aws

개발자 안내서, 버전 1

AWS IoT Greengrass

AWS IoT Greengrass: 개발자 안내서, 버전 1

Table of Contents

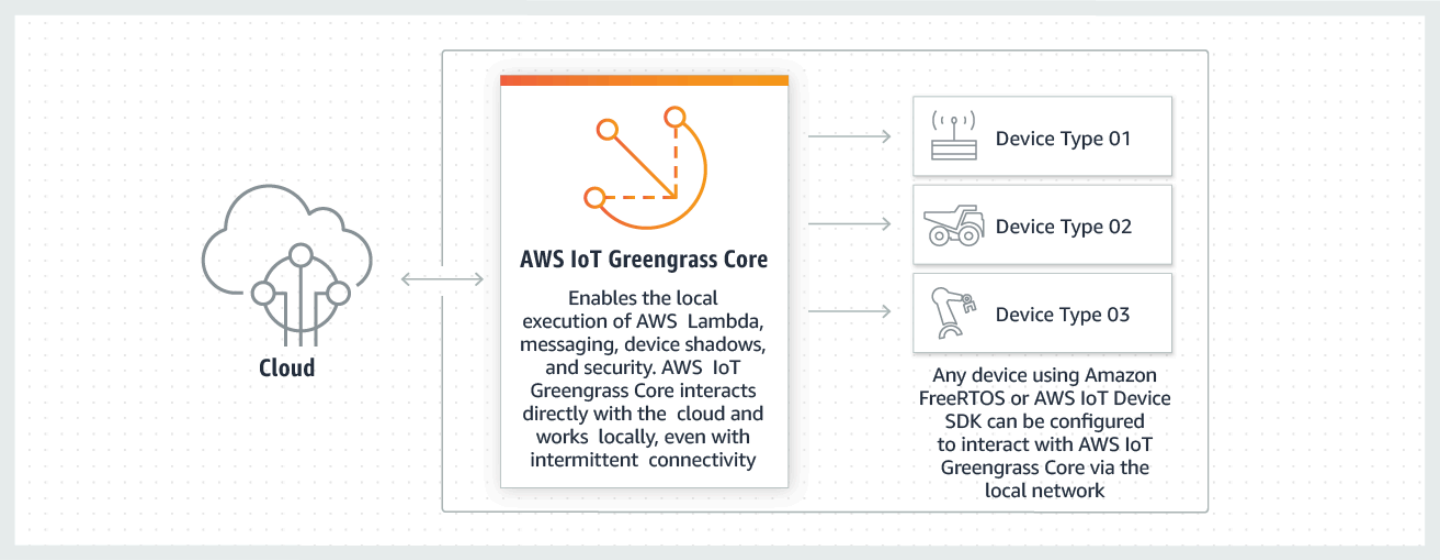
AWS IoT Greengrass Version 1 2023년 6월 30일에 수명 연장 단계에 들어갔습니다. AWS IoT Greengrass V1 관리형 정책에 대한 자세한 정보는 섹션을 참조하세요. 이 날짜 이후에는 기능, 개선 사항, 버그 수정 또는 보안 패치를 제공하는 업데이트가 AWS IoT Greengrass V1 릴리스되지 않습니다. 에서 실행되는 기기는 AWS IoT Greengrass V1 중단되지 않으며 계속 작동하고 클라우드에 연결됩니다. 새로운 기능이 크게 추가되고 추가 플랫폼에 대한 지원이 추가되는 으로 마이그레이션하는 AWS IoT Greengrass Version 2 것이 좋습니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

AWS IoT Greengrass(이)란 무엇인가요?

AWS IoT Greengrass는 클라우드 기능을 로컬 디바이스로 확장하는 소프트웨어입니다. 덕분에 디바이스는 정보 소스와 보다 밀접한 데이터를 수집 및 분석하고, 로컬 이벤트에 자율적으로 응답하며, 로컬 네트워크에서 서로 안전하게 통신할 수 있습니다. 또한 로컬 디바이스는 AWS IoT Core와 안전하게 통신하고 IoT 데이터를 AWS 클라우드로 내보낼 수 있습니다. AWS IoT Greengrass 개발자는 AWS Lambda 함수 및 미리 빌드된 커넥터를 사용하여 로컬 실행을 위해 디바이스에 배포되는 서버리스 애플리케이션을 생성할 수 있습니다.

다음 다이어그램은 AWS IoT Greengrass의 기본 아키텍처를 보여줍니다.



AWS IoT Greengrass가 있으면 고객이 IoT 디바이스와 애플리케이션 로직을 빌드할 수 있습니다. 특히 AWS IoT Greengrass는 디바이스에서 실행되는 애플리케이션 로직을 클라우드 기반으로 관리할 수 있는 기능을 제공합니다. 로컬로 배포된 Lambda 함수와 커넥터는 로컬 이벤트, 클라우드의 메시지 또는 기타 소스에서 트리거됩니다.

AWS IoT Greengrass에서, 디바이스는 클라우드에 연결하지 않고도 로컬 네트워크와 안전하게 통신하고 메시지를 교환할 수 있습니다. AWS IoT Greengrass은(는) 연결이 끊어지면 메시지를 지능적으로 버퍼해 클라우드로 들어오거나 나가는 메시지를 보존하는 로컬 게시/구독 메시지 관리자를 제공합니다.

AWS IoT Greengrass은(는) 다음과 같은 방법으로 사용자 데이터를 보호합니다.

- 보안 인증 및 디바이스 인증을 통해 보호합니다.
- 로컬 네트워크에서 보안 연결을 통해 보호합니다.

- 로컬 디바이스와 클라우드 간에 보호합니다.

클라우드와의 연결이 끊어져도 디바이스 보안 자격 증명은 해지될 때까지 하나의 그룹 내에서 작동하므로 디바이스가 로컬에서 계속 안전하게 통신을 실행할 수 있습니다.

AWS IoT GreengrassLambda 함수의 안전한 over-the-air 업데이트를 제공합니다.

AWS IoT Greengrass은(는) 다음 요소로 구성됩니다.

- 소프트웨어 배포
  - AWS IoT Greengrass 코어 소프트웨어
  - AWS IoT Greengrass 코어 SDK
- 클라우드 서비스
  - AWS IoT Greengrass API
- 특성
  - Lambda 런타임
  - 섀도우 구현
  - 메시지 관리자
  - 그룹 관리
  - 검색 서비스
  - O 업데이트 에이전트 ver-the-air
  - 스트림 관리자
  - 로컬 리소스 액세스
  - 로컬 기계 학습 추론
  - 로컬 암호 관리자
  - 서비스, 프로토콜 및 소프트웨어와의 통합이 내장된 커넥터

주제

- AWS IoT Greengrass 다운로드
- 연락을 기다리겠습니다.
- AWS IoT Greengrass 코어 소프트웨어 설치
- AWS IoT Greengrass 코어 구성

AWS IoT Greengrass 코어 소프트웨어

AWS IoT Greengrass 코어 소프트웨어는 다음과 같은 기능을 제공합니다.

- 커넥터 및 Lambda 함수의 배포 및 로컬 실행.
- AWS 클라우드로 자동 내보내기를 사용하여 로컬에서 데이터 스트림을 처리합니다.
- 관리형 구독을 사용한 디바이스, 커넥터, Lambda 함수 간의 로컬 네트워크를 통한 MQTT 메시징.
- 관리형 구독을 사용한 AWS IoT와 디바이스, 커넥터, Lambda 함수 간의 MQTT 메시징.
- 디바이스 인증 및 권한 부여를 사용하여 디바이스와 AWS 클라우드 간에 설정되는 보안 연결.
- 디바이스의 로컬 섀도우 동기화. 섀도우는 AWS 클라우드와 동기화하도록 구성할 수 있습니다.
- 로컬 디바이스 및 볼륨 리소스에 대한 액세스 제어.
- 로컬 추론 실행을 위한 클라우드 학습 머신러닝 모델 배포.
- 디바이스에서 Greengrass 코어 디바이스를 검색할 수 있도록 지원하는 자동 IP 주소 감지.
- 새 그룹 또는 업데이트된 그룹 구성의 중앙 배포. 구성 데이터를 다운로드한 후 코어 디바이스가 자동으로 다시 시작합니다.
- 사용자 정의 Lambda 함수의 안전한 over-the-air (OTA) 소프트웨어 업데이트
- 로컬 보안 암호의 안전하고 암호화된 저장 및 커넥터 및 Lambda 함수의 액세스 제어.

AWS IoT Greengrass 코어 인스턴스는 클라우드에 저장된 AWS IoT Greengrass 그룹 정의들을 생성하고 업데이트하는 AWS IoT Greengrass API를 통해 구성됩니다.

AWS IoT Greengrass 코어 소프트웨어 버전

AWS IoT Greengrass에서는 targ.gz 다운로드 파일, 빠른 시작 스크립트, 지원되는 Debian 플랫폼에서의 `apt` 설치를 포함하여 여러 가지 방법으로 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 자세한 설명은 the section called “AWS IoT Greengrass 코어 소프트웨어 설치” 섹션을 참조하세요.

다음 표는 AWS IoT Greengrass 코어 소프트웨어 버전의 새로운 기능과 변경 사항을 설명합니다.

GGC v1.11

1.11.6

버그 수정 및 개선 사항:

- 배포 중에 갑작스러운 전력 손실이 발생할 경우 복원력이 향상되었습니다.
- 스트림 관리자 데이터 손상으로 인해 AWS IoT Greengrass 코어 소프트웨어가 시작되지 않는 문제를 수정했습니다.
- 특정 시나리오에서 새 클라이언트 장치를 코어에 연결할 수 없는 문제를 수정했습니다.
- 스트림 관리자 스트림 이름이 `.log`를 포함할 수 없던 문제를 수정했습니다.

1.11.5

버그 수정 및 개선 사항:

- 일반 성능 향상 및 버그 수정.

1.11.4

버그 수정 및 개선 사항:

- AWS IoT Greengrass 코어 소프트웨어 v1.11.3으로의 업그레이드를 방해하는 스트림 관리자 관련 문제를 수정했습니다. 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 이제 OTA 업데이트를 사용하여 이전 v1.x 버전의 AWS IoT Greengrass Core 소프트웨어를 v1.11.4로 업그레이드할 수 있습니다.
- 일반 성능 향상 및 버그 수정.

1.11.3

버그 수정 및 개선 사항:

- Ubuntu 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 바로 실행하던 중 디바이스의 전원이 갑자기 끊긴 후 응답하지 않던 문제를 수정했습니다.
- 수명이 긴 Lambda 함수에 MQTT 메시지 전송이 지연되는 문제를 수정했습니다.
- `maxWorkItemCount` 값이 `1024`보다 큰 값으로 설정된 경우 MQTT 메시지가 제대로 전송되지 않던 문제를 수정했습니다.
- OTA 업데이트 에이전트가 `config.json`에서 `keepAlive` 속성에 지정된 MQTT `KeepAlive` 기간을 무시하던 문제를 수정했습니다.

- 일반 성능 향상 및 버그 수정.

> **Important**
>
> 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 이전 v1.x 버전에서 AWS IoT Greengrass 코어 소프트웨어 v1.11.3으로 업그레이드하지 마십시오. 스트림 관리자를 처음 활성화하는 경우 먼저 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 설치하는 것이 좋습니다.

1.11.1

버그 수정 및 개선 사항:

- 스트림 관리자의 메모리 사용량이 증가하는 문제가 수정되었습니다.
- 스트림 데이터의 지정된 기간 time-to-live (TTL) 보다 오랫동안 Greengrass 코어 디바이스의 전원이 꺼진 `0` 경우 스트림 관리자가 스트림의 시퀀스 번호를 재설정하던 문제를 수정했습니다.
- 스트림 관리자가 AWS 클라우드로 데이터를 내보내려는 재시도를 제대로 중단하지 못하던 문제를 수정했습니다.

1.11.0

새로운 기능:

- Greengrass 코어의 텔레메트리 에이전트는 로컬 텔레메트리 데이터를 수집하여 AWS 클라우드에 게시합니다. 추가 처리를 위해 원격 분석 데이터를 검색하려는 고객은 Amazon EventBridge 규칙을 생성하고 대상을 구독할 수 있습니다. 자세한 내용은 AWS IoT Greengrass 코어 디바이스에서 시스템 상태 텔레메트리 데이터 수집을 참조하십시오.
- AWS IoT Greengrass에서 시작된 로컬 작업자 프로세스의 현재 상태를 스냅샷으로 제공하는 로컬 HTTP API가 포함되어 있습니다. 자세한 내용은 로컬 상태 확인 API 호출을 참조하십시오.
- 스트림 관리자는 Amazon S3 및 AWS IoT SiteWise로 데이터를 자동으로 내보냅니다.

  새 스트림 관리자 파라미터를 사용하면 기존 스트림을 업데이트하고 데이터 내보내기를 일시 중지하거나 재개할 수 있습니다.
- 코어에서 Python 3.8.x Lambda 함수를 실행하기 위한 지원.
- Greengrass 코어 IPC 포트 번호를 구성하는 데 사용하는 `config.json`의 새 `ggDaemonPort` 속성입니다. 기본 포트 번호는 8000입니다.

Greengrass 코어 IPC 인증에 대한 제한 시간을 구성하는 데 사용하는 `config.json`의 새 `systemComponentAuthTimeout` 속성입니다. 기본값은 5,000밀리초입니다.

- AWS IoT Greengrass 그룹당 최대 AWS IoT 디바이스 수를 200개에서 2500개로 늘렸습니다.

  그룹당 최대 구독 수를 1000개에서 10000개로 늘렸습니다.

  자세한 내용은 AWS IoT Greengrass 엔드포인트 및 할당량을 참조하십시오.

버그 수정 및 개선 사항:

- Greengrass 서비스 프로세스의 메모리 사용률을 줄일 수 있는 일반 최적화.
- 새로운 런타임 구성 파라미터(`mountAllBlockDevices`)를 사용하면 Greengrass가 OverlayFS를 설정한 후 바인드 마운트를 사용하여 모든 블록 디바이스를 컨테이너에 마운트할 수 있습니다. 이 기능은 `/usr`이(가) `/` 계층 구조 아래에 있지 않을 경우 Greengrass 배포가 실패하는 문제를 해결했습니다.
- `/tmp`이 심볼릭 링크인 경우 AWS IoT Greengrass 코어 오류가 발생하는 문제를 수정했습니다.
- Greengrass 배포 에이전트가 `mlmodel_public` 폴더에서 사용하지 않는 기계 학습 모델 아티팩트를 제거하도록 하는 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

Extended life versions

1.10.5

버그 수정 및 개선 사항:

- 일반 성능 향상 및 버그 수정.

1.10.4

버그 수정 및 개선 사항:

- Ubuntu 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 바로 실행하던 중 디바이스의 전원이 갑자기 끊긴 후 응답하지 않던 문제를 수정했습니다.
- 수명이 긴 Lambda 함수에 MQTT 메시지 전송이 지연되는 문제를 수정했습니다.
- `maxWorkItemCount` 값이 `1024`보다 큰 값으로 설정된 경우 MQTT 메시지가 제대로 전송되지 않던 문제를 수정했습니다.

- OTA 업데이트 에이전트가 config.json에서 keepAlive 속성에 지정된 MQTT KeepAlive 기간을 무시하던 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

1.10.3

버그 수정 및 개선 사항:

- Greengrass 코어 IPC 인증에 대한 제한 시간을 구성하는 데 사용하는 config.json의 새 systemComponentAuthTimeout 속성입니다. 기본값은 5,000밀리초입니다.
- 스트림 관리자의 메모리 사용량이 증가하는 문제가 수정되었습니다.

1.10.2

버그 수정 및 개선 사항:

- AWS IoT Core와 MQTT 연결에서 게시, 구독 및 구독 취소 작업에 대한 제한 시간을 설정하는 데 사용하는 config.json의 새 mqttOperationTimeout 속성입니다.
- 일반 성능 향상 및 버그 수정.

1.10.1

버그 수정 및 개선 사항:

- 스트림 관리자는 파일 데이터 손상에 대한 복원력이 더 뛰어납니다.
- Linux 커널 5.1 이상을 사용하는 디바이스에 sysfs를 탑재하지 못하는 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

1.10.0

새로운 기능:

- 데이터 스트림을 로컬에서 처리하고 이를 AWS 클라우드로 자동으로 내보내는 스트림 관리자. 이 기능을 사용하려면 Greengrass 코어 디바이스에 Java 8이 필요합니다. 자세한 설명은 데이터 스트림 관리 섹션을 참조하세요.
- 코어 디바이스에서 Docker 애플리케이션을 실행하는 새로운 Greengrass Docker 애플리케이션 배포 커넥터. 자세한 설명은 the section called “Docker 애플리케이션 배포” 섹션을 참조하세요.
- 산업용 장치 데이터를 OPC-UA 서버의 자산 자산으로 전송하는 새로운 IoT SiteWise 커넥터. AWS IoT SiteWise 자세한 설명은 the section called “IoT SiteWise” 섹션을 참조하세요.
- 컨테이너화 없이 실행되는 Lambda 함수는 Greengrass 그룹의 기계 학습 리소스에 액세스할 수 있습니다. 자세한 설명은 the section called “기계 학습 리소스에 액세스” 섹션을 참조하세요.

- AWS IoT를 통해 MQTT 영구 세션 지원. 자세한 설명은 the section called “AWS IoT Core를 사용하는 MQTT 영구 세션” 섹션을 참조하세요.
- 로컬 MQTT 트래픽은 기본 포트 8883 이외의 포트를 통해 이동할 수 있습니다. 자세한 설명은 the section called “로컬 메시징을 위한 MQTT 포트” 섹션을 참조하세요.
- Lambda 함수에서 안정적인 메시지 게시를 위한 AWS IoT Greengrass 코어 SDK의 새로운 `queueFullPolicy` 옵션.
- 코어에서 Node.js 12.x Lambda 함수를 실행하기 위한 지원.
- 하드웨어 보안 통합을 통한 O ver-the-air (OTA) 업데이트는 OpenSSL 1.1을 사용하여 구성할 수 있습니다.
- 일반 성능 향상 및 버그 수정.

1.9.4

버그 수정 및 개선 사항:

- 일반 성능 향상 및 버그 수정.

1.9.3

새로운 기능:

- Armv6l에 대한 지원. AWS IoT Greengrass Armv6l 아키텍처의 Raspbian 배포(예: Raspberry Pi Zero 디바이스)에 코어 소프트웨어 v1.9.3 이상을 설치할 수 있습니다.
- ALPN을 통한 포트 443의 OTA 업데이트. MQTT 트래픽에 포트 443을 사용하는 Greengrass 코어는 이제 over-the-air (OTA) 소프트웨어 업데이트를 지원합니다. AWS IoT Greengrass애플리케이션 계층 프로토콜 네트워크 (ALPN) TLS 확장을 사용하여 이러한 연결을 활성화합니다. 자세한 내용은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 및 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.

버그 수정 및 개선 사항:

- v1.9.0에서 Python 2.7 Lambda 함수가 다른 Lambda 함수로 이진 페이로드를 보낼 수 없었던 버그를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

1.9.2

새로운 기능:

- 에 대한 지원 OpenWrt. AWS IoT Greengrass 코어 소프트웨어 v1.9.2 이상은 Armv8 (AArch64) 및 ARMv7L OpenWrt 아키텍처를 사용하는 배포판에 설치할 수 있습니다. OpenWrt 현재는 ML 추론을 지원하지 않습니다.

1.9.1

버그 수정 및 개선 사항:

- 주제에 와일드카드 문자가 포함된 `cloud`의 메시지를 삭제하는 v1.9.0의 버그를 수정합니다.

1.9.0

새로운 기능:

- Python 3.7 및 Node.js 8.10 Lambda 런타임을 지원합니다. Python 3.7 및 Node.js 8.10 런타임을 사용하는 Lambda 함수가 이제 AWS IoT Greengrass 코어에서 실행할 수 있습니다 (AWS IoT Greengrass는 Python 2.7 및 Node.js 6.10 런타임을 계속 지원함).
- MQTT 연결을 최적화했습니다. Greengrass 코어는 AWS IoT Core와의 더 적은 수의 연결을 설정합니다. 이 변경에 따라 연결 수를 기반으로 부과되는 요금의 운영 비용을 절감할 수 있습니다.
- 로컬 MQTT 서버에 EC(Elliptic Curve) 키를 사용할 수 있습니다. 로컬 MQTT 서버는 RSA 키 외에도 EC 키를 지원합니다. 키 유형과 상관없이 MQTT 서버 인증서에는 SHA-256 RSA 서명이 있습니다. 자세한 설명은 the section called “보안 주체” 섹션을 참조하세요.

버그 수정 및 개선 사항:

- 일반 성능 향상 및 버그 수정.

1.8.4

섀도우 동기화 및 디바이스 인증서 관리자 재연결 문제를 수정했습니다.

일반 성능 향상 및 버그 수정.

1.8.3

일반 성능 향상 및 버그 수정

1.8.2

일반 성능 향상 및 버그 수정

1.8.1

일반 성능 향상 및 버그 수정.

1.8.0

새로운 기능:

- 그룹의 Lambda 함수에 대해 구성 가능한 기본 액세스 자격 증명. 이 그룹 레벨 설정은 Lambda 함수를 실행하는 데 사용된 기본 권한을 결정합니다. 사용자 ID, 그룹 ID 또는 둘 다 설정할 수 있습니다. 개별 Lambda 함수는 해당 그룹의 기본 액세스 자격 증명을 재정의할 수 있습니다. 자세한 설명은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.
- 포트 443을 통한 HTTPS 트래픽 HTTPS 통신은 기본 포트 8443 대신에 포트 443을 통한 이동을 위해 구성됩니다. 이는 Application Layer Protocol Network (ALPN) TLS 확장에 대한 AWS IoT Greengrass 지원을 보완하고 모든 Greengrass 메시징 트래픽(MQTT 및 HTTPS 모두)이 포트 443을 사용할 수 있도록 합니다. 자세한 설명은 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.
- AWS IoT 연결에 대해 예상 가능한 이름의 클라이언트 ID 이 변경에 따라 AWS IoT Device Defender 및 AWS IoT 수명 주기 이벤트가 지원되므로 연결, 연결 끊기, 구독 및 구독 해제 이벤트에 대한 알림을 받을 수 있습니다. 또한 예상 가능한 이름을 지정하면 연결 ID를 중심으로 논리를 쉽게 만들 수 있습니다(예: 인증서 속성을 기준으로 구독 정책 템플릿을 만드는 경우). 자세한 설명은 the section called “AWS IoT를 통한 MQTT 연결용 클라이언트 ID” 섹션을 참조하세요.

버그 수정 및 개선 사항:

- 섀도우 동기화 및 디바이스 인증서 관리자 재연결 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

1.7.1

새로운 기능:

- 로컬 인프라, 디바이스 프로토콜, AWS, 기타 클라우드 서비스와 기본적으로 통합되는 Greengrass 커넥터. 자세한 설명은 커넥터를 사용하여 서비스 및 프로토콜과 통합 섹션을 참조하세요.
- AWS IoT Greengrass가 AWS Secrets Manager를 코어 디바이스로 확장함에 따라 커넥터와 Lambda 함수에 암호, 토큰, 기타 비밀 정보를 사용 가능합니다. 암호는 전송 및 저장 상태에서 암호화됩니다. 자세한 설명은 코어에 암호 배포 섹션을 참조하세요.
- 신뢰 보안 옵션의 하드웨어 루트에 대한 지원. 자세한 설명은 the section called “하드웨어 보안 통합” 섹션을 참조하세요.

- Lambda 함수가 Greengrass 컨테이너 없이 실행되고 지정된 사용자 및 그룹의 권한을 사용할 수 있도록 허용하는 격리 및 권한 설정. 자세한 설명은 the section called “Greengrass Lambda 함수 실행 제어” 섹션을 참조하세요.
- Greengrass 그룹을 컨테이너화 없이 실행하도록 구성하여 Windows, macOS 또는 Linux의 Docker 컨테이너에서 AWS IoT Greengrass를 실행할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.
- ALPN(Application Layer Protocol Negotiation) 또는 네트워크 프록시 경유 연결을 통한 포트 443의 MQTT 메시징. 자세한 설명은 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.
- SageMaker Neo 딥 러닝 런타임은 SageMaker Neo 딥 러닝 컴파일러로 최적화된 머신 러닝 모델을 지원합니다. Neo 딥 러닝 런타임에 관해 자세한 내용은 the section called “ML 추론을 위한 런타임 및 라이브러리” 섹션을 참조하십시오.
- Raspberry Pi 코어 디바이스에서 Raspbian Stretch(2018-06-27)에 대한 지원.

버그 수정 및 개선 사항:

- 일반 성능 향상 및 버그 수정.

그에 더해 이 릴리스로는 다음 기능을 사용할 수 있습니다.

- CPU 아키텍처, 커널 구성, 드라이버가 AWS IoT Greengrass에서 기능하는지 확인하는 데 사용 가능한 AWS IoT Greengrass용 AWS IoT 디바이스 테스터. 자세한 설명은 AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터 사용 섹션을 참조하세요.
- AWS IoT Greengrass코어 소프트웨어, AWS IoT Greengrass 코어 SDK 및 AWS IoT Greengrass Machine Learning SDK 패키지는 Amazon을 통해 다운로드할 수 있습니다. CloudFront 자세한 설명은 the section called “AWS IoT Greengrass 다운로드” 섹션을 참조하세요.

1.6.1

새로운 기능:

- Greengrass 코어에서 이진 코드를 실행하는 Lambda 실행 파일입니다. C용 새 AWS IoT Greengrass 코어 SDK를 사용하여 Lambda 실행 파일을 C 및 C++로 작성할 수 있습니다. 자세한 설명은 the section called “Lambda 실행 파일” 섹션을 참조하세요.
- 재시작 시에도 계속 유지되는 로컬 스토리지 메시지 캐시(선택 사항)입니다. 처리를 위해 대기 중인 MQTT 메시지에 대한 스토리지 설정을 구성할 수 있습니다. 자세한 설명은 the section called “MQTT 메시지 대기열” 섹션을 참조하세요.

- 코어 디바이스 연결이 해제될 때 구성할 수 있는 최대 재연결 시도 간격입니다. 자세한 내용은 `mqttMaxConnectionRetryInterval`의 the section called “AWS IoT Greengrass 코어 구성 파일” 속성을 참조하십시오.
- 호스트/proc 디렉터리에 대한 로컬 리소스 액세스입니다. 자세한 설명은 로컬 리소스에 액세스 섹션을 참조하세요.
- 구성 가능한 쓰기 디렉터리입니다. AWS IoT Greengrass 코어 소프트웨어는 읽기 전용 및 읽기-쓰기 위치에 배포가 가능합니다. 자세한 설명은 the section called “쓰기 디렉터리 구성” 섹션을 참조하세요.

버그 수정 및 개선 사항:

- Greengrass 코어 내에서, 그리고 디바이스와 코어 사이에서 메시지 게시 성능이 향상되었습니다.
- 사용자 정의 Lambda 함수에 의해 생성된 로그를 처리하기 위해 필요한 컴퓨팅 리소스가 감소되었습니다.

1.5.0

새로운 기능:

- AWS IoT Greengrass 기계 학습(ML) 추론이 일반적으로 사용 가능합니다. 클라우드에서 빌드 및 교육된 모델을 사용하여 AWS IoT Greengrass 디바이스에서 로컬 방식으로 ML 추론을 수행할 수 있습니다. 자세한 설명은 기계 학습 추론 수행 섹션을 참조하세요.
- Greengrass Lambda 함수는 이제 JSON 외에도 이진 데이터를 입력 페이로드로 지원합니다. 이 기능을 사용하려면 AWS IoT Greengrass 코어 SDK 버전 1.1.0으로 업그레이드해야 합니다. 이 버전은 AWS IoT Greengrass 코어 SDK 다운로드 페이지에서 다운로드할 수 있습니다.

버그 수정 및 개선 사항:

- 전체 메모리 공간을 줄였습니다.
- 클라우드 메시지 전송 성능이 개선되었습니다.
- 다운로드 에이전트, 디바이스 인증서 관리자(DCM) 및 OTA 업데이트 에이전트의 성능과 안정성이 개선되었습니다.
- 사소한 버그가 수정됨.

1.3.0

새로운 기능:

- 클라우드에 배포된 Greengrass 업데이트 작업을 처리할 수 있는 O ver-the-air (OTA) 업데이트 에이전트 에이전트는 새로운 `/greengrass/ota` 디렉터리에 있습니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.

- Greengrass Lambda 함수는 로컬 리소스 액세스 기능을 통해 주변기기 및 볼륨 같은 로컬 리소스에 액세스합니다. 자세한 설명은 Lambda 함수와 커넥터를 사용하여 로컬 리소스에 액세스 섹션을 참조하세요.

1.1.0

새로운 기능:

- 배포 완료된 AWS IoT Greengrass 그룹을 Lambda 함수, 구독 및 구성을 삭제하여 재설정할 수 있습니다. 자세한 설명은 the section called “배포 재설정” 섹션을 참조하세요.
- Python 2.7 외에 Node.js 6.10 및 Java 8 Lambda 런타임 지원.

AWS IoT Greengrass의 이전 버전에서 마이그레이션하려면:

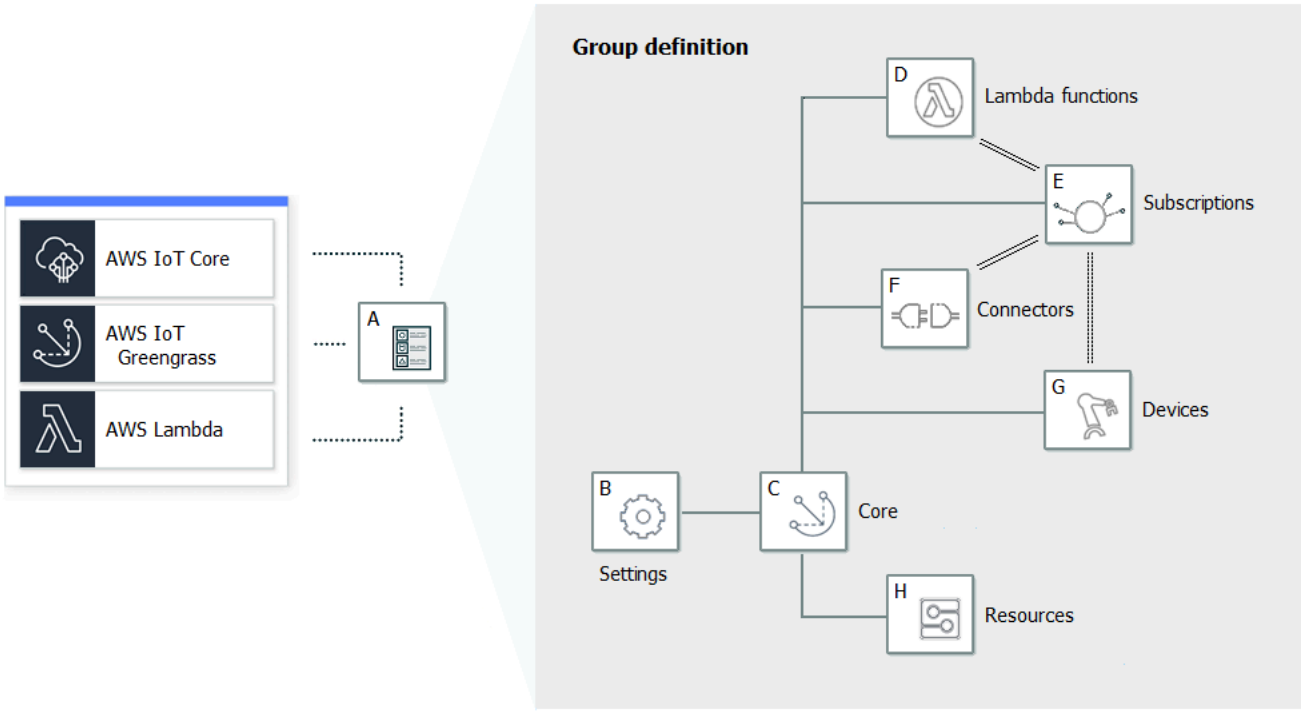
- `/greengrass/configuration/certs` 폴더에서 `/greengrass/certs`로 인증서를 복사합니다.
- `/greengrass/configuration/config.json`를 `/greengrass/config/config.json`에 복사합니다.
- `/greengrass/greengrassd` 대신 `/greengrass/ggc/core/greengrassd`를 실행합니다.
- 새 코어에 그룹을 배포합니다.

1.0.0

초기 버전

AWS IoT Greengrass 그룹

Greengrass 그룹은 Greengrass 코어, 디바이스 및 구독과 같은 설정 및 구성 요소의 모음입니다. 그룹은 상호 작용의 범위를 정의하는 데 사용됩니다. 예를 들어, 그룹은 건물의 한 층, 트럭 한 대 또는 전체 채광 현장을 나타낼 수 있습니다. 다음 다이어그램은 Greengrass 그룹을 구성할 수 있는 구성 요소를 보여줍니다.



앞서 제시한 다이어그램에서

A: Greengrass 그룹 정의

그룹 설정 및 구성 요소에 대한 정보.

B: Greengrass 그룹 설정

다음이 포함됩니다.

- Greengrass 그룹 역할.
- 인증 기관과 로컬 연결 구성.
- Greengrass 코어 연결 정보.
- 기본 Lambda 런타임 환경. 자세한 설명은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.
- CloudWatch 및 로컬 로그 구성 자세한 설명은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요.

C: Greengrass 코어

Greengrass 코어를 나타내는 AWS IoT 사물(디바이스). 자세한 설명은 the section called “AWS IoT Greengrass 코어 구성” 섹션을 참조하세요.

D: Lambda 함수 정의

관련 구성 데이터를 포함하여 코어에서 로컬로 실행되는 Lambda 함수의 목록입니다. 자세한 설명은 로컬 Lambda 함수 실행 섹션을 참조하세요.

E: 구독 정의

MQTT 메시지를 사용해 통신을 할 수 있게 해주는 구독 목록입니다. 구독은 다음을 정의합니다.

- 메시지 소스 및 메시지 대상. 이 항목은 클라이언트 디바이스, Lambda 함수, 커넥터, AWS IoT Core 및 로컬 섀도우 서비스일 수 있습니다.
- 메시지를 필터링하는 데 사용되는 주제 또는 제목.

자세한 설명은 the section called “MQTT 메시징 워크플우로의 관리형 구독” 섹션을 참조하세요.

F: 커넥터 정의

관련 구성 데이터를 포함하여 코어에서 로컬로 실행되는 커넥터의 목록입니다. 자세한 설명은 커넥터를 사용하여 서비스 및 프로토콜과 통합 섹션을 참조하세요.

G: 디바이스 정의

Greengrass 그룹의 구성원인 AWS IoT 사물(클라이언트 디바이스 또는 디바이스라고 함)을 연결된 구성 데이터와 함께 표시하는 목록입니다. 자세한 설명은 the section called “AWS IoT Greengrass 내 디바이스” 섹션을 참조하세요.

H: 리소스 정의

관련 구성 데이터를 포함하여 Greengrass 코어DP 있는 로컬 리소스, 기계 학습 리소스 및 암호 리소스의 목록입니다. 자세한 내용은 로컬 리소스에 액세스, 기계 학습 추론 수행, 코어에 암호 배포 단원을 참조하세요.

배포 시 Greengrass 그룹 정의, Lambda 함수, 커넥터, 리소스 및 구독 테이블이 코어 디바이스에 복사됩니다. 자세한 설명은 AWS IoT Greengrass 그룹 배포 섹션을 참조하세요.

AWS IoT Greengrass 내 디바이스

Greengrass 그룹에는 두 가지 유형의 AWS IoT 디바이스가 포함될 수 있습니다.

Greengrass 코어

Greengrass 코어는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 디바이스로 AWS IoT Core 및 AWS IoT Greengrass 서비스와 직접 통신할 수 있습니다. 코어에는 AWS IoT Core를 통해 인증

하는 데 사용되는 자체 디바이스 인증서가 있습니다. AWS IoT Core 레지스트리에 디바이스 섀도우와 항목이 있습니다. Greengrass 코어는 로컬 Lambda 런타임과 배포 에이전트를 실행할 뿐만 아니라, 클라이언트 디바이스가 그룹 및 코어 연결 정보를 자동으로 검색할 수 있도록 허용하기 위해 AWS IoT Greengrass 서비스에 IP 주소 정보를 전송하는 IP 주소 트래커도 실행합니다. 자세한 설명은 the section called “AWS IoT Greengrass 코어 구성” 섹션을 참조하세요.

> **Note**
>
> Greengrass 그룹은 정확히 하나의 코어를 포함해야 합니다.

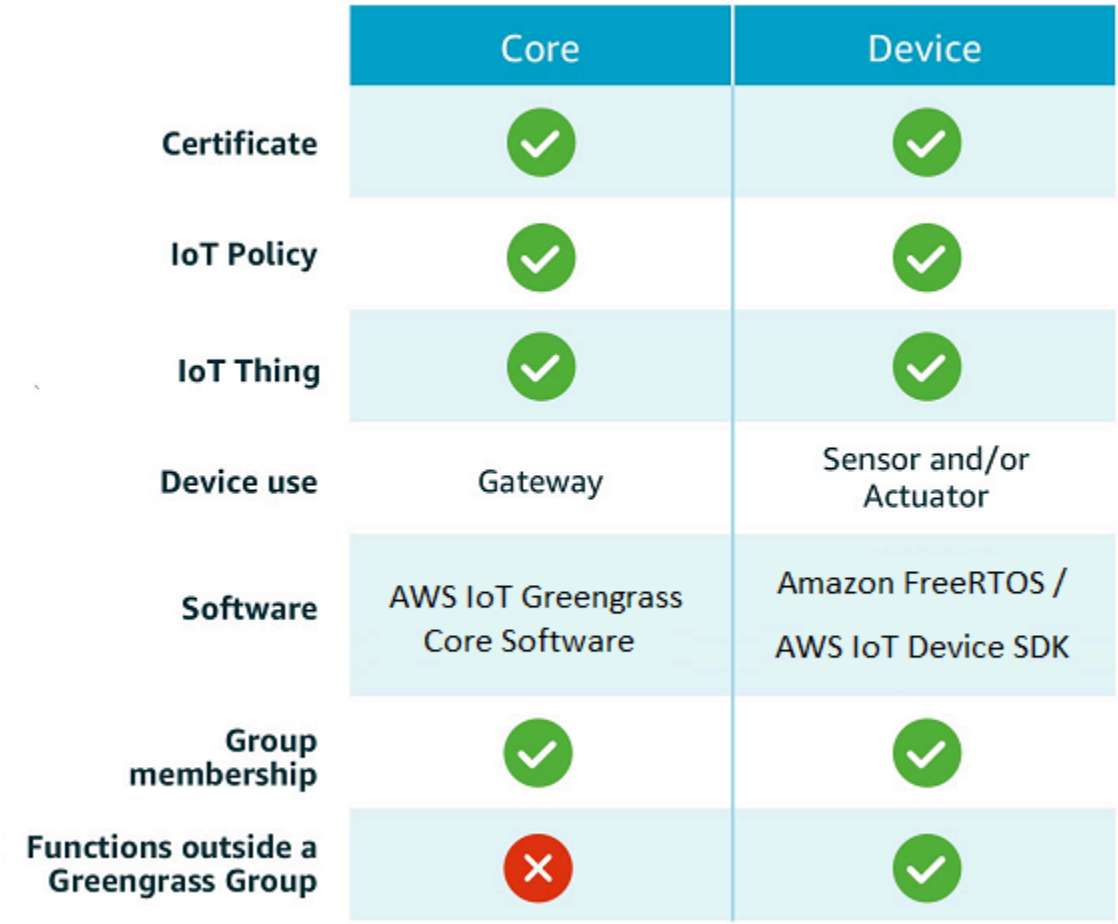
클라이언트 디바이스

클라이언트 디바이스(연결된 디바이스, Greengrass 디바이스 또는 디바이스라고도 함)는 MQTT를 통해 Greengrass 코어에 연결하는 디바이스입니다. 디바이스에는 AWS IoT Core 인증, 디바이스 섀도우 및 AWS IoT Core 레지스트리 항목에 대한 자체 인증서가 포함되어 있습니다. Greengrass 디바이스는 를 실행하거나 디바이스 SDKAWS IoT 또는 AWS IoT Greengrass 검색 API를 사용하여 동일한 Greengrass 그룹의 코어에 연결하고 인증하는 데 사용되는 검색 정보를 가져올 수 있습니다. AWS IoT 콘솔을 사용하여 AWS IoT Greengrass에 대한 클라이언트 디바이스를 만들고 구성하는 방법에 대한 자세한 내용은 the section called “모듈 4: AWS IoT Greengrass 그룹에서 클라이언트 디바이스와 상호 작용”를 참조하십시오. 또는 를 사용하여 클라이언트 장치를 만들고 구성하는 방법을 보여주는 예제는 AWS CLI명령 참조를 참조하십시오 create-device-definition. AWS CLI AWS IoT Greengrass

Greengrass 그룹에서는 클라이언트 디바이스가 MQTT를 통해 그룹 내에 있는 Lambda 함수, 커넥터 및 기타 클라이언트 디바이스, 그리고 또는 AWS IoT Core 로컬 섀도우 서비스와 통신하도록 허용하는 구독을 생성할 수 있습니다. MQTT 메시지는 코어를 통해 라우팅됩니다. 코어 디바이스와 클라우드의 연결이 끊어지면 클라이언트 디바이스는 로컬 네트워크를 통해 계속 통신할 수 있습니다. 클라이언트 디바이스는 마이크로 컨트롤러 기반 소형 디바이스부터 대형 어플라이언스까지 다양한 크기일 수 있습니다. 현재 Greengrass 그룹은 최대 2,500개의 클라이언트 디바이스를 포함할 수 있습니다. 한 클라이언트 디바이스는 최대 10개 그룹의 멤버일 수 있습니다.

> **Note**
>
> OPC-UA는 산업 통신을 위한 정보 교환 표준입니다. Greengrass 코어에서 OPC-UA에 대한 지원을 구현하려면 IoT 커넥터를 사용할 수 있습니다. SiteWise 이 커넥터는 OPC-UA 서버의 산업용 디바이스 데이터를 AWS IoT SiteWise의 자산 속성으로 보냅니다.

다음 테이블은 이러한 디바이스 유형들이 어떤 관계가 있는지를 보여줍니다.



AWS IoT Greengrass 코어 디바이스는 두 위치에 인증서를 저장합니다.

- */greengrass-root*/certs의 코어 디바이스 인증서. 일반적으로 코어 디바이스 인증서의 이름은 *hash*.cert.pem(예: 86c84488a5.cert.pem)입니다. 이 인증서는 코어가 AWS IoT Core 및 AWS IoT Greengrass 서비스에 연결될 때 AWS IoT 클라이언트가 상호 인증을 위해 사용합니다.
- */greengrass-root*/ggc/var/state/server의 MQTT 서버 인증서. MQTT 서버 인증서의 이름이 server.crt로 지정됩니다. 이 인증서는 로컬 MQTT 서버(Greengrass 코어에 있음) 사이의 상호 인증에 사용됩니다.

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

SDK

AWS IoT Greengrass와 작업하는 데 사용되는 AWS 제공 SDK는 다음과 같습니다.

AWS SDK

AWS SDK를 사용해 Amazon S3, Amazon DynamoDB, AWS IoT, AWS IoT Greengrass 등을 포함한 모든 AWS 서비스와 상호 작용하는 애플리케이션을 빌드합니다. AWS IoT Greengrass의 컨텍스트에서는 배포된 Lambda 함수 내 AWS SDK를 사용하여 모든 AWS 서비스를 직접 호출할 수 있습니다. 자세한 설명은 AWS SDK 섹션을 참조하세요.

> **Note**
>
> AWS SDK에서 사용할 수 있는 Greengrass 관련 작업은 AWS IoT Greengrass API 및 AWS CLI에서도 사용할 수 있습니다.

AWS IoT 디바이스 SDK

AWS IoT 디바이스 SDK는 디바이스의 AWS IoT Core 또는 AWS IoT Greengrass 연결을 지원합니다. 자세한 내용은 AWS IoT 개발자 안내서의 AWS IoT 디바이스 SDK 섹션을 참조하세요.

클라이언트 디바이스는 AWS IoT 디바이스 SDK V2를 사용하여 Greengrass 코어의 연결 정보를 검색할 수도 있습니다. 연결 정보에는 다음이 포함됩니다.

- 클라이언트 디바이스가 속한 Greengrass 그룹의 ID.
- 각 그룹에 있는 Greengrass 코어의 IP 주소. 이를 코어 엔드포인트라고도 합니다.
- 디바이스가 코어와의 상호 인증에 사용하는 그룹 CA 인증서. 자세한 설명은 the section called “디바이스 연결 워크플로” 섹션을 참조하세요.

> **Note**
>
> AWS IoT 장치 SDK v1에서는 C++ 및 Python 플랫폼만이 내장된 검색 지원을 제공합니다.

AWS IoT Greengrass 코어 SDK

AWS IoT Greengrass 코어 SDK는 Lambda 함수가 Greengrass 코어와의 상호 작용, AWS IoT에 메시지 게시, 로컬 섀도우 서비스와의 상호 작용, 배포된 다른 Lambda 함수의 간접 호출, 암호 리

소스 액세스를 수행할 수 있도록 지원합니다. 이 SDK는 AWS IoT Greengrass 코어에서 실행되는 Lambda 함수가 사용합니다. 자세한 설명은 AWS IoT Greengrass 코어 SDK 섹션을 참조하세요.

AWS IoT Greengrass 기계 학습 SDK

AWS IoT Greengrass 기계 학습 SDK는 Lambda 함수가 Greengrass 코어에 기계 학습 리소스로서 배포된 기계 학습 모델을 이용할 수 있도록 지원합니다. 이 SDK는 AWS IoT Greengrass 코어에서 실행되며 로컬 추론 서비스로 상호 작용하는 Lambda 함수에 사용됩니다. 자세한 설명은 AWS IoT Greengrass기계 학습 SDK 섹션을 참조하세요.

지원되는 플랫폼 및 요구 사항

다음 탭에는 AWS IoT Greengrass 코어 소프트웨어에 지원되는 플랫폼과 요구 사항이 나열되어 있습니다.

> **Note**
>
> AWS IoT Greengrass코어 소프트웨어 다운로드에서 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

GGC v1.11

지원되는 플랫폼:

- 아키텍처: ARMv7l
  - OS: Linux
  - OS: 리눅스 () OpenWrt
- 아키텍처: Armv8(AArch64)
  - OS: Linux
  - OS: 리눅스 (OpenWrt)
- 아키텍처: ARMv6l
  - OS: Linux
- 아키텍처: x86_64
  - OS: Linux

- Windows, macOS 및 Linux 플랫폼은 도커 컨테이너에서 AWS IoT Greengrass를 실행할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.

요구 사항:

- AWS IoT Greengrass 코어 소프트웨어에 사용할 수 있는 최소 128MB의 디스크 공간. OTA 업데이트 에이전트를 사용하는 경우 최소 용량은 400MB입니다.
- AWS IoT Greengrass 코어 소프트웨어에 최소 128MB의 RAM이 할당됩니다. 스트림 관리자가 활성화된 경우 최소 크기는 198MB RAM입니다.

> **Note**
>
> AWS IoT 콘솔의 기본 그룹 생성 옵션을 사용하여 Greengrass 그룹을 생성하는 경우 스트림 관리자가 기본적으로 활성화됩니다.

- Linux 커널 버전:
  - 컨테이너를 이용한 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 4.4 이상이 필요합니다.
  - 컨테이너 없이 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 3.17 이상이 필요합니다. 이 구성에서 Greengrass 그룹에 대한 기본 Lambda 함수 컨테이너화는 컨테이너 없음으로 설정해야 합니다. 지침은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.
- GNU C 라이브러리 (glibc) 버전 2.14 이상 OpenWrt 배포판에는 musl C 라이브러리 버전 1.1.16 이상이 필요합니다.
- `/var/run` 디렉터리가 디바이스에 있어야 합니다.
- `/dev/stdin`, `/dev/stdout` 및 `/dev/stderr` 파일이 사용 가능해야 합니다.
- 하드링크 및 소프트링크 보호를 활성화해야 합니다. 그렇지 않으면 AWS IoT Greengrass 플래그를 사용하여 안전하지 않은 모드에서만 `-i`를 실행할 수 있습니다.
- 디바이스에서 다음 Linux 커널 구성을 활성화해야 합니다.
  - 네임스페이스:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS

    - CONFIG_PID_NS
  - Cgroups:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    커널에서 cgroups를 지원해야 합니다. AWS IoT Greengrass컨테이너로 실행할 때는 다음 요구 사항이 적용됩니다.

    - AWS IoT Greengrass가 Lambda 함수의 메모리 제한을 설정하도록 하려면 메모리 cgroup을 활성화하고 마운트해야 합니다.
    - 로컬 리소스 액세스 권한이 있는 Lambda 함수를 사용하여 AWS IoT Greengrass 코어 디바이스에서 파일을 여는 경우 디바이스 cgroup을 활성화하고 마운트해야 합니다.
  - 기타:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- Amazon S3 및 AWS IoT의 루트 인증서가 시스템 트러스트 스토어에 있어야 합니다.
- 스트림 관리자에서는 기본 AWS IoT Greengrass 코어 소프트웨어 메모리 요구 사항 외에 Java 8 런타임과 최소 70MB의 RAM이 필요합니다. AWS IoT 콘솔의 기본 그룹 생성 옵션을 사용할 경우 스트림 관리자가 기본적으로 활성화됩니다. 배포판에서는 스트림 매니저가 지원되지 않습니다. OpenWrt
- 로컬로 실행할 Lambda 함수에 필요한 AWS Lambda런타임을 지원하는 라이브러리. 필수 라이브러리를 코어에 설치하고 PATH 환경 변수에 추가해야 합니다. 동일한 코어에 여러 라이브러리를 설치할 수 있습니다.
  - Python 3.8 런타임을 사용하는 기능의 경우 Python 버전 3.8.
  - Python 3.7 실행 시간을 사용하는 기능의 경우 Python 버전 3.7.
  - Python 2.7 실행 시간을 사용하는 기능의 경우 Python 버전 2.7.
  - Node.js 12.x 런타임을 사용하는 함수의 경우 Node.js 버전 12.x.

- Java 8 런타임을 사용하는 함수의 경우 Java 버전 8 이상

  > **Note**
  >
  > OpenWrt 배포판에서 Java를 실행하는 것은 공식적으로 지원되지 않습니다. 하지만 OpenWrt 빌드에서 Java를 지원하는 경우 Java로 작성된 Lambda 함수를 디바이스에서 실행할 수 있습니다. OpenWrt

  Lambda 런타임용 AWS IoT Greengrass 지원에 대한 자세한 내용은 로컬 Lambda 함수 실행 섹션을 참조하세요.

- (OTA) 업데이트 에이전트에는 다음과 같은 셸 명령 (BusyBox 변형 아님) 이 over-the-air 필요합니다.
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`
  - `rm`
  - `ln`
  - `cut`
  - `cat`
  - `/bin/bash`

GGC v1.10

지원되는 플랫폼:

- 아키텍처: ARMv7l
  - OS: Linux
  - OS: 리눅스 (OpenWrt)
- 아키텍처: Armv8(AArch64)
  - OS: Linux
  - OS: 리눅스 (OpenWrt)
- 아키텍처: ARMv6l
  - OS: Linux
- 아키텍처: x86_64
  - OS: Linux
- Windows, macOS 및 Linux 플랫폼은 도커 컨테이너에서 AWS IoT Greengrass를 실행할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.

요구 사항:

- AWS IoT Greengrass 코어 소프트웨어에 사용할 수 있는 최소 128MB의 디스크 공간. OTA 업데이트 에이전트를 사용하는 경우 최소 용량은 400MB입니다.
- AWS IoT Greengrass 코어 소프트웨어에 최소 128MB의 RAM이 할당됩니다. 스트림 관리자가 활성화된 경우 최소 크기는 198MB RAM입니다.

  > **Note**
  >
  > AWS IoT 콘솔의 기본 그룹 생성 옵션을 사용하여 Greengrass 그룹을 생성하는 경우 스트림 관리자가 기본적으로 활성화됩니다.

- Linux 커널 버전:
  - 컨테이너를 이용한 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 4.4 이상이 필요합니다.
  - 컨테이너 없이 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 3.17 이상이 필요합니다. 이 구성에서 Greengrass 그룹에 대한 기본 Lambda 함수 컨테이너화는 컨테이너 없음으

로 설정해야 합니다. 지침은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.

- GNU C 라이브러리 (glibc) 버전 2.14 이상 OpenWrt 배포판에는 musl C 라이브러리 버전 1.1.16 이상이 필요합니다.
- `/var/run` 디렉터리가 디바이스에 있어야 합니다.
- `/dev/stdin`, `/dev/stdout` 및 `/dev/stderr` 파일이 사용 가능해야 합니다.
- 하드링크 및 소프트링크 보호를 활성화해야 합니다. 그렇지 않으면 AWS IoT Greengrass 플래그를 사용하여 안전하지 않은 모드에서만 `-i`를 실행할 수 있습니다.
- 디바이스에서 다음 Linux 커널 구성을 활성화해야 합니다.
  - 네임스페이스:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    커널에서 cgroups를 지원해야 합니다. AWS IoT Greengrass컨테이너로 실행할 때는 다음 요구 사항이 적용됩니다.
    - AWS IoT Greengrass가 Lambda 함수의 메모리 제한을 설정하도록 하려면 메모리 cgroup을 활성화하고 마운트해야 합니다.
    - 로컬 리소스 액세스 권한이 있는 Lambda 함수를 사용하여 AWS IoT Greengrass 코어 디바이스에서 파일을 여는 경우 디바이스 cgroup을 활성화하고 마운트해야 합니다.
  - 기타:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP

    - CONFIG_SHMEM
- Amazon S3 및 AWS IoT의 루트 인증서가 시스템 트러스트 스토어에 있어야 합니다.
- 스트림 관리자에서는 기본 AWS IoT Greengrass 코어 소프트웨어 메모리 요구 사항 외에 Java 8 런타임과 최소 70MB의 RAM이 필요합니다. AWS IoT 콘솔의 기본 그룹 생성 옵션을 사용할 경우 스트림 관리자가 기본적으로 활성화됩니다. 배포판에서는 스트림 매니저가 지원되지 않습니다. OpenWrt
- 로컬로 실행할 Lambda 함수에 필요한 AWS Lambda런타임을 지원하는 라이브러리. 필수 라이브러리를 코어에 설치하고 PATH 환경 변수에 추가해야 합니다. 동일한 코어에 여러 라이브러리를 설치할 수 있습니다.
    - Python 3.7 실행 시간을 사용하는 기능의 경우 Python 버전 3.7.
    - Python 2.7 실행 시간을 사용하는 기능의 경우 Python 버전 2.7.
    - Node.js 12.x 런타임을 사용하는 함수의 경우 Node.js 버전 12.x.
    - Java 8 런타임을 사용하는 함수의 경우 Java 버전 8 이상

      > **Note**
      >
      > OpenWrt 배포판에서 Java를 실행하는 것은 공식적으로 지원되지 않습니다. 하지만 OpenWrt 빌드에서 Java를 지원하는 경우 Java로 작성된 Lambda 함수를 디바이스에서 실행할 수 있습니다. OpenWrt

    Lambda 런타임용 AWS IoT Greengrass 지원에 대한 자세한 내용은 로컬 Lambda 함수 실행 섹션을 참조하세요.
- (OTA) 업데이트 에이전트에는 다음과 같은 셸 명령 (BusyBox 변형 아님) 이 over-the-air 필요합니다.
    - `wget`
    - `realpath`
    - `tar`
    - `readlink`
    - `basename`
    - `dirname`
    - `pidof`
    - `df`

- `grep`
- `umount`
- `mv`
- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`
- `/bin/bash`

GGC v1.9

지원되는 플랫폼:

- 아키텍처: ARMv7l
  - OS: Linux
  - OS: 리눅스 (OpenWrt)
- 아키텍처: Armv8(AArch64)
  - OS: Linux
  - OS: 리눅스 (OpenWrt)
- 아키텍처: ARMv6l
  - OS: Linux
- 아키텍처: x86_64
  - OS: Linux
- Windows, macOS 및 Linux 플랫폼은 도커 컨테이너에서 AWS IoT Greengrass를 실행할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.

요구 사항:

- AWS IoT Greengrass 코어 소프트웨어에 사용할 수 있는 최소 128MB의 디스크 공간. OTA 업데이트 에이전트를 사용하는 경우 최소 용량은 400MB입니다.

- AWS IoT Greengrass 코어 소프트웨어에 최소 128MB의 RAM이 할당됩니다.
- Linux 커널 버전:
  - 컨테이너를 이용한 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 4.4 이상이 필요합니다.
  - 컨테이너 없이 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 3.17 이상이 필요합니다. 이 구성에서 Greengrass 그룹에 대한 기본 Lambda 함수 컨테이너화는 컨테이너 없음으로 설정해야 합니다. 지침은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.
- GNU C 라이브러리 (glibc) 버전 2.14 이상 OpenWrt 배포판에는 musl C 라이브러리 버전 1.1.16 이상이 필요합니다.
- `/var/run` 디렉터리가 디바이스에 있어야 합니다.
- `/dev/stdin`, `/dev/stdout` 및 `/dev/stderr` 파일이 사용 가능해야 합니다.
- 하드링크 및 소프트링크 보호를 활성화해야 합니다. 그렇지 않으면 AWS IoT Greengrass 플래그를 사용하여 안전하지 않은 모드에서만 `-i`를 실행할 수 있습니다.
- 디바이스에서 다음 Linux 커널 구성을 활성화해야 합니다.
  - 네임스페이스:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    커널에서 cgroups를 지원해야 합니다. AWS IoT Greengrass컨테이너로 실행할 때는 다음 요구 사항이 적용됩니다.
    - AWS IoT Greengrass가 Lambda 함수의 메모리 제한을 설정하도록 하려면 메모리 cgroup을 활성화하고 마운트해야 합니다.
    - 로컬 리소스 액세스 권한이 있는 Lambda 함수를 사용하여 AWS IoT Greengrass 코어 디바이스에서 파일을 여는 경우 디바이스 cgroup을 활성화하고 마운트해야 합니다.
  - 기타:
    - CONFIG_POSIX_MQUEUE

  - CONFIG_OVERLAY_FS
  - CONFIG_HAVE_ARCH_SECCOMP_FILTER
  - CONFIG_SECCOMP_FILTER
  - CONFIG_KEYS
  - CONFIG_SECCOMP
  - CONFIG_SHMEM
- Amazon S3 및 AWS IoT의 루트 인증서가 시스템 트러스트 스토어에 있어야 합니다.
- 로컬로 실행할 Lambda 함수에 필요한 AWS Lambda런타임을 지원하는 라이브러리. 필수 라이브러리를 코어에 설치하고 `PATH` 환경 변수에 추가해야 합니다. 동일한 코어에 여러 라이브러리를 설치할 수 있습니다.
  - Python 2.7 실행 시간을 사용하는 기능의 경우 Python 버전 2.7.
  - Python 3.7 실행 시간을 사용하는 기능의 경우 Python 버전 3.7.
  - Node.js 6.10 실행 시간을 사용하는 기능의 경우 Node.js 버전 6.10 이상.
  - Node.js 8.10 실행 시간을 사용하는 기능의 경우 Node.js 버전 8.10 이상.
  - Java 8 런타임을 사용하는 함수의 경우 Java 버전 8 이상

    > **Note**
    >
    > OpenWrt 배포판에서 Java를 실행하는 것은 공식적으로 지원되지 않습니다. 하지만 OpenWrt 빌드에서 Java를 지원하는 경우 Java로 작성된 Lambda 함수를 디바이스에서 실행할 수 있습니다. OpenWrt

  Lambda 런타임용 AWS IoT Greengrass 지원에 대한 자세한 내용은 로컬 Lambda 함수 실행 섹션을 참조하세요.
- (OTA) 업데이트 에이전트에는 다음과 같은 셸 명령 (BusyBox 변형 아님) 이 over-the-air 필요합니다.
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`

- `pidof`
- `df`
- `grep`
- `umount`
- `mv`
- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`

GGC v1.8

- 지원되는 플랫폼:
  - 아키텍처: Armv7l; OS: Linux
  - 아키텍처: x86_64; OS: Linux
  - 아키텍처: Armv8(AArch64); OS: Linux
  - Windows, macOS 및 Linux 플랫폼은 도커 컨테이너에서 AWS IoT Greengrass를 실행할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.
  - Linux 플랫폼은 Greengrass 스냅을 사용하여 AWS IoT Greengrass를 제한된 기능의 버전으로 실행하며, 이 스냅은 Snapcraft에서 구할 수 있습니다. 자세한 설명은 the section called “AWS IoT Greengrass 스냅 소프트웨어” 섹션을 참조하세요.
- 다음과 같은 항목이 필요합니다.
  - AWS IoT Greengrass 코어 소프트웨어에 사용할 수 있는 최소 128MB의 디스크 공간. OTA 업데이트 에이전트를 사용하는 경우 최소 용량은 400MB입니다.
  - AWS IoT Greengrass 코어 소프트웨어에 최소 128MB의 RAM이 할당됩니다.
  - Linux 커널 버전:
    - 컨테이너를 이용한 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 4.4 이상이 필요합니다.

    - 컨테이너 없이 AWS IoT Greengrass 실행을 지원하려면 Linux 커널 버전 3.17 이상이 필요합니다. 이 구성에서 Greengrass 그룹에 대한 기본 Lambda 함수 컨테이너화는 컨테이너 없음으로 설정해야 합니다. 지침은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.
- GNU C Library(glibc) 버전 2.14 이상.
- `/var/run` 디렉터리가 디바이스에 있어야 합니다.
- `/dev/stdin`, `/dev/stdout` 및 `/dev/stderr` 파일이 사용 가능해야 합니다.
- 하드링크 및 소프트링크 보호를 활성화해야 합니다. 그렇지 않으면 AWS IoT Greengrass 플래그를 사용하여 안전하지 않은 모드에서만 `-i`를 실행할 수 있습니다.
- 디바이스에서 다음 Linux 커널 구성을 활성화해야 합니다.
    - 네임스페이스:
        - CONFIG_IPC_NS
        - CONFIG_UTS_NS
        - CONFIG_USER_NS
        - CONFIG_PID_NS
    - Cgroups:
        - CONFIG_CGROUP_DEVICE
        - CONFIG_CGROUPS
        - CONFIG_MEMCG

      커널에서 cgroups를 지원해야 합니다. AWS IoT Greengrass컨테이너로 실행할 때는 다음 요구 사항이 적용됩니다.
        - AWS IoT Greengrass가 Lambda 함수의 메모리 제한을 설정하도록 하려면 메모리 cgroup을 활성화하고 마운트해야 합니다.
        - 로컬 리소스 액세스 권한이 있는 Lambda 함수를 사용하여 AWS IoT Greengrass 코어 디바이스에서 파일을 여는 경우 디바이스 cgroup을 활성화하고 마운트해야 합니다.
    - 기타:
        - CONFIG_POSIX_MQUEUE
        - CONFIG_OVERLAY_FS
        - CONFIG_HAVE_ARCH_SECCOMP_FILTER
        - CONFIG_SECCOMP_FILTER
        - CONFIG_KEYS

        - CONFIG_SECCOMP
        - CONFIG_SHMEM
    - Amazon S3 및 AWS IoT의 루트 인증서가 시스템 트러스트 스토어에 있어야 합니다.
- 조건에 따라 다음 항목이 필요합니다.
    - 로컬로 실행할 Lambda 함수에 필요한 AWS Lambda런타임을 지원하는 라이브러리. 필수 라이브러리를 코어에 설치하고 PATH 환경 변수에 추가해야 합니다. 동일한 코어에 여러 라이브러리를 설치할 수 있습니다.
        - Python 2.7 실행 시간을 사용하는 기능의 경우 Python 버전 2.7.
        - Node.js 6.10 실행 시간을 사용하는 기능의 경우 Node.js 버전 6.10 이상.
        - Java 8 런타임을 사용하는 함수의 경우 Java 버전 8 이상
    - (OTA) 업데이트 에이전트에는 다음과 같은 셸 명령 over-the-air (BusyBox 변형 제외) 이 필요합니다.
        - `wget`
        - `realpath`
        - `tar`
        - `readlink`
        - `basename`
        - `dirname`
        - `pidof`
        - `df`
        - `grep`
        - `umount`
        - `mv`
        - `gzip`
        - `mkdir`
        - `rm`
        - `ln`
        - `cut`
        - `cat`

AWS IoT Greengrass 할당량(한도)에 대한 자세한 내용은 Amazon Web Services 일반 참조의 서비스 할당량을 참조하세요.

요금 정보는 AWS IoT Greengrass 요금 및 AWS IoT Core 요금 섹션을 참조하십시오.

AWS IoT Greengrass 다운로드

다음 정보를 사용하여 AWS IoT Greengrass에 사용할 소프트웨어를 찾아 다운로드할 수 있습니다.

주제

- AWS IoT Greengrass 코어 소프트웨어
- AWS IoT Greengrass 스냅 소프트웨어
- AWS IoT Greengrass Docker 소프트웨어
- AWS IoT Greengrass 코어 SDK
- 지원되는 기계 학습 런타임 및 라이브러리
- AWS IoT Greengrass ML SDK 소프트웨어

AWS IoT Greengrass 코어 소프트웨어

AWS IoT Greengrass 코어 소프트웨어는 AWS 기능을 AWS IoT Greengrass 코어 디바이스로 확장하므로, 로컬 디바이스가 생성하는 데이터에 대해 로컬 작업을 수행할 수 있습니다.

v1.11

1.11.6

버그 수정 및 개선 사항:

- 배포 중에 갑작스러운 전력 손실이 발생할 경우 복원력이 향상되었습니다.
- 스트림 관리자 데이터 손상으로 인해 AWS IoT Greengrass 코어 소프트웨어가 시작되지 않는 문제를 수정했습니다.
- 특정 시나리오에서 새 클라이언트 장치를 코어에 연결할 수 없는 문제를 수정했습니다.
- 스트림 관리자 스트림 이름이 `.log`를 포함할 수 없던 문제를 수정했습니다.

1.11.5

버그 수정 및 개선 사항:

- 일반 성능 향상 및 버그 수정.

1.11.4

버그 수정 및 개선 사항:

- AWS IoT Greengrass 코어 소프트웨어 v1.11.3으로의 업그레이드를 방해하는 스트림 관리자 관련 문제를 수정했습니다. 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 이제 OTA 업데이트를 사용하여 이전 v1.x 버전의 AWS IoT Greengrass Core 소프트웨어를 v1.11.4로 업그레이드할 수 있습니다.
- 일반 성능 향상 및 버그 수정.

1.11.3

버그 수정 및 개선 사항:

- Ubuntu 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 바로 실행하던 중 디바이스의 전원이 갑자기 끊긴 후 응답하지 않던 문제를 수정했습니다.
- 수명이 긴 Lambda 함수에 MQTT 메시지 전송이 지연되는 문제를 수정했습니다.
- `maxWorkItemCount` 값이 `1024`보다 큰 값으로 설정된 경우 MQTT 메시지가 제대로 전송되지 않던 문제를 수정했습니다.
- OTA 업데이트 에이전트가 `config.json`에서 `keepAlive` 속성에 지정된 MQTT `KeepAlive` 기간을 무시하던 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

> **Important**
>
> 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 이전 v1.x 버전에서 AWS IoT Greengrass 코어 소프트웨어 v1.11.3으로 업그레이드하지 마십시오. 스트림 관리자를 처음 활성화하는 경우 먼저 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 설치하는 것이 좋습니다.

1.11.1

버그 수정 및 개선 사항:

- 스트림 관리자의 메모리 사용량이 증가하는 문제가 수정되었습니다.
- 스트림 데이터의 지정된 기간 time-to-live (TTL) 보다 오랫동안 Greengrass 코어 디바이스의 전원이 꺼진 `0` 경우 스트림 관리자가 스트림의 시퀀스 번호를 재설정하던 문제를 수정했습니다.

- 스트림 관리자가 AWS 클라우드로 데이터를 내보내려는 재시도를 제대로 중단하지 못하던 문제를 수정했습니다.

1.11.0

새로운 기능:

- Greengrass 코어의 텔레메트리 에이전트는 로컬 텔레메트리 데이터를 수집하여 AWS 클라우드에 게시합니다. 추가 처리를 위해 원격 분석 데이터를 검색하려는 고객은 Amazon EventBridge 규칙을 생성하고 대상을 구독할 수 있습니다. 자세한 내용은 AWS IoT Greengrass 코어 디바이스에서 시스템 상태 텔레메트리 데이터 수집을 참조하십시오.
- AWS IoT Greengrass에서 시작된 로컬 작업자 프로세스의 현재 상태를 스냅샷으로 제공하는 로컬 HTTP API가 포함되어 있습니다. 자세한 내용은 로컬 상태 확인 API 호출을 참조하십시오.
- 스트림 관리자는 Amazon S3 및 AWS IoT SiteWise로 데이터를 자동으로 내보냅니다.

  새 스트림 관리자 파라미터를 사용하면 기존 스트림을 업데이트하고 데이터 내보내기를 일시 중지하거나 재개할 수 있습니다.
- 코어에서 Python 3.8.x Lambda 함수를 실행하기 위한 지원.
- Greengrass 코어 IPC 포트 번호를 구성하는 데 사용하는 `config.json`의 새 `ggDaemonPort` 속성입니다. 기본 포트 번호는 8000입니다.

  Greengrass 코어 IPC 인증에 대한 제한 시간을 구성하는 데 사용하는 `config.json`의 새 `systemComponentAuthTimeout` 속성입니다. 기본값은 5,000밀리초입니다.
- AWS IoT Greengrass 그룹당 최대 AWS IoT 디바이스 수를 200개에서 2500개로 늘렸습니다.

  그룹당 최대 구독 수를 1000개에서 10000개로 늘렸습니다.

  자세한 내용은 AWS IoT Greengrass 엔드포인트 및 할당량을 참조하십시오.

버그 수정 및 개선 사항:

- Greengrass 서비스 프로세스의 메모리 사용률을 줄일 수 있는 일반 최적화.
- 새로운 런타임 구성 파라미터(`mountAllBlockDevices`)를 사용하면 Greengrass가 OverlayFS를 설정한 후 바인드 마운트를 사용하여 모든 블록 디바이스를 컨테이너에 마운트할 수 있습니다. 이 기능은 `/usr`이(가) `/` 계층 구조 아래에 있지 않을 경우 Greengrass 배포가 실패하는 문제를 해결했습니다.
- `/tmp`이 심볼릭 링크인 경우 AWS IoT Greengrass 코어 오류가 발생하는 문제를 수정했습니다.

- Greengrass 배포 에이전트가 `mlmodel_public` 폴더에서 사용하지 않는 기계 학습 모델 아티팩트를 제거하도록 하는 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 아키텍처 및 OS(운영 체제)를 위한 패키지를 다운로드한 후 시작하기 안내서의 단계를 따릅니다.

> **Tip**
>
> AWS IoT Greengrass에서는 다른 방법으로도 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 예를 들어 Greengrass 디바이스 설정을 사용하여 환경을 구성하고 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 또는 지원되는 Debian 플랫폼에서 APT 패키지 관리자를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 업그레이드할 수 있습니다. 자세한 설명은 the section called “AWS IoT Greengrass 코어 소프트웨어 설치” 섹션을 참조하세요.

아키텍처	운영 체제	링크
ARMv8(AArch64)	Linux	다운로드
ARMv8(AArch64)	Linux () OpenWrt	다운로드
Armv7l	Linux	다운로드
Armv7l	리눅스 (OpenWrt)	다운로드
Armv6l	Linux	다운로드
x86_64	Linux	다운로드

Extended life versions

1.10.5

v1.10의 새로운 기능:

- 데이터 스트림을 로컬에서 처리하고 이를 AWS 클라우드로 자동으로 내보내는 스트림 관리자. 이 기능을 사용하려면 Greengrass 코어 디바이스에 Java 8이 필요합니다. 자세한 설명은 데이터 스트림 관리 섹션을 참조하세요.
- 코어 디바이스에서 Docker 애플리케이션을 실행하는 새로운 Greengrass Docker 애플리케이션 배포 커넥터. 자세한 설명은 the section called “Docker 애플리케이션 배포” 섹션을 참조하세요.
- 산업용 장치 데이터를 OPC-UA 서버의 자산 자산으로 전송하는 새로운 IoT SiteWise 커넥터. AWS IoT SiteWise 자세한 설명은 the section called “IoT SiteWise” 섹션을 참조하세요.
- 컨테이너화 없이 실행되는 Lambda 함수는 Greengrass 그룹의 기계 학습 리소스에 액세스할 수 있습니다. 자세한 설명은 the section called “기계 학습 리소스에 액세스” 섹션을 참조하세요.
- AWS IoT를 통해 MQTT 영구 세션 지원. 자세한 설명은 the section called “AWS IoT Core를 사용하는 MQTT 영구 세션” 섹션을 참조하세요.
- 로컬 MQTT 트래픽은 기본 포트 8883 이외의 포트를 통해 이동할 수 있습니다. 자세한 설명은 the section called “로컬 메시징을 위한 MQTT 포트” 섹션을 참조하세요.
- Lambda 함수에서 안정적인 메시지 게시를 위한 AWS IoT Greengrass 코어 SDK의 새로운 `queueFullPolicy` 옵션.
- 코어에서 Node.js 12.x Lambda 함수를 실행하기 위한 지원.

버그 수정 및 개선 사항:

- 하드웨어 보안 통합을 통한 O ver-the-air (OTA) 업데이트는 OpenSSL 1.1을 사용하여 구성할 수 있습니다.
- 스트림 관리자는 파일 데이터 손상에 대한 복원력이 더 뛰어납니다.
- Linux 커널 5.1 이상을 사용하는 디바이스에 sysfs를 탑재하지 못하는 문제를 수정했습니다.
- AWS IoT Core와 MQTT 연결에서 게시, 구독 및 구독 취소 작업에 대한 제한 시간을 설정하는 데 사용하는 config.json의 새 `mqttOperationTimeout` 속성입니다.
- 스트림 관리자의 메모리 사용량이 증가하는 문제가 수정되었습니다.
- Greengrass 코어 IPC 인증에 대한 제한 시간을 구성하는 데 사용하는 `config.json`의 새 `systemComponentAuthTimeout` 속성입니다. 기본값은 5,000밀리초입니다.
- OTA 업데이트 에이전트가 `config.json`에서 `keepAlive` 속성에 지정된 MQTT `KeepAlive` 기간을 무시하던 문제를 수정했습니다.
- `maxWorkItemCount` 값이 `1024`보다 큰 값으로 설정된 경우 MQTT 메시지가 제대로 전송되지 않던 문제를 수정했습니다.

- 수명이 긴 Lambda 함수에 MQTT 메시지 전송이 지연되는 문제를 수정했습니다.
- Ubuntu 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 바로 실행하던 중 디바이스의 전원이 갑자기 끊긴 후 응답하지 않던 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 아키텍처 및 OS(운영 체제)를 위한 패키지를 다운로드한 후 시작하기 안내서의 단계를 따릅니다.

아키텍처	운영 체제	링크
ARMv8(AArch64)	Linux	다운로드
ARMv8(AArch64)	리눅스 () OpenWrt	다운로드
Armv7l	Linux	다운로드
Armv7l	리눅스 (OpenWrt)	다운로드
Armv6l	Linux	다운로드
x86_64	Linux	다운로드

1.9.4

v1.9의 새로운 기능:

- Python 3.7 및 Node.js 8.10 Lambda 런타임을 지원합니다. Python 3.7 및 Node.js 8.10 런타임을 사용하는 Lambda 함수가 이제 AWS IoT Greengrass 코어에서 실행할 수 있습니다(AWS IoT Greengrass는 Python 2.7 및 Node.js 6.10 런타임을 계속 지원함).
- MQTT 연결을 최적화했습니다. Greengrass 코어는 AWS IoT Core와의 더 적은 수의 연결을 설정합니다. 이 변경에 따라 연결 수를 기반으로 부과되는 요금의 운영 비용을 절감할 수 있습니다.
- 로컬 MQTT 서버에 EC(Elliptic Curve) 키를 사용할 수 있습니다. 로컬 MQTT 서버는 RSA 키 외에도 EC 키를 지원합니다. 키 유형과 상관없이 MQTT 서버 인증서에는 SHA-256 RSA 서명이 있습니다. 자세한 설명은 the section called “보안 주체” 섹션을 참조하세요.
- 에 대한 지원 OpenWrt. AWS IoT Greengrass 코어 소프트웨어 v1.9.2 이상은 Armv8(AArch64) 및 ARMv7L OpenWrt 아키텍처를 사용하는 배포판에 설치할 수 있습니다. OpenWrt 현재는 ML 추론을 지원하지 않습니다.

- Armv6l에 대한 지원. AWS IoT Greengrass Armv6l 아키텍처의 Raspbian 배포(예: Raspberry Pi Zero 디바이스)에 코어 소프트웨어 v1.9.3 이상을 설치할 수 있습니다.
- ALPN을 통한 포트 443의 OTA 업데이트. MQTT 트래픽에 포트 443을 사용하는 Greengrass 코어는 이제 over-the-air (OTA) 소프트웨어 업데이트를 지원합니다. AWS IoT Greengrass애플리케이션 계층 프로토콜 네트워크 (ALPN) TLS 확장을 사용하여 이러한 연결을 활성화합니다. 자세한 내용은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 및 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 아키텍처 및 OS(운영 체제)를 위한 패키지를 다운로드한 후 시작하기 안내서의 단계를 따릅니다.

아키텍처	운영 체제	링크
ARMv8(AArch64)	Linux	다운로드
ARMv8(AArch64)	리눅스 () OpenWrt	다운로드
Armv7l	Linux	다운로드
Armv7l	리눅스 (OpenWrt)	다운로드
Armv6l	Linux	다운로드
x86_64	Linux	다운로드

1.8.4

- 새로운 기능:
  - 그룹의 Lambda 함수에 대해 구성 가능한 기본 액세스 자격 증명. 이 그룹 레벨 설정은 Lambda 함수를 실행하는 데 사용된 기본 권한을 결정합니다. 사용자 ID, 그룹 ID 또는 둘 다 설정할 수 있습니다. 개별 Lambda 함수는 해당 그룹의 기본 액세스 자격 증명을 재정의할 수 있습니다. 자세한 설명은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.
  - 포트 443을 통한 HTTPS 트래픽 HTTPS 통신은 기본 포트 8443 대신에 포트 443을 통한 이동을 위해 구성됩니다. 이는 Application Layer Protocol Network (ALPN) TLS 확장에 대한 AWS IoT Greengrass 지원을 보완하고 모든 Greengrass 메시징 트래픽(MQTT 및 HTTPS 모두)이 포트 443을 사용할 수 있도록 합니다. 자세한 설명은 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.

- AWS IoT 연결에 대해 예상 가능한 이름의 클라이언트 ID 이 변경에 따라 AWS IoT Device Defender 및 AWS IoT 수명 주기 이벤트가 지원되므로 연결, 연결 끊기, 구독 및 구독 해제 이벤트에 대한 알림을 받을 수 있습니다. 또한 예상 가능한 이름을 지정하면 연결 ID를 중심으로 논리를 쉽게 만들 수 있습니다(예: 인증서 속성을 기준으로 구독 정책 템플릿을 만드는 경우). 자세한 설명은 the section called “AWS IoT를 통한 MQTT 연결용 클라이언트 ID” 섹션을 참조하세요.

버그 수정 및 개선 사항:

- 섀도우 동기화 및 디바이스 인증서 관리자 재연결 문제를 수정했습니다.
- 일반 성능 향상 및 버그 수정.

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 아키텍처 및 OS(운영 체제)를 위한 패키지를 다운로드한 후 시작하기 안내서의 단계를 따릅니다.

아키텍처	운영 체제	링크
ARMv8(AArch64)	Linux	다운로드
Armv7l	Linux	다운로드
x86_64	Linux	다운로드

이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하는 다른 방법에 대한 자세한 내용은 the section called “AWS IoT Greengrass 코어 소프트웨어 설치” 섹션을 참조하십시오.

AWS IoT Greengrass 스냅 소프트웨어

AWS IoT Greengrass snap 1.11.x를 사용하면 컨테이너화된 환경에서 필요한 모든 종속성과 함께 편리한 소프트웨어 패키지를 사용하여 제한된 버전의 AWS IoT Greengrass를 실행할 수 있습니다.

> **Note**
>
> AWS IoT Greengrass snap은 AWS IoT Greengrass 코어 소프트웨어 v1.11.x에서 사용할 수 있습니다. AWS IoT Greengrass는 v1.10.x에 snap을 제공하지 않습니다. 지원되지 않는 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다.

AWS IoT Greengrass snap은 커넥터 및 기계 학습(ML) 추론을 지원하지 않습니다.

자세한 설명은 the section called “스냅에서 AWS IoT Greengrass 실행” 섹션을 참조하세요.

AWS IoT Greengrass Docker 소프트웨어

AWS는 Docker 컨테이너에서 AWS IoT Greengrass를 쉽게 실행할 수 있도록 Dockerfile과 도커 이미지를 제공합니다.

Dockerfile

Dockerfile에는 사용자 지정 AWS IoT Greengrass 컨테이너 이미지를 빌드하기 위한 소스 코드가 포함되어 있습니다. 다른 플랫폼 아키텍처에서 실행하거나 이미지 크기를 줄이기 위해 이미지를 수정할 수 있습니다. 지침은 README 파일을 참조하세요.

대상 AWS IoT Greengrass Core 소프트웨어 버전을 다운로드합니다.

v1.11

- AWS IoT Greengrass v1.11.6용 Dockerfile.

Extended life versions

v1.10

AWS IoT Greengrass v1.10.5용 Dockerfile.

v1.9

AWS IoT Greengrass v1.9.4 용 Dockerfile.

v1.8

AWS IoT Greengrass v1.8.1용 Dockerfile.

도커 이미지

도커 이미지에는 Amazon Linux 2(x86_64) 및 Alpine Linux(x86_64, Armv7l 또는 AArch64) 기본 이미지에 설치된 AWS IoT Greengrass Core 소프트웨어 및 종속성이 포함되어 있습니다. 미리 빌드된 이미지를 사용하여 AWS IoT Greengrass 실험을 시작할 수 있습니다.

> **Important**
>
> 2022년 6월 30일에 AWS IoT Greengrass는 Amazon Elastic Container Registry(Amazon ECR) 및 Docker Hub에 게시된 AWS IoT Greengrass 코어 소프트웨어 v1.x 도커 이미지에 대한 유지 관리를 종료했습니다. 유지 관리 종료 후 1년이 되는 2023년 6월 30일까지 Amazon ECR 및 Docker Hub에서 이 Docker 이미지를 계속 다운로드할 수 있습니다. 그러나 AWS IoT Greengrass Core 소프트웨어 v1.x Docker 이미지는 2022년 6월 30일에 유지 관리가 종료된 이후 더 이상 보안 패치나 버그 수정을 받지 않습니다. 이러한 Docker 이미지를 사용하는 프로덕션 워크로드를 실행하는 경우 AWS IoT Greengrass이(가) 제공하는 Dockerfile을 사용하여 자체 Docker 이미지를 구축하는 것이 좋습니다. 자세한 설명은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

Docker Hub 또는 Amazon Elastic Container Registry(Amazon ECR)에서 이미지를 다운로드할 수 있습니다.

- Docker Hub의 경우 *##* 태그를 사용하여 Greengrass Docker 이미지의 특정 버전을 다운로드할 수 있습니다. 사용 가능한 모든 이미지에 대한 태그를 찾으려면 Docker Hub의 태그 페이지를 확인하십시오.
- Amazon ECR의 경우 `latest` 태그를 사용하여 사용 가능한 최신 버전의 Greengrass Docker 이미지를 다운로드할 수 있습니다. 사용 가능한 이미지 버전을 나열하고 Amazon ECR에서 이미지를 다운로드하는 방법에 대한 자세한 내용은 도커 컨테이너에서의 AWS IoT Greengrass 실행 섹션을 참조하세요.

> **Warning**
>
> AWS IoT Greengrass코어 소프트웨어 v1.11.6부터 Greengrass Docker 이미지에는 더 이상 Python 2.7이 포함되지 않습니다. Python 2.7이 end-of-life 2020년에 출시되었고 더 이상 보안 업데이트를 받지 않았기 때문입니다. 이러한 Docker 이미지로 업데이트하기로 선택한 경우, 업데이트를 프로덕션 디바이스에 배포하기 전에 애플리케이션이 새 Docker 이미지와 호환되는지 확인하는 것을 권장합니다. Greengrass Docker 이미지를 사용하는 애플리케이션에 Python 2.7이 필요한 경우, Python 2.7을 애플리케이션에 포함하도록 Greengrass Dockerfile을 수정할 수 있습니다.

AWS IoT Greengrass는 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.1의 Docker 이미지를 제공하지 않습니다.

> **Note**
>
> 기본적으로 `alpine-aarch64` 및 `alpine-armv7l` 이미지는 Arm 기반 호스트에서만 실행할 수 있습니다. x86 호스트에서 이러한 이미지를 실행하려면 QEMU를 설치하고 호스트에 QEMU 라이브러리를 탑재할 수 있습니다. 예:
>
> ```
> docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
> ```

AWS IoT Greengrass 코어 SDK

Lambda 함수는 AWS IoT Greengrass 코어 SDK를 사용해 로컬에서 AWS IoT Greengrass와 상호 작용합니다. 이를 통해 배포된 Lambda 함수로 다음 작업이 가능합니다.

- MQTT 메시지를 AWS IoT Core와 교환합니다.
- MQTT 메시지를 Greengrass 그룹의 커넥터, 디바이스, 기타 Lambda 함수와 교환합니다.
- 로컬 섀도우 서비스와 상호 작용합니다.
- 다른 로컬 Lambda 함수를 간접적으로 호출합니다.
- 암호 리소스에 액세스합니다.
- 스트림 관리자와 상호 작용합니다.

에서 해당 언어 또는 플랫폼에 맞는 AWS IoT Greengrass Core SDK를 다운로드하십시오. GitHub

- Java용 AWS IoT Greengrass 코어 SDK
- Node.js용 AWS IoT Greengrass 코어 SDK
- AWS IoT Greengrass Python용 코어 SDK
- AWS IoT Greengrass Core SDK for C

자세한 설명은 AWS IoT Greengrass 코어 SDK 섹션을 참조하세요.

지원되는 기계 학습 런타임 및 라이브러리

Greengrass 코어에 대한 추론을 수행하려면 ML 모델 유형에 맞는 기계 학습 런타임 또는 라이브러리를 설치해야 합니다.

AWS IoT Greengrass는 다음과 같은 ML 모델 유형을 지원합니다. 다음 링크에서 모델 유형 및 디바이스 플랫폼에 맞는 런타임 또는 라이브러리를 설치하는 방법에 대한 정보를 찾을 수 있습니다.

- 딥 러닝 런타임(DLR)
- MXNet
- TensorFlow

기계 학습 샘플

AWS IoT Greengrass에서는 지원되는 ML 런타임 및 라이브러리와 함께 사용할 수 있는 샘플을 제공합니다. 이러한 샘플은 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Deep learning runtime (DLR)

사용 중인 디바이스 플랫폼에 해당하는 샘플을 다운로드합니다.

- Raspberry Pi용 DLR 샘플
- NVIDIA Jetson TX2용 DLR 샘플
- Intel Atom용 DLR 샘플

DLR 샘플을 사용하는 튜토리얼은 the section called “최적화된 기계 학습 추론을 구성하는 방법” 섹션을 참조하십시오.

MXNet

사용 중인 디바이스 플랫폼에 해당하는 샘플을 다운로드합니다.

- Raspberry Pi용 MXNet 샘플
- NVIDIA Jetson TX2용 MXNet 샘플
- Intel Atom용 MXNet 샘플

MXNet 샘플을 사용하는 튜토리얼은 the section called “기계 학습 추론을 구성하는 방법” 섹션을 참조하십시오.

TensorFlow

디바이스 플랫폼용 TensorFlow 샘플을 다운로드하십시오. 이 샘플은 Raspberry Pi, NVIDIA Jetson TX2 및 Intel Atom과 함께 작동합니다.

AWS IoT Greengrass ML SDK 소프트웨어

AWS IoT Greengrass기계 학습 SDK을 사용하면 사용자가 작성하는 Lambda 함수가 로컬 기계 학습 모델을 사용하고 업로드 및 게시를 위해 ML 피드백 커넥터로 데이터를 전송할 수 있습니다.

v1.1.0

- Python 3.7.

v1.0.0

- Python 2.7.

연락을 기다리겠습니다.

우리는 여러분의 의견을 환영합니다. 문의하려면 AWS re:Post를 방문하여 AWS IoT Greengrass태그를 사용하십시오.

AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass 코어 소프트웨어는 AWS 기능을 AWS IoT Greengrass 코어 디바이스로 확장하므로, 로컬 디바이스가 생성하는 데이터에 대해 로컬 작업을 수행할 수 있습니다.

AWS IoT Greengrass에서는 다음 몇 가지 방법으로 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다.

- tar.gz 파일 다운로드 및 압축 해제
- Greengrass 디바이스 설정 스크립트 실행
- APT 리포지토리에서 설치

또한 AWS IoT Greengrass는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 컨테이너화된 환경을 제공합니다.

- 도커 컨테이너에서 AWS IoT Greengrass 실행
- 스냅에서 AWS IoT Greengrass 실행

AWS IoT Greengrass 코어 소프트웨어 패키지 다운로드 및 추출

플랫폼에 맞는 AWS IoT Greengrass 코어 소프트웨어를 선택하여 tar.gz 파일로 다운로드하고 디바이스에서 압축을 풀 수 있습니다. 최신 버전의 소프트웨어를 다운로드할 수 있습니다. 자세한 설명은 the section called “AWS IoT Greengrass 코어 소프트웨어” 섹션을 참조하세요.

Greengrass 디바이스 설정 스크립트 실행

Greengrass 디바이스 설정을 실행하여 디바이스를 구성하고 최신 AWS IoT Greengrass 코어 소프트웨어 버전을 설치하고 몇 분 안에 Hello World Lambda 함수를 배포할 수 있습니다. 자세한 설명은 the section called “빠른 시작: Greengrass 디바이스 설정” 섹션을 참조하세요.

APT 리포지토리에서 AWS IoT Greengrass 코어 소프트웨어 설치

> **Important**
>
> 2022년 2월 11일부터 더 이상 APT 저장소에서 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 업데이트할 수 없습니다. AWS IoT Greengrass 리포지토리를 추가한 디바이스에서는 소스 목록에서 리포지토리를 제거해야 합니다. APT 리포지토리에서 소프트웨어를 실행하는 기기는 계속 정상적으로 작동합니다. tar 파일을 사용하여 AWS IoT Greengrass 코어 소프트웨어를 업데이트하는 것이 좋습니다.

AWS IoT Greengrass에서 제공하는 APT 리포지토리에는 다음 패키지가 포함되어 있습니다.

- `aws-iot-greengrass-core`은(는) AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

- `aws-iot-greengrass-keyring`은(는) AWS IoT Greengrass 패키지 리포지토리에 서명하는 데 사용되는 GnuPG(GPG) 키를 설치합니다.

  이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

주제

- systemd 스크립트를 사용하여 Greengrass 대몬(daemon) 수명 주기 관리
- APT 리포지토리를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 제거합니다.
- AWS IoT Greengrass 코어 소프트웨어 리포지토리 소스를 제거합니다.

systemd 스크립트를 사용하여 Greengrass 대몬(daemon) 수명 주기 관리

`aws-iot-greengrass-core` 패키지는 AWS IoT Greengrass 코어 소프트웨어(대몬(daemon)) 수명 주기를 관리하는 데 사용할 수 있는 `systemd` 스크립트도 설치합니다.

- 부팅 중에 Greengrass 대몬(daemon):

```
systemctl enable greengrass.service
```

- Greengrass 대몬(daemon)을 시작하려면:

```
systemctl start greengrass.service
```

- Greengrass 대몬(daemon)을 중지하려면:

```
systemctl stop greengrass.service
```

- Greengrass 데몬의 상태를 확인하려면:

```
systemctl status greengrass.service
```

APT 리포지토리를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 제거합니다.

AWS IoT Greengrass 코어 소프트웨어를 제거할 때 디바이스 인증서, 그룹 정보, 로그 파일 등 AWS IoT Greengrass 코어 소프트웨어의 구성 정보를 보존할지 제거할지를 선택할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어를 제거하고 구성 정보를 보존하려면

- 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 패키지를 제거하고 `/greengrass` 폴더에 구성 정보를 보존합니다.

```
sudo apt remove aws-iot-greengrass-core aws-iot-greengrass-keyring
```

AWS IoT Greengrass 코어 소프트웨어를 제거하고 구성 정보를 제거하려면

1. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 패키지를 제거하고 `/greengrass folder`에서 구성 정보를 제거합니다.

```
sudo apt purge aws-iot-greengrass-core aws-iot-greengrass-keyring
```

2. 소스 목록에서 AWS IoT Greengrass 코어 소프트웨어 리포지토리를 제거합니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어 리포지토리 소스를 제거합니다. 섹션을 참조하세요.

AWS IoT Greengrass 코어 소프트웨어 리포지토리 소스를 제거합니다.

APT 리포지토리에서 AWS IoT Greengrass 코어 소프트웨어를 더 이상 설치하거나 업데이트할 필요가 없는 경우, AWS IoT Greengrass 코어 소프트웨어 리포지토리 소스를 제거할 수 있습니다. 2022년 2월 11일 이후에는 `apt update`을(를) 실행 시 오류가 발생하지 않도록 소스 목록에서 리포지토리를 제거해야 합니다.

소스 목록에서 APT 리포지토리를 제거하려면

- 다음 명령을 실행하여 소스 목록에서 AWS IoT Greengrass 코어 소프트웨어 리포지토리를 제거합니다.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

Docker 컨테이너에서 AWS IoT Greengrass을(를) 실행합니다.

AWS IoT Greengrass은(는) Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어를 쉽게 실행할 수 있도록 Dockerfile과 Docker 이미지를 제공합니다. 자세한 설명은 the section called “AWS IoT Greengrass Docker 소프트웨어” 섹션을 참조하세요.

> **Note**
>
> Greengrass 코어 디바이스에서 Docker 애플리케이션을 실행할 수도 있습니다. 이렇게 하려면 Greengrass Docker 애플리케이션 배포 커넥터를 사용하십시오

스냅에서 AWS IoT Greengrass 실행

AWS IoT Greengrass snap 1.11.x를 사용하면 컨테이너화된 환경에서 필요한 모든 종속성과 함께 편리한 소프트웨어 패키지를 사용하여 제한된 버전의 AWS IoT Greengrass를 실행할 수 있습니다.

2023년 12월 31일에 AWS IoT Greengrass이(가) snapcraft.io에 게시된 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.x Snap에 대한 유지 관리를 종료합니다. 현재 Snap을 실행하는 기기는 추후 공지가 있을 때까지 계속 작동합니다. 그러나 유지 관리 종료 후에는 AWS IoT Greengrass 코어 Snap에 더 이상 보안 패치나 버그 수정이 제공되지 않습니다.

스냅 개념

다음은 AWS IoT Greengrass 스냅 사용 방법을 이해하는 데 도움이 되는 필수 스냅 개념입니다.

Channel

업데이트를 위해 설치 및 추적되는 스냅 버전을 정의하는 스냅 구성 요소입니다. 스냅은 현재 채널의 최신 버전으로 자동 업데이트됩니다.

인터페이스

네트워크 및 사용자 파일과 같은 리소스에 대한 액세스 권한을 부여하는 스냅 구성 요소입니다.

AWS IoT Greengrass 스냅을 실행하려면 다음 인터페이스가 연결되어 있어야 합니다. `greengrass-support-no-container`이(가) 먼저 연결되어야 하고 연결이 끊어지면 안 됩니다.

```
- greengrass-support-no-container
- hardware-observe
- home-for-hooks
- hugepages-control
- log-observe
- mount-observe
- network
- network-bind
```

```
    - network-control
    - process-control
    - system-observe
```

다른 인터페이스는 선택 사항입니다. Lambda 함수에 특정 리소스에 대한 액세스가 필요한 경우, 적절한 인터페이스에 연결해야 할 수 있습니다.

새로 고침

스냅은 자동으로 업데이트됩니다. snapd 대몬(daemon)은 기본적으로 하루에 네 번 업데이트를 확인하는 스냅 패키지 관리자입니다. 각 업데이트 검사를 새로 고침이라고 합니다. 새로 고침이 발생하면 대몬(daemon)이 중지되고 스냅이 업데이트된 다음 대몬(daemon)이 다시 시작됩니다.

자세한 내용은 Snapcraft 웹 사이트를 참조하세요.

AWS IoT Greengrass 스냅 v1.11.x의 새로운 기능

다음은 AWS IoT Greengrass 스냅 버전 1.11.x의 새로운 기능과 변경된 기능에 대한 설명입니다.

- 이 버전은 사용자 ID(UID)으로 표시되는 snap_daemon 사용자 및 그룹(GID) 584788만 지원합니다.
- 이 버전은 컨테이너화되지 않은 Lambda 함수만 지원합니다.

> **Important**
>
> 컨테이너화되지 않은 Lambda 함수는 동일한 사용자(snap_daemon)를 공유해야 하므로 Lambda 함수는 서로 격리되지 않습니다. 자세한 내용은 그룹별 구성을 사용한 Greengrass Lambda 함수의 실행 제어를 참조하십시오.

- 이 버전은 C, C++, Java 8, Node.js 12.x, Python 2.7, Python 3.7, Python 3.8 런타임을 지원합니다.

> **Note**
>
> 중복 Python 런타임을 방지하기 위해 Python 3.7 Lambda 함수는 실제로 Python 3.8 런타임을 실행합니다.

AWS IoT Greengrass 스냅 시작하기

다음 절차는 디바이스에 AWS IoT Greengrass 스냅을 설치하고 구성하는 데 도움이 됩니다.

요구 사항

AWS IoT Greengrass 스냅을 실행하려면 다음을 수행해야 합니다.

- 지원되는 Linux 배포판(예: Ubuntu, Linux Mint, Debian, Fedora)에서 AWS IoT Greengrass 스냅을 실행합니다.
- 디바이스에 snapd 대몬(daemon)을 설치합니다. snap 도구가 포함된 snapd 대몬(daemon)은 디바이스의 스냅 환경을 관리합니다.

지원되는 Linux 배포판 목록과 설치 지침은 스냅 설명서의 snapd 설치를 참조하세요.

AWS IoT Greengrass 스냅 설치 및 구성

다음 자습서에서는 디바이스에 AWS IoT Greengrass 스냅을 설치하고 구성하는 방법을 보여줍니다.

> **Note**
>
> - 이 자습서에서는 Amazon EC2 인스턴스(x86 t2.micro Ubuntu 20.04)를 사용하지만 Raspberry Pi와 같은 물리적 하드웨어로 AWS IoT Greengrass 스냅을 실행할 수 있습니다.
> - snapd 대몬(daemon)은 Ubuntu에 사전 설치되어 있습니다.

1. 디바이스 터미널에서 다음 명령을 실행하여 core18 스냅을 설치합니다.

   ```
   sudo snap install core18
   ```

   core18 스냅은 일반적으로 사용되는 라이브러리가 포함된 런타임 환경을 제공하는 기본 스냅입니다. 이 스냅은 Ubuntu 18.04 LTS에서 만들어졌습니다.
2. 다음 명령을 실행하여 snapd을(를) 업그레이드합니다.

   ```
   sudo snap install --channel=edge snapd; sudo snap refresh --channel=edge snapd
   ```
3. snap list 명령을 실행하여 AWS IoT Greengrass 스냅이 설치되어 있는지 확인합니다.

   다음 예제 응답은 snapd은(는) 설치되었지만 aws-iot-greengrass은(는) 설치되지 않았음을 보여줍니다.

   ```
   Name              Version              Rev    Tracking         Publisher   Notes
   amazon-ssm-agent  3.0.161.0            2996   latest/stable/…  aws#        classic
   ```

```
core              16-2.48              10444  latest/stable    canonical#  core
core18            20200929             1932   latest/stable    canonical#  base
lxd               4.0.4                18150  4.0/stable/…     canonical#  -
snapd             2.48+git548.g929ccfb 10526  latest/edge      canonical#  snapd
```

4. 다음 옵션 중 하나를 선택하여 AWS IoT Greengrass 스냅 1.11.x를 설치합니다.

   - 다음 명령을 실행하여 AWS IoT Greengrass 스냅을 설치합니다.

     ```
     sudo snap install aws-iot-greengrass
     ```

     응답의 예:

     ```
     aws-iot-greengrass 1.11.5 from Amazon Web Services (aws) installed
     ```

   - 이전 버전에서 v1.11.x로 마이그레이션하거나 사용 가능한 최신 패치 버전으로 업데이트하려면 다음 명령을 실행합니다.

     ```
     sudo snap refresh --channel=1.11.x aws-iot-greengrass
     ```

   다른 스냅과 마찬가지로 AWS IoT Greengrass 스냅은 채널을 사용하여 마이너 버전을 관리합니다. 스냅은 현재 채널의 사용 가능한 최신 버전으로 자동 업데이트됩니다. 예를 들어, `--channel=1.11.x`을(를) 지정하면 AWS IoT Greengrass 스냅이 v1.11.5로 업데이트됩니다.

   `snap info aws-iot-greengrass` 명령을 실행하여 AWS IoT Greengrass에 사용 가능한 채널 목록을 가져올 수 있습니다.

   응답의 예:

   ```
   name:      aws-iot-greengrass
   summary:   AWS supported software that extends cloud capabilities to local devices.
   publisher: Amazon Web Services (aws#)
   store-url: https://snapcraft.io/aws-iot-greengrass
   contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
   license:   Proprietary
   description: |
     AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
    locally on the data
     they generate, while still using the cloud for management, analytics, and durable
    storage.
   ```

```
  AWS IoT Greenrgrass snap v1.11.0 enables you to run a limited version of AWS IoT
 Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
 inference.
  By downloading this software you agree to the Greengrass Core Software License
 Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
ggc.html#gg-snap-support) in
  the AWS IoT Greengrass Developer.
  If you need help, try the AWS IoT Greengrass tag on AWS re:Post
  (https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
 with an AWS IQ expert
  (https://iq.aws.amazon.com/services/aws/greengrass).
snap-id: SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
channels:
  latest/stable:    1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
  latest/beta:      1.11.3 2021-06-14 (59) 111MB -
  latest/edge:      1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. Lambda 함수에 필요한 특정 리소스에 액세스하기 위해 추가 인터페이스에 연결할 수 있습니다.

   다음 명령을 실행하여 AWS IoT Greengrass 스냅이 지원되는 인터페이스 목록을 파악합니다.

   ```
   snap connections aws-iot-greengrass
   ```

   응답의 예:

   ```
   Interface               Plug                                        Slot
               Notes
   camera                  aws-iot-greengrass:camera                   -
               -
   dvb                     aws-iot-greengrass:dvb                      -
               -
   ```

```
gpio                    aws-iot-greengrass:gpio                              -
               -
gpio-memory-control     aws-iot-greengrass:gpio-memory-control               -
               -
greengrass-support      aws-iot-greengrass:greengrass-support-no-container
  :greengrass-support  -
hardware-observe        aws-iot-greengrass:hardware-observe
  :hardware-observe    manual
hardware-random-control aws-iot-greengrass:hardware-random-control           -
               -
home                    aws-iot-greengrass:home-for-greengrassd              -
               -
home                    aws-iot-greengrass:home-for-hooks                    :home
               manual
hugepages-control       aws-iot-greengrass:hugepages-control
  :hugepages-control   manual
i2c                     aws-iot-greengrass:i2c                               -
               -
iio                     aws-iot-greengrass:iio                               -
               -
joystick                aws-iot-greengrass:joystick                          -
               -
log-observe             aws-iot-greengrass:log-observe                       :log-
observe         manual
mount-observe           aws-iot-greengrass:mount-observe
  :mount-observe       manual
network                 aws-iot-greengrass:network
  :network             -
network-bind            aws-iot-greengrass:network-bind
  :network-bind        -
network-control         aws-iot-greengrass:network-control
  :network-control     -
opengl                  aws-iot-greengrass:opengl
  :opengl              -
optical-drive           aws-iot-greengrass:optical-drive
  :optical-drive       -
process-control         aws-iot-greengrass:process-control
  :process-control     -
raw-usb                 aws-iot-greengrass:raw-usb                           -
               -
removable-media         aws-iot-greengrass:removable-media                   -
               -
serial-port             aws-iot-greengrass:serial-port                       -
               -
```

```
spi                       aws-iot-greengrass:spi                                -
               -
system-observe            aws-iot-greengrass:system-observe
  :system-observe      -
```

슬롯 열에 하이픈(-)이 보이면 해당 인터페이스가 연결되지 않은 것입니다.

6. AWS IoT Greengrass 코어 소프트웨어 설치를 따라 AWS IoT 사물, Greengrass 그룹, AWS IoT을(를) 통해 보안 통신을 가능하게 하는 보안 리소스, AWS IoT Greengrass 코어 소프트웨어 구성 파일을 생성합니다. 구성 파일 `config.json`에는 인증서 파일 위치 및 AWS IoT 디바이스 데이터 엔드포인트와 같은 Greengrass 코어와 관련된 구성이 포함되어 있습니다.

> **Note**
>
> 파일을 다른 디바이스에 다운로드한 경우, 이 단계에 따라 파일을 AWS IoT Greengrass 코어 디바이스로 전송합니다.

7. AWS IoT Greengrass 스냅을 위해서는 다음과 같이 config.json 파일을 업데이트해야 합니다.

   - *certificateId*의 각 인스턴스를 인증서 및 키 파일 이름의 인증서 ID로 교체합니다.
   - Amazon 루트 CA 1과 다른 아마존 루트 CA 인증서를 다운로드한 경우, *AmazonRootca1.pem* # 각 인스턴스를 Amazon 루트 CA 파일의 이름으로 교체하십시오.

```
{
  ...
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.keyy"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key",
        "certificatePath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-certificate.pem.crt"
      }
    },
    "caPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/AmazonRootCA1.pem"
  },
```

```
  "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
  "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

8. 다음 명령을 실행하여 AWS IoT Greengrass 인증서 및 구성 파일을 추가합니다.

```
sudo snap set aws-iot-greengrass gg-certs=/home/ubuntu/my-certs
```

Lambda 함수를 배포합니다.

이 섹션에서는 AWS IoT Greengrass 스냅에서 고객 관리형 Lambda 함수를 배포하는 방법을 보여 줍니다.

> **Important**
>
> AWS IoT Greengrass 스냅 v1.11은 컨테이너화되지 않은 Lambda 함수만 지원합니다.

1. 다음 명령을 실행하여 AWS IoT Greengrass 대몬(daemon)을 시작합니다.

```
sudo snap start aws-iot-greengrass
```

응답의 예:

```
Started.
```

> **Note**
>
> 오류가 발생하는 경우, `snap run` 명령을 사용하여 자세한 오류 메시지를 표시할 수 있습니다. 문제 해결에 대한 자세한 정보는 오류: 다음 작업을 수행할 수 없습니다. - snap "" ([start snap. aws-iot-greengrass aws-iot-greengrass[.greengrassd.service] 가 종료 상태 1: 스냅 작업 때문에 실패했습니다. aws-iot-greengrass제어 프로세스가 오류 코드와 함께 종료되어.greengrassd.service가 실패했습니다. “systemctl 상태 스냅”을 참조하십시오. aws-iot-greengrass자세한 내용은 .greengrassd.service 및 “journalctl -xe”를 참조하십시오.)을 참조하십시오.

2. 다음 명령을 사용하여 대몬(daemon)이 실행 중인지 확인합니다.

```
snap services aws-iot-greengrass.greengrassd
```

응답의 예:

```
Service                          Startup   Current  Notes
aws-iot-greengrass.greengrassd   disabled  active   -
```

3. 모듈 3 (1부): AWS IoT Greengrass의 Lambda 함수를 따라 Hello World Lambda 함수 생성하고 배포합니다. 하지만 Lambda 함수를 배포하기 전에 다음 단계를 완료합니다.
4. Lambda 함수가 컨테이너 없음 모드에서 snap_daemon 사용자로 실행되는지 확인합니다. Greengrass 그룹의 설정을 업데이트하려면 AWS IoT Greengrass 콘솔에서 다음과 같이 합니다.
   a. AWS IoT Greengrass 콘솔에 로그인합니다.
   b. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
   c. Greengrass 그룹에서 대상 그룹을 선택합니다.
   d. 그룹 구성 페이지의 탐색 창에서 Lambda 함수 탭을 선택합니다.
   e. 기본 Lambda 함수 런타임 환경에서 편집을 선택하고 다음을 수행합니다.
      i. 기본 시스템 사용자 및 그룹의 경우, 다른 사용자 ID/그룹 ID를 선택한 다음 시스템 사용자 ID(번호) 와 시스템 그룹 ID(숫자) 모두에 **584788**을 입력합니다.
      ii. 기본 Lambda 함수 컨테이너화의 경우, 컨테이너 없음을 선택합니다.
      iii. 저장을 선택합니다.

AWS IoT Greengrass 대몬(daemon) 중지

`snap stop`명령을 사용하여 서비스를 중지할 수 있습니다.

다음 명령을 실행해 AWS IoT Greengrass 대몬(daemon)을 중단합니다.

```
sudo snap stop aws-iot-greengrass
```

명령이 `Stopped.`을(를) 반환합니다.

스냅을 성공적으로 중지했는지 확인하려면 다음 명령을 실행합니다.

```
snap services aws-iot-greengrass.greengrassd
```

응답의 예:

```
Service                          Startup   Current   Notes
aws-iot-greengrass.greengrassd  disabled  inactive  -
```

AWS IoT Greengrass 스냅 제거

AWS IoT Greengrass 스냅을 제거하려면 다음 명령을 실행합니다.

```
sudo snap remove aws-iot-greengrass
```

응답의 예:

```
aws-iot-greengrass removed
```

AWS IoT Greengrass 스냅 문제 해결

다음 정보를 사용하면 AWS IoT Greengrass 스냅 문제 해결에 도움이 됩니다.

권한 거부 오류

해결책: 권한 거부 오류는 종종 인터페이스 누락으로 인해 발생합니다. 누락된 인터페이스 목록과 자세한 문제 해결 정보를 보려면 `snappy-debug` 도구를 사용할 수 있습니다.

다음 명령을 실행하여 도구를 설치합니다.

```
sudo snap install snappy-debug
```

응답의 예:

```
snappy-debug 0.36-snapd2.45.1 from Canonical# installed
```

별도의 터미널 세션에서 `sudo snappy-debug` 명령을 실행합니다. 권한 거부 오류가 발생할 때까지 작업이 계속됩니다.

예를 들어, Lambda 함수가 `$HOME` 디렉터리에 있는 파일을 읽으려고 하면 다음과 같은 응답을 받을 수 있습니다.

```
INFO: Following '/var/log/syslog'. If have dropped messages, use:
INFO: $ sudo journalctl --output=short --follow --all | sudo snappy-debug
kernel.printk_ratelimit = 0
= AppArmor =
Time: Dec  6 04:48:26
Log: apparmor="DENIED" operation="mknod" profile="snap.aws-iot-greengrass.greengrassd"
 name="/home/ubuntu/my-file.txt" pid=12345 comm="touch" requested_mask="c"
 denied_mask="c" fsuid=0 ouid=0
File: /home/ubuntu/my-file.txt (write)
Suggestion:
* add 'home' to 'plugs'
```

이 예제는 `/home/ubuntu/my-file.txt` 파일을 생성할 때 권한 오류가 발생했음을 보여줍니다. 또한 `plugs`에 home을(를) 추가할 것을 제안합니다. 하지만 이 제안은 해당되지 않습니다. `home-for-greengrassd` 및 `home-for-hooks` 플러그에는 읽기 전용 액세스 권한만 제공됩니다.

자세한 내용은 스냅 설명서의 The 스냅py-debug 스냅을 참조하세요.

오류: 다음 작업을 수행할 수 없습니다. - snap "" ([start snap. aws-iot-greengrass aws-iot-greengrass[.greengrassd.service] 가 종료 상태 1: 스냅 작업 때문에 실패했습니다. aws-iot-greengrass 제어 프로세스가 오류 코드와 함께 종료되어.greengrassd.service가 실패했습니다. “systemctl 상태 스냅”을 참조하십시오. aws-iot-greengrass자세한 내용은 .greengrassd.service 및 “journalctl -xe”를 참조하십시오.)

해결책: `snap start aws-iot-greengrass` 명령이 AWS IoT Greengrass 코어 소프트웨어가 시작하는 데 실패하면 이 오류가 발생할 수 있습니다.

자세한 문제 해결 정보를 보려면 다음 명령을 실행합니다.

```
sudo snap run aws-iot-greengrass.greengrassd
```

응답의 예:

```
Couldn't find /snap/aws-iot-greengrass/44/greengrass/config/config.json.
```

이 예제는 AWS IoT Greengrass이(가) `config.json` 파일을 찾을 수 없는 경우를 보여줍니다. 구성 및 인증서 파일을 확인할 수 있습니다.

/var/snap/ /current/ aws-iot-greengrass /packages/1.11.5/rootfs/merged는 절대 경로가 아니거나 심볼릭 링크입니다. ggc-write-directory

해결책: AWS IoT Greengrass 스냅은 컨테이너화되지 않은 Lambda 함수만 지원합니다. Lambda 함수를 컨테이너 없음 모드에서 실행해야 합니다. 자세한 내용은 AWS IoT Greengrass Version 1 개발자 안내서의 Lambda 함수 컨테이너화 선택 시 고려 사항을 참조하세요.

sudo snap refresh snapd 명령을 실행한 후 스냅d 대몬(daemon)을 다시 시작하지 못했습니다.

해결책: AWS IoT Greengrass 스냅 설치 및 구성에서 6~8단계에 따라 AWS IoT Greengrass스냅에 AWS IoT Greengrass 인증서 및 구성 파일을 추가합니다.

AWS IoT Greengrass 코어 소프트웨어 설치 아카이브

새로운 AWS IoT Greengrass 코어 소프트웨어 버전으로 업그레이드할 때 현재 설치된 버전을 아카이브할 수 있습니다. 그러면 현재 설치 환경이 보존되어 동일한 하드웨어에서 새 소프트웨어 버전을 테스트할 수 있습니다. 또한 어떤 이유이든 손쉽게 아카이브된 버전으로 롤백할 수 있습니다.

현재 버전을 아카이브하고 새 버전을 설치하려면

1. 업그레이드하려는 AWS IoT Greengrass 코어 소프트웨어 설치 패키지를 다운로드합니다.
2. 패키지를 대상 코어 디바이스에 복사합니다. 파일을 전송하는 방법을 보여 주는 지침은 이 단계를 참조하십시오.

   > **Note**
   >
   > 나중에 현재 인증서, 키 및 구성 파일을 새 설치에 복사합니다.

   코어 디바이스 터미널에서 다음 단계에 따라 명령을 실행합니다.
3. 코어 디바이스에서 Greengrass 대몬(daemon)이 중지되어 있는지 확인합니다.

   a. 대몬(daemon)이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 root에 대한 /greengrass/ggc/packages/*ggc-version*/bin/daemon 입력이 포함되어 있는 경우에는 대몬(daemon)이 실행 중인 것입니다.

> **Note**
>
> 이 절차는 AWS IoT Greengrass 코어 소프트웨어가 `/greengrass` 디렉터리에 설치되었다는 가정하에 작성되었습니다.

b. 대몬(daemon)을 종료할 경우:

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
```

4. 현재 Greengrass 루트 디렉터리를 다른 디렉터리로 이동합니다.

```
sudo mv /greengrass /greengrass_backup
```

5. 코어 디바이스에서 새 소프트웨어의 압축을 해제합니다. 명령에서 *`os-architecture`* 및 *`version`* 자리 표시자를 바꿉니다.

```
sudo tar -zxvf greengrass-os-architecture-version.tar.gz -C /
```

6. 아카이브된 인증서, 키 및 구성 파일을 새 설치에 복사합니다.

```
sudo cp /greengrass_backup/certs/* /greengrass/certs
sudo cp /greengrass_backup/config/* /greengrass/config
```

7. 대몬(daemon)을 시작합니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

이제, 그룹 배포를 만들어 새 설치를 테스트할 수 있습니다. 문제가 발생할 경우, 아카이브된 설치를 복원할 수 있습니다.

아카이브된 설치를 복원하려면

1. 대몬(daemon)을 중지합니다.
2. 새 `/greengrass` 디렉터리를 삭제합니다.
3. `/greengrass_backup` 디렉터리를 다시 `/greengrass`로 이동합니다.
4. 대몬(daemon)을 시작합니다.

AWS IoT Greengrass 코어 구성

AWS IoT Greengrass 코어는 엣지 환경에서 허브 또는 게이트웨이 역할을 하는 AWS IoT 사물(디바이스)입니다. 다른 AWS IoT 디바이스와 마찬가지로 코어는 레지스트리에 존재하고, 디바이스 섀도우를 보유하며, 디바이스 인증서를 사용해 AWS IoT Core 및 AWS IoT Greengrass를 인증합니다. 코어 디바이스는 Greengrass 그룹에 대한 로컬 프로세스(예: 통신, 섀도우 동기화 및 토큰 교환)를 관리할 수 있도록 해주는 AWS IoT Greengrass 코어 소프트웨어를 실행합니다.

AWS IoT Greengrass 코어 소프트웨어는 다음과 같은 기능을 제공합니다.

- 커넥터 및 Lambda 함수의 배포 및 로컬 실행.
- AWS 클라우드로 자동 내보내기를 사용하여 로컬에서 데이터 스트림을 처리합니다.
- 관리형 구독을 사용한 디바이스, 커넥터, Lambda 함수 간의 로컬 네트워크를 통한 MQTT 메시징.
- 관리형 구독을 사용한 AWS IoT와 디바이스, 커넥터, Lambda 함수 간의 MQTT 메시징.
- 디바이스 인증 및 권한 부여를 사용하여 디바이스와 AWS 클라우드 간에 설정되는 보안 연결.
- 디바이스의 로컬 섀도우 동기화. 섀도우는 AWS 클라우드와 동기화하도록 구성할 수 있습니다.
- 로컬 디바이스 및 볼륨 리소스에 대한 액세스 제어.
- 로컬 추론 실행을 위한 클라우드 학습 머신러닝 모델 배포.
- 디바이스에서 Greengrass 코어 디바이스를 검색할 수 있도록 지원하는 자동 IP 주소 감지.
- 새 그룹 또는 업데이트된 그룹 구성의 중앙 배포. 구성 데이터를 다운로드한 후 코어 디바이스가 자동으로 다시 시작합니다.
- 사용자 정의 Lambda 함수의 안전한 over-the-air (OTA) 소프트웨어 업데이트
- 로컬 보안 암호의 안전하고 암호화된 저장 및 커넥터 및 Lambda 함수의 액세스 제어.

AWS IoT Greengrass 코어 구성 파일

AWS IoT Greengrass 코어 소프트웨어에 대한 구성 파일은 `config.json`입니다. 이 파일은 `/greengrass-root/config` 디렉터리에 있습니다.

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 `/greengrass` 디렉터리입니다.
> AWS IoT Greengrass 콘솔에서 기본 그룹 생성 옵션을 사용하면 `config.json` 파일이 작동 상태로 코어 디바이스에 배포됩니다.

다음 명령을 실행하여 이 파일의 내용을 검토할 수 있습니다.

```
cat /greengrass-root/config/config.json
```

다음은 예 `config.json` 파일입니다. 이 버전은 AWS IoT Greengrass 콘솔에서 코어를 만들 때 생성됩니다.

GGC v1.11

```
{
    "coreThing": {
        "caPath": "root.ca.pem",
        "certPath": "hash.cert.pem",
        "keyPath": "hash.private.key",
        "thingArn": "arn:partition:iot:region:account-id:thing/core-thing-name",
        "iotHost": "host-prefix-ats.iot.region.amazonaws.com",
        "ggHost": "greengrass-ats.iot.region.amazonaws.com",
        "keepAlive": 600,
        "ggDaemonPort": 8000,
        "systemComponentAuthTimeout": 5000
    },
    "runtime": {
        "maxWorkItemCount": 1024,
        "maxConcurrentLimit": 25,
        "lruSize": 25,
        "mountAllBlockDevices": "no",
        "cgroup": {
            "useSystemd": "yes"
        }
    },
    "managedRespawn": false,
    "crypto": {
        "principals": {
            "SecretsManager": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key"
            },
            "IoTCertificate": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key",
                "certificatePath": "file:///greengrass/certs/hash.cert.pem"
            }
        },
        "caPath": "file:///greengrass/certs/root.ca.pem"
    },
```

```
    "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
    "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

config.json 파일은 다음 속성을 지원합니다.

coreThing

필드	설명	참고
caPath	*/greengrass-root* /certs 디렉터리에 상대적인 AWS IoT 루트 CA의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
certPath	*/greengrass-root* /certs 디렉터리에 상대적인 코어 디바이스 인증서의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
keyPath	*/greengrass-root* /certs 디렉터리에 상대적인 코어 프라이빗 키의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
thingArn	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 aws greengrass get-core-definition-version CLI 명령을 실행하여 AWS IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.

필드	설명	참고
`iotHost`	AWS IoT 엔드포인트입니다.	설정 아래에서 또는 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 명령을 실행하여 AWS IoT 콘솔에서 엔드포인트를 찾습니다. 이 명령은 Amazon Trust Services(ATS) 엔드포인트를 반환합니다. 자세한 내용은 서버 인증 설명서를 참조하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.

필드	설명	참고
`ggHost`	AWS IoT Greengrass 엔드포인트입니다.	호스트 접두사가 greengrass로 대체된 `iotHost` 엔드포인트입니다(예: `greengrass-ats.iot.`*`region`*`.amazonaws.com`). `iotHost`와 동일한 AWS 리전을 사용하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.
`iotMqttPort`	선택 사항으로, AWS IoT과(와)의 MQTT 통신에 사용하는 포트 번호.	유효한 값은 8883 또는 443입니다. 기본 값은 8883입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`iotHttpPort`	선택 사항으로, AWS IoT에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 8443 또는 443입니다. 기본 값은 8443입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.

필드	설명	참고
`ggMqttPort`	선택 사항으로, 로컬 네트워크를 통한 MQTT 통신에 사용할 포트 번호입니다.	유효한 값은 `1024 - 65535`입니다. 기본 값은 `8883`입니다. 자세한 내용은 the section called “로컬 메시징을 위한 MQTT 포트” 단원을 참조하십시오.
`ggHttpPort`	선택 사항으로, AWS IoT Greengrass 서비스에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 `8443` 또는 `443`입니다. 기본 값은 `8443`입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`keepAlive`	선택 사항으로, MQTT `KeepAlive` 주기(초)입니다.	유효한 범위는 30~1200초입니다. 기본 값은 `600`입니다.
`networkProxy`	선택 사항으로, 연결할 프록시 서버를 정의하는 객체입니다.	프록시 서버는 HTTP 또는 HTTPS일 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.
`mqttOperationTimeout`	선택 사항으로, Greengrass 코어가 AWS IoT Core에 대한 MQTT 연결에서 게시, 구독 또는 구독 취소 작업을 완료할 수 있는 시간(초)입니다.	기본값은 5입니다. 최소값은 5입니다.
`ggDaemonPort`	선택 사항으로, Greengrass 코어 IPC 포트 번호입니다.	이 속성은 AWS IoT Greengrass v1.11.0 이상에서 사용할 수 있습니다. 유효한 값은 1024~65535입니다. 기본값은 8000입니다.

필드	설명	참고
`systemComponentAuthTimeout`	선택 사항으로, Greengrass 코어 IPC가 인증을 완료할 시간(밀리초) 입니다.	이 속성은 AWS IoT Greengrass v1.11.0 이상에서 사용할 수 있습니다. 유효한 값은 500~5000입니다. 기본값은 5000입니다.

실행 시간

필드	설명	참고
`maxWorkItem##`	선택 사항으로, Greengrass 대몬(daemon)이 한 번에 처리할 수 있는 최대 작업 항목 수입니다. 이 제한을 초과하는 작업 항목은 무시됩니다. 작업 항목 대기열은 시스템 구성 요소, 사용자 정의 Lambda 함수 및 커넥터에서 공유됩니다.	기본값은 1,024입니다. 최대 값은 디바이스 하드웨어에 의해 제한됩니다. 이 값을 늘리면 AWS IoT Greengrass에서 사용하는 메모리가 증가합니다. 코어가 사용률이 높은 MQTT 메시지 트래픽을 수신할 것으로 예상되는 경우 이 값을 늘릴 수 있습니다.
`maxConcurrentLimit`	선택 사항으로, Greengrass 대몬(daemon)이 보유할 수 있는 최대 동시성 비고정 Lambda 작업자 수입니다. 사용자는 다른 정수를 지정하여 이 파라미터를 재정의할 수 있습니다.	기본값은 25입니다. 최소값은 `lruSize`에 의해 정의됩니다.
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.

필드	설명	참고
`mountAllBlock####`	Optional. Enables AWS IoT Greengrass to use bind mounts to mount all block devices into a container after setting up the OverlayFS.	이 속성은 AWS IoT Greengrass v1.11.0 이상에서 사용할 수 있습니다. 유효 값은 `yes` 및 `no`입니다. 기본 값은 `no`입니다. `/usr` 디렉터리가 `/` 계층 구조 아래에 있지 않은 경우 이 값을 `yes`로 설정하십시오.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `###`. Run the `check_ggc_dependencies` script in 모듈 1 to see if your device uses `systemd`.

crypto

`crypto`에는 PKCS#11 및 로컬 비밀 스토리지를 통해 하드웨어 보안 모듈(HSM)에서 프라이빗 키 스토리지를 지원하는 속성이 포함되어 있습니다. 자세한 내용은 the section called “보안 주체”, the section called “하드웨어 보안 통합”, 코어에 암호 배포 단원을 참조하세요. HSM 및 파일 시스템의 프라이빗 키 스토리지 구성이 지원됩니다.

필드	설명	참고
caPath	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
PKCS11		
OpenSSLEngine	선택 사항으로, OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진 `.so` 파일에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다. 하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 설명은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.
P11Provider	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
slotLabel	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
slotUserPin	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
principals		

필드	설명	참고
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoT ###. privateKeyPath`	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.
`MQTT ServerCertificate`	선택 사항으로, 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	
`#ServerCertificate . privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.

필드	설명	참고
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

다음 구성 속성도 지원됩니다.

필드	설명	참고
`mqttMaxConnectionRetryInterval`	선택 사항으로, 연결이 삭제된 경우, MQTT 연결 재시도 간의 최대 간격(초)입니다.	부호 없는 정수로 이 값을 지정합니다. 기본값은 `60`입니다.
`managedRespawn`	선택 사항으로, 업데이트 전 OTA 에이전트에서 사용자 지정 코드를 실행해야 함을 나타냅니다.	유효한 값은 `true` 또는 `false`입니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.
`writeDirectory`	선택 사항으로, AWS IoT Greengrass가 모든 읽기-쓰기 리소스를 생성하는 쓰기 디렉터리입니다.	자세한 설명은 AWS IoT Greengrass를 위한 쓰기 디렉터리 구성 섹션을 참조하세요.

필드	설명	참고
pidFileDirectory	선택 사항입니다. AWS IoT Greengrass는 프로세스 ID(PID)를 이 디렉터리에 저장합니다.	기본 값은 `/var/run`입니다.

Extended life versions

다음 버전의 AWS IoT Greengrass 코어 소프트웨어는 수명 연장 단계에 있습니다. 이 정보는 참고 용도로만 포함된 것입니다.

GGC v1.10

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600,
    "systemComponentAuthTimeout": 5000
  },
  "runtime" : {
    "maxWorkItemCount" : 1024,
    "maxConcurrentLimit" : 25,
    "lruSize": 25,
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
```

```
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

config.json 파일은 다음 속성을 지원합니다.

coreThing

필드	설명	참고
caPath	/*greengrass-root* /certs 디렉터리에 상대적인 AWS IoT 루트 CA의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
certPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 디바이스 인증서의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
keyPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 프라이빗 키의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
thingArn	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 aws greengrass get-core-definition-version _ CLI 명령을 실행하여 AWS

필드	설명	참고
		IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.
`iotHost`	AWS IoT 엔드포인트입니다.	설정 아래에서 또는 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 명령을 실행하여 AWS IoT 콘솔에서 엔드포인트를 찾습니다. 이 명령은 Amazon Trust Services(ATS) 엔드포인트를 반환합니다. 자세한 내용은 서버 인증 설명서를 참조하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.

필드	설명	참고
`ggHost`	AWS IoT Greengrass 엔드포인트입니다.	호스트 접두사가 greengrass로 대체된 `iotHost` 엔드포인트입니다(예: `greengrass-ats.iot.`*`region`*`.amazonaws.com`). `iotHost`와 동일한 AWS 리전을 사용하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.
`iotMqttPort`	선택 사항으로, AWS IoT과(와)의 MQTT 통신에 사용하는 포트 번호.	유효한 값은 8883 또는 443입니다. 기본 값은 8883입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`iotHttpPort`	선택 사항으로, AWS IoT에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 8443 또는 443입니다. 기본 값은 8443입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.

필드	설명	참고
`ggMqttPort`	선택 사항으로, 로컬 네트워크를 통한 MQTT 통신에 사용할 포트 번호입니다.	유효한 값은 `1024` - `65535`입니다. 기본 값은 `8883`입니다. 자세한 내용은 the section called “로컬 메시징을 위한 MQTT 포트” 단원을 참조하십시오.
`ggHttpPort`	선택 사항으로, AWS IoT Greengrass 서비스에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 `8443` 또는 `443`입니다. 기본 값은 `8443`입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`keepAlive`	선택 사항으로, MQTT `KeepAlive` 주기(초)입니다.	유효한 범위는 30~1200초입니다. 기본 값은 `600`입니다.
`networkProxy`	선택 사항으로, 연결할 프록시 서버를 정의하는 객체입니다.	프록시 서버는 HTTP 또는 HTTPS일 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.
`mqttOperationTimeout`	선택 사항으로, Greengrass 코어가 AWS IoT Core에 대한 MQTT 연결에서 게시, 구독 또는 구독 취소 작업을 완료할 수 있는 시간(초)입니다.	이 속성은 AWS IoT Greengrass v1.10.2부터 사용할 수 있습니다. 기본값은 5입니다. 최소값은 5입니다.

실행 시간

필드	설명	참고
`maxWorkItem##`	선택 사항으로, Greengrass 대몬(daemon)이 한 번에 처리할 수 있는 최대 작업 항목 수입니다. 이 제한을 초과하는 작업 항목은 무시됩니다. 작업 항목 대기열은 시스템 구성 요소, 사용자 정의 Lambda 함수 및 커넥터에서 공유됩니다.	기본값은 1,024입니다. 최대값은 디바이스 하드웨어에 의해 제한됩니다. 이 값을 늘리면 AWS IoT Greengrass에서 사용하는 메모리가 증가합니다. 코어가 사용률이 높은 MQTT 메시지 트래픽을 수신할 것으로 예상되는 경우 이 값을 늘릴 수 있습니다.
`maxConcurrentLimit`	선택 사항으로, Greengrass 대몬(daemon)이 보유할 수 있는 최대 동시성 비고정 Lambda 작업자 수입니다. 사용자는 다른 정수를 지정하여 이 파라미터를 재정의할 수 있습니다.	기본값은 25입니다. 최소값은 `lruSize`에 의해 정의됩니다.
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `##`. Run the `check_ggc_dependencies` script in

필드	설명	참고
		모듈 1 to see if your device uses `systemd`.

crypto

`crypto`에는 PKCS#11 및 로컬 비밀 스토리지를 통해 하드웨어 보안 모듈(HSM)에서 프라이빗 키 스토리지를 지원하는 속성이 포함되어 있습니다. 자세한 내용은 the section called “보안 주체”, the section called “하드웨어 보안 통합”, 코어에 암호 배포 단원을 참조하세요. HSM 및 파일 시스템의 프라이빗 키 스토리지 구성이 지원됩니다.

필드	설명	참고
`caPath`	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
`PKCS11`		
`OpenSSLEngine`	선택 사항으로, OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진 `.so` 파일에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다. 하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 설명은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.

필드	설명	참고
`P11Provider`	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
`slotLabel`	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
`slotUserPin`	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoT ###. privateKeyPath`	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.
`MQTT ServerCertificate`	선택 사항으로, 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	

필드	설명	참고
`#ServerCertificate . privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

다음 구성 속성도 지원됩니다.

필드	설명	참고
`mqttMaxConnectionRetryInterval`	선택 사항으로, 연결이 삭제된 경우, MQTT 연결 재시도 간의 최대 간격(초)입니다.	부호 없는 정수로 이 값을 지정합니다. 기본값은 `60`입니다.
`managedRespawn`	선택 사항으로, 업데이트 전 OTA 에이전트에서 사용자 지정 코드를 실행해야 함을 나타냅니다.	유효한 값은 `true` 또는 `false`입니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.
`writeDirectory`	선택 사항으로, AWS IoT Greengrass가 모든 읽기-쓰기 리소스를 생성하는 쓰기 디렉터리입니다.	자세한 설명은 AWS IoT Greengrass를 위한 쓰기 디렉터리 구성 섹션을 참조하세요.

GGC v1.9

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
```

```
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

config.json 파일은 다음 속성을 지원합니다.

coreThing

필드	설명	참고
caPath	/*greengrass-root* /certs 디렉터리에 상대적인 AWS IoT 루트 CA의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
certPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 디바이스 인증서의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
keyPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 프라이빗 키의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.

필드	설명	참고
`thingArn`	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 `aws greengrass get-core-definition-version` CLI 명령을 실행하여 AWS IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.
`iotHost`	AWS IoT 엔드포인트입니다.	설정 아래에서 또는 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 명령을 실행하여 AWS IoT 콘솔에서 엔드포인트를 찾습니다. 이 명령은 Amazon Trust Services(ATS) 엔드포인트를 반환합니다. 자세한 내용은 서버 인증 설명서를 참조하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.

필드	설명	참고
`ggHost`	AWS IoT Greengrass 엔드포인트입니다.	호스트 접두사가 greengrass로 대체된 `iotHost` 엔드포인트입니다(예: `greengrass-ats.iot.`*`region`*`.amazonaws.com`). `iotHost`와 동일한 AWS 리전을 사용하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.
`iotMqttPort`	선택 사항으로, AWS IoT과(와)의 MQTT 통신에 사용하는 포트 번호.	유효한 값은 8883 또는 443입니다. 기본 값은 8883입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`iotHttpPort`	선택 사항으로, AWS IoT에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 8443 또는 443입니다. 기본 값은 8443입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.

필드	설명	참고
`ggHttpPort`	선택 사항으로, AWS IoT Greengrass 서비스에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 8443 또는 443입니다. 기본 값은 8443입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`keepAlive`	선택 사항으로, MQTT `KeepAlive` 주기(초)입니다.	유효한 범위는 30~1200초입니다. 기본 값은 `600`입니다.
`networkProxy`	선택 사항으로, 연결할 프록시 서버를 정의하는 객체입니다.	프록시 서버는 HTTP 또는 HTTPS일 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

실행 시간

필드	설명	참고
`maxConcurrentLimit`	선택 사항으로, Greengrass 대몬(daemon)이 보유할 수 있는 최대 동시성 비고정 Lambda 작업자 수입니다. 사용자는 다른 정수를 지정하여 이 파라미터를 재정의할 수 있습니다.	기본값은 25입니다. 최소값은 `lruSize`에 의해 정의됩니다.
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting	The default timeout is 30 seconds (30000 ms).

필드	설명	참고
	that the Greengrass daemon waits for the health check to finish.	
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `###`. Run the `check_ggc_dependencies` script in 모듈 1 to see if your device uses `systemd`.

crypto

`crypto` 객체는 v1.7.0에서 추가되었습니다. 이 객체는 PKCS#11 및 로컬 보안 스토리지를 통해 HSM(하드웨어 보안 모듈)에 대한 프라이빗 키 스토리지를 지원하는 속성을 도입합니다. 자세한 내용은 the section called “보안 주체”, the section called “하드웨어 보안 통합”, 코어에 암호 배포 단원을 참조하세요. HSM 및 파일 시스템의 프라이빗 키 스토리지 구성이 지원됩니다.

필드	설명	참고
`caPath`	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
`PKCS11`		
`OpenSSLEngine`	선택 사항으로, OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진	파일 시스템에서는 파일에 대한 경로여야 합니다.

필드	설명	참고
	.so 파일에 대한 절대 경로입니다.	하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 설명은 the section called "OTA 업데이트 구성" 섹션을 참조하세요.
P11Provider	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
slotLabel	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
slotUserPin	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoT ###. privateKeyPath	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 file:///*absolute/path/to/file* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.

필드	설명	참고
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.
`MQTT ServerCertificate`	선택 사항으로, 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	
`#ServerCertificate . privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	

필드	설명	참고
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

다음 구성 속성도 지원됩니다.

필드	설명	참고
`mqttMaxConnectionRetryInterval`	선택 사항으로, 연결이 삭제된 경우, MQTT 연결 재시도 간의 최대 간격(초)입니다.	부호 없는 정수로 이 값을 지정합니다. 기본값은 `60`입니다.
`managedRespawn`	선택 사항으로, 업데이트 전 OTA 에이전트에서 사용자 지정 코드를 실행해야 함을 나타냅니다.	유효한 값은 `true` 또는 `false`입니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.
`writeDirectory`	선택 사항으로, AWS IoT Greengrass가 모든 읽기-쓰기 리소스를 생성하는 쓰기 디렉터리입니다.	자세한 설명은 AWS IoT Greengrass를 위한 쓰기 디렉터리 구성 섹션을 참조하세요.

GGC v1.8

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

`config.json` 파일은 다음 속성을 지원합니다.

coreThing

필드	설명	참고
`caPath`	*/greengrass-root* `/certs` 디렉터리에 상대적인 AWS IoT 루트 CA의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. `crypto` 객체가 있으면 이 속성이 무시됩니다.

필드	설명	참고
		Note 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
certPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 디바이스 인증서의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
keyPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 프라이빗 키의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
thingArn	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 aws greengrass get-core-definition-version CLI 명령을 실행하여 AWS IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.

필드	설명	참고
`iotHost`	AWS IoT 엔드포인트입니다.	설정 아래에서 또는 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 명령을 실행하여 AWS IoT 콘솔에서 엔드포인트를 찾습니다. 이 명령은 Amazon Trust Services(ATS) 엔드포인트를 반환합니다. 자세한 내용은 서버 인증 설명서를 참조하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.

필드	설명	참고
`ggHost`	AWS IoT Greengrass 엔드포인트입니다.	호스트 접두사가 greengrass로 대체된 `iotHost` 엔드포인트입니다(예: `greengrass-ats.iot.region.amazonaws.com`). `iotHost`와 동일한 AWS 리전을 사용하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.
`iotMqttPort`	선택 사항으로, AWS IoT과(와)의 MQTT 통신에 사용하는 포트 번호.	유효한 값은 8883 또는 443입니다. 기본 값은 8883입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`iotHttpPort`	선택 사항으로, AWS IoT에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 8443 또는 443입니다. 기본 값은 8443입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.

필드	설명	참고
ggHttpPort	선택 사항으로, AWS IoT Greengrass 서비스에 HTTPS 연결 생성을 위해 사용되는 포트 번호입니다.	유효한 값은 8443 또는 443입니다. 기본 값은 8443입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
keepAlive	선택 사항으로, MQTT KeepAlive 주기(초)입니다.	유효한 범위는 30~1200초입니다. 기본 값은 600입니다.
networkProxy	선택 사항으로, 연결할 프록시 서버를 정의하는 객체입니다.	프록시 서버는 HTTP 또는 HTTPS일 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

실행 시간

필드	설명	참고
cgroup		
useSystemd	Indicates whether your device uses systemd.	Valid values are yes or ###. Run the check_ggc_dependencies script in 모듈 1 to see if your device uses systemd.

crypto

crypto 객체는 v1.7.0에서 추가되었습니다. 이 객체는 PKCS#11 및 로컬 보안 스토리지를 통해 HSM(하드웨어 보안 모듈)에 대한 프라이빗 키 스토리지를 지원하는 속성을 도입합니다. 자세한 내용은 the section called “보안 주체”, the section called “하드웨어 보안 통합”, 코어에 암호 배포 단원을 참조하세요. HSM 및 파일 시스템의 프라이빗 키 스토리지 구성이 지원됩니다.

필드	설명	참고
`caPath`	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
`PKCS11`		
`OpenSSLEngine`	선택 사항으로, OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진 `.so` 파일에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다. 하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 설명은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.
`P11Provider`	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
`slotLabel`	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
`slotUserPin`	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.

필드	설명	참고
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoT ###. privateKeyPath`	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.
`MQTT ServerCertificate`	선택 사항으로, 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	

필드	설명	참고
`#ServerCertificate . privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

다음 구성 속성도 지원됩니다.

필드	설명	참고
`mqttMaxConnectionRetryInterval`	선택 사항으로, 연결이 삭제된 경우, MQTT 연결 재시도 간의 최대 간격(초)입니다.	부호 없는 정수로 이 값을 지정합니다. 기본값은 `60`입니다.
`managedRespawn`	선택 사항으로, 업데이트 전 OTA 에이전트에서 사용자 지정 코드를 실행해야 함을 나타냅니다.	유효한 값은 `true` 또는 `false`입니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.
`writeDirectory`	선택 사항으로, AWS IoT Greengrass가 모든 읽기-쓰기 리소스를 생성하는 쓰기 디렉터리입니다.	자세한 설명은 AWS IoT Greengrass를 위한 쓰기 디렉터리 구성 섹션을 참조하세요.

GGC v1.7

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
```

```
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

config.json 파일은 다음 속성을 지원합니다.

coreThing

필드	설명	참고
caPath	/*greengrass-root* /certs 디렉터리에 상대적인 AWS IoT 루트 CA의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
certPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 디바이스 인증서의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.
keyPath	/*greengrass-root* /certs 디렉터리에 상대적인 코어 프라이빗 키의 경로입니다.	1.7.0 이전 버전과의 호환성을 위한 용도. crypto 객체가 있으면 이 속성이 무시됩니다.

필드	설명	참고
`thingArn`	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 `aws greengrass get-core-definition-version` CLI 명령을 실행하여 AWS IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.
`iotHost`	AWS IoT 엔드포인트입니다.	설정 아래에서 또는 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 명령을 실행하여 AWS IoT 콘솔에서 엔드포인트를 찾습니다. 이 명령은 Amazon Trust Services(ATS) 엔드포인트를 반환합니다. 자세한 내용은 서버 인증 설명서를 참조하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.

필드	설명	참고
`ggHost`	AWS IoT Greengrass 엔드포인트입니다.	호스트 접두사가 greengrass로 대체된 `iotHost` 엔드포인트입니다(예: `greengrass-ats.iot.region.amazonaws.com`). `iotHost`와 동일한 AWS 리전을 사용하십시오. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다. 엔드포인트는 해당 AWS 리전과 일치해야 합니다.
`iotMqttPort`	선택 사항으로, AWS IoT과(와)의 MQTT 통신에 사용하는 포트 번호.	유효한 값은 8883 또는 443입니다. 기본 값은 8883입니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오.
`keepAlive`	선택 사항으로, MQTT `KeepAlive` 주기(초)입니다.	유효한 범위는 30~1200초입니다. 기본 값은 `600`입니다.
`networkProxy`	선택 사항으로, 연결할 프록시 서버를 정의하는 객체입니다.	프록시 서버는 HTTP 또는 HTTPS일 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

실행 시간

필드	설명	참고
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are `yes` or `###`. Run the `check_ggc_dependencies` script in 모듈 1 to see if your device uses `systemd`.

crypto

v1.7.0에 추가된 `crypto` 객체는 PKCS#11 및 로컬 비밀 스토리지를 통해 하드웨어 보안 모듈(HSM)에 대한 프라이빗 키 스토리지를 지원하는 속성을 도입합니다. 자세한 내용은 the section called “하드웨어 보안 통합” 및 코어에 암호 배포 섹션을 참조하세요. HSM 및 파일 시스템의 프라이빗 키 스토리지 구성이 지원됩니다.

필드	설명	참고
`caPath`	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
`PKCS11`		
`OpenSSLEngine`	선택 사항으로, OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진	파일 시스템에서는 파일에 대한 경로여야 합니다. 하드웨어 보안과 함께 Greengrass OTA 업데이트

필드	설명	참고
	.so 파일에 대한 절대 경로입니다.	에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 설명은 the section called "OTA 업데이트 구성" 섹션을 참조하세요.
P11Provider	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
slotLabel	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
slotUserPin	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	
IoT ###. privateKeyPath	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 file:///*absolute/path/to/file* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
IoTCertificate .certificatePath	코어 디바이스 인증서의 절대 경로입니다.	file:///*absolute/path/to/file* 형식의 파일 URI여야 합니다.

필드	설명	참고
`MQTT ServerCertificate`	선택 사항으로, 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	
`#ServerCertificate . privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	

필드	설명	참고
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

다음 구성 속성도 지원됩니다.

필드	설명	참고
`mqttMaxConnectionRetryInterval`	선택 사항으로, 연결이 삭제된 경우, MQTT 연결 재시도 간의 최대 간격(초)입니다.	부호 없는 정수로 이 값을 지정합니다. 기본값은 `60`입니다.
`managedRespawn`	선택 사항으로, 업데이트 전 OTA 에이전트에서 사용자 지정 코드를 실행해야 함을 나타냅니다.	유효한 값은 `true` 또는 `false`입니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.
`writeDirectory`	선택 사항으로, AWS IoT Greengrass가 모든 읽기-쓰기 리소스를 생성하는 쓰기 디렉터리입니다.	자세한 설명은 AWS IoT Greengrass를 위한 쓰기 디렉터리 구성 섹션을 참조하세요.

GGC v1.6

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600,
       "mqttMaxConnectionRetryInterval": 60
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
   "managedRespawn": true,
   "writeDirectory": "/write-directory"
}
```

> **Note**
>
> AWS IoT Greengrass 콘솔에서 기본 그룹 생성 옵션을 사용하는 경우, config.json 파일은 기본 구성을 지정하는 작동 상태로 코어 디바이스에 배포됩니다.

config.json 파일은 다음 속성을 지원합니다.

필드	설명	참고
caPath	/*greengrass-root* / certs 디렉터리에 상대적인 AWS IoT 루트 CA의 경로입니다.	파일을 /*greengrass-root*/certs에 저장합니다.
certPath	/*greengrass-root* / certs 디렉터리에 상대적인	파일을 /*greengrass-root*/certs에 저장합니다.

필드	설명	참고
	AWS IoT Greengrass 코어 인증서의 경로입니다.	
keyPath	*/greengrass-root* / certs 디렉터리에 상대적인 AWS IoT Greengrass 코어 프라이빗 키의 경로입니다.	파일을 */greengrass-root*/certs에 저장합니다.
thingArn	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 aws greengrass get-core-definition-version CLI 명령을 실행하여 AWS IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.
iotHost	AWS IoT 엔드포인트입니다.	AWS IoT 콘솔의 설정에서 또는 aws iot describe-endpoint CLI 명령을 실행하여 찾을 수 있습니다.
ggHost	AWS IoT Greengrass 엔드포인트입니다.	이 값은 greengrass.iot. *region*.amazonaws.com 형식을 사용합니다. iotHost와 같은 리전을 사용합니다.
keepAlive	MQTT KeepAlive 주기(초)입니다.	이 값은 선택 사항입니다. 기본값은 600입니다.
mqttMaxConnectionRetryInterval	연결이 삭제된 경우, MQTT 연결 재시도 간의 최대 간격(초)입니다.	부호 없는 정수로 이 값을 지정합니다. 이 값은 선택 사항입니다. 기본값은 60입니다.

필드	설명	참고
useSystemd	디바이스가 systemd를 사용하는지 여부를 나타냅니다.	유효한 값은 yes 또는 no입니다. 디바이스가 systemd를 사용하는지 알아보려면 모듈 1에서 check_ggc_dependencies 스크립트를 실행합니다.
managedRespawn	선택적 over-the-air (OTA) 업데이트 기능으로, 이는 OTA 에이전트가 업데이트 전에 사용자 지정 코드를 실행해야 함을 나타냅니다.	유효한 값은 true 또는 false입니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.
writeDirectory	AWS IoT Greengrass가 모든 읽기-쓰기 리소스를 생성하는 쓰기 디렉터리입니다.	이 값은 선택 사항입니다. 자세한 설명은 AWS IoT Greengrass를 위한 쓰기 디렉터리 구성 섹션을 참조하세요.

GGC v1.5

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
```

```
    "managedRespawn": true
}
```

config.json 파일은 */greengrass-root*/config에 있고 다음 파라미터를 포함하고 있습니다.

필드	설명	참고
caPath	*/greengrass-root* /certs 폴더에 상대적인 AWS IoT 루트 CA의 경로입니다.	파일을 폴더 */greengrass-root* /certs에 저장합니다.
certPath	*/greengrass-root* /certs 폴더에 상대적인 AWS IoT Greengrass 코어 인증서의 경로입니다.	파일을 폴더 */greengrass-root* /certs에 저장합니다.
keyPath	*/greengrass-root* /certs 폴더에 상대적인 AWS IoT Greengrass 코어 프라이빗 키의 경로입니다.	파일을 폴더 */greengrass-root* /certs에 저장합니다.
thingArn	AWS IoT Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	코어 아래에서 또는 aws greengrass get-core-definition-version CLI 명령을 실행하여 AWS IoT Greengrass 콘솔에서 코어의 ARN을 찾습니다.
iotHost	AWS IoT 엔드포인트입니다.	AWS IoT 콘솔의 설정에서 찾거나 aws iot describe-endpoint 명령을 실행하여 찾을 수 있습니다.
ggHost	AWS IoT Greengrass 엔드포인트입니다.	이 값은 greengrass.iot. *region*.amazonaw

필드	설명	참고
		s.com 형식을 사용합니다. iotHost와 같은 리전을 사용합니다.
keepAlive	MQTT KeepAlive 주기(초)입니다.	이 값은 선택 사항입니다. 기본값은 600초입니다.
useSystemd	디바이스가 systemd를 사용하는지 여부를 나타냅니다.	유효한 값은 yes 또는 no입니다. 디바이스가 systemd를 사용하는지 알아보려면 모듈 1에서 check_ggc_dependencies 스크립트를 실행합니다.
managedRespawn	선택적 over-the-air (OTA) 업데이트 기능으로, 이는 OTA 에이전트가 업데이트 전에 사용자 지정 코드를 실행해야 함을 나타냅니다.	자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.

GGC v1.3

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
```

```
    "managedRespawn": true
}
```

config.json 파일은 */greengrass-root*/config에 있고 다음 파라미터를 포함하고 있습니다.

필드	설명	참고
caPath	*/greengrass-root* /certs 폴더에 상대적인 AWS IoT 루트 CA의 경로입니다.	파일을 폴더 */greengrass-root* /certs에 저장합니다.
certPath	*/greengrass-root* /certs 폴더에 상대적인 AWS IoT Greengrass 코어 인증서의 경로입니다.	파일을 폴더 */greengrass-root* /certs에 저장합니다.
keyPath	*/greengrass-root* /certs 폴더에 상대적인 AWS IoT Greengrass 코어 프라이빗 키의 경로입니다.	파일을 폴더 */greengrass-root* /certs에 저장합니다.
thingArn	AWS IoT Greengrass 코어를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	이 값은 AWS IoT Greengrass 콘솔의 AWS IoT 사물에 대한 정의에서 찾을 수 있습니다.
iotHost	AWS IoT 엔드포인트입니다.	이 값은 AWS IoT 콘솔의 설정에서 찾을 수 있습니다.
ggHost	AWS IoT Greengrass 엔드포인트입니다.	이 값은 AWS IoT 콘솔의 설정에서 greengrass. 가 앞에 추가된 상태로 찾을 수 있습니다.
keepAlive	MQTT KeepAlive 주기(초)입니다.	이 값은 선택 사항입니다. 기본값은 600초입니다.

필드	설명	참고
useSystemd	디바이스에서 systemd를 사용하는 경우, 이진 플래그입니다.	유효한 값은 yes 또는 no입니다. 모듈 1의 종속성 스크립트를 사용하여 디바이스가 systemd을 사용하는지 확인합니다.
managedRespawn	선택적 over-the-air (OTA) 업데이트 기능으로, 이는 OTA 에이전트가 업데이트 전에 사용자 지정 코드를 실행해야 함을 나타냅니다.	자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요.

GGC v1.1

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   }
}
```

config.json 파일은 */greengrass-root*/config에 있고 다음 파라미터를 포함하고 있습니다.

필드	설명	참고
caPath	/*greengrass-root* /certs 폴더에 상대적인 AWS IoT 루트 CA의 경로입니다.	파일을 폴더 /*greengrass-root* /certs에 저장합니다.
certPath	/*greengrass-root* /certs 폴더에 상대적인 AWS IoT Greengrass 코어 인증서의 경로입니다.	파일을 폴더 /*greengrass-root* /certs에 저장합니다.
keyPath	/*greengrass-root* /certs 폴더에 상대적인 AWS IoT Greengrass 코어 프라이빗 키의 경로입니다.	파일을 폴더 /*greengrass-root* /certs에 저장합니다.
thingArn	AWS IoT Greengrass 코어를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	이 값은 AWS IoT Greengrass 콘솔의 AWS IoT 사물에 대한 정의에서 찾을 수 있습니다.
iotHost	AWS IoT 엔드포인트입니다.	이 값은 AWS IoT 콘솔의 설정에서 찾을 수 있습니다.
ggHost	AWS IoT Greengrass 엔드포인트입니다.	이 값은 AWS IoT 콘솔의 설정에서 greengrass. 가 앞에 추가된 상태로 찾을 수 있습니다.
keepAlive	MQTT KeepAlive 주기(초)입니다.	이 값은 선택 사항입니다. 기본값은 600초입니다.
useSystemd	디바이스에서 systemd를 사용하는 경우, 이진 플래그입니다.	유효한 값은 yes 또는 no입니다. 모듈 1의 종속성 스크립트를 사용하여 디바이스가 systemd을 사용하는지 확인합니다.

GGC v1.0

AWS IoT Greengrass Core v1.0에서 config.json은 *greengrass-root*/configuration에 배포됩니다.

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   }
}
```

config.json 파일은 */greengrass-root*/configuration에 있고 다음 파라미터를 포함하고 있습니다.

필드	설명	참고
caPath	*/greengrass-root* /configuration/certs 폴더에 상대적인 AWS IoT 루트 CA의 경로입니다.	파일을 폴더 */greengrass-root* /configuration/certs 에 저장합니다.
certPath	*/greengrass-root* /configuration/certs 폴더에 상대적인 AWS IoT Greengrass 코어 인증서의 경로입니다.	파일을 폴더 */greengrass-root* /configuration/certs 에 저장합니다.
keyPath	*/greengrass-root* /configuration/certs	파일을 폴더 */greengrass-root* /configur

필드	설명	참고
	폴더에 상대적인 AWS IoT Greengrass 코어 프라이빗 키의 경로입니다.	`ation/certs` 에 저장합니다.
`thingArn`	AWS IoT Greengrass 코어를 나타내는 AWS IoT 사물의 Amazon 리소스 이름(ARN)입니다.	이 값은 AWS IoT Greengrass 콘솔의 AWS IoT 사물에 대한 정의에서 찾을 수 있습니다.
`iotHost`	AWS IoT 엔드포인트입니다.	이 값은 AWS IoT 콘솔의 설정에서 찾을 수 있습니다.
`ggHost`	AWS IoT Greengrass 엔드포인트입니다.	이 값은 AWS IoT 콘솔의 설정에서 `greengrass.` 가 앞에 추가된 상태로 찾을 수 있습니다.
`keepAlive`	MQTT `KeepAlive` 주기(초)입니다.	이 값은 선택 사항입니다. 기본값은 600초입니다.
`useSystemd`	디바이스에서 `systemd`를 사용하는 경우 이진 플래그입니다.	유효한 값은 `yes` 또는 `no`입니다. 모듈 1의 종속성 스크립트를 사용하여 디바이스가 `systemd`을 사용하는지 확인합니다.

서비스 엔드포인트는 루트 CA 인증서 유형과 일치해야 합니다

AWS IoT Core 및 AWS IoT Greengrass 엔드포인트는 디바이스에 있는 루트 CA 인증서 유형과 일치해야 합니다. 엔드포인트와 인증서 유형이 일치하지 않으면 디바이스와 AWS IoT Core 또는 AWS IoT Greengrass 간의 인증 시도가 실패합니다. 자세한 내용은 AWS IoT 개발자 안내서의 서버 인증을 참조하세요.

디바이스에서 Amazon Trust Services(ATS) 루트 CA 인증서(선호되는 방법)를 사용하는 경우, 디바이스 관리 및 검색 데이터 영역 작업에도 ATS 엔드포인트를 사용해야 합니다. AWS IoT Core 엔드포인트에 대한 다음 구문과 같이 ATS 엔드포인트에는 `ats` 세그먼트가 포함됩니다.

```
prefix-ats.iot.region.amazonaws.com
```

> **Note**
>
> 이전 버전과의 호환성을 위해 AWS IoT Greengrass 현재 일부 AWS 리전 버전에서는 레거시 VeriSign 루트 CA 인증서 및 엔드포인트를 지원합니다. 레거시 VeriSign 루트 CA 인증서를 사용하는 경우 ATS 엔드포인트를 만들고 ATS 루트 CA 인증서를 대신 사용하는 것이 좋습니다. 그렇지 않을 경우 해당 레거시 엔드포인트를 사용하십시오. 자세한 내용은 Amazon Web Services 일반 참조의 지원되는 레거시 엔드포인트 섹션을 참조하세요.

config.json의 엔드포인트

Greengrass 코어 디바이스에서 엔드포인트는 `config.json` 파일의 `coreThing` 객체에 지정됩니다. `iotHost` 속성은 AWS IoT Core 엔드포인트를 나타냅니다. `ggHost` 속성은 AWS IoT Greengrass 엔드포인트를 나타냅니다. 다음 예제 코드 조각에서 이러한 속성은 ATS 엔드포인트를 지정합니다.

```
{
  "coreThing" : {
    ...
    "iotHost" : "abcde1234uwxyz-ats.iot.us-west-2.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
    ...
  },
```

AWS IoT Core 엔드포인트

적절한 `--endpoint-type` 파라미터와 함께 aws iot describe-endpoint CLI 명령을 실행하여 AWS IoT Core 엔드포인트를 가져올 수 있습니다.

- ATS 서명 엔드포인트를 반환하려면 다음을 실행합니다.

  ```
  aws iot describe-endpoint --endpoint-type iot:Data-ATS
  ```

- VeriSign 서명된 레거시 엔드포인트를 반환하려면 다음을 실행하십시오.

  ```
  aws iot describe-endpoint --endpoint-type iot:Data
  ```

AWS IoT Greengrass 엔드포인트

AWS IoT Greengrass 엔드포인트는 호스트 접두사가 greengrass로 대체된 `iotHost` 엔드포인트입니다. 예를 들어, ATS 서명 엔드포인트는 `greengrass-ats.iot.region.amazonaws.com`입니다. 이는 해당 AWS IoT Core 엔드포인트와 동일한 리전을 사용합니다.

포트 443에서 또는 네트워크 프록시를 통해 연결

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

Greengrass 코어는 TLS 클라이언트 인증으로 MQTT 메시징 프로토콜을 사용해 AWS IoT Core와 통신합니다. 일반적으로 MQTT over TLS는 포트 8883을 사용합니다. 그러나 보안 조치로서 제한적 환경하에서 작은 범위의 TCP 포트로의 인바운드 및 아웃바운드 트래픽이 제한될 수 있습니다. 예를 들어, 기업 방화벽은 HTTPS 트래픽용 포트 443을 열지만 MQTT 트래픽용 포트 8883과 같이 비교적 일반적이지 않은 프로토콜에 사용되는 기타 포트를 닫을 수 있습니다. 다른 제한적 환경에서는 모든 트래픽이 인터넷 연결 전에 HTTP 프록시를 경유해야 할 수 있습니다.

이와 같은 시나리오에서 통신을 활성화해야 할 경우 AWS IoT Greengrass에서는 다음 구성이 허용됩니다.

- 포트 443을 통해 TLS 클라이언트 인증을 사용하는 MQTT. 네트워크에서 포트 443에 대한 연결을 허용하는 경우 MQTT 트래픽에 기본 포트(8883) 대신 포트 443을 사용하도록 코어를 구성할 수 있습니다. 이는 포트 443으로 직접 연결되거나 네트워크 프록시 서버를 통해 연결되는 것일 수 있습니다.

  AWS IoT Greengrass는 ALPN(Application Layer Protocol Network) TLS 확장을 사용해 이러한 연결을 지원합니다. 기본 구성과 마찬가지로 포트 443에서의 MQTT over TLS는 인증서 기반 클라이언트 인증을 사용합니다.

  포트 443에 대한 직접 연결을 사용하도록 구성된 경우 코어는 AWS IoT Greengrass 소프트웨어에 대한 over-the-air (OTA) 업데이트를 지원합니다. 이 지원을 위해서는 AWS IoT Greengrass 코어 v1.9.3 이상이 필요합니다.
- 포트 443을 통한 HTTPS 통신. AWS IoT Greengrass는 기본적으로 포트 8443을 통해 HTTPS 트래픽을 보내지만 포트 443을 사용하도록 구성할 수 있습니다.
- 네트워크 프록시를 통한 연결. Greengrass 코어에 연결을 위한 중간 역할을 수행하도록 네트워크 프록시 서버를 구성할 수 있습니다. 기본 인증과 HTTP 및 HTTPS 프록시만 지원됩니다.

  프록시 구성은 `http_proxy`, `https_proxy` 및 `no_proxy` 환경 변수를 통해 사용자 정의 Lambda 함수로 전달됩니다. 프록시를 통해 연결하려면 사용자 정의 Lambda 함수가 이러한 전달된 설정을

사용해야 합니다. boto3 또는 cURL 및 python `requests` 패키지와 같은 연결을 만드는 데 Lambda 함수가 사용하는 공통 라이브러리는 일반적으로 이러한 환경 변수를 기본값으로 사용합니다. 또한 Lambda 함수가 이러한 동일한 환경 변수를 지정하는 경우 AWS IoT Greengrass가 이를 재정의하지 않습니다.

> **Important**
>
> 네트워크 프록시를 사용하도록 구성된 Greengrass 코어는 OTA 업데이트를 지원하지 않습니다.

포트 443을 통해 MQTT를 구성하려면

이 기능을 사용하려면 AWS IoT Greengrass 코어 v1.7 이상이 필요합니다.

이 절차를 통해 Greengrass 코어는 AWS IoT Core의 MQTT 메시징에 포트 443을 사용할 수 있습니다.

1. 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

2. su 사용자로서 편집을 하려면 *greengrass-root*`/config/config.json`을 엽니다.
3. 다음 예제에 나와 있는 것처럼 `coreThing` 객체에서 `iotMqttPort` 속성을 추가하고 그 값을 **443**으로 설정합니다.

   ```
   {
       "coreThing" : {
           "caPath" : "root.ca.pem",
           "certPath" : "12345abcde.cert.pem",
           "keyPath" : "12345abcde.private.key",
           "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
           "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
           "iotMqttPort" : 443,
           "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
           "keepAlive" : 600
       },
       ...
   }
   ```

4. 대몬(daemon)을 시작합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

포트 443을 통해 HTTPS를 구성하려면

이 기능을 사용하려면 AWS IoT Greengrass 코어 v1.8 이상이 필요합니다.

이 절차에서는 HTTPS 통신에 포트 443을 사용하도록 코어를 구성합니다.

1. 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

2. su 사용자로서 편집을 하려면 *greengrass-root*/config/config.json을 엽니다.
3. 다음 예제에 나와 있는 것처럼 coreThing 객체에서 iotHttpPort 및 ggHttpPort 속성을 추가합니다.

   ```
   {
       "coreThing" : {
           "caPath" : "root.ca.pem",
           "certPath" : "12345abcde.cert.pem",
           "keyPath" : "12345abcde.private.key",
           "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
           "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
           "iotHttpPort" : 443,
           "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
           "ggHttpPort" : 443,
           "keepAlive" : 600
       },
       ...
   }
   ```

4. 대몬(daemon)을 시작합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd start
   ```

네트워크 프록시를 구성하려면

이 기능을 사용하려면 AWS IoT Greengrass 코어 v1.7 이상이 필요합니다.

이 절차에서는 AWS IoT Greengrass가 HTTP 또는 HTTPS 네트워크 프록시를 통해 인터넷에 연결합니다.

1. 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. su 사용자로서 편집을 하려면 *greengrass-root*/config/config.json을 엽니다.
3. 다음 예제와 같이 coreThing 객체에서 networkProxy 객체를 추가합니다.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "keepAlive" : 600,
        "networkProxy": {
            "noProxyAddresses" : "http://128.12.34.56,www.mywebsite.com",
            "proxy" : {
                "url" : "https://my-proxy-server:1100",
                "username" : "Mary_Major",
                "password" : "pass@word1357"
            }
        }
    },
    ...
}
```

4. 대몬(daemon)을 시작합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

networkProxy 객체

networkProxy 객체를 사용하여 네트워크 프록시 관련 정보를 지정합니다. 이 객체에는 다음 속성이 있습니다.

필드	설명
noProxyAddresses	선택 사항으로, 프록시에서 제외되는 IP 주소 또는 호스트 이름을 쉼표로 구분한 목록입니다.

proxy	연결할 프록시입니다. 프록시에는 다음과 같은 속성이 있습니다.
	• url. scheme://userinfo@host:port 형식으로 표시되는 프록시 서버의 URL입니다.
	  • scheme. 체계입니다. http 또는 https이어야 합니다.
	  • userinfo. 선택 사항. 사용자 이름 및 암호 정보입니다. 지정한 경우 username 및 password 필드는 무시됩니다.
	  • host. 프록시 서버의 호스트 이름 또는 IP 주소입니다.
	  • port. 선택 사항. 포트 번호입니다. 지정되지 않은 경우 다음 기본값이 사용됩니다.
	  • http: 80
	  • https: 443
	• username. 선택 사항. 프록시 서버로 인증하는 데 사용되는 사용자 이름입니다.
	• password. 선택 사항. 프록시 서버로 인증하는 데 사용되는 암호입니다.

엔드포인트 허용

Greengrass 디바이스와 AWS IoT Core 또는 AWS IoT Greengrass 간의 통신은 인증이 필요합니다. 이 인증은 등록된 X.509 디바이스 인증서와 암호화 키를 기반으로 수행됩니다. 인증된 요청이 추가로 암호화되지 않고 프록시를 통해 전달되도록 하려면 다음 엔드포인트를 허용하십시오.

엔드포인트	포트	설명
`greengrass.` *`region`*`.amazonaws.com`	443	그룹 관리를 위한 제어 플레인 작업에 사용됩니다.
`prefix``-ats.iot.` *`region`*`.amazonaws.com` 또는 *`prefix`*`.iot.`*`region`*`.amazonaws.com`	MQTT: 8883 또는 443 HTTPS: 8443 또는 443	섀도우 동기화와 같이 디바이스 관리를 위한 데이터 플레인 작업에 사용됩니다. 코어 및 클라이언트 디바이스가 Amazon Trust Services 루트 CA 인증서(선호됨)를 사용하는지, 레거시 루트 CA 인증서를 사용하는지 또는 둘 다를 사용하는지에 따라 엔드포인트 하나 또는 둘 다를 사

엔드포인트	포트	설명
		용하도록 허용합니다. 자세한 설명은 the section called “서비스 엔드포인트는 인증서 유형과 일치해야 합니다” 섹션을 참조하세요.

엔드포인트	포트	설명

`greengrass-ats.iot. region.amazonaws.com`
 	
또는
 	
`greengrass.iot. region.amazonaws.com`
	8443 또는 443	디바이스 검색 작업에 사용됩니다.
		
		코어 및 클라이언트 디바이스가 Amazon Trust Services 루트 CA 인증서(선호됨)를 사용하는지, 레거시 루트 CA 인증서를 사용하는지 또는 둘 다를 사용하는지에 따라 엔드포인트 하나 또는 둘 다를 사용하도록 허용합니다. 자세한 설명은 the section called “서비스 엔드포인트는 인증서 유형과 일치해야 합니다” 섹션을 참조하세요.

엔드포인트	포트	설명
		Note 포트 443에서 연결하는 클라이언트는 ALPN(Application Layer Protocol Negotiation) TLS 확장을 구현하고 `ProtocolNameList` 에 `x-amzn-http-ca`를 `ProtocolName` 으로 전달해야 합니다. 자세

엔드포인트	포트	설명
		한 정보는 AWS IoT 개발자 안내서의 프로토콜을 참조하세요.
*.s3.amazonaws.com	443	배포 작업 및 over-the-air 업데이트에 사용됩니다. 엔드포인트는 내부적으로 제어되고 언제든 변경될 수 있으므로 이 형식에는 * 문자가 포함됩니다.

엔드포인트	포트	설명
`logs.region.amazonaws.com`	443	CloudWatch에 로그를 쓰도록 Greengrass 그룹을 구성한 경우 필요합니다.

AWS IoT Greengrass를 위한 쓰기 디렉터리 구성

이 기능은 AWS IoT Greengrass 코어 v1.6 이상에 사용할 수 있습니다.

기본적으로 AWS IoT Greengrass 코어 소프트웨어는 AWS IoT Greengrass이(가) 모든 읽기 및 쓰기 작업을 수행하는 단일 루트 디렉터리 아래에 배포됩니다. 하지만 디렉터리 및 파일 생성을 비롯한 모든 쓰기 작업에 대해 별도의 디렉터리를 사용하도록 AWS IoT Greengrass를 구성할 수도 있습니다. 이 경우 AWS IoT Greengrass는 2개의 최상위 디렉터리를 사용합니다.

- 하나는 읽기-쓰기로 남겨두거나 선택에 따라 읽기 전용으로 설정할 수 있는 *greengrass-root* 디렉터리입니다. 여기에는 AWS IoT Greengrass 코어 소프트웨어와 런타임 동안 변경이 불가능한 상태로 유지되어야 하는 기타 중요한 구성 요소(예: 인증서 및 `config.json`)가 포함되어 있습니다.
- 또 하나는 지정된 쓰기 디렉터리입니다. 여기에는 쓰기 가능한 내용(예: 로그, 상태 정보 및 배포된 사용자 정의 Lambda 함수)이 포함되어 있습니다.

이 구성은 다음과 같은 디렉터리 구조의 결과입니다.

Greengrass 루트 디렉터리

```
greengrass-root/
|-- certs/
|   |-- root.ca.pem
|   |-- hash.cert.pem
|   |-- hash.private.key
|   |-- hash.public.key
|-- config/
|   |-- config.json
|-- ggc/
```

```
|   |-- packages/
|       |-- package-version/
|           |-- bin/
|               |-- daemon
|           |-- greengrassd
|           |-- lambda/
|           |-- LICENSE/
|           |-- release_notes_package-version.html
|               |-- runtime/
|                   |-- java8/
|                   |-- nodejs8.10/
|                   |-- python3.8/
|   |-- core/
```

쓰기 디렉터리

```
write-directory/
|-- packages/
|   |-- package-version/
|       |-- ggc_root/
|       |-- rootfs_nosys/
|       |-- rootfs_sys/
|       |-- var/
|-- deployment/
|   |-- group/
|       |-- group.json
|   |-- lambda/
|   |-- mlmodel/
|-- var/
|   |-- log/
|   |-- state/
```

쓰기 디렉터리를 구성하려면

1. 다음 명령을 실행해 AWS IoT Greengrass 대몬(daemon)을 중단합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

2. su 사용자로서 편집을 하려면 *greengrass-root*/config/config.json을 엽니다.

3. 아래 예제에서와 같이 파라미터로 `writeDirectory`를 추가하고 대상 디렉터리에 대한 경로를 지정합니다.

```
{
    "coreThing": {
        "caPath": "root-CA.pem",
        "certPath": "hash.pem.crt",
        ...
    },
    ...
    "writeDirectory" : "/write-directory"
}
```

> **Note**
>
> 원하는 만큼 자주 `writeDirectory` 설정을 업데이트할 수 있습니다. 설정이 업데이트되고 나면 AWS IoT Greengrass은(는) 다음에 시작할 때 새로 지정된 쓰기 디렉터리를 사용하지만, 이전의 쓰기 디렉터리에서 내용을 마이그레이션하지는 않습니다.

4. 쓰기 디렉터리가 구성되면 이제 선택에 따라 *greengrass-root* 디렉터리를 읽기 전용으로 설정할 수 있습니다. 자세한 지침은 Greengrass 루트 디렉터리를 읽기 전용으로 설정하는 방법을 참조하십시오.

   그렇지 않으면 AWS IoT Greengrass 대몬(daemon)을 시작합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

Greengrass 루트 디렉터리를 읽기 전용으로 설정하려면

Greengrass 루트 디렉터리를 읽기 전용으로 설정하고 싶은 경우에만 이 단계를 수행합니다. 시작하기 전에 쓰기 디렉터리를 구성해야 합니다.

1. 필요한 디렉터리에 대한 액세스 권한을 부여하십시오:

   a. `config.json` 소유자에게 읽기 및 쓰기 권한을 제공합니다.

```
sudo chmod 0600 /greengrass-root/config/config.json
```

b. ggc_user를 인증서 및 시스템 Lambda 디렉터리의 소유자로 설정합니다.

```
sudo chown -R ggc_user:ggc_group /greengrass-root/certs/
sudo chown -R ggc_user:ggc_group /greengrass-root/ggc/packages/1.11.6/lambda/
```

> **Note**
>
> ggc_user 및 ggc_group 계정은 시스템 Lambda 함수를 실행하는 데 기본적으로 사용됩니다. 다른 계정을 사용하도록 그룹 레벨 기본 액세스 자격 증명을 구성한 경우 그 대신 해당 사용자(UID) 및 그룹(GID)에게 권한을 부여해야 합니다.

2. 원하는 메커니즘을 사용하여 *greengrass-root* 디렉터리를 읽기 전용으로 설정합니다.

> **Note**
>
> *greengrass-root* 디렉터리를 읽기 전용으로 설정하는 한 가지 방법은 디렉터리를 읽기 전용으로 마운트하는 것입니다. 하지만 마운트된 디렉터리의 AWS IoT Greengrass Core 소프트웨어에 over-the-air (OTA) 업데이트를 적용하려면 먼저 디렉터리를 마운트 해제한 다음 업데이트 후 다시 마운트해야 합니다. 이러한 `umount` 및 `mount` 작업들을 `ota_pre_update` 및 `ota_post_update` 스크립트에 추가할 수 있습니다. OTA 업데이트에 대한 자세한 내용은 the section called “Greengrass OTA 업데이트 에이전트” 및 the section called “OTA 업데이트가 지원되는 관리형 Respawn” 섹션을 참조하십시오.

3. 대몬(daemon)을 시작합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

1단계에서의 권한이 올바르게 설정되지 않으면 대몬(daemon)이 시작되지 않습니다.

MQTT 설정 구성

AWS IoT Greengrass 환경에서 로컬 디바이스, Lambda 함수, 커넥터 및 시스템 구성 요소는 서로 통신하고 AWS IoT Core와 통신할 수 있습니다. 모든 통신은 엔터티 간의 MQTT 통신을 인증하는 구독을 관리하는 코어를 거칩니다.

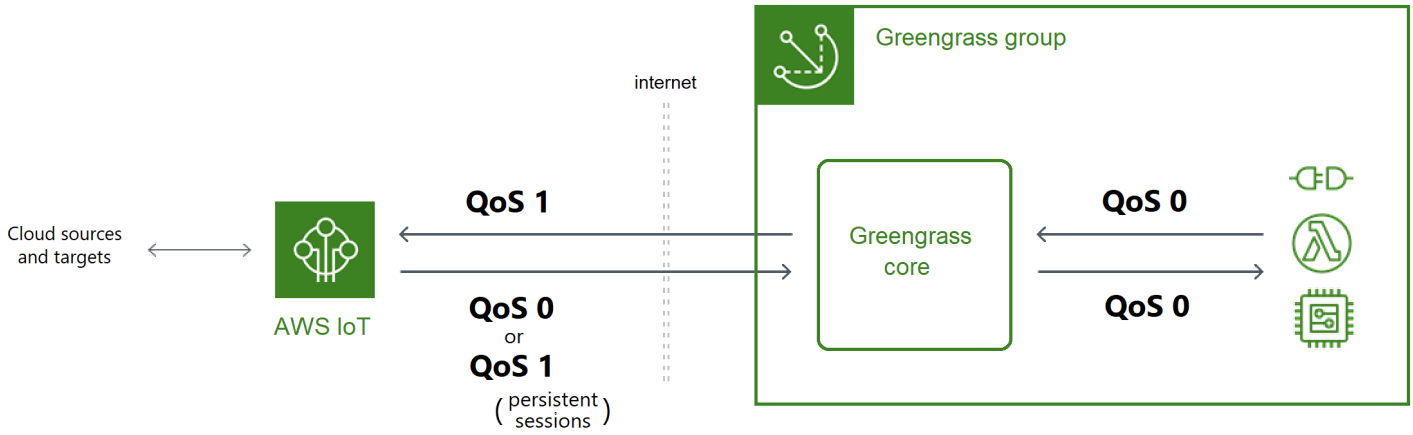
AWS IoT Greengrass에서 구성할 수 있는 MQTT 설정에 대한 자세한 내용은 다음 섹션을 참조하십시오.

- the section called “서비스 품질 메시지”
- the section called “MQTT 메시지 대기열”
- the section called “AWS IoT Core를 사용하는 MQTT 영구 세션”
- the section called “AWS IoT를 통한 MQTT 연결용 클라이언트 ID”
- 로컬 메시징을 위한 MQTT 포트
- the section called “AWS 클라우드를 통해 MQTT 연결 시 게시, 구독, 구독 취소 작업에 대한 제한 시간”

> **Note**
>
> OPC-UA는 산업 통신을 위한 정보 교환 표준입니다. Greengrass 코어에서 OPC-UA에 대한 지원을 구현하려면 IoT 커넥터를 사용할 수 있습니다. SiteWise 이 커넥터는 OPC-UA 서버의 산업용 디바이스 데이터를 AWS IoT SiteWise의 자산 속성으로 보냅니다.

서비스 품질 메시지

AWS IoT Greengrass은 구성과 통신 대상 및 방향에 따라 QoS(서비스 품질) 수준 0 또는 1을 지원합니다. Greengrass 코어는 클라이언트(AWS IoT Core와의 통신용)와 메시지 브로커(로컬 네트워크상의 통신용) 역할을 모두 수행합니다.



MQTT 및 QoS에 대한 자세한 내용은 MQTT 웹 사이트에서 시작하기를 참조하십시오.

AWS 클라우드와의 통신

- 아웃바운드 메시지는 QoS 1을 사용

  코어는 QoS 1을 사용하여 AWS 클라우드 대상으로 보내는 메시지를 전송합니다. AWS IoT Greengrass에서는 MQTT 메시지 대기열을 사용하여 이러한 메시지를 처리합니다. AWS IoT에서 메시지 전송이 확인되지 않으면 메시지가 나중에 재시도되도록 스풀링됩니다. 대기열이 가득 찼을 때는 메시지를 재시도할 수 없습니다. 메시지 전송 확인은 간헐적인 연결로 인한 데이터 손실을 최소화하는 데 도움이 될 수 있습니다.

  AWS IoT로 아웃바운드 메시지는 QoS 1을 사용하므로, Greengrass 코어가 메시지를 보낼 수 있는 최대 속도는 코어와 AWS IoT 사이의 지연 시간에 따라 달라집니다. 코어는 메시지를 보낼 때마다 AWS IoT가 메시지를 승인할 때까지 기다렸다가 다음 메시지를 전송합니다. 예를 들어 코어와 AWS 리전 사이의 왕복 시간이 50밀리초인 경우 코어는 초당 최대 20개의 메시지를 보낼 수 있습니다. 코어가 연결되는 AWS 리전을 선택할 때는 이 동작을 고려하세요. AWS 클라우드에 대용량 IoT 데이터를 수집하려면 스트림 관리자를 사용할 수 있습니다.

  AWS 클라우드 대상으로 보내는 메시지를 유지할 수 있도록 로컬 스토리지 캐시를 구성하는 방법을 비롯한 MQTT 메시지 대기열에 대한 자세한 내용은 the section called “MQTT 메시지 대기열” 섹션을 참조하세요.

- 인바운드 메시지는 QoS 0(기본값) 또는 QoS 1을 사용

  기본적으로 코어는 QoS 0로 AWS 클라우드 소스의 메시지를 구독합니다. 영구 세션을 활성화하면 코어가 QoS 1로 구독합니다. 이렇게 하면 간헐적인 연결로 인한 데이터 손실을 최소화할 수 있습니다. 이러한 구독에 대한 QoS를 관리하려면 로컬 스풀러 시스템 구성 요소에 대한 지속성 설정을 구성합니다.

  코어가 AWS 클라우드 대상에서 영구 세션을 설정하도록 활성화하는 방법을 포함한 자세한 내용은 the section called “AWS IoT Core를 사용하는 MQTT 영구 세션” 섹션을 참조하세요.

로컬 대상과의 통신

모든 로컬 통신은 QoS 0을 사용합니다. 코어는 Greengrass Lambda 함수, 커넥터 또는 클라이언트 디바이스 같은 로컬 대상에 메시지를 보내려고 시도합니다. 코어는 메시지를 저장하거나 전송 여부를 확인하지 않습니다. 메시지는 구성 요소 간 임의의 위치에서 끊길 수 있습니다.

> **Note**
>
> Lambda 함수 간의 직접 통신은 MQTT 메시징을 사용하지 않지만, 동작은 동일합니다.

클라우드 대상을 위한 MQTT 메시지 대기열

AWS 클라우드 대상을 목표로 하는 MQTT 메시지는 처리를 기다리기 위해 대기 상태가 됩니다. 대기 상태인 메시지는 FIFO(선입선출)에 따라 처리됩니다. 메시지가 처리되어 AWS IoT Core에 게시되면 해당 메시지가 대기열에서 제거됩니다.

기본적으로 Greengrass 코어는 AWS 클라우드 대상으로 보내는 처리되지 않은 메시지를 메모리에 저장합니다. 대신에 로컬 스토리지 캐시에 미처리 메시지를 저장하도록 코어를 구성할 수 있습니다. 인메모리 스토리지와 달리, 로컬 스토리지 캐시는 코어 재시작(예: 그룹 배포 또는 디바이스 재부팅 이후)에도 유지될 수 있어 AWS IoT Greengrass이(가) 가 계속해서 메시지를 처리할 수 있습니다. 또한 스토리지 크기를 구성할 수 있습니다.

> **Warning**
>
> Greengrass 코어는 MQTT 클라이언트가 오프라인 상태임을 감지하기 전에 게시 작업을 재시도하기 때문에, 연결이 끊어지면 중복된 MQTT 메시지를 대기열에 넣을 수 있습니다. 클라우드 대상에 대한 MQTT 메시지 중복을 방지하려면 코어의 `keepAlive` 값을 `mqttOperationTimeout` 값의 절반 미만으로 구성하십시오. 자세한 설명은 AWS IoT Greengrass 코어 구성 파일 섹션을 참조하세요.

AWS IoT Greengrass는 스풀러 시스템 구성 요소(`GGCloudSpooler` Lambda 함수)를 사용하여 메시지 대기열을 관리합니다. 다음 `GGCloudSpooler` 환경 변수를 사용하여 스토리지 설정을 구성할 수 있습니다.

- GG_CONFIG_STORAGE_TYPE. 메시지 대기열의 위치입니다. 유효한 값은 다음과 같습니다.
  - `FileSystem`. 미처리 메시지를 물리적 코어 디바이스의 디스크에 있는 로컬 스토리지 캐시에 저장합니다. 코어가 재시작될 때 대기 상태인 메시지가 처리를 위해 보관됩니다. 처리가 끝난 메시지는 삭제됩니다.
  - `Memory` (default). 처리되지 않은 메시지를 메모리에 저장합니다. 코어가 재시작될 때 대기 상태인 메시지가 사라집니다.

    이 옵션은 하드웨어 기능이 제한된 디바이스에 최적화되어 있습니다. 이 구성을 사용하는 경우에는 서비스 중단이 최소화될 때 그룹을 배포하거나 디바이스를 재시작하는 것이 좋습니다.

- GG_CONFIG_MAX_SIZE_BYTES. 스토리지 크기(바이트)입니다. 이 값은 262144 이상(256KB)인 음수가 아닌 정수면 어떤 것이든 가능하며, 크기가 더 작으면 AWS IoT Greengrass 코어 소프트웨어

가 시작되지 않습니다. 기본 크기는 2.5MB입니다. 크기가 한도에 도달하면 대기열에서 가장 오래된 메시지가 새로운 메시지로 대체됩니다.

> **Note**
>
> 이 기능은 AWS IoT Greengrass 코어 v1.6 이상에 사용할 수 있습니다. 이전 버전들은 대기열 크기가 2.5MB인 인 메모리 스토리지를 사용합니다. 이전 버전에서는 스토리지 설정을 구성할 수 없습니다.

로컬 스토리지의 캐시에 메시지를 저장하는 방법

파일 시스템에 메시지를 캐싱하여 코어 재시작에도 유지가 되도록 AWS IoT Greengrass을 구성할 수 있습니다. 이렇게 하려면 `GGCloudSpooler` 함수가 스토리지 유형을 `FileSystem`으로 설정하는 함수 정의 버전을 배포합니다. AWS IoT Greengrass API를 사용하여 로컬 스토리지 캐시를 구성해야 합니다. 콘솔에서는 이 작업을 수행할 수 없습니다.

다음 절차는 `create-function-definition-version` CLI 명령을 사용하여 파일 시스템에 대기 상태인 메시지를 저장하도록 스풀러를 구성합니다. 또한 2.6MB 대기열 크기를 구성합니다.

1. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

2. 출력에서 대상 그룹의 `Id` 및 `LatestVersion` 값을 복사합니다.
3. 최신 그룹 버전을 확인합니다.

- *group-id*를 복사한 Id로 바꿉니다.
- *latest-group-version-id*를 복사한 LatestVersion로 바꿉니다.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. 출력의 Definition 객체에서 CoreDefinitionVersionArn과 다른 모든 그룹 구성 요소의 ARN을 복사합니다(FunctionDefinitionVersionArn 제외). 새 그룹 버전을 만들 때 이러한 값들이 사용됩니다.
5. 출력의 FunctionDefinitionVersionArn에서 함수 정의의 ID를 복사합니다. 다음 예제에서와 같이, ARN에서 functions 세그먼트 다음에 나오는 GUID가 ID가 됩니다.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> 또는 create-function-definition 명령을 실행하여 함수 정의를 생성하고, 출력에서 ID를 복사할 수 있습니다.

6. 함수 정의에 함수 정의 버전을 추가합니다.

- 함수 정의를 위해 복사한 Id 것으로 *function-definition-id*바꾸십시오.
- 함수 이름 (예:) *arbitrary-function-id*으로 **spooler-function** 바꾸십시오.
- 이 버전에 포함시키고 싶은 Lambda 함수를 functions 배열에 추가합니다. get-function-definition-version 명령을 사용해 기존 함수 정의 버전에서 Greengrass Lambda 함수를 가져올 수 있습니다.

> **Warning**
>
> 262144보다 크거나 같은 GG_CONFIG_MAX_SIZE_BYTES에 대해 값을 지정해야 합니다. 크기가 더 작으면 AWS IoT Greengrass 코어 소프트웨어가 시작되지 않습니다.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
 "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
 {"Environment": {"Variables":
{"GG_CONFIG_MAX_SIZE_BYTES":"2621440","GG_CONFIG_STORAGE_TYPE":"FileSystem"}},"Executable":
 "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
function-id"}]'
```

> **Note**
>
> 이전에 GG_CONFIG_SUBSCRIPTION_QUALITY 환경 변수를 AWS IoT Core로 영구 세션을 지원하도록 설정한 경우 이 함수 인스턴스에 포함하십시오.

7. 출력에서 함수 정의 버전의 Arn를 복사합니다.
8. 시스템 Lambda 함수를 포함하는 그룹 버전을 만듭니다.

   - *group-id*를 그룹의 Id로 바꿉니다.
   - 최신 그룹 버전에서 복사한 것으로 *core-definition-version-arn*바꾸십시오. CoreDefinitionVersionArn
   - 새 함수 정의 버전용으로 복사한 것으로 *function-definition-version-arn*바꾸십시오. Arn
   - 최신 그룹 버전에서 복사한 다른 그룹 구성 요소의 ARN(예: SubscriptionDefinitionVersionArn 또는 DeviceDefinitionVersionArn)을 바꿉니다.
   - 사용되지 않은 파라미터를 모두 제거합니다. 예를 들어 그룹 버전에 리소스가 포함되어 있지 않으면 --resource-definition-version-arn을 제거합니다.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
```

```
--subscription-definition-version-arn subscription-definition-version-arn
```

9. 출력에서 `Version`을 복사합니다. 새 그룹 버전의 ID가 이 값이 됩니다.
10. 새로운 그룹 버전을 사용하여 그룹을 배포합니다.

    - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
    - 새 그룹 버전용으로 복사한 것으로 *group-version-id*바꾸십시오. `Version`

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

스토리지 설정을 업데이트하려면 AWS IoT Greengrass API를 사용해 구성이 업데이트된 `GGCloudSpooler` 함수를 포함하는 새로운 함수 정의 버전을 생성합니다. 그런 다음, 새 그룹 버전에 함수 정의 버전을 추가하고(다른 그룹 구성 요소와 함께) 그룹 버전을 배포합니다. 기본 구성을 복원하고 싶은 경우에는 `GGCloudSpooler` 함수가 포함되지 않은 함수 정의 버전을 배포할 수 있습니다.

이 시스템 Lambda 함수는 콘솔에 표시되지 않습니다. 하지만 함수가 최신 그룹 버전에 추가되고 난 후에는 콘솔에서 만든 배포에 포함이 됩니다(API를 사용해 이를 교체 또는 제거하지 않는 한).

AWS IoT Core를 사용하는 MQTT 영구 세션

이 기능은 AWS IoT Greengrass 코어 v1.10 이상에 사용할 수 있습니다.

Greengrass 코어는 AWS IoT 메시지 브로커를 사용해 영구 세션을 설정할 수 있습니다. 영구 세션은 코어가 오프라인 상태에서도 전송된 메시지를 수신할 수 있도록 해주는 지속적인 연결입니다. 코어는 연결 중인 클라이언트입니다.

영구 세션에서 AWS IoT 메시지 브로커는 연결 중에 코어에서 생성된 모든 구독을 저장합니다. 코어의 연결이 끊어지면 AWS IoT 메시지 브로커는 QoS 1로 게시되고 Lambda 함수 및 클라이언트 디바이스와 같은 로컬 대상으로 보내는 승인되지 않은 새 메시지를 저장합니다. 코어가 다시 연결되면 영구 세션이 재개되고 AWS IoT가 초당 최대 10개의 메시지 속도로 저장된 메시지를 코어로 보냅니다. 영구 세션의 기본 만료 기간은 1시간입니다. 이 기간은 메시지 브로커가 코어의 연결이 끊어졌음을 감지할 때 시작됩니다. 자세한 내용은 AWS IoT 개발자 안내서의 MQTT 영구 세션을 참조하세요.

AWS IoT Greengrass는 스풀러 시스템 구성 요소(`GGCloudSpooler` Lambda 함수)를 사용하여 소스로 AWS IoT가 있는 구독을 생성합니다. 다음 `GGCloudSpooler` 환경 변수를 사용하여 영구 세션을 구성할 수 있습니다.

- GG_CONFIG_SUBSCRIPTION_QUALITY. 소스로 AWS IoT를 가지고 있는 구독의 품질입니다. 유효한 값은 다음과 같습니다.
  - `AtMostOnce` (default). 영구 세션을 비활성화합니다. 구독은 QoS 0을 사용합니다.
  - `AtLeastOncePersistent`. 영구 세션을 활성화합니다. `CONNECT` 메시지에서 `cleanSession` 플래그를 `0`으로 설정하고 QoS 1로 구독합니다.

    코어가 수신하는 QoS 1로 게시된 메시지는 Greengrass 대몬(daemon)의 인 메모리 작업 대기열에 도달하도록 보장됩니다. 코어는 대기열에 추가된 이후에 메시지를 확인합니다. 대기열에서 로컬 대상(예: Greengrass Lambda 함수, 커넥터 또는 디바이스)과의 후속 통신은 QoS 0으로 전송됩니다. AWS IoT Greengrass는 로컬 대상에 대한 전달을 보장하지 않습니다.

    > **Note**
    >
    > maxWorkItemCount 구성 속성을 사용하여 작업 항목 대기열의 크기를 제어할 수 있습니다. 예를 들어 워크로드에 사용률이 높은 MQTT 트래픽이 필요한 경우 대기열 크기를 늘릴 수 있습니다.

    영구 세션이 활성화되면 코어는 MQTT 메시지를 AWS IoT와 교환하기 위해 적어도 하나의 추가 연결을 엽니다. 자세한 설명은 the section called “AWS IoT를 통한 MQTT 연결용 클라이언트 ID” 섹션을 참조하세요.

MQTT 영구 세션을 구성하려면

AWS IoT Core에서 영구 세션을 사용하도록 AWS IoT Greengrass를 구성할 수 있습니다. 이렇게 하려면 `GGCloudSpooler` 함수가 구독 품질을 `AtLeastOncePersistent`로 설정하는 함수 정의 버전을 배포합니다. 이 설정은 소스로 AWS IoT Core(`cloud`)를 보유한 모든 구독에 적용됩니다. 영구 세션을 구성하려면 AWS IoT Greengrass API를 사용해야 합니다. 콘솔에서는 이 작업을 수행할 수 없습니다.

다음 절차에서는 `create-function-definition-version` CLI 명령을 통해 영구 세션을 사용하도록 스풀러를 구성합니다. 이 절차에서는 기존 그룹의 최신 그룹 버전에 대한 구성을 업데이트한다고 가정합니다.

1. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

> **Note**
>
> AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

2. 출력에서 대상 그룹의 `Id` 및 `LatestVersion` 값을 복사합니다.
3. 최신 그룹 버전을 확인합니다.

   - *group-id*를 복사한 `Id`로 바꿉니다.
   - *latest-group-version-id*를 복사한 `LatestVersion`로 바꿉니다.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. 출력의 `Definition` 객체에서 `CoreDefinitionVersionArn`과 다른 모든 그룹 구성 요소의 ARN을 복사합니다(`FunctionDefinitionVersionArn` 제외). 새 그룹 버전을 만들 때 이러한 값들이 사용됩니다.
5. 출력의 `FunctionDefinitionVersionArn`에서 함수 정의의 ID를 복사합니다. 다음 예제에서와 같이, ARN에서 `functions` 세그먼트 다음에 나오는 GUID가 ID가 됩니다.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> 또는 create-function-definition 명령을 실행하여 함수 정의를 생성하고, 출력에서 ID를 복사할 수 있습니다.

6. 함수 정의에 함수 정의 버전을 추가합니다.

   - 함수 정의를 위해 복사한 것으로 *function-definition-id*바꾸십시오. `Id`
   - 함수 이름 (예:) *arbitrary-function-id*으로 **spooler-function** 바꾸십시오.
   - 이 버전에 포함시키고 싶은 Lambda 함수를 `functions` 배열에 추가합니다. get-function-definition-version 명령을 사용해 기존 함수 정의 버전에서 Greengrass Lambda 함수를 가져올 수 있습니다.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
 "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
 {"Environment": {"Variables":
{"GG_CONFIG_SUBSCRIPTION_QUALITY":"AtLeastOncePersistent"}},"Executable":
 "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
function-id"}]'
```

> **Note**
>
> 이전에 `GG_CONFIG_STORAGE_TYPE` 또는 `GG_CONFIG_MAX_SIZE_BYTES` 환경 변수를 설정하여 스토리지 설정을 정의한 경우 이 함수 인스턴스에 포함하십시오.

7. 출력에서 함수 정의 버전의 `Arn`를 복사합니다.
8. 시스템 Lambda 함수를 포함하는 그룹 버전을 만듭니다.

   - *group-id*를 그룹의 `Id`로 바꿉니다.
   - 최신 그룹 버전에서 복사한 것으로 *core-definition-version-arn*바꾸십시오. `CoreDefinitionVersionArn`
   - 새 함수 정의 버전용으로 복사한 것으로 *function-definition-version-arn*바꾸십시오. `Arn`

- 최신 그룹 버전에서 복사한 다른 그룹 구성 요소의 ARN(예: `SubscriptionDefinitionVersionArn` 또는 `DeviceDefinitionVersionArn`)을 바꿉니다.
- 사용되지 않은 파라미터를 모두 제거합니다. 예를 들어 그룹 버전에 리소스가 포함되어 있지 않으면 `--resource-definition-version-arn`을 제거합니다.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. 출력에서 `Version`을 복사합니다. 새 그룹 버전의 ID가 이 값이 됩니다.
10. 새로운 그룹 버전을 사용하여 그룹을 배포합니다.

    - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
    - 새 그룹 버전용으로 복사한 것으로 *group-version-id*바꾸십시오. `Version`

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

11. (선택 사항) 핵심 구성 파일의 maxWorkItem개수 속성을 늘립니다. 이렇게 하면 코어가 증가한 MQTT 트래픽 및 로컬 대상과의 통신을 처리하는 데 도움이 될 수 있습니다.

이러한 구성 변경 사항을 반영해 코어를 업데이트하려면 AWS IoT Greengrass API를 사용해 구성이 업데이트된 `GGCloudSpooler` 함수를 포함하는 새로운 함수 정의 버전을 생성합니다. 그런 다음, 새 그룹 버전에 함수 정의 버전을 추가하고(다른 그룹 구성 요소와 함께) 그룹 버전을 배포합니다. 기본 구성을 복원하고 싶은 경우에는 `GGCloudSpooler` 함수가 포함되지 않은 함수 정의 버전을 생성할 수 있습니다.

이 시스템 Lambda 함수는 콘솔에 표시되지 않습니다. 하지만 함수가 최신 그룹 버전에 추가되고 난 후에는 콘솔에서 만든 배포에 포함이 됩니다(API를 사용해 이를 교체 또는 제거하지 않는 한).

AWS IoT를 통한 MQTT 연결용 클라이언트 ID

이 기능은 AWS IoT Greengrass 코어 v1.8 이상에 사용할 수 있습니다.

Greengrass 코어는 섀도우 동기화 및 인증서 관리와 같은 작업을 위해 AWS IoT Core를 통해 MQTT 연결을 엽니다. 이러한 연결의 경우 코어는 코어 사물 이름을 기반으로 예측 가능한 클라이언트 ID를 생성합니다. AWS IoT Device Defender 및 AWS IoT 수명 주기 이벤트를 포함한 모니터링, 감사 및 요금 기능에 예측 가능한 클라이언트 ID를 사용할 수 있습니다. 또한 예상 가능한 클라이언트 ID를 중심으로 논리를 생성할 수 있습니다(예: 인증서 속성을 기반으로 한 구독 정책 템플릿).

GGC v1.9 and later

두 Greengrass 시스템 구성 요소에서 AWS IoT Core를 통해 MQTT 연결을 엽니다. 이러한 구성 요소는 다음 패턴을 사용하여 연결을 위한 클라이언트 ID를 생성합니다.

Operation	클라이언트 ID 패턴
배포	*core-thing-name* 예제: `MyCoreThing` 연결, 연결 해제, 구독 및 구독 해지 수명 주기 이벤트 알림에 이 클라이언트 ID를 사용합니다.
구독	*core-thing-name* `-c`*n* 예제: `MyCoreThing-c01` *n*은 00에서 시작하고 새 연결에 따라 최대 250까지 증분하는 정수입니다. 연결 수는 섀도우 상태를 AWS IoT Core(그룹당 최대 2,500개)와 동기화하는 디바이스 수 및 `cloud`를 그룹의 해당 소스로 포함하는 구독 수(그룹당 최대 10,000개)로 결정됩니다. 스풀러 시스템 구성 요소는 AWS IoT Core와의 연결을 설정하여 클라우드 소스 또는 대상과 구독에 대한 메시지를 교환합니다. 또한 스풀러는 AWS IoT Core와 로컬 섀도우 서비스

Operation	클라이언트 ID 패턴
	및 디바이스 인증서 관리자 간의 메시지 교환을 위한 프록시로 작동합니다.

그룹당 MQTT 연결 수를 계산하려면 다음 수식을 사용하십시오.

```
number of MQTT connections per group = number of connections for
Deployment Agent + number of connections for Subscriptions
```

여기서

- 배포 에이전트의 연결 수 = 1.
- 구독의 연결 수 = `(2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50`.
  - 여기서, `50` = AWS IoT Core가 지원할 수 있는 연결당 최대 구독 수입니다.

> **Note**
>
> AWS IoT Core 구독을 위해 영구 세션을 활성화하면 코어가 영구 세션에서 사용할 추가 연결을 하나 이상 엽니다. 시스템 구성 요소는 영구 세션을 지원하지 않으므로 해당 연결을 공유할 수 없습니다.

MQTT 연결 수를 줄이고 비용을 절감하기 위해 로컬 Lambda 함수를 사용하여 엣지에서 데이터를 집계할 수 있습니다. 그런 다음 집계된 데이터를 AWS 클라우드로 전송합니다. 따라서 AWS IoT Core에서 사용하는 MQTT 항목이 줄어듭니다. 자세한 내용은 AWS IoT Greengrass 요금을 참조하세요.

GGC v1.8

여러 Greengrass 시스템 구성 요소에서 AWS IoT Core를 통해 MQTT 연결이 열립니다. 이러한 구성 요소는 다음 패턴을 사용하여 연결을 위한 클라이언트 ID를 생성합니다.

Operation	클라이언트 ID 패턴
배포	*core-thing-name*

Operation	클라이언트 ID 패턴
	예제: `MyCoreThing` 연결, 연결 해제, 구독 및 구독 해지 수명 주기 이벤트 알림에 이 클라이언트 ID를 사용합니다.
AWS IoT Core를 통한 MQTT 메시지 교환	*`core-thing-name`* `-spr` 예제: `MyCoreThing-spr`
섀도우 동기화	*`core-thing-name`* `-s`*`nn`* 예제: `MyCoreThing-s01` *`nn`*은 00에서 시작하고 새 연결에 따라 최대 03까지 증분하는 정수입니다. 연결 수는 섀도우 상태를 AWS IoT Core(연결당 최대 50개 구독)와 동기화하는 디바이스 수(그룹당 최대 200개 디바이스)로 결정됩니다.
디바이스 인증서 관리	*`core-thing-name`* `-dcm` 예제: `MyCoreThing-dcm`

> **Note**
>
> 중복 ID를 동시 연결에 사용하면 무한 연결-연결 해제 루프가 발생할 수 있습니다. 이는 다른 디바이스가 연결에서 코어 디바이스 이름을 클라이언트 ID로 사용하도록 하드코딩되는 경우 발생할 수 있습니다. 자세한 내용은 이 문제 해결 단계를 참조하십시오.

또한 Greengrass 디바이스는 AWS IoT Device Management의 플릿 인덱싱 서비스와 완전 통합됩니다. 따라서 클라우드에서 디바이스 속성, 섀도우 상태, 연결 상태를 기반으로 디바이스를 인덱싱하고 검색할 수 있습니다. 예를 들어, 디바이스 연결 인덱싱을 사용하여 AWS IoT Core에 현재 연결되거나 연결 해제된 Greengrass 디바이스를 검색할 수 있도록 Greengrass 디바이스는 사물 이름을 클라이언트 ID로 사용하는 연결을 하나 이상 설정합니다. 자세한 내용은 AWS IoT 개발자 안내서의 플릿 인덱싱을 참조하세요.

로컬 메시징을 위한 MQTT 포트 구성

이 기능을 사용하려면 AWS IoT Greengrass 코어 v1.10 이상이 필요합니다.

Greengrass 코어는 로컬 Lambda 함수, 커넥터 및 클라이언트 디바이스 간의 MQTT 메시징을 위한 로컬 메시지 브로커 역할을 수행합니다. 기본적으로 코어는 로컬 네트워크의 MQTT 트래픽에 포트 8883을 사용합니다. 포트 8883에서 실행되는 다른 소프트웨어와의 충돌을 피하기 위해 포트를 변경할 수 있습니다.

코어가 로컬 MQTT 트래픽에 사용하는 포트 번호를 구성하려면

1. 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

2. su 사용자로서 편집을 하려면 *greengrass-root*/config/config.json을 엽니다.
3. coreThing 객체에서 ggMqttPort 속성을 추가하고 사용할 포트 번호로 값을 설정합니다. 유효한 값은 1024 ~ 65535입니다. 다음 예제에서는 포트 번호를 9000으로 설정합니다.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "ggMqttPort" : 9000,
        "keepAlive" : 600
    },
    ...
}
```

4. 대몬(daemon)을 시작합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

5. 코어에 대해 자동 IP 감지가 활성화된 경우에는 구성이 완료된 것입니다.

자동 IP 감지가 활성화되지 않은 경우 코어의 연결 정보를 업데이트해야 합니다. 이렇게 하면 클라이언트 디바이스가 검색 작업 중에 올바른 포트 번호를 수신하여 코어 연결 정보를 얻을 수 있습니다. AWS IoT 콘솔 또는 AWS IoT Greengrass API를 사용하여 코어 연결 정보를 업데이트할 수 있습니다. 이 절차에서는 포트 번호만 업데이트합니다. 코어의 로컬 IP 주소는 동일하게 유지됩니다.

코어에 대한 연결 정보를 업데이트하려면(콘솔)

1. 그룹 구성 페이지에서 Greengrass 코어를 선택합니다.
2. 코어 세부 정보 페이지에서 MQTT 브로커 엔드포인트 탭을 선택합니다.
3. 엔드포인트 관리를 선택한 다음 엔드포인트 추가를 선택합니다.
4. 현재 로컬 IP 주소와 새 포트 번호를 입력합니다. 다음 예제에서는 IP 주소 `192.168.1.8`의 포트 번호 `9000`를 설정합니다.
5. 사용되지 않는 엔드포인트를 제거한 다음 업데이트를 선택합니다.

코어에 대한 연결 정보를 업데이트하려면(API)

- UpdateConnectivityInfo 작업을 사용합니다. 다음 예제에서는 AWS CLI에서 `update-connectivity-info`를 사용하여 IP 주소 `192.168.1.8`의 포트 번호 `9000`를 설정합니다.

```
aws greengrass update-connectivity-info \
    --thing-name "MyGroup_Core" \
    --connectivity-info "[{\"Metadata\":\"\",\"PortNumber\":9000,
\"HostAddress\":\"192.168.1.8\",\"Id\":\"localIP_192.168.1.8\"},{\"Metadata
\":\"\",\"PortNumber\":8883,\"HostAddress\":\"127.0.0.1\",\"Id\":
\"localhost_127.0.0.1_0\"}]"
```

> **Note**
>
> AWS IoT Core를 사용하여 코어가 MQTT 메시징에 사용하는 포트를 구성할 수도 있습니다. 자세한 설명은 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.

AWS 클라우드를 통해 MQTT 연결 시 게시, 구독, 구독 취소 작업에 대한 제한 시간

이 기능은 AWS IoT Greengrass v1.10.2 이상에서 사용할 수 있습니다.

Greengrass 코어는 AWS IoT Core에 MQTT 연결 시 게시, 구독 또는 구독 취소 작업을 완료할 수 있는 시간(초)을 구성할 수 있습니다. 대역폭 제약 또는 긴 지연 시간으로 인해 작업 시간이 초과되는 경우 이 설정을 조정할 수 있습니다. config.json 파일에서 이 설정을 구성하려면 `coreThing` 객체의 `mqttOperationTimeout` 속성을 추가하거나 변경합니다. 예:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

기본 제한 시간은 5초입니다. 최소 제한 시간은 5초입니다.

자동 IP 감지 활성화

Greengrass 그룹의 클라이언트 장치가 Greengrass 코어를 자동으로 검색하도록 AWS IoT Greengrass를 구성할 수 있습니다. 활성화되면 코어는 IP 주소 변경을 감시합니다. 주소가 변경되면 코어는 업데이트된 주소 목록을 게시합니다. 이러한 주소는 코어와 동일한 Greengrass 그룹에 속한 클라이언트 디바이스에서 사용할 수 있습니다.

> **Note**
>
> 클라이언트 디바이스에 대한 AWS IoT 정책은 디바이스가 코어에 대한 연결 정보를 검색할 수 있도록 허용하는 `greengrass:Discover` 권한을 부여해야 합니다. 이 정책 설명에 대한 자세한 내용은 the section called “검색 권한” 섹션을 참조하십시오.

AWS IoT Greengrass 콘솔에서 이 기능을 활성화하려면 Greengrass 그룹을 처음 배포할 때 자동 감지를 선택합니다. Lambda 함수 탭을 선택하고 IP 감지기를 선택하여 그룹 구성 페이지에서 이 기능을 활성화하거나 비활성화할 수도 있습니다. MQTT 브로커 엔드포인트 자동 감지 및 재정의가 선택된 경우 자동 IP 감지가 활성화됩니다.

AWS IoT Greengrass API로 자동 검색을 관리하려면 `IPDetector` 시스템 Lambda 함수를 구성해야 합니다. 다음 절차는 create-function-definition-versionCLI 명령을 사용하여 Greengrass 코어의 자동 검색을 구성하는 방법을 보여줍니다.

1. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

2. 출력에서 대상 그룹의 `Id` 및 `LatestVersion` 값을 복사합니다.
3. 최신 그룹 버전을 확인합니다.

   - *group-id*를 복사한 `Id`로 바꿉니다.
   - *latest-group-version-id*를 복사한 `LatestVersion`로 바꿉니다.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. 출력의 `Definition` 객체에서 `CoreDefinitionVersionArn`과 다른 모든 그룹 구성 요소의 ARN을 복사합니다(`FunctionDefinitionVersionArn` 제외). 새 그룹 버전을 만들 때 이러한 값들이 사용됩니다.
5. 출력의 `FunctionDefinitionVersionArn`에서 함수 정의의 ID와 함수 정의 버전을 복사합니다.

```
arn:aws:greengrass:region:account-id:/greengrass/groups/function-definition-id/
versions/function-definition-version-id
```

> **Note**
>
> 선택에 따라 create-function-definition 명령을 실행하여 함수 정의를 만들고, 출력에서 ID를 복사할 수 있습니다.

6. get-function-definition-version 명령을 사용하여 현재 정의 상태를 가져옵니다. 복사한 *function-definition-id*내용을 함수 정의에 사용하십시오. 예를 들면 다음과 같습니다 *4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*.

```
aws greengrass get-function-definition-version
--function-definition-id function-definition-id
--function-definition-version-id function-definition-version-id
```

나열된 함수 구성을 적어 두십시오. 현재 정의 설정의 손실을 방지하기 위해 새 함수 정의 버전을 생성할 때 이 함수 구성을 포함시켜야 합니다.

7. 함수 정의에 함수 정의 버전을 추가합니다.

   - 함수 정의를 위해 복사한 `Id` 것으로 *function-definition-id*바꾸십시오. 예를 들면 다음과 같습니다 *4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*.
   - 함수 이름 (예:) *arbitrary-function-id*으로 **auto-detection-function** 바꾸십시오.
   - 이 버전에 포함시키고 싶은 모든 Lambda 함수(예: 이전 단계에서 나열된 함수)를 `functions` 배열에 추가합니다.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions
 '[{"FunctionArn":"arn:aws:lambda:::function:GGIPDetector:1","Id":"arbitrary-
function-id","FunctionConfiguration":
{"Pinned":true,"MemorySize":32768,"Timeout":3}}]'\
--region us-west-2
```

8. 출력에서 함수 정의 버전의 `Arn`를 복사합니다.
9. 시스템 Lambda 함수를 포함하는 그룹 버전을 만듭니다.

- *group-id*를 그룹의 Id로 바꿉니다.
- 최신 그룹 버전에서 복사한 것으로 *core-definition-version-arn*바꾸십시오. CoreDefinitionVersionArn
- 새 함수 정의 버전용으로 복사한 것으로 *function-definition-version-arn*바꾸십시오. Arn
- 최신 그룹 버전에서 복사한 다른 그룹 구성 요소의 ARN(예: SubscriptionDefinitionVersionArn 또는 DeviceDefinitionVersionArn)을 바꿉니다.
- 사용되지 않은 파라미터를 모두 제거합니다. 예를 들어 그룹 버전에 리소스가 포함되어 있지 않으면 --resource-definition-version-arn을 제거합니다.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

10. 출력에서 Version을 복사합니다. 새 그룹 버전의 ID가 이 값이 됩니다.
11. 새로운 그룹 버전을 사용하여 그룹을 배포합니다.

    - *group-id*를 해당 그룹에서 복사한 Id로 바꿉니다.
    - 새 그룹 버전용으로 복사한 것으로 *group-version-id*바꾸십시오. Version

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Greengrass 코어의 IP 주소를 수동으로 입력하려면 IPDetector 함수가 포함되지 않은 다른 함수 정의로 이 튜토리얼을 완료할 수 있습니다. 다음과 같이 하면 감지 함수가 Greengrass 코어 IP 주소를 찾아서 자동으로 입력할 수 없습니다.

이 시스템 Lambda 함수는 Lambda 콘솔에 표시되지 않습니다. 함수가 최신 그룹 버전에 추가된 후에는 API를 사용하여 이를 교체하거나 제거하지 않는 한 콘솔에서 만든 배치에 포함됩니다.

Greengrass 대몬(daemon)을 시작하도록 init 시스템 구성

특히 대규모 디바이스 집합을 관리할 때는 부팅 동안 Greengrass 대몬(daemon)을 시작하도록 init 시스템을 설정하는 것이 좋습니다.

> **Note**
>
> apt를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치한 경우 systemd 스크립트를 사용하여 부팅 시 시작을 활성화할 수 있습니다. 자세한 설명은 the section called “systemd 스크립트를 사용하여 Greengrass 대몬(daemon) 수명 주기 관리” 섹션을 참조하세요.

initd, systemd 및 SystemV 같이 다양한 유형의 init 시스템이 있으며, 이들은 비슷한 구성 파라미터를 사용합니다. 다음은 systemd를 위한 서비스 파일의 예입니다. greengrassd(Greengrass를 시작하는 데 사용)는 Greengrass 대몬(daemon) 프로세스를 분기시키기 때문에 `Type` 파라미터가 `forking`로 설정되었고, Greengrass가 실패 상태가 되면 systemd에 Greengrass 재시작을 지시하도록`Restart` 파라미터가 `on-failure`로 설정되었습니다.

> **Note**
>
> 디바이스가 systemd를 사용하는지 알아보려면 모듈 1에서 `check_ggc_dependencies` 스크립트를 실행합니다. 그런 다음 systemd를 사용하려면 `config.json`에서 `useSystemd` 파라미터가 `yes`로 설정되어 있어야 합니다.

```
[Unit]
Description=Greengrass Daemon

[Service]
Type=forking
PIDFile=/var/run/greengrassd.pid
Restart=on-failure
ExecStart=/greengrass/ggc/core/greengrassd start
ExecReload=/greengrass/ggc/core/greengrassd restart
ExecStop=/greengrass/ggc/core/greengrassd stop
```

```
[Install]
WantedBy=multi-user.target
```

다음 사항도 참조하십시오.

- AWS IoT Greengrass(이)란 무엇인가요?
- the section called “지원되는 플랫폼 및 요구 사항”
- 시작하기: AWS IoT Greengrass
- the section called “그룹 객체 모델 개요”
- the section called “하드웨어 보안 통합”

AWS IoT Greengrass Version 1 유지 관리 정책

이 AWS IoT Greengrass V1 유지 관리 정책을 사용하여 AWS IoT Greengrass V1 서비스 및 AWS IoT Greengrass 코어 소프트웨어 v1.x의 다양한 유지 관리 및 업데이트 수준을 이해합니다.

주제

AWS IoT Greengrass 버전 관리 체계

AWS IoT Greengrass은(는) AWS IoT Greengrass 코어 소프트웨어의 시맨틱 버저닝을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 이전 메이저 버전과 역호환되지 않는 기능 및 API 변경 시 메이저 버전이 증분됩니다. 이전 버전과 호환되는 새로운 기능을 추가하는 릴리스의 경우, 마이너 버전이 증분됩니다. 보안 패치 또는 버그 수정에 대한 패치 버전이 증분됩니다. 첫 번째 메이저 릴리스인 v1.0.0부터 AWS IoT Greengrass은(는) AWS IoT Greengrass 코어 소프트웨어 v1.x의 11개 마이너 버전을 릴리스하였으며, 여기서 v1.11.6이 최신 릴리스입니다. 새로운 기능, 향상된 기능 및 버그 수정이 활용되도록 AWS IoT Greengrass 코어 소프트웨어를 최신 버전으로 업데이트하는 것이 좋습니다.

2020년 12월, AWS IoT Greengrass은(는) 첫 번째 메이저 버전 업데이트를 릴리스했습니다. 이 업데이트에는 AWS IoT Greengrass 코어 소프트웨어의 AWS IoT Greengrass V2 서비스 및 버전 2.0.3이 포함되었습니다. 새 애플리케이션의 경우, AWS IoT Greengrass Version 2와(과) AWS IoT Greengrass 코어 소프트웨어 v2.x를 사용하는 것이 좋습니다. 버전 2에는 새로운 기능과 모든 주요 V1 기능이 포함되며 추가 플랫폼과 대규모 디바이스로의 지속적인 배포가 지원됩니다. 자세한 내용은 AWS IoT Greengrass V2란 무엇인가요?를 참조하세요.

AWS IoT Greengrass 코어 소프트웨어의 주요 버전에 대한 수명주기 단계

AWS IoT Greengrass 코어 소프트웨어의 각 주요 버전에는 다음과 같은 세 가지 순차적 수명주기 단계가 있습니다. 각 라이프사이클 단계는 최초 출시일 이후 일정 기간 동안 서로 다른 수준의 유지 관리를 제공합니다.

- 릴리스 단계 — AWS IoT Greengrass은(는) 다음과 같은 업데이트를 릴리스할 수 있습니다.
  - 기존 기능에 새로운 기능이나 향상된 기능을 제공하는 마이너 버전 업데이트
  - 보안 패치 및 버그 수정이 제공되는 패치 버전 업데이트
- 유지 관리 단계 - AWS IoT Greengrass은(는) 보안 패치와 버그 수정을 제공하는 패치 버전 업데이트를 릴리스할 수 있습니다. AWS IoT Greengrass은(는) 유지 관리 단계에서는 새 기능이나 기존 기능의 개선 사항을 릴리스하지 않습니다.
- 수명 연장 단계 — AWS IoT Greengrass은(는) 기능을 제공하는 업데이트, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 릴리스하지 않습니다. 하지만 AWS 클라우드 엔드포인트 및 API 작업은 AWS IoT Greengrass 서비스 수준 계약에 따라 계속 사용 가능하고 운영됩니다. AWS IoT Greengrass 코어 소프트웨어 v1.x를 실행하는 디바이스는 AWS 클라우드에 계속 연결하여 작동할 수 있습니다.

AWS IoT Greengrass의 메이저 버전의 수명 연장 단계가 종료되면 AWS 클라우드 엔드포인트 및 API 작업은 더 이상 사용되지 않으며 더 이상 사용할 수 없게 됩니다. AWS IoT Greengrass 코어 소프트웨어 v1.x를 실행하는 디바이스는 AWS 클라우드 서비스에 연결하여 작동할 수 없습니다.

AWS IoT Greengrass 코어 소프트웨어 유지 관리 정책

AWS IoT Greengrass 코어 소프트웨어 v1.x는 2023년 6월 30일에 수명 연장 단계에 들어갔습니다. 이 날짜 이후에도 AWS IoT Greengrass 코어 소프트웨어 v1.x는 추후 공지가 있을 때까지 수명 연장 단계로 유지됩니다.

AWS IoT Greengrass 코어 소프트웨어 v2.x는 현재 릴리스 단계에 있으며 추후 공지가 있을 때까지 릴리스 단계에 남아 있습니다. AWS IoT Greengrass은(는) AWS IoT Greengrass 코어 소프트웨어 v2.x에 새로운 기능과 향상된 기능을 계속 추가하고 있습니다. 예를 들어, AWS IoT Greengrass은(는) AWS IoT Greengrass 코어 소프트웨어 v2.5.0에서 Windows 지원을 릴리스했습니다. AWS IoT Greengrass은(는) 릴리스일로부터 최소 1년 동안 AWS IoT Greengrass 코어 v2.x의 모든 마이너 버전에 대한 보안 패치와 버그 수정을 릴리스합니다. 자세한 내용은 What's New in AWS IoT Greengrass V2?를 참조하세요.

유지 관리 단계 일정

2023년 6월 30일에 AWS IoT Greengrass 코어 소프트웨어 v1.11.x의 유지 관리 단계가 종료되었습니다. 2022년 3월 31일에 AWS IoT Greengrass 코어 소프트웨어 v1.10.x의 유지 관리 단계가 종료되었습니다. 특정 AWS IoT Greengrass 코어 소프트웨어 v1.x 아티팩트 및 기능에 대한 유지 관리 단계는 이 날짜 이전에 종료됩니다. 자세한 내용은 유지보수 일정 종료 섹션을 참조하세요.

AWS Support 플랜이 있는 경우, AWS IoT Greengrass 코어 소프트웨어 v1.x의 유지 관리 단계는 AWS Support 플랜에 영향을 주지 않습니다. 유지 관리 단계가 끝난 후에도 AWS Support 티켓을 계속 열 수 있습니다. 질문이나 우려 사항이 있는 경우, AWS Support 담당자에게 문의하거나 AWS IoT Greengrass 태그를 사용하여 AWS re:Post 질문하세요.

사용 중단 일정

현재로서는 AWS IoT Greengrass 코어 소프트웨어 v1.x의 지원을 중단할 계획이 없습니다. AWS IoT Greengrass V1 엔드포인트 및 API 작업은 추후 공지가 있을 때까지 계속 사용할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어 v1.11.6은 2023년 6월 30일에 수명 연장 단계에 들어갔습니다. 이 단계에서 AWS IoT Greengrass 코어 소프트웨어 v1.x를 실행하는 디바이스는 추후 공지가 있을 때까지 AWS IoT Greengrass V1 서비스에 계속 연결하여 작동할 수 있습니다.

AWS IoT Greengrass V1이(가) 향후 지원을 중단하는 경우, AWS IoT Greengrass은(는) 이 문제가 발생하기 12개월 전에 사전 통지할 예정입니다. 이렇게 하면 AWS IoT Greengrass V2와(과) AWS IoT Greengrass 코어 소프트웨어 v2.x을 사용할 애플리케이션의 업데이트를 계획하는 데 도움이 됩니다. 애플리케이션을 V2로 업데이트하는 방법에 대한 자세한 내용은 AWS IoT Greengrass V1에서 V2로 이동을 참조합니다.

Greengrass 코어 디바이스의 AWS Lambda 기능에 대한 지원 정책

AWS IoT Greengrass은(는) IoT 디바이스에서 AWS Lambda 기능을 실행할 수 있게 합니다. AWS Lambda은(는) AWS IoT Greengrass에서 Lambda 런타임에 대한 지원을 결정하는 지원 정책 및 일정을 제공합니다. Lambda 런타임이 지원 종료 단계에 도달하면 AWS IoT Greengrass은(는) 해당 런타임에 대한 지원도 종료합니다. 자세한 내용을 알아보려면 런타임 지원 정책에 대한 AWS Lambda 개발자 가이드를 참조하세요.

Lambda 런타임이 지원 종료에 도달하면 해당 런타임을 사용하는 Lambda 함수를 생성하거나 업데이트할 수 없습니다. 하지만 이러한 Lambda 함수를 Greengrass 코어 디바이스에 계속 배포하고 배포된 Lambda 함수를 간접적으로 호출할 수 있습니다. 이 정책은 AWS IoT Greengrass V2에도 적용됩니다.

AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터에 대한 지원 정책

AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터(IDT)를 사용하면 AWS Partner 디바이스 카탈로그에 포함시킬 AWS IoT Greengrass 디바이스를 검증하고 자격을 부여할 수 있습니다. 2022년 4월 4일부터 AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터(IDT)는 더 이상 서명된 자격 보고서를 생성하지 않습니다. 더 이상 AWS 디바이스 검증 프로그램을 통해 AWS Partner 디바이스 카탈로그에 나열할 새 AWS IoT Greengrass V1 디바이스를 검증할 수 없습니다. Greengrass V1 디바이스를 검증할 수는 없지만 계속해서 AWS IoT Greengrass V1용 IDT를 사용하여 Greengrass V1 디바이스를 테스트할 수 있습니다. AWS Partner 디바이스 카탈로그에서 Greengrass 디바이스를 검증하고 등재하려면 AWS IoT Greengrass V2용 IDT를 사용하는 것이 좋습니다. 자세한 내용은 AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터에 대한 지원 정책 섹션을 참조하세요.

유지보수 일정 종료

다음 표에는 AWS IoT Greengrass 코어 v1.x 아티팩트 및 기능의 유지 관리 종료 날짜가 나와 있습니다. 유지 관리 일정 또는 정책에 대한 질문이 있는 경우, AWS 지원에 문의하세요.

Artifact 또는 기능	유지 관리 종료일
Greengrass APT 리포지토리 설치	2022년 2월 11일
ML 이미지 분류 커넥터	2022년 3월 31일
ML 객체 감지 커넥터	2022년 3월 31일
ML 피드백 커넥터	2022년 3월 31일
AWS IoT Analytics 커넥터	2022년 3월 31일
Twilio 알림 커넥터	2022년 3월 31일
Splunk 통합 커넥터	2022년 3월 31일
Serial Stream 커넥터	2022년 3월 31일
ServiceNow MetricBase 통합 커넥터	2022년 3월 31일
Raspberry Pi GPIO 커넥터	2022년 3월 31일

Artifact 또는 기능	유지 관리 종료일
AWS IoT Greengrass 코어 소프트웨어 v1.10.x	2022년 3월 31일
AWS IoT Greengrass 코어 소프트웨어 v1.x Docker 이미지	2022년 6월 30일
AWS IoT Greengrass 코어 소프트웨어 v1.11.x	2023년 6월 30일
AWS IoT Greengrass 코어 소프트웨어 v1.11.x Snap	2023년 12월 31일

AWS IoT Greengrass 코어 소프트웨어 v1.x Docker 이미지에 대한 유지 관리 종료

2022년 6월 30일에 AWS IoT Greengrass은(는) Amazon Elastic Container Registry(Amazon ECR) 및 Docker Hub에 게시된 AWS IoT Greengrass 코어 소프트웨어 v1.x Docker 이미지에 대한 유지 관리를 종료하였습니다. 유지 관리 종료 후 1년이 되는 2023년 6월 30일까지 Amazon ECR 및 Docker Hub에서 이 Docker 이미지를 계속 다운로드할 수 있습니다. 그러나 AWS IoT Greengrass Core 소프트웨어 v1.x Docker 이미지는 2022년 6월 30일에 유지 관리가 종료된 이후 더 이상 보안 패치나 버그 수정을 받지 않습니다. 이러한 Docker 이미지를 사용하는 프로덕션 워크로드를 실행하는 경우, AWS IoT Greengrass이(가) 제공하는 Dockerfile을 사용하여 자체 Docker 이미지를 구축하는 것이 좋습니다. 자세한 내용은 AWS IoT Greengrass Docker 소프트웨어 섹션을 참조하세요.

AWS IoT Greengrass 코어 소프트웨어 v1.x APT 리포지토리 유지 관리 종료

2022년 2월 11일에 AWS IoT Greengrass이(가) APT 리포지토리에서 AWS IoT Greengrass 코어 소프트웨어 v1.x를 설치하는 옵션에 대한 유지 관리를 종료했습니다. 이 날짜에 APT 리포지토리가 제거되었으므로 더 이상 APT 리포지토리를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 업데이트하거나 새 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 없습니다. 리포지토리를 추가한 디바이스에서는 소스 목록에서 AWS IoT Greengrass 리포지토리를 제거해야 합니다. tar 파일을 사용하여 AWS IoT Greengrass 코어 소프트웨어 v1.x를 업데이트하는 것이 좋습니다.

AWS IoT Greengrass 코어 소프트웨어 v1.11.x 스냅에 대한 유지 관리 종료

2023년 12월 31일에 AWS IoT Greengrass은(는) snapcraft.io에 게시된 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.x Snap에 대한 유지 관리를 종료합니다. 현재 Snap을 실행하는 기기는 추후 공지

가 있을 때까지 계속 작동합니다. 그러나 유지 관리 종료 후에는 AWS IoT Greengrass 코어 Snap에 더 이상 보안 패치나 버그 수정이 제공되지 않습니다.

시작하기 AWS IoT Greengrass

이 시작하기 자습서에는 AWS IoT Greengrass 기본 사항을 보여주고 사용을 시작하는 데 도움이 되도록 설계된 여러 모듈이 포함되어 AWS IoT Greengrass있습니다. 이 튜토리얼에서는 다음과 같은 기본 개념을 다룹니다.

- AWS IoT Greengrass 코어 및 그룹 구성.
- 엣지에서 AWS Lambda 함수를 실행하기 위한 배포 프로세스.
- 클라이언트 AWS IoT 디바이스라고 하는 디바이스를 AWS IoT Greengrass 코어에 연결합니다.
- 로컬 Lambda 함수, 클라이언트 디바이스 및 간의 MQTT 통신을 허용하는 구독 생성 AWS IoT

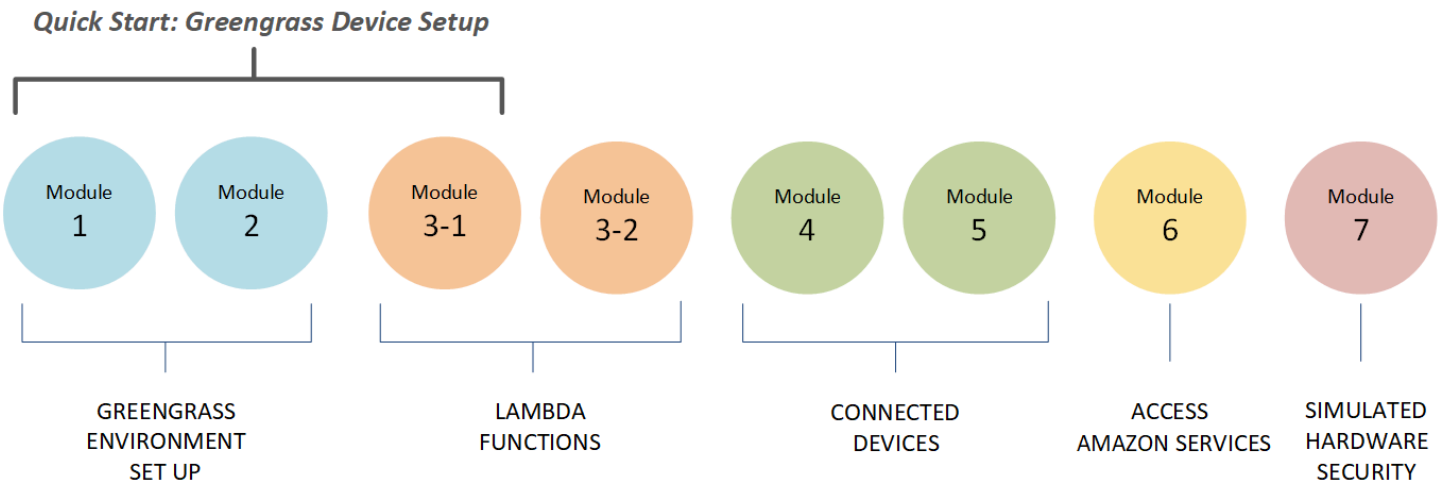
시작 방법을 선택하세요. AWS IoT Greengrass

이 튜토리얼을 사용하여 코어 장치를 설정하는 방법을 선택할 수 있습니다.

- 코어 디바이스에서 Greengrass 디바이스 설정을 실행하면 몇 분 안에 AWS IoT Greengrass 종속성 설치부터 Hello World Lambda 함수 테스트까지 완료할 수 있습니다. 이 스크립트는 모듈 1에서 모듈 3-1까지 단계를 재현합니다.

  – 또는 -

- 모듈 1에서 모듈 3-1까지 단계를 수행하여 Greengrass 요구 사항과 프로세스를 보다 면밀히 점검합니다. 그런 다음 코어 장치를 설정하고, Hello World Lambda 함수를 포함하는 Greengrass 그룹을 생성 및 구성하고, Greengrass 그룹을 배포합니다. 일반적으로 이 작업을 완료하는 데 1~2시간 정도 걸립니다.



빠른 시작

Greengrass 장치 설정은 코어 장치와 Greengrass 리소스를 구성합니다. 스크립트:

- 종속성을 설치합니다. AWS IoT Greengrass
- 루트 CA 인증서와 코어 장치 인증서 및 키를 다운로드합니다.
- 장치에 AWS IoT Greengrass Core 소프트웨어를 다운로드, 설치 및 구성합니다.
- 코어 장치에서 Greengrass 대몬(daemon) 프로세스를 시작합니다.
- 필요한 경우 Greengrass 서비스 역할을 생성하거나 업데이트합니다.
- Greengrass 그룹 및 Greengrass 코어를 생성합니다.
- (선택 사항) Hello World Lambda 함수, 구독 및 로컬 로깅 구성을 생성합니다.
- (선택 사항) Greengrass 그룹을 배포합니다.

모듈 1 및 2

모듈 1 및 모듈 2에서는 환경을 설정하는 방법을 설명합니다. 또는 Greengrass 장치 설정을 사용하여 이러한 모듈을 자동으로 실행합니다.

- Greengrass에 대한 코어 장치를 구성합니다.
- 종속성 확인 프로그램을 실행합니다.
- Greengrass 그룹 및 Greengrass 코어를 생성합니다.
- tar.gz 파일에서 최신 AWS IoT Greengrass Core 소프트웨어를 다운로드하고 설치합니다.
- 코어에서 Greengrass 대몬(daemon) 프로세스를 시작합니다.

> **Note**
>
> AWS IoT Greengrass 지원되는 데비안 플랫폼에서의 `apt` 설치를 포함하여 AWS IoT Greengrass Core 소프트웨어를 설치하기 위한 다른 옵션도 제공합니다. 자세한 정보는 the section called “AWS IoT Greengrass 코어 소프트웨어 설치”을 참조하세요.

모듈 3-1 및 3-2

모듈 3-1 및 모듈 3-2에서는 로컬 Lambda 함수를 사용하는 방법을 설명합니다. 또는 Greengrass 장치 설정을 사용하여 모듈 3-1을 자동으로 실행합니다.

- 에서 헬로월드 람다 함수를 생성합니다. AWS Lambda
- Greengrass 그룹에 Lambda 함수를 추가합니다.
- Lambda 함수와 간에 MQTT 통신을 허용하는 구독을 생성합니다. AWS IoT
- Greengrass 시스템 구성 요소 및 Lambda 함수에 대한 로컬 로깅을 구성합니다.
- Lambda 함수와 구독이 포함된 Greengrass 그룹을 배포합니다.
- 로컬 Lambda 함수에서 로 메시지를 보냅니다. AWS IoT
- 에서 로컬 Lambda 함수를 호출합니다. AWS IoT
- 온디맨드 및 수명이 긴 함수를 테스트합니다.

모듈 4 및 5

모듈 4에서는 클라이언트 장치를 코어에 연결하여 서로 통신하는 방법을 보여줍니다.

모듈 5에서는 클라이언트 장치에서 섀도우를 사용하여 상태를 제어하는 방법을 보여줍니다.

- AWS IoT 디바이스 등록 및 프로비저닝 (명령줄 터미널로 표시).
- AWS IoT Device SDK Python용 를 설치합니다. 이는 클라이언트 장치에서 Greengrass 코어를 검색하는 데 사용됩니다.
- Greengrass 그룹에 클라이언트 장치를 추가합니다.
- MQTT 통신을 허용하는 구독을 생성합니다.
- 클라이언트 장치가 포함된 Greengrass 그룹을 배포합니다.
- device-to-device 통신 테스트.
- 섀도우 상태 업데이트를 테스트합니다.

모듈 6

모듈 6에서는 Lambda 함수가 AWS 클라우드에 액세스하는 방법을 보여줍니다.

- Amazon DynamoDB 리소스에 대한 액세스를 허용하는 Greengrass 그룹 역할을 생성합니다.
- Greengrass 그룹에 Lambda 함수를 추가합니다. 이 함수는 Python용 AWS SDK를 사용하여 DynamoDB와 상호 작용합니다.
- MQTT 통신을 허용하는 구독을 생성합니다.
- DynamoDB와 상호 작용을 테스트합니다.

모듈 7

모듈 7에서는 Greengrass 코어와 함께 사용하도록 시뮬레이션된 하드웨어 보안 모듈(HSM)을 구성하는 방법을 보여줍니다.

> **Important**
>
> 이 고급 모듈은 실험과 초기 테스트를 위해서만 제공되며, 어떠한 종류의 실제 사용 용도로 제공되지 않습니다.

- 소프트웨어 기반 HSM 및 프라이빗 키를 설치하고 구성합니다.
- 하드웨어 보안을 사용하도록 Greengrass 코어를 구성합니다.
- 하드웨어 보안 구성을 테스트합니다.

요구 사항

이 튜토리얼을 완료하려면 다음이 필요합니다.

- Mac, Windows PC 또는 UNIX와 같은 시스템
- An. AWS 계정계정이 없는 경우 the section called “생성하기 AWS 계정” 단원을 참조하십시오.
- 지원하는 AWS 지역의 사용 AWS IoT Greengrass. 지원되는 지역 목록은 의 AWS 엔드포인트 및 할당량을 참조하십시오. AWS IoT GreengrassAWS 일반 참조

> **Note**
>
> 참고 사항을 AWS 리전 기록하고 이 자습서 전체에서 일관되게 사용되는지 확인하세요. 자습서를 AWS 리전 진행하는 동안 설정을 변경하면 단계를 완료하는 데 문제가 발생할 수 있습니다.

- 8GB 마이크로SD 카드가 장착된 Raspberry Pi 4 모델 B 또는 Raspberry Pi 3 모델 B/B+ 또는 Amazon EC2 인스턴스. AWS IoT Greengrass 는 물리적 하드웨어에 사용하는 것이 이상적이기 때문에 Raspberry Pi를 사용하는 것이 좋습니다.

  > **Note**
  >
  > Raspberry Pi의 모델을 확인하려면 다음 명령을 실행하십시오.
  >
  > ```
  > cat /proc/cpuinfo
  > ```
  >
  > 목록 하단 근처에서 `Revision` 속성의 값을 메모한 다음 Which Pi have I got? 표를 참조하십시오. 예를 들어 `Revision`의 값이 `a02082`이면 표는 Pi가 3 모델 B임을 보여 줍니다. Raspberry Pi의 아키텍처를 확인하려면 다음 명령을 실행하십시오.
  >
  > ```
  > uname -m
  > ```
  >
  > 이 튜토리얼에서 결과는 `armv71`보다 크거나 같아야 합니다.

- Python에 대한 기본적인 지식

이 튜토리얼은 Raspberry AWS IoT Greengrass Pi에서 실행하기 위한 것이지만 다른 AWS IoT Greengrass 플랫폼도 지원합니다. 자세한 정보는 the section called “지원되는 플랫폼 및 요구 사항”을 참조하세요.

생성하기 AWS 계정

계정이 없는 AWS 계정경우 다음 단계에 따라 계정을 만들고 활성화하세요 AWS 계정.

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 여세요.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중에는 전화를 받고 키패드로 인증 코드를 입력하는 과정이 있습니다.

에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스 액세스 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 가서 내 계정(My Account)을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요. AWS 계정 루트 사용자

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면AWS 로그인 사용 설명서의 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 참조하십시오.

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 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

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 포털 로그인을 참조하십시오.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.

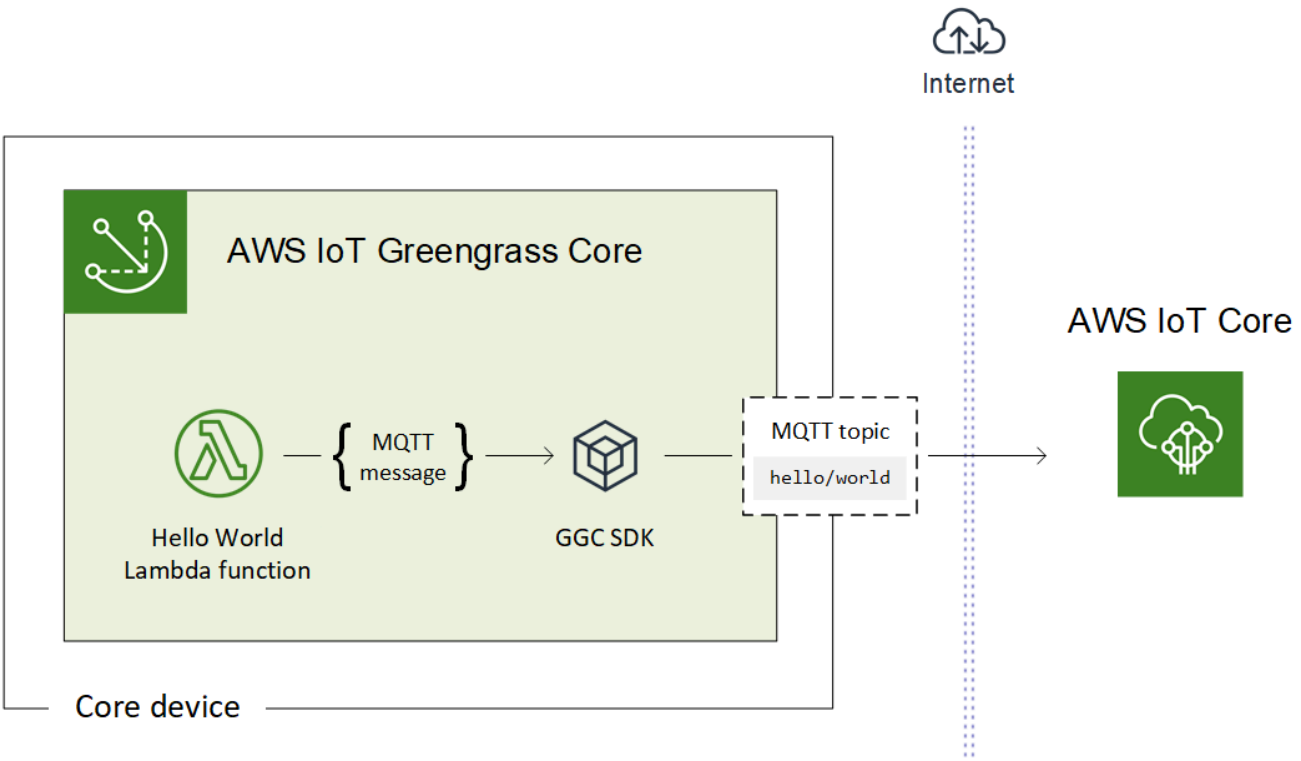
   지침은AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

> **Important**
>
> 이 튜토리얼에서는 IAM 사용자 계정에 관리자 액세스 권한이 있다고 가정합니다.

빠른 시작: Greengrass 디바이스 설정

Greengrass 디바이스 설정은 코어 디바이스를 몇 분 이내에 설정하는 스크립트이므로, AWS IoT Greengrass를 빠르게 사용할 수 있습니다. 이 스크립트를 사용하여 다음을 수행할 수 있습니다.

1. 디바이스를 구성하고 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.
2. 클라우드 기반 리소스를 구성합니다.
3. 선택적으로 MQTT 메시지를 AWS IoT Greengrass 코어에서 AWS IoT에 전송하는 Hello World Lambda 함수를 사용하여 Greengrass 그룹을 배포합니다. 이 단계에서는 다음 다이어그램에 표시된 Greengrass 환경을 설정합니다.



요구 사항

Greengrass 디바이스 설정에는 다음과 같은 요구 사항이 있습니다.

- 코어 디바이스에서 지원되는 플랫폼을 사용해야 합니다. 디바이스에 적절한 패키지 관리자(`apt`, `yum` 또는 `opkg`)가 설치되어 있어야 합니다.
- 스크립트를 실행하는 Linux 사용자가 `sudo`로 실행할 권한이 있어야 합니다.
- AWS 계정 자격 증명을 제공해야 합니다. 자세한 내용은 the section called “AWS 계정 자격 증명 제공” 섹션을 참조하세요.

> **Note**
>
> Greengrass 디바이스 설정은 디바이스에 최신 버전의 AWS IoT Greengrass Core 소프트웨어를 설치합니다. 이 AWS IoT Greengrass 코어 소프트웨어를 설치하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

Greengrass 디바이스 설정 실행

몇 단계만으로 Greengrass 디바이스 설정을 실행할 수 있습니다. AWS 계정 자격 증명을 제공하면 스크립트에서 Greengrass 코어 디바이스를 프로비저닝하고 몇 분 안에 Greengrass 그룹을 배포합니다. 대상 디바이스의 터미널 창에서 다음 명령을 실행합니다.

> **Note**
>
> 다음 단계에서는 대화형 모드에서 스크립트를 실행하는 방법을 보여줍니다. 이 모드에서는 각 입력 값을 입력하거나 수락하라는 메시지가 표시됩니다. 스크립트를 자동으로 실행하는 방법에 대한 자세한 내용은 the section called “자동 모드로 Greengrass 디바이스 설정 실행” 섹션을 참조하십시오.

1. 자격 증명을 제공합니다. 이 절차에서는 임시 보안 자격 증명을 환경 변수로 제공한다고 가정합니다.

   ```
   export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   > **Note**
   >
   > Raspbian 또는 OpenWRT 플랫폼에서 Greengrass 디바이스 설정을 실행하는 경우 다음 명령을 복사합니다. 디바이스를 재부팅한 후 다시 제공해야 합니다.

2. 스크립트를 다운로드하여 시작합니다. `wget` 또는 `curl` 키를 사용하여 스크립트를 다운로드할 수 있습니다.

   `wget`:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass-interactive
   ```

curl:

```
curl https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/
gg-device-setup-latest.sh > gg-device-setup-latest.sh && chmod +x ./gg-device-
setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-
interactive
```

3. 입력 값에 대한 명령 프롬프트를 계속 진행합니다. Enter 키를 눌러 기본값을 사용하거나 사용자 지정 값을 입력한 다음 Enter 키를 누를 수 있습니다.

   이 스크립트는 다음과 유사한 상태 메시지를 터미널에 기록합니다.

```
############### Greengrass Device Setup v1.0.0 ###############
[GreengrassDeviceSetup] The Greengrass Device Setup bootstrap log is available at: /tmp/greengrass-device-setup-bootstrap-1575933831.log
[GreengrassDeviceSetup] Using package management tool: yum...
[GreengrassDeviceSetup] Using runtime: python3.7...
[GreengrassDeviceSetup] Installing a dedicated pip for Greengrass Device Setup...
[GreengrassDeviceSetup] Validating and installing required dependencies...
[GreengrassDeviceSetup] The Greengrass Device Setup configuration is complete. Starting the Greengrass environment setup...
[GreengrassDeviceSetup] Forwarding command-line parameters: bootstrap-greengrass-interactive

[GreengrassDeviceSetup] Validating the device environment...
[GreengrassDeviceSetup] Validation of the device environment is complete.

[GreengrassDeviceSetup] Running the Greengrass environment setup...
[GreengrassDeviceSetup] The Greengrass environment setup is complete.

[GreengrassDeviceSetup] Configuring cloud-based Greengrass group management...
[GreengrassDeviceSetup] The Greengrass group configuration is complete.

[GreengrassDeviceSetup] Preparing the Greengrass core software...
[GreengrassDeviceSetup] The Greengrass core software is running.

[GreengrassDeviceSetup] Configuring the group deployment...
[GreengrassDeviceSetup] The group deployment is complete.
```

4. 코어 디바이스에서 Raspbian 또는 OpenWRT를 실행 중인 경우 메시지가 표시되면 디바이스를 재부팅하고 자격 증명을 제공한 다음 스크립트를 다시 시작합니다.

   a. 디바이스를 재부팅하라는 메시지가 표시되면 다음 명령 중 하나를 실행합니다.

      Raspbian 플랫폼의 경우:

```
sudo reboot
```

      OpenWRT 플랫폼의 경우:

```
reboot
```

   b. 디바이스가 재부팅된 후 터미널을 열고 자격 증명을 환경 변수로 제공합니다.

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

c. 스크립트를 다시 시작합니다.

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

d. 이전 세션의 입력 값을 사용할지 새 설치를 시작할지 여부를 묻는 메시지가 표시되면 yes를 입력하여 입력 값을 다시 사용합니다.

> **Note**
>
> 재부팅이 필요한 플랫폼에서는 이전 세션의 입력 값(자격 증명 제외)이 임시로 `GreengrassDeviceSetup.config.info` 파일에 저장됩니다.

설정이 완료되면 터미널에 다음과 유사한 성공 상태 메시지가 표시됩니다.

```
================================================================================

Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.


Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

================================================================================
```

5. 제공한 입력 값을 사용하여 스크립트가 구성하는 새 Greengrass 그룹을 검토하십시오.

a. 컴퓨터에서 AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.

> **Note**
>
> 콘솔에서 선택한 AWS 리전이 Greengrass 환경을 구성하는 데 사용한 것과 동일한지 확인합니다. 기본적으로 리전은 미국 서부(오레곤)입니다.

b. 탐색 창에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택하여 새로 만든 그룹을 찾습니다.

6. Hello World Lambda 함수를 포함한 경우 Greengrass 디바이스 설정은 Greengrass 그룹을 코어 디바이스에 배포합니다. Lambda 함수를 테스트하거나 그룹에서 Lambda 함수를 제거하는 방법을 알아보려면 시작하기 튜토리얼의 모듈 3-1에서 the section called “Lambda 함수가 코어 디바이스에서 실행 중인지 확인” 섹션을 계속 진행합니다.

> **Note**
>
> 콘솔에서 선택한 AWS 리전이 Greengrass 환경을 구성하는 데 사용한 것과 동일한지 확인합니다. 기본적으로 리전은 미국 서부(오레곤)입니다.

Hello World Lambda 함수를 포함하지 않은 경우 자신의 Lambda 함수를 생성하거나 다른 Greengrass 특성을 사용해 볼 수 있습니다. 예를 들어 Docker 애플리케이션 배포 커넥터를 그룹에 추가한 후 Docker 컨테이너를 코어 디바이스에 배포하는 데 사용할 수 있습니다.

문제 해결

다음 정보를 사용하면 AWS IoT Greengrass 디바이스 설정의 문제 해결에 도움이 됩니다.

오류: Python(python3.7)을 찾을 수 없습니다. 설치 시도 중...

해결 방법: Amazon EC2 인스턴스로 작업할 때 이 오류가 표시될 수 있습니다. 이 오류는 Python이 `/usr/bin/python3.7` 폴더에 설치되어 있지 않을 때 발생합니다. 이 오류를 해결하려면 Python을 설치한 후 올바른 디렉토리로 이동하십시오.

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

추가 문제 해결

AWS IoT Greengrass 디바이스 설정에 추가적인 문제를 해결하려면 로그 파일에서 디버그 정보를 확인할 수 있습니다.

- Greengrass 디바이스 설정 구성에 문제가 있는 경우 `/tmp/greengrass-device-setup-bootstrap-`*`epoch-timestamp`*`.log` 파일을 확인합니다.
- Greengrass 그룹 또는 코어 환경 설정에 문제가 있는 경우 `gg-device-setup-latest.sh`와 동일한 디렉터리 또는 지정한 위치에서 `GreengrassDeviceSetup-`*`date`*`-`*`time`*`.log` 파일을 확인합니다.

문제 해결 도움말은 문제 해결 섹션을 참조하거나 AWS re:Post의 AWS IoT Greengrass 태그를 확인하십시오.

Greengrass 디바이스 설정 구성 옵션

AWS 리소스에 액세스하고 Greengrass 환경을 설정하도록 Greengrass 디바이스 설정을 구성합니다.

AWS 계정 자격 증명 제공

Greengrass 디바이스 설정에서는 AWS 계정 자격 증명을 사용하여 AWS 리소스에 액세스합니다. IAM 사용자에 대한 장기 자격 증명 또는 IAM 역할의 임시 보안 자격 증명을 지원합니다.

먼저 자격 증명을 가져옵니다.

- 장기 자격 증명을 사용하려면 IAM 사용자의 액세스 키 ID와 비밀 액세스 키를 제공합니다. 장기 자격 증명을 위한 액세스 키 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자를 위한 액세스 키 관리를 참조하십시오.

- 임시 보안 자격 증명(권장)을 사용하려면 위임된 IAM 역할의 액세스 키 ID, 비밀 액세스 키 및 세션 토큰을 제공합니다. AWS STS `assume-role` 명령에서 임시 보안 자격 증명을 추출하는 것에 대한 자세한 내용은 IAM 사용 설명서의 AWS CLI를 이용해 임시 보안 자격 증명 사용을 참조하세요.

> **Note**
>
> 이 튜토리얼은 IAM 사용자 또는 IAM 역할에 관리자 액세스 권한이 있다고 가정합니다.

이후, 다음 두 가지 방법 중 하나로 Greengrass 디바이스 설정에 자격 증명을 제공합니다.

- 환경 변수로. the section called “Greengrass 디바이스 설정 실행”의 1단계에 표시된 대로 스크립트를 시작하기 전에 `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` 및 `AWS_SESSION_TOKEN`(필요한 경우) 환경 변수를 설정합니다.

- 입력 값으로. 스크립트를 시작한 후 터미널에 액세스 키 ID, 비밀 액세스 키 및 세션 토큰(필요한 경우) 값을 직접 입력합니다.

Greengrass 디바이스 설정에서는 자격 증명을 저장하지 않습니다.

입력 값 제공

대화형 모드에서 Greengrass 디바이스 설정은 입력 값을 입력하라는 메시지를 표시합니다. Enter 키를 눌러 기본값을 사용하거나 사용자 지정 값을 입력한 다음 Enter 키를 누를 수 있습니다. 자동 모드에서는 스크립트를 시작한 후 입력 값을 제공합니다.

입력 값입니다.

AWS 액세스 키 ID

장기 또는 임시 보안 자격 증명의 액세스 키 ID입니다. 자격 증명을 환경 변수로 제공하지 않는 경우에만 이 옵션을 입력 값으로 지정합니다. 자세한 내용은 the section called “AWS 계정 자격 증명 제공” 섹션을 참조하세요.

자동 모드의 옵션 이름: `--aws-access-key-id`

AWS 비밀 액세스 키

장기 또는 임시 보안 자격 증명의 비밀 액세스 키입니다. 자격 증명을 환경 변수로 제공하지 않는 경우에만 이 옵션을 입력 값으로 지정합니다. 자세한 내용은 the section called “AWS 계정 자격 증명 제공” 섹션을 참조하세요.

자동 모드의 옵션 이름: `--aws-secret-access-key`

AWS 세션 토큰

임시 보안 자격 증명의 세션 토큰입니다. 자격 증명을 환경 변수로 제공하지 않는 경우에만 이 옵션을 입력 값으로 지정합니다. 자세한 내용은 the section called “AWS 계정 자격 증명 제공” 섹션을 참조하세요.

자동 모드의 옵션 이름: `--aws-session-token`

AWS 리전

Greengrass 그룹을 생성하려는 AWS 리전입니다. 지원되는 AWS 리전 목록은 Amazon Web Services 일반 참조의 AWS IoT Greengrass를 참조하십시오.

기본값: `us-west-2`

자동 모드의 옵션 이름: `--region`

[Group name]

Greengrass 그룹의 이름입니다.

기본값: `GreengrassDeviceSetup_Group_`*`guid`*

자동 모드의 옵션 이름: `--group-name`

코어 이름

Greengrass 코어의 이름입니다. 코어는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 AWS IoT 디바이스(사물)입니다. 코어는 AWS IoT 레지스트리와 Greengrass 그룹에 추가됩니다. 이름을 제공하는 경우 이름은 AWS 계정 및 AWS 리전에서 고유해야 합니다.

기본값: `GreengrassDeviceSetup_Core_`*`guid`*

자동 모드의 옵션 이름: `--core-name`

AWS IoT Greengrass 코어 소프트웨어 설치 경로

AWS IoT Greengrass 코어 소프트웨어를 설치할 디바이스 파일 시스템의 위치입니다.

기본값: `/`

자동 모드의 옵션 이름: `--ggc-root-path`

Hello World 람다 함수

Greengrass 그룹에 Hello World Lambda 함수를 포함할지 여부를 나타냅니다. 이 함수는 5초마다 `hello/world` 주제에 MQTT 메시지를 게시합니다.

이 스크립트는 AWS Lambda에서 이 사용자 정의 Lambda 함수를 생성하여 Greengrass 그룹에 추가합니다. 또한 이 스크립트는 함수가 MQTT 메시지를 AWS IoT에 전송하도록 허용하는 그룹에서 구독을 생성합니다.

> **Note**
>
> 이는 Python 3.7 Lambda 함수입니다. Python 3.7이 디바이스에 설치되어 있지 않고 스크립트에서 설치할 수 없는 경우 스크립트는 터미널에 오류 메시지를 인쇄합니다. 그룹에 Lambda 함수를 포함하려면 Python 3.7을 수동으로 설치하고 스크립트를 다시 시작해야 합니다. Lambda 함수 없이 Greengrass 그룹을 생성하려면 스크립트를 다시 시작하고 함수를 포함할지 묻는 메시지가 표시되면 no를 입력합니다.

기본값: no

자동 모드의 옵션 이름: `--hello-world-lambda` - 이 옵션은 값을 갖고 있지 않습니다. 함수를 생성하려면 명령에 포함합니다.

배포 제한 시간

Greengrass 디바이스 설정이 Greengrass 그룹 배포의 상태 확인을 중지할 때까지 경과되는 시간(초)입니다. 그룹에 Hello World 람다 함수를 포함하는 경우에만 사용됩니다. 그렇지 않은 경우 그룹이 배포되지 않습니다.

배포 시간은 네트워크 속도에 따라 다릅니다. 네트워크 속도가 느린 경우 이 값을 늘릴 수 있습니다.

기본값: `180`

자동 모드의 옵션 이름: `--deployment-timeout`

로그 경로

Greengrass 그룹 및 코어 설정 작업에 대한 정보가 포함된 로그 파일의 위치입니다. 이 로그를 사용하여 Greengrass 그룹 및 코어 설정과 관련된 배포 및 기타 문제를 해결할 수 있습니다.

기본값: `./`

자동 모드의 옵션 이름: `--log-path`

Verbosity

스크립트가 실행되는 동안 터미널에서 자세한 로그 정보를 인쇄할지 여부를 나타냅니다. 이 정보를 사용하여 디바이스 설정 문제를 해결할 수 있습니다.

기본값: no

자동 모드의 옵션 이름: `--verbose` - 이 옵션은 값을 갖고 있지 않습니다. 자세한 로그 정보를 인쇄하려면 명령에 포함합니다.

자동 모드로 Greengrass 디바이스 설정 실행

스크립트가 값을 입력하라는 메시지를 표시하지 않도록 자동 모드에서 Greengrass 디바이스 설정을 실행할 수 있습니다. 자동 모드로 실행하려면 스크립트를 시작한 후 `bootstrap-greengrass` 모드 및 입력 값을 지정합니다. 기본값을 사용하려면 입력 값을 생략할 수 있습니다.

절차는 스크립트를 시작하기 전에 AWS 계정 자격 증명을 환경 변수로 제공하는지 아니면 스크립트를 시작한 후 입력 값으로 제공하는지에 따라 다릅니다.

자격 증명을 환경 변수로 제공

1. 자격 증명을 환경 변수로 제공합니다. 다음 예제에서는 세션 토큰을 포함하는 임시 자격 증명을 내보냅니다.

   ```
   export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
   export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   > **Note**
   >
   > Raspbian 또는 OpenWRT 플랫폼에서 Greengrass 디바이스 설정을 실행하는 경우 다음 명령을 복사합니다. 디바이스를 재부팅한 후 다시 제공해야 합니다.

2. 스크립트를 다운로드하여 시작합니다. 필요에 따라 입력 값을 제공합니다. 예:

   - 모든 기본값을 사용하려면:

     ```
     wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
     greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
     gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
     greengrass
     ```

   - 사용자 지정 값을 지정하려면:

     ```
     wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
     greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
     gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
     greengrass
     --region us-east-1
     ```

```
--group-name Custom_Group_Name
--core-name Custom_Core_Name
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

> **Note**
>
> curl을 사용하여 스크립트를 다운로드하려면 명령에서 `wget -q -O`를 `curl`로 바꿉니다.

3. 코어 디바이스에서 Raspbian 또는 OpenWRT를 실행 중인 경우 메시지가 표시되면 디바이스를 재부팅하고 자격 증명을 제공한 다음 스크립트를 다시 시작합니다.

   a. 디바이스를 재부팅하라는 메시지가 표시되면 다음 명령 중 하나를 실행합니다.

      Raspbian 플랫폼의 경우:

      ```
      sudo reboot
      ```

      OpenWRT 플랫폼의 경우:

      ```
      reboot
      ```

   b. 디바이스가 재부팅된 후 터미널을 열고 자격 증명을 환경 변수로 제공합니다.

      ```
      export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
      export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

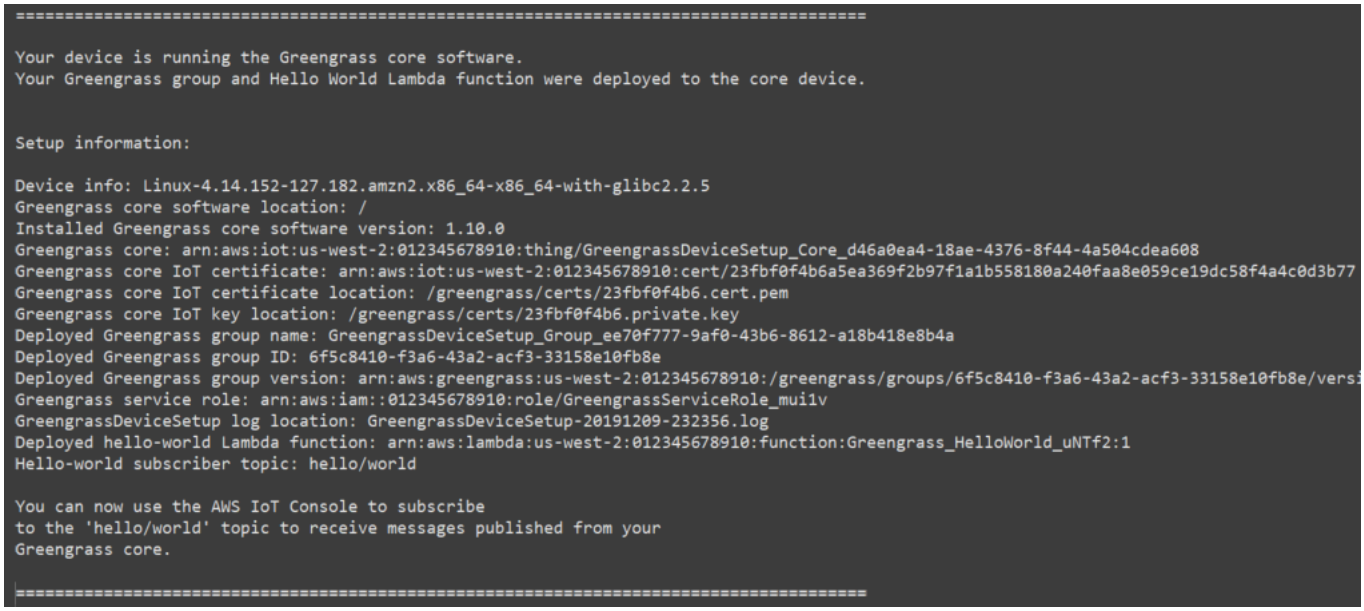
   c. 스크립트를 다시 시작합니다.

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      ```

   d. 이전 세션의 입력 값을 사용할지 새 설치를 시작할지 여부를 묻는 메시지가 표시되면 `yes`를 입력하여 입력 값을 다시 사용합니다.

> **Note**
>
> 재부팅이 필요한 플랫폼에서는 이전 세션의 입력 값(자격 증명 제외)이 임시로 `GreengrassDeviceSetup.config.info` 파일에 저장됩니다.

설정이 완료되면 터미널에 다음과 유사한 성공 상태 메시지가 표시됩니다.



4. Hello World Lambda 함수를 포함한 경우 Greengrass 디바이스 설정은 Greengrass 그룹을 코어 디바이스에 배포합니다. Lambda 함수를 테스트하거나 그룹에서 Lambda 함수를 제거하는 방법을 알아보려면 시작하기 튜토리얼의 모듈 3-1에서 the section called “Lambda 함수가 코어 디바이스에서 실행 중인지 확인” 섹션을 계속 진행합니다.

> **Note**
>
> 콘솔에서 선택한 AWS 리전이 Greengrass 환경을 구성하는 데 사용한 것과 동일한지 확인합니다. 기본적으로 리전은 미국 서부(오레곤)입니다.

Hello World Lambda 함수를 포함하지 않은 경우 자신의 Lambda 함수를 생성하거나 다른 Greengrass 특성을 사용해 볼 수 있습니다. 예를 들어 Docker 애플리케이션 배포 커넥터를 그룹에 추가한 후 Docker 컨테이너를 코어 디바이스에 배포하는 데 사용할 수 있습니다.

자격 증명을 입력 값으로 제공

1. 스크립트를 다운로드하여 시작합니다. 자격 증명 및 지정할 기타 입력 값을 제공합니다. 다음 예제에서는 세션 토큰을 포함하는 임시 자격 증명을 제공하는 방법을 보여줍니다.

   - 모든 기본값을 사용하려면:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   - 사용자 지정 값을 지정하려면:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   --region us-east-1
   --group-name Custom_Group_Name
   --core-name Custom_Core_Name
   --ggc-root-path /custom/ggc/root/path
   --deployment-timeout 300
   --log-path /customized/log/path
   --hello-world-lambda
   --verbose
   ```

> **Note**
>
> Raspbian 또는 OpenWrt 플랫폼에서 Greengrass 디바이스 설정을 실행하는 경우 자격 증명을 복사합니다. 디바이스를 재부팅한 후 다시 제공해야 합니다.
>
> `curl`을 사용하여 스크립트를 다운로드하려면 명령에서 `wget -q -O`를 `curl`로 바꿉니다.

2. 코어 디바이스에서 Raspbian 또는 OpenWRT를 실행 중인 경우 메시지가 표시되면 디바이스를 재부팅하고 자격 증명을 제공한 다음 스크립트를 다시 시작합니다.

   a. 디바이스를 재부팅하라는 메시지가 표시되면 다음 명령 중 하나를 실행합니다.

      Raspbian 플랫폼의 경우:

      ```
      sudo reboot
      ```

      OpenWRT 플랫폼의 경우:

      ```
      reboot
      ```

   b. 스크립트를 다시 시작합니다. 명령에 자격 증명을 포함해야 하지만 다른 입력 값은 포함하지 않아야 합니다. 예:

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      --aws-access-key-id AKIAIOSFODNN7EXAMPLE
      --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

   c. 이전 세션의 입력 값을 사용할지 새 설치를 시작할지 여부를 묻는 메시지가 표시되면 `yes`를 입력하여 입력 값을 다시 사용합니다.

      > **Note**
      >
      > 재부팅이 필요한 플랫폼에서는 이전 세션의 입력 값(자격 증명 제외)이 임시로 `GreengrassDeviceSetup.config.info` 파일에 저장됩니다.

   설정이 완료되면 터미널에 다음과 유사한 성공 상태 메시지가 표시됩니다.

```
================================================================================

Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.


Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

================================================================================
```

3. Hello World Lambda 함수를 포함한 경우 Greengrass 디바이스 설정은 Greengrass 그룹을 코어 디바이스에 배포합니다. Lambda 함수를 테스트하거나 그룹에서 Lambda 함수를 제거하는 방법을 알아보려면 시작하기 튜토리얼의 모듈 3-1에서 the section called “Lambda 함수가 코어 디바이스에서 실행 중인지 확인” 섹션을 계속 진행합니다.

> **Note**
>
> 콘솔에서 선택한 AWS 리전이 Greengrass 환경을 구성하는 데 사용한 것과 동일한지 확인합니다. 기본적으로 리전은 미국 서부(오레곤)입니다.

Hello World Lambda 함수를 포함하지 않은 경우 자신의 Lambda 함수를 생성하거나 다른 Greengrass 특성을 사용해 볼 수 있습니다. 예를 들어 Docker 애플리케이션 배포 커넥터를 그룹에 추가한 후 Docker 컨테이너를 코어 디바이스에 배포하는 데 사용할 수 있습니다.

모듈 1: Greengrass를 위한 환경 설정

이 모듈에서는 AWS IoT Greengrass가 AWS IoT Greengrass 코어 디바이스로 사용할 준비가 된 Raspberry Pi, Amazon EC2 인스턴스 또는 기타 디바이스를 즉시 사용하는 방법을 안내합니다.

> **Tip**
>
> 또는 코어 디바이스를 자동으로 설정하는 스크립트를 사용하려면 the section called “빠른 시작: Greengrass 디바이스 설정” 단원을 참조하십시오.

이 모듈은 완료하는 데 30분이 채 걸리지 않습니다.

시작하기 전에 이 자습서의 요구 사항을 읽으십시오. 그런 다음 다음 항목 중 하나의 설정 지침을 따릅니다. 코어 디바이스 유형에 적용되는 항목만 선택합니다.

주제

- Raspberry Pi 설정
- Amazon EC2 인스턴스 설정
- 다른 디바이스 설정

> **Note**
>
> 미리 만들어진 Docker 컨테이너에서 AWS IoT Greengrass를 실행하는 방법을 배우려면 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.”를 참조하십시오.

Raspberry Pi 설정

이 주제의 단계에 따라 AWS IoT Greengrass 코어로 사용할 Raspberry Pi를 설정합니다.

> **Tip**
>
> AWS IoT Greengrass에서는 다른 방법으로도 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 예를 들어 Greengrass 디바이스 설정을 사용하여 환경을 구성하고 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 또는 지원되는 Debian 플랫폼에서 APT 패키지 관리자를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 업그레이드할 수 있습니다. 자세한 내용은 the section called “AWS IoT Greengrass 코어 소프트웨어 설치” 섹션을 참조하세요.

처음으로 Raspberry Pi를 설정하는 경우, 아래 모든 단계에 따라야 합니다. 그렇지 않으면 9단계로 건너뛰어도 됩니다. 하지만 2단계에서 권장하는 대로 운영 체제를 사용해 Raspberry Pi의 이미지를 다시 생성하는 것이 좋습니다.

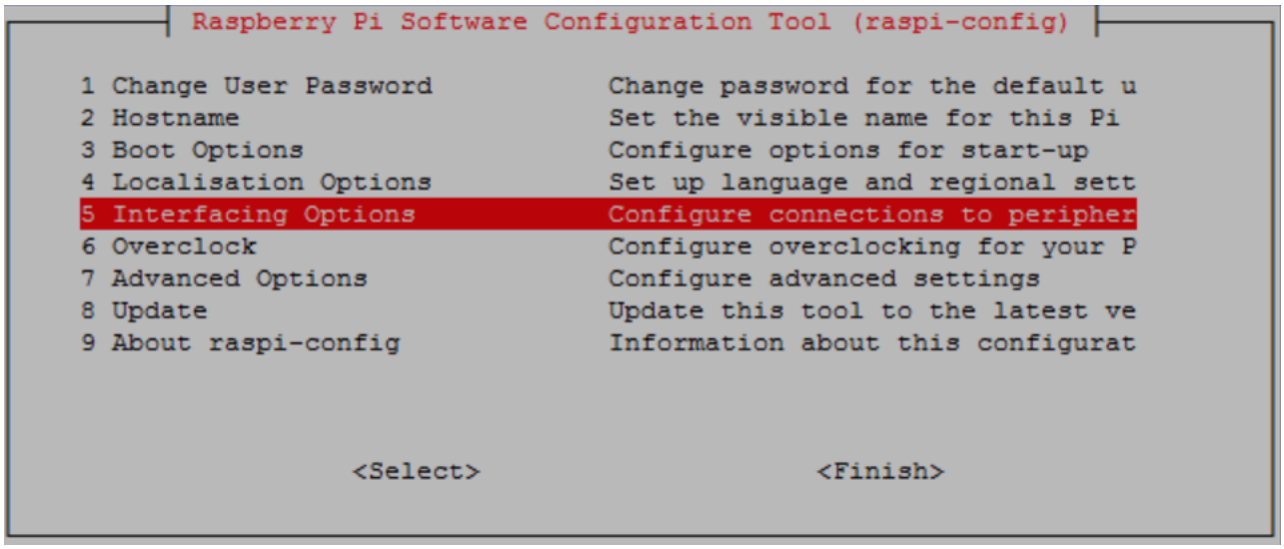
1. SD Memory Card Formatter와 같은 SD 카드 포맷터를 다운로드하고 설치합니다. SD 카드를 컴퓨터에 삽입합니다. 프로그램을 시작하고 SD 카드를 삽입한 드라이브를 선택합니다. SD 카드의 빠른 포맷을 수행할 수 있습니다.
2. Raspbian Buster 운영 체제를 `zip` 파일로 다운로드합니다.
3. SD 카드 쓰기 도구(예: Etcher)를 사용하여 도구의 지침에 따라 다운로드한 `zip` 파일을 SD 카드로 플래시합니다. 운영 체제 이미지가 크기 때문에 이 단계는 시간이 약간 걸릴 수 있습니다. 컴퓨터에서 SD 카드를 빼내고 Raspberry Pi에 microSD 카드를 삽입합니다.
4. 처음으로 부팅하는 경우, Raspberry Pi를 모니터(HDMI를 통해), 키보드, 마우스에 연결하는 것이 좋습니다. 그 다음 Pi를 마이크로 USB 전원에 연결하면 Raspbian 운영 체제가 시작됩니다.
5. 계속하기 전에 Pi의 키보드 레이아웃을 구성하는 것이 좋습니다. 그러려면 오른쪽 상단의 Raspberry 아이콘을 선택하고, 기본 설정과 Mouse and Keyboard Settings(마우스 및 키보드 설정)를 차례로 선택합니다. 그 다음 Keyboard(키보드) 탭에서 Keyboard Layout(키보드 레이아웃)을 선택한 다음 적절한 키보드 종류를 선택합니다.
6. 다음으로 Wi-Fi 네트워크 또는 이더넷 케이블을 통해 Raspberry Pi를 인터넷에 연결합니다.

   > **Note**
   >
   > 컴퓨터가 연결된 네트워크와 동일한 네트워크에 Raspberry Pi를 연결하고, 계속하기 전에 컴퓨터와 Raspberry Pi 모두 인터넷에 접속되어 있는지 확인합니다. 작업 환경에 있거나 방화벽 뒤에 있는 경우, Pi와 컴퓨터가 동일한 네트워크에 있도록 두 디바이스를 게스트 네트워크에 연결해야 할 수도 있습니다. 하지만 이렇게 하면 인트라넷과 같은 로컬 네트워크 리소스와 컴퓨터의 연결이 해제될 수 있습니다. 한 가지 해결책은 Pi를 게스트 Wi-Fi 네트워크에 연결하고 이더넷 케이블을 통해 컴퓨터를 게스트 Wi-Fi 네트워크 및 로컬 네트워크에 연결하는 것입니다. 이 구성에서는 컴퓨터를 게스트 Wi-Fi 네트워크를 통해 Raspberry Pi에, 이더넷 케이블을 통해 로컬 네트워크 리소스에 연결할 수 있습니다.

7. Pi에 원격으로 연결하려면 Pi에서 SSH를 설정해야 합니다. Raspberry Pi에서 터미널 창을 열고 다음 명령을 실행합니다.

   ```
   sudo raspi-config
   ```

다음과 같은 모양이어야 합니다.



아래로 스크롤해 [Interfacing Options]를 선택한 다음 [P2 SSH]를 선택합니다. 메시지가 나타나면 예를 선택합니다. (Tab 키를 누른 다음 Enter 키를 누릅니다). 이제 SSH가 활성화되어야 합니다. 확인(OK)을 선택합니다. Tab키를 사용하여 완료를 선택한 다음 Enter를 누릅니다. Raspberry Pi가 자동으로 재부팅되지 않는 경우 다음 명령을 실행합니다.

```
sudo reboot
```

8. Raspberry Pi의 터미널에서 다음 명령을 실행합니다.

```
hostname -I
```

이렇게 하면 Raspberry Pi의 IP 주소가 반환됩니다.

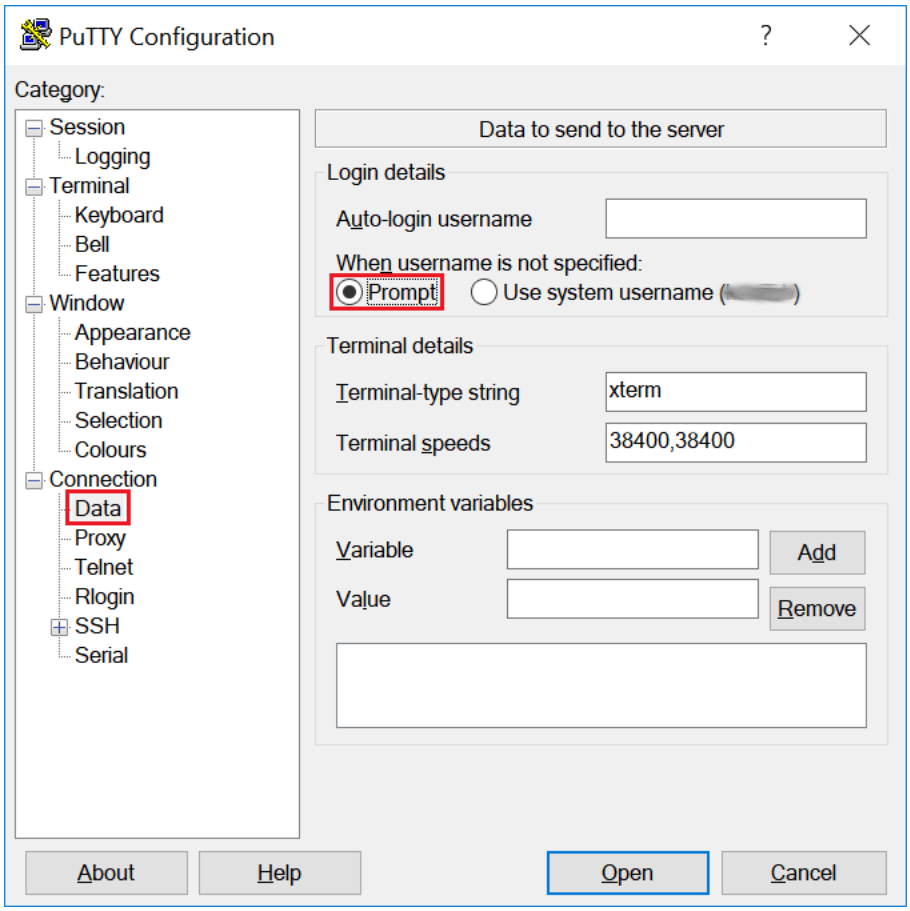
> **Note**
>
> 그 다음에 ECDSA 키 지문 관련 메시지(`Are you sure you want to continue connecting (yes/no)?`)를 받으면 `yes`를 입력합니다. Raspberry Pi의 기본 암호는 **`raspberry`**입니다.

macOS를 사용하는 경우에는 터미널 창을 열고 다음을 입력합니다.

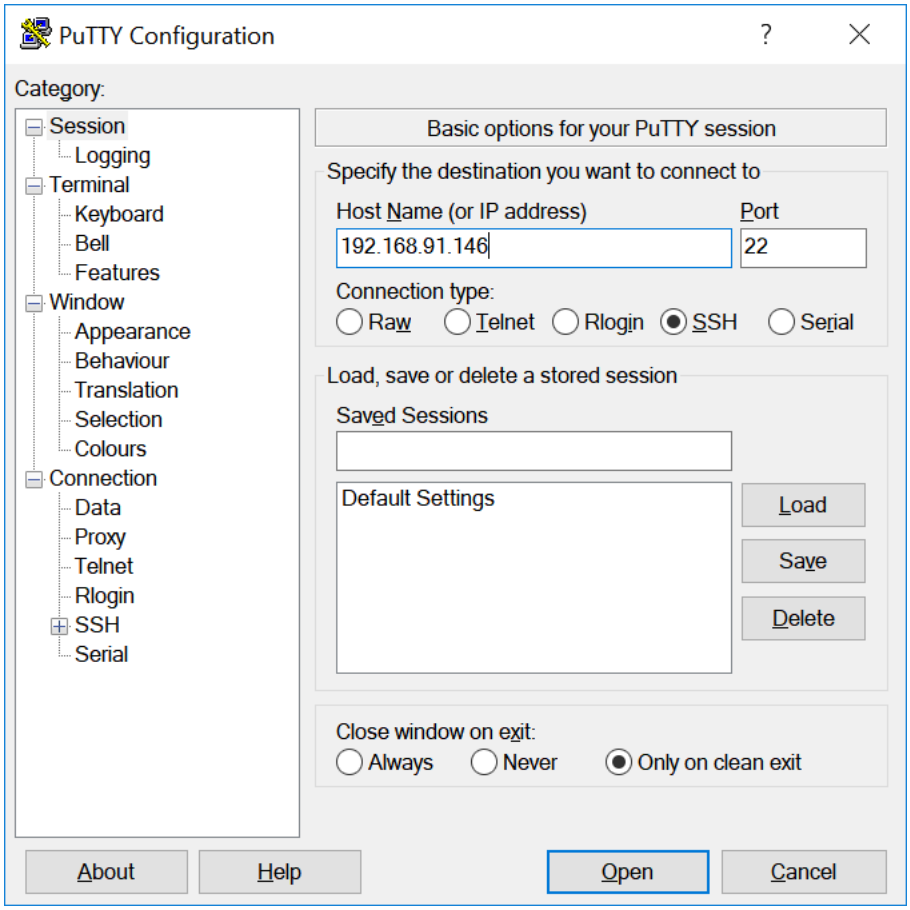
```
ssh pi@IP-address
```

*IP-address*는 `hostname -I` 명령을 사용하여 가져온 Raspberry Pi의 IP 주소입니다.

Windows를 사용하는 경우, PuTTY를 설치하고 구성해야 합니다. 연결을 확장하고 데이터를 선택한 다음 프롬프트가 선택되어 있는지 확인합니다.

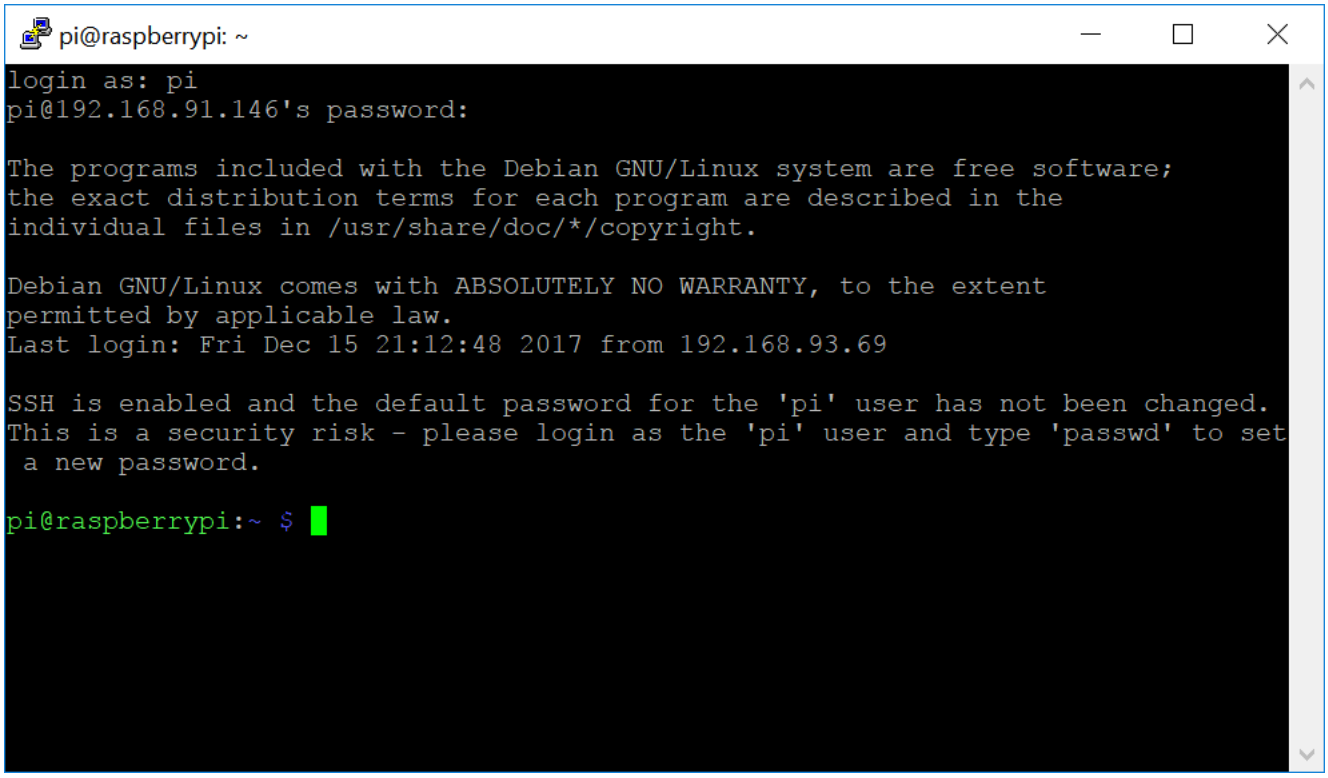


다음으로 Session(세션)을 선택하고 Raspberry Pi의 IP 주소를 입력한 다음 기본 설정을 사용하여 열기를 선택합니다.



PuTTY 보안 알림이 표시되면 예를 선택합니다.

기본 Raspberry Pi 로그인 및 암호는 각각 **pi**와 **raspberry**입니다.



> **Note**
>
> 컴퓨터가 VPN을 사용하여 원격 네트워크에 연결되어 있는 경우, SSH를 사용하여 컴퓨터에서 Raspberry Pi에 연결하기가 어려울 수 있습니다.

9. 이제 AWS IoT Greengrass에 대해 Raspberry Pi를 설정할 준비가 되었습니다. 먼저 로컬 Raspberry Pi 터미널 창 또는 SSH 터미널 창에서 다음 명령을 실행합니다.

> **Tip**
>
> AWS IoT Greengrass에서는 다른 방법으로도 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 예를 들어 Greengrass 디바이스 설정을 사용하여 환경을 구성하고 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 또는 지원되는 Debian 플랫폼에서 APT 패키지 관리자를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 업그레이드할 수 있습니다. 자세한 내용은 the section called “AWS IoT Greengrass 코어 소프트웨어 설치” 섹션을 참조하세요.

```
sudo adduser --system ggc_user
```

```
sudo addgroup --system ggc_group
```

10. Pi 디바이스에 대한 보안을 개선하려면 시작 시 운영 체제에서 hardlink 및 softlink(symlink) 보호를 활성화합니다.

    a. `98-rpi.conf` 파일로 이동합니다.

    ```
    cd /etc/sysctl.d
    ls
    ```

    > **Note**
    >
    > `98-rpi.conf` 파일이 보이지 않으면 `README.sysctl` 파일의 지침을 따릅니다.

    b. 텍스트 편집기(예: Leafpad, GNU nano 또는 vi)를 사용하여 파일 끝에 다음 두 줄을 추가합니다. `sudo` 명령을 사용하여 루트로 편집해야 할 수 있습니다(예: `sudo nano 98-rpi.conf`).

    ```
    fs.protected_hardlinks = 1
    fs.protected_symlinks = 1
    ```

    c. Pi를 재부팅합니다.

    ```
    sudo reboot
    ```

    약 1분 후 SSH를 사용하여 Pi에 연결하고 다음 명령을 실행하여 변경 사항을 확인합니다.

    ```
    sudo sysctl -a 2> /dev/null | grep fs.protected
    ```

    `fs.protected_hardlinks = 1`와 `fs.protected_symlinks = 1`가 보여야 합니다.

11. 명령줄 부트 파일을 편집하여 메모리 cgroup을 활성화하고 탑재합니다. 이렇게 하면 AWS IoT Greengrass에서 Lambda 함수에 대한 메모리 제한을 설정할 수 있습니다. 또한 Cgroup은 기본 컨테이너화 모드에서 AWS IoT Greengrass을(를) 실행해야 합니다.

    a. `boot` 디렉터리로 이동합니다.

    ```
    cd /boot/
    ```

b. 텍스트 편집기를 사용하여 `cmdline.txt`을 엽니다. 다음을 새 줄로 추가하지 말고 기존 줄의 끝에 추가합니다. `sudo` 명령을 사용하여 루트로 편집해야 할 수 있습니다(예: `sudo nano cmdline.txt`).

```
cgroup_enable=memory cgroup_memory=1
```

c. 이제 Pi를 재부팅합니다.

```
sudo reboot
```

이제 AWS IoT Greengrass에 Raspberry Pi를 사용할 준비가 되었습니다.

12. 선택 사항. Java 8 런타임을 설치합니다. 이 런타임은 스트림 관리자에 필요합니다. 이 튜토리얼에서는 스트림 관리자를 사용하지 않지만 기본적으로 스트림 관리자를 활성화하는 기본 그룹 생성 워크플로를 사용합니다. 그룹을 배포하기 전에 다음 명령을 사용하여 코어 디바이스에 Java 8 런타임을 설치하거나 스트림 관리자를 비활성화해야 합니다. 스트림 관리자를 비활성화하는 지침은 모듈 3에 나와 있습니다.

```
sudo apt install openjdk-8-jdk
```

13. 필요한 모든 종속성이 있는지 확인하려면 GitHub의 AWS IoT Greengrass 샘플 리포지토리에서 Greengrass 종속성 확인 프로그램을 다운로드하여 실행합니다. 이 명령은 `Downloads` 디렉터리에 있는 종속성 확인 스크립트의 압축을 풀고 실행합니다.

> **Note**
>
> Raspbian 커널 버전 5.4.51을 실행하는 경우 종속성 검사기가 실패할 수 있습니다. 이 버전은 메모리 cgroup을 제대로 마운트하지 않습니다. 이로 인해 컨테이너 모드에서 실행되는 Lambda 함수가 실패할 수 있습니다.
> 커널 업데이트에 대한 자세한 내용은 Raspberry Pi 포럼에서 커널 업그레이드 후 로드되지 않은 Cgroups를 참조하십시오.

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
```

```
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

more가 나타나면 Spacebar 키를 눌러 다른 텍스트 화면을 표시합니다.

> **Important**
>
> 이 튜토리얼에서 로컬 Lambda 함수를 실행하려면 Python 3.7 런타임이 필요합니다. 스트림 관리자를 활성화한 경우에도 Java 8 런타임이 필요합니다. `check_ggc_dependencies` 스크립트에서 이러한 누락된 런타임 사전 조건에 대한 경고를 생성하는 경우 계속하기 전에 해당 사전 조건을 설치해야 합니다. 선택 사항인 다른 런타임 사전 조건 누락 관련 경고는 무시해도 됩니다.

modprobe 명령에 대한 자세한 내용을 보려면 터미널에서 man modprobe를 실행합니다.

귀하의 Raspberry Pi 구성이 완료되었습니다. 계속해서 the section called “모듈 2: AWS IoT Greengrass 코어 소프트웨어 설치”로 이동하십시오.

Amazon EC2 인스턴스 설정

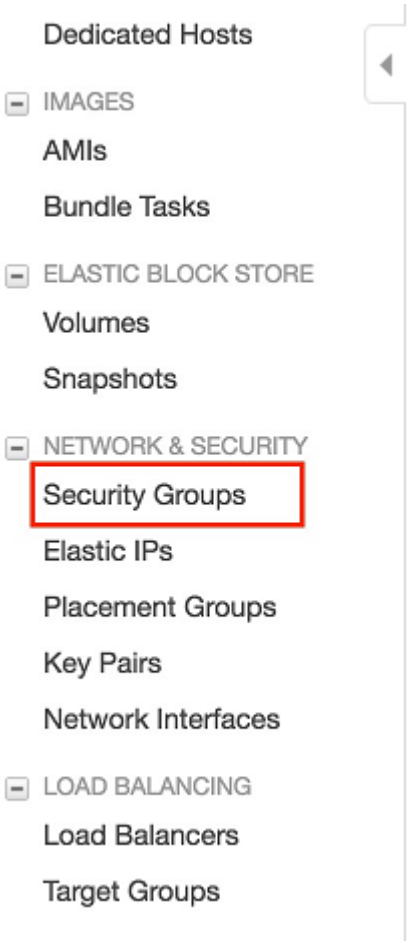
이 주제의 단계에 따라 AWS IoT Greengrass 코어로 사용할 Amazon EC2 인스턴스를 설정합니다.

> **Tip**
>
> 또는 환경을 설정하고 AWS IoT Greengrass Core 소프트웨어를 설치하는 스크립트를 사용하려면 을 참조하십시오the section called “빠른 시작: Greengrass 디바이스 설정”.

Amazon EC2 인스턴스를 사용하여 이 자습서를 완료할 수 있지만 물리적 하드웨어와 함께 사용하는 것이 가장 좋습니다. AWS IoT Greengrass 가능하면 Amazon EC2 인스턴스를 사용하는 대신 Raspberry Pi를 설정하는 것이 좋습니다. Raspberry Pi를 사용하는 경우 이 주제의 단계를 따를 필요가 없습니다.

1. Amazon Linux AMI를 사용하여 AWS Management Console에 로그인하고 Amazon EC2 인스턴스를실행합니다. Amazon EC2 인스턴스에 대한 자세한 내용은 Amazon EC2 시작 가이드를 참조하세요.
2. Amazon EC2 인스턴스가 실행된 후 포트 8883을 활성화하여 들어오는 MQTT 통신을 허용하여 다른 디바이스가 코어에 연결할 수 있도록 합니다. AWS IoT Greengrass

   a. Amazon EC2 콘솔의 탐색 창에서 보안 그룹을 선택합니다.



   b. 방금 시작한 인스턴스에 대한 보안 그룹을 선택한 다음 인바운드 규칙 탭을 선택합니다.

   c. 인바운드 규칙 편집을 선택합니다.

      포트 8883을 활성화하려면 사용자 지정 TCP 규칙을 보안 그룹에 추가합니다. 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가를 참조하십시오.

   d. 인바운드 규칙 편집 페이지에서 규칙 추가을 선택하고 다음 설정을 입력한 다음 저장를 선택합니다.

      - 유형의 경우 사용자 지정 TCP 규칙을 선택합니다.

- 포트 범위에 **8883**을 입력합니다.
- Source(소스)에서 Anywhere(위치 무관)를 선택합니다.
- 설명에 **MQTT Communications**를 입력합니다.

3. Amazon EC2 인스턴스에 연결합니다.

   a. 탐색 창에서 인스턴스를 선택하고 해당 인스턴스를 선택한 다음 연결을 선택합니다.

   b. Connect To Your Instance(인스턴스에 연결) 페이지의 지침에 따라 SSH와 프라이빗 키 파일을 사용하여 인스턴스에 연결합니다.

   Windows의 경우 PuTTY를 사용하거나 macOS의 경우 Terminal을 사용할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결을 참조하십시오.

   이제 AWS IoT Greengrass에 대한 Amazon EC2 인스턴스를 설정할 준비가 되었습니다.

4. Amazon EC2 인스턴스에 연결한 후 `ggc_user` 및 `ggc_group` 계정을 생성합니다.

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

> **Note**
>
> 시스템에서 `adduser` 명령을 사용할 수 없는 경우 다음 명령을 사용하십시오.
>
> ```
> sudo useradd --system ggc_user
> ```

5. 보안을 개선하려면 시작 시 Amazon EC2 인스턴스의 운영 체제에서 hardlink 및 softlink(symlink) 보호가 활성화되어 있는지 확인합니다.

> **Note**
>
> hardlink 및 softlink 보호를 활성화하는 단계는 운영 체제마다 다릅니다. 해당 배포의 설명서를 참조하십시오.

   a. 다음 명령을 실행하여 hardlink 및 softlink 보호가 활성화되었는지 확인합니다.

```
sudo sysctl -a | grep fs.protected
```

hardlink 및 softlink가 1로 설정된 경우 보호가 올바르게 활성화된 것입니다. 6단계로 이동합니다.

> **Note**
>
> Softlink는 `fs.protected_symlinks`로 표시됩니다.

b. hardlink 및 softlink가 1로 설정되지 않은 경우 이러한 보호를 활성화합니다. 시스템 구성 파일로 이동합니다.

```
cd /etc/sysctl.d
ls
```

c. 자주 사용하는 텍스트 편집기(Leafpad, GNU nano 또는 vi)를 사용하여 시스템 구성 파일 끝에 다음 두 줄을 추가합니다. Amazon Linux 1에서 이는 `00-defaults.conf` 파일입니다. Amazon Linux 2에서 이는 `99-amazon.conf` 파일입니다. 파일에 쓰기 위해 권한을 변경하거나(`chmod` 명령 사용), `sudo` 명령을 사용하여 루트로 편집해야 할 수 있습니다(예: `sudo nano 00-defaults.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

d. Amazon EC2 인스턴스를 재부팅합니다.

```
sudo reboot
```

몇 분 후 SSH를 사용하여 인스턴스에 연결하고 다음 명령을 실행하여 변경 사항을 확인합니다.

```
sudo sysctl -a | grep fs.protected
```

hardlink 및 softlink가 1로 설정된 것이 보여야 합니다.

6. 다음 스크립트를 추출하고 실행하여 Linux 제어 그룹(cgroups)을 탑재합니다. AWS IoT Greengrass 이를 통해 Lambda 함수의 메모리 제한을 설정할 수 있습니다. 또한 Cgroup은 기본 컨테이너화 AWS IoT Greengrass 모드에서 실행해야 합니다.

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-
mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

이제 Amazon EC2 인스턴스를 AWS IoT Greengrass에 사용할 준비가 되었습니다.

7. 선택 사항입니다. Java 8 런타임을 설치합니다. 이 런타임은 스트림 관리자에 필요합니다. 이 튜토리얼에서는 스트림 관리자를 사용하지 않지만 기본적으로 스트림 관리자를 활성화하는 기본 그룹 생성 워크플로를 사용합니다. 그룹을 배포하기 전에 다음 명령을 사용하여 코어 장치에 Java 8 런타임을 설치하거나 스트림 관리자를 비활성화해야 합니다. 스트림 관리자를 비활성화하는 지침은 모듈 3에 나와 있습니다.

   - Debian 기반 배포판의 경우:

     ```
     sudo apt install openjdk-8-jdk
     ```

   - Red Hat 기반 배포판의 경우:

     ```
     sudo yum install java-1.8.0-openjdk
     ```

8. 필요한 종속성이 모두 있는지 확인하려면 샘플 저장소에서AWS IoT Greengrass Greengrass 종속성 검사기를 다운로드하여 실행하십시오. GitHub 이 명령은 Amazon EC2 인스턴스에서 종속성 확인 스크립트를 다운로드하여 압축을 풀고 실행합니다.

   ```
   mkdir greengrass-dependency-checker-GGCv1.11.x
   cd greengrass-dependency-checker-GGCv1.11.x
   wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
   dependency-checker-GGCv1.11.x.zip
   unzip greengrass-dependency-checker-GGCv1.11.x.zip
   cd greengrass-dependency-checker-GGCv1.11.x
   sudo ./check_ggc_dependencies | more
   ```

   Important

   이 튜토리얼에서 로컬 Lambda 함수를 실행하려면 Python 3.7 런타임이 필요합니다. 스트림 관리자를 활성화한 경우에도 Java 8 런타임이 필요합니다. `check_ggc_dependencies` 스크립트에서 이러한 누락된 런타임 사전 조건에 대한 경고

를 생성하는 경우 계속하기 전에 해당 사전 조건을 설치해야 합니다. 선택 사항인 다른 런타임 사전 조건 누락 관련 경고는 무시해도 됩니다.

Amazon EC2 인스턴스 구성이 완료되었습니다. 계속해서 the section called “모듈 2: AWS IoT Greengrass 코어 소프트웨어 설치”로 이동하십시오.

다른 디바이스 설정

이 주제의 단계에 따라 AWS IoT Greengrass 코어로 사용할 디바이스(Raspberry Pi 제외)를 설정합니다.

> **Tip**
>
> 또는 환경을 설정하고 AWS IoT Greengrass Core 소프트웨어를 자동으로 설치하는 스크립트를 사용하려면 the section called “빠른 시작: Greengrass 디바이스 설정”을 참조하십시오.

AWS IoT Greengrass를 처음 사용하는 경우 Raspberry Pi 또는 Amazon EC2 인스턴스를 코어 디바이스로 사용하고 해당하는 디바이스의 설정 단계를 따르는 것이 좋습니다.

Yocto Project를 사용하여 사용자 지정 Linux 기반 시스템을 구축하려는 경우 `meta-aws` 프로젝트의 AWS IoT Greengrass Bitbake Recipe를 사용할 수 있습니다. 또한 이 레시피는 임베디드 애플리케이션용 AWS 엣지 소프트웨어를 지원하는 소프트웨어 플랫폼을 개발하는 데도 도움이 됩니다. Bitbake 빌드는 디바이스에 AWS IoT Greengrass Core 소프트웨어를 설치, 구성 및 자동으로 실행합니다.

Yocto Project

하드웨어 아키텍처에 관계없이 임베디드 애플리케이션을 위한 사용자 지정 Linux 기반 시스템을 구축할 수 있도록 지원하는 오픈 소스 협업 프로젝트입니다. 자세한 내용은 Yocto 프로젝트를 참조하십시오.

`meta-aws`

Yocto 레시피를 제공하는 AWS 관리형 프로젝트입니다. 레시피를 사용하여 OpenEmbedded 및 Yocto Project로 구축된 Linux 기반 시스템에서 AWS 엣지 소프트웨어를 개발할 수 있습니다. 커뮤니티에서 지원하는 이 기능에 대한 자세한 내용은 GitHub에서 `meta-aws` 프로젝트를 참조하십시오.

`meta-aws-demos`

`meta-aws` 프로젝트 데모가 포함된 AWS 관리형 프로젝트. 통합 프로세스에 대한 더 많은 예시는 GitHub의 `meta-aws-demos` 프로젝트를 참조하십시오.

다른 디바이스나 지원되는 플랫폼을 사용하려면 이 주제의 단계를 따르십시오.

1. 코어 디바이스가 NVIDIA Jetson 디바이스인 경우 먼저 JetPack 4.3 인스톨러에서 펌웨어를 플래시해야 합니다. 다른 디바이스를 구성하고 있는 경우 2단계로 건너뜁니다.

   > **Note**
   >
   > 사용하는 JetPack 설치 관리자 버전은 대상 CUDA Toolkit 버전에 따라 다릅니다. 다음 지침에서는 JetPack 4.3 및 CCUDA Toolkit 10.0를 사용합니다. 디바이스에 적합한 버전 사용에 대한 자세한 내용은 NVIDIA 설명서의 Jetpack 설치 방법을 참조하십시오.

   a. Ubuntu 16.04 이상을 실행하는 물리적 데스크톱에서 NVIDIA 설명서의 JetPack 다운로드 및 설치(4.3)를 따라 JetPack 4.3 설치 프로그램으로 펌웨어를 플래시합니다.

      설치 관리자의 지침을 따라 Jetson 보드의 모든 패키지 및 종속성을 설치하고, 데스크탑과 Micro-B 케이블을 연결해야 합니다.

   b. 정상 모드로 보드를 재부팅하고 디스플레이를 보드에 연결합니다.

      > **Note**
      >
      > SSH를 사용하여 Jetson 보드에 연결할 때 기본 사용자 이름(**nvidia**)과 기본 암호(**nvidia**)를 사용합니다.

2. 다음 명령을 실행하여 `ggc_user` 사용자와 `ggc_group` 그룹을 생성합니다. 실행하는 명령은 코어 디바이스에 설치된 배포에 따라 다릅니다.

   - 코어 디바이스가 OpenWrt를 실행하는 경우 다음 명령을 실행하십시오.

     ```
     opkg install shadow-useradd
     opkg install shadow-groupadd
     useradd --system ggc_user
     groupadd --system ggc_group
     ```

- 또는 다음 명령을 실행하십시오.

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

> **Note**
>
> 시스템에서 addgroup 명령을 사용할 수 없는 경우 다음 명령을 사용하십시오.
>
> ```
> sudo groupadd --system ggc_group
> ```

3. 선택 사항. Java 8 런타임을 설치합니다. 이 런타임은 스트림 관리자에 필요합니다. 이 자습서에서는 스트림 관리자를 사용하지 않지만 기본적으로 스트림 관리자를 활성화하는 기본 그룹 생성 워크플로를 사용합니다. 그룹을 배포하기 전에 다음 명령을 사용하여 코어 디바이스에 Java 8 런타임을 설치하거나 스트림 관리자를 비활성화해야 합니다. 스트림 관리자를 비활성화하는 지침은 모듈 3에 나와 있습니다.

   - Debian 기반 또는 Ubuntu 기반 배포판의 경우:

   ```
   sudo apt install openjdk-8-jdk
   ```

   - Red Hat 기반 배포판의 경우:

   ```
   sudo yum install java-1.8.0-openjdk
   ```

4. 필요한 모든 종속성이 있는지 확인하려면 GitHub의 AWS IoT Greengrass 샘플 리포지토리에서 Greengrass 종속성 확인 프로그램을 다운로드하여 실행합니다. 이 명령은 종속성 확인 스크립트의 압축을 풀고 실행합니다.

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

> **Note**
>
> `check_ggc_dependencies` 스크립트는 AWS IoT Greengrass 지원 플랫폼에서 실행되며 특정한 Linux 시스템 명령이 필요합니다. 자세한 내용은 종속성 확인 프로그램의 Readme를 참조하십시오.

5. 종속성 확인 프로그램 출력에 표시된 대로 필요한 모든 종속성을 디바이스에 설치합니다. 누락된 커널 수준 종속성을 위해 커널을 다시 컴파일해야 할 수 있습니다. Linux 제어 그룹(`cgroups`)을 탑재하려면 cgroupfs-mount 스크립트를 실행합니다. 이렇게 하면 AWS IoT Greengrass에서 Lambda 함수에 대한 메모리 제한을 설정할 수 있습니다. 또한 Cgroup은 기본 컨테이너화 모드에서 AWS IoT Greengrass을(를) 실행해야 합니다.

   출력에 오류가 나타나지 않으면 디바이스에서 AWS IoT Greengrass이(가) 성공적으로 실행될 수 있을 것입니다.

   > **Important**
   >
   > 이 튜토리얼에서 로컬 Lambda 함수를 실행하려면 Python 3.7 런타임이 필요합니다. 스트림 관리자를 활성화한 경우에도 Java 8 런타임이 필요합니다. `check_ggc_dependencies` 스크립트에서 이러한 누락된 런타임 사전 조건에 대한 경고를 생성하는 경우 계속하기 전에 해당 사전 조건을 설치해야 합니다. 선택 사항인 다른 런타임 사전 조건 누락 관련 경고는 무시해도 됩니다.

   AWS IoT Greengrass 요구 사항 및 종속성 목록은 the section called “지원되는 플랫폼 및 요구 사항” 단원을 참조하십시오.

모듈 2: AWS IoT Greengrass 코어 소프트웨어 설치

이 모듈에서는 선택한 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하는 방법을 살펴봅니다. 이 모듈에서는 먼저 Greengrass 그룹과 코어를 생성합니다. 그런 다음 코어 디바이스에서 소프트웨어를 다운로드, 구성 및 시작합니다. AWS IoT Greengrass 코어 소프트웨어 기능에 대한 자세한 내용은 the section called “AWS IoT Greengrass 코어 구성” 단원을 참조하십시오.

시작하기 전에 선택한 디바이스에 대해 모듈 1의 설정 단계를 완료했는지 확인하십시오.

> **Tip**
>
> AWS IoT Greengrass에서는 다른 방법으로도 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 예를 들어 Greengrass 디바이스 설정을 사용하여 환경을 구성하고 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다. 또는 지원되는 Debian 플랫폼에서 APT 패키지 관리자를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 업그레이드할 수 있습니다. 자세한 내용은 the section called “AWS IoT Greengrass 코어 소프트웨어 설치” 섹션을 참조하세요.

이 모듈은 완료하는 데 30분이 채 걸리지 않습니다.

주제

- Greengrass 코어로 사용할 AWS IoT 사물 프로비저닝
- 코어의 AWS IoT Greengrass 그룹 생성
- 코어 디바이스에 AWS IoT Greengrass 설치 및 실행

Greengrass 코어로 사용할 AWS IoT 사물 프로비저닝

Greengrass 코어는 AWS IoT Greengrass Core 소프트웨어를 실행하여 로컬 IoT 프로세스를 관리하는 디바이스입니다. Greengrass 코어를 설정하려면 AWS IoT에 연결된 디바이스 또는 논리적 엔터티를 나타내는 AWS IoT 사물을 생성합니다. 디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 AWS IoT에 대한 액세스를 허용하는 디지털 인증서와 키를 사용할 수 있습니다. AWS IoT 정책을 사용하여 디바이스가 AWS IoT 및 AWS IoT Greengrass 서비스와 통신할 수 있도록 허용합니다.

이 섹션에서는 디바이스를 Greengrass 코어로 사용하기 위한 AWS IoT 사물로 등록합니다.

AWS IoT 사물을 생성하려면

1. AWS IoT 콘솔로 이동합니다.
2. 관리에서 모든 디바이스를 확장한 다음 사물을 선택합니다.
3. 사물 페이지에서 사물 생성을 선택합니다.
4. 사물 생성 페이지에서 단일 사물 생성을 선택한 후 다음을 선택합니다.
5. 세부 사물 속성 페이지에서 다음 작업을 수행합니다.

    a. 사물 이름에 **`MyGreengrassV1Core`**와(과) 같이 디바이스를 나타내는 이름을 입력합니다.

    b. 다음(Next)을 선택합니다.

6. 디바이스 인증서 구성 페이지에서 다음을 선택합니다.
7. 인증서에 정책 첨부 페이지에서 다음 중 하나를 수행합니다.
   - 코어에 필요한 권한을 부여하는 기존 정책을 선택한 다음 사물 생성을 선택합니다.

     디바이스가 AWS 클라우드에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.
   - 핵심 디바이스 권한을 부여하는 새 정책을 만들고 첨부하십시오. 해결 방법:

     a. [정책 생성(Create policy)]을 선택합니다.

        [Create policy(정책 생성)] 페이지가 새 탭에서 열립니다.

     b. [Create policy(정책 생성)] 페이지에서 다음을 수행합니다.

        i. 정책 이름에는 **`GreengrassV1CorePolicy`**와(과) 같이 정책을 설명하는 이름을 입력합니다.

        ii. 정책 설명 탭의 정책 문서에서 JSON을 선택합니다.

        iii. 다음 정책 문서를 입력합니다. 이 정책을 통해 코어는 AWS IoT Core 서비스와 통신하고, 디바이스 섀도우와 상호 작용하고, AWS IoT Greengrass 서비스와 통신할 수 있습니다. 사용 사례에 따라 이 정책의 액세스를 제한하는 방법에 대한 자세한 내용은 AWS IoT Greengrass 코어 디바이스에 대한 최소 AWS IoT 정책을(를) 참조하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 생성(Create)을 선택하여 정책을 생성합니다.

c. 인증서에 정책 연결 페이지가 열려 있는 브라우저 탭으로 돌아가십시오. 해결 방법:

i. 정책 목록에서 GreengrassV1CorePolicy와 같이 만든 정책을 선택합니다.

새 정책이 나타나지 않으면 새로고침 아이콘을 선택합니다.

ii. 사물 생성(Create thing)을 선택합니다.

코어가 AWS IoT에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.

8. 인증서에 정책 연결 페이지가 열려 있는 브라우저 탭으로 돌아가십시오. 해결 방법:

a. 정책 목록에서 GreengrassV1CorePolicy과 같이 만든 정책을 선택합니다.

새 정책이 나타나지 않으면 새로고침 아이콘을 선택합니다.

b. 사물 생성(Create thing)을 선택합니다.

코어가 AWS IoT에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.

9. 인증서 및 키 다운로드 모달에서 디바이스의 인증서를 다운로드합니다.

> **Important**
>
> 완료를 선택하기 전에 보안 리소스를 다운로드합니다.

해결 방법:

a. 디바이스 인증서의 경우 다운로드를 선택하여 디바이스 인증서를 다운로드합니다.
b. 공개 키 파일의 경우 다운로드를 선택하여 인증서의 공개 키를 다운로드합니다.
c. 개인 키 파일의 경우 다운로드를 선택하여 인증서의 개인 키 파일을 다운로드합니다.
d. AWS IoT 개발자 안내서의 서버 인증을 검토하고 적절한 루트 CA 인증서를 선택합니다. Amazon Trust Services(ATS) 엔드포인트와 ATS 루트 CA 인증서를 사용하는 것이 좋습니다. 루트 CA 인증서에서 루트 CA 인증서용 다운로드를 선택합니다.
e. 완료(Done)를 선택합니다.

디바이스 인증서 및 키의 파일 이름에 공통적으로 나타나는 인증서 ID를 기록해 둡니다. 나중에 필요합니다.

코어의 AWS IoT Greengrass 그룹 생성

AWS IoT Greengrass 그룹에는 클라이언트 디바이스, Lambda 함수 및 커넥터와 같은 구성 요소에 대한 설정 및 기타 정보가 포함됩니다. 그룹은 구성 요소가 서로 상호 작용하는 방식을 포함하여 코어의 구성을 정의합니다.

이 부분에서는 코어에 대해 그룹을 생성합니다.

> **Tip**
>
> AWS IoT Greengrass API를 이용해 그룹을 생성하고 배포하는 예제는 GitHub에 있는 gg_group_setup 리포지토리를 참조하십시오.

코어의 그룹을 생성하려면

1. AWS IoT 콘솔로 이동합니다.
2. 관리에서 Greengrass 디바이스를 확장하고 그룹(V1)을 선택합니다.

> **Note**
>
> Greengrass 디바이스 메뉴가 표시되지 않으면 AWS IoT Greengrass V1을 지원하는 AWS 리전으로 변경합니다. 지원되는 리전 목록은 AWS 일반 참조에서 AWS IoT Greengrass V1 엔드포인트 및 할당량을 참조하십시오. AWS IoT Greengrass V1이 사용 가능한 리전에서 코어에 맞는 AWS IoT 사물을 생성해야 합니다.

3. Greengrass 그룹 페이지에서 그룹 생성을 선택합니다.
4. Greengrass 그룹 생성 페이지에서 다음을 수행합니다.
   a. Greengrass 그룹 이름에 그룹을 설명하는 이름을 입력합니다(예: **`MyGreengrassGroup`**).
   b. Greengrass 코어의 경우 이전에 만든 AWS IoT 사물을 선택합니다(예: MyGreengrassV1Core).

      콘솔은 자동으로 사물의 디바이스 인증서를 선택합니다.
   c. 그룹 생성을 선택합니다.

코어 디바이스에 AWS IoT Greengrass 설치 및 실행

> **Note**
>
> 이 튜토리얼은 Raspberry Pi에서 AWS IoT Greengrass 코어 소프트웨어를 실행하는 지침을 제공하지만 지원되는 다른 디바이스를 사용해도 됩니다.

이 섹션에서는 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 구성, 설치 및 실행합니다.

AWS IoT Greengrass를 설치 및 실행하려면

1. 이 안내서의 AWS IoT Greengrass 코어 소프트웨어에서 AWS IoT Greengrass 코어 소프트웨어 설치 패키지를 다운로드합니다. 해당 코어 디바이스의 CPU 아키텍처, 배포 및 OS에 가장 적합한 패키지를 선택하십시오.
   - Raspberry Pi의 경우 ARMv7L 아키텍처 및 Linux 운영 체제용 패키지를 다운로드하십시오.
   - Amazon EC2 인스턴스의 경우 x86_64 아키텍처 및 Linux 운영 체제용 패키지를 다운로드합니다.

- NVIDIA Jetson TX2의 경우 Armv8(AArch64) 아키텍처 및 Linux 운영 체제용 패키지를 다운로드합니다.
- Intel Atom의 경우, x86_64 아키텍처 및 Linux 운영 체제용 패키지를 다운로드합니다.

2. 이전 단계에서 다음 다섯 파일을 컴퓨터에 다운로드했습니다.

- `greengrass-`*`OS-architecture`*`-1.11.6.tar.gz` – 이 압축 파일에는 코어 디바이스에서 실행되는 AWS IoT Greengrass 코어 소프트웨어가 포함되어 있습니다.
- *`certificateId`*`-certificate.pem.crt` – 디바이스 인증서 파일.
- *`certificateId`*`-public.pem.key` – 디바이스 인증서의 공개 키 파일.
- *`certificateId`*`-private.pem.key` – 디바이스 인증서의 개인 키 파일.
- `AmazonRootCA1.pem` – Amazon 루트 인증 기관(CA) 파일.

이 단계에서는 이러한 파일을 컴퓨터에서 코어 디바이스로 전송합니다. 해결 방법:

a. Greengrass 코어 디바이스의 IP 주소를 모를 경우 코어 디바이스에서 터미널을 열고 다음 명령을 실행합니다.

> **Note**
>
> 일부 디바이스의 경우 이 명령이 올바른 IP 주소를 반환하지 않을 수 있습니다. 디바이스 IP 주소를 검색하려면 해당 디바이스의 설명서를 참조하십시오.

```
hostname -I
```

b. 컴퓨터에서 코어 디바이스로 이러한 파일을 전송합니다. 파일 전송 단계는 컴퓨터의 운영 체제에 따라 달라집니다. Raspberry Pi 디바이스에 파일을 전송하는 방법을 설명하는 단계를 보려면 해당 운영 체제를 선택하십시오.

> **Note**
>
> Raspberry Pi의 경우, 기본 사용자 이름은 **`pi`**이고 기본 암호는 **`raspberry`**입니다. NVIDIA Jetson TX2의 경우, 기본 사용자 이름은 **`nvidia`**이고 기본 암호는 **`nvidia`**입니다.

Windows

컴퓨터에 있는 압축 파일을 Raspberry Pi 코어 디바이스로 전송하려면 WinSCP 또는 PuTTY pscp 명령과 같은 도구를 사용합니다. pscp 명령을 사용하려면 컴퓨터에서 명령 프롬프트 창을 열고 다음을 실행합니다.

```
cd path-to-downloaded-files
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> 이 명령에서 버전 번호는 AWS IoT Greengrass 코어 소프트웨어 패키지의 버전과 일치해야 합니다.

macOS

Mac에 있는 압축 파일을 Raspberry Pi 코어 디바이스로 전송하려면 컴퓨터에서 터미널 창을 열고 다음 명령을 실행합니다. *path-to-downloaded-files*는 일반적으로 `~/Downloads`입니다.

> **Note**
>
> 암호를 두 개 입력하라는 메시지가 나타날 수 있습니다. 이 경우 첫 번째는 Mac의 sudo 명령에 사용할 암호이며 두 번째는 Raspberry Pi에 사용할 암호입니다.

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
```

```
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> 이 명령에서 버전 번호는 AWS IoT Greengrass 코어 소프트웨어 패키지의 버전과 일치해야 합니다.

UNIX-like system

컴퓨터에 있는 압축 파일을 Raspberry Pi 코어 디바이스에 전송하려면 컴퓨터에서 터미널 창을 열고 다음 명령을 실행합니다.

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> 이 명령에서 버전 번호는 AWS IoT Greengrass 코어 소프트웨어 패키지의 버전과 일치해야 합니다.

Raspberry Pi web browser

Raspberry Pi의 웹 브라우저를 사용하여 압축 파일을 다운로드한 경우 해당 파일은 Pi의 `~/Downloads` 폴더(예: `/home/pi/Downloads`)에 있어야 합니다. 그렇지 않다면 해당 압축 파일은 Pi의 ~ 폴더(예: `/home/pi`)에 있어야 합니다.

3. Greengrass 코어 디바이스에서 터미널을 열고 AWS IoT Greengrass 코어 소프트웨어 및 인증서가 들어 있는 폴더로 이동합니다. *path-to-transferred-files*를 코어 디바이스에서 파일을 전송한 경로로 바꾸십시오. 예를 들어, Raspberry Pi에서 `cd /home/pi`를 실행합니다.

   ```
   cd path-to-transferred-files
   ```

4. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어의 압축을 해제합니다. 다음 명령을 실행하여 코어 디바이스로 전송한 소프트웨어 아카이브의 압축을 풉니다. 이 명령은 `-C /` 인수를 사용하여 코어 디바이스의 루트 폴더에 `/greengrass` 폴더를 만듭니다.

   ```
   sudo tar -xzvf greengrass-OS-architecture-1.11.6.tar.gz -C /
   ```

   > **Note**
   >
   > 이 명령에서 버전 번호는 AWS IoT Greengrass 코어 소프트웨어 패키지의 버전과 일치해야 합니다.

5. 인증서와 키를 AWS IoT Greengrass 코어 소프트웨어 폴더로 이동합니다. 다음 명령을 실행하여 인증서용 폴더를 만들고 인증서와 키를 해당 폴더로 이동합니다. *path-to-transferred-files*를 코어 디바이스에서 파일을 전송한 경로로 바꾸고 *CertificateID*를 파일 이름의 인증서 ID로 바꾸십시오. 예를 들어, Raspberry Pi에서는 *path-to-transferred-files*를 **/home/pi**로 바꿉니다.

   ```
   sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/
   certs
   sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs
   sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs
   sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
   ```

6. AWS IoT Greengrass 코어 소프트웨어는 소프트웨어의 매개변수를 지정하는 구성 파일을 사용합니다. 이 구성 파일은 인증서 파일의 파일 경로와 사용할 AWS 클라우드 엔드포인트를 지정합니다. 이 단계에서는 코어의 AWS IoT Greengrass 코어 소프트웨어 구성 파일을 생성합니다. 해결 방법:

   a. 코어의 AWS IoT 사물에 대한 Amazon 리소스 이름(ARN)을 가져옵니다. 해결 방법:

      i. AWS IoT 콘솔의 관리 아래에 있는 Greengrass 디바이스에서 그룹(V1)을 선택합니다.

      ii. Greengrass 그룹 페이지에서 이전에 생성한 그룹을 선택합니다.

      iii. 개요에서 Greengrass 코어를 선택합니다.

      iv. 코어 세부 정보 페이지에서 AWS IoT 사물 ARN을 복사하고 AWS IoT Greengrass 코어 구성 파일에서 사용할 수 있도록 저장합니다.

   b. 현재 리전에서 사용자의 AWS 계정에 대한 AWS IoT 디바이스 데이터 엔드포인트를 가져오십시오. 디바이스는 이 엔드포인트를 사용하여 AWS IoT 사물로서 AWS에 연결합니다. 해결 방법:

i. AWS IoT 콘솔에서 설정을 선택합니다.

ii. 디바이스 데이터 엔드포인트에서 엔드포인트를 복사하고 AWS IoT Greengrass 코어 구성 파일에서 사용할 수 있도록 저장합니다.

c. AWS IoT Greengrass 코어 소프트웨어에 대한 구성 파일을 생성합니다. 예를 들어 다음 명령을 실행하여 GNU nano를 사용해 파일을 생성할 수 있습니다.

```
sudo nano /greengrass/config/config.json
```

다음 JSON 문서로 파일의 내용을 바꿉니다.

```
{
  "coreThing" : {
    "caPath": "AmazonRootCA1.pem",
    "certPath": "certificateId-certificate.pem.crt",
    "keyPath": "certificateId-private.pem.key",
    "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
    "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
  "crypto": {
    "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key",
        "certificatePath": "file:///greengrass/certs/certificateId-
certificate.pem.crt"
      }
    }
  }
```

```
}
```

뒤이어 다음과 같이 합니다.

- Amazon Root CA 1과 다른 Amazon root CA 인증서를 다운로드한 경우, *AmazonRootCA1.pem*의 각 인스턴스를 Amazon root CA 파일의 이름으로 교체합니다.
- *certificateId*의 각 인스턴스를 인증서 및 키 파일 이름의 인증서 ID로 교체합니다.
- *arn:aws:iot:region:account-id:thing/MyGreengrassV1Core*를 이전에 저장한 코어 사물의 ARN으로 바꿉니다.
- *MyGreengrassV1core*를 코어 사물의 이름으로 바꿉니다.
- *device-data-prefix-ats.iot.region.amazonaws.com*을 이전에 저장한 AWS IoT 디바이스 데이터 엔드포인트로 바꾸십시오.
- *##*을 사용자의 AWS 리전으로 바꿉니다.

이 구성 파일에서 사용할 수 있는 구성 옵션에 대한 자세한 내용은 AWS IoT Greengrass 코어 구성 파일를 참조하세요.

7. 코어 디바이스가 인터넷에 연결되어 있는지 확인합니다. 코어 디바이스에서 AWS IoT Greengrass를 시작합니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

`Greengrass successfully started` 메시지가 표시되어야 합니다. PID를 기록해 둡니다.

> **Note**
>
> 시스템 부팅 시 AWS IoT Greengrass을(를) 시작하도록 코어 디바이스를 설정하는 방법은 the section called “시스템 부팅 시 Greengrass 시작” 섹션을 참조하십시오.

다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어(Greengrass 대몬)가 작동 중인지 확인할 수 있습니다. *PID-number*를 사용자의 PID로 대체합니다.

```
ps aux | grep PID-number
```

실행 중인 Greengrass 대몬(daemon)의 경로와 함께 PID에 대한 항목이 표시되어야 합니다(예: `/greengrass/ggc/packages/1.11.6/bin/daemon`). AWS IoT Greengrass 시작에 문제가 발생하면 문제 해결 섹션을 참조하십시오.

모듈 3(1부): AWS IoT Greengrass의 Lambda 함수

이 모듈에서는 AWS IoT Greengrass 코어 디바이스에서 MQTT 메시지를 보내는 Lambda 함수를 만들고 배포하는 방법을 보여줍니다. 이 모듈에서는 Lambda 함수 구성, MQTT 메시징을 허용하는 데 사용되는 구독 및 코어 디바이스에 대한 배포를 설명합니다.

모듈 3(2부)에서는 AWS IoT Greengrass 코어에서 실행되는 온디맨드 Lambda 함수와 수명이 긴 Lambda 함수 간의 차이점을 다룹니다.

시작하기 전에 모듈 1과 모듈 2를 끝냈고 실행 중인 AWS IoT Greengrass 코어 디바이스가 있는지 확인합니다.

> **Tip**
>
> 또는 코어 디바이스를 자동으로 설정하는 스크립트를 사용하려면 the section called “빠른 시작: Greengrass 디바이스 설정” 단원을 참조하십시오. 스크립트는 이 모듈에 사용되는 Lambda 함수를 만들고 배포할 수도 있습니다.

이 모듈을 완료하는 데 약 30분 정도 걸립니다.

주제

- Lambda 함수 생성 및 패키징
- AWS IoT Greengrass에 대한 Lambda 함수 구성
- Greengrass 코어 디바이스에 클라우드 구성 배포
- Lambda 함수가 코어 디바이스에서 실행 중인지 확인

Lambda 함수 생성 및 패키징

이 모듈의 예제 Python Lambda 함수수는 Python용 AWS IoT Greengrass코어 SDK를 사용하여 MQTT 메시지를 게시합니다.

이 단계에서는 다음을 수행합니다.

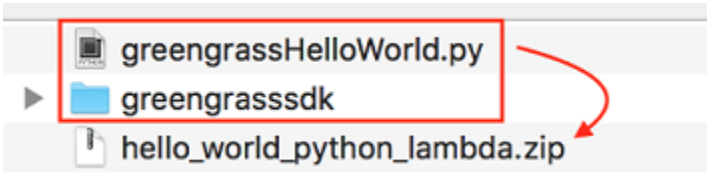
- Python용 AWS IoT Greengrass 코어 SDK를 컴퓨터(AWS IoT Greengrass 코어 디바이스가 아니라)에 다운로드합니다.
- 함수 코드와 종속성이 포함된 Lambda 함수 배포 패키지를 생성합니다.
- Lambda 콘솔을 사용하여 Lambda 함수를 만들고 배포 패키지를 업로드합니다.
- Lambda 함수의 버전을 게시하고 버전을 가리키는 별칭을 만듭니다.

이 모듈을 완료하려면 코어 디바이스에 Python 3.7을 설치해야 합니다.

1. AWS IoT Greengrass 코어 SDK 다운로드 페이지에서 Python용 AWS IoT Greengrass 코어 SDK를 다운로드합니다.
2. 다운로드한 패키지의 압축을 풀어 Lambda 함수 코드 및 SDK를 가져옵니다.

   이 모듈의 Lambda 함수는 다음을 사용합니다.

   - `examples\HelloWorld`의 `greengrassHelloWorld.py` 파일. 이 항목은 Lambda 함수 코드입니다. 이 함수는 5초마다 가능한 메시지 두 개 중 하나를 `hello/world` 주제에 게시합니다.
   - `greengrasssdk` 폴더입니다. 이는 SDK입니다.
3. `greengrasssdk` 폴더를 `greengrassHelloWorld.py`가 포함된 `HelloWorld` 폴더에 복사합니다.
4. Lambda 함수 배포 패키지를 생성하려면 `greengrassHelloWorld.py`와 `greengrasssdk` 폴더를 `hello_world_python_lambda.zip`이라는 압축된 `zip` 파일로 저장합니다. `.py` 파일과 `greengrasssdk` 폴더는 디렉터리의 루트에 있어야 합니다.

   UNIX 유사 시스템(Mac 터미널 포함)의 경우 다음 명령을 사용하여 파일과 폴더를 패키징할 수 있습니다.

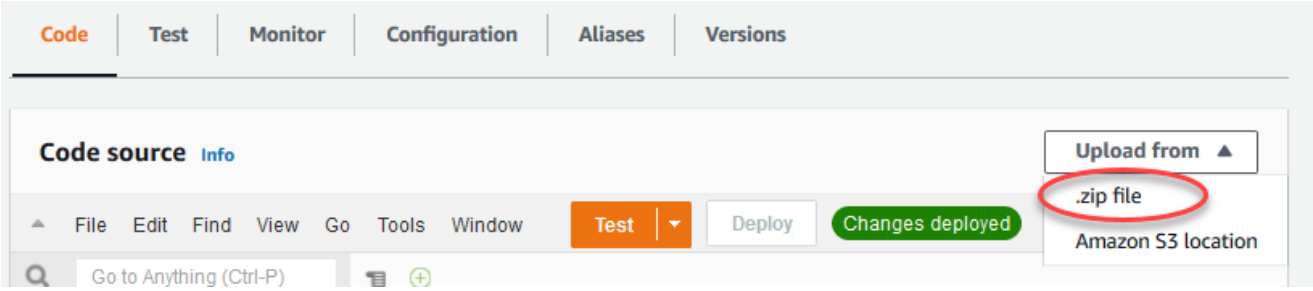
   ```
   zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
   ```

> **Note**
>
> 배포에 따라 먼저 `zip`을 설치해야 할 수도 있습니다(예를 들어 `sudo apt-get install zip`을 실행하여). 설치 명령은 배포에 따라 다를 수 있습니다.

이제 Lambda 함수를 만들고 배포 패키지를 업로드할 준비가 되었습니다.

5. Lambda 콘솔을 열고 함수 생성을 선택합니다.
6. 새로 작성을 선택합니다.
7. 함수 이름을 **`Greengrass_HelloWorld`**로 지정하고 나머지 필드를 다음과 같이 설정합니다.
   - 실행 시간에서 Python 3.7을 선택합니다.
   - 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.

   함수 생성(Create function)을 선택합니다.
8. Lambda 함수 배포 패키지를 업로드합니다.
   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



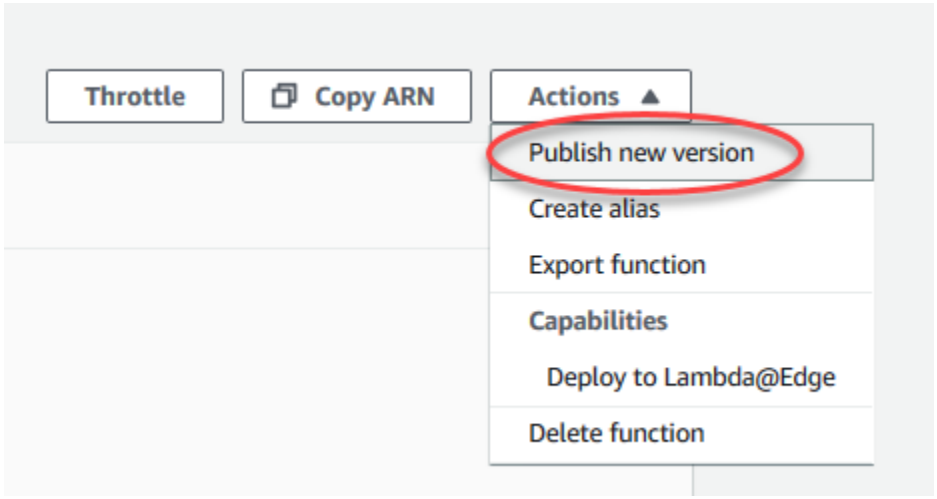
   b. 업로드를 선택한 다음 `hello_world_python_lambda.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.
   c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.
      - 실행 시간에서 Python 3.7을 선택합니다.
      - 핸들러에 **`greengrassHelloWorld.function_handler`**를 입력합니다.

d. Save를 선택합니다.

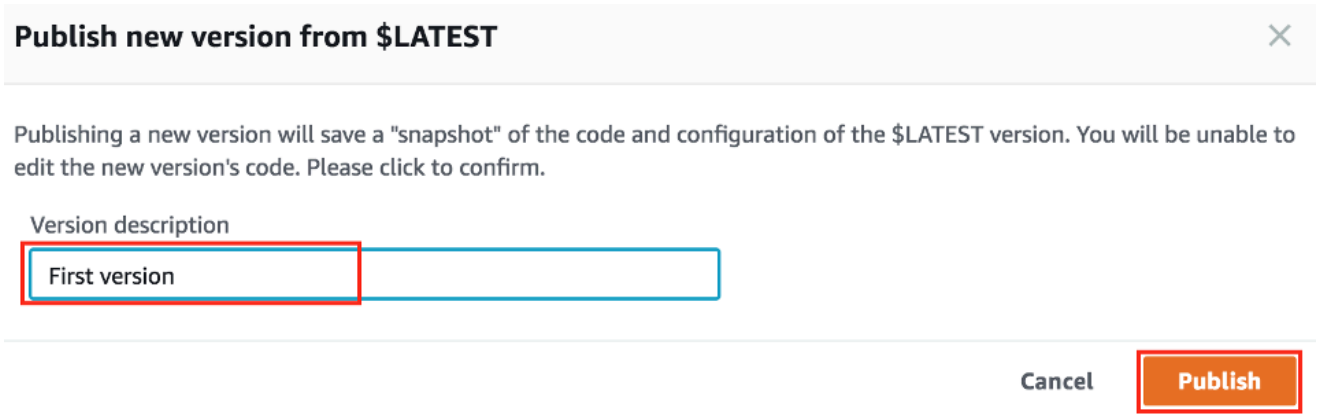
> **Note**
>
> AWS Lambda 콘솔의 테스트 버튼은 이 함수와 함께 작동하지 않습니다. AWS IoT Greengrass 코어 SDK에는 AWS Lambda 콘솔에서 Greengrass Lambda 함수를 독립적으로 실행하는 데 필요한 모듈이 포함되어 있지 않습니다. 이러한 모듈(예: `greengrass_common`)은 Greengrass 코어에 배포된 후 함수에 제공됩니다.

9. Lambda 함수를 게시하십시오.

a. 작업 메뉴에서 새 버전 게시를 선택합니다.



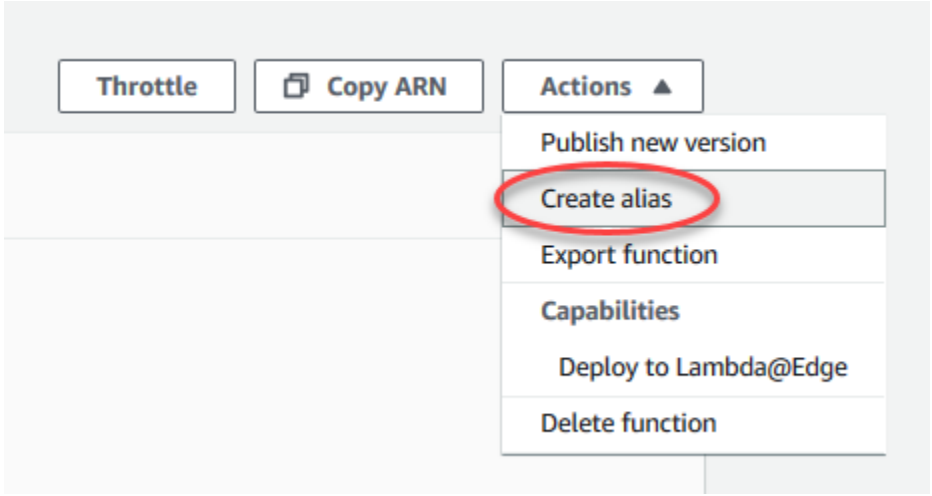
b. 버전 설명에 **`First version`**을 입력한 후 게시를 선택합니다.



10. Lambda 함수 버전의 별칭을 생성합니다.

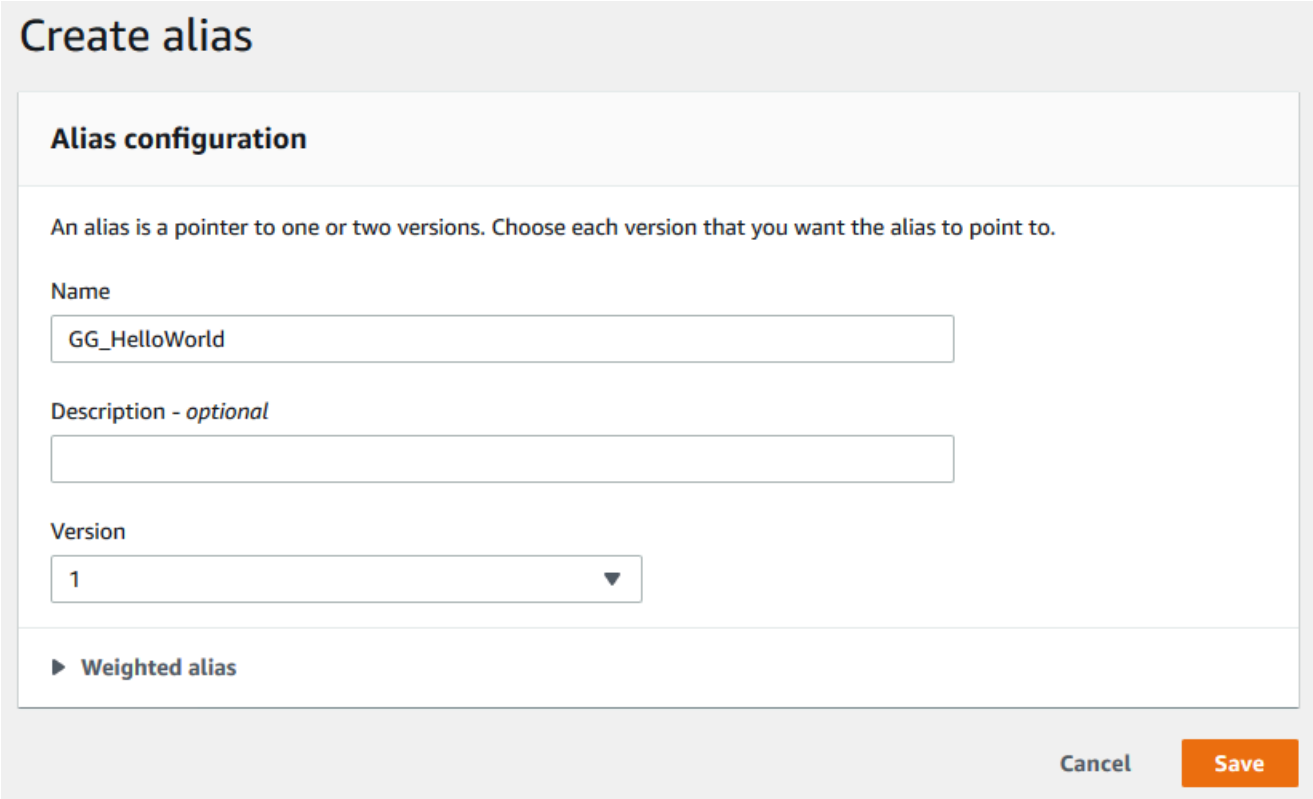
> **Note**
>
> Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

a. 페이지 상단의 작업 메뉴에서 별칭 생성을 선택합니다.



b. 별칭을 **GG_HelloWorld**(으)로 지정하고 버전을 **1**(사용자가 방금 게시한 버전에 해당)로 설정한 다음 저장을 선택합니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.



AWS IoT Greengrass에 대한 Lambda 함수 구성

이제 AWS IoT Greengrass에 대해 Lambda 함수를 구성할 준비가 되었습니다.

이 단계에서는 다음을 수행합니다.

- 먼저 AWS IoT 콘솔을 사용하여 Greengrass 그룹에 Lambda 함수를 추가합니다.
- Lambda 함수에 대한 그룹별 설정을 구성합니다.
- Lambda 함수가 MQTT 메시지를 AWS IoT에 게시할 수 있도록 그룹에 구독을 추가합니다.
- 그룹에 대한 로컬 로그 설정을 구성합니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. Greengrass 그룹에서 모듈 2에서 생성한 그룹을 선택합니다.
3. 그룹 구성 페이지에서 Lambda 함수 탭을 선택한 다음 내 Lambda 함수 섹션으로 스크롤하여 Lambda 함수 추가를 선택합니다.
4. 이전 단계에서 생성한 Lambda 함수의 이름을 선택합니다. (별칭 이름이 아닌 Greengrass_HelloWorld)
5. 버전에는 Alias: GG_HelloWorld를 선택합니다.
6. Lambda 함수 구성 섹션에서 다음과 같이 변경합니다.
   - 시스템 사용자 및 그룹을 그룹 기본값 사용으로 설정합니다.
   - 그룹 기본값 사용에 Lambda 함수 컨테이너화를 설정합니다.
   - 제한 시간을 25초로 설정합니다. 이 Lambda 함수는 매 간접 호출 전에 20초 동안 대기 상태를 유지합니다.
   - 고정된 경우 True를 선택합니다.

   > **Note**
   >
   > 수명이 긴(또는 고정) Lambda 함수는 AWS IoT Greengrass가 시작된 후 자동으로 시작되며, 자체 컨테이너에서 계속 실행됩니다. 이는 간접 호출되면 시작되고 실행할 작업이 남아 있지 않으면 중지되는 온디맨드 Lambda 함수와 상반됩니다. 자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.

7. Lambda 함수 추가를 선택하여 변경 사항을 저장합니다. Lambda 함수 속성에 대한 자세한 내용은 the section called “Greengrass Lambda 함수 실행 제어”을 참조하십시오.

   그런 다음 Lambda 함수가 MQTT 메시지를 AWS IoT Core로 전송하는 데 사용하는 구독을 생성합니다.

   Greengrass Lambda 함수는 MQTT 메시지를 다음과 교환할 수 있습니다.

   - Greengrass 그룹에서의 디바이스
   - 그룹 내의 커넥터.

- 그룹의 기타 Lambda 함수입니다.
- AWS IoT Core.
- 로컬 섀도우 서비스 자세한 내용은 the section called “모듈 5: 디바이스 섀도우와 상호 작용” 섹션을 참조하세요.

그룹은 구독을 사용하여 이러한 엔터티가 서로 통신하는 방법을 제어합니다. 구독은 예측 가능한 상호 작용과 보안 계층을 제공합니다.

구독은 소스, 대상 및 주제로 구성됩니다. 소스는 메시지의 전송 위치입니다. 대상은 메시지의 목적지입니다. 해당 주제를 사용하면 소스에서 대상으로 전송되는 데이터를 필터링할 수 있습니다. 소스 또는 대상은 Greengrass 디바이스, Lambda 함수, 커넥터, 디바이스 섀도우 또는 AWS IoT Core일 수 있습니다.

> **Note**
>
> 메시지가 특정 방향으로 흐른다는 점에서 구독에는 방향(소스에서 대상으로)이 있습니다. 양방향 통신이 가능하려면 2개의 구독을 설정해야 합니다.

> **Note**
>
> 현재 구독 주제 필터는 주제에 하나 이상의 + 글자를 포함할 수 없습니다. 주제 필터는 주제의 끝에 하나의 # 문자만 허용합니다.

`Greengrass_HelloWorld` Lambda 함수는 AWS IoT Core의 `hello/world`주제에만 메시지를 전송하므로 Lambda 함수에서 AWS IoT Core로의 구독을 하나만 생성해야 합니다. 이 작업은 다음 단계에서 생성합니다.

8. 그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.

   AWS CLI을(를) 사용하여 구독을 생성하는 방법을 보여주는 예제는 AWS CLI 명령 참조의 create-subscription-definition을 참조하십시오.

9. 소스 유형에서 Lambda 함수를 선택하고, 소스에서는 GreenGrass_HelloWorld를 선택합니다.
10. 대상 유형으로는 서비스를 선택하고, 대상에는 IoT Cloud를 선택합니다.
11. 주제 필터에 **`hello/world`**을(를) 입력한 다음 구독 생성을 선택합니다.

12. 그룹의 로깅 설정을 구성합니다. 이 자습서에서는 로그를 코어 디바이스의 파일 시스템에 쓸 수 있도록 AWS IoT Greengrass 시스템 구성 요소와 사용자 정의 Lambda 함수를 구성합니다.

    a. 그룹 구성 페이지에서 로그을 선택합니다.

    b. 로컬 로그 구성 섹션에서 편집을 선택합니다.

    c. 로컬 로그 구성 편집 대화 상자에서 로그 수준 및 저장소 크기 모두에 대한 기본값을 유지한 다음 저장을 선택합니다.

    로그를 사용하여 이 자습서를 실행할 때 발생할 수 있는 문제를 해결할 수 있습니다. 문제를 해결할 때 일시적으로 로깅 수준을 디버그로 변경할 수 있습니다. 자세한 내용은 the section called “파일 시스템 로그 액세스” 섹션을 참조하세요.

13. Java 8 런타임이 코어 디바이스에 설치되지 않은 경우 Java 8 런타임을 설치하거나 스트림 관리자를 비활성화해야 합니다.

    > **Note**
    >
    > 이 자습서에서는 스트림 관리자를 사용하지 않지만 기본적으로 스트림 관리자를 활성화하는 기본 그룹 생성 워크플로를 사용합니다. 스트림 관리자를 활성화했지만 Java 8이 설치되지 않은 경우 그룹 배포가 실패합니다. 자세한 내용은 스트림 관리자 요구 사항을 참조하십시오.

    스트림 관리자를 비활성화하려면 다음을 수행합니다.

    a. 그룹 설정 페이지에서 Lambda 함수 탭을 선택합니다.

    b. 시스템 Lambda 함수 섹션에서 스트림 관리자를 선택하고 편집을 선택합니다.

    c. 비활성화를 선택한 다음 저장을 선택합니다.

Greengrass 코어 디바이스에 클라우드 구성 배포

1. Greengrass 코어 디바이스가 인터넷에 연결되어 있는지 확인합니다. 예를 들어, 웹 페이지를 이동해 봅니다.
2. 코어 디바이스에서 Greengrass 대몬(daemon)이 실행 중인지 확인합니다. 코어 디바이스 터미널에서 다음 명령들을 실행하여 대몬(daemon)이 실행 중인지 확인하고 필요한 경우 시작하십시오.

    a. 대몬(daemon)이 실행 중인지 확인하려면:

```
ps aux | grep -E 'greengrass.*daemon'
```

출력에 root에 대한 /greengrass/ggc/packages/1.11.6/bin/daemon 입력이 포함되어 있는 경우에는 대몬(daemon)이 실행 중인 것입니다.

b. 대몬(daemon)을 시작하려면:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

이제 Greengrass 코어 디바이스에 Lambda 함수 및 구독 구성을 배포할 준비가 완료되었습니다.

3. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
4. Greengrass 그룹에서 모듈 2에서 생성한 그룹을 선택합니다.
5. 그룹 구성 페이지에서 배포를 선택합니다.
6. Lambda 함수 탭의 시스템 Lambda 함수 섹션에서 IP 감지기를 선택합니다.
7. 편집을 선택하고 MQTT 브로커 엔드포인트 자동 감지 및 재정의를 선택합니다. 이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

최초 배포는 몇 분 정도 걸릴 수 있습니다. 배포가 완료되면 배포 페이지의 상태 열에 성공적으로 완료됨이 표시되어야 합니다.

> **Note**
>
> 또한 배포 상태가 페이지 헤더에서 그룹의 이름 아래에 표시됩니다.

문제 해결에 대한 도움말은 문제 해결 섹션을 참조하십시오.

Lambda 함수가 코어 디바이스에서 실행 중인지 확인

1. AWS IoT 콘솔을 열고 탐색 패널에서 테스트를 선택하고 MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독 탭을 선택합니다.
3. **hello/world**주제 필터에 을(를) 입력하고 추가 구성을 확장합니다.

4. 다음 각 필드에 나열된 정보를 입력합니다.

   - 서비스 품질에서 0을 선택합니다.
   - MQTT 페이로드 디스플레이에서 페이로드를 문자열로 표시를 선택합니다.

5. 구독을 선택합니다.

디바이스에서 Lambda 함수가 실행 중이라고 가정하고 다음과 같이 `hello/world` 주제에 메시지가 게시됩니다.

Lambda 함수가 `hello/world` 주제로 계속 MQTT 메시지를 전송하지만 AWS IoT Greengrass 대몬을 중지하지 마십시오. 나머지 모듈은 데몬이 실행 중이라는 가정하에 작성되었습니다.

그룹에서 함수 및 구독을 삭제할 수 있습니다.

- 그룹 구성 페이지의 Lambda 함수 탭에서 제거하려는 Lambda 함수를 선택하고 제거를 선택합니다.
- 그룹 구성 페이지에서 구독 탭을 선택한 다음 삭제를 선택합니다.

다음 그룹 배포 중에 코어에서 함수 및 구독이 제거됩니다.

모듈 3(2부): AWS IoT Greengrass의 Lambda 함수

이 모듈에서는 AWS IoT Greengrass 코어에서 실행되는 온디맨드 Lambda 함수와 수명이 긴 Lambda 함수 간의 차이점을 살펴봅니다.

시작하기 전에 Greengrass 디바이스 설정 스크립트를 실행하거나 모듈 1, 모듈 2 및 모듈 3(1부)을 완료했는지 확인합니다.

이 모듈을 완료하는 데 약 30분 정도 걸립니다.

주제

- Lambda 함수 생성 및 패키징
- 수명이 긴 AWS IoT Greengrass용 Lambda 함수 구성
- 수명이 긴 Lambda 함수 테스트
- 온디맨드 Lambda 함수 테스트

Lambda 함수 생성 및 패키징

이 단계에서는 다음을 수행합니다.

- 함수 코드와 종속성이 포함된 Lambda 함수 배포 패키지를 생성합니다.
- Lambda 콘솔을 사용하여 Lambda 함수를 만들고 배포 패키지를 업로드합니다.
- Lambda 함수의 버전을 게시하고 버전을 가리키는 별칭을 만듭니다.

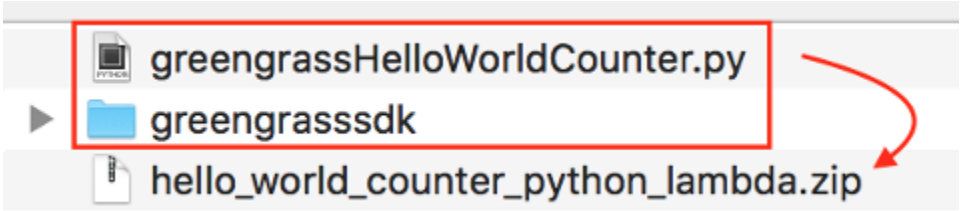
1. 컴퓨터에서, 모듈 3-1의 the section called “Lambda 함수 생성 및 패키징”에서 다운로드 및 압축 해제한 AWS IoT Greengrass Python용 코어 SDK로 이동하십시오.

   이 모듈의 Lambda 함수는 다음을 사용합니다.

   - `examples\HelloWorldCounter`의 `greengrassHelloWorldCounter.py` 파일. 이 항목은 Lambda 함수 코드입니다.
   - `greengrasssdk` 폴더입니다. 이는 SDK입니다.
2. Lambda 함수 배포 패키지 생성:

   a. `greengrasssdk` 폴더를 `greengrassHelloWorldCounter.py`가 포함된 `HelloWorldCounter` 폴더에 복사합니다.

   b. `greengrassHelloWorldCounter.py`와 `greengrasssdk` 폴더를 `hello_world_counter_python_lambda.zip`이라는 `zip` 파일에 저장합니다. `py` 파일과 `greengrasssdk` 폴더는 디렉터리의 루트에 있어야 합니다.



`zip`이 설치된 UNIX 유사 시스템(Mac 터미널 포함)의 경우 다음 명령을 사용하여 파일과 폴더를 패키징할 수 있습니다.

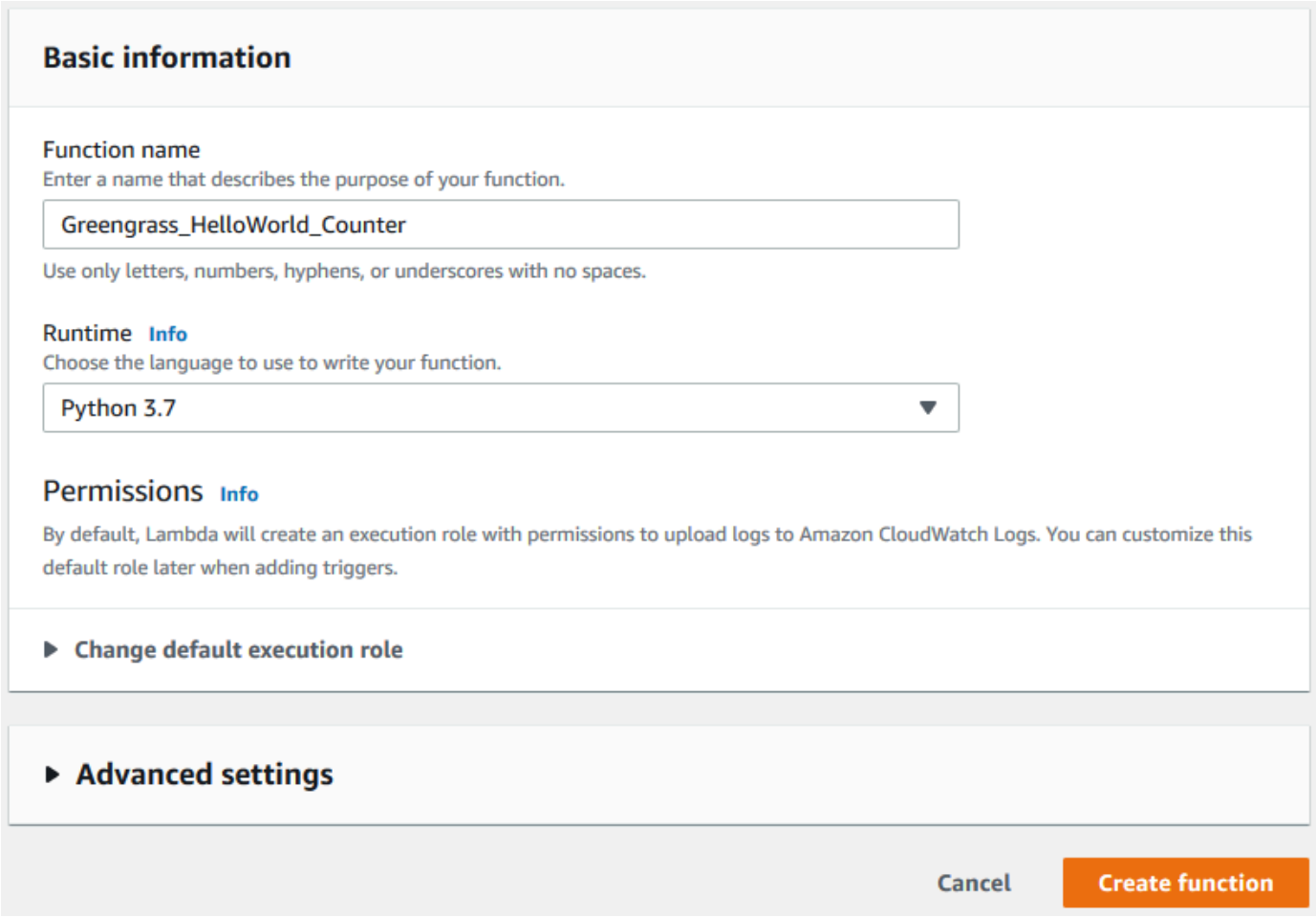
```
zip -r hello_world_counter_python_lambda.zip greengrasssdk
 greengrassHelloWorldCounter.py
```

이제 Lambda 함수를 만들고 배포 패키지를 업로드할 준비가 되었습니다.

3. Lambda 콘솔을 열고 함수 생성을 선택합니다.
4. 새로 작성을 선택합니다.
5. 함수 이름을 **`Greengrass_HelloWorld_Counter`**로 지정하고 나머지 필드를 다음과 같이 설정합니다.

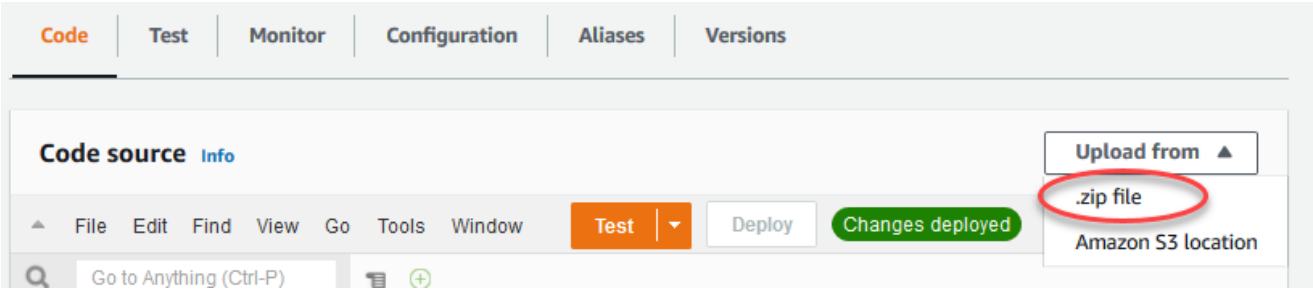
   - 실행 시간에서 Python 3.7을 선택합니다.
   - 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다. 또는 모듈 3-1에서 생성한 역할을 다시 사용할 수 있습니다.

   함수 생성을 선택합니다.



6. Lambda 함수 배포 패키지를 업로드합니다.

   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



   b. 업로드를 선택한 다음 `hello_world_counter_python_lambda.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.

   c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

      - 실행 시간에서 Python 3.7을 선택합니다.

- 핸들러에 **greengrassHelloWorldCounter.function_handler**를 입력합니다.

d. Save를 선택합니다.

> **Note**
>
> AWS Lambda 콘솔의 테스트 버튼은 이 함수와 함께 작동하지 않습니다. AWS IoT Greengrass코어 SDK에는 AWS Lambda 콘솔에서 Greengrass Lambda 함수를 독립적으로 실행하는 데 필요한 모듈이 포함되어 있지 않습니다. 이러한 모듈(예: `greengrass_common`)은 Greengrass 코어에 배포된 후 함수에 제공됩니다.

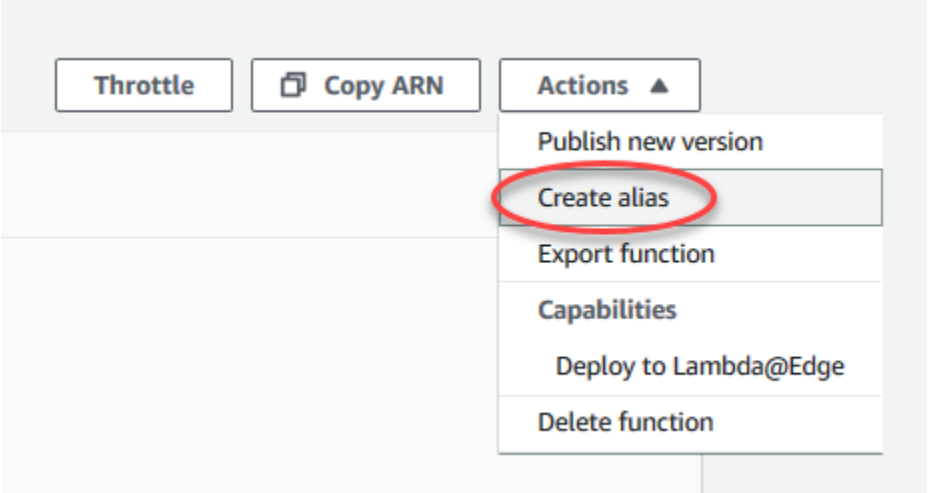
7. 함수의 첫 번째 버전을 게시합니다.

a. 작업 메뉴에서 새 버전 게시를 선택합니다. 버전 설명에 **First version**을 입력합니다.

b. [Publish]를 선택합니다.

8. 함수 버전의 별칭을 생성합니다.

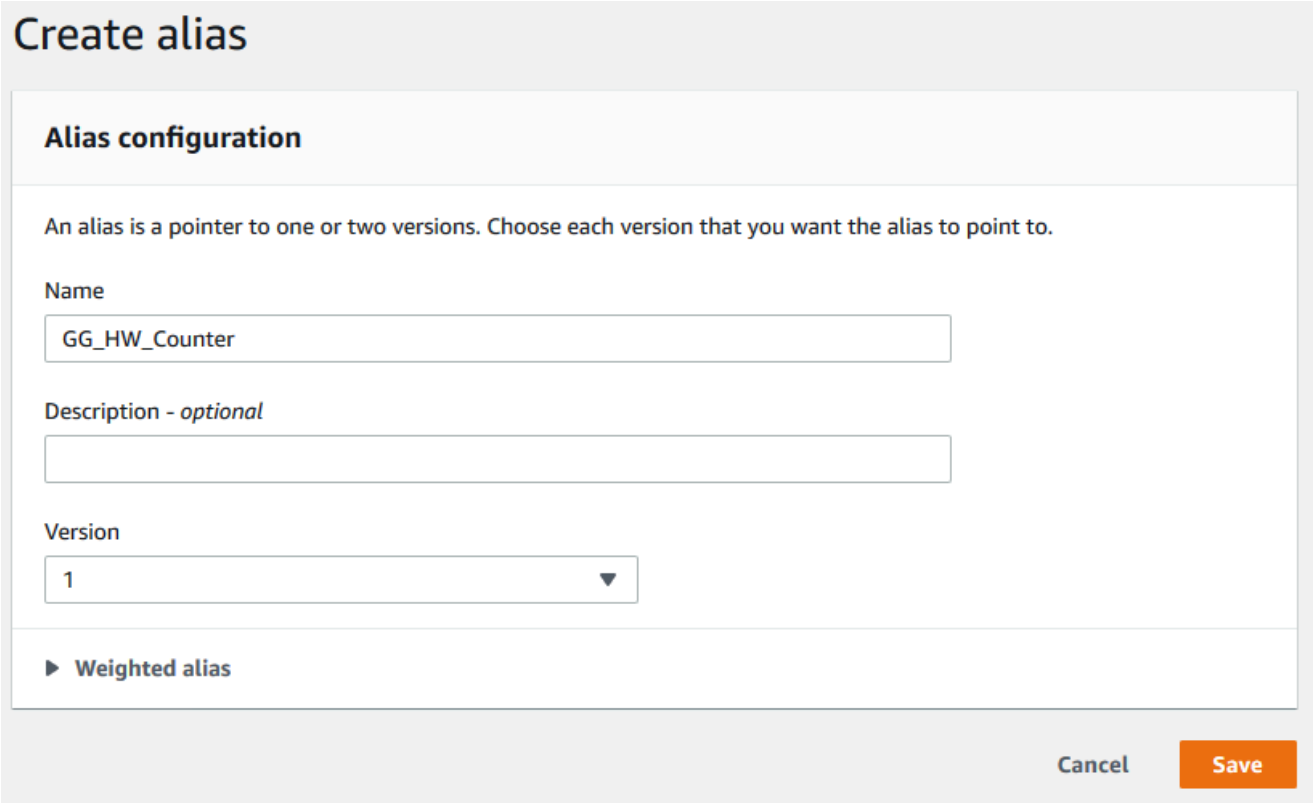
a. 페이지 상단의 작업 메뉴에서 별칭 생성을 선택합니다.



b. 이름(Name)에 **GG_HW_Counter**을 입력합니다.

c. 버전에서 1을 선택합니다.

d. Save를 선택합니다.



별칭은 Greengrass 디바이스가 구독할 수 있는 Lambda 함수에 대해 단일 엔터티를 생성합니다. 이렇게 하면 함수가 수정될 때마다 새 Lambda 함수 버전 번호로 구독을 업데이트할 필요가 없습니다.

수명이 긴 AWS IoT Greengrass용 Lambda 함수 구성

이제 AWS IoT Greengrass에 대해 Lambda 함수를 구성할 준비가 되었습니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. Greengrass 그룹에서 모듈 2에서 생성한 그룹을 선택합니다.
3. 그룹 구성 페이지에서 Lambda 함수수 탭을 선택한 다음 내 Lambda 함수수에서 추가를 선택합니다.
4. Lambda 함수의 경우 GreenGrass_HelloWorld_Counter를 선택하십시오.
5. Lambda 함수 버전에서 게시된 버전에 대한 별칭을 선택합니다.
6. 제한 시간(초)에는 **25**를 입력합니다. 이 Lambda 함수는 매 간접 호출 전에 20초 동안 대기 상태를 유지합니다.
7. 고정된 경우 True를 선택합니다.

8. 다른 모든 속성에 대해 기본 값을 유지하고 Lambda 함수 추가를 선택합니다.

수명이 긴 Lambda 함수 테스트

수명이 긴 Lambda 함수는 AWS IoT Greengrass 코어가 시작할 때 자동으로 시작됩니다(그리고 단일 컨테이너 또는 샌드박스에서 실행됩니다). 함수 핸들러 외부에서 정의된 변수 및 사전 처리 로직은 함수 핸들러를 호출할 때마다 유지됩니다. 함수 핸들러를 여러 번 호출하면 앞선 호출의 실행을 마칠 때까지 대기됩니다.

이 모듈에 사용되는 `greengrassHelloWorldCounter.py` 코드는 함수 핸들러 외부의 `my_counter` 변수를 정의합니다.

> **Note**
>
> AWS Lambda 콘솔이나 GitHub의 AWS IoT Greengrass Core SDK for Python에서 코드를 볼 수 있습니다.

이 단계에서는 Lambda 함수 및 AWS IoT를 사용하여 MQTT 메시지를 교환할 수 있는 구독을 생성합니다. 그런 다음 그룹을 배포하고 함수를 테스트합니다.

1. 그룹 구성 페이지에서 구독을 선택한 다음 추가를 선택합니다.
2. 소스 유형에서 Lambda 함수를 선택한 다음 Greengrass_HelloWorld_Counter를 선택합니다.
3. 대상 유형에서 서비스를 선택하고 IoT Cloud를 선택합니다.
4. 주제 필터에 **`hello/world/counter`**을 입력합니다.
5. 구독 생성을 선택합니다.

   이 단일 구독은 `Greengrass_HelloWorld_Counter` Lambda 함수에서 AWS IoT로 한 방향으로만 진행됩니다. 클라우드에서 이 Lambda 함수를 간접 호출하거나 트리거하려면 반대 방향의 구독을 생성해야 합니다.
6. 1 - 5단계를 수행하여 다음 값을 사용하는 다른 구독을 추가합니다. 이 구독은 Lambda 함수가 AWS IoT에서 메시지를 받을 수 있습니다. 함수를 간접 호출하는 AWS IoT 콘솔에서 메시지를 보낼 때 이 구독을 사용합니다.

   - 소스의 경우 서비스를 선택한 다음 IoT Cloud를 선택합니다.
   - 대상의 경우 Lambdas 함수를 선택한 다음 Greengrass_HelloWorld_Counter를 선택합니다.

- 주제 필터의 경우 **hello/world/counter/trigger**를 입력합니다.

  이 주제 필터에서 /trigger 확장이 사용되는 것은 2개의 구독을 생성했고 이 두 구독이 서로 간섭하지 않아야 하기 때문입니다.

7. 코어 디바이스로 클라우드 구성 배포 섹션에 설명된 대로 Greengrass 대몬(daemon)이 실행 중인지 확인합니다.
8. 그룹 구성 페이지에서 배포를 선택합니다.
9. 배포가 완료된 후 AWS IoT 콘솔 홈페이지로 돌아가 테스트를 선택합니다.
10. 다음 필드를 구성합니다.

    - 구독 주제에 **hello/world/counter**를 입력합니다.
    - 서비스 품질에서 0을 선택합니다.
    - MQTT 페이로드 디스플레이에서 페이로드를 문자열로 표시를 선택합니다.

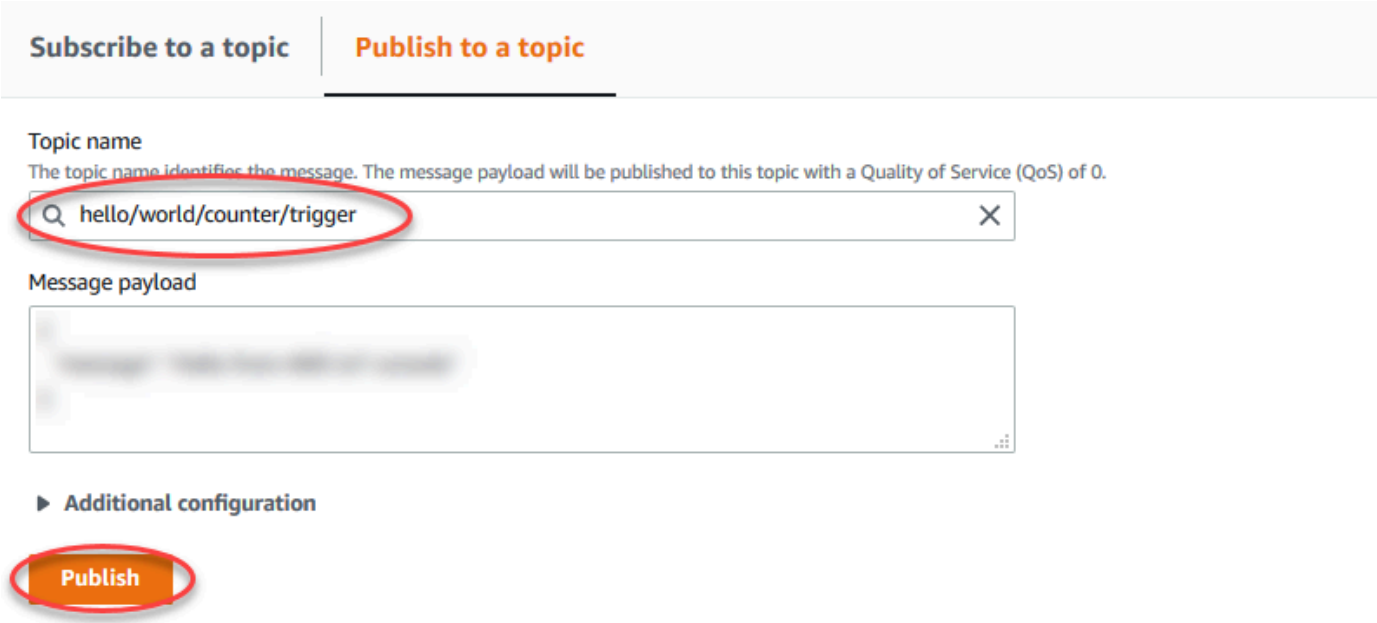
11. 구독을 선택합니다.

    이 모듈의 1부와 달리 hello/world/counter 구독 후에는 메시지가 표시되지 않습니다. 이는 hello/world/counter 주제에 게시하는 greengrassHelloWorldCounter.py 코드가 함수 핸들러 내에 있고 함수 핸들러는 함수가 간접 호출될 때에만 실행되기 때문입니다.

    이 모듈에서는 hello/world/counter/trigger 주제에 대한 MQTT 메시지를 받을 때 간접 호출되도록 Greengrass_HelloWorld_Counter Lambda 함수를 구성했습니다.

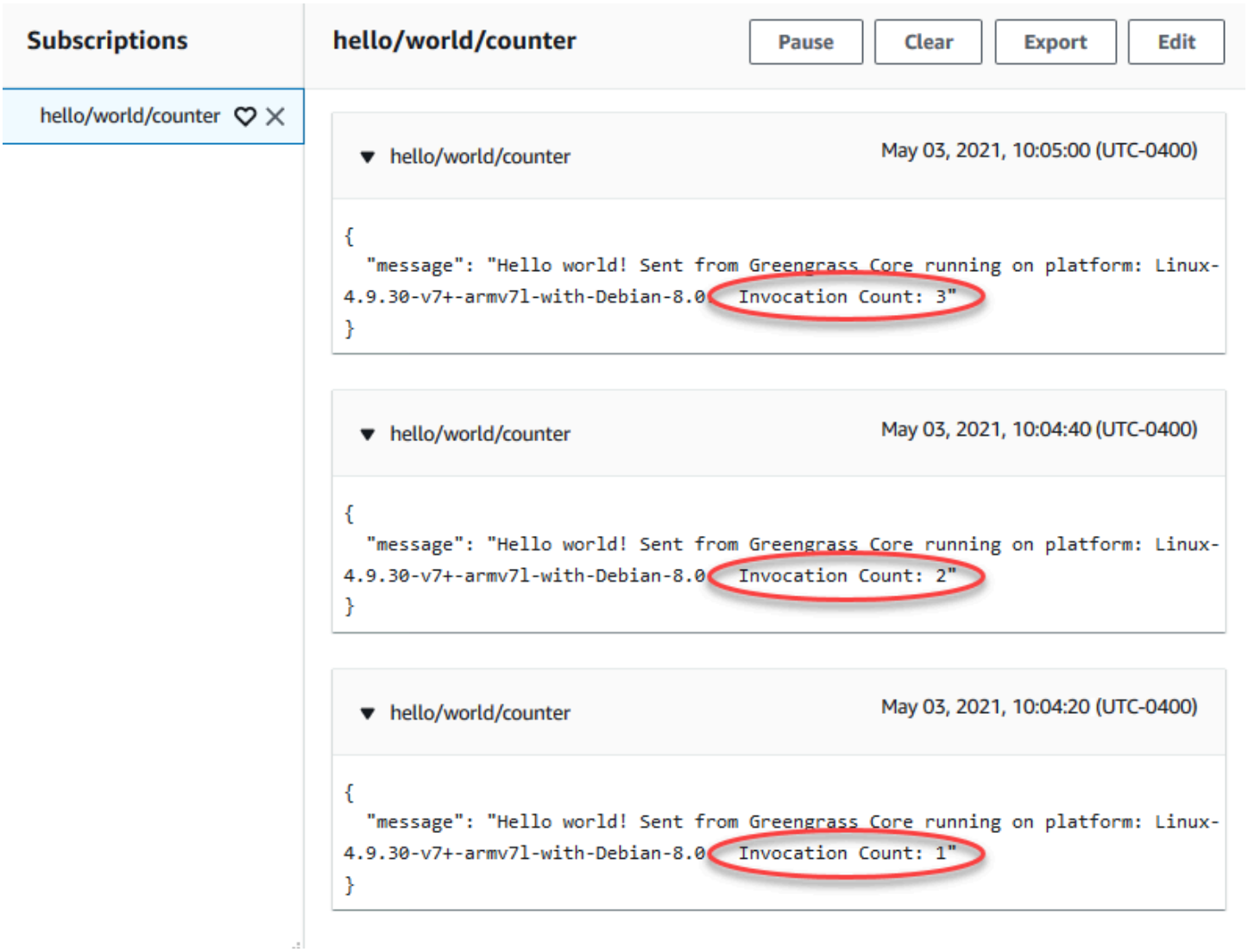
    Greengrass_HelloWorld_Counter에서 IoT Cloud로의 구독을 통해 함수는 hello/world/counter 주제의 AWS IoT에 메시지를 전송할 수 있습니다. IoT Cloud에서 Greengrass_HelloWorld_Counter로의 구독을 통해 AWS IoT는 hello/world/counter/trigger 주제의 함수에 메시지를 전송할 수 있습니다.

12. 긴 수명 주기를 테스트하려면 hello/world/counter/trigger 주제에 메시지를 게시하여 Lambda 함수를 간접 호출합니다. 기본 메시지를 사용할 수 있습니다.



> **Note**
>
> 이 `Greengrass_HelloWorld_Counter` 함수는 수신된 메시지의 내용을 무시합니다. 다만 `function_handler`에서 코드를 실행하며 코드는 `hello/world/counter` 주제에 메시지를 전송합니다. 이 코드는 GitHub의 AWS IoT Greengrass Core SDK for Python에서 검토할 수 있습니다.

메시지가 `hello/world/counter/trigger` 주제에 게시될 때마다 `my_counter` 변수가 증가합니다. 이 간접 호출 횟수는 Lambda 함수에서 전송된 메시지에 표시됩니다. 함수 핸들러에는 20초의 대기 주기(`time.sleep(20)`)가 포함되어 있기 때문에 핸들러는 AWS IoT Greengrass 코어의 응답을 반복해서 대기열에 넣습니다.

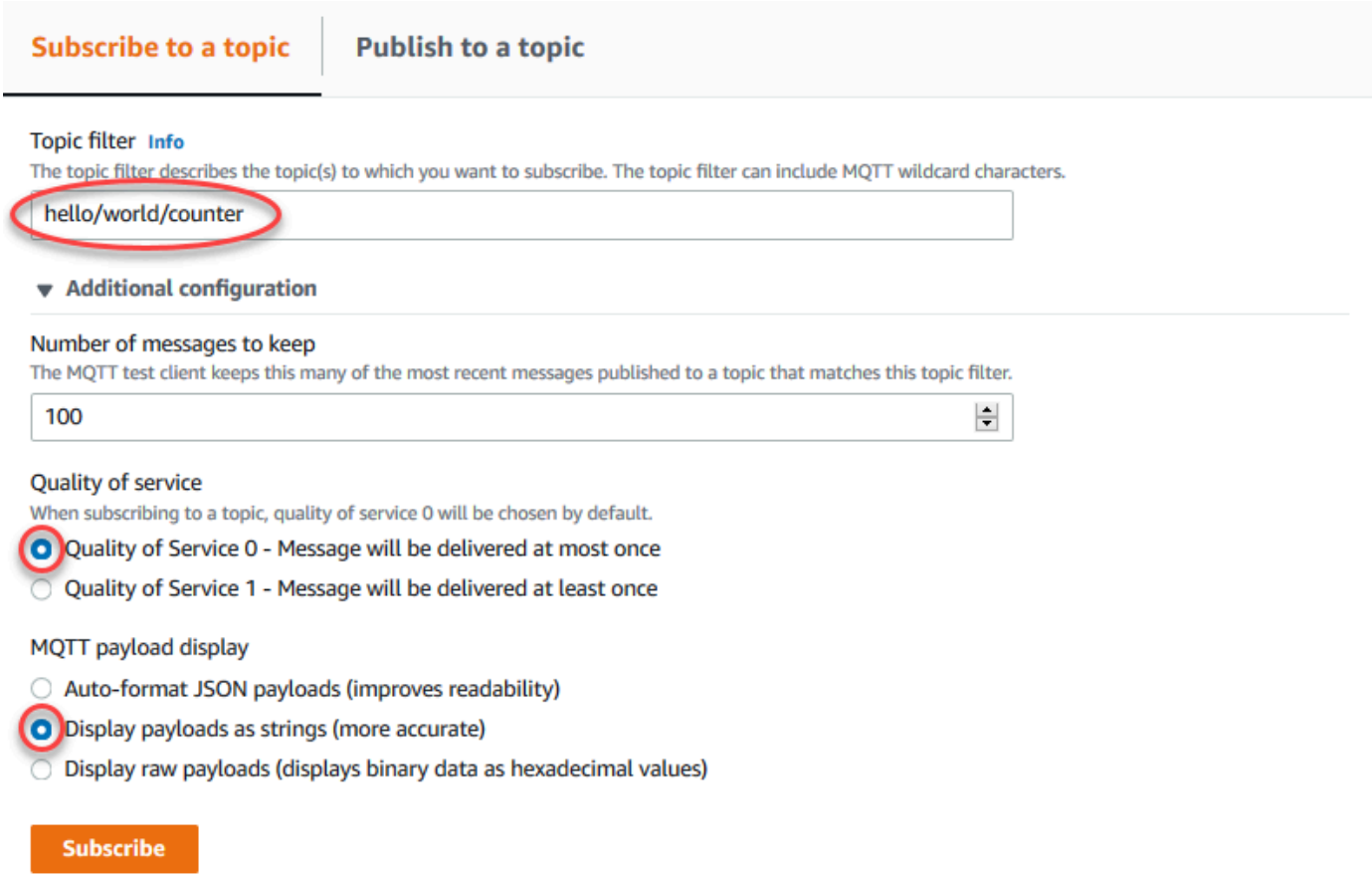


온디맨드 Lambda 함수 테스트

온디맨드 Lambda 함수는 클라우드 기반 AWS Lambda 함수의 기능과 유사합니다. 온디맨드 Lambda 함수에 대한 다중 호출은 병렬로 실행될 수 있습니다. Lambda 함수를 호출하면 호출을 처리하기 위해 별도의 컨테이너가 생성되거나, 리소스가 허용하는 경우 기존 컨테이너가 재사용됩니다. 함수 핸들러 외부에서 정의된 모든 변수 또는 사전 처리는 컨테이너 생성 시 유지되지 않습니다.

1. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
2. 내 Lambda 함수에서 Greengrass_HelloWorld_Counter Lambda 함수를 선택합니다.
3. Greengrass_HelloWorld_Counter 세부 정보 페이지에서 편집을 선택합니다.
4. 고정된 경우, 거짓을 선택한 다음 저장을 선택합니다.
5. 그룹 구성 페이지에서 배포를 선택합니다.

6. 배포가 완료된 후 AWS IoT 콘솔 홈페이지로 돌아가 테스트를 선택합니다.
7. 다음 필드를 구성합니다.
   - 구독 주제에 **hello/world/counter**를 입력합니다.
   - 서비스 품질에서 0을 선택합니다.
   - MQTT 페이로드 디스플레이에서 페이로드를 문자열로 표시를 선택합니다.

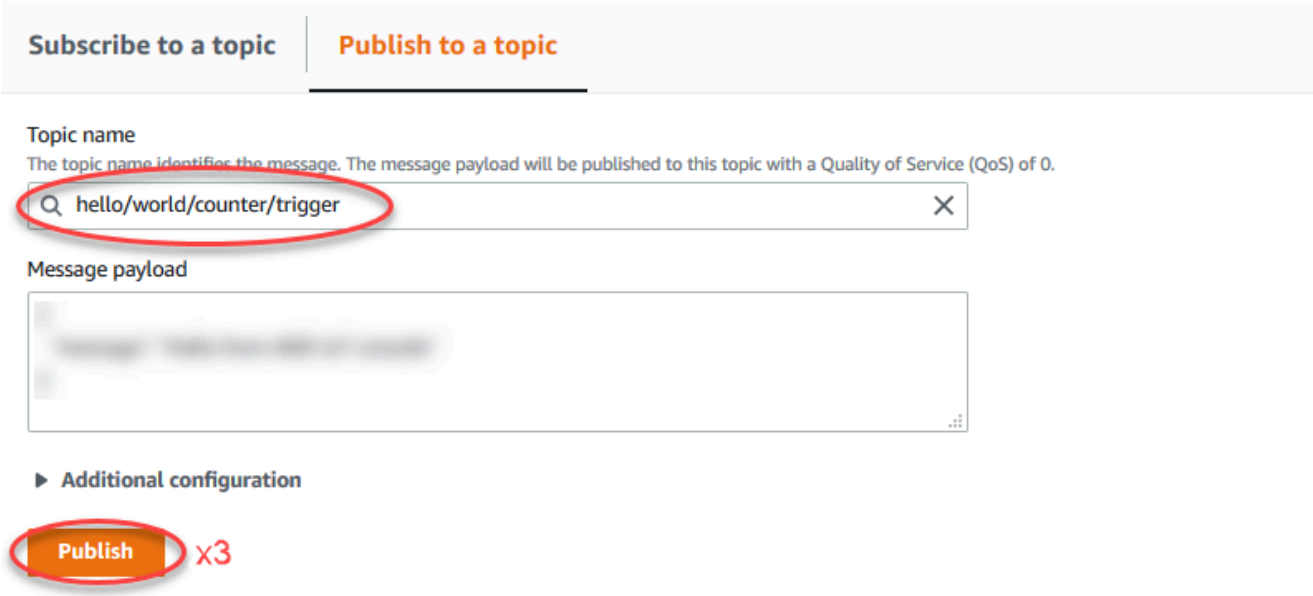


8. 구독을 선택합니다.

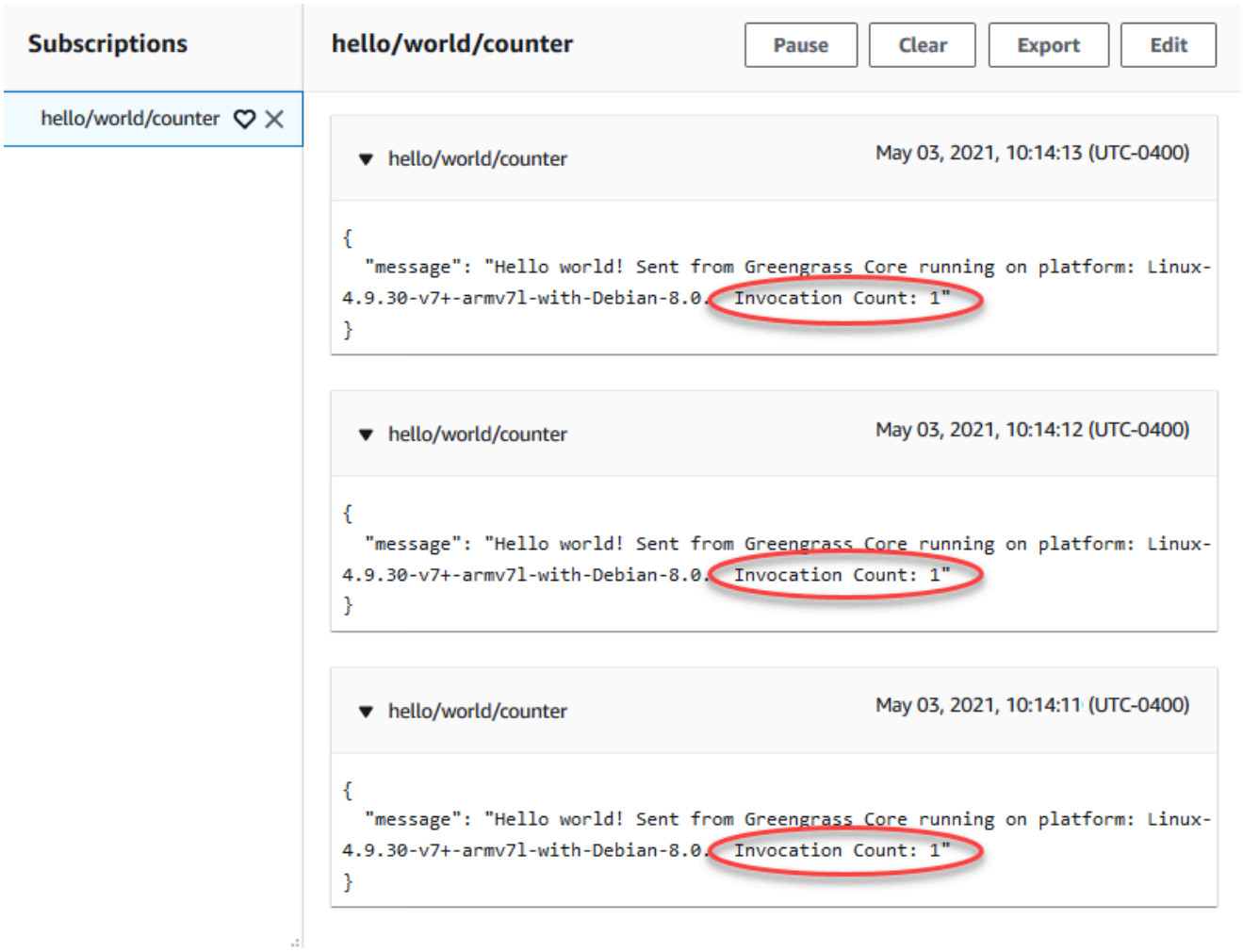
> **Note**
>
> 구독 후에는 어떤 메시지도 표시되지 않습니다.

9. 온디맨드 수명 주기를 테스트하려면 `hello/world/counter/trigger` 주제에 메시지를 게시하여 함수를 간접 호출합니다. 기본 메시지를 사용할 수 있습니다.

   a. 게시 버튼을 한 번 누른 후 5초 이내로 빠르게 3회 선택합니다.



각 게시가 함수 핸들러를 간접 호출하고 각 호출의 컨테이너를 생성합니다. 각각의 온디맨드 Lambda 함수에는 자체 컨테이너/샌드박스가 있기 때문에 3회의 함수 트리거에 호출 횟수가 증가하지는 않습니다.

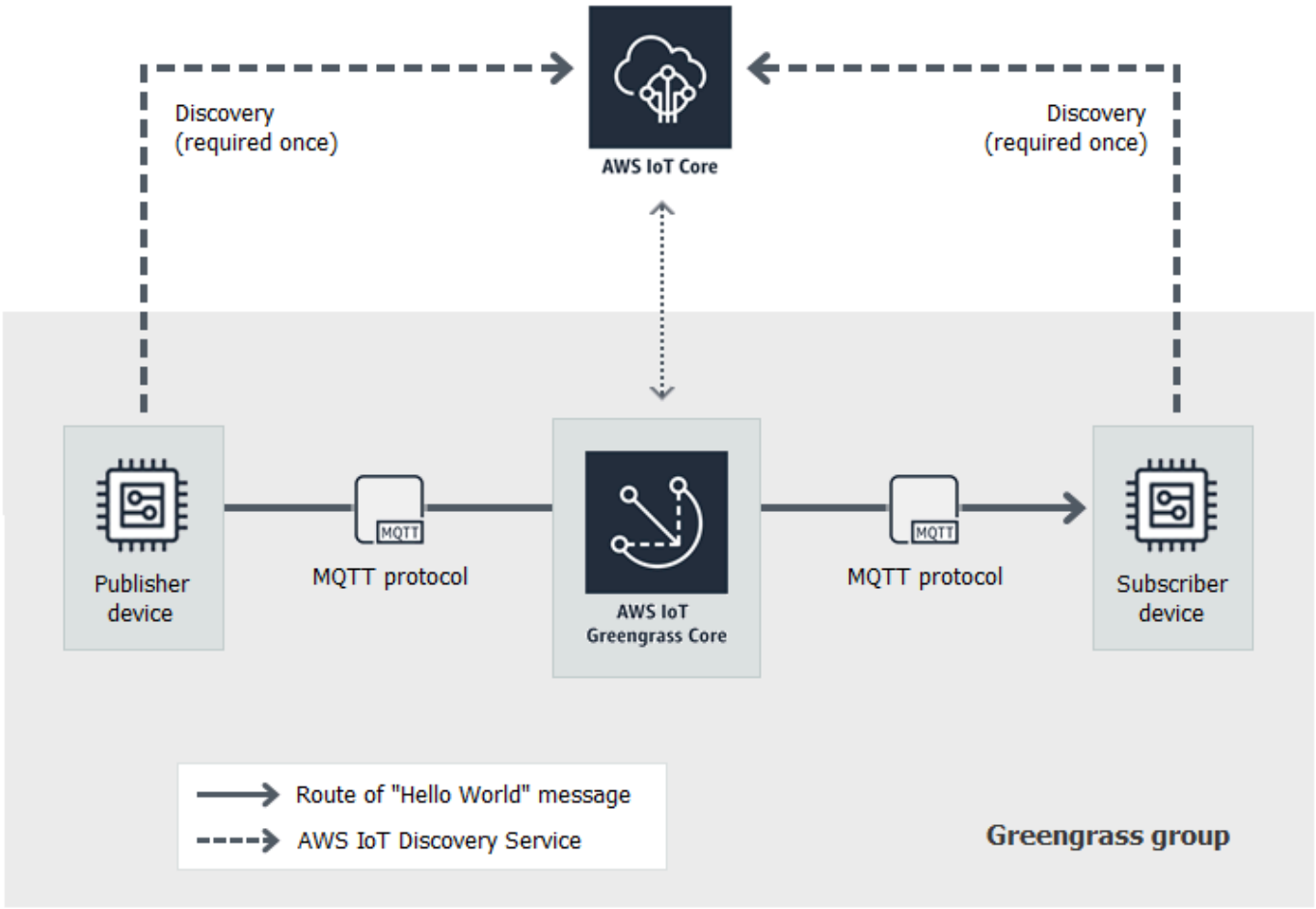


b. 약 30초 기다린 후 주제 게시를 선택합니다. 호출 횟수는 2로 증가되어야 합니다. 이것은 이전 호출에서 생성된 컨테이너가 재사용되고 있고, 함수 핸들러 외부의 사전 처리 변수가 저장되었음을 보여 줍니다.

이제 AWS IoT Greengrass 코어에서 실행할 수 있는 Lambda 함수의 두 가지 유형을 이해해야 합니다. 다음 모듈 4에서는 AWS IoT Greengrass 그룹에서 로컬 IoT 디바이스가 상호 작용하는 방법을 살펴봅니다.

모듈 4: AWS IoT Greengrass 그룹에서 클라이언트 디바이스와 상호 작용

이 모듈에서는 클라이언트 디바이스 또는 디바이스라고 하는 로컬 IoT 디바이스가 AWS IoT Greengrass 코어 디바이스에 연결하고 통신하는 방법을 보여줍니다. AWS IoT Greengrass 코어에 연결되는 클라이언트 디바이스는 AWS IoT Greengrass 그룹의 일부이며 AWS IoT Greengrass 프로그래밍 패러다임에 참여할 수 있습니다. 이 모듈에서는 하나의 클라이언트 디바이스가 Greengrass 그룹 내의 다른 클라이언트 디바이스에 Hello World 메시지를 보냅니다.



시작하기 전에 Greengrass 디바이스 설정 스크립트를 실행하거나 모듈 1 및 모듈 2를 완료합니다. 이 모듈에서는 두 시뮬레이션된 디바이스를 생성합니다. 다른 구성 요소나 디바이스는 필요하지 않습니다.

이 모듈은 완료하는 데 30분이 채 걸리지 않습니다.

주제

- AWS IoT Greengrass 그룹에서 클라이언트 디바이스 생성
- 구독 구성
- Python용 AWS IoT Device SDK 설치
- 통신 테스트

AWS IoT Greengrass 그룹에서 클라이언트 디바이스 생성

이 단계에서는 Greengrass 그룹에 두 개의 클라이언트 디바이스를 추가합니다. 이 프로세스에서는 디바이스를 AWS IoT 사물로 등록하고 인증서와 키를 구성하여 AWS IoT Greengrass에 연결할 수 있습니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 그룹 구성 페이지에서 클라이언트 디바이스를 선택한 다음 연결을 선택합니다.
4. 클라이언트 디바이스를 이 그룹에 연결 모달에서 새 AWS IoT 사물 만들기를 선택합니다.

   사물 생성 페이지가 새 탭에서 열립니다.
5. 사물 생성 페이지에서 단일 사물 생성을 선택한 후 다음을 선택합니다.
6. 사물 속성 지정 페이지에서 이 클라이언트 디바이스를 **`HelloWorld_Publisher`**(으)로 등록하고 다음을 선택합니다.
7. 디바이스 인증서 구성 페이지에서 다음을 선택합니다.
8. 인증서에 정책 첨부 페이지에서 다음 중 하나를 수행합니다.
   - 클라이언트 디바이스에 필요한 권한을 부여하는 기존 정책을 선택한 다음 사물 생성을 선택합니다.

     디바이스가 AWS 클라우드 및 코어에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.
   - 클라이언트 디바이스 권한을 부여하는 새 정책을 만들어 첨부하십시오. 해결 방법:

     a. [정책 생성(Create policy)]을 선택합니다.

        [Create policy(정책 생성)] 페이지가 새 탭에서 열립니다.

     b. [Create policy(정책 생성)] 페이지에서 다음을 수행합니다.

        i. 정책 이름에는 **`GreengrassV1ClientDevicePolicy`**와(과) 같이 정책을 설명하는 이름을 입력합니다.

        ii. 정책 설명 탭의 정책 문서에서 JSON을 선택합니다.

        iii. 다음 정책 문서를 입력합니다. 이 정책을 통해 클라이언트 디바이스는 Greengrass 코어를 검색하고 모든 MQTT 주제에 대해 통신할 수 있습니다. 이 정책의 액세스를 제한하는 방법에 대한 자세한 내용은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여을(를) 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 생성(Create)을 선택하여 정책을 생성합니다.

c. 인증서에 정책 연결 페이지가 열려 있는 브라우저 탭으로 돌아가십시오. 해결 방법:

i. 정책 목록에서 GreengrassV1ClientDevicePolicy과 같이 만든 정책을 선택합니다.

새 정책이 나타나지 않으면 새로고침 아이콘을 선택합니다.

ii. 사물 생성(Create thing)을 선택합니다.

디바이스가 AWS 클라우드 및 코어에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.

9. 인증서 및 키 다운로드 모달에서 디바이스의 인증서를 다운로드합니다.

> **Important**
>
> 완료를 선택하기 전에 보안 리소스를 다운로드합니다.

해결 방법:

a. 디바이스 인증서의 경우 다운로드를 선택하여 디바이스 인증서를 다운로드합니다.
b. 공개 키 파일의 경우 다운로드를 선택하여 인증서의 공개 키를 다운로드합니다.
c. 개인 키 파일의 경우 다운로드를 선택하여 인증서의 개인 키 파일을 다운로드합니다.
d. AWS IoT 개발자 안내서의 서버 인증을 검토하고 적절한 루트 CA 인증서를 선택합니다. Amazon Trust Services(ATS) 엔드포인트와 ATS 루트 CA 인증서를 사용하는 것이 좋습니다. 루트 CA 인증서에서 루트 CA 인증서용 다운로드를 선택합니다.
e. 완료(Done)를 선택합니다.

디바이스 인증서 및 키의 파일 이름에 공통적으로 나타나는 인증서 ID를 기록해 둡니다. 나중에 필요합니다.

10. 클라이언트 디바이스를 이 그룹에 연결 모달을 연 상태로 브라우저 탭으로 돌아가십시오. 해결 방법:

    a. AWS IoT 사물 이름의 경우 생성한 HelloWorld_Publisher을(를) 선택합니다.

       해당 사물이 보이지 않으면 새로 고침 버튼을 선택하십시오.

    b. 연결을 선택합니다.

11. 3~10단계를 반복하여 그룹에 두 번째 클라이언트 디바이스를 추가합니다.

    이 클라이언트 디바이스의 이름을 **`HelloWorld_Subscriber`**(으)로 지정합니다. 클라이언트 디바이스의 인증서 및 키를 컴퓨터에 다운로드합니다. 이번에도 HelloWorld_Subscriber 디바이스의 파일 이름에 공통적인 인증서 ID를 기록해 둡니다.

    이제 Greengrass 그룹에 두 개의 클라이언트 디바이스가 있어야 합니다.

    - HelloWorld_Publisher
    - HelloWorld_Subscriber

12. 컴퓨터에 이러한 클라이언트 디바이스의 보안 인증 정보를 저장할 폴더를 만드세요. 인증서와 키를 이 폴더에 복사합니다.

구독 구성

이 단계에서 HelloWorld_Publisher 클라이언트 디바이스가 MQTT 메시지를 HelloWorld_Subscriber 클라이언트 디바이스로 보낼 수 있게 합니다.

1. 그룹 구성 페이지에서 구독 탭을 선택한 다음 추가를 선택합니다.
2. 구독 생성 페이지에서 다음을 수행하여 구독을 구성합니다.

   a. 소스 유형에서 클라이언트 디바이스를 선택한 다음 HelloWorld_Publisher를 선택합니다.
   b. 대상 유형에서 클라이언트 디바이스, HelloWorld_Subscriber를 차례로 선택합니다.
   c. 주제 필터에 **`hello/world/pubsub`**을 입력합니다.

   > **Note**
   >
   > 이전 모듈에서 구독을 삭제할 수 있습니다. 그룹의 구독 페이지에서 삭제할 구독을 선택한 다음 삭제를 선택합니다.

   d. 구독 생성을 선택합니다.
3. Greengrass 코어가 IP 주소 목록을 게시할 수 있도록 자동 감지가 활성화되어 있는지 확인합니다. 클라이언트 디바이스는 이 정보를 사용하여 코어를 검색합니다. 해결 방법:

   a. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
   b. 시스템 Lambda 함수에서 IP 감지기를 선택한 다음 편집을 선택합니다.
   c. IP 감지기 설정 편집에서 MQTT 브로커 엔드포인트 자동 감지 및 재정의를 선택한 다음 저장을 선택합니다.
4. 코어 디바이스로 클라우드 구성 배포 섹션에 설명된 대로 Greengrass 대몬(daemon)이 실행 중인지 확인합니다.
5. 그룹 구성 페이지에서 배포를 선택합니다.

배포 상태가 페이지 헤더에서 그룹 이름 아래에 표시됩니다. 배포의 세부 정보를 보려면 배포 탭을 선택합니다.

Python용 AWS IoT Device SDK 설치

클라이언트 디바이스는 AWS IoT 및 AWS IoT Greengrass 코어 디바이스와 통신하는 데 Python용 AWS IoT Device SDK으로 사용될 수 있습니다(Python 프로그래밍 언어 사용). 요구 사항을 비롯한 자세한 내용은 GitHub의 Python용 AWS IoT Device SDK Readme 단원을 참조하십시오.

이 단계에서는 SDK를 설치하고 컴퓨터의 시뮬레이션된 클라이언트 디바이스에서 사용하는 `basicDiscovery.py` 샘플 함수를 가져옵니다.

1. 컴퓨터에 SDK와 모든 필수 구성 요소를 함께 설치하려면 운영 체제를 선택합니다.

   Windows

   1. 관리자 권한 명령 프롬프트를 열고 다음 명령을 실행합니다.

      ```
      python --version
      ```

      버전 정보가 반환되지 않았거나 버전 번호가 2.7 미만(Python 2) 또는 3.3 미만(Python 3)이면 Python 다운로드의 지침에 따라 Python 2.7 이상 또는 Python 3.3 이상을 설치합니다. 자세한 내용은 Windows에서 Python 사용을 참조하십시오.

   2. Python용 AWS IoT Device SDK을 `zip` 파일로 다운로드하고 컴퓨터의 적절한 위치에 압축을 해제합니다.

      `setup.py` 파일이 들어 있는 압축 해제된 `aws-iot-device-sdk-python-master` 폴더의 파일 경로를 메모합니다. 다음 단계에서 이 파일 경로는 *`path-to-SDK-folder`*로 표시됩니다.

   3. 관리자 권한 명령 프롬프트에서 다음을 실행합니다.

      ```
      cd path-to-SDK-folder
      python setup.py install
      ```

   macOS

   1. 터미널 창을 열고 다음 명령을 실행합니다.

      ```
      python --version
      ```

      버전 정보가 반환되지 않았거나 버전 번호가 2.7 미만(Python 2) 또는 3.3 미만(Python 3)이면 Python 다운로드의 지침에 따라 Python 2.7 이상 또는 Python 3.3 이상을 설치합니다. 자세한 내용은 Macintosh에서 Python 사용을 참조하십시오.

   2. 터미널 창에서 다음 명령을 실행하여 OpenSSL 버전을 확인합니다.

      ```
      python
      ```

```
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

OpenSSL 버전 값을 기록해 둡니다.

> **Note**
>
> Python 3를 실행 중인 경우 print(ssl.OPENSSL_VERSION)를 사용합니다.

Python 셸을 닫으려면 다음 명령을 실행합니다.

```
>>>exit()
```

OpenSSL 버전이 1.0.1 이상이면 c단계로 건너뜁니다. 그렇지 않은 경우 다음 단계를 따르십시오.

- 터미널 창에서 다음 명령을 실행하여 컴퓨터에서 Simple Python Version Management를 사용 중인지 확인합니다.

  ```
  which pyenv
  ```

파일 경로가 반환되면 [**pyenv** 사용] 탭을 선택합니다. 반환되지 않으면 [**pyenv** 사용 안 함] 탭을 선택합니다.

Using pyenv

1. Mac OS X용 Python 릴리스(또는 유사 항목)를 참조하여 안정적인 최신 Python 버전을 확인하십시오. 다음 예에서 이 값은 *latest-Python-version*으로 표시됩니다.
2. 터미널 창에서 다음 명령을 실행합니다.

   ```
   pyenv install latest-Python-version
   pyenv global latest-Python-version
   ```

   예를 들어 Python 2의 최신 버전이 2.7.14이면 이 명령은 다음과 같습니다.

   ```
   pyenv install 2.7.14
   ```

```
pyenv global 2.7.14
```

3. 터미널 창을 닫았다가 다시 열고 다음 명령을 실행합니다.

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

OpenSSL 버전은 1.0.1 이상이어야 합니다. 버전이 1.0.1 미만이면 업데이트가 실패한 것입니다. pyenv install 및 pyenv global 명령에서 사용된 Python 버전 값을 확인하고 다시 시도하십시오.

4. 다음 명령을 실행하여 Python 셸을 종료합니다.

```
 exit()
```

Not using pyenv

1. 터미널 창에서 다음 명령을 실행하여 brew가 설치되어 있는지 확인합니다.

```
which brew
```

파일 경로가 반환되지 않으면 다음과 같이 `brew`를 설치합니다.

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"
```

> **Note**
>
> 설치 프롬프트에 따릅니다. Xcode 명령줄 도구 다운로드에 약간의 시간이 걸릴 수 있습니다.

2. 다음 명령을 실행합니다:

```
brew update
brew install openssl
brew install python@2
```

Python용 AWS IoT Device SDK에는 Python 실행 파일로 컴파일된 OpenSSL 버전 1.0.1 이상이 필요합니다. brew install python 명령은 이 요구 사항을 충족하는 python2 실행 파일을 설치합니다. python2 실행 파일은 /usr/local/bin 디렉터리에 설치되며, PATH 환경 변수의 일부여야 합니다. 확인하려면 다음 명령을 실행합니다.

```
python2 --version
```

python2 버전 정보가 제공되면 다음 단계로 건너뜁니다. 그렇지 않으면 셸 프로필에 다음 줄을 추가하여 PATH 환경 변수에 대한 /usr/local/bin 경로를 영구적으로 추가합니다.

```
export PATH="/usr/local/bin:$PATH"
```

예를 들어 .bash_profile을 사용 중이거나 아직 셸 프로필이 없는 경우 터미널 창에서 다음 명령을 실행합니다.

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

이후 셸 프로필에 source를 지정한 다음 python2 --version에서 버전 정보를 제공하는지 확인합니다. 예를 들어 .bash_profile을 사용 중인 경우 다음 명령을 실행합니다.

```
source ~/.bash_profile
python2 --version
```

python2 버전 정보가 반환되어야 합니다.

3. 다음 줄을 셸 프로필에 추가합니다.

```
alias python="python2"
```

예를 들어 .bash_profile을 사용 중이거나 아직 셸 프로필이 없는 경우 다음 명령을 실행합니다.

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. 이후 셸 프로필에 source를 지정합니다. 예를 들어 `.bash_profile`을 사용 중인 경우 다음 명령을 실행합니다.

   ```
   source ~/.bash_profile
   ```

   python 명령을 호출하면 필수 OpenSSL 버전이 포함된 Python 실행 파일(예: `python2`)이 실행됩니다.

5. 다음 명령을 실행합니다:

   ```
   python
    import ssl
    print ssl.OPENSSL_VERSION
   ```

   OpenSSL 버전은 1.0.1 이상이어야 합니다.

6. Python 셸을 종료하려면 다음 명령을 실행합니다.

   ```
    exit()
   ```

3. 다음 명령을 사용하여 Python용 AWS IoT Device SDK을 설치합니다.

   ```
   cd ~
   git clone https://github.com/aws/aws-iot-device-sdk-python.git
   cd aws-iot-device-sdk-python
   sudo python setup.py install
   ```

UNIX-like system

1. 터미널 창에서 다음 명령을 실행합니다.

   ```
   python --version
   ```

   버전 정보가 반환되지 않았거나 버전 번호가 2.7 미만(Python 2) 또는 3.3 미만(Python 3)이면 Python 다운로드의 지침에 따라 Python 2.7 이상 또는 Python 3.3 이상을 설치합니다. 자세한 내용은 Unix 플랫폼에서 Python 사용을 참조하십시오.

2. 터미널에서 다음 명령을 실행하여 OpenSSL 버전을 확인합니다.

   ```
   python
   ```

```
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

OpenSSL 버전 값을 기록해 둡니다.

> **Note**
>
> Python 3를 실행 중인 경우 print(ssl.OPENSSL_VERSION)를 사용합니다.

Python 셸을 닫으려면 다음 명령을 실행합니다.

```
exit()
```

OpenSSL 버전이 1.0.1 이상이면 다음 단계로 건너뜁니다. 그렇지 않으면 해당 명령을 실행하여 배포용 OpenSSL을 업데이트합니다(예: `sudo yum update openssl`, `sudo apt-get update` 등).

다음 명령을 실행하여 OpenSSL 버전이 1.0.1 이상인지 확인합니다.

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. 다음 명령을 사용하여 AWS IoT Device SDK 디바이스 SDK for Python을 설치합니다.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. Python용 AWS IoT Device SDK가 설치되면 `samples` 폴더로 이동하여 `greengrass` 폴더를 엽니다.

   이 자습서에서는 the section called “AWS IoT Greengrass 그룹에서 클라이언트 디바이스 생성”에서 다운로드한 인증서와 키를 사용하는 `basicDiscovery.py` 샘플 함수를 복사합니다.

3. HelloWorld_Publisher 및 HelloWorld_Subscriber 디바이스 인증서 및 키가 포함된 폴더에 `basicDiscovery.py`를 복사합니다.

통신 테스트

1. 컴퓨터와 AWS IoT Greengrass 코어 장치가 동일한 네트워크를 사용하여 인터넷에 연결되어 있는지 확인합니다.

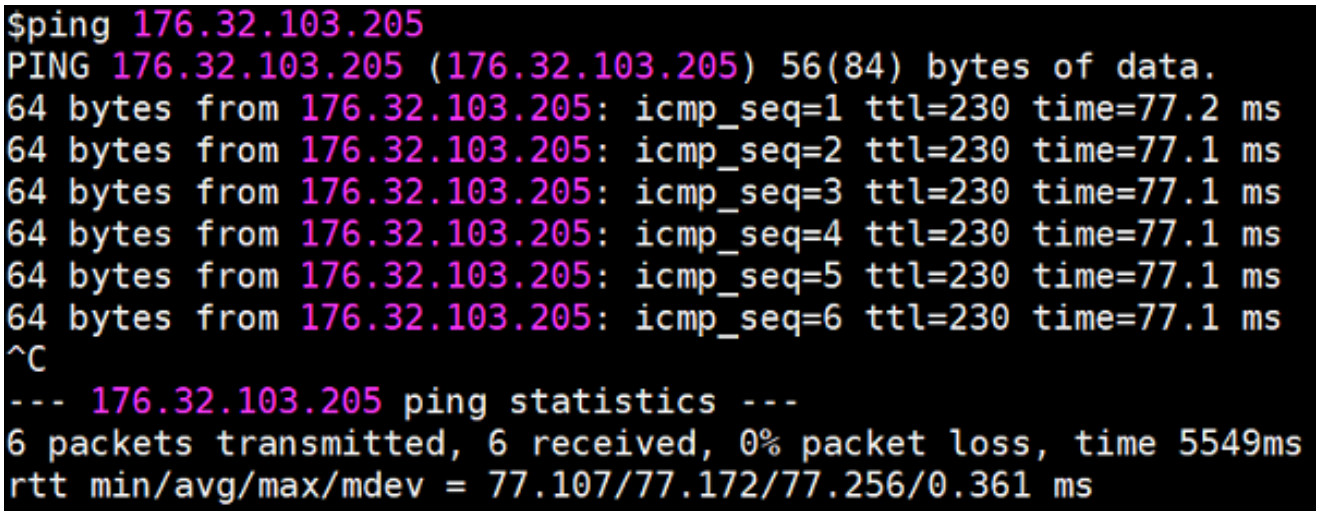
   a. AWS IoT Greengrass 코어 디바이스에서 다음 명령을 실행하여 해당 IP 주소를 찾습니다.

   ```
   hostname -I
   ```

   b. 컴퓨터에서 코어의 IP 주소를 사용하여 다음 명령을 실행합니다. Ctrl + C를 사용하여 ping 명령을 중지할 수 있습니다.

   ```
   ping IP-address
   ```

   다음과 비슷한 출력은 컴퓨터와 AWS IoT Greengrass 코어 장치 간의 통신 성공 (패킷 손실 0%) 을 나타냅니다.

   > **Note**
   >
   > 실행 중인 AWS IoT Greengrass EC2 인스턴스를 ping할 수 없는 경우 인스턴스의 인바운드 보안 그룹 규칙이 Echo 요청 메시지에 대한 ICMP 트래픽을 허용하는지 확인하십시오. 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가를 참조하십시오.
   >
   > Windows 호스트 컴퓨터에 어드밴스 보안 앱을 갖춘 Windows 방화벽에서 인바운드 에코 요청(예: File and Printer Sharing(파일 및 프린터 공유)(에코 요청 - ICMPv4-In))을 허용하는 인바운드 규칙을 활성화하거나 생성해야 할 수도 있습니다.

2. AWS IoT 엔드포인트 가져오기.

   a. AWS IoT 콘솔의 탐색 창에서 설정을 선택합니다.

   b. 장치 데이터 엔드포인트에서 엔드포인트의 값을 기록해 둡니다. 이 값을 사용하여 다음 단계의 명령에서 *AWS_IOT_ENDPOINT* 자리 표시자를 바꿉니다.

   > **Note**
   >
   > 엔드포인트는 해당 인증서 유형과 일치해야 합니다.

3. 컴퓨터 (AWS IoT Greengrass 코어 장치 아님) 에서 두 개의 명령줄 (터미널 또는 명령 프롬프트) 창을 엽니다. 한 창은 HelloWorld _Publisher 클라이언트 장치를 나타내고 다른 창은 _Subscriber 클라이언트 장치를 나타냅니다. HelloWorld

   실행 시 엔드포인트의 AWS IoT Greengrass 코어 위치에 대한 정보 수집을 `basicDiscovery.py` 시도합니다. 이 정보는 클라이언트 장치가 코어를 발견하고 성공적으로 연결한 후 저장됩니다. 이 경우 향후 메시징 및 작업이 로컬에서 실행될 수 있습니다(인터넷 연결 필요 없음).

   > **Note**
   >
   > MQTT 연결에 사용되는 클라이언트 ID는 클라이언트 장치의 사물 이름과 일치해야 합니다. `basicDiscovery.py` 스크립트는 MQTT 연결의 클라이언트 ID를 스크립트 실행 시 지정한 사물 이름으로 설정합니다.
   > `basicDiscovery.py` 파일이 포함된 폴더에서 다음 명령을 실행하면 상세한 스크립트 사용 정보를 알 수 있습니다.
   >
   > ```
   > python basicDiscovery.py --help
   > ```

4. HelloWorld_Publisher 클라이언트 장치 창에서 다음 명령을 실행합니다.

   - *path-to-certs-folder*를 인증서, 키 및 `basicDiscovery.py`를 포함하는 폴더의 경로로 바꿉니다.
   - *AWS_IOT_ENDPOINT*를 엔드포인트로 바꿉니다.
   - 두 *### CertId ##### HelloWorld _Publisher* 클라이언트 디바이스의 파일 이름에 있는 인증서 ID로 교체합니다.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key
 --thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --
message 'Hello, World! Sent from HelloWorld_Publisher'
```

`Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`와 같은 항목을 포함하는 다음과 비슷한 출력 화면이 보여야 합니다.

> **Note**
>
> 스크립트가 `error: unrecognized arguments` 메시지를 반환하는 경우 `--topic` 및 `--message` 파라미터의 작은따옴표를 큰따옴표로 변경하고 명령을 다시 실행합니다. 연결 문제를 해결하려면 수동 IP 감지를 사용합니다.

```
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}

2017-11-13 21:12:26,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:26,297 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:27,301 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}

2017-11-13 21:12:27,302 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:27,303 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:28,305 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}

2017-11-13 21:12:28,306 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:28,307 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:29,310 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
```

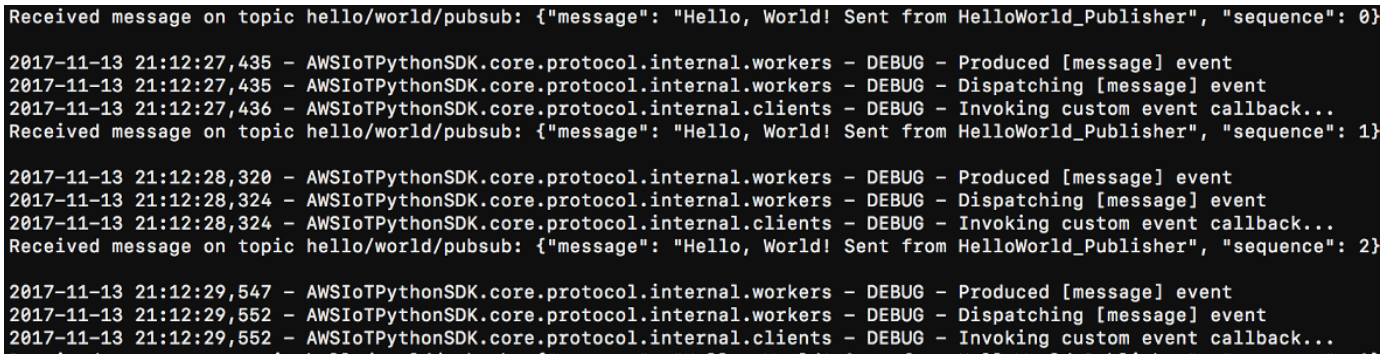
5. HelloWorld_Subscriber 클라이언트 디바이스 창에서 다음 명령을 실행합니다.

   - *path-to-certs-folder*를 인증서, 키 및 `basicDiscovery.py`를 포함하는 폴더의 경로로 바꿉니다.
   - *AWS_IOT_ENDPOINT*를 엔드포인트로 바꿉니다.
   - 두 *### CertId ##### HelloWorld _Subscriber* 클라이언트 디바이스의 파일 이름에 있는 인증서 ID로 교체합니다.

```
cd path-to-certs-folder
```

```
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
```

`Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`와 같은 항목을 포함하는 다음 출력 화면이 보여야 합니다.



HelloWorld_Subscriber 창에 메시지가 누적되지 않도록 하려면 HelloWorld_Publisher 창을 닫습니다.

기업 네트워크에 대한 테스트는 코어 연결을 방해할 수 있습니다. 차선책으로서 엔드포인트를 수동으로 입력할 수 있습니다. 이렇게 하면 `basicDiscovery.py` 스크립트가 코어 디바이스의 올바른 IP 주소에 연결됩니다. AWS IoT Greengrass

수동으로 엔드포인트를 입력하려면

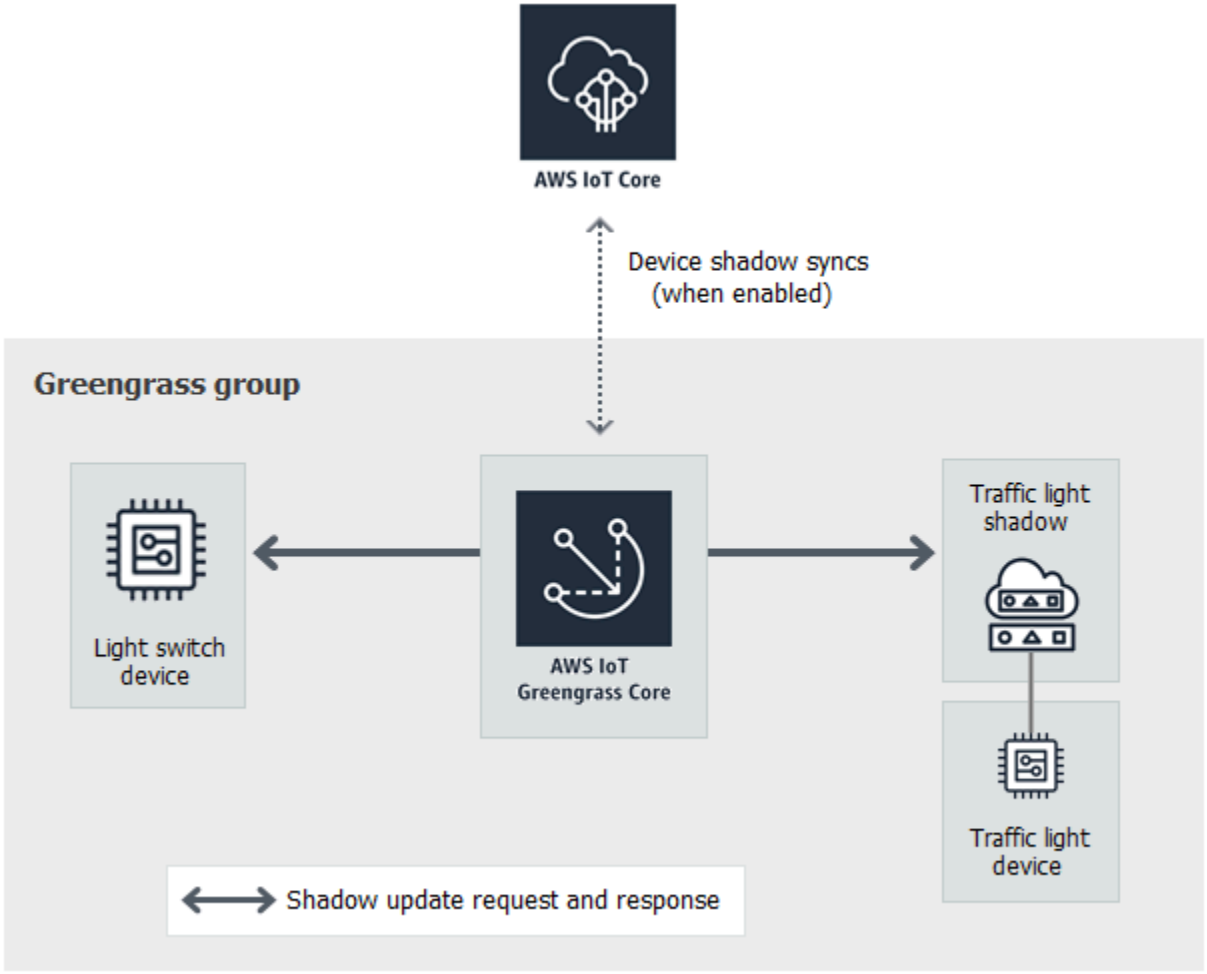
1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 장치를 확장한 다음 그룹 (V1) 을 선택합니다.
2. Greengrass 그룹에서 사용자 그룹을 선택합니다.
3. MQTT 브로커 엔드포인트를 수동으로 관리하도록 코어를 구성하십시오. 다음을 따릅니다.
   a. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
   b. 시스템 Lambda 함수에서 IP 감지기를 선택한 다음 편집을 선택합니다.
   c. IP 감지기 설정 편집에서 MQTT 브로커 엔드포인트 수동 관리를 선택한 다음 저장을 선택합니다.
4. 코어의 MQTT 브로커 엔드포인트를 입력합니다. 다음을 따릅니다.
   a. 개요에서 Greengrass 코어를 선택합니다.
   b. MQTT 브로커 엔드포인트에서 엔드포인트 관리를 선택합니다.

c. 엔드포인트 추가를 선택하고 엔드포인트 값이 하나뿐인지 확인합니다. 이 값은 AWS IoT Greengrass 코어 디바이스의 포트 8883의 IP 주소 엔드포인트여야 합니다 (예:). `192.168.1.4`

d. 업데이트를 선택합니다.

모듈 5: 디바이스 섀도우와 상호 작용

이 고급 모듈에서는 클라이언트 디바이스가 AWS IoT Greengrass 그룹 내의 AWS IoT 디바이스 섀도우와 상호 작용하는 방법을 보여줍니다. 섀도우는 사물의 현재 상태 정보 또는 원하는 상태 정보를 저장하는 데 사용되는 JSON 문서입니다. 이 모듈에서는 하나의 클라이언트 디바이스(`GG_Switch`)가 다른 클라이언트 디바이스(`GG_TrafficLight`)의 상태를 수정하는 방법과 이러한 상태를 AWS IoT Greengrass 클라우드와 동기화하는 방법을 살펴봅니다.



시작하기 전에 Greengrass 디바이스 설정 스크립트를 실행하거나 모듈 1 및 모듈 2를 완료했는지 확인합니다. 또한 클라이언트 디바이스를 AWS IoT Greengrass 코어에 연결하는 방법(모듈 4)을 알고 있어야 합니다. 다른 구성 요소나 디바이스는 필요하지 않습니다.

이 모듈을 완료하는 데 약 30분 정도 걸립니다.

주제

- 디바이스 및 구독 구성
- 필수 파일 다운로드
- 통신 테스트(장치 동기화 비활성화됨)
- 통신 테스트(디바이스 동기화 활성화됨)

디바이스 및 구독 구성

AWS IoT Greengrass 코어가 인터넷에 연결되면 섀도우가 AWS IoT에 동기화될 수 있습니다. 이 모듈에서는 우선 클라우드와 동기화하지 않고 로컬 섀도우를 사용합니다. 그런 다음, 클라우드 동기화를 활성화합니다.

각 클라이언트 디바이스에는 고유한 섀도우가 있습니다. 자세한 내용은 AWS IoT 개발자 안내서의 AWS IoT용 디바이스 섀도우 서비스를 참조하십시오.

1. 그룹 구성 페이지에서 클라이언트 디바이스 탭을 선택합니다.
2. 클라이언트 디바이스 탭에서 AWS IoT Greengrass 그룹에 두 개의 새 클라이언트 디바이스를 추가합니다. 이 프로세스에 대한 자세한 단계는 the section called “AWS IoT Greengrass 그룹에서 클라이언트 디바이스 생성”을 참조하십시오.

   - **`GG_Switch`** 및 **`GG_TrafficLight`** 클라리언트 디바이스의 이름을 지정합니다.
   - 두 클라이언트 디바이스의 원클릭 기본 보안 리소스를 생성하고 다운로드합니다.
   - 클라이언트 디바이스의 보안 리소스 파일 이름에 해시 구성 요소를 적어 둡니다. 나중에 이 값을 사용합니다.
3. 컴퓨터에 이러한 클라이언트 디바이스의 보안 인증 정보를 저장할 폴더를 만드세요. 인증서와 키를 이 폴더에 복사합니다.
4. 클라이언트 디바이스가 로컬 섀도우를 사용하도록 설정되어 있는지 AWS 클라우드 확인합니다. 그렇지 않은 경우 클라이언트 디바이스를 선택하고 섀도우 동기화를 선택한 다음 클라우드와의 섀도우 동기화 비활성화를 선택합니다.
5. 다음 표의 구독을 그룹에 추가합니다. 예를 들어 첫 번째 구독을 생성하려면

a. 그룹 구성 페이지에서 구독 탭을 선택한 다음 추가을(를) 선택합니다.

b. 소스 유형에서 클라이언트 디바이스를 선택한 다음 GG_Switch를 선택합니다.

c. 대상 유형에서 서비스를 선택한 다음 Local Shadow Service를 선택합니다.

d. 주제 필터에 **$aws/things/GG_TrafficLight/shadow/update**를 입력합니다.

e. 구독 생성을 선택합니다.

주제는 테이블에 표시된 것처럼 정확히 입력되어야 합니다. 와일드카드를 사용하여 일부 구독을 통합할 수 있지만 권장하지는 않습니다. 자세한 내용은 AWS IoT 개발자 안내서의 Shadow MQTT 주제를 참조하십시오.

소스	대상	주제	주의
GG_Switch	로컬 섀도우 서비스	$aws/things/GG_TrafficLight/shadow/update	GG_Switch는 업데이트 요청을 보내 주제를 업데이트합니다.
로컬 섀도우 서비스	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/accepted	GG_Switch는 업데이트 요청이 수락되었는지 알아야 합니다.
로컬 섀도우 서비스	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/rejected	GG_Switch는 업데이트 요청이 거부되었는지 알아야 합니다.
GG_TrafficLight	로컬 섀도우 서비스	$aws/things/GG_TrafficLight/shadow/update	GG_TrafficLight는 업데이트 주제로 상태 업데이트를 보냅니다.
로컬 섀도우 서비스	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/delta	로컬 섀도우 서비스는 델타 주제를 통해 GG_TrafficLight로 수신된 업데이트를 보냅니다.

소스	대상	주제	주의
로컬 섀도우 서비스	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/accepted	GG_TrafficLight는 상태 업데이트가 수락되었는지 알아야 합니다.
로컬 섀도우 서비스	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/rejected	GG_TrafficLight는 상태 업데이트가 거부되었는지 알아야 합니다.

구독 탭에 새 구독이 표시됩니다.

> **Note**
>
> $ 문자에 대한 정보는 예약된 주제를 참조하십시오.

6. Greengrass 코어가 IP 주소 목록을 게시할 수 있도록 자동 감지가 활성화되어 있는지 확인합니다. 클라이언트 디바이스는 이 정보를 사용하여 코어를 검색합니다. 해결 방법:

   a. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.

   b. 시스템 Lambda 함수에서 IP 감지기를 선택한 다음 편집을 선택합니다.

   c. IP 감지기 설정 편집에서 MQTT 브로커 엔드포인트 자동 감지 및 재정의를 선택한 다음 저장을 선택합니다.

7. 코어 디바이스로 클라우드 구성 배포 단원에 설명된 대로 Greengrass 데몬이 실행 중인지 확인합니다.

8. 그룹 구성 페이지에서 배포를 선택합니다.

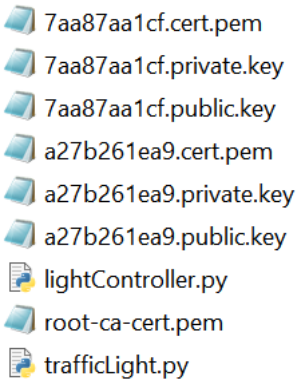
필수 파일 다운로드

1. 아직 설치하지 않았다면 Python용 AWS IoT Device SDK를 설치합니다. 지침은 the section called “Python용 AWS IoT Device SDK 설치”의 1단계를 참조하십시오.

   이 SDK는 AWS IoT 및 AWS IoT Greengrass 코어 디바이스와 통신하기 위해 클라이언트 디바이스가 사용합니다.

2. GitHub의 TrafficLight 예제 폴더에서 `lightController.py` 및 `trafficLight.py` 파일을 컴퓨터로 다운로드합니다. 파일을 GG_Switch 및 GG_TrafficLight 클라이언트 디바이스 인증서 및 키를 포함하는 폴더에 저장합니다.

   `lightController.py` 스크립트는 GG_Switch 디바이스에 해당하고, `trafficLight.py` 스크립트는 GG_TrafficLight 디바이스에 해당합니다.

   > **Note**
   >
   > 예제 Python 파일은 편의를 위해 AWS IoT Greengrass Core SDK for Python 리포지토리에 저장되지만 AWS IoT Greengrass 코어 SDK를 사용하지 않습니다.

통신 테스트(장치 동기화 비활성화됨)

1. 컴퓨터와 AWS IoT Greengrass 코어 장치가 동일한 네트워크를 사용하여 인터넷에 연결되어 있는지 확인합니다.

   a. AWS IoT Greengrass 코어 디바이스에서 다음 명령을 실행하여 해당 IP 주소를 찾습니다.

   ```
   hostname -I
   ```

   b. 컴퓨터에서 코어의 IP 주소를 사용하여 다음 명령을 실행합니다. Ctrl + C를 사용하여 ping 명령을 중지할 수 있습니다.

   ```
   ping IP-address
   ```

   다음과 비슷한 출력은 컴퓨터와 AWS IoT Greengrass 코어 장치 간의 통신 성공 (패킷 손실 0%) 을 나타냅니다.

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

> **Note**
>
> 실행 중인 AWS IoT Greengrass EC2 인스턴스를 ping할 수 없는 경우 인스턴스의 인바운드 보안 그룹 규칙이 Echo 요청 메시지에 대한 ICMP 트래픽을 허용하는지 확인하십시오. 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가를 참조하십시오.
>
> Windows 호스트 컴퓨터에 어드밴스 보안 앱을 갖춘 Windows 방화벽에서 인바운드 에코 요청(예: File and Printer Sharing(파일 및 프린터 공유)(에코 요청 - ICMPv4-In))을 허용하는 인바운드 규칙을 활성화하거나 생성해야 할 수도 있습니다.

2. AWS IoT 엔드포인트 가져오기.

   a. AWS IoT 콘솔의 탐색 창에서 설정을 선택합니다.

   b. 장치 데이터 엔드포인트에서 엔드포인트의 값을 기록해 둡니다. 이 값을 사용하여 다음 단계의 명령에서 *AWS_IOT_ENDPOINT* 자리 표시자를 바꿉니다.

> **Note**
>
> 엔드포인트는 해당 인증서 유형과 일치해야 합니다.

3. 컴퓨터 (AWS IoT Greengrass 코어 장치 아님) 에서 두 개의 명령줄 (터미널 또는 명령 프롬프트) 창을 엽니다. 한 창은 GG_Switch 클라이언트 장치를 나타내고 다른 창은 GG_ 클라이언트 장치를 나타냅니다. TrafficLight

   a. GG_Switch 클라이언트 장치 창에서 다음 명령을 실행합니다.

- *path-to-certs-folder*를 인증서, 키 및 Python 파일을 포함하는 폴더의 경로로 바꿉니다.
- *AWS_IOT_ENDPOINT*를 엔드포인트로 바꿉니다.
- 두 *### CertId* 인스턴스를 GG_Switch 클라이언트 디바이스의 파일 이름에 있는 인증서 ID로 교체합니다.

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA
 AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-
private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
```

b. GG_ TrafficLight 클라이언트 디바이스 창에서 다음 명령을 실행합니다.

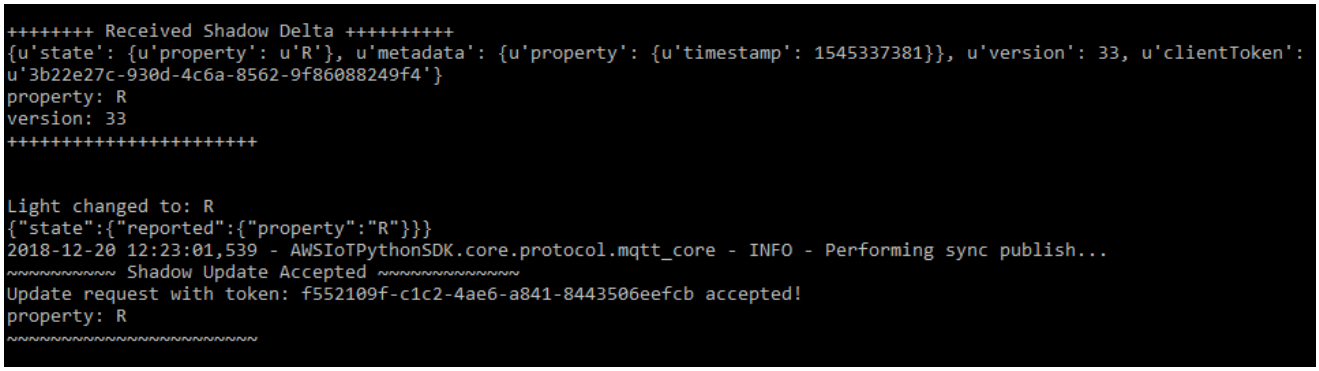
- *path-to-certs-folder*를 인증서, 키 및 Python 파일을 포함하는 폴더의 경로로 바꿉니다.
- *AWS_IOT_ENDPOINT*를 엔드포인트로 바꿉니다.
- 두 개의 *### CertId* 인스턴스를 GG_ TrafficLight 클라이언트 디바이스의 파일 이름에 있는 인증서 ID로 교체합니다.

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_TrafficLight
```

스위치는 20초마다 섀도우 상태를 G, Y 및 R로 업데이트하고, 전등은 다음에 나온 것처럼 새로운 상태를 표시합니다.

GG_Switch 출력:

GG_ 출력TrafficLight :



처음 실행될 때 각 클라이언트 장치 스크립트는 AWS IoT Greengrass 검색 서비스를 실행하여 인터넷을 통해 AWS IoT Greengrass 코어에 연결합니다. 클라이언트 장치가 검색되어 AWS IoT Greengrass 코어에 성공적으로 연결되면 향후 작업을 로컬에서 실행할 수 있습니다.

> **Note**
>
> `lightController.py` 및 `trafficLight.py` 스크립트는 스크립트와 동일한 폴더에 생성되는 `groupCA` 폴더에 연결 정보를 저장합니다. 연결 오류를 수신하는 경우 `ggc-host` 파일의 IP 주소가 이 단계에서 코어에 대해 구성한 단일 IP 주소 엔드포인트와 일치하는지 확인합니다.

4. AWS IoT 콘솔에서 AWS IoT Greengrass 그룹을 선택하고 클라이언트 장치 탭을 선택한 다음 TrafficLightGG_를 선택하여 클라이언트 장치의 AWS IoT 사물 세부 정보 페이지를 엽니다.
5. 장치 섀도우 탭을 선택합니다. GG_Switch의 상태가 변경된 후에는 이 섀도우에 대한 업데이트가 있으면 안 됩니다. GG_가 클라우드와의 섀도우 동기화 비활성화로 TrafficLight 설정되어 있기 때문입니다.
6. GG_Switch(`lightController.py`) 클라이언트 장치 창에서 Ctrl + C를 누릅니다. GG_TrafficLight (`trafficLight.py`) 창에서 상태 변경 메시지 수신이 중지되는 것을 확인할 수 있을 것입니다.

   다음 단원에서 명령을 실행할 수 있도록 이러한 창을 그대로 열어 둡니다.

통신 테스트(디바이스 동기화 활성화됨)

이 테스트의 경우 GG_TrafficLight 디바이스 섀도우가 AWS IoT와 동기화되도록 구성합니다. 이전 테스트의 경우와 동일한 명령을 실행하지만 이번에는 GG_Switch가 업데이트 요청을 보낼 때 클라우드의 섀도우 상태가 업데이트됩니다.

1. AWS IoT 콘솔에서 AWS IoT Greengrass 그룹을 선택한 다음 클라이언트 디바이스 탭을 선택합니다.
2. GG_TrafficLight 디바이스를 선택하고 섀도우 동기화를 선택한 다음 클라우드와 섀도우 동기화 활성화를 선택합니다.

   디바이스 섀도우 동기화 상태가 업데이트되었다는 알림을 수신해야 합니다.
3. 그룹 구성 페이지에서 배포를 선택합니다.
4. 명령줄 창 두 개에서 GG_Switch 및 GG_TrafficLight 클라이언트 디바이스에 대해 이전 테스트에서 명령을 실행합니다.
5. 이제 AWS IoT 콘솔의 섀도우 상태를 확인합니다. AWS IoT Greengrass 그룹을 선택하고, 클라이언트 디바이스 탭을 선택한 다음 GG_TrafficLight를 선택하고, 디바이스 섀도우 탭을 선택한 다음, 클래식 섀도우를 선택합니다.

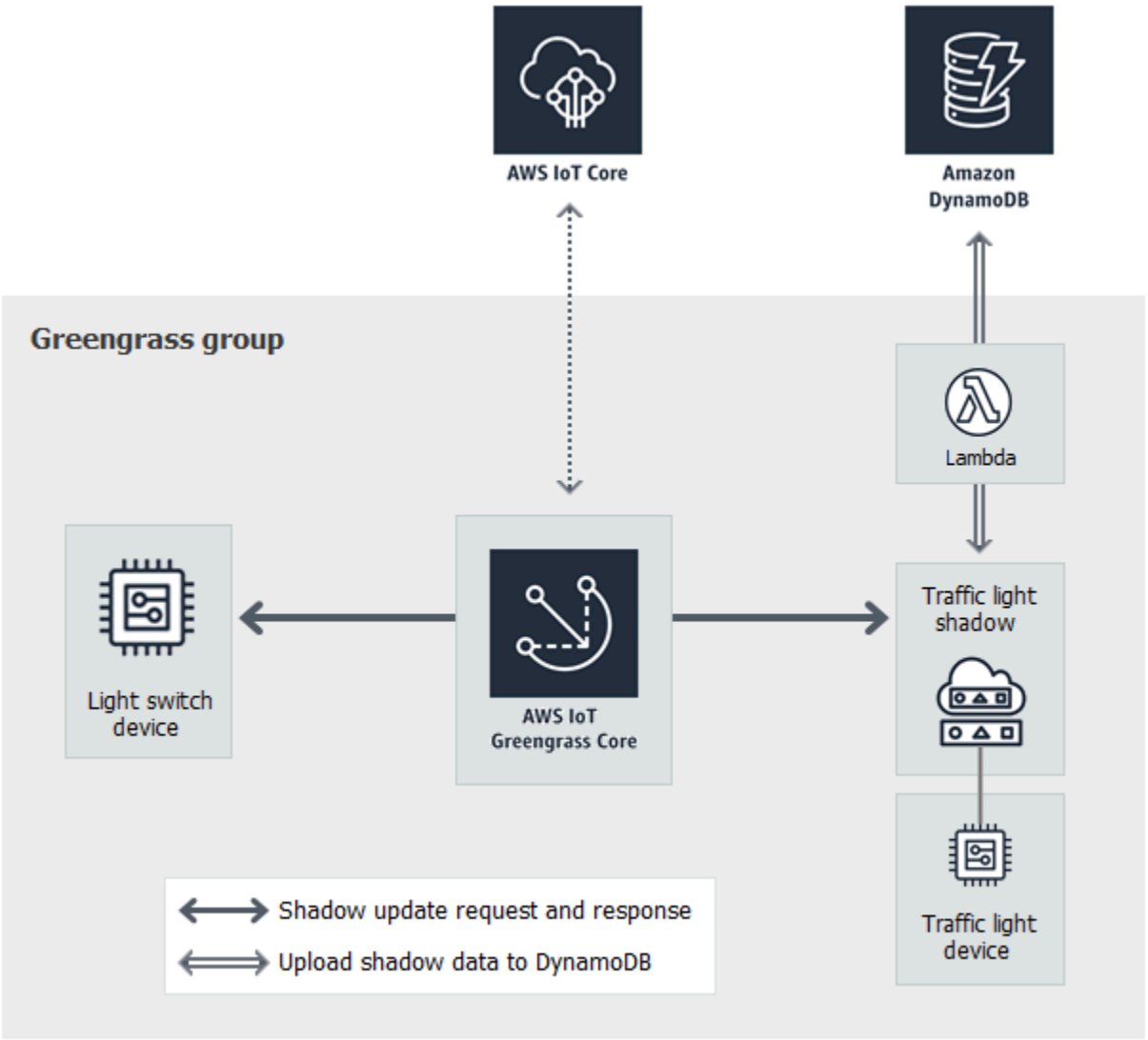
   AWS IoT에 대한 GG_TrafficLight 섀도우의 동기화를 활성화했기 때문에 GG_Switch가 업데이트를 보낼 때마다 클라우드의 섀도우 상태가 업데이트되어야 합니다. 이 기능은 AWS IoT에 클라이언트 디바이스의 상태를 노출하는 데 사용될 수 있습니다.

> **Note**
>
> 필요한 경우 AWS IoT Greengrass 코어 로그, 특히 `runtime.log`를 확인하여 문제를 해결할 수 있습니다.
>
> ```
> cd /greengrass/ggc/var/log
> sudo cat system/runtime.log | more
> ```
>
> `GGShadowSyncManager.log`와 `GGShadowService.log`도 볼 수 있습니다. 자세한 내용은 문제 해결 섹션을 참조하세요.

클라이언트 디바이스 및 구독 설정을 유지합니다. 다음 모듈에서 이 둘을 사용합니다. 동일한 명령도 실행합니다.

모듈 6: 다른 AWS 서비스에 액세스

이 고급 모듈에서는 AWS IoT Greengrass 코어가 클라우드에서 다른 AWS 서비스와 상호 작용하는 방법을 보여 줍니다. 이 모듈은 모듈 5의 신호등 예제를 기반으로 하며, 섀도우 상태를 처리하는 Lambda 함수를 추가하고 클라우드의 Amazon DynamoDB 테이블로 요약을 업로드합니다.



시작하기 전에 Greengrass 디바이스 설정 스크립트를 실행하거나 모듈 1 및 모듈 2를 완료했는지 확인합니다. 또한 모듈 5도 완료해야 합니다. 다른 구성 요소나 디바이스는 필요하지 않습니다.

이 모듈을 완료하는 데 약 30분 정도 걸립니다.

> **Note**
>
> 이 모듈에서는 DynamoDB에서 테이블을 생성하고 업데이트합니다. 대부분의 작업이 간단하고 Amazon Web Services 프리 티어 내에 속하지만 이 모듈의 일부 단계를 수행하면 계정에 요금이 청구될 수 있습니다. 요금에 대한 자세한 내용은 DynamoDB 요금 설명서를 참조하십시오.

주제

그룹 역할 구성

그룹 역할은 사용자가 생성하고 Greengrass 그룹에 연결하는 IAM 역할입니다. 이 역할은 배포된 Lambda 함수(및 다른 AWS IoT Greengrass 기능)가 AWS 서비스에 액세스하기 위해 사용하는 권한들을 포함하고 있습니다. 자세한 내용은 the section called “Greengrass 그룹 역할” 섹션을 참조하세요.

IAM 콘솔에서 그룹 역할을 생성하려면 다음과 같은 상위 단계를 사용합니다.

1. 하나 이상의 리소스에 대한 작업을 허용하거나 거부하는 정책을 생성합니다.
2. Greengrass 서비스를 신뢰할 수 있는 엔터티로 사용하는 역할을 생성합니다.
3. 역할에 정책을 연결합니다.

그런 다음 AWS IoT 콘솔에서 Greengrass 그룹에 역할을 추가합니다.

> **Note**
>
> Greengrass 그룹에는 하나의 그룹 역할이 있습니다. 권한을 추가하려면 연결된 정책을 편집하거나 더 많은 정책을 연결할 수 있습니다.

이 튜토리얼에서는 Amazon DynamoDB 테이블에서 작업 설명, 생성 및 업데이트를 허용하는 권한 정책을 생성합니다. 그런 다음, 정책을 새 역할로 연결한 뒤 해당 역할을 사용자의 Greengrass 그룹으로 연결합니다.

먼저 이 모듈에서 Lambda 함수가 요구하는 권한을 부여하는 고객 관리형 정책을 생성합니다.

1. IAM 콘솔의 탐색 창에서 정책을 선택한 후 정책 생성을 선택하세요.

2. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다. 이 모듈의 Lambda 함수는 `CarStats`라는 이름이 지정된 DynamoDB 테이블을 생성 및 업데이트하는 해당 권한들을 사용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionsForModule6",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:CreateTable",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
        }
    ]
}
```

3. Next: Tags(다음: 태그)를 선택한 후 Next: Review(다음: 검토)를 선택합니다. 태그가 이 튜토리얼에서 사용되지 않습니다.
4. 이름의 경우, **greengrass_CarStats_Table**를 입력한 후 정책 생성을 선택합니다.

   그런 다음, 새 정책을 사용하는 역할을 생성합니다.

5. 탐색 창에서 역할(Roles)을 선택한 후 역할 생성(Create role)을 선택합니다.
6. 신뢰할 수 있는 엔터티 유형에서 AWS 서비스를 선택합니다.
7. 사용 사례, 기타 AWS 서비스 사용 사례에서 Greengrass를 선택하고, Greengrass를 선택한 후, 다음을 선택합니다.
8. 권한 정책에서 새 **greengrass_CarStats_Table** 정책을 선택한 후 다음을 선택합니다.
9. Role name(역할 이름)에 **Greengrass_Group_Role**을 입력합니다.
10. 설명(Description)에 **Greengrass group role for connectors and user-defined Lambda functions**를 입력합니다.
11. 역할 생성을 선택합니다.

    이제 Greengrass 그룹에 해당 역할을 연결합니다.

12. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
13. Greengrass 그룹에서 사용자 그룹을 선택합니다.
14. 설정을 선택한 후 역할 연결을 선택합니다.
15. 역할 목록에서 Greengrass_Group_Role을 선택한 다음 역할 연결을 선택합니다.

Lambda 함수 생성 및 구성

이 단계에서는 신호등을 통과하는 차량들의 수를 추적하는 Lambda 함수를 만듭니다. 이 Lambda 함수는 `GG_TrafficLight` 섀도우 상태가 `G`로 변경될 때마다 임의의 수의 자동차(1~20대)가 통과하는 것을 시뮬레이션합니다. 세 번째 `G` 불빛이 바뀔 때마다 Lambda 함수는 최소값 및 최대값과 같은 기본 통계를 DynamoDB 테이블로 전송합니다.

1. 컴퓨터에 `car_aggregator`라는 이름의 폴더를 하나 만듭니다.
2. GitHub의 TrafficLight 예제 폴더에서 `carAggregator.py` 파일을 `car_aggregator` 폴더로 다운로드합니다. 이 항목은 Lambda 함수 코드입니다.

   > **Note**
   >
   > 이 예제 Python 파일은 편의를 위해 AWS IoT Greengrass Core SDK 리포지토리에 저장되지만 AWS IoT Greengrass Core SDK를 사용하지 않습니다.

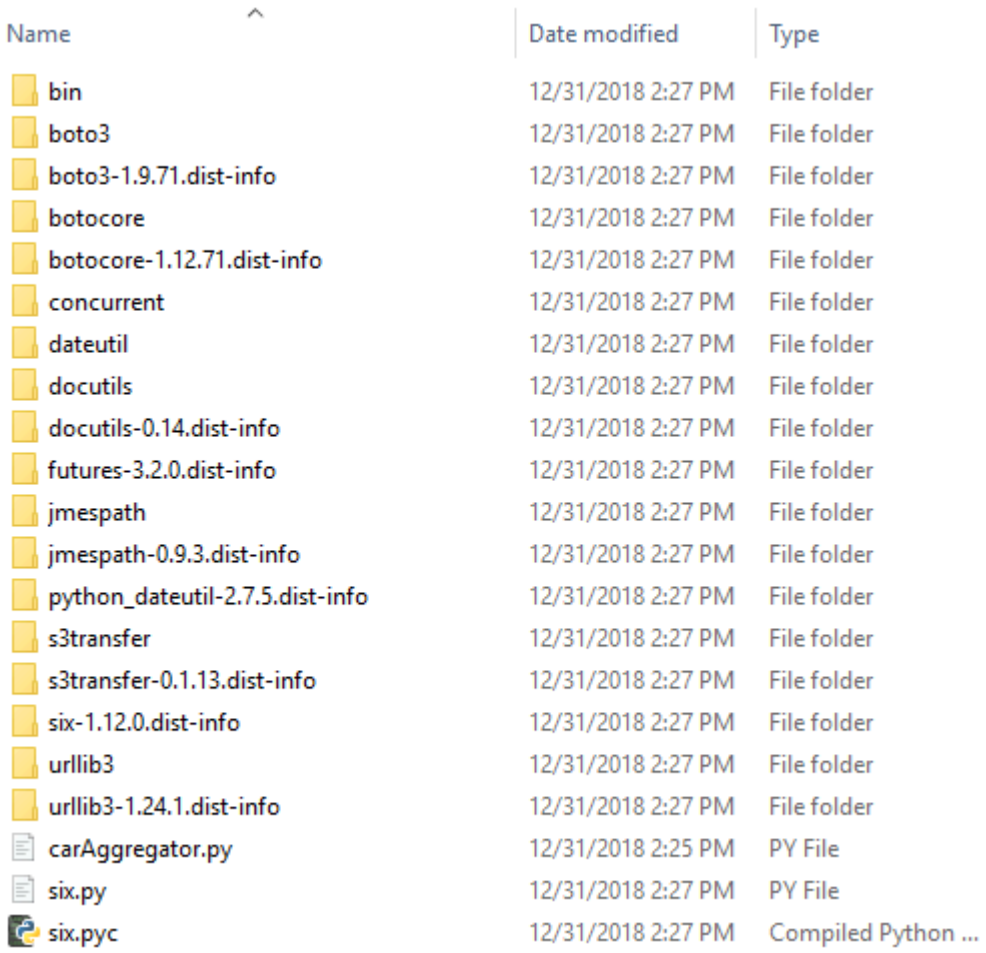
3. 미국 동부(버지니아 북부) 리전에서 근무하지 않는 경우 `carAggregator.py`를 열고 다음 줄의 `region_name`을 AWS IoT 콘솔에 현재 선택된 AWS 리전으로 변경합니다. 지원되는 AWS 리전 목록은 Amazon Web Services 일반 참조의 AWS IoT Greengrass을(를) 참조하십시오.

   ```
   dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
   ```

4. 명령줄 창에서 다음 명령을 실행하여 AWS SDK for Python (Boto3) 패키지와 종속 항목을 `car_aggregator` 폴더에 설치합니다. Greengrass Lambda 함수는 AWS SDK를 사용하여 다른 AWS 서비스에 액세스합니다. (Windows의 경우, 관리자 권한 명령 프롬프트를 사용합니다.)

   ```
   pip install boto3 -t path-to-car_aggregator-folder
   ```

   그러면 다음과 유사한 디렉터리 목록이 생성됩니다.

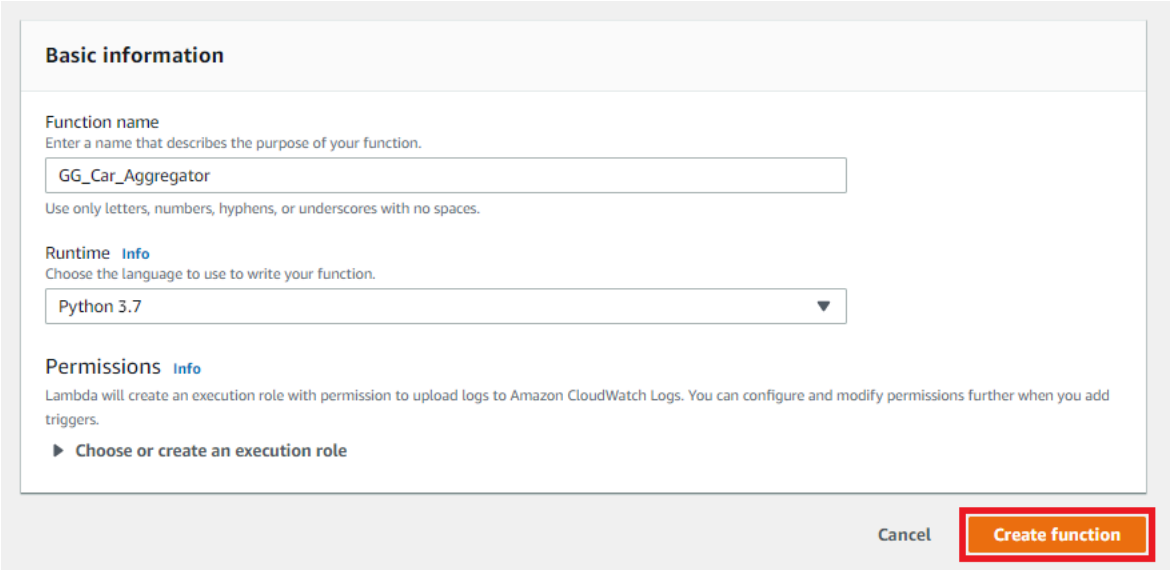


Name	Date modified	Type
bin	12/31/2018 2:27 PM	File folder
boto3	12/31/2018 2:27 PM	File folder
boto3-1.9.71.dist-info	12/31/2018 2:27 PM	File folder
botocore	12/31/2018 2:27 PM	File folder
botocore-1.12.71.dist-info	12/31/2018 2:27 PM	File folder
concurrent	12/31/2018 2:27 PM	File folder
dateutil	12/31/2018 2:27 PM	File folder
docutils	12/31/2018 2:27 PM	File folder
docutils-0.14.dist-info	12/31/2018 2:27 PM	File folder
futures-3.2.0.dist-info	12/31/2018 2:27 PM	File folder
jmespath	12/31/2018 2:27 PM	File folder
jmespath-0.9.3.dist-info	12/31/2018 2:27 PM	File folder
python_dateutil-2.7.5.dist-info	12/31/2018 2:27 PM	File folder
s3transfer	12/31/2018 2:27 PM	File folder
s3transfer-0.1.13.dist-info	12/31/2018 2:27 PM	File folder
six-1.12.0.dist-info	12/31/2018 2:27 PM	File folder
urllib3	12/31/2018 2:27 PM	File folder
urllib3-1.24.1.dist-info	12/31/2018 2:27 PM	File folder
carAggregator.py	12/31/2018 2:25 PM	PY File
six.py	12/31/2018 2:27 PM	PY File
six.pyc	12/31/2018 2:27 PM	Compiled Python ...

5. `car_aggregator` 폴더의 내용을 `car_aggregator.zip`이라는 이름의 `.zip` 파일로 압축합니다. (폴더의 내용(폴더가 아닌)을 압축합니다.) 이것이 Lambda 함수 배포 패키지입니다.
6. Lambda 콘솔에서 **`GG_Car_Aggregator`**(이)라는 함수를 생성하고 나머지 필드들을 다음과 같이 설정합니다.

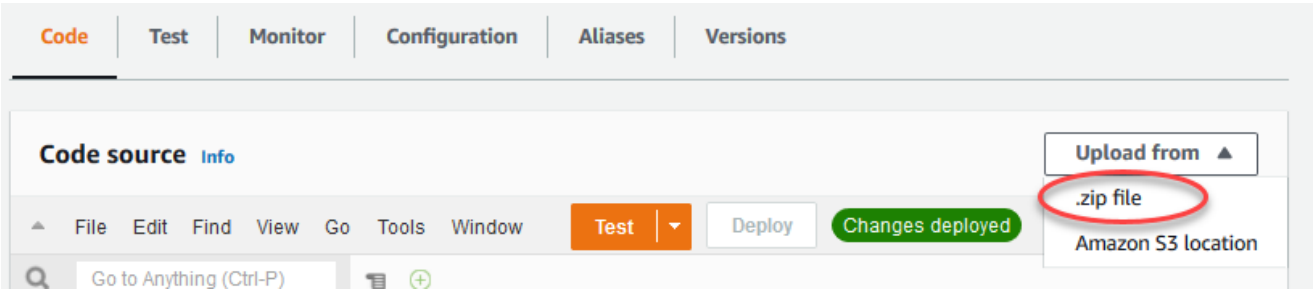
   - 실행 시간에서 Python 3.7을 선택합니다.
   - 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.

   함수 생성(Create function)을 선택합니다.



7. Lambda 함수 배포 패키지를 업로드합니다.

   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



   b. 업로드를 선택한 후 `car_aggregator.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.

   c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

      - 실행 시간에서 Python 3.7을 선택합니다.
      - 핸들러에 **`carAggregator.function_handler`**를 입력합니다.

   d. Save를 선택합니다.

8. Lambda 함수를 게시하고 **`GG_CarAggregator`**(이)라는 별칭을 생성합니다. 단계별 지침은 모듈 3 (1부)의 Lambda 함수 게시와 별칭 생성 단계를 참조하십시오.

9. AWS IoT 그룹에서 방금 생성한 Lambda 함수를 AWS IoT Greengrass 콘솔에 추가합니다.

   a. 그룹 구성 페이지에서 Lambda 함수를 선택한 다음 내 Lambda 함수에서 추가를 선택합니다.

   b. Lambda 함수의 경우 GG_CAR_Aggregator를 선택하십시오.

c. Lambda 함수 버전에서 게시한 버전에 대한 별칭을 선택합니다.

d. 메모리 제한에 **64 MB**를 입력합니다.

e. 고정된 경우 True를 선택합니다.

f. Lambda 함수 추가를 선택합니다.

> **Note**
>
> 이전 모듈의 다른 Lambda 함수를 제거해도 됩니다.

구독 구성

이 단계에서 GG_TrafficLight 섀도우가 업데이트된 상태 정보를 GG_Car_Aggregator Lambda 함수에 전송할 수 있게 해주는 구독을 생성합니다. 이 구독은 사용자가 모듈 5에서 생성한 구독들에 추가되며, 이 모두가 이 모듈에 필요한 것입니다.

1. 그룹 구성 페이지에서 구독 탭을 선택한 다음 추가를 선택합니다.
2. 구독 생성 페이지에서 다음을 수행합니다.

   a. 소스 유형의 경우, 서비스를 선택한 다음 로컬 섀도우 서비스를 선택합니다.

   b. 대상 유형에서 Lambda 함수를 선택한 다음 GG_Car_Aggregator을 선택합니다.

   c. 주제 필터에 **$aws/things/GG_TrafficLight/shadow/update/documents**를 입력합니다.

   d. 구독 생성을 선택합니다.

   이 모듈에서는 모듈 5에서 생성한 구독과 새 구독이 필요합니다.
3. 코어 디바이스로 클라우드 구성 배포 섹션에 설명된 대로 Greengrass 대몬(daemon)이 실행 중인지 확인합니다.
4. 그룹 구성 페이지에서 배포를 선택합니다.

통신 테스트

1. 컴퓨터에서 2개의 명령줄 창을 엽니다. 모듈 5에서처럼 창 하나는 GG_Switch 디바이스에, 다른 창 하나는 GG_TrafficLight 클라이언트 디바이스에 사용됩니다. 이러한 창을 사용하여 모듈 5에서 실행한 것과 동일한 명령을 실행합니다.

   GG_Switch 클라이언트 디바이스에 대해 다음 명령을 실행합니다.

   ```
   cd path-to-certs-folder
   python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
    --cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --
   thingName GG_TrafficLight --clientId GG_Switch
   ```

   GG_TrafficLight 클라이언트 디바이스에 대해 다음 명령을 실행합니다.

   ```
   cd path-to-certs-folder
   python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
   cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName
    GG_TrafficLight --clientId GG_TrafficLight
   ```

   스위치는 20초마다 섀도우 상태를 G, Y 및 R로 업데이트하고, 전등도 새로운 상태를 표시합니다.
2. 세 번째 녹색 불빛이 들어올 때마다(3분마다) Lambda 함수의 함수 핸들러가 트리거되고 새 레코드가 생성됩니다. `lightController.py` 및 `trafficLight.py`이 3분 동안 실행된 후 AWS Management Console으로 이동하여 DynamoDB 콘솔을 엽니다.
3. AWS 리전 메뉴에서 US East (N. Virginia)를 선택합니다. 이는 `GG_Car_Aggregator` 함수가 테이블을 생성하는 리전입니다.
4. 탐색 창에서 테이블을 선택한 다음 CarStats 테이블을 선택합니다.
5. 항목 보기를 선택하여 테이블의 항목을 확인합니다.

   통과한 차에 대한 기본 통계가 포함된 항목이 보여야 합니다(3분마다 항목 하나). 테이블에 대한 업데이트를 보려면 새로 고침 버튼을 선택해야 할 수도 있습니다.
6. 테스트가 실패하면 Greengrass 로그에서 문제 해결 정보를 찾을 수 있습니다.

   a. 루트 사용자로 전환하고 `log` 디렉터리로 이동합니다. AWS IoT Greengrass 로그 액세스에는 루트 권한이 필요합니다.

      ```
      sudo su
      ```

```
cd /greengrass/ggc/var/log
```

b. runtime.log에서 오류가 있는지 확인합니다.

```
cat system/runtime.log | grep 'ERROR'
```

c. Lambda 함수에 의해 생성되는 로그를 확인합니다.

```
cat user/region/account-id/GG_Car_Aggregator.log
```

lightController.py 및 trafficLight.py 스크립트는 스크립트와 동일한 폴더에 생성되는 groupCA 폴더에 연결 정보를 저장합니다. 연결 오류를 수신하는 경우 ggc-host 파일의 IP 주소가 이 단계에서 코어에 대해 구성한 단일 IP 주소 엔드포인트와 일치하는지 확인합니다.

자세한 내용은 문제 해결 섹션을 참조하세요.

기본 자습서의 마지막 부분입니다. 이제 AWS IoT Greengrass 프로그래밍 모델을 이해하고 AWS IoT Greengrass 코어, 그룹, 구독, 디바이스, 및 엣지에서 실행 중인 Lambda 함수의 배포 프로세스를 비롯한 기본적인 개념을 이해해야 합니다.

DynamoDB 표와 Greengrass Lambda 함수 및 구독을 삭제할 수 있습니다. AWS IoT Greengrass 코어 디바이스와 AWS IoT 클라우드 간의 통신을 중지하려면 코어 디바이스에서 터미널을 열고 다음 명령 중 하나를 실행합니다.

- AWS IoT Greengrass 코어 디바이스를 종료할 경우:

```
sudo halt
```

- AWS IoT Greengrass 대몬을 종료할 경우:

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
```

모듈 7: 하드웨어 보안 통합 시뮬레이션

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

이 고급 모듈에서는 Greengrass 코어와 함께 사용하도록 시뮬레이션된 하드웨어 보안 모듈(HSM)을 구성하는 방법을 보여줍니다. 이 구성에서 사용되는 SoftHSM은 PKCS#11 애플리케이션 프로그래밍 인터페이스(API)를 사용하는 일반 소프트웨어 구현입니다. 이 모듈의 목적은 PKCS#11 API의 소프트웨어 전용 구현을 배우고 초기 테스트를 수행할 수 있는 환경을 설정하도록 하는 것입니다. 이 모듈은 학습과 초기 테스트를 위해서만 제공되지 어떠한 종류의 실제 사용 용도로도 제공되지 않습니다.

이 구성을 통해 PKCS#11 호환 서비스를 사용하여 프라이빗 키를 저장하는 실험을 할 수 있습니다. 소프트웨어전용 구현에 대한 자세한 내용은 SoftHSM을 참조하십시오. 일반적인 요구 사항을 포함해 AWS IoT Greengrass 코어에 대한 하드웨어 보안 통합에 대한 자세한 내용은 the section called “하드웨어 보안 통합”를 참조하십시오.

> **Important**
>
> 이 모듈은 실험 전용으로만 제공됩니다. 잘못된 보안 감지가 늘어날 수 있기 때문에 프로덕션 환경에서는 SoftHSM을 사용하지 않는 것이 좋습니다. 이러한 구성은 실제 보안 이점을 전혀 제공하지 않습니다. SoftHSM에 저장되는 키는 Greengrass 환경에서 보안 암호를 저장하는 다른 수단보다 안전하게 저장되지 않습니다.
> 이 모듈의 목적은 앞으로 실제 하드웨어 기반 HSM을 사용할 계획이 있는 경우 PKCS#11 사양에 대해 배우고 소프트웨어의 초기 테스트를 수행할 수 있도록 하는 것입니다.
> SoftHSM에서 제공하는 PKCS#11 구현과 하드웨어 기반 구현에는 차이가 있을 수 있기 때문에 프로덕션 사용 전에 향후 하드웨어 구현을 별도로 철저하게 테스트해야 합니다.

지원되는 하드웨어 보안 모듈 온보딩과 관련하여 도움이 필요한 경우 AWS 엔터프라이즈 지원 담당자에게 문의하십시오.

시작하기 전에 Greengrass 디바이스 설정 스크립트를 실행하거나 시작하기 자습서의 모듈 1 및 모듈 2를 완료했는지 확인합니다. 이 모듈에서는 코어가 이미 프로비저닝되어 있고 AWS와 통신한다고 가정합니다. 이 모듈을 완료하는 데 약 30분 정도 걸립니다.

SoftHSM 소프트웨어 설치

이 단계에서는 SoftHSM을 설치하고, SoftHSM 인스턴스를 관리하는 데 사용되는 pkcs11 도구를 설치합니다.

- AWS IoT Greengrass 코어 디바이스의 터미널에서 다음 명령을 실행합니다.

```
sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
```

이러한 패키지에 대한 자세한 내용은 Install softhsm2, Install libsofthsm2-dev 및 Install pkcs11-dump를 참조하십시오.

> **Note**
>
> 시스템에서 이 명령을 사용할 때 문제가 발생하면 GitHub에서 SoftHSM 버전 2를 참조하십시오. 이 사이트에서는 소스에서 빌드하는 방법을 비롯하여 설치 정보를 더욱 다양하게 제공합니다.

SoftHSM 구성

이 단계에서는 SoftHSM을 구성합니다.

1. 루트 사용자로 전환합니다.

```
sudo su
```

2. 매뉴얼 페이지를 사용하여 시스템 전체에서 `softhsm2.conf` 위치를 찾습니다. 공통 위치는 `/etc/softhsm/softhsm2.conf`이며, 일부 시스템에서는 이 위치가 다를 수 있습니다.

```
man softhsm2.conf
```

3. 시스템 전체 위치에서 softhsm2 구성 파일의 디렉터리를 생성합니다. 이 예에서는 위치가 `/etc/softhsm/softhsm2.conf`라고 가정합니다.

```
mkdir -p /etc/softhsm
```

4. `/greengrass` 디렉터리에서 토큰 디렉터리를 생성합니다.

> **Note**
>
> 이 단계를 건너뛰면 softhsm2-util에서 `ERROR: Could not initialize the library`를 보고합니다.

```
mkdir -p /greengrass/softhsm2/tokens
```

5. 토큰 디렉터리를 구성합니다.

```
echo "directories.tokendir = /greengrass/softhsm2/tokens" > /etc/softhsm/
softhsm2.conf
```

6. 파일 기반 백엔드를 구성합니다.

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

> **Note**
>
> 이러한 구성 설정은 실험 전용입니다. 모든 구성 옵션을 보려면 해당 구성 파일의 매뉴얼 페이지를 읽어보십시오.
>
> ```
> man softhsm2.conf
> ```

SoftHSM으로 프라이빗 키 가져오기

이 단계에서는 SoftHSM 토큰을 초기화하고, 프라이빗 키 형식을 변환한 다음 프라이빗 키를 가져옵니다.

1. SoftHSM 토큰을 초기화합니다.

```
softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
```

> **Note**
>
> 메시지가 표시되면 SO 핀(12345)과 사용자 핀(1234)을 입력합니다. AWS IoT Greengrass에서는 SO(감독자) 핀을 사용하지 않으므로 임의의 값을 사용할 수 있습니다. `CKR_SLOT_ID_INVALID: Slot 0 does not exist` 오류가 발생하면 다음 명령을 대신 시도하십시오.
>
> ```
> softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin
>  1234
> ```

2. 프라이빗 키를 SoftHSM 가져오기 도구에서 사용할 수 있는 형식으로 변환합니다. 이 튜토리얼에서는 시작하기 튜토리얼의 모듈 2에 있는 기본 그룹 생성 옵션에서 얻은 개인 키를 변환합니다.

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -
out hash.private.pem
```

3. SoftHSM으로 프라이빗 키를 가져옵니다. softhsm2-util 버전에 따라 다음 명령 중 하나만 실행합니다.

   Raspbian softhsm2-util v2.2.0 구문

   ```
   softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id
    0000 --pin 12340
   ```

   Ubuntu softhsm2-util v2.0.0 구문

   ```
   softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin
    1234
   ```

   이 명령은 슬롯을 `0`으로 식별하고, 키 레이블을 `iotkey`로 정의합니다. 이러한 값은 다음 섹션에서 사용합니다.

프라이빗 키를 가져온 다음에는 `/greengrass/certs` 디렉터리에서 선택적으로 제거할 수 있습니다. 루트 CA 및 디바이스 인증서는 이 디렉터리에 보관합니다.

SoftHSM을 사용하도록 Greengrass 코어 구성

이 단계에서는 SoftHSM을 사용하도록 Greengrass 코어 구성 파일을 수정합니다.

1. 시스템에서 SoftHSM 공급자 라이브러리(`libsofthsm2.so`) 경로를 찾습니다.

   a. 이 라이브러리에 대해 설치된 패키지 목록을 가져옵니다.

   ```
   sudo dpkg -L libsofthsm2
   ```

   `libsofthsm2.so` 파일은 `softhsm` 디렉터리에 있습니다.

   b. 파일의 전체 경로를 복사합니다(예: `/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so`). 이 값은 나중에 사용합니다.

2. Greengrass 데몬을 중지합니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
```

3. Greengrass 구성 파일을 엽니다. config.json 파일은 /greengrass/config 디렉터리에 있습니다.

> **Note**
>
> 이 절차의 예제는 config.json 파일이 시작하기 튜토리얼의 모듈 2에 있는 기본 그룹 생성 옵션에서 생성된 형식을 사용한다는 가정 하에 작성되었습니다.

4. crypto.principals 객체에서 다음 MQTT 서버 인증서 객체를 삽입합니다. 유효한 JSON 파일을 생성하는 데 필요한 경우 쉼표를 추가합니다.

```
  "MQTTServerCertificate": {
    "privateKeyPath": "path-to-private-key"
  }
```

5. crypto 객체에서 다음 PKCS11 객체를 삽입합니다. 유효한 JSON 파일을 생성하는 데 필요한 경우 쉼표를 추가합니다.

```
  "PKCS11": {
    "P11Provider": "/path-to-pkcs11-provider-so",
    "slotLabel": "crypto-token-name",
    "slotUserPin": "crypto-token-user-pin"
  }
```

파일은 다음과 같아야 합니다.

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
```

```
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

> **Note**
>
> 하드웨어 보안에 무선(OTA) 업데이트를 사용하려면 PKCS11 개체에도 OpenSSLEngine 속성이 포함되어야 합니다. 자세한 내용은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.

6. 다음과 같이 `crypto` 객체를 편집합니다.

   a. PKCS11 객체를 구성합니다.

      - `P11Provider`에 `libsofthsm2.so`의 전체 경로를 입력합니다.
      - `slotLabel`에 `greengrass`을 입력합니다.
      - `slotUserPin`에 `1234`을 입력합니다.

b. `principals` 객체에 프라이빗 키 경로를 구성합니다. `certificatePath` 속성은 편집하지 마십시오.

   - `privateKeyPath` 속성에 다음 RFC 7512 PKCS#11 경로(키의 레이블을 지정하는 경로)를 입력합니다. `IoTCertificate`, `SecretsManager` 및 `MQTTServerCertificate` 보안 주체에 대해 동일하게 수행합니다.

     ```
     pkcs11:object=iotkey;type=private
     ```

c. `crypto` 객체를 확인합니다. 예를 들면 다음과 같아야 합니다.

```
"crypto": {
  "PKCS11": {
    "P11Provider": "/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so",
    "slotLabel": "greengrass",
    "slotUserPin": "1234"
  },
  "principals": {
    "MQTTServerCertificate": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "SecretsManager": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "IoTCertificate": {
      "certificatePath": "file://certs/core.crt",
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    }
  },
  "caPath": "file://certs/root.ca.pem"
}
```

7. `coreThing` 객체에서 `caPath`, `certPath` 및 `keyPath` 값을 제거합니다. 예를 들면 다음과 같아야 합니다.

```
"coreThing" : {
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
  "keepAlive" : 600
}
```

> **Note**
>
> 이 자습서에서는 모든 보안 주체에 대해 동일한 프라이빗 키를 지정합니다. 로컬 MQTT 서버에 대한 프라이빗 키 선택에 대한 자세한 내용은 성능을 참조하십시오. 로컬 보안 암호 관리자에 대한 자세한 내용은 코어에 암호 배포 단원을 참조하십시오.

구성 테스트

- Greengrass 데몬을 시작합니다.

  ```
  cd /greengrass/ggc/core/
  sudo ./greengrassd start
  ```

  데몬이 성공적으로 시작하면 코어가 제대로 구성된 것입니다.

  이제 PKCS#11 사양에 대해 배우고 SoftHSM 구현에서 제공하는 PKCS#11 API를 사용해 초기 테스트를 수행할 준비가 되었습니다.

  > **Important**
  >
  > 다시 말하지만, 이 모듈은 학습 및 테스트만을 위한 모듈입니다. 이 모듈이 Greengrass 환경의 보안 상태를 실제로 강화하지는 않습니다.
  > 대신 이 모듈의 용도는 앞으로 실제 하드웨어 기반 HSM을 사용할 것에 대비해 학습 및 테스트를 시작하도록 하는 데 있습니다. 현재, SoftHSM에서 제공하는 PKCS#11 구현과 하드웨어 기반 구현 사이에는 차이가 있을 수 있으므로 실제 사용 전에 하드웨어 기반 HSM에 대해 소프트웨어를 별도로 철저하게 테스트해야 합니다.

다음 사항도 참조하세요.

- PKCS #11 암호화 토큰 인터페이스 사용 설명서 버전 2.40. John Leiseboer 및 Robert Griffin 편집. 2014년 11월 16일 OASIS Committee Note 02. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. 최신 버전: http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
- RFC 7512

AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트

AWS IoT Greengrass 코어 소프트웨어 패키지에는 AWS IoT Greengrass 소프트웨어의 무선(OTA) 업데이트를 수행할 수 있는 업데이트 에이전트가 포함되어 있습니다. OTA 업데이트를 사용하여 하나 이상의 코어에 최신 버전의 AWS IoT Greengrass 코어 소프트웨어 또는 를 설치할 수 있습니다. OTA 업데이트를 사용하면 코어 디바이스가 물리적으로 존재할 필요가 없습니다.

가능하면 OTA 업데이트를 사용하는 것이 좋습니다. 업데이트 상태 및 업데이트 기록을 추적하는 데 사용할 수 있는 메커니즘을 제공합니다. 실패한 업데이트가 발생하면 OTA 업데이트 에이전트는 이전 소프트웨어 버전으로 롤백합니다.

> **Note**
>
> `apt`를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치할 때는 OTA 업데이트가 지원되지 않습니다. 이러한 설치의 경우 `apt`를 사용하여 소프트웨어를 업그레이드하는 것이 좋습니다. 자세한 내용은 the section called “APT 리포지토리에서 설치” 섹션을 참조하세요.

OTA 업데이트를 통해 다음을 보다 효율적으로 수행할 수 있습니다.

- 보안 취약성을 수정합니다.
- 소프트웨어 안정성 문제를 해결합니다.
- 새 기능 또는 향상된 기능을 배포합니다.

이 기능은 AWS IoT 작업과 통합됩니다.

요구 사항

다음 요구 사항은 AWS IoT Greengrass 소프트웨어의 OTA 업데이트에 적용됩니다.

- Greengrass 코어의 로컬 스토리지에는 최소 400MB의 디스크 공간이 있어야 합니다. OTA 업데이트 에이전트에는 AWS IoT Greengrass 코어 소프트웨어의 런타임 사용 요구 사항의 약 3배가 필요합니다. 자세한 내용은 Amazon Web Services 일반 참조의 Amazon EBS에 대한 서비스 할당량을 참조하세요.
- Greengrass 코어는 AWS 클라우드 클라우드와 연결되어 있어야 합니다.

- Greengrass 코어는 AWS IoT Core 및 AWS IoT Greengrass를 사용하여 인증하기 위한 인증서와 키로 올바르게 구성 및 프로비저닝되어야 합니다. 자세한 내용은 the section called “X.509 인증서” 섹션을 참조하세요.
- Greengrass 코어는 네트워크 프록시를 사용하도록 구성할 수 없습니다.

  > **Note**
  >
  > AWS IoT Greengrass v1.9.3부터는 MQTT 트래픽이 기본 포트 8883 대신 포트 443을 사용하도록 구성하는 코어에서 OTA 업데이트가 지원됩니다. 그러나 OTA 업데이트 에이전트는 네트워크 프록시를 지원하지 않습니다. 자세한 내용은 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.

- AWS IoT Greengrass 코어 소프트웨어가 포함된 파티션에서는 신뢰할 수 있는 부팅을 활성화할 수 없습니다.

  > **Note**
  >
  > 신뢰할 수 있는 부팅이 활성화된 파티션에 AWS IoT Greengrass 코어 소프트웨어를 설치하고 실행할 수 있지만 OTA 업데이트는 지원되지 않습니다.

- AWS IoT Greengrass에는 AWS IoT Greengrass 코어 소프트웨어가 포함된 파티션에 대한 읽기/쓰기 권한이 있어야 합니다.
- init 시스템을 사용하여 Greengrass 코어를 관리하는 경우 init 시스템과 통합되도록 OTA 업데이트를 구성해야 합니다. 자세한 내용은 the section called “Init 시스템과의 통합” 섹션을 참조하세요.
- AWS IoT Greengrass 소프트웨어 업데이트 아티팩트에 Amazon S3 URL을 미리 서명하는 데 사용되는 역할을 만들어야 합니다. 이 AWS IoT Core 서명자 역할을 통해 Amazon S3가 사용자를 대신하여 에 저장된 소프트웨어 업데이트 아티팩트에 액세스할 수 있습니다. 자세한 내용은 the section called “OTA 업데이트에 대한 IAM 권한” 섹션을 참조하세요.

OTA 업데이트에 대한 IAM 권한

AWS IoT Greengrass에서 새 버전의 AWS IoT Greengrass 코어 소프트웨어를 AWS IoT Greengrass 릴리스하면 에서 OTA 업데이트에 사용되는 Amazon S3에 저장된 소프트웨어 아티팩트를 업데이트합니다.

AWS 계정 계정에 이러한 아티팩트에 액세스하는 데 사용할 수 있는 Amazon S3 URL URL 서명자 역할이 포함되어야 합니다. 역할에는 대상 AWS 리전 리전의 버킷에 대한 `s3:GetObject` 작업을 허용

하는 권한 정책이 있어야 합니다. 역할에는 `iot.amazonaws.com`이 신뢰할 수 있는 엔터티로 역할을 수임할 수 있는 신뢰 정책도 있어야 합니다.

권한 정책

역할 권한의 경우 AWS 관리형 정책을 사용하거나 사용자 지정 정책을 생성할 수 있습니다.

- AWS 관리형 정책 사용

  GreengrassOTAUpdateArtifactAccess 관리형 정책은 AWS IoT Greengrass에서 제공합니다. 현재 및 향후 AWS IoT Greengrass에서 지원하는 모든 Amazon Web Services 리전에서 액세스를 허용하려면 이 정책을 사용합니다.

- 사용자 지정 정책 생성

  코어가 배포되는 Amazon Web Services 리전을 명시적으로 지정하려면 사용자 지정 정책을 생성해야 합니다. 다음 예제 정책에서는 6개의 리전에서 AWS IoT Greengrass 소프트웨어 업데이트에 대한 액세스를 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::us-east-1-greengrass-updates/*",
                "arn:aws:s3:::us-west-2-greengrass-updates/*",
                "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
                "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
                "arn:aws:s3:::eu-central-1-greengrass-updates/*",
                "arn:aws:s3:::eu-west-1-greengrass-updates/*"
            ]
        }
    ]
}
```

신뢰 정책

역할에 연결된 신뢰 정책은 `sts:AssumeRole` 작업을 허용하고 `iot.amazonaws.com`을 보안 주체로 정의해야 합니다. 이를 통해 AWS IoT Core가 신뢰할 수 있는 엔터티로 역할을 수임할 수 있습니다. 다음은 정책 문서의 예입니다.

```
{
     "Version": "2012-10-17",
     "Statement": [
         {
             "Sid": "AllowIotToAssumeRole",
             "Action": "sts:AssumeRole",
             "Principal": {
                  "Service": "iot.amazonaws.com"
             },
             "Effect": "Allow"
        }
    ]
}
```

또한 OTA 업데이트를 시작하는 사용자는 `greengrass:CreateSoftwareUpdateJob` 및 `iot:CreateJob`, `iam:PassRole`을 사용하여 서명자 역할의 권한을 전달할 권한이 있어야 합니다. 다음은 IAM 정책의 예입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "greengrass:CreateSoftwareUpdateJob"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn-of-s3-url-signer-role"
        }
    ]
}
```

고려 사항

Greengrass 코어 소프트웨어의 OTA 업데이트를 시작하기 전에 코어 디바이스와 해당 코어에 로컬로 연결된 클라이언트 디바이스 모두에서 Greengrass 그룹의 디바이스에 미치는 영향을 알고 있어야 합니다.

- 업데이트 중에 코어가 종료됩니다.
- 코어에서 실행 중인 모든 Lambda 함수가 종료됩니다. 해당 함수가 로컬 리소스에 쓸 경우 제대로 종료하지 않으면 해당 리소스가 잘못된 상태가 될 수 있습니다.
- 코어가 가동 중지된 동안에는 AWS 클라우드 클라우드와의 모든 연결이 끊어집니다. 클라이언트 디바이스에서 코어를 통해 라우팅하는 메시지가 손실됩니다.
- 자격 증명 캐시가 손실됩니다.
- Lambda 함수에 대한 보류 중인 작업이 있는 대기열이 손실됩니다.
- 수명이 긴 Lambda 함수는 동적 상태 정보를 잃고 보류 중인 작업이 모두 삭제됩니다.

OTA 업데이트 중에 다음 상태 정보가 유지됩니다.

- 코어 구성
- Greengrass 그룹 구성
- 로컬 섀도우
- Greengrass 로그
- OTA 업데이트 에이전트 로그

Greengrass OTA 업데이트 에이전트

Greengrass OTA 업데이트 에이전트는 클라우드에서 생성 및 배포된 업데이트 작업을 처리하는 디바이스의 소프트웨어 구성 요소입니다. OTA 업데이트 에이전트는 AWS IoT Greengrass 코어 소프트웨어와 동일한 소프트웨어 패키지로 배포됩니다. 에이전트는 */greengrass-root*/ota/ota_agent/ggc-ota에 있습니다. /var/log/greengrass/ota/ggc_ota.txt에 로그를 씁니다.

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

바이너리를 수동으로 실행하거나 systemd 서비스 파일과 같은 init 스크립트의 일부로 통합하여 OTA 업데이트 에이전트를 시작할 수 있습니다. 바이너리를 수동으로 실행하는 경우 루트로 실행해야 합니다. 바이너리가 시작되면 OTA 업데이트 에이전트가 AWS IoT Core에서 AWS IoT Greengrass 소프트웨어 업데이트 작업을 수신하여 순차적으로 실행합니다. AWS IoT OTA 업데이트 에이전트는 다른 IoT 작업 유형을 모두 무시합니다.

다음 발췌문은 OTA 업데이트 에이전트를 시작, 중지 및 재시작하는 systemd 서비스 파일의 예를 보여줍니다.

```
[Unit]
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota

[Install]
WantedBy=multi-user.target
```

업데이트의 대상인 코어는 OTA 업데이트 에이전트의 두 인스턴스를 실행해서는 안 됩니다. 그러면 두 에이전트가 동일한 작업을 처리하게 되어 충돌이 발생합니다.

Init 시스템과의 통합

OTA 업데이트 중에 OTA 업데이트 에이전트가 코어에서 바이너리를 다시 시작합니다. 바이너리가 실행 중인 경우 init 시스템에서 업데이트 중에 AWS IoT Greengrass 코어 소프트웨어 또는 에이전트의 상태를 모니터링하는 경우 충돌이 발생할 수 있습니다. OTA 업데이트 메커니즘을 사용자의 init 모니터링 전략과 통합하기 위해 업데이트 전후에 실행되는 셸 스크립트를 쓸 수 있습니다. 예를 들어 디바이스를 종료하기 전에 데이터를 백업하거나 프로세스를 중지하는 `ggc_pre_update.sh` 스크립트를 쓸 수 있습니다.

이러한 셸 스크립트를 실행하도록 OTA 업데이트 에이전트에 지시하려면 config.json 파일에 `"managedRespawn" : true` 플래그를 포함시켜야 합니다. 이 설정은 다음 발췌문에 표시됩니다.

```
{
   "coreThing": {
       …
   },
   "runtime": {
       …
   },
   "managedRespawn": true
       …
}
```

OTA 업데이트가 지원되는 관리형 Respawn

`managedRespawn`이(가) `true`(으)로 설정된 OTA 업데이트에는 다음 요구 사항이 적용됩니다.

- */greengrass-root*`/usr/scripts` 디렉터리에 다음 쉘 스크립트가 있어야 합니다.
  - `ggc_pre_update.sh`
  - `ggc_post_update.sh`
  - `ota_pre_update.sh`
  - `ota_post_update.sh`
- 스크립트가 성공적인 반환 코드를 반환해야 합니다.
- 스크립트는 루트가 소유하고 루트에 의해서만 실행 가능해야 합니다.
- `ggc_pre_update.sh` 스크립트는 Greengrass 대몬(daemon)을 중지해야 합니다.
- `ggc_post_update.sh` 스크립트는 Greengrass 대몬(daemon)을 시작해야 합니다.

> **Note**
>
> OTA 업데이트 에이전트가 자체 프로세스를 관리하므로 `ota_pre_update.sh` 및 `ota_post_update.sh` 스크립트는 OTA 서비스를 중지하거나 시작할 필요가 없습니다.

OTA 업데이트 에이전트는 */greengrass-root*`/usr/scripts`에서 스크립트를 실행합니다. 디렉터리 트리는 다음과 같아야 합니다.

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
|   |-- ota_pre_update.sh
|   |-- ota_post_update.sh
|-- ota
```

`managedRespawn`이 `true`로 설정된 경우 OTA 업데이트 에이전트는 소프트웨어 업데이트 전후에 해당 스크립트에 대한 */greengrass-root*`/usr/scripts` 디렉토리를 확인합니다. 스크립트가 없으면 업데이트가 실패합니다. AWS IoT Greengrass은(는) 이러한 스크립트의 내용을 확인하지 않습니다. 가장 좋은 방법은 스크립트가 제대로 작동하는지 확인하고 오류에 대한 적절한 종료 코드를 발행하는 것입니다.

AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트의 경우:

- 업데이트를 시작하기 전에 에이전트는 `ggc_pre_update.sh` 스크립트를 실행합니다. OTA 업데이트 에이전트가 AWS IoT Greengrass Core 소프트웨어 업데이트를 시작하기 전에 실행해야 하는 명령(예: 데이터 백업 또는 실행 중인 프로세스 중지)에는 이 스크립트를 사용하십시오. 다음 예제는 Greengrass 대몬(daemon)을 중지하는 간단한 스크립트를 보여줍니다.

  ```
  #!/bin/bash
  set -euo pipefail
  systemctl stop greengrass
  ```

- 업데이트를 완료한 후 에이전트는 `ggc_post_update.sh` 스크립트를 실행합니다. OTA 업데이트 에이전트가 AWS IoT Greengrass Core 소프트웨어 업데이트를 시작한 후에 실행해야 하는 명령(예:

프로세스 재시작)에는 이 스크립트를 사용하십시오. 다음 예제는 Greengrass 대몬(daemon)을 시작하는 간단한 스크립트를 보여줍니다.

```
#!/bin/bash
set -euo pipefail
systemctl start greengrass
```

OTA 업데이트 에이전트의 OTA 업데이트의 경우:

- 업데이트를 시작하기 전에 에이전트는 `ota_pre_update.sh` 스크립트를 실행합니다. OTA 업데이트 에이전트가 자체적으로 업데이트하기 전에 실행해야 하는 명령(예: 데이터 백업 또는 실행 중인 프로세스 중지)에는 이 스크립트를 사용하십시오.
- 업데이트를 완료한 후 에이전트는 `ota_post_update.sh` 스크립트를 실행합니다. OTA 업데이트 에이전트가 자체적으로 업데이트한 후 실행해야 하는 명령(예: 프로세스 재시작)에는 이 스크립트를 사용하십시오.

> **Note**
>
> `managedRespawn`이 `false`로 설정된 경우 OTA 업데이트 에이전트에서 스크립트를 실행하지 않습니다.

OTA 업데이트 생성

하나 이상의 코어에서 AWS IoT Greengrass 소프트웨어의 OTA 업데이트를 수행하려면 다음 단계를 따르십시오.

1. 코어가 OTA 업데이트에 대한 요구 사항을 충족하는지 확인하십시오.

   > **Note**
   >
   > AWS IoT Greengrass 코어 소프트웨어 또는 OTA 업데이트 에이전트를 관리하도록 init 시스템을 구성한 경우 코어에서 다음을 확인하십시오.
   >
   > - config.json 파일은 `"managedRespawn" : true`를 지정합니다.
   > - */greengrass-root*/usr/scripts 디렉토리에는 다음 스크립트가 포함되어 있습니다.
   >   - ggc_pre_update.sh

- ggc_post_update.sh
- ota_pre_update.sh
- ota_post_update.sh

자세한 내용은 the section called “Init 시스템과의 통합” 섹션을 참조하세요.

2. 코어 디바이스 터미널에서 OTA 업데이트 에이전트를 시작하십시오.

```
cd /greengrass-root/ota/ota_agent
sudo ./ggc-ota
```

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

충돌을 일으킬 수 있으므로 코어에서 OTA 업데이트 에이전트의 여러 인스턴스를 시작하지 마십시오.

3. 또는 AWS IoT Greengrass API를 사용하여 소프트웨어 업데이트 작업을 생성합니다.

   a. CreateSoftwareUpdateJob API를 호출합니다. 이 예제 절차에서는 AWS CLI 명령을 사용합니다.

   다음 명령은 하나의 코어에서 AWS IoT Greengrass 코어 소프트웨어를 업데이트하는 작업을 생성합니다. 예제 값을 바꾼 다음 명령을 실행합니다.

   Linux or macOS terminal

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

Windows command prompt

```
aws greengrass create-software-update-job ^
--update-targets-architecture x86_64 ^
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] ^
--update-targets-operating-system ubuntu ^
--software-to-update core ^
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^
--update-agent-log-level WARN ^
--amzn-client-token myClientToken1
```

이 명령은 다음 응답을 반환합니다.

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.10.1"
}
```

b. 응답에서 `IoTJobId`를 복사합니다.

c. AWS IoT Core API에서 DescribeJob을 직접적으로 호출하면 작업 상태를 확인할 수 있습니다. 예제 값을 작업 ID로 바꾼 다음 명령을 실행합니다.

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-
a1da0EXAMPLE
```

이 명령은 `status` 및 `jobProcessDetails`를 포함하여 작업에 대한 정보가 포함된 응답 객체를 반환합니다.

```
{
    "job": {
        "jobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "targetSelection": "SNAPSHOT",
        "status": "IN_PROGRESS",
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myCoreDevice"
```

```
        ],
        "description": "This job was created by Greengrass to update the
 Greengrass Cores in the targets with version 1.10.1 of the core software
 running on x86_64 architecture.",
        "presignedUrlConfig": {
            "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
            "expiresInSec": 3600
        },
        "jobExecutionsRolloutConfig": {},
        "createdAt": 1588718249.079,
        "lastUpdatedAt": 1588718253.419,
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfFailedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfQueuedThings": 1,
            "numberOfInProgressThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfTimedOutThings": 0
        },
        "timeoutConfig": {}
    }
}
```

문제 해결에 대한 도움말은 문제 해결 단원을 참조하십시오.

CreateSoftwareUpdateJob API

`CreateSoftwareUpdateJob` API를 사용하여 코어 디바이스의 AWS IoT Greengrass 코어 소프트웨어 또는 OTA 업데이트 에이전트 소프트웨어를 업데이트할 수 있습니다. 이 API는 업데이트를 사용할 수 있을 때 디바이스에 알리는 AWS IoT 스냅샷 작업을 생성합니다. `CreateSoftwareUpdateJob`을 호출한 후 다른 AWS IoT 작업 명령을 사용하여 소프트웨어 업데이트를 추적할 수 있습니다. 자세한 내용은 AWS IoT 개발자 가이드의 작업 섹션을 참조하세요.

다음 예제에서는 코어 디바이스에서 AWS CLI를 사용하여 AWS IoT Greengrass 코어 소프트웨어 업데이트 작업을 생성하는 방법을 보여줍니다.

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
```

```
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

`create-software-update-job` 명령은 작업 ID, 작업 ARN, 업데이트가 설치하는 소프트웨어 버전을 포함하는 JSON 응답을 반환합니다.

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-
ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.9.2"
}
```

`create-software-update-job`을 사용하여 코어 디바이스를 업데이트하는 방법을 보여 주는 단계는 the section called “OTA 업데이트 생성” 단원을 참조하십시오.

`create-software-update-job` 명령에는 다음 파라미터가 있습니다.

`--update-targets-architecture`

코어 디바이스의 아키텍처입니다.

유효한 값: `armv7l`, `armv6l`, `x86_64` 또는 `aarch64`

`--update-targets`

업데이트할 코어입니다. 목록에는 개별 코어의 ARN과 코어로 구성된 사물 그룹의 ARN이 포함될 수 있습니다. 사물 그룹에 대한 자세한 내용은 AWS IoT 개발자 안내서의 정적 사물 그룹을 참조하십시오.

`--update-targets-operating-system`

코어 디바이스의 운영 체제입니다.

유효한 값: `ubuntu`, `amazon_linux`, `raspbian` 또는 `openwrt`

`--software-to-update`

코어의 소프트웨어를 업데이트할지 아니면 OTA 에이전트 소프트웨어를 업데이트할지 지정합니다.

유효한 값: `core` 또는 `ota_agent`

`--s3-url-signer-role`

AWS IoT Greengrass 소프트웨어 업데이트 아티팩트에 연결되는 URL을 미리 서명하는 데 사용되는 IAM 역할의 ARN입니다. 역할의 연결된 권한 정책은 대상 AWS 리전 리전의 버킷에 대한 `s3:GetObject` 작업을 허용해야 합니다. 또한 역할은 `iot.amazonaws.com`이 신뢰할 수 있는 엔터티로 역할을 수임할 수 있도록 허용해야 합니다. 자세한 내용은 the section called “OTA 업데이트에 대한 IAM 권한” 섹션을 참조하세요.

`--amzn-client-token`

(선택 사항) 멱등(Idempotent) 요청 생성에 사용되는 클라이언트 토큰입니다. 내부 재시도로 인해 중복 업데이트가 생성되지 않도록 고유한 토큰을 제공하십시오.

`--update-agent-log-level`

(선택 사항) OTA 업데이트 에이전트에 의해 생성된 로그 명령문의 로깅 수준입니다. 기본값은 `ERROR`입니다.

유효한 값: `NONE`, `TRACE`, `DEBUG`, `VERBOSE`, `INFO`, `WARN`, `ERROR` 또는 `FATAL`

> **Note**
>
> `CreateSoftwareUpdateJob`은 다음과 같은 지원되는 아키텍처 및 운영 체제 조합에 대한 요청만 허용합니다.
>
> - ubuntu/x86_64
> - ubuntu/aarch64
> - amazon_linux/x86_64
> - raspbian/armv7l
> - raspbian/armv6l
> - openwrt/aarch64
> - openwrt/armv7l

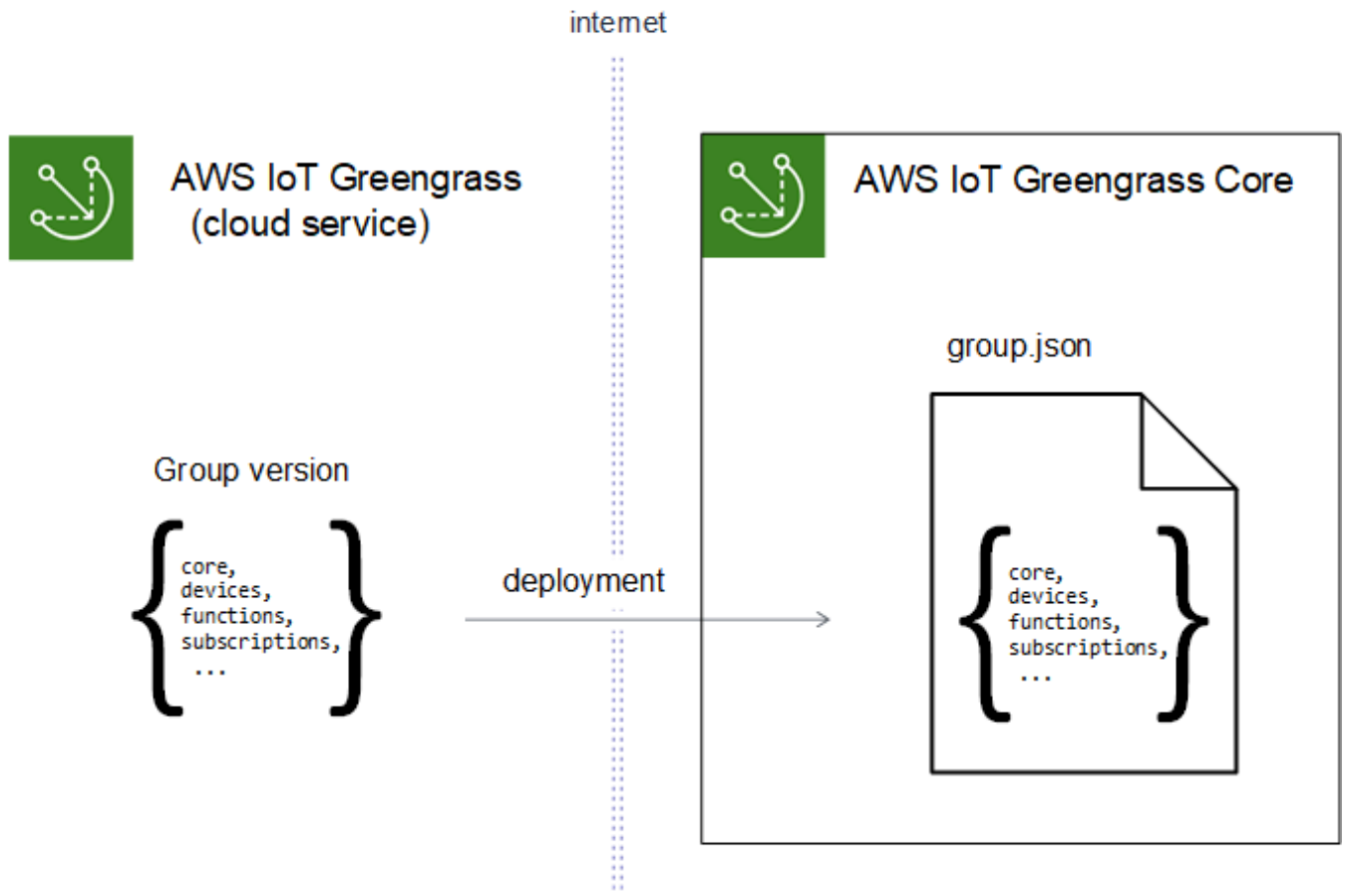
AWS IoT Greengrass 코어에 AWS IoT Greengrass 그룹 배포

AWS IoT Greengrass 그룹은 엣지 환경에서 엔터티를 구성하는 데 사용됩니다. 그룹을 사용해 그룹의 엔터티가 서로 그리고 AWS 클라우드와 상호 작용할 수 있는 방법을 제어합니다. 예를 들어 로컬 실행에는 그룹의 Lambda 함수만 배포되며, 그룹에 있는 디바이스만 로컬 MQTT 서버를 사용하여 통신할 수 있습니다.

그룹에는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 AWS IoT 디바이스인 코어가 포함되어야 합니다. 코어는 엣지 게이트웨이 역할을 하며, 엣지 환경에서 AWS IoT Core 기능을 제공합니다. 비즈니스 요구에 맞게 그룹에 다음 엔터티를 추가할 수도 있습니다.

- 클라이언트 디바이스. AWS IoT 레지스트리에서 사물로 표시됩니다. 이 디바이스는 FreeRTOS를 실행하거나 AWS IoT 디바이스 SDK 또는 AWS IoT Greengrass Discovery API를 사용하여 코어에 대한 연결 정보를 가져옵니다. 그룹 구성원인 클라이언트 디바이스만 코어에 연결할 수 있습니다.
- Lambda 함수. 코어에서 코드를 실행하는 사용자 정의된 서버리스 애플리케이션입니다. Lambda 함수는 AWS Lambda에서 작성되고 Greengrass 그룹에서 참조됩니다. 자세한 설명은 로컬 Lambda 함수 실행 섹션을 참조하세요.
- 커넥터. 코어에서 코드를 실행하는 미리 정의된 서버리스 애플리케이션입니다. 커넥터는 로컬 인프라, 디바이스 프로토콜, AWS, 기타 클라우드 서비스와 기본적으로 통합하는 기능을 제공합니다. 자세한 설명은 커넥터를 사용하여 서비스 및 프로토콜과 통합 섹션을 참조하세요.
- 구독. MQTT 통신을 위해 인증을 받는 MQTT 주제(또는 제목), 게시자 및 구독자를 정의합니다.
- 리소스. 로컬 디바이스 및 볼륨, 기계 학습 모델 및 보안에 대한 참조로 Greengrass Lambda 함수 및 커넥터에 의한 액세스 제어에 사용됩니다.
- 로그. AWS IoT Greengrass 시스템 구성 요소와 Lambda 함수에 대한 로깅 구성입니다. 자세한 설명은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요.

AWS 클라우드에서 Greegrass 그룹을 관리한 다음 코어에 배포합니다. 배포하면 코어 디바이스에서 `group.json` 파일로 그룹 구성이 복제됩니다. 이 파일은 *`greengrass-root`*`/ggc/deployments/group`에 위치합니다.



> **Note**
>
> 배포 중에는 코어 디바이스에서 Greengrass 대몬(daemon) 프로세스가 중지한 다음 다시 시작됩니다.

AWS IoT 콘솔에서 그룹 배포

AWS IoT 콘솔의 그룹 구성 페이지에서 그룹을 배포하고 그룹 배포를 관리할 수 있습니다.

> **Note**
>
> 콘솔에서 이 페이지를 열려면 Greengrass 디바이스를 선택한 다음 그룹(V1)을 선택하고 Greengrass 그룹에서 그룹을 선택합니다.

현재 그룹 버전을 배포하려면

- 그룹 구성 페이지에서 배포를 선택합니다.

그룹의 배포 이력을 확인하려면

그룹의 배포 이력에는 각 배포 시도의 날짜 및 시간, 그룹 버전, 상태를 포함하여 그룹의 배포 기록이 포함됩니다.

1. 그룹 구성 페이지에서 배포 탭을 선택합니다.
2. 오류 메시지를 포함하여 배포에 대한 자세한 내용을 보려면 AWS IoT 콘솔에서 Greengrass 디바이스 아래에 배포를 선택합니다.

그룹 배포를 재배포하려면

현재 배포가 실패할 경우 배포를 재배포하거나 다른 그룹 버전으로 되돌릴 수 있습니다.

1. AWS IoT 콘솔에서 Greengrass 디바이스를 선택한 다음 그룹(V1)을 선택합니다.
2. 배포 탭을 선택합니다.
3. 재배포하려는 배포를 선택하고 재배포를 선택합니다.

그룹 배포를 재설정하려면

그룹 배포를 재설정하여 그룹을 이동 또는 삭제하거나 배포 정보를 제거할 수 있습니다. 자세한 설명은 the section called “배포 재설정” 섹션을 참조하세요.

1. AWS IoT 콘솔에서 Greengrass 디바이스를 선택한 다음 그룹(V1)을 선택합니다.
2. 배포 탭을 선택합니다.
3. 재설정할 배포를 선택하려면 배포 재설정을 선택합니다.

AWS IoT Greengrass API를 사용하여 그룹 배포

AWS IoT Greengrass API는 다음 작업을 통해 AWS IoT Greengrass 그룹을 배포하고 그룹 배포를 관리할 수 있습니다. AWS CLI, AWS IoT Greengrass API 또는 AWS SDK에서 이러한 작업을 호출할 수 있습니다.

작업	설명
CreateDeployment	`NewDeployment` 또는 `Redeployment` 배포를 생성합니다. 현재 배포가 실패할 경우 배포를 재배포할 수 있습니다. 또는 재배포를 통해 다른 그룹 버전으로 되돌릴 수 있습니다.
GetDeploymentStatus	배포 상태를 `Building`, `InProgress` , `Success` 또는 `Failure`로 반환합니다. 배포 알림을 수신하도록 Amazon EventBridge 이벤트를 구성할 수 있습니다. 자세한 설명은 the section called “배포 알림 받기” 섹션을 참조하세요.
ListDeployments	그룹의 배포 이력을 반환합니다.
ResetDeployments	그룹에 대한 배포를 재설정합니다. 그룹 배포를 재설정하여 그룹을 이동 또는 삭제하거나 배포 정보를 제거할 수 있습니다. 자세한 설명은 the section called “배포 재설정” 섹션을 참조하세요.

> **Note**
>
> 대량 작업에 대한 자세한 내용은 the section called “대량 배포 생성” 섹션을 참조하십시오.

그룹 ID 가져오기

그룹 ID는 일반적으로 API 작업에 사용됩니다. ListGroups작업을 사용하여 그룹 목록에서 대상 그룹의 ID를 찾을 수 있습니다. 예를 들어 AWS CLI에서 `list-groups` 명령을 사용합니다.

```
aws greengrass list-groups
```

결과를 필터링하는 query 옵션을 포함할 수도 있습니다. 예:

- 가장 최근에 만든 그룹을 가져오려면 다음을 수행하십시오.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))[0]"
```

- 이름으로 그룹을 가져오려면 다음을 수행하십시오.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

  그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

다음은 list-groups 응답의 예입니다. 각 그룹에 대한 정보에는 속성에 있는 그룹의 ID(Id 속성에 있음) 및 최신 그룹 버전의 ID(LatestVersion 속성에 있음)가 포함됩니다. 그룹의 다른 버전 ID를 가져오려면 그룹 ID를 와 함께 사용하십시오 ListGroupVersions.

> **Note**
>
> AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.


```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
```

```
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

AWS 리전을 지정하지 않으면 AWS CLI 명령은 사용자 프로파일에서 기본 리전을 사용합니다. 다른 리전의 그룹을 반환하려면 ## 옵션을 포함합니다. 예:

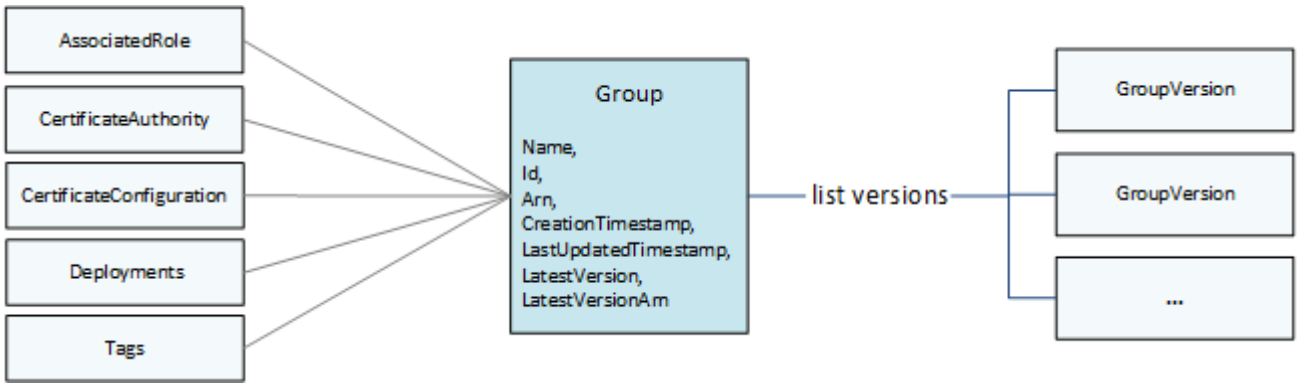
```
aws greengrass list-groups --region us-east-1
```

AWS IoT Greengrass 그룹 객체 모델 개요

AWS IoT Greengrass API를 프로그래밍할 때 Greengrass 그룹 객체 모델을 이해하는 것은 도움이 됩니다.

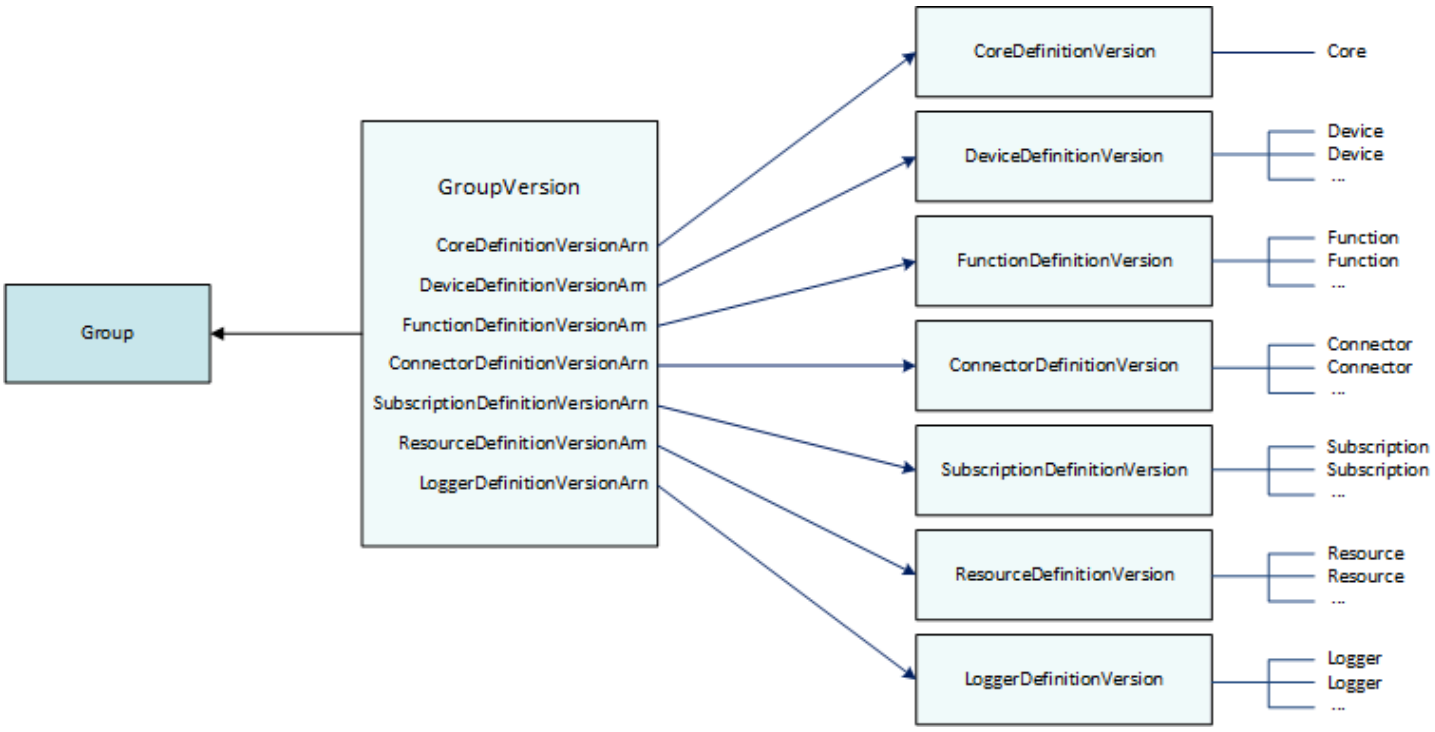
그룹

AWS IoT Greengrass API에서, 최상위 `Group` 객체는 메타데이터와 `GroupVersion` 객체 목록으로 구성되어 있습니다. `GroupVersion` 객체는 `Group`에 ID를 기준으로 연결되어 있습니다.



그룹 버전

GroupVersion 객체는 그룹 구성원을 정의합니다. 각 GroupVersion은 CoreDefinitionVersion과, ARN을 기준으로 다른 구성 요소 버전을 참조합니다. 이러한 참조를 통해 그룹에 포함할 엔터티가 결정됩니다.



예를 들어 그룹에 Lambda 함수 3개, 디바이스 1개, 구독 2개를 포함하려면 GroupVersion은 다음을 참조합니다.

- 필수 코어가 포함된 CoreDefinitionVersion입니다.
- 3개의 함수가 포함된 FunctionDefinitionVersion
- 클라이언트 디바이스가 포함된 DeviceDefinitionVersion.
- 2개의 구독이 포함된 SubscriptionDefinitionVersion

코어 디바이스에 배포된 GroupVersion은 로컬 환경에서 제공되는 개체와 해당 개체가 상호 작용하는 방식을 결정합니다.

그룹 구성 요소

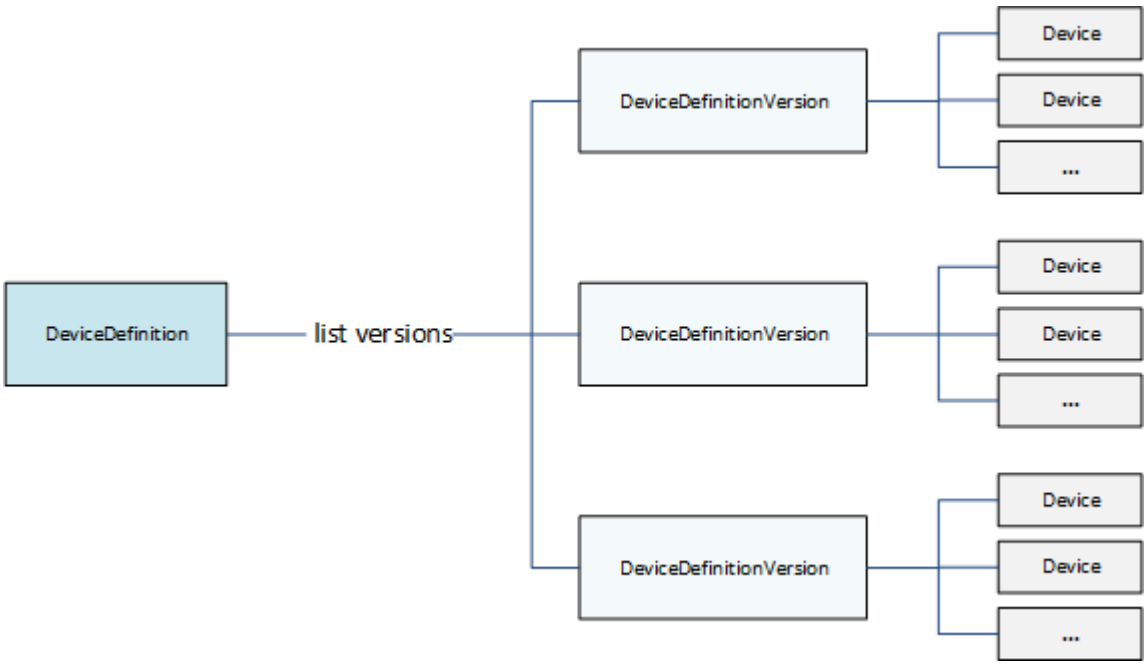
그룹에 추가한 구성 요소는 3개 레벨의 계층 구조로 되어 있습니다.

- 지정된 유형의 DefinitionVersion개체 목록을 참조하는 정의입니다. 예를 들어 `DeviceDefinition`은 `DeviceDefinitionVersion` 객체를 참조합니다.
- 지정된 유형의 엔티티 세트를 DefinitionVersion포함하는 A. 예를 들어 `DeviceDefinitionVersion`에는 `Device` 객체 목록이 포함되어 있습니다.
- 속성과 동작을 정의하는 개별 엔터티. 예를 들어, `Device`는 AWS IoT 레지스트리에서 해당하는 클라이언트 디바이스 ARN, 디바이스 인증서의 ARN, 그리고 클라우드에서 로컬 섀도우가 자동으로 동기화되는 위치를 정의합니다.

  다음 유형의 엔터티를 그룹에 추가할 수 있습니다.

  - 지원
  - Core
  - 디바이스
  - 함수
  - Logger
  - 리소스
  - Subscription

다음 예제는 `DeviceDefinition` 각각 여러 `Device` 객체를 포함하고 있는 3개의 `DeviceDefinitionVersion` 객체를 참조합니다. 그룹에서는 한 번에 한 개의 `DeviceDefinitionVersion`만 사용됩니다.



그룹 업데이트

AWS IoT Greengrass API에서 버전을 사용하여 그룹 구성을 업데이트할 수 있습니다. 버전은 변경할 수 없으므로 그룹 구성 요소를 추가, 제거 또는 변경하려면 새 엔티티나 업데이트된 DefinitionVersion엔티티를 포함하는 개체를 만들어야 합니다.

새 DefinitionVersions개체를 새 정의 개체 또는 기존 Definition 개체와 연결할 수 있습니다. 예를 들어, `CreateFunctionDefinition` 작업을 사용하여 `FunctionDefinitionVersion`을 초기 버전으로 포함하는 `FunctionDefinition`을 만들거나, `CreateFunctionDefinitionVersion` 작업을 사용하여 기존 `FunctionDefinition`을 참조할 수 있습니다.

그룹 구성 요소를 만든 후 그룹에 포함하려는 모든 DefinitionVersion개체를 포함하는 구성 요소를 만듭니다. `GroupVersion` 그런 다음 `GroupVersion`을 배포합니다.

`GroupVersion`을 배포하려면 `Core`가 정확히 1개 포함된 `CoreDefinitionVersion`을 참조해야 합니다. 참조된 모든 엔터티는 그룹의 구성원이어야 합니다. Greengrass 서비스 역할은 `GroupVersion`을 배포 중인 AWS 리전의 AWS 계정과 연결해야 합니다.

> **Note**
>
> API에서 `Update` 작업은 `Group` 또는 구성 요소 Definition 객체의 이름을 변경하는 데 사용됩니다.

AWS 리소스를 참조하는 엔터티 업데이트

Greengrass Lambda 함수와 보안 리소스는 Greengrass별 속성을 정의하며, 해당 AWS 리소스를 참조합니다. 이러한 엔터티를 업데이트하기 위해서는 Greengrass 객체 대신 해당하는 AWS 리소스를 변경할 수도 있습니다. 예를 들어 Lambda 함수는 AWS Lambda에서 함수를 참조하며, Greengrass 그룹에 적용되는 수명 주기와 기타 속성도 정의합니다.

- Lambda 함수 코드나 패키징된 종속성을 업데이트하려면 AWS Lambda에서 변경하십시오. 이러한 변경 사항은 그 다음 그룹 배포 시 AWS Lambda에서 가져와 로컬 환경에 복사됩니다.
- Greengrass별 속성을 업데이트하려면 업데이트된 `Function` 속성이 포함된 `FunctionDefinitionVersion`을 생성합니다.

> **Note**
>
> Greengrass Lambda 함수는 별칭 ARN 또는 버전 ARN을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭 ARN을 참조하면(권장 사항), AWS Lambda에서 새로운 함수 버전을 게시할 때 `FunctionDefinitionVersion`(또는 `SubscriptionDefinitionVersion`)을 업데이트하지 않아도 됩니다. 자세한 설명은 the section called “별칭 또는 버전을 기준으로 함수 참조” 섹션을 참조하세요.

다음 사항도 참조하십시오.

- the section called “배포 알림 받기”
- the section called “배포 재설정”
- the section called “대량 배포 생성”
- 배포 문제 해결

- AWS IoT Greengrass Version 1 API 참조
- AWS CLI 명령 참조에서 AWS IoT Greengrass 명령

배포 알림 받기

Amazon EventBridge 이벤트 규칙을 사용하여 Greengrass 그룹 배포의 상태 변경에 대한 알림을 받을 수 있습니다. EventBridge는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. AWS IoT Greengrass는 이러한 이벤트를 EventBridge에 한 번 이상 전송합니다. 즉, AWS IoT Greengrass는 전달을 보장하기 위해 특정 이벤트의 사본을 여러 개 전송할 수 있습니다. 또한 이벤트 리스너는 이벤트가 발생한 순서대로 이벤트를 수신하지 못할 수 있습니다.

> **Note**
>
> Amazon EventBridge는 애플리케이션을 Greengrass 코어 디바이스 및 배포 알림과 같이 다양한 소스의 데이터와 연결하는 데 사용할 수 있는 이벤트 버스 서비스입니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 섹션을 참조하세요.

AWS IoT Greengrass는 그룹 배포 상태가 변경될 때 이벤트를 발생시킵니다. 모든 상태 전환 또는 지정한 상태로의 전환에 대해 실행하는 EventBridge 규칙을 생성할 수 있습니다. 배포가 규칙을 트리거

하는 상태로 전환되면 EventBridge는 규칙에 정의된 대상 작업을 간접 호출합니다. 이렇게 하면 알림을 전송하고, 이벤트 정보를 캡처하고, 적절한 조치를 취하거나, 상태 변경에 대응하는 기타 이벤트를 시작할 수 있습니다. 예를 들면, 다음 사용 사례에 대한 규칙을 생성할 수 있습니다.

- 자산 다운로드 및 담당자 알림 전송과 같은 배포 후 작업을 트리거합니다.
- 배포 성공 또는 실패 시 알림 보내기
- 배포 이벤트에 대한 사용자 지정 지표 게시

AWS IoT Greengrass는 배포가 다음 상태 `Building`, `InProgress`, `Success` 및 `Failure`로 전환될 때 이벤트를 발생시킵니다.

> **Note**
>
> 대량 배포 작업 상태를 모니터링하는 기능은 현재 지원되지 않습니다. 하지만 AWS IoT Greengrass는 대량 배포에 포함된 개별 그룹 배포의 상태 변경 이벤트를 발생시킵니다.

그룹 배포 상태 변경 이벤트

배포 상태 변경에 대한 이벤트에서 사용하는 형식은 다음과 같습니다.

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "time":"2018-03-22T00:38:11Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
        "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "deployment-type": "NewDeployment|Redeployment|ResetDeployment|
ForceResetDeployment",
        "status": "Building|InProgress|Success|Failure"
    }
}
```

하나 이상의 그룹에 적용할 규칙을 생성할 수 있습니다. 다음 배포 유형과 배포 상태 중 하나로 규칙을 필터링할 수 있습니다.

배포 유형

- `NewDeployment`. 그룹 버전의 최초 배포.
- `ReDeployment`. 그룹 버전의 재배포.
- `ResetDeployment`. AWS 클라우드와 AWS IoT Greengrass 코어에 저장된 배포 정보를 삭제합니다. 자세한 내용은 the section called “배포 재설정” 섹션을 참조하세요.
- `ForceResetDeployment`. AWS 클라우드에 저장된 배포 정보를 삭제하며, 코어 대응을 대기하지 않고 성공을 보고합니다. 코어가 연결되었거나 다음에 연결할 때 코어에 저장된 배포 정보도 삭제합니다.

배포 상태

- `Building`. AWS IoT Greengrass가 그룹 구성을 검증하고 배포 아티팩트를 빌드합니다.
- `InProgress`. 배포가 AWS IoT Greengrass 코어에서 진행 중입니다.
- `Success`. 배포가 성공했습니다.
- `Failure`. 배포가 실패했습니다.

이벤트가 중복되거나 이벤트 순서가 잘못되었을 수 있습니다. 이벤트 순서를 정하려면 `time` 속성을 사용하세요.

> **Note**
>
> AWS IoT Greengrass에서는 `resources` 속성이 사용되지 않으므로 항상 비어 있습니다.

EventBridge 규칙 생성을 위한 사전 조건

AWS IoT Greengrass에 대한 EventBridge 규칙을 생성하기 전에 다음 수행해야 합니다.

- Eventbridge의 이벤트, 규칙, 대상을 숙지해야 합니다.
- EventBridge 규칙에 의해 간접 호출되는 대상을 생성하고 구성해야 합니다. 규칙은 다음을 비롯한 다양한 유형의 대상을 간접 호출할 수 있습니다.
  - Amazon Simple Notification Service (Amazon SNS)
  - AWS Lambda 함수
  - Amazon Kinesis Video Streams

- Amazon Simple Queue Service(Amazon SQS) 대기열

자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 및 Amazon EventBridge 시작하기 섹션을 참조하세요.

배포 알림 구성(콘솔)

다음 단계를 사용하여 그룹의 배포 상태가 변경될 때 Amazon SNS 주제를 게시하는 EventBridge 규칙을 생성합니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용자 안내서의 AWS 리소스에서 이벤트에 트리거하는 EventBridge 규칙 생성을 참조하세요.

1. Amazon EventBridge 콘솔을 엽니다.
2. 탐색 창에서 규칙을 선택합니다.
3. 규칙 생성을 선택합니다.
4. 규칙에 대해 이름과 설명을 입력합니다.

   규칙은 동일한 리전과 동일한 이벤트 버스의 다른 규칙과 동일한 이름을 가질 수 없습니다.
5. 이벤트 버스에서 이 규칙과 연결할 이벤트 버스를 선택합니다. 이 규칙이 자신의 계정에서 발생하는 이벤트와 일치하도록 하려면 AWS 기본 이벤트 버스를 선택합니다. 계정의 AWS 서비스가 이벤트를 출력하면 항상 계정의 기본 이벤트 버스로 이동합니다.
6. 규칙 유형(Rule type)에서 이벤트 패턴이 있는 규칙(Rule with an event pattern)을 생성합니다.
7. 다음(Next)을 선택합니다.
8. 이벤트 소스(Event source)에서 AWS 서비스(services)를 선택합니다.
9. 이벤트 패턴의 경우, AWS 서비스를 선택합니다.
10. AWS 서비스 이름에서 Greengrass를 선택합니다.
11. 이벤트 유형에에서 Greengrass 배포 상태 변경을 선택합니다.

> **Note**
>
> CloudTrail을 통한 AWS API 호출 이벤트 유형은 AWS IoT Greengrass와 AWS CloudTrail의 통합을 기반으로 합니다. 이 옵션을 사용하여 AWS IoT Greengrass API에 대한 읽기 또는 쓰기 호출에 의해 개시되는 규칙을 만들 수 있습니다. 자세한 내용은 the section called “AWS CloudTrail을 사용하여 AWS IoT Greengrass API 직접 호출 로깅” 섹션을 참조하세요.

12. 알림을 트리거하는 배포 상태를 선택합니다.

    - 모든 상태 변경 이벤트에 대한 알림을 받으려면 모든 상태를 선택합니다.
    - 일부 상태 변경 이벤트에 대해서만 알림을 받으려면 특정 상태를 선택한 다음 대상 상태를 선택합니다.

13. 알림을 트리거하는 배포 유형을 선택합니다.

    - 모든 배포 유형에 대한 알림을 받으려면 모든 상태를 선택합니다.
    - 일부 배포 유형에 대해서만 알림을 받으려면 특정 상태를 선택한 다음 대상 배포 유형을 선택합니다.

14. 다음(Next)을 선택합니다.
15. 대상 유형(Target types)에서 AWS서비스(service)를 선택합니다.
16. 대상 선택에서 대상을 구성합니다. 이 예제에서는 Amazon SNS 주제를 사용하지만 알림을 보내도록 다른 대상 유형을 구성할 수 있습니다.

    a. 대상에서 SNS 주제를 선택합니다.
    b. 주제에서 대상 주제를 선택합니다.
    c. 다음(Next)을 선택합니다.

17. 태그에서 규칙에 대한 태그를 정의하거나 필드를 비워 둡니다.
18. 다음(Next)을 선택합니다.
19. 규칙의 세부 정보를 검토하고 규칙 생성(Create rule)을 선택합니다.

배포 알림 구성(CLI)

다음 단계를 사용하여 그룹의 배포 상태가 변경될 때 Amazon SNS 주제를 게시하는 EventBridge 규칙을 생성합니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다.

1. 규칙을 생성합니다.

    - *group-id*를 AWS IoT Greengrass 그룹의 ID로 바꿉니다.

```
aws events put-rule \
  --name TestRule \
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":
 [\"group-id\"]}}"
```

패턴에서 생략된 속성은 무시됩니다.

2. 규칙 대상으로 주제를 추가합니다.

   - *topic-arn*을 Amazon SNS 주제의 ARN으로 바꿉니다.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> Amazon EventBridge가 대상 주제를 호출하도록 허용하려면 주제에 리소스 기반 정책을 추가해야 합니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon SNS 권한 섹션을 참조하세요.

자세한 내용은 Amazon EventBridge 사용 설명서에서 EventBridge의 이벤트 및 이벤트 패턴 섹션을 참조하세요.

배포 알림 구성(AWS CloudFormation)

AWS CloudFormation 템플릿을 사용하여 Greengrass 그룹 배포의 상태 변경에 대한 알림을 보내는 EventBridge 규칙을 만듭니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 Amazon EventBridge 리소스 유형 참조를 참조하세요.

다음 사항도 참조하세요.

- AWS IoT Greengrass 그룹 배포
- Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 섹션을 참조하세요.

배포 재설정

이 기능은 AWS IoT Greengrass 코어 v1.1 이상에 사용할 수 있습니다.

다음 작업을 수행하여 그룹의 배포를 재설정할 수 있습니다.

- 그룹의 코어를 다른 그룹으로 이동하려는 경우 또는 그룹의 코어가 다시 이미징된 경우 그룹을 삭제합니다. 그룹을 삭제하기 전에 다른 Greengrass 그룹과 함께 코어를 사용하도록 그룹 배포를 재설정해야 합니다.
- 그룹의 코어를 다른 그룹으로 이동합니다.
- 그룹의 상태를 배포 이전으로 되돌립니다.
- 코어 디바이스에서 배포 구성을 제거합니다.
- 코어 디바이스나 클라우드에서 중요한 데이터를 삭제합니다.
- 현재 그룹에서 코어를 다른 코어로 바꾸지 않고 새 그룹 구성을 코어에 배치합니다.

> **Note**
>
> 배포 재설정 기능은 AWS IoT Greengrass 코어 소프트웨어 v1.0.0에서는 제공되지 않습니다. v1.0.0을 이용해 배포된 그룹은 삭제할 수 없습니다.

재설정 배포 작업은 먼저 클라우드에 저장되어 있는 해당 그룹의 배포 정보를 모두 정리합니다. 그런 다음, 배포 관련 정보(Lambda 함수, 사용자 로그, 섀도우 데이터베이스 및 서버 인증서를 포함하지만 사용자가 정의한 `config.json` 또는 Greengrass 코어 인증서는 제외)도 모두 정리하도록 그룹의 코어 디바이스에 지시합니다. 그룹에 현재 `In Progress` 또는 `Building` 상태의 배포가 있는 경우 그룹에 대한 배포 재설정을 시작할 수 없습니다.

AWS IoT 콘솔에서 배포 재설정

AWS IoT 콘솔의 그룹 구성 페이지에서 그룹 배포를 재설정할 수 있습니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 배포 탭에서 배포 재설정을 선택합니다.
4. 이 Greengrass Group의 배포 재설정 대화 상자에 동의한다는 의미의 **`confirm`**을(를) 입력하고 배포 재설정을 선택합니다.

AWS IoT Greengrass API를 사용하여 배포 재설정

AWS CLI, AWS IoT Greengrass API 또는 AWS SDK에서 `ResetDeployments` 작업을 사용하여 배포를 재설정할 수 있습니다. 이 단원의 예제에서는 CLI를 사용합니다.

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

reset-deployments CLI 명령의 인수:

`--group-id`

그룹 ID입니다. `list-groups` 명령을 사용하여 이 값을 가져옵니다.

`--force`

선택 사항으로, 그룹의 코어 디바이스가 분실, 도난 또는 파괴된 경우 이 파라미터를 사용합니다. 이 옵션을 사용하면 배포 재설정 프로세스가 코어 디바이스의 응답을 기다리지 않고 클라우드에서 모든 배포 정보가 정리되고 난 후 성공을 보고합니다. 그러나 코어 디바이스가 활성 상태이거나 활성화되면 해당 정리 작업도 수행합니다.

`reset-deployments` CLI 명령의 출력은 다음과 같습니다.

```
{
    "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
    "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

다음과 같이 `get-deployment-status` CLI 명령을 사용하면 배포 재설정 상태를 확인할 수 있습니다.

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

get-deployment-status CLI 명령의 인수:

`--deployment-id`

배포 ID입니다.

`--group-id`

그룹 ID입니다.

`get-deployment-status` CLI 명령의 출력은 다음과 같습니다.

```
{
```

```
    "DeploymentStatus": "Success",
    "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

배포 재설정을 준비 중일 때 `DeploymentStatus`는 `Building`으로 설정됩니다. 배포 재설정이 준비된 상태에서 AWS IoT Greengrass 코어가 배포 재설정을 선택하지 않았다면 `DeploymentStatus`은(는) `InProgress`(으)로 표시됩니다.

작업이 실패하면 응답에 오류 정보가 반환됩니다.

다음 사항도 참조하십시오.

- AWS IoT Greengrass 그룹 배포
- ResetDeployments AWS IoT Greengrass Version 1API 레퍼런스에서
- GetDeploymentStatusAWS IoT Greengrass Version 1API 레퍼런스에서

그룹의 대량 배포 생성

간단한 API 호출을 사용하여 한 번에 많은 수의 Greengrass 그룹을 배포할 수 있습니다. 이러한 배포는 상한이 고정된 가변 레이트로 트리거됩니다.

이 자습서에서는 AWS CLI를 사용하여 AWS IoT Greengrass에서 그룹 대량 배포를 생성 및 모니터링하는 방법을 설명합니다. 이 자습서에 나오는 대량 배포 예제에는 여러 그룹이 포함되어 있습니다. 구현 시 이러한 예제를 사용해 필요한 수 만큼 그룹을 추가할 수 있습니다.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. 대량 배포 입력 파일 생성 및 업로드
2. 벌크 배포를 위한 IAM 실행 역할 생성 및 구성
3. 실행 역할이 S3 버킷에 액세스하도록 허용
4. 그룹 배포
5. 배포 테스트

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 배포 가능한 Greengrass 그룹 한 개 이상. AWS IoT Greengrass 그룹 및 코어 만들기에 대한 자세한 내용은 시작하기 AWS IoT Greengrass 단원을 참조하십시오.
- 시스템에 설치 및 구성된 AWS CLI. 자세한 내용은 AWS CLI 사용 설명서를 참조하십시오.
- AWS IoT Greengrass와 동일한 AWS 리전에서 생성된 S3 버킷. 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 첫 S3 버킷 생성 및 구성을 참조하세요.

> **Note**
>
> 현재 SSE KMS를 사용하는 버킷은 지원되지 않습니다.

1단계: 대량 배포 입력 파일 생성 및 업로드

이 단계에서는 배포 입력 파일을 생성하여 Amazon S3 버킷에 업로드합니다. 이 파일은 직렬화되어 행으로 구분된 JSON 파일로, 대량 배포의 각 그룹에 대한 정보를 포함하고 있습니다. AWS IoT Greengrass에서는 이러한 정보를 사용하여 대량 그룹 배포를 초기화할 때 사용자를 대신해 각 그룹을 배포합니다.

1. 다음 명령을 실행하여 배포하려는 각 그룹의 `groupId`를 가져옵니다. AWS IoT Greengrass에서 배포할 각 그룹을 식별할 수 있도록 대량 배포 입력 파일에 `groupId`를 입력합니다.

   > **Note**
   >
   > AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

   ```
   aws greengrass list-groups
   ```

   응답에는 AWS IoT Greengrass 계정의 각 그룹에 대한 정보가 들어 있습니다.

   ```
   {
     "Groups": [
       {
         "Name": "string",
   ```

```
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

다음 명령을 실행하여 배포하려는 각 그룹의 `groupVersionId`를 가져옵니다.

```
list-group-versions --group-id groupId
```

응답에는 그룹 내 모든 버전에 대한 정보가 들어 있습니다. 사용할 그룹 버전의 `Version` 값을 적어 둡니다.

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
      "Version": "string",
      "CreationTimestamp": "string"
    }
  ],
  "NextToken": "string"
}
```

2. 다음 예제에서 선택한 컴퓨터 터미널 또는 편집기에서 *MyBulkDeploymentInputFile* 파일을 생성합니다. 이 파일에는 대량 배포에 포함되는 각 AWS IoT Greengrass 그룹에 대한 정보가 포함되어 있습니다. 자습서에서 사용하기 위해 이 예제에서 여러 그룹을 정의하더라도 파일에 한 그룹만 포함할 수 있습니다.

> **Note**
>
> 이 파일의 크기는 100MB 미만이어야 합니다.

```
{"GroupId":"groupId1", "GroupVersionId":"groupVersionId1",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId2", "GroupVersionId":"groupVersionId2",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId3", "GroupVersionId":"groupVersionId3",
 "DeploymentType":"NewDeployment"}
...
```

각 레코드(또는 행)에는 그룹 객체가 포함되어 있습니다. 각 그룹 객체에는 해당하는 `GroupId`, `GroupVersionId` 및 `DeploymentType`이 들어 있습니다. 현재 AWS IoT Greengrass는 `NewDeployment` 대량 배포 유형만 지원합니다.

파일을 저장하고 닫습니다. 파일의 위치를 메모해 둡니다.

3. 터미널에서 다음 명령을 사용해 Amazon S3 버킷에 입력 파일을 업로드합니다. 파일의 위치 및 이름으로 파일 경로를 바꿉니다. 자세한 내용은 버킷에 객체 추가를 참조하십시오.

```
aws s3 cp path/MyBulkDeploymentInputFile s3://my-bucket/
```

2단계: IAM 실행 역할 생성 및 구성

이 단계에서는 IAM 콘솔을 사용하여 독립 실행형 실행 역할을 생성합니다. 그런 다음 이 역할과 AWS IoT Greengrass 간에 신뢰 관계를 설정해 IAM 사용자가 실행 역할에 대한 `PassRole` 권한을 갖도록 합니다. 그러면 AWS IoT Greengrass에서는 사용자의 실행 역할을 맡아 사용자를 대신해 배포를 생성합니다.

1. 다음 정책에 따라 실행 역할을 생성합니다. 이 정책 문서를 통해 AWS IoT Greengrass에서는 사용자를 대신해 배포를 생성할 때마다 대량 배포 입력 파일에 액세스할 수 있습니다.

   IAM 역할 생성 및 권한 위임에 대한 자세한 내용은 IAM 역할 생성을 참조하십시오.

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "greengrass:CreateDeployment",
            "Resource": [
                "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId1",
                "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId2",
                "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId3",
                ...
            ]
        }
    ]
}
```

> **Note**
>
> AWS IoT Greengrass에서 대량 배포 입력 파일의 각 그룹 또는 그룹 버전을 배포하려면 이 정책에는 리소스가 있어야 합니다. 모든 그룹에 대한 액세스를 허용하려면 다음과 같이 `Resource`에 별표를 지정합니다.
>
> ```
> "Resource": ["*"]
> ```

2. AWS IoT Greengrass를 포함하도록 실행 역할에 대한 신뢰 관계를 수정합니다. 그러면 AWS IoT Greengrass에서는 사용자의 실행 역할과 해당 역할에 연결된 권한을 사용할 수 있습니다. 자세한 내용은 기존 역할에서 신뢰 관계를 편집을 참조하십시오.

   혼동된 대리자 보안 문제를 방지하려면 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키도 포함하는 것이 좋습니다. 조건 컨텍스트 키는 지정된 계정 및 Greengrass 작업 영역에서 들어오는 요청만 허용하도록 액세스를 제한합니다. 혼동된 대리자 문제에 대한 자세한 내용은, 교차 서비스 혼동된 대리자 예방를 참조하세요.

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

```
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}
```

3. IAM 사용자에게 실행 역할에 대한 IAM PassRole 권한을 부여합니다. 이 IAM 사용자는 대량 배포를 시작하기 위해 사용한 사용자입니다. PassRole 권한을 사용하면 IAM 사용자가 실행 권한을 AWS IoT Greengrass에 사용할 수 있도록 전달할 수 있습니다. 자세한 내용은 사용자에게 AWS 서비스 역할을 전달할 수 있는 권한 부여를 참조하세요.

   다음 예제를 사용하여 실행 역할에 연결된 IAM 정책을 업데이트합니다. 필요에 따라 이 예제를 수정하십시오.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "Stmt1508193814000",
           "Effect": "Allow",
           "Action": [
               "iam:PassRole"
           ],
           "Resource": [
               "arn:aws:iam::account-id:user/executionRoleArn"
           ]
           "Condition": {
               "StringEquals": {
                   "iam:PassedToService": "greengrass.amazonaws.com"
               }
```

```
            }
        }
    ]
}
```

3단계: 실행 역할이 S3 버킷에 액세스하도록 허용

대량 배포를 시작하려면 실행 역할이 Amazon S3 버킷에서 대량 배포 입력 파일을 읽을 수 있어야 합니다. `GetObject` 권한이 실행 역할에 액세스할 수 있도록 Amazon S3 버킷에 다음 정책 예제를 연결합니다.

자세한 내용은 S3 버킷 정책을 추가하려면 어떻게 해야 합니까?를 참조하십시오.

```
{
    "Version": "2008-10-17",
    "Id": "examplePolicy",
    "Statement": [
        {
            "Sid": "Stmt1535408982966",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "executionRoleArn"
                ]
            },
            "Action": "s3:GetObject",
            "Resource":
            "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

터미널에서 다음 명령을 사용해 버킷의 정책을 확인할 수 있습니다.

```
aws s3api get-bucket-policy --bucket my-bucket
```

> **Note**
>
> Amazon S3 버킷의 `GetObject` 권한에 권한을 부여하도록 실행 역할을 직접 수정할 수 있습니다. 이렇게 하려면 실행 역할에 다음 정책 예제를 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

4단계: 그룹 배포

이 단계에서는 대량 배포 입력 파일에 구성된 모든 그룹 버전에 대해 대량 배포 작업을 시작합니다. 각 그룹 버전에 대한 배포 작업의 유형은 `NewDeploymentType`입니다.

> **Note**
>
> 동일한 계정의 또 다른 대량 배포 작업이 실행 중인 경우에는 StartBulkDeployment를 호출할 수 없습니다. 요청이 거부됩니다.

1. 다음 명령을 사용하여 대량 배포를 시작합니다.

   모든 StartBulkDeployment 요청에 `X-Amzn-Client-Token` 토큰을 포함하는 것이 좋습니다. 이러한 요청은 토큰 및 요청 파라미터에 관한 idempotent입니다. 이 토큰은 최대 64개의 ASCII 문자로 구성된 고유한 문자열일 수 있으며, 대소문자를 구분합니다.

   ```
   aws greengrass start-bulk-deployment --cli-input-json "{
            "InputFileUri":"URI of file in S3 bucket",
            "ExecutionRoleArn":"ARN of execution role",
   ```

```
            "AmznClientToken":"your Amazon client token"
            }"
```

이 명령은 다음 응답과 함께 성공한 상태 코드인 200을 반환해야 합니다.

```
{
  "bulkDeploymentId": UUID
}
```

대량 배포 ID를 메모해 둡니다. 이 ID는 대량 배포의 상태를 확인하는 데 사용할 수 있습니다.

> **Note**
>
> 현재 대량 배포 작업은 지원되지 않지만 개별 그룹에 대한 배포 상태 변경에 대한 알림을 받을 수 있는 Amazon EventBridge 이벤트 규칙을 생성할 수 있습니다. 자세한 내용은 the section called “배포 알림 받기” 섹션을 참조하세요.

2. 다음 명령을 사용하여 대량 배포 상태를 확인합니다.

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

이 명령은 정보의 JSON 페이로드 이외에 성공한 상태 코드인 200을 반환해야 합니다.

```
 {
  "BulkDeploymentStatus": Running,
  "Statistics": {
     "RecordsProcessed": integer,
     "InvalidInputRecords": integer,
     "RetryAttempts": integer
  },
  "CreatedAt": "string",
  "ErrorMessage": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
```

```
  ]
}
```

BulkDeploymentStatus에는 대량 실행의 현재 상태가 포함되어 있습니다. 실행의 상태는 6개 중 하나일 수 있습니다.

- Initializing. 대량 배포 요청이 수신되었고 실행이 시작 준비 중입니다.
- Running. 대량 배포 실행이 시작되었습니다.
- Completed. 대량 배포 실행이 모든 레코드 처리를 마쳤습니다.
- Stopping. 대량 배포 실행이 중지 명령을 수신했고 곧 종료됩니다. 이전 배포의 상태가 Stopping이면 새 대량 배포를 시작할 수 없습니다.
- Stopped. 대량 배포 실행이 수동으로 중지되었습니다.
- Failed. 대량 배포 실행에서 오류가 발생해 종료되었습니다. 오류에 대한 자세한 내용은 ErrorDetails 필드를 참조하십시오.

또한 JSON 페이로드에는 대량 배포 진행에 대한 통계 정보가 들어 있습니다. 이 정보를 사용해 처리된 그룹 수와 실패한 그룹 수를 확인할 수 있습니다. 통계 정보는 다음과 같습니다.

- RecordsProcessed: 시도한 그룹 레코드 수
- InvalidInputRecords: 재시도할 수 없는 오류를 반환한 총 레코드 수. 예를 들어, 입력 파일의 그룹 레코드가 잘못된 형식을 사용하거나 존재하지 않는 버전을 지정한 경우 또는 실행이 그룹 또는 그룹 버전을 배포할 권한을 부여하지 않은 경우 이러한 정보가 생성될 수 있습니다.
- RetryAttempts: 재시도할 수 있는 오류를 반환한 배포 시도 횟수. 예를 들어, 그룹 배포 시도가 조절 오류를 반환한 경우 재시도가 트리거됩니다. 그룹 배포는 최대 5회까지 재시도할 수 있습니다.

또한 대량 배포 실행에 실패한 경우 이 페이로드에는 문제 해결에 사용할 수 있는 ErrorDetails 섹션이 포함됩니다. 이 섹션에는 실행 실패의 원인에 대한 정보가 들어 있습니다.

대량 배포 상태를 정기적으로 확인해 예상한 대로 진행 중인지 확인할 수 있습니다. 배포가 완료되면 RecordsProcessed가 대량 배포 입력 파일의 배포 그룹 수와 동일해야 합니다. 이는 각 레코드가 처리되었음을 나타냅니다.

5단계: 배포 테스트

ListBulkDeployments 명령을 사용하여 대량 배포의 ID를 찾습니다.

```
aws greengrass list-bulk-deployments
```

이 명령을 `BulkDeploymentId`를 포함해 최근 대량 배포까지 모든 대량 배포 목록을 반환합니다.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

이제, ListBulkDeploymentDetailedReports 명령을 호출해 각 배포에 대한 자세한 정보를 수집합니다.

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

이 명령은 정보의 JSON 페이로드와 함께 성공한 상태 코드인 `200`을 반환해야 합니다.

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
```

```
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

일반적으로, 이 페이로드에는 최근까지 각 배포와 그 배포 상태로 구성된, 페이지가 매겨진 목록이 포함되어 있습니다. 또한 대량 배포 실행에 실패하면 추가 정보가 포함됩니다. 다시, 목록의 총 배포 수와 대량 배포 입력 파일에서 식별한 그룹 수가 동일해야 합니다.

배포가 종료 상태(성공 또는 실패)가 될 때까지 반환되는 정보가 바뀔 수 있습니다. 그때까지 이 명령을 정기적으로 호출할 수 있습니다.

대량 배포 문제 해결

대량 배포가 성공적이지 않은 경우 다음 문제 해결 단계를 시도할 수 있습니다. 터미널에서 이 명령을 실행합니다.

입력 파일 오류 문제 해결

대량 배포 입력 파일에 구문 오류가 있는 경우 대량 배포에 실패할 수 있습니다. 그러면 대량 배포 상태로 `Failed`가 반환되고 첫 번째 확인 오류의 행 번호를 나타내는 오류 메시지가 함께 표시됩니다. 다음과 같은 4가지 오류가 발생할 수 있습니다.

- ```
  InvalidInputFile: Missing GroupId at line number: line number
  ```

  이 오류는 주어진 입력 파일 행이 지정된 파라미터를 등록할 수 없음을 나타냅니다. 누락될 수 있는 파라미터는 `GroupId` 및 `GroupVersionId`입니다.

- ```
  InvalidInputFile: Invalid deployment type at line number : line number. Only valid
   type is 'NewDeployment'.
  ```

  이 오류는 주어진 입력 파일 줄에 잘못된 배포 유형이 나열되었음을 나타냅니다. 현재, 지원되는 유일한 배포 유형은 `NewDeployment`입니다.

- ```
  Line %s is too long in S3 File. Valid line is less than 256 chars.
  ```

  이 오류는 주어진 입력 파일 행이 너무 길어 줄여야 함을 나타냅니다.

- ```
  Failed to parse input file at line number: line number
  ```

  이 오류는 주어진 입력 파일 행이 올바른 json으로 간주되지 않음을 나타냅니다.

동시 대량 배포 확인

다른 배포가 실행 중이거나 종료 상태가 아닌 경우에는 새 대량 배포를 시작할 수 없습니다. 그러면 `Concurrent Deployment Error`가 발생할 수 있습니다. ListBulkDeployments 명령을 사용해 대량 배포가 현재 실행 중이 아닌지 확인할 수 있습니다. 이 명령은 최근까지 대량 배포를 나열합니다.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

맨 처음 나열된 대량 배포의 `BulkDeploymentId`를 사용하여 GetBulkDeploymentStatus 명령을 실행합니다. 최근 대량 배포의 상태가 실행 중이면(`Initializing` 또는 `Running`) 다음 명령을 사용하여 대량 배포를 중지합니다.

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

이 작업은 배포 상태가 `Stopped`가 될 때까지 `Stopping` 상태로 유지됩니다. 배포가 `Stopped` 상태가 되면 새 대량 배포를 시작할 수 있습니다.

오류 세부 정보 확인

`GetBulkDeploymentStatus` 명령을 실행하여 대량 배포 실행 실패에 대한 정보가 포함된 JSON 페이로드를 반환합니다.

```
"Message": "string",
"ErrorDetails": [
  {
    "DetailedErrorCode": "string",
    "DetailedErrorMessage": "string"
  }
]
```

오류가 있는 경우 이 호출로 반환된 `ErrorDetails` JSON 페이로드에는 대량 배포 실행 실패에 대한 추가 정보가 포함됩니다. 예를 들어, `400` 시리즈의 오류 상태 코드는 입력 파라미터 또는 호출자 종속성에 대한 입력 오류를 나타냅니다.

AWS IoT Greengrass 코어 로그 확인

AWS IoT Greengrass 코어 로그를 보고 문제를 해결할 수 있습니다. `runtime.log`를 보려면 다음 명령을 사용하십시오.

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

AWS IoT Greengrass 로깅에 대한 자세한 내용은 AWS IoT Greengrass 로그를 사용하여 모니터링 단원을 참조하십시오.

다음 사항도 참조하세요.

자세한 정보는 다음 자료를 참조하세요.

- AWS IoT Greengrass 그룹 배포
- AWS CLI명령 참조에서 Amazon S3 API 명령
- AWS CLI 명령 참조에서 AWS IoT Greengrass 명령

AWS IoT Greengrass 코어에서 Lambda 함수 실행

AWS IoT Greengrass은(는) 사용자가 AWS Lambda에 작성한 사용자 정의 코드를 위한 컨테이너화된 Lambda 런타임 환경을 제공합니다. AWS IoT Greengrass 코어에 배포된 Lambda 함수는 코어의 로컬 Lambda 런타임에서 실행됩니다. 로컬 Lambda 함수는 로컬 이벤트, 클라우드의 메시지 및 기타 소스에 의해 트리거가 가능하기 때문에 로컬 컴퓨팅 기능을 연결된 디바이스에 제공할 수 있습니다. 예를 들어 클라우드로 데이터를 전송하기 앞서 Lambda 함수를 사용해 디바이스 데이터를 필터링할 수 있습니다.

코어에 Lambda 함수를 배포하려면 Greengrass 그룹에 해당 함수를 추가하고(기존 Lambda 함수를 참조하여) 해당 함수에 대해 그룹별 설정을 구성한 다음, 그룹을 배포합니다. 함수가 AWS 서비스에 액세스하는 경우에는 Greengrass 그룹 역할에 필요한 모든 권한을 추가해야 합니다.

권한, 격리, 메모리 제한 등을 비롯해 Lambda 함수 실행 방법을 결정하는 파라미터를 구성할 수 있습니다. 자세한 설명은 the section called “Greengrass Lambda 함수 실행 제어” 섹션을 참조하세요.

> **Note**
>
> 이 설정으로 AWS IoT Greengrass를 Docker 컨테이너에서 실행할 수도 있습니다. 자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.

다음 표에는 지원되는 AWS Lambda 런타임 및 해당 런타임이 실행될 수 있는 AWS IoT Greengrass 코어 소프트웨어 버전이 나와 있습니다.

언어 또는 플랫폼	GGC 버전
Python 3.8	1.11
Python 3.7	1.9 이상
Python 2.7 *	1.0 이상
Java 8	1.1 이상
Node.js 12.x *	1.10 이상

언어 또는 플랫폼	GGC 버전
Node.js 8.10 *	12.7 이상
Node.js 6.10 *	12.7 이상
C, C++	1.6 이상

* 지원되는 버전의 AWS IoT Greengrass에서 이러한 런타임을 사용하는 Lambda 함수를 실행할 수 있지만 AWS Lambda에서 생성할 수는 없습니다. 디바이스의 런타임이 해당 함수에 지정된 AWS Lambda 런타임과 다른 경우 `FunctionDefintionVersion`의 `FunctionRuntimeOverride`을(를) 사용하여 자체 런타임을 선택할 수 있습니다. 자세한 내용은 을 참조하십시오 CreateFunctionDefinition. 지원되는 런타임에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 런타임 지원 정책을 참조하십시오.

Greengrass Lambda 함수용 SDK

AWS는 AWS IoT Greengrass에서 실행되는 Lambda 함수에서 사용 가능한 SDK를 3개 제공합니다. 이 SDK는 서로 다른 패키지에 포함되어 있기 때문에 두 패키지를 동시에 사용할 수 있습니다. Lambda 함수에서 SDK를 사용하려면 에 업로드하는 Lambda 함수 배포 패키지AWS Lambda에 SDK를 포함시킵니다.

AWS IoT Greengrass 코어 SDK

다음 사항을 위해 로컬 Lambda 함수를 활성화해 코어와 상호 작용합니다.

- MQTT 메시지를 AWS IoT Core와 교환합니다.
- MQTT 메시지를 Greengrass 그룹의 커넥터, 디바이스, 기타 Lambda 함수와 교환합니다.
- 로컬 섀도우 서비스와 상호 작용합니다.
- 다른 로컬 Lambda 함수를 간접적으로 호출합니다.
- 암호 리소스에 액세스합니다.
- 스트림 관리자와 상호 작용합니다.

AWS IoT Greengrass에서 다음 언어 및 플랫폼으로 AWS IoT Greengrass Core SDK를 제공합니다. GitHub

- Java용 AWS IoT Greengrass 코어 SDK
- Node.js용 AWS IoT Greengrass 코어 SDK

- AWS IoT Greengrass Python용 코어 SDK
- C용 AWS IoT Greengrass 코어 SDK

Lambda 함수 배포 패키지에 AWS IoT Greengrass 종속성을 포함시키려면 다음과 같이 하십시오.

1. Lambda 함수의 런타임과 일치하는 AWS IoT Greengrass 패키지의 언어 또는 플랫폼을 다운로드합니다.
2. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
3. 함수 코드가 포함된 Lambda 함수 배포 패키지에 `greengrasssdk`을 포함시킵니다. 이는 Lambda 함수를 생성할 때 AWS Lambda에 업로드하는 패키지입니다.

StreamManagerClient

AWS IoT Greengrass스트림 관리자 작업에는 다음 코어 SDK만 사용할 수 있습니다.

- Java SDK (v1.4.0 이상)
- Python SDK (v1.5.0 이상)
- Node.js SDK (v1.6.0 이상)

Python용 AWS IoT Greengrass Core SDK를 사용하여 스트림 관리자와 상호 작용하려면 Python 3.7 이상을 설치해야 합니다. 또한 Python Lambda 함수 배포 패키지에 포함시킬 종속성을 설치해야 합니다.

1. `requirements.txt` 파일이 포함된 SDK 디렉터리로 이동합니다. 이 파일은 종속성을 나열합니다.
2. SDK 종속성을 설치합니다. 예를 들어 다음 `pip` 명령을 실행하여 현재 디렉터리에 설치합니다.

```
pip install --target . -r requirements.txt
```

코어 디바이스에 Python용 AWS IoT Greengrass 코어 SDK를 설치합니다.

Python Lambda 함수를 실행하는 경우 `pip`를 사용하여 코어 디바이스에 Python의 AWS IoT Greengrass Core SDK를 설치할 수 있습니다. 그런 다음 Lambda 함수 배포 패키지에 SDK을 포함시키지 않고 함수를 배포할 수 있습니다. 자세한 내용은 greengrasssdk를 참조하십시오.

이 지원은 크기 제약이 있는 코어를 위한 것입니다. 가능하면 Lambda 함수 배포 패키지에 SDK를 포함시키는 것이 좋습니다.

AWS IoT Greengrass기계 학습 SDK

Lambda 함수가 Greengrass 코어에 기계 학습 리소스로서 배포된 기계 학습 모델을 이용할 수 있도록 지원합니다. Lambda 함수는 SDK를 사용하여 코어에 커넥터로 배포된 로컬 추론 서비스를 간접적으로 호출하고 상호 작용할 수 있습니다. Lambda 함수 및 ML 커넥터는 SDK를 사용하여 업로드 및 게시를 위해 ML 피드백 커넥터로 데이터를 전송할 수도 있습니다. SDK를 사용하는 코드 예제를 포함하여 자세한 내용은 the section called “ML 이미지 분류”, the section called “ML 객체 감지” 및 the section called “ML 피드백” 단원을 참조하십시오.

다음 표에는 SDK 버전 및 해당 버전을 실행할 수 있는 AWS IoT Greengrass 코어 소프트웨어 버전에 대해 지원되는 언어 및 플랫폼이 나와 있습니다.

SDK 버전	언어 또는 플랫폼	필요한 GGC 버전	Changelog
1.1.0	Python 3.7 또는 2.7	1.9.3 이상	Python 3.7 지원과 새로운 `feedback` 클라이언트를 추가했습니다.
1.0.0	Python 2.7	1.7 이상	최초 릴리스.

다운로드 정보는 the section called “AWS IoT Greengrass ML SDK 소프트웨어” 섹션을 참조하세요.

AWS SDK

Amazon S3, DynamoDB, AWS IoT, 및 AWS IoT Greengrass 같은 AWS 서비스를 직접 호출할 수 있도록 로컬 Lambda 함수를 활성화합니다. Lambda 함수에서 AWS SDK를 사용하려면 배포 패키지에 이를 포함시켜야 합니다. 또한 동일한 패키지에서 AWS IoT Greengrass Core SDK로 AWS

SDK를 사용하는 경우 Lambda 함수가 정확한 네임스페이스를 사용하는지 확인하십시오. Lambda 함수는 코어가 오프라인 상태일 때 AWS 또는 기타 클라우드 서비스와 통신을 할 수 없습니다.

시작 리소스 센터에서 AWS SDK를 다운로드하십시오.

배포 패키지를 만드는 방법에 대한 자세한 내용은 시작하기 튜토리얼의 the section called “Lambda 함수 생성 및 패키징” 또는 AWS Lambda 개발자 안내서의 배포 패키지 생성을 참조하십시오.

클라우드 기반 Lambda 함수 마이그레이션

AWS IoT Greengrass 코어 SDK는 AWS SDK 프로그래밍 모델을 따르기 때문에 클라우드를 위해 개발된 Lambda 함수를 AWS IoT Greengrass 코어에서 실행되는 Lambda 함수로 손쉽게 옮길 수 있습니다.

예를 들면 다음의 Python Lambda 함수는 AWS SDK for Python (Boto3)을 사용하여 클라우드의 `some/topic` 주제에 메시지를 게시합니다.

```
import boto3

iot_client = boto3.client("iot-data")
response = iot_client.publish(
    topic="some/topic", qos=0, payload="Some payload".encode()
)
```

AWS IoT Greengrass 코어에서 이 함수를 옮기려면 `import` 명령문 및 `client` 초기화에서 모듈 이름을 `boto3`에서 `greengrasssdk`(으)로 바꿉니다(다음 예제 참조).

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

> **Note**
>
> AWS IoT Greengrass 코어 SDK는 QoS = 0인 MQTT 메시지의 전송만 지원합니다. 자세한 설명은 the section called “서비스 품질 메시지” 섹션을 참조하세요.

프로그래밍 모델 간의 유사성 덕분에 클라우드에서 Lambda 함수를 개발한 다음 최소한의 노력으로 AWS IoT Greengrass에 마이그레이션할 수 있습니다. Lambda 실행 파일은 클라우드에서 실행되지 않으므로 배포 전에 AWS SDK를 사용하여 클라우드에서 개발할 수 없습니다.

별칭 또는 버전을 기준으로 Lambda 함수 참조

Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다. 별칭은 그룹 배포 동안 버전 번호로 확인됩니다. 별칭을 사용하면 확인된 버전이 배포 당시 별칭이 가리키고 있는 버전으로 업데이트됩니다.

AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다. $LATEST 버전은 변경 불가능한 게시 함수 버전으로 제한되지 않고 언제라도 변경이 가능한데, 이는 버전 불변성이라는 AWS IoT Greengrass 원칙과 상반됩니다.

코드 변경에 따라 Lambda 함수를 계속 업데이트하기 위한 일반적인 방법은 Greengrass 그룹 및 구독에서 **`PRODUCTION`**(이)라는 별칭을 사용하는 것입니다. 새로운 버전의 Lambda 함수를 프로덕션 환경으로 승격함에 따라 별칭이 안정적인 최신 버전을 가리키도록 하고 그룹을 재배포합니다. 또한 이전 버전으로 롤백하기 위해 이 방법을 사용할 수도 있습니다.

그룹별 구성을 사용한 Lambda 함수 실행 제어

AWS IoT Greengrass은 Lambda 함수에 대한 클라우드 기반 관리를 제공합니다. Lambda 함수의 코드와 종속성은 AWS Lambda를 사용하여 관리되지만 Greengrass 그룹에서 실행될 때 Lambda 함수가 동작하는 방식을 구성할 수 있습니다.

그룹별 구성 설정

AWS IoT Greengrass은 Lambda 함수에 대해 다음과 같은 그룹별 구성 설정을 제공합니다.

시스템 사용자 및 그룹

Lambda 함수 실행에 사용되는 액세스 ID입니다. 기본적으로 Lambda 함수는 그룹의 기본 액세스 확인으로 실행됩니다. 일반적으로 이 항목은 표준 AWS IoT Greengrass 시스템 계정(ggc_user and ggc_group)입니다. 설정을 변경할 수 있으며, Lambda 함수 실행에 필요한 권한을 가진 사용자 ID 및 그룹 ID를 선택할 수 있습니다. UID와 GID를 둘 다 재정의하거나 한 필드를 비워둘 경우 다른 하나만 재정의할 수도 있습니다. 이러한 설정으로 디바이스 리소스 액세스를 한층 세부적으로 제어할

수 있습니다. Lambda 함수 실행에 권한이 사용되는 사용자와 그룹에 대해 적절한 리소스 제한, 파일 권한, 디스크 할당량으로 Greengrass 하드웨어를 구성하는 것이 좋습니다.

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

> **Important**
>
> 반드시 필요한 경우 외에는 Lambda 함수를 루트로 실행하는 것은 피하는 것이 좋습니다. 루트로 실행하면 다음과 같은 위험이 증가합니다.
>
> - Lambda 함수를 루트로 실행하면 중요한 파일을 실수로 삭제하는 등 의도치 않은 변경이 발생할 리스크가 증가합니다.
> - 악의적인 개인으로 인한 데이터 및 디바이스 위험.
> - Docker 컨테이너가 `--net=host` 및 `UID=EUID=0`와(과) 함께 실행될 경우 컨테이너 이스케이프 위험이 커집니다.
>
> 루트로 실행할 필요가 없는 경우 AWS IoT Greengrass 구성을 업데이트해 이를 활성화해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

시스템 사용자 ID(번호)

Lambda 함수 실행에 필요한 권한을 가진 사용자의 사용자 ID입니다. 이 설정은 다른 사용자 ID/그룹 ID로 실행하기를 선택할 경우에만 사용 가능합니다. AWS IoT Greengrass 코어 디바이스에서 getent passwd 명령을 사용하여 Lambda 함수 실행에 사용할 사용자 ID를 조회할 수 있습니다.

Greengrass 코어 디바이스에서 동일한 UID를 사용하여 프로세스와 Lambda 함수를 실행하는 경우, Greengrass 그룹 역할은 프로세스에 임시 보안 인증을 부여할 수 있습니다. 프로세스는 Greengrass 코어 배포에 임시 보안 인증을 사용할 수 있습니다.

시스템 그룹 ID(번호)

Lambda 함수 실행에 필요한 권한을 가진 그룹의 그룹 ID입니다. 이 설정은 다른 사용자 ID/그룹 ID로 실행하기를 선택할 경우에만 사용 가능합니다. AWS IoT Greengrass 코어 디바이스에서 getent group 명령을 사용하여 Lambda 함수 실행에 사용할 그룹 ID를 조회할 수 있습니다.

Lambda 함수 컨테이너화

그룹의 기본 컨테이너화로 Lambda 함수를 실행할지 선택하거나 이 Lambda 함수에 항상 사용할 컨테이너화를 지정합니다.

Lambda 함수의 컨테이너화 모드는 격리 수준을 결정합니다.

- 컨테이너화된 Lambda 함수는 Greengrass 컨테이너 모드에서 실행됩니다. Lambda 함수는 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경(또는 네임스페이스)에서 실행됩니다.
- 컨테이너화되지 않은 Lambda 함수는 컨테이너 없음 모드에서 실행됩니다. Lambda 함수는 격리 없이 일반 Linux 프로세스로 실행됩니다.

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

컨테이너화 없이 실행할 것이 요구되는 사용 사례가 아니라면 Lambda 함수를 Greengrass 컨테이너에서 실행하는 것이 좋습니다. Lambda 함수가 Greengrass 컨테이너에서 실행되면 연결된 로컬 디바이스 리소스를 사용하고 격리와 보안 강화에 따른 이점을 누릴 수 있습니다. 컨테이너화 변경 전에 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항”를 참조하십시오.

> **Note**
>
> 디바이스 커널 네임스페이스와 cgroup의 활성화 없이 실행하려면 모든 Lambda 함수가 컨테이너화 없이 실행되어야 합니다. 이는 그룹의 기본 컨테이너화를 설정해 간편히 달성할 수 있습니다. 자세한 내용은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.

메모리 제한

함수에 할당된 메모리입니다. 기본값은 16MB입니다.

> **Note**
>
> Lambda 함수를 컨테이너화 없이 실행하기로 변경하면 메모리 제한 설정이 삭제됩니다. 컨테이너화 없이 실행되는 Lambda 함수에는 메모리 제한이 없습니다. 컨테이너화 없이 실행되도록 Lambda 함수 또는 그룹 기본 컨테이너화 설정을 변경하면 메모리 제한 설정이 무시됩니다.

제한 시간

함수 또는 요청이 종료되기 전까지의 시간입니다. 기본값은 3초입니다.

고정

Lambda 함수는 온디맨드 또는 긴 수명 주기를 가질 수 있습니다. 기본값은 온디맨드입니다.

온디맨드 Lambda 함수는 호출 시 새 컨테이너나 재사용된 컨테이너에서 시작됩니다. 함수에 대한 요청은 사용 가능한 어떤 컨테이너에서든 처리가 가능합니다. 수명이 긴 또는 고정 Lambda 함수는 AWS IoT Greengrass 시작 후 자동으로 시작되며 고유한 컨테이너(또는 샌드박스)에서 계속해서 실행됩니다. 함수에 대한 모든 요청은 동일한 컨테이너에 의해 처리됩니다. 자세한 설명은 the section called “수명 주기 구성” 섹션을 참조하세요.

/sys 디렉터리에 대한 읽기 액세스 권한

함수가 호스트의 /sys 폴더에 액세스할 수 있는지 여부입니다. 함수가 /sys에서 디바이스 정보를 읽어야 할 때 이 권한을 사용하십시오. 기본값은 false입니다.

> **Note**
>
> 이 설정은 Lambda 함수를 컨테이너화 없이 실행할 경우 사용할 수 없습니다. Lambda 함수를 컨테이너화 없이 실행하기로 변경하면 이 설정 값이 삭제됩니다.

인코딩 유형

함수에 대한 입력 페이로드의 예상 인코딩 유형(JSON 또는 이진)입니다. 기본값은 JSON입니다.

이진 인코딩 유형은 AWS IoT Greengrass 코어 소프트웨어 v1.5.0 및 AWS IoT Greengrass 코어 SDK v1.1.0부터 지원됩니다. 디바이스 데이터와 상호 작용하는 함수에서는 이진 입력 데이터를 수락하는 것이 유용할 수 있습니다. 왜냐하면 디바이스의 하드웨어 기능이 제한되어 있기 때문에 JSON 데이터 유형을 생성하는 것이 종종 어렵거나 불가능하기 때문입니다.

> **Note**
>
> Lambda 실행 파일은 이진 인코딩 유형만 지원하며, JSON은 지원하지 않습니다.

프로세스 인수

Lambda 함수가 실행될 때 전달할 명령줄 인수입니다.

환경 변수

함수 코드 및 라이브러리에 역동적으로 설정값을 전달할 수 있는 키-값 쌍입니다. 로컬 환경 변수는 AWS Lambda 함수 환경 변수와 동일한 방식으로 작동하지만, 코어 환경에서도 사용할 수 있습니다.

Resource access policies(리소스 액세스 정책)

Lambda 함수가 액세스할 수 있는 최대 10개의 로컬 리소스, 비밀 리소스, 기계 학습 리소스 및 해당 `read-only` 또는 `read-write` 권한의 목록입니다. 콘솔에서 이러한 제휴 리소스는 리소스 탭의 그룹 구성 페이지에 나열됩니다.

컨테이너화 모드는 Lambda 함수가 로컬 디바이스와 볼륨 리소스 및 기계 학습 리소스에 액세스하는 방법에 영향을 줍니다.

- 컨테이너화되지 않은 Lambda 함수는 코어 디바이스의 파일 시스템을 통해 직접 로컬 디바이스 및 볼륨 리소스에 액세스해야 합니다.
- 컨테이너화되지 않은 Lambda 함수가 Greengrass 그룹의 기계 학습 리소스에 액세스하도록 허용하려면 기계 학습 리소스에 대한 리소스 소유자 및 액세스 권한 속성을 설정해야 합니다. 자세한 설명은 the section called “기계 학습 리소스에 액세스” 섹션을 참조하세요.

AWS IoT GreengrassAPI를 사용하여 사용자 정의 Lambda 함수의 그룹별 구성 설정을 설정하는 방법에 대한 자세한 내용은 API 참조 또는 명령 AWS IoT Greengrass Version 1 참조를 CreateFunctionDefinition참조하십시오. create-function-definitionAWS CLI Greengrass 코어에 Lambda 함수를 배포하려면 함수가 포함된 함수 정의 버전을 생성하고, 함수 정의 버전 및 기타 그룹 구성 요소를 참조하는 그룹 버전을 생성한 다음 그룹을 배포합니다.

루트로서의 Lambda 함수 실행

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

하나 이상의 Lambda 함수를 루트로 실행할 수 있기 전에는 먼저 AWS IoT Greengrass 구성을 업데이트해 지원을 활성화해야 합니다. 루트로서의 Lambda 함수 실행 지원은 기본적으로 꺼져 있습니다. Lambda 함수를 루트로서 배포 및 실행하려 하고(UID 및 GID가 0) AWS IoT Greengrass 구성을 업데이트하지 않은 경우 배포에 실패합니다. 다음과 같은 오류 메시지가 런타임 로그(*greengrass_root*/ggc/var/log/system/runtime.log)에 표시됩니다.

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
 configured to run lambdas with root permissions
```

> **Important**
>
> 반드시 필요한 경우 외에는 Lambda 함수를 루트로 실행하는 것은 피하는 것이 좋습니다. 루트로 실행하면 다음과 같은 위험이 증가합니다.

- Lambda 함수를 루트로 실행하면 중요한 파일을 실수로 삭제하는 등 의도치 않은 변경이 발생할 리스크가 증가합니다.
- 악의적인 개인으로 인한 데이터 및 디바이스 위험.
- Docker 컨테이너가 `--net=host` 및 `UID=EUID=0`와(과) 함께 실행될 경우 컨테이너 이스케이프 위험이 커집니다.

Lambda 함수를 루트로 실행하도록 허용하려면

1. AWS IoT Greengrass 디바이스에서 *greengrass-root*/config 폴더로 이동합니다.

   > **Note**
   >
   > 기본적으로 *greengrass-root*는 /greengrass 디렉터리입니다.

2. config.json 파일을 편집하여 `"allowFunctionsToRunAsRoot" : "yes"`를 `runtime` 필드에 추가합니다. 예:

   ```
   {
     "coreThing" : {
       ...
     },
     "runtime" : {
       ...
       "allowFunctionsToRunAsRoot" : "yes"
     },
     ...
   }
   ```

3. 다음 명령을 사용해 AWS IoT Greengrass를 재시작합니다.

   ```
   cd /greengrass/ggc/core
   sudo ./greengrassd restart
   ```

   이제 Lambda 함수의 사용자 ID와 그룹 ID(UID/GID)를 0으로 설정해 Lambda 함수를 루트로 실행할 수 있습니다.

루트로서의 Lambda 함수 실행을 허용하지 않으려면 `"allowFunctionsToRunAsRoot"`의 값을 `"no"`로 변경하고 AWS IoT Greengrass를 재시작할 수 있습니다.

Lambda 함수 컨테이너화 선택 시 고려 사항

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

기본적으로 Lambda 함수는 AWS IoT Greengrass 컨테이너 내부에서 실행됩니다. 컨테이너를 통해 함수와 호스트를 서로 격리해, 컨테이너 내 호스트와 함수 모두의 보안을 강화할 수 있습니다.

컨테이너화 없이 실행할 것이 요구되는 사용 사례가 아니라면 Lambda 함수를 Greengrass 컨테이너에서 실행하는 것이 좋습니다. Greengrass 컨테이너에서 Lambda 함수를 실행하면 리소스 액세스 제한을 보다 세부적으로 제어할 수 있습니다.

다음은 컨테이너화 없이 실행할 경우에 대한 몇 가지 예제 사용 사례입니다.

- 컨테이너 모드를 지원하지 않는 디바이스에서 AWS IoT Greengrass를 실행하려는 경우(예를 들어 특수 Linux 배포를 사용하거나 너무 오래된 커널 버전이 있는 경우).
- 자체 OverlayFS를 사용하는 다른 컨테이너 환경에서 Lambda 함수를 실행하려 하지만 Greengrass 컨테이너에서 실행 시 OverlayFS 충돌이 발생하는 경우.
- 배포 시 결정할 수 없거나 플러그 가능 디바이스처럼 배포 후 변경될 수 있는 경로로 로컬 리소스에 액세스할 수 있어야 합니다.
- 프로세스로서 쓰여진 레거시 애플리케이션을 컨테이너화된 Lambda 함수로 실행할 때 문제가 발생한 경우.

컨테이너화의 차이점

컨테이너화	참고
Greengrass 컨테이너	• 모든 AWS IoT Greengrass 기능은 Greengrass 컨테이너에서 Lambda 함수를 실행할 때 사용 가능합니다. • Greengrass 컨테이너에서 실행되는 Lambda 함수는 동일한 그룹 ID로 실행된다 해도 다른 Lambda 함수의 배포된 코드에 액세스할 수 없습니다. 다시 말해, 함수는 상호 격리가 강화된 상태에서 실행됩니다.

컨테이너화	참고
	• AWS IoT Greengrass 컨테이너에서 실행되는 Lambda 함수는 모든 하위 프로세스가 Lambda 함수와 동일한 컨테이너에서 실행되도록 하며, 따라서 함수 종료 시에는 하위 프로세스 또한 종료됩니다.

컨테이너 없음	• 컨테이너화되지 않은 Lambda 함수에는 다음 기능을 사용할 수 없습니다.
	  • Lambda 함수 메모리 제한.
	  • 로컬 디바이스 및 볼륨 리소스. 이러한 리소스는 Greengrass 그룹의 멤버로 액세스하는 대신 코어 디바이스에서 직접 액세스해야 합니다.
	• 컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 액세스하는 경우 리소스 소유자를 식별하고 Lambda 함수가 아닌 리소스에 대한 액세스 권한을 설정해야 합니다. 이 지원을 위해서는 AWS IoT Greengrass 코어 소프트웨어 v1.10 이상이 필요합니다. 자세한 설명은 the section called “기계 학습 리소스에 액세스” 섹션을 참조하세요.
	• Lambda 함수는 동일한 그룹 ID로 실행되는 다른 함수의 배포된 코드에 대해 읽기 전용 액세스 권한을 가집니다.
	• 다른 프로세스 세션에서나 nohup 유틸리티 같은 재정의된 SIGHUP(신호 중단) 핸들러로 하위 프로세스를 생성하는 AWS IoT Greengrass 함수는 상위 Lambda 함수가 종료되어도 에 의해 자동으로 종료되지 않습니다.

> **Note**
>
> Greengrass 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

Lambda 함수의 컨테이너화를 변경하면 배포 시 문제가 발생할 수 있습니다. 더 이상 새 컨테이너화 설정으로 사용할 수 없는 Lambda 함수에 로컬 리소스를 할당했다면 배포에 실패합니다.

- Lambda 함수를 Greengrass 컨테이너에서 실행하는 방식에서 컨테이너화 없이 실행하는 방식으로 변경하면 해당 함수에 대해 설정된 메모리 제한이 삭제됩니다. 연결된 로컬 리소스를 사용하기보다 파일 시스템에 직접 액세스해야 합니다. 배포하기 전에 연결된 모든 리소스를 제거해야 합니다.
- Lambda 함수를 컨테이너화 없이 실행하는 방식에서 컨테이너에서 실행하는 방식으로 변경하면 Lambda 함수는 파일 시스템에 대한 직접 액세스 권한을 상실합니다. 각 함수에 대한 메모리 제한을 정의하거나 16MB 기본값을 수락해야 합니다. 배포하기 전에 각 Lambda 함수에 대해 이러한 설정을 구성할 수 있습니다.

Lambda 함수의 컨테이너화 설정을 변경하려면

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 설정을 변경하려는 Lambda 함수가 포함된 그룹을 선택합니다.
3. Lambda 함수탭을 선택합니다.
4. 변경하려는 Lambda 함수에서 줄임표(...)를 선택한 다음 구성 편집을 선택합니다.
5. 컨테이너화 설정을 변경합니다. Lambda 함수가 Greengras 컨테이너에서 실행되도록 구성할 경우, 메모리 제한과 /sys 디렉터리에 대한 읽기 액세스. 권한 또한 설정해야 합니다.
6. 저장을 선택한 다음 확인을 선택하여 Lambda 함수에 변경 내용을 저장합니다.

변경 사항은 그룹이 배포될 때 적용됩니다.

API 참조에서 CreateFunctionDefinition및 CreateFunctionDefinitionVersion를 사용할 수도 있습니다. AWS IoT Greengrass 컨테이너화 설정을 변경할 경우, 다른 파라미터 또한 업데이트해야 합니다. 예를 들어 Lambda 함수를 Greengrass 컨테이너에서 실행하는 방식에서 컨테이너화 없이 실행하는 방식으로 변경한다면 `MemorySize` 파라미터를 지워야 합니다.

Greengrass 디바이스로 지원되는 격리 모드 결정

AWS IoT Greengrass 종속성 확인 프로그램을 사용해 Greengrass 디바이스가 어떤 격리 모드(Greengrass 컨테이너 있음/없음)를 지원하는지 확인할 수 있습니다.

AWS IoT Greengrass 종속성 확인 프로그램을 실행하려면

1. 리포지토리에서 AWS IoT Greengrass 종속성 검사기를 다운로드하여 실행합니다. GitHub

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

2. more가 나타나면 Spacebar 키를 눌러 다른 텍스트 페이지를 표시합니다.

modprobe 명령에 대한 자세한 내용을 보려면 터미널에서 man modprobe를 실행합니다.

그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정

이 기능은 AWS IoT Greengrass 코어 v1.8 이상에 사용할 수 있습니다.

디바이스 리소스에 대한 액세스를 자세히 제어하기 위해, 그룹에서 Lambda 함수를 실행하는 데 사용되는 기본 액세스 자격 증명을 구성할 수 있습니다. 이 설정은 Lambda 함수가 코어 디바이스에서 실행될 때 함수에 제공된 기본 권한을 결정합니다. 그룹에 있는 개별 함수의 설정을 재정의하려면 함수의 Run as(실행 방식) 속성을 사용할 수 있습니다. 자세한 내용은 실행 방식을 참조하십시오.

기본 AWS IoT Greengrass 코어 소프트웨어를 실행하기 위해 이 그룹 수준 설정이 사용되기도 합니다. 이 설정은 메시지 라우팅, 로컬 섀도우 동기화, 자동 IP 주소 감지 등의 작업을 관리하는 시스템 Lambda 함수로 이루어집니다.

표준 AWS IoT Greengrass 시스템 계정(ggc_user 및 ggc_group)으로 실행되거나 다른 사용자나 그룹의 권한을 사용하도록 기본 액세스 자격 증명을 구성할 수 있습니다. 사용자 정의 또는 시스템 Lambda 함수 실행에 권한이 사용되는 사용자와 그룹에 대해 적절한 리소스 제한, 파일 권한, 디스크 할당량으로 Greengrass 하드웨어를 구성하는 것이 좋습니다.

AWS IoT Greengrass 그룹의 기본 액세스 자격 증명을 수정하려면

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 설정을 변경하려는 그룹을 선택합니다.
3. Lambda 함수 탭을 선택하고 기본 Lambda 함수 런타임 환경 섹션에서 편집을 선택합니다.
4. 기본 Lambda 함수 런타임 환경 편집 페이지의 기본 시스템 사용자 및 그룹에서 다른 사용자 ID/그룹 ID를 선택합니다.

   이 옵션을 선택하면 시스템 사용자 ID (숫자) 및 시스템 그룹 ID (숫자) 필드가 표시됩니다.

5. 사용자 ID, 그룹 ID 또는 둘 다 입력합니다. 필드를 비워 두면 해당 Greengrass 시스템 계정(ggc_user 또는 ggc_group)이 사용됩니다.

   - 시스템 사용자 ID(번호)에 그룹에서 Lambda 함수를 실행하는 데 기본적으로 사용하려는 권한을 가진 사용자의 사용자 ID를 입력합니다. getent passwd 명령을 AWS IoT Greengrass 디바이스에 사용하여 사용자 ID를 조회할 수 있습니다.
   - 시스템 그룹 ID(번호)에 그룹에서 Lambda 함수를 실행하는 데 기본적으로 사용하려는 권한을 가진 그룹의 그룹 ID를 입력합니다. getent group 명령을 AWS IoT Greengrass 디바이스에 사용하여 그룹 ID를 조회할 수 있습니다.

   > **Important**
   >
   > 루트 사용자로 실행하면 데이터 및 디바이스의 위험이 커집니다. 비즈니스 사례에 필요하지 않으면 루트(UID/GID=0)로 실행하지 마십시오. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

변경 사항은 그룹이 배포될 때 적용됩니다.

그룹 내 Lambda 함수의 기본 컨테이너화 설정

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

Greengrass 그룹의 컨테이너화 설정은 그룹의 Lambda 함수에 대한 기본 컨테이너화를 결정합니다.

- Greengrass 컨테이너 모드에서 Lambda 함수는 기본적으로 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다.
- 컨테이너 없음 모드에서 Lambda 함수는 기본적으로 일반 Linux 프로세스로 실행됩니다.

그룹 설정을 수정해 그룹 내 Lambda 함수의 기본 컨테이너화를 지정할 수 있습니다. Lambda 함수를 그룹 기본값과 다른 컨테이너화로 실행하도록 하려면 그룹 내 하나 이상의 함수에 대해 이 설정을 재정의할 수 있습니다. 컨테이너화 설정 변경 전에 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항”를 참조하십시오.

> **Important**
>
> 그룹의 기본 컨테이너화를 변경하려고 하지만 서로 다른 컨테이너화를 사용하는 함수가 하나 이상인 경우, 그룹 설정 변경 전에 Lambda 함수 설정을 변경합니다. 그룹 컨테이너화 설정을

먼저 변경할 경우, 메모리 제한과 /sys 디렉터리에 대한 읽기 액세스 권한 설정의 값이 삭제됩니다.

AWS IoT Greengrass 그룹의 컨테이너화 설정을 변경하려면

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 설정을 변경하려는 그룹을 선택합니다.
3. Lambda 함수탭을 선택합니다.
4. 기본 Lambda 함수 런타임 환경에서 편집을 선택합니다.
5. 기본 Lambda 함수 런타임 환경 편집 페이지의 기본 Lambda 함수 컨테이너화에서 컨테이너화 설정을 변경합니다.
6. 저장을 선택합니다.

변경 사항은 그룹이 배포될 때 적용됩니다.

Greengrass Lambda 함수를 위한 통신 흐름

Lambda 함수는 AWS, 로컬 서비스 및 클라우드 서비스(AWS IoT Greengrass 서비스 포함)의 다른 구성원들과의 여러 통신 방법을 지원합니다.

MQTT 메시지를 사용한 통신

Lambda 함수는 구독에 의해 제어되는 게시-구독 패턴을 사용하여 MQTT 메시지를 전송하고 수신할 수 있습니다.

이러한 통신 흐름을 통해 Lambda 함수는 메시지를 다음 엔터티와 교환할 수 있습니다.

- 그룹 내 클라이언트 디바이스.
- 그룹 내의 커넥터.
- 그룹의 기타 Lambda 함수입니다.
- AWS IoT.
- 로컬 디바이스 섀도우 서비스.

구독은 메시지 소스와 메시지 대상, 소스에서 대상까지의 메시지 라우팅에 사용된 주제(제목)를 정의합니다. Lambda 함수에 게시되는 메시지는 함수의 등록 핸들러에 전달됩니다. 구독은 보안 강화 및 예측

가능한 상호 작용을 지원합니다. 자세한 설명은 the section called “MQTT 메시징 워크플우로의 관리형 구독” 섹션을 참조하세요.

> **Note**
>
> 코어가 오프라인 상태일 때 Lambda 함수는 디바이스, 커넥터, 기타 함수 및 로컬 섀도우와 메시지를 교환할 수 있지만, AWS IoT에 대한 메시지는 대기됩니다. 자세한 설명은 the section called “MQTT 메시지 대기열” 섹션을 참조하세요.

기타 통신 흐름

- Lambda 함수는 로컬 디바이스 및 볼륨 리소스와 코어 디바이스의 기계 학습 모델과 상호 작용하기 위해 플랫폼 고유의 운영 체제 인터페이스를 사용합니다. 예를 들어 Python 함수에서 os 모듈의 `open` 메서드를 사용할 수 있습니다. 함수가 리소스에 액세스하도록 허용하려면 함수가 해당 리소스와 연계되고 `read-only` 또는 `read-write` 권한을 부여 받아야 합니다. AWS IoT Greengrass 코어 버전 가용성을 포함해 자세한 내용은 로컬 리소스에 액세스 및 the section called “Lambda 함수 코드에서 기계 학습 리소스에 액세스” 를 참조하십시오.

  > **Note**
  >
  > 컨테이너화 없이 Lambda 함수를 실행하면 연결된 로컬 디바이스 및 볼륨 리소스를 사용할 수 없으며 해당 리소스에 직접 액세스해야 합니다.

- Lambda 함수는 AWS IoT Greengrass 코어 SDK에서 `Lambda` 클라이언트를 사용하여 Greengrass 그룹의 다른 Lambda 함수를 호출할 수 있습니다.
- Lambda 함수는 AWS SDK를 사용해 AWS 서비스와 통신할 수 있습니다. 자세한 내용은 AWS SDK를 참조하세요.
- Lambda 함수는 타사 인터페이스를 사용하여 클라우드 기반 Lambda 함수처럼 외부 클라우드 서비스와 통신할 수 있습니다.

> **Note**
>
> Lambda 함수는 코어가 오프라인 상태일 때 AWS 또는 기타 클라우드 서비스와 통신을 할 수 없습니다.

입력 MQTT 주제(또는 제목) 검색

AWS IoT Greengrass는 구독을 사용하여 그룹 내의 디바이스, Lambda 함수 및 커넥터와 AWS IoT 또는 로컬 섀도우 서비스 간의 MQTT 메시지 교환을 제어합니다. 구독은 메시지 원본, 메시지 대상 및 메시지 라우팅에 사용되는 MQTT 주제를 정의합니다. 대상이 Lambda 함수이면 소스가 메시지를 게시할 때 함수 핸들러가 호출됩니다. 자세한 설명은 the section called “MQTT 메시지를 사용한 통신” 섹션을 참조하세요.

다음 예에서는 Lambda 함수가 핸들러에 전달되는 `context`에서 입력 항목을 가져올 수 있는 방법을 보여 줍니다. 컨텍스트 계층 구조(`context.client_context.custom['subject']`)에서 `subject` 키에 액세스하여 이 작업을 수행합니다. 또한 예제에서는 입력 JSON 메시지를 구문 분석한 후 구문 분석된 주제와 메시지를 게시합니다.

> **Note**
>
> AWS IoT Greengrass API에서 구독의 주제는 `subject` 속성으로 나타냅니다.

```
import greengrasssdk
import logging

client = greengrasssdk.client('iot-data')

OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
```

```
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
input_message)
        logging.info(response)
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return
```

함수를 테스트하려면 기본 구성 설정을 사용하여 그룹에 함수를 추가하십시오. 그리고 나서 다음 구독을 추가하고 그룹을 배포합니다. 지침은 the section called “모듈 3(1부): AWS IoT Greengrass의 Lambda 함수”을(을) 참조하십시오.

대상	
선택	
(필	ource
)	터

IoT	st/
클라	opic
주제	_message

IoT	st/
클라	opic
주제	_results

배포가 완료되면 함수를 호출합니다.

1. AWS IoT 콘솔에서 MQTT 테스트 클라이언트 페이지를 엽니다.
2. 주제 구독 탭을 선택하여 `test/topic_results` 주제를 구독하십시오.
3. 주제에 게시 탭을 선택하여 `test/input_message` 주제에 메시지를 게시하십시오. 이 예제에서는 `test-key` 속성을 JSON 메시지에 포함해야 합니다.

```
{
  "test-key": "Some string value"
}
```

성공하면 함수가 입력 주제에 메시지 문자열을 `test/topic_results` 주제에 게시합니다.

Greengrass Lambda 함수의 라이프사이클 구성

Lambda 함수 수명 주기에 따라 함수가 시작되는 시점과 컨테이너를 생성 및 사용하는 방법이 결정됩니다. 또한 수명 주기는 함수 핸들러 외부에 있는 변수 및 사전 처리 로직이 보관되는 방법을 결정합니다.

AWS IoT Greengrass은(는) 온디맨드(기본값) 또는 수명이 긴 수명 주기를 지원합니다.

- 온디맨드 함수는 호출될 때 시작되고 실행할 작업이 남아 있지 않을 때 종료됩니다. 기존 컨테이너를 재사용할 수 있는 경우가 아닌 한, 함수를 호출하면 호출을 처리하기 위해 별도의 컨테이너(또는 샌드박스)가 생성됩니다. 함수에 전송되는 데이터는 어떤 컨테이너에 의해서든 가져올 수 있습니다.

  온디맨드 함수에 대한 다중 호출은 병렬로 실행될 수 있습니다.

  함수 핸들러 외부에서 정의된 변수 및 사전 처리 로직은 새 컨테이너 생성 시 보관되지 않습니다.

- 수명이 긴(또는 고정) 함수는 AWS IoT Greengrass 코어가 시작될 때 자동으로 시작되며 단일 컨테이너에서 실행됩니다. 함수에 전송되는 모든 데이터는 동일한 컨테이너가 가져옵니다.

  이전 호출이 실행될 때까지 다중 호출은 대기 상태가 됩니다.

  함수 핸들러 외부에서 정의된 변수 또는 사전 처리 로직은 핸들러를 호출할 때마다 보관됩니다.

  수명이 긴 Lambda 함수는 초기 입력 없이 작업을 시작해야 할 때 유용합니다. 예를 들어 수명이 긴 함수는 함수가 디바이스 데이터 수신을 시작할 때 준비가 되도록 ML 모델을 로드해서 처리를 시작할 수 있습니다.

> **Note**
>
> 수명이 긴 함수는 핸들러 호출과 관련해 제한 시간이 있다는 점에 유의하십시오. 무한정 실행되는 코드를 실행하고 싶은 경우에는 핸들러 외부에서 이를 시작해야 합니다. 핸들러 외부에 함수의 초기화 완료를 방해하는 차단 코드가 없는지 확인하십시오.

코어가 중지되지 않으면(예: 그룹 배포 또는 디바이스 재부팅 도중) 이 함수가 실행됩니다. 또는 함수가 오류 상태(핸들러 시간 초과, 확인되지 않은 예외 또는 메모리 제한을 초과하는 경우 등)가 됩니다.

컨테이너 재사용에 대한 자세한 내용은 AWS 컴퓨팅 블로그의 AWS Lambda에서 컨테이너 재사용 이해하기를 참조하십시오.

Lambda 실행 파일

이 기능은 AWS IoT Greengrass 코어 v1.6 이상에 사용할 수 있습니다.

Lambda 실행 파일은(는) 코어 환경에서 이진 코드를 실행하는 데 사용할 수 있는Greengrass Lambda 함수의 유형입니다. 이를 통해 디바이스별 기능을 기본적으로 실행할 수 있으며 컴파일된 코드의 공간이 더 작아지는 이점을 누릴 수 있습니다. Lambda 실행 파일은 이벤트에 의해 간접적으로 호출되고, 다른 함수를 간접적으로 호출하고, 로컬 리소스에 액세스할 수 있습니다.

Lambda 실행 파일은 이진 인코딩 유형만 지원하지만(JSON는 지원하지 않음), 그렇지 않고 다른 Lambda 함수처럼 Greengrass 그룹에서 이들을 관리하고 배포할 수 있습니다. 그러나 을(를) 생성하는 프로세스는 Python, Java 및 Node.js Lambda 함수를 생성하는 프로세스와는 다릅니다.

- AWS Lambda 콘솔을 사용하여 을(를) Lambda 만들 수는(또는 관리할 수는) 없습니다. Lambda 실행 파일은 AWS Lambda API로만 생성할 수 있습니다.
- 함수 코드를 AWS Lambda에 C용 AWS IoT Greengrass Core SDK를 포함하는 컴파일된 실행 파일로 업로드합니다.
- 함수 핸들러로 실행 파일 이름을 지정합니다.

Lambda 실행 파일은 함수 코드에 특정한 호출 및 프로그래밍 패턴을 구현해야 합니다. 예를 들어 `main` 메서드는 다음과 같이 해야 합니다.

- `gg_global_init`를 직접 호출하여 Greengrass 내부 전역 변수를 초기화합니다. 스레드를 생성하기 전에, 그리고 다른 AWS IoT Greengrass 코어 SDK 함수를 호출하기 전에 이 함수를 호출해야 합니다.
- `gg_runtime_start`을 직접 호출하여 Greengrass Lambda 런타임에 함수 핸들러를 등록합니다. 초기화 중에 이 함수를 호출해야 합니다. 이 함수를 호출하면 현재 스레드가 실행 시간에 사용됩니다. 선택 사항인 `GG_RT_OPT_ASYNC` 파라미터는 이 함수에게 차단하는 대신, 실행 시간을 위한 새 스레드를 생성하라고 지시합니다. 이 함수는 `SIGTERM` 핸들러를 사용합니다.

다음 스니펫은 simple_handler.c 코드 예제의 `main` 메서드입니다. GitHub

```
int main() {
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

요건, 제약 조건 및 기타 구현 세부 정보에 대한 자세한 내용은 AWS IoT GreengrassC용 코어 SDK을 참조하십시오.

Lambda 실행 파일 생성

SDK와 함께 코드를 컴파일하고 난 후에는 AWS Lambda API를 사용하여 Lambda 함수를 생성하고 컴파일된 실행 파일을 업로드합니다.

> **Note**
>
> C89와 호환 가능한 컴파일러로 함수를 컴파일해야 합니다.

다음 예제에서는 create-function CLI 명령을 사용하여 Lambda 실행 파일을 생성합니다. 이 명령은 다음을 지정합니다.

- 핸들러를 위한 실행 파일의 이름. 컴파일된 실행 파일의 정확한 이름이어야 합니다.
- 컴파일된 실행 파일이 포함된 `.zip` 파일에 대한 경로.
- 실행 시간을 위한 `arn:aws:greengrass:::runtime/function/executable`. 모든 Lambda 실행 파일의 런타임입니다.

> **Note**
>
> `role`의 경우에는 모든 Lambda 실행 파일의 ARN을 지정할 수 있습니다. AWS IoT Greengrass는 이 역할을 사용하지 않지만, 함수를 생성하려면 이 파라미터가 필요합니다. Lambda 실행 파일에 대한 자세한 내용은 AWS Lambda 개발자 가이드의 AWS LambdaLambda 실행 파일을 참조하세요.

```
aws lambda create-function \
--region aws-region \
--function-name function-name \
--handler executable-name \
--role role-arn \
--zip-file fileb://file-name.zip \
--runtime arn:aws:greengrass:::runtime/function/executable
```

그런 다음 AWS Lambda API를 사용하여 버전을 게시하고 별칭을 생성합니다.

- 함수 버전을 게시하려면 게시-버전을 사용하십시오.

  ```
  aws lambda publish-version \
  --function-name function-name \
  --region aws-region
  ```

- 방금 게시한 버전을 가리키는 별칭을 생성하려면 생성-별칭을 사용하십시오. Greengrass 그룹에 추가할 때는 별칭을 기준으로 Lambda 함수를 참조하는 것이 좋습니다.

  ```
  aws lambda create-alias \
  --function-name function-name \
  --name alias-name \
  --function-version version-number \
  --region aws-region
  ```

> **Note**
>
> AWS Lambda 콘솔에는 Lambda 실행 파일이 표시되지 않습니다. 함수 코드를 업데이트하려면 AWS Lambda API를 사용해야 합니다.

그리고 Greengrass 그룹에 Lambda 실행 파일을 추가하고 그룹별 설정에서 이진 입력 데이터를 수용하도록 이를 구성한 다음, 그룹을 배포합니다. 이 작업은 AWS IoT Greengrass 콘솔에서 또는 AWS IoT Greengrass API를 사용하여 수행할 수 있습니다.

도커 컨테이너에서의 AWS IoT Greengrass 실행

AWS IoT Greengrass를 도커 컨테이너에서 실행하도록 구성할 수 있습니다.

Amazon CloudFront를 통해 AWS IoT Greengrass 코어 소프트웨어와 종속성이 설치된 Dockerfile을 다운로드할 수 있습니다. 다른 플랫폼 아키텍처에서 실행되도록 Docker 이미지를 수정하거나 Docker 이미지의 크기를 줄이려면 Docker 패키지 다운로드에 있는 `README` 파일을 참조하십시오.

AWS IoT Greengrass 실험을 시작할 수 있도록 AWS는 AWS IoT Greengrass Core 소프트웨어 및 종속성이 설치된 미리 빌드된 도커 이미지도 제공합니다. Docker Hub 또는 Amazon Elastic Container Registry(Amazon ECR)에서 이미지를 다운로드할 수 있습니다. 이러한 미리 빌드된 이미지는 Amazon Linux 2(x86_64) 및 Alpine Linux(x86_64, Armv7l 또는 AArch64) 기본 이미지를 사용합니다.

> **Important**
>
> 2022년 6월 30일에 AWS IoT Greengrass은(는) Amazon Elastic Container Registry(Amazon ECR) 및 Docker Hub에 게시된 AWS IoT Greengrass 코어 소프트웨어 v1.x Docker 이미지에 대한 유지 관리를 종료하였습니다. 유지 관리 종료 후 1년이 되는 2023년 6월 30일까지 Amazon ECR 및 Docker Hub에서 이 Docker 이미지를 계속 다운로드할 수 있습니다. 그러나 AWS IoT Greengrass Core 소프트웨어 v1.x Docker 이미지는 2022년 6월 30일에 유지 관리가 종료된 이후 더 이상 보안 패치나 버그 수정을 받지 않습니다. 이러한 Docker 이미지를 사용하는 프로덕션 워크로드를 실행하는 경우 AWS IoT Greengrass이(가) 제공하는 Dockerfile을 사용하여 자체 Docker 이미지를 구축하는 것이 좋습니다. 자세한 내용은 AWS IoT Greengrass Docker 소프트웨어 섹션을 참조하세요.

이 주제에서는 에서 AWS IoT Greengrass Docker 이미지를 Amazon ECR 다운로드하는 방법과 이를 Windows, macOS 또는 Linux(x86_64) 플랫폼에서 실행하는 방법을 설명합니다. 이 주제에는 다음 단계가 포함되어 있습니다.

1. Amazon ECR에서 AWS IoT Greengrass 컨테이너 이미지 가져오기
2. Greengrass 그룹 및 코어 생성 및 구성
3. 로컬로 AWS IoT Greengrass 실행
4. 그룹에 대한 "컨테이너 없음" 컨테이너화 구성

5. Docker 컨테이너너에 Lambda 함수 배포
6. (선택 사항) Docker 컨테이너에서 Greengrass와 상호 작용하는 디바이스 배포

도커 컨테이너에서 AWS IoT Greengrass를 실행할 때 다음 기능은 지원되지 않습니다.

- Greengrass 컨테이너 모드에서 실행되는 커넥터 Docker 컨테이너에서 커넥터를 실행하려면 커넥터가 컨테이너 없음 모드로 실행되어야 합니다. 컨테이너 없음 모드를 지원하는 커넥터를 찾으려면 the section called “AWS에서 제공한 Greengrass 커넥터” 단원을 참조하십시오. 이러한 커넥터 중 일부에는 컨테이너 없음으로 설정해야 하는 격리 모드 파라미터가 있습니다.
- 로컬 디바이스 및 볼륨 리소스. Docker 컨테이너에서 실행되는 사용자 정의 Lambda 함수는 코어의 디바이스 및 볼륨에 직접 액세스해야 합니다.

Greengrass 그룹에 대한 Lambda 런타임 환경을 컨테이너 없음으로 설정하면 이러한 기능이 지원되지 않습니다. 이 설정은 Docker 컨테이너에서 AWS IoT Greengrass를 실행하는 데 필수적입니다.

필수 조건

이 자습서를 시작하기 전에 다음 작업을 수행해야 합니다.

- 선택한 AWS Command Line Interface(AWS CLI) 버전에 따라 호스트 컴퓨터에 다음 소프트웨어 및 버전을 설치해야 합니다.

  AWS CLI version 2

  - Docker, 버전 18.09 이상. 이전 버전도 작동할 수 있지만 18.09 이상을 사용하는 것이 좋습니다.
  - AWS CLI 버전 2.0.0 이상
    - AWS CLI 버전 2를 설치하려면 AWS CLI 버전 2 설치를 참조하십시오.
    - AWS CLI을(를) 구성하려면 AWS CLI 구성을 참조하십시오.

    > **Note**
    >
    > Windows 컴퓨터에서 최신 AWS CLI 버전 2로 업그레이드하려면 MSI 설치 프로세스를 반복해야 합니다.

  AWS CLI version 1

  - Docker, 버전 18.09 이상. 이전 버전도 작동할 수 있지만 18.09 이상을 사용하는 것이 좋습니다.

- Python, 버전 3.6 이상.
- pip 버전 18.1 이상
- AWS CLI 버전 1.17.10 이상
  - AWS CLI 버전 1을 설치하려면 AWS CLI 버전 1 설치를 참조하십시오.
  - AWS CLI을(를) 구성하려면 AWS CLI 구성을 참조하십시오.
  - AWS CLI 버전 1의 최신 버전으로 업데이트하려면 다음 명령을 실행합니다.

```
pip install awscli --upgrade --user
```

> **Note**
>
> Windows에서 AWS CLI 버전 1의 MSI 설치를 사용하는 경우 다음 사항에 유의하십시오.
>
> - 이 AWS CLI 버전 1 설치에서 botocore를 설치하지 못하는 경우 Python 및 pip 설치를 사용해 봅니다.
> - 최신 AWS CLI 버전 1로 업그레이드하려면 MSI 설치 프로세스를 반복해야 합니다.

- Amazon Elastic Container Registry(Amazon ECR) 리소스에 액세스하려면 다음 권한을 부여해야 합니다.
  - Amazon ECR의 요구 사항에 따라 사용자가 레지스트리에 대해 인증하고 Amazon ECR 리포지토리에서 이미지를 푸시 또는 풀하기 전에 AWS Identity and Access Management (IAM) 정책을 통해 `ecr:GetAuthorizationToken` 권한을 부여해야 합니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR 리포지토리 정책 예시 및 One Amazon ECR 리포지토리 액세스를 참조하십시오.

1단계: Amazon ECR에서 AWS IoT Greengrass 컨테이너 이미지 가져오기

AWS는 AWS IoT Greengrass Core 소프트웨어가 설치된 도커 이미지를 제공합니다.

> **Warning**
>
> AWS IoT Greengrass 코어 소프트웨어 v1.11.6부터 Greengrass Docker 이미지에 Python 2.7이 더 이상 포함되지 않습니다. Python 2.7이 2020년에 수명을 다했고 더 이상 보안 업데이트가 이루어지지 않기 때문입니다. 이러한 Docker 이미지로 업데이트하기로 선택한 경우, 업데이트를 프로덕션 디바이스에 배포하기 전에 애플리케이션이 새 Docker 이미지와 호환되는지 확

인하는 것을 권장합니다. Greengrass Docker 이미지를 사용하는 애플리케이션에 Python 2.7이 필요한 경우, Python 2.7을 애플리케이션에 포함하도록 Greengrass Dockerfile을 수정할 수 있습니다.

Amazon ECR에서 `latest` 이미지를 끌어오는 방법을 보여 주는 단계를 알아보려면 해당 운영 체제를 선택하십시오.

컨테이너 이미지 가져오기(Linux)

컴퓨터 터미널에서 다음 명령을 실행합니다.

1. Amazon ECR에서 AWS IoT Greengrass 레지스트리에 로그인합니다.

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   성공하면 `Login Succeeded` 출력이 인쇄됩니다.

2. AWS IoT Greengrass 컨테이너 이미지를 가져옵니다.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > `latest` 이미지에는 Amazon Linux 2 기본 이미지에 설치된 안정적인 최신 버전의 AWS IoT Greengrass Core 소프트웨어가 포함되어 있습니다. 또한 리포지토리에서 다른 이미지를 끌어올 수도 있습니다. 사용 가능한 모든 이미지에 대한 태그를 찾으려면 Docker Hub의 태그 페이지를 확인하거나 aws ecr list-images 명령을 사용하십시오. 예:
   >
   > ```
   > aws ecr list-images --region us-west-2 --registry-id 216483018798 --
   > repository-name aws-iot-greengrass
   > ```

3. symlink 및 hardlink 보호를 활성화합니다. 컴퓨터에서 AWS IoT Greengrass 실행을 실험하는 경우 현재 부트에 대해서만 설정을 활성화할 수 있습니다.

> **Note**
>
> sudo를 사용해 이러한 명령을 실행해야 할 수 있습니다.

- 현재 부트에 대해서만 설정을 활성화하려면:

```
echo 1 > /proc/sys/fs/protected_hardlinks
echo 1 > /proc/sys/fs/protected_symlinks
```

- 재시작 후에도 유지되도록 설정을 활성화하려면:

```
echo '# AWS IoT Greengrass' >> /etc/sysctl.conf
echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf
echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf

sysctl -p
```

4. IPv4 네트워크 전달을 활성화합니다. 이 기능은 AWS IoT Greengrass 클라우드 배포 및 MQTT 통신이 Linux에서 작동하는 데 필요합니다. `/etc/sysctl.conf` 파일에서 `net.ipv4.ip_forward`를 1로 설정한 다음 `sysctls`를 다시 로드합니다.

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

> **Note**
>
> nano 대신 사용자가 선택한 편집기를 사용할 수 있습니다.

컨테이너 이미지 가져오기(macOS)

컴퓨터 터미널에서 다음 명령을 실행합니다.

1. Amazon ECR에서 AWS IoT Greengrass 레지스트리에 로그인합니다.

```
aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

성공하면 `Login Succeeded` 출력이 인쇄됩니다.

2. AWS IoT Greengrass 컨테이너 이미지를 가져옵니다.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> `latest` 이미지에는 Amazon Linux 2 기본 이미지에 설치된 안정적인 최신 버전의 AWS IoT Greengrass Core 소프트웨어가 포함되어 있습니다. 또한 리포지토리에서 다른 이미지를 끌어올 수도 있습니다. 사용 가능한 모든 이미지에 대한 태그를 찾으려면 Docker Hub의 태그 페이지를 확인하거나 aws ecr list-images 명령을 사용하십시오. 예:
>
> ```
> aws ecr list-images --region us-west-2 --registry-id 216483018798 --
> repository-name aws-iot-greengrass
> ```

컨테이너 이미지 가져오기(Windows)

명령 프롬프트에서 다음 명령을 실행합니다. Windows에서 도커 명령을 사용하려면 먼저 도커 데스크톱이 실행 중이어야 합니다.

1. Amazon ECR에서 AWS IoT Greengrass 레지스트리에 로그인합니다.

```
aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

성공하면 `Login Succeeded` 출력이 인쇄됩니다.

2. AWS IoT Greengrass 컨테이너 이미지를 가져옵니다.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> `latest` 이미지에는 Amazon Linux 2 기본 이미지에 설치된 안정적인 최신 버전의 AWS IoT Greengrass Core 소프트웨어가 포함되어 있습니다. 또한 리포지토리에서 다른 이

미지를 끌어올 수도 있습니다. 사용 가능한 모든 이미지에 대한 태그를 찾으려면 Docker Hub의 태그 페이지를 확인하거나 aws ecr list-images 명령을 사용하십시오. 예:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

2단계: Greengrass 그룹 및 코어 생성 및 구성

Docker 이미지에는 AWS IoT Greengrass 코어 소프트웨어가 설치되어 있지만, Greengrass 그룹 및 코어를 생성해야 합니다. 이 단계에는 인증서와 코어 구성 파일을 다운로드하는 작업도 포함됩니다.

- the section called "모듈 2: AWS IoT Greengrass 코어 소프트웨어 설치" 단원의 단계를 따르십시오. 다운로드하는 단계를 건너뛰고 AWS IoT Greengrass 코어 소프트웨어를 실행합니다. 소프트웨어와 해당 런타임 종속성은 Docker 이미지에 이미 설치되어 있습니다.

3단계: 로컬로 AWS IoT Greengrass 실행

그룹이 구성된 후에는 코어를 구성하고 시작할 준비가 된 것입니다. 이렇게 하는 방법을 보여 주는 단계를 알아보려면 운영 체제를 선택하십시오.

로컬로 Greengrass 실행(Linux)

컴퓨터 터미널에서 다음 명령을 실행합니다.

1. 디바이스의 보안 리소스를 위한 폴더를 생성하고 인증서와 키를 이 폴더로 옮깁니다. 다음 명령을 실행합니다. *path-to-security-files*을 보안 리소스 경로로 바꾸고 *certificateId*를 파일 이름의 인증서 ID로 대체합니다.

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. 디바이스 구성을 위한 폴더를 만들고 AWS IoT Greengrass 코어 구성 파일을 이 폴더로 옮깁니다. 다음 명령을 실행합니다. *path-to-config-file*를 구성 파일의 경로로 바꿉니다.

   ```
   mkdir /tmp/config
   ```

```
mv path-to-config-file/config.json /tmp/config
```

3. AWS IoT Greengrass를 시작하고 Docker 컨테이너에서 인증서와 구성 파일을 바인드 탑재합니다.

   인증서와 구성 파일의 압축을 푼 경로로 /tmp를 바꿉니다.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
   --entrypoint /greengrass-entrypoint.sh \
   -v /tmp/certs:/greengrass/certs \
   -v /tmp/config:/greengrass/config \
   -p 8883:8883 \
   216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   출력은 다음 예시와 같은 형식이어야 합니다.

   ```
   Setting up greengrass daemon
   Validating hardlink/softlink protection
   Waiting for up to 30s for Daemon to start

   Greengrass successfully started with PID: 10
   ```

로컬로 Greengrass 실행(macOS)

컴퓨터 터미널에서 다음 명령을 실행합니다.

1. 디바이스의 보안 리소스를 위한 폴더를 생성하고 인증서와 키를 이 폴더로 옮깁니다. 다음 명령을 실행합니다. *path-to-security-files*을 보안 리소스 경로로 바꾸고 *certificateId*를 파일 이름의 인증서 ID로 대체합니다.

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. 디바이스 구성을 위한 폴더를 만들고 AWS IoT Greengrass 코어 구성 파일을 이 폴더로 옮깁니다. 다음 명령을 실행합니다. *path-to-config-file*를 에이전트 구성 파일의 경로로 바꿉니다.

   ```
   mkdir /tmp/config
   ```

```
mv path-to-config-file/config.json /tmp/config
```

3. AWS IoT Greengrass를 시작하고 Docker 컨테이너에서 인증서와 구성 파일을 바인드 탑재합니다.

   인증서와 구성 파일의 압축을 푼 경로로 /tmp를 바꿉니다.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
   --entrypoint /greengrass-entrypoint.sh \
   -v /tmp/certs:/greengrass/certs \
   -v /tmp/config:/greengrass/config \
   -p 8883:8883 \
   216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   출력은 다음 예시와 같은 형식이어야 합니다.

   ```
   Setting up greengrass daemon
   Validating hardlink/softlink protection
   Waiting for up to 30s for Daemon to start

   Greengrass successfully started with PID: 10
   ```

로컬로 Greengrass 실행(Windows)

1. 디바이스의 보안 리소스를 위한 폴더를 생성하고 인증서와 키를 이 폴더로 옮깁니다. 명령 프롬프트에서 다음 명령을 실행합니다. *path-to-security-files*을 보안 리소스 경로로 바꾸고 *certificateId*를 파일 이름의 인증서 ID로 대체합니다.

   ```
   mkdir C:\Users\%USERNAME%\Downloads\certs
   move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
   ```

2. 디바이스 구성을 위한 폴더를 만들고 AWS IoT Greengrass 코어 구성 파일을 이 폴더로 옮깁니다. 명령 프롬프트에서 다음 명령을 실행합니다. *path-to-config-file*를 에이전트 구성 파일의 경로로 바꿉니다.

```
mkdir C:\Users\%USERNAME%\Downloads\config
move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
```

3. AWS IoT Greengrass를 시작하고 Docker 컨테이너에서 인증서와 구성 파일을 바인드 탑재합니다. 명령 프롬프트에서 다음 명령을 실행합니다.

```
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs
 -v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883
 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

도커에서 `C:\` 드라이브를 도커 데몬과 공유하라는 메시지가 나타나면 이 작업을 허용하여 도커 컨테이너 내부에 `C:\` 디렉터리를 바인드 탑재합니다. 자세한 내용은 도커 설명서의 공유 드라이브를 참조하십시오.

출력은 다음 예시와 같은 형식이어야 합니다.

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

> **Note**
>
> 컨테이너가 셸을 열지 않고 즉시 종료되는 경우 이미지 시작 시 Greengrass 런타임 로그를 바인드 탑재하여 문제를 디버그할 수 있습니다. 자세한 내용은 the section called “도커 컨테이너 외부의 Greengrass 런타임 로그를 유지하려면” 섹션을 참조하세요.

4단계: Greengrass 그룹에 대한 "컨테이너 없음" 컨테이너화 구성

Docker 컨테이너에서 AWS IoT Greengrass를 실행하면 모든 Lambda 함수가 컨테이너화 없이 실행되어야 합니다. 이 단계에서는 그룹의 기본 컨테이너화를 컨테이너 없음으로 설정합니다. 처음으로 그룹을 배포하기 전에 이 작업을 수행해야 합니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.

2. 설정을 변경하려는 그룹을 선택합니다.
3. Lambda 함수탭을 선택합니다.
4. 기본 Lambda 함수 런타임 환경에서 편집을 선택합니다.
5. 기본 Lambda 함수 런타임 환경 편집의 기본 Lambda 함수 컨테이너화에서 컨테이너화 설정을 변경합니다.
6. Save를 선택합니다.

변경 사항은 그룹이 배포될 때 적용됩니다.

자세한 내용은 the section called “그룹 내 Lambda 함수의 기본 컨테이너화 설정” 섹션을 참조하세요.

> **Note**
>
> 기본적으로 Lambda 함수는 그룹 컨테이너화 설정을 사용합니다. AWS IoT Greengrass이(가) Docker 컨테이너에서 실행 중일 때 Lambda 함수에 컨테이너 없음 설정을 재정의하면 배포가 실패합니다.

5단계: AWS IoT Greengrass Docker 컨테이너에 Lambda 함수 배포

수명이 긴 Lambda 함수를 Greengrass Docker 컨테이너에 배포할 수 있습니다.

- the section called “모듈 3(1부): AWS IoT Greengrass의 Lambda 함수”의 단계를 따라 수명이 긴 Hello World Lambda 함수를 컨테이너에 배포합니다.

6단계: (선택 사항) Docker 컨테이너에서 실행 중인 Greengrass와 상호 작용하는 클라이언트 디바이스 배포

Docker 컨테이너에서 실행 중일 때 AWS IoT Greengrass와 상호 작용하는 클라이언트 디바이스를 배포할 수도 있습니다.

- the section called “모듈 4: AWS IoT Greengrass 그룹에서 클라이언트 디바이스와 상호 작용”의 단계에 따라 코어에 연결하고 MQTT 메시지를 전송하는 클라이언트 디바이스를 배포합니다.

AWS IoT Greengrass 도커 컨테이너 중단

AWS IoT Greengrass 도커 컨테이너를 중지하려면 터미널 또는 명령 프롬프트에서 Ctrl+C를 누릅니다. 이 작업은 `SIGTERM`을 Greengrass 대몬 프로세스로 전송하여 Greengrass 대몬 프로세스와 해당 대몬 프로세스로 시작된 모든 Lambda 프로세스를 제거합니다. Docker 컨테이너는 `/dev/init` 프로세스를 통해 PID 1로 초기화됩니다. 이렇게 하면 남아 있는 좀비 프로세스를 제거하는 데 도움이 됩니다. 자세한 내용은 Docker 실행 참조를 확인하십시오.

도커 컨테이너에서 AWS IoT Greengrass 문제 해결

다음 정보를 사용하면 도커 컨테이너에서 AWS IoT Greengrass 실행 중 발생하는 문제를 해결하는 데 도움이 됩니다.

오류: TTY가 아닌 디바이스에서 대화형 로그인을 수행할 수 없습니다.

해결책: `aws ecr get-login-password` 명령을 실행할 때 이 오류가 발생할 수 있습니다. 가장 최신의 AWS CLI 버전 2 또는 1 이상이 설치되어 있는지 확인합니다. AWS CLI 버전 2를 사용하는 것이 좋습니다. 자세한 내용은 AWS Command Line Interface 사용 설명서에서 AWS CLI 설치를 참조하세요.

오류: 알 수 없는 옵션: no-include-email

해결책: `aws ecr get-login` 명령을 실행할 때 이 오류가 발생할 수 있습니다. 최신 AWS CLI 버전이 설치되어 있는지 확인합니다(예를 들어, `pip install awscli --upgrade --user` 실행). Windows를 사용하고 있고 MSI 설치 관리자를 사용하여 CLI를 설치한 경우 설치 프로세스를 반복해야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 Microsoft Windows에 AWS Command Line Interface 설치를 참조하십시오.

경고: IPv4가 비활성화되어 있습니다. 네트워킹이 작동하지 않습니다.

해결책: Linux 컴퓨터에서 AWS IoT Greengrass를 실행할 때 이 경고 또는 비슷한 메시지를 받을 수 있습니다. 이 단계의 설명에 따라 IPv4 네트워크 전달을 활성화합니다. IPv4 전달을 활성화하지 않으면 AWS IoT Greengrass 클라우드 배포 및 MQTT 통신이 작동하지 않습니다. 자세한 내용은 Docker 설명서의 런타임에 네임스페이스 커널 파라미터(sysctls) 구성을 참조하십시오.

오류: 방화벽이 Windows와 컨테이너 간의 파일 공유를 차단하고 있습니다.

해결책: Windows 컴퓨터에서 Docker를 실행할 때 이 오류 또는 `Firewall Detected` 메시지를 받을 수 있습니다. 이 오류는 VPN(가상 프라이빗 네트워크)에 로그인되어 있고 네트워크 설정 때문에 공유

드라이브가 탑재되지 않는 경우에도 발생할 수 있습니다. 이러한 경우에는 VPN을 끄고 Docker 컨테이너를 재실행합니다.

<account-id><user-name>오류: GetAuthorizationToken 작업을 호출하는 동안 오류가 발생했습니다(AccessDenieException): 사용자: arn:aws:iam::<account-id>:user/<user-name>에게 수행할 권한이 없습니다: ecr:GetAuthorizationToken on resource: *

Amazon ECR 리포지토리에 액세스할 충분한 권한이 없는데 `aws ecr get-login-password` 명령을 실행할 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 Amazon ECR 사용 설명서의 Amazon ECR 리포지토리 정책 예제 및 하나의 Amazon ECR 리포지토리에 액세스를 참조하십시오.

일반적인 AWS IoT Greengrass 문제 해결 도움말은 문제 해결 단원을 참조하십시오.

도커 컨테이너에서 AWS IoT Greengrass 디버깅

도커 컨테이너로 문제를 디버그하려면 Greengrass 런타임 로그를 유지하거나 대화형 셸을 도커 컨테이너에 연결할 수 있습니다.

도커 컨테이너 외부의 Greengrass 런타임 로그를 유지하려면

`/greengrass/ggc/var/log` 디렉터리를 바인드 탑재한 후 AWS IoT Greengrass 도커 컨테이너를 실행할 수 있습니다. 컨테이너가 종료되거나 제거된 후에도 로그는 유지됩니다.

Linux 또는 macOS에서는

1. 호스트에서 실행 중인 Greengrass Docker 컨테이너를 중지하고 터미널에서 다음 명령을 실행합니다. 이렇게 하면 Greengrass `log` 디렉터리가 바인드 탑재되고 Docker 이미지가 시작됩니다.

   인증서와 구성 파일의 압축을 푼 경로로 `/tmp`를 바꿉니다.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
         --entrypoint /greengrass-entrypoint.sh \
         -v /tmp/certs:/greengrass/certs \
         -v /tmp/config:/greengrass/config \
         -v /tmp/log:/greengrass/ggc/var/log \
         -p 8883:8883 \
         216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   그런 다음 Docker 컨테이너 내부에서 Greengrass가 실행되는 중에 호스트의 `/tmp/log`에서 로그를 점검하여 무슨 일이 발생했는지 확인할 수 있습니다.

Windows

호스트에서 실행 중인 Greengrass Docker 컨테이너를 중지하고 명령 프롬프트에서 다음 명령을 실행합니다. 이렇게 하면 Greengrass `log` 디렉터리가 바인드 탑재되고 Docker 이미지가 시작됩니다.

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-
west-2.amazonaws.com/aws-iot-greengrass:latest
```

그런 다음 Docker 컨테이너 내부에서 Greengrass가 실행되는 중에 호스트의 `C:/Users/%USERNAME%/Downloads/log`에서 로그를 점검하여 무슨 일이 발생했는지 확인할 수 있습니다.

대화형 셸을 도커 컨테이너에 연결하려면

대화형 셸을 실행 중인 AWS IoT Greengrass 도커 컨테이너에 연결할 수 있습니다. 이는 Greengrass 도커 컨테이너의 상태를 조사하는 데 도움이 될 수 있습니다.

Linux 또는 macOS에서는

Greengrass Docker 컨테이너가 실행되는 동안 별도의 터미널에서 다음 명령을 실행합니다.

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

Windows

Greengrass Docker 컨테이너가 실행되는 동안 별도의 명령 프롬프트에서 다음 명령을 실행합니다.

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

*gg-container-id*를 이전 명령의 `container_id` 결과로 바꿉니다.

```
docker exec -it gg-container-id /bin/bash
```

Lambda 함수와 커넥터를 사용하여 로컬 리소스에 액세스

이 기능은 AWS IoT Greengrass 코어 v1.3 이상에 사용할 수 있습니다.

AWS IoT Greengrass를 사용하는 개발자는 클라우드에서 AWS Lambda 함수를 작성하고 커넥터를 구성하여 로컬에서 실행하기 위해 코어 디바이스에 배포할 수 있습니다. Linux를 실행하는 Greengrass 코어에서 이러한 로컬로 배포된 Lambda 함수와 커넥터는 Greengrass 코어 디바이스에 실제로 있는 로컬 리소스에 액세스할 수 있습니다. 예를 들어 Modbus 또는 CANbus를 통해 연결된 디바이스와 통신하기 위해 Lambda 함수가 코어 디바이스의 직렬 포트에 액세스하도록 할 수 있습니다. 로컬 리소스에 대한 보안 액세스를 구성하려면 물리적 하드웨어의 보안과 Greengrass 코더 디바이스 OS를 보증해야 합니다.

로컬 리소스에 대한 액세스를 시작하려면 다음 자습서를 참조하십시오.

- AWS 명령줄 인터페이스를 사용하여 로컬 리소스 액세스를 구성하는 방법
- AWS Management Console을 사용하여 로컬 리소스 액세스를 구성하는 방법

지원되는 리소스 유형

볼륨 리소스와 디바이스 리소스라는 두 가지 로컬 리소스 유형에 액세스할 수 있습니다.

볼륨 리소스

루트 파일 시스템의 파일 또는 디렉터리입니다(`/sys`, `/dev` 또는 `/var`에 있는 파일 또는 디렉터리 제외). 다음이 포함됩니다.

- Greengrass Lambda 함수에서 정보를 읽거나 쓰는 데 사용되는 폴더 또는 파일(예: `/usr/lib/python2.x/site-packages/local`).
- 호스트의 /proc 파일 시스템에 있는 폴더 또는 파일(예: `/proc/net` 또는 `/proc/stat`). v1.6 이상에서 지원됩니다. 추가적인 필수 사항은 the section called “/proc 디렉터리에 있는 볼륨 리소스” 섹션을 참조하세요.

> **Tip**
>
> `/var`, `/var/run` 및 `/var/lib` 디렉터리를 볼륨 리소스로 구성하려면 먼저 디렉터리를 다른 폴더에 탑재한 다음 해당 폴더를 볼륨 리소스로 구성하십시오.

볼륨 리소스를 구성할 때 소스 경로와 대상 경로를 지정합니다. 소스 경로는 호스트에 있는 리소스의 절대 경로입니다. 대상 경로는 Lambda 네임스페이스 환경 내부에 있는 리소스의 절대 경로입니다. 이는 Lambda 함수 또는 커넥터가 실행되는 컨테이너입니다. 대상 경로의 모든 변경 사항은 호스트 파일 시스템의 소스 경로에 반영됩니다.

> **Note**
>
> 대상 경로의 파일은 Lambda 네임스페이스에만 표시됩니다. 일반적인 Linux 네임스페이스에서는 이를 볼 수 없습니다.

디바이스 리소스

/dev에 있는 파일. /dev에 있는 문자 디바이스 또는 블록 디바이스만 디바이스 리소스로 허용됩니다. 다음이 포함됩니다.

- 직렬 포트를 통해 연결된 디바이스와 통신하는 데 사용되는 직렬 포트(예: `/dev/ttyS0`, `/dev/ttyS1`).
- USB 주변 장치를 연결하는 데 사용되는 USB(예: `/dev/ttyUSB0` 또는 `/dev/bus/usb`).
- GPIO를 통해 센서 및 액추에이터에 사용되는 GPIO(예: `/dev/gpiomem`).
- 온보드 GPU를 사용하여 기계 학습을 가속화하는 데 사용되는 GPU(예: `/dev/nvidia0`).
- 이미지 및 동영상을 캡처하는 데 사용되는 카메라(예: `/dev/video0`).

> **Note**
>
> `/dev/shm`은 예외로, 볼륨 리소스로만 구성할 수 있습니다. `/dev/shm`에 있는 리소스에는 `rw` 권한을 부여해야 합니다.

AWS IoT Greengrass은(는) 또한 기계 학습 추론을 수행하는 데 사용되는 리소스 유형을 지원합니다. 자세한 내용은 기계 학습 추론 수행 섹션을 참조하세요.

요구 사항

로컬 리소스에 대한 보안 액세스 구성에는 다음 요구 사항이 적용됩니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.3 이상을 사용해야 합니다. 호스트의 /proc 디렉터리에서 리소스를 생성하려면 반드시 v1.6 이상을 사용하고 있어야 합니다.

- 로컬 리소스(필요한 드라이버 및 라이브러리 포함)는 Greengrass 코어 디바이스에 올바르게 설치되어 있어야 하며, 사용 중에 지속적으로 사용할 수 있어야 합니다.
- 원하는 리소스 작업과 리소스에 대한 액세스에는 루트 권한이 필요하지 않습니다.
- `read` 또는 `read and write` 권한만 사용할 수 있습니다. Lambda 함수는 리소스에 대해 특권 작업을 수행할 수 없습니다.
- Greengrass 코어 디바이스의 운영 체제에서 로컬 리소스의 전체 경로를 제공해야 합니다.
- 리소스 이름이나 ID는 최대 128자이며, 패턴 `[a-zA-Z0-9:_-]+`을 사용해야 합니다.

/proc 디렉터리에 있는 볼륨 리소스

호스트의 /proc 디렉터리에 있는 볼륨 리소스에 다음 고려 사항이 적용됩니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.6 이상을 사용해야 합니다.
- 사용자는 Lambda 함수에 대해 읽기 전용 액세스를 허용할 수 있지만, 읽기-쓰기 액세스는 허용할 수 없습니다. 이러한 액세스 수준은 AWS IoT Greengrass에서 관리됩니다.
- 파일 시스템에서 읽기 액세스를 사용할 수 있도록 OS 그룹에 권한을 부여해야 할 수도 있습니다. 예를 들어 소스 디렉터리나 파일이 660 파일 권한을 가지고 있어서 그룹의 소유자나 사용자만 읽기 및 쓰기 액세스 권한을 갖는다고 가정해 봅시다. 이 경우에는 OS 그룹 소유자의 권한을 리소스에 추가해야 합니다. 자세한 내용은 the section called “그룹 소유자 파일 액세스 권한” 섹션을 참조하세요.
- 호스트 환경과 Lambda 네임스페이스 모두 /proc 디렉터리를 포함하고 있기 때문에 대상 경로를 지정할 때 이름 충돌이 발생하지 않도록 해야 합니다. 예를 들어 /proc이 소스 경로인 경우에는 /host-proc을 대상 경로(또는 "/proc" 이외의 모든 경로 이름)로 지정할 수 있습니다.

그룹 소유자 파일 액세스 권한

AWS IoT Greengrass Lambda 함수 프로세스는 일반적으로 `ggc_user` 및 `ggc_group`으로 실행됩니다. 하지만 사용자는 다음과 같이 로컬 리소스 정의에서 추가적인 파일 액세스 권한을 Lambda 함수 프로세스에 제공할 수 있습니다.

- 리소스를 소유한 Linux 그룹의 권한을 추가하려면 `GroupOwnerSetting#AutoAddGroupOwner` 파라미터 또는 리소스를 소유한 시스템 그룹의 파일 시스템 권한 자동 추가 콘솔 옵션을 사용합니다.
- 다른 Linux 그룹의 권한을 추가하려면 `GroupOwnerSetting#GroupOwner` 파라미터 또는 다른 OS 그룹을 지정해 권한 추가 콘솔 옵션을 사용합니다. `GroupOwnerSetting#AutoAddGroupOwner`가 true이면 `GroupOwner` 값이 무시됩니다.

AWS IoT Greengrass Lambda 함수 프로세스는 `ggc_user`, `ggc_group` 및 Linux 그룹(추가한 경우)의 모든 파일 시스템 권한을 상속합니다. Lambda 함수가 리소스에 액세스할 수 있으려면 Lambda 함수 프로세스가 리소스에 대해 필요한 권한을 가지고 있어야 합니다. `chmod(1)` 명령을 사용해 필요한 경우 리소스의 권한을 변경할 수 있습니다.

다음 사항도 참조하세요.

- Amazon Web Services 일반 참조의 리소스에 대한 서비스 할당량

AWS 명령줄 인터페이스를 사용하여 로컬 리소스 액세스를 구성하는 방법

이 기능은 AWS IoT Greengrass 코어 v1.3 이상에 사용할 수 있습니다.

로컬 리소스를 사용하려면 Greengrass 코어 디바이스에 배포되는 그룹 정의에 리소스 정의를 추가해야 합니다. 그룹 정의에는 Lambda 함수에 로컬 리소스에 대한 액세스 권한을 부여하는 Lambda 함수 정의도 포함되어야 합니다. 요구 사항과 제한 조건을 비롯한 자세한 내용은 Lambda 함수와 커넥터를 사용하여 로컬 리소스에 액세스 섹션을 참조하십시오.

이 자습서에서는 AWS Command Line Interface(CLI)를 사용하여 로컬 리소스를 만들고 로컬 리소스에 대한 액세스 권한을 구성하는 프로세스에 대해 설명합니다. 자습서의 단계들을 따르려면 시작하기 AWS IoT Greengrass에서 설명한 대로 Greengrass 그룹을 이미 만들었어야 합니다.

AWS Management Console을(를) 사용하는 튜토리얼은 AWS Management Console을 사용하여 로컬 리소스 액세스를 구성하는 방법 섹션을 참조하십시오.

로컬 리소스 생성

먼저, `CreateResourceDefinition` 명령을 사용하여 액세스할 리소스를 지정하는 리소스 정의를 만듭니다. 이 예에서는 `TestDirectory` 및 `TestCamera`라는 두 개의 리소스를 만듭니다.

```
aws greengrass create-resource-definition  --cli-input-json '{
    "Name": "MyLocalVolumeResource",
    "InitialVersion": {
        "Resources": [
            {
                "Id": "data-volume",
                "Name": "TestDirectory",
```

```
                "ResourceDataContainer": {
                    "LocalVolumeResourceData": {
                        "SourcePath": "/src/LRAtest",
                        "DestinationPath": "/dest/LRAtest",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            },
            {
                "Id": "data-device",
                "Name": "TestCamera",
                "ResourceDataContainer": {
                    "LocalDeviceResourceData": {
                        "SourcePath": "/dev/video0",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            }
        ]
    }
}'
```

Resources: Greengrass 그룹의 `Resource` 객체 목록입니다. 하나의 Greengrass 그룹에는 최대 50개의 리소스가 있을 수 있습니다.

Resource#Id: 리소스의 고유한 식별자입니다. ID는 Lambda 함수 구성에서 리소스를 참조하는 데 사용됩니다. 최대 길이는 128자입니다. 패턴: [a-zA-Z0-9:_-]+.

Resource#Name: 리소스의 이름입니다. 리소스 이름은 Greengrass 콘솔에 표시됩니다. 최대 길이는 128자입니다. 패턴: [a-zA-Z0-9:_-]+.

LocalDeviceResourceData# SourcePath: 기기 리소스의 로컬 절대 경로입니다. 디바이스 리소스의 소스 경로는 `/dev`에서 문자 디바이스 또는 블록 디바이스만 참조할 수 있습니다.

LocalVolumeResourceData# SourcePath: Greengrass 코어 디바이스에 있는 볼륨 리소스의 로컬 절대 경로입니다. 이 위치는 안에서 함수가 실행되는 컨테이너의 외부입니다. 볼륨 리소스 유형의 소스 경로는 `/sys`로 시작할 수 없습니다.

LocalVolumeResourceData# DestinationPath: Lambda 환경 내 볼륨 리소스의 절대 경로입니다. 이 위치는 안에서 함수가 실행되는 컨테이너의 내부입니다.

GroupOwnerSetting: Lambda 프로세스에 대한 추가 그룹 권한을 구성할 수 있습니다. 이 필드는 선택 사항입니다. 자세한 설명은 그룹 소유자 파일 액세스 권한 섹션을 참조하세요.

GroupOwnerSetting# AutoAddGroupOwner: true인 경우 Greengrass는 리소스의 지정된 Linux OS 그룹 소유자를 Lambda 프로세스 권한에 자동으로 추가합니다. 따라서 Lambda 프로세스는 추가된 Linux 그룹의 파일 액세스 권한을 갖게 됩니다.

GroupOwnerSetting# GroupOwner: Lambda 프로세스에 권한이 추가된 Linux OS 그룹의 이름을 지정합니다. 이 필드는 선택 사항입니다.

리소스 정의 버전 ARN은 `CreateResourceDefinition`에서 반환됩니다. ARN은 그룹 정의를 업데이트할 때 사용해야 합니다.

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
    "Name": "MyLocalVolumeResource",
    "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
    "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
    "CreationTimestamp": "2017-11-15T01:18:42.153Z",
    "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

Greengrass 함수 생성

리소스를 만든 후에는 `CreateFunctionDefinition` 명령을 사용하여 Greengrass 함수를 만들고 리소스에 함수 액세스 권한을 부여하십시오.

```
aws greengrass create-function-definition --cli-input-json '{
    "Name": "MyFunctionDefinition",
    "InitialVersion": {
        "Functions": [
            {
                "Id": "greengrassLraTest",
                "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
```

```
                "FunctionConfiguration": {
                    "Pinned": false,
                    "MemorySize": 16384,
                    "Timeout": 30,
                    "Environment": {
                        "ResourceAccessPolicies": [
                            {
                                "ResourceId": "data-volume",
                                "Permission": "rw"
                            },
                            {
                                "ResourceId": "data-device",
                                "Permission": "ro"
                            }
                        ],
                        "AccessSysfs": true
                    }
                }
            }
        ]
    }
}'
```

ResourceAccessPolicies: Lambda 함수에 리소스에 대한 액세스 권한을 `permission` 부여하는 `resourceId` 및 를 포함합니다. Lambda 함수는 최대 20개의 리소스에 액세스할 수 있습니다.

ResourceAccessPolicy#Permission: Lambda 함수가 리소스에 대해 갖는 권한을 지정합니다. 사용 가능한 옵션은 `rw`(읽기/쓰기) 또는 `ro`(읽기 전용)입니다.

AccessSysfs: true인 경우 Lambda 프로세스는 Greengrass 코어 `/sys` 디바이스의 폴더에 대한 읽기 액세스 권한을 가질 수 있습니다. 이는 Greengrass Lambda 함수가 `/sys`에서 디바이스 정보를 읽어야 하는 경우에 사용됩니다.

다시 말하지만, `CreateFunctionDefinition`은 함수 정의 버전 ARN을 반환합니다. ARN은 그룹 정의 버전에서 사용해야 합니다.

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
    "Name": "MyFunctionDefinition",
    "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
```

```
    "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
    "CreationTimestamp": "2017-11-22T02:28:02.325Z",
    "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}
```

그룹에 Lambda 함수를 추가합니다.

마지막으로 CreateGroupVersion를 사용하여 그룹에 함수를 추가하십시오. 예:

```
aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-
caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbcb6/
versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

> **Note**
>
> 이 명령과 함께 사용할 그룹 ID를 가져오는 방법을 알아보려면 the section called “그룹 ID 가져오기” 단원을 참조하십시오.

새 그룹 버전이 반환됩니다.

```
{
    "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/
b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
    "Version": "291917fb-ec54-4895-823e-27b52da25481",
    "CreationTimestamp": "2017-11-22T01:47:22.487Z",
    "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

이제 Greengrass 그룹에 두 리소스인 및 에 액세스할 수 있는 LraTest Lambda 함수가 포함됩니다. TestDirectory TestCamera

다음은 Python으로 작성된 Lambda 함수 `lraTest.py`의 예제입니다. 이 함수는 로컬 볼륨 리소스에 씁니다.

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

이 명령은 Greengrass API에서 제공하며 리소스 정의 및 리소스 정의 버전을 작성하고 관리합니다.

- CreateResourceDefinition

- CreateResourceDefinitionVersion
- DeleteResourceDefinition
- GetResourceDefinition
- GetResourceDefinitionVersion
- ListResourceDefinitions
- ListResourceDefinitionVersions
- UpdateResourceDefinition

문제 해결

- Q: Greengrass 그룹 배포가 다음과 같은 오류와 함께 실패하는 이유는 무엇입니까?

```
group config is invalid:
    ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
```

  A: 이 오류는 Lambda 프로세스에 지정된 리소스에 대한 권한이 없음을 나타냅니다. 해결 방법은 Lambda가 리소스에 액세스할 수 있도록 리소스의 파일 권한을 변경하는 것입니다. (자세한 내용은 그룹 소유자 파일 액세스 권한 단원을 참조하십시오).

- Q: `/var/run`을 볼륨 리소스로 구성할 때 Lambda 함수가 runtime.log의 오류 메시지와 함께 시작하지 못하는 이유는 무엇입니까?

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
 container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \\\"/var/run\\\" to rootfs \\\"/
greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
rootfs_sys/run\\\"
caused \\\"invalid argument\\\"\""
```

  A: AWS IoT Greengrass 코어는 현재 `/var`, `/var/run` 및 `/var/lib`의 구성을 볼륨 리소스로 지원하지 않습니다. 한 가지 해결 방법은 먼저 `/var`, `/var/run` 또는 `/var/lib`를 다른 폴더에 탑재한 다음 해당 폴더를 볼륨 리소스로 구성하는 것입니다.

- Q: `/dev/shm`을 읽기 전용 권한을 가진 볼륨 리소스로 구성할 때 runtime.log에 오류가 발생하면서 Lambda 함수가 시작하지 못하는 이유는 무엇입니까?

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
 container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \\\"/dev/shm\\\" to rootfs \\\"/
greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
rootfs_sys/dev/shm\\\"
caused \\\"operation not permitted\\\"\""”
```

A: `/dev/shm`은 읽기/쓰기로만 구성할 수 있습니다. 리소스 권한을 `rw`로 변경하면 문제가 해결됩니다.

AWS Management Console을 사용하여 로컬 리소스 액세스를 구성하는 방법

이 기능은 AWS IoT Greengrass 코어 v1.3 이상에 사용할 수 있습니다.

호스트 Greengrass 코어 디바이스에서 로컬 리소스에 안전하게 액세스하도록 Lambda 함수를 구성할 수 있습니다. 로컬 리소스란 물리적으로 호스트에 있는 버스와 주변 정치 또는 호스트 OS의 파일 시스템 볼륨을 가리킵니다. 요구 사항과 제한 조건을 비롯한 자세한 내용은 Lambda 함수와 커넥터를 사용하여 로컬 리소스에 액세스 섹션을 참조하십시오.

이 튜토리얼에서는 AWS Management Console을 사용하여 AWS IoT Greengrass 코어 디바이스에 존재하는 로컬 리소스에 대한 액세스를 구성하는 방법을 설명합니다. 자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Lambda 함수 배포 패키지 생성
2. Lambda 함수 생성 및 게시
3. 그룹에 Lambda 함수를 추가합니다.
4. 그룹에 로컬 리소스 추가
5. 그룹에 구독 추가
6. 그룹 배포

AWS Command Line Interface을(를) 사용하는 튜토리얼은 AWS 명령줄 인터페이스를 사용하여 로컬 리소스 액세스를 구성하는 방법 섹션을 참조하십시오.

필수 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- Greengrass 그룹 및 Greengrass 코어(v1.3 이상). Greengrass 그룹 또는 코어를 생성하는 방법에 대해 알아보려면 시작하기 AWS IoT Greengrass 섹션을 참조하십시오.
- Greengrass 코어 디바이스의 다음 디렉터리:
  - /src/LRAtest
  - /dest/LRAtest

  이러한 디렉터리의 소유자 그룹은 디렉터리에 대한 읽기 및 쓰기 권한을 보유해야 합니다. 다음 명령을 사용하여 액세스를 부여할 수 있습니다.

```
sudo chmod 0775 /src/LRAtest
```

1단계: Lambda 함수 배포 패키지 생성

이 단계에서는 함수의 코드와 종속성이 포함된 ZIP 파일인 Lambda 함수 배포 패키지를 생성합니다. 또한 종속성으로 패키지에 포함할 AWS IoT Greengrass 코어 SDK도 다운로드합니다.

1. 컴퓨터에서 다음 Python 스크립트를 `lraTest.py`라는 로컬 파일에 복사합니다. 이는 Lambda 함수에 대한 앱 로직입니다.

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
```

```
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass
 Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

2. AWS IoT Greengrass 코어 SDK 다운로드 페이지에서 Python용 AWS IoT Greengrass 코어 SDK를 다운로드합니다.
3. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
4. 다음 항목을 `lraTestLambda.zip`라는 파일로 압축합니다.

   - `lraTest.py`. 앱 로직.
   - `greengrasssdk`. 모든 Python Lambda 함수에 대한 필수 라이브러리입니다.

   `lraTestLambda.zip` 파일이 Lambda 함수 배포 패키지입니다. 이제 Lambda 함수를 만들고 배포 패키지를 업로드할 준비가 되었습니다.

2단계: Lambda 함수 생성 및 게시

이 단계에서는 AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성한 후 배포 패키지를 사용하도록 구성합니다. 그런 다음 함수 버전을 게시하고 별칭을 생성합니다.

먼저, Lambda 함수를 생성합니다.

1. AWS Management Console에서 [Services]를 선택한 다음 AWS Lambda 콘솔을 엽니다.
2. 함수를 선택합니다.
3. 함수 생성을 선택한 다음 새로 작성을 선택합니다.

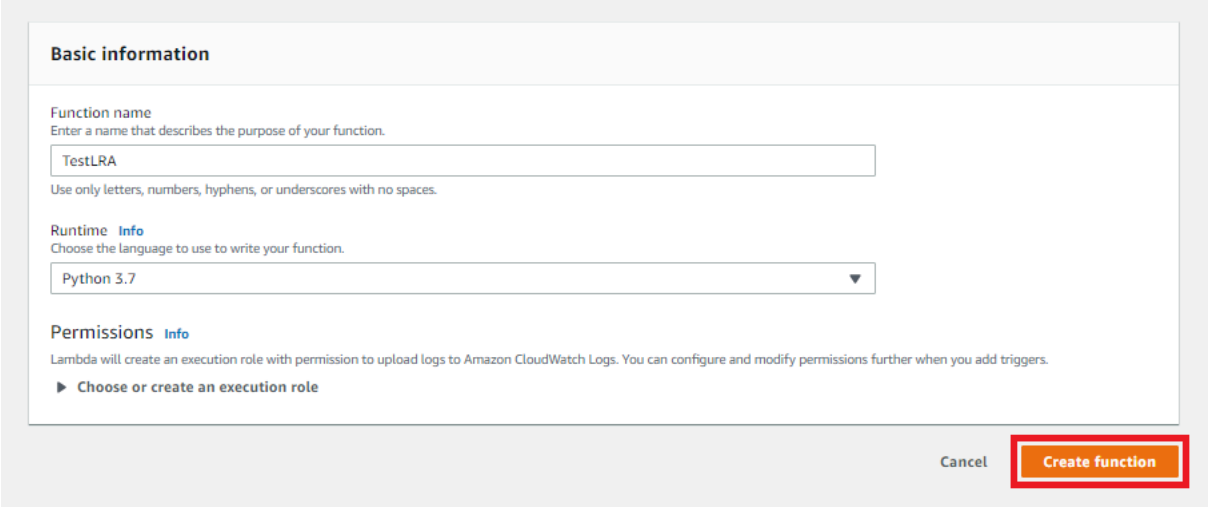
4. 기본 정보 섹션에서 다음 값을 지정합니다.

   a. [함수 이름(Function name)]에 **`TestLRA`**을 입력합니다.

   b. 실행 시간에서 Python 3.7을 선택합니다.

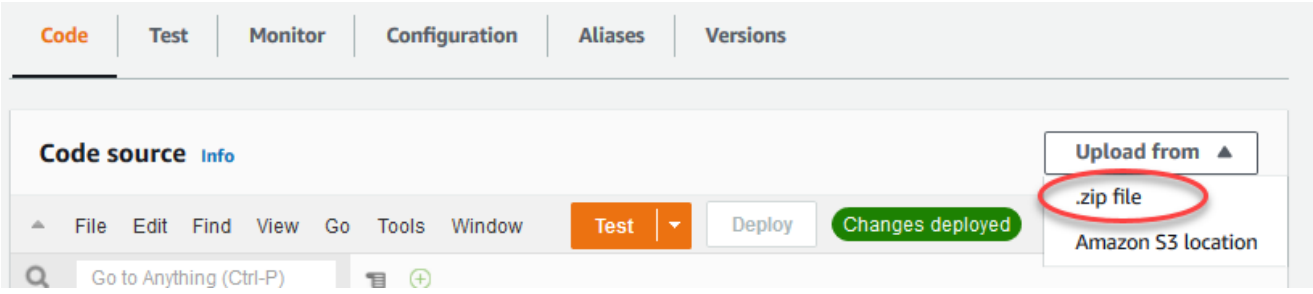
   c. 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.

5. 함수 생성(Create function)을 선택합니다.



6. 이제 Lambda 함수 배포 패키지를 업로드하고 핸들러를 등록합니다.

   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



   b. 업로드를 선택한 다음 `lraTestLambda.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.

   c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

   - 실행 시간에서 Python 3.7을 선택합니다.
   - 핸들러에 대해 lraTest.function_handler를 입력합니다.

d. Save를 선택합니다.

> **Note**
>
> AWS Lambda 콘솔의 테스트 버튼은 이 함수와 함께 작동하지 않습니다. AWS IoT Greengrass 코어 SDK에는 AWS Lambda 콘솔에서 Greengrass Lambda 함수를 독립적으로 실행하는 데 필요한 모듈이 포함되어 있지 않습니다. 이러한 모듈(예: `greengrass_common`)은 Greengrass 코어에 배포된 후 함수에 제공됩니다.

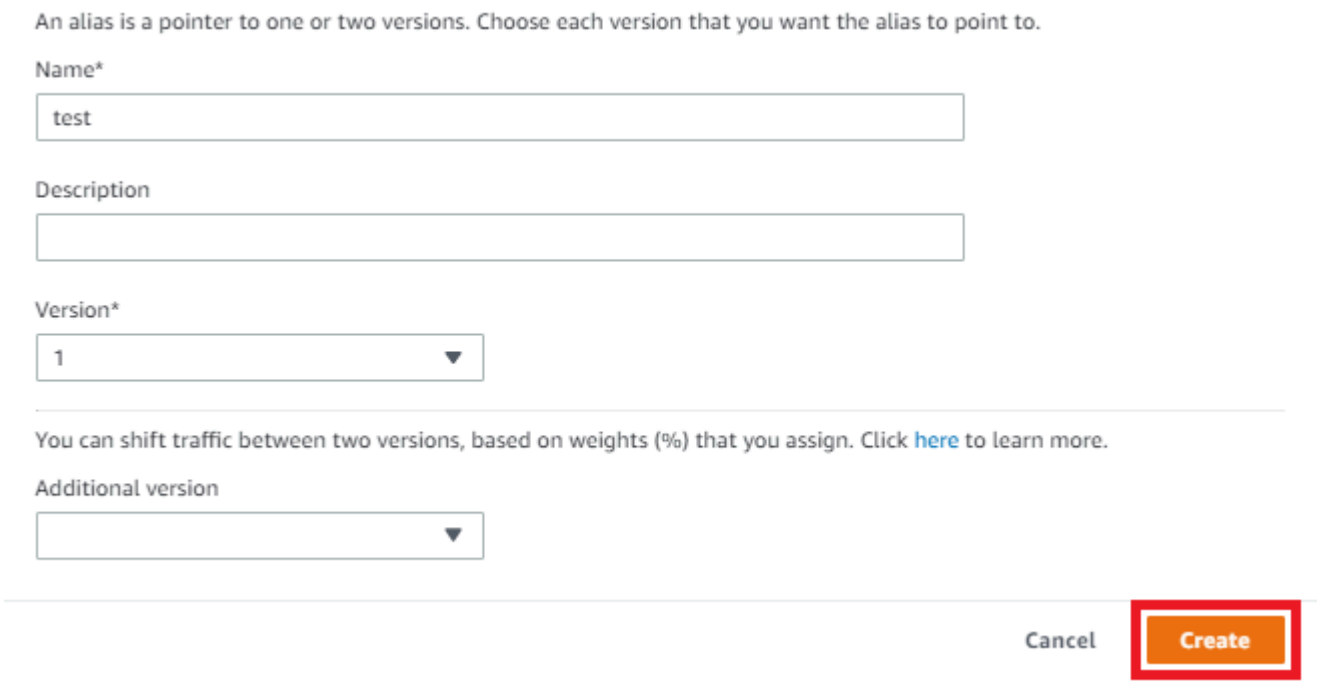
그런 다음 Lambda 함수의 첫 번째 버전을 게시합니다. 그런 다음 버전의 별칭을 생성합니다.

Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

7. 작업에서 새 버전 게시를 선택합니다.
8. 버전 설명에 **`First version`**을 입력한 후 게시를 선택합니다.
9. [TestLRA: 1] 구성 페이지의 [Actions]에서 [Create alias]를 선택합니다.
10. 별칭 작성 페이지에서 이름에 **`test`**를 입력하십시오. 버전에서 1을 선택합니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

11. 생성을 선택합니다.



이제 Greengrass 그룹에 Lambda 함수를 추가할 수 있습니다.

3단계: Greengrass 그룹에 Lambda 함수 추가

이 단계에서 그룹에 함수를 추가한 다음 함수의 수명 주기를 구성합니다.

먼저 Greengrass 그룹에 Lambda 함수를 추가합니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. Lambda 함수를 추가하려는 Greengrass 그룹을 선택합니다.
3. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
4. 내 Lambda 함수 섹션에서 추가를 선택합니다.
5. Lambda 함수 추가 페이지에서 Lambda 함수를 선택합니다. Select **`TestLRA`**.
6. Lambda 함수 버전을 선택합니다.
7. Lambda 함수 구성 섹션에서 시스템 사용자 및 그룹과 Lambda 함수 컨테이너화를 선택합니다.

   그런 다음 Lambda 함수의 수명 주기를 구성합니다.
8. Timeout에서, 30 seconds를 선택합니다.

> **Important**
>
> (이 절차에서 설명된 바와 같이) 로컬 리소스를 사용하는 Lambda 함수는 Greengrass 컨테이너에서 실행해야 합니다. 그렇지 않으면 함수 배포를 시도하는 경우 배포가 실패합니다. 자세한 내용은 컨테이너화를 참조하십시오.

9. 페이지 하단에서 Lambda 함수 추가를 선택합니다.

4단계: Greengrass 그룹에 로컬 리소스 추가

이 단계에서는 로컬 볼륨 리소스를 Greengrass 그룹에 추가하고, 함수에 리소스에 대한 읽기 및 쓰기 액세스를 부여합니다. 로컬 리소스의 범위는 그룹 레벨입니다. 리소스에 액세스하기 위해 그룹에서 모든 Lambda 함수에 대한 권한을 부여할 수 있습니다.

1. 그룹 구성 페이지에서 리소스를 선택합니다.
2. 로컬 리소스 섹션에서 추가를 선택합니다.
3. 로컬 리소스 추가 페이지에서 다음 값을 사용합니다.

   a. 리소스 이름에 **testDirectory**을 입력합니다.

   b. [리소스 유형(Resource type)]에서 [볼륨(Volume)]을 선택합니다.

   c. 디바이스 경로에 **/src/LRAtest**를 입력합니다. 이 경로가 호스트 OS에 존재해야 합니다.

      로컬 디바이스 경로는 코어 디바이스의 파일 시스템에 있는 리소스의 로컬 절대 경로입니다. 이 위치는 안에서 함수가 실행되는 컨테이너의 외부입니다. 경로는 /sys로 시작할 수 없습니다.

   d. [Destination path]에 [**/dest/LRAtest**]을 입력합니다. 이 경로가 호스트 OS에 존재해야 합니다.

      대상 경로는 Lambda 네임스페이스에서 리소스의 절대 경로입니다. 이 위치는 안에서 함수가 실행되는 컨테이너의 내부입니다.

   e. 시스템 그룹 소유자 및 파일 액세스 권한에서 리소스를 소유한 시스템 그룹의 파일 시스템 권한 자동 추가를 선택합니다.

      시스템 그룹 소유자 및 파일 액세스 권한 옵션을 사용하면 Lambda 프로세스에 추가 파일 액세스 권한을 부여할 수 있습니다. 자세한 내용은 그룹 소유자 파일 액세스 권한 섹션을 참조하세요.

4. 리소스 추가를 선택합니다. [Resources] 페이지에 새 testDirectory 리소스가 표시됩니다.

5단계: Greengrass 그룹에 구독 추가

이 단계에서는 Greengrass 그룹에 구독을 추가합니다. 이러한 구독은 Lambda 함수와 AWS IoT 사이의 양방향 통신을 가능하게 합니다.

먼저 AWS IoT에 메시지를 보내도록 Lambda 함수에 대한 구독을 생성합니다.

1. 그룹 구성 페이지에서 구독 탭을 선택합니다.
2. 추가(Add)를 선택합니다.
3. 구독 생성 페이지에서 다음과 같이 원본과 대상을 구성합니다.

   a. 소스 유형에서 Lambda 함수를 선택한 다음 TestLRA를 선택합니다.

   b. 대상 유형에서 서비스를 선택한 다음 IoT 클라우드를 선택합니다.

   c. 주제 필터에 **`LRA/test`**를 입력한 다음 구독 생성을 선택합니다.
4. [Subscriptions] 페이지에 새 구독이 표시됩니다.

   그런 다음 AWS IoT에서 함수를 간접 호출하는 구독을 구성합니다.
5. 구독 페이지에서 구독 추가를 선택합니다.
6. [Select your source and target] 페이지에서 다음과 같이 원본과 대상을 구성합니다.

   a. 소스 유형에서 Lambda 함수를 선택한 다음 IoT 클라우드를 선택합니다.

   b. 대상 유형에서 서비스를 선택한 다음 TestLRA를 선택합니다.

   c. 다음(Next)을 선택합니다.
7. 데이터를 주제에 대해 필터링 페이지의 주제 필터 필드에 **`invoke/LRAFunction`**을 입력한 후 다음을 선택합니다.
8. [마침]을 클릭합니다. [Subscriptions] 페이지에 2개의 구독이 모두 표시됩니다.

6단계: AWS IoT Greengrass 그룹 배포

이 단계에서는 그룹 정의의 현재 버전을 배포합니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 대몬(daemon)이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 `root`에 대한 `/greengrass/ggc/packages/1.11.6/bin/daemon` 입력이 포함되어 있는 경우에는 대몬(daemon)이 실행 중인 것입니다.

   > **Note**
   >
   > 경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

   b. 대몬(daemon)을 시작하려면:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. 그룹 구성 페이지에서 배포를 선택합니다.

   > **Note**
   >
   > 컨테이너화 없이 Lambda 함수를 실행하고 연결된 로컬 리소스에 액세스하려고 하면 배포에 실패합니다.

3. 메시지가 표시되면 Lambda 함수 탭의 시스템 Lambda 함수에서 IP 감지기를 선택한 다음 편집, 자동 감지를 선택합니다.

   이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

> **Note**
>
> 메시지가 표시되면 Greengrass 서비스 역할을 생성할 권한을 부여하고 이 권한을 현재 AWS 리전의 AWS 계정과 연결합니다. 이 역할은 AWS IoT Greengrass가 AWS 서비스의 리소스에 액세스할 수 있습니다.

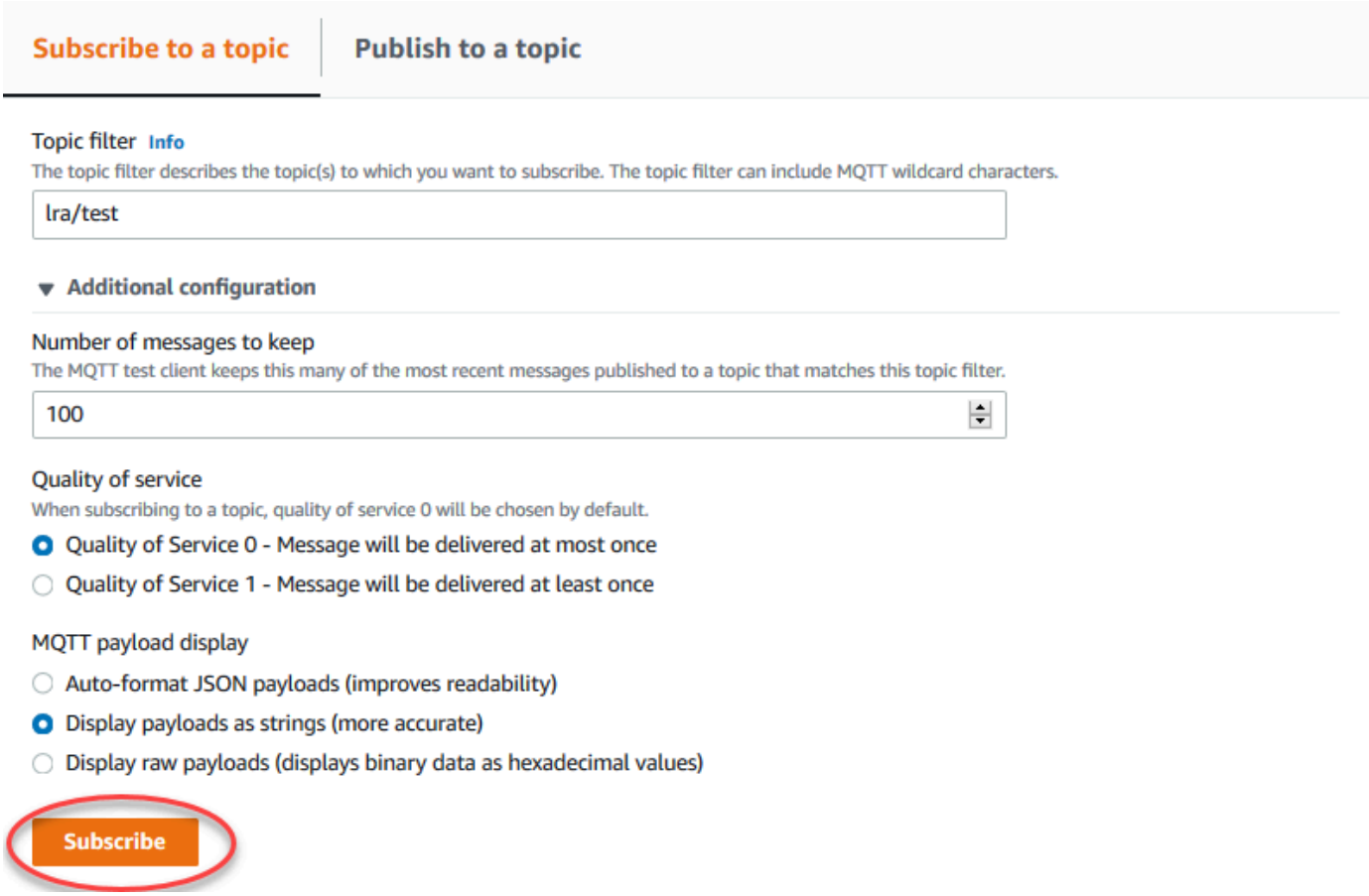
배포 페이지에 배포 타임스탬프, 버전 ID, 상태가 표시됩니다. 완료되면 배포 상태는 완료됨이 됩니다.

문제 해결에 대한 도움말은 문제 해결 섹션을 참조하십시오.

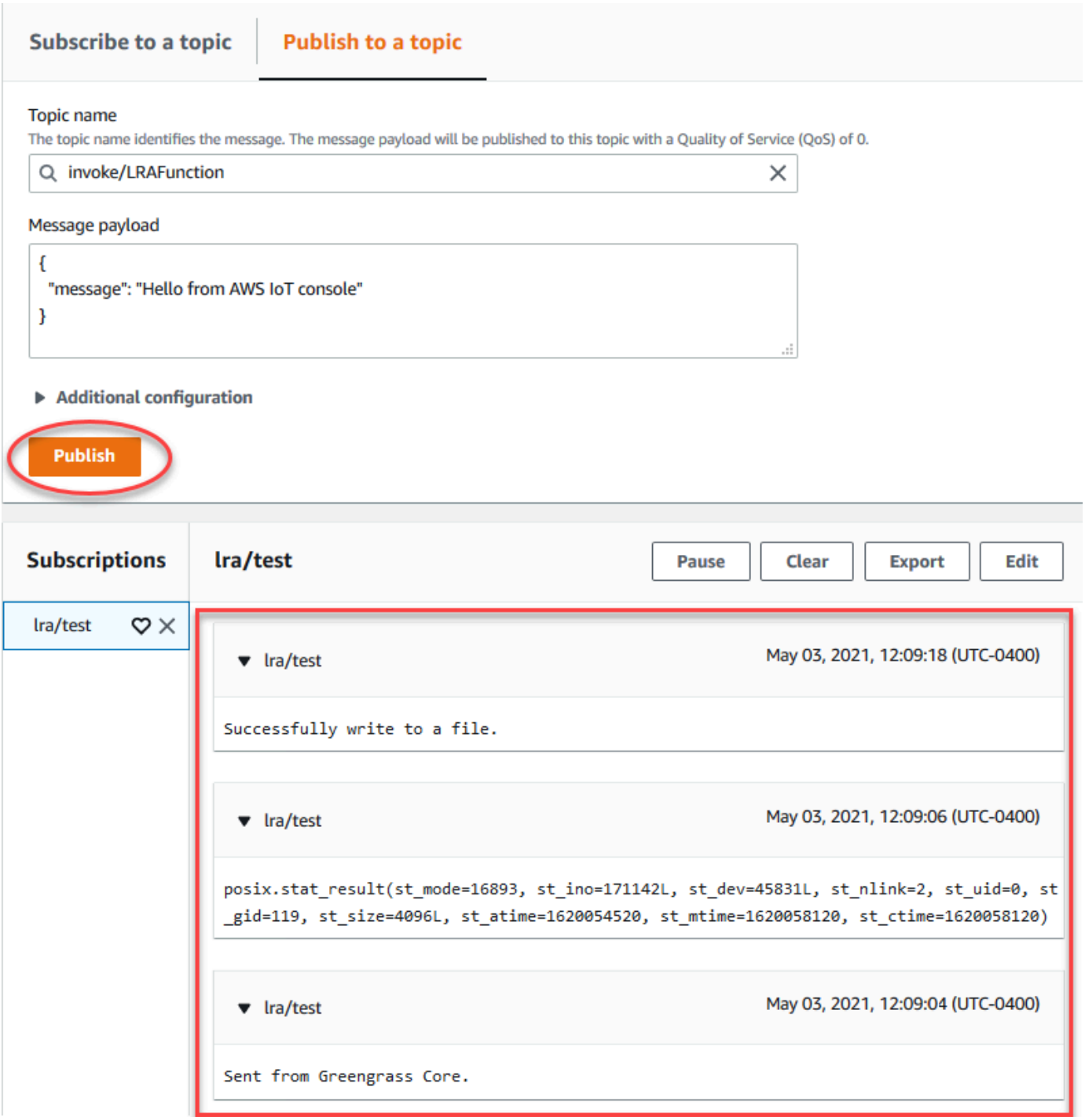
로컬 리소스 액세스 테스트

이제 로컬 리소스 액세스가 올바르게 구성되었는지 확인할 수 있습니다. 테스트하려면 `LRA/test` 주제를 구독하고 `invoke/LRAFunction` 주제에 게시합니다. Lambda 함수가 예상되는 페이로드를 AWS IoT에 전송하면 테스트가 성공한 것입니다.

1. AWS IoT 콘솔을 열고 탐색 창에서 테스트를 선택하고 MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독에서 주제 필터에 대해 **`LRA/test`**를 입력합니다.
3. 추가 정보에서 MQTT 페이로드 표시의 경우 페이로드를 문자열로 표시를 선택합니다.
4. 구독을 선택합니다. Lambda 함수가 LRA/test 주제에 게시합니다.



5. 주제에 게시에서 주제 이름에 **`invoke/LRAFunction`**을 입력한 다음 게시를 선택하여 Lambda 함수를 간접 호출합니다. 페이지에 함수의 3개 메시지 페이로드가 표시되면 테스트가 성공한 것입니다.



Lambda 함수로 만든 테스트 파일은 Greengrass 코어 디바이스의 `/src/LRAtest` 디렉터리에 있습니다. Lambda 함수는 `/dest/LRAtest` 디렉터리의 파일에 쓰지만 해당 파일은 Lambda 네임스페이스에서만 볼 수 있습니다. 일반적인 Linux 네임스페이스에서는 이를 볼 수 없습니다. 대상 경로의 모든 변경 사항은 파일 시스템의 소스 경로에 반영됩니다.

문제 해결에 대한 도움말은 문제 해결 섹션을 참조하십시오.

기계 학습 추론 수행

이 특성은 AWS IoT Greengrass 코어 v1.6 이상에 사용할 수 있습니다.

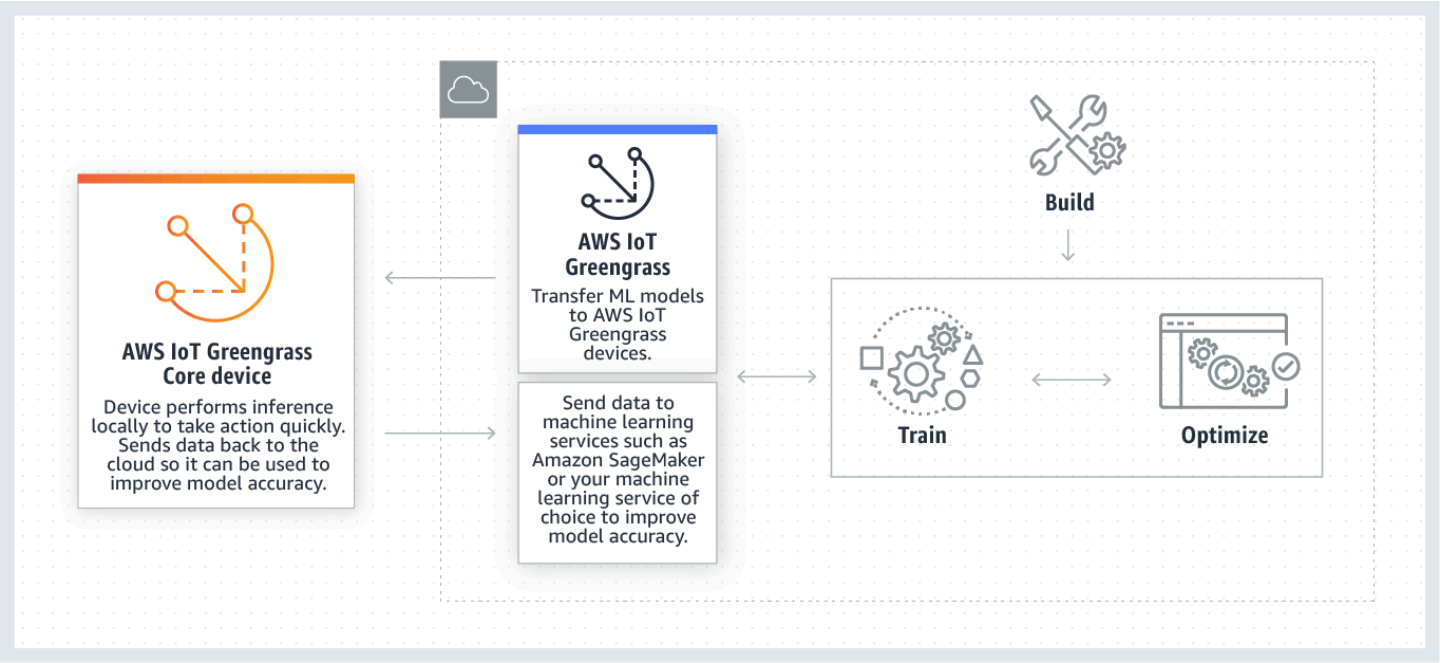
AWS IoT Greengrass을(를) 사용하여 클라우드 교육 모델을 사용하여 로컬 방식으로 생성된 데이터의 엣지에서 기계 학습(ML) 추론을 수행할 수 있습니다. 이를 통해 로컬 추론 실행의 낮은 지연 시간과 비용 절감이라는 이점을 얻을 수 있습니다. 그러면서도 모델 훈련 및 복잡한 처리에 필요한 클라우드 컴퓨팅 능력을 활용할 수 있습니다.

로컬 추론 수행을 시작하려면 the section called “기계 학습 추론을 구성하는 방법” 단원을 참조하십시오.

AWS IoT Greengrass ML 추론 작동 방식

어디서든 추론 모델을 교육하고, 이를 Greengrass 그룹에서 기계 학습 리소스로서 로컬 방식으로 배포한 다음 Greengrass Lambda 함수에서 액세스할 수 있습니다. 예를 들어 SageMaker에서 딥 러닝 모델을 빌드 및 교육한 다음 이를 Greengrass 코어에 배포할 수 있습니다. Lambda 함수는 로컬 모델을 사용하여 연결된 디바이스에서 추론을 수행하고 새 교육 데이터를 클라우드로 전송할 수 있습니다.

다음 다이어그램은 AWS IoT Greengrass ML 추론 워크플로를 보여줍니다.



AWS IoT Greengrass ML 추론은 다음과 같은 ML 워크플로의 각 단계를 간소화합니다.

- ML 프레임워크 프로토타입 빌드 및 배포.
- 클라우드 교육 모델 액세스 및 Greengrass 코어 디바이스에 배포.
- 하드웨어 액셀러레이터(GPU 및 FPGA 등)에 로컬 리소스로서 액세스할 수 있는 추론 앱 생성.

기계 학습 리소스

기계 학습 리소스는 AWS IoT Greengrass 코어에 배포된 클라우드 교육 추론 모델을 나타냅니다. 기계 학습 리소스를 배포하려면 리소스를 Greengrass 그룹에 추가한 다음 그룹의 Lambda 함수가 어떻게 액세스할 수 있는지 정의합니다. 그룹 배포 도중 AWS IoT Greengrass은(는) 클라우드에서 소스 모델 패키지를 검색하고 Lambda 런타임 네임스페이스 내에 있는 디렉터리에 이를 압축 해제합니다. 그런 다음 Greengrass Lambda 함수는 로컬 방식으로 배포된 모델을 사용하여 추론을 수행합니다.

로컬 방식으로 배포된 모델을 업데이트하려면 기계 학습 리소스에 해당하는 (클라우드상의) 소스 모델을 업데이트한 다음 그룹을 배포합니다. 배포 도중 AWS IoT Greengrass은(는) 소스의 변경 사항을 확인합니다. 변경 사항이 감지되는 경우 AWS IoT Greengrass은(는) 로컬 모델을 업데이트합니다.

지원되는 모델 소스

AWS IoT Greengrass은(는) 기계 학습 리소스에 SageMaker 및 Amazon S3 모델을 지원합니다.

다음 요구 사항은 모델 소스에 적용됩니다.

- SageMaker와 Amazon S3 모델 소스를 저장하는 S3 버킷은 SSE-C를 사용하여 암호화해서는 안 됩니다. 서버 측 암호화를 사용하는 버킷의 경우 AWS IoT Greengrass ML 추론은 현재 SSE-S3 또는 SSE-KMS 암호화 옵션만 지원합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 서버 측 암호화 옵션을 사용하여 데이터 보호를 참조하세요.
- SageMaker 및 Amazon S3 모델 소스를 저장하는 S3 버킷의 이름에 마침표(`.`)를 포함해서는 안 됩니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 이름 지정 규칙에서 SSL과 함께 가상 호스팅 스타일 버킷을 사용하는 것과 관련된 규칙을 참조하십시오.
- 서비스 수준 AWS 리전 지원을 AWS IoT Greengrass 및 SageMaker 모두에 제공해야 합니다. 현재 AWS IoT Greengrass는 다음 리전에서 SageMaker 모델을 지원합니다.
  - 미국 동부(오하이오)
  - 미국 동부(버지니아 북부)
  - 미국 서부(오레건)
  - 아시아 태평양(뭄바이)
  - 아시아 태평양(서울)

- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)

- AWS IoT Greengrass은(는) 다음 단원에서 설명한 대로 모델 소스에 대한 `read` 권한을 보유해야 합니다.

SageMaker

AWS IoT Greengrass은(는) SageMaker 교육 작업으로 저장된 모델을 지원합니다. SageMaker는 내장형 또는 사용자 지정 알고리즘을 사용하여 모델을 빌드 및 교육할 수 있는 완전 관리형 ML 서비스입니다. 자세한 내용은 SageMaker 개발자 가이드의 SageMaker란?을 참조하세요.

이름에 `sagemaker`가 들어간 버킷을 생성하여 SageMaker 환경을 구성한 경우 AWS IoT Greengrass는 교육 작업에 액세스하는 적절한 권한을 보유합니다. `AWSGreengrassResourceAccessRolePolicy` 관리형 정책에서는 이름에 문자열 `sagemaker`가 포함된 버킷에 대한 액세스를 허용합니다. 이 정책은 Greengrass 서비스 역할에 연결됩니다.

그렇지 않으면 교육 작업이 저장된 버킷에 대한 AWS IoT Greengrass `read` 권한을 부여합니다. 이를 수행하려면 서비스 역할에 다음 인라인 정책을 포함시킵니다. 여러 버킷 ARN을 나열할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
```

```
}
```

Amazon S3

AWS IoT Greengrass은(는) Amazon S3에 `tar.gz` 또는 `.zip` 파일로 저장된 모델을 지원합니다.

AWS IoT Greengrass이(가) Amazon S3 버킷에 저장된 모델에 액세스하도록 하려면 다음 중 하나를 수행하여 버킷에 액세스할 AWS IoT Greengrass `read` 권한을 부여해야 합니다.

- 이름에 `greengrass`가 포함된 버킷에 모델을 저장합니다.

  `AWSGreengrassResourceAccessRolePolicy` 관리형 정책에서는 이름에 문자열 `greengrass`가 포함된 버킷에 대한 액세스를 허용합니다. 이 정책은 Greengrass 서비스 역할에 연결됩니다.

- Greengrass 서비스 역할에 다음 인라인 정책을 포함시킵니다.

  버킷 이름에 `greengrass`가 포함되지 않는 경우 서비스 역할에 다음 인라인 정책을 추가합니다. 여러 버킷 ARN을 나열할 수 있습니다.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "s3:GetObject"
              ],
              "Resource": [
                  "arn:aws:s3:::my-bucket-name"
              ]
          }
      ]
  }
  ```

  자세한 내용은 IAM 사용 설명서의 인라인 정책 임베딩을 참조하세요.

요구 사항

기계 학습 리소스 생성 및 사용에는 다음 요구 사항이 적용됩니다.

- 반드시 AWS IoT Greengrass Core v1.6 이상을 사용해야 합니다.
- 사용자 정의 Lambda 함수는 리소스에 대한 `read` 또는 `read and write` 작업을 수행할 수 없습니다. 다른 작업을 할 수 있는 권한은 제공되지 않습니다. 연결된 Lambda 함수의 컨테이너화 모드는 액세스 권한을 설정하는 방법을 결정합니다. 자세한 내용은 the section called “기계 학습 리소스에 액세스” 섹션을 참조하세요.
- 코어 디바이스의 운영 체제에서 리소스의 전체 경로를 제공해야 합니다.
- 리소스 이름이나 ID는 최대 128자이며, 패턴 `[a-zA-Z0-9:_-]+`을 사용해야 합니다.

ML 추론을 위한 런타임 및 라이브러리

AWS IoT Greengrass에서 다음 ML 런타임 및 라이브러리를 사용할 수 있습니다.

- Amazon SageMaker Neo 딥 러닝 런타임
- Apache MXNet
- TensorFlow

이러한 런타임 및 라이브러리는 NVIDIA Jetson TX2, Intel Atom 및 Raspberry Pi 플랫폼에 설치될 수 있습니다. 다운로드 정보는 the section called “지원되는 기계 학습 런타임 및 라이브러리” 섹션을 참조하세요. 코어 디바이스에 직접 설치할 수 있습니다.

호환성 및 제한 사항에 대한 다음 정보를 참조해야 합니다.

SageMaker Neo 딥 러닝 런타임

SageMaker Neo 딥러닝 런타임을 사용해 AWS IoT Greengrass 디바이스에서 최적화된 기계 학습 모델로 추론을 수행할 수 있습니다. 이러한 모델은 기계 학습 추론 예측 속도를 개선하기 위해 SageMaker Neo 딥러닝 컴파일러를 사용해 최적화되었습니다. SageMaker의 모델 최적화에 대한 자세한 내용은 SageMaker Neo 설명서를 참조하십시오.

> **Note**
>
> 현재는 특정 Amazon Web Services 지역에서만 Neo 딥러닝 컴파일러를 사용하여 기계 학습 모델을 최적화할 수 있습니다. 하지만 AWS IoT Greengrass 코어가 지원되는 각 AWS 리전에서는 최적화된 모델과 함께 Neo 딥 러닝 런타임을 사용할 수 있습니다. 자세한 내용은 최적화된 Machine Learning 추론을 구성하는 방법을 참조하십시오.

MXNet 버전 관리

Apache MXNet은 현재 다음 버전과의 호환성을 보장하지 않습니다. 따라서 이후 버전의 프레임워크를 사용하여 교육한 모델은 이전 버전의 프레임워크에서 제대로 작동하지 않을 수 있습니다. 모델 교육 및 모델 서비스 단계 사이의 충돌을 피하고, 일관된 종합적 경험을 제공하려면 모든 단계에서 동일한 MXNet 프레임워크 버전을 사용해야 합니다.

Raspberry Pi의 MXNet

로컬 MXNet 모델에 액세스하는 Lambda 함수는 다음과 같은 환경 변수를 설정해야 합니다.

```
MXNET_ENGINE_TYPE=NativeEngine
```

환경 변수를 함수 코드에 설정하거나 함수의 그룹별 구성에 추가할 수 있습니다. 환경 변수를 구성 설정으로 추가하는 예는 이 단계를 참조하십시오.

> **Note**
>
> 타사 코드 예제 실행과 같이 MXNet 프레임워크를 일반적으로 사용하는 경우에는 환경 변수를 Raspberry Pi에서 구성해야 합니다.

Raspberry Pi에서의 TensorFlow 모델 서비스 제한 사항

추론 결과 개선을 위한 다음 권장 사항은 Raspberry Pi 플랫폼에서의 TensorFlow 32비트 ARM 라이브러리를 사용한 테스트를 기반으로 합니다. 이러한 권장 사항은 고급 사용자의 참조용만을 목적으로 하며, 어떤 것도 보장되지 않습니다.

- Checkpoint 형식을 사용하여 교육된 모델은 서비스 전에 프로토콜 버퍼에 대해 고정되어야 합니다. 예를 들어 TensorFlow-Slim 이미지 분류 모델 라이브러리를 참조하십시오.
- 교육 또는 추론 코드에서 TF-Estimator 및 TF-Slim 라이브러리를 사용하지 마십시오. 대신 다음 예제에 표시된 `.pb` 파일 모델 로드 패턴을 사용하십시오.

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
  tf.import_graph_def(graph_def)
```

> **Note**
>
> TensorFlow에 대해 지원되는 플랫폼에 대한 자세한 내용은 TensorFlow 설명서의 Installing TensorFlow를 참조하십시오.

Lambda 함수에서 기계 학습 리소스에 액세스

사용자 정의 Lambda 함수는 기계 학습 리소스에 액세스하여 AWS IoT Greengrass에서 로컬 추론을 실행할 수 있습니다. 기계 학습 리소스는 훈련된 모델과 코어 디바이스로 다운로드되는 기타 아티팩트로 구성됩니다.

Lambda 함수가 코어의 기계 학습 리소스에 액세스할 수 있도록 허용하려면 해당 리소스를 Lambda 함수에 연결하고 액세스 권한을 정의해야 합니다. 제휴(또는 연결된) Lambda 함수의 컨테이너화 모드에 따라 이 작업을 수행하는 방법이 결정됩니다.

기계 학습 리소스에 대한 액세스 권한

AWS IoT Greengrass 코어 v1.10.0부터는 기계 학습 리소스의 리소스 소유자를 정의할 수 있습니다. 리소스 소유자는 AWS IoT Greengrass가 리소스 아티팩트를 다운로드하는 데 사용하는 OS 그룹 및 권한을 나타냅니다. 리소스 소유자가 정의되지 않은 경우 다운로드한 리소스 아티팩트는 루트에 대해서만 액세스가 가능합니다.

- 컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 액세스하는 경우 컨테이너에서 권한 제어가 없으므로 리소스 소유자를 정의해야 합니다. 컨테이너화되지 않은 Lambda 함수는 리소스 소유자 권한을 상속하여 리소스에 액세스하는 데 사용할 수 있습니다.

- 컨테이너화된 Lambda 함수만 리소스에 액세스하는 경우 리소스 소유자를 정의하는 대신 함수 수준 권한을 사용하는 것이 좋습니다.

리소스 소유자 속성

리소스 소유자는 그룹 소유자 및 그룹 소유자 권한을 지정합니다.

그룹 소유자. 코어 디바이스에 있는 기존 Linux OS 그룹의 그룹 ID(GID)입니다. 그룹의 사용 권한이 Lambda 프로세스에 추가됩니다. 특히 GID가 Lambda 함수의 보충 그룹 ID에 추가됩니다.

Greengrass 그룹의 Lambda 함수가 기계 학습 리소스의 리소스 소유자와 동일한 OS 그룹으로 실행되도록 구성된 경우 해당 리소스가 Lambda 함수에 연결되어야 합니다. 그렇지 않으면 이 구성이 AWS IoT Greengrass 인증 없이 리소스에 액세스하는 데 Lambda 함수가 사용할 수 있는 암시적 권한을 부여하기 때문에 배포가 실패합니다. Lambda 함수가 루트(UID=0)로 실행되는 경우 배포 유효성 검사를 건너뜁니다.

Greengrass 코어의 다른 리소스, Lambda 함수 또는 파일에서 사용하지 않는 OS 그룹을 사용하는 것이 좋습니다. 공유 OS 그룹을 사용하면 연결된 Lambda 함수에 필요한 것보다 더 많은 액세스 권한이 부여됩니다. 공유 OS 그룹을 사용하는 경우, 연결된 Lambda 함수가 공유 OS 그룹을 사용하는 모든 기계 학습 리소스에도 연결되어야 합니다. 그렇지 않으면 배포가 실패합니다.

그룹 소유자 권한. Lambda 프로세스에 추가할 읽기 전용 또는 읽기 및 쓰기 권한입니다.

컨테이너화되지 않은 Lambda 함수는 리소스에 대한 이러한 액세스 권한을 상속해야 합니다. 컨테이너화된 Lambda 함수는 이러한 리소스 수준 권한을 상속하거나 함수 수준 권한을 정의할 수 있습니다. 함수 수준 권한을 정의하는 경우, 권한은 리소스 수준 권한과 같거나 더 제한적이어야 합니다.

다음 테이블은 지원되는 액세스 권한 구성을 보여줍니다.

GGC v1.10 or later

속성	컨테이너화된 Lambda 함수만 리소스에 액세스하는 경우	컨테이너화되지 않은 Lambda 함수가 리소스에 액세스하는 경우
함수 수준 속성		
권한(읽기/쓰기)	리소스가 리소스 소유자를 정의하지 않는 한 필수입니다. 리소스 소유자가 정의된 경우, 함수 수준 권한은 리소스 소유자 권한과 같거나 더 제한적이어야 합니다.	컨테이너화되지 않은 Lambda 함수: 지원하지 않음. 컨테이너화되지 않은 Lambda 함수는 리소스 수준 권한을 상속해야 합니다. 컨테이너화된 Lambda 함수:

속성	컨테이너화된 Lambda 함수만 리소스에 액세스하는 경우	컨테이너화되지 않은 Lambda 함수가 리소스에 액세스하는 경우
	컨테이너화된 Lambda 함수만 리소스에 액세스하는 경우 리소스 소유자를 정의하지 않는 것이 좋습니다.	선택 사항이지만 리소스 수준 권한과 같거나 더 제한적이어야 합니다.
리소스 수준 속성		
리소스 소유자	선택 사항 (권장하지 않음).	필수 사항입니다.
권한(읽기/쓰기)	선택 사항 (권장하지 않음).	필수 사항입니다.

GGC v1.9 or earlier

속성	컨테이너화된 Lambda 함수만 리소스에 액세스하는 경우	컨테이너화되지 않은 Lambda 함수가 리소스에 액세스하는 경우
함수 수준 속성		
권한(읽기/쓰기)	필수 사항입니다.	지원하지 않음.
리소스 수준 속성		
리소스 소유자	지원하지 않음.	지원하지 않음.
권한(읽기/쓰기)	지원하지 않음.	지원하지 않음.

> **Note**
>
> AWS IoT Greengrass API를 사용하여 Lambda 함수와 리소스를 구성하는 경우 함수 수준 `ResourceId` 속성도 필요합니다. `ResourceId` 속성은 기계 학습 리소스를 Lambda 함수에 연결합니다.

Lambda 함수에 대한 액세스 권한 정의(콘솔)

AWS IoT 콘솔에서는 기계 학습 리소스를 구성하거나 Lambda 함수에 연결할 때 액세스 권한을 정의합니다.

컨테이너화된 Lambda 함수

컨테이너화된 Lambda 함수만 기계 학습 리소스에 연결되는 경우:

- 기계 학습 리소스의 리소스 소유자로 시스템 그룹 없음을 선택합니다. 컨테이너화된 Lambda 함수만 기계 학습 리소스에 액세스할 때 권장되는 설정입니다. 아니면 연결된 Lambda 함수에 필요한 것보다 많은 액세스 권한을 부여할 수 있습니다.

컨테이너화되지 않은 Lambda 함수(GGC v1.10 이상 필요)

컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 연결된 경우:

- 기계 학습 리소스의 리소스 소유자로 사용할 시스템 그룹의 ID(GID)를 지정합니다. 시스템 그룹 및 권한 지정을 선택하고 GID를 입력합니다. 코어 디바이스의 `getent group` 명령을 사용하여 시스템 그룹의 ID를 조회할 수 있습니다.

- 시스템 그룹 권한으로 읽기 전용 액세스 또는 읽기 및 쓰기 액세스를 선택합니다.

Lambda 함수에 대한 액세스 권한 정의(API)

AWS IoT Greengrass API의 경우, Lambda 함수의 `ResourceAccessPolicy` 속성이나 리소스의 `OwnerSetting` 속성에서 기계 학습 리소스에 대한 권한을 정의합니다.

컨테이너화된 Lambda 함수

컨테이너화된 Lambda 함수만 기계 학습 리소스에 연결되는 경우:

- 컨테이너화된 Lambda 함수의 경우 `ResourceAccessPolicies` 속성의 `Permission` 속성에서 액세스 권한을 정의합니다. 예:

```
"Functions": [
    {
        "Id": "my-containerized-function",
```

```
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw"
                    }
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- 기계 학습 리소스의 경우 `OwnerSetting` 속성을 생략합니다. 예:

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package"
            }
        }
    }
]
```

컨테이너화된 Lambda 함수만 기계 학습 리소스에 액세스할 때 권장되는 구성입니다. 아니면 연결된 Lambda 함수에 필요한 것보다 많은 액세스 권한을 부여할 수 있습니다.

컨테이너화되지 않은 Lambda 함수(GGC v1.10 이상 필요)

컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 연결된 경우:

- 컨테이너화되지 않은 Lambda 함수의 경우, `ResourceAccessPolicies`에서 `Permission` 속성을 생략합니다. 이 구성은 필수이며 함수가 리소스 수준 권한을 상속할 수 있게 해줍니다. 예:

```
"Functions": [
    {
        "Id": "my-non-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "Execution": {
                    "IsolationMode": "NoContainer",
                },
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id"
                    }
                ]
            },
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- 기계 학습 리소스에도 액세스하는 컨테이너화된 Lambda 함수의 경우, `ResourceAccessPolicies`에서 `Permission` 속성을 생략하거나 리소스 수준 권한과 같거나 더 제한적인 권한을 정의합니다. 예:

```
"Functions": [
    {
        "Id": "my-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw" // Optional, but cannot exceed
 the GroupPermission defined for the resource.
                    }
                ]
            },
            "MemorySize": 512,
```

```
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- 기계 학습 리소스의 경우, 하위 `GroupOwner` 및 `GroupPermission` 속성을 포함해 `OwnerSetting` 속성을 정의합니다. 예:

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package",
                "OwnerSetting": {
                    "GroupOwner": "os-group-id",
                    "GroupPermission": "ro-or-rw"
                }
            }
        }
    }
]
```

Lambda 함수 코드에서 기계 학습 리소스에 액세스

사용자 정의 Lambda 함수는 플랫폼별 OS 인터페이스를 사용하여 코어 디바이스의 기계 학습 리소스에 액세스합니다.

GGC v1.10 or later

컨테이너화된 Lambda 함수의 경우, 리소스는 Greengrass 컨테이너 내부에 마운트되며 해당 리소스에 대해 정의된 로컬 대상 경로에서 사용할 수 있습니다. 컨테이너화되지 않은 Lambda 함수의 경우, 리소스는 Lambda 특정 작업 디렉터리에 symlink되어 Lambda 프로세스의 `AWS_GG_RESOURCE_PREFIX` 환경 변수로 전달됩니다.

기계 학습 리소스의 다운로드된 아티팩트에 대한 경로를 얻기 위해 Lambda 함수는 리소스에 대해 정의된 로컬 대상 경로에 `AWS_GG_RESOURCE_PREFIX` 환경 변수를 추가합니다. 컨테이너화된 Lambda 함수의 경우 반환된 값은 단일 슬래시(`/`)입니다.

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

기계 학습 리소스의 다운로드된 아티팩트는 리소스에 대해 정의된 로컬 대상 경로에 위치합니다. 컨테이너화된 Lambda 함수만 AWS IoT Greengrass 코어 v1.9 이전 버전의 기계 학습 리소스에 액세스할 수 있습니다.

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

모델 로딩 구현은 ML 라이브러리에 따라 다릅니다.

문제 해결

다음 정보를 사용하여 기계 학습 리소스 액세스와 관련된 문제를 해결할 수 있습니다.

주제

- InvalidML ModelOwner - GroupOwnerSetting ML 모델 리소스에 제공되었지만 존재하지 GroupOwner 않음 GroupPermission
- NoContainer 함수는 Machine Learning 리소스를 연결할 때 권한을 구성할 수 없습니다. <function-arn>리소스 액세스 정책에 <resource-id>권한이 <ro/rw> 있는 기계 학습 리소스를 말합니다.
- 함수는 <function-arn><resource-id> ResourceAccessPolicy 및 리소스 모두에서 권한이 누락된 Machine Learning 리소스를 가리킵니다 OwnerSetting.
- 함수는 <function-arn><resource-id>\ "rw\" 권한이 있는 Machine Learning 리소스를 가리키는 반면, 리소스 소유자 설정은\ "ro\ GroupPermission "만 허용합니다.
- NoContainer 함수는 <function-arn>중첩된 대상 경로의 리소스를 가리킵니다.
- Lambda <function-arn>은 동일한 그룹 소유자 ID를 공유하여 리소스 <resource-id>에 대한 액세스 권한을 획득합니다.

InvalidML ModelOwner - GroupOwnerSetting ML 모델 리소스에 제공되었지만 존재하지 GroupOwner 않음 GroupPermission

해결 방법: 기계 학습 리소스에 ResourceDownloadOwnerSetting개체가 포함되어 있지만 필수 `GroupOwner` 또는 `GroupPermission` 속성이 정의되지 않은 경우 이 오류가 발생합니다. 이 문제를 해결하려면 누락된 속성을 정의합니다.

NoContainer 함수는 Machine Learning 리소스를 연결할 때 권한을 구성할 수 없습니다. <function-arn>리소스 액세스 정책에 <resource-id>권한이 <ro/rw> 있는 기계 학습 리소스를 말합니다.

해결 방법: 컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 대한 함수 수준 권한을 지정하는 경우 이 오류가 발생합니다. 컨테이너화되지 않은 함수는 기계 학습 리소스에 정의된 리소스 소유자 권한으로부터 권한을 상속해야 합니다. 이 문제를 해결하려면 리소스 소유자 권한(콘솔)을 상속하거나, Lambda 함수의 리소스 액세스 정책(API)에서 권한을 제거하도록 선택하십시오.

함수는 <function-arn><resource-id> ResourceAccessPolicy 및 리소스 모두에서 권한이 누락된 Machine Learning 리소스를 가리킵니다 OwnerSetting.

해결 방법: 기계 학습 리소스에 대한 권한이 연결된 Lambda 함수 또는 리소스에 대해 구성되지 않은 경우 이 오류가 발생합니다. 이 문제를 해결하려면 Lambda 함수의 ResourceAccessPolicy속성 또는 리소스의 속성에서 권한을 OwnerSetting구성하십시오.

함수는 <function-arn><resource-id>\ "rw\" 권한이 있는 Machine Learning 리소스를 가리키는 반면, 리소스 소유자 설정은\ "ro\ GroupPermission "만 허용합니다.

해결 방법: 연결된 Lambda 함수에 대해 정의된 액세스 권한이 기계 학습 리소스에 대해 정의된 리소스 소유자 권한을 초과하는 경우 이 오류가 발생합니다. 이 문제를 해결하려면 Lambda 함수에 대해 더 제한적인 권한을 설정하거나, 리소스 소유자에 대해 덜 제한적인 권한을 설정합니다.

NoContainer 함수는 <function-arn>중첩된 대상 경로의 리소스를 가리킵니다.

해결 방법: 컨테이너화되지 않은 Lambda 함수에 연결된 여러 개의 기계 학습 리소스가 동일한 대상 경로 또는 중첩된 대상 경로를 사용하는 경우 이 오류가 발생합니다. 이 문제를 해결하려면 리소스에 대해 별도의 대상 경로를 지정합니다.

Lambda <function-arn>은 동일한 그룹 소유자 ID를 공유하여 리소스 <resource-id>에 대한 액세스 권한을 획득합니다.

해결 방법: 동일한 OS 그룹이 Lambda 함수의 다음으로 실행 ID와 기계 학습 리소스의 리소스 소유자로 지정되었지만 리소스가 Lambda 함수에 연결되지 않은 경우 `runtime.log`에 이 오류가 발생합니다. 이 구성은 Lambda 함수에 AWS IoT Greengrass 인증 없이 리소스에 액세스하는 데 사용할 수 있는 암시적 권한을 부여합니다.

이 문제를 해결하려면 속성 중 하나에 대해 다른 OS 그룹을 사용하거나 기계 학습 리소스를 Lambda 함수에 연결합니다.

다음 사항도 참조하십시오.

- 기계 학습 추론 수행
- the section called “기계 학습 추론을 구성하는 방법”
- the section called “최적화된 기계 학습 추론을 구성하는 방법”
- AWS IoT Greengrass Version 1 API 참조

AWS Management Console을 사용하여 기계 학습 추론을 구성하는 방법

이 튜토리얼의 단계를 수행하려면 AWS IoT Greengrass Core v1.10 이상이 필요합니다.

로컬로 생성된 데이터를 사용하여 Greengrass 코어 디바이스에서 로컬 방식으로 기계 학습(ML) 추론을 수행할 수 있습니다. 요구 사항과 제한 조건을 비롯한 자세한 내용은 기계 학습 추론 수행 단원을 참조하십시오.

이 자습서는 AWS Management Console을(를) 사용하고 Greengrass 그룹을 구성하여 클라우드로 데이터를 전송하지 않고 로컬 방식으로 카메라의 이미지를 인식하는 추론 앱을 실행합니다. 추론 앱은

Raspberry Pi의 카메라 모듈에 액세스하고 오픈 소스 SqueezeNet 모델을 사용하여 추론을 실행합니다.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Raspberry Pi 구성
2. MXNet 프레임워크 설치
3. 모델 패키지 생성
4. Lambda 함수 생성 및 게시
5. 그룹에 Lambda 함수를 추가합니다.
6. 그룹에 리소스 추가
7. 그룹에 구독 추가
8. 그룹 배포
9. 앱 테스트

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- Raspberry Pi 4 모델 B 또는 Raspberry Pi 3 모델 B/B+는 AWS IoT Greengrass와(과) 함께 사용하도록 설정 및 구성되었습니다. AWS IoT Greengrass로 Raspberry Pi를 설정하기 위해, Greengrass 디바이스 설정 스크립트를 실행하거나 모듈 1 및 모듈 2를 완료했는지 시작하기 AWS IoT Greengrass 확인합니다.

  > **Note**
  >
  > Raspberry Pi는 일반적으로 이미지 분류에 사용되는 딥 러닝 프레임워크를 실행하기 위해 2.5A 전원 공급 장치가 필요할 수 있습니다. 정격이 더 낮은 전원 공급 장치는 디바이스가 재부팅될 수 있습니다.

- Raspberry Pi Camera Module V2 – 8메가픽셀, 1080p. 카메라 설정 방법을 알아보려면 Raspberry Pi 설명서의 Connecting the camera를 참조하십시오.
- Greengrass 그룹 및 Greengrass 코어. Greengrass 그룹 또는 코어를 생성하는 방법에 대한 자세한 내용은 시작하기 AWS IoT Greengrass을(를) 참조하세요.

> **Note**
>
> 이 자습서는 Raspberry Pi를 사용하지만 AWS IoT Greengrass는 Intel Atom과 NVIDIA Jetson TX2 등의 다른 플랫폼도 지원합니다. Jetson TX2의 예제에서는 카메라에서 스트리밍된 이미지 대신 정적 이미지를 사용할 수 있습니다. Jetson TX2 예제를 사용하는 경우 Python 3.7 대신 Python 3.6을 설치해야 할 수도 있습니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있도록 디바이스를 구성하는 방법에 대한 자세한 내용은 the section called “다른 디바이스 설정”를 참조하십시오.
>
> AWS IoT Greengrass이(가) 지원하지 않는 타사 플랫폼의 경우, Lambda 함수를 비컨테이너식 모드에서 실행해야 합니다. 비컨테이너식 모드에서 실행하려면 Lambda 함수를 루트로 실행해야 합니다. 자세한 정보는 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 및 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

1단계: Raspberry Pi 구성

이 단계에서는 Raspbian 운영 체제에 업데이트를 설치하고, 카메라 모듈 소프트웨어와 Python 종속성을 설치하며, 카메라 인터페이스를 활성화합니다.

Raspberry Pi 터미널에서 다음 명령을 실행합니다.

1. Raspbian에 업데이트를 설치합니다.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. 이 자습서에 필요한 카메라 모듈에 대한 `picamera` 인터페이스 및 기타 Python 라이브러리를 설치합니다.

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   설치를 검사합니다.

   - Python 3.7 설치에 pip가 포함되어 있는지 확인합니다.

     ```
     python3 -m pip
     ```

     pip가 설치되어 있지 않으면 pip 웹 사이트에서 다운로드한 후 다음 명령을 실행합니다.

```
python3 get-pip.py
```

- Python 버전이 3.7 이상인지 확인합니다.

```
python3 --version
```

출력에 이전 버전이 나열되면 다음 명령을 실행합니다.

```
sudo apt-get install -y python3.7-dev
```

- Setuptools 및 Picamera가 성공적으로 설치되었는지 확인합니다.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

출력에 오류가 포함되어 있지 않으면 검사가 성공한 것입니다.

> **Note**
>
> 디바이스에 설치된 Python 실행 파일이 `python3.7`인 경우 이 자습서의 명령에 `python3` 대신 `python3.7`을 사용합니다. 종속성 오류를 피하기 위해 pip 설치가 올바른 `python3.7` 또는 `python3` 버전에 매핑되는지 확인하십시오.

3. Raspberry Pi를 재부팅합니다.

```
sudo reboot
```

4. Raspberry Pi 구성 도구를 엽니다.

```
sudo raspi-config
```

5. 화살표 키를 사용하여 [Interfacing Options]를 열고 카메라 인터페이스를 활성화합니다. 메시지가 나타나면 디바이스 재부팅을 허용합니다.
6. 다음 명령을 사용하여 카메라 설정을 테스트합니다.

```
raspistill -v -o test.jpg
```

그러면 Raspberry Pi의 미리 보기 창이 열리고, 현재 디렉터리에 `test.jpg`라는 이름의 사진이 저장되며, 카메라에 대한 정보가 Raspberry Pi 터미널에 표시됩니다.

2단계: MXNet 프레임워크 설치

이 단계에서는 Raspberry Pi에 MXNet 라이브러리를 설치합니다.

1. Raspberry Pi에 원격으로 로그인합니다.

   ```
   ssh pi@your-device-ip-address
   ```

2. MXNet 설명서의 MXNet 설치에 나온 지침을 따라 디바이스에 MXNet을 설치합니다.

   > **Note**
   >
   > 디바이스 충돌을 방지하려면 이 튜토리얼에서는 버전 1.5.0을 설치하고 소스에서 MXNet을 구축하는 것을 권장합니다.

3. MXNet을 설치한 후 다음 구성을 검사합니다.

   - `ggc_user` 시스템 계정에서 MXNet 프레임워크를 사용할 수 있는지 확인합니다.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
     ```

   - NumPy가 설치되어 있는지 확인합니다.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

3단계: MXNet 모델 패키지 생성

이 단계에서는 Amazon Simple Storage Service (Amazon S3)에 업로드할 사전 훈련된 MXNet 모델 샘플을 포함하는 모델 패키지를 생성합니다. AWS IoT Greengrass는 의 모델 패키지(tar.gz 또는 zip 형식인 경우)를 사용할 수 있습니다.

1. the section called “기계 학습 샘플”에 나온 Raspberry Pi용 MXNet 샘플을 컴퓨터에 다운로드합니다.
2. 다운로드한 `mxnet-py3-armv7l.tar.gz` 파일의 압축을 풉니다.

3. `squeezenet` 디렉터리로 이동합니다.

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
```

이 디렉터리의 `squeezenet.zip` 파일은 모델 패키지로, 이미지 분류 모델에 대한 SqueezeNet 오픈 소스 모델 아티팩트를 포함하고 있습니다. 이후에 이 모델 패키지를 Amazon S3에 업로드합니다.

4단계: Lambda 함수 생성 및 게시

이 단계에서는 Lambda 함수 배포 패키지 및 Lambda 함수를 생성합니다. 그런 다음 함수 버전을 게시하고 별칭을 생성합니다.

먼저 Lambda 함수 배포 패키지를 생성합니다.

1. 컴퓨터에서 the section called “모델 패키지 생성”에서 압축을 푼 샘플 패키지의 `examples` 디렉터리로 이동합니다.

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
```

`examples` 디렉터리에는 함수 코드와 종속성이 포함되어 있습니다.

- `greengrassObjectClassification.py`는 이 자습서에서 사용되는 추론 코드입니다. 이 코드를 템플릿으로 사용하여 사용자 고유의 추론 함수를 생성할 수 있습니다.
- `greengrasssdk`은(는) Python용 AWS IoT Greengrass 코어 SDK 버전 1.5.0입니다.

> **Note**
>
> 새 버전을 사용할 수 있는 경우 해당 버전을 다운로드하여 배포 패키지의 SDK 버전을 업그레이드할 수 있습니다. 자세한 내용은 GitHub의 Python용 AWS IoT Greengrass 코어 SDK를 참조하세요.

2. `examples` 디렉터리의 내용을 `greengrassObjectClassification.zip`라는 파일로 압축합니다. 이것은 배포 패키지입니다.

```
zip -r greengrassObjectClassification.zip .
```

> **Note**
>
> `.py` 파일 및 종속성이 디렉터리의 루트에 있는지 확인합니다.

다음으로, Lambda 함수를 생성합니다.

3. AWS IoT 콘솔에서 함수를 선택하고 함수 생성을 선택합니다.
4. 새로 작성을 선택하고 다음 값을 사용하여 함수를 생성합니다.

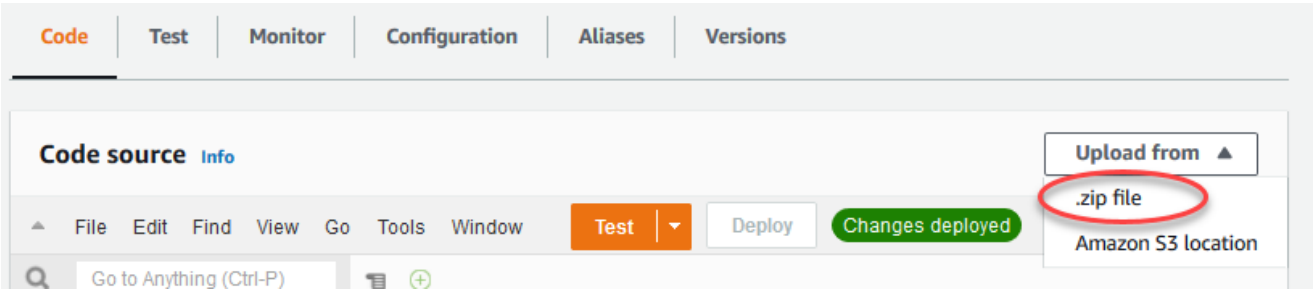
   - [함수 이름(Function name)]에 **`greengrassObjectClassification`**을 입력합니다.
   - 실행 시간에서 Python 3.7을 선택합니다.

   권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.
5. 함수 생성(Create function)을 선택합니다.

이제 Lambda 함수 배포 패키지를 업로드하고 핸들러를 등록합니다.

6. Lambda 함수를 선택하고 Lambda 함수 배포 패키지를 업로드하십시오.

   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



   b. 업로드를 선택한 다음 `greengrassObjectClassification.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.

   c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

      - 실행 시간에서 Python 3.7을 선택합니다.

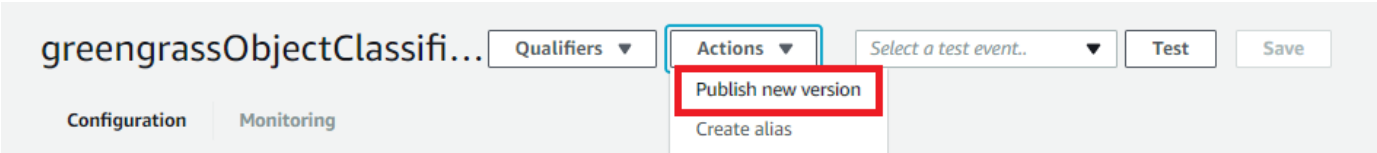
- 핸들러에 **greengrassObjectClassification.function_handler**를 입력합니다.

Save를 선택합니다.

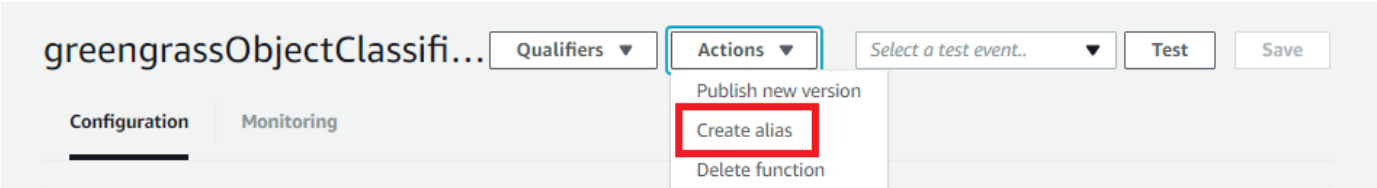
그런 다음 Lambda 함수의 첫 번째 버전을 게시합니다. 그런 다음 버전의 별칭을 생성합니다.

> **Note**
>
> Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

7. [Actions] 메뉴에서 [Publish new revision]을 선택합니다.



8. 버전 설명에 **First version**을 입력한 후 게시를 선택합니다.
9. [greengrassObjectClassification: 1] 구성 페이지의 [Actions] 메뉴에서 [Create alias]를 선택합니다.



10. [Create a new alias] 페이지에서 다음 값을 사용합니다.

    - 이름(Name)에 **mlTest**을 입력합니다.
    - 버전에 **1**를 입력합니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

11. Save를 선택합니다.

이제 Greengrass 그룹에 Lambda 함수를 추가합니다.

5단계: Greengrass 그룹에 Lambda 함수 추가

이 단계에서 그룹에 Lambda 함수를 추가한 다음 수명 주기와 환경 변수를 구성합니다.

먼저 Greengrass 그룹에 Lambda 함수를 추가합니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
3. 내 Lambda 함수 섹션에서 추가를 선택합니다.
4. Lambda 함수에서 greengrassObjectClassification를 선택합니다.
5. Lambda 함수 버전의 경우 Alias:mlTest를 선택하십시오.

그 다음, Lambda 함수의 수명 주기와 환경 변수를 구성합니다.

6. Lambda 함수 구성 섹션에서 다음과 같이 업데이트하십시오.

> **Note**
>
> 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

a. 컨테이너화 없이 실행하려면:

- 시스템 사용자 및 그룹의 경우 **`Another user ID/group ID`**을(를) 선택합니다. 시스템 사용자 ID에 **`0`**을(를) 입력합니다. 시스템 그룹 ID에 **`0`**을(를) 입력합니다.

  이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

> **Tip**
>
> 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.
- 시간 제한에 **`10 seconds`**를 입력합니다.
- 고정된 경우 True를 선택합니다.

  자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 시스템 사용자 및 그룹의 경우 그룹 기본값 사용을 선택합니다.
- Lambda 함수 컨테이너화의 경우 그룹 기본값 사용을 선택합니다.
- 메모리 제한에 **`96 MB`**를 입력합니다.
- 시간 제한에 **`10 seconds`**를 입력합니다.
- 고정된 경우 True를 선택합니다.

  자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.

7. 환경 변수에서 키-값 페어를 생성합니다. 키-값 페어는 Raspberry Pi에서 MXNet 모델과 상호 작용하는 함수에 필요합니다.

   키에 MXNET_ENGINE_TYPE를 사용합니다. 값에 NaiveEngine을 사용합니다.

> **Note**
>
> 자신의 사용자 정의 Lambda 함수에서는 함수 코드에 환경 변수를 선택적으로 설정할 수 있습니다.

8. 다른 모든 속성을 위해 기본 값을 유지하고 Lambda 함수 추가를 선택합니다.

6단계: Greengrass 그룹에 리소스 추가

이 단계에서 카메라 모듈 및 ML 추론 모델에 대한 리소스를 생성하고 Lambda 함수와 리소스를 연결하여 Lambda 함수가 코어 디바이스의 리소스에 액세스할 수 있도록 합니다.

> **Note**
>
> 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 이 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다.

우선 카메라에 대해 2개의 로컬 디바이스 리소스를 생성합니다. 하나는 공유 메모리용이고, 다른 하나는 디바이스 인터페이스용입니다. 로컬 리소스 액세스에 대한 자세한 내용은 Lambda 함수와 커넥터를 사용하여 로컬 리소스에 액세스 단원을 참조하십시오.

1. 그룹 구성 페이지에서 리소스를 선택합니다.
2. 로컬 리소스 탭에서 로컬 리소스 추가를 선택합니다.
3. 로컬 리소스 생성 페이지에서 다음 값을 사용합니다.

   - 리소스 이름에 **`videoCoreSharedMemory`**을 입력합니다.
   - [Resource type]에서 [Device]를 선택합니다.
   - 디바이스 경로에 **`/dev/vcsm`**를 입력합니다.

     디바이스 경로는 디바이스 리소스의 로컬 절대 경로입니다. 이 경로는 `/dev` 아래에 있는 문자 디바이스 또는 블록 디바이스만 참조할 수 있습니다.
   - System group owner and file access permissions에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.

시스템 그룹 소유자 및 파일 액세스 권한 옵션을 사용하면 Lambda 프로세스에 추가 파일 액세스 권한을 부여할 수 있습니다. 자세한 내용은 그룹 소유자 파일 액세스 권한 섹션을 참조하세요.

4. 다음으로 카메라 인터페이스에 대한 로컬 디바이스 리소스를 추가합니다.
5. 로컬 리소스 추가를 선택합니다.
6. 로컬 리소스 생성 페이지에서 다음 값을 사용합니다.

   - 리소스 이름에 **videoCoreInterface**을 입력합니다.
   - [Resource type]에서 [Device]를 선택합니다.
   - 디바이스 경로에 **/dev/vchiq**를 입력합니다.
   - [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.
7. 페이지 하단에서 다른 리소스 추가를 선택합니다.

이제 기계 학습 리소스로서 추론 모델을 추가합니다. 이 단계에서 `squeezenet.zip` 모델 패키지를 Amazon S3에 업로드합니다.

1. 리소스 페이지에서 기계 학습을 선택한 후 기계 학습 리소스 추가를 선택합니다.
2. 머신 러닝 리소스 생성 페이지에서 리소스 이름에 **squeezenet_model**을 입력합니다.
3. 모델 소스의 경우 S3에 저장된 모델(예: 딥 러닝 컴파일러를 통해 최적화된 모델)을 선택합니다.
4. S3 URI의 경우 S3 버킷이 저장된 경로를 입력합니다.
5. S3 찾아보기(Browse S3)를 선택합니다. 그러면 새로운 Amazon S3 콘솔 탭이 열립니다.
6. Amazon S3 콘솔 탭에서 `squeezenet.zip` 파일을 S3 버킷에 업로드합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 S3 버킷에 파일 및 폴더를 업로드하려면 어떻게 해야 합니까?를 참조하십시오.

> **Note**
>
> S3 버킷에 액세스할 수 있으려면 버킷 이름에 문자열 **greengrass**이(가) 포함되어야 하고 버킷은 AWS IoT Greengrass을(를) 사용하는 리전과 동일한 리전에 있어야 합니다. 고유한 이름을 선택합니다(예: **greengrass-bucket-*user-id*-*epoch-time***). 버킷 이름에는 마침표(.)를 사용하지 마십시오.

7. AWS IoT Greengrass 콘솔 탭에서S3 버킷을 찾아 선택합니다. 업로드한 `squeezenet.zip` 파일을 찾아 선택을 선택합니다. 사용 가능한 버킷과 파일의 목록을 업데이트하려면 새로 고침을 선택하십시오.
8. [Destination path]에 [**`/greengrass-machine-learning/mxnet/squeezenet`**]을 입력합니다.

   Lambda 런타임 네임스페이스에 있는 로컬 모델의 대상입니다. 그룹을 배포할 때 AWS IoT Greengrass은(는) 소스 모델 패키지를 검색하고 지정된 디렉터리에 내용을 압축 해제합니다. 이 자습서에 대한 샘플 Lambda 함수는 이미 (`model_path` 변수에서) 이 경로를 사용하도록 구성되었습니다.
9. 시스템 그룹 소유자 및 파일 액세스 권한에서 시스템 그룹 없음을 선택합니다.
10. 리소스 추가를 선택합니다.

SageMaker 학습 모델 사용

이 자습서는 Amazon S3에 저장된 모델을 사용합니다. 하지만 모델 또한 손쉽게 사용할 수 있습니다. AWS IoT Greengrass 콘솔에는 통합이 내장되어 있습니다. 따라서 수동으로 이러한 모델을 Amazon S3로 업로드할 필요가 없습니다. SageMaker 모델 사용 요구 사항 및 제한 사항은 the section called “지원되는 모델 소스”을 참조하십시오.

SageMaker 모델을 사용하려면:

- Model source에서 Use a model trained in AWS SageMaker을 선택한 다음 모델의 교육 작업 이름을 선택합니다.
- 대상 경로에 Lambda 함수가 모델을 찾는 디렉터리 경로를 입력합니다.

7단계: Greengrass 그룹에 구독 추가

이 단계에서는 그룹에 구독을 추가합니다. 이 구독을 통해 Lambda 함수가 MQTT 주제를 게시함으로써 AWS IoT에 결과를 전송할 수 있습니다.

1. 그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.
2. 구독 세부정보 페이지에서 다음과 같이 원본 및 대상을 구성합니다.

   a. 소스 유형에서 Lambda 함수를 선택한 다음 greengrassObjectClassification을 선택합니다.

   b. 대상 선택에서 서비스를 선택한 후 IoT Cloud를 선택합니다.

3. 주제 필터에서 **hello/world**을(를) 입력한 다음 구독 생성을 선택합니다.

8단계: Greengrass 그룹 배포

이 단계에서는 그룹 정의의 현재 버전을 Greengrass 코어 디바이스로 배포합니다. 정의에는 추가한 Lambda 함수, 리소스 및 구독 구성이 포함됩니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 데몬이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

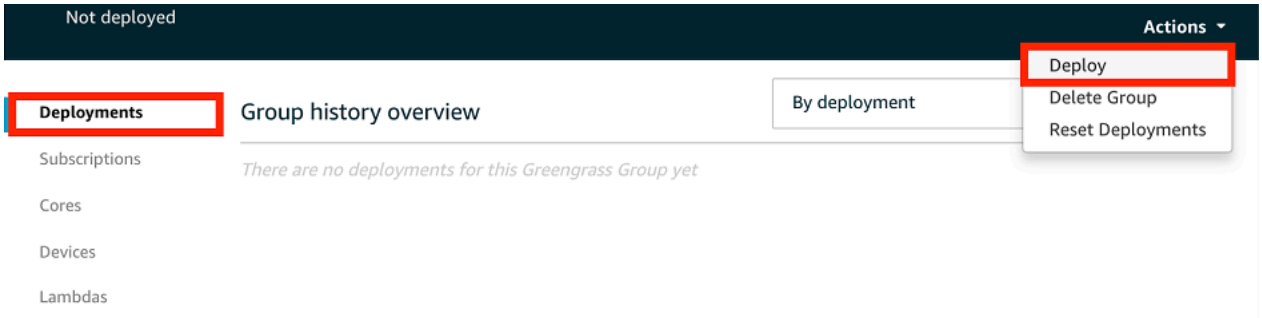
   출력에 root에 대한 /greengrass/ggc/packages/1.11.6/bin/daemon 입력이 포함되어 있는 경우에는 데몬이 실행 중인 것입니다.

   > **Note**
   > 경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

   b. 데몬을 시작하려면:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. 그룹 구성 페이지에서 배포를 선택합니다.



3. Lambda 함수 탭의 시스템 Lambda 함수 섹션에서 IP 감지기를 선택하고 편집을 선택합니다.
4. IP 감지기 설정 편집 대화 상자에서 MQTT 브로커 엔드포인트 자동 탐지 및 재정의를 선택합니다.

5. Save를 선택합니다.

   이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

   > **Note**
   >
   > 메시지가 표시되면 Greengrass 서비스 역할을 생성할 권한을 부여하고 이 권한을 현재 AWS 리전의 AWS 계정과 연결합니다. 이 역할은 AWS IoT Greengrass가 AWS 서비스의 리소스에 액세스할 수 있습니다.

   배포 페이지에 배포 타임스탬프, 버전 ID, 상태가 표시됩니다. 완료되면 배포에 대해 성공적으로 완료했습니다 상태가 표시되어야 합니다.

   배포에 대한 자세한 내용은 AWS IoT Greengrass 그룹 배포 섹션을 참조하세요. 문제 해결에 대한 도움말은 문제 해결 단원을 참조하십시오.

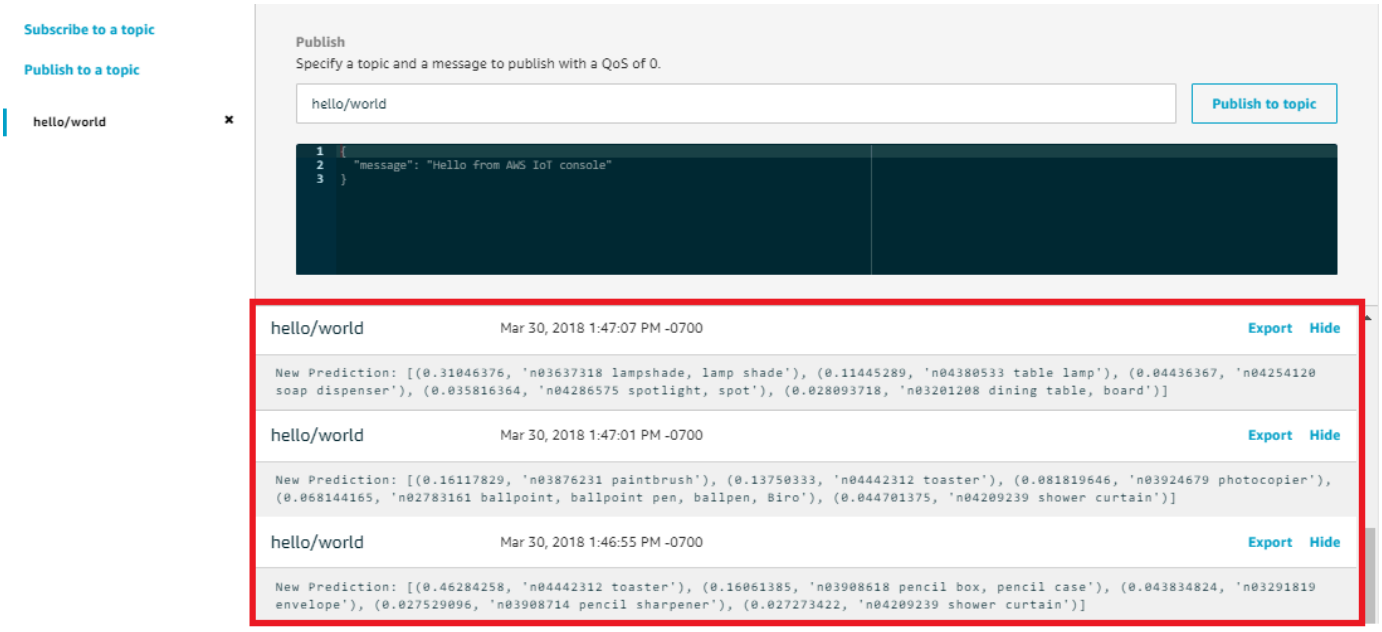
9단계: 추론 앱 테스트

이제 배포가 올바르게 구성되었는지 확인할 수 있습니다. 테스트하려면 `hello/world` 주제를 구독하고 Lambda 함수가 게시하는 예측 결과를 봅니다.

> **Note**
>
> Raspberry Pi에 모니터가 연결된 경우 라이브 카메라 피드가 미리 보기 창에 표시됩니다.

1. AWS IoT 콘솔의 테스트에서 MQTT 테스트 클라이언트를 선택합니다.
2. [Subscriptions]에서 다음 값을 사용합니다.
   - 구독 주제에 hello/world를 사용합니다.
   - MQTT 페이로드 표시의 경우 추가 구성에서 페이로드를 문자열로 표시를 선택합니다.
3. 구독을 선택합니다.

테스트가 성공적인 경우 페이지 하단에 Lambda 함수로부터의 메시지가 표시됩니다. 각 메시지에는 이미지의 상위 5개 예측 결과가 포함되며 가능성, 예측된 클래스 ID 및 해당 클래스 이름 형식을 사용합니다.



AWS IoT Greengrass ML 추론 문제 해결

테스트가 성공적이지 않은 경우 다음 문제 해결 단계를 시도할 수 있습니다. Raspberry Pi 터미널에서 명령을 실행합니다.

오류 로그 확인

1. 루트 사용자로 전환하고 `log` 디렉터리로 이동합니다. AWS IoT Greengrass 로그 액세스에는 루트 권한이 필요합니다.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. `system` 디렉터리에서 `runtime.log` 또는 `python_runtime.log`를 확인합니다.

   `user/`*`region`*`/`*`account-id`* 디렉터리에서 `greengrassObjectClassification.log`를 확인합니다.

   자세한 내용은 the section called “로그 문제 해결” 섹션을 참조하세요.

runtime.log의 압축 해제 오류

`runtime.log`에 다음과 유사한 오류가 포함된 경우 `tar.gz` 소스 모델 패키지에 상위 디렉터리가 있어야 합니다.

```
Greengrass deployment error: unable to download the artifact model-arn: Error while
 processing.
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /
greengrass/ggc/deployment/path/model-arn,
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/
squeezenet_v1.1-0000.params: no such file or directory
```

패키지에 모델 파일을 포함하는 상위 디렉터리가 없는 경우 다음 명령을 사용하여 모델을 리패키지합니다.

```
tar -zcvf model.tar.gz ./model
```

예:

```
#$ tar -zcvf test.tar.gz ./test
./test
./test/some.file
./test/some.file2
./test/some.file3
```

> **Note**
>
> 이 명령에 후행 `/*` 문자를 포함시키지 마십시오.

Lambda 함수가 성공적으로 배포되었는지 확인

1. `/lambda` 디렉터리에 있는 배포된 Lambda의 내용을 나열합니다. 명령을 실행하기 전에 자리 표시자 값을 바꿉니다.

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. 디렉터리에 4단계: Lambda 함수 생성 및 게시에 업로드한 `greengrassObjectClassification.zip` 배포 패키지와 동일한 콘텐츠가 들어 있는지 확인하십시오.

   `.py` 파일 및 종속성이 디렉터리의 루트에 있는지 확인합니다.

추론 모델이 성공적으로 배포되었는지 확인

1. Lambda 런타임 프로세스의 프로세스 식별 번호(PID)를 찾습니다.

   ```
   ps aux | grep 'lambda-function-name*'
   ```

   출력의 Lambda 런타임 프로세스에 대한 행의 두 번째 열에 PID가 표시됩니다.

2. Lambda 런타임 네임스페이스를 입력합니다. 명령을 실행하기 전에 자리 표시자 *pid* 값을 바꿔야 합니다.

   > **Note**
   >
   > 이 디렉터리 및 내용은 Lambda 런타임 네임스페이스입니다. 따라서 정규 Linux 네임스페이스에서는 보이지 않습니다.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. ML 리소스에 대해 지정한 로컬 디렉터리의 내용을 나열합니다.

   ```
   cd /greengrass-machine-learning/mxnet/squeezenet/
   ls -ls
   ```

   다음 파일이 표시됩니다.

   ```
   32 -rw-r--r-- 1 ggc_user ggc_group   31675 Nov 18 15:19 synset.txt
   32 -rw-r--r-- 1 ggc_user ggc_group   28707 Nov 18 15:19 squeezenet_v1.1-symbol.json
   4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19
    squeezenet_v1.1-0000.params
   ```

다음 단계

지금부터는 다른 추론 앱을 살펴봅니다. AWS IoT Greengrass은(는) 로컬 추론을 시도할 때 사용할 수 있는 기타 Lambda 함수를 제공합니다. the section called “MXNet 프레임워크 설치” 단계에서 다운로드한 프리컴파일 라이브러리 폴더에서 예제 패키지를 찾을 수 있습니다.

Intel Atom 구성

Intel Atom 디바이스에서 이 자습서를 실행하려면 소스 이미지를 제공하고 Lambda 함수를 구성한 후 다른 로컬 디바이스 리소스를 추가합니다. 추론에 GPU를 사용하려면 다음 소프트웨어가 디바이스에 설치되어 있어야 합니다.

- OpenCL 버전 1.0 이상
- Python 3.7 및 pip

> **Note**
>
> 디바이스가 Python 3.6으로 미리 빌드된 경우 Python 3.7에 대한 심볼 링크를 대신 만들 수 있습니다. 자세한 내용은 Step 2 섹션을 참조하세요.

- NumPy
- OpenCV on Wheels

1. 이미지 분류에 사용하기 위해 Lambda 함수에 대한 정적 PNG 또는 JPG 이미지를 다운로드합니다. 이 예제는 작은 이미지 파일에 가장 효과적입니다.

   `greengrassObjectClassification.py` 파일이 들어 있는 디렉터리(또는 이 디렉터리의 하위 디렉터리)에 이미지 파일을 저장합니다. 이 파일은 the section called “Lambda 함수 생성 및 게시”에서 업로드하는 Lambda 함수 배포 패키지에 있습니다.

   > **Note**
   >
   > AWS DeepLens를 사용하는 경우, 온보드 카메라를 사용하거나 본인의 카메라를 장착해 정적 이미지가 아닌 캡처된 이미지에 대해 추론을 수행할 수 있습니다. 그러나 정적 이미지부터 시작하는 것이 좋습니다.

카메라를 사용하는 경우 awscam APT 패키지가 설치되어 있고 최신 상태여야 합니다. 자세한 내용은 AWS DeepLens 개발자 안내서의 AWS DeepLens 코어 디바이스 설정 섹션을 참조하세요.

2. Python 3.7을 사용하고 있다면 Python 3.x에서 Python 3.7로 심볼 링크를 만들어야 합니다. 이렇게 하면 디바이스가 AWS IoT Greengrass와 함께 Python 3를 사용하도록 구성됩니다. 다음 명령을 실행하여 Python 설치를 찾습니다.

```
which python3
```

다음 명령을 실행하여 심볼 링크(symlink)를 생성합니다.

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

디바이스를 재부팅합니다.

3. Lambda 함수 구성을 편집합니다. the section called “그룹에 Lambda 함수를 추가합니다.”의 프로시저를 따르세요.

> **Note**
>
> 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

a. 컨테이너화 없이 실행하려면:

- 시스템 사용자 및 그룹의 경우 **Another user ID/group ID**을(를) 선택합니다. 시스템 사용자 ID에 **0**을(를) 입력합니다. 시스템 그룹 ID에 **0**을(를) 입력합니다.

  이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

> **Tip**
>
> 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.
- 제한 시간 값을 5초로 업데이트합니다. 이렇게 하면 요청이 제한 시간에 너무 빨리 도달하는 경우를 피할 수 있습니다. 설정 후 추론이 실행되려면 몇 분이 걸립니다.
- 고정에서 True를 선택합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.
- Lambda 수명 주기에 대해 Make this function long-lived and keep it running indefinitely를 선택합니다.

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 제한 시간 값을 5초로 업데이트합니다. 이렇게 하면 요청이 제한 시간에 너무 빨리 도달하는 경우를 피할 수 있습니다. 설정 후 추론이 실행되려면 몇 분이 걸립니다.
- 고정된 경우 True를 선택합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

4. 컨테이너화 모드에서 실행 중인 경우 필요한 로컬 디바이스 리소스를 추가하여 디바이스 GPU에 대한 액세스 권한을 부여하십시오.

> **Note**
>
> 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 디바이스 리소스를 구성하지 않고도 디바이스 GPU에 액세스할 수 있습니다.

a. 그룹 구성 페이지에서 리소스를 선택합니다.

b. 로컬 리소스 추가를 선택합니다.

c. 리소스를 정의합니다.

- 리소스 이름에 **renderD128**을 입력합니다.
- 리소스 유형에서 로컬 디바이스를 선택합니다.
- 디바이스 경로에 **/dev/dri/renderD128**을 입력합니다.
- [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.
- Lambda 함수 제휴의 경우 Lambda 함수에 대한 읽기 및 쓰기 액세스 권한을 부여하십시오.

NVIDIA Jetson TX2 구성

NVIDIA Jetson TX2에서 이 자습서를 실행하려면 소스 이미지를 제공하고 Lambda 함수를 구성합니다. GPU를 사용하는 경우 로컬 디바이스 리소스도 추가해야 합니다.

1. AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있도록 Jetson 디바이스가 구성되어 있는지 확인합니다. 디바이스 구성에 대한 자세한 내용은 the section called “다른 디바이스 설정” 단원을 참조하십시오.
2. MXNet 설명서의 Jetson에 MXNet 설치에 나온 지침을 따라 Jetson 디바이스에 MXNet을 설치합니다.

> **Note**
>
> 소스에서 MXNet을 빌드하려면 지침을 따라 공유 라이브러리를 빌드합니다. Jetson TX2 디바이스에서 작동하도록 `config.mk` 파일에서 다음 설정을 편집합니다.
>
> - `CUDA_ARCH` 설정에 `-gencode arch=compute-62, code=sm_62`를 추가합니다.
> - CUDA를 활성화합니다.
>
> ```
> USE_CUDA = 1
> ```

3. 이미지 분류에 사용하기 위해 Lambda 함수에 대한 정적 PNG 또는 JPG 이미지를 다운로드합니다. 이 앱은 작은 이미지 파일에 가장 적합합니다. 또는 Jetson 보드의 카메라를 구성하여 소스 이미지를 캡처할 수 있습니다.

   `greengrassObjectClassification.py` 파일이 들어 있는 디렉터리에 이미지 파일을 저장합니다. 이 디렉터리의 하위 디렉터리에 저장할 수도 있습니다. 이 디렉터리는 the section called "Lambda 함수 생성 및 게시"에서 업로드하는 Lambda 함수 배포 패키지에 있습니다.

4. AWS IoT Greengrass와 함께 Python 3을 사용하려면 Python 3.7에서 Python 3.6으로 심볼 링크를 생성합니다. 다음 명령을 실행하여 Python 설치를 찾습니다.

   ```
   which python3
   ```

   다음 명령을 실행하여 심볼 링크(symlink)를 생성합니다.

   ```
   sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
   ```

   디바이스를 재부팅합니다.

5. `ggc_user` 시스템 계정에서 MXNet 프레임워크를 사용할 수 있는지 확인합니다.

   ```
   "sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
   ```

6. Lambda 함수 구성을 편집합니다. the section called "그룹에 Lambda 함수를 추가합니다."의 프로시저를 따르세요.

   > **Note**
   >
   > 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

   a. 컨테이너화 없이 실행하려면:

      - 시스템 사용자 및 그룹의 경우 **Another user ID/group ID**을(를) 선택합니다. 시스템 사용자 ID에 **0**을(를) 입력합니다. 시스템 그룹 ID에 **0**을(를) 입력합니다.

이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

> **Tip**
>
> 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.
- 환경 변수에서 다음 키-값 페어를 Lambda 함수에 추가합니다. 이렇게 하면 MXNet 프레임워크를 사용하도록 AWS IoT Greengrass가 구성됩니다.

키	값
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 메모리 제한 값을 늘립니다. CPU의 경우 500MB를, GPU의 경우 최소 2,000MB를 사용합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.
- 환경 변수에서 다음 키-값 페어를 Lambda 함수에 추가합니다. 이렇게 하면 MXNet 프레임워크를 사용하도록 AWS IoT Greengrass가 구성됩니다.

키	값
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

7. 컨테이너화 모드에서 실행 중인 경우 다음의 로컬 디바이스 리소스를 추가하여 디바이스 GPU에 대한 액세스 권한을 부여하십시오. the section called “그룹에 리소스 추가”의 프로시저를 따르세요.

> **Note**
>
> 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 디바이스 리소스를 구성하지 않고도 디바이스 GPU에 액세스할 수 있습니다.

각 리소스의 경우:

- [Resource type]에서 [Device]를 선택합니다.
- [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.

이름	디바이스 경로
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. 컨테이너화 모드에서 실행 중인 경우 다음 로컬 볼륨 리소스를 추가하여 디바이스 카메라에 대한 액세스 권한을 부여하세요. the section called “그룹에 리소스 추가”의 프로시저를 따르세요.

> **Note**
>
> 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 볼륨 리소스를 구성하지 않고도 디바이스 카메라에 액세스할 수 있습니다.

- [리소스 유형(Resource type)]에서 [볼륨(Volume)]을 선택합니다.
- [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.

이름	소스 경로	대상 경로
shm	/dev/shm	/dev/shm
/tmp	/tmp	/tmp

AWS Management Console을 사용하여 최적화된 기계 학습 추론을 구성하는 방법

이 자습서의 단계를 따르려면 AWS IoT Greengrass Core v1.10 이상을 사용해야 합니다.

SageMaker Neo 딥러닝 컴파일러를 사용해 Tensorflow, Apache MXNet, PyTorch, ONNX 및 XGBoost 프레임워크에서 기본 기계 학습 추론 모델의 예측 효율성을 최적화함으로써 설치 공간을 줄이고 성능을 높일 수 있습니다. 그런 다음 더 빠른 추론을 위해 최적화된 모델을 다운로드하고 SageMaker Neo 딥러닝 컴파일러을 설치해 AWS IoT Greengrass 디바이스에 배포할 수 있습니다.

이 자습서에서는 AWS Management Console을 사용하여 Greengrass 그룹을 구성한 다음 데이터를 클라우드로 전송하지 않고 로컬 방식으로 카메라에서 이미지를 인식하는 Lambda 추론 예제를 실행하는 방법을 설명합니다. 이 추론 예제는 Raspberry Pi에서 카메라 모듈에 액세스합니다. 이 자습서에서는 Resnet-50에서 교육하고 Neo 딥러닝 컴파일러에서 최적화되어 있는 사전 패키징된 모델을 다운로드합니다. 그런 다음 이 모델을 사용하여 AWS IoT Greengrass 디바이스에서 로컬 이미지 분류를 수행합니다.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Raspberry Pi 구성
2. Neo 딥 러닝 런타임 설치
3. 추론 Lambda 함수를 생성합니다.
4. 그룹에 Lambda 함수를 추가합니다.
5. Neo 최적화 모델 리소스를 그룹에 추가
6. 그룹에 카메라 디바이스 리소스 추가
7. 그룹에 구독 추가
8. 그룹 배포
9. 예제 테스트

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- Raspberry Pi 4 모델 B 또는 Raspberry Pi 3 모델 B/B+는 AWS IoT Greengrass와(과) 함께 사용하도록 설정 및 구성되었습니다. AWS IoT Greengrass로 Raspberry Pi를 설정하기 위해, Greengrass 디

바이스 설정 스크립트를 실행하거나 모듈 1 및 모듈 2를 완료했는지 시작하기 AWS IoT Greengrass 확인합니다.

> **Note**
>
> Raspberry Pi는 일반적으로 이미지 분류에 사용되는 딥 러닝 프레임워크를 실행하기 위해 2.5A 전원 공급 장치가 필요할 수 있습니다. 정격이 더 낮은 전원 공급 장치는 디바이스가 재부팅될 수 있습니다.

- Raspberry Pi Camera Module V2 – 8메가픽셀, 1080p. 카메라 설정 방법을 알아보려면 Raspberry Pi 설명서의 Connecting the camera를 참조하십시오.
- Greengrass 그룹 및 Greengrass 코어. Greengrass 그룹 또는 코어를 생성하는 방법에 대해 알아보려면 시작하기 AWS IoT Greengrass 단원을 참조하십시오.

> **Note**
>
> 이 자습서는 Raspberry Pi를 사용하지만 AWS IoT Greengrass는 Intel Atom과 NVIDIA Jetson TX2 등의 다른 플랫폼도 지원합니다. Intel Atom 예제를 사용하는 경우 Python 3.7 대신 Python 3.6을 설치해야 할 수도 있습니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있도록 디바이스를 구성하는 방법에 대한 자세한 내용은 the section called “다른 디바이스 설정” 단원을 참조하십시오.
>
> AWS IoT Greengrass이(가) 지원하지 않는 타사 플랫폼의 경우, Lambda 함수를 비컨테이너식 모드에서 실행해야 합니다. 비컨테이너식 모드에서 실행하려면 Lambda 함수를 루트로 실행해야 합니다. 자세한 정보는 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 및 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

1단계: Raspberry Pi 구성

이 단계에서는 Raspbian 운영 체제에 업데이트를 설치하고, 카메라 모듈 소프트웨어와 Python 종속성을 설치하며, 카메라 인터페이스를 활성화합니다.

Raspberry Pi 터미널에서 다음 명령을 실행합니다.

1. Raspbian에 업데이트를 설치합니다.

```
sudo apt-get update
```

```
sudo apt-get dist-upgrade
```

2. 이 자습서에 필요한 카메라 모듈에 대한 picamera 인터페이스 및 기타 Python 라이브러리를 설치합니다.

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   설치를 검사합니다.

   - Python 3.7 설치에 pip가 포함되어 있는지 확인합니다.

     ```
     python3 -m pip
     ```

     pip가 설치되어 있지 않으면 pip 웹 사이트에서 다운로드한 후 다음 명령을 실행합니다.

     ```
     python3 get-pip.py
     ```

   - Python 버전이 3.7 이상인지 확인합니다.

     ```
     python3 --version
     ```

     출력에 이전 버전이 나열되면 다음 명령을 실행합니다.

     ```
     sudo apt-get install -y python3.7-dev
     ```

   - Setuptools 및 Picamera가 성공적으로 설치되었는지 확인합니다.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
     sudo -u ggc_user bash -c 'python3 -c "import picamera"'
     ```

     출력에 오류가 포함되어 있지 않으면 검사가 성공한 것입니다.

   > **Note**
   >
   > 디바이스에 설치된 Python 실행 파일이 python3.7인 경우 이 자습서의 명령에 python3 대신 python3.7을 사용합니다. 종속성 오류를 피하기 위해 pip 설치가 올바른 python3.7 또는 python3 버전에 매핑되는지 확인하십시오.

3. Raspberry Pi를 재부팅합니다.

```
sudo reboot
```

4. Raspberry Pi 구성 도구를 엽니다.

```
sudo raspi-config
```

5. 화살표 키를 사용하여 [Interfacing Options]를 열고 카메라 인터페이스를 활성화합니다. 메시지가 나타나면 디바이스 재부팅을 허용합니다.
6. 다음 명령을 사용하여 카메라 설정을 테스트합니다.

```
raspistill -v -o test.jpg
```

그러면 Raspberry Pi의 미리 보기 창이 열리고, 현재 디렉터리에 `test.jpg`라는 이름의 사진이 저장되며, 카메라에 대한 정보가 Raspberry Pi 터미널에 표시됩니다.

2단계: Amazon SageMaker Neo 딥 러닝 런타임 설치

이 단계에서는 Raspberry Pi에 (DLR)을 설치합니다.

> **Note**
>
> 이 자습서에서는 버전 1.1.0을 설치하는 것이 좋습니다.

1. Raspberry Pi에 원격으로 로그인합니다.

```
ssh pi@your-device-ip-address
```

2. DLR 설명서의 DLR 설치에서 Raspberry Pi 디바이스의 휠 URL을 찾습니다. 그런 다음 지침에 따라 디바이스에 DLR을 설치합니다. 예를 들어 pip를 사용할 수 있습니다.

```
pip3 install rasp3b-wheel-url
```

3. DLR을 설치한 후 다음 구성을 검사합니다.
   - `ggc_user` 시스템 계정에서 DLR 라이브러리를 사용할 수 있는지 확인합니다.

```
sudo -u ggc_user bash -c 'python3 -c "import dlr"'
```

- NumPy가 설치되어 있는지 확인합니다.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

3단계: 추론 Lambda 함수 생성

이 단계에서는 Lambda 함수 배포 패키지 및 Lambda 함수를 생성합니다. 그런 다음 함수 버전을 게시하고 별칭을 생성합니다.

1. the section called “기계 학습 샘플”에 나온 Raspberry Pi용 DLR 샘플을 컴퓨터에 다운로드합니다.
2. 다운로드한 `dlr-py3-armv7l.tar.gz` 파일의 압축을 풉니다.

   ```
   cd path-to-downloaded-sample
   tar -xvzf dlr-py3-armv7l.tar.gz
   ```

   추출된 샘플 패키지의 `examples` 디렉터리에는 함수 코드와 종속성이 포함되어 있습니다.

   - `inference.py`는 이 자습서에서 사용되는 추론 코드입니다. 이 코드를 템플릿으로 사용하여 사용자 고유의 추론 함수를 생성할 수 있습니다.
   - `greengrasssdk`은(는) Python용 AWS IoT Greengrass 코어 SDK 버전 1.5.0입니다.

   > **Note**
   >
   > 새 버전을 사용할 수 있는 경우 해당 버전을 다운로드하여 배포 패키지의 SDK 버전을 업그레이드할 수 있습니다. 자세한 내용은 GitHub의 Python용 AWS IoT Greengrass 코어 SDK를 참조하세요.

3. `examples` 디렉터리의 내용을 `optimizedImageClassification.zip`라는 파일로 압축합니다. 이것은 배포 패키지입니다.

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/examples
   zip -r optimizedImageClassification.zip .
   ```

배포 패키지에는 함수 코드와 종속성이 포함되어 있습니다. 여기에는 Neo 딥 러닝 컴파일러 모델로 추론을 수행하기 위해 Neo 딥 러닝 런타임 Python API를 간접적으로 호출하는 코드가 포함됩니다.

> **Note**
>
> `.py` 파일 및 종속성이 디렉터리의 루트에 있는지 확인합니다.

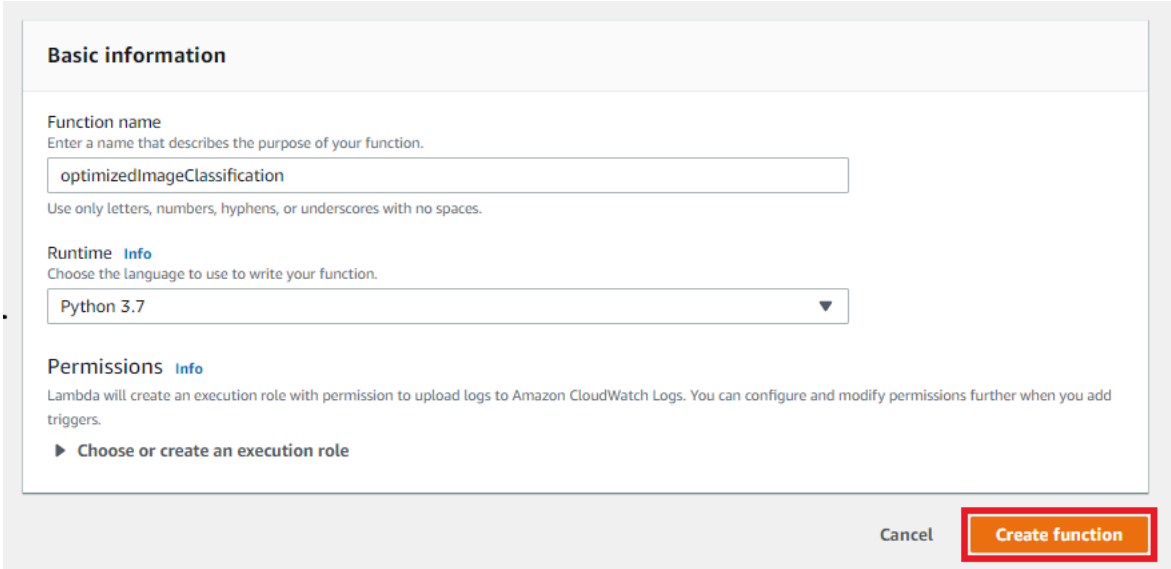
4. 이제 Greengrass 그룹에 Lambda 함수를 추가합니다.

   Lambda 콘솔 페이지의 함수를 선택하고 함수 생성을 선택합니다.

5. 새로 작성을 선택하고 다음 값을 사용하여 함수를 생성합니다.

   - [함수 이름(Function name)]에 **`optimizedImageClassification`**을 입력합니다.
   - 실행 시간에서 Python 3.7을 선택합니다.

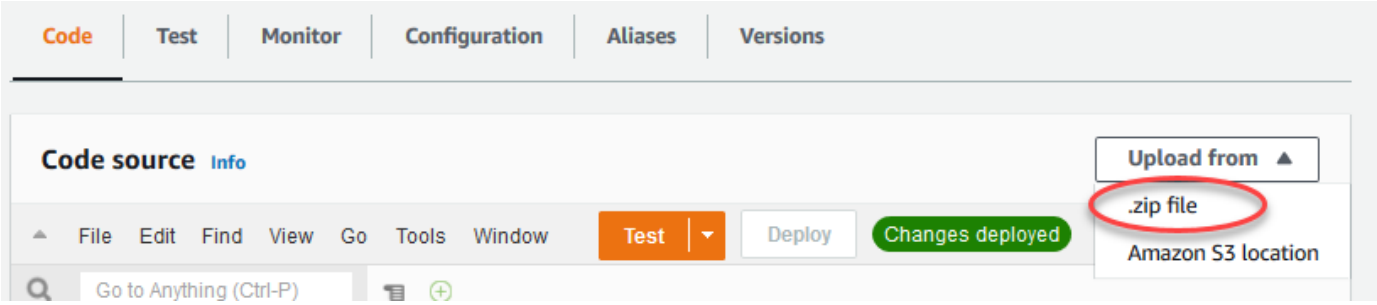
   권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.



6. 함수 생성(Create function)을 선택합니다.

이제 Lambda 함수 배포 패키지를 업로드하고 핸들러를 등록합니다.

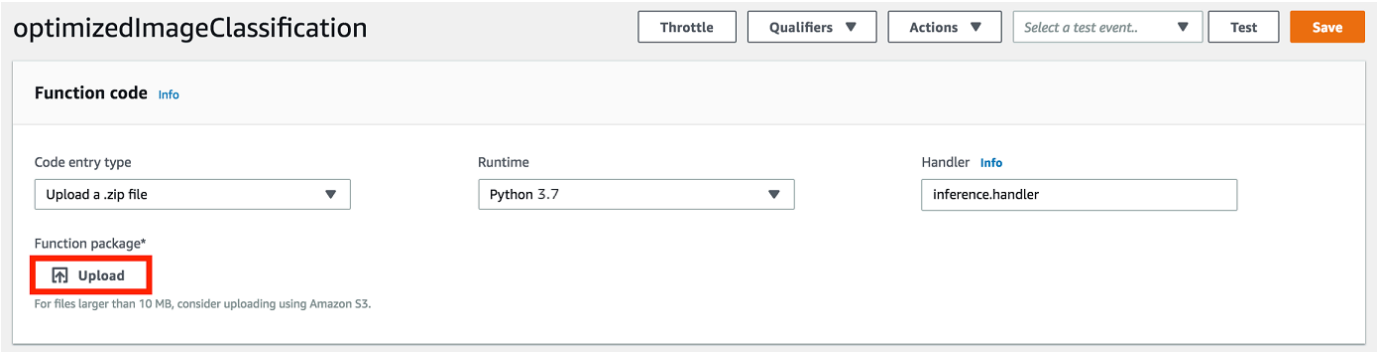
1. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



2. `optimizedImageClassification.zip` 배포 패키지를 선택한 다음 저장을 선택합니다.
3. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

   - 실행 시간에서 Python 3.7을 선택합니다.
   - 핸들러에 **`inference.handler`**를 입력합니다.

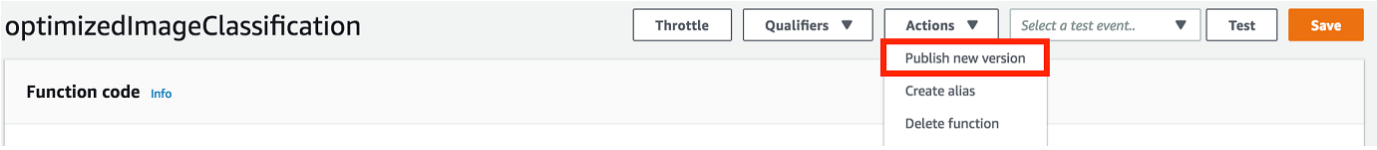
   Save를 선택합니다.



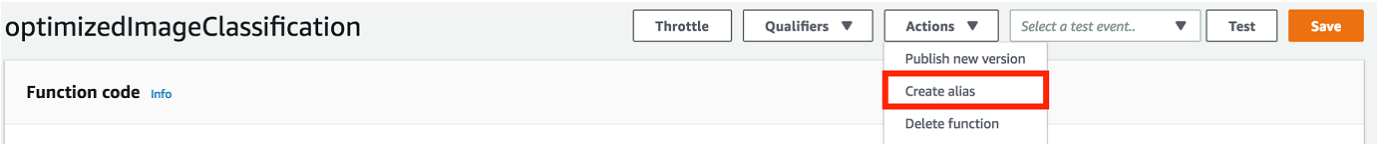
그런 다음 Lambda 함수의 첫 번째 버전을 게시합니다. 그런 다음 버전의 별칭을 생성합니다.

> **Note**
>
> Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

1. [Actions] 메뉴에서 [Publish new revision]을 선택합니다.



2. 버전 설명에 **`First version`**을 입력한 후 게시를 선택합니다.
3. optimizedImageClassification: 1 구성 페이지의 작업 메뉴에서 별칭 생성을 선택합니다.



4. [Create a new alias] 페이지에서 다음 값을 사용합니다.
   - 이름(Name)에 **`mlTestOpt`**을 입력합니다.
   - 버전에 **`1`**를 입력합니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

5. 생성을 선택합니다.

   이제 Greengrass 그룹에 Lambda 함수를 추가합니다.

4단계: Greengrass 그룹에 Lambda 함수 추가

이 단계에서 그룹에 Lambda 함수를 추가한 다음 수명 주기를 구성합니다.

먼저 Greengrass 그룹에 Lambda 함수를 추가합니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 그룹 구성 페이지에서 Lambda 함수 탭을 선택한 다음 Lambda 추가를 선택합니다.
3. Lambdas 함수를 선택하고 optimizedImageClassification를 선택합니다.
4. Lambda 함수 버전에서 게시한 버전에 대한 별칭을 선택합니다.

그런 다음 Lambda 함수의 수명 주기를 구성합니다.

1. Lambda 함수 구성 섹션에서 다음과 같이 업데이트하십시오.

> **Note**
>
> 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

a. 컨테이너화 없이 실행하려면:

- 시스템 사용자 및 그룹의 경우 **Another user ID/group ID**을(를) 선택합니다. 시스템 사용자 ID에 **0**을(를) 입력합니다. 시스템 그룹 ID에 **0**을(를) 입력합니다.

  이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

  > **Tip**
  >
  > 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.

- 시간 제한에 **10 seconds**를 입력합니다.
- 고정된 경우 True를 선택합니다.

  자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.

- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 시스템 사용자 및 그룹의 경우 그룹 기본값 사용을 선택합니다.
- Lambda 함수 컨테이너화의 경우 그룹 기본값 사용을 선택합니다.
- 메모리 제한에 **`1024 MB`**를 입력합니다.
- 시간 제한에 **`10 seconds`**를 입력합니다.
- 고정된 경우 True를 선택합니다.

  자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

2. Lambda 함수 추가를 선택합니다.

5단계: SageMaker Neo 최적화 모델 리소스를 Greengrass 그룹에 추가

이 단계에서 최적화된 ML 추론 모델에 대한 리소스를 생성해 Amazon S3 버킷에 업로드합니다. 그런 다음 AWS IoT Greengrass 콘솔에서 Amazon S3 업로드 모델을 찾고 새로 생성된 리소스를 Lambda 함수와 연결합니다. 이렇게 하면 함수가 코어 디바이스의 리소스에 액세스할 수 있습니다.

1. 컴퓨터에서 the section called “추론 Lambda 함수를 생성합니다.”에서 압축을 푼 샘플 패키지의 `resnet50` 디렉터리로 이동합니다.

   > **Note**
   >
   > NVIDIA Jetson 예제를 사용하는 경우 이 대신 샘플 패키지의 `resnet18` 디렉터리를 사용해야 합니다. 자세한 내용은 the section called “NVIDIA Jetson TX2 구성” 섹션을 참조하세요.

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
   ```

이 디렉터리에는 Resnet-50에서 교육된 이미지 분류 모델을 위해 미리 컴파일된 모델 아티팩트가 들어 있습니다.

2. `resnet50` 디렉터리 내의 파일을 `resnet50.zip`이라는 파일로 압축합니다.

```
zip -r resnet50.zip .
```

3. AWS IoT Greengrass 그룹에 대한 그룹 구성 페이지에서 리소스 탭을 선택합니다. Machine Learning 섹션으로 이동하고 Machine Learning 리소스 추가를 선택합니다. 머신 러닝 리소스 생성 페이지에서 리소스 이름에 **`resnet50_model`**을 입력합니다.
4. 모델 소스의 경우 S3에 저장된 모델(예: 딥 러닝 컴파일러를 통해 최적화된 모델)을 선택합니다.
5. S3 URI에서 S3 찾아보기를 선택합니다.

> **Note**
>
> 현재, 최적화된 SageMaker 모델은 자동으로 Amazon S3에 저장됩니다. 이 옵션을 사용해 Amazon S3 버킷에서 최적화된 모델을 찾을 수 있습니다. SageMaker의 모델 최적화에 대한 자세한 내용은 SageMaker Neo 설명서를 참조하십시오.

6. 모델 업로드를 선택합니다.
7. Amazon S3 콘솔 탭에서 zip 파일을 Amazon S3 버킷으로 업로드합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 S3 버킷에 파일 및 폴더를 업로드하려면 어떻게 해야 합니까?를 참조하십시오.

> **Note**
>
> 버킷 이름이 문자열 **`greengrass`**를 포함해야 합니다. 고유한 이름을 선택합니다(예: **`greengrass-dlr-bucket-`*`user-id`*`-`*`epoch-time`***). 버킷 이름에는 마침표(`.`)를 사용하지 마십시오.

8. AWS IoT Greengrass 콘솔 탭에서 Amazon S3 버킷을 찾아 선택합니다. 업로드한 `resnet50.zip` 파일을 찾아 선택을 선택합니다. 사용 가능한 버킷과 파일의 목록을 업데이트하려면 페이지를 새로 고쳐야 할 수 있습니다.
9. Destination path에 **`/ml_model`**을 입력합니다.

Lambda 런타임 네임스페이스에 있는 로컬 모델의 대상입니다. 그룹을 배포할 때 AWS IoT Greengrass은(는) 소스 모델 패키지를 검색하고 지정된 디렉터리에 내용을 압축 해제합니다.

> **Note**
>
> 로컬 경로에 제공된 정확한 경로를 사용하는 것이 좋습니다. 이 단계에서 다른 로컬 모델 대상 경로를 사용하면 이 자습서에서 제공하는 일부 문제 해결 명령이 부정확하게 됩니다. 다른 경로를 사용하는 경우 여기서 입력하는 정확한 경로를 사용하는 `MODEL_PATH` 환경 변수를 설정해야 합니다. 환경 변수에 대한 자세한 내용은 AWS Lambda 환경 변수를 참조하십시오.

10. 컨테이너화 모드에서 실행 중인 경우:

    a. 시스템 그룹 소유자 및 파일 액세스 권한에서 시스템 그룹 및 권한 지정을 선택합니다.

    b. 작업을 선택한 다음 리소스 추가를 선택합니다.

6단계: Greengrass 그룹에 카메라 디바이스 리소스 추가

이 단계에서는 카메라 모듈에 대한 리소스를 생성하고 이 리소스를 Lambda 함수와 연결하여 Lambda 함수가 코어 디바이스의 리소스에 액세스할 수 있도록 합니다.

> **Note**
>
> 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 이 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다.

1. 그룹 구성 페이지에서 리소스를 선택합니다.
2. 로컬 리소스 탭에서 로컬 리소스 추가를 선택합니다.
3. 로컬 리소스 생성 페이지에서 다음 값을 사용합니다.

    - 리소스 이름에 **`videoCoreSharedMemory`**을 입력합니다.

- [Resource type]에서 [Device]를 선택합니다.
- 디바이스 경로에 **/dev/vcsm**를 입력합니다.

  디바이스 경로는 디바이스 리소스의 로컬 절대 경로입니다. 이 경로는 /dev 아래에 있는 문자 디바이스 또는 블록 디바이스만 참조할 수 있습니다.
- System group owner and file access permissions에서 Automatically add file system permissions of the system group that owns the resource를 선택합니다.

  [Group owner file access permission] 옵션을 사용하면 Lambda 프로세스에 추가 파일 액세스 권한을 부여할 수 있습니다. 자세한 내용은 그룹 소유자 파일 액세스 권한 섹션을 참조하세요.

4. 페이지 하단에서 다른 리소스 추가를 선택합니다.
5. 리소스 탭에서 추가를 선택하여 다른 로컬 리소스를 생성하고 다음 값을 사용하십시오.
   - 리소스 이름에 **videoCoreInterface**을 입력합니다.
   - [Resource type]에서 [Device]를 선택합니다.
   - 디바이스 경로에 **/dev/vchiq**를 입력합니다.
   - [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.
6. 리소스 추가를 선택합니다.

7단계: Greengrass 그룹에 구독 추가

이 단계에서는 그룹에 구독을 추가합니다. 이러한 구독을 통해 Lambda 함수는 MQTT 주제에 게시하여 예측 결과를 AWS IoT에 전송할 수 있습니다.

1. 그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.
2. 구독 생성 페이지에서 다음과 같이 원본과 대상을 구성합니다.
   a. 소스 유형에서 Lambda 함수를 선택한 후 optimizedImageClassification을 선택합니다.
   b. 대상 선택에서 서비스를 선택한 후 IoT Cloud를 선택합니다.
   c. 주제 필터에서 **/resnet-50/predictions**을(를) 입력한 다음 구독 생성을 선택합니다.
3. 두 번째 구독을 추가합니다. 구독 탭을 선택하고 구독 추가를 선택한 다음 다음과 같이 원본과 대상을 구성합니다.
   a. 소스 유형에서 서비스를 선택한 후 IoT Cloud를 선택합니다.

b. 대상 선택에서 Lambda 함수를 선택한 후 optimizedImageClassification을 선택합니다.

c. 주제 필터에서 **/resnet-50/test**을(를) 입력한 다음 구독 생성을 선택합니다.

8단계: Greengrass 그룹 배포

이 단계에서는 그룹 정의의 현재 버전을 Greengrass 코어 디바이스로 배포합니다. 정의에는 추가한 Lambda 함수, 리소스 및 구독 구성이 포함됩니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 데몬이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 root에 대한 /greengrass/ggc/packages/*latest-core-version*/bin/daemon 입력이 포함되어 있는 경우에는 데몬이 실행 중인 것입니다.

   b. 데몬을 시작하려면:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. 그룹 구성 페이지에서 배포를 선택합니다.
3. Lambda 함수 탭에서 IP 감지기를 선택하고 편집을 선택합니다.
4. IP 감지기 설정 편집 대화 상자에서 MQTT 브로커 엔드포인트 자동 감지 및 재정의를 선택하고 저장을 선택합니다.

   이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

   > **Note**
   >
   > 메시지가 표시되면 Greengrass 서비스 역할을 생성할 권한을 부여하고 이 권한을 현재 AWS 리전의 AWS 계정과 연결합니다. 이 역할은 AWS IoT Greengrass가 AWS 서비스의 리소스에 액세스할 수 있습니다.

배포 페이지에 배포 타임스탬프, 버전 ID, 상태가 표시됩니다. 완료되면 배포에 대해 성공적으로 완료했습니다 상태가 표시되어야 합니다.

배포에 대한 자세한 내용은 AWS IoT Greengrass 그룹 배포 섹션을 참조하세요. 문제 해결에 대한 도움말은 문제 해결 단원을 참조하십시오.

추론 예제 테스트

이제 배포가 올바르게 구성되었는지 확인할 수 있습니다. 테스트하려면 `/resnet-50/predictions` 주제를 구독하고 `/resnet-50/test` 주제에 메시지를 게시합니다. 그러면 Raspberry Pi로 사진을 찍고 캡처한 이미지에 대해 추론을 수행하는 Lambda 함수가 트리거됩니다.

> **Note**
>
> NVIDIA Jetson 예제를 사용하는 경우 이 대신 `resnet-18/predictions` 및 `resnet-18/test` 주제를 사용해야 합니다.

> **Note**
>
> Raspberry Pi에 모니터가 연결된 경우 라이브 카메라 피드가 미리 보기 창에 표시됩니다.

1. AWS IoT 콘솔 홈 페이지의 테스트에서 MQTT 테스트 클라이언트를 선택합니다.
2. 구독의 경우 주제 구독을 선택합니다. 다음 값을 사용합니다. 나머지 옵션을 기본값으로 둡니다.
   - 구독 주제에 **`/resnet-50/predictions`**를 입력합니다.
   - MQTT 페이로드 표시의 경우 추가 구성에서 페이로드를 문자열로 표시를 선택합니다.
3. 구독을 선택합니다.
4. 주제에 게시 탭의 **`/resnet-50/test`**주제 이름에 을 입력한 다음 게시를 선택합니다.
5. 테스트가 성공적인 경우 메시지를 게시하면 Raspberry Pi 카메라가 이미지를 캡처합니다. 페이지 하단에 Lambda 함수의 메시지가 나타납니다. 이 메시지에는 예측된 클래스 이름, 확률, 피크 메모리 사용량의 형식을 사용하여 이미지의 예측 결과가 포함됩니다.

Intel Atom 구성

Intel Atom 디바이스에서 이 자습서를 실행하려면 소스 이미지를 제공하고 Lambda 함수를 구성한 후 다른 로컬 디바이스 리소스를 추가합니다. 추론에 GPU를 사용하려면 다음 소프트웨어가 디바이스에 설치되어 있어야 합니다.

- OpenCL 버전 1.0 이상
- Python 3.7 및 pip
- NumPy
- OpenCV on Wheels

1. 이미지 분류에 사용하기 위해 Lambda 함수에 대한 정적 PNG 또는 JPG 이미지를 다운로드합니다. 이 예제는 작은 이미지 파일에 가장 효과적입니다.

   `inference.py` 파일이 들어 있는 디렉터리(또는 이 디렉터리의 하위 디렉터리)에 이미지 파일을 저장합니다. 이 파일은 the section called “추론 Lambda 함수를 생성합니다.”에서 업로드하는 Lambda 함수 배포 패키지에 있습니다.

   > **Note**
   >
   > AWS DeepLens를 사용하는 경우, 온보드 카메라를 사용하거나 본인의 카메라를 장착해 정적 이미지가 아닌 캡처된 이미지에 대해 추론을 수행할 수 있습니다. 그러나 정적 이미지부터 시작하는 것이 좋습니다.
   > 카메라를 사용하는 경우 `awscam` APT 패키지가 설치되어 있고 최신 상태여야 합니다. 자세한 내용은 AWS DeepLens 개발자 안내서의 AWS DeepLens 코어 디바이스 설정 섹션을 참조하세요.

2. Lambda 함수 구성을 편집합니다. the section called “그룹에 Lambda 함수를 추가합니다.”의 프로시저를 따르세요.

   > **Note**
   >
   > 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

a. 컨테이너화 없이 실행하려면:

- 시스템 사용자 및 그룹의 경우 **Another user ID/group ID**을(를) 선택합니다. 시스템 사용자 ID에 **0**을(를) 입력합니다. 시스템 그룹 ID에 **0**을(를) 입력합니다.

  이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

  > **Tip**
  >
  > 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.

- 제한 시간 값을 2분으로 늘립니다. 이렇게 하면 요청이 제한 시간에 너무 빨리 도달하는 경우를 피할 수 있습니다. 설정 후 추론이 실행되려면 몇 분이 걸립니다.
- 고정된 경우 True를 선택합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 메모리 제한 값을 3,000MB로 늘립니다.
- 제한 시간 값을 2분으로 늘립니다. 이렇게 하면 요청이 제한 시간에 너무 빨리 도달하는 경우를 피할 수 있습니다. 설정 후 추론이 실행되려면 몇 분이 걸립니다.
- 고정된 경우 True를 선택합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

3. Neo 최적화 모델 리소스를 그룹에 추가합니다. the section called “추론 Lambda 함수를 생성합니다.”에서 압축을 푼 샘플 패키지의 `resnet50` 디렉터리에 있는 모델 리소스를 업로드합니다. 이 디렉터리에는 Resnet-50에서 교육된 이미지 분류 모델을 위해 미리 컴파일된 모델 아티팩트가 들어 있습니다. 다음 업데이트 사항과 함께 the section called “Neo 최적화 모델 리소스를 그룹에 추가”의 절차를 따릅니다.

   - `resnet50` 디렉터리 내의 파일을 `resnet50.zip`이라는 파일로 압축합니다.
   - 머신 러닝 리소스 생성 페이지에서 리소스 이름에 **`resnet50_model`**을 입력합니다.
   - `resnet50.zip` 파일을 업로드합니다.

4. 컨테이너화 모드에서 실행 중인 경우 필요한 로컬 디바이스 리소스를 추가하여 디바이스 GPU에 대한 액세스 권한을 부여하십시오.

   > **Note**
   >
   > 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 디바이스 리소스를 구성하지 않고도 디바이스 GPU에 액세스할 수 있습니다.

   a. 그룹 구성 페이지에서 리소스를 선택합니다.
   b. 로컬 리소스 탭에서 로컬 리소스 추가를 선택합니다.
   c. 리소스를 정의합니다.

      - 리소스 이름에 **`renderD128`**을 입력합니다.
      - [Resource type]에서 [Device]를 선택합니다.
      - 디바이스 경로에 **`/dev/dri/renderD128`**를 입력합니다.
      - [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.

NVIDIA Jetson TX2 구성

NVIDIA Jetson TX2에서 이 자습서를 실행하려면 소스 이미지를 제공하고 Lambda 함수를 구성한 후 추가 로컬 디바이스 리소스를 추가합니다.

1. AWS IoT Greengrass 코어 소프트웨어를 설치하고 추론에 GPU를 사용할 수 있도록 Jetson 디바이스가 구성되어 있는지 확인합니다. 디바이스 구성에 대한 자세한 내용은 the section called “다른 디바이스 설정” 단원을 참조하십시오. NVIDIA Jetson TX2에서 추론에 GPU를 사용하려면

Jetpack 4.3으로 보드의 이미지를 생성할 때 CUDA 10.0 및 cuDNN 7.0을 디바이스에 설치해야 합니다.

2. 이미지 분류에 사용하기 위해 Lambda 함수에 대한 정적 PNG 또는 JPG 이미지를 다운로드합니다. 이 예제는 작은 이미지 파일에 가장 효과적입니다.

   `inference.py` 파일이 들어 있는 디렉터리에 이미지 파일을 저장합니다. 이 디렉터리의 하위 디렉터리에 저장할 수도 있습니다. 이 디렉터리는 the section called "추론 Lambda 함수를 생성합니다."에서 업로드하는 Lambda 함수 배포 패키지에 있습니다.

   > **Note**
   >
   > 또는 Jetson 보드의 카메라를 구성해 소스 이미지를 캡처하기로 선택할 수 있습니다. 그러나 정적 이미지부터 시작하는 것이 좋습니다.

3. Lambda 함수 구성을 편집합니다. the section called "그룹에 Lambda 함수를 추가합니다."의 프로시저를 따르세요.

   > **Note**
   >
   > 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

   a. 컨테이너화 없이 실행하려면:

      - 다음으로 실행에서 **Another user ID/group ID**을(를) 선택합니다. uid에 **0**을 입력합니다. GUID의 경우 **0**을(를) 입력합니다.

        이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called "그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정" 섹션을 참조하세요.

> **Tip**
>
> 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.
- 제한 시간 값을 5분으로 늘립니다. 이렇게 하면 요청이 제한 시간에 너무 빨리 도달하는 경우를 피할 수 있습니다. 설정 후 추론이 실행되려면 몇 분이 걸립니다.
- 고정된 경우 True를 선택합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 메모리 제한 값을 늘립니다. GPU 모드에서 제공되는 모델을 사용하려면 2,000MB 이상을 사용합니다.
- 제한 시간 값을 5분으로 늘립니다. 이렇게 하면 요청이 제한 시간에 너무 빨리 도달하는 경우를 피할 수 있습니다. 설정 후 추론이 실행되려면 몇 분이 걸립니다.
- 고정된 경우 True를 선택합니다.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

4. Neo 최적화 모델 리소스를 그룹에 추가합니다. the section called “추론 Lambda 함수를 생성합니다.”에서 압축을 푼 샘플 패키지의 `resnet18` 디렉터리에 있는 모델 리소스를 업로드합니다. 이 디렉터리에는 Resnet-18에서 훈련된 이미지 분류 모델을 위해 미리 컴파일된 모델 아티팩트가 들어 있습니다. 다음 업데이트 사항과 함께 the section called “Neo 최적화 모델 리소스를 그룹에 추가”의 절차를 따릅니다.

   - `resnet18` 디렉터리 내의 파일을 `resnet18.zip`이라는 파일로 압축합니다.

- 머신 러닝 리소스 생성 페이지에서 리소스 이름에 **`resnet18_model`**을 입력합니다.
- `resnet18.zip` 파일을 업로드합니다.

5. 컨테이너화 모드에서 실행 중인 경우 필요한 로컬 디바이스 리소스를 추가하여 디바이스 GPU에 대한 액세스 권한을 부여하십시오.

> **Note**
>
> 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 디바이스 리소스를 구성하지 않고도 디바이스 GPU에 액세스할 수 있습니다.

a. 그룹 구성 페이지에서 리소스를 선택합니다.

b. 로컬 리소스 탭에서 로컬 리소스 추가를 선택합니다.

c. 각 리소스를 정의합니다.

- 리소스 이름 및 디바이스 경로에 다음 표의 값을 사용합니다. 표의 각 행에 대해 디바이스 리소스를 하나씩 생성합니다.
- [Resource type]에서 [Device]를 선택합니다.
- [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.

이름	디바이스 경로
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap

이름	디바이스 경로
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. 컨테이너화 모드에서 실행 중인 경우 다음 로컬 볼륨 리소스를 추가하여 디바이스 카메라에 대한 액세스 권한을 부여하세요. the section called “Neo 최적화 모델 리소스를 그룹에 추가”의 프로시저를 따르세요.

   > **Note**
   >
   > 비컨테이너화 모드에서 실행하는 경우 AWS IoT Greengrass은(는) 디바이스 리소스를 구성하지 않고도 디바이스 카메라에 액세스할 수 있습니다.

   - [리소스 유형(Resource type)]에서 [볼륨(Volume)]을 선택합니다.
   - [Group owner file access permission]에서 [Automatically add OS group permissions of the Linux group that owns the resource]를 선택합니다.

이름	소스 경로	대상 경로
shm	/dev/shm	/dev/shm
/tmp	/tmp	/tmp

7. 올바른 디렉터리를 사용하도록 그룹 구독을 업데이트합니다. 다음 업데이트 사항과 함께 the section called “그룹에 구독 추가”의 절차를 따릅니다.
   - 첫 번째 주제 필터에 **`/resnet-18/predictions`**를 입력합니다.
   - 두 번째 주제 필터에 **`/resnet-18/test`**를 입력합니다.
8. 올바른 디렉터리를 사용하도록 테스트 구독을 업데이트합니다. 다음 업데이트 사항과 함께 the section called “예제 테스트”의 절차를 따릅니다.
   - 구독의 경우 주제 구독을 선택합니다. 구독 주제에 **`/resnet-18/predictions`**를 입력합니다.
   - `/resnet-18/predictions` 페이지에서 게시할 `/resnet-18/test` 주제를 지정합니다.

AWS IoT Greengrass ML 추론 문제 해결

테스트가 성공적이지 않은 경우 다음 문제 해결 단계를 시도할 수 있습니다. Raspberry Pi 터미널에서 명령을 실행합니다.

오류 로그 확인

1. 루트 사용자로 전환하고 `log` 디렉터리로 이동합니다. AWS IoT Greengrass 로그 액세스에는 루트 권한이 필요합니다.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. `runtime.log`의 오류를 확인합니다.

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   사용자 정의 Lambda 함수 로그에서 오류를 검색할 수도 있습니다.

   ```
   cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
   ```

   자세한 내용은 the section called “로그 문제 해결” 섹션을 참조하세요.

Lambda 함수가 성공적으로 배포되었는지 확인

1. `/lambda` 디렉터리에 있는 Lambda 배포된 의 내용을 나열합니다. 명령을 실행하기 전에 자리 표시자 값을 바꿉니다.

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. 디렉터리에 3단계: 추론 Lambda 함수 생성에 업로드한 `optimizedImageClassification.zip` 배포 패키지와 동일한 콘텐츠가 들어 있는지 확인하십시오.

   `.py` 파일 및 종속성이 디렉터리의 루트에 있는지 확인합니다.

추론 모델이 성공적으로 배포되었는지 확인

1. Lambda 런타임 프로세스의 프로세스 식별 번호(PID)를 찾습니다.

   ```
   ps aux | grep lambda-function-name
   ```

   출력의 Lambda 런타임 프로세스에 대한 행의 두 번째 열에 PID가 표시됩니다.

2. Lambda 런타임 네임스페이스를 입력합니다. 명령을 실행하기 전에 자리 표시자 *pid* 값을 바꿔야 합니다.

   > **Note**
   >
   > 이 디렉터리 및 내용은 Lambda 런타임 네임스페이스입니다. 따라서 정규 Linux 네임스페이스에서는 보이지 않습니다.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. ML 리소스에 대해 지정한 로컬 디렉터리의 내용을 나열합니다.

   > **Note**
   >
   > ML 리소스 경로가 `ml_model`이 아닌 경우 여기서 대체해야 합니다.

   ```
   cd /ml_model
   ls -ls
   ```

   다음 파일이 표시됩니다.

   ```
       56 -rw-r--r-- 1 ggc_user ggc_group     56703 Oct 29 20:07 model.json
   196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params
      256 -rw-r--r-- 1 ggc_user ggc_group    261848 Oct 29 20:07 model.so
       32 -rw-r--r-- 1 ggc_user ggc_group     30564 Oct 29 20:08 synset.txt
   ```

Lambda 함수에서 **/dev/dri/renderD128**을 찾을 수 없습니다.

이 문제는 OpenCL이 그에 필요한 GPU 디바이스에 연결될 수 없을 때 발생합니다. Lambda 함수에 필요한 디바이스에 대해 디바이스 리소스를 생성해야 합니다.

다음 단계

다음에는 기타 최적화된 모델을 탐색합니다. 자세한 내용은 SageMaker Neo 설명서를 참조하십시오.

AWS IoT Greengrass 코어에서 데이터 스트림 관리

AWS IoT Greengrass 스트림 관리자를 사용하면 대용량 IoT 데이터를 AWS 클라우드로 보다 쉽고 안정적으로 전송할 수 있습니다. 스트림 관리자는 데이터 스트림을 로컬에서 처리하고 이를 AWS 클라우드로 자동으로 내보냅니다. 이 기능은 AWS 클라우드 또는 로컬 스토리지 대상으로 내보내기 전에 데이터가 로컬에서 처리되고 분석되는 기계 학습(ML) 추론과 같은 일반적인 에지 시나리오에 통합됩니다.

스트림 관리자는 애플리케이션 개발을 단순화합니다. IoT 애플리케이션은 사용자 지정 스트림 관리 기능을 구축하는 대신 표준화된 메커니즘을 사용하여 대용량 스트림을 처리하고 로컬 데이터 보존 정책을 관리할 수 있습니다. IoT 애플리케이션은 스트림에 대해 읽기 및 쓰기를 수행할 수 있습니다. 스트림별로 스토리지 유형, 크기 및 데이터 보존에 대한 정책을 정의하여 스트림 관리자가 스트림을 처리하고 내보내는 방법을 제어할 수 있습니다.

스트림 관리자는 연결이 간헐적이거나 제한된 환경에서 작동하도록 설계되었습니다. 대역폭 사용, 제한 시간 동작, 코어가 연결되거나 연결이 끊어질 때 스트림 데이터가 처리되는 방식을 정의할 수 있습니다. 중요한 데이터의 경우 우선 순위를 설정하여 스트림을 AWS 클라우드로 내보내는 순서를 제어할 수 있습니다.

사용자는 저장 또는 추가 처리 및 분석을 위해 AWS 클라우드로 자동 내보내기를 구성할 수 있습니다. 스트림 관리자는 다음 AWS 클라우드 목적지로 내보내기를 지원합니다.

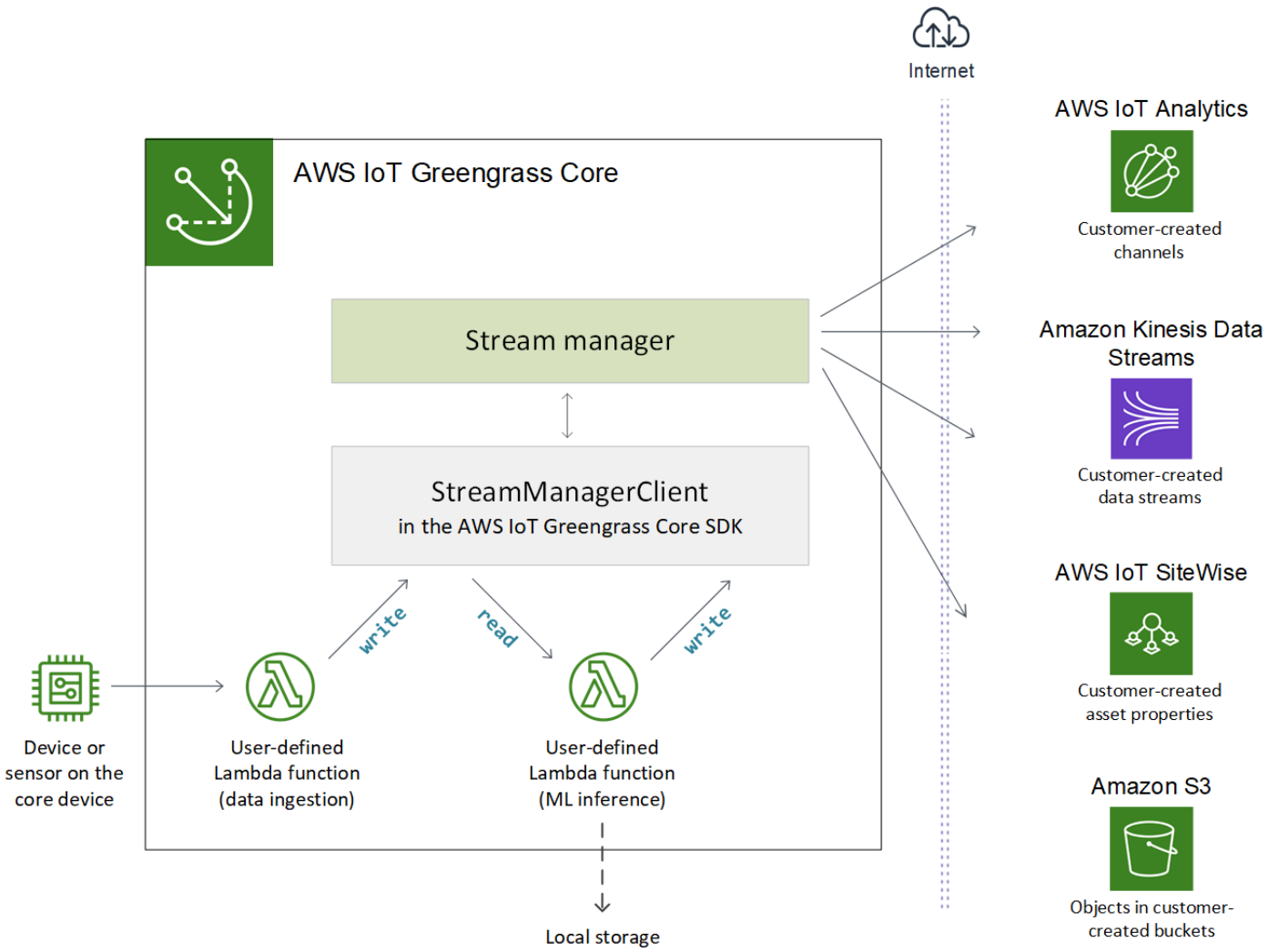
- AWS IoT Analytics의 채널. AWS IoT Analytics를 이용하면 데이터에 대한 고급 분석을 수행하여 비즈니스 결정을 내리고 기계 학습 모델을 개선할 수 있습니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 AWS IoT Analytics란 무엇인가요?를 참조하세요.
- Kinesis Data Streams의 스트림. Kinesis Data Streams는 일반적으로 대용량 데이터를 종합하여 데이터 웨어하우스나 map-reduce 클러스터에 로드하는 데 사용됩니다. 자세한 내용은 Amazon Kinesis 개발자 안내서의 Amazon Kinesis Data Streams이란 무엇입니까?를 참조하세요.
- AWS IoT SiteWise의 자산 속성. AWS IoT SiteWise는 대규모로 산업 장비 데이터를 수집, 조직 및 분석할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 AWS IoT SiteWise란 무엇인가요?를 참조하세요.
- Amazon S3 객체. Amazon S3를 사용하여 대량의 데이터를 저장 및 검색할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 Amazon S3란 무엇인가요?를 참조하세요.

스트림 관리 워크플로우

IoT 애플리케이션은 AWS IoT Greengrass 코어 SDK를 통해 스트림 관리자와 상호 작용합니다. 간단한 워크플로우에서 Greengrass 코어에서 실행 중인 사용자 정의 Lambda 함수는 시계열 온도 및 압력 지표와 같은 IoT 데이터를 소비합니다. 이 Lambda 함수는 데이터를 필터링 또는 압축한 다음, AWS IoT Greengrass 코어 SDK를 호출하여 스트림 관리자의 스트림에 데이터를 기록할 수 있습니다. 스트림 관리자는 스트림에 대해 정의된 정책에 따라 자동으로 스트림을 AWS 클라우드로 내보낼 수 있습니다. 사용자 정의 Lambda 함수는 데이터를 로컬 데이터베이스 또는 스토리지 리포지토리로 직접 보낼 수도 있습니다.

IoT 애플리케이션에는 스트림에 대해 읽기 또는 쓰기를 수행하는 여러 개의 사용자 정의 Lambda 함수가 포함될 수 있습니다. 이러한 로컬 Lambda 함수는 스트림 읽기 및 쓰기를 수행하여 로컬에서 데이터를 필터링, 집계 및 분석할 수 있습니다. 따라서 데이터가 코어에서 클라우드 또는 로컬 대상으로 전송되기 전에 로컬 이벤트에 신속하게 대응하고 중요한 정보를 추출하는 것이 가능합니다.

다음 다이어그램에는 예제 워크플로우가 나와 있습니다.



스트림 관리자를 사용하려면 먼저 스트림 관리자 파라미터를 구성하여 Greengrass 코어의 모든 스트림에 적용되는 그룹 수준 런타임 설정을 정의합니다. 이러한 사용자 정의 가능한 설정을 통해 비즈니스 요구 사항 및 환경 제약에 따라 스트림 관리자가 스트림을 저장, 처리 및 내보내는 방법을 제어할 수 있습니다. 자세한 내용은 the section called “스트림 관리자 구성” 섹션을 참조하세요.

스트림 관리자를 구성한 후 IoT 애플리케이션을 생성하고 배포할 수 있습니다. 이들은 일반적으로 AWS IoT Greengrass 코어 SDK에서 스트림을 생성하고 스트림과 상호 작용하는 데 `StreamManagerClient`를 사용하는 사용자 정의 Lambda 함수입니다. 스트림 생성 중에 Lambda 함수는 내보내기 대상, 우선순위, 지속성과 같은 스트림별 정책을 정의합니다. `StreamManagerClient` 작업을 위한 코드 스니펫을 포함한 자세한 내용은 the section called “StreamManagerClient를 사용하여 스트림 작업”를 참조하세요.

간단한 워크플로우를 구성하는 자습서는 또는 the section called “데이터 스트림 내보내기(콘솔)” 또는 the section called “데이터 스트림 내보내기(CLI)”를 참조하세요.

요구 사항

스트림 관리자 사용을 위해 다음 요구 사항이 적용됩니다.

- 스트림 관리자가 활성화된 AWS IoT Greengrass Core 소프트웨어 v1.10 이상을 사용해야 합니다. 자세한 내용은 the section called “스트림 관리자 구성” 섹션을 참조하세요.

  스트림 관리자는 OpenWrt 배포에서 지원되지 않습니다.
- Java 8 런타임(JDK 8)이 코어에 설치되어 있어야 합니다.
  - Debian 기반 배포판(Raspbian 포함) 또는 Ubuntu 기반 배포판의 경우 다음 명령을 실행합니다.

    ```
    sudo apt install openjdk-8-jdk
    ```
  - Red Hat 기반 배포판(Amazon Linux 포함) 의 경우 다음 명령을 실행합니다.

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    자세한 내용은 OpenJDK 설명서의 OpenJDK 패키지 다운로드 및 설치 방법을 참조하십시오.
- 스트림 관리자에는 기본 AWS IoT Greengrass 코어 소프트웨어 외에 최소 70MB의 RAM이 필요합니다. 총 메모리 요구 사항은 워크로드에 따라 다릅니다.
- 사용자 정의 Lambda 함수는 AWS IoT Greengrass 코어 SDK를 사용하여 스트림 관리자와 상호 작용해야 합니다. AWS IoT Greengrass 코어 SDK는 여러 언어로 사용할 수 있지만, 다음 버전만 스트림 관리자 작업을 지원합니다.
  - Java SDK(v1.4.0 이상)
  - Python SDK(v1.5.0 이상)
  - Node.js SDK(v1.6.0 이상)

  Lambda 함수 런타임에 해당하는 버전의 SDK를 다운로드하여 Lambda 함수 배포 패키지에 포함시킵니다.

> **Note**
>
> Python용 AWS IoT Greengrass 코어 SDK에는 Python 3.7 이상이 필요하며 다른 패키지 종속성이 있습니다. 자세한 내용은 Lambda 함수 배포 패키지 생성(콘솔) 또는 Lambda 함수 배포 패키지 생성(CLI)을 참조하십시오.

- 스트림에 대한 AWS 클라우드 내보내기 대상을 정의하는 경우, 내보내기 대상을 생성하고 Greengrass 그룹 역할에서 이들에 액세스할 수 있는 권한을 부여해야 합니다. 대상에 따라 다른 요구 사항도 적용될 수 있습니다. 자세한 내용은 다음을 참조하세요.
  - the section called “AWS IoT Analytics 채널”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “AWS IoT SiteWise 자산 속성”
  - the section called “Amazon S3 객체”

  이러한 AWS 클라우드 리소스를 관리할 책임은 사용자에게 있습니다.

데이터 보안

스트림 관리자를 사용할 때는 다음과 같은 보안 고려 사항에 유의하십시오.

로컬 데이터 보안

AWS IoT Greengrass은 코어 디바이스의 구성 요소 간에 로컬로 저장 시 또는 전송 중에 스트림 데이터를 암호화하지 않습니다.

- 저장 시 데이터. 스트림 데이터는 Greengrass 코어의 스토리지 디렉터리에 로컬로 저장됩니다. 데이터 보안을 위해 AWS IoT Greengrass는 Unix 파일 사용 권한 및 전체 디스크 암호화를 사용합니다(활성화된 경우). 선택적 STREAM_MANAGER_STORE_ROOT_DIR 파라미터를 사용하여 스토리지 디렉터리를 지정할 수 있습니다. 다른 스토리지 디렉터리를 사용하도록 나중에 이 파라미터를 변경하는 경우, AWS IoT Greengrass에서 이전의 스토리지 디렉터리 또는 해당 내용이 삭제되지 않습니다.

- 로컬로 전송 중인 데이터. AWS IoT Greengrass는 데이터 소스, Lambda 함수, AWS IoT Greengrass 코어 SDK 및 스트림 관리자 간에 로컬 전송 시 스트림 데이터를 암호화하지 않습니다.

- AWS 클라우드로 전송 중인 데이터. 스트림 관리자가 AWS 클라우드로 내보낸 데이터 스트림은 전송 계층 보안(TLS)에서 표준 AWS 서비스 클라이언트 암호화를 사용합니다.

자세한 내용은 the section called “데이터 암호화” 섹션을 참조하세요.

클라이언트 인증

스트림 관리자 클라이언트는 AWS IoT Greengrass 코어 SDK를 사용하여 스트림 관리자와 통신합니다. 클라이언트 인증이 활성화되면 Greengrass 그룹의 Lambda 함수만 스트림 관리자의 스트림과 상호 작용할 수 있습니다. 클라이언트 인증이 비활성화되면 Greengrass 코어(예: Docker 컨테이너)에서 실행 중인 모든 프로세스가 스트림 관리자의 스트림과 상호 작용할 수 있습니다. 비즈니스 사례에 필요한 경우에만 인증을 비활성화해야 합니다.

STREAM_MANAGER_AUTHENTICATE_CLIENT 파라미터를 사용하여 클라이언트 인증 모드를 설정합니다. 콘솔 또는 AWS IoT Greengrass API에서 이 파라미터를 구성할 수 있습니다. 변경 사항은 그룹이 배포된 후에 적용됩니다.

	활성화됨	비활성화됨
파라미터 값	`true`(기본값 및 권장)	`false`
허용된 클라이언트	Greengrass 그룹의 사용자 정의 Lambda 함수	Greengrass 그룹의 사용자 정의 Lambda 함수 Greengrass 코어 디바이스에서 실행 중인 기타 프로세스

다음 사항도 참조하세요.

- the section called “스트림 관리자 구성”
- the section called “StreamManagerClient를 사용하여 스트림 작업”
- the section called “지원되는 AWS 클라우드 대상의 구성 내보내기”
- the section called “데이터 스트림 내보내기(콘솔)”
- the section called “데이터 스트림 내보내기(CLI)”

AWS IoT Greengrass 스트림 관리자 구성

AWS IoT Greengrass 코어에서 스트림 관리자는 IoT 디바이스 데이터를 저장, 처리 및 내보낼 수 있습니다. 스트림 관리자는 그룹 수준의 런타임 설정을 구성하는 데 사용하는 파라미터를 제공합니다. 이러한 설정은 Greengrass 코어의 모든 스트림에 적용됩니다. AWS IoT 콘솔 또는 AWS IoT Greengrass API를 사용하여 스트림 관리자 설정을 구성할 수 있습니다. 변경 사항은 그룹이 배포된 후에 적용됩니다.

> **Note**
>
> 스트림 관리자를 구성한 후에는 Greengrass 코어에서 실행되는 IoT 애플리케이션을 생성 및 배포하고 스트림 관리자와 상호 작용할 수 있습니다. 이러한 IoT 애플리케이션은 일반적으로 사용자 정의 Lambda 함수입니다. 자세한 내용은 the section called “StreamManagerClient를 사용하여 스트림 작업” 섹션을 참조하세요.

스트림 관리자 파라미터

스트림 관리자는 그룹 수준 설정을 정의할 수 있도록 다음과 같은 파라미터를 제공합니다. 모든 파라미터는 선택 사항입니다.

스토리지 디렉터리

파라미터 이름: `STREAM_MANAGER_STORE_ROOT_DIR`

스트림을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로입니다. 이 값은 슬래시로 시작해야 합니다(예: `/data`).

스트림 데이터 보안에 대한 자세한 내용은 the section called “로컬 데이터 보안” 섹션을 참조하십시오.

최소 AWS IoT Greengrass 코어 버전: 1.10.0

[Server port]

파라미터 이름: `STREAM_MANAGER_SERVER_PORT`

스트림 관리자와 통신하는 데 사용되는 로컬 포트 번호입니다. 기본값은 `8088`입니다.

최소 AWS IoT Greengrass 코어 버전: 1.10.0

클라이언트 인증

파라미터 이름: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

스트림 관리자와 상호 작용하기 위해 클라이언트가 인증되어야 하는지 여부를 나타냅니다. 클라이언트와 스트림 관리자 간의 모든 상호 작용은 AWS IoT Greengrass Core SDK에 의해 제어됩니다. 이 파라미터는 AWS IoT Greengrass Core SDK를 호출하여 스트림 작업을 수행할 수 있는 클라이언트를 결정합니다. 자세한 내용은 the section called “클라이언트 인증” 섹션을 참조하세요.

유효한 값은 `true` 또는 `false`입니다. 기본값은 `true`입니다(권장).

- `true`. Greengrass Lambda 함수만 클라이언트로 사용할 수 있습니다. Lambda 함수 클라이언트는 AWS IoT Greengrass 코어 프로토콜을 사용하여 AWS IoT Greengrass Core SDK로 인증합니다.
- `false`. AWS IoT Greengrass 코어에서 실행되는 모든 프로세스를 클라이언트로 허용합니다. 비즈니스 사례에 필요한 경우가 아니면 `false`로 설정하지 마십시오. 예를 들어 코어 디바이스의 비 Lambda 프로세스가 스트림 관리자(예: 코어에서 실행되는 Docker 컨테이너)와 직접 통신해야 하는 경우에만 이 값을 `false`로 설정합니다.

최소 AWS IoT Greengrass 코어 버전: 1.10.0

최대 대역폭

파라미터 이름: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

데이터를 내보내는 데 사용할 수 있는 평균 최대 대역폭(초당 킬로비트)입니다. 기본값은 사용 가능한 대역폭을 무제한으로 사용할 수 있도록 허용합니다.

최소 AWS IoT Greengrass 코어 버전: 1.10.0

스레드 풀 크기

파라미터 이름: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

데이터를 내보내는 데 사용할 수 있는 최대 활성 스레드 수입니다. 기본값은 5입니다.

최적의 크기는 하드웨어, 스트림 볼륨 및 계획된 내보내기 스트림 수에 따라 다릅니다. 내보내기 속도가 느린 경우 이 설정을 조정하여 하드웨어 및 비즈니스 사례에 가장 적합한 크기를 찾을 수 있습니다. 코어 디바이스 하드웨어의 CPU 및 메모리는 제한적인 요소입니다. 시작을 위해 이 값을 디바이스의 프로세서 코어 수와 동일하게 설정해 볼 수 있습니다.

하드웨어가 지원할 수 있는 크기 보다 크게 설정하지 않도록 주의하십시오. 각 스트림은 하드웨어 리소스를 소비하므로 제한된 디바이스에서 내보내기 스트림 수를 제한해야 합니다.

최소 AWS IoT Greengrass 코어 버전: 1.10.0

JVM 인수

파라미터 이름: `JVM_ARGS`

시작 시 스트림 관리자에게 전달할 사용자 정의 Java 가상 머신 인수입니다. 여러 인수는 공백으로 구분해야 합니다.

JVM에서 사용하는 기본 설정을 재정의해야 하는 경우에만 이 파라미터를 사용합니다. 예를 들어 많은 수의 스트림을 내보낼 계획이 있다면 기본 힙 크기를 늘려야 할 수 있습니다.

최소 AWS IoT Greengrass 코어 버전: 1.10.0

읽기 전용 입력 파일 디렉터리

파라미터 이름: `STREAM_MANAGER_READ_ONLY_DIRS`

입력 파일을 저장하는 루트 파일 시스템 외부의 디렉터리에 대한 절대 경로를 쉼표로 구분한 목록입니다. 스트림 관리자는 Amazon S3에 파일을 읽고 업로드하고 디렉터리를 읽기 전용으로 마운트합니다. Amazon S3에 로그 데이터 내보내기에 대한 자세한 내용은 the section called “Amazon S3 객체” 섹션을 참조하세요.

다음 조건이 충족되는 경우에만 이 파라미터를 사용하십시오.

- Amazon S3로 내보내는 스트림의 입력 파일 디렉터리는 다음 위치 중 하나에 있습니다.
  - 루트 파일 시스템이 아닌 파티션.
  - 루트 파일 시스템의 `/tmp` 아래.
- Greengrass 그룹의 기본 컨테이너화는 Greengrass 컨테이너입니다.

예제 값: `/mnt/directory-1,/mnt/directory-2,/tmp`

최소 AWS IoT Greengrass 코어 버전: 1.11.0

멀티파트 업로드의 최소 크기

파라미터 이름:
`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

Amazon S3에 대한 멀티파트 업로드의 최소 부분 크기(바이트). 스트림 관리자는 이 설정과 입력 파일의 크기를 사용하여 멀티파트 PUT 요청에서 데이터를 일괄 처리하는 방법을 결정합니다. 기본값 및 최소값은 `5242880` 바이트(5MB)입니다.

> **Note**
>
> 스트림 관리자는 스트림의 `sizeThresholdForMultipartUploadBytes` 속성을 사용하여 Amazon S3에 단일 업로드로 내보낼지 멀티파트 업로드로 내보낼지 결정합니다. 사용자 정의 Lambda 함수는 Amazon S3로 내보내는 스트림을 생성할 때 이 임계값을 설정합니다. 기본 임계값 크기는 5MB입니다.

최소 AWS IoT Greengrass 코어 버전: 1.11.0

스트림 관리자 설정 구성(콘솔)

다음과 같은 관리 작업에서 AWS IoT 콘솔을 사용할 수 있습니다.

- 스트림 관리자가 활성화되어 있는지 확인
- 그룹 생성 중에 스트림 관리자 활성화 또는 비활성화
- 기존 그룹에 대해 스트림 관리자를 활성화 또는 비활성화
- 스트림 관리자 설정 변경

변경 사항은 Greengrass 그룹이 배포된 후에 적용됩니다. 스트림 관리자와 상호 작용하는 Lambda 함수를 포함하는 Greengrass 그룹을 배포하는 방법을 보여주는 튜토리얼은 the section called “데이터 스트림 내보내기(콘솔)”을 참조하십시오.

> **Note**
>
> 콘솔을 사용하여 스트림 관리자를 활성화하고 그룹을 배포하는 경우 스트림 관리자의 메모리 크기가 기본 4GB로 설정됩니다. 메모리 크기를 최소 128,000KB로 설정하는 것이 좋습니다.

스트림 관리자가 활성화되어 있는지 확인하려면(콘솔)

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. Lambda 함수 탭을 선택합니다.

4. 시스템 Lambda 함수에서 스트림 관리자를 선택하고 편집을 선택합니다.
5. 활성화 또는 비활성화 상태를 확인합니다. 구성된 모든 사용자 지정 스트림 관리자 설정도 표시됩니다.

그룹 생성 중에 스트림 관리자를 활성화 또는 비활성화하려면(콘솔)

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 그룹 생성을 선택합니다. 다음 페이지에서 선택한 항목에 따라 그룹에 대한 스트림 관리자를 구성하는 방법이 결정됩니다.
3. 그룹 이름 지정을 진행하고 Greengrass 코어 페이지를 선택합니다.
4. 그룹 생성을 선택합니다.
5. 그룹 구성 페이지에서 Lambda 함수 탭을 선택하고 스트림 관리자를 선택한 다음 편집을 선택합니다.

   - 스트림 관리자를 기본 설정으로 활성화하려면 기본 설정으로 활성화를 선택합니다.

   - 사용자 지정 설정으로 스트림 관리자를 활성화하려면 설정 사용자 지정을 선택합니다.
     1. 스트림 관리자 구성 페이지에서 사용자 지정 설정으로 활성화를 선택합니다.
     2. 사용자 정의 설정에 스트림 관리자 파라미터의 값을 입력합니다. 자세한 내용은 the section called “스트림 관리자 파라미터” 섹션을 참조하세요. AWS IoT Greengrass이 기본값을 사용할 수 있도록 필드를 비워 둡니다.

   - 스트림 관리자를 비활성화하려면 비활성화를 선택합니다.
     1. Configure stream manager(스트림 관리자 구성) 페이지에서 비활성화를 선택합니다.

6. Save를 선택합니다.
7. 나머지 페이지를 계속 진행하여 그룹을 생성합니다.
8. 클라이언트 디바이스 페이지에서 보안 리소스를 다운로드하고 정보를 검토한 다음, 완료를 선택합니다.

> **Note**
>
> 스트림 관리자가 활성화된 경우 그룹을 배포하기 전에 코어 디바이스에 Java 8 런타임을 설치해야 합니다.

기존 그룹에 대해 스트림 관리자를 활성화 또는 비활성화하려면(콘솔)

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. Lambda 함수 탭을 선택합니다.
4. 시스템 Lambda 함수에서 스트림 관리자를 선택하고 편집을 선택합니다.
5. 활성화 또는 비활성화 상태를 확인합니다. 구성된 모든 사용자 지정 스트림 관리자 설정도 표시됩니다.

스트림 관리자 설정을 변경하려면(콘솔)

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. Lambda 함수 탭을 선택합니다.
4. 시스템 Lambda 함수에서 스트림 관리자를 선택하고 편집을 선택합니다.
5. 활성화 또는 비활성화 상태를 확인합니다. 구성된 모든 사용자 지정 스트림 관리자 설정도 표시됩니다.
6. Save를 선택합니다.

스트림 관리자 설정 구성(CLI)

AWS CLI에서는 시스템 `GGStreamManager` Lambda 함수를 사용하여 스트림 관리자를 구성합니다. 시스템 Lambda 함수는 AWS IoT Greengrass 코어 소프트웨어의 구성 요소입니다. 스트림 관리자 및 일부 기타 시스템 Lambda 함수의 경우 Greengrass 그룹에서 해당 `Function` 및

`FunctionDefinitionVersion` 객체를 관리하여 Greengrass 기능을 구성할 수 있습니다. 자세한 내용은 the section called “그룹 객체 모델 개요” 섹션을 참조하세요.

다음과 같은 관리 작업에서 API를 사용할 수 있습니다. 이 섹션의 예제는 AWS CLI를 사용하는 방법을 보여 주지만 AWS IoT Greengrass API를 직접 호출하거나 AWS SDK를 사용할 수도 있습니다.

- 스트림 관리자가 활성화되어 있는지 확인
- 스트림 관리자를 활성화, 비활성화 또는 구성

변경 사항은 그룹이 배포된 후에 적용됩니다. 스트림 관리자와 상호 작용하는 Lambda 함수를 사용하여 Greengrass 그룹을 배포하는 방법을 보여주는 튜토리얼은 the section called “데이터 스트림 내보내기(CLI)” 섹션을 참조하십시오.

> **Tip**
>
> 스트림 관리자가 활성화되어 있고 코어 디바이스에서 실행 중인지 확인하려면 디바이스의 터미널에서 다음 명령을 실행할 수 있습니다.
>
> ```
> ps aux | grep -i 'streammanager'
> ```

스트림 관리자가 활성화되어 있는지 확인하려면(CLI)

배포된 함수 정의 버전에 시스템 `GGStreamManager` Lambda 함수가 포함되어 있으면 스트림 관리자가 활성화됩니다. 확인하려면 다음을 수행하십시오.

1. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

> **Note**
>
> AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

2. 출력에서 대상 그룹의 `Id` 및 `LatestVersion` 값을 복사합니다.
3. 최신 그룹 버전을 확인합니다.

   - *`group-id`*를 복사한 `Id`로 바꿉니다.
   - *`latest-group-version-id`*를 복사한 `LatestVersion`으로 바꿉니다.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. 출력의 `FunctionDefinitionVersionArn`에서 함수 정의 및 함수 정의 버전의 ID를 복사합니다.

   - 함수 정의 ID는 Amazon 리소스 이름(ARN)의 `functions` 세그먼트 뒤에 나오는 GUID입니다.
   - 함수 정의 버전 ID는 ARN의 `versions` 세그먼트 뒤에 나오는 GUID입니다.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/function-definition-id/versions/function-definition-version-id
```

5. 함수 정의 버전을 가져옵니다.

   - *`function-definition-id`*를 함수 정의 ID로 바꿉니다.
   - *`function-definition-version-id`*를 함수 정의 버전 ID로 바꿉니다.

```
aws greengrass get-function-definition-version \
--function-definition-id function-definition-id \
--function-definition-version-id function-definition-version-id
```

출력의 `functions` 배열이 `GGStreamManager` 함수를 포함하는 경우, 스트림 관리자가 활성화됩니다. 함수에 대해 정의된 모든 환경 변수는 스트림 관리자의 사용자 지정 설정을 나타냅니다.

스트림 관리자를 활성화, 비활성화 또는 구성하려면(CLI)

AWS CLI에서는 시스템 `GGStreamManager` Lambda 함수를 사용하여 스트림 관리자를 구성합니다. 변경 사항은 그룹을 배포한 후에 적용됩니다.

- 스트림 관리자를 활성화하려면 함수 정의 버전의 `functions` 배열에 `GGStreamManager`을 포함시킵니다. 사용자 지정 설정을 구성하려면 해당 스트림 관리자 파라미터에 대한 환경 변수를 정의합니다.
- 스트림 관리자를 비활성화하려면 함수 정의 버전의 `functions` 배열에서 `GGStreamManager`을 제거합니다.

기본 설정의 스트림 관리자

다음 예제 구성에서는 기본 설정으로 스트림 관리자를 활성화합니다. 임의의 함수 ID를 `streamManager`로 설정합니다.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

> **Note**
>
> `FunctionConfiguration` 속성의 경우 다음 사항을 알고 있을 수 있습니다.
>
> - `MemorySize`는 기본 설정으로 4194304KB(4GB)로 설정됩니다. 이 값은 언제든지 변경할 수 있습니다. 최소 128,000KB로 `MemorySize`를 설정하는 것이 좋습니다.
> - `Pinned`를 `true`로 설정해야 합니다.
> - `Timeout`은 함수 정의 버전에서 필요하지만, `GGStreamManager`에서는 이를 사용하지 않습니다.

사용자 지정 설정의 스트림 관리자

다음 예제 구성에서는 스토리지 디렉터리, 서버 포트 및 스레드 풀 크기 파라미터에 대한 사용자 지정 값을 사용하여 스트림 관리자를 활성화합니다.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                "STREAM_MANAGER_SERVER_PORT": "1234",
                "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

AWS IoT Greengrass는 환경 변수로 지정되지 않은 스트림 관리자 파라미터에 대해 기본값을 사용합니다.

Amazon S3 내보내기에 대한 사용자 지정 설정이 포함된 스트림 관리자

다음 예제 구성을 사용하면 스트림 관리자가 업로드 디렉터리 및 최소 멀티파트 업로드 크기 파라미터에 대한 사용자 지정 값을 사용할 수 있습니다.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",

 "STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
 "10485760"
            }
        },
        "MemorySize": 4194304,
```

```
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

스트림 관리자를 활성화, 비활성화 또는 구성하려면(CLI)

1. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

2. 출력에서 대상 그룹의 `Id` 및 `LatestVersion` 값을 복사합니다.
3. 최신 그룹 버전을 확인합니다.

   - *group-id*를 복사한 `Id`로 바꿉니다.
   - *latest-group-version-id*를 복사한 `LatestVersion`으로 바꿉니다.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. FunctionDefinitionVersionArn을 제외한 CoreDefinitionVersionArn 및 모든 버전 ARN을 출력에서 복사합니다. 나중에 새 그룹 버전을 만들 때 이러한 값들이 사용됩니다.
5. 출력의 FunctionDefinitionVersionArn에서 함수 정의의 ID를 복사합니다. 다음 예제에서와 같이, ARN에서 functions 세그먼트 다음에 나오는 GUID가 ID가 됩니다.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> 또는 create-function-definition 명령을 실행하여 함수 정의를 생성하고, 출력에서 ID를 복사할 수 있습니다.

6. 함수 정의에 함수 정의 버전을 추가합니다.

   - *function-definition-id*를 함수 정의에서 복사한 Id로 바꿉니다.
   - functions 배열에 Greengrass 코어에 사용할 수 있게 하려는 다른 모든 함수를 포함시킵니다. get-function-definition-version 명령을 사용하여 기존 함수 목록을 가져올 수 있습니다.

기본 설정으로 스트림 관리자 활성화

다음 예제에서는 functions 배열에 GGStreamManager 함수를 포함하여 스트림 관리자를 활성화합니다. 이 예제에서는 스트림 관리자 파라미터에 기본값을 사용합니다.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
```

```
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> 이 예제의 myLambdaFunction 함수는 사용자 정의 Lambda 함수 중 하나를 나타냅니다.

사용자 지정 설정으로 스트림 관리자 활성화

다음 예제에서는 functions 배열에 GGStreamManager 함수를 포함하여 스트림 관리자를 활성화합니다. 기본값을 변경하지 않는 한 모든 스트림 관리자 설정은 선택 사항입니다. 이 예제에서는 환경 변수를 사용하여 사용자 지정 값을 설정하는 방법을 보여줍니다.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "Environment": {
                    "Variables": {
                        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                        "STREAM_MANAGER_SERVER_PORT": "1234",
                        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                    }
                },
```

```
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> FunctionConfiguration 속성의 경우 다음 사항을 알고 있을 수 있습니다.
>
> - MemorySize는 기본 설정으로 4194304KB(4GB)로 설정됩니다. 이 값은 언제든지 변경할 수 있습니다. 최소 128,000KB로 MemorySize를 설정하는 것이 좋습니다.
> - Pinned를 true로 설정해야 합니다.
> - Timeout은 함수 정의 버전에서 필요하지만, GGStreamManager에서는 이를 사용하지 않습니다.

스트림 관리자 비활성화

다음 예제에서는 GGStreamManager 함수를 생략하여 스트림 관리자를 비활성화합니다.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
```

```
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> Lambda 함수를 배포하지 않으려면 함수 정의 버전을 완전히 생략할 수 있습니다.

7. 출력에서 함수 정의 버전의 `Arn`를 복사합니다.
8. 시스템 Lambda 함수를 포함하는 그룹 버전을 만듭니다.

   - *group-id*를 그룹의 `Id`로 바꿉니다.
   - *core-definition-version-arn*을 최신 그룹 버전에서 복사한 `CoreDefinitionVersionArn`으로 바꿉니다.
   - *function-definition-version-arn*을 새 함수 정의 버전에서 복사한 `Arn`으로 바꿉니다.
   - 최신 그룹 버전에서 복사한 다른 그룹 구성 요소의 ARN(예: `SubscriptionDefinitionVersionArn` 또는 `DeviceDefinitionVersionArn`)을 바꿉니다.
   - 사용되지 않은 파라미터를 모두 제거합니다. 예를 들어 그룹 버전에 리소스가 포함되어 있지 않으면 `--resource-definition-version-arn`을 제거합니다.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
```

```
--subscription-definition-version-arn subscription-definition-version-arn
```

9. 출력에서 `Version`을 복사합니다. 새 그룹 버전의 ID가 이 값이 됩니다.
10. 새로운 그룹 버전을 사용하여 그룹을 배포합니다.

    - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
    - *group-version-id*를 새 그룹 버전에서 복사한 `Version`으로 바꿉니다.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

나중에 스트림 관리자 설정을 다시 편집하려면 이 절차를 따르십시오. 업데이트된 구성으로 `GGStreamManager` 함수가 포함된 함수 정의 버전을 생성하십시오. 그룹 버전은 코어에 배포하려는 모든 구성 요소 버전 ARN을 참조해야 합니다. 변경 사항은 그룹이 배포된 후에 적용됩니다.

다음 사항도 참조하세요.

- 데이터 스트림 관리
- the section called “StreamManagerClient를 사용하여 스트림 작업”
- the section called “지원되는 AWS 클라우드 대상의 구성 내보내기”
- the section called “데이터 스트림 내보내기(콘솔)”
- the section called “데이터 스트림 내보내기(CLI)”

StreamManagerClient를 사용하여 스트림 작업

AWS IoT Greengrass 코어에서 실행되는 사용자 정의 Lambda 함수는 AWS IoT Greengrass Core SDK의 `StreamManagerClient`객체를 사용하여 스트림 관리자에서 스트림을 생성한 다음 스트림과 상호 작용할 수 있습니다. Lambda 함수는 스트림을 생성할 때 스트림에 대한 AWS 클라우드 대상, 우선 순위, 기타 내보내기 및 데이터 보존 정책을 정의합니다. 스트림 관리자로 데이터를 전송하기 위해 Lambda 함수는 데이터를 스트림에 추가합니다. 스트림에 대해 내보내기 대상이 정의된 경우 스트림 관리자는 스트림을 자동으로 내보냅니다.

> **Note**
>
> 일반적으로 스트림 관리자의 클라이언트는 사용자 정의 Lambda 함수입니다. 비즈니스 사례에서 필요한 경우 Greengrass 코어(예: Docker 컨테이너)에서 실행되는 비 Lambda 프로세스 스트림 관리자와 상호 작용하도록 허용할 수 있습니다. 자세한 내용은 the section called “클라이언트 인증” 섹션을 참조하세요.

이 주제의 코드 조각은 클라이언트가 스트림 작업을 위해 `StreamManagerClient` 메서드를 직접적으로 호출하여 사용하는 방법을 보여줍니다. 메서드 및 해당 인수에 대한 구현 세부 정보를 보려면 각 코드 조각 다음에 나열된 SDK 참조에 대한 링크를 사용합니다. 전체 Python Lambda 함수를 포함하는 자습서는 the section called “데이터 스트림 내보내기(콘솔)” 단원 또는 the section called “데이터 스트림 내보내기(CLI)” 단원을 참조하십시오.

Lambda 함수는 함수 핸들러 외부에서 `StreamManagerClient`을(를) 인스턴스화해야 합니다. 핸들러에서 인스턴스화된 함수는 호출될 때마다 스트림 관리자에 대한 `client` 및 연결을 만듭니다.

> **Note**
>
> 핸들러에서 `StreamManagerClient`을(를) 인스턴스한 경우, `client`이(가) 작업을 완료하면 사용자가 `close()` 메서드를 명시적으로 호출해야 합니다. 그렇지 않으면 `client`은(는) 연결을 열어두고 스크립트가 종료될 때까지 다른 스레드를 실행합니다.

`StreamManagerClient`에서는 다음 작업을 지원합니다.

- the section called “메시지 스트림 생성”
- the section called “메시지 추가”
- the section called “메시지 읽기”
- the section called “스트림 나열”
- the section called “메시지 스트림 설명”
- the section called “메시지 스트림 업데이트”
- the section called “메시지 스트림 삭제”

메시지 스트림 생성

스트림을 생성하기 위해 사용자 정의 Lambda 함수가 생성 메서드를 직접적으로 호출하고 `MessageStreamDefinition` 객체를 전달합니다. 이 객체는 스트림의 고유한 이름을 지정하고 최대 스트림 크기에 도달했을 때 스트림 관리자가 새 데이터를 처리하는 방법을 정의합니다. `MessageStreamDefinition` 및 해당 데이터 유형(예: `ExportDefinition`, `StrategyOnFull` 및 `Persistence`)을 사용하여 다른 스트림 속성을 정의할 수 있습니다. 다음이 포함됩니다.

- 자동 내보내기를 위한 대상 AWS IoT Analytics, Kinesis Data Streams, AWS IoT SiteWise, 및 Amazon S3 대상. 자세한 내용은 the section called “지원되는 AWS 클라우드 대상의 구성 내보내기” 섹션을 참조하세요.
- 내보내기 우선 순위. 스트림 관리자는 우선 순위가 낮은 스트림보다 우선 순위가 높은 스트림을 먼저 내보냅니다.
- AWS IoT Analytics, Kinesis Data Streams 및 AWS IoT SiteWise 대상에 대한 최대 배치 크기 및 배치 간격 스트림 관리자는 두 조건 중 하나가 충족되면 메시지를 내보냅니다.
- TTL(Time-to-Live). 스트림 데이터를 처리에 사용할 수 있도록 보장하는 시간입니다. 이 기간 내에 데이터를 사용할 수 있는지 확인해야 합니다. 이는 삭제 정책이 아닙니다. TTL 기간 직후에는 데이터가 삭제되지 않을 수 있습니다.
- 스트림 지속성. 스트림을 파일 시스템에 저장하여 코어 재시작 시 데이터를 유지하거나 메모리에 스트림을 저장하도록 선택합니다.
- 시작 시퀀스 번호 내보내기에서 시작 메시지로 사용할 메시지의 시퀀스 번호를 지정합니다.

`MessageStreamDefinition`에 대한 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하세요.

- Java SDK의 MessageStreamDefinition
- Node.js SDK의 MessageStreamDefinition
- Python SDK의 MessageStreamDefinition

> **Note**
>
> `StreamManagerClient`는 스트림을 HTTP 서버로 내보내는 데 사용할 수 있는 대상을 제공합니다. 이 대상은 테스트 목적으로만 사용됩니다. 이 대상은 안정적이지 않으며 프로덕션 환경에서 사용할 수 없습니다.

스트림이 생성되면 Lambda 함수는 스트림에 메시지를 추가하여 내보내기용 데이터를 보내고 로컬 처리를 위해 스트림에서 메시지를 읽을 수 있습니다. 생성하는 스트림 수는 하드웨어 기능 및 비즈니스 사례에 따라 다릅니다. 한 가지 전략은 AWS IoT Analytics 또는 Kinesis 데이터 스트림의 각 대상 채널에 대해 스트림을 생성하는 것입니다(하나의 스트림에 대해 여러 개의 대상을 정의할 수 있음). 스트림은 안정적인 수명을 가지고 있습니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

> **Note**
>
> AWS IoT SiteWise 또는 Amazon S3 내보내기 대상으로 스트림을 생성하려면 다음 요구 사항이 있습니다.
>
> - 최소 AWS IoT Greengrass 코어 버전: 1.11.0
> - 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

예시

다음 코드 조각은 `StreamName`(이)라는 스트림을 생성합니다. 이는 `MessageStreamDefinition` 및 하위 데이터 유형의 스트림 속성을 정의합니다.

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",  # Required.
        max_size=268435456,  # Default is 256 MB.
        stream_segment_size=16777216,  # Default is 16 MB.
        time_to_live_millis=None,  # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,  # Required.
        persistence=Persistence.File,  # Default is File.
```

```
        flush_on_write=False,  # Default is false.
        export_definition=ExportDefinition(  # Optional. Choose where/how the stream
 is exported to the AWS ####.
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: create_message_stream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)  // Default is 256 MB.
                    .withStreamSegmentSize(16777216L)  // Default is 16 MB.
                    .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                    .withPersistence(Persistence.File)  // Default is File.
                    .withFlushOnWrite(false)  // Default is false.
                    .withExportDefinition(  // Optional. Choose where/how the stream
 is exported to the AWS ####.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3TaskExecutor(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
```

```
}
```

Java SDK 참조: createMessageStream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.createMessageStream(
            new MessageStreamDefinition()
                .withName("StreamName") // Required.
                .withMaxSize(268435456)  // Default is 256 MB.
                .withStreamSegmentSize(16777216)  // Default is 16 MB.
                .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                .withPersistence(Persistence.File)  // Default is File.
                .withFlushOnWrite(false)  // Default is false.
                .withExportDefinition(  // Optional. Choose where/how the stream is
 exported to the AWS ####.
                    new ExportDefinition()
                        .withKinesis(null)
                        .withIotAnalytics(null)
                        .withIotSitewise(null)
                        .withS3TaskExecutor(null)
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: createMessageStream | MessageStreamDefinition

내보내기 대상 구성에 대한 자세한 내용은 the section called “지원되는 AWS 클라우드 대상의 구성 내보내기” 단원을 참조하세요.

메시지 추가

내보내기를 위해 스트림 관리자로 데이터를 전송하려면 Lambda 함수가 데이터를 대상 스트림에 추가합니다. 내보내기 대상은 이 메소드에 전달할 데이터 유형을 결정합니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

> **Note**
>
> AWS IoT SiteWise 또는 Amazon S3 내보내기 대상과 함께 메시지를 첨부하려면 다음 요구 사항이 적용됩니다.
>
> - 최소 AWS IoT Greengrass 코어 버전: 1.11.0
> - 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

예시

AWS IoT Analytics 또는 Kinesis Data Streams 내보내기 대상

다음 코드 조각은 `StreamName`이라는 스트림에 메시지를 추가합니다. AWS IoT Analytics 또는 Kinesis Data Streams 대상의 경우 Lambda 함수는 데이터 덩어리를 추가합니다.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Python

```
client = StreamManagerClient()

try:
```

```
    sequence_number = client.append_message(stream_name="StreamName",
 data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: append_message

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: appendMessage

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const sequenceNumber = await client.appendMessage("StreamName",
 Buffer.from("Arbitrary byte array"));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: appendMessage

AWS IoT SiteWise 내보내기 목적지

다음 코드 조각은 `StreamName`이라는 스트림에 메시지를 추가합니다. AWS IoT SiteWise 대상의 경우, Lambda 함수는 직렬화된 `PutAssetPropertyValueEntry` 객체를 추가합니다. 자세한 내용은 the section called “AWS IoT SiteWise로 내보내기” 섹션을 참조하세요.

> **Note**
>
> AWS IoT SiteWise(으)로 데이터를 보낼 때 데이터는 `BatchPutAssetPropertyValue` 작업의 요구 사항을 충족해야 합니다. 자세한 내용은 AWS IoT SiteWise API 참조의 BatchPutAssetPropertyValue를 참조하세요.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.11.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages. Add some randomness to
 time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
```

```
    sequence_number = client.append_message(stream_name="StreamName",
 data=Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: append_message | PutAssetPropertyValueEntry

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
 in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;

    // IoTSiteWise requires unique timestamps in all messages. Add some randomness
 to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
```

```
    // Properly handle exception.
}
```

Java SDK 참조: appendMessage | PutAssetPropertyValueEntry

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
 defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
 Math.floor(Math.random() - maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);

        const putAssetPropertyValueEntry =  new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: appendMessage | PutAssetPropertyValueEntry

Amazon S3 대상

다음 코드 조각은 `StreamName`이라는 스트림에 내보내기 작업을 추가합니다. Amazon S3 대상의 경우, Lambda 함수는 소스 입력 파일 및 대상 Amazon S3 객체에 대한 정보가 포함된 직렬화된 `S3ExportTaskDefinition` 객체를 추가합니다. 지정된 객체가 없는 경우 Stream Manager가 자동으로 생성합니다. 자세한 내용은 the section called “Amazon S3로 내보내기” 섹션을 참조하세요.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.11.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
 bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
 data=Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: append_message | S3ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
```

```
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: appendMessage | S3ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: appendMessage | S3ExportTaskDefinition

메시지 읽기

스트림의 메시지 읽기

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

예시

다음 코드 조각은 StreamName이라는 스트림에서 메시지를 읽습니다. 읽기 메서드는 읽기를 시작할 시퀀스 번호, 읽기를 수행할 최소 및 최대 숫자, 메시지 읽기 시간 제한을 지정하는 선택적 ReadMessagesOptions 객체를 가져옵니다.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
 the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
 0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is raised. By default, this is 1.
            max_message_count=100,  # Accept up to 100 messages. By default this is
 1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_messsage_count to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: read_messages | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
```

```
    List<Message> messages = client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: readMessages | ReadMessagesOptions

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messages = await client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                // Try to read from sequence number 100 or greater. By default this
 is 0.
                .withDesiredStartSequenceNumber(100)
                // Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is thrown. By default, this is 1.
                .withMinMessageCount(10)
                // Accept up to 100 messages. By default this is 1.
                .withMaxMessageCount(100)
                // Try to wait at most 5 seconds for the minMessageCount to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                .withReadTimeoutMillis(5 * 1000)
```

```
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: readMessages | ReadMessagesOptions

스트림 나열

스트림 매니저에서 스트림 목록을 가져옵니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

예시

다음 코드 조각은 스트림 관리자에서 이름별로 스트림 목록을 가져옵니다.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
```

```
    # Properly handle errors.
```

Python SDK 참조: list_streams

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: listStreams

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const streams = await client.listStreams();
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: listStreams

메시지 스트림 설명

스트림 정의, 크기, 내보내기 상태 등 스트림에 대한 메타데이터를 가져옵니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

예시

다음 코드 조각은 스트림의 정의, 크기 및 내보내기 상태를 포함하여 `StreamName`라는 스트림에 대한 메타데이터를 가져옵니다.

Python

```
client = StreamManagerClient()

try:
    stream_description = client.describe_message_stream(stream_name="StreamName")
    if stream_description.export_statuses[0].error_message:
        # The last export of export destination 0 failed with some error
        # Here is the last sequence number that was successfully exported
        stream_description.export_statuses[0].last_exported_sequence_number

    if (stream_description.storage_status.newest_sequence_number >
            stream_description.export_statuses[0].last_exported_sequence_number):
        pass
        # The end of the stream is ahead of the last exported sequence number
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: describe_message_stream

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
```

```
            description.getExportStatuses().get(0).getLastExportedSequenceNumber();
        }

        if (description.getStorageStatus().getNewestSequenceNumber() >
                description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (StreamManagerException e) {
        // Properly handle exception.
    }
```

Java SDK 참조: describeMessageStream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: describeMessageStream

메시지 스트림 업데이트

기존 스트림의 속성을 업데이트합니다. 스트림이 생성된 후 요구 사항이 변경되면 스트림을 업데이트하는 것이 좋습니다. 예:

- AWS 클라우드 대상에 새 내보내기 구성을 추가합니다.
- 스트림의 최대 크기를 늘려 데이터를 내보내거나 유지하는 방법을 변경합니다. 예를 들어 스트림 크기와 전체 설정 전략을 함께 사용하면 스트림 관리자가 데이터를 처리하기 전에 데이터가 삭제되거나 거부될 수 있습니다.
- 예를 들어, 내보내기 작업이 오래 걸리고 업로드 데이터를 할당하려는 경우 내보내기를 일시 중지했다가 다시 시작합니다.

Lambda 함수는 다음과 같은 상위 수준 프로세스에 따라 스트림을 업데이트합니다.

1. 스트림에 대한 설명을 가져옵니다.
2. 해당 `MessageStreamDefinition` 및 하위 객체의 대상 속성을 업데이트합니다.
3. 업데이트된 `MessageStreamDefinition`을(를) 전달합니다. 업데이트된 스트림에 대한 전체 객체 정의를 포함해야 합니다. 정의되지 않은 속성은 기본값으로 되돌아갑니다.

   내보내기에서 시작 메시지로 사용할 메시지의 시퀀스 번호를 지정할 수 있습니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.11.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

예시

다음 코드 조각은 `StreamName`(이)라는 스트림을 업데이트합니다. Kinesis Data Streams로 내보내는 스트림의 여러 속성을 업데이트합니다.

Python

```
client = StreamManagerClient()
```

```
try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=
        [KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: updateMessageStream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
            // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)  // Update TTL to 1 hour.
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the AWS ###
#.
```

```
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: update_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)  // Default is 256 MB. Updating Max Size to
 512 MB.
                .withStreamSegmentSize(33554432)  // Default is 16 MB. Updating
 Segment Size to 32 MB.
                .withTimeToLiveMillis(3600000)  // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)  // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)  // Default is File. Update the
 persistence to Memory
                .withFlushOnWrite(true)  // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the AWS
 ####.
                    messageStreamInfo.definition.exportDefinition
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
```

```
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: updateMessageStream | MessageStreamDefinition

스트림 업데이트에 대한 제약조건

스트림을 업데이트할 때는 다음 제약조건이 적용됩니다. 다음 목록에 명시되어 있지 않는 한 업데이트는 즉시 적용됩니다.

- 스트림의 지속성을 업데이트할 수 없습니다. 이 동작을 변경하려면 스트림을 삭제하고 새 지속성 정책을 정의하는 스트림을 생성합니다.
- 다음 조건에서만 스트림의 최대 크기를 업데이트할 수 있습니다.
  - 최대 크기는 스트림의 현재 크기 이상이어야 합니다. 이 정보를 찾으려면 스트림을 설명하고 반환된 `MessageStreamInfo` 객체의 스토리지 상태를 확인합니다.
  - 최대 크기는 스트림의 세그먼트 크기 이상이어야 합니다.
- 스트림 세그먼트 크기를 스트림의 최대 크기보다 작은 값으로 업데이트할 수 있습니다. 업데이트된 설정은 새 세그먼트에 적용됩니다.
- Time to Live(TTL) 속성 업데이트는 새 추가 작업에 적용됩니다. 이 값을 줄이면 스트림 관리자가 TTL을 초과하는 기존 세그먼트도 삭제할 수 있습니다.
- 전체 속성에 대한 전략 업데이트는 새 추가 작업에 적용됩니다. 가장 오래된 데이터를 덮어쓰도록 전략을 설정하는 경우 스트림 관리자가 새 설정에 따라 기존 세그먼트를 덮어쓸 수도 있습니다.
- flush on write 속성 업데이트는 새 메시지에 적용됩니다.
- 내보내기 구성 업데이트는 새 내보내기에 적용됩니다. 업데이트 요청에는 지원할 모든 내보내기 구성이 포함되어야 합니다. 그렇지 않으면 스트림 관리자가 해당 파일을 삭제합니다.
  - 내보내기 구성을 업데이트할 때 대상 내보내기 구성의 식별자를 지정합니다.
  - 내보내기 구성을 추가하려면 새 내보내기 구성의 고유 식별자를 지정합니다.
  - 내보내기 구성을 삭제하려면 내보내기 구성을 생략합니다.

- 스트림에서 내보내기 구성의 시작 시퀀스 번호를 업데이트하려면 최신 시퀀스 번호보다 작은 값을 지정해야 합니다. 이 정보를 찾으려면 스트림을 설명하고 반환된 `MessageStreamInfo` 객체의 스토리지 상태를 확인합니다.

메시지 스트림 삭제

스트림을 삭제합니다. 스트림을 삭제하면 스트림에 저장된 모든 데이터가 디스크에서 삭제됩니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 AWS IoT Greengrass 코어 버전: 1.10.0
- 최소 AWS IoT Greengrass 코어 SDK 버전: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

예시

다음 코드 조각은 `StreamName`라는 스트림을 삭제합니다.

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: deleteMessageStream

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
```

```
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: delete_message_stream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.deleteMessageStream("StreamName");
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

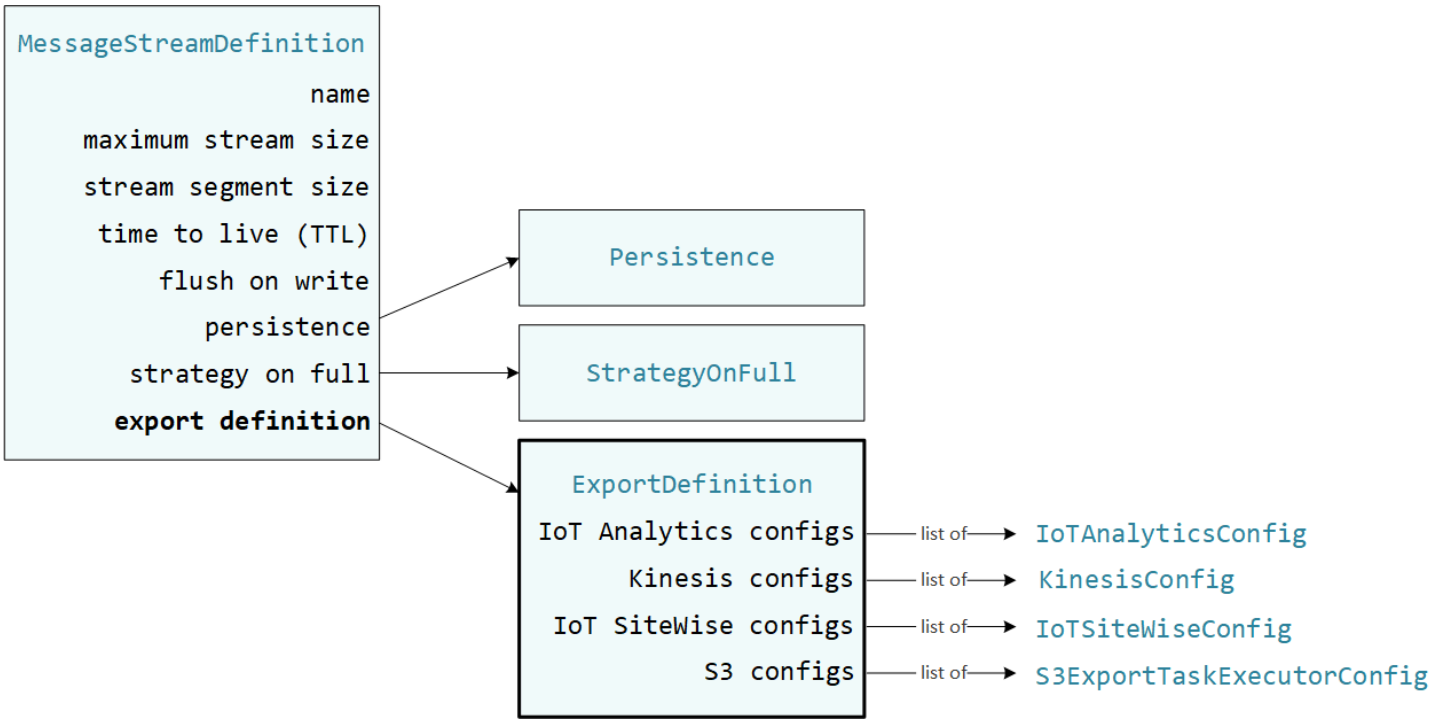
Node.js SDK 참조: deleteMessageStream

다음 사항도 참조하세요.

- 데이터 스트림 관리
- the section called “스트림 관리자 구성”
- the section called “지원되는 AWS 클라우드 대상의 구성 내보내기”
- the section called “데이터 스트림 내보내기(콘솔)”
- the section called “데이터 스트림 내보내기(CLI)”
- AWS IoT Greengrass 코어 SDK 참조의 `StreamManagerClient`은 다음과 같습니다.
  - Python
  - Java
  - Node.js

지원되는 AWS 클라우드 대상의 구성 내보내기

사용자 정의 Lambda 함수는 AWS IoT Greengrass 코어 SDK의 `StreamManagerClient`를 사용하여 스트림 관리자와 상호 작용합니다. Lambda 함수는 스트림을 생성하거나 스트림을 업데이트할 때, 내보내기 정의를 포함하여 스트림 속성을 나타내는 `MessageStreamDefinition` 객체를 전달합니다. `ExportDefinition` 객체에는 스트림에 대해 정의된 내보내기 구성이 포함됩니다. 스트림 관리자는 이러한 내보내기 구성을 사용하여 스트림을 내보내는 위치와 방법을 결정합니다.



단일 대상 유형에 대한 여러 내보내기 구성을 포함하여 스트림에 0개 이상의 내보내기 구성을 정의할 수 있습니다. 예를 들어 하나의 스트림을 2개의 AWS IoT Analytics 채널과 하나의 Kinesis 데이터 스트림으로 내보낼 수 있습니다.

내보내기 시도가 실패한 경우 스트림 관리자는 최대 5분 간격으로 계속해서 데이터를 AWS 클라우드로 다시 내보내려고 시도합니다. 재시도 횟수는 최대 한도가 없습니다.

> **Note**
>
> `StreamManagerClient`는 스트림을 HTTP 서버로 내보내는 데 사용할 수 있는 대상을 제공합니다. 이 대상은 테스트 목적으로만 사용됩니다. 이 대상은 안정적이지 않으며 프로덕션 환경에서 사용할 수 없습니다.

지원되는 AWS 클라우드 대상

- AWS IoT Analytics 채널
- Amazon Kinesis Data Streams
- AWS IoT SiteWise 자산 속성
- Amazon S3 객체

이러한 AWS 클라우드 리소스를 관리할 책임은 사용자에게 있습니다.

AWS IoT Analytics 채널

스트림 관리자는 AWS IoT Analytics로 자동 내보내기를 지원합니다. AWS IoT Analytics를 이용해 데이터에 대한 고급 분석을 수행하여 비즈니스 결정을 내리고 기계 학습 모델을 개선할 수 있습니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 AWS IoT Analytics란 무엇인가요?를 참조하세요.

AWS IoT Greengrass Core SDK에서 Lambda 함수는 `IoTAnalyticsConfig`를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python SDK의 IoTAnalyticsConfig
- 자바 SDK의 IoTAnalyticsConfig
- Node.js SDK의 IoTAnalyticsConfig

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- AWS IoT Analytics의 대상 채널은 Greengrass 그룹과 동일한 AWS 계정 및 AWS 리전에 있어야 합니다.
- the section called “Greengrass 그룹 역할”은 채널을 타겟팅할 수 있는 `iotanalytics:BatchPutMessage` 권한을 허용해야 합니다. 예:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
```

```
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
    ]
}
```

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

AWS IoT Analytics로 내보내기

AWS IoT Analytics로 내보내는 스트림을 생성하기 위해 Lambda 함수는 하나 이상의 `IoTAnalyticsConfig` 객체를 포함하는 내보내기 정의를 가지는 스트림을 생성합니다. 이 객체는 대상 채널, 배치 크기, 배치 간격, 우선 순위와 같은 내보내기 설정을 정의합니다.

Lambda 함수는 디바이스로부터 데이터를 수신할 때 대상 스트림에 데이터 묶음이 포함된 메시지를 추가합니다.

그런 다음 스트림 관리자는 스트림의 내보내기 구성에 정의된 배치 설정 및 우선 순위에 따라 데이터를 내보냅니다.

Amazon Kinesis Data Streams

스트림 관리자는 Amazon Kinesis Data Streams로의 자동 내보내기를 지원합니다. Kinesis Data Streams는 일반적으로 대용량 데이터를 종합하여 데이터 웨어하우스나 map-reduce 클러스터에 로드하는 데 사용됩니다. 자세한 내용은 Amazon Kinesis 개발자 안내서의 Amazon Kinesis Data Streams이란 무엇입니까?를 참조하세요.

AWS IoT Greengrass Core SDK에서 Lambda 함수는 `KinesisConfig`를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python SDK의 KinesisConfig
- 자바 SDK의 KinesisConfig
- Node.js SDK의 KinesisConfig

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- Kinesis Data Streams의 대상 스트림은 Greengrass 그룹과 동일한 AWS 계정 및 AWS 리전에 있어야 합니다.
- the section called “Greengrass 그룹 역할”은 데이터 스트림을 타겟팅할 수 있는 `kinesis:PutRecords` 권한을 허용해야 합니다. 예:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/stream_1_name",
                "arn:aws:kinesis:region:account-id:stream/stream_2_name"
            ]
        }
    ]
}
```

  리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Kinesis Data Streams로 내보내기

Kinesis Data Streams로 내보내는 스트림을 생성하기 위해 Lambda 함수는 하나 이상의 `KinesisConfig` 객체를 포함하는 내보내기 정의를 가진 스트림을 생성합니다. 이 객체는 대상 데이터 스트림, 배치 크기, 배치 간격, 우선 순위와 같은 내보내기 설정을 정의합니다.

Lambda 함수는 디바이스로부터 데이터를 수신할 때 대상 스트림에 데이터 묶음이 포함된 메시지를 추가합니다. 그런 다음 스트림 관리자는 스트림의 내보내기 구성에 정의된 배치 설정 및 우선 순위에 따라 데이터를 내보냅니다.

스트림 관리자는 Amazon Kinesis에 업로드된 각 레코드의 파티션 키로 고유한 임의 UUID를 생성합니다.

AWS IoT SiteWise 자산 속성

스트림 관리자는 AWS IoT SiteWise로 자동 내보내기를 지원합니다. AWS IoT SiteWise는 대규모로 산업 장비 데이터를 수집, 구성 및 분석할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 AWS IoT SiteWise란 무엇인가요?를 참조하세요.

AWS IoT Greengrass Core SDK에서 Lambda 함수는 `IoTSiteWiseConfig`를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python SDK에서의 IoTSiteWiseConfig
- Java SDK의 IoTSiteWiseConfig
- Node.js SDK의 IoTSiteWiseConfig

> **Note**
>
> 또한 AWS는 OPC-UA 소스와 함께 사용할 수 있는 사전 구축된 솔루션인 the section called "IoT SiteWise"도 제공합니다.

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- AWS IoT SiteWise의 대상 자산 속성은 Greengrass 그룹과 동일한 AWS 계정 및 AWS 리전에 있어야 합니다.

  > **Note**
  >
  > AWS IoT SiteWise가 지원하는 리전 목록은 AWS 일반 참조의 AWS IoT SiteWise 엔드포인트 및 할당량을 참조하세요.

- the section called "Greengrass 그룹 역할"은 대상 자산 속성에 `iotsitewise:BatchPutAssetPropertyValue` 권한을 허용해야 합니다. 다음 예제의 정책은 `iotsitewise:assetHierarchyPath` 조건 키를 사용하여 대상 루트 자산과 그 하위 자산에 대한 액세스 권한을 부여합니다. 정책에서 `Condition`을 제거하여 모든 AWS IoT SiteWise 자산에 대한 액세스를 허용하거나 개별 자산의 ARN을 지정할 수 있습니다.

  ```
  {
  ```

```
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

중요한 보안 정보는 AWS IoT SiteWise 사용 설명서의 BatchPutAssetPropertyValue 권한 부여를 참조하십시오.

AWS IoT SiteWise로 내보내기

AWS IoT SiteWise로 내보내는 스트림을 생성하기 위해 Lambda 함수는 하나 이상의 `IoTSiteWiseConfig` 객체를 포함하는 내보내기 정의를 가지는 스트림을 생성합니다. 이 객체는 배치 크기, 배치 간격 및 우선 순위와 같은 내보내기 설정을 정의합니다.

Lambda 함수는 디바이스로부터 데이터를 수신할 때 대상 스트림에 데이터를 포함하는 메시지를 추가합니다. 메시지는 하나 이상의 자산 속성에 대한 속성 값을 포함하는 JSON 직렬화 `PutAssetPropertyValueEntry` 객체입니다. 자세한 내용은 AWS IoT SiteWise 내보내기 대상에 대한 메시지 추가를 참조하십시오.

> **Note**
>
> AWS IoT SiteWise로 데이터를 보낼 때 데이터는 `BatchPutAssetPropertyValue` 작업의 요구 사항을 충족해야 합니다. 자세한 내용은 AWS IoT SiteWise API 참조의 BatchPutAssetPropertyValue를 참조하세요.

그런 다음 스트림 관리자는 스트림의 내보내기 구성에 정의된 배치 설정 및 우선 순위에 따라 데이터를 내보냅니다.

스트림 관리자 설정 및 Lambda 함수 로직을 조정하여 내보내기 전략을 설계할 수 있습니다. 예:

- 거의 실시간으로 내보내려면 배치 크기 및 간격을 낮게 설정하고 스트림이 수신되면 스트림에 데이터를 추가하십시오.
- 배치 처리를 최적화하고, 대역폭 제약을 완화하거나, 비용을 최소화하기 위해 Lambda 함수는 스트림에 데이터를 추가하기 전에 단일 자산 속성에 대해 수신된 타임스탬프-품질-값(TQV) 데이터 포인트를 풀링할 수 있습니다. 한 가지 전략은 동일한 속성에 대해 둘 이상의 항목을 보내는 대신 최대 10개의 서로 다른 속성-자산 조합 또는 속성 별칭에 대한 항목을 하나의 메시지로 일괄 처리하는 것입니다. 이렇게 하면 스트림 관리자가 AWS IoT SiteWise 할당량 이내로 유지할 수 있습니다.

Amazon S3 객체

스트림 관리자는 Amazon S3로의 자동 내보내기를 지원합니다. Amazon S3를 사용하여 대량의 데이터를 저장 및 검색할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 Amazon S3란 무엇인가요?를 참조하세요.

AWS IoT Greengrass Core SDK에서 Lambda 함수는 `S3ExportTaskExecutorConfig`를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python SDK의 S3ExportTaskExecutorConfig
- Java SDK의 S3ExportTaskExecutorConfig
- Node.js SDK의 S3ExportTaskExecutorConfig

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 대상 Amazon S3 버킷은 Greengrass 그룹과 동일한 AWS 계정에 있어야 합니다.
- Greengrass 그룹의 기본 컨테이너화가 Greengrass 컨테이너인 경우, STREAM_MANAGER_READ_ONLY_DIRS 파라미터를 설정하여 `/tmp` 아래에 있거나 루트 파일 시스템에 있지 않은 입력 파일 디렉터리를 사용해야 합니다.

- Greengrass 컨테이너 모드에서 실행되는 Lambda 함수가 입력 파일 디렉터리에 입력 파일을 쓰는 경우, 디렉터리에 대한 로컬 볼륨 리소스를 생성하고 쓰기 권한을 사용하여 디렉터리를 컨테이너에 마운트해야 합니다. 이렇게 하면 파일이 루트 파일 시스템에 기록되고 컨테이너 외부에서 볼 수 있습니다. 자세한 내용은 로컬 리소스에 액세스 섹션을 참조하세요.
- the section called “Greengrass 그룹 역할”은 대상 버킷에 다음 권한을 허용해야 합니다. 예:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-1-name/*",
                "arn:aws:s3:::bucket-2-name/*"
            ]
        }
    ]
}
```

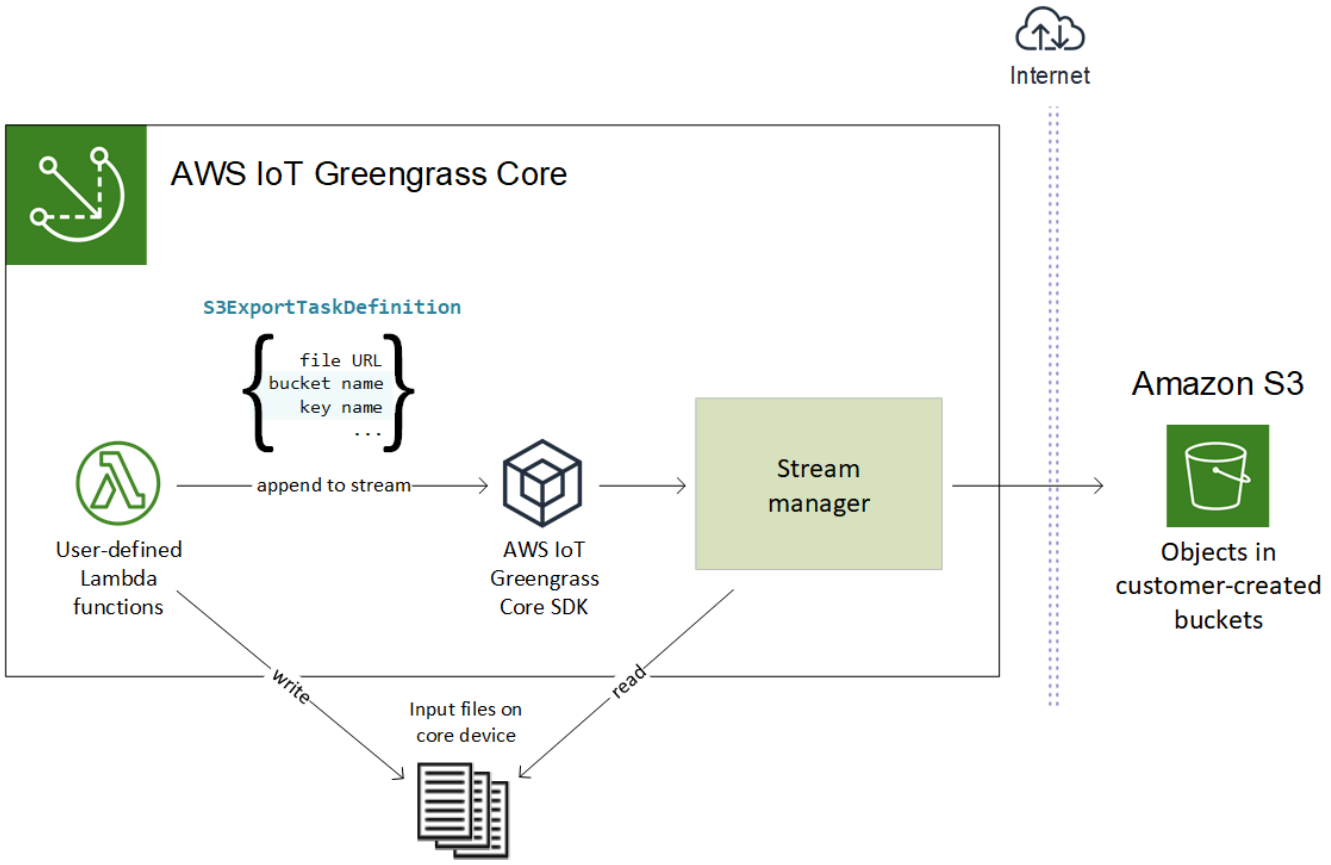
  리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Amazon S3로 내보내기

Amazon S3로 내보내는 스트림을 생성하기 위해, Lambda 함수는 객체를 사용하여 `S3ExportTaskExecutorConfig` 내보내기 정책을 구성합니다. 정책은 멀티파트 업로드 임계값 및 우선 순위와 같은 내보내기 설정을 정의합니다. Amazon S3 내보내기의 경우 스트림 관리자는 코어 디바이스의 로컬 파일에서 읽은 데이터를 업로드합니다. 업로드를 시작하기 위해 Lambda 함수는 대상 스트림에 내보내기 작업을 추가합니다. 내보내기 작업에는 입력 파일 및 대상 Amazon S3 객체에 대한 정보가 포함됩니다. 스트림 관리자는 스트림에 추가된 순서대로 작업을 실행합니다.

> **Note**
>
> 대상 버킷은 AWS 계정 내에 이미 있어야 합니다. 지정된 키의 객체가 존재하지 않는 경우 스트림 관리자가 대신 객체를 생성합니다.

이러한 대략적 워크플로우는 다음 다이어그램에 나와 있습니다.



스트림 관리자는 멀티파트 업로드 임계값 속성, 최소 파트 크기 설정 및 입력 파일 크기를 사용하여 데이터 업로드 방법을 결정합니다. 멀티파트 업로드 임계값은 최소 파트 크기보다 크거나 이와 동일해야 합니다. 데이터를 병렬로 업로드하려는 경우 여러 스트림을 생성할 수 있습니다.

대상 Amazon S3 객체를 지정하는 키는 `!{timestamp:`*`value`*`}` 자리표시자에 유효한 Java DateTimeFormatter 문자열을 포함할 수 있습니다. 이러한 타임스탬프 자리표시자를 사용하여 입력 파일 데이터가 업로드된 시간을 기준으로 Amazon S3의 데이터를 분할할 수 있습니다. 예를 들어, 다음 키 이름은 `my-key/2020/12/31/data.txt`와 같은 값으로 해석됩니다.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> 스트림의 내보내기 상태를 모니터링하려면 먼저 상태 스트림을 만든 다음 이를 사용하도록 내보내기 스트림을 구성하십시오. 자세한 내용은 the section called “내보내기 작업 모니터링” 섹션을 참조하세요.

입력 데이터 관리

IoT 애플리케이션이 입력 데이터의 라이프사이클을 관리하는 데 사용하는 코드를 작성할 수 있습니다. 다음 예제 워크플로는 Lambda 함수를 사용하여 이 데이터를 관리하는 방법을 보여줍니다.

1. 로컬 프로세스는 디바이스 또는 주변 기기로부터 데이터를 수신한 다음 코어 디바이스의 디렉터리에 있는 파일에 데이터를 씁니다. 스트림 관리자용 입력 파일입니다.

   > **Note**
   >
   > 입력 파일 디렉터리에 대한 액세스를 구성해야 하는지 확인하려면 STREAM_MANAGER_READ_ONLY_DIRS 파라미터를 참조하십시오.
   > 스트림 관리자가 실행하는 프로세스는 그룹에 대한 기본 액세스 ID의 모든 파일 시스템 권한을 상속합니다. 스트림 관리자는 입력 파일에 액세스할 수 있는 권한이 있어야 합니다. `chmod(1)` 명령을 사용해 필요한 경우 파일의 권한을 변경할 수 있습니다.

2. Lambda 함수는 디렉토리를 스캔하고 새 파일이 생성될 때 대상 스트림에 내보내기 작업을 추가합니다. 작업은 입력 파일의 URL, 대상 Amazon S3 버킷 및 키, 선택적 사용자 메타데이터를 지정하는 JSON 직렬화 `S3ExportTaskDefinition` 객체입니다.
3. 스트림 관리자는 입력 파일을 읽고 추가된 작업의 순서대로 Amazon S3로 데이터를 내보냅니다. 대상 버킷은 AWS 계정 내에 이미 있어야 합니다. 지정된 키의 객체가 존재하지 않는 경우 스트림 관리자가 대신 객체를 생성합니다.
4. Lambda 함수는 상태 스트림에서 메시지를 읽어 내보내기 상태를 모니터링합니다. 내보내기 작업이 완료된 후 Lambda 함수는 해당 입력 파일을 삭제할 수 있습니다. 자세한 내용은 the section called “내보내기 작업 모니터링” 섹션을 참조하세요.

내보내기 작업 모니터링

IoT 애플리케이션이 Amazon S3 내보내기 상태를 모니터링하는 데 사용하는 코드를 작성할 수 있습니다. Lambda 함수는 상태 스트림을 생성한 다음 상태 스트림에 상태 업데이트를 기록하도록 내보내기

스트림을 구성해야 합니다. 단일 상태 스트림은 Amazon S3로 내보내는 여러 스트림으로부터 상태 업데이트를 받을 수 있습니다.

먼저 상태 스트림으로 사용할 스트림을 생성합니다. 스트림의 크기 및 보존 정책을 구성하여 상태 메시지의 수명을 제어할 수 있습니다. 예:

- 상태 메시지를 저장하지 않으려는 경우 `Persistence`를 `Memory`로 설정합니다.
- 새 상태 메시지가 손실되지 않도록 `StrategyOnFull`을 `OverwriteOldestData`로 설정합니다.

그런 다음, 상태 스트림을 사용하도록 내보내기 스트림을 생성하거나 업데이트하십시오. 특히 스트림의 `S3ExportTaskExecutorConfig` 내보내기 구성의 상태 구성 속성을 설정하십시오. 그러면 스트림 관리자에게 내보내기 작업에 대한 상태 메시지를 상태 스트림에 기록하도록 지시합니다. `StatusConfig` 객체에서 상태 스트림의 이름과 상세 수준을 지정합니다. 지원되는 다음 값의 범위는 최소 세부 정보 표시(`ERROR`)에서 최대 세부 정보 표시(`TRACE`)까지입니다. 기본값은 `INFO`입니다.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

다음 예제 워크플로우는 Lambda 함수가 상태 스트림을 사용하여 내보내기 상태를 모니터링하는 방법을 보여줍니다.

1. 이전 워크플로우에서 설명한 것처럼 Lambda 함수는 내보내기 작업에 대한 상태 메시지를 상태 스트림에 기록하도록 구성된 스트림에 내보내기 작업을 추가합니다. 추가 작업은 작업 ID를 나타내는 시퀀스 번호를 반환합니다.
2. Lambda 함수는 상태 스트림에서 메시지를 순차적으로 읽은 다음 스트림 이름과 작업 ID 또는 메시지 컨텍스트의 내보내기 작업 속성을 기반으로 메시지를 필터링합니다. 예를 들어, Lambda 함수는 메시지 컨텍스트에서 `S3ExportTaskDefinition` 객체로 표시되는 내보내기 작업의 입력 파일 URL을 기준으로 필터링할 수 있습니다.

   다음 상태 코드는 내보내기 작업이 완료됨 상태에 도달했음을 나타냅니다.

   - `Success`. 업로드가 성공적으로 완료되었습니다.

- `Failure`. 스트림 관리자에서 오류가 발생했습니다. 예를 들어, 지정된 버킷이 존재하지 않습니다. 문제를 해결한 후 내보내기 작업을 스트림에 다시 추가할 수 있습니다.
- `Canceled`. 스트림 또는 내보내기 정의가 삭제되었거나 작업의 TTL(Time-to-Live) 기간이 만료되어 작업이 중단되었습니다.

> **Note**
>
> 작업 상태가 `InProgress` 또는 `Warning` 상태를 가질 수도 있습니다. 이벤트가 작업 실행에 영향을 주지 않는 오류를 반환하면 스트림 관리자에서 경고를 발행합니다. 예를 들어 중단된 부분 업로드를 정리하지 못하면 경고가 반환됩니다.

3. 내보내기 작업이 완료된 후 Lambda 함수는 해당 입력 파일을 삭제할 수 있습니다.

다음 예제는 Lambda 함수가 상태 메시지를 읽고 처리하는 방법을 보여줍니다.

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from greengrasssdk.stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
```

```
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 참조: read_messages | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
```

```
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
 ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
 ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
 StatusMessage.class);
                    // Check the status of the status message. If the status is
 "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
                        System.out.println("Successfully uploaded file at path " +
 FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                     } else if (Status.Failure.equals(statusMessage.getStatus()) ||
 Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
 path %s to S3. Message %s",

 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
```

```
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
```

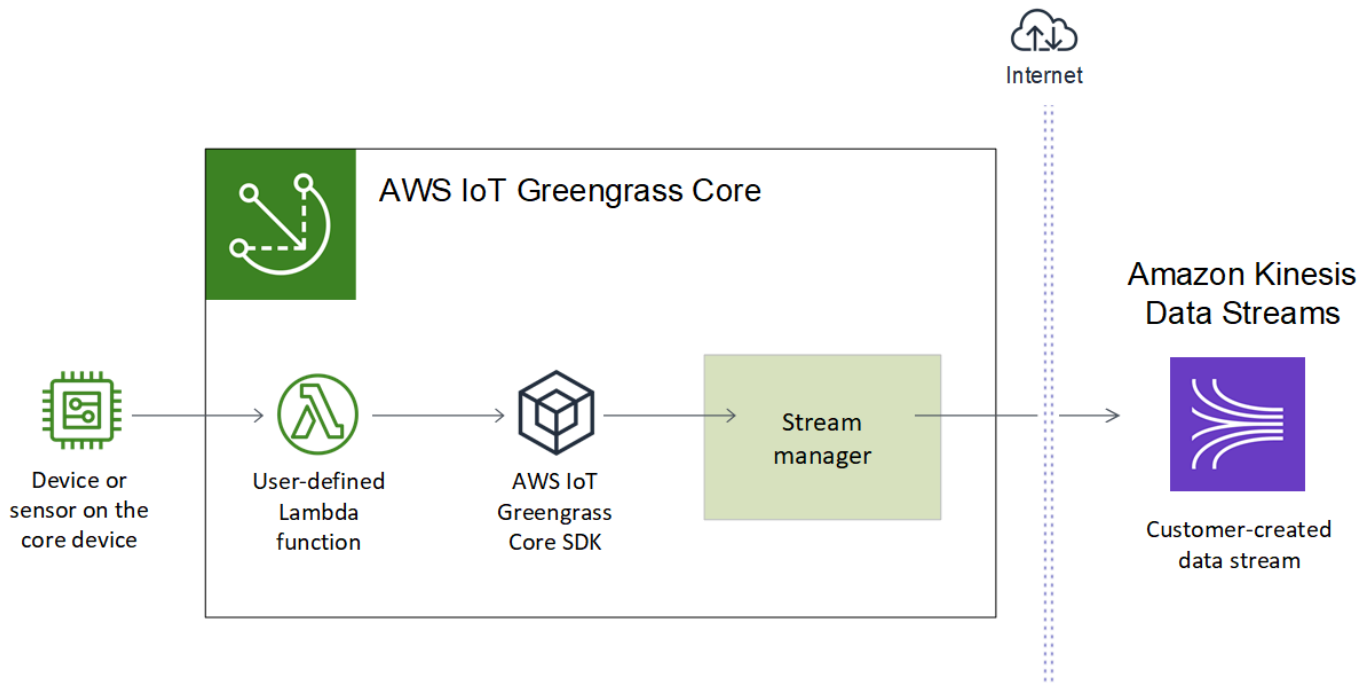
```
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
            }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: readMessages | StatusMessage

AWS 클라우드 클라우드로 데이터 스트림 내보내기(콘솔)

이 튜토리얼에서는 AWS IoT 콘솔을 사용하여 스트림 관리자가 활성화된 상태에서 AWS IoT Greengrass 그룹을 구성하고 배포하는 방법을 보여 줍니다. 이 그룹에는 스트림 관리자의 스트림에 기록하는 사용자 정의 Lambda 함수가 포함되어 있으며, 이 함수는 자동으로 AWS 클라우드 클라우드로 내보내집니다.

스트림 관리자를 사용하면 대용량 데이터 스트림을 보다 쉽고 안정적으로 수집, 처리 및 내보낼 수 있습니다. 이 튜토리얼에서는 IoT 데이터를 사용하는 `TransferStream` Lambda 함수를 생성합니다. Lambda 함수는 AWS IoT Greengrass Core SDK를 사용하여 스트림 관리자에서 스트림을 생성한 다음, 읽기 및 쓰기 작업을 수행합니다. 그러면 스트림 관리자가 Kinesis Data Streams로 스트림을 내보냅니다. 다음 다이어그램은 이 워크플로를 보여 줍니다.



이 튜토리얼은 사용자 정의 Lambda 함수가 AWS IoT Greengrass Core SDK의 `StreamManagerClient` 객체를 사용하여 스트림 관리자와 상호 작용하는 방법을 보여 줍니다. 간편성을 위해 이 튜토리얼에서 생성한 Python Lambda 함수는 시뮬레이션된 디바이스 데이터를 생성합니다.

필수 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- Greengrass 그룹 및 Greengrass 코어(v1.10 이상). Greengrass 그룹 및 코어를 생성하는 방법에 대한 자세한 내용은 시작하기: AWS IoT Greengrass 섹션을 참조하세요. 시작하기 자습서에는 AWS IoT Greengrass 코어 소프트웨어를 설치하는 단계도 포함되어 있습니다.

  > **Note**
  >
  > 스트림 관리자는 OpenWrt 배포에서 지원되지 않습니다.

- 코어 디바이스에 설치된 Java 8 런타임(JDK 8).
  - Debian 기반 배포판(Raspbian 포함) 또는 Ubuntu 기반 배포판의 경우 다음 명령을 실행합니다.

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Red Hat 기반 배포판(Amazon Linux 포함) 의 경우 다음 명령을 실행합니다.

```
sudo yum install java-1.8.0-openjdk
```

자세한 내용은 OpenJDK 설명서의 OpenJDK 패키지 다운로드 및 설치 방법을 참조하십시오.

- AWS IoT Greengrass Core SDK for Python v1.5.0 이상. AWS IoT Greengrass Core SDK for Python에서 `StreamManagerClient`를 사용하려면 다음을 수행해야 합니다.
  - Python 3.7 이상을 코어 디바이스에 설치합니다.
  - Lambda 함수 배포 패키지에 SDK와 그 종속성을 포함시킵니다. 지침은 이 튜토리얼에 나와 있습니다.

> **Tip**
>
> Java 또는 NodeJS로 `StreamManagerClient`를 사용할 수 있습니다. 예제 코드는 AWS IoT Greengrass Core SDK for Java 및 AWS IoT Greengrass Core SDK for Node.js를 참조하십시오.

- Greengrass 그룹과 동일한 AWS 리전에 Amazon Kinesis Data Streams에서 생성된 **`MyKinesisStream`**이라는 대상 스트림입니다. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서에서 스트림 생성을 참조하십시오.

> **Note**
>
> 이 튜토리얼에서는 스트림 관리자가 데이터를 Kinesis Data Streams로 내보내고 이로 인해 AWS 계정에 요금이 청구됩니다. 요금 정보는 Kinesis Data Streams 요금을 참조하십시오. 요금이 부과되지 않도록 하기 위해 Kinesis 데이터 스트림을 생성하지 않고 이 튜토리얼을 실행할 수 있습니다. 이 경우 로그를 확인하여 스트림 관리자가 스트림을 Kinesis Data Streams로 내보내려고 시도했는지 확인합니다.

- 다음 예제에 나오는 것처럼 대상 전송 스트림에 대한 `kinesis:PutRecords` 작업을 허용하는 the section called “Greengrass 그룹 역할”에 추가된 IAM 정책입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
```

```
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Lambda 함수 배포 패키지 생성
2. Lambda 함수 생성
3. 그룹에 함수 추가
4. 스트림 관리자 활성화
5. 로컬 로깅 구성
6. 그룹 배포
7. 애플리케이션 테스트

이 자습서를 완료하는 데 약 20분 정도 걸립니다.

1단계: Lambda 함수 배포 패키지 생성

이 단계에서는 Python 함수 코드와 종속성이 포함된 Lambda 함수 배포 패키지를 생성합니다. 나중에 AWS Lambda에서 Lambda 함수를 생성할 때 이 패키지를 업로드합니다. Lambda 함수는 AWS IoT Greengrass Core SDK를 사용하여 로컬 스트림을 생성하고 상호 작용합니다.

> **Note**
>
> 사용자 정의 Lambda 함수는 AWS IoT Greengrass Core SDK를 사용하여 스트림 관리자와 상호 작용해야 합니다. Greengrass 스트림 관리자의 요구 사항에 대한 자세한 내용은 Greengrass 스트림 관리자 요구 사항을 참조하십시오.

1. AWS IoT Greengrass Core SDK for Python v1.5.0 이상을 다운로드하십시오.
2. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
3. Lambda 함수 배포 패키지에 SDK와 함께 포함시킬 패키지 종속성을 설치합니다.

1. `requirements.txt` 파일이 포함된 SDK 디렉터리로 이동합니다. 이 파일은 종속성을 나열합니다.
2. SDK 종속성을 설치합니다. 예를 들어 다음 `pip` 명령을 실행하여 현재 디렉터리에 설치합니다.

```
pip install --target . -r requirements.txt
```

4. `transfer_stream.py`이라는 로컬 파일에 다음과 같은 Python 코드 함수를 저장합니다.

> **Tip**
>
> Java 및 NodeJS를 사용하는 예제 코드는 GitHub의 AWS IoT Greengrass Core SDK for Java 및 AWS IoT Greengrass Core SDK for Node.js를 참조하십시오.

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
```

```
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
 kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
 strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
```

```
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
 ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
 to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. 다음 항목을 `transfer_stream_python.zip`라는 파일로 압축합니다. 이것이 Lambda 함수 배포 패키지입니다.

   - transfer_stream.py. 앱 로직.
   - greengrasssdk. MQTT 메시지를 게시하는 Python Greengrass Lambda 함수에 대한 필수 라이브러리입니다.

     스트림 관리자 작업은 AWS IoT Greengrass Core SDK for Python 버전 1.5.0 이상에서 사용할 수 있습니다.
   - AWS IoT Greengrass Core SDK for Python에 대해 설치한 종속성(예: `cbor2` 디렉터리).

   `zip` 파일을 생성할 때 이러한 항목만 포함시키고 포함 폴더는 포함시키지 않습니다.

2단계: Lambda 함수 생성

이 단계에서는 AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성한 후 배포 패키지를 사용하도록 구성합니다. 그런 다음 함수 버전을 게시하고 별칭을 생성합니다.

1. 먼저, Lambda 함수를 생성합니다.

   a. AWS Management Console에서 [Services]를 선택한 다음 AWS Lambda 콘솔을 엽니다.

   b. 함수 생성을 선택한 다음 새로 작성을 선택합니다.

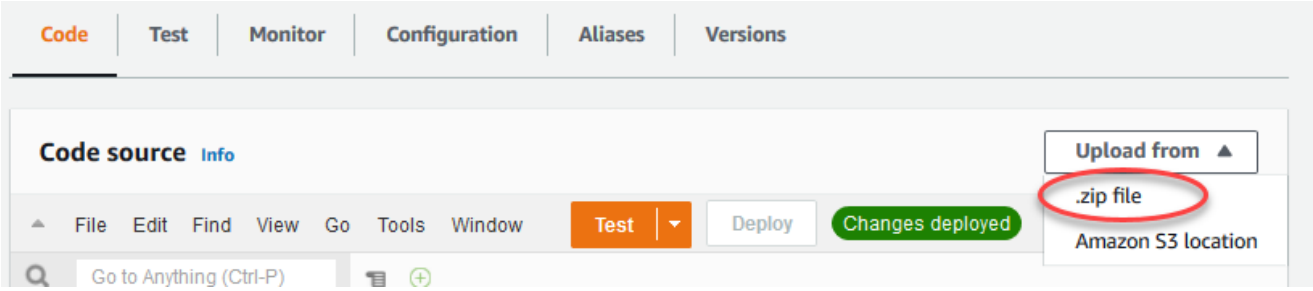
   c. 기본 정보 섹션에서 다음 값을 사용합니다.

      - [함수 이름]에 **`TransferStream`**을 입력합니다.
      - 실행 시간에서 Python 3.7을 선택합니다.
      - 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.

   d. 페이지 하단에서 함수 생성을 선택합니다.

2. 이제 핸들러를 등록하고 Lambda 함수 배포 패키지를 업로드합니다.

   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



   b. 업로드를 선택한 다음 `transfer_stream_python.zip` 배포 패키지를 선택합니다. 그런 다음 저장을 선택합니다.

   c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

      - 실행 시간에서 Python 3.7을 선택합니다.
      - 핸들러에 **`transfer_stream.function_handler`**를 입력합니다.

   d. Save를 선택합니다.

> **Note**
>
> AWS Lambda 콘솔의 테스트 버튼은 이 함수와 함께 작동하지 않습니다. AWS IoT Greengrass 코어 SDK에는 AWS Lambda 콘솔에서 Greengrass Lambda 함수를 독립적으로 실행하는 데 필요한 모듈이 포함되어 있지 않습니다. 이러한 모듈(예: `greengrass_common`)은 Greengrass 코어에 배포된 후 함수에 제공됩니다.

3. 이제 Lambda 함수의 첫 번째 버전을 게시하고 버전의 별칭을 생성합니다.

> **Note**
>
> Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

   a. [Actions] 메뉴에서 [Publish new revision]을 선택합니다.
   b. 버전 설명에 **`First version`**을 입력한 후 게시를 선택합니다.
   c. TransferStream: 1 구성 페이지의 작업 메뉴에서 별칭 생성을 선택합니다.
   d. [Create a new alias] 페이지에서 다음 값을 사용합니다.
      - 이름에 **`GG_TransferStream`**을 입력합니다.
      - 버전에서 1을 선택합니다.

      > **Note**
      >
      > AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

   e. 생성을 선택합니다.

이제 Greengrass 그룹에 Lambda 함수를 추가할 준비가 되었습니다.

3단계: Greengrass 그룹에 Lambda 함수 추가

이 단계에서 그룹에 Lambda 함수를 추가한 다음 수명 주기와 환경 변수를 구성합니다. 자세한 내용은 the section called “Greengrass Lambda 함수 실행 제어” 섹션을 참조하세요.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
4. 내 Lambda 함수에서 추가를 선택합니다.
5. Lambda 함수 추가 페이지에서 Lambda 함수에 대한 Lambda 함수를 선택합니다.
6. Lambda 버전의 경우 Alias:GG_TransferStream을 선택합니다.

   이제 Greengrass 그룹에서 Lambda 함수의 동작을 결정하는 속성을 구성합니다.
7. Lambda 함수 구성 섹션에서 다음과 같이 변경합니다.

   - 메모리 제한을 32MB로 설정합니다.
   - 고정된 경우 True를 선택합니다.

   > **Note**
   >
   > 수명이 긴(또는 고정) Lambda 함수는 AWS IoT Greengrass가 시작된 후 자동으로 시작되며, 자체 컨테이너에서 계속 실행됩니다. 이는 간접 호출되면 시작되고 실행할 작업이 남아 있지 않으면 중지되는 온디맨드 Lambda 함수와 상반됩니다. 자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.
8. Lambda 함수 추가를 선택합니다.

4단계: 스트림 관리자 활성화

이 단계에서는 스트림 관리자가 활성화되어 있는지 확인합니다.

1. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
2. 시스템 Lambda 함수에서 스트림 관리자를 선택하고 상태를 확인합니다. 비활성화된 경우 편집을 선택합니다. 그런 다음 활성화 및 저장을 선택합니다. 이 튜토리얼의 기본 파라미터 설정을 사용할 수 있습니다. 자세한 내용은 the section called “스트림 관리자 구성” 섹션을 참조하세요.

> **Note**
>
> 콘솔을 사용하여 스트림 관리자를 활성화하고 그룹을 배포하는 경우 스트림 관리자의 메모리 크기가 기본 4GB로 설정됩니다. 메모리 크기를 최소 128,000KB로 설정하는 것이 좋습니다.

5단계: 로컬 로깅 구성

이 단계에서는 로그를 코어 디바이스의 파일 시스템에 쓸 수 있도록 그룹에서 AWS IoT Greengrass 시스템 구성 요소, 사용자 정의 Lambda 함수 및 커넥터를 구성합니다. 로그를 사용하여 발생할 수 있는 문제를 해결할 수 있습니다. 자세한 내용은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요.

1. 로컬 로그 구성에서 로컬 로깅이 구성되어 있는지 확인합니다.
2. Greengrass 시스템 구성 요소 또는 사용자 정의 Lambda 함수에 대해 로그가 구성되지 않은 경우에는 편집을 선택합니다.
3. 사용자 Lambda 함수 로그 수준 및 Greengrass 시스템 로그 수준을 선택합니다.
4. 로깅 수준 및 디스크 공간 제한을 기본값으로 유지한 후 저장을 선택합니다.

6단계: Greengrass 그룹 배포

코어 디바이스에 그룹을 배포합니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 대몬(daemon)이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 `root`에 대한 `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon` 입력이 포함되어 있는 경우에는 대몬(daemon)이 실행 중인 것입니다.

   > **Note**
   >
   > 경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

b. 대몬(daemon)을 시작하려면:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. 그룹 구성 페이지에서 배포를 선택합니다.
3. a. Lambda 함수 탭의 시스템 Lambda 함수 섹션에서 IP 감지기를 선택하고 편집을 선택합니다.
   b. IP 감지기 설정 편집 대화 상자에서 MQTT 브로커 엔드포인트 자동 탐지 및 재정의를 선택합니다.
   c. Save를 선택합니다.

   이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

   > **Note**
   > 메시지가 표시되면 Greengrass 서비스 역할을 생성할 권한을 부여하고 이 권한을 현재 AWS 리전의 AWS 계정과 연결합니다. 이 역할은 AWS IoT Greengrass가 AWS 서비스의 리소스에 액세스할 수 있습니다.

   배포 페이지에 배포 타임스탬프, 버전 ID, 상태가 표시됩니다. 완료되면 배포에 대해 성공적으로 완료했습니다 상태가 표시되어야 합니다.

   문제 해결에 대한 도움말은 문제 해결 섹션을 참조하십시오.

7단계: 애플리케이션 테스트

이 `TransferStream` Lambda 함수는 시뮬레이션된 디바이스 데이터를 생성합니다. 스트림 관리자가 대상 Kinesis 데이터 스트림으로 내보내는 스트림에 데이터를 기록합니다.

1. Amazon Kinesis 콘솔의 Kinesis Data Streams에서 MyKinesisStream을 선택합니다.

   > **Note**
   > 대상 Kinesis 데이터 스트림 없이 튜토리얼을 실행한 경우 스트림 관리자(`GGStreamManager`)의 로그 파일을 확인하십시오. 오류 메시지에 `export stream`

`MyKinesisStream doesn't exist`이 포함되어 있으면 테스트가 성공적입니다. 이 오류는 서비스가 스트림으로의 내보내기를 시도했지만 스트림이 존재하지 않는다는 것을 의미합니다.

2. MyKinesisStream 페이지에서 모니터링을 선택합니다. 테스트가 성공적이면 Put Records(레코드 넣기) 차트에 데이터가 표시되어야 합니다. 연결에 따라 데이터가 표시되기까지 1분 정도 걸릴 수 있습니다.

   Important

   테스트를 마쳤을 때 추가 요금이 발생하지 않도록 Kinesis 데이터 스트림을 삭제합니다. 또는 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다. 이렇게 하면 테스트를 계속할 준비가 될 때까지 코어가 메시지를 보내지 못하게 됩니다.

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd stop
   ```

3. 코어에서 TransferStream Lambda 함수를 제거합니다.

   a. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.

   b. Greengrass 그룹에서 사용자 그룹을 선택합니다.

   c. Lambdas 페이지에서 TransferStream 함수에 줄임표(...)를 선택한 다음 함수 제거를 선택합니다.

   d. 작업에서 배포를 선택합니다.

로깅 정보를 보거나 스트림 문제를 해결하려면 로그에서 `TransferStream` 및 `GGStreamManager` 함수를 확인합니다. 파일 시스템에서 AWS IoT Greengrass 로그를 읽을 수 있는 `root` 권한이 있어야 합니다.

- `TransferStream`은 *greengrass-root*`/ggc/var/log/user/`*region*`/`*account-id*`/TransferStream.log`에 로그 항목을 기록합니다.
- `GGStreamManager`은 *greengrass-root*`/ggc/var/log/system/GGStreamManager.log`에 로그 항목을 기록합니다.

추가 문제 해결 정보가 필요한 경우 사용자 Lambda로그의 로깅 수준을 디버그 로그로 설정한 다음 그룹을 다시 배포할 수 있습니다.

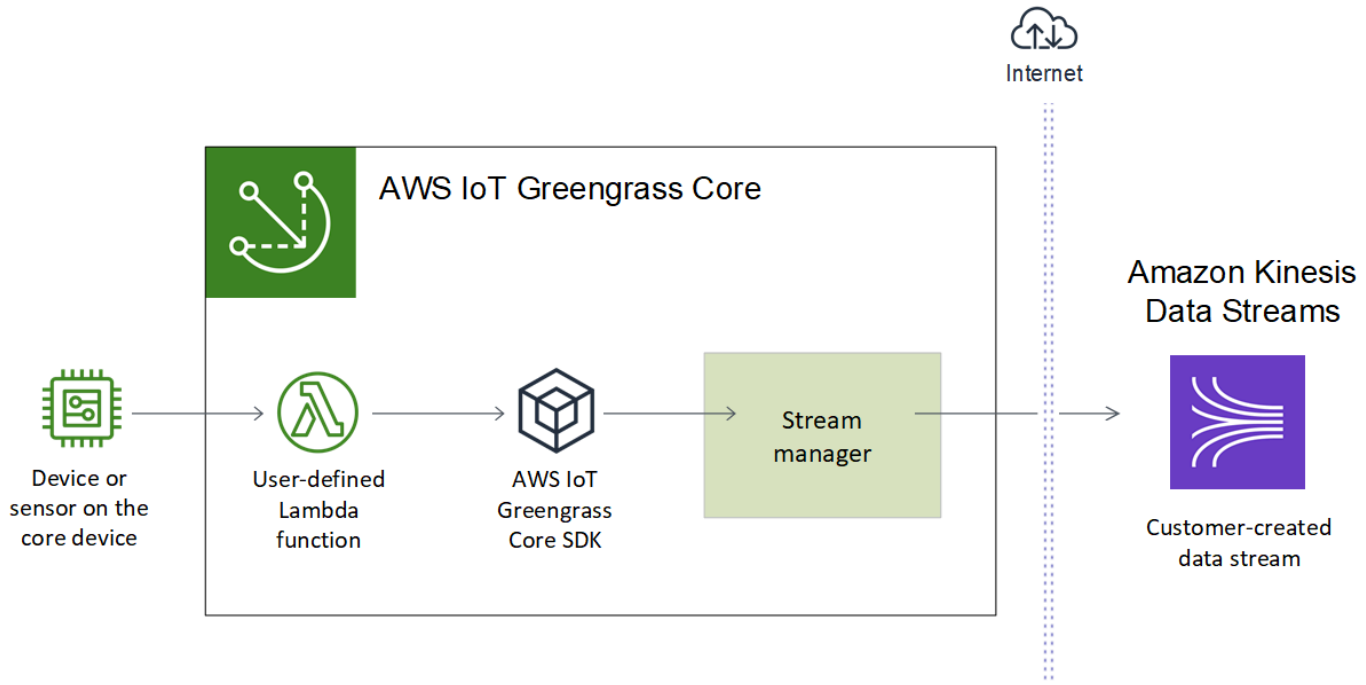
다음 사항도 참조하세요.

- 데이터 스트림 관리
- the section called “스트림 관리자 구성”
- the section called “StreamManagerClient를 사용하여 스트림 작업”
- the section called “지원되는 AWS 클라우드 대상의 구성 내보내기”
- the section called “데이터 스트림 내보내기(CLI)”

AWS 클라우드로 데이터 스트림 내보내기(CLI)

이 튜토리얼에서는 AWS CLI를 사용하여 스트림 관리자가 활성화된 상태에서 AWS IoT Greengrass 그룹을 구성하고 배포하는 방법을 보여 줍니다. 이 그룹에는 스트림 관리자의 스트림에 기록하는 사용자 정의 Lambda 함수가 포함되어 있으며, 이 함수는 자동으로 AWS 클라우드 클라우드로 내보내집니다.

스트림 관리자를 사용하면 대용량 데이터 스트림을 보다 쉽고 안정적으로 수집, 처리 및 내보낼 수 있습니다. 이 튜토리얼에서는 IoT 데이터를 사용하는 `TransferStream` Lambda 함수를 생성합니다. Lambda 함수는 AWS IoT Greengrass Core SDK를 사용하여 스트림 관리자에서 스트림을 생성한 다음, 읽기 및 쓰기 작업을 수행합니다. 그러면 스트림 관리자가 Kinesis Data Streams로 스트림을 내보냅니다. 다음 다이어그램은 이 워크플로를 보여 줍니다.



이 튜토리얼은 사용자 정의 Lambda 함수가 AWS IoT Greengrass Core SDK의 `StreamManagerClient` 객체를 사용하여 스트림 관리자와 상호 작용하는 방법을 보여 줍니다. 간편성을 위해 이 튜토리얼에서 생성한 Python Lambda 함수는 시뮬레이션된 디바이스 데이터를 생성합니다.

AWS IoT Greengrass API(AWS CLI의 Greengrass CLI 명령 포함)를 사용하여 그룹을 생성할 때 스트림 관리자가 기본적으로 비활성화됩니다. 코어에서 스트림 관리자를 활성화하려면 시스템 `GGStreamManager` Lambda 함수를 포함하는 함수 정의 버전과 새 함수 정의 버전을 참조하는 그룹 버전을 생성합니다. 그런 다음 그룹을 배포합니다.

필수 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- Greengrass 그룹 및 Greengrass 코어(v1.10 이상). Greengrass 그룹 및 코어를 생성하는 방법에 대한 자세한 내용은 시작하기: AWS IoT Greengrass 섹션을 참조하세요. 시작하기 자습서에는 AWS IoT Greengrass 코어 소프트웨어를 설치하는 단계도 포함되어 있습니다.

  > **Note**
  >
  > 스트림 관리자는 OpenWrt 배포에서 지원되지 않습니다.

- 코어 디바이스에 설치된 Java 8 런타임(JDK 8).

- Debian 기반 배포판(Raspbian 포함) 또는 Ubuntu 기반 배포판의 경우 다음 명령을 실행합니다.

  ```
  sudo apt install openjdk-8-jdk
  ```

- Red Hat 기반 배포판(Amazon Linux 포함) 의 경우 다음 명령을 실행합니다.

  ```
  sudo yum install java-1.8.0-openjdk
  ```

  자세한 내용은 OpenJDK 설명서의 OpenJDK 패키지 다운로드 및 설치 방법을 참조하십시오.

- AWS IoT Greengrass Core SDK for Python v1.5.0 이상. AWS IoT Greengrass Core SDK for Python에서 `StreamManagerClient`를 사용하려면 다음을 수행해야 합니다.
  - Python 3.7 이상을 코어 디바이스에 설치합니다.
  - Lambda 함수 배포 패키지에 SDK와 그 종속성을 포함시킵니다. 지침은 이 튜토리얼에 나와 있습니다.

  > **Tip**
  >
  > Java 또는 NodeJS로 `StreamManagerClient`를 사용할 수 있습니다. 예제 코드는 AWS IoT Greengrass Core SDK for Java 및 AWS IoT Greengrass Core SDK for Node.js를 참조하십시오.

- Greengrass 그룹과 동일한 AWS 리전에 Amazon Kinesis Data Streams에서 생성된 **`MyKinesisStream`**이라는 대상 스트림입니다. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서에서 스트림 생성을 참조하십시오.

  > **Note**
  >
  > 이 튜토리얼에서는 스트림 관리자가 데이터를 Kinesis Data Streams로 내보내고 이로 인해 AWS 계정에 요금이 청구됩니다. 요금 정보는 Kinesis Data Streams 요금을 참조하십시오. 요금이 부과되지 않도록 하기 위해 Kinesis 데이터 스트림을 생성하지 않고 이 튜토리얼을 실행할 수 있습니다. 이 경우 로그를 확인하여 스트림 관리자가 스트림을 Kinesis Data Streams로 내보내려고 시도했는지 확인합니다.

- 다음 예제에 나오는 것처럼 대상 전송 스트림에 대한 `kinesis:PutRecords` 작업을 허용하는 the section called “Greengrass 그룹 역할”에 추가된 IAM 정책입니다.

  ```
  {
  ```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

- 컴퓨터에 설치 및 구성된 AWS CLI. 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS Command Line Interface 설치 및 AWS CLI 구성을 참조하세요.

  이 튜토리얼의 예제 명령은 Linux 및 기타 Linux 기반 시스템을 위해 작성된 것입니다. Windows를 사용하는 경우 구문의 차이를 알아보려면 AWS 명령줄 인터페이스에 대한 파라미터 값 지정을 참조하십시오.

  명령에 JSON 문자열이 포함되어 있는 경우 이 자습서에서는 한 행에 JSON이 포함된 예제를 제공합니다. 일부 시스템에서는 이 형식을 사용해 보다 쉽게 명령을 편집 및 실행할 수 있습니다.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Lambda 함수 배포 패키지 생성
2. Lambda 함수 생성
3. 함수 정의 및 버전 생성
4. 로거 정의 및 버전 생성
5. 코어 정의 버전의 ARN 가져오기
6. 그룹 버전 생성
7. 배포 만들기
8. 애플리케이션 테스트

이 자습서를 완료하는 데 약 30분 정도 걸립니다.

1단계: Lambda 함수 배포 패키지 생성

이 단계에서는 Python 함수 코드와 종속성이 포함된 Lambda 함수 배포 패키지를 생성합니다. 나중에 AWS Lambda에서 Lambda 함수를 생성할 때 이 패키지를 업로드합니다. Lambda 함수는 AWS IoT Greengrass Core SDK를 사용하여 로컬 스트림을 생성하고 상호 작용합니다.

> **Note**
>
> 사용자 정의 Lambda 함수는 AWS IoT Greengrass Core SDK를 사용하여 스트림 관리자와 상호 작용해야 합니다. Greengrass 스트림 관리자의 요구 사항에 대한 자세한 내용은 Greengrass 스트림 관리자 요구 사항을 참조하십시오.

1. AWS IoT Greengrass Core SDK for Python v1.5.0 이상을 다운로드하십시오.
2. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
3. Lambda 함수 배포 패키지에 SDK와 함께 포함시킬 패키지 종속성을 설치합니다.

   1. `requirements.txt` 파일이 포함된 SDK 디렉터리로 이동합니다. 이 파일은 종속성을 나열합니다.
   2. SDK 종속성을 설치합니다. 예를 들어 다음 `pip` 명령을 실행하여 현재 디렉터리에 설치합니다.

      ```
      pip install --target . -r requirements.txt
      ```

4. `transfer_stream.py`이라는 로컬 파일에 다음과 같은 Python 코드 함수를 저장합니다.

   > **Tip**
   >
   > Java 및 NodeJS를 사용하는 예제 코드는 GitHub의 AWS IoT Greengrass Core SDK for Java 및 AWS IoT Greengrass Core SDK for Node.js를 참조하십시오.

   ```
   import asyncio
   import logging
   import random
   import time

   from greengrasssdk.stream_manager import (
   ```

```
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
 kinesis_stream_name=kinesis_stream_name)]
```

```
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
 strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
 ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
 to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return
```

```
logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. 다음 항목을 `transfer_stream_python.zip`라는 파일로 압축합니다. 이것이 Lambda 함수 배포 패키지입니다.

   - transfer_stream.py. 앱 로직.
   - greengrasssdk. MQTT 메시지를 게시하는 Python Greengrass Lambda 함수에 대한 필수 라이브러리입니다.

     스트림 관리자 작업은 AWS IoT Greengrass Core SDK for Python 버전 1.5.0 이상에서 사용할 수 있습니다.
   - AWS IoT Greengrass Core SDK for Python에 대해 설치한 종속성(예: `cbor2` 디렉터리).

   `zip` 파일을 생성할 때 이러한 항목만 포함시키고 포함 폴더는 포함시키지 않습니다.

2단계: Lambda 함수 생성

1. 함수 생성 시 역할 ARN에서 전달할 수 있도록 IAM 역할을 생성합니다.

   JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
           "Action": "sts:AssumeRole"
        }
    ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
 "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
 "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

> **Note**
>
> Greengrass 그룹 역할에 Greengrass Lambda 함수에 대한 권한이 지정되어 있기 때문에 AWS IoT Greengrass에서는 이 역할을 사용하지 않습니다. 이 튜토리얼에서는 빈 역할을 생성합니다.

2. 출력에서 `Arn`을 복사합니다.
3. AWS Lambda API를 사용하여 `TransferStream` 함수를 생성합니다. 다음 명령은 zip 파일이 현재 디렉터리에 있다고 가정합니다.

   - *`role-arn`*을 복사한 `Arn`으로 바꿉니다.

```
aws lambda create-function \
--function-name TransferStream \
--zip-file fileb://transfer_stream_python.zip \
--role role-arn \
--handler transfer_stream.function_handler \
--runtime python3.7
```

4. 이 함수의 버전을 게시합니다.

```
aws lambda publish-version --function-name TransferStream --description 'First
 version'
```

5. 게시된 버전에 대한 별칭을 생성합니다.

Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

6. 출력에서 `AliasArn`을 복사합니다. AWS IoT Greengrass에서 이 함수를 구성할 때 이 값을 사용합니다.

이제 AWS IoT Greengrass에 대해 함수를 구성할 준비가 되었습니다.

3단계: 함수 정의 및 버전 생성

이 단계는 시스템 `GGStreamManager` Lambda 함수와 사용자 정의 `TransferStream` Lambda 함수를 참조하는 함수 정의 버전을 생성합니다. AWS IoT Greengrass API를 사용할 때 스트림 관리자를 활성화하려면 함수 정의 버전에 `GGStreamManager` 함수가 포함되어야 합니다.

1. 시스템 및 사용자 정의 Lambda 함수를 포함하는 초기 버전이 포함된 함수 정의를 생성합니다.

   다음 정의 버전은 기본 파라미터 설정으로 스트림 관리자를 활성화합니다. 사용자 지정 설정을 구성하려면 해당 스트림 관리자가 파라미터에 대한 환경 변수를 정의합니다. 예제는 the section called “스트림 관리자를 활성화, 비활성화 또는 구성” 섹션을 참조하십시오. AWS IoT Greengrass는 생략된 파라미터에서 기본값을 사용합니다. `MemorySize`는 적어도 `128000`여야 합니다. `Pinned`를 `true`로 설정해야 합니다.

   > **Note**
   >
   > 수명이 긴(또는 고정) Lambda 함수는 AWS IoT Greengrass가 시작된 후 자동으로 시작되며, 자체 컨테이너에서 계속 실행됩니다. 이는 간접 호출되면 시작되고 실행할 작업이 남아 있지 않으면 중지되는 온디맨드 Lambda 함수와 상반됩니다. 자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.

   - *`arbitrary-function-id`*를 **`stream-manager`** 같은 함수에 대한 이름으로 바꿉니다.
   - *`alias-arn`*을 `TransferStream` Lambda 함수에 대한 별칭을 생성했을 때 복사한 `AliasArn`으로 바꿉니다.

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "arbitrary-function-id",
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize": 128000,
                "Pinned": true,
                "Timeout": 3
            }
        },
        {
            "Id": "TransferStreamFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "transfer_stream.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON single

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-
id","FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
 "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data","STREAM_MANAGER_SERVER_PORT":
 "1234","STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}},"MemorySize":
 128000,"Pinned": true,"Timeout": 3}},{"Id": "TransferStreamFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
```

```
"transfer_stream.function_handler", "MemorySize": 16000,"Pinned":
true,"Timeout": 5}}]}'
```

> **Note**
>
> `Timeout`은 함수 정의 버전에서 필요하지만, `GGStreamManager`에서는 이를 사용하지 않습니다. `Timeout` 및 기타 그룹 수준 설정에 대한 자세한 내용은 the section called "Greengrass Lambda 함수 실행 제어"을 참조하십시오.

2. 출력에서 `LatestVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 함수 정의 버전을 추가합니다.

4단계: 로거 정의 및 버전 생성

그룹의 로깅 설정을 구성합니다. 이 튜토리얼에서는 로그를 코어 디바이스의 파일 시스템에 쓸 수 있도록 AWS IoT Greengrass 시스템 구성 요소, 사용자 정의 Lambda 함수 및 커넥터를 구성합니다. 로그를 사용하여 발생할 수 있는 문제를 해결할 수 있습니다. 자세한 내용은 the section called "AWS IoT Greengrass 로그를 사용하여 모니터링" 섹션을 참조하세요.

1. 최초 버전이 포함된 로거 정의를 생성합니다.

   JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
    "Loggers": [
        {
            "Id": "1",
            "Component": "GreengrassSystem",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        },
        {
            "Id": "2",
            "Component": "Lambda",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
```

```
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \
    --name "LoggingConfigs" \
    --initial-version '{"Loggers":
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSyste
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. 출력에서 로거 정의의 `LatestVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 로거 정의 버전을 추가합니다.

5단계: 코어 정의 버전의 ARN 가져오기

새 그룹 버전에 추가할 코어 정의 버전의 ARN을 가져옵니다. 그룹 버전을 배포하려면 정확히 하나의 코어를 포함하는 코어 정의 버전을 참조해야 합니다.

1. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

> **Note**
>
> AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

2. 출력에서 대상 그룹의 `Id`을 복사합니다. 이 값을 사용하여 그룹을 배포할 때 코어 정의 버전을 가져옵니다.

3. 그룹에 추가된 최신 버전의 ID가 되도록 출력에서 `LatestVersion`을 복사합니다. 이 값을 사용하여 코어 정의 버전을 가져옵니다.
4. 코어 정의 버전의 ARN 가져오기:

   a. 그룹 버전을 가져옵니다.

   - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
   - *group-version-id*를 해당 그룹에서 복사한 `LatestVersion`으로 바꿉니다.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id group-version-id
   ```

   b. 출력에서 `CoreDefinitionVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 코어 정의 버전을 추가합니다.

6단계: 그룹 버전 생성

이제, 배포하고 싶은 모든 항목이 포함된 그룹 버전을 생성할 준비가 되었습니다. 이렇게 하려면 이 각 구성 요소 유형의 대상 버전을 참조하는 그룹 버전을 생성합니다. 이 튜토리얼에서는 코어 정의 버전, 함수 정의 버전 및 로거 정의 버전을 포함합니다.

1. 그룹 버전을 만듭니다.

   - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
   - *core-definition-version-arn*을 새 함수 정의 버전에서 복사한 `CoreDefinitionVersionArn`으로 바꿉니다.
   - *function-definition-version-arn*을 새 함수 정의 버전에서 복사한 `LatestVersionArn`로 바꿉니다.
   - *logger-definition-version-arn*을 새 로거 정의 버전에서 복사한 `LatestVersionArn`로 바꿉니다.

   ```
   aws greengrass create-group-version \
   --group-id group-id \
   --core-definition-version-arn core-definition-version-arn \
   --function-definition-version-arn function-definition-version-arn \
   ```

```
--logger-definition-version-arn logger-definition-version-arn
```

2. 출력에서 `Version`을 복사합니다. 새 그룹 버전의 ID가 이 값이 됩니다.

7단계: 배포 생성

코어 디바이스에 그룹을 배포합니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 대몬(daemon)이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 `root`에 대한 `/greengrass/ggc/packages/ggc-version/bin/daemon` 입력이 포함되어 있는 경우에는 대몬(daemon)이 실행 중인 것입니다.

   > **Note**
   >
   > 경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

   b. 대몬(daemon)을 시작하려면:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. 배포를 생성합니다.

   - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
   - *group-version-id*를 새 그룹 버전에서 복사한 `Version`으로 바꿉니다.

   ```
   aws greengrass create-deployment \
   --deployment-type NewDeployment \
   --group-id group-id \
   --group-version-id group-version-id
   ```

3. 출력에서 `DeploymentId`을 복사합니다.

4. 배포 상태를 가져옵니다.

   - *group-id*를 해당 그룹에서 복사한 Id로 바꿉니다.
   - *deployment-id*를 배포를 위해 복사한 DeploymentId로 바꿉니다.

   ```
   aws greengrass get-deployment-status \
   --group-id group-id \
   --deployment-id deployment-id
   ```

   상태가 Success이면 배포에 성공한 것입니다. 문제 해결에 대한 도움말은 문제 해결 섹션을 참조하십시오.

8단계: 애플리케이션 테스트

이 TransferStream Lambda 함수는 시뮬레이션된 디바이스 데이터를 생성합니다. 스트림 관리자가 대상 Kinesis 데이터 스트림으로 내보내는 스트림에 데이터를 기록합니다.

1. Amazon Kinesis 콘솔의 Kinesis Data Streams에서 MyKinesisStream을 선택합니다.

   > **Note**
   >
   > 대상 Kinesis 데이터 스트림 없이 튜토리얼을 실행한 경우 스트림 관리자(GGStreamManager)의 로그 파일을 확인하십시오. 오류 메시지에 `export stream MyKinesisStream doesn't exist`이 포함되어 있으면 테스트가 성공적입니다. 이 오류는 서비스가 스트림으로의 내보내기를 시도했지만 스트림이 존재하지 않는다는 것을 의미합니다.

2. MyKinesisStream 페이지에서 모니터링을 선택합니다. 테스트가 성공적이면 Put Records(레코드 넣기) 차트에 데이터가 표시되어야 합니다. 연결에 따라 데이터가 표시되기까지 1분 정도 걸릴 수 있습니다.

   > **Important**
   >
   > 테스트를 마쳤을 때 추가 요금이 발생하지 않도록 Kinesis 데이터 스트림을 삭제합니다. 또는 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다. 이렇게 하면 테스트를 계속할 준비가 될 때까지 코어가 메시지를 보내지 못하게 됩니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
```

3. 코어에서 TransferStream Lambda 함수를 제거합니다.

   a. the section called “그룹 버전 생성”에 따라 새 그룹 버전을 생성하되, `create-group-version` 명령에서 `--function-definition-version-arn` 옵션을 제거합니다. 아니면 TransferStream Lambda 함수를 포함하지 않는 함수 정의 버전을 생성합니다.

   > **Note**
   >
   > 배포된 그룹 버전에서 시스템 `GGStreamManager` Lambda 함수를 생략하면 코어에서 스트림 관리를 비활성화할 수 있습니다.

   b. the section called “배포 만들기”에 따라 새 그룹 버전을 배포합니다.

로깅 정보를 보거나 스트림 문제를 해결하려면 로그에서 `TransferStream` 및 `GGStreamManager` 함수를 확인합니다. 파일 시스템에서 AWS IoT Greengrass 로그를 읽을 수 있는 `root` 권한이 있어야 합니다.

- `TransferStream`은 *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`에 로그 항목을 기록합니다.
- `GGStreamManager`은 *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`에 로그 항목을 기록합니다.

추가적인 문제 해결 정보가 필요한 경우 `DEBUG`로 `Lambda` 로깅 수준을 설정하고 새 그룹 버전을 생성 및 배포합니다.

다음 사항도 참조하세요.

- 데이터 스트림 관리
- the section called “StreamManagerClient를 사용하여 스트림 작업”
- the section called “지원되는 AWS 클라우드 대상의 구성 내보내기”
- the section called “스트림 관리자 구성”
- the section called “데이터 스트림 내보내기(콘솔)”

- AWS CLI 명령 참조에서 AWS Identity and Access Management(IAM) 명령
- AWS CLI 명령 참조에서 AWS Lambda 명령
- AWS CLI 명령 참조에서 AWS IoT Greengrass 명령

AWS IoT Greengrass 코어에 암호 배포

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

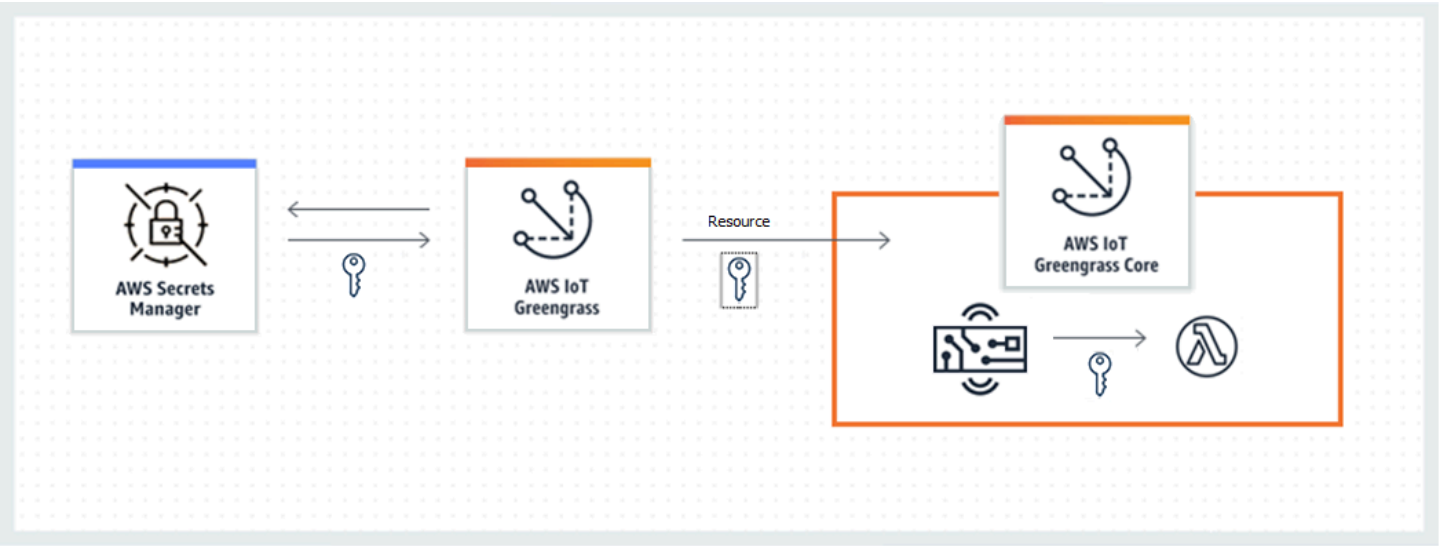
AWS IoT Greengrass를 사용하면 하드 코딩된 암호, 토큰 또는 기타 암호를 사용하지 않고도 Greengrass 디바이스의 서비스와 애플리케이션에 인증할 수 있습니다.

AWS Secrets Manager는 클라우드에서 보안 암호를 안전하게 저장하고 관리하는 데 사용할 수 있는 서비스입니다. AWS IoT Greengrass는 Secrets Manager를 Greengrass 코어 디바이스로 확장하며, 따라서 커넥터 및 Lambda 함수가 로컬 보안 암호를 사용해 서비스 및 애플리케이션과 상호 작용할 수 있습니다. 예를 들어, Twilio 알림 커넥터는 로컬로 저장된 인증 토큰을 사용합니다.

보안 암호를 Greengrass 그룹에 통합하려면 Secrets Manager 보안 암호를 참조하는 그룹 리소스를 생성합니다. 이 암호 리소스는 ARN을 통해 클라우드 암호를 참조합니다. 암호 리소스의 생성, 관리, 사용 방법을 알아보려면 the section called “암호 리소스 작업”을 참조하십시오.

AWS IoT Greengrass는 전송 중이거나 상주 중인 암호를 암호화합니다. 그룹 배포 중에 AWS IoT Greengrass는 Secrets Manager에서 암호를 가져오고 코어에서 암호화된 로컬 복사본을 생성합니다. Secrets Manager에서 클라우드 암호를 교체한 후 그룹을 다시 배포하여 업데이트된 값을 코어에 전파합니다.

다음 다이어그램에서는 암호를 코어에 배포하는 프로세스를 자세히 보여줍니다. 암호는 전송 및 저장 상태에서 암호화됩니다.



AWS IoT Greengrass를 사용하여 로컬로 암호를 저장하는 경우 다음과 같은 이점이 있습니다.

- 코드에서 분리됨(하드 코딩되지 않음). 중앙에서 관리되는 자격 증명을 지원하며 민감한 데이터를 손상 위험으로부터 보호할 수 있습니다.
- 오프라인 시나리오에 사용할 수 있습니다. 인터넷 연결이 끊어진 상태에서 커넥터와 함수가 로컬 서비스 및 소프트웨어에 안전하게 액세스할 수 있습니다.
- 암호에 제어된 액세스. 그룹에서 승인된 커넥터와 함수만 암호에 액세스할 수 있습니다. AWS IoT Greengrass는 프라이빗 키 암호화를 사용하여 암호를 보호합니다. 암호는 전송 및 저장 상태에서 암호화됩니다. 자세한 내용은 the section called “암호 암호화” 섹션을 참조하세요.
- 제어된 회전. Secrets Manager에서 암호를 교체한 후 Greengrass 그룹을 다시 배포하여 암호의 로컬 사본을 업데이트합니다. 자세한 내용은 the section called “보안 암호 생성 및 관리” 섹션을 참조하세요.

> **Important**
>
> AWS IoT Greengrass는 클라우드 버전 교체 후 로컬 암호 값을 자동으로 업데이트하지 않습니다. 로컬 값을 업데이트하려면 그룹을 다시 배포해야 합니다.

암호 암호화

AWS IoT Greengrass는 전송 및 유휴 상태의 암호를 암호화합니다.

> **Important**
>
> 사용자 정의한 Lambda 함수에서 보안 암호를 안전하게 취급해야 하고, 보안 암호에 저장된 기밀 데이터를 로그에 기록하지 말아야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Lambda 함수 로깅 및 디버깅의 위험 완화를 참조하십시오. 이 설명서에서는 특별히 교체 기능을 언급하지만 권장 사항은 Greengrass Lambda 함수에도 적용됩니다.

전송 중 데이터 암호화

AWS IoT Greengrass는 TLS(전송 계층 보안)를 사용하여 인터넷 및 로컬 네트워크를 통한 모든 통신을 암호화합니다. 그러면 Secrets Manager에서 암호를 검색하여 코어에 배포하는 과정에서 전송 중인 암호가 보호됩니다. 지원되는 TLS 암호 그룹은 the section called “TLS 암호 그룹 지원” 섹션을 참조하십시오.

저장된 데이터 암호화

AWS IoT Greengrass는 `config.json`에 지정된 프라이빗 키를 사용해 코어에 저장된 암호를 암호화합니다. 이 때문에 로컬 암호를 보호하려면 프라이빗 키의 보안 스토리지가 필요합니다. AWS 공동 책임 모델에서는 코어 디바이스에서 프라이빗 키의 보안 스토리지를 보장할 책임이 고객에게 있습니다.

AWS IoT Greengrass는 두 가지 프라이빗 키 스토리지 모드를 지원합니다.

- 하드웨어 보안 모듈 사용 자세한 내용은 the section called “하드웨어 보안 통합” 섹션을 참조하세요.

> **Note**
>
> 현재 AWS IoT Greengrass는 하드웨어 기반 프라이빗 키를 사용할 경우 로컬 보안 암호의 암호화 및 해독에 PKCS#1 v1.5 패딩 메커니즘만 지원합니다. 공급업체에서 제공한 지침에 따라 하드웨어 기반 프라이빗 키를 수동으로 생성하는 경우, PKCS#1 v1.5를 선택해야 합니다. AWS IoT Greengrass는 OAEP(Optimal Asymmetric Encryption Padding)를 지원하지 않습니다.

- 파일 시스템 권한 사용(기본값)

프라이빗 키는 로컬 암호를 암호화하는 데 사용되는 데이터 키를 보호하는 데 사용됩니다. 데이터 키는 각 그룹 배포로 교체됩니다.

AWS IoT Greengrass는 프라이빗 키에 액세스할 수 있는 유일한 엔터티입니다. 암호 리소스와 연계된 Greengrass 커넥터 또는 Lambda 함수는 코어에서 암호의 값을 가져옵니다.

요구 사항

로컬 암호 지원을 위한 요구 사항은 다음과 같습니다.

- AWS IoT Greengrass Core v1.7 이상을 사용하고 있어야 합니다.
- 로컬 비밀 값을 가져오려면 사용자 정의 Lambda 함수가 AWS IoT Greengrass 코어 SDK v1.3.0 이상을 사용해야 합니다.
- 로컬 암호 암호화에 사용되는 프라이빗 키를 Greengrass 구성 파일에서 지정해야 합니다. 기본적으로 AWS IoT Greengrass는 파일 시스템에 저장된 코어 프라이빗 키를 사용합니다. 사용자 본인의 프라이빗 키를 제공하려면 the section called “암호 암호화를 위한 프라이빗 키 지정” 섹션을 참조하십시오. RSA 키 유형만 지원됩니다.

> **Note**
>
> 현재 AWS IoT Greengrass는 하드웨어 기반 프라이빗 키를 사용할 경우 로컬 보안 암호의 암호화 및 해독에 PKCS#1 v1.5 패딩 메커니즘만 지원합니다. 공급업체에서 제공한 지침에 따라 하드웨어 기반 프라이빗 키를 수동으로 생성하는 경우, PKCS#1 v1.5를 선택해야 합니다. AWS IoT Greengrass는 OAEP(Optimal Asymmetric Encryption Padding)를 지원하지 않습니다.

- AWS IoT Greengrass에 암호 값을 가져올 수 있는 권한을 부여해야 합니다. 그러면 AWS IoT Greengrass가 그룹 배포 중에 값을 가져올 수 있습니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 AWS IoT Greengrass는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다. 액세스를 사용자 지정하려면 the section called “AWS IoT Greengrass의 암호 값 가져오기 허용”을 참조하십시오.

> **Note**
>
> 권한을 사용자 지정하는 경우에도 이 명령 규칙을 사용하여 AWS IoT Greengrass가 액세스할 수 있는 암호를 식별하는 것이 좋습니다. 콘솔에서는 다른 권한을 사용하여 암호를 판독하므로 AWS IoT Greengrass가 가져올 권한이 없는 암호를 콘솔에서 선택할 수 있습니다. 명명 규칙을 사용하면 배포 오류가 발생하는 권한 충돌을 방지할 수 있습니다.

암호 암호화를 위한 프라이빗 키 지정

이 절차에서는 로컬 보안 암호화에 사용되는 프라이빗 키의 경로를 제공합니다. 이는 최소 2048비트 길이의 RSA 키여야 합니다. AWS IoT Greengrass 코어에서 사용되는 프라이빗 키에 대한 자세한 내용은 the section called “보안 주체” 섹션을 참조하세요.

AWS IoT Greengrass에서는 두 가지 프라이빗 키 스토리지 모드인 하드웨어 기반 또는 파일 시스템 기반(기본값)을 지원합니다. 자세한 내용은 the section called “암호 암호화” 섹션을 참조하세요.

파일 시스템에서 코어 프라이빗 키를 사용하는 기본 구성을 변경하려는 경우에만 이 절차를 따르십시오. 이 단계는 시작하기 저습서의 모듈 2에 설명된 대로 그룹 및 코어를 생성했다는 가정하에 작성되었습니다.

1. */greengrass-root*/config 디렉터리에 있는 config.json 파일은 여십시오.

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

2. crypto.principals.SecretsManager 객체에서 privateKeyPath 속성에 프라이빗 키의 경로를 입력합니다.

   - 프라이빗 키가 파일 시스템에 저장된 경우 키의 절대 경로를 지정합니다. 예:

     ```
     "SecretsManager" : {
       "privateKeyPath" : "file:///somepath/hash.private.key"
     }
     ```

   - 프라이빗 키가 HSM(하드웨어 보안 모듈)에 저장되는 경우, RFC 7512 PKCS#11 URI 체계를 사용해 경로를 지정합니다. 예:

     ```
     "SecretsManager" : {
       "privateKeyPath" : "pkcs11:object=private-key-label;type=private"
     }
     ```

     자세한 내용은 the section called “하드웨어 보안 구성” 섹션을 참조하세요.

     > **Note**
     >
     > 현재 AWS IoT Greengrass는 하드웨어 기반 프라이빗 키를 사용할 경우 로컬 보안 암호의 암호화 및 해독에 PKCS#1 v1.5 패딩 메커니즘만 지원합니다. 공급업체에서 제공한 지침에 따라 하드웨어 기반 프라이빗 키를 수동으로 생성하는 경우, PKCS#1 v1.5를 선택해야 합니다. AWS IoT Greengrass는 OAEP(Optimal Asymmetric Encryption Padding)를 지원하지 않습니다.

AWS IoT Greengrass의 암호 값 가져오기 허용

이 절차에서는 AWS IoT Greengrass가 암호 값을 가져오도록 허용하는 인라인 정책을 Greengrass 서비스 역할에 추가합니다.

암호에 대한 AWS IoT Greengrass 사용자 지정 권한을 허용하거나 Greengrass 서비스 역할에 `AWSGreengrassResourceAccessRolePolicy` 관리형 정책이 포함되지 않는 경우에만 이 절차를 따르십시오. `AWSGreengrassResourceAccessRolePolicy`는 greengrass-로 시작하는 이름을 가진 암호에 대한 액세스 권한을 부여합니다.

1. 다음 CLI 명령을 실행해 Greengrass 서비스 역할의 ARN을 가져옵니다.

   ```
   aws greengrass get-service-role-for-account --region region
   ```

   반환되는 ARN에는 역할 이름이 포함되어 있습니다.

   ```
   {
     "AssociatedAt": "time-stamp",
     "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
   }
   ```

   다음 단계에서는 ARN 또는 이름을 사용합니다.

2. `secretsmanager:GetSecretValue` 작업을 허용하는 인라인 정책을 추가합니다. 지침은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하십시오.

   암호를 명시적으로 나열하거나 와일드카드 * 이름 지정 체계를 사용하여 세분화된 액세스 권한을 부여하거나 버전이 지정되거나 태그 지정된 암호에 대한 조건부 액세스를 허용할 수 있습니다. 예를 들어, 다음 정책은 AWS IoT Greengrass에 지정된 암호만 읽을 수 있도록 허용합니다.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "secretsmanager:GetSecretValue"
               ],
               "Resource": [
                   "arn:aws:secretsmanager:region:account-id:secret:greengrass-
   SecretA-abc",
                   "arn:aws:secretsmanager:region:account-id:secret:greengrass-
   SecretB-xyz"
               ]
           }
       ]
   ```

```
}
```

> **Note**
>
> 고객 관리형 AWS KMS 키를 사용해 암호를 암호화할 경우, Greengrass 서비스 역할이 `kms:Decrypt` 작업을 허용해야 합니다.

Secrets Manager의 IAM 정책에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager에 대한 인증 및 액세스 제어 및 AWS Secrets Manager에 대한 IAM 정책 또는 보안 정책에서 사용할 수 있는 작업, 리소스 및 컨텍스트 키를 참조하십시오.

다음 사항도 참조하세요.

- AWS Secrets Manager 사용 설명서의 AWS Secrets Manager란 무엇입니까?
- PKCS #1: RSA 암호화 버전1.5

암호 리소스 작업

AWS IoT Greengrass는 암호 리소스를 사용해 AWS Secrets Manager의 암호를 Greengrass 그룹으로 통합합니다. 암호 리소스는 Secrets Manager의 암호에 대한 참조입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

AWS IoT Greengrass 코어 디바이스에서, 커넥터 및 Lambda 함수는 암호, 토큰 또는 다른 자격 증명을 하드 코딩하지 않고 보안 암호 리소스를 사용하여 서비스와 애플리케이션을 인증할 수 있습니다.

보안 암호 생성 및 관리

Greengrass 그룹에서 암호 리소스는 Secrets Manager 암호의 ARN을 참조합니다. 암호 리소스가 코어에 배포되면 암호의 값이 암호화되어 연계된 커넥터 및 Lambda 함수에 사용할 수 있게 됩니다. 자세한 내용은 the section called “암호 암호화” 섹션을 참조하세요.

Secrets Manager을 사용하여 암호의 클라우드 버전을 생성하고 관리합니다. AWS IoT Greengrass를 사용하여 암호 리소스를 생성, 관리, 배포합니다.

> **Important**
>
> Secrets Manager에서 암호 교체 모범 사례를 따르는 것이 좋습니다. 그런 다음 Greengrass 그룹을 배포하여 암호의 로컬 사본을 업데이트합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서에서 AWS Secrets Manager 비밀 교체를 참조하세요.

Greengrass 코어에서 암호를 사용할 수 있도록 지정하려면

1. Secrets Manager에서 보안 암호를 생성합니다. 이 암호는 Secrets Manager에서 중앙 집중식으로 저장되고 관리되는 클라우드 버전 암호입니다. 관리 작업에는 암호 값 교체와 리소스 정책 적용이 포함됩니다.
2. AWS IoT Greengrass에서 암호 리소스를 생성합니다. 암호 리소스는 ARN을 통해 클라우드 암호를 참조하는 그룹 리소스의 한 유형입니다. 암호를 그룹당 한 번만 참조할 수 있습니다.
3. 커넥터 또는 Lambda 함수를 구성합니다. 해당 파라미터 또는 속성을 지정하여 리소스를 커넥터 또는 함수와 연계해야 합니다. 그러면 함수를 사용하여 로컬로 배포된 암호 리소스의 값을 가져올 수 있습니다. 자세한 내용은 the section called “로컬 암호 사용” 섹션을 참조하세요.
4. Greengrass 그룹을 배포합니다. 배포 중에 AWS IoT Greengrass는 클라우드 암호 값을 가져오고 코어에서 로컬 암호를 생성하거나 업데이트합니다.

AWS CloudTrail에서 암호 값을 검색할 때마다 Secrets Manager는 AWS IoT Greengrass에 이벤트를 기록합니다. AWS IoT Greengrass는 로컬 암호의 배포 또는 사용과 관련된 이벤트를 기록하지 않습니다. Secrets Manager 로깅에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager 비밀 사용 모니터링을 참조하십시오.

암호 리소스에 스테이징 레이블 포함

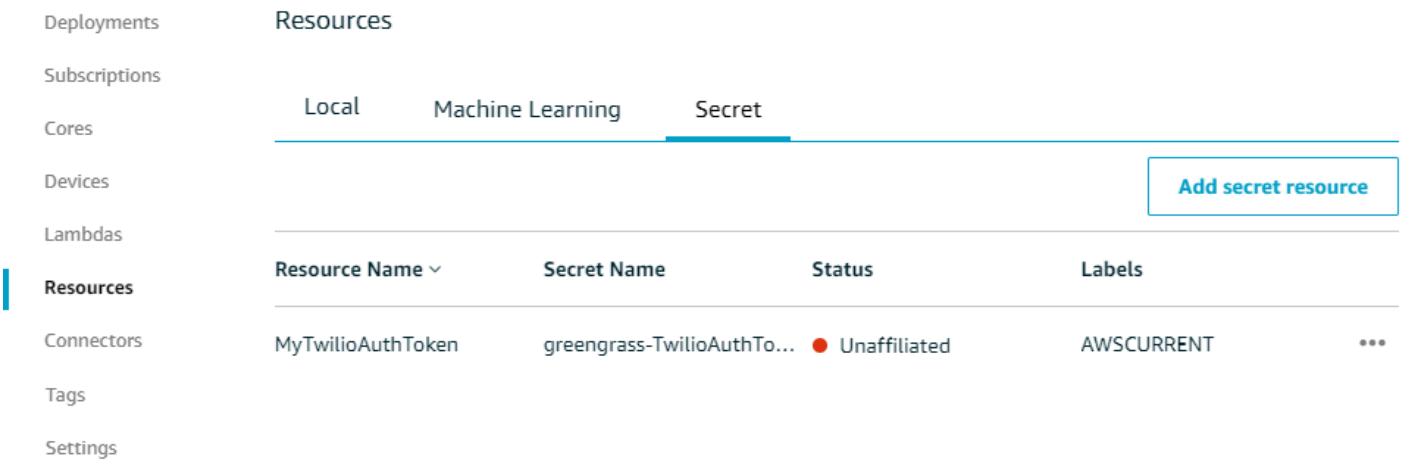
Secrets Manager는 스테이징 레이블을 사용하여 암호 값의 특정 버전을 식별합니다. 스테이징 레이블은 시스템에서 정의하거나 사용자가 정의할 수 있습니다. Secrets Manager는 비밀 값의 최신 버전에 `AWSCURRENT` 레이블을 지정합니다. 스테이징 레이블은 일반적으로 암호 교체를 관리하는 데 사용됩니다. Secrets Manager 버전 관리에 대한 자세한 정보는 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager의 주요 개념 및 용어를 참조하세요.

암호 리소스는 항상 `AWSCURRENT` 스테이징 레이블을 포함하며, Lambda 함수 또는 커넥터에 필요한 경우 다른 스테이징 레이블을 선택적으로 포함할 수 있습니다. 그룹 배포 중에 AWS IoT Greengrass는 그룹에서 참조되는 스테이징 레이블 값을 검색한 후 코어에서 해당 값을 생성하거나 업데이트합니다.

암호 리소스 생성 및 관리(콘솔)

암호 리소스 생성(콘솔)

AWS IoT Greengrass 콘솔에서는 그룹 리소스 페이지의 비밀 탭에서 비밀 리소스를 생성하고 관리할 수 있습니다. 암호 리소스를 생성하여 그룹에 추가하는 방법을 설명하는 자습서는 the section called "암호 리소스 생성 방법(콘솔)" 및 the section called "커넥터 시작하기(콘솔)" 단원을 참조하십시오.



> **Note**
>
> 대안으로, 커넥터 또는 Lambda 함수를 구성할 때 콘솔에서 보안 암호와 보안 암호 리소스를 생성할 수도 있습니다. 커넥터의 파라미터 구성 페이지 또는 Lambda 함수의 리소스 페이지에서 이 작업을 수행할 수 있습니다.

암호 리소스 관리(콘솔)

Greengrass 그룹의 보안 암호 리소스에 대한 관리 작업에는 그룹에 보안 암호 리소스 추가, 그룹에서 보안 암호 리소스 제거, 보안 암호 리소스에 포함된 스테이징 레이블 세트 변경이 포함됩니다.

Secrets Manager에서 다른 암호를 가리키는 경우 암호를 사용하는 커넥터도 편집해야 합니다.

1. 그룹 구성 페이지에서 Connectors(커넥터)를 선택합니다.
2. 커넥터의 컨텍스트 메뉴에서 Edit(편집)를 선택합니다.
3. 파라미터 편집 페이지에 암호 ARN이 변경되었다고 알려주는 메시지가 표시됩니다. 변경을 확인하려면 저장을 선택합니다.

Secrets Manager에서 보안 암호를 삭제할 경우 그룹 및 해당 보안 리소스를 참조하는 커넥터 및 Lambda 함수에서 보안 리소스를 제거해야 합니다. 이렇게 하지 않으면 그룹 배포 중 AWS IoT Greengrass는 보안 암호를 찾을 수 없다는 오류를 반환합니다. 또한 필요한 경우 Lambda 함수 코드를 업데이트합니다.

암호 리소스 생성 및 관리(CLI)

암호 리소스 생성(CLI)

AWS IoT Greengrass API에서 암호는 그룹 리소스의 한 유형입니다. 다음 예에서는 `MySecretResource`라는 암호 리소스를 포함하는 초기 버전으로 리소스 정의를 생성합니다. 암호 리소스를 생성하여 그룹 버전에 추가하는 방법을 설명하는 자습서는 the section called “커넥터 시작하기(CLI)” 단원을 참조하십시오.

암호 리소스는 해당 Secrets Manager 암호의 ARN을 참조하며, 항상 포함되는 `AWSCURRENT` 이외에 두 스테이징 레이블을 포함합니다.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
    "Resources": [
        {
            "Id": "my-resource-id",
            "Name": "MySecretResource",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2"
                    ]
                }
            }
        }
    ]
}'
```

암호 리소스 관리(CLI)

Greengrass 그룹의 보안 암호 리소스에 대한 관리 작업에는 그룹에 보안 암호 리소스 추가, 그룹에서 보안 암호 리소스 제거, 보안 암호 리소스에 포함된 스테이징 레이블 세트 변경이 포함됩니다.

AWS IoT Greengrass API에서 이러한 변경은 버전을 사용하여 구현됩니다.

AWS IoT Greengrass API는 버전을 사용하여 그룹을 관리합니다. 버전은 변경할 수 없으므로 그룹 구성 요소(예: 그룹의 클라이언트 디바이스, 기능 및 리소스)를 추가하거나 변경하려면 새 구성 요소나 업데이트된 구성 요소의 버전을 생성해야 합니다. 그런 다음 각 구성 요소의 대상 버전을 포함하는 그룹 버전을 생성하고 배포합니다. 그룹에 대한 자세한 내용은 the section called “AWS IoT Greengrass 그룹”을 참조하십시오.

예를 들어, 암호 리소스에 대한 스테이징 레이블 세트를 변경하려면

1. 업데이트된 암호 리소스를 포함하는 리소스 정의 버전을 생성합니다. 다음 예에서는 이전 단원의 암호 리소스에 세 번째 스테이징 레이블을 추가합니다.

> **Note**
>
> 더 많은 리소스를 버전에 추가하려면 `Resources` 어레이에 해당 리소스를 포함합니다.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
    "Resources": [
        {
            "Id": "my-resource-id",
            "Name": "MySecretResource",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2",
                        "Label3"
                    ]
                }
            }
        }
    ]
}'
```

2. 암호 리소스의 ID가 변경된 경우 암호 리소스를 사용하는 커넥터 및 함수를 업데이트합니다. 새 버전에서 리소스 ID에 해당하는 파라미터 또는 속성을 업데이트합니다. 암호의 ARN이 변경된 경우 암호를 사용하는 커넥터에 대한 해당 파라미터를 변경해야 합니다.

   > **Note**
   >
   > 리소스 ID는 고객이 제공하는 임의의 식별자입니다.

3. 코어에 보낼 각 구성 요소의 대상 버전을 포함하는 그룹 버전을 생성합니다.
4. 그룹 버전을 배포합니다.

암호 리소스, 커넥터 및 함수를 생성하고 배포하는 방법을 보여주는 자습서는 the section called “커넥터 시작하기(CLI)” 단원을 참조하십시오.

Secrets Manager에서 보안 암호를 삭제할 경우 그룹 및 해당 보안 리소스를 참조하는 커넥터 및 Lambda 함수에서 보안 리소스를 제거해야 합니다. 이렇게 하지 않으면 그룹 배포 중 AWS IoT Greengrass는 보안 암호를 찾을 수 없다는 오류를 반환합니다. 또한 필요한 경우 Lambda 함수 코드를 업데이트합니다. 해당 암호 리소스를 포함하지 않는 리소스 정의 버전을 배포하여 로컬 암호를 제거할 수 있습니다.

커넥터 및 Lambda 함수에서 로컬 암호 사용

Greengrass 커넥터 및 Lambda 함수는 로컬 암호를 사용해 타사 서비스 및 애플리케이션과 상호 작용합니다. `AWSCURRENT` 값이 기본적으로 사용되지만 암호 리소스에 포함된 다른 스테이징 레이블의 값을 사용할 수도 있습니다.

로컬 암호에 액세스하려면 커넥터 및 함수를 구성해야 합니다. 이렇게 하면 암호 리소스가 커넥터 또는 함수와 연계됩니다.

커넥터

커넥터는 로컬 암호에 액세스해야 하는 경우 암호에 액세스하는 데 필요한 정보로 구성된 파라미터를 제공합니다.

- AWS IoT Greengrass에서 이 작업을 수행하는 방법은 the section called “커넥터 시작하기(콘솔)” 단원을 참조하십시오.
- AWS IoT Greengrass CLI로 이 작업을 수행하는 방법은 the section called “커넥터 시작하기(CLI)”를 참조하십시오.

개별 커넥터를 위한 요구 사항에 대한 자세한 내용은 the section called “AWS에서 제공한 Greengrass 커넥터”를 참조하십시오.

암호를 액세스하고 사용하기 위한 로직은 커넥터에 기본 제공됩니다.

Lambda 함수

Lambda 함수가 로컬 암호에 액세스하도록 허용하려면 함수의 속성을 구성합니다.

- AWS IoT Greengrass에서 이 작업을 수행하는 방법은 the section called “암호 리소스 생성 방법(콘솔)” 단원을 참조하십시오.
- AWS IoT Greengrass API에서 이 작업을 수행하려면 `ResourceAccessPolicies` 속성에 다음 정보를 제공합니다.
  - `ResourceId`: Greengrass 그룹에 있는 암호 리소스의 ID입니다. 해당 Secrets Manager 암호의 ARN을 참조하는 리소스입니다.
  - `Permission`: 함수가 리소스에 대해 갖는 액세스 권한 유형입니다. 암호 리소스에 대해 `ro`(읽기 전용) 권한만 지원됩니다.

다음 예에서는 `MyApiKey` 암호 리소스에 액세스할 수 있는 Lambda 함수를 생성합니다.

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-
version '{
    "Functions": [
        {
            "Id": "MyLambdaFunction",
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1",
            "FunctionConfiguration": {
                "Pinned": false,
                "MemorySize": 16384,
                "Timeout": 10,
                "Environment": {
                    "ResourceAccessPolicies": [
                        {
                            "ResourceId": "MyApiKey",
                            "Permission": "ro"
                        }
                    ],
                    "AccessSysfs": true
                }
            }
        }
    ]
```

```
}'
```

Greengrass Lambda 함수는 런타임에 로컬 암호에 액세스하기 위해 AWS IoT Greengrass Core SDK (v1.3.0 이상)의 `secretsmanager` 클라이언트에서 `get_secret_value` 함수를 직접 호출합니다.

다음 예에서는 함수가 Python용 AWS IoT Greengrass Core SDK를 사용해 암호를 받는 방법을 보여줍니다. 여기서는 암호 이름이 `get_secret_value` 함수로 전달됩니다. `SecretId`는 (암호 리소스가 아니라) Secrets Manager 암호의 이름 또는 ARN일 수 있습니다.

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"


def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

텍스트 형식 암호의 경우에는 `get_secret_value` 함수가 문자열을 반환합니다. 이진수 형식 암호의 경우에는 base64 인코딩 문자열을 반환합니다.

> **Important**
>
> 사용자 정의한 Lambda 함수에서 보안 암호를 안전하게 취급해야 하고, 보안 암호에 저장된 기밀 데이터를 로그에 기록하지 말아야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Lambda 함수 로깅 및 디버깅의 위험 완화를 참조하십시오. 이 설명서에서는 특별히 교체 기능을 언급하지만 권장 사항은 Greengrass Lambda 함수에도 적용됩니다.

암호의 현재 값은 기본적으로 반환됩니다. 이는 `AWSCURRENT` 스테이징 레이블이 연결된 버전입니다. 다른 버전에 액세스하려면 선택적 `VersionStage` 인수에 대해 해당하는 스테이징 레이블의 이름을 전달합니다. 예:

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
secret_version = "MyTargetLabel"


# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

`get_secret_value`를 호출하는 다른 예제 함수는 Lambda 함수 배포 패키지 생성를 참조하십시오.

암호 리소스 생성 방법(콘솔)

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

이 자습서에서는 AWS Management Console을 사용하여 암호 리소스를 Greengrass 그룹에 추가하는 방법을 보여줍니다. 암호 리소스는 AWS Secrets Manager의 암호에 대한 참조입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

AWS IoT Greengrass 코어 디바이스에서, 커넥터 및 Lambda 함수는 암호, 토큰 또는 다른 자격 증명을 하드 코딩하지 않고 보안 암호 리소스를 사용하여 서비스와 애플리케이션을 인증할 수 있습니다.

이 자습서에서는 암호를 AWS Secrets Manager 콘솔에서 생성하면서 시작합니다. 그 다음에는 AWS IoT Greengrass 콘솔에서 그룹 리소스 페이지의 Greengrass 그룹에 암호 리소스를 추가합니다. 이 암호 리소스는 Secrets Manager 암호를 참조합니다. 이후에는 암호 리소스를 Lambda 함수에 연결하는데, 이를 통해 함수가 로컬 암호 값을 가져올 수 있습니다.

> **Note**
>
> 대안으로, 커넥터 또는 Lambda 함수를 구성할 때 콘솔에서 보안 암호와 보안 암호 리소스를 생성할 수도 있습니다. 커넥터의 파라미터 구성 페이지 또는 Lambda 함수의 리소스 페이지에서 이 작업을 수행할 수 있습니다.
> 암호에 대한 파라미터를 포함하는 커넥터에서만 암호에 액세스할 수 있습니다. Twilio Notifications 커넥터에서 로컬로 저장된 인증 토큰을 사용하는 방법을 보여주는 자습서는 the section called “커넥터 시작하기(콘솔)”을 참조하십시오.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Secrets Manager 보안 암호 생성
2. 그룹에 보안 암호 리소스 추가
3. Lambda 함수 배포 패키지 생성
4. Lambda 함수 생성
5. 그룹에 함수 추가
6. 암호 리소스의 함수 연결
7. 그룹에 구독 추가
8. 그룹 배포
9. the section called “Lambda 함수 테스트”

이 자습서를 완료하는 데 약 20분 정도 걸립니다.

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- Greengrass 그룹 및 Greengrass 코어(v1.7 이상). Greengrass 그룹 및 코어를 생성하는 방법에 대해 알아보려면 시작하기: AWS IoT Greengrass단원을 참조하십시오. 시작하기 자습서에는 AWS IoT Greengrass 코어 소프트웨어를 설치하는 단계도 포함되어 있습니다.
- 로컬 암호를 지원하도록 AWS IoT Greengrass를 구성해야 합니다. 자세한 내용은 암호 요구 사항을 참조하십시오.

> **Note**
>
> 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 로컬 비밀 값을 가져오려면 사용자 정의 Lambda 함수가 AWS IoT Greengrass 코어 SDK v1.3.0 이상을 사용해야 합니다.

1단계: Secrets Manager 보안 암호 생성

이 단계에서는 AWS Secrets Manager 콘솔을 사용하여 암호를 생성합니다.

1. AWS Secrets Manager 콘솔에 로그인합니다.

   > **Note**
   >
   > 이 프로세스에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 1단계: 비밀 만들기 및 AWS Secrets Manager에 저장을 참조하십시오.

2. Store a new secret(새 보안 암호 저장)을 선택합니다.
3. 암호 유형 선택에서 다른 암호 유형을 선택합니다.
4. 이 보안 암호에 저장할 키-값 쌍 지정에서 다음을 수행합니다.
   - 키(Key)에 **`test`**를 입력합니다.
   - 값에 **`abcdefghi`**을(를) 입력합니다.
5. 암호화 키로 aws/secretsmanager가 선택된 상태를 유지하고 다음을 선택합니다.

   > **Note**
   >
   > Secrets Manager가 계정에서 생성하는 기본 AWS 관리형 키를 사용하는 경우 AWS KMS에서 요금이 청구되지 않습니다.

6. Secret name(보안 암호 이름)에 **`greengrass-TestSecret`**을 입력한 후 다음을 선택합니다.

> **Note**
>
> 기본적으로 Greengrass 서비스 역할은 AWS IoT Greengrass이(가) 이름이 greengrass–로 시작하는 비밀의 값을 가져오도록 허용합니다. 자세한 내용은 암호 요구 사항을 참조하십시오.

7. 이 자습서에서는 교체할 필요가 없으므로 자동 교체 비활성화를 선택하고 다음을 선택합니다.
8. Review(검토) 페이지에서 설정을 검토한 다음 Store(저장)를 선택합니다.

   이제 해당 암호를 참조하는 암호 리소스를 Greengrass 그룹에 생성합니다.

2단계: Greengrass 그룹에 보안 암호 리소스 추가

이 단계에서는 Secrets Manager 암호를 참조하는 그룹 리소스를 구성합니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 암호 리소스를 추가할 그룹을 선택합니다.
3. 그룹 구성 페이지에서 리소스 탭을 선택한 다음 비밀 섹션까지 아래로 스크롤합니다. 비밀 섹션에 그룹에 속하는 비밀 리소스가 표시됩니다. 이 탭에서 암호 리소스를 추가, 편집, 제거할 수 있습니다.

   > **Note**
   >
   > 대안으로, 커넥터 또는 Lambda 함수를 구성할 때 콘솔에서 보안 암호와 보안 암호 리소스를 생성할 수도 있습니다. 커넥터의 파라미터 구성 페이지 또는 Lambda 함수의 리소스 페이지에서 이 작업을 수행할 수 있습니다.

4. 비밀 섹션에서 추가를 선택합니다.
5. 비밀 리소스 추가 페이지에서 리소스 이름에 **`MyTestSecret`**을(를) 입력합니다.
6. 비밀에서 greengrass-TestSecret을 선택합니다.
7. 레이블 선택(선택 사항) 섹션에서 AWSCURRENT 스테이징 레이블은 비밀의 최신 버전을 나타냅니다. 이 레이블은 암호 리소스에 항상 포함되어 있습니다.

> **Note**
>
> 이 자습서에서는 AWSCURRENT 레이블만 있으면 됩니다. Lambda 함수 또는 커넥터에 필요한 레이블을 선택적으로 포함시킬 수 있습니다.

8. 리소스 추가를 선택합니다.

3단계: Lambda 함수 배포 패키지 생성

Lambda 함수를 생성하려면 먼저 함수 코드와 종속성을 포함하는 Lambda 함수 배포 패키지를 생성해야 합니다. Greengrass Lambda 함수는 코어 환경에서 MQTT 메시지와 통신하고 로컬 비밀에 액세스하는 등의 작업을 위해 AWS IoT Greengrass Core SDK가 필요합니다. 이 자습서는 Python 함수를 생성하므로 배포 패키지의 Python 버전 SDK를 사용할 수 있습니다.

> **Note**
>
> 로컬 비밀 값을 가져오려면 사용자 정의 Lambda 함수가 AWS IoT Greengrass 코어 SDK v1.3.0 이상을 사용해야 합니다.

1. AWS IoT Greengrass 코어 SDK 다운로드 페이지에서 Python용 AWS IoT Greengrass 코어 SDK를 다운로드합니다.
2. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
3. `secret_test.py`이라는 로컬 파일에 다음과 같은 Python 코드 함수를 저장합니다.

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"


def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
    """
```

```
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret_value = response.get("SecretString")
    message = (
        f"Failed to retrieve secret {secret_name}."
        if secret_value is None
        else f"Successfully retrieved secret {secret_name}."
    )
    iot_client.publish(topic=send_topic, payload=message)
    print("Published: " + message)
```

`get_secret_value` 함수는 `SecretId` 값에 암호의 이름 또는 ARN을 지원합니다. 이 예에서는 암호 이름을 사용합니다. 이 예제 암호에서 AWS IoT Greengrass는 `{"test":"abcdefghi"}` 키-값 쌍을 반환합니다.

> **Important**
>
> 사용자 정의한 Lambda 함수에서 보안 암호를 안전하게 취급해야 하고, 보안 암호에 저장된 기밀 데이터를 로그에 기록하지 말아야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Lambda 함수 로깅 및 디버깅의 위험 완화를 참조하십시오. 이 설명서에서는 특별히 교체 기능을 언급하지만 권장 사항은 Greengrass Lambda 함수에도 적용됩니다.

4. 다음 항목을 `secret_test_python.zip`라는 파일로 압축합니다. ZIP 파일을 만들 때 상위 폴더가 아닌 코드 및 종속성만 포함합니다.

   - secret_test.py. 앱 로직.
   - greengrasssdk. 모든 Python Greengrass Lambda 함수에 대한 필수 라이브러리입니다.

   이것이 Lambda 함수 배포 패키지입니다.

4단계: Lambda 함수 생성

이 단계에서는 AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성한 후 배포 패키지를 사용하도록 구성합니다. 그런 다음 함수 버전을 게시하고 별칭을 생성합니다.

1. 먼저, Lambda 함수를 생성합니다.

a. AWS Management Console에서 [Services]를 선택한 다음 AWS Lambda 콘솔을 엽니다.

b. 함수 생성을 선택한 다음 새로 작성을 선택합니다.

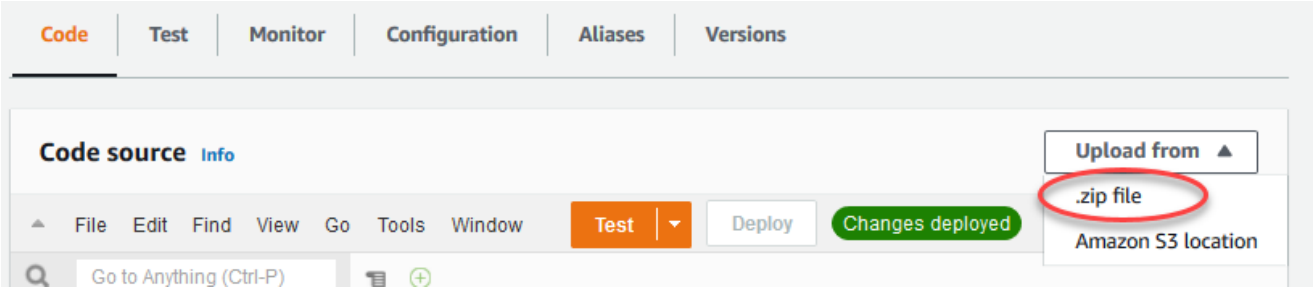
c. 기본 정보 섹션에서 다음 값을 사용합니다.

- [함수 이름(Function name)]에 **SecretTest**을 입력합니다.
- 실행 시간에서 Python 3.7을 선택합니다.
- 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.

d. 페이지 하단에서 함수 생성을 선택합니다.

2. 이제 핸들러를 등록하고 Lambda 함수 배포 패키지를 업로드합니다.

a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



b. 업로드를 선택한 다음 `secret_test_python.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.

c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

- 실행 시간에서 Python 3.7을 선택합니다.
- 핸들러에 **secret_test.function_handler**를 입력합니다.

d. Save를 선택합니다.

> **Note**
>
> AWS Lambda 콘솔의 테스트 버튼은 이 함수와 함께 작동하지 않습니다. AWS IoT Greengrass 코어 SDK에는 AWS Lambda 콘솔에서 Greengrass Lambda 함수를 독립적으로 실행하는 데 필요한 모듈이 포함되어 있지 않습니다. 이러한 모듈(예: `greengrass_common`)은 Greengrass 코어에 배포된 후 함수에 제공됩니다.

3. 이제 Lambda 함수의 첫 번째 버전을 게시하고 버전의 별칭을 생성합니다.

> **Note**
>
> Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

a. [Actions] 메뉴에서 [Publish new revision]을 선택합니다.

b. 버전 설명에 **`First version`**을 입력한 후 게시를 선택합니다.

c. SecretTest: 1 구성 페이지의 작업 메뉴에서 별칭 생성을 선택합니다.

d. [Create a new alias] 페이지에서 다음 값을 사용합니다.

- 이름(Name)에 **`GG_SecretTest`**을 입력합니다.
- 버전에서 1을 선택합니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

e. 생성을 선택합니다.

이제 Greengrass 그룹에 Lambda 함수를 추가하고 암호 리소스를 연결할 준비가 되었습니다.

5단계: Greengrass 그룹에 함수 추가

AWS IoT 이 단계에서는 의 Greengrass 그룹에 Lambda 함수를 추가합니다.

1. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
2. 내 Lambda 함수 섹션에서 추가를 선택합니다.
3. Lambda 함수의 경우 SecretTest를 선택합니다.
4. Lambda 함수 버전에서 게시한 버전에 대한 별칭을 선택합니다.

그런 다음 Lambda 함수의 수명 주기를 구성합니다.

1. Lambda 함수 구성 섹션에서 다음과 같이 업데이트하십시오.

> **Note**
>
> 비즈니스 사례에서 요구하지 않는 한 컨테이너화 없이 Lambda 함수를 실행하는 것이 좋습니다. 이렇게 하면 디바이스 리소스를 구성하지 않고도 디바이스 GPU와 카메라에 액세스할 수 있습니다. 컨테이너화 없이 실행하는 경우 AWS IoT Greengrass Lambda 함수에 대한 루트 액세스 권한도 부여해야 합니다.

a. 컨테이너화 없이 실행하려면:

- 시스템 사용자 및 그룹의 경우 **Another user ID/group ID**을(를) 선택합니다. 시스템 사용자 ID에 **0**을(를) 입력합니다. 시스템 그룹 ID에 **0**을(를) 입력합니다.

  이렇게 하면 Lambda 함수를 루트로 실행할 수 있습니다. 작업 실행 방법에 대한 자세한 내용은 the section called “그룹에 있는 Lambda 함수의 기본 액세스 자격 증명 설정” 섹션을 참조하세요.

  > **Tip**
  >
  > 또한 Lambda 함수에 루트 액세스 권한을 부여하도록 `config.json` 파일을 업데이트해야 합니다. 이 절차는 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

- Lambda 함수 컨테이너화의 경우 컨테이너 없음을 선택합니다.

  속도 제어 자동화 실행에 대한 자세한 내용은 the section called “Lambda 함수 컨테이너화 선택 시 고려 사항” 섹션을 참조하세요.

- 시간 제한에 **10 seconds**를 입력합니다.
- 고정된 경우 True를 선택합니다.

  자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.

- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

b. 대신 컨테이너화 모드에서 실행하려면:

> **Note**
>
> 비즈니스 사례에서 요구하는 경우가 아니면 컨테이너화 모드에서 실행하지 않는 것이 좋습니다.

- 시스템 사용자 및 그룹의 경우 그룹 기본값 사용을 선택합니다.
- Lambda 함수 컨테이너화의 경우 그룹 기본값 사용을 선택합니다.
- 메모리 제한에 **`1024 MB`**를 입력합니다.
- 시간 제한에 **`10 seconds`**를 입력합니다.
- 고정된 경우 True를 선택합니다.

  자세한 내용은 the section called “수명 주기 구성” 섹션을 참조하세요.
- 추가 매개 변수에서 /sys 디렉터리에 대한 읽기 액세스에 대해 활성화를 선택합니다.

2. Lambda 함수 추가를 선택합니다.

그 다음에는 암호 리소스를 함수에 연계합니다.

6단계: 암호 리소스를 Lambda 함수에 연결합니다.

이 단계에서는 암호 리소스를 Greengrass 그룹의 Lambda 함수에 연결합니다. 그러면 리소스가 함수에 연결되는데, 이를 통해 함수가 로컬 암호 값을 가져올 수 있습니다.

1. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
2. SecretTest 함수를 선택합니다.
3. 함수의 세부정보 페이지에서 리소스를 선택합니다.
4. 비밀 섹션으로 스크롤하여 연결을 선택합니다.
5. MyTestSecret을 선택한 다음 연결을 선택합니다.

7단계: Greengrass 그룹에 구독 추가

이 단계에서는 AWS IoT 및 Lambda 함수를 사용하여 메시지를 교환할 수 있는 구독을 추가합니다. 구독에서 AWS IoT를 사용하여 함수를 호출하고, 함수를 사용하여 출력 데이터를 AWS IoT에 보낼 수 있습니다.

1. 그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.
2. AWS IoT를 사용해 함수에 메시지를 게시할 수 있는 구독을 생성합니다.

   그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.
3. 구독 생성 페이지에서 다음과 같이 원본과 대상을 구성합니다.

   a. 소스 유형에서 Lambda 함수를 선택한 다음 IoT Cloud를 선택합니다.

   b. 대상 유형에서 서비스를 선택한 다음 SecretTest를 선택합니다.

   c. 주제 필터에서 **`secrets/input`**을(를) 입력한 다음 구독 생성을 선택합니다.
4. 두 번째 구독을 추가합니다. 구독 탭을 선택하고 구독 추가를 선택한 다음, 다음과 같이 원본과 대상을 구성합니다.

   a. 소스 유형에서 서비스를 선택한 다음 SecretTest를 선택합니다.

   b. 대상 유형에서 Lambda 함수를 선택한 다음 IoT Cloud를 선택합니다.

   c. 주제 필터에서 **`secrets/output`**을(를) 입력한 다음 구독 생성을 선택합니다.

8단계: Greengrass 그룹 배포

코어 디바이스에 그룹을 배포합니다. 배포 중에는 Secrets Manager에서 암호 값을 AWS IoT Greengrass가 가져오고 코어에서 암호화된 로컬 복사본을 생성합니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 데몬이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 `root`에 대한 `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon` 입력이 포함되어 있는 경우에는 데몬이 실행 중인 것입니다.

   > **Note**
   >
   > 경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

   b. 데몬을 시작하려면:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. 그룹 구성 페이지에서 배포를 선택합니다.
3. a. Lambda 함수 탭의 시스템 Lambda 함수 섹션에서 IP 감지기를 선택하고 편집을 선택합니다.
   b. IP 감지기 설정 편집 대화 상자에서 MQTT 브로커 엔드포인트 자동 탐지 및 재정의를 선택합니다.
   c. Save를 선택합니다.

   이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

   > **Note**
   > 메시지가 표시되면 Greengrass 서비스 역할을 생성할 권한을 부여하고 이 권한을 현재 AWS 리전의 AWS 계정과 연결합니다. 이 역할은 AWS IoT Greengrass가 AWS 서비스의 리소스에 액세스할 수 있습니다.

   배포 페이지에 배포 타임스탬프, 버전 ID, 상태가 표시됩니다. 완료되면 배포에 대해 성공적으로 완료했습니다 상태가 표시되어야 합니다.

   문제 해결에 대한 도움말은 문제 해결 단원을 참조하십시오.

Lambda 함수 테스트

1. AWS IoT 홈 페이지에서 테스트를 선택합니다.
2. 구독에서 다음 값을 사용한 다음 주제 구독을 선택합니다.

속성	값
구독 주제	암호/출력
MQTT 페이로드 디스플레이	페이로드를 문자열로 표시

3. 주제 게시에서 다음 값을 사용한 후 게시를 선택하여 함수를 간접 호출합니다.

속성	값
주제	암호/입력
메시지	기본 메시지를 유지합니다. 메시지를 게시하여 Lambda 함수를 간접 호출하지만, 이 자습서의 함수는 메시지 본문을 처리하지 않습니다.

성공하면 함수가 "성공" 메시지를 게시합니다.

다음 사항도 참조하세요.

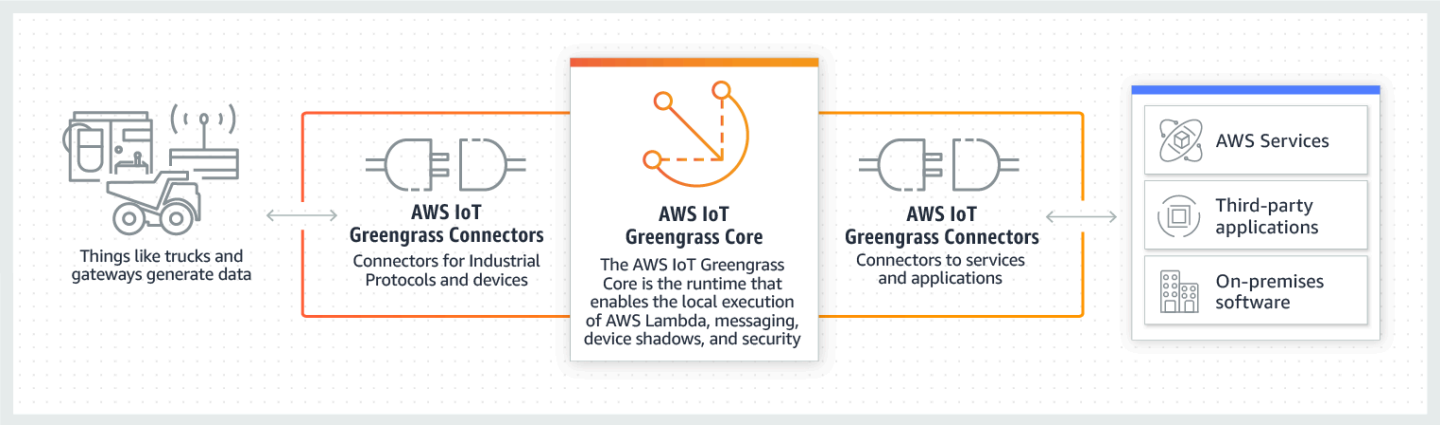
- 코어에 암호 배포

Greengrass 커넥터를 사용하여 서비스 및 프로토콜과 통합

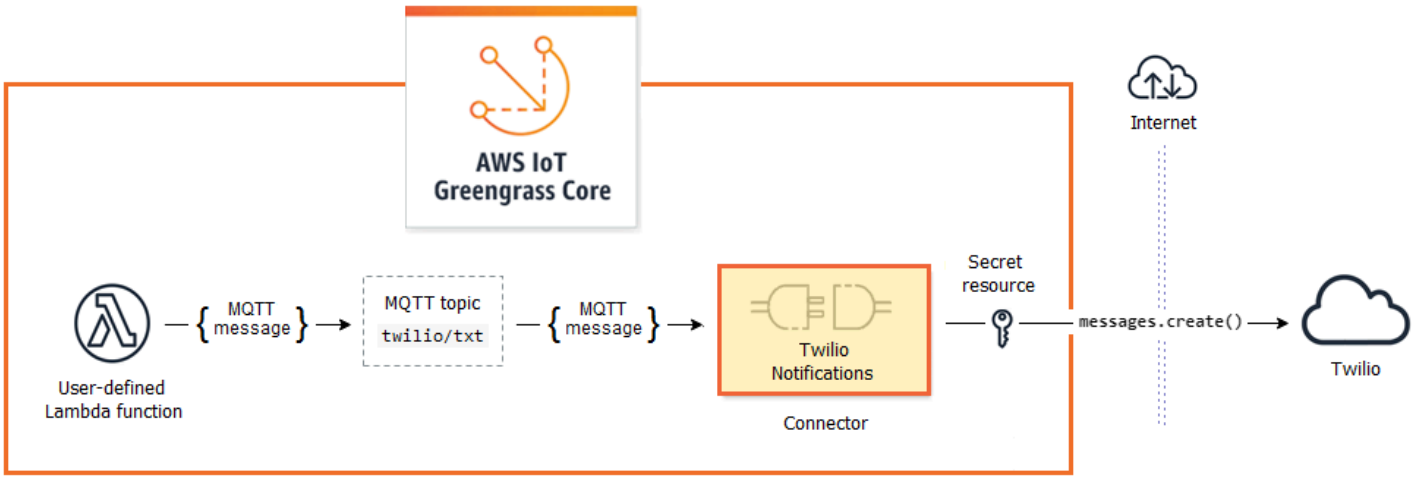
이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

AWS IoT Greengrass의 Connectors는 로컬 인프라, 디바이스 프로토콜, AWS, 기타 클라우드 서비스와 기본적으로 통합되는 모듈입니다. 커넥터를 사용하면 새 프로토콜 및 API를 학습하는 시간을 단축하여 더 많은 시간을 업무에 중요한 로직에 집중할 수 있습니다.

다음 다이어그램은 커넥터가 AWS IoT Greengrass 랜드스케이프에 적합할 수 있는 지점을 보여줍니다.



다수의 커넥터는 MQTT 메시지를 사용하여 그룹 내의 디바이스 및 Lambda 함수 또는 AWS IoT 및 로컬 섀도우 서비스와 통신합니다. 다음 예제에서 AWS Secrets Manager 커넥터는 사용자 정의 Lambda 함수에서 MQTT 메시지를 수신하고, 의 암호에 대한 로컬 참조를 사용하며, Twilio API를 호출합니다.



이 솔루션을 생성하는 자습서는 the section called “커넥터 시작하기(콘솔)” 및 the section called “커넥터 시작하기(CLI)” 단원을 참조하십시오.

Greengrass 커넥터는 디바이스 기능을 확장하거나 단일 용도 디바이스를 생성하는 데 도움이 될 수 있습니다. 커넥터를 사용하여 다음을 수행할 수 있습니다.

- 재사용 가능한 비즈니스 로직을 구현합니다.
- AWS 및 타사 서비스를 비롯해 클라우드 및 로컬 서비스와 상호 작용합니다.
- 디바이스 데이터를 수집해 처리합니다.
- MQTT 주제 구독 및 사용자 정의 Lambda 함수를 사용하여 디바이스 간 직접 호출을 가능하게 합니다.

AWS는 공통 서비스 및 데이터 원본과의 상호 작용을 간소화하는 Greengrass 커넥터 세트를 제공합니다. 이와 같은 미리 빌드된 모듈은 로깅 및 진단, 보충, 산업용 데이터 처리, 경보 및 메시징 시나리오를 지원합니다. 자세한 내용은 the section called “AWS에서 제공한 Greengrass 커넥터” 섹션을 참조하세요.

요구 사항

커넥터를 사용하려면 다음 사항을 염두에 두십시오.

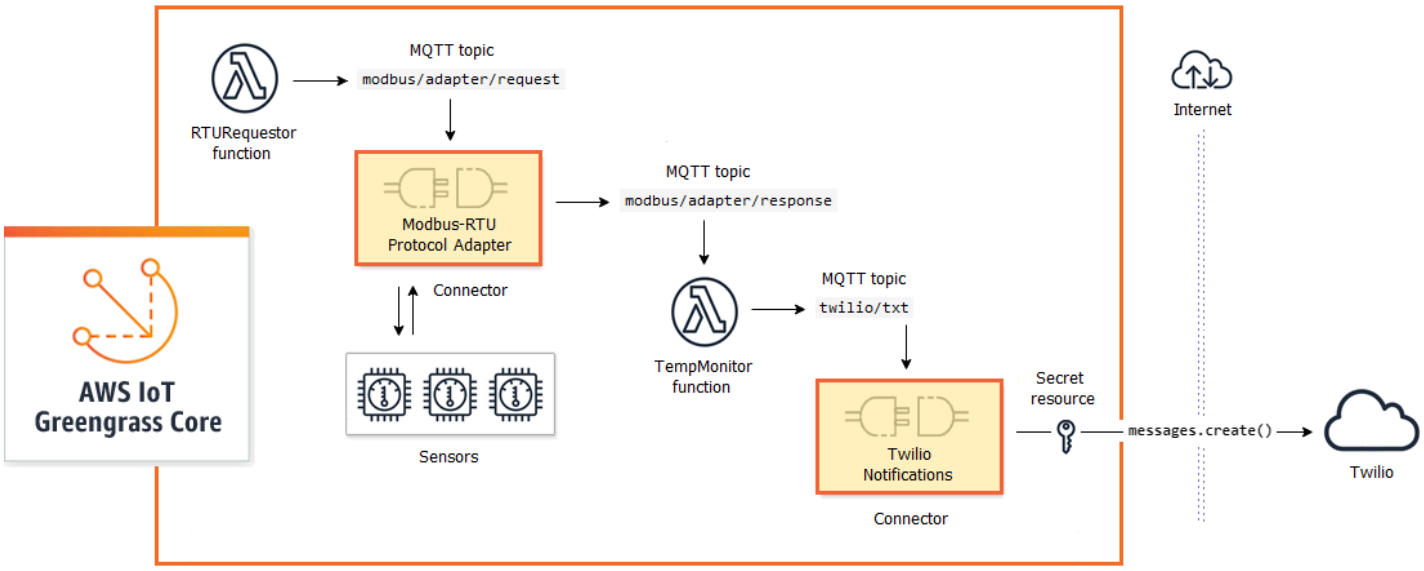
- 사용 중인 각 커넥터의 요구 사항을 충족해야 합니다. 이러한 요구 사항에는 최소 AWS IoT Greengrass 코어 소프트웨어 버전, 디바이스 사전 조건, 필수 권한 및 제한이 포함될 수 있습니다. 자세한 내용은 the section called “AWS에서 제공한 Greengrass 커넥터” 섹션을 참조하세요.
- Greengrass 그룹에는 지정된 커넥터의 구성된 인스턴스가 하나만 포함될 수 있습니다. 그러나 여러 구독에서 인스턴스를 사용할 수 있습니다. 자세한 내용은 the section called “구성 파라미터” 섹션을 참조하세요.
- Greengrass 그룹의 기본 컨테이너화가 컨테이너 없음으로 설정된 경우 그룹의 커넥터는 컨테이너화 없이 실행되어야 합니다. 컨테이너 없음 모드를 지원하는 커넥터를 찾으려면 the section called “AWS에서 제공한 Greengrass 커넥터” 단원을 참조하십시오.

Greengrass 커넥터 사용

커넥터란 일종의 그룹 구성 요소입니다. 디바이스, 사용자 정의 Lambda 함수 등의 기타 그룹 구성 요소와 마찬가지로, 사용자가 커넥터를 그룹에 추가하고 커넥터의 설정을 구성하며 커넥터를 AWS IoT Greengrass 코어에 배포합니다. 커넥터는 코어 환경에서 실행됩니다.

일부 커넥터는 간단한 독립형 애플리케이션으로 배포할 수 있습니다. 예를 들어 커넥터는 코어 디바이스에서 시스템 지표를 읽은 다음 분석을 위해 AWS IoT Device Defender에 전송합니다.

기타 커넥터는 더욱 광범위한 솔루션에서 빌딩 블록으로 추가할 수 있습니다. 다음 예제 솔루션에서는 Modbus-RTU Protocol Adapter 커넥터를 사용하여 센서의 메시지를 처리하고 Twilio Notifications 커넥터를 사용하여 Twilio 메시지를 트리거합니다.



일반적으로 솔루션에는 커넥터 옆에서 커넥터가 전송하거나 수신하는 데이터를 처리하는 사용자 정의 Lambda 함수가 포함되어 있습니다. 이 예제에서 TempMonitor 함수는 Modbus-RTU Protocol Adapter에서 데이터를 수신하고, 일부 비즈니스 로직을 실행한 다음 데이터를 에 보냅니다.

솔루션을 생성해 배포하려면 다음과 같은 일반적을 프로세스를 따릅니다.

1. 상위 수준 데이터 흐름을 시작합니다. 사용해야 하는 데이터 소스, 데이터 채널, 서비스, 프로토콜 및 리소스를 식별합니다. 이 예제 솔루션에서는 Modbus RTU 프로토콜, 물리적 Modbus 직렬 포트 및 Twilio에 대한 데이터가 포함되어 있습니다.
2. 솔루션에 포함할 커넥터를 식별하고 그룹에 추가합니다. 예제 솔루션은 Modbus-RTU 프로토콜 어댑터와 Twilio 알림을 사용합니다. 시나리오에 적용할 커넥터를 찾고 개별 요구 사항에 대해 알아보려면 the section called “AWS에서 제공한 Greengrass 커넥터” 단원을 참조하십시오.
3. 사용자 정의 Lambda 함수, 디바이스 또는 리소스가 필요한지 확인한 후 생성하여 그룹에 추가합니다. 여기에는 비즈니스 로직이 포함되어 있거나 솔루션에서 다른 엔터티에 필요한 형식으로 데이터를 처리하는 함수가 포함되어 있을 수 있습니다. 이 예제 솔루션에서는 함수를 사용하여 Modbus RTU 요청을 보내고 Twilio 알림을 시작합니다. 또한 Modbus RTU 직렬 포트에 필요한 로컬 디바이스 리소스와 Twilio 인증 토큰에 필요한 보안 암호 리소스가 포함되어 있습니다.

> **Note**
>
> 보안 암호 리소스는 AWS Secrets Manager의 암호, 토큰, 및 기타 보안 암호를 참조합니다. 암호는 커넥터 및 Lambda 함수가 서비스와 애플리케이션에 인증하는 데 사용할 수 있습니다. 기본적으로 AWS IoT Greengrass는 "greengrass-"로 시작하는 이름의 보안 암호에 액세스할 수 있습니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

4. 솔루션의 개체가 MQTT 메시지를 교환하도록 허용하는 구독을 생성합니다. 커넥터가 이 구독에 사용되는 경우 커넥터 및 메시지 소스 또는 대상은 커넥터에서 지원하는 미리 정의된 주제 구문을 사용해야 합니다. 자세한 내용은 the section called “입력 및 출력” 섹션을 참조하세요.
5. Greengrass 코어에 그룹을 배포합니다.

커넥터 생성 및 배포에 대한 자세한 내용은 다음 튜토리얼을 참조하세요.

- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

구성 파라미터

다수의 커넥터가 동작 또는 출력을 사용자 지정하는 데 사용할 수 있는 파라미터를 제공합니다. 이러한 파라미터는 초기화 중, 실행 시간에 또는 커넥터 수명 주기의 기타 시점에 사용됩니다.

파라미터 유형 및 사용은 커넥터별로 다릅니다. 예를 들어, SNS 커넥터에는 기본 SNS 주제를 구성하는 파라미터가 있고, Device Defender에는 데이터 샘플링 비율을 구성하는 파라미터가 있습니다.

한 그룹 버전에 여러 커넥터가 포함될 수 있지만, 지정된 커넥터의 인스턴스는 한 번에 하나만 있을 수 있습니다. 즉, 그룹의 각 커넥터에는 활성 구성이 하나만 있을 수 있습니다. 그러나 커넥터 인스턴스는 그룹의 여러 구독에 사용할 수 있습니다. 예를 들어 여러 디바이스에서 Kinesis Firehose 커넥터에 데이터를 전송하도록 허용하는 구독을 생성할 수 있습니다.

그룹 리소스 액세스에 사용되는 파라미터

Greengrass 커넥터는 그룹 리소스를 사용하여 코어 디바이스의 파일 시스템, 포트, 주변 장치 및 기타 로컬 리소스에 액세스합니다. 커넥터가 그룹 리소스에 액세스해야 하는 경우 커넥터는 관련 구성 파라미터를 제공합니다.

그룹 리소스에는 다음 항목이 포함됩니다.

- 로컬 리소스. Greengrass 코어 디바이스에 있는 디렉터리, 파일, 포트, 핀 및 주변 장치.
- Machine Learning 리소스. 클라우드에서 교육되고 로컬 추론을 위해 코어로 배포되는 기계 학습 모델.
- 보안 암호 리소스. AWS Secrets Manager의 암호, 키, 토큰 또는 임의 텍스트의 암호화된 로컬 복사본. 커넥터는 이러한 로컬 암호에 안전하게 액세스할 수 있으며 이러한 로컬 암호를 사용하여 서비스 또는 로컬 인프라를 인증할 수 있습니다.

예를 들어, 의 파라미터를 사용하면 호스트 `/proc` 디렉터리에서 시스템 지표에 액세스할 수 있고, Twilio Notifications의 파라미터를 사용하면 로컬에 저장된 Twilio 인증 토큰에 액세스할 수 있습니다.

커넥터 파라미터 업데이트

커넥터가 Greengrass 그룹에 추가될 때 파라미터가 구성됩니다. 커넥터가 추가된 후 파라미터 값을 변경할 수 있습니다.

- 콘솔에서: 그룹 구성 페이지에서 Connectors(커넥터)를 열고 커넥터의 컨텍스트 메뉴에서 Edit(편집)를 선택합니다.

  > **Note**
  >
  > 커넥터가 다른 암호를 참조하기 위해 나중에 변경되는 보안 암호 리소스를 사용하는 경우 커넥터의 파라미터를 편집해 변경을 확인해야 합니다.

- API에서: 새 구성을 정의하는 커넥터의 다른 버전을 생성합니다.

  AWS IoT Greengrass API는 버전을 사용하여 그룹을 관리합니다. 버전은 변경할 수 없으므로 그룹 구성 요소(예: 그룹의 클라이언트 디바이스, 기능 및 리소스)를 추가하거나 변경하려면 새 구성 요소나 업데이트된 구성 요소의 버전을 생성해야 합니다. 그런 다음 각 구성 요소의 대상 버전을 포함하는 그룹 버전을 생성하고 배포합니다.

커넥터 구성을 변경한 후에는 그룹을 배포하여 변경 사항을 코어에 전파해야 합니다.

입력 및 출력

많은 Greengrass 커넥터는 MQTT 메시지를 전송하고 수신하여 다른 개체와 통신할 수 있습니다. MQTT 통신은 커넥터가 Greengrass 그룹 내의 Lambda 함수, 디바이스 및 기타 커넥터 또는 AWS IoT 및 로컬 섀도우 서비스와 데이터를 교환하도록 허용하는 구독을 통해 제어됩니다. 이러한 통신을 허용

하려면 커넥터가 속한 그룹에서 구독을 생성해야 합니다. 자세한 내용은 the section called “MQTT 메시징 워크플우로의 관리형 구독” 섹션을 참조하세요.

커넥터는 메시지 게시자, 메시지 구독자 또는 둘 다일 수 있습니다. 각 커넥터는 게시 또는 구독하는 MQTT 주제를 정의합니다. 이러한 미리 정의된 주제는 커넥터가 메시지 소스 또는 메시지 대상이 되는 구독에서 사용되어야 합니다. 커넥터에 대한 구독을 구성하는 단계가 포함된 자습서는 the section called “커넥터 시작하기(콘솔)” 및 the section called “커넥터 시작하기(CLI)” 단원을 참조하십시오.

> **Note**
>
> 또한 다수의 커넥터에는 클라우드 또는 로컬 서비스와 상호 작용하기 위한 기본 제공 통신 모드도 있습니다. 이러한 통신 모드는 커넥터에 따라 다르며, 파라미터를 구성하거나 그룹 역할에 권한을 추가해야 할 수 있습니다. 커넥터 요구 사항에 대한 자세한 내용은 the section called “AWS에서 제공한 Greengrass 커넥터” 단원을 참조하십시오.

입력 주제

대부분의 커넥터는 MQTT 주제에 대한 입력 데이터를 수신합니다. 일부 커넥터는 입력 데이터에 대한 여러 주제를 구독합니다. 예를 들어 Serial Stream 커넥터는 다음 두 가지 주제를 지원합니다.

- `serial/+/read/#`
- `serial/+/write/#`

이 커넥터의 경우 읽기 및 쓰기 요청이 해당 주제에 전송됩니다. 구독을 생성할 때 구현에 맞는 주제를 사용해야 합니다.

이전 예에서 + 및 # 문자는 와일드카드입니다. 이러한 와일드카드를 사용하면 구독자가 여러 주제 및 게시자에 대한 메시지를 수신해 게시할 주제를 사용자 지정할 수 있습니다.

- + 와일드카드는 주제 계층 구조 어디든지 나타날 수 있습니다. 이 와일드카드는 하나의 계층 항목으로 대체될 수 있습니다.

  예를 들어 `sensor/+/input` 주제의 경우 메시지를 `sensor/id-123/input` 주제에 게시할 수 있지만 `sensor/group-a/id-123/input`에는 게시할 수 없습니다.

- # 와일드카드는 주제 계층 구조의 끝에만 나타날 수 있습니다. 이 와일드카드는 항목이 0개 이상의 계층 항목으로 대체될 수 있습니다.

예를 들어 `sensor/#` 주제의 경우 메시지를 `sensor/`, `sensor/id-123` 및 `sensor/group-a/id-123` 주제에 게시할 수 있지만 `sensor`에는 게시할 수 없습니다.

와일드카드 문자는 주제를 구독하는 경우에만 유효합니다. 와일드카드가 포함된 주제에는 메시지를 게시할 수 없습니다. 입력 또는 출력 주제 요구 사항에 대해 자세히 알아보려면 커넥터 설명서를 참조하십시오. 자세한 내용은 the section called “AWS에서 제공한 Greengrass 커넥터” 섹션을 참조하세요.

컨테이너화 지원

기본적으로 대부분의 커넥터는 AWS IoT Greengrass에서 관리하는 격리된 런타임 환경의 Greengrass 코어에서 실행됩니다. 컨테이너라고 하는 이러한 런타임 환경은 커넥터와 호스트 시스템 간의 격리를 제공하여 호스트와 커넥터에 대한 보안을 강화합니다.

그러나 이 Greengrass 컨테이너화는 Docker 컨테이너 또는 cgroup이 없는 이전 Linux 커널에서 AWS IoT Greengrass를 실행하는 것과 같은 일부 환경에서는 지원되지 않습니다. 이러한 환경에서 커넥터는 컨테이너 없음 모드에서 실행되어야 합니다. 컨테이너 없음 모드를 지원하는 커넥터를 찾으려면 the section called “AWS에서 제공한 Greengrass 커넥터” 단원을 참조하십시오. 일부 커넥터는 기본적으로 이 모드에서 실행되며 일부 커넥터에서는 격리 모드를 설정할 수 있습니다.

Greengrass 컨테이너화를 지원하는 환경에서는 격리 모드를 컨테이너 없음으로 설정할 수도 있지만 가능하면 Greengrass 컨테이너 모드를 사용하는 것이 좋습니다.

> **Note**
>
> Greengrass 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

커넥터 버전 업그레이드

커넥터 공급자는 기능을 추가하거나 문제를 수정하거나 성능을 향상시키는 새 버전의 커넥터를 출시할 수 있습니다. 사용 가능한 버전 및 관련 변경 사항에 대한 자세한 내용은 각 커넥터의 설명서를 참조하십시오.

AWS IoT 콘솔에서 Greengrass 그룹의 커넥터에 대한 새 버전을 확인할 수 있습니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. Greengrass 그룹에서 사용자 그룹을 선택합니다.

3. 커넥터를 선택하여 그룹에 커넥터를 표시합니다.

   커넥터에 새 버전이 있는 경우 업그레이드 열에 사용 가능 버튼이 나타납니다.

4. 커넥터 버전을 업그레이드하려면:

   a. 커넥터 페이지의 업그레이드 열에서 사용 가능을 선택합니다. 커넥터 업그레이드 페이지가 열리고 해당하는 경우 현재 파라미터 설정이 표시됩니다.

      새 커넥터 버전을 선택하고 필요에 따라 파라미터를 정의한 다음 업그레이드를 선택합니다.

   b. 구독 페이지에서 그룹에 새 구독을 추가하여 커넥터를 소스 또는 대상으로 사용하는 모든 구독을 대체합니다. 그런 다음 이전 구독을 제거합니다.

      구독은 버전별로 커넥터를 참조하므로 그룹에서 커넥터 버전을 변경하면 커넥터가 유효하지 않게 됩니다.

   c. 작업 메뉴에서 배포를 선택하여 변경 사항을 코어에 배포합니다.

AWS IoT Greengrass API에서 커넥터를 업그레이드하려면 업데이트된 커넥터 및 구독을 포함하는 그룹 버전을 생성하고 배포합니다. 그룹에 커넥터를 추가할 때와 동일한 프로세스를 사용합니다. AWS CLI를 사용하여 예제 커넥터를 구성 및 배포하는 방법을 보여주는 자세한 단계는 the section called “커넥터 시작하기(CLI)”를 참조하십시오.

커넥터에 대한 로깅

Greengrass 커넥터에는 Greengrass 로그에 이벤트 및 오류를 쓰는 Lambda 함수가 포함되어 있습니다. 그룹 설정에 따라 로그는 CloudWatch Logs, 로컬 파일 시스템 또는 둘 다에 기록됩니다. 커넥터의 로그에는 해당 함수의 ARN이 포함됩니다. 다음 예제 ARN은 Kinesis Firehose 커넥터에서 나옵니다.

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

기본 로깅 구성은 다음 디렉터리 구조를 사용하여 파일 시스템에 정보 수준 로그를 씁니다.

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

Greengrass 로깅에 대한 자세한 내용은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요.

AWS에서 제공한 Greengrass 커넥터

AWS 일반적인 AWS IoT Greengrass 시나리오를 지원하는 다음 커넥터를 제공합니다. 커넥터의 작동 방식에 대한 자세한 내용은 다음 설명서를 참조하십시오.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- 커넥터 시작하기(콘솔) 또는 커넥터 시작하기(CLI)

커넥터	설명	지원되는 Lambda 런타임	컨테이너 없음 모드 지원
CloudWatch 지표	CloudWatchAmazon에 사용자 지정 지표를 게시합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	예
Device Defender	에 시스템 지표를 전송합니다 AWS IoT Device Defender.	• Python 3.8 * • Python 3.7 • Python 2.7	아니요
Docker 애플리케이션 배포	Docker Compose 파일을 실행하여 코어 디바이스에서 Docker 애플리케이션을 시작합니다.	• Python 3.8 • Python 3.7	예
IoT Analytics	장치 및 센서의 데이터를 로 보냅니다 AWS IoT Analytics.	• Python 3.8 * • Python 3.7 • Python 2.7	예
IoT 이더넷 IP 프로토콜 어댑터	이더넷/IP 디바이스에서 데이터를 수집합니다.	• Java 8	예

커넥터	설명	지원되는 Lambda 런타임	컨테이너 없음 모드 지원
IoT SiteWise	장치 및 센서에서 AWS IoT SiteWise의 자산 속성으로 데이터를 전송합니다.	• Java 8	예
Kinesis Firehose	Amazon Data Firehose 전송 스트림으로 데이터를 전송합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	예
ML Feedback	기계 학습 모델 입력을 클라우드에 게시하고 출력을 MQTT 주제에 게시합니다.	• Python 3.8 • Python 3.7	아니요
ML Image Classification	로컬 이미지 분류 추론 서비스를 실행합니다. 이 커넥터는 여러 플랫폼에 맞는 버전을 제공합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	아니요
ML Object Detection	로컬 객체 감지 추론 서비스를 실행합니다. 이 커넥터는 여러 플랫폼에 맞는 버전을 제공합니다.	• Python 3.8 • Python 3.7	아니요
Modbus-RTU 프로토콜 어댑터	Modbus RTU 디바이스에 요청을 보냅니다.	• Python 3.8 * • Python 3.7 • Python 2.7	아니요
Modbus-TCP 프로토콜 어댑터	ModbusTCP 디바이스에서 데이터를 수집합니다.	• Java 8	예

커넥터	설명	지원되는 Lambda 런타임	컨테이너 없음 모드 지원
Raspberry Pi GPIO	Raspberry Pi 코어 디바이스에 대한 GPIO를 제어합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	아니요
Serial Stream	코어 디바이스에서 읽고 직렬 포트에 씁니다.	• Python 3.8 * • Python 3.7 • Python 2.7	아니요
ServiceNow MetricBase 통합	시계열 지표를 에 ServiceNow MetricBase 게시합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	예
SNS	Amazon SNS 주제로 메시지를 전송합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	예
Splunk 통합	Splunk HEC에 데이터를 게시합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	예
Twilio 알림	Twilio 문자 또는 음성 메시지를 트리거합니다.	• Python 3.8 * • Python 3.7 • Python 2.7	예

* Python 3.8 런타임을 사용하려면 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만들어야 합니다. 자세한 내용은 커넥터 요구 사항을 참조하십시오.

> **Note**
>
> 커넥터 버전을 Python 2.7에서 Python 3.7로 업그레이드하는 것이 좋습니다. Python 2.7 커넥터에 대한 지속적인 지원은 AWS Lambda 런타임 지원에 따라 달라집니다. 자세한 내용을 알아보려면 런타임 지원 정책에 대한 AWS Lambda 개발자 가이드를 참조하세요.

CloudWatch 메트릭 커넥터

CloudWatch 지표 커넥터는 Greengrass 디바이스의 사용자 지정 지표를 Amazon에 게시합니다. CloudWatch 커넥터는 CloudWatch 지표 게시를 위한 중앙 집중식 인프라를 제공하며, 이를 사용하여 Greengrass 핵심 환경을 모니터링 및 분석하고 로컬 이벤트에 대한 조치를 취할 수 있습니다. 자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 측정치 사용을 참조하십시오.

이 커넥터는 지표 데이터를 MQTT 메시지로 수신합니다. 커넥터는 동일한 네임스페이스에 있는 메트릭을 일괄 처리하여 CloudWatch 정기적으로 게시합니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
5	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/5`
4	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/4`
3	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/3`
2	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/2`

버전	ARN
1	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3 - 5

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

  > **Note**
  >
  > Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 다음 예제에 표시된 것처럼 `cloudwatch:PutMetricData` 작업을 허용하는 Greengrass 그룹 역할에 추가된 AWS Identity and Access Management (IAM) 정책.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
```

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

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

CloudWatch 권한에 대한 자세한 내용은 IAM 사용 설명서의 Amazon CloudWatch 권한 참조를 참조하십시오.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- 다음 예제에 표시된 것처럼 `cloudwatch:PutMetricData` 작업을 허용하는 Greengrass 그룹 역할에 추가된 AWS Identity and Access Management (IAM) 정책.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

CloudWatch 권한에 대한 자세한 내용은 IAM 사용 설명서의 Amazon CloudWatch 권한 참조를 참조하십시오.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Versions 4 - 5

`PublishInterval`

지정된 네임스페이스에 대해 일괄 처리된 지표를 게시하기 전에 대기해야 하는 최대 시간(초). 최대값은 900입니다. 지표가 (배치 처리 없이) 수신될 때 지표를 게시하도록 커넥터를 구성하려면 0을 지정합니다.

커넥터는 동일한 네임스페이스에서 20개의 지표를 수신한 CloudWatch 후 또는 지정된 간격 후에 게시합니다.

> **Note**
>
> 이 커넥터는 게시 이벤트의 순서를 보장하지 않습니다.

AWS IoT 콘솔의 표시 이름: 게시 간격

필수: `true`

유형: `string`

유효값: `0 - 900`

유효한 패턴: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

AWS 리전메트릭을 게시할 대상 CloudWatch 이 값은 기본 Greengrass 지표 리전을 재정의합니다. 교차 리전 지표를 게시하는 경우에만 필요합니다.

AWS IoT 콘솔의 표시 이름: 게시 리전

필수: `false`

유형: `string`

유효한 패턴: `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

커넥터에 할당할 메모리(KB).

AWS IoT 콘솔의 표시 이름: 메모리 크기

필수: `true`

유형: `string`

유효한 패턴: `^[0-9]+$`

`MaxMetricsToRetain`

새 지표로 바뀌기 전에 메모리에 저장할 모든 네임스페이스 간 최대 지표 수. 최소값은 2000입니다.

이러한 제한은 인터넷에 연결되어 있지 않거나 커넥터가 나중에 게시할 지표를 버퍼링하기 시작할 때 적용됩니다. 버퍼가 꽉 차면 가장 오래된 지표가 새 지표로 바뀝니다. 지정된 네임스페이스의 지표는 동일한 네임스페이스의 지표로만 바뀝니다.

> **Note**
>
> 커넥터에 대한 호스트 프로세스가 중단되면 지표가 저장되지 않습니다. 예를 들어, 그룹 배포 중 또는 디바이스가 다시 시작될 때 중단이 발생할 수 있습니다.

AWS IoT 콘솔의 표시 이름: 보존할 최대 지표

필수: `true`

유형: `string`

유효한 패턴: `^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 `GreengrassContainer`이며 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
>
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

유형: `string`

유효한 값: `GreengrassContainer`또는 `NoContainer`

유효한 패턴: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`PublishInterval`

지정된 네임스페이스에 대해 일괄 처리된 지표를 게시하기 전에 대기해야 하는 최대 시간(초). 최대값은 900입니다. 지표가 (배치 처리 없이) 수신될 때 지표를 게시하도록 커넥터를 구성하려면 0을 지정합니다.

커넥터는 동일한 네임스페이스에서 20개의 메트릭을 수신한 후 또는 지정된 간격 후에 게시합니다. CloudWatch

> **Note**
> 이 커넥터는 게시 이벤트의 순서를 보장하지 않습니다.

AWS IoT 콘솔의 표시 이름: 게시 간격

필수: `true`

유형: `string`

유효값: `0 - 900`

유효한 패턴: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

AWS 리전메트릭을 게시할 대상 CloudWatch 이 값은 기본 Greengrass 지표 리전을 재정의합니다. 교차 리전 지표를 게시하는 경우에만 필요합니다.

AWS IoT 콘솔의 표시 이름: 게시 리전

필수: `false`

유형: `string`

유효한 패턴: `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

커넥터에 할당할 메모리(KB).

AWS IoT 콘솔의 표시 이름: 메모리 크기

필수: `true`

유형: `string`

유효한 패턴: `^[0-9]+$`

`MaxMetricsToRetain`

새 지표로 바뀌기 전에 메모리에 저장할 모든 네임스페이스 간 최대 지표 수. 최소값은 2000입니다.

이러한 제한은 인터넷에 연결되어 있지 않거나 커넥터가 나중에 게시할 지표를 버퍼링하기 시작할 때 적용됩니다. 버퍼가 꽉 차면 가장 오래된 지표가 새 지표로 바뀝니다. 지정된 네임스페이스의 지표는 동일한 네임스페이스의 지표로만 바뀝니다.

> **Note**
>
> 커넥터에 대한 호스트 프로세스가 중단되면 지표가 저장되지 않습니다. 예를 들어, 그룹 배포 중 또는 디바이스가 다시 시작될 때 중단이 발생할 수 있습니다.

AWS IoT 콘솔의 표시 이름: 보존할 최대 지표

필수: `true`

유형: `string`

유효한 패턴: `^([2-9]\d{3}|[1-9]\d{4,})$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 CloudWatch Metrics 커넥터가 `ConnectorDefinition` 포함된 초기 버전을 사용하여 를 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyCloudWatchMetricsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/
versions/4",
            "Parameters": {
                "PublishInterval" : "600",
                "PublishRegion" : "us-west-2",
                "MemorySize" : "16",
                "MaxMetricsToRetain" : "2500",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 설명은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 주제에 대한 메트릭을 수락하고 메트릭을 게시합니다. CloudWatch 입력 메시지는 JSON 형식이어야 합니다.

구독의 주제 필터

`cloudwatch/metric/put`

메시지 속성

`request`

이 메시지에는 지표에 대한 정보가 포함되어 있습니다.

요청 객체에는 CloudWatch에 게시할 지표 데이터가 포함되어 있습니다. 지표 값은 `PutMetricData` API의 사양을 충족해야 합니다. `namespace`, `metricData.metricName` 및 `metricData.value` 속성만 필요합니다.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`namespace`

이 요청의 메트릭 데이터에 대한 사용자 정의 네임스페이스입니다. CloudWatch 네임스페이스를 지표 데이터 포인트의 컨테이너로 사용합니다.

> **Note**
>
> 예약 문자어 `AWS/`로 시작하는 네임스페이스는 지정할 수 없습니다.

필수: `true`

유형: `string`

유효한 패턴: `[^:].*`

`metricData`

지표에 대한 데이터.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`metricName`

지표의 이름.

필수: `true`

유형: `string`

`dimensions`

지표와 연결된 차원. 차원은 지표와 해당 데이터에 대한 추가 정보를 제공합니다. 지표는 최대 10개 차원을 정의할 수 있습니다.

이 커넥터에는 이름이 `coreName`인 차원이 자동으로 포함되며, 여기서 값은 코어의 이름입니다.

필수: `false`

유형: 다음 속성을 포함하는 차원 객체의 `array`

`name`

차원 이름입니다.

필수: `false`

유형: `string`

`value`

차원 값입니다.

필수: `false`

유형: `string`

`timestamp`

지표 데이터가 수신된 시간, `Jan 1, 1970 00:00:00 UTC` 이후 밀리초로 표시됩니다. 이 값을 생략하면 커넥터는 메시지를 수신한 시간을 사용합니다.

필수: `false`

유형: `timestamp`

> **Note**
>
> 이 커넥터의 버전 1~4를 사용하는 경우 단일 소스에서 여러 지표를 전송할 때 각 지표에 대한 타임스탬프를 개별적으로 검색하는 것이 좋습니다. 변수를 사용하여 타임스탬프를 저장하지 마세요.

`value`

지표에 대한 값.

> **Note**
>
> CloudWatch 너무 작거나 너무 큰 값은 거부합니다. 값의 범위는 `8.515920e-109` ~ `1.174271e+108`(기본 10) 또는 `2e-360` ~ `2e360`(기본 2) 이어야 합니다. 특수 값(예: `NaN`, `+Infinity`, `-Infinity`)은 지원되지 않습니다.

필수: `true`

유형: `double`

`unit`

측정치의 단위입니다.

필수: `false`

유형: `string`

유효값: `Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/Second, Terabits/Second, Count/Second, None`

Limits

CloudWatch `PutMetricData`API에서 부과하는 모든 제한은 이 커넥터를 사용할 때 지표에 적용됩니다. 다음은 중요한 제한 사항입니다.

- API 페이로드에 대한 40KB 제한
- API 요청당 20개 지표
- `PutMetricData` API에 대한 150개의 초당 트랜잭션(TPS)

자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch 한도를 참조하십시오.

입력 예

```
{
   "request": {
       "namespace": "Greengrass",
       "metricData":
           {
               "metricName": "latency",
               "dimensions": [
                   {
                       "name": "hostname",
                       "value": "test_hostname"
                   }
               ],
               "timestamp": 1539027324,
```

```
                "value": 123.0,
                "unit": "Seconds"
            }
    }
}
```

출력 데이터

이 커넥터는 상태 정보를 MQTT 주제에 출력 데이터로 게시합니다.

구독의 주제 필터

```
cloudwatch/metric/put/status
```

출력 예: 성공

응답에는 메트릭 데이터의 네임스페이스와 응답의 `RequestId` 필드가 포함됩니다. CloudWatch

```
{
   "response": {
        "cloudwatch_rid":"70573243-d723-11e8-b095-75ff2EXAMPLE",
        "namespace": "Greengrass",
        "status":"success"
    }
}
```

출력 예: 실패

```
{
   "response" : {
        "namespace": "Greengrass",
        "error": "InvalidInputException",
        "error_message":"cw metric is invalid",
        "status":"fail"
   }
}
```

> **Note**
>
> 커넥터가 재시도 가능한 오류(예: 연결 오류)를 감지하면 다음 배치에서 게시를 재시도합니다.

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.

   그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 커넥터를 추가하고 해당 파라미터를 구성합니다.

   c. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.

4. 그룹을 배포합니다.
5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))
```

```
publish_basic_message()

def lambda_handler(event, context):
    return
```

라이선스

CloudWatch Metrics 커넥터에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
5	입력 데이터의 중복 타임스탬프에 대한 지원을 추가하도록 수정했습니다.
4	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	과도한 로깅을 줄이도록 고정합니다.

버전	변경
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드”을 참조하십시오.

다음 사항도 참조하십시오.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- Amazon 사용 CloudWatch 설명서에서 Amazon CloudWatch 메트릭스 사용하기
- PutMetricData아마존 CloudWatch API 레퍼런스에서

Defice Defender 커넥터

Device Defender 커넥터는 관리자에게 Greengrass 코어 디바이스의 상태 변경을 알립니다. 그러면 손상된 디바이스를 나타낼 수 있는 비정상적인 동작을 식별할 수 있습니다.

이 커넥터는 코어 디바이스의 `/proc` 디렉터리에서 시스템 지표를 읽은 다음 AWS IoT Device Defender에 게시합니다. 지표 보고 세부 정보는 AWS IoT 개발자 안내서의 디바이스 지표 문서 사양을 참조하십시오.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
3	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/3`
2	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/2`

버전	ARN
1	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

  > **Note**
  >
  > Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- Detect 기능을 사용하여 위반을 추적하도록 구성된 AWS IoT Device Defender. 자세한 정보는 AWS IoT 개발자 안내서의 감지 사용 방법을 참조하세요.
- `/proc` 디렉터리를 가리키는 Greengrass 그룹의 로컬 볼륨 리소스 이 리소스는 다음 속성을 사용해야 합니다.
  - 소스 경로: `/proc`
  - 대상 경로: `/host_proc`(또는 유효한 패턴과 일치하는 값)
  - AutoAddGroupOwner: `true`
- Greengrass 코어에 설치된 psutil 라이브러리입니다. 버전 5.7.0은 커넥터와 함께 사용하도록 확인된 최신 버전입니다.

- Greengrass 코어에 설치된 cbor 라이브러리입니다. 버전 1.0.0은 커넥터와 함께 사용하도록 확인된 최신 버전입니다.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Detect 기능을 사용하여 위반을 추적하도록 구성된 AWS IoT Device Defender. 자세한 정보는 AWS IoT 개발자 안내서의 감지 사용 방법을 참조하세요.
- `/proc` 디렉터리를 가리키는 Greengrass 그룹의 로컬 볼륨 리소스 이 리소스는 다음 속성을 사용해야 합니다.
  - 소스 경로: `/proc`
  - 대상 경로: `/host_proc`(또는 유효한 패턴과 일치하는 값)
  - AutoAddGroupOwner: `true`
- Greengrass 코어에 설치된 psutil 라이브러리입니다.
- Greengrass 코어에 설치된 cbor 라이브러리입니다.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

`SampleIntervalSeconds`

각 지표 수집 및 보고 주기 사이 시간(초). 최소값은 300초(5분)입니다.

AWS IoT 콘솔의 표시 이름: 메트릭 보고 간격

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

`ProcDestinationPath-ResourceId`

`/proc` 볼륨 리소스의 ID.

> **Note**
>
> 이 커넥터에는 리소스에 대한 읽기 전용 액세스 권한이 부여됩니다.

AWS IoT 콘솔의 표시 이름: /proc 디렉토리의 리소스

필수: `true`

형식: `string`

유효한 패턴: `[a-zA-Z0-9_-]+`

`ProcDestinationPath`

`/proc` 볼륨 리소스의 대상 경로.

AWS IoT 콘솔의 표시 이름: /proc 리소스의 대상 경로

필수: `true`

형식: `string`

유효한 패턴: `\/[a-zA-Z0-9_-]+`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Device Defender 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyDeviceDefenderConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/
versions/3",
            "Parameters": {
                "SampleIntervalSeconds": "600",
                "ProcDestinationPath": "/host_proc",
                "ProcDestinationPath-ResourceId": "my-proc-resource"
            }
        }
    ]
```

```
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

출력 데이터

이 커넥터는 보안 지표를 AWS IoT Device Defender에 출력 데이터로 게시합니다.

구독의 주제 필터

```
$aws/things/+/defender/metrics/json
```

> **Note**
>
> AWS IoT Device Defender에서 기대하는 주제 구문입니다. 이 커넥터는 + 와일드카드를 디바이스 이름으로 바꿉니다(예: `$aws/things/thing-name/defender/metrics/json`).

출력 예

지표 보고 세부 정보는 AWS IoT 개발자 안내서의 디바이스 지표 문서 사양을 참조하십시오.

```
{
    "header": {
        "report_id": 1529963534,
        "version": "1.0"
    },
    "metrics": {
        "listening_tcp_ports": {
            "ports": [
                {
```

```
                    "interface": "eth0",
                    "port": 24800
                },
                {
                    "interface": "eth0",
                    "port": 22
                },
                {
                    "interface": "eth0",
                    "port": 53
                }
            ],
            "total": 3
        },
        "listening_udp_ports": {
            "ports": [
                {
                    "interface": "eth0",
                    "port": 5353
                },
                {
                    "interface": "eth0",
                    "port": 67
                }
            ],
            "total": 2
        },
        "network_stats": {
            "bytes_in": 1157864729406,
            "bytes_out": 1170821865,
            "packets_in": 693092175031,
            "packets_out": 738917180
        },
        "tcp_connections": {
            "established_connections":{
                "connections": [
                    {
                    "local_interface": "eth0",
                    "local_port": 80,
                    "remote_addr": "192.168.0.1:8000"
                    },
                    {
                    "local_interface": "eth0",
                    "local_port": 80,
```

```
                    "remote_addr": "192.168.0.1:8000"
                    }
                ],
                "total": 2
            }
        }
    }
}
```

라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	과도한 로깅을 줄이도록 고정합니다.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- AWS IoT 개발자 안내서의 Device Defender

Docker 애플리케이션 배포 커넥터

Greengrass Docker 애플리케이션 배포 커넥터를 사용하면 AWS IoT Greengrass 코어에서 Docker 이미지를 더 쉽게 실행할 수 있습니다. 커넥터는 Docker Compose를 사용하여 `docker-compose.yml` 파일에서 멀티컨테이너 Docker 애플리케이션을 시작합니다. 특히 커넥터는 `docker-compose` 명령을 실행하여 단일 코어 디바이스에서 Docker 컨테이너를 관리합니다. 자세한 내용은 Docker 설명서의 Docker Compose 개요를 참조하십시오. 커넥터는 Amazon Elastic Container Registry(Amazon ECR), Docker 허브 및 프라이빗 Docker 신뢰 레지스트리와 같은 Docker 컨테이너 레지스트리에 저장된 Docker 이미지에 액세스 할 수 있습니다.

Greengrass 그룹을 배포하고 난 후 커넥터는 최신 이미지를 가져오고 Docker 컨테이너를 시작합니다. `docker-compose pull` 및 `docker-compose up` 명령을 실행합니다. 커낵터는 output MQTT topic에 명령의 상태를 게시합니다. 또한 Docker 컨테이너 실행에 대한 상태 정보를 기록합니다. 이렇게 하면 Amazon에서 애플리케이션 로그를 모니터링할 수 CloudWatch 있습니다. 자세한 설명은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요. 또한 커넥터는 Greengrass 대몬(daemon)이 다시 시작될 때마다 Docker 컨테이너를 시작합니다. 코어에서 실행할 수 있는 Docker 컨테이너의 수는 하드웨어에 따라 다릅니다.

Docker 컨테이너는 코어 디바이스의 Greengrass 도메인 밖에서 실행되므로 코어의 프로세스 간 통신(IPC) 에 액세스 할 수 없습니다. 그러나 로컬 Lambda 함수 같은 Greengrass 구성 요소와의 일부 통신 채널을 구성할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너와 통신” 섹션을 참조하세요.

코어 디바이스에서 웹 서버 또는 MySQL 서버를 호스팅하는 등의 시나리오에서 커넥터를 사용할 수 있습니다. Docker 애플리케이션의 로컬 서비스는 서로 통신할 수 있으며, 로컬 환경의 다른 프로세스 및 클라우드 서비스와도 통신할 수 있습니다. 예를 들어, 코어에서 웹 서버를 실행하여 Lambda 함수에서 클라우드의 웹 서비스로 요청을 보낼 수 있습니다.

이 커넥터는 컨테이너 없음 격리 모드에서 실행되므로 Greengrass 컨테이너화 없이 실행되는 Greengrass 그룹에 배포할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
7	`arn:aws:greengrass:  region::/connectors/DockerApplicationDeployment/versions/7`

버전	ARN
6	arn:aws:greengrass:*region*::/connectors/DockerApplicationDeployment/versions/6
5	arn:aws:greengrass:*region*::/connectors/DockerApplicationDeployment/versions/5
4	arn:aws:greengrass:*region*::/connectors/DockerApplicationDeployment/versions/4
3	arn:aws:greengrass:*region*::/connectors/DockerApplicationDeployment/versions/3
2	arn:aws:greengrass:*region*::/connectors/DockerApplicationDeployment/versions/2
1	arn:aws:greengrass:*region*::/connectors/DockerApplicationDeployment/versions/1

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.10 이상.

> **Note**
>
> OpenWrt 배포판에서는 이 커넥터가 지원되지 않습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 커넥터가 실행 중인 Docker 컨테이너를 모니터링하기 위해서는 Greengrass 코어에서 최소 36MB RAM이 필요합니다. 총 메모리 요구 사항은 코어에서 실행되는 Docker 컨테이너의 수에 따라 다릅니다.
- Greengrass 코어에 설치된 Docker Engine 1.9.1 이상입니다. 버전 19.0.3은 커넥터와 함께 사용하도록 확인된 최신 버전입니다.

  `docker` 실행 파일은 `/usr/bin` 또는 `/usr/local/bin` 디렉터리에 있어야 합니다.

  > **Important**
  >
  > Docker 자격 증명의 로컬 복사본을 보호하려면 자격 증명 저장소를 설치하는 것이 좋습니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

  Amazon Linux 배포판에 Docker를 설치하는 방법에 대한 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 Amazon ECS용 Docker 기본 사항을 참조하십시오.
- Greengrass 코어에 설치된 Docker Compose입니다. `docker-compose` 실행 파일은 `/usr/bin` 또는 `/usr/local/bin` 디렉터리에 있어야 합니다.

  다음 Docker Compose 버전은 커넥터와 함께 사용하도록 확인되었습니다.

커넥터 버전	확인된 Docker Compose 버전
7	1.25.4
6	1.25.4

커넥터 버전	확인된 Docker Compose 버전
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- Amazon Simple Storage Service(S3)에 저장된 단일 Docker Compose 파일(예: `docker-compose.yml`) 형식은 코어에 설치된 Docker Compose의 버전과 호환되어야 합니다. 코어에서 사용하기 전에 파일을 테스트해야 합니다. Greengrass 그룹을 배포한 후 파일을 편집하는 경우, 그룹을 다시 배포하여 코어의 로컬 복사본을 업데이트해야 합니다.
- 로컬 Docker 대몬(daemon)을 호출하고 Compose 파일의 로컬 복사본을 저장하는 디렉터리에 쓸 수 있는 권한이 있는 Linux 사용자입니다. 자세한 설명은 코어에서 Docker 사용자 설정 섹션을 참조하세요.
- Compose 파일이 포함된 S3 버킷에서 `s3:GetObject` 작업을 허용하는 Greengrass 그룹 역할에 추가된 정책입니다. 이 권한은 다음 IAM 정책 예제에 나와 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToComposeFileS3Bucket",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

> **Note**
>
> S3 버킷에서 버전 관리를 사용하는 경우, `s3:GetObjectVersion` 작업을 허용하도록 역할을 구성해야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버전 관리 사용을 참조하세요.

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

- Docker Compose 파일이 Amazon ECR에 저장된 Docker 이미지를 참조하는 경우, 다음을 허용하도록 Greengrass 그룹 역할이 구성됩니다.
  - Docker 이미지가 포함된 Amazon ECR 리포지토리에 대한 `ecr:GetDownloadUrlForLayer` 및 `ecr:BatchGetImage` 작업입니다.
  - 리소스에 대한 `ecr:GetAuthorizationToken` 작업입니다.

  리포지토리는 커넥터와 동일한 AWS 계정 및 AWS 리전에 있어야 합니다.

> **Important**
>
> 그룹 역할의 권한은 Greengrass 그룹의 모든 Lambda 함수 및 커넥터에서 수임할 수 있습니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

이러한 권한은 다음 예제 정책에 나와 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetEcrRepositories",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:region:account-id:repository/repository-name"
```

```
            ]
        },
        {
            "Sid": "AllowGetEcrAuthToken",
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

자세한 내용을 알아보려면 Amazon ECR 사용 설명서의 Amazon ECR 리포지토리 정책 예시를 참조하세요.

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

- Docker Compose 파일이 AWS Marketplace에서 Docker 이미지를 참조하는 경우, 커넥터에는 다음과 같은 요구 사항이 있습니다.
  - AWS Marketplace 컨테이너 제품을 구독해야 합니다. 자세한 내용은 AWS Marketplace 구독자 가이드의 컨테이너 제품 검색 및 구독을 참조하십시오.
  - AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 비밀을 지원하도록 구성해야 합니다. 커넥터는 이 기능을 사용하여 AWS Secrets Manager에서 사용자의 비밀을 검색하되 저장하지는 않습니다.
  - Compose 파일에서 참조 된 Docker 이미지를 저장하는 각 AWS Marketplace 레지스트리에 대해 Secrets Manager에서 비밀을 생성해야 합니다. 자세한 설명은 the section called “프라이빗 리포지토리에서 도커 이미지에 액세스” 섹션을 참조하세요.
- Docker Compose 파일이 Docker Hub 같이 Amazon ECR 이외의 레지스트리의 프라이빗 레지스트리에서 Docker 이미지를 참조하는 경우, 커넥터에는 다음과 같은 요구 사항도 있습니다.
  - AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 비밀을 지원하도록 구성해야 합니다. 커넥터는 이 기능을 사용하여 AWS Secrets Manager에서 사용자의 비밀을 검색하되 저장하지는 않습니다.
  - Compose 파일에서 참조된 Docker 이미지를 저장하는 각 프라이빗 리포지토리에 대해 Secrets Manager에서 비밀을 생성해야 합니다. 자세한 설명은 the section called “프라이빗 리포지토리에서 도커 이미지에 액세스” 섹션을 참조하세요.
- 이 커넥터가 포함된 Greengrass 그룹을 배포할 때 Docker 대몬(daemon)이 실행 중이어야 합니다.

프라이빗 리포지토리에서 도커 이미지에 액세스

자격 증명을 사용하여 Docker 이미지에 액세스하는 경우, 커넥터가 여기에 액세스할 수 있도록 허용해야 합니다. 이 작업을 수행하는 방법은 Docker 이미지의 위치에 따라 다릅니다.

Amazon ECR에 저장된 Docker 이미지의 경우, Greengrass 그룹 역할에서 권한 부여 토큰을 얻을 수 있는 권한을 부여합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

다른 프라이빗 리포지토리 또는 레지스트리에 저장된 Docker 이미지의 경우, 로그인 정보를 저장하기 위해 AWS Secrets Manager에서 비밀을 생성해야 합니다. 여기에는 AWS Marketplace에서 구독한 Docker 이미지가 포함됩니다. 각 리포지토리에 대해 하나의 비밀을 생성합니다. Secrets Manager에서 비밀을 업데이트하면 다음에 그룹을 배포할 때 변경 내용이 코어에 전파됩니다.

> **Note**
>
> Secrets Manager는 AWS 클라우드에서 자격 증명, 키 및 기타 비밀을 안전하게 저장하고 관리하는 데 사용할 수 있는 서비스입니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 (AWS Secrets Manager이)란 무엇입니까? 섹션을 참조하십시오.

각 비밀에는 다음 키가 포함되어야 합니다.

키	값
`username`	리포지토리 또는 레지스트리에 액세스하는 데 사용되는 사용자 이름입니다.
`password`	리포지토리 또는 레지스트리 액세스에 사용되는 암호입니다.
`registryUrl`	레지스트리의 엔드포인트입니다. 이것은 Compose 파일의 해당 레지스트리 URL과 일치해야 합니다.

> **Note**
>
> 기본적으로 AWS IoT Greengrass이 비밀에 액세스 할 수 있게 하려면 암호의 이름이 greengrass-로 시작해야 합니다. 그렇지 않으면 Greengrass 서비스 역할이 액세스 권한을 부

여해야 합니다. 자세한 설명은 the section called “AWS IoT Greengrass의 암호 값 가져오기 허용” 섹션을 참조하세요.

AWS Marketplace에서 Docker 이미지에 대한 로그인 정보를 얻으려면

1. `aws ecr get-login-password` 명령을 사용하여 AWS Marketplace에서 Docker 이미지의 암호를 가져오십시오. 자세한 내용은 AWS CLI 명령 레퍼런스의 get-login-password 섹션을 참조하십시오.

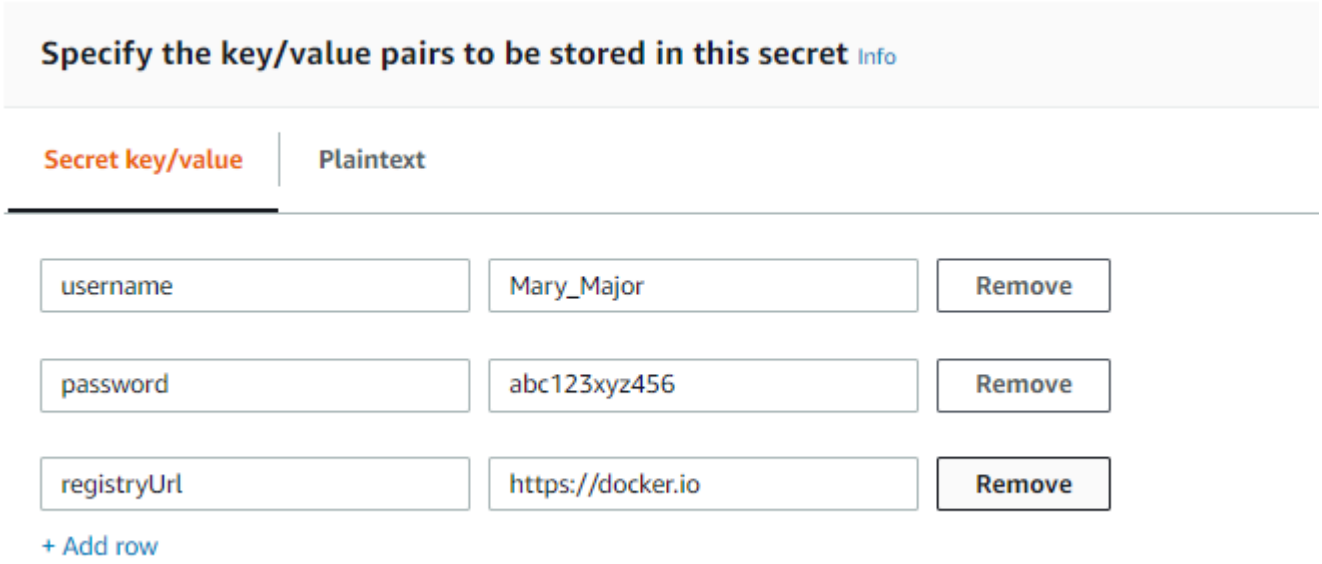
   ```
   aws ecr get-login-password
   ```

2. Docker 이미지의 레지스트리 URL을 검색합니다. AWS Marketplace 웹 사이트를 열고 컨테이너 제품 시작 페이지를 엽니다. 컨테이너 이미지에서 컨테이너 이미지 세부 정보 보기를 선택하여 사용자 이름과 레지스트리 URL을 찾습니다.

검색된 사용자 이름, 암호 및 레지스트리 URL을 사용하여 Compose 파일에서 참조되는 Docker 이미지를 저장하는 각 AWS Marketplace 레지스트리의 비밀을 생성합니다.

비밀을 생성하려면(콘솔)

AWS Secrets Manager 콘솔에서 다른 유형의 비밀을 선택합니다. Specify the key-value pairs to be stored for this secret(이 비밀에 저장할 키-값 페어 지정)에서 `username`, `password` 및 `registryUrl`에 행을 추가합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하세요.



비밀을 만들려면(CLI)

다음의 예제를 참고하여 AWS CLI에서 Secrets Manager `create-secret` 명령을 사용합니다. 자세한 내용은 AWS CLI 명령 참조의 create-secret을 참조하세요.

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string
 [{"username":"Mary_Major"},{"password":"abc123xyz456"},{"registryUrl":"https://
docker.io"}]
```

Important

Docker Compose 파일을 저장하는 `DockerComposeFileDestinationPath` 디렉터리와 프라이빗 리포지토리에서 Docker 이미지에 대한 자격 증명을 보호하는 것은 사용자의 책임입니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Version 7

`DockerComposeFileS3Bucket`

Docker Compose 파일을 포함하는 S3 버킷의 이름입니다. 버킷을 생성할 때는 Amazon Simple Storage Service 사용 설명서에 설명된 버킷 이름 규칙을 따르십시오.

AWS IoT 콘솔에 이름 표시: S3의 Docker Compose 파일

Note

콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3의 Docker Compose 파일의 객체 키입니다. 이름 지정 지침에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 키 및 메타데이터를 참조하세요.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileS3Version`

Amazon S3의 Docker Compose 파일의 객체 버전입니다. 객체 키 이름 지정 지침을 비롯한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버전 관리 사용을 참조하십시오.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `false`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileDestinationPath`

Docker Compose 파일의 복사본을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로입니다. 이 디렉터리는 기존 디렉터리여야 합니다. `DockerUserId`에 지정된 사용자는 이 디렉터리에서 파

일을 작성할 수 있는 권한을 가져야 합니다. 자세한 설명은 the section called “코어에서 Docker 사용자 설정” 섹션을 참조하세요.

> **Important**
>
> 이 디렉터리는 Docker Compose 파일과 프라이빗 리포지토리에서 Docker 이미지에 대한 자격 증명을 저장합니다. 이 디렉터리의 보안을 유지하는 것은 사용자의 책임입니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 로컬 Compose 파일의 디렉터리 경로

필수: `true`

유형: `string`

유효한 패턴 `\/.*\/?`

예제: `/home/username/myCompose`

`DockerUserId`

커넥터가 실행되는 Linux 사용자의 UID입니다. 이 사용자는 코어 디바이스의 `docker` Linux 그룹에 속해야 하며 `DockerComposeFileDestinationPath` 디렉터리에 대한 쓰기 권한이 있어야 합니다. 자세한 설명은 코어에서 Docker 사용자 설정 섹션을 참조하세요.

> **Note**
>
> 반드시 필요한 경우, 외에는 루트로 실행하는 것은 피하는 것이 좋습니다. 루트 사용자를 지정하는 경우, Lambda 함수가 AWS IoT Greengrass 코어에서 루트로 실행되도록 허용해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

AWS IoT 콘솔의 디스플레이 이름: Docker 사용자 ID

필수: `false`

유형: `string`

유효한 패턴: `^[0-9]{1,5}$`

`AWSSecretsArnList`

프라이빗 리포지토리의 Docker 이미지에 액세스하는 데 사용되는 로그인 정보가 포함된 AWS Secrets Manager의 비밀의 Amazon 리소스 이름(ARN)입니다. 자세한 설명은 the section called “프라이빗 리포지토리에서 도커 이미지에 액세스” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 개인 리포지토리의 자격 증명

필수: `false`. 이 파라미터는 프라이빗 리포지토리에 저장된 Docker 이미지에 액세스하는 데 필요합니다.

유형: `string`의 `array`

유효한 패턴: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

커넥터가 코어에서 실행 중인 Docker 컨테이너에 대한 상태 정보를 기록하는 빈도(초)입니다. 기본값은 300초(5분)입니다.

AWS IoT 콘솔의 디스플레이 이름: 로깅 빈도

필수: `false`

유형: `string`

유효한 패턴: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

마지막 배포의 잘못된 정리로 인해 Docker 배포가 실패할 경우, 이를 강제로 수행할지 여부를 나타냅니다. 기본 값은 `False`입니다.

AWS IoT 콘솔의 표시 이름: 강제 배포

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

`DockerPullBeforeUp`

동작을 `docker-compose pull` 실행하기 전에 `docker-compose up` 배포자를 pull-down-up 실행해야 하는지 여부를 나타냅니다. 기본 값은 `True`입니다.

AWS IoT 콘솔의 디스플레이 이름: Docker Pull Before Up

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

`StopContainersOnNewDeployment`

GGC가 중지될 때 커넥터가 Docker Deployer 관리 Docker 컨테이너를 중지해야 하는지 여부를 나타냅니다(새 그룹이 배포되거나 커널이 종료되면 GGC가 중지됨). 기본 값은 `True`입니다.

AWS IoT 콘솔에 이름 표시: 새 배포 시 Docker 중지

> **Note**
>
> 이 파라미터를 기본 `True`값으로 유지하는 것이 좋습니다. `False` 파라미터를 사용하면 AWS IoT Greengrass 코어를 종료하거나 새 배포를 시작한 후에도 Docker 컨테이너가 계속 실행됩니다. 이 파라미터를 `False`로 설정하는 경우, `docker-compose` 서비스 이름이 변경되거나 추가되는 경우, 필요에 따라 Docker 컨테이너를 유지 관리해야 합니다. 자세한 내용은 `docker-compose` 작성 파일 설명서를 참조하십시오.

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

`DockerOfflineMode`

오프라인으로 AWS IoT Greengrass을(를) 시작할 때 기존 Docker Compose 파일을 사용할지 여부를 나타냅니다. 기본 값은 `False`입니다.

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

Version 6

`DockerComposeFileS3Bucket`

Docker Compose 파일을 포함하는 S3 버킷의 이름입니다. 버킷을 생성할 때는 Amazon Simple Storage Service 사용 설명서에 설명된 버킷 이름 규칙을 따르십시오.

AWS IoT 콘솔에 이름 표시: S3의 Docker Compose 파일

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3의 Docker Compose 파일의 객체 키입니다. 이름 지정 지침에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 키 및 메타데이터를 참조하세요.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileS3Version`

Amazon S3의 Docker Compose 파일의 객체 버전입니다. 객체 키 이름 지정 지침을 비롯한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버전 관리 사용을 참조하십시오.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `false`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileDestinationPath`

Docker Compose 파일의 복사본을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로입니다. 이 디렉터리는 기존 디렉터리여야 합니다. `DockerUserId`에 지정된 사용자는 이 디렉터리에서 파일을 작성할 수 있는 권한을 가져야 합니다. 자세한 설명은 the section called “코어에서 Docker 사용자 설정” 섹션을 참조하세요.

> **Important**
>
> 이 디렉터리는 Docker Compose 파일과 프라이빗 리포지토리에서 Docker 이미지에 대한 자격 증명을 저장합니다. 이 디렉터리의 보안을 유지하는 것은 사용자의 책임입니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 로컬 Compose 파일의 디렉터리 경로

필수: `true`

유형: `string`

유효한 패턴 `\/.*\/?`

예제: `/home/username/myCompose`

`DockerUserId`

커넥터가 실행되는 Linux 사용자의 UID입니다. 이 사용자는 코어 디바이스의 `docker` Linux 그룹에 속해야 하며 `DockerComposeFileDestinationPath` 디렉터리에 대한 쓰기 권한이 있어야 합니다. 자세한 설명은 코어에서 Docker 사용자 설정 섹션을 참조하세요.

> **Note**
>
> 반드시 필요한 경우, 외에는 루트로 실행하는 것은 피하는 것이 좋습니다. 루트 사용자를 지정하는 경우, Lambda 함수가 AWS IoT Greengrass 코어에서 루트로 실행되도록 허용해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

AWS IoT 콘솔의 디스플레이 이름: Docker 사용자 ID

필수: `false`

유형: `string`

유효한 패턴: `^[0-9]{1,5}$`

`AWSSecretsArnList`

프라이빗 리포지토리의 Docker 이미지에 액세스하는 데 사용되는 로그인 정보가 포함된 AWS Secrets Manager의 비밀의 Amazon 리소스 이름(ARN)입니다. 자세한 설명은 the section called “프라이빗 리포지토리에서 도커 이미지에 액세스” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 개인 리포지토리의 자격 증명

필수: `false`. 이 파라미터는 프라이빗 리포지토리에 저장된 Docker 이미지에 액세스하는 데 필요합니다.

유형: `string`의 `array`

유효한 패턴: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

커넥터가 코어에서 실행 중인 Docker 컨테이너에 대한 상태 정보를 기록하는 빈도(초)입니다. 기본값은 300초(5분)입니다.

AWS IoT 콘솔의 디스플레이 이름: 로깅 빈도

필수: `false`

유형: `string`

유효한 패턴: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

마지막 배포의 잘못된 정리로 인해 Docker 배포가 실패할 경우, 이를 강제로 수행할지 여부를 나타냅니다. 기본 값은 `False`입니다.

AWS IoT 콘솔의 표시 이름: 강제 배포

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

`DockerPullBeforeUp`

동작을 `docker-compose pull` 실행하기 전에 `docker-compose up` 배포자를 pull-down-up 실행해야 하는지 여부를 나타냅니다. 기본 값은 `True`입니다.

AWS IoT 콘솔의 디스플레이 이름: Docker Pull Before Up

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

`StopContainersOnNewDeployment`

GGC가 중지될 때(새 그룹 배포가 수행되거나 커넬이 종료될 때) 커넥터가 Docker Deployer 관리 Docker 컨테이너를 중지해야 하는지 여부를 나타냅니다. 기본 값은 `True`입니다.

AWS IoT 콘솔에 이름 표시: 새 배포 시 Docker 중지

> **Note**
>
> 이 파라미터를 기본 `True`값으로 유지하는 것이 좋습니다. `False` 파라미터를 사용하면 AWS IoT Greengrass 코어를 종료하거나 새 배포를 시작한 후에도 Docker 컨테이너가 계속 실행됩니다. 이 파라미터를 `False`로 설정하는 경우, `docker-compose` 서비스 이름이 변경되거나 추가되는 경우, 필요에 따라 Docker 컨테이너를 유지 관리해야 합니다.

자세한 내용은 `docker-compose` 작성 파일 설명서를 참조하십시오.

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

Version 5

`DockerComposeFileS3Bucket`

Docker Compose 파일을 포함하는 S3 버킷의 이름입니다. 버킷을 생성할 때는 Amazon Simple Storage Service 사용 설명서에 설명된 버킷 이름 규칙을 따르십시오.

AWS IoT 콘솔에 이름 표시: S3의 Docker Compose 파일

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3의 Docker Compose 파일의 객체 키입니다. 이름 지정 지침에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 키 및 메타데이터를 참조하세요.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileS3Version`

Amazon S3의 Docker Compose 파일의 객체 버전입니다. 객체 키 이름 지정 지침을 비롯한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버전 관리 사용을 참조하십시오.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `false`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileDestinationPath`

Docker Compose 파일의 복사본을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로입니다. 이 디렉터리는 기존 디렉터리여야 합니다. `DockerUserId`에 지정된 사용자는 이 디렉터리에서 파일을 작성할 수 있는 권한을 가져야 합니다. 자세한 설명은 the section called “코어에서 Docker 사용자 설정” 섹션을 참조하세요.

> **Important**
>
> 이 디렉터리는 Docker Compose 파일과 프라이빗 리포지토리에서 Docker 이미지에 대한 자격 증명을 저장합니다. 이 디렉터리의 보안을 유지하는 것은 사용자의 책임입니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 로컬 Compose 파일의 디렉터리 경로

필수: `true`

유형: `string`

유효한 패턴 `\/.*\/?`

예제: `/home/username/myCompose`

`DockerUserId`

커넥터가 실행되는 Linux 사용자의 UID입니다. 이 사용자는 코어 디바이스의 `docker` Linux 그룹에 속해야 하며 `DockerComposeFileDestinationPath` 디렉터리에 대한 쓰기 권한이 있어야 합니다. 자세한 설명은 코어에서 Docker 사용자 설정 섹션을 참조하세요.

> **Note**
>
> 반드시 필요한 경우, 외에는 루트로 실행하는 것은 피하는 것이 좋습니다. 루트 사용자를 지정하는 경우, Lambda 함수가 AWS IoT Greengrass 코어에서 루트로 실행되도록 허용해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

AWS IoT 콘솔의 디스플레이 이름: Docker 사용자 ID

필수: `false`

유형: `string`

유효한 패턴: `^[0-9]{1,5}$`

`AWSSecretsArnList`

프라이빗 리포지토리의 Docker 이미지에 액세스하는 데 사용되는 로그인 정보가 포함된 AWS Secrets Manager의 비밀의 Amazon 리소스 이름(ARN)입니다. 자세한 설명은 the section called “프라이빗 리포지토리에서 도커 이미지에 액세스” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 개인 리포지토리의 자격 증명

필수: `false`. 이 파라미터는 프라이빗 리포지토리에 저장된 Docker 이미지에 액세스하는 데 필요합니다.

유형: `string`의 `array`

유효한 패턴: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

커넥터가 코어에서 실행 중인 Docker 컨테이너에 대한 상태 정보를 기록하는 빈도(초)입니다. 기본값은 300초(5분)입니다.

AWS IoT 콘솔의 디스플레이 이름: 로깅 빈도

필수: `false`

유형: `string`

유효한 패턴: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

마지막 배포의 잘못된 정리로 인해 Docker 배포가 실패할 경우, 이를 강제로 수행할지 여부를 나타냅니다. 기본 값은 `False`입니다.

AWS IoT 콘솔의 표시 이름: 강제 배포

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

`DockerPullBeforeUp`

동작을 `docker-compose pull` 실행하기 전에 `docker-compose up` 배포자를 pull-down-up 실행해야 하는지 여부를 나타냅니다. 기본 값은 `True`입니다.

AWS IoT 콘솔의 디스플레이 이름: Docker Pull Before Up

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

Docker Compose 파일을 포함하는 S3 버킷의 이름입니다. 버킷을 생성할 때는 Amazon Simple Storage Service 사용 설명서에 설명된 버킷 이름 규칙을 따르십시오.

AWS IoT 콘솔에 이름 표시: S3의 Docker Compose 파일

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3의 Docker Compose 파일의 객체 키입니다. 이름 지정 지침에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 키 및 메타데이터를 참조하세요.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileS3Version`

Amazon S3의 Docker Compose 파일의 객체 버전입니다. 객체 키 이름 지정 지침을 비롯한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버전 관리 사용을 참조하십시오.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `false`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileDestinationPath`

Docker Compose 파일의 복사본을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로입니다. 이 디렉터리는 기존 디렉터리여야 합니다. `DockerUserId`에 지정된 사용자는 이 디렉터리에서 파일을 작성할 수 있는 권한을 가져야 합니다. 자세한 설명은 the section called “코어에서 Docker 사용자 설정” 섹션을 참조하세요.

> **Important**
>
> 이 디렉터리는 Docker Compose 파일과 프라이빗 리포지토리에서 Docker 이미지에 대한 자격 증명을 저장합니다. 이 디렉터리의 보안을 유지하는 것은 사용자의 책임입니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 로컬 Compose 파일의 디렉터리 경로

필수: `true`

유형: `string`

유효한 패턴 `\/.*\/?`

예제: `/home/username/myCompose`

`DockerUserId`

커넥터가 실행되는 Linux 사용자의 UID입니다. 이 사용자는 코어 디바이스의 `docker` Linux 그룹에 속해야 하며 `DockerComposeFileDestinationPath` 디렉터리에 대한 쓰기 권한이 있어야 합니다. 자세한 설명은 코어에서 Docker 사용자 설정 섹션을 참조하세요.

> **Note**
>
> 반드시 필요한 경우, 외에는 루트로 실행하는 것은 피하는 것이 좋습니다. 루트 사용자를 지정하는 경우, Lambda 함수가 AWS IoT Greengrass 코어에서 루트로 실행되도록 허용해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

AWS IoT 콘솔의 디스플레이 이름: Docker 사용자 ID

필수: `false`

유형: `string`

유효한 패턴: `^[0-9]{1,5}$`

`AWSSecretsArnList`

프라이빗 리포지토리의 Docker 이미지에 액세스하는 데 사용되는 로그인 정보가 포함된 AWS Secrets Manager의 비밀의 Amazon 리소스 이름(ARN)입니다. 자세한 설명은 the section called “프라이빗 리포지토리에서 도커 이미지에 액세스” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 개인 리포지토리의 자격 증명

필수: `false`. 이 파라미터는 프라이빗 리포지토리에 저장된 Docker 이미지에 액세스하는 데 필요합니다.

유형: `string`의 `array`

유효한 패턴: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

커넥터가 코어에서 실행 중인 Docker 컨테이너에 대한 상태 정보를 기록하는 빈도(초)입니다. 기본값은 300초(5분)입니다.

AWS IoT 콘솔의 디스플레이 이름: 로깅 빈도

필수: `false`

유형: `string`

유효한 패턴: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

마지막 배포의 잘못된 정리로 인해 Docker 배포가 실패할 경우, 이를 강제로 수행할지 여부를 나타냅니다. 기본 값은 `False`입니다.

AWS IoT 콘솔의 표시 이름: 강제 배포

필수: `false`

유형: `string`

유효한 패턴: `^(true|false)$`

Version 1

`DockerComposeFileS3Bucket`

Docker Compose 파일을 포함하는 S3 버킷의 이름입니다. 버킷을 생성할 때는 Amazon Simple Storage Service 사용 설명서에 설명된 버킷 이름 규칙을 따르십시오.

AWS IoT 콘솔에 이름 표시: S3의 Docker Compose 파일

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3의 Docker Compose 파일의 객체 키입니다. 이름 지정 지침에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 키 및 메타데이터를 참조하세요.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `true`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileS3Version`

Amazon S3의 Docker Compose 파일의 객체 버전입니다. 객체 키 이름 지정 지침을 비롯한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버전 관리 사용을 참조하십시오.

> **Note**
>
> 콘솔에서 Docker Compose file in S3(S3의 Docker Compose 파일) 속성은 `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` 및 `DockerComposeFileS3Version` 파라미터를 결합합니다.

필수: `false`

유형: `string`

유효한 패턴 `.+`

`DockerComposeFileDestinationPath`

Docker Compose 파일의 복사본을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로입니다. 이 디렉터리는 기존 디렉터리여야 합니다. `DockerUserId`에 지정된 사용자는 이 디렉터리에서 파

일을 작성할 수 있는 권한을 가져야 합니다. 자세한 설명은 the section called “코어에서 Docker 사용자 설정” 섹션을 참조하세요.

> **Important**
>
> 이 디렉터리는 Docker Compose 파일과 프라이빗 리포지토리에서 Docker 이미지에 대한 자격 증명을 저장합니다. 이 디렉터리의 보안을 유지하는 것은 사용자의 책임입니다. 자세한 설명은 the section called “보안 참고 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 로컬 Compose 파일의 디렉터리 경로

필수: `true`

유형: `string`

유효한 패턴 `\/.*\/?`

예제: `/home/username/myCompose`

`DockerUserId`

커넥터가 실행되는 Linux 사용자의 UID입니다. 이 사용자는 코어 디바이스의 `docker` Linux 그룹에 속해야 하며 `DockerComposeFileDestinationPath` 디렉터리에 대한 쓰기 권한이 있어야 합니다. 자세한 설명은 코어에서 Docker 사용자 설정 섹션을 참조하세요.

> **Note**
>
> 반드시 필요한 경우, 외에는 루트로 실행하는 것은 피하는 것이 좋습니다. 루트 사용자를 지정하는 경우, Lambda 함수가 AWS IoT Greengrass 코어에서 루트로 실행되도록 허용해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

AWS IoT 콘솔의 디스플레이 이름: Docker 사용자 ID

필수: `false`

유형: `string`

유효한 패턴: `^[0-9]{1,5}$`

`AWSSecretsArnList`

프라이빗 리포지토리의 Docker 이미지에 액세스하는 데 사용되는 로그인 정보가 포함된 AWS Secrets Manager의 비밀의 Amazon 리소스 이름(ARN)입니다. 자세한 설명은 the section called "프라이빗 리포지토리에서 도커 이미지에 액세스" 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 개인 리포지토리의 자격 증명

필수: `false`. 이 파라미터는 프라이빗 리포지토리에 저장된 Docker 이미지에 액세스하는 데 필요합니다.

유형: `string`의 `array`

유효한 패턴: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

커넥터가 코어에서 실행 중인 Docker 컨테이너에 대한 상태 정보를 기록하는 빈도(초)입니다. 기본값은 300초(5분)입니다.

AWS IoT 콘솔의 디스플레이 이름: 로깅 빈도

필수: `false`

유형: `string`

유효한 패턴: `^[1-9]{1}[0-9]{0,3}$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Greengrass Docker 애플리케이션 배포 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을(를) 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyDockerAppplicationDeploymentConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
DockerApplicationDeployment/versions/5",
            "Parameters": {
```

```
                "DockerComposeFileS3Bucket": "myS3Bucket",
                "DockerComposeFileS3Key": "production-docker-compose.yml",
                "DockerComposeFileS3Version": "123",
                "DockerComposeFileDestinationPath": "/home/username/myCompose",
                "DockerUserId": "1000",
                "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\",\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
                "DockerContainerStatusLogFrequency": "30",
                "ForceDeploy": "True",
                "DockerPullBeforeUp": "True"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

입력 데이터

이 커넥터는 입력 데이터를 요구하거나 허용하지 않습니다.

출력 데이터

이 커넥터는 `docker-compose up` 명령의 상태를 출력 데이터로 게시합니다.

구독의 주제 필터

```
dockerapplicationdeploymentconnector/message/status
```

출력 예: 성공

```
{
  "status":"success",
  "GreengrassDockerApplicationDeploymentStatus":"Successfully triggered docker-
compose up",
  "S3Bucket":"myS3Bucket",
  "ComposeFileName":"production-docker-compose.yml",
  "ComposeFileVersion":"123"
```

```
}
```

출력 예: 실패

```
{
  "status":"fail",
  "error_message":"description of error",
  "error":"InvalidParameter"
}
```

오류 유형은 `InvalidParameter` 또는 `InternalError`일 수 있습니다.

AWS IoT Greengrass 코어에서 Docker 사용자 설정

Greengrass Docker 애플리케이션 배포 커넥터는 `DockerUserId` 파라미터에 지정한 사용자로 실행됩니다. 값을 지정하지 않으면 커넥터가 기본 Greengrass 액세스 ID인 `ggc_user`(으)로 실행됩니다.

커넥터가 Docker 대몬(daemon)과 상호 작용할 수 있게 하려면 Docker 사용자가 코어의 `docker` Linux 그룹에 속해야 합니다. Docker 사용자는 `DockerComposeFileDestinationPath` 디렉터리에 대한 쓰기 권한이 있어야 합니다. 여기에 커넥터가 로컬 `docker-compose.yml` 파일 및 Docker 자격 증명을 저장합니다.

> **Note**
>
> - 기본값 `ggc_user`을 사용하는 대신 Linux 사용자를 생성하는 것이 좋습니다. 그렇지 않으면 Greengrass 그룹의 모든 Lambda 함수가 Compose 파일 및 Docker 자격 증명에 액세스 할 수 있습니다.
> - 반드시 필요한 경우, 외에는 루트로 실행하는 것은 피하는 것이 좋습니다. 루트 사용자를 지정하는 경우, Lambda 함수가 AWS IoT Greengrass 코어에서 루트로 실행되도록 허용해야 합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

1. 사용자를 생성합니다. `useradd` 명령을 실행하고 UID를 할당하는 `-u` 옵션을 포함시킬 수 있습니다. 예:

   ```
   sudo useradd -u 1234 user-name
   ```

2. 코어의 docker 그룹에 사용자를 추가합니다. 예:

   ```
   sudo usermod -aG docker user-name
   ```

   docker 그룹을 생성하는 방법을 포함하여 자세한 내용은 Docker 설명서의 Docker를 루트가 아닌 사용자로 관리를 참조하십시오.

3. 사용자에게 DockerComposeFileDestinationPath 파라미터에 지정된 디렉터리에 쓰기를 할 수 있는 권한을 부여합니다. 예:

   a. 사용자를 디렉터리의 소유자로 설정합니다. 이 예제에서는 1단계의 UID를 사용합니다.

      ```
      chown 1234 docker-compose-file-destination-path
      ```

   b. 소유자에게 읽기 및 쓰기 권한을 제공하려면

      ```
      chmod 700 docker-compose-file-destination-path
      ```

      자세한 내용은 Linux Foundation 설명서의 Linux에서 파일 및 폴더 권한을 관리하는 방법을 참조하십시오.

   c. 사용자를 생성할 때 UID를 할당하지 않았거나 기존 사용자를 사용한 경우에는 id 명령을 실행하여 UID를 조회합니다.

      ```
      id -u user-name
      ```

      UID를 사용하여 커넥터에 대한 DockerUserId 파라미터를 구성합니다.

사용 정보

Greengrass Docker 애플리케이션 배포 커넥터를 사용할 때는 다음과 같은 구현별 사용 정보를 알고 있어야 합니다.

- 프로젝트 이름의 고정 접두사. 커넥터는 시작하는 Docker 컨테이너의 이름 앞에 접두어 greengrassdockerapplicationdeployment을 추가합니다. 커넥터는 이 접두어를 커넥터에서 실행되는 docker-compose 명령에서 프로젝트 이름으로 사용합니다.
- 로깅 동작. 커넥터는 상태 정보와 문제 해결 정보를 로그 파일에 기록합니다. 로그를 AWS IoT Greengrass Logs로 전송하고 로컬에 CloudWatch 로그를 기록하도록 구성할 수 있습니다. 자세한 설명은 the section called “로그” 섹션을 참조하세요. 커넥터에 대한 로컬 로그의 경로입니다.

```
/greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
```

로컬 로그에 액세스하려면 루트 권한이 있어야 합니다.

- Docker 이미지 업데이트. Docker는 코어 디바이스에 이미지를 캐싱합니다. Docker 이미지를 업데이트하고 변경 사항을 코어 디바이스에 전파하고 싶은 경우에는 Compose 파일에서 이미지의 태그를 변경해야 합니다. 변경 사항은 Greengrass 그룹이 배포된 후에 적용됩니다.
- 정리 작업에 대한 10분의 제한 시간. 다시 시작하는 동안 Greengrass 대몬(daemon)이 중지되면 `docker-compose down` 명령이 시작됩니다. 모든 Docker 컨테이너는 `docker-compose down`가 시작된 후 정리 작업이 수행되기까지 최대 10분이 소요됩니다. 정리가 10분 안에 완료되지 않으면 나머지 컨테이너를 수동으로 정리해야 합니다. 자세한 내용은 Docker CLI 설명서의 docker rm을 참조하십시오.
- Docker 명령 실행. 문제를 해결하기 위해 코어 디바이스의 터미널 창에서 Docker 명령을 실행할 수 있습니다. 예를 들어 다음 명령을 실행하여 커넥터에서 시작된 Docker 컨테이너를 확인합니다.

```
docker ps --filter name="greengrassdockerapplicationdeployment"
```

- 예약 리소스 ID입니다. 커넥터는 Greengrass 그룹에서 생성하는 Greengrass 리소스에 대한 `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_index` ID를 사용합니다. 리소스 ID는 그룹에서 고유해야 하므로 이 예약 리소스 ID와 충돌할 수 있는 리소스 ID를 할당해서는 안 됩니다.
- 오프라인 모드. `DockerOfflineMode` 구성 파라미터를 `True`(으)로 설정하면 Docker 커넥터가 오프라인 모드에서 작동할 수 있습니다. 이는 코어 디바이스가 오프라인 상태일 때 Greengrass 그룹 배포가 다시 시작되고 커넥터가 Docker Compose 파일을 검색하기 위해 Amazon S3 또는 Amazon ECR에 대한 연결을 설정할 수 없는 경우, 발생할 수 있습니다.

  오프라인 모드가 활성화되면 커넥터는 Compose 파일을 다운로드하고 정상적으로 다시 시작할 때와 마찬가지로 `docker login` 명령을 실행하려고 시도합니다. 이러한 시도가 실패하면 커넥터는 `DockerComposeFileDestinationPath` 파라미터를 사용하여 지정된 폴더에서 로컬로 저장된 Compose 파일을 찾습니다. 로컬 Compose 파일이 있는 경우, 커넥터는 일반적인 `docker-compose` 명령 시퀀스에 따라 로컬 이미지에서 가져옵니다. Compose 파일이나 로컬 이미지가 없는 경우, 커넥터에 장애가 발생합니다. `ForceDeploy` 및 `StopContainersOnNewDeployment` 파라미터의 동작은 오프라인 모드에서도 동일하게 유지됩니다.

Docker 컨테이너와 통신

AWS IoT Greengrass은 다음과 같은 Greengrass 구성 요소와 Docker 컨테이너 간의 통신 채널을 지원합니다.

- Greengrass Lambda 함수는 REST API를 사용하여 Docker 컨테이너의 프로세스와 통신 할 수 있습니다. 포트를 여는 Docker 컨테이너에서 서버를 설정할 수 있습니다. Lambda 함수는 이 포트의 컨테이너와 통신할 수 있습니다.
- Docker 컨테이너의 프로세스는 로컬 Greengrass 메시지 브로커를 통해 MQTT 메시지를 교환할 수 있습니다. Docker 컨테이너를 Greengrass 그룹에서 클라이언트 디바이스로 설정한 다음, 컨테이너가 그룹의 Greengrass Lambda 함수, 클라이언트 디바이스 및 기타 커넥터나 AWS IoT 및 로컬 섀도우 서비스와 통신할 수 있도록 허용하는 구독을 생성할 수 있습니다. 자세한 설명은 the section called “Docker 컨테이너와의 MQTT 통신 구성” 섹션을 참조하세요.
- Greengrass Lambda 함수는 공유 파일을 업데이트하여 Docker 컨테이너에 정보를 전달할 수 있습니다. Compose 파일을 사용하여 Docker 컨테이너의 공유 파일 경로를 바인드 마운트 할 수 있습니다.

Docker 컨테이너와의 MQTT 통신 구성

Docker 컨테이너를 클라이언트 디바이스로 구성하여 Greengrass 그룹에 추가할 수 있습니다. 그런 다음 Docker 컨테이너와 Greengrass 구성 요소 또는 AWS IoT 간의 MQTT 통신을 허용하는 구독을 생성할 수 있습니다. 다음 절차에서는 Docker 컨테이너 장치가 로컬 섀도우 서비스에서 섀도우 업데이트 메시지를 수신 할 수 있도록 허용하는 구독을 생성합니다. 이 패턴을 따라 다른 구독을 생성할 수 있습니다.

> **Note**
>
> 이 절차에서는 이미 Greengrass 그룹과 Greengrass 코어(v1.10 이상)를 생성했다고 가정합니다. Greengrass 그룹 또는 코어를 생성하는 방법에 대한 자세한 내용은 시작하기: AWS IoT Greengrass을(를) 참조하세요.

Docker 컨테이너를 클라이언트 디바이스로 구성하여 Greengrass 그룹에 추가하려면

1. 코어 디바이스에 폴더를 생성하여 Greengrass 디바이스를 인증하는 데 사용되는 인증서와 키를 저장합니다.

시작하고 싶은 Docker 컨테이너에 파일 경로가 마운트 되어야 합니다. 다음 코드 조각은 Compose 파일에 파일 경로를 마운트하는 방법을 보여줍니다. 이 예제에서 는 이 단계에서 만든 폴더를 *path-to-device-certs*나타냅니다.

```
version: '3.3'
services:
  myService:
    image: user-name/repo:image-tag
    volumes:
      - /path-to-device-certs/:/path-accessible-in-container
```

2. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
3. 대상 그룹을 선택합니다.
4. 그룹 구성 페이지에서 클라이언트 디바이스를 선택한 다음 연결을 선택합니다.
5. 클라이언트 디바이스를 이 그룹에 연결 모달에서 새 AWS IoT 사물 만들기를 선택합니다.

   사물 생성 페이지가 새 탭에서 열립니다.
6. 사물 생성 페이지에서 단일 사물 생성을 선택한 후 다음을 선택합니다.
7. 사물 속성 지정 페이지에서 디바이스 이름을 입력하고 다음을 선택합니다.
8. 디바이스 인증서 구성 페이지에서 다음을 선택합니다.
9. 인증서에 정책 첨부 페이지에서 다음 중 하나를 수행합니다.
   - 클라이언트 디바이스에 필요한 권한을 부여하는 기존 정책을 선택한 다음 사물 생성을 선택합니다.

     디바이스가 AWS 클라우드 및 코어에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.
   - 클라이언트 디바이스 권한을 부여하는 새 정책을 만들어 첨부하십시오. 다음을 따릅니다.

     a. 정책 생성을 선택합니다.

        정책 생성 페이지가 새 탭에서 열립니다.

     b. [Create policy(정책 생성)] 페이지에서 다음을 수행합니다.

        i. 정책 이름에는 **GreengrassV1ClientDevicePolicy**와(과) 같이 정책을 설명하는 이름을 입력합니다.

        ii. 정책 설명 탭의 정책 문서에서 JSON을 선택합니다.

iii. 다음 정책 문서를 입력합니다. 이 정책을 통해 클라이언트 디바이스는 Greengrass 코어를 검색하고 모든 MQTT 주제에 대해 통신할 수 있습니다. 이 정책의 액세스를 제한하는 방법에 대한 자세한 내용은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여을(를) 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 생성(Create)을 선택하여 정책을 생성합니다.

c. 인증서에 정책 첨부 페이지가 열려 있는 브라우저 탭으로 돌아가십시오. 다음을 따릅니다.

i. 정책 목록에서 GreengrassV1ClientDevicePolicy과 같이 만든 정책을 선택합니다.

새 정책이 나타나지 않으면 새로고침 아이콘을 선택합니다.

ii. 사물 생성(Create thing)을 선택합니다.

디바이스가 AWS 클라우드 및 코어에 연결하는 데 사용하는 인증서 및 키를 다운로드할 수 있는 모달이 열립니다.

10. 인증서 및 키 다운로드 모달에서 디바이스의 인증서를 다운로드합니다.

> **Important**
>
> 완료를 선택하기 전에 보안 리소스를 다운로드합니다.

다음을 따릅니다.

a. 디바이스 인증서의 경우, 다운로드를 선택하여 디바이스 인증서를 다운로드합니다.
b. 공개 키 파일의 경우, 다운로드를 선택하여 인증서의 공개 키를 다운로드합니다.
c. 개인 키 파일의 경우, 다운로드를 선택하여 인증서의 개인 키 파일을 다운로드합니다.
d. AWS IoT 개발자 안내서의 서버 인증을 검토하고 적절한 루트 CA 인증서를 선택합니다. Amazon Trust Services(ATS) 엔드포인트와 ATS 루트 CA 인증서를 사용하는 것이 좋습니다. 루트 CA 인증서에서 루트 CA 인증서용 다운로드를 선택합니다.
e. 완료를 선택합니다.

디바이스 인증서 및 키의 파일 이름에 공통적으로 나타나는 인증서 ID를 기록해 둡니다. 나중에 필요합니다.

11. 1단계에서 생성한 폴더에 인증서와 키를 복사합니다.

그런 다음 그룹에 구독을 생성합니다. 이 예제에서는 Docker 컨테이너 디바이스가 로컬 섀도우 서비스에서 MQTT 메시지를 수신할 수 있도록 해주는 구독을 생성합니다.

> **Note**
>
> 섀도우 문서의 최대 크기는 8KB입니다. 자세한 내용은 AWS IoT 개발자 안내서에서 AWS IoT 할당량을 참조하세요.

Docker 컨테이너 디바이스가 로컬 섀도우 서비스에서 MQTT 메시지를 수신 할 수 있게 해주는 구독을 생성하려면

1. 그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.

2. [Select your source and target] 페이지에서 다음과 같이 원본과 대상을 구성합니다.

   a. 소스 선택에서 서비스를 선택한 다음 Local Shadow Service(로컬 섀도우 서비스)를 선택합니다.

   b. 대상 선택에서 디바이스를 선택한 다음 해당 디바이스를 선택합니다.

   c. 다음을 선택합니다.

   d. 데이터를 주제에 대해 필터링 페이지의 주제 필터에서 **$aws/things/*MyDockerDevice*/shadow/update/accepted**을(를) 선택한 다음 다음을 선택합니다. 이전에 만든 디바이스의 *MyDockerDevice*이름으로 바꾸십시오.

   e. 마침을 클릭합니다.

Compose 파일에서 참조하는 Docker 이미지에 다음과 같은 코드 조각을 포함시킵니다. 이것은 Greengrass 디바이스 코드입니다. 또한 컨테이너 내부에서 Greengrass 디바이스를 시작하는 코드를 Docker 컨테이너에 추가합니다. 이미지 또는 별도의 스레드에서 별도의 프로세스로 실행할 수 있습니다.

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker
 container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"
```

```
# Discover Greengrass cores.
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10)  # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None

try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
    coreList = discoveryInfo.getAllCores()

    # Use first discovery result.
    groupId, ca = caList[0]
    coreInfo = coreList[0]

    # Save the group CA to a local file.
    groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
    if not os.path.exists(GROUP_CA_PATH):
        os.makedirs(GROUP_CA_PATH)
    groupCAFile = open(groupCA, "w")
    groupCAFile.write(ca)
    groupCAFile.close()
    discovered = True
except DiscoveryInvalidRequestException as e:
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
```

```
myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)


# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
    if connected:
        break

if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
    time.sleep(1)
```

보안 참고 사항

Greengrass Docker 애플리케이션 배포 커넥터를 사용할 때는 다음과 같은 보안 고려 사항에 유의하십시오.

- Docker Compose 파일의 로컬 저장소

커넥터는 Compose 파일의 복사본을 `DockerComposeFileDestinationPath` 파라미터에 지정된 디렉터리에 저장합니다.

이 디렉터리의 보안을 유지하는 것은 사용자의 책임입니다. 파일 시스템 권한을 사용하여 디렉터리에 대한 액세스를 제한해야 합니다.

Docker 자격 증명의 로컬 스토리지

Docker 이미지가 프라이빗 리포지토리에 저장되어 있는 경우, 커넥터는 `DockerComposeFileDestinationPath` 파라미터에 지정된 디렉터리에 Docker 자격 증명을 저장합니다.

이러한 자격 증명을 보호하는 것은 사용자의 책임입니다. 예를 들어 Docker Engine을 설치할 때 코어 디바이스에서 credential-helper를 사용해야 합니다.

신뢰할 수 있는 소스에서 Docker Engine 설치

신뢰할 수 있는 소스에서 Docker Engine을 설치하는 것은 사용자의 책임입니다. 이 커넥터는 코어 디바이스에서 Docker 대몬(daemon)을 사용하여 Docker 자산에 액세스하고 Docker 컨테이너를 관리합니다.

Greengrass 그룹 역할 권한의 범위

Greengrass 그룹 역할에 추가하는 권한은 Greengrass 그룹의 모든 Lambda 함수 및 커넥터에서 수임할 수 있습니다. 이 커넥터는 S3 버킷에 저장된 Docker Compose 파일에 액세스해야 합니다. 또한 Docker 이미지가 Amazon ECR의 프라이빗 리포지토리에 저장되어있는 경우, Amazon ECR 인증 토큰에 액세스해야 합니다 .

라이선스

Greengrass Docker 애플리케이션 배포 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스

- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
7	오프라인으로 AWS IoT Greengrass을(를) 시작할 때 기존 Docker Compose 파일을 사용하도록 `DockerOfflineMode` 이(가) 추가되었습니다. `docker login` 명령에 대한 재시도를 구현했습니다. 32비트 UID를 지원합니다.
6	새로 배포하거나 GGC가 중지될 때 컨테이너 정리를 재정의하도록 `StopContainersOnNewDeployment` (이)가 추가되었습니다. 더 안전한 종료 및 시작 메커니즘. YAML 유효성 검사 버그 수정.
5	`docker-compose down` 실행 전에 이미지를 가져옵니다.
4	Docker 이미지를 업데이트하는 pull-before-up 동작이 추가되었습니다.
3	환경 변수를 찾는 문제가 해결되었습니다.
2	`ForceDeploy` 파라미터를 추가했습니다.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드”을 참조하십시오.

다음 사항도 참조하십시오.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

IoT Analytics 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

IoT Analytics 커넥터는 로컬 디바이스 데이터를 AWS IoT Analytics에 전송합니다. 이 커넥터를 중앙 허브로 사용하여 Greengrass 코어 디바이스의 센서 및 연결된 클라이언트 디바이스에서 데이터를 수집할 수 있습니다. 커넥터는 현재 AWS 계정 계정 및 리전의 AWS IoT Analytics 채널에 데이터를 전송합니다. 기본 대상 채널 및 동적으로 지정된 채널에 데이터를 전송할 수 있습니다.

> **Note**
>
> AWS IoT Analytics는 IoT 데이터를 수집, 저장, 처리 및 쿼리하는 데 사용할 수 있는 완전 관리형 서비스입니다. AWS IoT Analytics에서 데이터를 추가로 분석하고 처리할 수 있습니다. 예를 들어, 이 기능을 사용하여 머신 상태를 모니터링하도록 ML 모델을 훈련하거나 새 모델링 전략을 테스트할 수 있습니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 AWS IoT Analytics(이)란 무엇입니까? 섹션을 참조하세요.

커넥터는 입력 MQTT 주제에 대한 형식이 지정된 데이터 및 형식이 지정되지 않은 데이터를 수락합니다. 이 커넥터는 대상 채널이 인라인으로 지정되는 두 개의 미리 지정된 주제를 지원합니다. 또한 구독에서 구성되는 고객 정의 주제에 대한 메시지도 수신할 수 있습니다. 이 커넥터를 사용하여 고정 주제에 게시되는 클라이언트 디바이스의 메시지를 라우팅하거나 리소스 제한 디바이스의 비정형 또는 스택 종속 데이터를 처리할 수 있습니다.

이 커넥터는 `BatchPutMessage` API를 사용하여 데이터를(JSON 또는 base64 인코딩 문자열로) 대상 채널에 전송합니다. 이 커넥터는 원시 데이터를 API 요구 사항에 적합한 형식으로 처리할 수 있습니다. 이 커넥터는 채널별 대기열에서 입력 메시지를 버퍼링하고 배치를 비동기적으로 처리합니다. 이 커넥터는 대기열 작성 및 배치 작성 동작을 제어하고 메모리 사용을 제한할 수 있는 파라미터를 제공합니다. 예를 들어 최대 대기열 크기, 배치 간격, 메모리 크기, 활성 채널 수를 구성할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
4	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/4`
3	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/3`
2	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/2`
1	`arn:aws:greengrass:region::/connectors/IoTAnalytics/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3 - 4

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```

이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 이 커넥터는 AWS IoT Greengrass와 AWS IoT Analytics에서 모두 지원되는 Amazon Web Services 리전에서만 사용할 수 있습니다.
- 모든 관련 AWS IoT Analytics 엔터티 및 워크플로우가 생성되고 구성됩니다. 엔터티에는 채널, 파이프라인, 데이터스토어 및 데이터 세트가 포함됩니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 AWS CLI 또는 콘솔 절차를 참조하십시오.

> **Note**
>
> 대상 AWS IoT Analytics 채널은 이 커넥터와 동일한 계정을 사용해야 하며 동일한 AWS 리전에 있어야 합니다.

- 다음 예제와 같이, 대상 채널에 대한 `iotanalytics:BatchPutMessage` 작업을 허용하는 Greengrass 그룹 역할에 IAM 정책을 추가해야 합니다. 채널은 현재 AWS 계정 및 리전에 있어야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
    ]
}
```

그룹 역할 요구 사항의 경우 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- 이 커넥터는 AWS IoT Greengrass와 AWS IoT Analytics에서 모두 지원되는 Amazon Web Services 리전에서만 사용할 수 있습니다.
- 모든 관련 AWS IoT Analytics 엔터티 및 워크플로우가 생성되고 구성됩니다. 엔터티에는 채널, 파이프라인, 데이터스토어 및 데이터 세트가 포함됩니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 AWS CLI 또는 콘솔 절차를 참조하십시오.

  > **Note**
  >
  > 대상 AWS IoT Analytics 채널은 이 커넥터와 동일한 계정을 사용해야 하며 동일한 AWS 리전에 있어야 합니다.

- 다음 예제와 같이, 대상 채널에 대한 `iotanalytics:BatchPutMessage` 작업을 허용하는 Greengrass 그룹 역할에 IAM 정책을 추가해야 합니다. 채널은 현재 AWS 계정 계정 및 리전에 있어야 합니다.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Sid": "Stmt1528133056761",
              "Action": [
                  "iotanalytics:BatchPutMessage"
              ],
              "Effect": "Allow",
              "Resource": [
                  "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                  "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
              ]
          }
      ]
  }
  ```

  그룹 역할 요구 사항의 경우 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

파라미터

`MemorySize`

이 커넥터에 할당할 메모리 양(KB)입니다.

AWS IoT 콘솔의 표시 이름: 메모리 크기

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`PublishRegion`

AWS IoT Analytics 채널이 생성되는 AWS 리전입니다. 커넥터와 동일한 리전을 사용합니다.

> **Note**
> 이 리전은 그룹 역할에서 지정되는 채널의 리전과도 일치해야 합니다.

AWS IoT 콘솔의 표시 이름: 게시 리전

필수: `false`

형식: `string`

유효한 패턴: `^$|([a-z]{2}-[a-z]+-\\d{1})`

`PublishInterval`

수신된 데이터의 배치를 AWS IoT Analytics에 게시하는 간격(초)입니다.

AWS IoT 콘솔의 표시 이름: 게시 간격

필수: `false`

형식: `string`

기본값: `1`

유효한 패턴: `$|^[0-9]+$`

`IotAnalyticsMaxActiveChannels`

커넥터가 활성으로 감시하는 최대 AWS IoT Analytics 채널 수입니다. 이 값은 0보다 커야 하며 최소한 커넥터가 지정된 시간에 게시할 것으로 예상되는 채널 수와 같아야 합니다.

이 파라미터를 사용하면 커넥터가 지정된 시간에 관리할 수 있는 총 대기열 수를 제한하여 메모리 사용을 제한할 수 있습니다. 대기된 메시지를 모두 전송하면 대기열이 삭제됩니다.

AWS IoT 콘솔의 표시 이름: 활성 채널의 최대 수

필수: `false`

형식: `string`

기본값: `50`

유효한 패턴: `^$|^[1-9][0-9]*$`

`IotAnalyticsQueueDropBehavior`

대기열이 가득 차 있을 때 채널 대기열에서 메시지를 삭제하기 위한 동작입니다.

AWS IoT 콘솔의 표시 이름: 대기열 삭제 동작

필수: `false`

형식: `string`

유효한 값: `DROP_NEWEST` 또는 `DROP_OLDEST`

기본값: `DROP_NEWEST`

유효한 패턴: `^DROP_NEWEST$|^DROP_OLDEST$`

`IotAnalyticsQueueSizePerChannel`

메시지가 제출되거나 삭제되기 전에 메모리에 유지할 최대 메시지 수입니다(채널당). 이 값은 0보다 커야 합니다.

AWS IoT 콘솔의 표시 이름: 채널당 최대 대기열 크기

필수: `false`

형식: `string`

기본값: 2048

유효한 패턴: ^$|^[1-9][0-9]*$

`IotAnalyticsBatchSizePerChannel`

배치 요청에서 AWS IoT Analytics 채널에 전송할 최대 메시지 수입니다. 이 값은 0보다 커야 합니다.

AWS IoT 콘솔의 표시 이름: 채널당 일괄 처리할 최대 메시지 수

필수: `false`

형식: `string`

기본값: 5

유효한 패턴: ^$|^[1-9][0-9]*$

`IotAnalyticsDefaultChannelName`

이 커넥터가 고객 정의 입력 주제로 전송되는 메시지에 사용하는 AWS IoT Analytics 채널의 이름입니다.

AWS IoT 콘솔의 표시 이름: 기본 채널 이름

필수: `false`

형식: `string`

유효한 패턴: ^[a-zA-Z0-9_]$

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 `GreengrassContainer`이며 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
>
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

형식: `string`

유효한 값: `GreengrassContainer` 또는 `NoContainer`

유효한 패턴: `^NoContainer$|^GreengrassContainer$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 IoT Analytics 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTAnalyticsApplication",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTAnalytics/
versions/3",
            "Parameters": {
                "MemorySize": "65535",
                "PublishRegion": "us-west-1",
                "PublishInterval": "2",
                "IotAnalyticsMaxActiveChannels": "25",
                "IotAnalyticsQueueDropBehavior": "DROP_OLDEST",
                "IotAnalyticsQueueSizePerChannel": "1028",
                "IotAnalyticsBatchSizePerChannel": "5",
                "IotAnalyticsDefaultChannelName": "my_channel"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 사전 정의 및 고객 정의 MQTT 주제에 대한 데이터를 수락합니다. 게시자는 Greengrass 디바이스, Lambda 함수 또는 기타 커넥터일 수 있습니다.

미리 정의된 주제

이 커넥터는 게시자가 인라인으로 채널 이름을 지정하도록 허용하는 다음 두 가지 정형 MQTT 주제를 지원합니다.

- `iotanalytics/channels/+/messages/put` 주제에 대한 형식이 지정된 메시지입니다. 이러한 입력 메시지에 있는 IoT 데이터는 JSON 또는 base64 인코딩 문자열로 형식 지정해야 합니다.
- `iotanalytics/channels/+/messages/binary/put` 주제에 대한 형식이 지정되지 않은 메시지입니다. 이 주제에서 수신된 입력 메시지는 바이너리 데이터로 처리되며 모든 데이터 유형을 포함할 수 있습니다.

  미리 정의된 주제에 게시하려면 + 와일드카드를 채널 이름으로 바꿉니다. 예:

  ```
  iotanalytics/channels/my_channel/messages/put
  ```

고객 정의 주제

커넥터는 구독에서 구성한 모든 MQTT 주제에 대한 입력 메시지를 수락하도록 허용하는 # 주제 구문을 지원합니다. 구독에 # 와일드카드만 사용하는 대신 주제 경로를 지정하는 것이 좋습니다. 이러한 메시지는 커넥터에 지정하는 기본 채널로 전송됩니다.

고객 정의 주제에 대한 입력 메시지는 바이너리 데이터로 처리됩니다. 이러한 메시지는 모든 메시지 형식을 사용할 수 있으며 모든 데이터 유형을 포함할 수 있습니다. 고객 정의 주제를 사용하여 고정 주제에 게시하는 디바이스의 메시지를 라우팅할 수 있습니다. 또한 이 주제를 사용하여 데이터를 커넥터에 전송할 형식 지정된 메시지로 처리할 수 없는 클라이언트 디바이스의 입력 데이터를 수락할 수 있습니다.

구독 및 MQTT 주제에 대한 자세한 내용은 the section called “입력 및 출력” 단원을 참조하십시오.

그룹 역할은 모든 대상 채널에 대한 `iotanalytics:BatchPutMessage` 작업을 허용해야 합니다. 자세한 내용은 the section called “요구 사항” 섹션을 참조하세요.

주제 필터: `iotanalytics/channels/+/messages/put`

이 주제를 사용하여 형식 지정된 메시지를 커넥터에 전송하고 대상 채널을 동적으로 지정합니다. 또한 이 주제를 사용하여 응답 출력에서 반환되는 ID를 지정할 수 있습니다. 이 커넥터는 AWS IoT Analytics에 전송하는 발신 `BatchPutMessage` 요청의 각 메시지에 대해 ID가 고유한지 확인합니다. 중복 ID가 있는 메시지는 취소됩니다.

이 주제에 전송되는 입력 데이터는 다음 메시지 형식을 사용해야 합니다.

메시지 속성

`request`

지정된 채널에 전송되는 데이터입니다.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`message`

JSON 또는 base64 인코딩 문자열인 디바이스 또는 센서 데이터입니다.

필수: `true`

형식: `string`

`id`

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다. 지정하면 응답 객체의 `id` 속성이 이 값으로 설정됩니다. 이 속성을 생략하면 커넥터가 ID를 생성합니다.

필수: `false`

형식: `string`

유효한 패턴: `.*`

입력 예

```
{
    "request": {
        "message" : "{\"temp\":23.33}"
    },
    "id" : "req123"
}
```

주제 필터: `iotanalytics/channels/+/messages/binary/put`

이 주제를 사용하여 형식이 지정되지 않은 메시지를 커넥터에 전송하고 대상 채널을 동적으로 지정합니다.

커넥터 데이터는 이 주제에 대해 수신된 입력 메시지를 구문 분석하지 않습니다. 이러한 메시지는 바이너리로 처리합니다. 메시지를 AWS IoT Analytics에 전송하기 전에 커넥터는 `BatchPutMessage` API 요구 사항에 적합하도록 메시지를 인코딩하고 형식 지정합니다.

- 커넥터는 원시 데이터를 base64로 인코딩하고 인코딩된 페이로드를 발신 `BatchPutMessage` 요청에 포함시킵니다.
- 커넥터는 ID를 생성하고 각 입력 메시지에 할당합니다.

> **Note**
> 커넥터의 응답 출력에는 이러한 입력 메시지에 대한 ID 상호 관계가 포함되지 않습니다.

메시지 속성

없음.

주제 필터: #

이 주제를 사용하여 모든 메시지 형식을 기본 채널로 전송합니다. 이 기능은 클라이언트 디바이스가 고정 주제에 게시하는 경우 또는 데이터를 커넥터 지원 메시지 형식으로 처리할 수 없는 클라이언트 디바이스의 기본 채널로 데이터를 전송하려는 경우에 유용합니다.

생성하는 구독에서 주제 구문을 정의하여 이 커넥터를 데이터 소스에 연결합니다. 구독에 # 와일드카드만 사용하는 대신 주제 경로를 지정하는 것이 좋습니다.

커넥터 데이터는 이 입력 주제에 게시되는 메시지를 구문 분석하지 않습니다. 모든 입력 메시지는 바이너리 데이터로 처리됩니다. 메시지를 AWS IoT Analytics에 전송하기 전에 커넥터는 `BatchPutMessage` API 요구 사항에 적합하도록 메시지를 인코딩하고 형식 지정합니다.

- 커넥터는 원시 데이터를 base64로 인코딩하고 인코딩된 페이로드를 발신 `BatchPutMessage` 요청에 포함시킵니다.
- 커넥터는 ID를 생성하고 각 입력 메시지에 할당합니다.

> **Note**
> 커넥터의 응답 출력에는 이러한 입력 메시지에 대한 ID 상호 관계가 포함되지 않습니다.

메시지 속성

없음.

출력 데이터

이 커넥터는 상태 정보를 MQTT 주제에 출력 데이터로 게시합니다. 이 정보에는 AWS IoT Analytics에서 수신하고 전송하는 각 입력 메시지에 대해 AWS IoT Analytics가 반환하는 응답이 포함됩니다.

구독의 주제 필터

`iotanalytics/messages/put/status`

출력 예: 성공

```
{
    "response" : {
        "status" : "success"
    },
    "id" : "req123"
}
```

출력 예: 실패

```
{
    "response" : {
        "status" : "fail",
        "error" : "ResourceNotFoundException",
        "error_message" : "A resource with the specified name could not be found."
    },
    "id" : "req123"
}
```

> **Note**
>
> 커넥터가 재시도 가능한 오류(예: 연결 오류)를 감지하면 다음 배치에서 게시를 재시도합니다. 지수 백오프는 AWS SDK에서 처리됩니다. 재시도할 수 있는 오류가 있는 요청은 `IotAnalyticsQueueDropBehavior` 파라미터에 따라 추가 게시를 위해 다시 채널 대기열에 추가됩니다.

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.

   그룹 역할 요구 사항의 경우 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 커넥터를 추가하고 해당 파라미터를 구성합니다.

   c. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.

4. 그룹을 배포합니다.
5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'iotanalytics/channels/my_channel/messages/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "message" : "{\"temp\":23.33}"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

제한

이 커넥터에는 다음 제한이 적용됩니다.

- AWS IoT Analytics `batch_put_message` 작업에 대해 AWS SDK for Python (Boto3)에서 부과한 모든 제한.
- AWS IoT Analytics BatchPutMessage API에서 부과하는 모든 할당량. 자세한 내용은 AWS 일반 참조의 AWS IoT Analytics 서비스 할당량을 참조하세요.
  - 채널별로 초당 100,000개의 메시지.
  - 배치당 100개의 메시지.
  - 메시지당 128KB.

  이 API는 채널 이름(채널 ARN이 아님)을 사용하므로, 데이터를 교차 리전 또는 교차 계정 채널로 전송하는 작업은 지원되지 않습니다.
- AWS IoT Greengrass 코어에 의해 부과된 모든 할당량. 자세한 내용은 AWS 일반 참조의 AWS IoT Greengrass 코어에 대한 서비스 할당량을 참조하세요.

  다음 할당량도 특정하게 적용될 수 있습니다.
  - 장치에서 전송되는 최대 메시지 크기는 128KB입니다.
  - Greengrass 코어 라우터에서 최대 메시지 대기열 크기는 2.5MB입니다.
  - 주제 문자열의 최대 길이는 256바이트의 UTF-8 인코딩 문자입니다.

라이선스

IoT Analytics 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
4	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	과도한 로깅을 줄이도록 고정합니다.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- AWS IoT Analytics 사용 설명서의 AWS IoT Analytics란 무엇입니까?

IoT 이더넷 IP 프로토콜 어댑터 커넥터

IoT 이더넷 IP 프로토콜 어댑터 커넥터는 EtherNet/IP 프로토콜을 사용하여 로컬 디바이스에서 데이터를 수집합니다. 이 커넥터를 사용하여 여러 디바이스에서 데이터를 수집하여 `StreamManager` 메시지 스트림에 게시할 수 있습니다.

또한 이 커넥터를 IoT SiteWise 커넥터 및 IoT SiteWise 게이트웨이와 함께 사용할 수 있습니다. 게이트웨이는 커넥터에 대한 구성을 제공해야 합니다. 자세한 내용은 IoT SiteWise 사용 설명서의 이더넷/IP(EIP) 소스 구성을 참조하십시오.

> **Note**
>
> 이 커넥터는 컨테이너 없음 격리 모드에서 실행되므로, Docker 컨테이너에서 실행되는 AWS IoT Greengrass 그룹에 이 커넥터를 배포할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
2(권장)	`arn:aws:greengrass:region::/connectors/IoTEIPProtocolAdaptor/versions/2`
1	`arn:aws:greengrass:region::/connectors/IoTEIPProtocolAdaptor/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 1 and 2

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- AWS IoT Greengrass 그룹에 스트림 관리자가 활성화되어 있습니다.
- 코어 디바이스에 설치되고 `PATH` 환경 변수에 추가된 Java 8입니다.
- 최소 256MB의 추가 램이 필요합니다. 이 요구 사항과 AWS IoT Greengrass 코어 메모리 요구 사항은 별개입니다.

> **Note**
>
> 이 커넥터는 다음 AWS 리전에서만 사용할 수 있습니다.
>
> - cn-north-1

- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

커넥터 파라미터

이 커넥터는 다음 파라미터를 지원합니다.

`LocalStoragePath`

IoT SiteWise 커넥터가 영구 데이터를 기록할 수 있는 AWS IoT Greengrass 호스트의 디렉터리입니다. 기본 디렉터리는 `/var/sitewise`입니다.

AWS IoT 콘솔의 표시 이름: 로컬 스토리지 경로

필수: `false`

형식: `string`

유효한 패턴: `^\s*$|\/.`

`ProtocolAdapterConfiguration`

커넥터가 데이터를 수집하거나 연결하는 EtherNet/IP 컬렉터 구성 세트. 빈 목록일 수 있습니다.

AWS IoT 콘솔의 표시 이름: 프로토콜 어댑터 구성

필수: `true`

유형: 지원되는 피드백 구성 세트를 정의하는 올바른 형식의 JSON 문자열입니다.

다음은 `ProtocolAdapterConfiguration`의 예제입니다.

```
{
    "sources": [
```

```
        {
            "type": "EIPSource",
            "name": "TestSource",
            "endpoint": {
                "ipAddress": "52.89.2.42",
                "port": 44818
            },
            "destination": {
                "type": "StreamManager",
                "streamName": "MyOutput_Stream",
                "streamBufferSize": 10
            },
            "destinationPathPrefix": "EIPSource_Prefix",
            "propertyGroups": [
                {
                    "name": "DriveTemperatures",
                    "scanMode": {
                        "type": "POLL",
                        "rate": 10000
                    },
                    "tagPathDefinitions": [
                        {
                            "type": "EIPTagPath",
                            "path": "arrayREAL[0]",
                            "dstDataType": "double"
                        }
                    ]
                }
            ]
        }
    ]
}
```

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 IoT Ethernet IP Protocol Adapter 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
    "Connectors": [
        {
```

```
            "Id": "MyIoTEIPProtocolConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
            "Parameters": {
                "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
 \"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
 \"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
 \"arrayREAL[0]\", \"dstDataType\": \"double\" }]}]}]}",
                "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

입력 데이터

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

출력 데이터

이 커넥터는 `StreamManager`에 데이터를 게시합니다. 대상 메시지 스트림을 구성해야 합니다. 출력 메시지는 다음 구조를 취합니다.

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

라이선스

IoT Ethernet IP Protocol Adapter 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- 이더넷/IP 클라이언트
- MapDB
- Elsa

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경	날짜
2	이 버전에는 버그 수정이 포함되어 있습니다.	2021년 12월 23일
1	최초 릴리스.	2020년 12월 15일

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

IoT SiteWise 커넥터

IoT SiteWise 커넥터는 로컬 장치 및 장비 데이터를 의 자산 자산으로 보냅니다AWS IoT SiteWise. 이 커넥터를 사용하여 여러 OPC-UA 서버에서 데이터를 수집하고 IoT에 게시할 수 있습니다. SiteWise 커넥터는 데이터를 현재 AWS 계정 및 리전의 자산 속성으로 보냅니다.

> **Note**
>
> SiteWise IoT는 산업용 장치 및 장비로부터 데이터를 수집, 처리 및 시각화하는 완전 관리형 서비스입니다. 이 커넥터에서 자산의 측정 속성으로 전송된 원시 데이터를 처리하는 자산 속성을 구성할 수 있습니다. 예를 들어 디바이스의 섭씨 온도 데이터 포인트를 화씨로 변환하는 변환 속성을 정의하거나 시간당 평균 온도를 계산하는 지표 속성을 정의할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의(AWS IoT SiteWise이)란 무엇입니까? 섹션을 참조하십시오.

커넥터는 OPC-UA 서버에서 전송된 OPC-UA 데이터 스트림 경로를 SiteWise 사용하여 IoT로 데이터를 전송합니다. 예를 들어, 데이터 흐름 경로 `/company/windfarm/3/turbine/7/temperature`는 풍력 발전소 #3 터빈 #7의 온도 센서를 나타낼 수 있습니다. AWS IoT Greengrass 코어에서 인터넷 연결이 끊어지면 커넥터는 AWS 클라우드 클라우드에 성공적으로 연결할 수 있을 때까지 데이터를 캐싱합니다. 데이터 캐싱에 사용되는 최대 디스크 버퍼 크기를 구성할 수 있습니다. 캐시 크기가 최대 디스크 버퍼 크기를 초과하면 커넥터는 대기열에서 가장 오래된 데이터를 폐기합니다.

IoT SiteWise 커넥터를 구성하고 배포한 후 IoT 콘솔에서 게이트웨이 및 OPC-UA 소스를 추가할 수 있습니다. SiteWise 콘솔에서 소스를 구성할 때 IoT 커넥터가 전송하는 OPC-UA 데이터 스트림 경로를 필터링하거나 접두사를 붙일 수 있습니다. SiteWise 게이트웨이 및 소스 설정을 완료하는 방법에 대한 지침은 AWS IoT SiteWise 사용 설명서의 게이트웨이 추가를 참조하십시오.

IoT는 IoT SiteWise 자산의 측정 속성에 매핑한 데이터 스트림에서만 데이터를 SiteWise 수신합니다. 데이터 스트림을 자산 속성에 매핑하려면 속성의 별칭을 OPC-UA 데이터 스트림 경로와 동일하게 설정할 수 있습니다. 자산 모델 정의 및 자산 생성에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서의 산업 자산 모델링을 참조하십시오.

> **참고**
>
> 스트림 관리자를 사용하여 OPC-UA 서버 이외의 SiteWise 소스에서 IoT로 데이터를 업로드할 수 있습니다. 스트림 관리자는 또한 지속성 및 대역폭 관리를 위한 맞춤형 지원을 제공합니다. 자세한 설명은 데이터 스트림 관리 섹션을 참조하세요.
> 이 커넥터는 컨테이너 없음 격리 모드에서 실행되므로, Docker 컨테이너에서 실행되는 Greengrass 그룹에 이 커넥터를 배포할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
12(권장)	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/12
11	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/11
10	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/10
9	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/9
8	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/8
7	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/7
6	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/6
5	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/5
4	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/4

버전	ARN
3	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/3`
2	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/2`
1	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 9, 10, 11, and 12

> **Important**
> 이 버전에는 AWS IoT Greengrass 코어 소프트웨어 v1.10.0 및 스트림 관리자라는 새로운 요구 사항이 도입되었습니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.10.2.
- Greengrass 그룹에 스트림 관리자가 활성화되어 있습니다.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Java 8입니다.
- 이 SiteWise 커넥터는 IoT가 모두 AWS IoT Greengrass지원되는 Amazon Web Services 지역에서만 사용할 수 있습니다.
- Greengrass 그룹 역할에 추가된 IAM 정책. 이 역할을 사용하면 다음 예제와 같이 AWS IoT Greengrass 그룹이 대상 루트 자산 및 해당 하위 자산에서 `iotsitewise:BatchPutAssetPropertyValue` 작업에 액세스할 수 있습니다.

`Condition`정책에서 를 제거하여 커넥터가 모든 IoT SiteWise 자산에 액세스하도록 허용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Versions 6, 7, and 8

> **Important**
>
> 이 버전에는 AWS IoT Greengrass 코어 소프트웨어 v1.10.0 및 스트림 관리자라는 새로운 요구 사항이 도입되었습니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.10.0.
- Greengrass 그룹에 스트림 관리자가 활성화되어 있습니다.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Java 8입니다.
- 이 SiteWise 커넥터는 IoT가 모두 AWS IoT Greengrass지원되는 Amazon Web Services 지역에서만 사용할 수 있습니다.
- Greengrass 그룹 역할에 추가된 IAM 정책. 이 역할을 사용하면 다음 예제와 같이 AWS IoT Greengrass 그룹이 대상 루트 자산 및 해당 하위 자산에서

`iotsitewise:BatchPutAssetPropertyValue` 작업에 액세스할 수 있습니다. `Condition`정책에서 를 제거하여 커넥터가 모든 IoT SiteWise 자산에 액세스하도록 허용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Version 5

- AWS IoT Greengrass 코어 소프트웨어 v1.9.4 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Java 8입니다.
- 이 SiteWise 커넥터는 IoT가 모두 AWS IoT Greengrass지원되는 Amazon Web Services 지역에서만 사용할 수 있습니다.
- Greengrass 그룹 역할에 추가된 IAM 정책. 이 역할을 사용하면 다음 예제와 같이 AWS IoT Greengrass 그룹이 대상 루트 자산 및 해당 하위 자산에서 `iotsitewise:BatchPutAssetPropertyValue` 작업에 액세스할 수 있습니다. `Condition`정책에서 를 제거하여 커넥터가 모든 IoT SiteWise 자산에 액세스하도록 허용할 수 있습니다.

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

```
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Version 4

- AWS IoT Greengrass 코어 소프트웨어 v1.10.0.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Java 8입니다.
- 이 SiteWise 커넥터는 IoT가 모두 AWS IoT Greengrass지원되는 Amazon Web Services 지역에서만 사용할 수 있습니다.
- Greengrass 그룹 역할에 추가된 IAM 정책. 이 역할을 사용하면 다음 예제와 같이 AWS IoT Greengrass 그룹이 대상 루트 자산 및 해당 하위 자산에서 `iotsitewise:BatchPutAssetPropertyValue` 작업에 액세스할 수 있습니다. `Condition`정책에서 를 제거하여 커넥터가 모든 IoT SiteWise 자산에 액세스하도록 허용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
```

```
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
        }
    ]
}
```

자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Version 3

- AWS IoT Greengrass 코어 소프트웨어 v1.9.4 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Java 8입니다.
- 이 SiteWise 커넥터는 IoT가 모두 AWS IoT Greengrass지원되는 Amazon Web Services 지역에서만 사용할 수 있습니다.
- Greengrass 그룹 역할에 추가된 IAM 정책. 이 역할을 사용하면 다음 예제와 같이 AWS IoT Greengrass 그룹이 대상 루트 자산 및 해당 하위 자산에서 `iotsitewise:BatchPutAssetPropertyValue` 작업에 액세스할 수 있습니다. `Condition`정책에서 를 제거하여 커넥터가 모든 IoT SiteWise 자산에 액세스하도록 허용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
        }
```

```
    ]
}
```

자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Versions 1 and 2

- AWS IoT Greengrass 코어 소프트웨어 v1.9.4 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Java 8입니다.
- 이 SiteWise 커넥터는 IoT가 모두 AWS IoT Greengrass지원되는 Amazon Web Services 지역에서만 사용할 수 있습니다.
- 다음 예제와 같이 대상 루트 자산 및 하위 자산에서 AWS IoT Core 및 `iotsitewise:BatchPutAssetPropertyValue` 작업에 대한 액세스를 허용하는 Greengrass 그룹 역할에 추가된 IAM 정책. `Condition`정책에서 를 제거하여 커넥터가 모든 IoT SiteWise 자산에 액세스하도록 허용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        },
        {
            "Effect": "Allow",
            "Action": [
                 "iot:Connect",
                 "iot:DescribeEndpoint",
                 "iot:Publish",
                 "iot:Receive",
                 "iot:Subscribe"
```

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

자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하세요.

파라미터

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

`SiteWiseLocalStoragePath`

IoT SiteWise 커넥터가 영구 데이터를 쓸 수 있는 AWS IoT Greengrass 호스트의 디렉터리입니다. 기본값은 `/var/sitewise`입니다.

AWS IoT 콘솔의 표시 이름: 로컬 스토리지 경로

필수: `false`

유형: `string`

유효한 패턴: `^\s*$|\/.`

`AWSSecretsArnList`

각각에 OPC-UA 사용자 이름과 암호 키-값 페어가 포함된 AWS Secrets Manager의 암호 목록입니다. 각 암호는 키-값 페어 유형 암호여야 합니다.

AWS IoT 콘솔의 표시 이름: OPC-UA 사용자 이름/비밀번호 암호의 ARN 목록

필수: `false`

유형: `JsonArrayOfStrings`

유효한 패턴: `\[( ?,? ?\"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+\/)*[a-zA-Z0-9\/_+=,.@\\-]+-[a-zA-Z0-9]+)*\")*\]`

`MaximumBufferSize`

IoT SiteWise 디스크 사용량의 최대 크기 (GB) 기본값은 10GB입니다.

AWS IoT 콘솔의 표시 이름: 최대 디스크 버퍼 크기

필수: `false`

유형: `string`

유효한 패턴: `^\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

IoT SiteWise 커넥터가 영구 데이터를 쓸 수 있는 AWS IoT Greengrass 호스트의 디렉터리입니다. 기본값은 `/var/sitewise`입니다.

AWS IoT 콘솔의 표시 이름: 로컬 스토리지 경로

필수: `false`

유형: `string`

유효한 패턴: `^\s*$|\/.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

OPC-UA 사용자 이름과 암호 키-값 페어가 포함된 AWS Secrets Manager의 암호입니다. 이 암호는 키-값 페어 유형 암호여야 합니다.

AWS IoT 콘솔의 표시 이름: OPC-UA 사용자 이름/암호 비밀의 ARN

필수: `false`

유형: `string`

유효한 패턴: `^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+/)*[a-zA-Z0-9/_+=,.@\\-]+-[a-zA-Z0-9]+`

`SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId`

OPC-UA 사용자 이름 및 비밀번호 암호를 참조하는 AWS IoT Greengrass 그룹의 암호 리소스입니다.

AWS IoT 콘솔의 표시 이름: OPC-UA 사용자 이름/암호 비밀 리소스

필수: `false`

유형: `string`

유효한 패턴: `^$|.+`

`MaximumBufferSize`

IoT SiteWise 디스크 사용량의 최대 크기 (GB) 기본값은 10GB입니다.

AWS IoT 콘솔의 표시 이름: 최대 디스크 버퍼 크기

필수: `false`

유형: `string`

유효한 패턴: `^\s*$|[0-9]+`

커넥터 만들기 예(AWS CLI)

다음 AWS CLI 명령은 IoT SiteWise 커넥터가 `ConnectorDefinition` 포함된 초기 버전을 사용하여 a를 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTSiteWiseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/
versions/11"
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 설명은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

출력 데이터

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

Limits

이 커넥터에는 다음을 포함하여 SiteWise IoT에서 부과하는 다음과 같은 모든 제한이 적용됩니다. 자세한 내용은 AWS 일반 참조의 AWS IoT SiteWise 엔드포인트 및 할당량을 참조하십시오.

- AWS 계정당 최대 게이트웨이 수입니다.
- 게이트웨이당 최대 OPC-UA 소스 수입니다.
- 저장되는 데이터 포인트의 최대 속도 timestamp-quality-value (TQV) 입니다. AWS 계정
- 자산 속성별로 저장된 TQV 데이터 포인트의 최대 비율입니다.

라이선스

Version 9, 10, 11, and 12

IoT SiteWise 커넥터에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- MapDB
- Elsa
- Eclipse Milo

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Versions 6, 7, and 8

IoT SiteWise 커넥터에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- Milo/EDL 1.0

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Versions 1, 2, 3, 4, and 5

IoT SiteWise 커넥터에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- Milo/EDL 1.0

- Chronicle-Queue/Apache 라이선스 2.0

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경	날짜
12	• 이 버전에는 버그 수정이 포함되어 있습니다.	2021년 12월 22일
11	• 숨겨진 문자 또는 인쇄할 수 없는 문자가 포함된 문자열을 지원합니다. 숨겨진 문자와 인쇄할 수 없는 문자는 문자열이 AWS 클라우드(으)로 전송되기 전에 자동으로 제거됩니다. • IoT SiteWise 게이트웨이가 잘못된 요청을 무한 재시도하던 문제를 수정했습니다. • IoT SiteWise 게이트웨이가 고주파 데이터 소스에 연결되었을 때 체크포인트가 손상되는 문제를 수정했습니다. • 게이트웨이 구성 문제를 해결하는 데 도움이 되도록 오류 메시지를 개선했습니다.	2021년 3월 24일
10	소스 연결이 끊겼다가 다시 설정되었을 때 처리를 개선하도록 `StreamManager` 을(를) 구성하였습니다. 또한 이 버전은	2021년 1월 22일

버전	변경	날짜
	`SourceTimestamp` 을(를) 사용할 수 없을 때 `ServerTimestamp` 와(과) 함께 OPC-UA 값을 허용합니다.	
9	사용자 지정 Greengrass `StreamManager` 스트림 대상, OPC-UA 데드밴딩, 사용자 지정 스캔 모드 및 사용자 지정 스캔 속도에 대한 지원이 시작되었습니다. 또한 IoT SiteWise 게이트웨이에서 구성을 업데이트하는 동안 개선된 성능도 포함됩니다.	2020년 12월 15일
8	커넥터에 간헐적인 네트워크 연결이 발생할 때의 안정성이 개선되었습니다.	2020년 11월 19일
7	게이트웨이 지표 관련 문제가 해결되었습니다.	2020년 8월 14일
6	새로운 OPC-UA 태그의 CloudWatch 메트릭 및 자동 검색에 대한 지원이 추가되었습니다. 이 버전은 스트림 관리자와 AWS IoT Greengrass 코어 소프트웨어 v1.10.0 이상이 필요합니다.	2020년 4월 29일
5	AWS IoT Greengrass 코어 소프트웨어 v1.9.4와의 호환성 문제가 해결되었습니다.	2020년 2월 12일
4	OPC-UA 서버 재연결 문제가 해결되었습니다.	2020년 2월 7일

버전	변경	날짜
3	`iot:*` 권한 요구 사항이 제거되었습니다.	2019년 12월 17일
2	여러 OPC-UA 암호 리소스에 대한 지원이 추가되었습니다.	2019년 12월 10일
1	최초 릴리스.	2019년 12월 2일

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드”을 참조하십시오.

다음 사항도 참조하십시오.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- AWS IoT SiteWise 사용 설명서의 다음 항목을 참조하십시오.
  - AWS IoT SiteWise란 무엇입니까?
  - 게이트웨이 사용
  - 게이트웨이 메트릭 CloudWatch
  - IoT SiteWise 게이트웨이 문제 해결

Kinesis Firehose

Kinesis Firehose 커넥터는 Amazon Data Firehose 전송 스트림을 통해 Amazon S3, Amazon Redshift 또는 Amazon 서비스와 같은 대상에 데이터를 게시합니다. OpenSearch

이 커넥터는 Kinesis 전송 스트림을 위한 데이터 생산자입니다. MQTT 주제에 대한 입력 데이터를 수신해 지정된 전송 스트림으로 보냅니다. 그런 다음 전송 스트림에서는 해당 데이터 레코드를 구성된 대상(예: S3 버킷)으로 보냅니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
5	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/5`
4	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/4`
3	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/3`
2	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/2`
1	`arn:aws:greengrass:region::/connectors/KinesisFirehose/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 4 - 5

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 구성된 Kinesis 전송 스트림입니다. 자세한 내용은 Amazon Kinesis Firehose 개발자 안내서의 Amazon Data Firehose 전송 스트림 생성을 참조하십시오.
- 다음 예제에 표시된 것처럼 Greengrass 그룹 역할에 추가된, 대상 전송 스트림에 대한 `firehose:PutRecord` 및 `firehose:PutRecordBatch` 작업을 허용하는 IAM 정책입니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
}
```

이 커넥터를 사용하면 입력 메시지 페이로드에서 기본 전송 스트림을 동적으로 재정의할 수 있습니다. 이 기능을 사용하여 구현하는 경우, IAM 정책에 모든 대상 스트림이 리소스로 포함되어야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해).

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)”를 참조하세요.

Versions 2 - 3

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- 구성된 Kinesis 전송 스트림입니다. 자세한 내용은 Amazon Kinesis Firehose 개발자 안내서의 Amazon Data Firehose 전송 스트림 생성을 참조하십시오.
- 다음 예제에 표시된 것처럼 Greengrass 그룹 역할에 추가된, 대상 전송 스트림에 대한 `firehose:PutRecord` 및 `firehose:PutRecordBatch` 작업을 허용하는 IAM 정책입니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
 }
```

이 커넥터를 사용하면 입력 메시지 페이로드에서 기본 전송 스트림을 동적으로 재정의할 수 있습니다. 이 기능을 사용하여 구현하는 경우, IAM 정책에 모든 대상 스트림이 리소스로 포함되어야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해).

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)”를 참조하세요.

Version 1

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.

- 구성된 Kinesis 전송 스트림입니다. 자세한 내용은 Amazon Kinesis Firehose 개발자 안내서의 Amazon Data Firehose 전송 스트림 생성을 참조하십시오.
- 다음 예제에 표시된 것처럼 Greengrass 그룹 역할에 추가된, 대상 전송 스트림에 대한 `firehose:PutRecord` 작업을 허용하는 IAM 정책입니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
}
```

이 커넥터를 사용하면 입력 메시지 페이로드에서 기본 전송 스트림을 동적으로 재정의할 수 있습니다. 이 기능을 사용하여 구현하는 경우, IAM 정책에 모든 대상 스트림이 리소스로 포함되어야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해).

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)”를 참조하세요.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Versions 5

`DefaultDeliveryStreamArn`

데이터를 전송할 기본 Firehose 전송 스트림의 ARN입니다. 전송 스트림은 입력 메시지 페이로드에서 `delivery_stream_arn` 속성으로 재정의될 수 있습니다.

> **Note**
>
> 그룹 역할이 모든 대상 전송 스트림에 대해 적절한 작업을 허용해야 합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 기본 전송 스트림 ARN

필수: `true`

유형: `string`

유효한 패턴: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

동일한 전송 스트림에 대한 새 레코드가 거부되기 전에 메모리에 유지할 최대 레코드 수입니다. 최소값은 2000입니다.

AWS IoT 콘솔의 표시 이름: 버퍼링할 최대 레코드 수 (스트림당)

필수: `true`

유형: `string`

유효한 패턴: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

이 커넥터에 할당할 메모리 양(KB)입니다.

AWS IoT 콘솔의 디스플레이 이름: 메모리 크기

필수: `true`

유형: `string`

유효한 패턴: `^[0-9]+$`

`PublishInterval`

Firehose에 레코드를 게시하는 간격 (초) 입니다. 배치를 비활성화하려면 이 값을 0으로 설정합니다.

AWS IoT 콘솔의 표시 이름: 게시 간격

필수: `true`

유형: `string`

유효값: `0 - 900`

유효한 패턴: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 이며`GreengrassContainer`, 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

유형: `string`

유효한 값: `GreengrassContainer`또는 `NoContainer`

유효한 패턴: `^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

`DefaultDeliveryStreamArn`

데이터를 전송할 기본 Firehose 전송 스트림의 ARN입니다. 전송 스트림은 입력 메시지 페이로드에서 `delivery_stream_arn` 속성으로 재정의될 수 있습니다.

> **Note**
> 그룹 역할이 모든 대상 전송 스트림에 대해 적절한 작업을 허용해야 합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 기본 전송 스트림 ARN

필수: `true`

유형: `string`

유효한 패턴: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

동일한 전송 스트림에 대한 새 레코드가 거부되기 전에 메모리에 유지할 최대 레코드 수입니다. 최소값은 2000입니다.

AWS IoT 콘솔의 표시 이름: 버퍼링할 최대 레코드 수 (스트림당)

필수: `true`

유형: `string`

유효한 패턴: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

이 커넥터에 할당할 메모리 양(KB)입니다.

AWS IoT 콘솔의 디스플레이 이름: 메모리 크기

필수: `true`

유형: `string`

유효한 패턴: `^[0-9]+$`

`PublishInterval`

Firehose에 레코드를 게시하는 간격 (초) 입니다. 배치를 비활성화하려면 이 값을 0으로 설정합니다.

AWS IoT 콘솔의 표시 이름: 게시 간격

필수: `true`

유형: `string`

유효값: `0 - 900`

유효한 패턴: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

`DefaultDeliveryStreamArn`

데이터를 전송할 기본 Firehose 전송 스트림의 ARN입니다. 전송 스트림은 입력 메시지 페이로드에서 `delivery_stream_arn` 속성으로 재정의될 수 있습니다.

> **Note**
>
> 그룹 역할이 모든 대상 전송 스트림에 대해 적절한 작업을 허용해야 합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 기본 전송 스트림 ARN

필수: `true`

유형: `string`

유효한 패턴: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyKinesisFirehoseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
            "Parameters": {
                "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
                "DeliveryStreamQueueSize": "5000",
                "MemorySize": "65535",
                "PublishInterval": "10",
                "IsolationMode" : "GreengrassContainer"
```

```
            }
        }
    ]
}'
```

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 설명은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 주제에 대한 스트림 콘텐츠를 수락한 다음, 이 콘텐츠를 대상 전송 스트림에 전송합니다. 다음과 같은 두 가지 유형의 입력 데이터를 수락합니다.

- `kinesisfirehose/message` 주제에 대한 JSON 데이터.
- `kinesisfirehose/message/binary/#` 주제에 대한 이진 데이터.

Versions 2 - 5

주제 필터: `kinesisfirehose/message`

이 주제는 JSON 데이터가 포함된 메시지를 보내는데 사용합니다.

메시지 속성

`request`

기본 스트림과 다른 경우 전송 스트림과 대상 전송 스트림으로 보낸 데이터입니다.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`data`

전송 스트림으로 보낸 데이터입니다.

필수: `true`

유형: `string`

`delivery_stream_arn`

대상 Kinesis 전송 스트림의 ARN입니다. 기본 전송 스트림을 재정의하려면 이 속성을 포함합니다.

필수: false

유형: string

유효한 패턴: arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$

id

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다. 지정하면 응답 객체의 id 속성이 이 값으로 설정됩니다. 이 기능을 사용하지 않는 경우 이 속성을 생략하거나 빈 문자열로 지정할 수 있습니다.

필수: false

유형: string

유효한 패턴: .*

입력 예

```
{
     "request": {
        "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
        "data": "Data to send to the delivery stream."
     },
     "id": "request123"
}
```

주제 필터: kinesisfirehose/message/binary/#

이 주제는 이진 데이터가 포함된 메시지를 보내는데 사용합니다. 이 커넥터는 이진 데이터를 구문 분석하지 않습니다. 이진 데이터는 있는 그대로 스트리밍됩니다.

입력 요청을 출력 응답에 매핑하려면 메시지 주제의 # 와일드카드를 임의의 요청 ID로 바꿉니다. 예를 들어, 메시지를 kinesisfirehose/message/binary/request123에 게시한 경우 응답 객체의 id 속성이 request123으로 설정됩니다.

요청을 응답에 매핑하지 않으려는 경우에는 메시지를 kinesisfirehose/message/binary/에 게시할 수 있습니다. 반드시 후행 슬래시를 포함해야 합니다.

Version 1

주제 필터: kinesisfirehose/message

이 주제는 JSON 데이터가 포함된 메시지를 보내는데 사용합니다.

메시지 속성

request

기본 스트림과 다른 경우 전송 스트림과 대상 전송 스트림으로 보낸 데이터입니다.

필수: true

유형: 다음 속성을 포함하는 object:

data

전송 스트림으로 보낸 데이터입니다.

필수: true

유형: string

delivery_stream_arn

대상 Kinesis 전송 스트림의 ARN입니다. 기본 전송 스트림을 재정의하려면 이 속성을 포함합니다.

필수: false

유형: string

유효한 패턴: arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$

id

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다. 지정하면 응답 객체의 id 속성이 이 값으로 설정됩니다. 이 기능을 사용하지 않는 경우 이 속성을 생략하거나 빈 문자열로 지정할 수 있습니다.

필수: false

유형: string

유효한 패턴: `.*`

입력 예

```
{
     "request": {
        "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
        "data": "Data to send to the delivery stream."
     },
     "id": "request123"
}
```

주제 필터: `kinesisfirehose/message/binary/#`

이 주제는 이진 데이터가 포함된 메시지를 보내는데 사용합니다. 이 커넥터는 이진 데이터를 구문 분석하지 않습니다. 이진 데이터는 있는 그대로 스트리밍됩니다.

입력 요청을 출력 응답에 매핑하려면 메시지 주제의 # 와일드카드를 임의의 요청 ID로 바꿉니다. 예를 들어, 메시지를 `kinesisfirehose/message/binary/request123`에 게시한 경우 응답 객체의 `id` 속성이 `request123`으로 설정됩니다.

요청을 응답에 매핑하지 않으려는 경우에는 메시지를 `kinesisfirehose/message/binary/`에 게시할 수 있습니다. 반드시 후행 슬래시를 포함해야 합니다.

출력 데이터

이 커넥터는 상태 정보를 MQTT 주제에 출력 데이터로 게시합니다.

Versions 2 - 5

구독의 주제 필터

`kinesisfirehose/message/status`

출력 예

응답에는 배치로 전송된 각 데이터 레코드의 상태가 포함됩니다.

```
{
```

```
    "response": [
        {
            "ErrorCode": "error",
            "ErrorMessage": "test error",
            "id": "request123",
            "status": "fail"
        },
        {
            "firehose_record_id": "xyz2",
            "id": "request456",
            "status": "success"
        },
        {
            "firehose_record_id": "xyz3",
            "id": "request890",
            "status": "success"
        }
    ]
}
```

> **Note**
>
> 커넥터가 재시도 가능한 오류(예: 연결 오류)를 감지하면 다음 배치에서 게시를 재시도합니다. 지수 백오프는 SDK에서 처리합니다. AWS 재시도할 수 있는 오류로 실패한 요청은 추가 게시를 위해 다시 채널 대기열에 추가됩니다.

Version 1

구독의 주제 필터

```
kinesisfirehose/message/status
```

출력 예: 성공

```
{
   "response": {
       "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",
       "status": "success"
    },
    "id": "request123"
}
```

출력 예: 실패

```
{
   "response" : {
       "error": "ResourceNotFoundException",
       "error_message": "An error occurred (ResourceNotFoundException) when
 calling the PutRecord operation: Firehose test1 not found under account
 123456789012.",
       "status": "fail"
   },
   "id": "request123"
}
```

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.

   그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)”를 참조하세요.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.

3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 커넥터를 추가하고 해당 파라미터를 구성합니다.

   c. 커넥터가 JSON 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core 를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 콘솔에서 상태 메시지를 볼 수 있습니다. AWS IoT

4. 그룹을 배포합니다.

5. AWS IoT 콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하면 커넥터의 상태 메시지를 볼 수 있습니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다. 이 메시지에는 JSON 데이터가 포함되어 있습니다.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }
```

```
def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

라이선스

Kinesis Firehose 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
5	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
4	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.

버전	변경
3	과도한 로깅을 줄이기 위한 수정 및 기타 사소한 버그 수정입니다.
2	지정된 간격으로 Firehose에 일괄 데이터 레코드를 전송하는 지원이 추가되었습니다. • 또한 그룹 역할에서 `firehose:PutRecordBatch` 작업이 필요합니다. • 새로운 `MemorySize`, `DeliveryStreamQueueSize` 및 `PublishInterval` 파라미터입니다. • 출력 메시지에는 게시된 데이터 레코드에 대한 일련의 상태 응답이 포함됩니다.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드”을 참조하십시오.

다음 사항도 참조하십시오.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- Amazon Kinesis 개발자 안내서의 Amazon Kinesis Data Firehose란 무엇인가요?

ML 피드백 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

ML 피드백 커넥터는 모델 재교육 및 분석을 위해 보다 쉽게 기계 학습(ML) 모델 데이터에 액세스할 수 있게 해줍니다. 이 커넥터는

- ML 모델이 사용하는 입력 데이터(샘플)를 Amazon S3로 업로드합니다. 모델 입력은 이미지, JSON, 오디오 등 임의의 형식일 수 있습니다. 샘플을 클라우드로 업로드한 후 이를 사용하여 모델을 재교육함으로써 예측의 정확성 및 정밀도를 개선할 수 있습니다. 예를 들어 SageMaker Ground Truth를 사용하여 샘플을 라벨링하고 SageMaker를 사용하여 모델을 재교육할 수 있습니다.
- 모델의 예측 결과를 MQTT 메시지로 게시합니다. 이를 통해 실시간으로 모델의 추론 품질을 모니터링 및 분석할 수 있습니다. 또한 예측 결과를 저장하고 이를 사용하여 경시적 추세를 분석할 수 있습니다.
- 샘플 업로드 및 샘플 데이터에 대한 지표를 Amazon CloudWatch에 게시합니다.

이 커넥터를 구성하려면 지원되는 피드백 구성을 JSON 형식으로 설명해야 합니다. 피드백 구성은 Amazon S3 버킷, 콘텐츠 유형, 샘플링 전략.과 같은 속성을 정의합니다. (샘플링 전략은 어느 샘플을 업로드할지 결정하는 데 사용됩니다.)

다음과 같은 상황에서 ML 피드백 커넥터를 사용할 수 있습니다.

- 사용자 정의 Lambda 함수와 함께 사용합니다. 로컬 추론 Lambda 함수는 AWS IoT Greengrass 기계 학습 SDK를 사용하여 이 커넥터를 간접 호출하고 대상 피드백 구성, 모델 입력, 모델 출력(예측 결과)을 전달합니다. 예시는 the section called “사용 예”에서 확인하세요.
- ML Image Classification 커넥터(v2) 사용. 이 커넥터를 ML Image Classification 커넥터와 함께 사용하려면 ML Image Classification 커넥터에 대해 `MLFeedbackConnectorConfigId` 파라미터를 구성합니다.
- ML Object Detection 커넥터 사용. 이 커넥터를 ML Object Detection 커넥터와 함께 사용하려면 ML Object Detection 커넥터에 대해 `MLFeedbackConnectorConfigId` 파라미터를 구성합니다.

ARN: `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 하나 이상의 Amazon S3 버킷. 사용하는 버킷 수는 샘플링 전략에 따라 달라집니다.
- 다음 예제와 같이, 대상 Amazon S3 버킷의 객체에 대한 `s3:PutObject` 작업을 허용하는 Greengrass 그룹 역할에 IAM 정책을 추가해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

  이 정책은 모든 대상 버킷을 리소스로 포함해야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해).

  그룹 역할 요구 사항의 경우 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

- loudWatch Metrics 커넥터를 Greengrass 그룹에 추가 및 구성해야 합니다. 지표 보고 기능을 사용하려는 경우에만 필요합니다.
- 이 커넥터와 상호 작용하려면 AWS IoT Greengrass 기계 학습 SDK v1.1.0이 필요합니다.

파라미터

`FeedbackConfigurationMap`

커넥터가 샘플을 Amazon S3에 업로드하는 데 사용할 수 있는 하나 이상의 피드백 구성의 집합입니다. 피드백 구성은 대상 버킷, 콘텐츠 유형, 샘플링 전략과 같은 파라미터를 정의합니다. 이 커넥터가 간접 호출될 경우 직접 호출하는 Lambda 함수 또는 커넥터가 대상 피드백 구성을 지정합니다.

AWS IoT 콘솔의 표시 이름: 피드백 구성 맵

필수: `true`

유형: 지원되는 피드백 구성 세트를 정의하는 올바른 형식의 JSON 문자열입니다. 예시는 the section called “FeedbackConfigurationMap 예제”에서 확인하세요.

피드백 구성 객체의 ID는 다음 요구 사항을 충족해야 합니다.

ID:

- 구성 객체 간에 고유해야 합니다.
- 문자 또는 숫자로 시작해야 합니다. 소문자 및 대문자, 숫자, 하이픈(-)을 포함할 수 있습니다.
- 2~63자 길이여야 합니다.

필수: `true`

형식: `string`

유효한 패턴: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

예: `MyConfig0`, `config-a`, `12id`

피드백 구성 객체의 본문은 다음 속성을 포함해야 합니다.

`s3-bucket-name`

대상 Amazon S3 버킷의 이름입니다.

> **Note**
>
> 그룹 역할은 모든 대상 버킷에 대한 `s3:PutObject` 작업을 허용해야 합니다. 자세한 내용은 the section called “요구 사항” 섹션을 참조하세요.

필수: `true`

형식: `string`

유효한 패턴: `^[a-z0-9\.\-]{3,63}$`

`content-type`

업로드할 샘플의 콘텐츠 유형입니다. 개별 피드백 구성에 대한 모든 콘텐츠는 동일한 유형이어야 합니다.

필수: `true`

형식: `string`

예: `image/jpeg`, `application/json`, `audio/ogg`

`s3-prefix`

업로드된 샘플에 사용할 키 접두사입니다. 접두사는 디렉터리 이름과 유사합니다. 이를 사용해 한 버킷의 동일한 디렉터리 아래에 유사한 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 키 및 메타데이터를 참조하십시오.

필수: `false`

형식: `string`

`file-ext`

업로드된 샘플에 사용할 파일 확장명입니다. 해당 콘텐츠 유형에 유효한 파일 확장명이어야 합니다.

필수: `false`

형식: `string`

예: `jpg`, `json`, `ogg`

`sampling-strategy`

업로드할 샘플을 필터링하는 데 사용할 샘플링 전략입니다. 생략할 경우 커넥터가 수신하는 모든 샘플을 업로드하려고 시도합니다.

필수: `false`

유형: 다음 속성을 포함하는 올바른 형식의 JSON 문자열입니다.

`strategy-name`

샘플링 전략의 이름입니다.

필수: `true`

형식: `string`

유효한 값: `RANDOM_SAMPLING`, `LEAST_CONFIDENCE`, `MARGIN` 또는 `ENTROPY`

`rate`

무작위 샘플링 전략의 비율입니다.

필수: `strategy-name`가 `RANDOM_SAMPLING`인 경우 `true`.

형식: `number`

유효한 값: `0.0 - 1.0`

`threshold`

최소 신뢰도, 마진 또는 엔트로피 샘플링 전략의 임계값입니다.

필수: `strategy-name`가 `LEAST_CONFIDENCE`, `MARGIN`, 또는 `ENTROPY`인 경우, `true`.

형식: `number`

유효 값:

- `LEAST_CONFIDENCE` 또는 `MARGIN` 전략의 경우 `0.0 - 1.0`.
- `ENTROPY` 전략의 경우 `0.0 - no limit`.

`RequestLimit`

커넥터가 한 번에 처리할 수 있는 요청의 최대 수입니다.

이 파라미터를 사용하여 커넥터가 동시에 처리할 수 있는 요청 수를 제한하면 메모리 사용을 제한할 수 있습니다. 이 제한을 초과하는 요청은 무시됩니다.

AWS IoT 콘솔의 표시 이름: 요청 제한

필수: `false`

형식: `string`

유효한 값: `0 - 999`

유효한 패턴: `^$|^[0-9]{1,3}$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 ML 피드백 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyMLFeedbackConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
            "Parameters": {
                "FeedbackConfigurationMap": "{  \"RandomSamplingConfiguration\":
 {  \"s3-bucket-name\": \"my-aws-bucket-random-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name
\": \"RANDOM_SAMPLING\",  \"rate\": 0.5  } },  \"LeastConfidenceConfiguration\": {
  \"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name\":
 \"LEAST_CONFIDENCE\",  \"threshold\": 0.4  } } }",
                "RequestLimit": "10"
            }
        }
    ]
}'
```

FeedbackConfigurationMap 예제

다음은 `FeedbackConfigurationMap` 파라미터의 확장된 예제 값입니다. 이 예제에는 서로 다른 샘플링 전략을 사용하는 여러 피드백 구성이 포함되어 있습니다.

```
{
    "ConfigID1": {
        "s3-bucket-name": "my-aws-bucket-random-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "RANDOM_SAMPLING",
```

```
            "rate": 0.5
        }
    },
    "ConfigID2": {
        "s3-bucket-name": "my-aws-bucket-margin-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "MARGIN",
            "threshold": 0.4
        }
    },
    "ConfigID3": {
        "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "LEAST_CONFIDENCE",
            "threshold": 0.4
        }
    },
    "ConfigID4": {
        "s3-bucket-name": "my-aws-bucket-entropy-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "ENTROPY",
            "threshold": 2
        }
    },
    "ConfigID5": {
        "s3-bucket-name": "my-aws-bucket-no-sampling",
        "s3-prefix": "DeviceA",
        "content-type": "application/json"
    }
}
```

샘플링 전략

이 커넥터는 커넥터로 전달되는 샘플을 업로드할지 여부를 결정하는 4개의 샘플링 전략을 지원합니다. 샘플은 모델이 예측을 위해 사용하는 개별 데이터 인스턴스입니다. 샘플링 전략을 사용하여 모델 정확성을 가장 개선할 수 있는 샘플을 선택할 수 있습니다.

`RANDOM_SAMPLING`

지정된 비율에 따라 무작위로 샘플을 업로드합니다. 무작위로 생성된 값이 비율보다 작은 경우 샘플을 업로드합니다. 비율이 높을수록 많은 샘플이 업로드됩니다.

> **Note**
>
> 이 전략은 제공된 모델 예측을 모두 무시합니다.

`LEAST_CONFIDENCE`

최대 신뢰도 확률이 지정된 임계값보다 낮은 샘플을 업로드합니다.

예제 시나리오:

임계값: `.6`

모델 예측: `[.2, .2, .4, .2]`

최대 신뢰도 확률: `.4`

Result:

최대 신뢰도 확률(`.4`) <= 임계값(`.6`)이므로 샘플을 사용합니다.

`MARGIN`

두 신뢰도 확률 간 마진이 지정된 임계값 이내일 경우 샘플을 업로드합니다. 마진은 2개의 최대 확률 간 차이입니다.

예제 시나리오:

임계값: `.02`

모델 예측: `[.3, .35, .34, .01]`

2개의 최대 신뢰도 확률: `[.35, .34]`

마진: `.01(.35 - .34)`

Result:

마진(`.01`) <= 임계값(`.02`)이므로 샘플을 사용합니다.

ENTROPY

엔트로피가 지정된 임계값보다 높은 샘플을 사용합니다. 모델 예측의 정규화된 엔트로피를 사용합니다.

예제 시나리오:

임계값: `0.75`

모델 예측: `[.5, .25, .25]`

예측 엔트로피: `1.03972`

Result:

엔트로피(`1.03972`) > 임계값(`0.75`)이므로 샘플을 사용합니다.

입력 데이터

사용자 정의 `publish` 함수는 AWS IoT Greengrass 기계 학습 SDK에서 `feedback` 클라이언트의 Lambda 함수를 사용하여 커넥터를 간접 호출합니다. 예시는 the section called “사용 예”에서 확인하세요.

> **Note**
>
> 이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

`publish` 함수는 다음 인수를 사용합니다.

ConfigId

대상 피드백 구성의 ID입니다. ML 피드백 커넥터에 대한 FeedbackConfigurationMap 파라미터에 정의된 피드백 구성의 ID와 일치해야 합니다.

필수: True

유형: 문자열

ModelInput

추론을 위해 모델에 전달된 입력 데이터입니다. 이 입력 데이터는 샘플링 전략에 의해 제외되지 않는 한 대상 구성을 사용하여 업로드됩니다.

필수: True

유형: 바이트

ModelPrediction

모델의 예측 결과입니다. 결과 형식은 사전 또는 목록입니다. 예를 들어 ML Image Classification 커넥터의 예측 결과는 확률 목록입니다(예: `[0.25, 0.60, 0.15]`). 이 데이터는 `/feedback/message/prediction` 주제에 게시됩니다.

필수: True

유형:`float` 값의 목록 또는 사전

Metadata

업로드된 샘플에 연결되고 `/feedback/message/prediction` 주제에 게시되는 고객이 정의한 애플리케이션 특정 메타데이터입니다. 또한 커넥터는 타임스탬프 값을 포함하여 `publish-ts` 키를 메타데이터에 삽입합니다.

필수: 거짓

유형: 사전

예: `{"some-key": "some value"}`

출력 데이터

이 커넥터는 다음 3개의 MQTT 주제에 데이터를 게시합니다.

- `feedback/message/status` 주제에 대한 커넥터의 상태 정보.
- `feedback/message/prediction` 주제에 대한 예측 결과.
- CloudWatch로 전송할 `cloudwatch/metric/put` 주제에 대한 지표.

커넥터가 MQTT 주제에서 통신할 수 있게 허용하도록 구독을 구성해야 합니다. 자세한 내용은 the section called “입력 및 출력” 섹션을 참조하세요.

주제 필터: `feedback/message/status`

이 주제를 사용하여 샘플 업로드 상태 및 삭제된 샘플을 모니터링합니다. 커넥터는 요청을 수신할 때마다 이 주제에 게시합니다.

출력 예: 샘플 업로드가 성공함

```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
 "lbbqaDVFOhMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
 rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
          "x-amz-request-id": "79104EXAMPLEB723",
          "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
          "date": "Thu, 11 Jul 2019 00:12:50 GMT",
          "x-amz-server-side-encryption": "AES256"
        }
      },
      "bucket": "greengrass-feedback-connector-data-us-west-2",
      "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
      "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
      "key": "s3-key-prefix/UUID.file_ext",
      "ServerSideEncryption": "AES256"
    }
  },
  "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}
```

커넥터가 Amazon S3로부터의 응답에 `bucket` 및 `key` 필드를 추가합니다. Amazon S3 응답에 대한 자세한 내용은 Amazon Simple Storage Service API 참조의 PUT 객체를 참조하십시오.

출력 예: 샘플링 전략 때문에 샘플이 삭제됨

```
{
  "response": {
```

```
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

출력 예: 샘플 업로드가 실패함

실패 상태는 오류 메시지(error_message 값)와 예외 클래스(error 값)을 포함합니다.

```
{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed
 to upload model input data due to exception. Model prediction will not be
 published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket)
 when calling the PutObject operation: The specified bucket does not exist",
    "error": "NoSuchBucket"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

출력 예: 요청 제한 때문에 요청이 조절됨

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
 request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

주제 필터: feedback/message/prediction

이 주제를 사용하여 업로드된 샘플 데이터를 기반으로 예측을 수신 대기합니다. 그러면 모델 성능을 실시간으로 분석할 수 있습니다. 데이터가 성공적으로 Amazon S3에 업로드된 경우에만 모델 예측이 게시됩니다. 이 주제에 게시되는 메시지는 JSON 형식입니다. 여기에 업로드된 데이터 객체에 대한 링크, 모델 예측 및 요청에 포함된 메타데이터가 포함됩니다.

또한 예측 결과를 저장하고 이를 사용하여 경시적 추세를 보고 및 분석할 수 있습니다. 추세는 가치 있는 통찰을 제공할 수 있습니다. 예를 들어 경시적으로 정확성이 감소하는 추세는 모델을 재교육해야 할지 여부를 결정하는 데 도움이 될 수 있습니다.

출력 예

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/
UUID.file_ext",
  "model-prediction": [
    0.5,
    0.2,
    0.2,
    0.1
  ],
  "config-id": "ConfigID2",
  "metadata": {
    "publish-ts": "2019-07-11 00:12:48.816752"
  }
}
```

> **Tip**
>
> 이 주제를 구독하고 추가 분석 또는 기록 분석을 위해 정보를 AWS IoT Analytics에 전송하도록 IoT Analytics 커넥터를 구성할 수 있습니다.

주제 필터: `cloudwatch/metric/put`

CloudWatch에 지표를 게시하는 데 사용되는 출력 주제입니다. 이 기능을 사용하려면 CloudWatch Metrics 커넥터를 설치 및 구성해야 합니다.

지표는 다음과 같습니다.

- 업로드된 샘플 수.
- 업로드된 샘플의 크기.
- Amazon S3로 업로드 시 오류 수.
- 샘플링 전략에 따라 삭제된 샘플 수.
- 조정된 요청 수.

출력 예: 데이터 샘플 크기(실제 업로드 전 게시됨)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

출력 예: 샘플 업로드가 성공함

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

출력 예: 성공한 샘플 업로드 및 게시된 예측 결과

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleAndPredictionPublished"
    }
  }
}
```

출력 예: 샘플 업로드가 실패함

```
{
```

```
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
}
```

출력 예: 샘플링 전략 때문에 샘플이 삭제됨

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleNotUsed"
    }
  }
}
```

출력 예: 요청 제한 때문에 요청이 조절됨

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

사용 예

다음 예제는 AWS IoT Greengrass 기계 학습 SDK를 사용하여 ML 피드백 커넥터로 데이터를 전송하는 사용자 정의 Lambda 함수입니다.

> **Note**
>
> AWS IoT Greengrass 다운로드 페이지에서 AWS IoT Greengrass 기계 학습 SDK를 다운로드할 수 있습니다.

```
import json
import logging
import os
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')

try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
 'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
    sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
```

```
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

라이선스

ML 피드백 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스
- six/MIT

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

ML 이미지 분류 커넥터

> **Warning**
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

ML 이미지 분류 커넥터는 AWS IoT Greengrass에서 실행되는 ML(기계 학습) 추론 서비스를 제공합니다. 이 로컬 추론 서비스는 SageMaker 이미지 분류 알고리즘이 훈련한 모델을 사용해 이미지를 분류합니다.

사용자 정의 Lambda 함수는 AWS IoT Greengrass 기계 학습을 사용하여 로컬 추론 서비스에 추론 요청을 제출합니다. 이 서비스를 로컬에서 추론을 실행하고 입력 이미지가 특정 범주에 속할 가능성을 반환합니다.

AWS IoT Greengrass는 여러 플랫폼에서 사용할 수 있는 이 커넥터에 대해 다음 버전을 제공합니다.

Version 2

커넥터	설명 및 ARN
ML 이미지 분류: Aarch64 JTX2	NVIDIA Jetson TX2용 이미지 분류 추론 서비스입니다. GPU 가속화를 지원합니다. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationAarch64JTX2/versions/2`
ML 이미지 분류 x86_64	x86_64 플랫폼용 이미지 분류 추론 서비스입니다. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationx86-64/versions/2`

커넥터	설명 및 ARN
ML 이미지 분류 ARMv7	ARMv7 플랫폼용 이미지 분류 추론 서비스입니다. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationARMv7/versions/2`

Version 1

커넥터	설명 및 ARN
ML 이미지 분류: Aarch64 JTX2	NVIDIA Jetson TX2용 이미지 분류 추론 서비스입니다. GPU 가속화를 지원합니다. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationAarch64JTX2/versions/1`
ML 이미지 분류 x86_64	x86_64 플랫폼용 이미지 분류 추론 서비스입니다. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationx86-64/versions/1`
ML 이미지 분류 Armv7	ARMv7 플랫폼용 이미지 분류 추론 서비스입니다. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationARMv7/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이러한 커넥터에는 다음 요구 사항이 있습니다.

Version 2

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

  > **Note**
  >
  > Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 코어 디바이스에 설치된 Apache MXNet 프레임워크에 대한 종속성. 자세한 내용은 the section called “MXNet 종속성 설치” 섹션을 참조하세요.
- SageMaker 모델 소스를 참조하는 Greengrass 그룹의 ML 리소스입니다. 이 모델은 SageMaker 이미지 분류 알고리즘이 학습해야 합니다. 자세한 내용은 Amazon SageMaker 개발자 안내서의 이미지 분류 알고리즘을 참조하세요.
- ML 피드백 커넥터를 Greengrass 그룹에 추가 및 구성해야 합니다. 이 파라미터는 이 커넥터를 사용하여 모델 입력 데이터를 업로드하고 예측을 MQTT 주제에 게시하려는 경우에만 필요합니다.
- 다음 예제 IAM 정책에 나오는 것처럼 Greengrass 그룹 역할이 대상 훈련 작업의 `sagemaker:DescribeTrainingJob` 작업을 허용하도록 구성되었습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
```

```
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
    ]
}
```

그룹 역할 요구 사항의 경우 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해). 앞으로 대상 훈련 작업을 변경하는 경우 필요에 따라 그룹 역할을 업데이트하십시오.

- 이 커넥터와 상호 작용하려면 AWS IoT Greengrass 기계 학습 SDK v1.1.0이 필요합니다.

Version 1

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 2.7입니다.
- 코어 디바이스에 설치된 Apache MXNet 프레임워크에 대한 종속성. 자세한 내용은 the section called “MXNet 종속성 설치” 섹션을 참조하세요.
- SageMaker 모델 소스를 참조하는 Greengrass 그룹의 ML 리소스입니다. 이 모델은 SageMaker 이미지 분류 알고리즘이 학습해야 합니다. 자세한 내용은 Amazon SageMaker 개발자 안내서의 이미지 분류 알고리즘을 참조하세요.
- 다음 예제 IAM 정책에 나오는 것처럼 Greengrass 그룹 역할이 대상 훈련 작업의 `sagemaker:DescribeTrainingJob` 작업을 허용하도록 구성되었습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
```

```
    ]
}
```

그룹 역할 요구 사항의 경우 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해). 앞으로 대상 훈련 작업을 변경하는 경우 필요에 따라 그룹 역할을 업데이트하십시오.

- 이 커넥터와 상호 작용하려면 AWS IoT Greengrass 기계 학습 SDK v1.0.0 이상이 필요합니다.

커넥터 파라미터

이러한 커넥터는 다음 파라미터를 제공합니다.

Version 2

`MLModelDestinationPath`

Lambda 환경 내 ML 리소스의 절대 로컬 경로입니다. ML 리소스에 대해 지정된 대상 경로입니다.

> **Note**
> 콘솔에서 ML 리소스를 생성한 경우 이 경로는 로컬 경로입니다.

AWS IoT 콘솔의 표시 이름: 모델 대상 경로

필수: `true`

형식: `string`

유효한 패턴: `.+`

`MLModelResourceId`

소스 모델을 참조하는 ML 리소스의 ID입니다.

AWS IoT 콘솔의 표시 이름: SageMaker 작업 ARN 리소스

필수: `true`

형식: `string`

유효한 패턴: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

SageMaker 모델 소스를 나타내는 SageMaker 훈련 작업의 ARN입니다. 이 모델은 SageMaker 이미지 분류 알고리즘이 훈련 받아야 합니다.

AWS IoT 콘솔의 표시 이름: SageMaker 작업 ARN

필수: `true`

형식: `string`

유효한 패턴: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

로컬 추론 서비스의 이름입니다. 사용자 정의 Lambda 함수가 AWS IoT Greengrass 기계 학습 SDK의 `invoke_inference_service` 함수에 이름을 전달해 이 서비스를 간접 호출합니다. 예제는 the section called “사용 예”에서 확인하세요.

AWS IoT 콘솔의 표시 이름: 로컬 추론 서비스 이름

필수: `true`

형식: `string`

유효한 패턴: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

추론 요청이 종료되기 전까지의 시간(초)입니다. 최소값은 1입니다.

AWS IoT 콘솔의 표시 이름: 제한 시간(초)

필수: `true`

형식: `string`

유효한 패턴: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

서비스에서 액세스할 수 있는 메모리의 양(KB)입니다. 최소값은 1입니다.

AWS IoT 콘솔의 표시 이름: 메모리 제한(KB)

필수: `true`

형식: `string`

유효한 패턴: `[1-9][0-9]*`

`GPUAcceleration`

CPU 또는 GPU(가속) 컴퓨팅 컨텍스트. 이 속성은 ML 이미지 분류 Aarch64 JTX2 커넥터에만 적용됩니다.

AWS IoT 콘솔의 표시 이름: GPU 가속화

필수: `true`

형식: `string`

유효한 값: `CPU` 또는 `GPU`

`MLFeedbackConnectorConfigId`

모델 입력 데이터를 업로드할 때 사용할 피드백 구성의 ID입니다. ML Feedback 커넥터에 정의된 피드백 구성의 ID와 일치해야 합니다.

이 파라미터는 ML 피드백 커넥터를 사용하여 모델 입력 데이터를 업로드하고 예측을 MQTT 주제에 게시하려는 경우에만 필요합니다.

AWS IoT 콘솔의 표시 이름: ML 피드백 커넥터 구성 ID

필수: `false`

형식: `string`

유효한 패턴: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Version 1

`MLModelDestinationPath`

Lambda 환경 내 ML 리소스의 절대 로컬 경로입니다. ML 리소스에 대해 지정된 대상 경로입니다.

> **Note**
> 콘솔에서 ML 리소스를 생성한 경우 이 경로는 로컬 경로입니다.

AWS IoT 콘솔의 표시 이름: 모델 대상 경로

필수: `true`

형식: `string`

유효한 패턴: `.+`

`MLModelResourceId`

소스 모델을 참조하는 ML 리소스의 ID입니다.

AWS IoT 콘솔의 표시 이름: SageMaker 작업 ARN 리소스

필수: `true`

형식: `string`

유효한 패턴: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

SageMaker 모델 소스를 나타내는 SageMaker 훈련 작업의 ARN입니다. 이 모델은 SageMaker 이미지 분류 알고리즘이 훈련 받아야 합니다.

AWS IoT 콘솔의 표시 이름: SageMaker 작업 ARN

필수: `true`

형식: `string`

유효한 패턴: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

로컬 추론 서비스의 이름입니다. 사용자 정의 Lambda 함수가 AWS IoT Greengrass 기계 학습 SDK의 `invoke_inference_service` 함수에 이름을 전달해 이 서비스를 간접 호출합니다. 예제는 the section called “사용 예”에서 확인하세요.

AWS IoT 콘솔의 표시 이름: 로컬 추론 서비스 이름

필수: `true`

형식: `string`

유효한 패턴: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

추론 요청이 종료되기 전까지의 시간(초)입니다. 최소값은 1입니다.

AWS IoT 콘솔의 표시 이름: 제한 시간(초)

필수: `true`

형식: `string`

유효한 패턴: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

서비스에서 액세스할 수 있는 메모리의 양(KB)입니다. 최소값은 1입니다.

AWS IoT 콘솔의 표시 이름: 메모리 제한(KB)

필수: `true`

형식: `string`

유효한 패턴: `[1-9][0-9]*`

`GPUAcceleration`

CPU 또는 GPU(가속) 컴퓨팅 컨텍스트. 이 속성은 ML 이미지 분류 Aarch64 JTX2 커넥터에만 적용됩니다.

AWS IoT 콘솔의 표시 이름: GPU 가속화

필수: `true`

형식: `string`

유효한 값: `CPU` 또는 `GPU`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 ML 이미지 분류 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`를 생성합니다.

예: CPU 인스턴스

이 예제에서는 ML 이미지 분류 ARMv7l 커넥터의 인스턴스를 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationARMv7/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

예: GPU 인스턴스

이 예제에서는 ML 이미지 분류 Aarch64 JTX2 커넥터의 인스턴스를 생성하며, 이 커넥터는 NVIDIA Jetson TX2 보드에서 GPU 가속화를 지원합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationAarch64JTX2/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "GPUAcceleration": "GPU",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이러한 커넥터는 입력으로 이미지 파일을 사용합니다. 입력 이미지 파일은 `jpeg` 또는 `png` 형식이어야 합니다. 자세한 내용은 the section called “사용 예” 섹션을 참조하세요.

이러한 커넥터는 MQTT 메시지를 입력 데이터로 받아들이지 않습니다.

출력 데이터

이러한 커넥터는 입력 이미지에서 감지된 객체의 형식 지정된 예측을 반환합니다.

```
[0.3,0.1,0.04,...]
```

예측에는 모델 교육 도중 교육 데이터 세트에 사용된 범주와 일치하는 값의 목록이 포함됩니다. 모든 값은 이미지가 해당 범주에 속할 확률을 나타냅니다. 확률이 가장 높은 범주가 지배적 예측입니다.

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

사용 예

다음 예제 Lambda 함수는 AWS IoT Greengrass 기계 학습 SDK를 사용하여 ML 이미지 분류 커넥터와 상호 작용합니다.

> **Note**
>
> 사용자는 AWS IoT Greengrass 기계 학습 SDK 다운로드 페이지에서 SDK를 다운로드할 수 있습니다.

이 예제에서는 SDK 클라이언트를 초기화하고 SDK의 `invoke_inference_service` 함수를 호출해 로컬 추론 서비스를 간접 호출합니다. 그러면 알고리즘 유형, 서비스 이름, 이미지 유형 및 이미지 콘텐츠를 전달합니다. 그런 다음 이 예제에서는 서비스 응답을 구문 분석해 가능성 결과(예측)를 얻습니다.

Python 3.7

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')
```

```
    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__,
 e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')

    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()

def function_handler(event, context):
    return
```

Python 2.7

```
import logging
from threading import Timer

import numpy
```

```
import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")


def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    predictions_arr = numpy.fromstring(predictions, sep=",")
    logging.info("Split into %s predictions.", len(predictions_arr))

    # Perform business logic that relies on predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
```

```
infer()


# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return
```

AWS IoT Greengrass 기계 학습 SDK의 `invoke_inference_service` 함수는 다음 인수를 받습니다.

인수	설명
`AlgoType`	추론에 사용할 알고리즘 유형의 이름입니다. 현재 `image-classification` 만 지원됩니다. 필수: `true` 형식: `string` 유효한 값: `image-classification`
`ServiceName`	로컬 추론 서비스의 이름입니다. 커넥터를 구성할 때 `LocalInferenceServiceName` 파라미터에 대해 지정한 이름을 사용합니다. 필수: `true` 형식: `string`
`ContentType`	입력 이미지의 mime 유형입니다. 필수: `true` 형식: `string` 유효한 값: `image/jpeg, image/png`
`Body`	입력 이미지 파일의 콘텐츠입니다.

인수	설명
	필수: `true` 형식: `binary`

AWS IoT Greengrass 코어에 MXNet 종속성 설치

ML 이미지 분류 커넥터를 사용하려면 코어 디바이스에 Apache MXNet 프레임워크에 대한 종속성을 설치해야 합니다. 커넥터는 이 프레임워크를 사용해 ML 모델을 제공합니다.

> **Note**
>
> 이러한 커넥터는 미리 컴파일된 MXNet 라이브러리와 번들로 제공되므로, 코어 디바이스에 MXNet 프레임워크를 설치할 필요가 없습니다.

AWS IoT Greengrass에서는 다음 일반 플랫폼 및 디바이스에 대한 종속성 설치를 위한(또는 해당 항목 설치 시 참조용으로 사용할) 스크립트를 제공합니다. 다른 플랫폼 또는 디바이스를 사용하는 경우 MXNet 설명서로 구성을 확인하십시오.

MXNet 종속성 설치 전에 필요한 시스템 라이브러리(지정된 최소 버전)가 디바이스에 있는지 확인합니다.

NVIDIA Jetson TX2

1. CUDA Toolkit 9.0 및 cuDNN 7.0을 설치합니다. 시작하기 자습서에서 the section called “다른 디바이스 설정”의 지침을 따르십시오.
2. 커넥터가 커뮤니티에서 유지 관리하는 오픈 소프트웨어를 설치할 수 있도록 범용 리포지토리를 활성화합니다. 자세한 내용은 Ubuntu 설명서의 리포지토리/Ubuntu를 참조하십시오.

   a. `/etc/apt/sources.list` 파일을 엽니다.

   b. 다음 행의 주석 처리를 해제하십시오.

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. 다음 설치 스크립트의 사본을 코어 디바이스에 `nvidiajtx2.sh` 파일로 저장합니다.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-
dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 이 스크립트를 사용하여 OpenCV가 성공적으로 설치되지 않을 경우 소스로부터 빌드를 시도할 수 있습니다. 자세한 내용은 OpenCV 설명서의 Linux에서 설치를 참조하거나 현재 플랫폼의 다른 온라인 리소스를 참조하십시오.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
 python-dev
```

```
echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. 파일을 저장한 디렉터리에서 다음 명령을 실행합니다.

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. 다음 설치 스크립트의 사본을 코어 디바이스에 x86_64.sh 파일로 저장합니다.

   Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
```

```
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 이 스크립트를 사용하여 OpenCV가 성공적으로 설치되지 않을 경우 소스로부터 빌드를 시도할 수 있습니다. 자세한 내용은 OpenCV 설명서의 Linux에서 설치를 참조하거나 현재 플랫폼의 다른 온라인 리소스를 참조하십시오.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
 python-pip
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade
```

```
  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
pip
else
  echo "OS Release not supported: $release"
  exit 1
fi

pip install numpy==1.15.0 scipy opencv-python

echo 'Dependency installation/upgrade complete.'
```

2. 파일을 저장한 디렉터리에서 다음 명령을 실행합니다.

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. 다음 설치 스크립트의 사본을 코어 디바이스에 `armv7l.sh` 파일로 저장합니다.

   Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 이 스크립트를 사용하여 OpenCV가 성공적으로 설치되지 않을 경우 소스로부터 빌드를 시도할 수 있습니다. 자세한 내용은 OpenCV 설명서의 Linux에서 설치를 참조하거나 현재 플랫폼의 다른 온라인 리소스를 참조하십시오.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
 repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
 installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. 파일을 저장한 디렉터리에서 다음 명령을 실행합니다.

```
sudo bash armv7l.sh
```

> **Note**
>
> Raspberry Pi에서는 pip를 사용한 기계 학습 종속성 설치가 메모리 집약적 작업이므로 디바이스가 메모리 부족으로 응답하지 않게 될 수 있습니다. 차선책으로서 스왑 크기를 일시적으로 늘릴 수 있습니다.
> /etc/dphys-swapfile에서는 CONF_SWAPSIZE 변수의 값을 늘리고 다음 명령을 실행해 dphys-swapfile을 재시작합니다.
>
> ```
> /etc/init.d/dphys-swapfile restart
> ```

로깅 및 문제 해결

그룹 설정에 따라 이벤트 및 오류 로그는 CloudWatch 로그, 로컬 파일 시스템 또는 둘 모두에 기록됩니다. 이 커넥터의 로그는 접두사 LocalInferenceServiceName을 사용합니다. 커넥터가 예기치 않은 방식으로 동작할 경우 커넥터의 로그를 확인합니다. 일반적으로 이러한 로그에는 누락된 ML 라이브러리 종속성 또는 커넥터 시작 실패 원인과 같은 유용한 디버깅 정보가 포함됩니다.

AWS IoT Greengrass 그룹이 로컬로 로그를 기록하도록 구성된 경우 커넥터는 로그 파일을 *greengrass-root*/ggc/var/log/user/*region*/aws/에 기록합니다. Greengrass 로깅에 대한 자세한 내용은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요.

다음 정보를 사용하면 ML 이미지 분류 커넥터의 문제 해결에 도움이 됩니다.

필수 시스템 라이브러리

다음 탭에는 각 ML 이미지 분류 커넥터에 필요한 시스템 라이브러리가 나열되어 있습니다.

ML Image Classification Aarch64 JTX2

라이브러리	최소 버전
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	해당 사항 없음

라이브러리	최소 버전
libcudart.so.9.0	해당 사항 없음
libcudnn.so.7	해당 사항 없음
libcufft.so.9.0	해당 사항 없음
libcurand.so.9.0	해당 사항 없음
libcusolver.so.9.0	해당 사항 없음
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Image Classification x86_64

라이브러리	최소 버전
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5

라이브러리	최소 버전
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

라이브러리	최소 버전
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

문제

증상	솔루션
Raspberry Pi에서는 다음 오류 메시지가 기록되며 카메라를 사용하지 않습니다. `Failed to initialize libdc1394`	드라이버를 비활성화하려면 다음 명령을 실행합니다. `sudo ln /dev/null /dev/raw1394` 이 작업은 임시적이며 심볼 링크는 재부팅 이후 사라집니다. 재부팅 링크 자동 생성 방법을 알아보려면 OS 배포 매뉴얼을 참조하십시오.

라이선스

ML 이미지 분류 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

- Deep Neural Network Library(DNNL)/Apache 라이선스 2.0
- OpenMP* Runtime Library/Intel OpenMP Runtime Library 라이선스 참조.
- mxnet/Apache 라이선스 2.0
- six/MIT

Intel OpenMP Runtime Library 라이선스. Intel® OpenMP* 런타임은 Intel® Parallel Studio XE Suite 제품에 포함된 상용(COM) 라이선스와 BSD 오픈 소스(OSS) 라이선스 두 가지 방식으로 라이선스를 제공합니다.

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
2	ML 피드백 커넥터를 사용하여 모델 입력 데이터를 업로드하고, 예측을 MQTT 주제에 게시하고, 지표를 Amazon CloudWatch에 게시할 수 있도록 지원하는 `MLFeedbackConnectorConfigId` 파라미터를 추가했습니다.

버전	변경
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 섹션을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- 기계 학습 추론 수행
- Amazon SageMaker 개발자 안내서의 이미지 분류 알고리즘

ML 객체 감지 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

ML 객체 감지 커넥터는 AWS IoT Greengrass 코어에서 실행되는 ML(기계 학습) 추론 서비스를 제공합니다. 이 로컬 추론 서비스 플랫폼은 SageMaker Neo 딥 러닝 컴파일러에 의해 컴파일된 객체 감지 모델을 사용하여 객체 감지를 수행합니다. 두 가지 객체 감지 모델이 지원됩니다. SSD(Single Shot Multibox Detector) 및 YOLO(You Only Look Once) v3. 자세한 내용은 객체 감지 모델 요구 사항을 참조하십시오.

사용자 정의 Lambda 함수는 AWS IoT Greengrass 기계 학습 SDK를 사용하여 로컬 추론 서비스에 추론 요청을 제출합니다. 이 서비스는 입력 이미지에 대해 로컬 추론을 수행하고 이미지에서 감지된 각 객체에 대한 예측의 목록을 반환합니다. 각 예측에는 객체 범주, 예측 신뢰도 점수, 예측된 객체 주위의 경계 상자를 지정하는 픽셀 좌표가 포함됩니다.

AWS IoT Greengrass는 여러 플랫폼을 위한 ML 객체 감지 커넥터를 제공합니다.

커넥터	설명 및 ARN
ML 객체 감지 Arch64 JTX2	NVIDIA Jetson TX2용 객체 감지 추론 서비스입니다. GPU 가속화를 지원합니다. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionAarch64JTX2/versions/1`
ML 객체 감지 x86_64	x86_64 플랫폼용 객체 감지 추론 서비스입니다. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionx86-64/versions/1`
ML 객체 감지 ARMv7	ARMv7 플랫폼용 객체 감지 추론 서비스입니다. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionARMv7/versions/1`

요구 사항

이러한 커넥터에는 다음 요구 사항이 있습니다.

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 코어 디바이스에 설치된 SageMaker Neo 딥러닝 런타임에 대한 종속성. 자세한 내용은 the section called “Neo 딥 러닝 런타임 종속성 설치” 섹션을 참조하세요.
- Greengrass 그룹의 ML 리소스입니다. ML 리소스는 객체 감지 모델이 있는 Amazon S3 버킷을 참조해야 합니다. 자세한 내용은 Amazon S3 모델 소스를 참조하세요.

> **Note**
>
> 모델은 Single Shot Multibox Detector 또는 You Only Look Once v3 객체 감지 모델 유형이어야 합니다. SageMaker Neo 딥 러닝 컴파일러를 사용하여 컴파일해야 합니다. 자세한 내용은 객체 감지 모델 요구 사항을 참조하십시오.

- ML 피드백 커넥터를 Greengrass 그룹에 추가 및 구성해야 합니다. 이 파라미터는 이 커넥터를 사용하여 모델 입력 데이터를 업로드하고 예측을 MQTT 주제에 게시하려는 경우에만 필요합니다.
- 이 커넥터와 상호 작용하려면 AWS IoT Greengrass 기계 학습 SDK v1.1.0이 필요합니다.

객체 감지 모델 요구 사항

ML 객체 감지 커넥터는 SSD(Single Shot Multibox Detector) 및 YOLO(You Only Look Once) v3 객체 감지 모델 유형을 지원합니다. GluonCV에서 제공하는 객체 감지 구성 요소를 사용하여 자체 데이터 세트로 모델을 교육시킬 수 있습니다. 또는 GluonCV Model Zoo에서 사전 교육된 모델을 사용할 수 있습니다.

- 사전 교육된 SSD 모델
- 사전 교육된 YOLO v3 모델

객체 감지 모델은 512 x 512 입력 이미지로 교육해야 합니다. GluonCV Model Zoo의 사전 교육된 모델은 이미 이 요구 사항을 충족합니다.

훈련된 객체 감지 모델은 SageMaker Neo 딥 러닝 컴파일러를 사용하여 컴파일해야 합니다. 컴파일 시 대상 하드웨어가 Greengrass 코어 디바이스의 하드웨어와 일치해야 합니다. 자세한 내용은 Amazon SageMaker 개발자 안내서의 SageMaker Neo를 참조하세요.

컴파일된 모델은 커넥터와 동일한 Greengrass 그룹에 ML 리소스(Amazon S3 모델 리소스)로 추가되어야 합니다.

커넥터 파라미터

이러한 커넥터는 다음 파라미터를 제공합니다.

`MLModelDestinationPath`

Neo 호환 ML 모델이 들어 있는 Amazon S3 버킷의 절대 경로입니다. ML 모델 리소스에 대해 지정된 대상 경로입니다.

AWS IoT 콘솔의 표시 이름: 모델 대상 경로

필수: `true`

형식: `string`

유효한 패턴: `.+`

`MLModelResourceId`

소스 모델을 참조하는 ML 리소스의 ID입니다.

AWS IoT 콘솔의 표시 이름: Greengrass 그룹 ML 리소스

필수: `true`

형식: `S3MachineLearningModelResource`

유효한 패턴: `^[a-zA-Z0-9:_-]+$`

`LocalInferenceServiceName`

로컬 추론 서비스의 이름입니다. 사용자 정의 Lambda 함수가 AWS IoT Greengrass 기계 학습 SDK의 `invoke_inference_service` 함수에 이름을 전달해 이 서비스를 간접 호출합니다. 예제는 the section called “사용 예”에서 확인하세요.

AWS IoT 콘솔의 표시 이름: 로컬 추론 서비스 이름

필수: `true`

형식: `string`

유효한 패턴: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

`LocalInferenceServiceTimeoutSeconds`

추론 요청이 종료되기 전까지의 시간(초)입니다. 최소값은 1입니다. 기본값은 10입니다.

AWS IoT 콘솔의 표시 이름: 제한 시간(초)

필수: `true`

형식: `string`

유효한 패턴: `^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

서비스에서 액세스할 수 있는 메모리의 양(KB)입니다. 최소값은 1입니다.

AWS IoT 콘솔의 표시 이름: 메모리 제한

필수: `true`

형식: `string`

유효한 패턴: `^[1-9][0-9]*$`

`GPUAcceleration`

CPU 또는 GPU(가속) 컴퓨팅 컨텍스트. 이 속성은 ML 이미지 분류 Aarch64 JTX2 커넥터에만 적용됩니다.

AWS IoT 콘솔의 표시 이름: GPU 가속화

필수: `true`

형식: `string`

유효한 값: `CPU` 또는 `GPU`

`MLFeedbackConnectorConfigId`

모델 입력 데이터를 업로드할 때 사용할 피드백 구성의 ID입니다. ML 피드백 커넥터에 정의된 피드백 구성의 ID와 일치해야 합니다.

이 파라미터는 ML 피드백 커넥터를 사용하여 모델 입력 데이터를 업로드하고 예측을 MQTT 주제에 게시하려는 경우에만 필요합니다.

AWS IoT 콘솔의 표시 이름: ML 피드백 커넥터 구성 ID

필수: `false`

형식: `string`

유효한 패턴: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 ML 객체 감지 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`를 생성합니다. 이 예제에서는 ML 객체 감지 ARMv7l 커넥터의 인스턴스를 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyObjectDetectionConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ObjectDetectionARMv7/versions/1",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "LocalInferenceServiceName": "objectDetection",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이러한 커넥터는 입력으로 이미지 파일을 사용합니다. 입력 이미지 파일은 `jpeg` 또는 `png` 형식이어야 합니다. 자세한 내용은 the section called “사용 예” 섹션을 참조하세요.

이러한 커넥터는 MQTT 메시지를 입력 데이터로 받아들이지 않습니다.

출력 데이터

이러한 커넥터는 입력 이미지에서 식별된 객체에 대한 예측 결과의 형식 지정된 목록을 반환합니다.

```
{
    "prediction": [
        [
            14,
            0.9384938478469849,
            0.37763649225234985,
            0.5110225081443787,
            0.6697432398796082,
            0.8544386029243469
        ],
        [
            14,
            0.8859519958496094,
            0,
            0.43536216020584106,
            0.3314110040664673,
            0.9538808465003967
        ],
        [
            12,
            0.04128098487854004,
            0.5976729989051819,
            0.5747185945510864,
            0.704264223575592,
            0.857937216758728
        ],
        ...
    ]
}
```

목록의 각 예측은 대괄호로 묶여 있으며 6개 값을 포함합니다.

- 첫 번째 값은 식별된 객체의 예측된 객체 범주를 나타냅니다. 객체 범주 및 해당 값은 Neo 딥 러닝 컴파일러에서 객체 감지 기계 학습 모델을 훈련할 때 결정됩니다.
- 두 번째 값은 객체 범주 예측의 신뢰도 점수입니다. 이 값은 예측이 정확할 확률을 나타냅니다.
- 마지막 4개 값은 이미지에서 예측된 객체 주위의 경계 상자를 표현하는 픽셀 치수에 해당합니다.

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

사용 예

다음 예제 Lambda 함수는 AWS IoT Greengrass 기계 학습 SDK를 사용하여 ML 객체 감지 커넥터와 상호 작용합니다.

> **Note**
>
> 사용자는 AWS IoT Greengrass 기계 학습 SDK 다운로드 페이지에서 SDK를 다운로드할 수 있습니다.

이 예제에서는 SDK 클라이언트를 초기화하고 SDK의 `invoke_inference_service` 함수를 호출해 로컬 추론 서비스를 간접 호출합니다. 그러면 알고리즘 유형, 서비스 이름, 이미지 유형 및 이미지 콘텐츠를 전달합니다. 그런 다음 이 예제에서는 서비스 응답을 구문 분석해 가능성 결과(예측)를 얻습니다.

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
```

```
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__, e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))
    predictions = eval(predictions)

    # Perform business logic that relies on the predictions.

    # Schedule the infer() function to run again in ten second.
    Timer(10, infer).start()
    return

infer()

def function_handler(event, context):
    return
```

AWS IoT Greengrass 기계 학습 SDK의 `invoke_inference_service` 함수는 다음 인수를 수락합니다.

인수	설명
`AlgoType`	추론에 사용할 알고리즘 유형의 이름입니다. 현재 `object-detection` 만 지원됩니다. 필수: `true` 형식: `string` 유효한 값: `object-detection`
`ServiceName`	로컬 추론 서비스의 이름입니다. 커넥터를 구성할 때 `LocalInferenceServiceName` 파라미터에 대해 지정한 이름을 사용합니다. 필수: `true`

인수	설명
	형식: `string`
`ContentType`	입력 이미지의 mime 유형입니다. 필수: `true` 형식: `string` 유효한 값: `image/jpeg, image/png`
`Body`	입력 이미지 파일의 콘텐츠입니다. 필수: `true` 형식: `binary`

AWS IoT Greengrass 코어에 Neo 딥 러닝 런타임 종속성 설치

ML 객체 감지 커넥터는 SageMaker Neo 딥 러닝 런타임(DLR)과 함께 번들로 제공됩니다. 커넥터는 이 런타임을 사용해 ML 모델을 제공합니다. 이들 커넥터를 사용하려면 코어 디바이스에 DLR 종속성을 설치해야 합니다.

DLR 종속성을 설치하기 전에 필요한 시스템 라이브러리(지정된 최소 버전)가 디바이스에 있는지 확인합니다.

NVIDIA Jetson TX2

1. CUDA Toolkit 9.0 및 cuDNN 7.0을 설치합니다. 시작하기 자습서에서 the section called “다른 디바이스 설정”의 지침을 따르십시오.
2. 커넥터가 커뮤니티에서 유지 관리하는 오픈 소프트웨어를 설치할 수 있도록 범용 리포지토리를 활성화합니다. 자세한 내용은 Ubuntu 설명서의 리포지토리/Ubuntu를 참조하십시오.

   a. `/etc/apt/sources.list` 파일을 엽니다.

   b. 다음 행의 주석 처리를 해제하십시오.

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

```
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. 다음 설치 스크립트의 사본을 코어 디바이스에 `nvidiajtx2.sh` 파일로 저장합니다.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 이 스크립트를 사용하여 OpenCV가 성공적으로 설치되지 않을 경우 소스로부터 빌드를 시도할 수 있습니다. 자세한 내용은 OpenCV 설명서의 Linux에서 설치를 참조하거나 현재 플랫폼의 다른 온라인 리소스를 참조하십시오.

4. 파일을 저장한 디렉터리에서 다음 명령을 실행합니다.

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. 다음 설치 스크립트의 사본을 코어 디바이스에 `x86_64.sh` 파일로 저장합니다.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
```

```
release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 이 스크립트를 사용하여 OpenCV가 성공적으로 설치되지 않을 경우 소스로부터 빌드를 시도할 수 있습니다. 자세한 내용은 OpenCV 설명서의 Linux에서 설치를 참조하거나 현재 플랫폼의 다른 온라인 리소스를 참조하십시오.

2. 파일을 저장한 디렉터리에서 다음 명령을 실행합니다.

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. 다음 설치 스크립트의 사본을 코어 디바이스에 `armv7l.sh` 파일로 저장합니다.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 이 스크립트를 사용하여 OpenCV가 성공적으로 설치되지 않을 경우 소스로부터 빌드를 시도할 수 있습니다. 자세한 내용은 OpenCV 설명서의 Linux에서 설치를 참조하거나 현재 플랫폼의 다른 온라인 리소스를 참조하십시오.

2. 파일을 저장한 디렉터리에서 다음 명령을 실행합니다.

```
sudo bash armv7l.sh
```

> **Note**
>
> Raspberry Pi에서는 `pip`를 사용한 기계 학습 종속성 설치가 메모리 집약적 작업이므로 디바이스가 메모리 부족으로 응답하지 않게 될 수 있습니다. 차선책으로서 스왑 크기를 일시적으로 늘릴 수 있습니다. `/etc/dphys-swapfile`에서는 `CONF_SWAPSIZE` 변수의 값을 늘리고 다음 명령을 실행해 `dphys-swapfile`을 재시작합니다.

```
/etc/init.d/dphys-swapfile restart
```

로깅 및 문제 해결

그룹 설정에 따라 이벤트 및 오류 로그는 CloudWatch 로그, 로컬 파일 시스템 또는 둘 모두에 기록됩니다. 이 커넥터의 로그는 접두사 `LocalInferenceServiceName`을 사용합니다. 커넥터가 예기치 않은 방식으로 동작할 경우 커넥터의 로그를 확인합니다. 일반적으로 이러한 로그에는 누락된 ML 라이브러리 종속성 또는 커넥터 시작 실패 원인과 같은 유용한 디버깅 정보가 포함됩니다.

AWS IoT Greengrass 그룹이 로컬로 로그를 기록하도록 구성된 경우 커넥터는 로그 파일을 *greengrass-root*/ggc/var/log/user/*region*/aws/에 기록합니다. Greengrass 로깅에 대한 자세한 내용은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링” 섹션을 참조하세요.

다음 정보를 사용하면 ML 객체 감지 커넥터의 문제 해결에 도움이 됩니다.

필수 시스템 라이브러리

다음 탭에는 각 ML 객체 감지 커넥터에 필요한 시스템 라이브러리가 나열되어 있습니다.

ML Object Detection Aarch64 JTX2

라이브러리	최소 버전
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	해당 사항 없음
libcudart.so.9.0	해당 사항 없음
libcudnn.so.7	해당 사항 없음
libcufft.so.9.0	해당 사항 없음
libcurand.so.9.0	해당 사항 없음
libcusolver.so.9.0	해당 사항 없음

라이브러리	최소 버전
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libnvinfer.so.4	해당 사항 없음
libnvrm_gpu.so	해당 사항 없음
libnvrm.so	해당 사항 없음
libnvidia-fatbinaryloader.so.28.2.1	해당 사항 없음
libnvos.so	해당 사항 없음
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Object Detection x86_64

라이브러리	최소 버전
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5

라이브러리	최소 버전
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

라이브러리	최소 버전
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

문제

증상	솔루션
Raspberry Pi에서는 다음 오류 메시지가 기록되며 카메라를 사용하지 않습니다. `Failed to initialize libdc1394`	드라이버를 비활성화하려면 다음 명령을 실행합니다. `sudo ln /dev/null /dev/raw1394` 이 작업은 임시적입니다. 이 심볼 링크는 재부팅 후 사라집니다. 재부팅 링크 자동 생성 방법을 알아보려면 OS 배포 매뉴얼을 참조하십시오.

라이선스

ML 객체 감지 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

- Deep Learning Runtime/Apache 라이선스 2.0
- six/MIT

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- 기계 학습 추론 수행
- Amazon SageMaker 개발자 안내서의 객체 감지 알고리즘

Modbus-RTU 프로토콜 어댑터 커넥터

Modbus-RTU Protocol Adapter 커넥터는 AWS IoT Greengrass 그룹에 있는 Modbus RTU 디바이스에서 정보를 폴링합니다.

이 커넥터는 사용자 정의 Lambda 함수로부터 Modbus RTU 요청에 대한 파라미터를 수신합니다. 해당 요청을 전송한 다음 대상 디바이스의 응답을 MQTT 메시지로 게시합니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

  > **Note**
  >
  > Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- AWS IoT Greengrass 코어와 Modbus 디바이스 간의 물리적 연결입니다. 코어는 직렬 포트(예: USB 포트)를 통해 Modbus RTU 네트워크에 물리적으로 연결해야 합니다.
- 물리적 Modbus 직렬 포트를 가리키는 Greengrass 그룹의 로컬 디바이스 리소스입니다.

- Modbus RTU 요청 파라미터를 이 커넥터에 전송하는 사용자 정의 Lambda 함수입니다. 요청 파라미터는 예상 패턴을 따라야 하며 Modbus RTU 네트워크에 있는 대상 디바이스의 ID와 주소를 포함해야 합니다. 자세한 내용은 the section called “입력 데이터” 섹션을 참조하세요.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- AWS IoT Greengrass와 Modbus 장치 간의 물리적 연결입니다. 코어는 직렬 포트(예: USB 포트)를 통해 Modbus RTU 네트워크에 물리적으로 연결해야 합니다.
- 물리적 Modbus 직렬 포트를 가리키는 Greengrass 그룹의 로컬 디바이스 리소스입니다.
- Modbus RTU 요청 파라미터를 이 커넥터에 전송하는 사용자 정의 Lambda 함수입니다. 요청 파라미터는 예상 패턴을 따라야 하며 Modbus RTU 네트워크에 있는 대상 디바이스의 ID와 주소를 포함해야 합니다. 자세한 내용은 the section called “입력 데이터” 섹션을 참조하세요.

커넥터 파라미터

이 커넥터는 다음 파라미터를 지원합니다.

`ModbusSerialPort-ResourceId`

물리적 Modbus 직렬 포트를 나타내는 로컬 디바이스 리소스의 ID입니다.

> **Note**
>
> 이 커넥터에는 리소스에 대한 쓰기 전용 액세스 권한이 부여됩니다.

AWS IoT 콘솔의 표시 이름: Modbus 직렬 포트 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

`ModbusSerialPort`

디바이스에 있는 물리적 Modbus 직렬 포트의 절대 경로입니다. Modbus 로컬 디바이스 리소스에 대해 지정된 소스 경로입니다.

AWS IoT 콘솔의 표시 이름: Modbus 직렬 포트 리소스의 소스 경로

필수: `true`

형식: `string`

유효한 패턴: `.+`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Modbus-RTU Protocol Adapter 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyModbusRTUProtocolAdapterConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ModbusRTUProtocolAdapter/versions/3",
            "Parameters": {
                "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",
                "ModbusSerialPort": "/path-to-port"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

> **Note**
>
> Modbus-RTU Protocol Adapter 커넥터를 배포한 후 AWS IoT Things Graph를 사용하여 그룹의 디바이스 간의 상호 작용을 조정할 수 있습니다. 자세한 내용을 알아보려면 AWS IoT Things Graph사용 설명서의 Modbus를 참조하세요.

입력 데이터

이 커넥터는 MQTT 주제에 대한 사용자 정의 Lambda 함수로부터 Modbus RTU 요청 파라미터를 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

구독의 주제 필터

```
modbus/adapter/request
```

메시지 속성

요청 메시지는 메시지가 나타내는 Modbus RTU 요청 유형에 따라 다릅니다. 다음은 모든 요청에 필수적인 속성입니다.

- `request` 객체에서:
  - `operation`. 실행할 작업의 이름입니다. 예를 들어, 코일을 읽도록 `"operation": "ReadCoilsRequest"`를 지정합니다. 이 값은 유니코드 문자열이어야 합니다. 지원되는 작업 목록은 the section called “Modbus RTU 요청 및 응답” 단원을 참조하십시오.
  - `device`. 요청의 대상 디바이스입니다. 이 값은 `0 - 247` 범위여야 합니다.
- `id` 속성입니다. 요청의 ID입니다. 이 값은 데이터 중복 제거에 사용되며 오류 응답을 포함하여 모든 응답의 `id` 속성에 있는 그대로 반환됩니다. 이 값은 유니코드 문자열이어야 합니다.

> **Note**
>
> 요청에 주소 필드가 포함된 경우 값을 정수로 지정해야 합니다. 예: `"address": 1`.

요청에 포함할 다른 파라미터는 작업에 따라 다릅니다. 따로 처리되는 CRC를 제외하고 모든 요청 파라미터가 필수입니다. 예를 보려면 the section called “예제 요청 및 응답” 섹션을 참조하세요.

입력 예: 코일 요청 읽기

```
{
    "request": {
```

```
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

출력 데이터

이 커넥터는 수신 Modbus RTU 요청에 대한 응답을 게시합니다.

구독의 주제 필터

```
modbus/adapter/response
```

메시지 속성

응답 메시지의 형식은 해당 요청 및 응답 상태에 따라 다릅니다. 예를 보려면 the section called “예제 요청 및 응답” 섹션을 참조하세요.

> **Note**
>
> 쓰기 작업에 대한 응답은 단순히 요청의 에코일 뿐입니다. 쓰기 응답에 대해 의미 있는 정보가 반환되지 않더라도 응답의 상태를 확인하는 것이 좋습니다.

모든 응답에는 다음 속성이 포함됩니다.

- `response` 객체에서:
  - `status`. 요청 상태입니다. 상태는 다음 값 중 하나일 수 있습니다.
    - `Success`. 요청이 올바르고, Modbus RTU 네트워크에 전송되었으며, 응답이 반환되었습니다.
    - `Exception`. 요청이 올바르고, Modbus RTU 네트워크에 전송되었으며, 예외 응답이 반환되었습니다. 자세한 내용은 the section called “응답 상태: 예외” 섹션을 참조하세요.
    - `No  Response`. 요청이 잘못되었고, 요청이 Modbus RTU 네트워크를 통해 전송되기 전에 커넥터에서 오류가 발견되었습니다. 자세한 내용은 the section called “응답 상태: 응답 없음” 섹션을 참조하세요.
  - `device`. 요청이 전송된 디바이스입니다.

  - `operation`. 전송된 요청 유형입니다.
  - `payload`. 반환된 응답 콘텐츠입니다. `status`가 `No Response`인 경우 이 객체에는 오류 설명이 있는 `error` 속성만 포함됩니다(예: `"error": "[Input/Output] No Response received from the remote unit"`).
- `id` 속성입니다. 데이터 중복 제거에 사용되는 요청의 ID입니다.

출력 예: 성공

```
{
    "response" : {
        "status" : "success",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

출력 예: 실패

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Internal Error",
        "error": "Exception",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 129,
         "exception_code": 2
     }
     },
     "id" : "TestRequest"
}
```

더 많은 예제는 the section called “예제 요청 및 응답”를 참조하세요.

Modbus RTU 요청 및 응답

이 커넥터는 Modbus RTU 요청 파라미터를 입력 데이터로 수락하고 응답을 출력 데이터로 게시합니다.

지원되는 일반적인 작업은 다음과 같습니다.

요청 시 작업 이름	응답의 함수 코드
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

예제 요청 및 응답

다음은 지원되는 작업에 대한 예제 요청 및 응답입니다.

코일 읽기

요청 예:

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
```

```
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

개별 입력 읽기

요청 예:

```
{
    "request": {
        "operation": "ReadDiscreteInputsRequest",
        "device": 1,
        "address": 1,
        "count": 1
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
        "operation": "ReadDiscreteInputsRequest",
        "payload": {
```

```
            "function_code": 2,
            "bits": [1]
        }
     },
     "id" : "TestRequest"
}
```

보류 레지스터 읽기

요청 예:

```
{
    "request": {
        "operation": "ReadHoldingRegistersRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadHoldingRegistersRequest",
     "payload": {
         "function_code": 3,
            "registers": [20,30]
     }
     },
     "id" : "TestRequest"
}
```

입력 레지스터 읽기

요청 예:

```
{
    "request": {
        "operation": "ReadInputRegistersRequest",
```

```
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

단일 코일 쓰기

요청 예:

```
{
    "request": {
        "operation": "WriteSingleCoilRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteSingleCoilRequest",
     "payload": {
         "function_code": 5,
         "address": 1,
         "value": true
     }
     },
     "id" : "TestRequest"
```

단일 레지스터 쓰기

요청 예:

```
{
    "request": {
        "operation": "WriteSingleRegisterRequest",
```

```
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

다중 코일 쓰기

요청 예:

```
{
    "request": {
        "operation": "WriteMultipleCoilsRequest",
     "device": 1,
     "address": 1,
     "values": [1,0,0,1]
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteMultipleCoilsRequest",
     "payload": {
         "function_code": 15,
         "address": 1,
         "count": 4
     }
     },
     "id" : "TestRequest"
}
```

다중 레지스터 쓰기

요청 예:

```
{
    "request": {
```

```
        "operation": "WriteMultipleRegistersRequest",
     "device": 1,
     "address": 1,
     "values": [20,30,10]
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteMultipleRegistersRequest",
     "payload": {
         "function_code": 23,
         "address": 1,
         "count": 3
     }
     },
     "id" : "TestRequest"
}
```

레지스터 쓰기 마스크

요청 예:

```
{
    "request": {
        "operation": "MaskWriteRegisterRequest",
     "device": 1,
     "address": 1,
        "and_mask": 175,
        "or_mask": 1
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
```

```
        "status": "success",
        "device": 1,
     "operation": "MaskWriteRegisterRequest",
     "payload": {
         "function_code": 22,
            "and_mask": 0,
            "or_mask": 8
     }
     },
     "id" : "TestRequest"
}
```

다중 레지스터 쓰기 읽기

요청 예:

```
{
    "request": {
        "operation": "ReadWriteMultipleRegistersRequest",
     "device": 1,
     "read_address": 1,
        "read_count": 2,
        "write_address": 3,
        "write_registers": [20,30,40]
    },
    "id": "TestRequest"
}
```

응답 예:

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadWriteMultipleRegistersRequest",
     "payload": {
         "function_code": 23,
         "registers": [10,20,10,20]
     }
     },
     "id" : "TestRequest"
}
```

> **Note**
>
> 이 응답에서 반환된 레지스터는 읽은 레지스터입니다.

응답 상태: 예외

요청 형식이 올바르지만 요청이 성공적으로 완료되지 않으면 예외가 발생합니다. 이 경우 응답에는 다음 정보가 포함됩니다.

- `status`는 `Exception`으로 설정됩니다.
- `function_code`는 요청의 코드 함수 + 128과 같습니다.
- `exception_code`에는 예외 코드가 포함되어 있습니다. 자세한 내용은 Modbus 예외 코드를 참조하십시오.

예:

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Internal Error",
        "error": "Exception",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 129,
         "exception_code": 2
     }
     },
     "id" : "TestRequest"
}
```

응답 상태: 응답 없음

이 커넥터는 Modbus 요청에 대해 확인 점검을 수행합니다. 예를 들어 잘못된 형식과 누락된 필드가 있는지 점검합니다. 확인이 실패하면 커넥터는 요청을 전송하지 않습니다. 그 대신 다음 정보가 포함된 응답을 반환합니다.

- `status`는 `No Response`으로 설정됩니다.

- `error`에는 오류 이유가 포함되어 있습니다.
- `error_message`에는 오류 메시지가 포함되어 있습니다.

예제:

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Invalid address field. Expected <type 'int'>, got <type
 'str'>",
        "error": "No Response",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "error": "Invalid address field. Expected <type 'int'>, got <type 'str'>"
     }
     },
     "id" : "TestRequest"
}
```

요청이 존재하지 않는 디바이스를 대상으로 하거나 Modbus RTU 네트워크가 작동하지 않는 경우 응답 없음 형식을 사용하는 `ModbusIOException`가 발생할 수 있습니다.

```
{
    "response" : {
        "status" : "fail",
        "error_message": "[Input/Output] No Response received from the remote unit",
        "error": "No Response",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "error": "[Input/Output] No Response received from the remote unit"
     }
     },
     "id" : "TestRequest"
}
```

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 필요한 로컬 디바이스 리소스를 추가하고 Lambda 함수에 대한 읽기/쓰기 액세스 권한을 부여합니다.

   c. 커넥터를 추가하고 해당 파라미터를 구성합니다.

   d. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.
4. 그룹을 배포합니다.
5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
 request = {
  "request": {
   "operation": "ReadCoilsRequest",
   "device": 1,
   "address": 1,
   "count": 1
  },
  "id": "TestRequest"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_read_coils_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

라이선스

Modbus-RTU Protocol Adapter 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- pymodbus/BSD
- pyserial/BSD

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	AWS 리전 지원용 커넥터 ARN이 업데이트됨. 오류 로깅을 개선함.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

Modbus-TCP 프로토콜 어댑터 커넥터

Modbus-TCP 프로토콜 어댑터 커넥터는 ModbusTCP 프로토콜을 통해 로컬 디바이스에서 데이터를 수집하고, 이를 선택한 `StreamManager` 스트림에 게시합니다.

또한 이 커넥터를 IoT SiteWise 커넥터 및 IoT SiteWise 게이트웨이와 함께 사용할 수 있습니다. 게이트웨이는 커넥터에 대한 구성을 제공해야 합니다. 자세한 내용은 IoT SiteWise 사용 설명서의 Modbus TCP 소스 구성을 참조하십시오.

> **Note**
>
> 이 커넥터는 컨테이너 없음 격리 모드에서 실행되므로, Docker 컨테이너에서 실행되는 AWS IoT Greengrass 그룹에 이 커넥터를 배포할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 1 - 3

- AWS IoT Greengrass 코어 소프트웨어 v1.10.2 이상.
- AWS IoT Greengrass 그룹에 스트림 관리자가 활성화되어 있습니다.
- 코어 디바이스에 설치되고 `PATH` 환경 변수에 추가된 Java 8입니다.

> **Note**
>
> 커넥터는 다음 리전에서만 사용할 수 있습니다.
>
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1

- us-west-2
- cn-north-1

커넥터 파라미터

이 커넥터는 다음 파라미터를 지원합니다.

`LocalStoragePath`

IoT SiteWise 커넥터가 영구 데이터를 기록할 수 있는 AWS IoT Greengrass 호스트의 디렉터리입니다. 기본 디렉터리는 `/var/sitewise`입니다.

AWS IoT 콘솔의 표시 이름: 로컬 스토리지 경로

필수: `false`

형식: `string`

유효한 패턴: `^\s*$|\/.`

`MaximumBufferSize`

IoT SiteWise 디스크 사용량을 위한 최대 크기(GB) 입니다. 기본 크기는 10GB입니다.

AWS IoT 콘솔의 표시 이름: 최대 디스크 버퍼 크기

필수: `false`

형식: `string`

유효한 패턴: `^\s*$|[0-9]+`

`CapabilityConfiguration`

커넥터가 데이터를 수집하고 연결하는 Modbus TCP 컬렉터 구성 세트.

AWS IoT 콘솔의 표시 이름: CapabilityConfiguration

필수: `false`

유형: 지원되는 피드백 구성 세트를 정의하는 올바른 형식의 JSON 문자열입니다.

다음은 CapabilityConfiguration의 예제입니다.

```
{
    "sources": [
        {
            "type": "ModBusTCPSource",
            "name": "SourceName1",
            "measurementDataStreamPrefix": "SourceName1_Prefix",
            "destination": {
                "type": "StreamManager",
                "streamName": "SiteWise_Stream_1",
                "streamBufferSize": 8
            },
            "endpoint": {
                "ipAddress": "127.0.0.1",
                "port": 8081,
                "unitId": 1
            },
            "propertyGroups": [
                {
                    "name": "GroupName",
                    "tagPathDefinitions": [
                        {
                            "type": "ModBusTCPAddress",
                            "tag": "TT-001",
                            "address": "30001",
                            "size": 2,
                            "srcDataType": "float",
                            "transformation": "byteWordSwap",
                            "dstDataType": "double"
                        }
                    ],
                    "scanMode": {
                        "type": "POLL",
                        "rate": 100
                    }
                }
            ]
        }
    ]
}
```

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Modbus-TCP Protocol Adapter 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
    "Connectors": [
        {
            "Id": "MyModbusTCPConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
            "Parameters": {
                "capability_configuration": "{\"version\":1,\"namespace\":
\"iotsitewise:modbuscollector:1\",\"configuration\":\"{\"sources\":[{\"type
\":\"ModBusTCPSource\",\"name\":\"SourceName1\",\"measurementDataStreamPrefix
\":\"\",\"endpoint\":{\"ipAddress\":\"127.0.0.1\",\"port\":8081,\"unitId\":1},
\"propertyGroups\":[{\"name\":\"PropertyGroupName\",\"tagPathDefinitions\":[{\"type
\":\"ModBusTCPAddress\",\"tag\":\"TT-001\",\"address\":\"30001\",\"size\":2,
\"srcDataType\":\"hexdump\",\"transformation\":\"noSwap\",\"dstDataType\":\"string
\"}],\"scanMode\":{\"rate\":200,\"type\":\"POLL\"}}],\"destination\":{\"type\":
\"StreamManager\",\"streamName\":\"SiteWise_Stream\",\"streamBufferSize\":10},
\"minimumInterRequestDuration\":200}]}\"}"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

입력 데이터

이 커넥터는 MQTT 메시지를 출력 데이터로 게시하지 않습니다.

출력 데이터

이 커넥터는 `StreamManager`에 데이터를 게시합니다. 대상 메시지 스트림을 구성해야 합니다. 출력 메시지는 다음 구조를 취합니다.

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

라이선스

Modbus-TCP Protocol Adapter 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- Digital Petri Modbus

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경	날짜
3(권장)	이 버전에는 버그 수정이 포함되어 있습니다.	2022년 12월 22일
2	ASCII, UTF8 및 ISO8859 인코딩 소스 문자열에 대한 지원이 추가되었습니다.	2021년 5월 24일
1	최초 릴리스.	2020년 12월 15일

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

Raspberry Pi GPIO 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

Raspberry Pi GPIO 커넥터 는 Raspberry Pi 코어 디바이스에서 범용 입력/출력(GPIO) 핀을 제어합니다.

이 커넥터는 입력 핀을 지정된 간격으로 폴링하고 상태 변경을 MQTT 주제에 게시합니다. 또한 사용자 정의 Lambda 함수의 읽기 및 쓰기 요청을 MQTT 메시지로 수락합니다. 쓰기 요청은 핀을 높은 또는 낮은 전압으로 유지하는 데 사용됩니다.

이 커넥터는 입력 및 출력 핀을 지정하는 데 사용하는 파라미터를 제공합니다. 이러한 동작은 그룹 배포 전에 구성합니다. 런타임 시 변경할 수 없습니다.

- 입력 핀은 주변 디바이스에서 데이터를 수신하는 데 사용할 수 있습니다.
- 출력 핀은 주변 장치를 제어하거나 주변 장치로 데이터를 보내는데 사용할 수 있습니다.

다음과 같은 여러 시나리오에 이 커넥터를 사용할 수 있습니다.

- 신호등으로 녹색, 노란색 및 빨간색 LED 조명을 제어합니다.
- 습도 센서의 데이터를 기반으로 팬(전기 릴레이에 연결됨)을 제어합니다.
- 고객이 버튼을 누르면 소매점 직원에게 알립니다.
- 스마트 전등 스위치를 사용하여 다른 IoT 디바이스를 제어합니다.

> **Note**
>
> 이 커넥터는 실시간 요구 사항이 있는 애플리케이션에 적합하지 않습니다. 기간이 짧은 이벤트가 누락되었을 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
3	`arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/versions/3`
2	`arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/versions/2`
1	`arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Raspberry Pi 4 Model B 또는 Raspberry Pi 3 Model B/B+ Raspberry Pi의 핀 순서를 알고 있어야 합니다. 자세한 내용은 the section called “GPIO 핀 순서” 섹션을 참조하세요.
- Raspberry Pi의 `/dev/gpiomem`을 가리키는 Greengrass 그룹의 로컬 디바이스 리소스. 콘솔에서 리소스를 생성하는 경우 리소스를 소유한 Linux 그룹의 OS 그룹 권한 자동 추가 옵션을 선택

해야 합니다. API에서 `GroupOwnerSetting.AutoAddGroupOwner` 속성을 `true`(으)로 설정합니다.

- Raspberry Pi에 설치된 RPi.GPIO 모듈. Raspbian에서 이 모듈은 기본적으로 설치되어 있습니다. 다음 명령을 사용해 다시 설치할 수 있습니다.

```
sudo pip install RPi.GPIO
```

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Raspberry Pi 4 Model B 또는 Raspberry Pi 3 Model B/B+ Raspberry Pi의 핀 순서를 알고 있어야 합니다. 자세한 내용은 the section called “GPIO 핀 순서” 섹션을 참조하세요.
- Raspberry Pi의 `/dev/gpiomem`을 가리키는 Greengrass 그룹의 로컬 디바이스 리소스. 콘솔에서 리소스를 생성하는 경우 리소스를 소유한 Linux 그룹의 OS 그룹 권한 자동 추가 옵션을 선택해야 합니다. API에서 `GroupOwnerSetting.AutoAddGroupOwner` 속성을 `true`(으)로 설정합니다.
- Raspberry Pi에 설치된 RPi.GPIO 모듈. Raspbian에서 이 모듈은 기본적으로 설치되어 있습니다. 다음 명령을 사용해 다시 설치할 수 있습니다.

```
sudo pip install RPi.GPIO
```

GPIO 핀 순서

Raspberry Pi GPIO 커넥터는 GPIO 핀의 물리적 레이아웃이 아니라 기본 시스템 온 칩(SoC)의 번호 지정 체계를 기준으로 GPIO 핀을 참조합니다. Raspberry Pi 버전에서 핀의 물리적 순서가 다를 수 있습니다. 자세한 내용은 Raspberry Pi 설명서에서 GPIO를 참조하세요.

이 커넥터는 구성한 입력 및 출력 핀이 Raspberry Pi의 기본 하드웨어에 올바르게 매핑되는지 확인하지 않습니다. 핀 구성이 잘못된 경우 커넥터는 디바이스에서 시작하려고 할 때 실행 시간 오류를 반환합니다. 이 문제를 해결하려면 커넥터를 재구성한 다음 다시 배포합니다.

> **Note**
>
> 구성 요소 손상을 방지하기 위해 GPIO 핀의 주변 장치가 적절하게 연결되어 있는지 확인합니다.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

`InputGpios`

입력으로 구성할 GPIO 핀 번호의 쉼표로 구분된 목록입니다. 선택적으로 `U`를 추가해 핀의 풀업 저항을 설정하거나, `D`를 추가해 풀다운 저항을 설정합니다. 예: `"5,6U,7D"`.

AWS IoT 콘솔의 표시 이름: 입력 GPIO 핀

필수: `false`. 입력 핀, 출력 핀 또는 둘 다를 지정해야 합니다.

형식: `string`

유효한 패턴: `^$|^[0-9]+[UD]?(,[0-9]+[UD]?)*$`

`InputPollPeriod`

상태가 변경되었는지 입력 GPIO 핀을 확인하는 각 폴링 작업 간의 간격(밀리초). 최소값은 1입니다.

이 값은 시나리오와 폴링되는 디바이스 유형에 따라 달라집니다. 예를 들어 값 `50`은 버튼 누르기를 감지할 수 있을 정도로 빨라야 합니다.

AWS IoT 콘솔의 표시 이름: 입력 GPIO 폴링 기간

필수: `false`

형식: `string`

유효한 패턴: `^$|^[1-9][0-9]*$`

`OutputGpios`

출력으로 구성할 GPIO 핀 번호의 쉼표로 구분된 목록입니다. 선택적으로 `H`를 추가해 높음 상태(1)를 설정하거나, `L`을 추가해 낮음 상태(0)로 설정합니다. 예: `"8H,9,27L"`.

AWS IoT 콘솔의 표시 이름: 출력 GPIO 핀

필수: `false`. 입력 핀, 출력 핀 또는 둘 다를 지정해야 합니다.

형식: `string`

유효한 패턴: `^$|^[0-9]+[HL]?(,[0-9]+[HL]?)*$`

`GpioMem-ResourceId`

`/dev/gpiomem`을 나타내는 로컬 디바이스 리소스의 ID입니다.

> **Note**
> 이 커넥터에는 리소스에 대한 쓰기 전용 액세스 권한이 부여됩니다.

AWS IoT 콘솔의 표시 이름: /dev/gpiomem 디바이스의 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Raspberry Pi GPIO 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyRaspberryPiGPIOConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/
versions/3",
            "Parameters": {
                "GpioMem-ResourceId": "my-gpio-resource",
                "InputGpios": "5,6U,7D",
                "InputPollPeriod": 50,
                "OutputGpios": "8H,9,27L"
            }
        }
    ]
```

```
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 두 개의 MQTT 주제에서 GPIO 핀에 대한 읽기 또는 쓰기 요청을 수락합니다.

- `gpio/+/+/read` 주제에 대한 읽기 요청.
- `gpio/+/+/write` 주제에 대한 쓰기 요청.

이러한 주제에 게시하려면 + 와일드카드를 각각 코어 사물 이름과 대상 핀 번호로 바꿉니다. 예:

```
gpio/core-thing-name/gpio-number/read
```

> **Note**
>
> 현재 Raspberry Pi GPIO 커넥터를 사용하는 구독을 생성할 때는 주제에서 하나 이상의 + 와일드카드에 대한 값을 지정해야 합니다.

주제 필터: `gpio/+/+/read`

이 주제를 사용하여 주제에서 지정된 GPIO 핀의 상태를 읽도록 커넥터에 지시할 수 있습니다.

커넥터는 해당 출력 주제에 응답을 게시합니다(예: `gpio/core-thing-name/gpio-number/state`).

메시지 속성

없음. 이 주제로 전송된 메시지는 무시됩니다.

주제 필터: `gpio/+/+/write`

이 주제는 GPIO 핀에 쓰기 요청을 전송하는 데 사용합니다. 그러면 주제에서 지정된 GPIO 핀을 낮은 전압 또는 높은 전압으로 설정하도록 커넥터에 지시합니다.

- `0`은 핀을 낮은 전압으로 설정합니다.
- `1`은 핀을 높은 전압으로 설정합니다.

커넥터는 해당 출력 `/state` 주제에 응답을 게시합니다(예: `gpio/`*`core-thing-name`*`/`*`gpio-number`*`/state`).

메시지 속성

값 `0` 또는 `1`(정수 또는 문자열)

입력 예

```
0
```

출력 데이터

이 커넥터는 다음 두 개의 주제에 데이터를 게시합니다.

- `gpio/+/+/state` 주제에 대한 높음 또는 낮음 상태 변경.
- `gpio/+/error` 주제에 대한 오류.

주제 필터: `gpio/+/+/state`

이 주제는 입력 핀에 대한 상태 변경과 읽기 요청에 대한 응답을 수신하는 데 사용합니다. 커넥터는 핀이 낮음 상태인 경우 `"0"` 문자열을 반환하거나, 핀이 높음 상태인 경우 `"1"` 문자열을 반환합니다.

이 주제에 게시하는 경우 커넥터는 `+` 와일드카드를 각각 코어 사물 이름 및 대상 핀으로 바꿉니다. 예:

```
gpio/core-thing-name/gpio-number/state
```

> **Note**
>
> 현재 Raspberry Pi GPIO 커넥터를 사용하는 구독을 생성할 때는 주제에서 하나 이상의 + 와일드카드에 대한 값을 지정해야 합니다.

출력 예

```
0
```

주제 필터: `gpio/+/error`

이 주제는 오류를 수신하는 데 사용합니다. 커넥터는 잘못된 요청(예: 입력 핀에 대한 상태 변경이 요청되는 경우)의 결과로서 이 주제에 게시합니다.

이 주제에 게시하는 경우 커넥터는 + 와일드카드를 코어 사물 이름으로 바꿉니다.

출력 예

```
{
   "topic": "gpio/my-core-thing/22/write",
   "error": "Invalid GPIO operation",
   "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations
 are not permitted."
 }
```

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 필요한 로컬 디바이스 리소스를 추가하고 Lambda 함수에 대한 읽기/쓰기 액세스 권한을 부여합니다.

   c. 커넥터를 추가하고 해당 파라미터를 구성합니다.

   d. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.
4. 그룹을 배포합니다.
5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다. 이 예제에서는 입력 GPIO 핀 집합에 대한 읽기 요청을 보냅니다. 이 예제에서는 코어 사물 이름과 핀 번호를 사용하여 주제를 구성하는 방법을 보여줍니다.

```
import greengrasssdk
import json
```

```
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOS:
     read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
    return
```

라이선스

Raspberry Pi GPIO 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- RPi.GPIO/MIT

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	AWS 리전 지원용 커넥터 ARN이 업데이트됨.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- Raspberry Pi 문서의 GPIO

Serial Stream 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

Serial Stream 커넥터는 AWS IoT Greengrass 코어 디바이스의 직렬 포트에서 읽고 씁니다.

이 커넥터는 다음 두 가지 작업 모드를 지원합니다.

- 온디맨드 읽기 MQTT 주제에 대한 읽기 및 쓰기 요청을 수신하고 읽기 작업의 응답 또는 쓰기 작업의 상태를 게시합니다.
- 폴링 읽기 직렬 포트에서 정기적으로 읽습니다. 이 모드에서는 온디맨드 읽기 요청도 지원합니다.

> **Note**
>
> 읽기 요청의 최대 읽기 길이는 63,994바이트로 제한됩니다. 쓰기 요청의 최대 데이터 길이는 128,000바이트로 제한됩니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
3	`arn:aws:greengrass:region::/connectors/SerialStream/versions/3`
2	`arn:aws:greengrass:region::/connectors/SerialStream/versions/2`
1	`arn:aws:greengrass:region::/connectors/SerialStream/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 대상 직렬 포트를 가리키는 Greengrass 그룹의 로컬 디바이스 리소스입니다.

> **Note**
>
> 이 커넥터를 배포하기 전에 직렬 포트를 설정하고 해당 포트에서 읽고 쓸 수 있는지 확인하는 것이 좋습니다.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- 대상 직렬 포트를 가리키는 Greengrass 그룹의 로컬 디바이스 리소스입니다.

> **Note**
>
> 이 커넥터를 배포하기 전에 직렬 포트를 설정하고 해당 포트에서 읽고 쓸 수 있는지 확인하는 것이 좋습니다.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

`BaudRate`

직렬 연결의 전송 속도입니다.

AWS IoT 콘솔의 표시 이름: Baud 속도

필수: `true`

형식: `string`

유효한 값: `110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400`

유효한 패턴: `^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

`Timeout`

읽기 작업에 대한 제한 시간(초)입니다.

AWS IoT 콘솔의 표시 이름: 제한 시간

필수: `true`

형식: `string`

유효한 값: `1 - 59`

유효한 패턴: `^([1-9]|[1-5][0-9])$`

`SerialPort`

디바이스에서 물리적 직렬 포트의 절대 경로입니다. 로컬 디바이스 리소스에 대해 지정된 소스 경로입니다.

AWS IoT 콘솔의 표시 이름: 직렬 포트

필수: `true`

형식: `string`

유효한 패턴: `[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

물리적 직렬 포트를 나타내는 로컬 디바이스 리소스의 ID입니다.

> **Note**
> 이 커넥터에는 리소스에 대한 쓰기 전용 액세스 권한이 부여됩니다.

AWS IoT 콘솔의 표시 이름: 직렬 포트 리소스

필수: `true`

형식: `string`

유효한 패턴: `[a-zA-Z0-9_-]+`

`PollingRead`

읽기 모드(폴링 읽기 또는 온디맨드 읽기)를 설정합니다.

- 폴링 읽기 모드의 경우 `true`를 지정합니다. 이 모드에서는 `PollingInterval`, `PollingReadType` 및 `PollingReadLength` 속성이 필요합니다.
- 온디맨드 읽기 모드의 경우 `false`를 지정합니다. 이 모드에서는 유형 및 길이 값이 읽기 요청에서 지정됩니다.

AWS IoT 콘솔의 표시 이름: 읽기 모드

필수: `true`

형식: `string`

유효한 값: `true, false`

유효한 패턴: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`PollingReadLength`

각 폴링 읽기 작업에서 읽을 데이터(바이트)의 길이입니다. 이는 폴링 읽기 모드를 사용하는 경우에만 적용됩니다.

AWS IoT 콘솔의 표시 이름: 폴링 읽기 길이

필수: `false`. `PollingRead`가 `true`인 경우 이 속성은 필수입니다.

형식: `string`

유효한 패턴: `^(|[1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])$`

`PollingReadInterval`

폴링 읽기를 수행하는 간격(초)입니다. 이는 폴링 읽기 모드를 사용하는 경우에만 적용됩니다.

AWS IoT 콘솔의 표시 이름: 폴링 읽기 간격

필수: `false`. `PollingRead`가 `true`인 경우 이 속성은 필수입니다.

형식: `string`

유효한 값은 1~999입니다.

유효한 패턴: `^(|[1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

`PollingReadType`

폴링 스레드가 읽는 데이터 형식입니다. 이는 폴링 읽기 모드를 사용하는 경우에만 적용됩니다.

AWS IoT 콘솔의 표시 이름: 폴링 읽기 유형

필수: `false`. `PollingRead`가 `true`인 경우 이 속성은 필수입니다.

형식: `string`

유효한 값: `ascii, hex`

유효한 패턴: `^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

`RtsCts`

RTS/CTS 흐름 제어를 활성화할지 여부를 나타냅니다. 기본값은 `false`입니다. 자세한 내용은 RTS, CTS 및 RTR을 참조하십시오.

AWS IoT 콘솔의 표시 이름: RTS/CTS 흐름 제어

필수: `false`

형식: `string`

유효한 값: `true, false`

유효한 패턴: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`XonXoff`

소프트웨어 흐름 제어를 활성화할지 여부를 나타냅니다. 기본값은 `false`입니다. 자세한 내용은 소프트웨어 흐름 제어를 참조하십시오.

AWS IoT 콘솔의 표시 이름: 소프트웨어 흐름 제어

필수: `false`

형식: `string`

유효한 값: `true, false`

유효한 패턴: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`Parity`

직렬 포트의 패리티입니다. 기본값은 N입니다. 자세한 내용은 패리티를 참조하십시오.

AWS IoT 콘솔의 표시 이름: 직렬 포트 패리티

필수: `false`

형식: `string`

유효한 값: `N, E, O, S, M`

유효한 패턴: `^(|[NEOSMneosm])$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Serial Stream 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다. 폴링 읽기 모드에 대해 커넥터를 구성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySerialStreamConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/
versions/3",
            "Parameters": {
                "BaudRate" : "9600",
                "Timeout" : "25",
                "SerialPort" : "/dev/serial1",
                "SerialPort-ResourceId" : "my-serial-port-resource",
                "PollingRead" : "true",
                "PollingReadLength" : "30",
                "PollingReadInterval" : "30",
                "PollingReadType" : "hex"
            }
        }
    ]
}'
```

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 두 개의 MQTT 주제에서 직렬 포트에 대한 읽기 또는 쓰기 요청을 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

- `serial/+/read/#` 주제에 대한 읽기 요청.
- `serial/+/write/#` 주제에 대한 쓰기 요청.

이러한 주제에 게시하려면 + 와일드카드를 코어 사물 이름으로 바꾸고 # 와일드카드를 직렬 포트의 경로로 바꿉니다. 예:

```
serial/core-thing-name/read/dev/serial-port
```

주제 필터: `serial/+/read/#`

이 주제를 사용하여 온디맨드 읽기 요청을 직렬 핀에 전송합니다. 읽기 요청의 최대 읽기 길이는 63,994바이트로 제한됩니다.

메시지 속성

`readLength`

직렬 포트에서 읽을 데이터의 길이입니다.

필수: `true`

형식: `string`

유효한 패턴: `^[1-9][0-9]*$`

`type`

읽을 데이터의 유형입니다.

필수: `true`

형식: `string`

유효한 값: `ascii, hex`

유효한 패턴: `(?i)^(ascii|hex)$`

`id`

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다.

필수: false

형식: string

유효한 패턴: .+

입력 예

```
{
    "readLength": "30",
    "type": "ascii",
    "id": "abc123"
}
```

주제 필터: serial/+/write/#

이 주제를 사용하여 쓰기 요청을 직렬 핀에 전송합니다. 쓰기 요청의 최대 데이터 길이는 128,000바이트로 제한됩니다.

메시지 속성

data

직렬 포트에 쓸 문자열입니다.

필수: true

형식: string

유효한 패턴: ^[1-9][0-9]*$

type

읽을 데이터의 유형입니다.

필수: true

형식: string

유효한 값: ascii, hex

유효한 패턴: ^(ascii|hex|ASCII|HEX)$

id

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다.

필수: false

형식: `string`

유효한 패턴: `.+`

입력 예: ASCII 요청

```
{
    "data": "random serial data",
    "type": "ascii",
    "id": "abc123"
}
```

입력 예: 16진수 요청

```
{
    "data": "base64 encoded data",
    "type": "hex",
    "id": "abc123"
}
```

출력 데이터

이 커넥터는 다음 두 가지 주제에 대한 출력 데이터를 게시합니다.

- `serial/+/status/#` 주제에 대한 커넥터의 상태 정보.
- `serial/+/read_response/#` 주제에 대한 읽기 요청의 응답입니다.

이 주제에 게시하는 경우 커넥터는 + 와일드 카드를 코어 사물 이름으로 바꾸고 # 와일드카드를 직렬 포트 경로로 바꿉니다. 예:

```
serial/core-thing-name/status/dev/serial-port
```

주제 필터: `serial/+/status/#`

이 주제를 사용하여 읽기 및 쓰기 요청의 상태를 수신합니다. `id` 속성이 응답에 포함된 경우 속성이 응답으로 반환됩니다.

출력 예: 성공

```
{
```

```
    "response": {
        "status": "success"
    },
    "id": "abc123"
}
```

출력 예: 실패

실패 응답은 읽기 또는 쓰기 작업을 수행하는 동안 발생한 오류 또는 제한 시간을 설명하는 `error_message` 속성을 포함합니다.

```
{
    "response": {
        "status": "fail",
        "error_message": "Could not write to port"
    },
    "id": "abc123"
}
```

주제 필터: `serial/+/read_response/#`

이 주제를 사용하여 읽기 작업에서 응답 데이터를 수신합니다. 형식이 `hex`인 경우 응답 데이터는 base64로 인코딩됩니다.

출력 예

```
{
    "data": "output of serial read operation"
    "id": "abc123"
}
```

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.

- 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 필요한 로컬 디바이스 리소스를 추가하고 Lambda 함수에 대한 읽기/쓰기 액세스 권한을 부여합니다.

   c. 그룹에 커넥터를 추가하고 해당 파라미터를 구성합니다.

   d. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 그룹에 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.
4. 그룹을 배포합니다.
5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
 request = {
  "data": "TEST",
  "type": "ascii",
  "id": "abc123"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_serial_stream_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

라이선스

Serial Stream 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- pyserial/BSD

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.

버전	변경
2	AWS 리전 지원용 커넥터 ARN이 업데이트됨.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

ServiceNow MetricBase 통합 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

ServiceNow MetricBase 통합 커넥터는 Greengrass 디바이스의 시계열 지표를 ServiceNow MetricBase에 게시합니다. 이렇게 하면 Greengrass 코어 환경의 시계열 데이터를 저장, 분석, 시각화하고 로컬 이벤트를 처리할 수 있습니다.

이 커넥터는 MQTT 주제에 대한 시계열 데이터를 수신하고 정기적으로 이 데이터를 ServiceNow API에 게시합니다.

이 커넥터를 사용하여 다음과 같은 시나리오를 지원할 수 있습니다.

- Greengrass 디바이스에서 수집한 시계열 데이터를 토대로 임계값 기반의 알림과 경보를 생성합니다.
- ServiceNow 플랫폼에 내장된 사용자 지정 애플리케이션에 Greengrass 디바이스의 시간 서비스 데이터를 사용합니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
4	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/4`
3	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/3`
2	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3 - 4

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상. AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 암호를 지원하도록 구성해야 합니다.

  > **Note**
  >
  > 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다..

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- MetricBase 구독이 활성 상태인 ServiceNow 계정입니다. 이와 함께 해당 계정에 지표 및 지표 테이블이 생성되어 있어야 합니다. 자세한 내용은 ServiceNow 설명서의 MetricBase를 참조하십시오.
- 기본 인증으로 ServiceNow 인스턴스에 로그인하기 위해 사용자 이름과 암호를 저장하는 AWS Secrets Manager의 텍스트 유형 암호입니다. 이 암호에는 "사용자" 및 "암호" 키와 해당 값이 들어 있어야 합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하세요.
- Secrets Manager 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상. AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 암호를 지원하도록 구성해야 합니다.

> **Note**
>
> 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- MetricBase 구독이 활성 상태인 ServiceNow 계정입니다. 이와 함께 해당 계정에 지표 및 지표 테이블이 생성되어 있어야 합니다. 자세한 내용은 ServiceNow 설명서의 MetricBase를 참조하십시오.

- 기본 인증으로 ServiceNow 인스턴스에 로그인하기 위해 사용자 이름과 암호를 저장하는 AWS Secrets Manager의 텍스트 유형 암호입니다. 이 암호에는 "사용자" 및 "암호" 키와 해당 값이 들어 있어야 합니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하세요.
- Secrets Manager 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Version 4

`PublishInterval`

ServiceNow에 대한 이벤트 게시 간격의 최대 대기 시간(초)입니다. 최대값은 900입니다.

이 커넥터는 `PublishBatchSize`이(가) 도달 또는 `PublishInterval`이(가) 만료되면 ServiceNow에 게시합니다.

AWS IoT 콘솔의 표시 이름: 게시 간격(초)

필수: `true`

형식: `string`

유효한 값: `1 - 900`

유효한 패턴: `[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

배치로 묶어 ServiceNow에 게시할 수 있는 지표 값의 최대 개수입니다.

이 커넥터는 `PublishBatchSize`이(가) 도달 또는 `PublishInterval`이(가) 만료되면 ServiceNow에 게시합니다.

AWS IoT 콘솔의 표시 이름: 배치 크기 게시

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`InstanceName`

ServiceNow에 연결할 때 사용하는 인스턴스의 이름입니다.

AWS IoT 콘솔의 표시 이름: ServiceNow 인스턴스의 이름

필수: `true`

형식: `string`

유효한 패턴: `.+`

`DefaultTableName`

시계열 MetricBase 데이터베이스와 연결된 `GlideRecord`가 들어 있는 테이블의 이름입니다. 입력 메시지 페이로드의 `table` 속성으로 이 값을 재정의할 수 있습니다.

AWS IoT 콘솔의 표시 이름: 메트릭을 포함할 테이블 이름

필수: `true`

형식: `string`

유효한 패턴: `.+`

`MaxMetricsToRetain`

새 지표로 바뀌기 전에 메모리에 저장할 수 있는 최대 지표 수입니다.

이러한 제한은 인터넷에 연결되어 있지 않거나 커넥터가 나중에 게시할 지표를 버퍼링하기 시작할 때 적용됩니다. 버퍼가 꽉 차면 가장 오래된 지표가 새 지표로 바뀝니다.

> **Note**
>
> 커넥터에 대한 호스트 프로세스가 중단되면 지표가 저장되지 않습니다. 예를 들어, 그룹 배포 중 또는 디바이스가 다시 시작될 때 발생할 수 있습니다.

이 값은 배치 크기보다 커야 하고, MQTT 메시지의 수신 속도에 따라 메시지를 보관할 수 있을 만큼 커야 합니다.

AWS IoT 콘솔의 표시 이름: 메모리에 보관할 최대 지표

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`AuthSecretArn`

ServiceNow 사용자 이름 및 암호를 저장하는 AWS Secrets Manager의 암호입니다. 텍스트 유형의 암호여야 합니다. 이 암호에는 "사용자" 및 "암호" 키와 해당 값이 들어 있어야 합니다.

AWS IoT 콘솔의 표시 이름: 인증 암호의 ARN

필수: `true`

형식: `string`

유효한 패턴: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

ServiceNow 보안 인증의 Secrets Manager 암호를 참조하는 그룹의 암호 리소스입니다.

AWS IoT 콘솔의 표시 이름: 인증 토큰 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 `GreengrassContainer`이며 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
>
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

형식: `string`

유효한 값: `GreengrassContainer` 또는 `NoContainer`

유효한 패턴: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`PublishInterval`

ServiceNow에 대한 이벤트 게시 간격의 최대 대기 시간(초)입니다. 최대값은 900입니다.

이 커넥터는 `PublishBatchSize`이(가) 도달 또는 `PublishInterval`이(가) 만료되면 ServiceNow에 게시합니다.

AWS IoT 콘솔의 표시 이름: 게시 간격(초)

필수: `true`

형식: `string`

유효한 값: `1 - 900`

유효한 패턴: `[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

배치로 묶어 ServiceNow에 게시할 수 있는 지표 값의 최대 개수입니다.

이 커넥터는 `PublishBatchSize`이(가) 도달 또는 `PublishInterval`이(가) 만료되면 ServiceNow에 게시합니다.

AWS IoT 콘솔의 표시 이름: 배치 크기 게시

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`InstanceName`

ServiceNow에 연결할 때 사용하는 인스턴스의 이름입니다.

AWS IoT 콘솔의 표시 이름: ServiceNow 인스턴스의 이름

필수: `true`

형식: `string`

유효한 패턴: `.+`

`DefaultTableName`

시계열 MetricBase 데이터베이스와 연결된 `GlideRecord`가 들어 있는 테이블의 이름입니다. 입력 메시지 페이로드의 `table` 속성으로 이 값을 재정의할 수 있습니다.

AWS IoT 콘솔의 표시 이름: 메트릭을 포함할 테이블 이름

필수: `true`

형식: `string`

유효한 패턴: `.+`

`MaxMetricsToRetain`

새 지표로 바뀌기 전에 메모리에 저장할 수 있는 최대 지표 수입니다.

이러한 제한은 인터넷에 연결되어 있지 않거나 커넥터가 나중에 게시할 지표를 버퍼링하기 시작할 때 적용됩니다. 버퍼가 꽉 차면 가장 오래된 지표가 새 지표로 바뀝니다.

> **Note**
>
> 커넥터에 대한 호스트 프로세스가 중단되면 지표가 저장되지 않습니다. 예를 들어, 그룹 배포 중 또는 디바이스가 다시 시작될 때 발생할 수 있습니다.

이 값은 배치 크기보다 커야 하고, MQTT 메시지의 수신 속도에 따라 메시지를 보관할 수 있을 만큼 커야 합니다.

AWS IoT 콘솔의 표시 이름: 메모리에 보관할 최대 지표

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`AuthSecretArn`

ServiceNow 사용자 이름 및 암호를 저장하는 AWS Secrets Manager의 암호입니다. 텍스트 유형의 암호여야 합니다. 이 암호에는 "사용자" 및 "암호" 키와 해당 값이 들어 있어야 합니다.

AWS IoT 콘솔의 표시 이름: 인증 암호의 ARN

필수: `true`

형식: `string`

유효한 패턴: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

ServiceNow 보안 인증의 Secrets Manager 암호를 참조하는 그룹의 암호 리소스입니다.

AWS IoT 콘솔의 표시 이름: 인증 토큰 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 ServiceNow MetricBase 통합 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyServiceNowMetricBaseIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ServiceNowMetricBaseIntegration/versions/4",
```

```
            "Parameters": {
                "PublishInterval" : "10",
                "PublishBatchSize" : "50",
                "InstanceName" : "myinstance",
                "DefaultTableName" : "u_greengrass_app",
                "MaxMetricsToRetain" : "20000",
                "AuthSecretArn" : "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "AuthSecretArn-ResourceId" : "MySecretResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 주제에 대한 시계열 지표를 수락하고 이 지표를 ServiceNow에 게시합니다. 입력 메시지는 JSON 형식이어야 합니다.

구독의 주제 필터

`servicenow/metricbase/metric`

메시지 속성

`request`

테이블, 레코드, 지표에 대한 정보입니다. 이 요청은 시계열 POST 요청에서 `seriesRef` 객체를 나타냅니다. 자세한 내용은 Clotho 시계열 API - POST를 참조하십시오.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`subject`

표에 있는 특정 레코드의 `sys_id`입니다.

필수: `true`

형식: `string`

`metric_name`

지표의 필드 이름입니다.

필수: `true`

형식: `string`

`table`

레코드를 저장할 테이블의 이름입니다. 이 값을 지정하여 `DefaultTableName` 파라미터를 재정의할 수 있습니다.

필수: `false`

형식: `string`

`value`

개별 데이터 포인트의 값입니다.

필수: `true`

형식: `float`

`timestamp`

개별 데이터 포인트의 타임스탬프입니다. 기본값은 현재 시간입니다.

필수: `false`

형식: `string`

입력 예

```
{
    "request": {
```

```
        "subject":"ef43c6d40a0a0b5700c77f9bf387afe3",
        "metric_name":"u_count",
        "table": "u_greengrass_app"
        "value": 1.0,
        "timestamp": "2018-10-14T10:30:00"
    }
}
```

출력 데이터

이 커넥터는 상태 정보를 MQTT 주제에 출력 데이터로 게시합니다.

구독의 주제 필터

```
servicenow/metricbase/metric/status
```

출력 예: 성공

```
{
    "response": {
        "metric_name": "Errors",
        "table_name": "GliderProd",
        "processed_on": "2018-10-14T10:35:00",
        "response_id": "khjKSkj132qwr23fcba",
        "status": "success",
        "values": [
            {
                "timestamp": "2016-10-14T10:30:00",
                "value": 1.0
            },
            {
                "timestamp": "2016-10-14T10:31:00",
                "value": 1.1
            }
        ]
    }
}
```

출력 예: 실패

```
{
    "response": {
```

```
        "error": "InvalidInputException",
        "error_message": "metric value is invalid",
        "status": "fail"
    }
}
```

> **Note**
>
> 커넥터가 재시도 가능한 오류(예: 연결 오류)를 감지하면 다음 배치에서 게시를 재시도합니다.

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 필요한 보안 리소스를 추가하고 Lambda 함수에 대한 읽기 액세스 권한을 부여합니다.

c. 커넥터를 추가하고 해당 파라미터를 구성합니다.

d. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

- Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
- 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.

4. 그룹을 배포합니다.
5. AWS IoT 콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
             "subject": '2efdf6badbd523803acfae441b961961',
             "metric_name": 'u_count',
             "value": 1234,
             "timestamp": '2018-10-20T20:22:20',
             "table": 'u_greengrass_metricbase_test'
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=SEND_TOPIC,
```

```
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

라이선스

ServiceNow MetricBase 통합 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- pysnow/MIT

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
4	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	과도한 로깅을 줄이도록 고정합니다.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드”을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”

- the section called “커넥터 시작하기(CLI)”

SNS 커넥터

SNS 커넥터는 Amazon SNS 주제에 메시지를 게시합니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 메시지 구독자가 Greengrass 그룹의 이벤트에 응답할 수 있게 됩니다.

이 커넥터는 MQTT 주제에 대한 SNS 메시지 정보를 수신하고 이 메시지를 지정된 SNS 주제에 전송합니다. 선택적으로 사용자 지정 Lambda 함수를 사용하여 메시지에 대한 필터링 또는 형식 지정 로직을 구현한 후 메시지를 이 커넥터에 게시할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
4	`arn:aws:greengrass:region::/connectors/SNS/versions/4`
3	`arn:aws:greengrass:region::/connectors/SNS/versions/3`
2	`arn:aws:greengrass:region::/connectors/SNS/versions/2`
1	`arn:aws:greengrass:region::/connectors/SNS/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3 - 4

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

> **Note**
>
> Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- 구성된 SNS 주제입니다. 자세한 설명은 Amazon Simple Notification Service 개발자 안내서에서 Amazon SNS 주제 생성을 참조하세요.
- 다음 예제 IAM 정책에서 보듯이, 대상 Amazon SNS 주제에 대해 `sns:Publish` 작업을 할 수 있도록 Greengrass 그룹 역할이 구성되었습니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
    ]
}
```

이 커넥터를 사용하면 입력 메시지 페이로드의 기본 주제를 동적으로 재정의할 수 있습니다. 이 기능을 사용하여 구현하는 경우, IAM 정책은 모든 대상 주제에 대해 `sns:Publish` 권한을 허용해야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해).

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 2.7입니다.
- 구성된 SNS 주제입니다. 자세한 설명은 Amazon Simple Notification Service 개발자 안내서에서 Amazon SNS 주제 생성을 참조하세요.
- 다음 예제 IAM 정책에서 보듯이, 대상 Amazon SNS 주제에 대해 `sns:Publish` 작업을 할 수 있도록 Greengrass 그룹 역할이 구성되었습니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
    ]
}
```

이 커넥터를 사용하면 입력 메시지 페이로드의 기본 주제를 동적으로 재정의할 수 있습니다. 이 기능을 사용하여 구현하는 경우, IAM 정책은 모든 대상 주제에 대해 `sns:Publish` 권한을 허용해야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 통해).

그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Version 4

`DefaultSNSArn`

메시지를 게시할 기본 SNS 주제의 ARN입니다. 입력 메시지 페이로드의 `sns_topic_arn` 속성으로 대상 주제를 재정의할 수 있습니다.

> **Note**
>
> 그룹 역할은 모든 대상 주제에 대해 `sns:Publish` 권한을 허용해야 합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 기본 SNS 주제 ARN

필수: `true`

유형: `string`

유효한 패턴: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 `GreengrassContainer`이며 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
>
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

유형: `string`

유효한 값: `GreengrassContainer`또는 `NoContainer`

유효한 패턴: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`DefaultSNSArn`

메시지를 게시할 기본 SNS 주제의 ARN입니다. 입력 메시지 페이로드의 `sns_topic_arn` 속성으로 대상 주제를 재정의할 수 있습니다.

> **Note**
>
> 그룹 역할은 모든 대상 주제에 대해 `sns:Publish` 권한을 허용해야 합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

AWS IoT 콘솔의 표시 이름: 기본 SNS 주제 ARN

필수: `true`

유형: `string`

유효한 패턴: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 SNS 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySNSConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",
            "Parameters": {
                "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 설명은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 주제에 대한 SNS 메시지 정보를 수락한 다음 이 메시지를 있는 그대로 대상 SNS 주제에 게시합니다. 입력 메시지는 JSON 형식이어야 합니다.

구독의 주제 필터

`sns/message`

메시지 속성

`request`

SNS 주제로 보낼 메시지에 대한 정보입니다.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`message`

문자열 또는 JSON 형식의 메시지 내용입니다. 예제는 입력 예를 참조하십시오.

JSON을 보내려면 `message_structure` 속성이 `json`으로 설정되어야 하며 메시지는 `default` 키가 포함되고 문자열 인코딩을 거친 JSON 객체여야 합니다.

필수: `true`

유형: `string`

유효한 패턴: `.*`

`subject`

메시지의 제목입니다.

필수: `false`

유형: 100자 이하의 ASCII 텍스트여야 합니다. 문자, 숫자 또는 구두점으로 시작해야 합니다. 줄 바꿈이나 컨트롤 문자를 넣으면 안 됩니다.

유효한 패턴: `.*`

`sns_topic_arn`

메시지를 게시할 SNS 주제의 ARN입니다. 지정된 경우 커넥터는 기본 주제 대신 이 주제에 게시합니다.

> **Note**
>
> 그룹 역할은 모든 대상 주제에 대해 `sns:Publish` 권한을 허용해야 합니다. 자세한 설명은 the section called “요구 사항” 섹션을 참조하세요.

필수: `false`

유형: `string`

유효한 패턴: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`message_structure`

메시지의 구조입니다.

필수: `false`. JSON 메시지를 전송하도록 지정해야 합니다.

유형: `string`

유효값: `json`

`id`

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다. 지정하면 응답 객체의 `id` 속성이 이 값으로 설정됩니다. 이 기능을 사용하지 않는 경우 이 속성을 생략하거나 빈 문자열로 지정할 수 있습니다.

필수: `false`

유형: `string`

유효한 패턴: `.*`

Limits

메시지 크기는 최대 SNS 메시지 크기인 256KB로 제한됩니다.

입력 예: 문자열 메시지

이 예제에서는 문자열 메시지를 보냅니다. 기본 대상 주제를 재정의하는 선택적 `sns_topic_arn` 속성을 지정합니다.

```
{
    "request": {
        "subject": "Message subject",
        "message": "Message data",
        "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
    },
    "id": "request123"
}
```

입력 예: JSON 메시지

이 예제에서는 메시지를 `default` 키가 포함되며 문자열 인코딩을 거친 JSON 객체로서 보냅니다.

```
{
    "request": {
        "subject": "Message subject",
        "message": "{ \"default\": \"Message data\" }",
        "message_structure": "json"
    },
    "id": "request123"
}
```

출력 데이터

이 커넥터는 상태 정보를 MQTT 주제에 출력 데이터로 게시합니다.

구독의 주제 필터

`sns/message/status`

출력 예: 성공

```
{
    "response": {
        "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
        "status": "success"
    },
```

```
    "id": "request123"
}
```

출력 예: 실패

```
{
   "response" : {
        "error": "InvalidInputException",
        "error_message": "SNS Topic Arn is invalid",
        "status": "fail"
   },
   "id": "request123"
}
```

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.

   그룹 역할 요구 사항의 경우, 필수 권한을 부여하도록 역할을 구성하고 역할이 그룹에 추가되었는지 확인해야 합니다. 자세한 내용은 the section called “그룹 역할 관리(콘솔)” 또는 the section called “그룹 역할 관리(CLI)” 섹션을 참조하세요.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.

3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 커넥터를 추가하고 해당 파라미터를 구성합니다.

   c. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

      - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
      - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.

4. 그룹을 배포합니다.

5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
```

```
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

라이선스

SNS 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- AWS SDK for Python (Boto3)/Apache 라이선스 2.0
- botocore/Apache 라이선스 2.0
- dateutil/PSF 라이선스
- docutils/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- jmespath/MIT 라이선스
- s3transfer/Apache 라이선스 2.0
- urllib3/MIT 라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
4	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	과도한 로깅을 줄이도록 고정합니다.

버전	변경
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드”을 참조하십시오.

다음 사항도 참조하십시오.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- Boto 3 설명서의 게시 작업
- Amazon Simple Notification Service 개발자 가이드의 Amazon Simple Notification Service란?

Splunk 통합 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

Splunk Integration 커넥터는 Greengrass 디바이스의 데이터를 Splunk에 게시합니다. 이렇게 하면 Greengrass 코어 환경을 Splunk로 모니터링 및 분석하고 로컬 이벤트를 처리할 수 있습니다. 이 커넥터는 HEC(HTTP Event Collector)와 통합됩니다. 자세한 내용은 Splunk 설명서의 Introduction to Splunk HTTP Event Collector를 참조하십시오.

이 커넥터는 MQTT 주제에 대한 로깅 및 이벤트 데이터를 수신하고 이 데이터를 있는 그대로 Splunk API에 게시합니다.

이 커넥터를 사용하여 다음과 같은 산업 시나리오를 지원할 수 있습니다.

- 작업자는 액추에이터와 센서에서 주기적으로 보내는 데이터(예: 온도, 압력, 물 판독값)를 사용하여 값이 특정 임계값을 초과하면 경보를 시작할 수 있습니다.

- 개발자는 산업 장비에서 수집한 데이터로 ML 모델을 빌드하고, 이것으로 장비를 모니터링하여 잠재적 문제를 파악할 수 있습니다.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
4	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/4`
3	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/3`
2	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 3 - 4

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상. AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 암호를 지원하도록 구성해야 합니다.

> **Note**
>
> 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

  > **Note**
  >
  > Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- Splunk에서 HTTP Event Collector 기능이 활성화되어 있어야 합니다. 자세한 내용은 Splunk 설명서의 Set up and use HTTP Event Collector in Splunk Web을 참조하십시오.
- Splunk HTTP Event Collector 토큰을 저장하는 AWS Secrets Manager의 텍스트 유형 암호입니다. 자세한 내용은 Splunk 설명서의 이벤트 컬렉터 토큰에 대한 정보 및 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하십시오.

  > **Note**
  >
  > Secrets Manager 콘솔에서 시크릿을 생성하려면 Plaintext 탭에 토큰을 입력하세요. 따옴표나 기타 서식을 포함하지 마세요. API에서 토큰을 `SecretString` 속성 값으로 지정합니다.

- Secrets Manager 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

Versions 1 - 2

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상. AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 암호를 지원하도록 구성해야 합니다.

> **Note**
>
> 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Splunk에서 HTTP Event Collector 기능이 활성화되어 있어야 합니다. 자세한 내용은 Splunk 설명서의 Set up and use HTTP Event Collector in Splunk Web을 참조하십시오.
- Splunk HTTP Event Collector 토큰을 저장하는 AWS Secrets Manager의 텍스트 유형 암호입니다. 자세한 내용은 Splunk 설명서의 이벤트 컬렉터 토큰에 대한 정보 및 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하십시오.

> **Note**
>
> Secrets Manager 콘솔에서 시크릿을 생성하려면 Plaintext 탭에 토큰을 입력하세요. 따옴표나 기타 서식을 포함하지 마세요. API에서 토큰을 `SecretString` 속성 값으로 지정합니다.

- Secrets Manager 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Version 4

`SplunkEndpoint`

Splunk 인스턴스의 엔드포인트입니다. 이 값은 프로토콜, 호스트 이름 및 포트를 포함해야 합니다.

AWS IoT 콘솔의 표시 이름: Splunk 엔드포인트

필수: `true`

형식: `string`

유효한 패턴: `^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

커넥터에 할당할 메모리 양(KB)입니다.

AWS IoT 콘솔의 표시 이름: 메모리 크기

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`SplunkQueueSize`

메모리에 저장할 수 있는 최대 항목 수로, 이 수를 넘어서면 항목을 제출하거나 폐기합니다. 이 한도에 도달하면 대기열에서 가장 오래된 항목부터 새 항목으로 바꿉니다. 이러한 제한은 흔히 인터넷에 연결되어 있지 않을 때 적용됩니다.

AWS IoT 콘솔의 표시 이름: 유지할 수 있는 최대 항목

필수: `true`

형식: `string`

유효한 패턴: `^[0-9]+$`

`SplunkFlushIntervalSeconds`

수신된 데이터를 Splunk HEC에 게시하는 간격(초)입니다. 최대값은 900입니다. 항목이 수신될 때 항목을 게시하도록 커넥터를 구성하려면 0을 지정합니다.

AWS IoT 콘솔의 표시 이름: Splunk 게시 간격

필수: `true`

형식: `string`

유효한 패턴: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

Splunk 토큰을 저장하는 AWS Secrets Manager의 암호입니다. 텍스트 유형의 암호여야 합니다.

AWS IoT 콘솔의 표시 이름: Splunk 인증 토큰 비밀의 ARN

필수: `true`

형식: `string`

유효한 패턴: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Splunk 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다.

AWS IoT 콘솔의 표시 이름: Splunk 인증 토큰 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

`SplunkCustomCALocation`

Splunk에 대한 사용자 지정 CA(인증 기관)의 파일 경로입니다(예: `/etc/ssl/certs/splunk.crt`).

AWS IoT 콘솔의 표시 이름: Splunk 사용자 지정 인증 기관 위치

필수: `false`

형식: `string`

유효한 패턴: `^$|/.*`

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 `GreengrassContainer`이며 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

형식: string

유효한 값: GreengrassContainer 또는 NoContainer

유효한 패턴: ^NoContainer$|^GreengrassContainer$

Version 1 - 3

SplunkEndpoint

Splunk 인스턴스의 엔드포인트입니다. 이 값은 프로토콜, 호스트 이름 및 포트를 포함해야 합니다.

AWS IoT 콘솔의 표시 이름: Splunk 엔드포인트

필수: true

형식: string

유효한 패턴: ^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$

MemorySize

커넥터에 할당할 메모리 양(KB)입니다.

AWS IoT 콘솔의 표시 이름: 메모리 크기

필수: true

형식: string

유효한 패턴: ^[0-9]+$

SplunkQueueSize

메모리에 저장할 수 있는 최대 항목 수로, 이 수를 넘어서면 항목을 제출하거나 폐기합니다. 이 한도에 도달하면 대기열에서 가장 오래된 항목부터 새 항목으로 바꿉니다. 이러한 제한은 흔히 인터넷에 연결되어 있지 않을 때 적용됩니다.

AWS IoT 콘솔의 표시 이름: 유지할 수 있는 최대 항목

필수: true

형식: `string`

유효한 패턴: `^[0-9]+$`

`SplunkFlushIntervalSeconds`

수신된 데이터를 Splunk HEC에 게시하는 간격(초)입니다. 최대값은 900입니다. 항목이 수신될 때 항목을 게시하도록 커넥터를 구성하려면 0을 지정합니다.

AWS IoT 콘솔의 표시 이름: Splunk 게시 간격

필수: `true`

형식: `string`

유효한 패턴: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

Splunk 토큰을 저장하는 AWS Secrets Manager의 암호입니다. 텍스트 유형의 암호여야 합니다.

AWS IoT 콘솔의 표시 이름: Splunk 인증 토큰 비밀의 ARN

필수: `true`

형식: `string`

유효한 패턴: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Splunk 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다.

AWS IoT 콘솔의 표시 이름: Splunk 인증 토큰 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

`SplunkCustomCALocation`

Splunk에 대한 사용자 지정 CA(인증 기관)의 파일 경로입니다(예: `/etc/ssl/certs/splunk.crt`).

AWS IoT 콘솔의 표시 이름: Splunk 사용자 지정 인증 기관 위치

필수: `false`

형식: `string`

유효한 패턴: `^$|/.*`

커넥터 만들기 예(AWS CLI)

다음 CLI 명령은 Splunk 통합 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySplunkIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/
versions/4",
            "Parameters": {
                "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",
                "MemorySize": 200000,
                "SplunkQueueSize": 10000,
                "SplunkFlushIntervalSeconds": 5,
                "SplunkTokenSecretArn":"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "SplunkTokenSecretArn-ResourceId": "MySplunkResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

> **Note**
>
> 이 커넥터의 Lambda 함수에는 수명이 긴 수명 주기가 있습니다.

AWS IoT Greengrass 콘솔에서는 그룹의 커넥터 페이지에서 커넥터를 추가할 수 있습니다. 자세한 내용은 the section called “커넥터 시작하기(콘솔)” 섹션을 참조하세요.

입력 데이터

이 커넥터는 MQTT 주제에 대한 로깅 및 이벤트 데이터를 수락하고 수신된 데이터를 있는 그대로 Splunk API에 게시합니다. 입력 메시지는 JSON 형식이어야 합니다.

구독의 주제 필터

`splunk/logs/put`

메시지 속성

`request`

Splunk API로 보낼 이벤트 데이터입니다. 이벤트는 services/collector API의 사양을 충족해야 합니다.

필수: `true`

유형: `object`. `event` 속성만 필수 입력 사항입니다.

`id`

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 상태에 매핑하는 데 사용됩니다.

필수: `false`

형식: `string`

제한

이 커넥터를 사용할 때는 Splunk API에서 부과하는 제한이 모두 적용됩니다. 자세한 내용은 services/collector를 참조하십시오.

입력 예

```
{
    "request": {
        "event": "some event",
        "fields": {
            "severity": "INFO",
            "category": [
                "value1",
                "value2"
            ]
        }
    },
```

```
    "id": "request123"
}
```

출력 데이터

이 커넥터는 다음 두 가지 주제에 대한 출력 데이터를 게시합니다.

- `splunk/logs/put/status` 주제에 대한 상태 정보
- `splunk/logs/put/error` 주제에 대한 오류.

주제 필터: `splunk/logs/put/status`

요청의 상태를 수신하려면 이 주제를 사용합니다. 커넥터는 수신된 데이터 배치를 Splunk API에 전송할 때마다 성공한 요청과 실패한 요청의 ID 목록을 게시합니다.

출력 예

```
{
    "response": {
        "succeeded": [
            "request123",
            ...
        ],
        "failed": [
            "request789",
            ...
        ]
    }
}
```

주제 필터: `splunk/logs/put/error`

커넥터로부터 오류를 수신하려면 이 주제를 사용합니다. 요청을 처리하는 동안 발생한 오류 또는 시간 초과를 설명하는 `error_message` 속성입니다.

출력 예

```
{
    "response": {
        "error": "UnauthorizedException",
        "error_message": "invalid splunk token",
```

```
        "status": "fail"
    }
}
```

> **Note**
>
> 커넥터가 재시도 가능한 오류(예: 연결 오류)를 감지하면 다음 배치에서 게시를 재시도합니다.

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> - 다른 Python 런타임을 사용하는 경우 Python3.x에서 Python 3.7로의 심볼릭 링크를 만들 수 있습니다.
> - 커넥터 시작하기(콘솔) 및 커넥터 시작하기(CLI) 주제에는 예제 Twilio 알림 커넥터를 구성하고 배포하는 방법을 보여주는 자세한 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 필요한 보안 리소스를 추가하고 Lambda 함수에 대한 읽기 액세스 권한을 부여합니다.

   c. 커넥터를 추가하고 해당 파라미터를 구성합니다.

d. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

   - Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
   - 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.

4. 그룹을 배포합니다.
5. AWS IoT콘솔의테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()
```

```
def lambda_handler(event, context):
    return
```

라이선스

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
4	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
3	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
2	과도한 로깅을 줄이도록 고정합니다.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”

Twilio 알림 커넥터

> **Warning**
>
> 이 커넥터는 수명 연장 단계로 전환되었으며 AWS IoT Greengrass은(는) 기능, 기존 기능의 개선 사항, 보안 패치 또는 버그 수정을 제공하는 업데이트를 릴리스하지 않을 예정입니다. 자세한 내용은 AWS IoT Greengrass Version 1 유지 관리 정책 섹션을 참조하세요.

Twilio Notifications 커넥터는 Twilio를 통해 자동 전화 통화를 수행하고 텍스트 메시지를 전송합니다. 이 커넥터를 사용하면 Greengrass 그룹의 이벤트에 응답하여 알림을 전송할 수 있습니다. 전화 통화의 경우 커넥터는 수신자에게 음성 메시지를 전달할 수 있습니다.

이 커넥터는 MQTT 주제에 대한 Twilio 메시지 정보를 수신한 다음 Twilio 알림을 트리거합니다.

> **Note**
>
> Twilio Notifications 커넥터를 사용하는 방법을 보여 주는 자습서는 the section called “커넥터 시작하기(콘솔)” 또는 the section called “커넥터 시작하기(CLI)” 단원을 참조하십시오.

이 커넥터의 버전은 다음과 같습니다.

버전	ARN
5	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/5`
4	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/4`
3	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/3`

버전	ARN
2	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/2`
1	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/1`

버전 변경 사항에 대한 자세한 내용은 Changelog를 참조하십시오.

요구 사항

이 커넥터에는 다음과 같은 요구 사항이 있습니다.

Version 4 - 5

- AWS IoT Greengrass 코어 소프트웨어 v1.9.3 이상. AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 암호를 지원하도록 구성해야 합니다.

  > **Note**
  >
  > 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7 또는 3.8입니다.

  > **Note**
  >
  > Python 3.8을 사용하려면 다음 명령을 실행하여 기본 Python 3.7 설치 폴더에서 설치된 Python 3.8 바이너리로 연결되는 심볼릭 링크를 만드십시오.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다.

- Twilio 계정 SID, 인증 토큰 및 Twilio 사용 전화번호. Twilio 프로젝트를 생성하면 프로젝트 대시보드에서 이러한 값을 확인할 수 있습니다.

  > **Note**
  >
  > Twilio 시험 계정을 사용할 수 있습니다. 평가판 계정을 사용하는 경우 인증된 전화번호 목록에 Twilio 수신자가 아닌 전화번호를 추가해야 합니다. 자세한 내용은 무료 Twilio 시험 계정을 사용하는 방법.을 참조하십시오.

- Twilio 인증 토큰을 저장하는 AWS Secrets Manager의 텍스트 유형 암호입니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하세요.

  > **Note**
  >
  > Secrets Manager 콘솔에서 시크릿을 생성하려면 Plaintext 탭에 토큰을 입력하세요. 따옴표나 기타 서식을 포함하지 마세요. API에서 토큰을 `SecretString` 속성 값으로 지정합니다.

- Secrets Manager 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

Versions 1 - 3

- AWS IoT Greengrass 코어 소프트웨어 v1.7 이상. AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 암호를 지원하도록 구성해야 합니다.

  > **Note**
  >
  > 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Twilio 계정 SID, 인증 토큰 및 Twilio 사용 전화번호. Twilio 프로젝트를 생성하면 프로젝트 대시보드에서 이러한 값을 확인할 수 있습니다.

> **Note**
>
> Twilio 시험 계정을 사용할 수 있습니다. 평가판 계정을 사용하는 경우 인증된 전화번호 목록에 Twilio 수신자가 아닌 전화번호를 추가해야 합니다. 자세한 내용은 무료 Twilio 시험 계정을 사용하는 방법.을 참조하십시오.

- Twilio 인증 토큰을 저장하는 AWS Secrets Manager의 텍스트 유형 암호입니다. 자세한 정보는 AWS Secrets Manager 사용 설명서의 기본 비밀 생성을 참조하세요.

> **Note**
>
> Secrets Manager 콘솔에서 시크릿을 생성하려면 Plaintext 탭에 토큰을 입력하세요. 따옴표나 기타 서식을 포함하지 마세요. API에서 토큰을 `SecretString` 속성 값으로 지정합니다.

- Secrets Manager 암호를 참조하는 Greengrass 그룹의 암호 리소스입니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

커넥터 파라미터

이 커넥터는 다음 파라미터를 제공합니다.

Version 5

`TWILIO_ACCOUNT_SID`

Twilio API를 호출하는 데 사용되는 Twilio 계정 SID입니다.

AWS IoT 콘솔의 표시 이름: Twilio 계정 SID

필수: `true`

형식: `string`

유효한 패턴: `.+`

`TwilioAuthTokenSecretArn`

Twilio 인증 토큰을 저장하는 Secrets Manager 암호의 ARN입니다.

> **Note**
>
> 이 항목은 코어에서 로컬 암호의 값에 액세스하는 데 사용됩니다.

AWS IoT 콘솔의 표시 이름: Twilio 인증 토큰 암호의 ARN

필수: `true`

형식: `string`

유효한 패턴: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

Greengrass 그룹에서 Twilio 인증 토큰의 암호를 참조하는 암호 리소스의 ID입니다.

AWS IoT 콘솔의 표시 이름: Twilio 인증 토큰 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

`DefaultFromPhoneNumber`

Twilio가 메시지를 전송하는 데 사용하는 기본 Twilio 지원 전화번호입니다. Twilio는 이 번호를 사용하여 텍스트나 통화를 초기화합니다.

- 기본 전화번호를 구성하지 않는 경우 입력 메시지 본문의 `from_number` 속성에서 전화번호를 지정해야 합니다.
- 기본 전화번호를 구성하는 경우 입력 메시지 본문의 `from_number` 속성을 지정하여 기본값을 선택적으로 재정의할 수 있습니다.

AWS IoT 콘솔의 표시 이름: 전화번호의 기본값

필수: `false`

형식: `string`

유효한 패턴: `^$|\+[0-9]+`

`IsolationMode`

이 커넥터의 컨테이너화 모드입니다. 기본값은 `GreengrassContainer`이며 이는 커넥터가 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됨을 의미합니다.

> **Note**
> 그룹의 기본 컨테이너화 설정은 커넥터에는 적용되지 않습니다.

AWS IoT 콘솔의 표시 이름: 컨테이너 격리 모드

필수: `false`

형식: `string`

유효한 값: `GreengrassContainer` 또는 `NoContainer`

유효한 패턴: `^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

Twilio API를 호출하는 데 사용되는 Twilio 계정 SID입니다.

AWS IoT 콘솔의 표시 이름: Twilio 계정 SID

필수: `true`

형식: `string`

유효한 패턴: `.+`

`TwilioAuthTokenSecretArn`

Twilio 인증 토큰을 저장하는 Secrets Manager 암호의 ARN입니다.

> **Note**
> 이 항목은 코어에서 로컬 암호의 값에 액세스하는 데 사용됩니다.

AWS IoT 콘솔의 표시 이름: Twilio 인증 토큰 암호의 ARN

필수: `true`

형식: `string`

유효한 패턴: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

Greengrass 그룹에서 Twilio 인증 토큰의 암호를 참조하는 암호 리소스의 ID입니다.

AWS IoT 콘솔의 표시 이름: Twilio 인증 토큰 리소스

필수: `true`

형식: `string`

유효한 패턴: `.+`

`DefaultFromPhoneNumber`

Twilio가 메시지를 전송하는 데 사용하는 기본 Twilio 지원 전화번호입니다. Twilio는 이 번호를 사용하여 텍스트나 통화를 초기화합니다.

- 기본 전화번호를 구성하지 않는 경우 입력 메시지 본문의 `from_number` 속성에서 전화번호를 지정해야 합니다.
- 기본 전화번호를 구성하는 경우 입력 메시지 본문의 `from_number` 속성을 지정하여 기본값을 선택적으로 재정의할 수 있습니다.

AWS IoT 콘솔의 표시 이름: 전화번호의 기본값

필수: `false`

형식: `string`

유효한 패턴: `^$|\+[0-9]+`

커넥터 만들기 예(AWS CLI)

다음 예제 CLI 명령은 Twilio Notifications 커넥터가 포함된 초기 버전을 사용하여 `ConnectorDefinition`을 생성합니다.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
```

```
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/5",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "abcd12345xyz",
                "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
                "DefaultFromPhoneNumber": "+19999999999",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

Twilio Notifications 커넥터를 그룹에 추가하는 방법을 보여 주는 자습서는 the section called “커넥터 시작하기(CLI)” 및 the section called “커넥터 시작하기(콘솔)” 단원을 참조하십시오.

입력 데이터

이 커넥터는 다음 두 가지 MQTT 주제에 대한 Twilio 메시지 정보를 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

- `twilio/txt` 주제에 대한 텍스트 메시지 정보.
- `twilio/call` 주제에 대한 전화 메시지 정보.

> **Note**
>
> 입력 메시지 페이로드에는 텍스트 메시지(`message`) 또는 음성 메시지(`voice_message_location`)가 포함될 수 있지만, 둘 다 포함될 수는 없습니다.

주제 필터**`twilio/txt`**

메시지 속성

`request`

Twilio 알림에 대한 정보입니다.

필수: `true`

유형: 다음 속성을 포함하는 object:

recipient

메시지 수신자입니다. 한 명의 수신자만 지원됩니다.

필수: true

유형: 다음 속성을 포함하는 object

name

수신자의 이름입니다.

필수: true

형식: string

유효한 패턴: .*

phone_number

수신자의 전화번호입니다.

필수: true

형식: string

유효한 패턴: \+[1-9]+

message

텍스트 메시지의 텍스트 내용입니다. 이 주제에는 텍스트 메시지만 지원됩니다. 음성 메시지의 경우 twilio/call을 사용합니다.

필수: true

형식: string

유효한 패턴: .+

from_number

발신자의 전화번호입니다. Twilio는 이 번호를 사용하여 메시지를 초기화합니다. DefaultFromPhoneNumber 파라미터가 구성되지 않은 경우 이 속성은 필수입니다. DefaultFromPhoneNumber가 구성된 경우 이 속성을 사용하여 기본값을 재정의할 수 있습니다.

필수: false

형식: string

유효한 패턴: \+[1-9]+

retries

재시도 횟수입니다. 기본값은 0입니다.

필수: false

형식: integer

id

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다.

필수: true

형식: string

유효한 패턴: .+

입력 예

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "message": "Hello from the edge"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

주제 필터**twilio/call**

메시지 속성

request

Twilio 알림에 대한 정보입니다.

필수: `true`

유형: 다음 속성을 포함하는 `object`:

`recipient`

메시지 수신자입니다. 한 명의 수신자만 지원됩니다.

필수: `true`

유형: 다음 속성을 포함하는 `object`

`name`

수신자의 이름입니다.

필수: `true`

형식: `string`

유효한 패턴: `.+`

`phone_number`

수신자의 전화번호입니다.

필수: `true`

형식: `string`

유효한 패턴: `\+[1-9]+`

`voice_message_location`

음성 메시지를 위한 오디오 콘텐츠의 URL입니다. 이 값은 TwiML 형식이어야 합니다. 이 주제에는 음성 메시지만 지원됩니다. 텍스트 메시지의 경우 `twilio/txt`를 사용합니다.

필수: `true`

형식: `string`

유효한 패턴: `.+`

`from_number`

발신자의 전화번호입니다. Twilio는 이 번호를 사용하여 메시지를 초기화합니다. `DefaultFromPhoneNumber` 파라미터가 구성되지 않은 경우 이 속성은 필수입니다.

DefaultFromPhoneNumber가 구성된 경우 이 속성을 사용하여 기본값을 재정의할 수 있습니다.

필수: false

형식: string

유효한 패턴: \+[1-9]+

retries

재시도 횟수입니다. 기본값은 0입니다.

필수: false

형식: integer

id

요청에 대한 임의의 ID입니다. 이 속성은 입력 요청을 출력 응답에 매핑하는 데 사용됩니다.

필수: true

형식: string

유효한 패턴: .+

입력 예

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "voice_message_location": "https://some-public-TwiML"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

출력 데이터

이 커넥터는 상태 정보를 MQTT 주제에 출력 데이터로 게시합니다.

구독의 주제 필터

```
twilio/message/status
```

출력 예: 성공

```
{
    "response": {
        "status": "success",
        "payload": {
            "from_number": "+19999999999",
            "messages": {
                "message_status": "queued",
                "to_number": "+12345000000",
                "name": "Darla"
            }
        }
    },
    "id": "request123"
}
```

출력 예: 실패

```
{
    "response": {
        "status": "fail",
        "error_message": "Recipient name cannot be None",
        "error": "InvalidParameter",
        "payload": None
        }
    },
    "id": "request123"
}
```

출력의 payload 속성은 메시지가 전송될 때 Twilio API의 응답입니다. 커넥터에서 입력 데이터가 잘못되었음을 감지하면(예: 필수 입력 필드를 지정하지 않음) 커넥터는 오류를 반환하고 값을 None으로 설정합니다. 다음은 예제 페이로드입니다.

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
```

```
        'message_status':'undelivered'
    }
}
```

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'queued'
    }
}
```

사용 예

다음 상위 수준 단계를 사용하여 커넥터를 사용해 보는 데 이용할 수 있는 예제 Python 3.7 Lambda 함수를 설정합니다.

> **Note**
>
> the section called “커넥터 시작하기(콘솔)” 및 the section called “커넥터 시작하기(CLI)” 주제에는 Twilio 알림 커넥터를 설정, 배포 및 테스트하는 방법을 보여주는 엔드 투 엔드 단계가 포함되어 있습니다.

1. 커넥터에 대한 요구 사항을 충족하는지 확인합니다.
2. 입력 데이터를 커넥터로 보내는 Lambda 함수를 생성하고 게시합니다.

   예제 코드를 PY 파일로 저장합니다. Python용 AWS IoT Greengrass 코어 SDK를 다운로드하고 압축을 풉니다. 그런 다음 루트 수준에서 PY 파일과 `greengrasssdk` 폴더를 포함하는 zip 패키지를 생성합니다. 이 zip 패키지는 AWS Lambda에 업로드하는 배포 패키지입니다.

   Python 3.7 Lambda 함수를 생성한 후 함수 버전을 게시하고 별칭을 만듭니다.
3. Greengrass 그룹을 구성합니다.

   a. 별칭으로 Lambda 함수를 추가합니다(권장). Lambda 수명 주기를 수명이 긴 함수(또는 CLI의 `"Pinned": true`)로 구성합니다.

   b. 필요한 보안 리소스를 추가하고 Lambda 함수에 대한 읽기 액세스 권한을 부여합니다.

c. 커넥터를 추가하고 해당 파라미터를 구성합니다.

d. 커넥터가 입력 데이터를 수신하고 지원되는 주제 필터에서 출력 데이터를 전송할 수 있도록 허용하는 구독을 추가합니다.

- Lambda 함수를 소스로, 커넥터를 대상으로 설정하고 지원되는 입력 주제 필터를 사용합니다.
- 커넥터를 소스로, AWS IoT Core를 대상으로 설정하고 지원되는 출력 주제 필터를 사용합니다. 이 구독을 사용하여 AWS IoT에서 상태 메시지를 확인합니다.

4. 그룹을 배포합니다.
5. AWS IoT콘솔의 테스트 페이지에서 출력 데이터 주제를 구독하여 커넥터의 상태 메시지를 확인합니다. 예제 Lambda 함수는 수명이 긴 함수로 그룹이 배포된 직후 메시지 전송을 시작합니다.

   테스트를 마치면 Lambda 수명 주기를 온디맨드 함수(또는 CLI의 `"Pinned": false`)로 설정하고 그룹을 배포할 수 있습니다. 이렇게 하면 함수가 메시지 전송을 중지합니다.

예

다음 예제 Lambda 함수는 커넥터에 입력 메시지를 보냅니다. 이 예제에서는 텍스트 메시지를 트리거합니다.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():

    txt = {
        "request": {
            "recipient" : {
                "name": "Darla",
                "phone_number": "+12345000000",
                "message": 'Hello from the edge'
            },
            "from_number" : "+19999999999"
        },
        "id" : "request123"
    }
```

```
    print("Message To Publish: ", txt)

    client.publish(topic=TXT_INPUT_TOPIC,
                   payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return
```

라이선스

Twilio Notifications 커넥터에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- twilio-python/MIT

이 커넥터는 Greengrass 코어 소프트웨어 라이선스 계약에 따라 릴리스됩니다.

Changelog

다음 표에서는 의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

버전	변경
5	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터가 추가되었습니다.
4	Lambda 런타임 요구 사항을 변경하는 Python 3.7로 런타임을 업그레이드했습니다.
3	과도한 로깅을 줄이도록 고정합니다.
2	경미한 버그 수정 및 개선 사항.
1	최초 릴리스.

Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “커넥터 시작하기(콘솔)”
- the section called “커넥터 시작하기(CLI)”
- Twilio API 참조

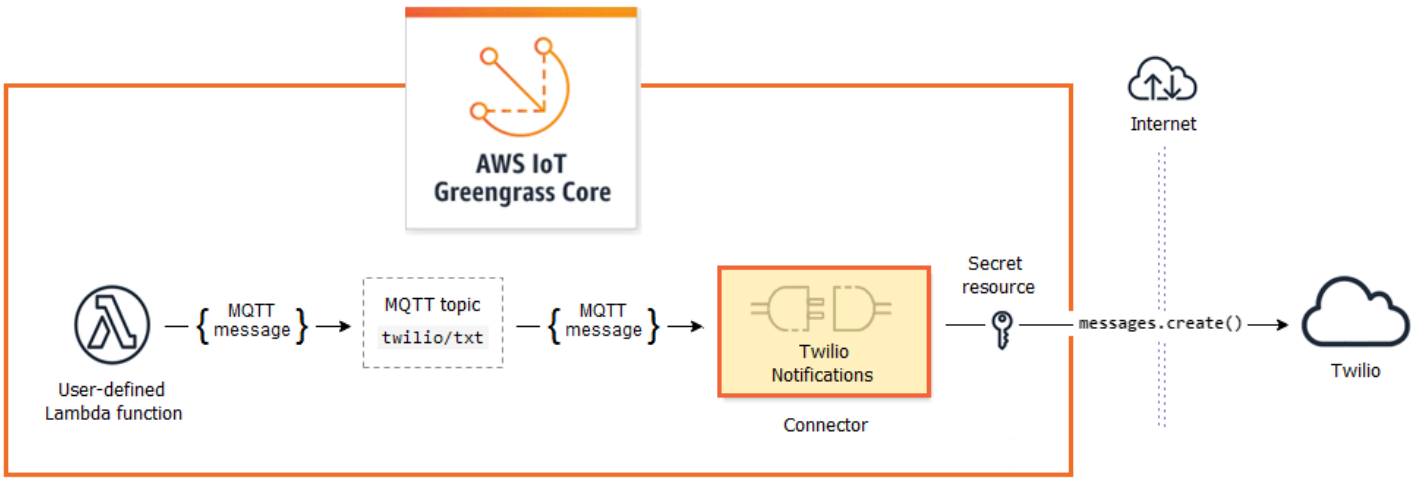
Greengrass 커넥터 시작하기(콘솔)

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

이 자습서는 AWS Management Console에서 커넥터로 작업하는 방법을 보여줍니다.

커넥터를 사용하여 개발 수명 주기를 가속화합니다. 커넥터는 서비스, 프로토콜 및 리소스와 더 쉽게 상호 작용하기 위해 사용할 수 있는 재사용 가능한 미리 빌드된 모듈입니다. 또한 Greengrass 디바이스에 비즈니스 로직을 보다 빠르게 배포할 수 있도록 지원합니다. 자세한 내용은 커넥터를 사용하여 서비스 및 프로토콜과 통합 섹션을 참조하세요.

이 자습서에서는 Twilio Notifications 커넥터를 구성하고 배포합니다. 커넥터는 Twilio 메시지 정보를 입력 데이터로 수신한 다음 Twilio 텍스트 메시지를 트리거합니다. 다음 다이어그램에는 데이터 흐름이 나와 있습니다.



커넥터를 구성한 후 Lambda 함수 및 구독을 생성합니다.

- 이 함수는 온도 센서에서 시뮬레이션된 데이터를 평가합니다. 또한 MQTT 주제에 Twilio 메시지 정보를 조건부로 게시합니다. 커넥터가 구독하는 주제입니다.

- 이 구독을 통해 함수는 주제에 게시할 수 있으며 커넥터는 이 주제에서 데이터를 수신할 수 있습니다.

Twilio Notifications 커넥터가 Twilio API와 상호 작용하려면 Twilio 인증 토큰이 필요합니다. 이러한 토큰은 AWS Secrets Manager에서 생성된 텍스트 유형 보안 함수로, 그룹 리소스에서 참조됩니다. AWS IoT Greengrass는 이 토큰을 사용하여 Greengrass 코어에서 암호의 로컬 사본을 생성할 수 있으며, 여기서 토큰은 암호화되어 커넥터에 사용 가능하게 됩니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Secrets Manager 보안 암호 생성
2. 그룹에 보안 암호 리소스 추가
3. 그룹에 커넥터 추가
4. Lambda 함수 배포 패키지 생성
5. Lambda 함수 생성
6. 그룹에 함수 추가
7. 그룹에 구독 추가
8. 그룹 배포
9. the section called “솔루션 테스트”

이 자습서를 완료하는 데 약 20분 정도 걸립니다.

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- Greengrass 그룹 및 Greengrass 코어(v1.9.3 이상). Greengrass 그룹 및 코어를 생성하는 방법에 대해 알아보려면 시작하기: AWS IoT Greengrass단원을 참조하십시오. 시작하기 자습서에는 AWS IoT Greengrass 코어 소프트웨어를 설치하는 단계도 포함되어 있습니다.
- AWS IoT Greengrass 코어 디바이스에 설치된 Python 3.7입니다.
- AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 비밀을 지원하도록 구성해야 합니다.

> **Note**
>
> 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- Twilio 계정 SID, 인증 토큰 및 Twilio 사용 전화번호. Twilio 프로젝트를 생성하면 프로젝트 대시보드에서 이러한 값을 확인할 수 있습니다.

> **Note**
>
> Twilio 시험 계정을 사용할 수 있습니다. 평가판 계정을 사용하는 경우 인증된 전화번호 목록에 Twilio 수신자가 아닌 전화번호를 추가해야 합니다. 자세한 내용은 무료 Twilio 시험 계정을 사용하는 방법.을 참조하십시오.

1단계: Secrets Manager 보안 암호 생성

이 단계에서는 AWS Secrets Manager 콘솔을 사용하여 Twilio 인증 토큰에 대한 텍스트 유형 보안 암호를 만듭니다.

1. AWS Secrets Manager 콘솔에 로그인합니다.

   > **Note**
   >
   > 이 프로세스에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 1단계: 비밀 만들기 및 AWS Secrets Manager에 저장을 참조하십시오.

2. Store a new secret(새 보안 암호 저장)을 선택합니다.
3. 암호 유형 선택에서 다른 암호 유형을 선택합니다.
4. 이 비밀에 저장할 키/값 쌍 지정의 일반 텍스트 탭에 Twilio 인증 토큰을 입력하십시오. JSON 형식 지정을 모두 제거하고 토큰 값만 입력합니다.
5. 암호화 키로 aws/secretsmanager가 선택된 상태를 유지하고 다음을 선택합니다.

> **Note**
>
> Secrets Manager가 계정에서 생성하는 기본 AWS 관리형 키를 사용하는 경우 AWS KMS에서 요금이 청구되지 않습니다.

6. Secret name(보안 암호 이름)에 **greengrass-TwilioAuthToken**을 입력한 후 다음을 선택합니다.

> **Note**
>
> 기본적으로 Greengrass 서비스 역할은 AWS IoT Greengrass이(가) 이름이 greengrass–로 시작하는 비밀의 값을 가져오도록 허용합니다. 자세한 내용은 암호 요구 사항을 참조하십시오.

7. 이 자습서에서는 교체할 필요가 없으므로 자동 교체 비활성화를 선택하고 다음을 선택합니다.
8. Review(검토) 페이지에서 설정을 검토한 다음 Store(저장)를 선택합니다.

   이제 해당 암호를 참조하는 암호 리소스를 Greengrass 그룹에 생성합니다.

2단계: Greengrass 그룹에 보안 암호 리소스 추가

이 단계에서는 Greengrass 그룹에 보안 암호 리소스를 추가합니다. 이 리소스는 이전 단계에서 생성한 보안 암호에 대한 참조입니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 암호 리소스를 추가할 그룹을 선택합니다.
3. 그룹 구성 페이지에서 리소스 탭을 선택한 다음 비밀 섹션까지 아래로 스크롤합니다. 비밀 섹션에 그룹에 속하는 비밀 리소스가 표시됩니다. 이 섹션에서 암호 리소스를 추가, 편집, 제거할 수 있습니다.

> **Note**
>
> 대안으로, 커넥터 또는 Lambda 함수를 구성할 때 콘솔에서 보안 암호와 보안 암호 리소스를 생성할 수도 있습니다. 커넥터의 파라미터 구성 페이지 또는 Lambda 함수의 리소스 페이지에서 이 작업을 수행할 수 있습니다.

4. 비밀 섹션에서 추가를 선택합니다.

5. 비밀 리소스 추가 페이지에서 리소스 이름으로 **MyTwilioAuthToken**을(를) 입력합니다.
6. 비밀로 greengrass-TwilioAuthToken을 선택하세요.
7. 레이블 선택(선택 사항) 섹션에서 AWSCURRENT 스테이징 레이블은 비밀의 최신 버전을 나타냅니다. 이 레이블은 암호 리소스에 항상 포함되어 있습니다.

> **Note**
>
> 이 자습서에서는 AWSCURRENT 레이블만 있으면 됩니다. Lambda 함수 또는 커넥터에 필요한 레이블을 선택적으로 포함시킬 수 있습니다.

8. 리소스 추가를 선택합니다.

3단계: Greengrass 그룹에 커넥터 추가

이 단계에서는 Twilio Notifications 커넥터에 대한 파라미터를 구성해 그룹에 추가합니다.

1. 그룹 구성 페이지에서 커넥터를 선택한 다음 커넥터 추가를 선택합니다.
2. 커넥터 추가 페이지에서 Twilio 알림을 선택합니다.
3. 버전을 선택합니다.
4. 구성 섹션에서:
   - Twilio 인증 토큰 리소스에 이전 단계에서 생성한 리소스를 입력합니다.

     > **Note**
     >
     > 이 리소스를 입력하면 Twilio 인증 토큰 보안 암호의 ARN 속성이 자동으로 채워집니다.

   - Default from phone number(전화번호의 기본값)에 Twilio 사용 전화번호를 입력합니다.
   - Twilio 계정 SID에 Twilio 계정 SID를 입력합니다.
5. 리소스 추가를 선택합니다.

4단계: Lambda 함수 배포 패키지 생성

Lambda 함수를 생성하려면 먼저 함수 코드와 종속성을 포함하는 Lambda 함수 배포 패키지를 생성해야 합니다. Greengrass Lambda 함수는 코어 환경에서 MQTT 메시지와 통신하고 로컬 비밀에 액세스

하는 등의 작업을 위해 AWS IoT Greengrass Core SDK가 필요합니다. 이 자습서는 Python 함수를 생성하므로 배포 패키지의 Python 버전 SDK를 사용할 수 있습니다.

1. AWS IoT Greengrass 코어 SDK 다운로드 페이지에서 Python용 AWS IoT Greengrass 코어 SDK를 다운로드합니다.
2. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
3. `temp_monitor.py`이라는 로컬 파일에 다음과 같은 Python 코드 함수를 저장합니다.

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
```

```
        "id": "request_" + str(random.randint(1,101))
    }
```

4. 다음 항목을 temp_monitor_python.zip라는 파일로 압축합니다. ZIP 파일을 만들 때 상위 폴더가 아닌 코드 및 종속성만 포함합니다.

   - temp_monitor.py. 앱 로직.
   - greengrasssdk. MQTT 메시지를 게시하는 Python Greengrass Lambda 함수에 대한 필수 라이브러리입니다.

   이것이 Lambda 함수 배포 패키지입니다.

이제, 배포 패키지를 사용하는 Lambda 함수를 생성합니다.

4단계: AWS Lambda 콘솔에서 Lambda 함수 생성

이 단계에서는 AWS Lambda 콘솔을 사용하여 Lambda 함수를 생성한 후 배포 패키지를 사용하도록 구성합니다. 그런 다음 함수 버전을 게시하고 별칭을 생성합니다.

1. 먼저, Lambda 함수를 생성합니다.

   a. AWS Management Console에서 [Services]를 선택한 다음 AWS Lambda 콘솔을 엽니다.

   b. 함수 생성을 선택한 다음 새로 작성을 선택합니다.

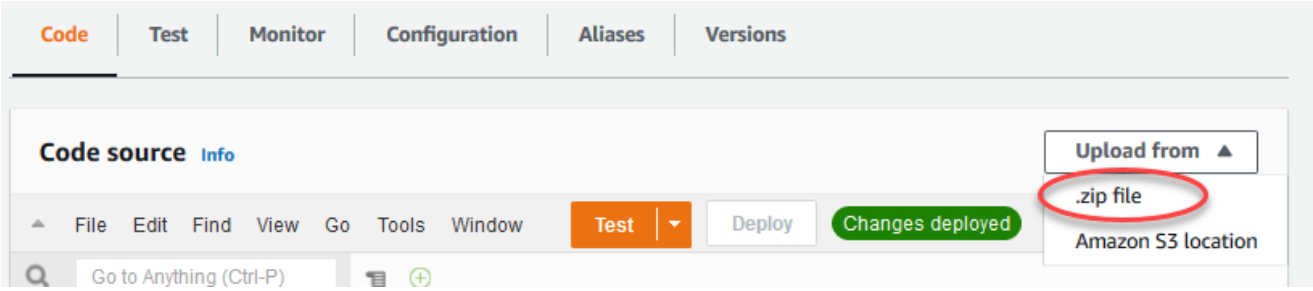
   c. 기본 정보 섹션에서 다음 값을 사용합니다.

      - [함수 이름(Function name)]에 **TempMonitor**을 입력합니다.
      - 실행 시간에서 Python 3.7을 선택합니다.
      - 권한의 경우, 기본 설정을 유지합니다. 이를 통해 기본 Lambda 권한을 부여하는 실행 역할을 생성합니다. 이 역할은 AWS IoT Greengrass에서 사용되지 않습니다.

   d. 페이지 하단에서 함수 생성을 선택합니다.

2. 이제 핸들러를 등록하고 Lambda 함수 배포 패키지를 업로드합니다.

   a. 코드 탭의 코드 소스에서 다음에서 업로드를 선택합니다. 드롭다운에서 .zip 파일을 선택합니다.



b. 업로드를 선택한 다음 `temp_monitor_python.zip` 배포 패키지를 선택합니다. 그런 다음 저장(Save)을 선택합니다.

c. 함수의 코드 탭에 있는 런타임 설정에서 편집을 선택하고 다음 값을 입력합니다.

- 실행 시간에서 Python 3.7을 선택합니다.
- 핸들러에 **`temp_monitor.function_handler`**를 입력합니다.

d. Save를 선택합니다.

> **Note**
>
> AWS Lambda 콘솔의 테스트 버튼은 이 함수와 함께 작동하지 않습니다. AWS IoT Greengrass 코어 SDK에는 AWS Lambda 콘솔에서 Greengrass Lambda 함수를 독립적으로 실행하는 데 필요한 모듈이 포함되어 있지 않습니다. 이러한 모듈(예: `greengrass_common`)은 Greengrass 코어에 배포된 후 함수에 제공됩니다.

3. 이제 Lambda 함수의 첫 번째 버전을 게시하고 버전의 별칭을 생성합니다.

> **Note**
>
> Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

a. [Actions] 메뉴에서 [Publish new revision]을 선택합니다.

b. 버전 설명에 **`First version`**을 입력한 후 게시를 선택합니다.

c. TempMonitor: 1 구성 페이지의 작업 메뉴에서 별칭 생성을 선택합니다.

d. [Create a new alias] 페이지에서 다음 값을 사용합니다.

- 이름(Name)에 **GG_TempMonitor**을 입력합니다.
- 버전에서 1을 선택합니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

e. 생성을 선택합니다.

이제 Greengrass 그룹에 Lambda 함수를 추가할 준비가 되었습니다.

6단계: Greengrass 그룹에 Lambda 함수 추가

이 단계에서 그룹에 Lambda 함수를 추가한 다음 수명 주기와 환경 변수를 구성합니다. 자세한 내용은 the section called “Greengrass Lambda 함수 실행 제어” 섹션을 참조하세요.

1. 그룹 구성 페이지에서 Lambda 함수 탭을 선택합니다.
2. 내 Lambda 함수에서 추가를 선택합니다.
3. Lambda 함수 추가 페이지에서 Lambda 함수에 대한 TempMonitor를 선택합니다.
4. Lambda 함수 버전의 경우 Alias: GG_TempMonitor를 선택합니다.
5. Lambda 함수 추가를 선택합니다.

7단계: Greengrass 그룹에 구독 추가

이 단계에서는 Lambda 함수가 커넥터에 입력 데이터를 전송하는 데 사용할 수 있는 구독을 추가합니다. 커넥터는 자신이 구독하는 MQTT 주제를 정의하므로, 이 구독은 이러한 주제 중 하나를 사용합니다. 이러한 주제는 예제 함수가 게시할 주제와 동일합니다.

이 자습서에서는 함수가 AWS IoT에서 시뮬레이션된 온도 판독값을 수신하고 AWS IoT가 커넥터에서 상태 정보를 수신하도록 허용하는 구독도 생성합니다.

1. 그룹 구성 페이지에서 구독을 선택한 다음 구독 추가를 선택합니다.
2. 구독 생성 페이지에서 다음과 같이 원본과 대상을 구성합니다.

   a. 소스 유형에서 Lambda 함수를 선택한 다음 TempMonitor를 선택합니다.
   b. 대상 유형에서 커넥터를 선택한 다음 Twilio 알림을 선택합니다.

3. 주제 필터의 경우 **twilio/txt**를 선택합니다.
4. 구독 생성을 선택합니다.
5. 1~4단계를 반복하여 AWS IoT에서 함수에 메시지를 게시하도록 허용하는 구독을 생성합니다.

   a. 소스 유형의 경우 서비스를 선택한 다음 IoT Cloud를 선택합니다.
   b. 대상 선택에서 Lambda 함수를 선택한 다음 TempMonitor를 선택합니다.
   c. 주제 필터에 **temperature/input**을 입력합니다.
6. 1~4단계를 반복하여 커넥터가 AWS IoT에 메시지를 게시하도록 허용하는 구독을 생성합니다.

   a. 소스 유형에서 커넥터를 선택한 다음 Twilio 알림을 선택합니다.
   b. 대상 유형에서 서비스를 선택한 다음 IoT Cloud를 선택합니다.
   c. 주제 필터에 **twilio/message/status**가 입력됩니다. 이것은 커넥터가 게시하는 미리 정의된 주제입니다.

8단계: Greengrass 그룹 배포

코어 디바이스에 그룹을 배포합니다.

1. AWS IoT Greengrass 코어가 실행 중인지 확인합니다. 필요한 경우 Raspberry Pi 터미널에서 다음 명령을 실행합니다.

   a. 데몬이 실행 중인지 확인하려면:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   출력에 root에 대한 /greengrass/ggc/packages/*ggc-version*/bin/daemon 입력이 포함되어 있는 경우에는 데몬이 실행 중인 것입니다.

   > **Note**
   > 경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

   b. 데몬을 시작하려면:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. 그룹 구성 페이지에서 배포를 선택합니다.
3. a. Lambda 함수 탭의 시스템 Lambda 함수 섹션에서 IP 감지기를 선택하고 편집을 선택합니다.
   b. IP 감지기 설정 편집 대화 상자에서 MQTT 브로커 엔드포인트 자동 탐지 및 재정의를 선택합니다.
   c. Save를 선택합니다.

   이렇게 하면 디바이스가 IP 주소, DNS, 포트 번호 등 코어의 연결 정보를 자동으로 획득할 수 있습니다. 자동 탐지가 바람직하지만 AWS IoT Greengrass은(는) 수동으로 지정된 엔드포인트도 지원합니다. 그룹이 처음 배포될 때만 검색 방법 메시지가 표시됩니다.

   > **Note**
   >
   > 메시지가 표시되면 Greengrass 서비스 역할을 생성할 권한을 부여하고 이 권한을 현재 AWS 리전의 AWS 계정과 연결합니다. 이 역할은 AWS IoT Greengrass가 AWS 서비스의 리소스에 액세스할 수 있습니다.

   배포 페이지에 배포 타임스탬프, 버전 ID, 상태가 표시됩니다. 완료되면 배포에 대해 성공적으로 완료했습니다 상태가 표시되어야 합니다.

   문제 해결에 대한 도움말은 문제 해결 단원을 참조하십시오.

> **Note**
>
> Greengrass 그룹은 한 번에 하나의 커넥터 버전만 포함할 수 있습니다. 커넥터 버전 업그레이드에 대한 자세한 내용은 the section called “커넥터 버전 업그레이드” 단원을 참조하십시오.

솔루션 테스트

1. AWS IoT 홈 페이지에서 테스트를 선택합니다.
2. 구독에서 다음 값을 사용한 다음 주제 구독을 선택합니다. Twilio Notifications 커넥터는 이 주제에 상태 정보를 게시합니다.

속성	값
구독 주제	twilio/message/status
MQTT 페이로드 디스플레이	페이로드를 문자열로 표시

3. 주제 게시에서 다음 값을 사용한 후 게시를 선택하여 함수를 간접 호출합니다.

속성	값
주제	온도/입력

메시지	*recipient-name* 은 이름으로, *recipient-phone-number* 는 문자 메시지 수신자의 전화번호로 바꿉니다. 예: +12345000000
	{
	"to_name": " *recipient-name* ",
	"to_number": " *recipient-phone-number* ",
	"temperature": 31
	}
	평가판 계정을 사용하는 경우 인증된 전화번호 목록에 Twilio 수신자가 아닌 전화번호를 추가해야 합니다. 자세한 내용은 개인 전화번호 확인을 참조하십시오.

성공하면 수신자가 문자 메시지를 수신하고 콘솔에 출력 데이터의 `success` 상태가 표시됩니다.

이제 입력 메시지에서 `temperature`를 **29**로 변경하고 게시합니다. 이 값이 30보다 작기 때문에 TempMonitor 기능이 Twilio 메시지를 트리거하지 않습니다.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “AWS에서 제공한 Greengrass 커넥터”

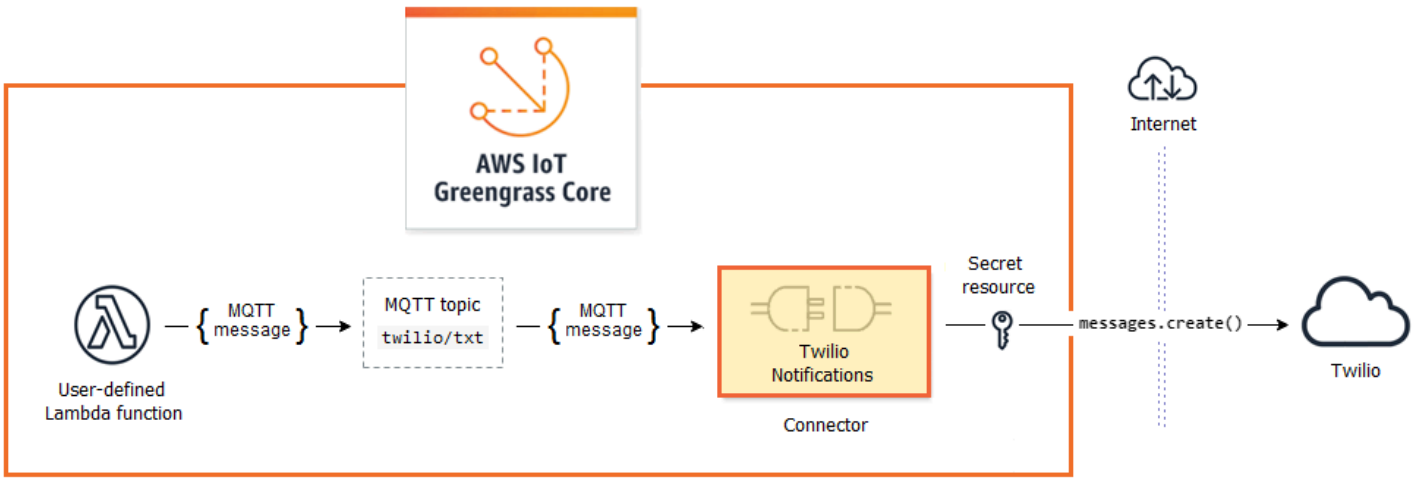
Greengrass 커넥터 시작하기(CLI)

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

이 자습서는 AWS CLI에서 커넥터로 작업하는 방법을 보여줍니다.

커넥터를 사용하여 개발 수명 주기를 가속화합니다. 커넥터는 서비스, 프로토콜 및 리소스와 더 쉽게 상호 작용하기 위해 사용할 수 있는 재사용 가능한 미리 빌드된 모듈입니다. 또한 Greengrass 디바이스에 비즈니스 로직을 보다 빠르게 배포할 수 있도록 지원합니다. 자세한 내용은 커넥터를 사용하여 서비스 및 프로토콜과 통합 섹션을 참조하세요.

이 자습서에서는 Twilio Notifications 커넥터를 구성하고 배포합니다. 커넥터는 Twilio 메시지 정보를 입력 데이터로 수신한 다음 Twilio 텍스트 메시지를 트리거합니다. 다음 다이어그램에는 데이터 흐름이 나와 있습니다.



커넥터를 구성한 후 Lambda 함수 및 구독을 생성합니다.

- 이 함수는 온도 센서에서 시뮬레이션된 데이터를 평가합니다. 또한 MQTT 주제에 Twilio 메시지 정보를 조건부로 게시합니다. 커넥터가 구독하는 주제입니다.
- 이 구독을 통해 함수는 주제에 게시할 수 있으며 커넥터는 이 주제에서 데이터를 수신할 수 있습니다.

Twilio Notifications 커넥터가 Twilio API와 상호 작용하려면 Twilio 인증 토큰이 필요합니다. 이러한 토큰은 AWS Secrets Manager에서 생성된 텍스트 유형 보안 함수로, 그룹 리소스에서 참조됩니다. AWS IoT Greengrass는 이 토큰을 사용하여 Greengrass 코어에서 암호의 로컬 사본을 생성할 수 있으며, 여기서 토큰은 암호화되어 커넥터에 사용 가능하게 됩니다. 자세한 내용은 코어에 암호 배포 섹션을 참조하세요.

자습서에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. Secrets Manager 보안 암호 생성
2. 리소스 정의 및 버전 생성
3. 커넥터 정의 및 버전 생성
4. Lambda 함수 배포 패키지 생성
5. Lambda 함수 생성
6. 함수 정의 및 버전 생성
7. 구독 정의 및 버전 생성
8. 그룹 버전 생성
9. 배포 만들기
10. the section called “솔루션 테스트”

이 자습서를 완료하는 데 약 30분 정도 걸립니다.

AWS IoT Greengrass API 사용

Greengrass 그룹 및 그룹 구성 요소(예: 커넥터, 함수 및 그룹의 리소스)를 사용할 때 다음 패턴을 이해하면 도움이 됩니다.

- 계층 구조의 맨 위에 있는 구성 요소에는 버전 개체의 컨테이너인 정의 개체가 있습니다. 그 다음에, 버전은 커넥터, 함수 또는 기타 구성 요소 유형에 대한 컨테이너입니다.
- Greengrass 코어에 배포하는 경우 특정 그룹 버전을 배포합니다. 그룹 버전에는 구성 요소의 각 유형에 대한 버전이 하나 있을 수 있습니다. 코어가 하나 필요하지만 다른 코어는 필요에 따라 포함됩니다.
- 버전은 변경 불가능하기 때문에 변경하려면 새 버전을 생성해야 합니다.

> **Tip**
>
> AWS CLI 명령 실행 시 오류가 수신되면 `--debug` 파라미터를 추가한 다음 이 명령을 다시 실행해 오류에 대한 추가 정보를 얻습니다.

AWS IoT Greengrass API를 사용하여 구성 요소 유형에 대해 여러 정의를 생성할 수 있습니다. 예를 들어, `FunctionDefinitionVersion`을 생성할 때마다 `FunctionDefinition` 객체를 생성하거나 기존 정의에 새 버전을 추가할 수 있습니다. 이러한 유연성 덕분에 버전 관리 시스템을 사용자 지정할 수 있습니다.

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- Greengrass 그룹 및 Greengrass 코어(v1.9.3 이상). Greengrass 그룹 및 코어를 생성하는 방법에 대해 알아보려면 시작하기: AWS IoT Greengrass단원을 참조하십시오. 시작하기 자습서에는 AWS IoT Greengrass 코어 소프트웨어를 설치하는 단계도 포함되어 있습니다.
- AWS IoT Greengrass 코어 디바이스에 설치된 Python 3.7입니다.
- AWS IoT Greengrass은(는) 비밀 요구 사항에 설명된 대로 로컬 비밀을 지원하도록 구성해야 합니다.

  > **Note**
  >
  > 이 요구 사항에는 Secrets Manager 암호에 대한 액세스 허용이 포함됩니다. 기본 Greengrass 서비스 역할을 사용 중인 경우 는 이미 greengrass-로 시작하는 이름을 가진 암호에 액세스할 수 있습니다.

- Twilio 계정 SID, 인증 토큰 및 Twilio 사용 전화번호. Twilio 프로젝트를 생성하면 프로젝트 대시보드에서 이러한 값을 확인할 수 있습니다.

  > **Note**
  >
  > Twilio 시험 계정을 사용할 수 있습니다. 평가판 계정을 사용하는 경우 인증된 전화번호 목록에 Twilio 수신자가 아닌 전화번호를 추가해야 합니다. 자세한 내용은 무료 Twilio 시험 계정을 사용하는 방법.을 참조하십시오.

- 시스템에 설치 및 구성된 AWS CLI. 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS Command Line Interface 설치 및 AWS CLI 구성을 참조하세요.

  이 자습서의 예제는 Linux 및 기타 Linux 기반 시스템에 맞춰 작성되어 있습니다. Windows를 사용하는 경우 구문의 차이를 알아보려면 AWS Command Line Interface에 대한 파라미터 값 지정을 참조하십시오.

  명령에 JSON 문자열이 포함되어 있는 경우 이 자습서에서는 한 행에 JSON이 포함된 예제를 제공합니다. 일부 시스템에서는 이 형식을 사용해 보다 쉽게 명령을 편집 및 실행할 수 있습니다.

1단계: Secrets Manager 보안 암호 생성

이 단계에서는 AWS Secrets Manager API를 사용하여 Twilio 인증 토큰에 대한 보안 암호를 만듭니다.

1. 먼저 보안 암호를 만듭니다.

   - *twilio-auth-token*을 Twilio 인증 토큰으로 바꿉니다.

   ```
   aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-
   string twilio-auth-token
   ```

   > **Note**
   >
   > 기본적으로 Greengrass 서비스 역할은 AWS IoT Greengrass이(가) 이름이 greengrass–로 시작하는 비밀의 값을 가져오도록 허용합니다. 자세한 내용은 암호 요구 사항을 참조하십시오.

2. 출력에서 보안 암호의 ARN을 복사합니다. 이 정보를 사용하여 암호 리소스를 생성하고 Twilio Notifications 커넥터를 구성합니다.

2단계: 리소스 정의 및 버전 생성

이 단계에서는 AWS IoT Greengrass API를 사용하여 보안 암호에 대한 보안 암호 리소스를 생성합니다.

1. 최초 버전이 포함된 리소스 정의를 생성합니다.

- *secret-arn*을 이전 단계에서 복사한 비밀의 ARN(으)로 바꾸십시오.

JSON Expanded

```
aws greengrass create-resource-definition --name MyGreengrassResources --
initial-version '{
    "Resources": [
        {
            "Id": "TwilioAuthToken",
            "Name": "MyTwilioAuthToken",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "secret-arn"
                }
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-resource-definition \
--name MyGreengrassResources \
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",
 "Name": "MyTwilioAuthToken", "ResourceDataContainer":
 {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. 출력에서 리소스 정의의 `LatestVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 리소스 정의 버전을 추가합니다.

3단계: 커넥터 정의 및 버전 생성

이 단계에서는 Twilio Notifications 커넥터에 대한 파라미터를 구성합니다.

1. 초기 버전을 사용하여 커넥터 정의를 생성합니다.

   - Twilio 계정 SID로 *account-sid*를 바꿉니다.

- *secret-arn*을 보안 암호의 ARN으로 바꿉니다. 커넥터는 이 값을 사용하여 로컬 암호의 값을 가져옵니다.
- *phone-number*를 Twilio 사용 전화번호로 바꿉니다. Twilio는 이 값을 사용해 문자 메시지를 초기화합니다. 이 값은 입력 메시지 페이로드에서 재정의될 수 있습니다. +19999999999 형식을 사용합니다.

JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --
initial-version '{
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "account-sid",
                "TwilioAuthTokenSecretArn": "secret-arn",
                "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
                "DefaultFromPhoneNumber": "phone-number"
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-connector-definition \
--name MyGreengrassConnectors \
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",
 "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",
 "TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}}]}'
```

> **Note**
>
> `TwilioAuthToken`은 이전 단계에서 보안 암호 리소스를 생성하는 데 사용한 ID입니다.

2. 출력에서 커넥터 정의의 `LatestVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 커넥터 정의 버전을 추가합니다.

4단계: Lambda 함수 배포 패키지 생성

Lambda 함수를 생성하려면 먼저 함수 코드와 종속성을 포함하는 Lambda 함수 배포 패키지를 생성해야 합니다. Greengrass Lambda 함수는 코어 환경에서 MQTT 메시지와 통신하고 로컬 비밀에 액세스하는 등의 작업을 위해 AWS IoT Greengrass Core SDK가 필요합니다. 이 자습서는 Python 함수를 생성하므로 배포 패키지의 Python 버전 SDK를 사용할 수 있습니다.

1. AWS IoT Greengrass 코어 SDK 다운로드 페이지에서 Python용 AWS IoT Greengrass 코어 SDK를 다운로드합니다.
2. 다운로드한 패키지의 압축을 풀어 SDK를 가져옵니다. SDK는 `greengrasssdk` 폴더입니다.
3. `temp_monitor.py`이라는 로컬 파일에 다음과 같은 Python 코드 함수를 저장합니다.

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return
```

```
# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. 다음 항목을 temp_monitor_python.zip라는 파일로 압축합니다. ZIP 파일을 만들 때 상위 폴더가 아닌 코드 및 종속성만 포함합니다.

   - temp_monitor.py. 앱 로직.
   - greengrasssdk. MQTT 메시지를 게시하는 Python Greengrass Lambda 함수에 대한 필수 라이브러리입니다.

   이것이 Lambda 함수 배포 패키지입니다.

5단계: Lambda 함수 생성

이제, 배포 패키지를 사용하는 Lambda 함수를 생성합니다.

1. 함수 생성 시 역할 ARN에서 전달할 수 있도록 IAM 역할을 생성합니다.

   JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "lambda.amazonaws.com"
            },
           "Action": "sts:AssumeRole"
        }
    ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
 "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
 "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

> **Note**
>
> Greengrass 그룹 역할에 Greengrass Lambda 함수에 대한 권한이 지정되어 있기 때문에 AWS IoT Greengrass에서는 이 역할을 사용하지 않습니다. 이 자습서에서는 빈 역할을 생성합니다.

2. 출력에서 `Arn`을 복사합니다.
3. AWS Lambda API를 사용하여 TempMonitor 함수를 생성합니다. 다음 명령은 zip 파일이 현재 디렉터리에 있다고 가정합니다.

   - *`role-arn`*을 복사한 `Arn`으로 바꿉니다.

```
aws lambda create-function \
--function-name TempMonitor \
--zip-file fileb://temp_monitor_python.zip \
--role role-arn \
--handler temp_monitor.function_handler \
--runtime python3.7
```

4. 이 함수의 버전을 게시합니다.

```
aws lambda publish-version --function-name TempMonitor --description 'First
 version'
```

5. 게시된 버전에 대한 별칭을 생성합니다.

Greengrass 그룹은 별칭(권장) 또는 버전을 기준으로 Lambda 함수를 참조할 수 있습니다. 별칭을 사용하면 함수 코드를 업데이트할 때 구독 테이블이나 그룹 정의를 변경할 필요가 없으므로 코드 업데이트를 더 쉽게 관리할 수 있습니다. 그 대신 새 함수 버전에 대한 별칭을 가리킵니다.

> **Note**
>
> AWS IoT Greengrass은(는) $LATEST 버전에서 Lambda 별칭을 지원하지 않습니다.

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --
function-version 1
```

6. 출력에서 `AliasArn`을 복사합니다. 이 값은 AWS IoT Greengrass에 대한 함수를 구성할 때 그리고 구독을 생성할 때 사용합니다.

이제 AWS IoT Greengrass에 대해 함수를 구성할 준비가 되었습니다.

6단계: 함수 정의 및 버전 생성

AWS IoT Greengrass 코어에 대해 Lambda 함수를 사용하려면 별칭으로 Lambda 함수를 참조하는 함수 정의 버전을 생성하고 그룹 수준 구성을 정의합니다. 자세한 내용은 the section called “Greengrass Lambda 함수 실행 제어” 섹션을 참조하세요.

1. 최초 버전이 포함된 함수 정의를 생성합니다.

   - *`alias-arn`*을 별칭을 생성했을 때 복사한 `AliasArn`로 바꿉니다.

   JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "TempMonitorFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "temp_monitor.function_handler",
```

```
                "MemorySize": 16000,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. 출력에서 `LatestVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 함수 정의 버전을 추가합니다.
3. 출력에서 `Id`을 복사합니다. 이 값은 나중에 함수를 업데이트할 때 사용합니다.

7단계: 구독 정의 및 버전 생성

이 단계에서는 Lambda 함수가 커넥터에 입력 데이터를 전송하는 데 사용할 수 있는 구독을 추가합니다. 커넥터는 자신이 구독하는 MQTT 주제를 정의하므로, 이 구독은 이러한 주제 중 하나를 사용합니다. 이러한 주제는 예제 함수가 게시할 주제와 동일합니다.

이 자습서에서는 함수가 AWS IoT에서 시뮬레이션된 온도 판독값을 수신하고 AWS IoT가 커넥터에서 상태 정보를 수신하도록 허용하는 구독도 생성합니다.

1. 구독이 포함된 초기 버전이 들어 있는 구독 정의를 생성합니다.

   - *alias-arn*을 함수에 대한 별칭을 생성했을 때 복사한 `AliasArn`로 바꿉니다. 이 ARN을 두 구독에 사용합니다.

JSON Expanded

```
aws greengrass create-subscription-definition --initial-version '{
    "Subscriptions": [
        {
```

```
            "Id": "TriggerNotification",
            "Source": "alias-arn",
            "Subject": "twilio/txt",
            "Target": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4"
        },
        {
            "Id": "TemperatureInput",
            "Source": "cloud",
            "Subject": "temperature/input",
            "Target": "alias-arn"
        },
        {
            "Id": "OutputStatus",
            "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Subject": "twilio/message/status",
            "Target": "cloud"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-subscription-definition \
--initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
 "alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
connectors/TwilioNotifications/versions/4"},{"Id": "TemperatureInput",
 "Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":
 "cloud"}]}'
```

2. 출력에서 `LatestVersionArn`을 복사합니다. 이 값을 사용하여 코어에 배포할 그룹 버전에 구독 정의 버전을 추가합니다.

8단계: 그룹 버전 생성

이제, 배포할 모든 항목이 포함된 그룹 버전을 생성할 준비가 되었습니다. 이렇게 하려면 이 각 구성 요소 유형의 대상 버전을 참조하는 그룹 버전을 생성합니다.

먼저, 코어 정의 버전의 그룹 ID와 ARN을 가져옵니다. 이러한 값은 그룹 버전을 생성하는 데 필요합니다.

1. 그룹 및 최신 그룹 버전의 ID 가져오기:

   a. 대상 Greengrass 그룹 및 그룹 버전의 ID를 확인합니다. 이 절차에서는 이것이 최신 그룹 및 그룹 버전이라고 가정합니다. 다음 쿼리는 가장 최근에 생성된 그룹을 반환합니다.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups,
    &CreationTimestamp))[0]"
   ```

   또는 이름으로 쿼리할 수도 있습니다. 그룹 이름은 고유한 이름이 아니어도 되므로 여러 그룹을 반환할 수도 있습니다.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > AWS IoT 콘솔에서도 이러한 값을 찾을 수 있습니다. 그룹 ID는 그룹의 설정 페이지에 표시됩니다. 그룹 버전 ID는 그룹의 배포 탭에 표시됩니다.

   b. 출력에서 대상 그룹의 `Id`을 복사합니다. 이 값을 사용하여 그룹을 배포할 때 코어 정의 버전을 가져옵니다.

   c. 그룹에 추가된 최신 버전의 ID가 되도록 출력에서 `LatestVersion`을 복사합니다. 이 값을 사용하여 코어 정의 버전을 가져옵니다.

2. 코어 정의 버전의 ARN 가져오기:

   a. 그룹 버전을 가져옵니다. 이 단계에서는 최신 그룹 버전에 코어 정의 버전이 포함되어 있다고 가정합니다.

   - *group-id*를 해당 그룹에서 복사한 `Id`로 바꿉니다.
   - *group-version-id*를 해당 그룹에서 복사한 `LatestVersion`으로 바꿉니다.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id group-version-id
   ```

   b. 출력에서 `CoreDefinitionVersionArn`을 복사합니다.

3. 그룹 버전을 만듭니다.

   - *group-id*를 해당 그룹에서 복사한 Id로 바꿉니다.
   - *core-definition-version-arn*을 새 함수 정의 버전에서 복사한 CoreDefinitionVersionArn으로 바꿉니다.
   - *resource-definition-version-arn*을 리소스 정의에 대해 복사한 LatestVersionArn으로 바꿉니다.
   - *connector-definition-version-arn*을 커넥터 정의에 대해 복사한 LatestVersionArn으로 바꿉니다.
   - *function-definition-version-arn*을 함수 정의 버전에서 복사한 LatestVersionArn으로 바꿉니다.
   - *subscription-definition-version-arn*을 구독 정의에 대해 복사한 LatestVersionArn으로 바꿉니다.

   ```
   aws greengrass create-group-version \
   --group-id group-id \
   --core-definition-version-arn core-definition-version-arn \
   --resource-definition-version-arn resource-definition-version-arn \
   --connector-definition-version-arn connector-definition-version-arn \
   --function-definition-version-arn function-definition-version-arn \
   --subscription-definition-version-arn subscription-definition-version-arn
   ```

4. 출력에서 Version 값을 복사합니다. 이것이 그룹 버전의 ID입니다. 이 값을 사용하여 그룹 버전을 배포합니다.

9단계: 배포 생성

코어 디바이스에 그룹을 배포합니다.

1. 코어 디바이스 터미널에서 AWS IoT Greengrass 데몬이 실행 중인지 확인합니다.

   a. 데몬이 실행 중인지 확인하려면:

      ```
      ps aux | grep -E 'greengrass.*daemon'
      ```

      출력에 root에 대한 /greengrass/ggc/packages/1.11.6/bin/daemon 입력이 포함되어 있는 경우에는 데몬이 실행 중인 것입니다.

b. 데몬을 시작하려면:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. 배포를 생성합니다.

- *group-id*를 해당 그룹에서 복사한 Id로 바꿉니다.
- *group-version-id*를 새 그룹 버전에서 복사한 Version으로 바꿉니다.

```
aws greengrass create-deployment \
--deployment-type NewDeployment \
--group-id group-id \
--group-version-id group-version-id
```

3. 출력에서 DeploymentId을 복사합니다.
4. 배포 상태를 가져옵니다.

- *group-id*를 해당 그룹에서 복사한 Id로 바꿉니다.
- *deployment-id*를 배포를 위해 복사한 DeploymentId로 바꿉니다.

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

상태가 Success이면, 배포에 성공한 것입니다. 문제 해결에 대한 도움말은 문제 해결 단원을 참조하십시오.

솔루션 테스트

1. AWS IoT 홈 페이지에서 테스트를 선택합니다.
2. 구독에서 다음 값을 사용한 다음 주제 구독을 선택합니다. Twilio Notifications 커넥터는 이 주제에 상태 정보를 게시합니다.

속성	값
구독 주제	twilio/message/status
MQTT 페이로드 디스플레이	페이로드를 문자열로 표시

3. 주제 게시에서 다음 값을 사용한 후 게시를 선택하여 함수를 간접 호출합니다.

속성	값
주제	온도/입력
메시지	*recipient-name* 은 이름으로, *recipient-phone-number* 는 문자 메시지 수신자의 전화번호로 바꿉니다. 예: +12345000000 `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` 평가판 계정을 사용하는 경우 인증된 전화번호 목록에 Twilio 수신자가 아닌 전화번호를 추가해야 합니다. 자세한 내용은 개인 전화번호 확인을 참조하십시오.

성공하면 수신자가 문자 메시지를 수신하고 콘솔에 출력 데이터의 `success` 상태가 표시됩니다.

이제 입력 메시지에서 `temperature`를 **29**로 변경하고 게시합니다. 이 값이 30보다 작기 때문에 TempMonitor 기능이 Twilio 메시지를 트리거하지 않습니다.

다음 사항도 참조하세요.

- 커넥터를 사용하여 서비스 및 프로토콜과 통합
- the section called “AWS에서 제공한 Greengrass 커넥터”
- the section called “커넥터 시작하기(콘솔)”
- AWS CLI 명령 참조에서 AWS Secrets Manager 명령
- AWS CLI명령 참조에서 AWS Identity and Access Management (IAM) 명령
- AWS CLI 명령 참조에서 AWS Lambda 명령
- AWS CLI 명령 참조에서 AWS IoT Greengrass 명령

Greengrass Discovery RESTful API

AWS IoT Greengrass와(과) 통신하는 모든 클라이언트 디바이스는 Greengrass 그룹의 멤버여야 합니다. 각 그룹에는 Greengrass 코어가 있어야 합니다. Discovery API를 사용하면 각 클라이언트 디바이스는 같은 Greengrass 그룹에 속한 Greengrass 코어에 연결하는 데 필요한 정보를 검색할 수 있습니다. 클라이언트 디바이스가 먼저 온라인 상태가 되면 AWS IoT Greengrass 서비스에 연결할 수 있으며 Discovery API를 사용하여 다음 사항들을 찾을 수도 있습니다.

- 디바이스가 속한 그룹. 한 클라이언트 디바이스는 최대 10개 그룹의 멤버일 수 있습니다.
- 해당 그룹 내 Greengrass 코어에 대한 IP 주소 및 포트.
- Greengrass 코어 디바이스를 인증하는 데 사용할 수 있는 해당 그룹 CA 인증서.

> **Note**
>
> 클라이언트 디바이스는 AWS IoT 디바이스 SDK를 사용하여 Greengrass 코어의 연결 정보를 검색할 수도 있습니다. 자세한 내용은 AWS IoT 디바이스 SDK 섹션을 참조하세요.

이 API를 사용하려면 Discovery API 엔드포인트에 HTTP 요청을 전송합니다. 예:

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

AWS IoT Greengrass 디스커버리 API에 지원되는 Amazon Web Services 리전 및 엔드포인트 목록은 AWS 일반 참조의 AWS IoT Greengrass 엔드포인트 및 할당량을 참조하십시오. 이것은 데이터 영역 전용 API입니다. 그룹 관리 및 AWS IoT Core 작업을 위한 엔드포인트는 Discovery API 엔드포인트와 다릅니다.

요청

다음 예제와 같이 요청은 표준 HTTP 헤더를 포함하고 있으며, Greengrass Discovery 엔드포인트로 전송됩니다.

포트 번호는 코어가 포트 8443 또는 포트 443을 통해 HTTPS 트래픽을 보내도록 구성되었는지 여부에 따라 다릅니다. 자세한 내용은 the section called “포트 443에서 또는 네트워크 프록시를 통해 연결” 섹션을 참조하세요.

포트 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/
thing/thing-name
```

포트 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/
thing/thing-name
```

포트 443에서 연결하는 클라이언트는 ALPN(Application Layer Protocol Negotiation) TLS 확장을 구현하고 `ProtocolNameList`에 `x-amzn-http-ca`를 `ProtocolName`으로 전달해야 합니다. 자세한 정보는 AWS IoT 개발자 안내서의 프로토콜을 참조하세요.

> **Note**
>
> 이 예제에서는 ATS 루트 CA 인증서에 사용되는 Amazon Trust Services(ATS) 엔드포인트를 사용합니다(권장). 엔드포인트는 CA 인증서 유형과 일치해야 합니다 자세한 내용은 the section called “서비스 엔드포인트는 인증서 유형과 일치해야 합니다” 섹션을 참조하세요.

응답

성공 시, 응답은 표준 HTTP 헤더와 다음 코드 및 본문을 포함합니다.

```
HTTP 200
BODY: response document
```

자세한 내용은 검색 응답 문서 예제 섹션을 참조하세요.

검색 권한

연결 정보를 검색하려면 호출자가 `greengrass:Discover` 작업을 수행할 수 있도록 허용하는 정책이 필요합니다. 클라이언트 인증서와의 TLS 상호 인증은 유일하게 허용된 인증 형식입니다. 다음은 호출자가 이러한 작업을 수행할 수 있도록 허용하는 정책 예제입니다.

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

```
        "Effect": "Allow",
        "Action": "greengrass:Discover",
        "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
     }]
}
```

검색 응답 문서 예제

다음 문서는 Greengrass 코어 엔드포인트 및 그룹 CA 인증서를 각각 하나씩 포함하는 그룹의 멤버에 속한 클라이언트 디바이스에 대한 응답을 보여줍니다.

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

다음 문서는 1개의 Greengrass 코어와 다수의 엔드포인트 및 그룹 CA 인증서를 포함하는 두 그룹의 멤버에 속한 클라이언트 디바이스에 대한 응답을 보여줍니다.

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
```

```
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-connection-1-description"
            },
            {
              "id": "core-01-connection-id-2",
              "hostAddress": "core-01-address-2",
              "portNumber": core-01-port-2,
              "metadata": "core-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "gg-group-02-id",
      "Cores": [
         {
            "thingArn":"core-02-thing-arn",
            "Connectivity" : [
            {
              "id": "core-02-connection-id",
              "hostAddress": "core-02-address",
              "portNumber": core-02-port,
              "metadata": "core-02-connection-1-description"
            }
            ],
            "CAs": [
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
            ]
        }
```

```
    ]
    }
}
```

> **Note**
>
> Greengrass 그룹에는 각각 정확히 1개의 Greengrass 코어가 정의되어야 합니다. Greengrass 코어 목록을 포함하는 AWS IoT Greengrass 서비스로부터의 모든 응답은 단 1개의 Greengrass 코어를 포함합니다.

cURL을 설치했을 경우 검색 요청을 테스트할 수 있습니다. 예:

```
$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-
ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":
[{"thingArn":"arn:aws:iot:us-west-2:1234567
89012:thing/MyFirstGroup_Core","Connectivity":
[{"Id":"AUTOIP_192.168.1.4_1","HostAddress":"192.168.1.5","PortNumber
":8883,"Metadata":""}]}],"CAs":["-----BEGIN CERTIFICATE-----\ncert-contents\n-----END
 CERTIFICATE-----\n"]}]}
```

AWS IoT Greengrass의 보안

AWS에서 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 매우 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드의 보안 – AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. 또한, AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램의 일환으로 보안 효과를 정기적으로 테스트하고 검증합니다. AWS IoT Greengrass에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램의 범위에 속하는 AWS 서비스 를 참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

AWS IoT Greengrass를 사용하면, 사용자도 디바이스, 로컬 네트워크 연결 및 프라이빗 키를 안전하게 보호할 책임이 있습니다.

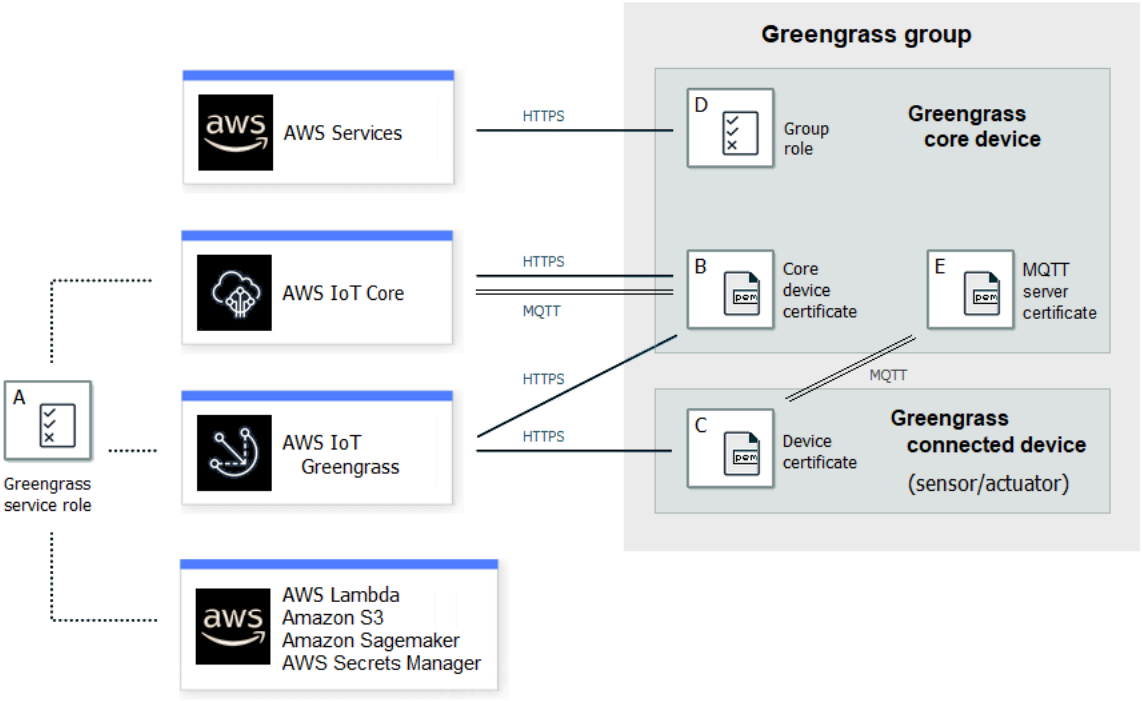
이 설명서는 AWS IoT Greengrass 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS IoT Greengrass를 구성하는 방법을 보여줍니다. 또한 AWS IoT Greengrass 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

주제

AWS IoT Greengrass 보안 개요

AWS IoT Greengrass 는 X.509 인증서, AWS IoT 정책, IAM 정책 및 역할을 사용하여 로컬 Greengrass 환경의 디바이스에서 실행되는 애플리케이션을 보호합니다.

다음 다이어그램은 보안 모델의 구성 요소를 보여줍니다. AWS IoT Greengrass



A - Greengrass 서비스 역할

AWS IoT Core AWS Lambda, 및 기타 서비스에서 AWS 리소스에 액세스할 AWS IoT Greengrass 때 위임되는 고객 생성 IAM 역할. AWS 자세한 설명은 the section called “Greengrass 서비스 역할” 섹션을 참조하세요.

B - 코어 디바이스 인증서

및 를 사용하여 Greengrass 코어를 인증하는 데 사용되는 X.509 인증서 AWS IoT Core AWS IoT Greengrass 자세한 설명은 the section called “디바이스 인증 및 권한 부여” 섹션을 참조하세요.

C - 디바이스 인증서

클라이언트 장치 (연결된 장치라고도 함) 를 인증하는 데 사용되는 X.509 인증서 (및) AWS IoT Core AWS IoT Greengrass 자세한 설명은 the section called “디바이스 인증 및 권한 부여” 섹션을 참조하세요.

D - 그룹 역할

Greengrass AWS 코어에서 서비스를 AWS IoT Greengrass 호출할 때 맡는 고객 생성 IAM 역할입니다.

이 역할을 사용하여 사용자 정의 Lambda 함수 및 커넥터가 DynamoDB와 같은 AWS 서비스에 액세스하는 데 필요한 액세스 권한을 지정합니다. 또한 이를 AWS IoT Greengrass 사용하여 스트림 관리자 스트림을 AWS 서비스로 내보내고 Logs에 쓸 수 있습니다. CloudWatch 자세한 설명은 the section called “Greengrass 그룹 역할” 섹션을 참조하세요.

> **Note**
>
> AWS IoT Greengrass Lambda 함수의 클라우드 버전에 지정된 AWS Lambda Lambda 실행 역할을 사용하지 않습니다.

E - MQTT 서버 인증서

Greengrass 코어 디바이스와 Greengrass 그룹의 연결된 디바이스 간 전송 계층 보안(TLS) 상호 인증에 사용되는 인증서입니다. 이 인증서는 AWS 클라우드 클라우드에 저장된 그룹 CA 인증서에 의해 서명됩니다.

디바이스 연결 워크플로

이 섹션에서는 클라이언트 디바이스를 AWS IoT Greengrass 서비스 및 Greengrass 코어 디바이스에 연결하는 방법을 설명합니다. 클라이언트 디바이스는 코어 AWS IoT Core 디바이스와 동일한 Greengrass 그룹에 속하는 등록된 디바이스입니다.

- Greengrass 코어 디바이스는 디바이스 인증서, 개인 키 및 AWS IoT Core 루트 CA 인증서를 사용하여 서비스에 연결합니다. AWS IoT Greengrass 코어 디바이스에서 구성 파일의 `crypto` 개체는 이러한 항목의 파일 경로를 지정합니다.
- Greengrass 코어 디바이스는 AWS IoT Greengrass 서비스에서 그룹 멤버십 정보를 다운로드합니다.
- Greengrass 코어 디바이스로 배포가 실행될 때 Device Certificate Manager(DCM)는 Greengrass 코어 디바이스에 대한 로컬 서버 인증서 관리를 처리합니다.
- 클라이언트 장치는 장치 인증서, 개인 키 및 AWS IoT Core 루트 CA 인증서를 사용하여 AWS IoT Greengrass 서비스에 연결합니다. 연결이 되면 클라이언트 디바이스는 Greengrass Discovery

Service를 사용해 Greengrass 코어 디바이스의 IP 주소를 찾습니다. 또한 클라이언트 디바이스는 Greengrass 코어 디바이스와의 TLS 상호 인증에 사용되는 그룹 CA 인증서를 다운로드합니다.

- 클라이언트 디바이스는 Greengrass 코어 디바이스와 연결을 시도하며 자체 디바이스 인증서와 클라이언트 ID를 전달합니다. 클라이언트 ID가 클라이언트 디바이스의 사물 이름과 일치하며 인증서가 유효한 경우(Greengrass 그룹의 일부), 연결이 이루어집니다. 그렇지 않으면 연결이 종료됩니다.

클라이언트 디바이스에 대한 AWS IoT 정책은 클라이언트 디바이스가 코어의 연결 정보를 검색할 수 있도록 허용하는 `greengrass:Discover` 권한을 부여해야 합니다. 이 정책 설명에 대한 자세한 내용은 the section called “검색 권한” 섹션을 참조하십시오.

AWS IoT Greengrass 보안 구성

Greengrass 애플리케이션의 보안을 구성하려면

1. Greengrass 코어 디바이스를 위한 AWS IoT Core 무언가를 만드세요.
2. Greengrass 코어 디바이스에 대한 키 페어와 디바이스 인증서를 생성합니다.
3. AWS IoT 정책을 생성해 해당 디바이스 인증서에 연결합니다. 인증서 및 정책은 Greengrass 코어 디바이스가 서비스에 액세스할 수 있도록 AWS IoT Core 허용합니다. AWS IoT Greengrass 자세한 설명은 코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

   > **Note**
   >
   > 핵심 장치에 대한 정책에서 사물 AWS IoT 정책 변수 (`iot:Connection.Thing.*`) 를 사용하는 것은 지원되지 않습니다. 코어는 동일한 장치 인증서를 사용하여 여러 번 AWS IoT Core 연결하지만 연결의 클라이언트 ID가 핵심 사물 이름과 정확히 일치하지 않을 수 있습니다.

4. Greengrass 서비스 역할을 생성합니다. 이 IAM 역할은 사용자를 AWS IoT Greengrass 대신하여 다른 AWS 서비스의 리소스에 액세스할 수 있는 권한을 부여합니다. AWS IoT Greengrass 이를 통해 AWS Lambda 기능 검색 및 디바이스 섀도 관리와 같은 필수 작업을 수행할 수 있습니다.

   AWS 리전s 전체에 동일한 서비스 역할을 사용할 수 있지만 사용하는 모든 AWS 계정 AWS 리전 위치에서 해당 서비스 역할이 사용자와 연결되어야 합니다. AWS IoT Greengrass
5. (선택 사항) Greengrass 그룹 역할을 생성합니다. 이 IAM 역할은 Greengrass 코어에서 실행되는 Lambda 함수 및 커넥터에 서비스를 호출할 수 있는 권한을 부여합니다. AWS 예를 들어 Kinesis Firehose 커넥터에는 Amazon Data Firehose 전송 스트림에 레코드를 쓸 수 있는 권한이 필요합니다.

Greengrass 그룹에는 하나의 역할만 연결할 수 있습니다.

6. Greengrass 코어에 연결되는 각 장치에 대해 AWS IoT Core 무언가를 만드세요.

> **Note**
>
> 기존 AWS IoT Core 사물과 인증서를 사용할 수도 있습니다.

7. Greengrass 코어에 연결되는 각 장치에 대한 장치 인증서, 키 페어 및 AWS IoT 정책을 생성합니다.

AWS IoT Greengrass 핵심 보안 원칙

Greengrass 코어는 AWS IoT 클라이언트, 로컬 MQTT 서버 및 로컬 비밀 관리자와 같은 보안 주체를 사용합니다. 이러한 보안 주체에 대한 구성은 `config.json` 구성 파일의 `crypto` 객체에 저장됩니다. 자세한 설명은 the section called “AWS IoT Greengrass 코어 구성 파일” 섹션을 참조하세요.

이 구성에는 인증 및 암호화를 위해 주체 구성 요소가 사용하는 프라이빗 키의 경로가 포함되어 있습니다. AWS IoT Greengrass 는 두 가지 프라이빗 키 스토리지 모드인 하드웨어 기반 또는 파일 시스템 기반(기본값)을 지원합니다. 하드웨어 보안 모듈에 키를 저장하는 방법에 대한 자세한 내용은 the section called “하드웨어 보안 통합” 단원을 참조하십시오.

AWS IoT 클라이언트

AWS IoT 클라이언트 (IoT 클라이언트) 는 Greengrass 코어와 Greengrass 코어 간의 인터넷 통신을 관리합니다. AWS IoT Core AWS IoT Greengrass 이 통신을 위한 TLS 연결을 설정할 때 상호 인증을 위해 공개 및 개인 키가 있는 X.509 인증서를 사용합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 X.509 인증서 및 AWS IoT Core을(를) 참조하십시오.

IoT 클라이언트는 RSA 및 EC 인증서와 키를 지원합니다. `config.json`에서 `IoTCertificate` 주체에 대한 인증서 및 프라이빗 키 경로가 지정되어 있습니다.

MQTT 서버

로컬 MQTT 서버는 로컬 네트워크를 통해 그룹 내 Greengrass 코어와 클라이언트 장치 간의 통신을 관리합니다. AWS IoT Greengrass 이 통신을 위한 TLS 연결을 설정할 때 상호 인증을 위해 공개 및 개인 키가 있는 X.509 인증서를 사용합니다.

기본적으로 RSA AWS IoT Greengrass 개인 키가 자동으로 생성됩니다. 코어에서 다른 프라이빗 키를 사용하도록 구성하려면 `config.json`에서 `MQTTServerCertificate` 주체에 대한 키 경로를 제공해야 합니다. 고객이 제공한 키를 교체할 책임은 사용자에게 있습니다.

프라이빗 키 지원

	RSA 키	EC 키
키 유형	Supported	Supported
키 파라미터	Minimum 2048-bit length	NIST P-256 or NIST P-384 curve
디스크 형식	PKCS#1, PKCS#8	SECG1, PKCS#8
최소 GGC 버전	• 기본 RSA 키 사용: 1.0 • RSA 키 지정: 1.7	• EC 키 지정: 1.9

프라이빗 키의 구성은 관련 프로세스를 결정합니다. Greengrass 코어에서 서버로 지원하는 암호 제품군 목록은 the section called “TLS 암호 그룹 지원” 단원을 참조하십시오.

프라이빗 키가 지정되지 않은 경우(기본값)

- AWS IoT Greengrass 순환 설정에 따라 키를 회전합니다.
- 코어는 인증서를 생성하는 데 사용되는 RSA 키를 생성합니다.
- MQTT 서버 인증서에는 RSA 퍼블릭 키 및 SHA-256 RSA 서명이 있습니다.

RSA 프라이빗 키가 지정된 경우(GGC v1.7 이상 필요)

- 키는 직접 교체해야 합니다.
- 코어는 지정된 키를 사용하여 인증서를 생성합니다.
- RSA 키의 최소 길이는 2048비트여야 합니다.
- MQTT 서버 인증서에는 RSA 퍼블릭 키 및 SHA-256 RSA 서명이 있습니다.

EC 프라이빗 키가 지정된 경우(GGC v1.9 이상 필요)

- 키는 직접 교체해야 합니다.
- 코어는 지정된 키를 사용하여 인증서를 생성합니다.
- EC 프라이빗 키는 NIST P-256 또는 NIST P-384 Curve를 사용해야 합니다.
- MQTT 서버 인증서에는 EC 퍼블릭 키 및 SHA-256 RSA 서명이 있습니다.

키 유형과 상관없이 코어에서 제공하는 MQTT 서버 인증서에는 SHA-256 RSA 서명이 있습니다. 이러한 이유 때문에 클라이언트는 코어와의 보안 연결을 설정하도록 SHA-256 RSA 인증서 유효성 검사를 지원해야 합니다.

보안 관리자

로컬 비밀 관리자는 사용자가 만든 암호의 로컬 사본을 안전하게 관리합니다. AWS Secrets Manager 보안을 암호화하는 데 사용되는 데이터 키의 보안을 유지하는 데 프라이빗 키를 사용합니다. 자세한 설명은 코어에 암호 배포 섹션을 참조하세요.

기본적으로 IoT 클라이언트 프라이빗 키가 사용되지만 `config.json`에서 `SecretsManager` 주체에 대해 다른 프라이빗 키를 지정할 수 있습니다. RSA 키 유형만 지원됩니다. 자세한 설명은 the section called “암호 암호화를 위한 프라이빗 키 지정” 섹션을 참조하세요.

> **Note**
>
> 현재는 하드웨어 기반 개인 키를 사용할 때 로컬 암호의 암호화 및 복호화를 위한 PKCS #1 v1.5 패딩 메커니즘만 AWS IoT Greengrass 지원합니다. 공급업체가 제공하는 지침에 따라 하드웨어 기반 개인 키를 수동으로 생성하려면 PKCS #1 v1.5를 선택해야 합니다. AWS IoT Greengrass OAEP (최적 비대칭 암호화 패딩) 는 지원하지 않습니다.

프라이빗 키 지원

	RSA 키	EC 키
키 유형	Supported	Not supported
키 파라미터	Minimum 2048-bit length	Not applicable
디스크 형식	PKCS#1, PKCS#8	Not applicable
최소 GGC 버전	1.7	Not applicable

MQTT 메시징 워크플우로의 관리형 구독

AWS IoT Greengrass 구독 테이블을 사용하여 Greengrass 그룹의 클라이언트 장치, 함수 및 커넥터 간에, 그리고 로컬 섀도우 서비스와 함께 AWS IoT Core MQTT 메시지를 교환하는 방법을 정의합니

다. 각 구독은 메시지를 보내거나 받는 소스, 대상 및 MQTT 주제 (또는 제목) 를 지정합니다. AWS IoT Greengrass 해당 구독이 정의된 경우에만 소스에서 대상으로 메시지를 보낼 수 있습니다.

구독은 소스에서 대상으로의 한 방향으로만 메시지 흐름을 정의합니다. 양방향 메시지 교환을 지원하려면 각 방향당 하나씩 총 두 개의 구독을 생성해야 합니다.

TLS 암호 그룹 지원

AWS IoT Greengrass AWS IoT Core 전송 보안 모델을 사용하여 TLS 암호 제품군을 사용하여 클라우드와의 통신을 암호화합니다. 또한 AWS IoT Greengrass 데이터는 유휴 상태일 때 (클라우드에) 암호화됩니다. AWS IoT Core 전송 보안 및 지원되는 암호 제품군에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 전송 보안을 참조하십시오.

로컬 네트워크 통신에 대한 지원되는 암호 제품군

반대로 AWS IoT Greengrass 코어는 AWS IoT Core 인증서 서명 알고리즘을 위한 다음과 같은 로컬 네트워크 TLS 암호 제품군을 지원합니다. 파일 시스템에 프라이빗 키가 저장되어 있으면 이러한 암호 제품군이 모두 지원됩니다. 코어가 HSM(하드웨어 보안 모듈)을 사용하도록 구성된 경우 하위 세트가 지원됩니다. 자세한 내용은 the section called “보안 주체” 및 the section called “하드웨어 보안 통합” 섹션을 참조하세요. 이 표에는 지원에 필요한 Core 소프트웨어의 최소 버전도 포함되어 있습니다 AWS IoT Greengrass .

	암호	HSM 지원	최소 GGC 버전
TLSv1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Supported	1.0

	암호	HSM 지원	최소 GGC 버전
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_128_GCM_SHA256	Not supported	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_256_GCM_SHA384	Not supported	1.0
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Supported	1.9
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Supported	1.9
TLSv1.1	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0

	암호	HSM 지원	최소 GGC 버전
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0
TLSv1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0

AWS IoT Greengrass의 데이터 보호

AWS 공동 책임 모델은 AWS IoT Greengrass의 데이터 보호에 적용됩니다. 이 모델에서 설명하는 것처럼 AWS은(는) 모든 AWS 클라우드을(를) 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하십시오. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정 보안 인증 정보를 보호하고 AWS IAM Identity Center 또는 AWS Identity and Access Management(IAM)을 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이 방식을 사용하면 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2가 필수이며 TLS 1.3을 권장합니다.

- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정합니다.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용합니다.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-2 섹션을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 AWS IoT Greengrass 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 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 보안 인증를 URL에 포함시켜서는 안 됩니다.

AWS IoT Greengrass에서 민감한 정보를 보호하는 방법에 대한 자세한 내용은 the section called “민감한 정보를 기록하지 않음” 섹션을 참조하세요.

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

주제

- 데이터 암호화
- 하드웨어 보안 통합

데이터 암호화

AWS IoT Greengrass는 암호화를 사용하여 전송 중(인터넷 또는 로컬 네트워크를 통해) 데이터와 유휴 상태의 데이터를 보호합니다(AWS 클라우드에 저장).

AWS IoT Greengrass 환경의 디바이스는 종종 추가 처리를 위해 AWS 서비스로 전송되는 데이터를 수집합니다. 다른 AWS 서비스의 데이터 암호화에 대한 자세한 내용은 해당 서비스의 보안 설명서를 참조하세요.

주제

- 전송 중 데이터 암호화

- 유휴 시 암호화
- Greengrass 코어 디바이스를 위한 키 관리

전송 중 데이터 암호화

AWS IoT Greengrass에서는 세 가지 통신 모드로 데이터를 전송할 수 있습니다.

- the section called “인터넷을 통해 전송 중인 데이터”. 인터넷을 통한 Greengrass 코어와 AWS IoT Greengrass 간 통신은 암호화됩니다.
- the section called “로컬 네트워크를 통해 전송 중인 데이터”. 로컬 네트워크를 통한 Greengrass 코어와 연결된 디바이스 간 통신은 암호화됩니다.
- the section called “코어 디바이스의 데이터”. Greengrass 코어 디바이스의 구성 요소 간 통신은 암호화되지 않습니다.

인터넷을 통해 전송 중인 데이터

AWS IoT Greengrass는 TLS(전송 계층 보안)를 사용하여 인터넷을 통한 모든 통신을 암호화합니다. AWS 클라우드로 전송되는 모든 데이터는 MQTT 또는 HTTPS 프로토콜을 사용하여 TLS 연결을 통해 전송되므로 기본적으로 안전합니다. AWS IoT Greengrass는 AWS IoT 전송 보안 모델을 사용합니다. 자세한 정보는 AWS IoT Core 개발자 안내서의 전송 보안을 참조하세요.

로컬 네트워크를 통해 전송 중인 데이터

AWS IoT Greengrass는 TLS를 사용하여 로컬 네트워크를 통한 Greengrass 코어와 클라이언트 디바이스 간 모든 통신을 암호화합니다. 자세한 내용은 로컬 네트워크 통신에 대한 지원되는 암호 제품군을 참조하십시오.

로컬 네트워크와 프라이빗 키를 보호하는 것은 사용자의 책임입니다.

Greengrass 코어 디바이스의 경우 다음과 같은 책임은 사용자에게 있습니다.

- 최신 보안 패치로 커널을 업데이트합니다.
- 최신 보안 패치로 시스템 라이브러리를 업데이트합니다.
- 프라이빗 키를 보호합니다. 자세한 내용은 the section called “키 관리” 섹션을 참조하세요.

클라이언트 디바이스의 경우 다음 책임은 사용자에게 있습니다.

- TLS 스택을 최신 상태로 유지합니다.
- 프라이빗 키를 보호합니다.

코어 디바이스의 데이터

데이터가 디바이스를 떠나지 않기 때문에 AWS IoT Greengrass는 Greengrass 코어 디바이스에서 로컬로 교환되는 데이터를 암호화하지 않습니다. 여기에는 사용자 정의 Lambda 함수, 커넥터, AWS IoT Greengrass 코어 SDK, 스트림 관리자와 같은 시스템 구성 요소 간의 통신이 포함됩니다.

유휴 시 암호화

AWS IoT Greengrass가 데이터를 저장합니다.

- the section called “AWS 클라우드에 저장된 데이터”. 이 데이터는 암호화됩니다.
- the section called “Greengrass 코어에 저장된 데이터”. 이 데이터는 암호화되지 않습니다(암호의 로컬 사본 제외).

AWS 클라우드에 저장된 데이터

AWS IoT Greengrass는 AWS 클라우드에 저장된 고객 데이터를 암호화합니다. 이 데이터는 AWS IoT Greengrass에서 관리하는 AWS KMS 키를 사용하여 보호됩니다.

Greengrass 코어에 저장된 데이터

AWS IoT Greengrass는 Unix 파일 권한 및 전체 디스크 암호화(활성화된 경우)를 사용하여 코어에 저장된 유휴 상태의 데이터를 보호합니다. 파일 시스템 및 디바이스의 보안을 유지하는 것은 사용자의 책임입니다.

그러나 AWS IoT Greengrass는 AWS Secrets Manager에서 검색한 암호의 로컬 복사본을 암호화합니다. 자세한 내용은 the section called “암호 암호화” 섹션을 참조하세요.

Greengrass 코어 디바이스를 위한 키 관리

Greengrass 코어 디바이스에 암호화(퍼블릭 및 프라이빗) 키를 안전하게 저장하는 것은 고객의 책임입니다. AWS IoT Greengrass는 다음과 같은 시나리오에 퍼블릭 키와 프라이빗 키를 사용합니다.

- IoT 클라이언트 키는 IoT 인증서와 함께 사용되어 Greengrass 코어가 AWS IoT Core에 연결될 때 TLS(전송 계층 보안) 핸드셰이크를 인증합니다. 자세한 내용은 the section called “디바이스 인증 및 권한 부여” 섹션을 참조하세요.

> **Note**
>
> 키와 인증서를 코어 프라이빗 키와 코어 디바이스 인증서라고도 합니다.

- MQTT 서버 키는 MQTT 서버 인증서가 코어 및 클라이언트 디바이스 간의 TLS 연결을 인증하는 데 사용됩니다. 자세한 내용은 the section called “디바이스 인증 및 권한 부여” 섹션을 참조하세요.
- 로컬 암호 관리자는 IoT 클라이언트 키를 사용하여 로컬 보안 암호를 암호화하는 데 사용되는 데이터 키를 보호하지만 사용자 고유의 프라이빗 키를 제공할 수도 있습니다. 자세한 내용은 the section called “암호 암호화” 섹션을 참조하세요.

Greengrass 코어는 파일 시스템 권한, 하드웨어 보안 모듈 또는 둘 다 사용하여 프라이빗 키 스토리지를 지원합니다. 파일 시스템 기반 프라이빗 키를 사용하는 경우, 코어 디바이스의 보안 스토리지에 대한 책임은 사용자에게 있습니다.

Greengrass 코어에서 프라이빗 키의 위치는 `config.json` 파일의 `crypto` 섹션에 지정됩니다. 고객이 제공한 키를 MQTT 서버 인증서에 사용하도록 코어를 구성하는 경우, 키를 교체하는 것은 사용자의 책임입니다. 자세한 내용은 the section called “보안 주체” 섹션을 참조하세요.

클라이언트 디바이스의 경우 TLS 스택을 최신 상태로 유지하고 프라이빗 키를 보호하는 것은 사용자의 책임입니다. 프라이빗 키는 AWS IoT Greengrass 서비스와 TLS 연결을 인증하기 위해 디바이스 인증서와 함께 사용됩니다.

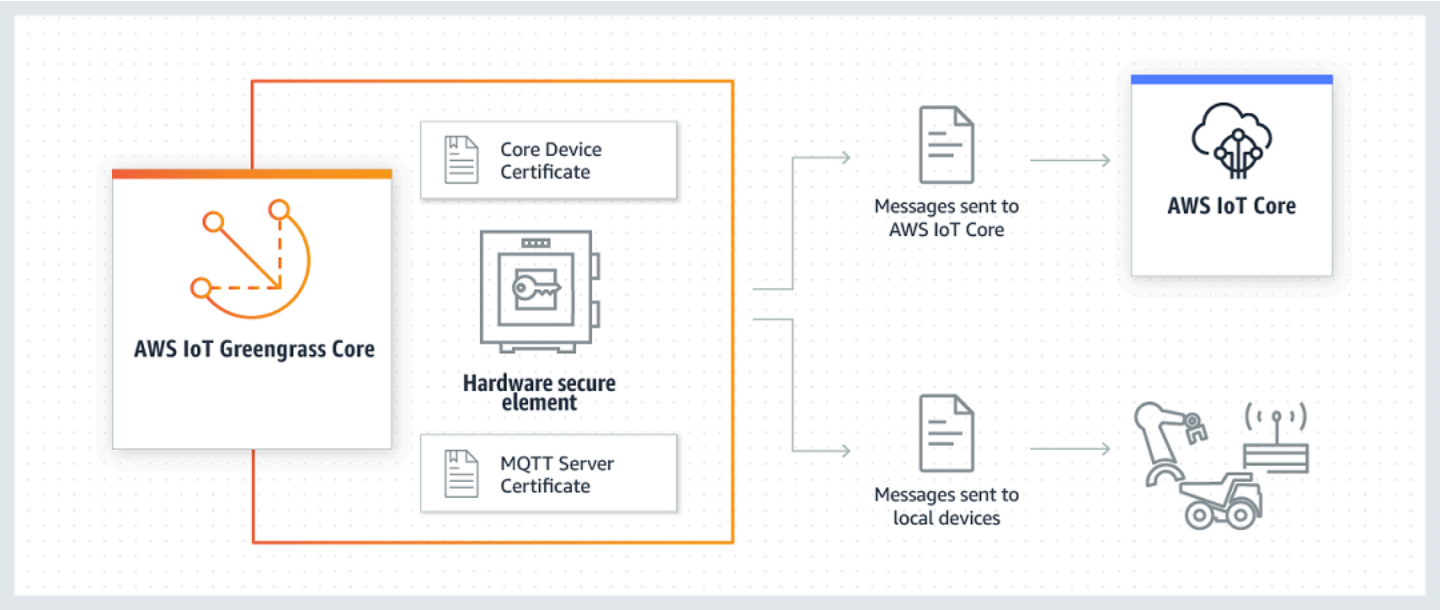
하드웨어 보안 통합

이 기능은 AWS IoT Greengrass 코어 v1.7 이상에 사용할 수 있습니다.

AWS IoT Greengrass에서는 안전한 저장과 프라이빗 키의 오프로딩을 위해 PKCS#11 인터페이스를 통한 HSM(하드웨어 보안 모듈)의 사용을 지원합니다. 그러면 키가 노출되거나 소프트웨어에서 중복되는 경우를 방지할 수 있습니다. 프라이빗 키는 HSM, TPM(Trusted Platform Module) 또는 기타 암호화 요소 등과 같은 하드웨어 모듈에 안전하게 저장할 수 있습니다.

AWS Partner 디바이스 카탈로그에서 이 기능에 적합한 디바이스를 검색하십시오.

다음 다이어그램은 AWS IoT Greengrass 코어의 하드웨어 보안 아키텍처를 나타냅니다.



표준 설치에서 AWS IoT Greengrass는 프라이빗 키 두 개를 사용합니다. 키 하나는 Greengrass 코어가 AWS IoT Core에 연결할 때 TLS(전송 계층 보안) 핸드셰이크 중 AWS IoT 클라이언트(IoT 클라이언트) 구성 요소가 사용합니다. 이 키는 코어 프라이빗 키라고도 합니다. 다른 키는 로컬 MQTT 서버에서 사용하는 데, Greengrass 디바이스가 Greengrass 코어와 통신하도록 합니다. 두 구성 요소에 하드웨어 보안을 사용하려는 경우 공유 프라이빗 키 또는 개별 프라이빗 키를 사용할 수 있습니다. 자세한 내용은 the section called “ 프로비저닝 연습” 섹션을 참조하세요.

> **Note**
>
> 표준 설치 시 로컬 보안 관리자도 해당 암호화 프로세스에 IoT 클라이언트 키를 사용하지만 사용자가 자신의 프라이빗 키를 사용할 수 있습니다. 이는 최소 2048비트 길이의 RSA 키여야 합니다. 자세한 내용은 the section called “암호 암호화를 위한 프라이빗 키 지정” 섹션을 참조하세요.

요구 사항

Greengrass 코어에 대한 하드웨어 보안을 구성하려면 다음 항목이 있어야 합니다.

- IoT 클라이언트, 로컬 MQTT 서버 및 로컬 보안 관리자 구성 요소에 대한 대상 프라이빗 키 구성을 지원하는 HSM(하드웨어 보안 모듈). 구성에는 구성 요소에서 키를 공유하도록 구성했는지 여부에 따라 하드웨어 기반 프라이빗 키가 한 개, 두 개 또는 세 개 포함될 수 있습니다. 프라이빗 키 지원에 대한 자세한 내용은 the section called “보안 주체” 단원을 참조하십시오.
  - RSA 키의 경우: RSA-2048 키 크기(또는 이상) 및 PKCS#1 v1.5 서명 체계

- EC 키의 경우: NIST P-256 또는 NIST P-384 Curve

> **Note**
>
> AWS Partner 디바이스 카탈로그에서 이 기능에 적합한 디바이스를 검색하십시오.

- 런타임에 로드할 수 있고(libdl 사용) PKCS#11을 제공하는 PKCS#11 공급자 라이브러리.
- 하드웨어 모듈은 PKCS#11 사양에 정의된 것처럼 슬롯 레이블로 확인할 수 있어야 합니다.
- 프라이빗 키는 공급업체에서 제공하는 프로비저닝 도구를 사용하여 HSM에서 생성 및 로드해야 합니다.
- 프라이빗 키는 객체 레이블로 확인할 수 있어야 합니다.
- 코어 디바이스 인증서. 프라이빗 키에 해당하는 IoT 클라이언트 인증서입니다.
- Greengrass OTA 업데이트 에이전트를 사용하는 경우 OpenSSL libp11 PKCS#11 래퍼 라이브러리 설치가 필요합니다. 자세한 내용은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.

또한 다음 조건을 충족하는지 확인하십시오.

- 프라이빗 키와 연결된 IoT 클라이언트 인증서가 AWS IoT에 등록되었고 활성화되어 있어야 합니다. AWS IoT 콘솔의 관리에서 모든 디바이스를 확장하고, 사물을 선택하고, 핵심 사물에 대한 인증서 탭을 선택하여 이를 확인할 수 있습니다.
- 시작하기 튜토리얼의 모듈 2에서 설명한 대로 AWS IoT Greengrass Core 소프트웨어 v1.7 이상이 코어 디바이스에 설치되어 있습니다. 버전 1.9 이상의 경우 MQTT 서버에 대해 EC 키를 사용해야 합니다.
- 인증서가 Greengrass 코어에 연결되어 있어야 합니다. 관리 페이지에서 AWS IoT 콘솔의 코어 사물에 대해 이러한 조건이 충족되었는지 확인할 수 있습니다.

> **Note**
>
> 현재 AWS IoT Greengrass에서는 HSM에서의 CA 인증서 또는 IoT 클라이언트 인증서 직접 로딩을 지원하지 않습니다. 인증서는 파일 시스템에서 Greengrass가 읽을 수 있는 위치에 일반 텍스트 파일로 로드해야 합니다.

AWS IoT Greengrass 코어에 대한 하드웨어 보안 구성

하드웨어 보안은 Greengrass 구성 파일에 구성됩니다. 이 config.json 파일은 */greengrass-root*/config 디렉터리에 위치합니다.

> **Note**
>
> 일반 소프트웨어 구현을 사용하여 HSM 구성을 설정하는 프로세스를 살펴보려면 the section called “모듈 7: 하드웨어 보안 통합 시뮬레이션”단원을 참조하십시오.
>
> > **Important**
> >
> > 이 예제의 시뮬레이션된 구성에서는 어떠한 보안 이점도 제공하지 않습니다. 이 모듈은 PKCS#11 사양에 대해 배우고, 앞으로 실제 하드웨어 기반 HSM을 사용할 계획이 있는 경우 소프트웨어의 초기 테스트를 수행하도록 마련되었습니다.

AWS IoT Greengrass에서 하드웨어 보안을 구성하려면 config.json에서 crypto 객체를 편집합니다.

하드웨어 보안을 사용하는 경우 다음 예제에 나와 있는 것처럼 crypto 객체가 PKCS#11 공급자 라이브러리의 인증서, 프라이빗 키 및 자산 경로를 지정하는 데 사용됩니다.

```
"crypto": {
  "PKCS11" : {
    "OpenSSLEngine" : "/path-to-p11-openssl-engine",
    "P11Provider" : "/path-to-pkcs11-provider-so",
    "slotLabel" : "crypto-token-name",
    "slotUserPin" : "crypto-token-user-pin"
  },
  "principals" : {
    "IoTCertificate" : {
      "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
      "certificatePath" : "file:///path-to-core-device-certificate"
    },
    "MQTTServerCertificate" : {
      "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
    },
    "SecretsManager" : {
      "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
```

```
    }
  },
  "caPath" : "file:///path-to-root-ca"
```

crypto 객체는 다음 속성을 포함하고 있습니다.

필드	설명	주의
caPath	AWS IoT 루트 CA의 절대 경로입니다.	file:///*absolute/path/to/file* 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
PKCS11		
OpenSSLEngine	선택 사항. OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진 .so 파일에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다. 하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 내용은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.
P11Provider	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
slotLabel	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.

필드	설명	주의
`slotUserPin`	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertificate .privateKeyPath`	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.
`MQTTServerCertificate`	선택 사항. 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	

필드	설명	주의
`MQTTServerCertificate .privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

필드	설명	주의
`caPath`	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
`PKCS11`		
`OpenSSLEngine`	선택 사항. OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진 `.so` 파일에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다. 하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 내용은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.
`P11Provider`	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
`slotLabel`	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
`slotUserPin`	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
`principals`		

필드	설명	주의
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertificate .privateKeyPath`	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.
`MQTTServerCertificate`	선택 사항. 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	
`MQTTServerCertificate .privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.

필드	설명	주의
`secretsmanager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

필드	설명	주의
`caPath`	AWS IoT 루트 CA의 절대 경로입니다.	`file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. **Note** 엔드포인트는 해당 인증서 유형과 일치해야 합니다.
`PKCS11`		
`OpenSSLEngine`	선택 사항. OpenSSL에 대한 PKCS#11 지원을 가능하게 하는 OpenSSL 엔진 `.so` 파일에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.

필드	설명	주의
		하드웨어 보안과 함께 Greengrass OTA 업데이트 에이전트를 사용하는 경우 이 속성이 필요합니다. 자세한 내용은 the section called “OTA 업데이트 구성” 섹션을 참조하세요.
`P11Provider`	PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.	파일 시스템에서는 파일에 대한 경로여야 합니다.
`slotLabel`	하드웨어 모듈을 식별하는 데 사용되는 슬롯 모듈입니다.	PKCS#11 레이블 사양을 준수해야 합니다.
`slotUserPin`	모듈에 대해 Greengrass 코어를 인증하는 데 사용되는 사용자 핀입니다.	구성된 프라이빗 키를 사용하여 C_Sign을 수행하려면 충분한 권한이 있어야 합니다.
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertificate .privateKeyPath`	코어 프라이빗 키의 경로입니다.	파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다.
`IoTCertificate .certificatePath`	코어 디바이스 인증서의 절대 경로입니다.	`file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다.

필드	설명	주의
`MQTTServerCertificate`	선택 사항. 코어가 MQTT 서버 또는 게이트웨이로 작동하기 위해 인증서와 함께 사용하는 프라이빗 키입니다.	
`MQTTServerCertificate.privateKeyPath`	로컬 MQTT 서버 프라이빗 키의 경로입니다.	이 값은 로컬 MQTT 서버에 대한 고유한 프라이빗 키를 지정하는 데 사용합니다. 파일 시스템 스토리지의 경우 `file:///`*`absolute/path/to/file`* 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. 이 속성을 생략하면 AWS IoT Greengrass에서는 교체 설정을 기반으로 키를 교체합니다. 지정된 경우, 고객이 키 교체를 책임집니다.
`secretsmanager`	The private key that secures the data key used for encryption. For more information, see 코어에 암호 배포.	

필드	설명	주의
`SecretsManager .privateKeyPath`	로컬 보안 관리자 프라이빗 키의 경로입니다.	RSA 키만 지원됩니다. 파일 시스템 스토리지의 경우 `file:///absolute/path/to/file` 형식의 파일 URI여야 합니다. HSM 스토리지의 경우 객체 레이블을 지정하는 RFC 7512 PKCS#11 경로여야 합니다. PKCS#1 v1.5 패딩 메커니즘을 사용하여 프라이빗 키를 생성해야 합니다.

AWS IoT Greengrass 하드웨어 보안에 대한 프로비저닝 연습

다음은 보안 및 성능 관련 프로비저닝 연습입니다.

보안

- 내부 하드웨어 난수 생성기를 사용하여 HSM에서 직접 프라이빗 키를 생성합니다.

  > **Note**
  >
  > 프라이빗 키를 구성해 이 기능을 사용할 경우(하드웨어 공급업체 제공 지침을 준수함), 현재 AWS IoT Greengrass에서는 로컬 암호 암호화 및 복호화에 대해 PKCS1 v1.5 패딩 메커니즘만 지원한다는 점에 유의하십시오. AWS IoT Greengrass에서는 OAEP(Optimal Asymmetric Encryption Padding)를 지원하지 않습니다.

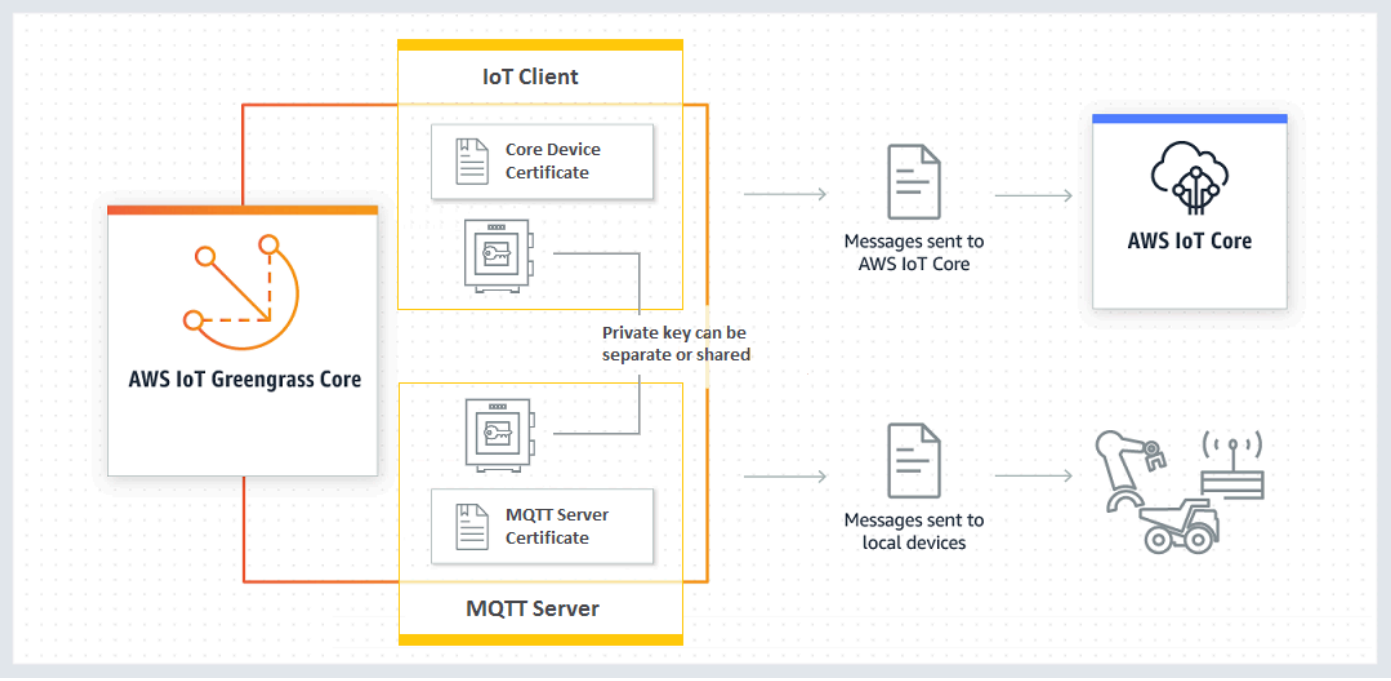
- 내보내기를 금지하도록 프라이빗 키를 구성합니다.
- 하드웨어 공급업체에서 제공한 프로비저닝 도구를 사용해 하드웨어 보호 프라이빗 키로 CSR(인증서 서명 요청)을 생성한 후 AWS IoT 콘솔을 사용해 클라이언트 인증서를 생성합니다.

> **Note**
>
> 프라이빗 키가 HSM에서 생성되는 경우에는 키 교체 연습이 적용되지 않습니다.

성능

다음 다이어그램에는 AWS IoT Greengrass 코어의 IoT 클라이언트 구성 요소와 로컬 MQTT 서버가 나와 있습니다. 두 구성 요소에 HSM 구성을 사용하려는 경우 동일한 프라이빗 키 또는 개별 프라이빗 키를 사용할 수 있습니다. 개별 키를 사용하는 경우 해당 키는 동일한 슬롯에 저장되어야 합니다.

> **Note**
>
> AWS IoT Greengrass에서는 HSM에 저장할 키 개수에 대한 한도를 지정하지 않으므로 IoT 클라이언트, MQTT 서버 및 보안 관리자 구성 요소에 대한 프라이빗 키를 저장할 수 있습니다. 그러나 슬롯에 저장할 수 있는 키 개수에 대한 한도를 지정하는 HSM 공급업체가 있을 수 있습니다.



AWS IoT Greengrass 코어 소프트웨어가 클라우드에 대해 연결을 오래 유지하기 때문에 일반적으로 IoT 클라이언트 키는 자주 사용되지 않습니다. 그러나 MQTT 서버 키는 Greengrass 디바이스가 코어에 연결할 때마다 사용됩니다. 이러한 상호 작용은 성능에 직접적인 영향을 미칩니다.

MQTT 서버 키가 HSM에 저장되어 있는 경우 디바이스가 연결할 수 있는 속도는 HSM이 수행할 수 있는 초당 RSA 서명 작업 수에 따라 달라집니다. 예를 들어, HSM이 RSA-2048 프라이빗 키에 대해 RSASSA-PKCS1-v1.5 서명을 수행하는 데 300밀리초가 걸리는 경우 1초에 Greengrass에는 디바이스 3개만 연결할 수 있습니다. 연결되면 HSM은 더 이상 사용되지 않고 표준 AWS IoT Greengrass 할당이 적용됩니다.

성능 병목 현상을 완화하기 위해 MQTT 서버에 대한 프라이빗 키를 HSM이 아니라 파일 시스템에 저장할 수 있습니다. 이러한 구성을 사용하면 MQTT 서버는 마치 하드웨어 보안이 활성화되지 않은 것처럼 작동합니다.

AWS IoT Greengrass에서는 IoT 클라이언트 및 MQTT 서버 구성 요소에 대한 여러 키 스토리지 구성을 지원하므로 보안 및 성능 요구 사항을 최적화할 수 있습니다. 다음 표에는 구성의 예가 포함되어 있습니다.

구성	IoT 키	MQTT 키	성능
HSM 공유 키	HSM: 키 A	HSM: 키 A	HSM 또는 CPU에서 제한됨
HSM 개별 키	HSM: 키 A	HSM: 키 B	HSM 또는 CPU에서 제한됨
IoT용 HSM만	HSM: 키 A	파일 시스템: 키 B	CPU에서 제한됨
레거시	파일 시스템: 키 A	파일 시스템: 키 B	CPU에서 제한됨

MQTT 서버에 파일 시스템 기반 키를 사용하도록 Greengrass 코어를 구성하려면 `config.json`에서 `principals.MQTTServerCertificate` 섹션을 생략합니다(또는 AWS IoT Greengrass에서 생성한 기본 키를 사용하지 않는 경우 키의 파일 기반 경로 지정). 결과로 나온 `crypto` 객체는 다음과 같습니다.

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
```

```
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

하드웨어 보안 통합을 위해 지원되는 암호 제품군

AWS IoT Greengrass에서는 코어가 하드웨어 보안에 맞게 구성된 경우 일련의 암호 제품군을 지원합니다. 이는 코어가 파일 기반 보안을 사용하도록 구성된 경우 지원되는 암호 제품군의 하위 세트입니다. 자세한 내용은 the section called “TLS 암호 그룹 지원” 섹션을 참조하세요.

> **Note**
>
> 로컬 네트워크를 통해 Greengrass 디바이스에서 Greengrass 코어에 연결할 때 지원되는 암호 제품군 중 하나를 사용하여 TLS 연결을 설정해야 합니다.

무선 업데이트에 대한 지원 구성

하드웨어 보안을 사용할 때 AWS IoT Greengrass 코어 소프트웨어의 OTA(무선) 업데이트를 활성화하려면 OpenSC libp11 PKCS#11 래퍼 라이브러리를 설치하고 Greengrass 구성 파일을 편집해야 합니다. OTA 업데이트에 대한 자세한 내용은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트단원을 참조하십시오.

1. Greengrass 데몬을 중지합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

2. OpenSSL 엔진을 설치합니다. OpenSSL 1.0 또는 1.1이 지원됩니다.

```
sudo apt-get install libengine-pkcs11-openssl
```

3. 시스템에서 OpenSSL 엔진(libpkcs11.so) 경로를 찾습니다.

   a. 이 라이브러리에 대해 설치된 패키지 목록을 가져옵니다.

   ```
   sudo dpkg -L libengine-pkcs11-openssl
   ```

   libpkcs11.so 파일은 engines 디렉터리에 있습니다.

   b. 파일의 전체 경로를 복사합니다(예: /usr/lib/ssl/engines/libpkcs11.so).

4. Greengrass 구성 파일을 엽니다. config.json 파일은 */greengrass-root*/config 디렉터리에 있습니다.

5. OpenSSLEngine 속성에 libpkcs11.so 파일 경로를 입력합니다.

```
{
 "crypto": {
   "caPath" : "file:///path-to-root-ca",
   "PKCS11" : {
     "OpenSSLEngine" : "/path-to-p11-openssl-engine",
     "P11Provider" : "/path-to-pkcs11-provider-so",
     "slotLabel" : "crypto-token-name",
     "slotUserPin" : "crypto-token-user-pin"
    },
    ...
  }
  ...
}
```

> **Note**
>
> PKCS11 객체에 OpenSSLEngine 속성이 없으면 해당 속성을 추가합니다.

6. Greengrass 데몬을 시작합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd start
```

AWS IoT Greengrass 코어 소프트웨어의 이전 버전과의 호환성

하드웨어 보안 지원 기능이 있는 AWS IoT Greengrass 코어 소프트웨어는 v1.6 이전에 대해 생성된 config.json 파일과 완벽하게 호환됩니다. crypto 객체가 config.json 구성 파일에 없으면 AWS IoT Greengrass에서는 파일 기반 coreThing.certPath, coreThing.keyPath 및 coreThing.caPath 속성을 사용합니다. 이러한 이전 버전과의 호환성은 Greengrass OTA 업데이트에 적용되고, config.json에 지정된 파일 기반 구성은 덮어쓰지 않습니다.

PKCS#11 지원 기능이 없는 하드웨어

PKCS#11 라이브러리는 일반적으로 하드웨어 공급업체가 제공하고 오픈 소스입니다. 예를 들어, 표준 호환 하드웨어(예: TPM1.2)를 사용하면 기존 오픈 소스 소프트웨어를 사용할 수 있습니다. 그러나 하드웨어에 해당하는 PKCS#11 라이브러리 구현이 없는 경우 또는 사용자 지정 PKCS#11 공급자를 쓰려는 경우 통합과 관련해 궁금한 점이 있으면 AWS Enterprise Support 담당자에게 문의해야 합니다.

다음 사항도 참조하세요.

- PKCS #11 암호화 토큰 인터페이스 사용 설명서 버전 2.40. John Leiseboer 및 Robert Griffin 편집. 2014년 11월 16일 OASIS Committee Note 02. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. 최신 버전: http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
- RFC 7512
- PKCS #1: RSA 암호화 버전1.5

AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여

AWS IoT Greengrass 환경의 디바이스는 인증에 X.509 인증서를 사용하고 권한 부여에는 AWS IoT 정책을 사용합니다. 인증서 및 정책을 통해 디바이스끼리, 그리고 AWS IoT Core 및 AWS IoT Greengrass와 안전하게 연결할 수 있습니다.

X.509 인증서는 X.509 퍼블릭 키 인프라 표준을 사용하여 퍼블릭 키를 인증서에 포함된 자격 증명과 연결하는 디지털 인증서입니다. X.509 인증서는 인증 기관(CA)이라고 부르는, 신뢰할 수 있는 엔터티가 발행합니다. CA는 X.509 인증서 발행하는 데 사용되는 CA 인증서라고 하는 하나 이상의 특수 인증서를 유지 관리합니다. 인증 기관만 CA 인증서에 액세스할 수 있습니다.

AWS IoT 정책은 AWS IoT 디바이스에 허용되는 작업 집합을 정의합니다. 특히 MQTT 메시지 게시 및 디바이스 섀도우 검색과 같은 AWS IoT Core 및 AWS IoT Greengrass 데이터 플레인 작업에 대한 액세스를 허용하거나 거부합니다.

모든 디바이스에는 AWS IoT Core 레지스트리의 항목, AWS IoT 정책이 연결된 활성화된 X.509 인증서가 필요합니다. 디바이스는 두 가지 범주로 나뉩니다.

- Greengrass 코어. 코어 디바이스는 인증서와 AWS IoT 정책을 사용하여 AWS IoT Core에 안전하게 연결됩니다. 또한 인증서와 정책은 AWS IoT Greengrass가 구성 정보, Lambda 함수, 커넥터 및 관리형 구독을 코어 디바이스에 배포할 수 있도록 합니다.
- 클라이언트 디바이스. 클라이언트 디바이스(연결된 디바이스, Greengrass 디바이스 또는 디바이스라고도 함)는 MQTT를 통해 Greengrass 코어에 연결하는 디바이스입니다. 인증서와 정책을 사용하여 AWS IoT Core와 AWS IoT Greengrass 서비스에 연결합니다. 이렇게 하면 클라이언트 디바이스는 AWS IoT Greengrass Discovery Service를 사용하여 코어 디바이스를 찾아 연결할 수 있습니다. 클라이언트 디바이스는 AWS IoT Core 디바이스 게이트웨이 및 코어 디바이스를 연결하는 데 동일한 인증서를 사용합니다. 또한 클라이언트 디바이스는 코어 디바이스와의 상호 인증을 위해 검색 정보를 사용합니다. 자세한 내용은 the section called “디바이스 연결 워크플로” 및 the section called “Greengrass 코어를 사용한 디바이스 인증 관리” 섹션을 참조하세요.

X.509 인증서

코어 디바이스와 클라이언트 디바이스 간의 통신과 디바이스 간, 그리고 AWS IoT Core 또는 AWS IoT Greengrass와의 통신은 반드시 인증되어야 합니다. 이 상호 인증은 등록된 X.509 디바이스 인증서와 암호화 키를 기반으로 수행됩니다.

AWS IoT Greengrass 환경에서 디바이스는 다음 TLS(전송 계층 보안) 연결에 퍼블릭 키와 프라이빗 키가 있는 인증서를 사용합니다.

- 인터넷을 통해 AWS IoT Core와 AWS IoT Greengrass에 연결하는 Greengrass 코어의 AWS IoT 클라이언트 구성 요소.
- 인터넷을 통해 코어 검색 정보를 얻기 위해 AWS IoT Greengrass에 연결되어 있는 클라이언트 연결 디바이스.
- 로컬 네트워크를 통해 그룹의 클라이언트 디바이스에 연결되어 있는 Greengrass 코어의 MQTT 서버 구성 요소.

AWS IoT Greengrass 코어 디바이스는 두 위치에 인증서를 저장합니다.

- */greengrass-root*/certs의 코어 디바이스 인증서. 일반적으로 코어 디바이스 인증서의 이름은 *hash*.cert.pem(예: 86c84488a5.cert.pem)입니다. 이 인증서는 코어가 AWS IoT Core 및 AWS IoT Greengrass 서비스에 연결될 때 AWS IoT 클라이언트가 상호 인증을 위해 사용합니다.
- */greengrass-root*/ggc/var/state/server의 MQTT 서버 인증서. MQTT 서버 인증서의 이름이 server.crt로 지정됩니다. 이 인증서는 로컬 MQTT 서버(Greengrass 코어에 있음) 사이의 상호 인증에 사용됩니다.

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

자세한 설명은 the section called “보안 주체” 섹션을 참조하세요.

CA(인증 기관) 인증서

코어 디바이스와 클라이언트 연결 디바이스는 AWS IoT Core 및 AWS IoT Greengrass 서비스 인증에 사용되는 루트 CA 인증서를 다운로드합니다. Amazon Root CA 1 같은 Amazon Trust Services(ATS) 루트 CA 인증서를 사용하는 것이 좋습니다. 서버 인증용 CA 인증서에 대한 자세한 내용은 AWS IoT Core개발자 설명서의 인증(IoT)를 참조하세요.

> **Note**
>
> 루트 CA 인증서 유형은 엔드포인트와 일치해야 합니다. ATS 엔드포인트가 있는 ATS 루트 CA 인증서 (권장) 또는 레거시 엔드포인트가 있는 VeriSign 루트 CA 인증서를 사용합니다. 일부 Amazon Web Services 리전만 레거시 엔드포인트를 지원합니다. 자세한 설명은 the section called “서비스 엔드포인트는 인증서 유형과 일치해야 합니다” 섹션을 참조하세요.

클라이언트 디바이스는 Greengrass 그룹 CA 인증서도 다운로드합니다. 상호 인증 중에 Greengrass 코어에서 MQTT 서버 인증서를 검증하는 데 사용됩니다. 자세한 설명은 the section called “디바이스 연결 워크플로” 섹션을 참조하세요. MQTT 서버 인증서의 기본 만료 기간은 7일입니다.

로컬 MQTT 서버에서의 인증서 교체

클라이언트 디바이스는 Greengrass 코어 디바이스와의 상호 인증을 위해 로컬 MQTT 서버 인증서를 사용합니다. 이 인증서는 기본적으로 7일이 경과하면 만료됩니다. 이 제한된 기간은 보안 모범 사례를 기반으로 합니다. MQTT 서버 인증서는 클라우드에 저장된 그룹 CA 인증서에 의해 서명됩니다.

Greengrass 코어 디바이스가 온라인 상태 및 지속적으로 AWS IoT Greengrass 서비스에 직접적으로 액세스가 가능해야 인증서 교체를 할 수 있습니다. 인증서가 만료되면 코어 디바이스는 새 인증서를 획득하기 위해 AWS IoT Greengrass 서비스와의 연결을 시도합니다. 연결이 성공하면 코어 디바이스는 새 MQTT 서버 인증서를 다운로드하고 로컬 MQTT 서비스를 다시 시작합니다. 이 시점에서 코어와 연결된 모든 클라이언트 디바이스는 연결이 해제됩니다. 코어 디바이스가 만료 시 오프라인이면 대체 인증서를 받지 않습니다. 코어 디바이스에 시도되는 모든 연결은 거절됩니다. 기존 연결은 영향을 받지 않습니다. 클라이언트 디바이스는 AWS IoT Greengrass 서비스와의 연결이 복원되기 전까지 코어와 연결할 수 없으며 새로운 MQTT 서버 인증서를 다운로드할 수 있습니다.

만료 기한은 필요에 따라 7일에서 30일 사이로 설정할 수 있습니다. 자주 교체하기 위해서는 잦은 클라우드 연결이 요구됩니다. 교체가 자주 이루어지지 않으면 보안 문제를 초래할 수 있습니다. 인증서 만료 값을 30일 이상으로 설정하려면 AWS Support에 문의하십시오.

AWS IoT 콘솔에서는 그룹의 설정 페이지에서 인증서를 관리할 수 있습니다. AWS IoT GreengrassAPI에서 UpdateGroupCertificateConfiguration작업을 사용할 수 있습니다.

MQTT 서버 인증서가 만료되면 인증서의 유효성을 검사하는 시도가 실패합니다. 클라이언트 디바이스는 장애를 감지하고 연결을 종료할 수 있어야 합니다.

데이터 영역 작업에 대한 AWS IoT 정책

AWS IoT 정책을 사용하여 AWS IoT Core 및 AWS IoT Greengrass 데이터 플레인에 대한 액세스 권한을 부여합니다. AWS IoT Core 데이터 플레인은 AWS IoT Core 연결 및 주제 구독과 같은 디바이스, 사용자 및 애플리케이션 작업으로 구성됩니다. AWS IoT Greengrass 데이터 플레인은 배포 검색 및 연결 정보 업데이트와 같은 Greengrass 디바이스 작업으로 구성됩니다.

AWS IoT 정책은 IAM 정책과 유사한 JSON 문서입니다. 여기에는 다음 속성을 지정하는 하나 이상의 정책 설명이 포함됩니다.

- `Effect`. 액세스 모드(`Allow` 또는 `Deny` 일 수 있음)

- `Action`. 정책에서 허용하거나 거부하는 작업 목록.
- `Resource`. 작업이 허용되거나 거부되는 리소스 목록입니다.

AWS IoT 정책은 *을(를) 와일드카드 문자로 지원하며 MQTT 와일드카드 문자(+ 및 #)를 리터럴 문자열로 취급합니다. * 와일드카드에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 리소스 ARN에서 와일드카드 사용을 참조하십시오.

자세한 내용은 AWS IoT Core 개발자 안내서의 AWS IoT 정책 및 AWS IoT 정책 조치를 참조하십시오.

> **Note**
>
> AWS IoT Core은(는) 사물 그룹에 AWS IoT 정책을 연결하여 디바이스 그룹에 대한 권한을 정의할 수 있게 합니다. 사물 그룹 정책은 AWS IoT Greengrass 데이터 영역 작업에 대한 액세스를 허용하지 않습니다. AWS IoT Greengrass 데이터 영역 작업에 대한 사물 액세스를 허용하려면 사물 인증서에 연결하는 AWS IoT 정책에 권한을 추가하세요.

AWS IoT Greengrass 정책 작업

Greengrass 코어 작업

AWS IoT Greengrass는 Greengrass 코어 디바이스가 AWS IoT 정책에서 사용할 수 있는 다음 정책 작업을 정의합니다.

`greengrass:AssumeRoleForGroup`

토큰 교환 서비스(TES) 시스템 Lambda 함수를 사용하여 보안 인증을 검색할 수 있는 Greengrass 핵심 디바이스에 대한 권한. 검색된 자격 증명에 연결된 권한은 구성된 그룹 역할에 연결된 정책을 기반으로 합니다.

이 권한은 Greengrass 코어 디바이스가 자격 증명을 검색하려고 할 때 확인됩니다(자격 증명이 로컬로 캐시되지 않는다고 가정).

`greengrass:CreateCertificate`

Greengrass 코어 디바이스가 자체 서버 인증서를 만들 수 있는 권한입니다.

이 권한은 Greengrass 코어 디바이스가 인증서를 만들 때 확인됩니다. Greengrass 코어 디바이스는 처음 실행될 때, 코어의 연결 정보가 변경될 때 및 지정된 교체 기간에 서버 인증서 생성을 시도합니다.

`greengrass:GetConnectivityInfo`

Greengrass 코어 디바이스가 자체 연결 정보를 검색할 수 있는 권한입니다.

이 권한은 Greengrass 코어 디바이스가 AWS IoT Core에서 연결 정보를 검색하려고 할 때 확인됩니다.

`greengrass:GetDeployment`

Greengrass 코어 디바이스가 배포를 검색할 수 있는 권한입니다.

이 권한은 Greengrass 코어 디바이스가 클라우드에서 배포 및 배포 상태를 검색하려고 할 때 확인됩니다.

`greengrass:GetDeploymentArtifacts`

그룹 정보 또는 Lambda 함수 같은 배포 아티팩트를 검색할 수 있는 Greengrass 코어 디바이스에 대한 권한입니다.

이 권한은 Greengrass 코어 디바이스가 배포를 수신한 다음 배포 아티팩트를 검색하려고 할 때 확인됩니다.

`greengrass:UpdateConnectivityInfo`

Greengrass 코어 디바이스가 IP 또는 호스트 이름 정보로 자체 연결 정보를 업데이트할 수 있는 권한입니다.

이 권한은 Greengrass 코어 디바이스가 클라우드에서 연결 정보를 업데이트하려고 할 때 확인됩니다.

`greengrass:UpdateCoreDeploymentStatus`

Greengrass 코어 디바이스가 배포 상태를 업데이트할 수 있는 권한입니다.

이 권한은 Greengrass 코어 디바이스가 배포를 수신한 다음 배포 상태를 업데이트하려고 할 때 확인됩니다.

Greengrass 디바이스 작업

AWS IoT Greengrass는 클라이언트 디바이스가 AWS IoT 정책에서 사용할 수 있는 다음과 같은 정책 작업을 정의합니다.

`greengrass:Discover`

클라이언트 디바이스가 Discovery API를 사용하여 그룹의 코어 연결 정보 및 그룹 인증 기관을 검색할 수 있는 권한입니다.

이 권한은 클라이언트 디바이스가 TLS 상호 인증을 사용하여 검색 API를 직접 호출할 때 확인됩니다.

AWS IoT Greengrass 코어 디바이스에 대한 최소 AWS IoT 정책

다음 예제 정책에는 코어 디바이스용 기본 Greengrass 기능을 지원하는 데 필요한 최소 작업 세트가 포함됩니다.

- 이 정책은 코어 디바이스가 섀도우 상태에 사용되는 주제를 포함하여 메시지를 게시하고 구독하고 메시지를 수신할 수 있는 MQTT 정책과 주제 필터를 나열합니다. Greengrass 그룹의 AWS IoT Core, Lambda 함수, 커넥터 및 디바이스 간에 메시지 교환을 지원하려면 허용할 주제와 주제 필터를 지정합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 게시/구독 정책 예제를 참조하십시오.
- 정책에는 AWS IoT Core가 코어 디바이스의 섀도우를 가져오고 업데이트하고 삭제하도록 허용하는 섹션이 포함됩니다. Greengrass 그룹의 클라이언트 디바이스에 대한 섀도우 동기화를 허용하려면 `Resource` 목록에서 대상 Amazon 리소스 이름(ARN)을 지정합니다(예: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`device-name`*).
- 핵심 디바이스에 대한 AWS IoT 정책에서 사물 정책 변수(`iot:Connection.Thing.*`)를 사용하는 것은 지원되지 않습니다. 코어는 동일한 디바이스 인증서를 사용하여 AWS IoT Core에 여러 번 연결하지만 연결의 클라이언트 ID가 핵심 사물 이름과 정확히 일치하지 않을 수 있습니다.
- `greengrass:UpdateCoreDeploymentStatus` 권한의 경우 `Resource` ARN의 최종 세그먼트는 코어 디바이스의 URL로 인코딩된 ARN입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:client/core-name-*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:AssumeRoleForGroup",
                "greengrass:CreateCertificate"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "greengrass:GetDeployment"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeploymentArtifacts"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:UpdateCoreDeploymentStatus"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*/cores/arn%3Aaws%3Aiot%3Aregion%3Aaccount-id%3Athing%2Fcore-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        }
    ]
}
```

> **Note**
>
> 클라이언트 디바이스용 AWS IoT 정책에는 일반적으로 `iot:Connect`, `iot:Publish`, `iot:Receive` 및 `iot:Subscribe` 작업에 대한 유사한 권한이 필요합니다. 디바이스가 속한 Greengrass 그룹의 코어에 대한 연결 정보를 클라이언트 디바이스가 자동으로 감지하도록 허용하려면 연결된 디바이스에 대한 AWS IoT 정책에 `greengrass:Discover` 작업이 포함되어야 합니다. `Resource` 섹션에서 Greengrass 코어 디바이스의 ARN이 아닌 클라이언트 디바이스의 ARN을 지정합니다. 예:
>
> ```
> {
>     "Effect": "Allow",
>     "Action": [
>         "greengrass:Discover"
>     ],
>     "Resource": [
>         "arn:aws:iot:region:account-id:thing/device-name"
>     ]
> }
> ```
>
> Greengrass 코어가 클라이언트 디바이스에 대한 섀도우 동기화 작업을 처리하기 때문에 연결된 디바이스용 AWS IoT 정책에는 일반적으로 `iot:GetThingShadow`, `iot:UpdateThingShadow` 또는 `iot:DeleteThingShadow` 작업에 대한 권한이 필요하지 않습니다. 이 경우 코어 AWS IoT 정책의 섀도우 작업에 대한 `Resource` 섹션에 연결된 디바이스의 ARN이 포함되어 있는지 확인합니다.

AWS IoT 콘솔에서는 코어 인증서에 연결된 정책을 보고 편집할 수 있습니다.

1. 탐색 창의 관리에서 모든 디바이스를 확장한 다음 사물을 선택합니다.
2. 코어를 선택합니다.
3. 코어의 구성 페이지에서 인증서 탭을 선택합니다.
4. 인증서 페이지에서 인증서를 선택합니다.
5. 인증서의 구성 페이지에서 Policies(정책)을 선택한 다음 정책을 선택합니다.

   정책을 편집하려면 활성 버전 편집을 선택합니다.
6. 정책을 검토하고 필요에 따라 권한을 추가, 제거 또는 편집합니다.

7. 새 정책 버전을 활성 버전으로 설정하려면 정책 버전 상태에서 편집한 버전을 이 정책의 활성 버전으로 설정을 선택합니다.
8. 새 버전으로 저장을 선택합니다.

AWS IoT Greengrass의 I자격 증명 및 액세스 관리

AWS Identity and Access Management(IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 통제할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 AWS IoT Greengrass 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

> **Note**
>
> 이 주제에서는 IAM 개념과 기능에 대해 설명합니다. AWS IoT Greengrass에서 지원하는 IAM 기능에 대한 자세한 내용은 the section called “AWS IoT Greengrass에서 IAM을 사용하는 방식” 섹션을 참조하세요.

고객

AWS Identity and Access Management(IAM)을 사용하는 방법은 AWS IoT Greengrass에서 수행하는 작업에 따라 달라집니다.

서비스 사용자 - AWS IoT Greengrass 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증과 권한을 관리자가 제공합니다. 더 많은 AWS IoT Greengrass 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. AWS IoT Greengrass의 기능에 액세스할 수 없는 경우 AWS IoT Greengrass의 자격 증명 및 액세스 문제 해결 단원을 참조하십시오.

서비스 관리자 - 회사에서 AWS IoT Greengrass 리소스를 책임지고 있는 담당자라면 AWS IoT Greengrass에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 AWS IoT Greengrass 기능과 리소스를 결정합니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 AWS IoT Greengrass에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 AWS IoT Greengrass에서 IAM을 사용하는 방식 단원을 참조하십시오.

IAM 관리자 - IAM 관리자라면 AWS IoT Greengrass에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 AWS IoT Greengrass 자격 증명 기반 정책 예제를 보려면 AWS IoT Greengrass 자격 증명 기반 정책 예제 섹션을 참조하십시오.

보안 인증을 통한 인증

인증은 ID 보안 인증 정보를 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자이나 IAM 사용자로 또는 IAM 역할을 수임하여 인증(AWS에 로그인)되어야 합니다.

보안 인증 정보 소스를 통해 제공된 보안 인증 정보를 사용하여 페더레이션형 ID로 AWS에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증, Google 또는 Facebook 보안 인증 정보가 페더레이션형 ID의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라 AWS Management Console 또는AWS 액세스 포털에 로그인할 수 있습니다. AWS에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법을 참조하십시오.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 보안 인증 정보를 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청에 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS는 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하십시오.

AWS 계정 루트 사용자

AWS 계정를 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 자격 증명은 AWS 계정 루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에는 루트 사용자를 가급적 사용하지 않는 것이 좋습니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하십시오.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 보안 인증이 있는 IAM 사용자를 생성하는 대신 임시 보안 인증을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하십시오.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 귀하는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins(이)라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 보안 인증을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 정보는 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하십시오.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 ID입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할 전환하여 AWS Management Console에서 IAM 역할을 임시로 수임할 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM 역할 사용을 참조하십시오.

임시 보안 인증 정보가 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 자격 증명에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 Creating a role for a third-party Identity Provider(서드 파티 자격 증명 공급자의 역할 만들기) 부분을 참조하십시오. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 아이덴티티가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연결합니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하십시오.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스: IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입

니다. 그러나 일부 AWS 서비스를 사용하면 역할을(프록시로 사용하는 대신) 리소스에 정책을 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하십시오.

- 교차 서비스 액세스 - 일부 AWS 서비스는 다른 AWS 서비스의 기능을 사용합니다. 예를 들어 서비스에서 직접적으로 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 보안 주체의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하십시오.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.
  - 서비스 연결 역할 - 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.
- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 보안 인증을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

정책을 사용한 액세스 관리

정책을 생성하고 AWS 자격 증명 또는 리소스에 연결하여 AWS 내 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 보안 주체(사용

자, 루트 사용자 또는 역할 세션)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되는지 또는 거부되는지를 결정합니다. 대부분의 정책은 AWS에 JSON 설명서로서 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하십시오.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는 지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI또는 AWSAPI에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오.

자격 증명 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 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 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3, AWS WAF 및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 참조하십시오.

기타 정책 유형

AWS은(는) 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 유형은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 엔터티의 ID 기반 정책 및 해당 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔터티에 대한 권한 경계를 참조하십시오.
- 서비스 제어 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations은 기업이 소유하는 여러 개의 AWS 계정를 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자을 비롯하여 멤버 계정의 엔터티에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 정보는 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하십시오.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책 및 세션 정책의 교집합입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하십시오.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

다음 사항도 참조하십시오.

- the section called “AWS IoT Greengrass에서 IAM을 사용하는 방식”
- the section called “자격 증명 기반 정책 예제”
- the section called “자격 증명 및 액세스 문제 해결”

AWS IoT Greengrass에서 IAM을 사용하는 방식

IAM을 사용하여 AWS IoT Greengrass에 액세스를 관리하기 전에 AWS IoT Greengrass에서 사용할 수 있는 IAM 기능을 이해해야 합니다.

IAM 특성	Greengrass에서 지원합니까?
리소스 수준 권한이 있는 자격 증명 기반 정책	예
리소스 기반 정책	아니요
액세스 제어 목록(ACL)	아니요
태그 기반 승인	예
임시 보안 인증	예
서비스 연결 역할	아니요
서비스 역할	예

기타 AWS 서비스가 IAM과 작동하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

AWS IoT Greengrass에 대한 ID 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업 및 리소스를 지정할 수 있을 뿐 아니라 작업이 허용되거나 거부되는 조건도 지정할 수 있습니다. AWS IoT Greengrass는 특정 작업, 리소스 및 조건 키를 지원합니다. 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 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. 일반적으로 정책 작업의 이름은 연결된 AWS API 작업의 이름과 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함합니다.

AWS IoT Greengrass의 정책 작업은 작업 앞에 `greengrass:` 접두사를 사용합니다. 예를 들어, 다른 사람이 `ListGroups` API 작업을 사용하여 `greengrass:ListGroups`의 그룹을 나열할 수 있도록 하려면 해당 AWS 계정 작업을 정책에 포함시킵니다. 정책 명령문에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다. AWS IoT Greengrass는 이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 집합을 정의합니다.

명령문 하나에 여러 작업을 지정하려면 다음과 같이 괄호 안에 나열한 후(`[ ]`) 각 작업을 쉼표로 구분합니다.

```
"Action": [
    "greengrass:action1",
    "greengrass:action2",
    "greengrass:action3"
]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `List`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "greengrass:List*"
```

> **Note**
>
> 와일드카드를 사용하여 서비스에 대해 사용 가능한 모든 작업을 지정하는 것은 권장하지 않습니다. 가장 좋은 방법은 정책에 최소한의 권한을 부여하고 권한을 좁히는 것입니다. 자세한 설명은 the section called “가능한 최소 권한 부여” 섹션을 참조하세요.

AWS IoT Greengrass 작업의 전체 목록을 보려면 IAM 사용 설명서의 AWS IoT Greengrass에서 정의한 작업을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 명령문에는 `Resource` 또는 `NotResource` 요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

다음 표에는 정책 설명의 `Resource` 요소에 사용할 수 있는 AWS IoT Greengrass 리소스 ARN이 나와 있습니다. AWS IoT Greengrass 작업에 지원되는 리소스 수준 권한 매핑은 IAM 사용 설명서에서 AWS IoT Greengrass에서 정의한 작업을 참조하세요.

Resource	ARN
Group	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}`
GroupVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/versions/${VersionId}`
CertificateAuthority	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/certificateauthorities/${CertificateAuthorityId}`
Deployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/deployments/${DeploymentId}`
BulkDeployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/bulk/deployments/${BulkDeploymentId}`

Resource	ARN
ConnectorDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}
ConnectorDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}/versions/${VersionId}
CoreDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}
CoreDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}/versions/${VersionId}
DeviceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}
DeviceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}/versions/${VersionId}
FunctionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}
FunctionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}/versions/${VersionId}
LoggerDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}

Resource	ARN
LoggerDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}/versions/${VersionId}
ResourceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}
ResourceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}/versions/${VersionId}
SubscriptionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}
SubscriptionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}/versions/${VersionId}
ConnectivityInfo	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/things/${ThingName}/connectivityInfo

다음 예제 Resource 요소는 AWS 계정 123456789012의 미국 서부(오레곤) 리전에 있는 그룹의 ARN을 지정합니다.

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

또는 특정 AWS 리전의 AWS 계정에 속한 모든 그룹을 지정하려면 그룹 ID 대신 와일드카드를 사용합니다.

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

일부 AWS IoT Greengrass 작업(예: 일부 목록 작업)은 특정 리소스에 대해 수행할 수 없습니다. 이러한 경우 와일드카드만 사용해야 합니다.

```
"Resource": "*"
```

한 명령에 여러 리소스 ARN을 지정하려면 다음과 같이 괄호 사이에 나열하고([ ]) 쉼표로 구분합니다.

```
"Resource": [
    "resource-arn1",
    "resource-arn2",
    "resource-arn3"
]
```

ARN 형식에 대한 자세한 내용은 Amazon Web Services 일반 참조에서 Amazon 리소스 이름(ARN) 및 AWS 서비스 네임스페이스를 참조하세요.

조건 키

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우 AWS는 논리적 AND 연산을 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우 AWS는 논리적 OR 연산을 사용하여 조건을 평가합니다. 명령문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키 섹션을 참조하십시오.

AWS IoT Greengrass는 다음과 같은 전역 조건 키를 지원합니다.

키	설명
aws:CurrentTime	현재 날짜 및 시간에 대한 날짜/시간 조건을 확인하여 액세스를 필터링합니다.

키	설명
aws:EpochTime	Epoch 또는 UNIX 시간의 현재 날짜 및 시간에 대한 날짜/시간 조건을 확인하여 액세스를 필터링합니다.
aws:MultiFactorAuthAge	Multi-Factor Authentication(MFA)을 사용하여 발행된 요청에서 MFA가 유효성을 검사한 보안 자격 증명이 경과한 시간(단위: 초)을 확인하여 액세스를 필터링합니다.
aws:MultiFactorAuthPresent	Multi-Factor Authentication(MFA)을 사용하여 현재 요청을 수행한 임시 보안 자격 증명의 유효성을 검사했는지 여부를 확인하여 액세스를 필터링합니다.
aws:RequestTag/${TagKey}	각 필수 태그에 허용되는 값의 집합에 따라 생성 요청을 필터링합니다.
aws:ResourceTag/${TagKey}	리소스와 연결된 태그 값을 기준으로 작업을 필터링합니다.
aws:SecureTransport	SSL을 사용하여 요청을 보냈는지 여부를 확인하여 액세스를 필터링합니다.
aws:TagKeys	요청에 필수 태그가 있는지 여부를 기준으로 생성 요청을 필터링합니다.
aws:UserAgent	요청자의 클라이언트 애플리케이션을 기준으로 액세스를 필터링합니다.

자세한 내용은 IAM 사용 설명서의 AWS 글로벌 조건 컨텍스트 키를 참조하세요.

예제

AWS IoT Greengrass 자격 증명 기반 정책의 예제를 보려면 the section called “자격 증명 기반 정책 예제” 섹션을 참조하세요.

AWS IoT Greengrass를 위한 리소스 기반 정책

AWS IoT Greengrass는 리소스 기반 정책을 지원하지 않습니다.

액세스 제어 목록(ACL)

AWS IoT Greengrass는 ACL을 지원하지 않습니다.

AWS IoT Greengrass 태그 기반 인증

태그를 지원되는 AWS IoT Greengrass 리소스에 연결하거나 요청을 통해 태그를 AWS IoT Greengrass에 전달할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. 자세한 설명은 Greengrass 리소스에 태그 지정 섹션을 참조하세요.

AWS IoT Greengrass의 IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 엔터티입니다.

AWS IoT Greengrass에서 임시 보안 인증 정보 사용

임시 보안 인증 정보는 페더레이션을 통해 로그인하거나, IAM 역할을 수임하거나, 교차 계정 역할을 수임하는 데 사용됩니다. AWS STS API 작업(예: AssumeRole 또는 GetFederationToken)을 호출하여 임시 보안 인증 정보를 가져옵니다.

Greengrass 코어에서는 그룹 역할에 대한 임시 보안 인증 정보를 사용자 정의 Lambda 함수 및 커넥터에 사용할 수 있습니다. Lambda 함수가 AWS SDK를 사용하는 경우 AWS가 이를 대신 수행하므로 자격 증명을 얻기 위해 로직을 추가할 필요가 없습니다.

서비스 연결 역할

AWS IoT Greengrass에서는 서비스 연결 역할을 지원하지 않습니다.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 서비스 역할을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

AWS IoT Greengrass는 서비스 역할을 사용하여 사용자를 대신하여 일부 AWS 리소스에 액세스합니다. 자세한 설명은 the section called “Greengrass 서비스 역할” 섹션을 참조하세요.

AWS IoT Greengrass 콘솔에서 IAM 역할 선택

AWS IoT Greengrass 콘솔에서 계정의 IAM 역할 목록의 Greengrass 서비스 역할 또는 Greengrass 그룹 역할을 선택해야 할 수 있습니다.

- Greengrass 서비스 역할을 통해 AWS IoT Greengrass는 사용자를 대신하여 다른 서비스의 AWS 리소스에 액세스 할 수 있습니다. 일반적으로 콘솔이 서비스 역할을 만들고 구성할 수 있으므로 사용자는 서비스 역할을 선택할 필요가 없습니다. 자세한 설명은 the section called “Greengrass 서비스 역할” 섹션을 참조하세요.
- Greengrass 그룹 역할은 그룹의 Lambda 함수 및 커넥터가 AWS 리소스에 액세스할 수 있도록 하는 데 사용됩니다. 또한 스트림을 AWS 서비스로 내보내고 CloudWatch 로그를 쓸 수 있는 AWS IoT Greengrass 권한을 부여할 수 있습니다. 자세한 내용은 the section called “Greengrass 그룹 역할”을(를) 참조하세요.

Greengrass 서비스 역할

Greengrass 서비스 역할은 사용자를 대신하여 AWS 서비스의 리소스에 액세스할 수 있도록 AWS IoT Greengrass에 권한을 부여하는 AWS Identity and Access Management (IAM) 서비스 역할입니다. AWS IoT Greengrass는 이 역할을 사용하여 AWS Lambda 함수 검색 및 AWS IoT 섀도우 관리 등의 필수 작업을 수행할 수 있습니다.

AWS IoT Greengrass가 리소스에 액세스하도록 허용하려면 Greengrass 서비스 역할을 AWS 계정 계정과 연결하고 AWS IoT Greengrass를 신뢰할 수 있는 엔터티로 지정해야 합니다. 역할에는 AWSGreengrassResourceAccessRolePolicy 관리형 정책 또는 사용 중인 AWS IoT Greengrass 기능에 대해 동등한 권한을 정의하는 사용자 지정 정책이 포함되어야 합니다. 이 정책은 AWS에서 관리하며, AWS IoT Greengrass이 AWS 리소스에 액세스하는 데 사용하는 권한 집합을 정의합니다.

여러 AWS 리전에서 동일한 Greengrass 서비스 역할을 재사용할 수 있지만, AWS IoT Greengrass를 사용하는 모든 AWS 리전에서 해당 역할을 AWS 계정과 연결해야 합니다. 현재 AWS 계정 및 리전에 서비스 역할이 없는 경우 그룹 배치가 실패합니다.

다음 단원에서는 AWS Management Console 또는 AWS CLI에서 Greengrass 서비스를 생성 및 관리하는 방법에 대해 설명합니다.

- 서비스 역할 관리(콘솔)
- 서비스 역할 관리(CLI)

> **Note**
>
> 서비스 레벨 액세스에 권한을 부여하는 서비스 역할 외에도, 그룹 역할을 AWS IoT Greengrass 그룹에 할당할 수 있습니다. 이 그룹 역할은 그룹의 Lambda 함수 및 커넥터가 AWS 서비스에 액세스하는 방식을 제어하는 별도의 IAM 역할입니다.

Greengrass 서비스 역할 관리(콘솔)

AWS IoT 콘솔에서는 Greengrass 서비스 역할을 손쉽게 관리할 수 있습니다. 예를 들어 Greengrass 그룹을 생성 또는 배포할 때 이 콘솔은 현재 콘솔에서 선택된 AWS 계정의 AWS 리전이 Greengrass 서비스 역할에 연결되어 있는지 여부를 확인합니다. 연결이 되어 있지 않으면 콘솔이 사용자를 대신하여 서비스 역할을 생성 및 구성할 수 있습니다. 자세한 내용은 the section called “Greengrass 서비스 역할 생성” 섹션을 참조하세요.

다음과 같은 역할 관리 작업에서 AWS IoT 콘솔을 사용할 수 있습니다.

- Greengrass 서비스 역할 검색
- Greengrass 서비스 역할 생성
- Greengrass 서비스 역할 변경
- Greengrass 서비스 역할 분리

> **Note**
>
> 콘솔에 로그인한 사용자는 해당 서비스 역할을 보고, 생성하고, 변경할 수 있는 권한을 가져야 합니다.

Greengrass 서비스 역할 검색(콘솔)

다음 절차를 사용해 현재 AWS 리전에서 AWS IoT Greengrass이 사용 중인 서비스 역할을 검색합니다.

1. AWS IoT 콘솔의 탐색 창에서 설정을 선택합니다.
2. Greengrass service role(Greengrass 서비스 역할) 섹션으로 스크롤하여 서비스 역할과 그 정책을 확인합니다.

   서비스 역할이 표시되지 않으면 콘솔에게 사용자를 대신해 역할을 생성 또는 구성하도록 할 수 있습니다. 자세한 내용은 Greengrass 서비스 역할 생성 섹션을 참조하세요.

Greengrass 서비스 역할 생성(콘솔)

콘솔이 사용자를 대신하여 기본 Greengrass 서비스 역할을 생성 및 구성할 수 있습니다. 이 역할에는 다음 속성이 있습니다.

속성	값
이름	`Greengrass_ServiceRole`
신뢰할 수 있는 엔터티.	`AWS service: greengrass`
Policy	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Greengrass 디바이스 설정에서 서비스 역할을 생성하는 경우 역할 이름은 `GreengrassServiceRole_`*`random-string`*입니다.

AWS IoT에서 Greengrass 그룹을 생성 또는 배포할 때 이 콘솔은 현재 콘솔에서 선택된 AWS 계정 리전의 AWS 리전 계정이 Greengrass 서비스 역할에 연결되어 있는지 여부를 확인합니다. 연결되지 않은 경우 AWS IoT Greengrass가 사용자를 대신해 AWS 서비스를 읽고 쓸 수 있도록 허용하라는 메시지가 콘솔에 표시됩니다.

권한을 부여하는 경우, 콘솔은 `Greengrass_ServiceRole` 이라는 이름의 역할이 AWS 계정 계정에 존재하는지 여부를 확인합니다.

- 역할이 존재하면 콘솔은 해당 서비스 역할을 현재 AWS 리전 리전의 AWS 계정 계정에 연결합니다.
- 역할이 존재하지 않으면 콘솔은 기본 Greengrass 서비스 역할을 생성하여 이를 현재 AWS 리전 리전의 AWS 계정 계정에 연결합니다.

> **Note**
>
> 사용자 지정 역할 정책을 사용하여 서비스를 생성하려는 경우에는 IAM 콘솔을 사용하여 역할을 생성하거나 수정하세요. 자세한 정보는 IAM 사용 설명서의AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요. 역할이 사용하는 기능 및 리소스에 대한 `AWSGreengrassResourceAccessRolePolicy` 관리형 정책과 동일한 권한을 부여하는

지 확인합니다. 혼동된 대리자 보안 문제를 방지하려면 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키도 포함하는 것이 좋습니다. 조건 컨텍스트 키는 지정된 계정 및 Greengrass 작업 영역에서 들어오는 요청만 허용하도록 액세스를 제한합니다. 혼동된 대리자 문제에 대한 자세한 내용은, 교차 서비스 혼동된 대리자 예방를 참조하세요. 서비스 역할을 생성하는 경우 AWS IoT 콘솔로 돌아가서 해당 역할을 그룹에 연결합니다. 그룹 설정 페이지의 Greengrass 서비스 역할에서 이 작업을 수행할 수 있습니다.

Greengrass 서비스 역할 변경(콘솔)

다음 절차를 사용하여 현재 콘솔에서 선택된 AWS 리전의 AWS 계정에 연결할 다른 Greengrass 서비스 경로를 선택합니다.

1. AWS IoT 콘솔의 탐색 창에서 설정을 선택합니다.
2. Greengrass 서비스 역할에서 역할 선택을 선택합니다.

   Greengrass 서비스 역할 업데이트 대화 상자가 열리고 AWS IoT Greengrass을(를) 신뢰할 수 있는 엔터티로 정의하는 AWS 계정의 IAM 역할이 표시됩니다.
3. 연결할 Greengrass 서비스 역할을 선택합니다.
4. 역할 연결을 선택합니다.

> **Note**
>
> 콘솔이 사용자를 대신해 기본 Greengrass 서비스 역할을 생성하도록 허용하려면 목록에서 역할을 선택하는 대신 Create role for me(사용자를 대신해 역할 생성)를 선택합니다. 사용자의 AWS 계정에 이름이 `Greengrass_ServiceRole`인 역할이 있는 경우 나를 위한 역할 생성 링크가 표시되지 않습니다.

Greengrass 서비스 역할 분리(콘솔)

다음 절차를 사용하여 현재 콘솔에서 선택된 AWS 리전의 AWS 계정에서 Greengrass 서비스 역할을 분리합니다. 이렇게 하면 AWS IoT Greengrass가 현재 AWS 리전 리전에서 AWS 서비스에 액세스할 수 있는 권한이 취소됩니다.

> **Important**
> 서비스 역할을 분리하면 진행 중인 작업에 방해가 될 수 있습니다.

1. AWS IoT 콘솔의 탐색 창에서 설정을 선택합니다.
2. Greengrass 서비스 역할에서 분리를 선택합니다.
3. 확인 대화 상자에서 분리(Detach)를 선택합니다.

> **Note**
> 역할이 더 이상 필요하지 않으면 IAM 콘솔에서 이를 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서에서 역할 또는 인스턴스 프로필 삭제를 참조하세요.
> 다른 역할들은 AWS IoT Greengrass가 리소스에 액세스하도록 허용할 수 있습니다. AWS IoT Greengrass이(가) 사용자를 대신하여 권한을 위임할 수 있는 모든 역할을 찾으려면 IAM 콘솔의 역할 페이지의 신뢰할 수 있는 엔터티 열에서 AWS 서비스: greengrass가 포함된 역할을 찾아보십시오.

Greengrass 서비스 역할 관리(CLI)

다음 절차에서는 AWS CLI가 설치되고 AWS 계정 ID를 사용하도록 구성되어 있다고 가정합니다. 자세한 내용은 AWS CLI 명령줄 인터페이스 AWS Command Line Interface 사용 설명서의 명령줄 인터페이스 AWS 설치를 참조하십시오.

다음과 같은 역할 관리 작업에서 AWS CLI을 사용할 수 있습니다.

- Greengrass 서비스 역할 가져오기
- Greengrass 서비스 역할 생성
- Greengrass 서비스 역할 제거

Greengrass 서비스 역할 가져오기(CLI)

다음 절차를 사용하여 Greengrass 서비스 역할이 AWS 리전의 AWS 계정과 연결되어 있는지 알아냅니다.

- 서비스 역할을 가져옵니다. 을 해당 AWS 리전 *##*(예: us-west-2)으로 바꿉니다.

  ```
  aws Greengrass get-service-role-for-account --region region
  ```

  Greengrass 서비스 역할이 계정과 이미 연결되어 있는 경우 다음 역할 메타데이터가 반환됩니다.

  ```
  {
    "AssociatedAt": "timestamp",
    "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  역할 메타데이터가 반환되지 않는 경우 서비스 역할을 생성하고(없는 경우) 해당 역할을 AWS 리전의 계정과 연결해야 합니다.

Greengrass 서비스 역할 생성(CLI)

다음 단계에 따라 역할을 생성하고 AWS 계정과 연결합니다.

IAM을 사용하여 서비스 역할을 생성하려면

1. AWS IoT Greengrass이 역할을 수임하도록 허용하는 신뢰 정책이 있는 역할을 생성합니다. 이 예제에서는 `Greengrass_ServiceRole`라는 역할을 생성하지만, 다른 이름을 사용할 수 있습니다. 혼동된 대리자 보안 문제를 방지하려면 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키도 포함하는 것이 좋습니다. 조건 컨텍스트 키는 지정된 계정 및 Greengrass 작업 영역에서 들어오는 요청만 허용하도록 액세스를 제한합니다. 혼동된 대리자 문제에 대한 자세한 내용은, 교차 서비스 혼동된 대리자 예방를 참조하세요.

   Linux, macOS, or Unix

   ```
   aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
   ```

```
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}'
```

Windows command prompt

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

2. 출력의 역할 메타데이터에서 역할 ARN을 복사합니다. ARN을 사용하여 역할을 계정과 연결합니다.
3. `AWSGreengrassResourceAccessRolePolicy` 정책을 역할에 연결합니다.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

계정에 AWS 계정 서비스 역할을 연결하려면 다음을 수행합니다.

- 역할을 계정과 연결합니다. *role-arn*을 서비스 역할 ARN과 바꾸고 AWS 리전을 *##*(예: `us-west-2`)과 바꿉니다.

```
aws greengrass associate-service-role-to-account --role-arn role-arn --
region region
```

성공하면 다음 응답이 반환됩니다.

```
{
```

```
  "AssociatedAt": "timestamp"
}
```

Greengrass 서비스 역할 제거(CLI)

다음 단계를 사용하여 AWS 계정에서 Greengrass 서비스 역할의 연결을 해제합니다.

- 계정에서 서비스 역할의 연결을 해제합니다. AWS 리전을 해당 *##*(예: `us-west-2`)으로 바꿉니다.

  ```
  aws greengrass disassociate-service-role-from-account --region region
  ```

  성공하면 다음 응답이 반환됩니다.

  ```
  {
    "DisassociatedAt": "timestamp"
  }
  ```

  > **Note**
  >
  > AWS 리전에서 사용하지 않을 경우 서비스를 삭제해야 합니다. 먼저 delete-role-policy를 사용하여 역할에서 `AWSGreengrassResourceAccessRolePolicy` 관리형 정책을 연결 해제하고, delete-role을 사용하여 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서에서 역할 또는 인스턴스 프로필 삭제를 참조하세요.

다음 사항도 참조하세요.

- IAM 사용 설명서의 역할을 만들어 AWS 사용자에게 권한 위임.
- IAM 사용자 설명서에서 역할 수정
- 자세한 내용은 IAM 사용 설명서에서 역할 또는 인스턴스 프로필 삭제를 참조하세요.
- AWS CLI명령 참조에서 사용 가능한 AWS IoT Greengrass 명령
  - associate-service-role-to-account
  - disassociate-service-role-from-account
  - get-service-role-for-account

- AWS CLI 명령 참조에서 사용 가능한 IAM 명령
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Greengrass 그룹 역할

Greengrass 그룹 역할은 Greengrass 코어에서 실행되는 코드가 AWS 리소스에 액세스하도록 권한을 부여하는 역할입니다. AWS Identity and Access Management (IAM)에서 역할을 만들고 권한을 관리하고 Greengrass 그룹에 역할을 연결합니다. Greengrass 그룹에는 하나의 그룹 역할이 있습니다. 권한을 추가하거나 변경하려면 다른 역할을 연결하거나 역할에 연결된 IAM 정책을 변경하면 됩니다.

이 역할은 AWS IoT Greengrass를 신뢰할 수 있는 엔터티로 정의해야 합니다. 비즈니스 사례에 따라 그룹 역할에 다음을 정의하는 IAM 정책이 포함될 수 있습니다.

- 사용자 정의 Lambda 함수의 AWS 서비스에 대한 액세스 권한.
- 커넥터가 AWS 서비스에 액세스하기 위한 권한.
- 스트림 관리자가 Kinesis 데이터 스트림 및 AWS IoT Analytics로 스트림을 내보낼 수 있는 권한.
- CloudWatch 로깅에 대한 권한

다음 섹션에서는 AWS Management Console 또는 AWS CLI에서 Greengrass 그룹 역할을 연결하거나 분리하는 방법에 대해 설명합니다.

- 그룹 역할 관리(콘솔)
- 그룹 역할 관리(CLI)

> **Note**
>
> Greengrass 코어에서 액세스를 승인하는 그룹 역할 외에도 AWS IoT Greengrass이(가) 사용자를 대신하여 AWS 리소스에 액세스할 수 있는 Greengrass 서비스 역할을 할당할 수 있습니다.

Greengrass 그룹 역할 관리(콘솔)

다음과 같은 역할 관리 작업에서 AWS IoT을 사용할 수 있습니다.

- Greengrass 그룹 역할 찾기
- Greengrass 그룹 역할 추가 또는 변경
- Greengrass 그룹 역할 제거

> **Note**
> 콘솔에 로그인한 사용자는 해당 역할을 관리할 수 있는 권한을 가져야 합니다.

Greengrass 그룹 역할 찾기(콘솔)

Greengrass 그룹에 연결된 역할을 찾으려면 다음 단계를 수행하십시오.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 그룹 구성 페이지에서 설정 보기를 선택합니다.

역할이 그룹에 연결된 경우 그룹 역할 아래에 나타납니다.

Greengrass 그룹 역할 추가 또는 변경(콘솔)

다음 단계에 따라 AWS 계정에서 Greengrass 그룹에 추가할 IAM 역할을 선택합니다.

그룹 역할에는 다음과 같은 요구사항이 있습니다.

- 신뢰할 수 있는 엔터티로 정의된 AWS IoT Greengrass.
- 이 역할에 연결된 권한 정책은 그룹의 Lambda 함수 및 커넥터와 Greengrass 시스템 구성 요소에 필요한 AWS 리소스에 권한을 부여해야 합니다.

> **Note**
>
> 혼동된 대리자 보안 문제를 방지하려면 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키도 포함하는 것이 좋습니다. 조건 컨텍스트 키는 지정된 계정 및 Greengrass 작업 영역에서 들어오는 요청만 허용하도록 액세스를 제한합니다. 혼동된 대리자 문제에 대한 자세한 내용은, 교차 서비스 혼동된 대리자 예방를 참조하세요.

IAM 콘솔을 사용하여 역할과 권한을 만들고 구성합니다. Amazon DynamoDB 테이블에 대한 액세스를 허용하는 예제 역할을 만드는 단계는 the section called “그룹 역할 구성”을 참조하십시오. 일반적인 단계는 IAM 사용 설명서의 AWS 서비스에 대한 역할 생성(콘솔)을 참조하십시오.

역할을 구성한 후 AWS IoT 콘솔을 사용하여 그룹에 역할을 추가합니다.

> **Note**
>
> 이 절차는 그룹에 대한 역할을 선택하는 경우에만 필요합니다. 현재 선택한 그룹 역할의 권한을 변경한 후에는 필요하지 않습니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 그룹 구성 페이지에서 설정을 선택합니다.
4. 그룹 역할에서 역할을 추가하거나 변경하도록 선택합니다.
   - 역할을 추가하려면 역할 연결을 선택한 다음 역할 목록에서 역할을 선택합니다. AWS 계정에서 AWS IoT Greengrass를 신뢰할 수 있는 엔터티로 정의하는 역할입니다.
   - 다른 역할을 선택하려면 역할 편집을 선택한 다음 역할 목록에서 역할을 선택합니다.
5. Save를 선택합니다.

Greengrass 그룹 역할 제거(콘솔)

Greengrass 그룹에서 역할을 분리하려면 다음 단계를 수행하십시오.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 그룹 구성 페이지에서 설정을 선택합니다.
4. 그룹 역할에서 역할 연결 해제를 선택합니다.
5. 확인 대화 상자에서 연결 해제를 선택합니다. 이 단계는 그룹에서 역할을 제거하지만 역할은 삭제하지 않습니다. 역할을 삭제하려면 IAM 콘솔을 사용합니다.

Greengrass 그룹 역할 관리(CLI)

다음과 같은 역할 관리 작업에서 AWS CLI을 사용할 수 있습니다.

- Greengrass 그룹 역할 가져오기
- Greengrass 그룹 역할 생성
- Greengrass 그룹 역할 제거

Greengrass 그룹 역할 가져오기(CLI)

Greengrass 그룹에 연결된 역할이 있는지 확인하려면 다음 단계를 수행하십시오.

1. 그룹 목록에서 대상 그룹의 ID를 가져옵니다.

   ```
   aws greengrass list-groups
   ```

   다음은 `list-groups` 응답의 예입니다. 응답의 각 그룹에는 그룹 ID가 포함된 `Id` 속성이 포함됩니다.

   ```
   {
       "Groups": [
           {
               "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
   groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
   a50e-7d356EXAMPLE",
               "Name": "MyFirstGroup",
               "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
               "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
               "CreationTimestamp": "2019-11-11T05:47:31.435Z",
   ```

```
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

query 옵션을 사용하여 결과를 필터링하는 예제를 비롯한 자세한 내용은 the section called “그룹 ID 가져오기” 단원을 참조하십시오.

2. 출력에서 대상 그룹의 Id을 복사합니다.
3. 그룹 역할을 가져옵니다. *group-id*를 대상 그룹의 ID로 바꿉니다.

```
aws greengrass get-associated-role --group-id group-id
```

역할이 Greengrass 그룹과 연결된 경우, 다음 역할 메타데이터가 반환됩니다.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

그룹에 연결된 역할이 없으면 다음 오류가 반환됩니다.

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
 attach an IAM role to this deployment group.
```

Greengrass 그룹 역할 생성(CLI)

역할을 생성하고 Greengrass 그룹에 연결하려면 다음 단계를 수행하십시오.

IAM을 사용하여 그룹 역할을 생성하려면

1. AWS IoT Greengrass이 역할을 수임하도록 허용하는 신뢰 정책이 있는 역할을 생성합니다. 이 예제에서는 `MyGreengrassGroupRole`라는 역할을 생성하지만, 다른 이름을 사용할 수 있습니다. 혼동된 대리자 보안 문제를 방지하려면 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키도 포함하는 것이 좋습니다. 조건 컨텍스트 키는 지정된 계정 및 Greengrass 작업 영역에서 들어오는 요청만 허용하도록 액세스를 제한합니다. 혼동된 대리자 문제에 대한 자세한 내용은, 교차 서비스 혼동된 대리자 예방를 참조하세요.

   Linux, macOS, or Unix

   ```
   aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "StringEquals": {
             "aws:SourceAccount": "account-id"
           },
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/
   groups/group-id"
           }
         }
       }
     ]
   }'
   ```

Windows command prompt

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:/greengrass/groups/group-id\"},
\"StringEquals\":{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

2. 출력의 역할 메타데이터에서 역할 ARN을 복사합니다. ARN을 사용하여 역할을 그룹과 연결합니다.
3. 비즈니스 사례를 지원할 관리형 또는 인라인 정책을 역할에 연결합니다. 예를 들어 사용자 정의 Lambda 함수를 Amazon S3에서 읽는 경우 `AmazonS3ReadOnlyAccess` 관리형 정책을 역할에 연결할 수 있습니다.

```
aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
 arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

성공하면 응답이 반환되지 않습니다.

역할을 Greengrass 그룹과 연결하려면

1. 그룹 목록에서 대상 그룹의 ID를 가져옵니다.

```
aws greengrass list-groups
```

다음은 `list-groups` 응답의 예입니다. 응답의 각 그룹에는 그룹 ID가 포함된 `Id` 속성이 포함됩니다.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
```

```
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

query 옵션을 사용하여 결과를 필터링하는 예제를 비롯한 자세한 내용은 the section called “그룹 ID 가져오기” 단원을 참조하십시오.

2. 출력에서 대상 그룹의 Id을 복사합니다.
3. 역할을 그룹과 연결합니다. *group-id*를 대상 그룹의 ID로 바꾸고 *role-arn*을 그룹 역할의 ARN으로 바꿉니다.

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

성공하면 다음 응답이 반환됩니다.

```
{
  "AssociatedAt": "timestamp"
}
```

Greengrass 그룹 역할 제거(CLI)

Greengrass 그룹에서 그룹 역할의 연결을 해제하려면 다음 단계를 수행하십시오.

1. 그룹 목록에서 대상 그룹의 ID를 가져옵니다.

```
aws greengrass list-groups
```

다음은 `list-groups` 응답의 예입니다. 응답의 각 그룹에는 그룹 ID가 포함된 `Id` 속성이 포함됩니다.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

`query` 옵션을 사용하여 결과를 필터링하는 예제를 비롯한 자세한 내용은 the section called “그룹 ID 가져오기” 단원을 참조하십시오.

2. 출력에서 대상 그룹의 `Id`을 복사합니다.
3. 그룹에서 역할의 연결을 해제합니다. *`group-id`*를 대상 그룹의 ID로 바꿉니다.

```
aws greengrass disassociate-role-from-group --group-id group-id
```

성공하면 다음 응답이 반환됩니다.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> 그룹 역할을 사용하지 않는 경우 삭제할 수 있습니다. 먼저 delete-role-policy를 사용하여 역할에서 각 관리형 정책을 연결 해제하고, delete-role을 사용하여 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서에서 역할 또는 인스턴스 프로필 삭제를 참조하세요.

다음 사항도 참조하세요.

- IAM 사용자 설명서의 관련 주제
  - 역할을 생성하여 AWS 서비스에 대한 권한 위임
  - 역할 변경
  - IAM 자격 증명 권한 추가 및 제거
  - 역할 또는 인스턴스 프로파일 삭제
- AWS CLI명령 참조에서 사용 가능한 AWS IoT Greengrass 명령
  - list-groups
  - associate-role-to-group
  - disassociate-role-from-group
  - get-associated-role
- AWS CLI명령 참조에서 사용 가능한 IAM 명령
  - attach-role-policy

- create-role
- delete-role
- delete-role-policy

교차 서비스 혼동된 대리자 예방

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에 작업을 수행하도록 강요할 수 있는 보안 문제입니다. AWS에서는 교차 서비스 가장으로 인해 혼동된 대리자 문제가 발생할 수 있습니다. 교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출되는 서비스)를 호출할 때 발생할 수 있습니다. 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해 AWS에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다.

AWS IoT Greengrass가 리소스에 다른 서비스를 제공하는 권한을 제한하려면 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하는 것이 좋습니다. 두 글로벌 조건 컨텍스트 키를 모두 사용하는 경우 `aws:SourceAccount` 값과 `aws:SourceArn` 값의 계정은 동일한 정책 문에서 사용할 경우 동일한 계정 ID를 사용해야 합니다.

`aws:SourceArn`의 값은 `sts:AssumeRole` 요청과 관련된 Greengrass 고객 리소스여야 합니다.

혼동된 대리자 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용하는 것입니다. 리소스의 전체 ARN을 모를 경우 또는 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분에 대해 와일드카드(*)를 포함한 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용합니다. 예: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:*`.

`aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하는 정책의 추가 예는 다음 주제를 참조하세요.

- Greengrass 서비스 역할 생성
- Greengrass 그룹 역할 생성
- 벌크 배포를 위한 IAM 실행 역할 생성 및 구성

AWS IoT Greengrass 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 AWS IoT Greengrass 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS CLI 또는 AWS API를 사용해 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 태스크를 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 AWS IoT Greengrass 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 따르세요.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 관리형 정책은 AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS managed policies(관리형 정책) 또는 AWS managed policies for job functions(직무에 대한 관리형 정책)를 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우 태스크를 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 사용 설명서에 있는 Policies and permissions in IAM(IAM의 정책 및 권한)을 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. 특정 AWS 서비스(예: AWS CloudFormation)를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 검증하여 안전하고 기능적인 권한 보장 – IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 검증합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 권장 사항을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 정보는 IAM 사용 설명서의 IAM Access Analyzer policy validation(IAM Access Analyzer 정책 검증)을 참조하세요.
- 다중 인증(MFA) 필요 – AWS 계정 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 설정합니다. API 작업을 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용 설명서의 Configuring MFA-protected API access(MFA 보호 API 액세스 구성)를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

AWS IoT Greengrass의 AWS 관리형 정책

AWS IoT Greengrass는 IAM 사용자와 역할에 권한을 부여하는 데 사용할 수 있는 다음과 같은 AWS 관리형 정책을 유지 관리합니다.

Policy	설명
AWSGreengrassFullAccess	모든 리소스에 대해 AWS IoT Greengrass 및 AWS 작업을 허용합니다. 이 정책은 AWS IoT Greengrass 서비스 관리자 또는 테스트 목적으로 권장됩니다.
AWSGreengrassReadOnlyAccess	모든 AWS 리소스에 대해 `List` 및 `Get` AWS IoT Greengrass 작업을 허용합니다.
AWSGreengrassResourceAccessRolePolicy	AWS 서비스에서 AWS Lambda 및 AWS IoT IoT 디바이스 섀도우를 포함한 리소스에 액세스할 수 있습니다. Greengrass 서비스 역할에 사용되는 기본 정책입니다. 이 정책은 일반적인 접근성을 제공하도록 설계되었습니다. 보다 제한적인 사용자 지정 정책을 정의할 수 있습니다.
GreengrassOTAUpdateArtifactAccess	모든 AWS 리전에서 AWS IoT Greengrass 코어 소프트웨어의 무선 업데이트(OTA) 아티팩트에 읽기 전용 액세스를 허용합니다.

정책 예

다음 예제의 고객 정의 정책은 일반적인 시나리오에 대한 권한을 부여합니다.

예시

- 사용자가 자신이 권한을 볼 수 있도록 허용

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

사용자가 자신이 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 태스크를 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": "*"
        }
    ]
}
```

AWS IoT Greengrass의 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 AWS IoT Greengrass 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

문제

- AWS IoT Greengrass에서 작업을 수행할 권한이 없음
- 오류: Greengrass is not authorized to assume the Service Role associated with this account 또는 오류: Failed: TES service role is not associated with this account.
- 오류: 역할을 사용하려고 할 때 권한이 거부되었습니다.<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.
- 디바이스 섀도우가 클라우드와 동기화하지 않습니다.
- iam:PassRole을 수행하도록 인증되지 않음
- 관리자인데, 다른 사용자가 AWS IoT Greengrass에 액세스할 수 있게 허용하려고 합니다
- 내 AWS 계정 외부의 사람이 내 AWS IoT Greengrass 리소스에 액세스할 수 있게 허용하기를 원합니다.

일반적인 문제 해결 도움말은 문제 해결 단원을 참조하십시오.

AWS IoT Greengrass에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 수신되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` IAM 사용자가 코어 정의 버전에 대한 세부 정보를 보려고 하지만 `greengrass:GetCoreDefinitionVersion` 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

이 경우 Mateo는 `arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` 태스크를 사용하여 `greengrass:GetCoreDefinitionVersion` 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

오류: Greengrass is not authorized to assume the Service Role associated with this account 또는 오류: Failed: TES service role is not associated with this account.

해결책: 배포에 실패하면 이 오류가 발생할 수 있습니다. Greengrass 서비스 역할이 현재 AWS 리전의 AWS 계정 계정과 연결되어 있는지 확인합니다. 자세한 내용은 the section called “서비스 역할 관리(CLI)” 또는 the section called “서비스 역할 관리(콘솔)” 섹션을 참조하세요.

오류: 역할을 사용하려고 할 때 권한이 거부되었습니다.<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.

해결책: 무선(OTA) 업데이트가 실패하면 이 오류가 발생할 수 있습니다. 서명자 역할 정책에서 대상 AWS 리전을 `Resource`로서 추가합니다. 이 서명자 역할은 AWS IoT Greengrass 소프트웨어 업데이트의 S3 URL을 미리 서명하는 데 사용됩니다. 자세한 내용은 S3 URL 서명자 역할을 참조하십시오.

디바이스 섀도우가 클라우드와 동기화하지 않습니다.

해결 방법: AWS IoT Greengrass에 Greengrass 서비스 역할의 `iot:UpdateThingShadow` 및 `iot:GetThingShadow` 작업에 대한 권한이 있는지 확인하십시오. 서비스 역할이 `AWSGreengrassResourceAccessRolePolicy` 관리형 정책을 사용하는 경우 이러한 권한은 기본적으로 포함됩니다.

섀도우 동기화 제한 시간 문제 해결 섹션을 참조하세요.

다음은 AWS IoT Greengrass에서 작업할 때 발생할 수 있는 일반적인 IAM 문제입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS IoT Greengrass에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신, 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예시 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 AWS IoT Greengrass에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스에 서비스 역할이 부여한 권한이 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
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 보안 인증 정보를 제공한 사람입니다.

관리자인데, 다른 사용자가 AWS IoT Greengrass에 액세스할 수 있게 허용하려고 합니다

다른 사용자가 AWS IoT Greengrass에 액세스하도록 허용하려면 액세스 권한이 필요한 사용자나 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 다른 사용자들은 해당 엔터티에 대한 자격 증명을 사용해 AWS에 액세스합니다. 그런 다음 AWS IoT Greengrass에 대한 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다.

바로 시작하려면 IAM 사용 설명서의 첫 번째 IAM 위임 사용자 및 그룹 생성을 참조하세요.

내 AWS 계정 외부의 사람이 내 AWS IoT Greengrass 리소스에 액세스할 수 있게 허용하기를 원합니다.

다른 계정의 사용자 또는 조직 외부의 사람이 AWS 리소스에 액세스하는 데 사용할 수 있는 IAM 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 자세한 내용은 IAM 사용자 설명서에서 소유하고 있는 또다른 AWS 계정에서 IAM 사용자에게 액세스 권한 제공 및 타사가 소유한 Amazon Web Services 계정에 대한 액세스 제공을 참조하십시오.

AWS IoT Greengrass는 리소스 기반 정책 또는 ACL(액세스 제어 목록)을 기반으로 하는 교차 계정 액세스를 지원하지 않습니다.

AWS IoT Greengrass의 규정 준수 확인

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면AWS 서비스 규정 준수 프로그램의AWS 서비스 범위별, 규정 참조하여 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 AWS 규정 준수 프로그램AWS 보증 프로그램 규정AWS 참조하십시오.

를 사용하여 AWS Artifact타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- 보안 및 규정 준수 퀵 스타트 가이드 - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.

  > **Note**
  >
  > 모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하십시오.

- AWS 규정 준수 리소스AWS — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub— 이를 AWS 서비스 통해 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하십시오.
- Amazon GuardDuty — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- AWS Audit Manager— 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

AWS IoT Greengrass의 복원성

AWS 글로벌 인프라는 Amazon Web Services 리전 및 가용 영역을 중심으로 구축됩니다. 각 AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간,

높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

Amazon Web Services 리전 및 가용 영역에 대한 자세한 내용을 알아보려면 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 뿐만 아니라 AWS IoT Greengrass도 데이터 복원력과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

- 코어의 인터넷 연결이 끊어져도 클라이언트 디바이스는 로컬 네트워크를 통해 계속 통신할 수 있습니다.
- AWS 클라우드 대상으로 향하는 처리되지 않은 메시지를 인 메모리 스토리지 대신 로컬 스토리지 캐시에 저장하도록 코어를 구성할 수 있습니다. 로컬 스토리지 캐시는 코어를 재시작해도(예: 그룹 배포 또는 디바이스 재부팅 후) 지속될 수 있으므로 AWS IoT Greengrass는 AWS IoT Core로 전달되는 메시지를 계속 처리할 수 있습니다. 자세한 내용은 the section called “MQTT 메시지 대기열” 섹션을 참조하세요.
- AWS IoT Core 메시지 브로커와 영구 세션을 설정하도록 코어를 구성할 수 있습니다. 이렇게 하면 코어가 오프라인 상태일 때 전송된 메시지를 수신할 수 있습니다. 자세한 내용은 the section called “AWS IoT Core를 사용하는 MQTT 영구 세션” 섹션을 참조하세요.
- 로컬 파일 시스템 및 CloudWatch Logs에 로그를 기록하도록 Greengrass 그룹을 구성할 수 있습니다. 코어의 연결이 끊어지면 로컬 로깅을 계속할 수 있지만 제한된 재시도 횟수로 CloudWatch 로그가 전송됩니다. 재시도 횟수가 모두 사용된 후에는 이벤트가 삭제됩니다. 로깅 제한도 알고 있어야 합니다.
- 스트림 관리자 스트림을 읽고 로컬 스토리지 대상으로 데이터를 전송하는 Lambda 함수를 작성할 수 있습니다.

AWS IoT Greengrass의 인프라 보안

관리형 서비스인 AWS IoT Greengrass는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 AWS IoT Greengrass에 액세스합니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

AWS IoT Greengrass 환경에서 디바이스는 X.509 인증서와 암호화 키를 사용하여 AWS 클라우드에 연결하고 인증합니다. 자세한 내용은 the section called “디바이스 인증 및 권한 부여” 섹션을 참조하세요.

AWS IoT Greengrass의 구성 및 취약성 분석

IoT 환경은 다양한 기능을 수행하고 장기적으로 사용되며 지리적으로 분산된 다수의 디바이스로 구성될 수 있습니다. 이러한 특성으로 인해 디바이스 설정이 복잡해지고 오류가 발생하기 쉬워집니다. 디바이스가 컴퓨팅 파워, 메모리 및 스토리지 기능에서 제한되는 경우가 있으므로 디바이스 자체에서 암호화 및 다른 보안 형태의 사용이 제한됩니다. 또한 디바이스에서 알려진 취약성이 있는 소프트웨어를 사용하는 경우도 있습니다. 이러한 요소로 인해 IoT 디바이스가 해커의 매력적인 대상이 되며, 디바이스를 지속적으로 보호하기 어렵게 됩니다.

AWS IoT Device Defender는 보안 문제 및 모범 사례와의 차이를 식별하는 도구를 제공하여 이러한 문제를 해결합니다. AWS IoT Device Defender를 사용하여 연결된 디바이스를 분석, 감사 및 모니터링하여 비정상적인 동작을 감지하고 보안 위험을 완화할 수 있습니다. AWS IoT Device Defender는 디바이스를 감사하여 보안 모범 사례를 준수하고 디바이스에서 비정상적인 동작을 감지할 수 있습니다. 따라서 디바이스 전반에 걸쳐 일관된 보안 정책을 시행하고 디바이스가 손상된 경우 신속하게 대응할 수 있습니다. AWS IoT Core와(과) 연결하면 AWS IoT Greengrass이(가) AWS IoT Device Defender 기능과 함께 사용할 수 있는 예측 가능한 클라이언트 ID를 생성합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 AWS IoT Device Defender을 참조하세요.

AWS IoT Greengrass 환경에서는 다음 고려 사항을 알고 있어야 합니다.

- 물리적 디바이스, 디바이스의 파일 시스템 및 로컬 네트워크를 보호하는 것은 사용자의 책임입니다.
- Greengrass 컨테이너에서 실행되는지 여부에 관계없이 AWS IoT Greengrass은(는) 사용자 정의 Lambda 함수에 대한 네트워크 격리를 시행하지 않습니다. 따라서 Lambda 함수가 시스템 또는 네트워크를 통해 외부에서 실행 중인 다른 프로세스와 통신할 수 있습니다.

Greengrass 코어 디바이스를 제어할 수 없는 경우 연결된 디바이스가 코어로 데이터를 전송하지 못하도록 하려면 다음을 수행하십시오.

1. Greengrass 그룹에서 Greengrass 코어를 제거합니다.
2. 그룹 CA 인증서를 교체합니다. AWS IoT 콘솔에서는 그룹의 설정 페이지에서 CA 인증서를 교체할 수 있습니다. AWS IoT GreengrassAPI에서 CreateGroupCertificateAuthority작업을 사용할 수 있습니다.

   또한 코어 디바이스의 하드 드라이브가 도난에 취약할 경우 전체 디스크 암호화를 사용하는 것이 좋습니다.

AWS IoT Greengrass 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트을 생성하여 VPC와 AWS IoT Greengrass 간에 프라이빗 연결을 설정할 수 있습니다. 이 엔드포인트를 사용하여 AWS IoT Greengrass 서비스의 그룹, Lambda 함수, 배포 및 기타 리소스를 관리할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 비공개로 AWS IoT Greengrass API에 액세스할 수 있도록 지원하는 AWS PrivateLink 기술로 구동됩니다. VPC의 인스턴스는 AWS IoT Greengrass API와 통신하는 데 퍼블릭 IP 주소를 필요로 하지 않습니다. VPC와 AWS IoT Greengrass 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

> **Note**
>
> 현재는 VPC 내에서 Greengrass 코어 디바이스가 완전히 작동하도록 구성할 수 없습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

주제

- AWS IoT Greengrass VPC 엔드포인트 고려 사항
- AWS IoT Greengrass 컨트롤 플레인 작업을 위한 인터페이스 VPC 엔드포인트 생성
- AWS IoT Greengrass에 대한 VPC 엔드포인트 정책 생성

AWS IoT Greengrass VPC 엔드포인트 고려 사항

AWS IoT Greengrass에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한 사항을 검토하세요. 추가적으로, 다음 사항을 고려하세요.

- AWS IoT Greengrass은 VPC에서 모든 컨트롤 플레인 API 작업에 대한 직접 호출 수행을 지원합니다. 컨트롤 플레인에는 CreateDeployment, StartBulkDeployment와 같은 작업이 포함됩니다. 컨트롤 플레인은 데이터 영역 작업인 GetDeployment, Discover와 같은 작업을 포함하지 않습니다.
- AWS IoT Greengrass용 VPC 엔드포인트는 현재 AWS 중국 리전에서 지원되지 않습니다.

AWS IoT Greengrass 컨트롤 플레인 작업을 위한 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 AWS IoT Greengrass 컨트롤 플레인에 대한 VPC 엔드포인트를 생성할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

다음 서비스 이름을 사용하여 AWS IoT Greengrass용 VPC 종단점을 생성합니다.

- com.amazonaws.*region*.greengrass

엔드포인트에 프라이빗 DNS를 사용하도록 설정하는 경우, 리전에 대한 기본 DNS 이름(예: `greengrass.us-east-1.amazonaws.com`)을 사용하여 AWS IoT Greengrass에 API 요청을 할 수 있습니다. 프라이빗 DNS는 기본적으로 활성화되어 있습니다.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하세요.

AWS IoT Greengrass에 대한 VPC 엔드포인트 정책 생성

AWS IoT Greengrass 컨트롤 플레인 운영에 대한 컨트롤 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체.
- 보안 주체가 수행할 수 있는 작업입니다.
- 보안 주체가 작업을 수행할 수 있는 리소스입니다.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

Example 예제: AWS IoT Greengrass 작업에 대한 VPC 엔드포인트 정책

다음은 AWS IoT Greengrass에 대한 엔드포인트 정책의 예입니다. 이 정책은 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에 대한 액세스 권한을 나열된 AWS IoT Greengrass 작업에 부여합니다.

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:StartBulkDeployment"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS IoT Greengrass의 보안 모범 사례

이 주제에는 AWS IoT Greengrass에 대한 보안 모범 사례가 포함되어 있습니다.

가능한 최소 권한 부여

IAM 역할에 최소 권한 집합을 사용하여 최소 권한 원칙을 따릅니다. IAM 정책의 `Action` 및 `Resource` 속성에 대한 `*` 와일드카드 사용을 제한합니다. 대신, 가능한 경우 한정된 작업과 리소스를 선언합니다. 최소 권한 및 기타 정책 모범 사례에 대한 자세한 내용은 the section called “정책 모범 사례”을(를) 참조하세요.

최소 권한 모범 사례는 Greengrass 코어 및 연결된 디바이스에 연결하는 AWS IoT 정책에도 적용됩니다.

Lambda 함수에서 보안 인증을 하드 코딩하지 않음

사용자 정의 Lambda 함수에서 보안 인증을 하드 코딩하지 마십시오. 자격 증명에 대한 보호를 강화하려면 다음을 수행하십시오.

- AWS 서비스와 상호 작용하려면 Greengrass 그룹 역할에서 특정 작업 및 리소스에 대한 권한을 정의합니다.
- 로컬 암호를 사용하여 자격 증명을 저장합니다. 또는 함수가 AWS SDK를 사용하는 경우 기본 보안 인증 공급자 체인의 보안 인증을 사용합니다.

민감한 정보를 기록하지 않음

자격 증명 및 기타 개인 식별 정보(PII)의 로깅을 방지해야 합니다. 코어 디바이스의 로컬 로그에 액세스하려면 루트 권한이 필요하고 로그에 액세스하려면 IAM 권한이 필요하지만 다음 보호 장치를 구현하는 것이 CloudWatch 좋습니다.

- MQTT 주제 경로에는 민감한 정보를 사용하지 마십시오.
- AWS IoT Core 레지스트리의 디바이스(사물) 이름, 유형 및 속성에 민감한 정보를 사용하지 마십시오.
- 사용자 정의 Lambda 함수에 민감한 정보를 로그하지 마십시오.
- Greengrass 리소스의 이름과 ID에 민감한 정보를 사용하지 마십시오.
  - 커넥터
  - 코어
  - 디바이스
  - 함수
  - 그룹
  - Loggers
  - 리소스(로컬, 기계 학습 또는 암호)
  - 구독

대상 구독 만들기

구독은 서비스, 디바이스, Lambda 함수 사이에 메시지가 교환되는 방식을 정의하여 Greengrass 그룹의 정보 흐름을 제어합니다. 애플리케이션이 의도한 동작만 수행하도록 하려면, 구독을 통해 게시자가 특정 주제에만 메시지를 보내도록 하고, 구독자가 기능에 필요한 주제에서만 메시지를 받도록 제한해야 합니다.

디바이스의 시계를 동기화 상태로 유지

디바이스에서는 정확한 시간을 유지하는 것이 중요합니다. X.509 인증서에는 만료 날짜와 시간이 있습니다. 디바이스의 시계는 서버 인증서가 여전히 유효한지 확인하는 데 사용됩니다. 디바이스 시계는 시간이 지나 드리프트 상태가 되거나 배터리가 방전될 수 있습니다.

자세한 내용은 AWS IoT Core 개발자 안내서의 디바이스 시계를 동기화된 상태로 유지 모범 사례를 참조하십시오.

Greengrass 코어를 사용한 디바이스 인증 관리

클라이언트 디바이스는 FreeRTOS를 실행하거나 AWS IoT디바이스 SDK 또는 AWS IoT Greengrass 검색 API를 사용하여 동일한 Greengrass 그룹의 코어에 연결하고 인증하는 데 사용되는 검색 정보를 가져올 수 있습니다. 검색 정보에는 다음이 포함됩니다.

- 클라이언트 디바이스와 동일한 Greengrass 그룹에 있는 Greengrass 코어에 대한 연결 정보입니다. 이 정보에는 코어 디바이스에 대한 각 엔드포인트의 호스트 주소와 포트 번호가 포함됩니다.
- 로컬 MQTT 서버 인증서에 서명하는 데 사용되는 그룹 CA 인증서입니다. 클라이언트 디바이스는 그룹 CA 인증서를 사용하여 코어가 제공하는 MQTT 서버 인증서를 검증합니다.

다음은 연결된 클라이언트 디바이스가 Greengrass 코어를 사용해 상호 인증을 관리하는 모범 사례입니다. 이러한 모범 사례는 핵심 디바이스가 손상된 경우 위험을 완화하는 데 도움이 될 수 있습니다.

각 연결에 대한 로컬 MQTT 서버 인증서를 검증합니다.

  클라이언트 디바이스는 코어와 연결할 때마다 코어가 제공하는 MQTT 서버 인증서를 검증해야 합니다. 이 검증은 코어 디바이스와 연결된 디바이스 간 상호 인증의 클라이언트 디바이스 측입니다. 클라이언트 디바이스는 장애를 감지하고 연결을 종료할 수 있어야 합니다.

검색 정보를 하드코딩하지 마십시오.

  코어가 고정 IP 주소를 사용하는 경우에도 클라이언트 디바이스는 검색 작업에 의존하여 코어 연결 정보와 그룹 CA 인증서를 가져와야 합니다. 클라이언트 디바이스는 이 검색 정보를 하드 코딩해서는 안 됩니다.

검색 정보를 주기적으로 업데이트합니다.

  클라이언트 디바이스는 주기적으로 검색을 실행하여 코어 연결 정보 및 그룹 CA 인증서를 업데이트해야 합니다. 클라이언트 디바이스가 코어와 연결하기 전에 이 정보를 업데이트하는 것이 좋습니다. 검색 작업 간의 지속시간이 짧을수록 노출 가능 시간을 최소화할 수 있습니다. 따라서 정기적으로 클라이언트 디바이스의 연결을 끊었다가 다시 연결하여 업데이트를 트리거하는 것이 좋습니다.

Greengrass 코어 디바이스를 제어할 수 없는 경우 연결된 디바이스가 코어로 데이터를 전송하지 못하도록 하려면 다음을 수행하십시오.

1. Greengrass 그룹에서 Greengrass 코어를 제거합니다.
2. 그룹 CA 인증서를 교체합니다. AWS IoT 콘솔에서는 그룹의 설정 페이지에서 CA 인증서를 교체할 수 있습니다. AWS IoT GreengrassAPI에서 작업을 사용할 수 있습니다. CreateGroupCertificateAuthority

   또한 코어 디바이스의 하드 드라이브가 도난에 취약할 경우 전체 디스크 암호화를 사용하는 것이 좋습니다.

자세한 설명은 the section called “디바이스 인증 및 권한 부여” 섹션을 참조하세요.

다음 사항도 참조하십시오.

- AWS IoT 개발자 안내서의 AWS IoT Core 내 보안 모범 사례
- AWS공식 블로그 사물 인터넷의 산업용 IoT 솔루션을 위한 10가지 보안 황금률

AWS IoT Greengrass의 로깅 및 모니터링

모니터링은 AWS IoT Greengrass와 사용자 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 중요한 역할을 합니다. 발생하는 다중 지점 실패를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. AWS IoT Greengrass에 대한 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 작성해야 합니다

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

모니터링 도구

AWS는 AWS IoT Greengrass를 모니터링하는 데 사용할 수 있는 도구를 제공합니다. 이러한 도구 중 일부를 구성하여 모니터링을 수행할 수 있습니다. 일부 도구는 수동 개입이 필요합니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

다음과 같은 자동 모니터링 도구를 사용하여 AWS IoT Greengrass를 모니터링하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch Logs – AWS CloudTrail 또는 기타 소스의 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 로그 파일 모니터링을 참조하세요.
- AWS CloudTrail Log Monitoring – 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs에 전송하여 실시간으로 모니터링하며, Java에서 로그 처리 애플리케이션을 작성하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하세요.
- Amazon EventBridge – EventBridge 이벤트 규칙을 사용하여 Greengrass 그룹 배포 또는 CloudTrail로 로깅된 API 호출의 상태 변경에 대한 알림을 받습니다. 자세한 내용은 the section called “배포 알림 받기” 또는 Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 섹션을 참조하세요.
- Greengrass 시스템 상태 텔레메트리 — Greengrass 코어에서 전송된 텔레메트리 데이터를 수신하려면 구독하십시오. 자세한 내용은 the section called “시스템 상태 원격 측정 데이터 수집” 섹션을 참조하세요.

- 로컬 상태 확인 — 상태 API를 사용하여 코어 디바이스의 로컬 AWS IoT Greengrass 프로세스 상태에 대한 스냅샷을 얻을 수 있습니다. 자세한 내용은 the section called “로컬 상태 확인 API 직접 호출” 섹션을 참조하세요.

다음 사항도 참조하세요.

- the section called “AWS IoT Greengrass 로그를 사용하여 모니터링”
- the section called “AWS CloudTrail을 사용하여 AWS IoT Greengrass API 직접 호출 로깅”
- the section called “배포 알림 받기”

AWS IoT Greengrass 로그를 사용하여 모니터링

AWS IoT Greengrass은(는) 클라우드 서비스와 AWS IoT Greengrass 코어 소프트웨어로 구성됩니다. AWS IoT GreengrassCore 소프트웨어는 Amazon CloudWatch 및 코어 디바이스의 로컬 파일 시스템에 로그를 기록할 수 있습니다. 코어에서 실행되는 Lambda 함수 및 커넥터는 로그 및 로컬 파일 시스템에 로그를 CloudWatch 기록할 수도 있습니다. 로그를 사용하여 이벤트를 모니터링하고 문제를 해결할 수 있습니다. 모든 AWS IoT Greengrass 로그 항목에는 타임스탬프, 로그 수준 및 이벤트에 대한 정보가 포함됩니다. 로깅 설정에 대한 변경 사항은 그룹을 배포한 후에 적용됩니다.

로깅은 그룹 수준에서 구성됩니다. Greengrass 그룹에 대한 로깅을 구성하는 방법을 보여주는 단계는 the section called “AWS IoT Greengrass의 로깅 구성” 단원을 참조하십시오.

로그 액세스 CloudWatch

CloudWatch 로깅을 구성하면 Amazon CloudWatch 콘솔의 로그 페이지에서 로그를 볼 수 있습니다. AWS IoT Greengrass 로그의 로그 그룹은 다음과 같은 이름 지정 규칙을 사용합니다.

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

각 로그 그룹에는 다음 명명 규칙을 사용하는 로그 스트림이 포함되어 있습니다.

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

CloudWatch 로그를 사용할 때는 다음 고려 사항이 적용됩니다.

- 인터넷에 연결할 수 없는 경우 제한된 횟수의 재시도 횟수로 CloudWatch 로그가 Logs로 전송됩니다. 재시도 횟수가 모두 사용된 후에는 이벤트가 삭제됩니다.
- 트랜잭션, 메모리 및 기타 제한이 적용됩니다. 자세한 설명은 the section called “로깅 제한” 섹션을 참조하세요.
- Greengrass 그룹 역할은 AWS IoT Greengrass 로그에 쓸 수 있도록 허용해야 합니다. CloudWatch 권한을 부여하려면 그룹 역할에 다음 인라인 정책을 포함시킵니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

> **Note**
>
> 로그 리소스에보다 세분화된 액세스 권한을 부여할 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch 로그에 대한 ID 기반 정책 (IAM 정책) 사용을 참조하십시오.

그룹 역할은 사용자가 생성하고 Greengrass 그룹에 연결하는 IAM 역할입니다. 콘솔이나 AWS IoT Greengrass API를 사용하여 그룹 역할을 관리할 수 있습니다.

콘솔 사용

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 대상 그룹을 선택합니다.
3. 보기 설정을 선택합니다. 그룹 역할에서 그룹 역할을 보거나, 연결하거나, 연결을 끊을 수 있습니다.

   그룹 역할을 연결하는 방법을 보여주는 단계는 그룹 역할을 참조하십시오.

CLI 사용

- 그룹 역할을 찾으려면 명령을 사용하십시오. get-associated-role
- 그룹 역할을 연결하려면 associate-role-to-group명령을 사용합니다.
- 그룹 역할을 제거하려면 disassociate-role-from-group명령을 사용합니다.

이러한 명령과 함께 사용할 그룹 ID를 가져오는 방법은 the section called “그룹 ID 가져오기” 단원을 참조하십시오.

파일 시스템 로그 액세스

파일 시스템 로깅을 구성하면 로그 파일이 코어 디바이스에 *greengrass-root*/ggc/var/log 아래 저장됩니다. 다음은 상위 디렉터리 구조입니다.

```
greengrass-root/ggc/var/log
    - crash.log
    - system
        - log files for each Greengrass system component
    - user
        - region
            - account-id
                - log files generated by each user-defined Lambda function
            - aws
                - log files generated by each connector
```

> **Note**
>
> 기본적으로 *greengrass-root*는 /greengrass 디렉터리입니다. 쓰기 디렉터리가 구성되어 있는 경우에는 로그가 이 디렉터리 아래 있습니다.

파일 시스템 로그 사용 시 다음 사항을 고려하십시오.

- 파일 시스템에서 AWS IoT Greengrass 로그를 읽으려면 루트 권한이 필요합니다.
- AWS IoT Greengrass은(는) 로그 데이터의 양이 구성 한도에 근접할 때 크기 기반의 교체 및 자동 정리를 지원합니다.
- crash.log 파일은 파일 시스템 로그에서만 사용할 수 있습니다. 이 로그는 CloudWatch 로그에 기록되지 않습니다.
- 디스크 사용 제한이 적용됩니다. 자세한 설명은 the section called “로깅 제한” 섹션을 참조하세요.

> **Note**
>
> AWS IoT Greengrass 코어 소프트웨어 v1.0에 대한 로그는 *greengrass-root*/var/log 디렉터리 아래에 저장됩니다.

기본 로깅 구성

로깅 설정이 명시적으로 구성되어 있지 않으면 AWS IoT Greengrass은(는) 첫 번째 그룹 배포 후 다음과 같은 기본 로깅 구성을 사용합니다.

AWS IoT Greengrass 시스템 구성 요소

- Type - FileSystem
- 구성 요소 - GreengrassSystem
- Level - INFO
- 공간 - 128 KB

사용자 정의 Lambda 함수

- Type - FileSystem
- 구성 요소 - Lambda
- Level - INFO

- 공간 - 128 KB

> **Note**
>
> 첫 번째 배포 전에는 배포된 사용자 정의 Lambda 함수가 없기 때문에 시스템 구성 요소 쓰기만 파일 시스템에 로깅됩니다.

AWS IoT Greengrass의 로깅 구성

AWS IoT 또는 AWS IoT Greengrass API를 사용하여 AWS IoT Greengrass 로깅을 구성할 수 있습니다.

> **Note**
>
> AWS IoT Greengrass로그에 로그를 쓸 수 CloudWatch 있으려면 그룹 역할이 필요한 CloudWatch 로그 작업을 허용해야 합니다.

로깅 구성(콘솔)

그룹의 설정 페이지에서 로깅을 구성할 수 있습니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.
2. 로깅을 구성하려는 그룹을 선택합니다.
3. 그룹 구성 페이지에서 로그를 선택합니다.
4. 다음과 같이 로깅 위치를 선택합니다.

   - CloudWatch 로깅을 구성하려면 CloudWatch 로그 구성에서 편집을 선택합니다.
   - 파일 시스템 로깅을 구성하려면 로컬 로그 구성에서 편집을 선택합니다.

   한 위치 또는 두 위치 모두에 대한 로깅을 구성할 수 있습니다.
5. 로그 편집 구성 모달에서 Greengrass 시스템 로그 수준 또는 사용자 Lambda 함수 로그 수준을 선택합니다. 한 구성 요소 또는 두 구성 요소를 모두 선택할 수 있습니다.
6. 로깅하려는 최저 수준의 이벤트를 선택합니다. 이 임계값보다 낮은 이벤트는 필터링되어 저장되지 않습니다.

7. 저장을 선택합니다. 변경 사항은 그룹을 배포한 후에 적용됩니다.

로깅 구성(API)

AWS IoT Greengrass 로거 API를 사용하여 프로그래밍 방식으로 로깅을 구성할 수 있습니다. 예를 들어, `CreateLoggerDefinition` 작업을 사용하여 다음 구문을 사용하는 `LoggerDefinitionVersion` 페이로드를 기반으로 로거 정의를 생성합니다.

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
      "Space": "integer"
    },
    {
      "Id": "string",
      ...
    }
  ]
}
```

`LoggerDefinitionVersion`은 다음 속성이 있는 하나 이상의 `Logger` 객체 배열입니다.

`Id`

로거의 식별자입니다.

`Type`

로그 이벤트의 스토리지 메커니즘입니다. 를 사용하면 로그 이벤트가 CloudWatch Logs로 전송됩니다. `AWSCloudWatch` `FileSystem`이 사용되면 로그 이벤트는 로컬 파일 시스템에 저장됩니다.

유효값: `AWSCloudWatch`, `FileSystem`

`Component`

로그 이벤트의 원본입니다. `GreengrassSystem`이 사용되면 Greengrass 시스템 구성 요소의 이벤트가 로깅됩니다. `Lambda`가 사용되면 사용자 정의 Lambda 함수의 이벤트가 로깅됩니다.

유효값: `GreengrassSystem`, `Lambda`

`Level`

로그 수준 임계값입니다. 이 임계값보다 낮은 로그 이벤트는 필터링되어 저장되지 않습니다.

유효한 값: `DEBUG`, `INFO`(권장), `WARN`, `ERROR`, `FATAL`

`Space`

로그를 저장하는 데 사용할 로컬 스토리지의 최대 용량(단위: KB)입니다. 이 필드는 `Type`이 `FileSystem`으로 설정된 경우에만 적용됩니다.

구성 예

다음 `LoggerDefinitionVersion` 예제는 다음과 같은 작업을 수행하는 로깅 구성을 지정합니다.

- AWS IoT Greengrass 시스템 구성 요소에 대한 파일 시스템 `ERROR` 및 그 이상 로깅을 켭니다.
- 사용자 정의 Lambda 함수에 대한 파일 시스템 `INFO` 및 그 이상 로깅을 켭니다.
- 사용자 정의 Lambda 함수에 대한 CloudWatch `INFO` (또는 그 이상) 로깅을 활성화합니다.

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
```

```
        "Type": "AWSCloudWatch"
      }
    ]
  }
}
```

로거 정의 버전을 생성한 후에는 그룹을 배포하기 전에 해당 버전 ARN을 사용하여 그룹 버전을 생성할 수 있습니다.

로깅 제한

AWS IoT Greengrass에는 다음과 같은 로깅 제한이 있습니다.

초당 트랜잭션 수

CloudWatch 로깅이 활성화되면 로깅 구성 요소가 이벤트를 전송하기 전에 로컬로 이벤트를 일괄 처리하므로 로그 스트림당 초당 5개 이상의 요청 속도로 로깅할 수 있습니다. CloudWatch

메모리

Lambda 함수가 로그를 전송하도록 구성되어 CloudWatch 있고 Lambda 함수가 초당 5MB를 초과하여 장기간 로깅하는 경우 AWS IoT Greengrass 내부 처리 파이프라인이 결국 꽉 차게 됩니다. 이론상 최악의 경우는 Lambda 함수 1개당 6MB입니다.

클록 스큐

CloudWatch 로깅이 활성화되면 로깅 구성 요소는 일반 서명 버전 4 서명 프로세스를 CloudWatch 사용하여 요청에 서명합니다. AWS IoT Greengrass 코어 디바이스에서 시스템 시간이 15분 이상 동기화되지 않을 경우, 해당 요청은 거부됩니다.

디스크 사용량

다음 공식을 사용하면 로깅 시 총 디스크 사용량의 최대값을 계산할 수 있습니다.

```
greengrass-system-component-space * 8    // 7 if automatic IP detection is disabled
  + 128KB                                // the internal log for the local logging
 component
  + lambda-space * lambda-count          // different versions of a Lambda function are
 treated as one
```

위치:

`greengrass-system-component-space`

AWS IoT Greengrass 시스템 구성 요소 로그를 저장할 로컬 스토리지의 최대 용량입니다.

`lambda-space`

Lambda 함수 로그를 저장할 로컬 스토리지의 최대 용량입니다.

`lambda-count`

배포된 Lambda 함수의 수입니다.

로그 손실

AWS IoT Greengrass코어 디바이스가 로그인만 하도록 CloudWatch 구성되어 있고 인터넷에 연결되어 있지 않으면 현재 메모리에 있는 로그를 검색할 방법이 없습니다.

Lambda 함수가 종료되면 (예: 배포 중) 몇 초 분량의 로그가 기록되지 않습니다. CloudWatch

CloudTrail 로그

AWS IoT Greengrass는 AWS IoT Greengrass에서 사용자, 역할, 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail와 함께 실행됩니다. 자세한 내용은 the section called “AWS CloudTrail을 사용하여 AWS IoT Greengrass API 직접 호출 로깅”을(를) 참조하세요.

AWS CloudTrail을 사용하여 AWS IoT Greengrass API 직접 호출 로깅

AWS IoT Greengrass에서 사용자AWS CloudTrail, 역할 또는 서비스가 수행한 작업의 기록을 제공하는 AWS 서비스와 통합됩니다AWS IoT Greengrass. CloudTrail 모든 API 호출을 AWS IoT Greengrass 이벤트로 캡처합니다. 캡처되는 호출에는 AWS IoT Greengrass 콘솔로부터의 호출과 AWS IoT Greengrass API 작업에 대한 코드 호출이 포함됩니다. 트레일을 생성하면 에 대한 이벤트를 포함하여 Amazon S3 버킷으로 CloudTrail 이벤트를 지속적으로 전송할 수 AWS IoT Greengrass 있습니다. 트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근 이벤트를 계속 볼 수 있습니다. 에서 수집한 CloudTrail 정보를 사용하여 요청을 받은 사람AWS IoT Greengrass, 요청한 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 CloudTrail 내용은 AWS CloudTrail사용 설명서를 참조하십시오.

AWS IoT Greengrass에 대한 정보 CloudTrail

CloudTrail 계정을 만들 AWS 계정 때 활성화됩니다. 에서 AWS IoT Greengrass 활동이 발생하면 해당 활동이 CloudTrail 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 이벤트 기록으로 CloudTrail 이벤트 보기를 참조하십시오.

AWS IoT Greengrass에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. 트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 지역의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하십시오.

- 추적 생성 개요
- CloudTrail 지원되는 서비스 및 통합
- 에 대한 Amazon SNS 알림 구성 CloudTrail
- 여러 지역에서 CloudTrail 로그 파일 수신 및 여러 계정으로부터 CloudTrail 로그 파일 수신

모든 AWS IoT Greengrass 작업은 AWS IoT GreengrassAPI 참조에 의해 CloudTrail 기록되고 문서화됩니다. 예를 들어, `AssociateServiceRoleToAccount` `GetGroupVersionGetConnectivityInfo`, 및 `CreateFunctionDefinition` 작업에 대한 호출은 CloudTrail 로그 파일에 항목을 생성합니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 AWS Identity and Access Management(IAM) 사용자 보안 인증 정보로 했는지
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증 정보를 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 CloudTrail userIdentity 요소를 참조하십시오.

AWS IoT Greengrass 로그 파일 항목 이해

트레일은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 전송할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 `AssociateServiceRoleToAccount` 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:04:02Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "AssociateServiceRoleToAccount",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "errorCode": "BadRequestException",
    "requestParameters": null,
    "responseElements": {
        "Message": "That role ARN is invalid."
    },
    "requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
    "eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

다음 예제는 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다. `GetGroupVersion`

```
{
    "eventVersion": "1.05",
```

```
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-10-17T18:14:57Z"
            }
        },
        "invokedBy": "apimanager.amazonaws.com"
    },
    "eventTime": "2018-10-17T18:15:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetGroupVersion",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
        "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
    },
    "responseElements": null,
    "requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
    "eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

다음 예제는 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다. `GetConnectivityInfo`

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:02:12Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetConnectivityInfo",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "ThingName": "us-east-1_CIS_1539795000000_"
    },
    "responseElements": null,
    "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
    "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

다음 예제는 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다. `CreateFunctionDefinition`

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T18:01:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateFunctionDefinition",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "InitialVersion": "***"
    },
    "responseElements": {
        "CreationTimestamp": "2018-10-17T18:01:11.449Z",
        "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
```

```
        "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-
b983-ee5cfEXAMPLE",
        "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
        "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
        "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/
functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
    },
    "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
    "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


다음 사항도 참조하십시오.

- AWS CloudTrail 사용 설명서의 AWS CloudTrail란 무엇입니까?
- Amazon 사용 EventBridge 설명서를 사용하여 AWS API CloudTrail 호출에서 트리거되는 EventBridge 규칙 생성
- AWS IoT Greengrass API 참조

AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집

시스템 상태 원격 측정 데이터는 Greengrass 코어 디바이스의 중요 작업 성능을 모니터링하는 데 도움이 되는 진단 데이터입니다. Greengrass 코어의 원격 측정 에이전트는 로컬 원격 측정 데이터를 수집하여 고객 상호 작용 없이 Amazon EventBridge에 게시합니다. 코어 디바이스는 최선의 노력으로 EventBridge에 원격 측정 데이터를 게시합니다. 예를 들어, 코어 디바이스는 오프라인 상태에서 원격 측정 데이터를 제공하지 못할 수 있습니다.

> **Note**
>
> Amazon EventBridge는 애플리케이션을 Greengrass 코어 디바이스 및 배포 알림과 같은 다양한 소스의 데이터와 연결하는 데 사용할 수 있는 이벤트 버스 서비스입니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 단원을 참조하세요.

엣지 디바이스에서 원격 측정 데이터를 검색, 분석, 변환 및 보고하는 프로젝트와 애플리케이션을 만들 수 있습니다. 프로세스 엔지니어와 같은 도메인 전문가는 이러한 애플리케이션을 사용하여 플릿 상태에 대한 통찰력을 얻을 수 있습니다.

Greengrass 엣지 구성 요소가 제대로 작동하도록 하기 위해 AWS IoT Greengrass은(는) 개발 및 품질 개선 목적으로 데이터를 사용합니다. 또한 이 기능은 새롭고 향상된 엣지 기능을 알려주는 데도 도움이 됩니다. AWS IoT Greengrass은(는) 원격 측정 데이터를 최대 7일간 보존합니다.

이 기능은 AWS IoT Greengrass Core 소프트웨어 v1.11.0에서 사용할 수 있으며 기존 코어를 포함한 모든 Greengrass 코어에 기본적으로 활성화되어 있습니다. AWS IoT Greengrass Core 소프트웨어 v1.11.0 이상으로 업그레이드하는 즉시 자동으로 데이터를 받기 시작합니다.

게시된 원격 측정 데이터에 액세스하거나 관리하는 방법에 대한 자세한 내용은 the section called “원격 측정 데이터 수신 구독” 단원을 참조하세요.

원격 측정 에이전트는 다음 시스템 지표를 수집하고 게시합니다.

원격 측정 지표

이름	설명	소스
`SystemMemUsage`	운영 체제를 포함하여 Greengrass 코어 디바이스의 모든 애플리케이션에서 현재 사용 중인 메모리의 양입니다.	시스템
`CpuUsage`	운영 체제를 포함하여 Greengrass 코어 디바이스의 모든 애플리케이션에서 현재 사용 중인 CPU의 양입니다.	시스템
`TotalNumberOfFDs`	Greengrass 코어 디바이스의 운영 체제에 저장된 파일 디스크립터 수입니다. 하나의 파일 디스크립터는 열려 있는 파일 하나를 고유하게 식별합니다.	시스템
`LambdaOutOfMemory`	Lambda 함수에서 메모리 부족이 발생한 실행 횟수입니다.	시스템

이름	설명	소스
DroppedMessageCount	AWS IoT Core 대상 메시지 중 삭제된 메시지 수.	GGCloudSpooler 시스템 구성 요소
LambdaTimeout	사용자 정의 Lambda 함수 실행에 대한 시간 초과 횟수입니다.	사용자 정의 Lambda 함수, AWS 클라우드 및 시스템
LambdaUngracefullyKilled	사용자 정의 Lambda 함수가 완료하지 못한 실행 횟수입니다.	사용자 정의 Lambda 함수, AWS 클라우드 및 시스템
LambdaError	사용자 정의 Lambda 함수에서 오류 로그를 작성하는 실행 횟수입니다.	사용자 정의 Lambda 함수, AWS 클라우드 및 시스템
BytesAppended	스트림 관리자에 추가된 데이터의 바이트 수입니다.	GGStreamManager 시스템 구성 요소
BytesUploadedToIoTAnalytics	스트림 관리자가 AWS IoT Analytics에서 채널로 내보내는 데이터의 바이트 수입니다.	GGStreamManager 시스템 구성 요소
BytesUploadedToKinesis	스트림 관리자가 Amazon Kinesis Data Streams의 스트림으로 내보내는 데이터의 바이트 수입니다.	GGStreamManager 시스템 구성 요소
BytesUploadedToIoTSiteWise	스트림 관리자가 AWS IoT SiteWise에서 에셋 속성으로 내보내는 데이터의 바이트 수입니다.	GGStreamManager 시스템 구성 요소
BytesUploadedToS3ExportTaskExecutor	스트림 관리자가 Amazon S3의 객체로 내보내는 데이터의 바이트 수입니다.	GGStreamManager 시스템 구성 요소

이름	설명	소스
BytesUploadedToHTTP	스트림 관리자가 HTTP로 내보내는 데이터의 바이트 수입니다.	GGStreamManager 시스템 구성 요소

원격 측정 설정 구성

Greengrass 원격 측정은 다음 설정을 사용합니다.

- 원격 측정 에이전트는 1시간마다 원격 측정 데이터를 집계합니다.
- 원격 측정 에이전트는 24시간마다 원격 측정 메시지를 게시합니다.

> **Note**
>
> 설정은 변경할 수 없습니다.

Greengrass 코어 디바이스에 대한 원격 측정 기능을 활성화하거나 비활성화할 수 있습니다. AWS IoT Greengrass은(는) 섀도우를 사용하여 원격 측정 구성을 관리합니다. 코어가 AWS IoT Core와(과) 연결되면 변경 내용이 즉시 적용됩니다.

원격 측정 에이전트는 서비스 품질(QoS) 수준이 0인 MQTT 프로토콜을 사용하여 데이터를 게시합니다. 즉, 전송을 확인하거나 게시 시도를 재시도하지 않습니다. 원격 측정 메시지는 AWS IoT Core 대상으로 하는 구독에 대한 다른 메시지와 MQTT 연결을 공유합니다.

데이터 링크 비용을 제외하고 코어에서 AWS IoT Core(으)로의 데이터 전송에는 요금이 부과되지 않습니다. 이는 에이전트가 AWS 예약된 주제에 게시하기 때문입니다. 하지만 사용 사례에 따라 데이터를 받거나 처리할 때 비용이 발생할 수 있습니다.

요구 사항

원격 측정 설정을 구성할 때는 다음 요구 사항이 적용됩니다.

- AWS IoT Greengrass Core 소프트웨어 v1.11.0 이상을 사용해야 합니다.

> **Note**
>
> 이전 버전을 실행 중이고 원격 측정 기능을 사용하지 않으려는 경우, 아무 작업도 수행할 필요가 없습니다.

- 원격 측정 설정을 업데이트하기 전에 코어(사물) 섀도우를 업데이트하고 구성 API를 직접적으로 호출할 수 있는 IAM 권한을 제공해야 합니다.

  다음 예제 IAM 정책을 사용하면 특정 코어의 섀도우 및 런타임 구성을 관리할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowManageShadow",
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:DescribeThing"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Sid": "AllowManageRuntimeConfig",
            "Effect": "Allow",
            "Action": [
                "greengrass:GetCoreRuntimeConfiguration",
                "greengrass:UpdateCoreRuntimeConfiguration"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name"
            ]
        }
    ]
}
```

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해서). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

설정 원격 측정 구성(콘솔)

다음은 AWS IoT Greengrass 콘솔에서 Greengrass 코어의 원격 측정 설정을 업데이트하는 방법을 보여줍니다.

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1) 을 선택합니다.
2. Greengrass 그룹에서 대상 그룹을 선택합니다.
3. 그룹 구성 페이지의 개요 섹션에서 Greengrass 코어를 선택합니다.
4. 코어의 구성 페이지에서 원격 측정 탭을 선택합니다.
5. 시스템 상태 원격 측정 섹션에서 구성을 선택합니다.
6. 원격 측정 구성에서 원격 측정을 선택하여 원격 측정 상태를 활성화하거나 비활성화합니다.

> **Important**
>
> 기본적으로 원격 측정 기능은 AWS IoT Greengrass Core 소프트웨어 v1.11.0 이상에서 활성화됩니다.

변경 사항이 런타임에 적용됩니다. 그룹을 배포할 필요가 없습니다.

원격 측정 설정 구성(CLI)

AWS IoT Greengrass API에서 `TelemetryConfiguration` 객체는 Greengrass 코어의 원격 측정 설정을 나타냅니다. 이 개체는 코어와 연결된 `RuntimeConfiguration` 객체의 일부입니다. AWS IoT Greengrass API, AWS CLI 또는 AWS SDK를 사용하여 Greengrass 원격 측정을 관리할 수 있습니다. 이 단원의 예제에서는 AWS CLI를 사용합니다.

원격 측정 설정을 확인하려면

다음 명령은 Greengrass 코어의 원격 측정 설정을 가져옵니다.

- *`core-thing-name`*을 대상 코어의 이름으로 바꾸십시오.

  사물 이름을 가져오려면 get-core-defintion-version 명령을 사용합니다. 이 명령은 사물 이름이 포함된 사물의 ARN을 반환합니다.

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

이 명령은 JSON 응답에서 `GetCoreRuntimeConfigurationResponse` 객체를 반환합니다. 예:

```
{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "OutOfSync",
            "Telemetry": "On"
        }
    }
}
```

원격 측정 설정을 구성하려면

다음 명령은 Greengrass 코어의 원격 측정 설정을 업데이트합니다.

- *core-thing-name*을 대상 코어의 이름으로 바꾸십시오.

  사물 이름을 가져오려면 get-core-defintion-version 명령을 사용합니다. 이 명령은 사물 이름이 포함된 사물의 ARN을 반환합니다.

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration  '{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "InSync",
            "Telemetry": "Off"
        }
    }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus
\":\"InSync\",\"Telemetry\":\"Off\"}}"
```

`ConfigurationSyncStatus`이(가) `InSync`인 경우, 원격 측정 설정 변경 사항이 적용된 것입니다. 변경 사항이 런타임에 적용됩니다. 그룹을 배포할 필요가 없습니다.

TelemetryConfiguration 객체

`TelemetryConfiguration` 객체에는 다음 속성이 있습니다.

`ConfigurationSyncStatus`

원격 측정 설정이 동기화되어 있는지 확인합니다. 이 속성을 변경할 수 없습니다.

유형: 문자열

유효한 값: `InSync` 또는 `OutOfSync`

`Telemetry`

원격 측정 사용/사용 안 함을 설정합니다. 기본값은 `On`입니다.

유형: 문자열

유효한 값: `On` 또는 `Off`

원격 측정 데이터 수신 구독

Amazon EventBridge에서 Greengrass 코어 디바이스에서 게시된 원격 측정 데이터를 처리하는 방법을 정의하는 규칙을 생성할 수 있습니다. EventBridge는 데이터를 수신하면 규칙에 정의된 대상 작업을 간접적으로 호출합니다. 예를 들어 알림을 보내거나, 이벤트 정보를 저장하거나, 교정 작업을 수행하거나, 기타 이벤트를 간접적으로 호출하는 이벤트 규칙을 생성할 수 있습니다.

원격 측정 이벤트

원격 측정 데이터를 포함하는 배포 상태 변경에 대한 이벤트에서 사용하는 형식은 다음과 같습니다.

```
{
    "version": "0",
    "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
    "detail-type": "Greengrass Telemetry Data",
    "source": "aws.greengrass",
    "account": "123456789012",
    "time": "2020-07-28T20:45:53Z",
    "region": "us-west-1",
    "resources": [],
    "detail": {
        "ThingName": "CoolThing",
```

```
        "Schema": "2020-06-30",
        "ADP": [
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToKinesis",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToHTTP",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToIoTAnalytics",
                        "Sum": 11,
                        "U": "Bytes"
                    }
```

```
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToIoTSiteWise",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
            "M": [
                {
                    "N": "LambdaTimeout",
                    "Sum": 15,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "CloudSpooler",
            "M": [
                {
                    "N": "DroppedMessageCount",
                    "Sum": 15,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 1593727692,
            "NS": "SystemMetrics",
            "M": [
                {
                    "N": "SystemMemUsage",
                    "Sum": 11.23,
                    "U": "Megabytes"
                },
```

```
                    {
                        "N": "CpuUsage",
                        "Sum": 35.63,
                        "U": "Percent"
                    },
                    {
                        "N": "TotalNumberOfFDs",
                        "Sum": 416,
                        "U": "Count"
                    }
                ]
            },
            {
                "TS": 1593727692,
                "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
                "M": [
                    {
                        "N": "LambdaOutOfMemory",
                        "Sum": 12,
                        "U": "Count"
                    },
                    {
                        "N": "LambdaUngracefullyKilled",
                        "Sum": 100,
                        "U": "Count"
                    },
                    {
                        "N": "LambdaError",
                        "Sum": 7,
                        "U": "Count"
                    }
                ]
            }
        ]
    }
}
```

ADP 배열에는 다음과 같은 속성을 가진 집계된 데이터 포인트 목록이 포함됩니다.

TS

필수. 데이터가 집계된 시기의 타임스탬프입니다.

`NS`

필수. 시스템의 네임스페이스

`M`

필수. 지표 목록 이 지표는 다음 속성을 포함하고 있습니다.

`N`

지표의 이름입니다.

`Sum`

집계된 지표 값입니다. 원격 측정 에이전트는 이전 총계에 새 값을 추가하므로 합계는 계속 증가하는 값입니다. 타임스탬프를 사용하여 특정 집계의 값을 찾을 수 있습니다. 예를 들어 가장 최근에 집계된 값을 찾으려면 타임스탬프가 지정된 최신 값에서 이전 타임스탬프 값을 빼십시오.

`U`

지표 값의 단위입니다.

`ThingName`

필수. 대상으로 하는 사물 디바이스의 이름입니다.

EventBridge 규칙 생성을 위한 사전 조건

AWS IoT Greengrass에 대한 EventBridge 규칙을 생성하기 전에 다음을 수행해야 합니다.

- Eventbridge의 이벤트, 규칙, 대상을 숙지해야 합니다.
- EventBridge 규칙에 의해 호출되는 대상을 생성하고 구성해야 합니다. 규칙은 Amazon Kinesis 스트림, AWS Lambda, 함수, Amazon SNS 주제, Amazon SQS 대기열 등 다양한 유형의 대상을 간접적으로 호출할 수 있습니다.

  EventBridge 규칙 및 관련 대상은 Greengrass 리소스를 생성한 AWS 리전에 있어야 합니다. 자세한 내용은 AWS 일반 참조의 서비스 엔드포인트 및 할당량을 참조하십시오.

자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge란? 및 Amazon EventBridge 시작하기 단원을 참조하세요.

원격 측정 데이터를 가져오기 위한 이벤트 규칙 생성(콘솔)

다음 단계에 따라 AWS Management Console을(를) 사용하여 Greengrass 코어에서 게시한 원격 측정 데이터를 수신하는 EventBridge 규칙을 생성할 수 있습니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다. 자세한 내용은 Amazon EventBridge User Guide의 Creating an EventBridge Rule That Triggers on an Event from an AWS Resource를 참조하세요.

1. Amazon EventBridge 콘솔을 열고 규칙 생성을 선택합니다.
2. 이름 및 설명에 규칙의 이름과 설명을 입력합니다.
3. 이벤트 버스를 선택하고 선택한 이벤트 버스에서 규칙을 활성화합니다.
4. 규칙 유형을 선택하고 이벤트 패턴이 있는 규칙을 선택합니다.
5. 다음(Next)을 선택합니다.
6. 이벤트 소스(Event source)에서 AWS 이벤트 또는 EventBridge 파트너 이벤트(Events or EventBridge partner events)를 선택합니다.
7. 샘플 이벤트에서 AWS이벤트를 선택하고 Greengrass 원격 측정 데이터를 선택합니다.
8. 이벤트 패턴에서 다음을 선택합니다.
   a. 이벤트 소스(Event source)에서 AWS 서비스(services)를 선택합니다.
   b. AWS서비스에서 Greengrass를 선택합니다.
   c. 이벤트 유형에서 Greengrass 원격 측정 데이터를 선택합니다.
9. 다음(Next)을 선택합니다.
10. 대상 1에서 AWS 서비스를 선택합니다.
11. 대상 선택에서 SQS 대기열을 선택합니다.
12. 대기열에서 함수를 선택합니다.

원격 측정 데이터(CLI) 를 가져오기 위한 이벤트 규칙 생성

다음 단계에 따라 AWS CLI을(를) 사용하여 Greengrass 코어에서 게시한 원격 측정 데이터를 수신하는 EventBridge 규칙을 생성할 수 있습니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다.

1. 규칙을 생성합니다.
   - ## ##을 코어의 사물 이름으로바꾸십시오.

사물 이름을 가져오려면 get-core-defintion-version 명령을 사용합니다. 이 명령은 사물 이름이 포함된 사물의 ARN을 반환합니다.

```
aws events put-rule \
  --name TestRule \
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\":
 [\"thing-name\"]}}"
```

패턴에서 생략된 속성은 무시됩니다.

2. 규칙 대상으로 주제를 추가합니다. 다음 예제는 Amazon SQS를 사용하지만 다른 대상 유형을 구성할 수 있습니다.

   - *queue-arn*을 Amazon SQS 대기열의 ARN으로대체하십시오.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="queue-arn"
```

> **Note**
>
> Amazon EventBridge가 대상 주제를 간접적으로 호출하도록 허용하려면 대기열에 리소스 기반 정책을 추가해야 합니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon SQS 권한 단원을 참조하세요.

자세한 내용은 Amazon EventBridge 사용 설명서에서 EventBridge의 이벤트 및 이벤트 패턴 단원을 참조하세요.

AWS IoT Greengrass 원격 측정 문제 해결

다음 정보를 사용하면 AWS IoT Greengrass 원격 측정 구성 문제 해결에 도움이 됩니다.

오류: get-thing-runtime-configuration 명령을 실행한 후 응답에 “ConfigurationStatus”: “OutOfSync”가 포함됨

솔루션:

- AWS IoT 디바이스 섀도우 서비스는 런타임 구성 업데이트를 처리하고 Greengrass 코어 디바이스에 업데이트를 전달하는 데 시간이 걸립니다. 잠시 기다렸다가 나중에 원격 측정 설정이 동기화되었는지 확인할 수 있습니다.
- 코어 디바이스가 온라인 상태인지 확인합니다.
- AWS IoT Core에서 Amazon CloudWatch Logs를 활성화하여 섀도우를 모니터링할 수 있습니다.
- AWS IoT 지표를 사용하여 사물을 모니터링합니다.

로컬 상태 확인 API 직접 호출

AWS IoT Greengrass에는 AWS IoT Greengrass에서 시작된 로컬 작업자 프로세스의 현재 상태를 스냅샷으로 제공하는 로컬 HTTP API가 포함되어 있습니다. 이 스냅샷에는 사용자 정의 Lambda 함수와 시스템 Lambda 함수가 포함되어 있습니다. 시스템 Lambda 함수는 AWS IoT Greengrass 코어 소프트웨어의 구성 요소입니다. 코어 디바이스에서 로컬 작업자 프로세스로 실행되며 메시지 라우팅, 로컬 섀도 동기화, 자동 IP 주소 감지와 같은 작업을 관리합니다.

상태 확인 API는 다음 요청을 지원합니다.

- `GET` 요청을 보내 모든 작업자의 상태 정보를 받아보세요.
- `POST` 요청을 보내 지정된 작업자의 상태 정보를 받아보세요.

요청은 디바이스에서 로컬로 전송되며 인터넷 연결이 필요하지 않습니다.

모든 작업자의 상태 정보를 확인하세요.

`GET` 요청을 보내 실행 중인 모든 작업자의 상태 정보를 받아보세요.

- *##*를 IPC의 포트 번호로 바꾸십시오.

```
GET http://localhost:port/2016-11-01/health/workers
```

`port`

IPC의 포트 번호입니다.

값은 1024에서 65535 사이입니다. 기본값은 8000입니다.

이 포트 번호를 변경하려면 `config.json` 파일에서 `ggDaemonPort` 속성을 업데이트하면 됩니다. 자세한 내용은 AWS IoT Greengrass 코어 구성 파일 섹션을 참조하세요.

요청 예제

다음 예제 `curl` 요청은 모든 작업자의 상태 정보를 가져옵니다.

```
curl http://localhost:8000/2016-11-01/health/workers
```

JSON 응답

이 요청은 작업자 상태 정보 개체 배열을 반환합니다.

응답의 예

다음 예제 응답에는 AWS IoT Greengrass에서 시작한 모든 작업자 프로세스의 상태 정보 개체가 나열되어 있습니다.

```
[
    {
      "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
      "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
      "ProcessId": "1234",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:::function:GGSecretManager:1",
      "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
      "ProcessId": "9798",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
      "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
      "ProcessId": "11837",
      "WorkerState": "Waiting"
    },
    ...
]
```

지정된 작업자에 대한 상태 정보를 얻으십시오.

POST 요청을 보내 지정된 작업자의 상태 정보를 받아보세요. ##를 IPC의 포트 번호로 바꾸십시오. 기본값은 8000입니다.

```
POST http://localhost:port/2016-11-01/health/workers
```

요청 예제

다음 예제 curl 요청은 지정된 작업자의 상태 정보를 가져옵니다.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

다음은 body.json 요청 본문의 예제입니다.

```
{
    "FuncArns": [
        "arn:aws:lambda:::function:GGShadowService:1",
        "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
    ]
}
```

요청 본문에는 FuncArns 배열이 들어 있습니다.

FuncArns

대상 작업자를 나타내는 Lambda 함수에 대한 Amazon 리소스 이름(ARN) 목록입니다.

- 사용자 정의 Lambda 함수의 경우 현재 배포된 버전의 ARN을 지정합니다. 별칭 ARN을 사용하여 그룹에 Lambda 함수를 추가한 경우, GET 요청을 사용하여 모든 작업자를 가져온 다음 쿼리하려는 ARN을 선택할 수 있습니다.
- 시스템 Lambda 함수의 경우 해당 Lambda 함수의 ARN을 지정합니다. 자세한 내용은 the section called “시스템 Lambda 함수” 섹션을 참조하세요.

형식: 문자열 배열

최소 길이: 1

최대 길이: 코어 디바이스의 AWS IoT Greengrass에서 시작한 총 작업자의 수입니다.

JSON 응답

이 요청은 `Workers` 배열과 `InvalidArns` 배열을 반환합니다.

`Workers`

지정된 작업자의 상태 정보 개체 목록.

유형: 상태 정보 개체의 배열

`InvalidArns`

연결된 작업자가 없는 함수 ARN을 포함한 유효하지 않은 함수 ARN 목록.

형식: 문자열 배열

응답의 예

다음 예제 응답에는 지정된 작업자의 상태 정보 개체가 나열되어 있습니다.

```
{
    "Workers": [
        {
            "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
            "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
            "ProcessId": "1234",
            "WorkerState": "Waiting"
        },
        {
            "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function:3",
            "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
            "ProcessId": "11837",
            "WorkerState": "Waiting"
        }
    ],
    "InvalidArns" : [
        "some-malformed-arn",
        "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
    ]
}
```

이 요청은 다음 오류를 반환합니다.

400 잘못된 요청

요청 본문이 잘못되었습니다. 이 문제를 해결하려면 다음 형식을 사용하여 요청을 다시 전송하시기 바랍니다.

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 요청이 최대 작업자 수를 초과했습니다.

FuncArns 배열에 지정된 ARN 수가 작업자 수를 초과합니다.

작업자 상태 정보

정상 정보 객체는 다음 속성을 포함하고 있습니다.

FuncArn

작업자를 나타내는 시스템 Lambda 함수의 ARN입니다.

형식: string

WorkerId

작업자의 ID입니다. 이 속성은 디버깅에 유용합니다. runtime.log 파일 및 Lambda 함수 로그에는 작업자 ID가 포함되어 있으므로 이 속성은 여러 인스턴스를 구동하는 온디맨드 Lambda 함수를 디버깅하는 데 특히 유용할 수 있습니다.

형식: string

ProcessId

작업자 프로세스의 프로세스 ID(PID)

형식: int

WorkerState

작업자의 상태.

형식: string

가능한 작업 상태 표시는 다음과 같습니다.

`Working`

메시지 처리 중.

`Waiting`

메시지 대기 중. 대몬(daemon) 또는 독립형 프로세스로 실행되는 수명이 긴 Lambda 함수에 적용됩니다.

`Starting`

스핀업, 시작합니다.

`FailedInitialization`

초기화에 실패했습니다.

`Terminated`

Greengrass 대몬(daemon)에 의해 중지되었습니다.

`NotStarted`

시작에 실패하여 다시 시작을 시도합니다.

`Initialized`

성공적으로 초기화되었습니다.

시스템 Lambda 함수

다음 시스템 Lambda 함수에 대한 상태 정보를 요청할 수 있습니다.

`GGCloudSpooler`

AWS IoT Core이(가) 소스 또는 대상으로 있는 MQTT 메시지의 대기열을 관리합니다.

ARN: `arn:aws:lambda:::function:GGCloudSpooler:1`

`GGConnManager`

Greengrass 코어와 클라이언트 디바이스 간에 MQTT 메시지를 라우팅합니다.

ARN: `arn:aws:lambda:::function:GGConnManager`

`GGDeviceCertificateManager`

코어 IP 엔드포인트의 변경 사항에 대해 AWS IoT 섀도우를 수신하고 GGConnManager에서 상호 인증에 사용하는 서버 측 인증서를 생성합니다.

ARN: `arn:aws:lambda:::function:GGDeviceCertificateManager`

`GGIPDetector`

디바이스에서 Greengrass 코어 디바이스를 검색할 수 있도록 지원하는 자동 IP 주소 관리 IP 주소를 수동으로 제공하는 경우에는 이 서비스를 적용할 수 없습니다.

ARN: `arn:aws:lambda:::function:GGIPDetector:1`

`GGSecretManager`

사용자 정의 Lambda 및 커넥터를 통해 로컬 비밀의 안전한 저장과 액세스를 관리합니다.

ARN: `arn:aws:lambda:::function:GGSecretManager:1`

`GGShadowService`

클라이언트 디바이스의 로컬 섀도우를 관리합니다.

ARN: `arn:aws:lambda:::function:GGShadowService`

`GGShadowSyncManager`

디바이스의 `syncShadow` 속성이 `true`(으)로 설정된 경우 로컬 섀도우를 코어 디바이스 및 클라이언트 디바이스의 AWS 클라우드와(과) 동기화합니다.

ARN: `arn:aws:lambda:::function:GGShadowSyncManager`

`GGStreamManager`

데이터 스트림을 로컬에서 처리하고 AWS 클라우드(으)로 자동 내보내기를 수행합니다.

ARN: `arn:aws:lambda:::function:GGStreamManager:1`

`GGTES`

로컬 코드가 AWS 서비스에 액세스하는 데 사용하는 Greengrass 그룹 역할에 정의된 IAM 보안 인증을 검색하는 로컬 토큰 교환 서비스입니다.

ARN: `arn:aws:lambda:::function:GGTES`

AWS IoT Greengrass 리소스에 태그 지정

태그는 AWS IoT Greengrass 그룹을 구성하고 관리하는 데 도움이 될 수 있습니다. 태그를 사용하여 그룹, 대량 배포, 그리고 그룹에 추가되는 코어, 디바이스 및 기타 리소스에 메타데이터를 할당할 수 있습니다. 또한 IAM 정책에서 태그를 사용하여 Greengrass 리소스에 대한 조건부 액세스를 정의할 수 있습니다.

> **Note**
>
> 현재 AWS IoT 결제 그룹 또는 비용 할당 보고서에서는 Greengrass 리소스 태그가 지원되지 않습니다.

태그 기본 사항

태그를 사용하면 용도, 소유자 또는 환경 등을 기준으로 하는 AWS IoT Greengrass 리소스를 분류할 수 있습니다. 동일한 유형의 리소스가 많을 때 리소스에 연결한 태그를 기반으로 리소스를 빠르게 식별할 수 있습니다. 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 각 리소스 유형에 대한 태그 키 세트를 설계하는 것이 좋습니다. 일관된 태그 키 세트를 사용하면 리소스를 보다 쉽게 관리할 수 있습니다. 예를 들어, 그룹에 대한 태그 세트를 정의하여 코어 디바이스의 공장 위치를 추적할 수 있습니다. 자세한 내용은 AWS 태그 지정 전략을 참조하세요.

AWS IoT 콘솔에서 태그 지정 지원

AWS IoT 콘솔에서 Greengrass `Group` 리소스에 대한 태그를 생성하고, 보고, 관리할 수 있습니다. 태그를 생성하기 전에 태그 지정에 대한 제한 사항을 유의하십시오. 자세한 내용은 Amazon Web Services 일반 참조의 태그 이름 지정 및 사용 규칙을 참조하세요.

그룹을 생성할 때 태그를 할당하려면

그룹을 생성할 때 그룹에 태그를 할당할 수 있습니다. 태그 섹션에서 새 태그 추가를 선택하여 태깅 입력 필드를 표시합니다.

그룹 구성 페이지에서 태그를 보고 관리하려면

설정 보기를 선택해 그룹 구성 페이지에서 태그를 보고 관리할 수 있습니다. 그룹의 태그 섹션에서 그룹 태그 추가, 편집 또는 제거를 위해 태그 관리를 선택합니다.

AWS IoT Greengrass API에서 태그 지정 지원

AWS IoT Greengrass API를 사용하여 태그 지정을 지원하는 AWS IoT Greengrass 리소스에 대한 태그를 생성하고, 나열하고, 관리할 수 있습니다. 태그를 생성하기 전에 태그 지정에 대한 제한 사항을 유의하십시오. 자세한 내용은 Amazon Web Services 일반 참조의 태그 이름 지정 및 사용 규칙을 참조하세요.

- 리소스 생성 중에 태그를 추가하려면 리소스의 `tags` 속성에서 태그를 정의합니다.
- 리소스가 생성된 후 태그를 추가하거나 태그 값을 업데이트하려면 `TagResource` 작업을 사용합니다.
- 리소스에서 태그를 제거하려면 `UntagResource` 작업을 사용합니다.
- 리소스와 연결된 태그를 검색하려면 `ListTagsForResource` 작업을 사용하거나 리소스를 가져와서 해당 `tags` 속성을 검사합니다.

다음 표에는 AWS IoT Greengrass API에서 태그를 지정할 수 있는 리소스와 해당 `Create` 및 `Get` 작업이 나열됩니다.

Resource	생성	Get
`Group`	`CreateGroup`	`GetGroup`
`ConnectorDefinition`	`CreateConnectorDefinition`	`GetConnectorDefinition`
`CoreDefinition`	`CreateCoreDefinition`	`GetCoreDefinition`
`DeviceDefinition`	`CreateDeviceDefinition`	`GetDeviceDefinition`
`FunctionDefinition`	`CreateFunctionDefinition`	`GetFunctionDefinition`
`LoggerDefinition`	`CreateLoggerDefinition`	`GetLoggerDefinition`
`ResourceDefinition`	`CreateResourceDefinition`	`GetResourceDefinition`

Resource	생성	Get
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentStatus

다음 작업을 사용하여 태그 지정을 지원하는 리소스에 대한 태그를 나열하고 관리합니다.

- TagResource. 태그를 리소스에 추가합니다. 또한 태그의 키-값 페어의 값을 변경하는 데 사용됩니다.
- ListTagsForResource. 리소스에 대한 태그를 나열합니다.
- UntagResource. 리소스에서 태그를 제거합니다.

언제든지 리소스에서 태그를 추가하거나 제거할 수 있습니다. 태그 키의 값을 변경하려면 동일한 키와 새로운 값을 정의하는 리소스에 태그를 추가합니다. 새 값은 기존 값을 덮어씁니다. 값을 빈 문자열로 설정할 수 있지만,값을 Null로 설정할 수는 없습니다.

리소스를 삭제하면 리소스와 연결된 태그도 삭제됩니다.

> **Note**
>
> AWS IoT 사물에 할당할 수 있는 속성과 리소스 태그를 혼동하지 마십시오. Greengrass 코어는 AWS IoT 사물이지만 이 주제에서 설명하는 리소스 태그는 코어 사물에 연결되지 않고 CoreDefinition에 연결됩니다.

IAM 정책에 태그 사용

IAM 정책에서는 리소스 태그를 사용하여 사용자 액세스 및 권한을 제어할 수 있습니다. 예를 들어, 정책은 사용자가 특정 태그가 있는 리소스만 생성하도록 허용할 수 있습니다. 또한 정책은 사용자가 특정 태그가 있는 리소스를 생성하거나 수정하는 것을 제한할 수 있습니다. 생성 중에 리소스에 태그를 지정하면(생성 시 태그 지정이라고 함) 나중에 사용자 지정 태그 지정 스크립트를 실행할 필요가 없습니다. 태그를 사용하여 새 환경을 시작하면 해당 IAM 권한이 자동으로 적용됩니다.

다음 조건 텍스트 키 및 값은 정책의 Condition 요소에서 사용할 수 있습니다(Condition 블록이라고 함).

`greengrass:ResourceTag/tag-key: tag-value`

특정 태그가 있는 리소스에 대한 사용자 작업을 허용하거나 거부합니다.

`aws:RequestTag/tag-key: tag-value`

태그 지정 가능한 리소스에서 태그를 생성하거나 수정하는 API 요청을 수행할 때 특정 태그를 사용하도록(또는 사용하지 않도록) 요구합니다.

`aws:TagKeys: [tag-key, ...]`

태그 지정 가능한 리소스에서 태그를 생성하거나 수정하는 API 요청을 수행할 때 특정 태그 세트를 사용하도록(또는 사용하지 않도록) 요구합니다.

조건 컨텍스트 키 및 값은 태그 지정 가능한 리소스에 대한 작업을 수행하는 AWS IoT Greengrass 작업에만 사용할 수 있습니다. 이러한 작업은 리소스를 필수 파라미터로 간주합니다. 예를 들어, GetGroupVersion에서는 조건부 액세스를 설정할 수 있습니다. AssociateServiceRoleToAccount에서는 요청에서 태그 지정 가능한 리소스(예: 그룹, 코어 정의 또는 디바이스 정의)가 참조되지 않기 때문에 조건부 액세스를 설정할 수 없습니다.

자세한 내용은 IAM 사용 설명서에서 태그를 이용한 액세스 제어 및 IAM JSON 정책 참조를 참조하세요. JSON 정책 참조에는 IAM에 있는 JSON 정책의 요소, 변수 및 평가 로직에 대한 자세한 구문, 설명 및 예제가 포함되어 있습니다.

예제 IAM 정책

다음 예제 정책은 베타 사용자를 베타 리소스 작업으로만 제한하는 태그 기반 권한을 적용합니다.

- 첫 번째 문은 IAM 사용자가 env=beta 태그가 있는 리소스에 대해서만 작업하도록 허용합니다.
- 두 번째 문은 IAM 사용자가 env=beta 태그를 리소스에서 제거하는 것을 방지합니다. 이 문을 사용하여 사용자가 자신의 액세스를 제거하지 않도록 보호할 수 있습니다.

> **Note**
>
> 태그를 사용하여 리소스에 대한 액세스를 제어하는 경우 사용자가 동일한 리소스에서 태그를 추가하거나 제거하도록 허용하는 권한도 관리해야 합니다. 그렇지 않으면 경우에 따라 사용자가 태그를 수정하여 제한을 우회하고 리소스에 대한 액세스 권한을 얻을 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "greengrass:ResourceTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "greengrass:UntagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "beta"
                }
            }
        }
    ]
}
```

사용자가 생성 시 태그를 지정하도록 허용하려면 사용자에게 적절한 권한을 부여해야 합니다. 다음 예제 정책에는 `greengrass:TagResource` 및 `greengrass:CreateGroup` 작업에 대한 `"aws:RequestTag/env": "beta"` 조건이 포함되어 있습니다. 이 조건을 통해 사용자는 그룹에 env=beta로 태그를 지정하는 경우에만 그룹을 만들 수 있습니다. 이 방법으로 사용자가 새 그룹에 태그를 지정하도록 효과적으로 강제할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
```

```
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "greengrass:CreateGroup",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        }
    ]
}
```

다음 코드 조각은 태그를 목록 안에 넣어 한 태그 키에 대해 여러 태그 값을 지정할 수 있는 방법을 보여 줍니다.

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

다음 사항도 참조하십시오.

- Amazon Web Services 일반 참조에서 AWS 리소스 태그 지정

AWS IoT Greengrass에 대한 AWS CloudFormation 지원

AWS CloudFormation은 AWS 리소스를 생성, 관리 및 복제하는 데 도움이 될 수 있는 서비스입니다. AWS CloudFormation 템플릿을 사용하여 AWS IoT Greengrass 그룹과 배포할 클라이언트 디바이스, 구독 및 기타 구성 요소를 정의할 수 있습니다. 예시는 the section called “템플릿 예제” 단원을 참조하세요.

템플릿에서 생성하는 리소스와 인프라를 스택이라고 합니다. 모든 리소스를 하나의 템플릿에서 정의하거나 다른 스택의 리소스를 참조할 수 있습니다. AWS CloudFormation 템플릿과 특성에 대한 자세한 내용은 AWS CloudFormation사용자 가이드의 AWS CloudFormation란?을 참조하십시오.

리소스 생성

AWS CloudFormation 템플릿은 AWS 리소스의 속성과 관계를 설명하는 JSON 또는 YAML 문서입니다. 지원되는 AWS IoT Greengrass 리소스는 다음과 같습니다.

- 그룹
- 코어
- 클라이언트 디바이스(디바이스)
- Lambda 함수
- 커넥터
- 리소스(로컬, 기계 학습, 비밀)
- 구독
- 로거(로깅 구성)

AWS CloudFormation 템플릿에서 Greengrass 리소스의 구조와 구문은 AWS IoT Greengrass API를 기반으로 합니다. 예를 들어, 이 예제 템플릿은 최상위 수준 `DeviceDefinition`을 개별 클라이언트 디바이스가 포함된 `DeviceDefinitionVersion`과 연결합니다. 자세한 설명은 the section called “그룹 객체 모델 개요” 섹션을 참조하세요.

AWS CloudFormation 사용 설명서의 AWS IoT Greengrass 리소스 유형 참조에는 AWS CloudFormation와(과) 함께 관리할 수 있는 Greengrass 리소스가 설명되어 있습니다. AWS CloudFormation 템플릿을 사용하여 Greengrass 리소스를 생성할 때는 AWS CloudFormation에서만 리소스를 관리하는 것이 좋습니다. 예를 들어, 디바이스를 추가, 변경 또는 제거하려는 경우(AWS IoT Greengrass API 또는 AWS IoT 콘솔을 사용하는 대신) 템플릿을 업데이트해야 합니다. 이렇게 하면 롤

백 및 기타 AWS CloudFormation 변경 관리 기능을 사용할 수 있습니다. AWS CloudFormation을(를) 사용하여 리소스와 스택을 생성하고 관리하는 방법에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 스택 작업을 참조하십시오.

AWS CloudFormation 템플릿에서 AWS IoT Greengrass 리소스를 생성하고 배포하는 방법에 대한 연습을 보려면 AWS 공식 블로그의 사물 인터넷에 있는 AWS CloudFormation을 통해 AWS IoT Greengrass 설정 자동화를 참조하십시오.

리소스 배포

그룹 버전이 포함된 AWS CloudFormation 스택을 생성한 후에는 AWS CLI 또는 AWS IoT 콘솔을 사용하여 스택을 배포할 수 있습니다.

> **Note**
>
> 그룹을 배포하려면 AWS 계정과 연결된 Greengrass 서비스 역할이 있어야 합니다. AWS IoT Greengrass는 서비스 역할을 사용하여 AWS Lambda 및 기타 AWS 서비스에서 사용자의 리소스에 액세스할 수 있습니다. 현재 AWS 리전에서 Greengrass 그룹을 이미 배포한 경우 이 역할이 존재해야 합니다. 자세한 설명은 the section called “Greengrass 서비스 역할” 섹션을 참조하세요.

그룹을 배포하려면(AWS CLI)

- create-deployment 명령을 실행합니다.

  ```
  aws greengrass create-deployment --group-id GroupId --group-version-
  id GroupVersionId --deployment-type NewDeployment
  ```

  > **Note**
  >
  > 예제 템플릿의 `CommandToDeployGroup` 명령문은 스택을 생성할 때 그룹 및 그룹 버전 ID와 함께 명령을 출력하는 방법을 보여줍니다.

그룹을 배포하려면(콘솔)

1. AWS IoT 콘솔 탐색 창의 관리에서 Greengrass 디바이스를 확장한 다음 그룹(V1)을 선택합니다.

2. 그룹을 선택합니다.
3. 그룹 구성 페이지에서 배포를 선택합니다.

템플릿 예제

다음 예제 템플릿은 코어, 클라이언트 디바이스, 함수, 로거, 구독 및 두 개의 리소스가 포함된 Greengrass 그룹을 생성합니다. 이렇게 하기 위해 템플릿은 AWS IoT Greengrass API의 객체 모델을 따릅니다. 예를 들어, 그룹에 추가하려는 클라이언트 디바이스가 `DeviceDefinition` 리소스와 연결된 `DeviceDefinitionVersion` 리소스에 포함되어 있는 경우, 디바이스를 그룹에 추가하기 위해 그룹 버전은 `DeviceDefinitionVersion`의 ARN을 참조합니다.

이 템플릿에는 코어 및 디바이스의 인증서 ARN과 소스 Lambda 함수(AWS Lambda 리소스)의 버전 ARN을 지정하는 데 사용할 수 있는 파라미터가 포함되어 있습니다. 이 템플릿은 `Ref` 및 `GetAtt` 내장 함수를 사용하여 Greengrass 리소스를 생성하는 데 필요한 ID, ARN 및 기타 속성을 참조합니다.

또한 이 템플릿은 Greengrass 그룹에 추가되는 코어 및 클라이언트 디바이스를 나타내는 두 개의 AWS IoT 디바이스(사물)를 정의합니다.

Greengrass 리소스가 있는 스택을 생성한 후에는 AWS CLI 또는 AWS IoT 콘솔을 사용하여 그룹을 배포할 수 있습니다.

> **Note**
>
> 예제의 `CommandToDeployGroup` 문은 그룹을 배포하는 데 사용할 수 있는 전체 create-deployment CLI 명령을 출력하는 방법을 보여 줍니다.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "AWS IoT Greengrass example template that creates a group version
 with a core, device, function, logger, subscription, and resources.",
    "Parameters": {
        "CoreCertificateArn": {
            "Type": "String"
        },
        "DeviceCertificateArn": {
            "Type": "String"
        },
```

```
        "LambdaVersionArn": {
            "Type": "String"
        }
    },
    "Resources": {
        "TestCore1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
                "ThingName": "TestCore1"
            }
        },
        "TestCoreDefinition": {
            "Type": "AWS::Greengrass::CoreDefinition",
            "Properties": {
                "Name": "DemoTestCoreDefinition"
            }
        },
        "TestCoreDefinitionVersion": {
            "Type": "AWS::Greengrass::CoreDefinitionVersion",
            "Properties": {
                "CoreDefinitionId": {
                    "Ref": "TestCoreDefinition"
                },
                "Cores": [
                    {
                        "Id": "TestCore1",
                        "CertificateArn": {
                            "Ref": "CoreCertificateArn"
                        },
                        "SyncShadow": "false",
                        "ThingArn": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "thing/TestCore1"
                                ]
                            ]
```

```
                    }
                }
            ]
        }
    },
    "TestClientDevice1": {
        "Type": "AWS::IoT::Thing",
        "Properties": {
            "ThingName": "TestClientDevice1"
        }
    },
    "TestDeviceDefinition": {
        "Type": "AWS::Greengrass::DeviceDefinition",
        "Properties": {
            "Name": "DemoTestDeviceDefinition"
        }
    },
    "TestDeviceDefinitionVersion": {
        "Type": "AWS::Greengrass::DeviceDefinitionVersion",
        "Properties": {
            "DeviceDefinitionId": {
                "Fn::GetAtt": [
                    "TestDeviceDefinition",
                    "Id"
                ]
            },
            "Devices": [
                {
                    "Id": "TestClientDevice1",
                    "CertificateArn": {
                        "Ref": "DeviceCertificateArn"
                    },
                    "SyncShadow": "true",
                    "ThingArn": {
                        "Fn::Join": [
                            ":",
                            [
                                "arn:aws:iot",
                                {
                                    "Ref": "AWS::Region"
                                },
                                {
                                    "Ref": "AWS::AccountId"
                                },
```

```
                                "thing/TestClientDevice1"
                            ]
                        ]
                    }
                }
            ]
        }
    },
    "TestFunctionDefinition": {
        "Type": "AWS::Greengrass::FunctionDefinition",
        "Properties": {
            "Name": "DemoTestFunctionDefinition"
        }
    },
    "TestFunctionDefinitionVersion": {
        "Type": "AWS::Greengrass::FunctionDefinitionVersion",
        "Properties": {
            "FunctionDefinitionId": {
                "Fn::GetAtt": [
                    "TestFunctionDefinition",
                    "Id"
                ]
            },
            "DefaultConfig": {
                "Execution": {
                    "IsolationMode": "GreengrassContainer"
                }
            },
            "Functions": [
                {
                    "Id": "TestLambda1",
                    "FunctionArn": {
                        "Ref": "LambdaVersionArn"
                    },
                    "FunctionConfiguration": {
                        "Pinned": "true",
                        "Executable": "run.exe",
                        "ExecArgs": "argument1",
                        "MemorySize": "512",
                        "Timeout": "2000",
                        "EncodingType": "binary",
                        "Environment": {
                            "Variables": {
                                "variable1": "value1"
```

```
                            },
                            "ResourceAccessPolicies": [
                                {
                                    "ResourceId": "ResourceId1",
                                    "Permission": "ro"
                                },
                                {
                                    "ResourceId": "ResourceId2",
                                    "Permission": "rw"
                                }
                            ],
                            "AccessSysfs": "false",
                            "Execution": {
                                "IsolationMode": "GreengrassContainer",
                                "RunAs": {
                                    "Uid": "1",
                                    "Gid": "10"
                                }
                            }
                        }
                    }
                }
            ]
        }
    },
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {
                "Ref": "TestLoggerDefinition"
            },
            "Loggers": [
                {
                    "Id": "TestLogger1",
                    "Type": "AWSCloudWatch",
                    "Component": "GreengrassSystem",
                    "Level": "INFO"
                }
```

```
                ]
            }
        },
        "TestResourceDefinition": {
            "Type": "AWS::Greengrass::ResourceDefinition",
            "Properties": {
                "Name": "DemoTestResourceDefinition"
            }
        },
        "TestResourceDefinitionVersion": {
            "Type": "AWS::Greengrass::ResourceDefinitionVersion",
            "Properties": {
                "ResourceDefinitionId": {
                    "Ref": "TestResourceDefinition"
                },
                "Resources": [
                    {
                        "Id": "ResourceId1",
                        "Name": "LocalDeviceResource",
                        "ResourceDataContainer": {
                            "LocalDeviceResourceData": {
                                "SourcePath": "/dev/TestSourcePath1",
                                "GroupOwnerSetting": {
                                    "AutoAddGroupOwner": "false",
                                    "GroupOwner": "TestOwner"
                                }
                            }
                        }
                    },
                    {
                        "Id": "ResourceId2",
                        "Name": "LocalVolumeResourceData",
                        "ResourceDataContainer": {
                            "LocalVolumeResourceData": {
                                "SourcePath": "/dev/TestSourcePath2",
                                "DestinationPath": "/volumes/TestDestinationPath2",
                                "GroupOwnerSetting": {
                                    "AutoAddGroupOwner": "false",
                                    "GroupOwner": "TestOwner"
                                }
                            }
                        }
                    }
                ]
```

```
            }
        },
        "TestSubscriptionDefinition": {
            "Type": "AWS::Greengrass::SubscriptionDefinition",
            "Properties": {
                "Name": "DemoTestSubscriptionDefinition"
            }
        },
        "TestSubscriptionDefinitionVersion": {
            "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
            "Properties": {
                "SubscriptionDefinitionId": {
                    "Ref": "TestSubscriptionDefinition"
                },
                "Subscriptions": [
                    {
                        "Id": "TestSubscription1",
                        "Source": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "thing/TestClientDevice1"
                                ]
                            ]
                        },
                        "Subject": "TestSubjectUpdated",
                        "Target": {
                            "Ref": "LambdaVersionArn"
                        }
                    }
                ]
            }
        },
        "TestGroup": {
            "Type": "AWS::Greengrass::Group",
            "Properties": {
                "Name": "DemoTestGroupNewName",
```

```
                "RoleArn": {
                    "Fn::Join": [
                        ":",
                        [
                            "arn:aws:iam:",
                            {
                                "Ref": "AWS::AccountId"
                            },
                            "role/TestUser"
                        ]
                    ]
                },
                "InitialVersion": {
                    "CoreDefinitionVersionArn": {
                        "Ref": "TestCoreDefinitionVersion"
                    },
                    "DeviceDefinitionVersionArn": {
                        "Ref": "TestDeviceDefinitionVersion"
                    },
                    "FunctionDefinitionVersionArn": {
                        "Ref": "TestFunctionDefinitionVersion"
                    },
                    "SubscriptionDefinitionVersionArn": {
                        "Ref": "TestSubscriptionDefinitionVersion"
                    },
                    "LoggerDefinitionVersionArn": {
                        "Ref": "TestLoggerDefinitionVersion"
                    },
                    "ResourceDefinitionVersionArn": {
                        "Ref": "TestResourceDefinitionVersion"
                    }
                },
                "Tags": {
                    "KeyName0": "value",
                    "KeyName1": "value",
                    "KeyName2": "value"
                }
            }
        }
    },
    "Outputs": {
        "CommandToDeployGroup": {
            "Value": {
                "Fn::Join": [
```

```
                    " ",
                    [
                        "groupVersion=$(cut -d'/' -f6 <<<",
                        {
                            "Fn::GetAtt": [
                                "TestGroup",
                                "LatestVersionArn"
                            ]
                        },
                        ");",
                        "aws --region",
                        {
                            "Ref": "AWS::Region"
                        },
                        "greengrass create-deployment --group-id",
                        {
                            "Ref": "TestGroup"
                        },
                        "--deployment-type NewDeployment --group-version-id",
                        "$groupVersion"
                    ]
                ]
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  AWS IoT Greengrass example template that creates a group version with a core,
  device, function, logger, subscription, and resources.
Parameters:
  CoreCertificateArn:
    Type: String
  DeviceCertificateArn:
    Type: String
  LambdaVersionArn:
    Type: String
Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
```

```
  Properties:
    ThingName: TestCore1
TestCoreDefinition:
  Type: 'AWS::Greengrass::CoreDefinition'
  Properties:
    Name: DemoTestCoreDefinition
TestCoreDefinitionVersion:
  Type: 'AWS::Greengrass::CoreDefinitionVersion'
  Properties:
    CoreDefinitionId: !Ref TestCoreDefinition
    Cores:
      - Id: TestCore1
        CertificateArn: !Ref CoreCertificateArn
        SyncShadow: 'false'
        ThingArn: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestCore1
TestClientDevice1:
  Type: 'AWS::IoT::Thing'
  Properties:
    ThingName: TestClientDevice1
TestDeviceDefinition:
  Type: 'AWS::Greengrass::DeviceDefinition'
  Properties:
    Name: DemoTestDeviceDefinition
TestDeviceDefinitionVersion:
  Type: 'AWS::Greengrass::DeviceDefinitionVersion'
  Properties:
    DeviceDefinitionId: !GetAtt
      - TestDeviceDefinition
      - Id
    Devices:
      - Id: TestClientDevice1
        CertificateArn: !Ref DeviceCertificateArn
        SyncShadow: 'true'
        ThingArn: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
```

```
  TestFunctionDefinition:
    Type: 'AWS::Greengrass::FunctionDefinition'
    Properties:
      Name: DemoTestFunctionDefinition
  TestFunctionDefinitionVersion:
    Type: 'AWS::Greengrass::FunctionDefinitionVersion'
    Properties:
      FunctionDefinitionId: !GetAtt
        - TestFunctionDefinition
        - Id
      DefaultConfig:
        Execution:
          IsolationMode: GreengrassContainer
      Functions:
        - Id: TestLambda1
          FunctionArn: !Ref LambdaVersionArn
          FunctionConfiguration:
            Pinned: 'true'
            Executable: run.exe
            ExecArgs: argument1
            MemorySize: '512'
            Timeout: '2000'
            EncodingType: binary
            Environment:
              Variables:
                variable1: value1
              ResourceAccessPolicies:
                - ResourceId: ResourceId1
                  Permission: ro
                - ResourceId: ResourceId2
                  Permission: rw
              AccessSysfs: 'false'
              Execution:
                IsolationMode: GreengrassContainer
                RunAs:
                  Uid: '1'
                  Gid: '10'
  TestLoggerDefinition:
    Type: 'AWS::Greengrass::LoggerDefinition'
    Properties:
      Name: DemoTestLoggerDefinition
  TestLoggerDefinitionVersion:
    Type: 'AWS::Greengrass::LoggerDefinitionVersion'
    Properties:
```

```
      LoggerDefinitionId: !Ref TestLoggerDefinition
      Loggers:
        - Id: TestLogger1
          Type: AWSCloudWatch
          Component: GreengrassSystem
          Level: INFO
  TestResourceDefinition:
    Type: 'AWS::Greengrass::ResourceDefinition'
    Properties:
      Name: DemoTestResourceDefinition
  TestResourceDefinitionVersion:
    Type: 'AWS::Greengrass::ResourceDefinitionVersion'
    Properties:
      ResourceDefinitionId: !Ref TestResourceDefinition
      Resources:
        - Id: ResourceId1
          Name: LocalDeviceResource
          ResourceDataContainer:
            LocalDeviceResourceData:
              SourcePath: /dev/TestSourcePath1
              GroupOwnerSetting:
                AutoAddGroupOwner: 'false'
                GroupOwner: TestOwner
        - Id: ResourceId2
          Name: LocalVolumeResourceData
          ResourceDataContainer:
            LocalVolumeResourceData:
              SourcePath: /dev/TestSourcePath2
              DestinationPath: /volumes/TestDestinationPath2
              GroupOwnerSetting:
                AutoAddGroupOwner: 'false'
                GroupOwner: TestOwner
  TestSubscriptionDefinition:
    Type: 'AWS::Greengrass::SubscriptionDefinition'
    Properties:
      Name: DemoTestSubscriptionDefinition
  TestSubscriptionDefinitionVersion:
    Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
    Properties:
      SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
      Subscriptions:
        - Id: TestSubscription1
          Source: !Join
            - ':'
```

```
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestClientDevice1
          Subject: TestSubjectUpdated
          Target: !Ref LambdaVersionArn
  TestGroup:
    Type: 'AWS::Greengrass::Group'
    Properties:
      Name: DemoTestGroupNewName
      RoleArn: !Join
        - ':'
        - - 'arn:aws:iam:'
          - !Ref 'AWS::AccountId'
          - role/TestUser
      InitialVersion:
        CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
        DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
        FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
        SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
        LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
        ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
      Tags:
        KeyName0: value
        KeyName1: value
        KeyName2: value
Outputs:
  CommandToDeployGroup:
    Value: !Join
      - ' '
      - - groupVersion=$(cut -d'/' -f6 <<<
        - !GetAtt
          - TestGroup
          - LatestVersionArn
        - );
        - aws --region
        - !Ref 'AWS::Region'
        - greengrass create-deployment --group-id
        - !Ref TestGroup
        - '--deployment-type NewDeployment --group-version-id'
        - $groupVersion
```

지원되는 AWS 리전

현재는 다음 AWS 리전와(과) 같은 경우에만 AWS IoT Greengrass 리소스를 생성하고 관리할 수 있습니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(오레곤)
- 아시아 태평양(뭄바이)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 중국(베이징)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- AWS GovCloud (미국 서부)

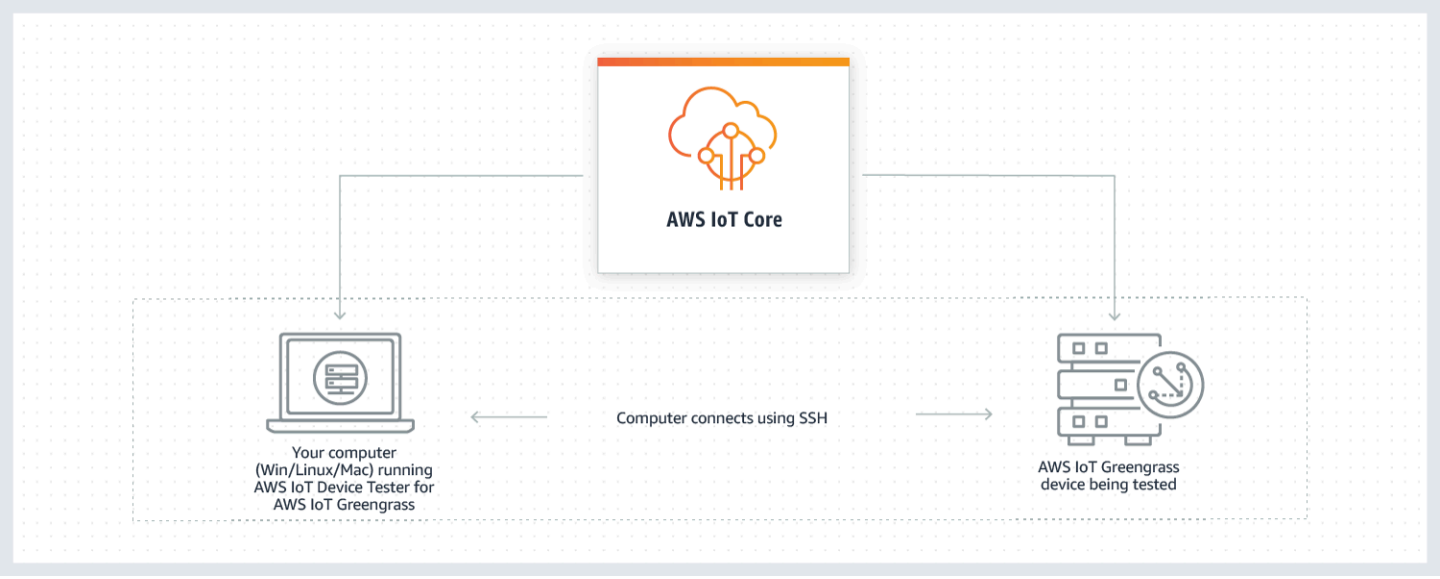
AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터 사용

AWS IoT 디바이스 테스터 (IDT) 는 IoT 디바이스를 검증할 수 있는 다운로드 가능한 테스트 프레임워크입니다. 유지 관리 모드로 전환되었기 때문에 IDT AWS IoT Greengrass Version 1 for는 더 AWS IoT Greengrass V1 이상 서명된 자격 보고서를 생성하지 않습니다. 더 이상 장치 검증 프로그램을 통해 장치 카탈로그에 새 AWS IoT Greengrass V1 장치를 등록할 자격을 부여할 수 없습니다.AWS Partner AWS 하지만 Greengrass V1 디바이스를 AWS IoT Greengrass V1 테스트하기 위해 IDT를 계속 사용할 수 있습니다. AWS Partner 장치 카탈로그에서 Greengrass 장치를 검증하고 등재하려면 AWS IoT Greengrass V2용 IDT를 사용하는 것이 좋습니다.

IDT는 테스트할 장치에 연결된 호스트 컴퓨터 (Windows, macOS 또는 Linux) 에서 AWS IoT Greengrass 실행됩니다. 이 IDT는 테스트를 실행하고 결과를 집계합니다. 또한 테스트 프로세스를 관리하기 위한 명령줄 인터페이스를 제공합니다.

AWS IoT Greengrass 자격 제품군

AWS IoT Greengrass IDT를 사용하여 AWS IoT Greengrass Core 소프트웨어가 하드웨어에서 실행되고 하드웨어와 통신할 수 있는지 확인하십시오. AWS 클라우드또한 를 사용하여 end-to-end AWS IoT Core테스트를 수행합니다. 예를 들어 장치가 MQTT 메시지를 송수신하고 올바르게 처리할 수 있는지 확인합니다.



AWS IoT 기기 테스터 용은 테스트 도구 모음 및 테스트 그룹의 개념을 사용하여 테스트를 AWS IoT Greengrass 구성합니다.

- 테스트 제품군은 장치가 AWS IoT Greengrass의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 Greengrass 그룹 배포 및 MQTT 메시징 같은 특정 기능과 관련된 개별 테스트 집합입니다.

자세한 정보는 IDT를 이용해 AWS IoT Greengrass 검증 제품군 실행을 참조하세요.

사용자 지정 테스트 도구 모음

IDT v4.0.0부터 IDT for는 표준화된 구성 설정 및 결과 형식을 장치 및 장치 소프트웨어에 대한 사용자 지정 테스트 도구 모음을 개발할 수 있는 테스트 도구 모음 환경과 AWS IoT Greengrass 결합합니다. 자체 내부 검증을 위한 사용자 지정 테스트를 추가하거나 장치 검증을 위해 고객에게 제공할 수 있습니다.

테스트 작성자가 사용자 지정 테스트 도구 모음을 구성하는 방법에 따라 사용자 지정 테스트 도구 모음을 실행하는 데 필요한 설정 구성이 결정됩니다. 자세한 내용은 IDT를 사용하여 자체 테스트 도구 모음을 개발하고 실행하십시오.을(를) 참조하세요.

AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터의 지원되는 버전

AWS IoT Greengrass Version 1이 유지 관리 모드로 전환되었기 때문에 AWS IoT Greengrass V1용 IDT는 더 이상 서명된 자격 보고서를 생성하지 않습니다. AWS IoT Greengrass V2용 IDT 사용을 권장합니다.

AWS IoT Greengrass V2용 IDT에 대한 자세한 내용은 AWS IoT Greengrass V2 개발자 안내서에서 AWS IoT Greengrass V2용 AWS IoT 디바이스 테스터 사용을 참조하십시오.

> **Note**
>
> AWS IoT Greengrass용 IDT가 사용 중인 AWS IoT Greengrass 버전과 호환되지 않을 경우 테스트 실행을 시작할 때 알림이 제공됩니다.

소프트웨어를 다운로드하면 AWS IoT Device Tester 라이선스 계약에 동의하는 것입니다.

AWS IoT Greengrass용 IDT의 지원되지 않는 버전

이 주제에서는 AWS IoT Greengrass용 IDT의 지원되지 않는 버전을 나열합니다. 지원되지 않는 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다. 자세한 설명은 the section called “AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터에 대한 지원 정책” 섹션을 참조하세요.

AWS IoT Greengrass 버전 v1.11.6, v1.10.5용 IDT v4.4.1

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 v1.11.6 및 v1.10.5를 실행하는 디바이스를 확인하고 검증할 수 있습니다.
- 사소한 버그 수정 사항을 포함합니다.

테스트 제품군 버전:

`GGQ_1.3.1`

- 2021년 12월 20일 출시

AWS IoT Greengrass 버전 v1.11.4, v1.10.4용 IDT v4.1.0

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 v1.11.4 및 v1.10.4를 실행하는 디바이스를 확인하고 검증할 수 있습니다.
- 테스트 실행 중에 표시되는 로그가 중복 태그를 사용하던 문제를 수정합니다.

테스트 제품군 버전:

`GGQ_1.3.0`

- 2021년 6월 23일에 출시되었습니다.
- Lambda, IAM 및 AWS STS에 대한 API 호출 재시도를 추가하고 제한 또는 서버 문제에 대한 처리를 개선합니다.
- ML 및 Docker 테스트 케이스에 Python 3.8에 대한 지원을 추가합니다.

AWS IoT Greengrass 버전 v1.11.1, v1.11.0, v1.10.3용 IDT v4.0.2

릴리스 정보:

- IDT가 하드웨어 보안 통합(HSI) 오류를 마스킹하는 문제를 수정했습니다.
- AWS IoT Greengrass용 AWS IoT 디바이스 테스터를 사용하여 사용자 지정 테스트 제품군을 개발하고 실행할 수 있습니다. 자세한 설명은 IDT를 사용하여 자체 테스트 도구 모음을 개발하고 실행하십시오. 섹션을 참조하세요.

- macOS 및 Windows용 코드 서명 IDT 애플리케이션을 제공합니다. macOS에서 보안 경고 메시지가 표시되면 IDT에 보안 예외를 부여해야 할 수 있습니다. 자세한 설명은 macOS에서의 보안 예외 섹션을 참조하세요.

> **Note**
>
> AWS IoT Greengrass는 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.1에 대한 도커파일 또는 Docker 이미지를 제공하지 않습니다. 디바이스의 Docker 검증을 테스트하려면 이전 버전의 AWS IoT Greengrass 코어 소프트웨어를 사용하십시오.

AWS IoT Greengrass 버전 v1.11.0, v1.10.1, v1.10.0용 IDT v3.2.0

릴리스 정보:

- 기본적으로 IDT는 검증을 위한 필수 테스트만 실행합니다. 추가 기능을 검증하려면 `device.json` 파일을 수정하면 됩니다.
- `device.json`에 SSH 연결용으로 구성할 수 있는 포트 번호를 추가했습니다.
- Docker는 컨테이너화 없이 스트림 관리자 및 기계 학습(ML)만 지원합니다. Docker 디바이스에는 컨테이너, 도커 및 HSI(하드웨어 보안 통합)를 사용할 수 없습니다.
- `device-ml.json`과 `device-hsm.json`을 `device.json`으로 병합했습니다.

AWS IoT Greengrass 버전 v1.10.x, v1.9.x, v1.8.x용 IDT v3.1.3

릴리스 정보:

- AWS IoT Greengrass v1.10.x 및 v1.9.x에 대한 ML 기능 검증에 대한 지원이 추가되었습니다. 이제 IDT를 사용하여 디바이스가 클라우드에 저장되고 교육된 모델을 사용하여 로컬로 ML 추론을 수행할 수 있는지 확인할 수 있습니다.
- `run-suite` 명령에 대한 `--stop-on-first-failure`가 추가되었습니다. 이 옵션을 사용하여 첫 번째 실패 시 실행을 중지하도록 IDT를 구성할 수 있습니다. 테스트 그룹 수준에서 디버깅 스테이지가 진행되는 동안 이 옵션을 사용하는 것이 좋습니다.
- 테스트 중인 디바이스가 올바른 시스템 시간을 사용하는지 확인하기 위해 MQTT 테스트에 대한 클록 드리프트 검사가 추가되었습니다. 사용 시간은 허용 가능한 시간 범위 내에 있어야 합니다.
- `run-suite` 명령에 대한 `--update-idt`가 추가되었습니다. 이 옵션을 사용하여 IDT를 업데이트하라는 프롬프트에 대한 응답을 설정할 수 있습니다.

- `run-suite` 명령에 대한 `--update-managed-policy`가 추가되었습니다. 이 옵션을 사용하여 관리형 정책을 업데이트하라는 프롬프트에 대한 응답을 설정할 수 있습니다.
- IDT 테스트 제품군 버전의 자동 업데이트에 대한 버그 수정이 추가되었습니다. IDT는 사용 중인 AWS IoT Greengrass 버전에 사용할 수 있는 최신 테스트 제품군을 실행할 수 있습니다.

AWS IoT Greengrass용 IDT v3.0.1

릴리스 정보:

- AWS IoT Greengrass v1.10.1에 대한 지원을 추가했습니다.
- IDT 테스트 제품군 버전의 자동 업데이트. IDT는 사용 중인 AWS IoT Greengrass 버전에 사용할 수 있는 최신 테스트 제품군을 다운로드할 수 있습니다. 이 기능의 내용은 다음과 같습니다.
  - 테스트 제품군은 *`major`*`.`*`minor`*`.`*`patch`* 형식을 사용하여 버전이 지정됩니다. 초기 테스트 제품군 버전은 `GGQ_1.0.0`입니다.
  - 새 테스트 제품군을 명령줄 인터페이스에서 대화식으로 다운로드하거나 IDT를 시작할 때 `upgrade-test-suite` 플래그를 설정할 수 있습니다.

  자세한 설명은 the section called “테스트 제품군 버전” 섹션을 참조하세요.
- `list-supported-products` 추가. 이 명령을 사용하여 설치된 IDT 버전에서 지원하는 테스트 제품군 버전과 AWS IoT Greengrass를 나열할 수 있습니다.
- `list-test-cases` 추가. 이 명령을 사용하여 테스트 그룹에서 사용할 수 있는 테스트 케이스를 나열할 수 있습니다.
- `run-suite` 명령에 대한 `test-id`가 추가되었습니다. 이 옵션을 사용하여 테스트 그룹에서 개별 테스트 케이스를 실행할 수 있습니다.

AWS IoT Greengrass v1.10, v1.9.x 및 v1.8.x용 IDT v2.3.0

물리적 디바이스에서 테스트하는 경우 AWS IoT Greengrass v1.10, v1.9.x 및 v1.8.x가 지원됩니다.

Docker 컨테이너에서 테스트하는 경우 AWS IoT Greengrass v1.10 및 v1.9.x가 지원됩니다.

릴리스 정보:

- the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 지원이 추가되었습니다. 이제 IDT를 사용하여 디바이스가 Docker 컨테이너에서 AWS IoT Greengrass를 실행할 수 있는지 확인하고 검증할 수 있습니다.

- AWS IoT 디바이스 테스터를 실행하는 데 필요한 권한을 정의하는 AWS 관리형 정책(`AWSIoTDeviceTesterForGreengrassFullAccess`)이 추가되었습니다. 새 릴리스에 추가 권한이 필요한 경우 AWS는 해당 권한을 이 관리형 정책에 추가하므로 사용자가 IAM 권한을 업데이트할 필요가 없습니다.
- 테스트 사례를 실행하기 전에 사용자 환경(예: 디바이스 연결 및 인터넷 연결)이 올바르게 설정되어 있는지 확인하는 검사가 도입되었습니다.
- IDT의 Greengrass 종속성 확인 프로그램이 디바이스에서 보다 유연하게 libc를 확인할 수 있도록 개선되었습니다.

AWS IoT Greengrass v1.10, v1.9.x 및 v1.8.x용 IDT v2.2.0

릴리스 정보:

- AWS IoT Greengrass v1.10에 대한 지원이 추가되었습니다.
- Greengrass Docker 애플리케이션 배포 커넥터에 대한 지원이 추가되었습니다.
- AWS IoT Greengrass 스트림 관리자에 대한 지원이 추가되었습니다.
- 중국(베이징) 리전에 대한 AWS IoT Greengrass 지원이 추가되었습니다.

AWS IoT Greengrass v1.9.x, v1.8.x 및 v1.7.x용 IDT v2.1.0

릴리스 정보:

- AWS IoT Greengrass v1.9.4에 대한 지원을 추가했습니다.
- Linux-ARMv6l 디바이스에 대한 지원을 추가했습니다.

AWS IoT Greengrass v1.9.3, v1.9.2, v.1.9.1, v1.9.0, v1.8.4, v1.8.3 및 v1.8.2용 IDT v2.0.0

릴리스 정보:

- 테스트 대상 디바이스에서 Python 종속성을 제거했습니다.
- 테스트 제품군 실행 시간을 50% 이상 단축하여 검증 프로세스가 빨라졌습니다.
- 실행 파일 크기를 50% 축소하여 다운로드 및 설치가 빨라졌습니다.
- 모든 테스트 사례에서 제한 시간 승수 지원을 개선했습니다.
- 오류를 보다 신속하게 해결할 수 있도록 진단 후 메시지를 개선했습니다.
- IDT를 실행하는 데 필요한 권한 정책 템플릿을 업데이트했습니다.

- AWS IoT Greengrass v1.9.3에 대한 지원을 추가했습니다.

AWS IoT Greengrass v1.9.2, v1.9.1, v1.9.0, v1.8.3 및 v1.8.2용 IDT v1.3.3

릴리스 정보:

- Greengrass v1.9.2 및 v1.8.3에 대한 지원을 추가했습니다.
- OpenWrtGreengrass에 대한 지원이 추가되었습니다.
- SSH 사용자 이름 및 암호 디바이스 로그인을 추가했습니다.
- OpenWrt-ARMv7L 플랫폼에 대한 네이티브 테스트 버그 수정이 추가되었습니다.

AWS IoT Greengrass v1.8.1용 IDT v1.2

릴리스 정보:

- 제한 시간 문제를 처리하고 해결하기 위해 구성 가능한 제한 시간 승수를 추가했습니다(예: 낮은 대역폭 연결).

AWS IoT Greengrass v1.8.0용 IDT v1.1

릴리스 정보:

- AWS IoT Greengrass 하드웨어 보안 통합(HSI)에 대한 지원을 추가했습니다.
- AWS IoT Greengrass 컨테이너 및 컨테이너 없음에 대한 지원을 추가했습니다.
- 자동화된 AWS IoT Greengrass 서비스 역할 생성을 추가했습니다.
- 테스트 리소스 정리를 개선했습니다
- 텍스트 실행 요약 보고서를 추가했습니다.

AWS IoT Greengrass v1.7.1용 IDT v1.1

릴리스 정보:

- AWS IoT Greengrass 하드웨어 보안 통합(HSI)에 대한 지원을 추가했습니다.
- AWS IoT Greengrass 컨테이너 및 컨테이너 없음에 대한 지원을 추가했습니다.
- 자동화된 AWS IoT Greengrass 서비스 역할 생성을 추가했습니다.

- 테스트 리소스 정리를 개선했습니다
- 텍스트 실행 요약 보고서를 추가했습니다.

AWS IoT Greengrass v1.6.1용 IDT v1.0

릴리스 정보:

- 향후 AWS IoT Greengrass 버전 호환성을 위해 OTA 테스트 버그 수정을 추가했습니다.

> **Note**
>
> AWS IoT Greengrass v1.6.1용 IDT v1.0을 사용하는 경우 Greengrass 서비스 역할을 생성해야 합니다. 이후 버전에서는 IDT가 서비스 역할을 자동으로 생성합니다.

IDT를 이용해 AWS IoT Greengrass 검증 제품군 실행

AWS IoT Greengrass용 AWS IoT 디바이스 테스터(IDT)를 사용하면 AWS IoT Greengrass 코어 소프트웨어가 하드웨어에서 실행되고 AWS 클라우드와 통신할 수 있는지 확인할 수 있습니다. 또한 이 IDT는 AWS IoT Core와의 엔드 투 엔드 테스트를 수행합니다. 예를 들어 디바이스가 MQTT 메시지를 송수신하고 올바르게 처리할 수 있는지 확인합니다.

AWS IoT Greengrass Version 1이 유지 관리 모드로 전환되었기 때문에 AWS IoT Greengrass V1용 IDT는 더 이상 서명된 자격 보고서를 생성하지 않습니다. AWS Partner 디바이스 카탈로그에 하드웨어를 추가하려면 AWS IoT Greengrass V2 검증 제품군을 실행하여 AWS IoT에 제출할 수 있는 테스트 보고서를 생성하십시오. 자세한 내용은 AWS 디바이스 검증 프로그램 및 AWS IoT Greengrass V2 지원 IDT 버전을 참조하십시오.

AWS IoT Greengrass용 IDT는 디바이스 테스트 외에 검증 프로세스를 쉽게 진행하기 위해 사용자의 AWS 계정에 리소스(예: AWS IoT 사물, AWS IoT Greengrass 그룹, Lambda 함수 등)를 생성합니다.

이러한 리소스를 생성하기 위해 AWS IoT Greengrass용 IDT는 `config.json` 파일에서 구성된 AWS 자격 증명을 사용하여 사용자를 대신해서 API를 호출합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

AWS IoT Greengrass용 IDT를 사용하여 AWS IoT Greengrass 검증 제품군을 실행하는 경우 IDT는 다음 단계를 수행합니다.

1. 디바이스 및 자격 증명 구성을 로드하고 검증합니다.

2. 필수 로컬 및 클라우드 리소스를 사용하여 선택한 테스트를 수행합니다.
3. 로컬 및 클라우드 리소스를 정리합니다.
4. 디바이스에서 검증에 필요한 테스트를 통과했는지를 나타내는 테스트 보고서를 생성합니다.

테스트 제품군 버전

AWS IoT Greengrass용 IDT는 테스트 제품군 및 테스트 그룹으로 테스트를 구성합니다.

- 테스트 제품군은 디바이스가 AWS IoT Greengrass의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 Greengrass 그룹 배포 및 MQTT 메시징 같은 특정 기능과 관련된 개별 테스트 집합입니다.

IDT v3.0.0부터는 *major*.*minor*.*patch* 형식(예: `GGQ_1.0.0`)을 사용하여 테스트 제품군의 버전이 관리됩니다. IDT를 다운로드하면 패키지에 최신 테스트 제품군 버전이 포함됩니다.

> **Important**
>
> IDT는 디바이스 검증을 위해 세 가지 최신 테스트 제품군 버전을 지원합니다. 자세한 내용은 the section called “AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터에 대한 지원 정책” 섹션을 참조하세요.
>
> `list-supported-products`를 실행하여 현재 사용 중인 IDT 버전에서 지원하는 AWS IoT Greengrass 버전과 테스트 제품군을 나열할 수 있습니다. 지원되지 않는 테스트 제품군 버전의 테스트는 디바이스 검증에 유효하지 않습니다. IDT는 지원되지 않는 버전에 대한 검증 보고서를 인쇄하지 않습니다.

IDT 구성 설정에 대한 업데이트

새로운 테스트에서는 새로운 IDT 구성 설정을 도입할 수 있습니다.

- 설정이 선택 사항인 경우 IDT는 테스트를 계속 실행합니다.
- 설정이 필요한 경우 IDT는 사용자에게 이를 알리고 실행을 중지합니다. 설정을 구성한 후 테스트 실행을 다시 시작하십시오.

  구성 설정은 *<device-tester-extract-location>*`/configs` 폴더에 있습니다. 자세한 내용은 the section called “IDT 설정 구성” 섹션을 참조하세요.

업데이트된 테스트 제품군 버전이 구성 설정을 추가하는 경우 IDT는 *<device-tester-extract-location>*/configs에 원래 구성 파일의 복사본을 만듭니다.

테스트 그룹 설명

IDT v2.0.0 and later

코어 검증을 위한 필수 테스트 그룹

이러한 테스트 그룹은 AWS IoT Greengrass 디바이스의 AWS Partner 디바이스 카탈로그를 검증하기 위해 필요합니다.

AWS IoT Greengrass 코어 종속성

디바이스가 AWS IoT Greengrass 코어 소프트웨어에 대한 모든 소프트웨어 및 하드웨어 요구 사항을 충족하는지 여부를 확인합니다.

이 테스트 그룹의 `Software Packages Dependencies` 테스트 케이스는 Docker 컨테이너에서 테스트하는 경우 적용할 수 없습니다.

배포

디바이스에 Lambda 함수를 배포할 수 있는지 검증합니다.

MQTT

Greengrass 코어와 로컬 IoT 디바이스인 클라이언트 디바이스 간 로컬 통신을 점검하여 AWS IoT Greengrass 메시지 라우터 기능을 확인합니다.

무선(OTA)

디바이스가 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트를 성공적으로 수행할 수 있는지 검증합니다.

이 테스트 그룹은 Docker 컨테이너에서 테스트하는 경우 적용할 수 없습니다.

버전

제공된 AWS IoT Greengrass의 버전이 사용하는 AWS IoT 디바이스 테스터 버전과 호환되는지 확인합니다.

선택적 테스트 그룹

이러한 테스트 그룹은 선택 사항입니다. 선택적 테스트 자격을 선택하면 디바이스가 AWS Partner 디바이스 카탈로그에 추가 기능과 함께 나열됩니다.

컨테이너 종속성

디바이스가 Greengrass 코어에서 컨테이너 모드로 Lambda 함수를 실행하기 위한 모든 소프트웨어 및 하드웨어 요구 사항을 충족하는지 여부를 검증합니다.

이 테스트 그룹은 Docker 컨테이너에서 테스트하는 경우 적용할 수 없습니다.

배포 컨테이너

디바이스에 Lambda 함수를 배포하고 Greengrass 코어에서 컨테이너 모드로 실행할 수 있는지 검증합니다.

이 테스트 그룹은 Docker 컨테이너에서 테스트하는 경우 적용할 수 없습니다.

Docker 종속성(IDT v2.2.0 이상에서 지원됨)

디바이스가 Greengrass Docker 애플리케이션 배포 커넥터를 사용하여 컨테이너를 실행하는 데 필요한 모든 기술 종속성을 충족하는지 검증합니다.

이 테스트 그룹은 Docker 컨테이너에서 테스트하는 경우 적용할 수 없습니다.

HSI(하드웨어 보안 통합)

제공된 HSI 공유 라이브러리가 하드웨어 보안 모듈(HSM)과 인터페이스할 수 있고 필요한 PKCS#11 API를 올바로 구현하는지 확인합니다. HSM 및 공유 라이브러리가 CSR에 서명하고 TLS 작업을 수행하며 올바른 키 길이와 퍼블릭 키 알고리즘을 제공할 수 있어야 합니다.

스트림 관리자 종속성(IDT v2.2.0 이상에서 지원됨)

디바이스가 AWS IoT Greengrass 스트림 관리자를 실행하는 데 필요한 모든 기술적 종속성을 충족하는지 검증합니다.

기계 학습 종속성(IDT v3.1.0 이상에서 지원됨)

디바이스가 로컬로 ML 추론을 실행하는 데 필요한 모든 기술적 종속성을 충족하는지 검증합니다.

기계 학습 추론 테스트(IDT v3.1.0 이상에서 지원됨)

ML 추론이 테스트 중인 지정된 디바이스에서 수행될 수 있는지 검증합니다. 자세한 내용은 the section called “선택 사항: ML 검증을 위해 디바이스 구성” 섹션을 참조하세요.

기계 학습 추론 컨테이너 테스트(IDT v3.1.0 이상에서 지원됨)

ML 추론이 테스트 중인 지정된 디바이스에서 수행될 수 있는지와 Greengrass 코어에서 컨테이너 모드로 실행될 수 있는지 검증합니다. 자세한 내용은 the section called “선택 사항: ML 검증을 위해 디바이스 구성” 섹션을 참조하세요.

IDT v1.3.3 and earlier

코어 검증을 위한 필수 테스트 그룹

이러한 테스트는 AWS IoT Greengrass 디바이스의 AWS Partner 디바이스 카탈로그를 검증하기 위해 필요합니다.

AWS IoT Greengrass 코어 종속성

디바이스가 AWS IoT Greengrass 코어 소프트웨어에 대한 모든 소프트웨어 및 하드웨어 요구 사항을 충족하는지 여부를 확인합니다.

조합(디바이스 보안 상호 작용)

Greengrass 클라우드의 그룹에서 연결 정보를 변경하여 Greengrass 코어 디바이스에서 디바이스 인증서 관리자 및 IP 감지 기능을 확인합니다. 테스트 그룹은 AWS IoT Greengrass 서버 인증서를 교체하고 AWS IoT Greengrass가 연결을 허용하는지 확인합니다.

배포(IDT v1.2 및 이전 버전에 필요)

디바이스에 Lambda 함수를 배포할 수 있는지 검증합니다.

Device Certificate Manager(DCM)

AWS IoT Greengrass 디바이스 인증서 관리자가 시작 시 서버 인증서를 생성하고 곧 만료되는 인증서를 교체할 수 있는지 확인합니다.

IPD(IP 감지)

Greengrass 코어 디바이스에서 IP 주소가 변경될 때 코어 연결 정보가 업데이트되는지 확인합니다. 자세한 내용은 자동 IP 감지 활성화 섹션을 참조하세요.

로그

AWS IoT Greengrass 로깅 서비스가 Python으로 작성된 사용자 Lambda 함수를 사용하여 로그 파일에 쓸 수 있는지 확인합니다.

MQTT

두 Lambda 함수로 라우팅되는 주제에 대한 메시지를 전송하여 AWS IoT Greengrass 메시지 라우터 기능을 확인합니다.

기본

AWS IoT Greengrass에서 기본 (컴파일된) Lambda 함수를 실행할 수 있는지 확인합니다.

무선(OTA)

디바이스가 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트를 성공적으로 수행할 수 있는지 검증합니다.

침투

하드 링크/소프트 링크 보호 및 seccomp가 활성화되지 않은 경우 AWS IoT Greengrass Core 소프트웨어가 시작되지 않는지 확인합니다. 이 그룹은 기타 보안 관련 기능을 확인하는 데도 사용됩니다.

섀도우

로컬 섀도우와 섀도우 클라우드의 동기화 기능을 확인합니다.

스풀러

MQTT 메시지가 기본 스풀러 구성으로 대기되었는지 확인합니다.

토큰 교환 서비스(TES)

AWS IoT Greengrass가 유효한 AWS 자격 증명에 대한 코어 인증서를 교환할 수 있는지 확인합니다.

버전

제공된 AWS IoT Greengrass의 버전이 사용하는 AWS IoT 디바이스 테스터 버전과 호환되는지 확인합니다.

선택적 테스트 그룹

이러한 테스트는 선택 사항입니다. 선택적 테스트 자격을 선택하면 디바이스가 AWS Partner 디바이스 카탈로그에 추가 기능과 함께 나열됩니다.

컨테이너 종속성

디바이스가 Lambda 함수를 컨테이너 모드로 실행하는 데 필요한 모든 종속성을 충족하는지 확인합니다.

HSI(하드웨어 보안 통합)

제공된 HSI 공유 라이브러리가 하드웨어 보안 모듈(HSM)과 인터페이스할 수 있고 필요한 PKCS#11 API를 올바로 구현하는지 확인합니다. HSM 및 공유 라이브러리가 CSR에 서명하고 TLS 작업을 수행하며 올바른 키 길이와 퍼블릭 키 알고리즘을 제공할 수 있어야 합니다.

로컬 리소스 액세스

AWS IoT Greengrass LRA(로컬 리소스 액세스) API를 통해 컨테이너화된 Lambda 함수에 다양한 Linux 사용자와 그룹이 소유한 로컬 파일과 디렉터리에 대한 액세스 권한을 제공함으로써 AWS IoT Greengrass의 LRA 기능을 확인합니다. Lambda 함수는 로컬 리소스 액세스 구성을 기반으로 로컬 리소스에 대한 액세스를 허용하거나 거부해야 합니다.

네트워크

Lambda 함수에서 소켓 연결을 설정할 수 있는지 확인합니다. 이러한 소켓 연결은 Greengrass 코어 구성에 기반하여 허용되거나 거부되어야 합니다.

검증 제품군 실행을 위한 사전 요구 사항 AWS IoT Greengrass

이 섹션에서는 검증 제품군을 AWS IoT Greengrass 실행하기 위한 AWS IoT 장치 테스터 (IDT) 를 사용하기 위한 사전 요구 사항을 설명합니다. AWS IoT Greengrass

최신 버전의 AWS IoT 디바이스 테스터를 다운로드하십시오. AWS IoT Greengrass

최신 버전의 IDT를 다운로드하여 읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 소프트웨어의 압축을 풉니다.

> **Note**
>
> 여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하는 것은 지원되지 않습니다. 로컬 드라이브에 IDT 패키지의 압축을 풀고 로컬 워크스테이션에서 IDT 바이너리를 실행하는 것이 좋습니다.
> Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용 중인 경우 경로를 260자 제한 아래로 유지하도록 IDT 압축을 `C:\` 또는 `D:\` 같은 루트 디렉터리에 풉니다.

생성 및 구성 AWS 계정

IDT를 AWS IoT Greengrass사용하기 전에 다음 단계를 수행해야 합니다.

1. 생성하기. AWS 계정 이미 가지고 있다면 2단계로 건너뛰세요. AWS 계정
2. IDT에 대한 권한을 구성합니다.

이러한 계정 권한을 통해 IDT는 사용자를 대신하여 AWS 서비스에 액세스하고 AWS IoT 사물, Greengrass 그룹, Lambda 함수 등의 AWS 리소스를 생성할 수 있습니다.

이러한 리소스를 생성하기 위해 IDT for는 `config.json` 파일에 구성된 AWS 자격 증명을 AWS IoT Greengrass 사용하여 사용자를 대신하여 API 호출을 수행합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

> **Note**
>
> 대부분의 테스트에서 Amazon Web Services 프리 티어를 사용할 수 있지만 AWS 계정에 가입할 때는 신용 카드를 등록해야 합니다. 자세한 내용은 계정에 프리 티어가 적용되는데 결제 방법이 필요한 이유는 무엇입니까?를 참조하십시오.

1단계: 만들기 AWS 계정

이 단계에는 AWS 계정을 생성하고 구성합니다. 이미 가지고 있다면 AWS 계정다음으로 건너뛰세요. the section called “2단계: IDT에 대한 권한 구성”

가입해 보세요 AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 여세요.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중에는 전화를 받고 키패드로 인증 코드를 입력하는 과정이 있습니다.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스 액세스 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 가서 내 계정(My Account)을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요. AWS 계정 루트 사용자

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면AWS 로그인 사용 설명서의 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 참조하십시오.

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 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

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 참조하십시오.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.

   지침은AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

2단계: IDT에 대한 권한 구성

이 단계에서는 IDT for 가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 AWS IoT Greengrass 사용하는 권한을 구성합니다. AWS Management Console 또는 AWS Command Line Interface (AWS CLI) 를 사용하여 IAM 정책을 생성하고 IDT용 테스트 사용자를 생성한 다음 정책을 사용자에게 연결할 수 있습니다. IDT에 대한 테스트 사용자를 이미 작성한 경우 the section called “IDT 테스트를 실행하도록 디바이스 구성” 또는 the section called “선택 사항: 도커 컨테이너 구성” 단계로 건너뜁니다.

- IDT에 대한 권한 구성(콘솔)
- IDT에 대한 권한 구성(AWS CLI)

IDT에 대한 권한을 구성하려면(콘솔)

콘솔을 사용하여 AWS IoT Greengrass용 IDT에 대한 권한을 구성하려면 다음 단계를 수행하십시오.

1. IAM 콘솔에 로그인합니다.
2. 특정 권한으로 역할을 생성하는 권한을 부여하는 고객 관리형 정책을 만듭니다.

   a. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.

   b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
```

```
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ]
        }
    ]
}
```

Important

다음 정책은 AWS IoT Greengrass에서 IDT에 필요한 역할을 생성하고 관리하는 권한을 부여합니다. 여기에는 다음과 같은 AWS 관리형 정책을 연결할 수 있는 권한이 포함됩니다.

- AWSGreengrassResourceAccessRolePolicy
- 그린그라소타 UpdateArtifactAccess
- AWSLambdaBasicExecutionRole

c. 다음: 태그를 선택합니다.

d. 다음: 검토를 선택합니다.

e. 이름에 **IDTGreengrassIAMPermissions**를 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.

f. 정책 생성(Create policy)을 선택합니다.

3. IAM 사용자를 만들고 AWS IoT Greengrass용 IDT에 필요한 권한을 연결합니다.

a. IAM 사용자를 생성합니다. IAM 사용 설명서에서 IAM 사용자 생성(콘솔)의 1~5단계를 따르십시오.

b. IAM 사용자에게 권한을 연결합니다.

i. 권한 설정 페이지에서 기존 정책 직접 연결을 선택합니다.

ii. 이전 단계에서 만든 IDTGreengrassIAMPermissions 정책을 검색합니다. 확인란을 선택합니다.

iii. 정책을 검색하세요. AWSIoTDeviceTesterForGreengrassFullAccess 확인란을 선택합니다.

> **Note**
>
> AWSIoTDeviceTesterForGreengrassFullAccessIDT가 테스트에 사용되는 AWS 리소스를 만들고 액세스하는 데 필요한 권한을 정의하는 AWS 관리형 정책입니다. 자세한 정보는 the section called “AWS IDT용 관리형 정책”을 참조하세요.

c. 다음: 태그를 선택합니다.

d. Next: Review(다음: 검토)를 선택하여 선택 사항의 요약을 봅니다.

e. 사용자 생성을 선택합니다.

f. 사용자의 액세스 키(액세스 키 ID와 비밀 액세스 키)를 보려면 암호와 액세스 키 옆에 있는 Show(표시)를 선택합니다. 액세스 키를 저장하려면 Download .csv(csv 다운로드)를 선택한 후 안전한 위치에 파일을 저장합니다. 나중에 이 정보를 사용하여 AWS 자격 증명 파일을 구성합니다.

4. 다음 단계: 물리적 장치를 구성합니다.

IDT에 대한 권한을 구성하려면(AWS CLI)

다음 단계에 따라 를 사용하여 AWS CLI IDT에 대한 권한을 구성하십시오. AWS IoT Greengrass콘솔에서 권한을 이미 구성한 경우 the section called “IDT 테스트를 실행하도록 디바이스 구성” 또는 the section called “선택 사항: 도커 컨테이너 구성” 단계로 건너뜁니다.

1. 컴퓨터에 를 설치하고 구성하십시오 (아직 설치되지 않은 AWS CLI 경우). AWS Command Line Interface 사용 설명서 AWS CLI설치 단계를 따르십시오.

   > **Note**
   >
   > 명령줄 셸에서 AWS 서비스와 상호 작용하는 데 사용할 수 있는 오픈 소스 도구입니다. AWS CLI

2. IDT 및 AWS IoT Greengrass 역할을 관리할 수 있는 권한을 부여하는 고객 관리형 정책을 만듭니다.

   Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
```

```
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ]
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --
policy-document '{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid
\": \"ManageRolePoliciesForIDTGreengrass\",\"Effect\": \"Allow\",
\"Action\": [\"iam:DetachRolePolicy\", \"iam:AttachRolePolicy\"],
 \"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"],\"Condition\": {\"ArnEquals\": {\"iam:PolicyARN\":
 [\"arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
\",\"arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess
\",\"arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole\"]}}},
{\"Sid\": \"ManageRolesForIDTGreengrass\",\"Effect\": \"Allow\",\"Action\":
 [\"iam:CreateRole\",\"iam:DeleteRole\", \"iam:PassRole\", \"iam:GetRole
\"],\"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"]}]}'
```

> **Note**
>
> Linux, macOS 또는 Unix 터미널 명령과 다른 JSON 구문을 사용하기 때문에 이 단계에는 Windows 명령 프롬프트 예제가 포함되어 있습니다.

3. IAM 사용자를 만들고 AWS IoT Greengrass용 IDT에 필요한 권한을 연결합니다.

   a. IAM 사용자를 생성합니다. 이 예제 설정에서 사용자의 이름은 `IDTGreengrassUser`입니다.

   ```
   aws iam create-user --user-name IDTGreengrassUser
   ```

   b. 2단계에서 생성한 `IDTGreengrassIAMPermissions` 정책을 IAM 사용자에게 연결합니다. <account-id>명령의 ID를 사용자 ID로 바꾸십시오. AWS 계정

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

   c. `AWSIoTDeviceTesterForGreengrassFullAccess` 정책을 IAM 사용자에게 연결합니다.

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
   ```

   > **Note**
   >
   > AWSIoTDeviceTesterForGreengrassFullAccessIDT가 테스트에 사용되는 AWS 리소스를 만들고 액세스하는 데 필요한 권한을 정의하는 AWS 관리형 정책입니다. 자세한 정보는 the section called “AWS IDT용 관리형 정책”을 참조하세요.

4. 사용자에 대한 보안 액세스 키를 만듭니다.

   ```
   aws iam create-access-key --user-name IDTGreengrassUser
   ```

   출력을 안전한 위치에 저장합니다. 나중에 이 정보를 사용하여 AWS 자격 증명 파일을 구성합니다.

5. 다음 단계: 물리적 장치를 구성합니다.

AWSAWS IoT 디바이스 테스터에 대한 관리형 정책

AWSIoTDeviceTesterForGreengrassFullAccess관리형 정책을 통해 IDT는 작업을 실행하고 사용량 지표를 수집할 수 있습니다. 이 정책은 다음 IDT 권한을 부여합니다.

- `iot-device-tester:CheckVersion`. 세트 AWS IoT Greengrass, 테스트 스위트 및 IDT 버전이 호환되는지 확인하세요.
- `iot-device-tester:DownloadTestSuite`. 테스트 제품군을 다운로드합니다.
- `iot-device-tester:LatestIdt`. 다운로드할 수 있는 최신 IDT 버전에 대한 정보를 가져옵니다.
- `iot-device-tester:SendMetrics`. IDT가 테스트에 대해 수집하는 사용량 데이터를 게시합니다.
- `iot-device-tester:SupportedVersion`. IDT에서 지원하는 테스트 스위트 버전 목록 AWS IoT Greengrass 및 테스트 도구 모음 버전을 확인하세요. 이 정보는 명령줄 창에 표시됩니다.

IDT 테스트를 실행하도록 디바이스 구성

디바이스를 구성하려면 AWS IoT Greengrass 종속성을 설치하고, AWS IoT Greengrass 코어 소프트웨어를 구성하고, 디바이스에 액세스할 호스트 컴퓨터를 구성하고, 디바이스의 사용자 권한을 구성해야 합니다.

테스트 대상 디바이스에서 AWS IoT Greengrass 종속성 확인

AWS IoT Greengrass용 IDT가 디바이스를 테스트하기 전에 AWS IoT Greengrass 시작하기에 설명된 대로 디바이스를 설정했는지 확인합니다. 지원되는 플랫폼에 대한 자세한 내용은 지원되는 플랫폼을 참조하십시오.

AWS IoT Greengrass 소프트웨어 구성

AWS IoT Greengrass용 IDT는 디바이스가 특정 AWS IoT Greengrass 버전과 호환되는지 테스트합니다. IDT는 디바이스에서 AWS IoT Greengrass를 테스트하는 두 가지 옵션을 제공합니다.

- AWS IoT Greengrass 코어 소프트웨어의 한 버전을 다운로드하여 사용합니다. IDT에서 자동으로 해당 소프트웨어를 설치합니다.
- 디바이스에 이미 설치된 AWS IoT Greengrass 코어 소프트웨어 버전을 사용하십시오.

> **Note**
>
> AWS IoT Greengrass의 각 버전에는 해당하는 IDT 버전이 있습니다. 사용 중인 AWS IoT Greengrass 버전에 해당하는 IDT 버전을 다운로드해야 합니다.

다음 단원에서는 이러한 옵션에 대해 설명합니다. 이러한 옵션 중 하나만 수행해야 합니다.

옵션 1: AWS IoT Greengrass 코어 소프트웨어를 다운로드하고 이를 사용하도록 AWS IoT 디바이스 테스터 구성

AWS IoT Greengrass코어 소프트웨어 다운로드 페이지에서 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

1. 올바른 아키텍처와 Linux 배포를 찾은 다음 Download(다운로드)를 선택합니다.
2. tar.gz 파일을 *<device-tester-extract-location>*/products/greengrass/ggc에 복사합니다.

> **Note**
>
> AWS IoT Greengrass tar.gz 파일의 이름은 변경하지 마십시오. 동일한 운영 체제 및 아키텍처에 대해 이 디렉터리에 여러 파일을 배치하지 마십시오. 예를 들어 해당 디렉터리에 greengrass-linux-armv7l-1.7.1.tar.gz 파일과 greengrass-linux-armv7l-1.8.1.tar.gz 파일이 모두 있으면 테스트가 실패합니다.

옵션 2: 기존 AWS IoT Greengrass 설치를 AWS IoT 디바이스 테스터에 사용

greengrassLocation 속성을 *<device-tester-extract-location>*/configs 폴더의 device.json 파일에 추가하여 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어를 테스트하도록 IDT를 구성합니다. 예:

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

device.json 파일에 대한 자세한 내용은 device.json 구성를 참조하세요.

Linux 디바이스에서 AWS IoT Greengrass 코어 소프트웨어의 기본 위치는 /greengrass입니다.

> **Note**
>
> 디바이스에 아직 시작되지 않은 AWS IoT Greengrass 코어 소프트웨어가 설치되어 있어야 합니다.
> 디바이스에 `ggc_user` 사용자 및 `ggc_group`을 추가했는지 확인합니다. 자세한 내용은 AWS IoT Greengrass에 대한 환경 설정을 참조하십시오.

테스트 대상 디바이스에 액세스하도록 호스트 컴퓨터 구성

IDT는 호스트 컴퓨터에서 실행되며, SSH를 사용하여 디바이스에 연결할 수 있어야 합니다. IDT가 테스트 대상 디바이스에 대한 SSH 액세스를 획득하도록 허용하는 옵션은 두 가지가 있습니다.

1. 여기에서 설명하는 지침에 따라 SSH 키 페어를 생성하고 해당 키가 암호를 지정하지 않고 테스트 대상 디바이스에 로그인할 수 있도록 권한을 부여합니다.
2. `device.json` 파일의 각 디바이스에 사용자 이름 및 암호를 제공합니다. 자세한 내용은 device.json 구성 섹션을 참조하세요.

임의의 SSL 구현을 사용하여 SSH 키를 생성할 수 있습니다. 다음 지침은 SSH-KEYGEN 또는 PuTTYgen(Windows)을 사용하는 방법을 보여줍니다. 다른 SSL 구현을 사용 중인 경우 해당 구현에 대한 설명서를 참조하십시오.

IDT는 SSH 키를 사용하여 테스트 대상 디바이스에 인증합니다.

SSH-KEYGEN을 사용하여 SSH 키를 생성하려면

1. SSH 키를 생성합니다,

   공개 SSH ssh-keygen 명령을 사용하여 SSH 키 페어를 생성할 수 있습니다. 이미 호스트 컴퓨터에 SSH 키 페어가 있는 경우 IDT 전용 SSH 키 페어를 생성하는 것이 가장 좋습니다. 그러면 테스트를 완료한 후 호스트 컴퓨터가 더 이상 암호 없이 디바이스에 연결할 수 없습니다. 또한 원하는 사용자만 원격 디바이스에 액세스할 수 있도록 제한할 수 있습니다.

   > **Note**
   >
   > Windows에는 SSH 클라이언트가 설치되어 있지 않습니다. Windows에 SSH 클라이언트 설치에 대한 내용은 SSH 클라이언트 소프트웨어 다운로드를 참조하십시오.

ssh-keygen 명령은 키 페어 저장 이름과 경로를 입력하라는 메시지를 표시합니다. 기본적으로 키 페어 파일은 `id_rsa`(프라이빗 키) 및 `id_rsa.pub`(퍼블릭 키)로 이름 지정됩니다. macOS와 Linux에서 이러한 파일의 기본 위치는 `~/.ssh/`입니다. Windows에서 기본 위치는 `C:\Users\<user-name>\.ssh`입니다.

메시지가 표시되면 SSH 키를 보호하기 위한 키 구문을 입력합니다. 자세한 내용은 새 SSH 키 생성을 참조하십시오.

2. 테스트 대상 디바이스에 권한 있는 SSH 키를 추가합니다,

   IDT는 SSH 프라이빗 키를 사용하여 테스트 대상 디바이스에 로그인해야 합니다. 테스트 대상 디바이스에 로그인하도록 SSH 프라이빗 키를 승인하려면 호스트 컴퓨터의 ssh-copy-id 명령을 사용합니다. 이 명령은 테스트 대상 디바이스에서 `~/.ssh/authorized_keys` 파일에 퍼블릭 키를 추가합니다. 예:

   $ ssh-copy-id ***<remote-ssh-user>***@***<remote-device-ip>***

   여기서 *remote-ssh-user*는 테스트 대상 디바이스에 로그인하는 데 사용하는 사용자 이름이고, *remote-device-ip*는 테스트를 실행할 테스트 대상 디바이스의 IP 주소입니다. 예:

   ssh-copy-id pi@192.168.1.5

   메시지가 표시되면 ssh-copy-id 명령에서 지정한 사용자 이름에 대한 암호를 입력합니다.

   ssh-copy-id는 퍼블릭 키가 `id_rsa.pub`로 이름 지정되고 기본 위치에 저장된다고 가정합니다(macOS와 Linux에서는 `~/.ssh/`, Windows에서는 `C:\Users\<user-name>\.ssh`) 퍼블릭 키의 이름을 다르게 지정하거나 다른 위치에 저장한 경우, ssh-copy-id에 -i 옵션을 사용하여 SSH 퍼블릭 키의 정규화된 경로를 지정해야 합니다(예: ssh-copy-id -i ~/my/path/myKey.pub). SSH 키 생성 및 퍼블릭 키 복사에 대한 자세한 내용은 SSH-COPY-ID를 참조하십시오.

PuTTYgen을 사용하여 SSH 키를 생성하려면(Windows만 해당)

1. 테스트 대상 디바이스에 OpenSSH 서버 및 클라이언트가 설치되어 있는지 확인합니다. 자세한 내용은 OpenSSH를 참조하십시오.
2. 테스트 대상 디바이스에 PuTTYgen을 설치합니다.
3. PuTTYgen을 엽니다.
4. 생성을 선택하고 마우스를 상자 안으로 이동하여 프라이빗 키를 생성합니다.

5. Conversions(변환) 메뉴에서 Export OpenSSH key(OpenSSH 키 내보내기)를 선택하고 프라이빗 키를 `.pem` 파일 확장명으로 저장합니다.
6. 테스트 대상 디바이스에서 `/home/<user>/.ssh/authorized_keys` 파일에 퍼블릭 키를 추가합니다.

   a. PuTTYgen 창에서 퍼블릭 키 텍스트를 복사합니다.

   b. PuTTY를 사용하여 테스트 대상 디바이스에서 세션을 생성합니다.

      i. 명령 프롬프트 또는 Windows Powershell 창에서 다음 명령을 실행합니다.

      C:/*<path-to-putty>*/putty.exe -ssh *<user>*@*<dut-ip-address>*

      ii. 메시지가 표시되면 디바이스의 암호를 입력합니다.

      iii. vi 또는 다른 텍스트 편집기를 사용하여 테스트 대상 디바이스의 `/home/<user>/.ssh/authorized_keys` 파일에 퍼블릭 키를 추가합니다.
7. 각 테스트 대상 디바이스에 대해 사용자 이름, IP 주소, 방금 호스트 컴퓨터에 저장한 프라이빗 키 파일의 경로로 `device.json` 파일을 업데이트합니다. 자세한 내용은 the section called “device.json 구성” 섹션을 참조하세요. 프라이빗 키에 전체 경로 및 파일 이름을 제공하고 슬래시('/')를 사용해야 합니다. 예를 들어 Windows 경로 `C:\DT\privatekey.pem`의 경우 `device.json` 파일에 `C:/DT/privatekey.pem`을 사용합니다.

디바이스에서 사용자 권한 구성

IDT는 테스트 대상 디바이스에서 다양한 디렉터리와 파일에 대해 작업을 수행합니다. 이러한 작업 중 일부는 승격된 권한(sudo 사용)이 필요합니다. 이러한 작업을 자동화하기 위해서는 AWS IoT Greengrass용 IDT가 암호 입력 메시지 없이 sudo를 사용하여 명령을 실행할 수 있어야 합니다.

암호 입력 메시지 없이 sudo 액세스를 허용하려면 테스트 대상 디바이스에서 다음 단계를 수행합니다.

> **Note**
>
> `username`은 IDT가 테스트 대상 디바이스에 액세스하는 데 사용하는 SSH 사용자를 나타냅니다.

sudo 그룹에 사용자를 추가하려면

1. 테스트 대상 디바이스에서 `sudo usermod -aG sudo <username>`을 실행합니다.
2. 변경 사항을 적용하려면 로그아웃했다가 다시 로그인하십시오.

3. 사용자 이름이 성공적으로 추가되었는지 확인하려면 sudo echo test를 실행합니다. 암호 입력 메시지가 표시되지 않으면 사용자가 제대로 구성된 것입니다.
4. `/etc/sudoers` 파일을 열고 파일 끝에 다음 줄을 추가합니다.

   ```
   <ssh-username> ALL=(ALL) NOPASSWD: ALL
   ```

선택적 기능을 테스트하도록 디바이스 구성

다음 주제에서는 선택적 기능에 대한 IDT 테스트를 실행하도록 디바이스를 구성하는 방법을 설명합니다. 이러한 기능을 테스트하려는 경우에만 다음 구성 단계를 수행하세요. 그렇지 않은 경우 the section called “IDT 설정 구성”를 계속 진행합니다.

주제

- 선택 사항: AWS IoT Greengrass에 IDT용 도커 컨테이너 구성
- 선택 사항: ML 검증을 위해 디바이스 구성

선택 사항: AWS IoT Greengrass에 IDT용 도커 컨테이너 구성

AWS IoT Greengrass는 도커 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 쉽게 실행할 수 있도록 도커 이미지와 도커 파일을 제공합니다. AWS IoT Greengrass 컨테이너를 설정한 후 IDT 테스트를 실행할 수 있습니다. 현재 AWS IoT Greengrass에서 IDT를 실행하는 데는 x86_64 도커 아키텍처만 지원됩니다.

이 기능을 사용하려면 IDT v2.3.0 이상이 필요합니다.

IDT 테스트를 실행하도록 도커 컨테이너를 설정하는 프로세스는 AWS IoT Greengrass에서 제공하는 도커 이미지를 사용하는지 아니면 도커 파일을 사용하는지에 따라 다릅니다.

- 도커 이미지 사용. 도커 이미지에는 AWS IoT Greengrass 코어 소프트웨어와 종속성이 설치되어 있습니다.
- 도커 파일 사용. 도커 파일에는 사용자 지정 AWS IoT Greengrass 컨테이너 이미지를 작성하는 데 사용할 수 있는 소스 코드가 포함되어 있습니다. 다른 플랫폼 아키텍처에서 실행하거나 이미지 크기를 줄이기 위해 이미지를 수정할 수 있습니다.

> **Note**
>
> AWS IoT Greengrass은(는) AWS IoT Greengrass 코어 소프트웨어 버전 1.11.1의 Docker파일 일 또는 Docker 이미지를 제공하지 않습니다. 사용자 지정 컨테이너 이미지에서 IDT 테스

트를 실행하려면 AWS IoT Greengrass에서 제공하는 도커 파일에 정의된 종속성이 이미지에 포함되어야 합니다.

도커 컨테이너에서 AWS IoT Greengrass를 실행할 때 다음 기능은 사용할 수 없습니다.

- Greengrass 컨테이너 모드에서 실행되는 커넥터 Docker 컨테이너에서 커넥터를 실행하려면 커넥터가 컨테이너 없음 모드로 실행되어야 합니다. 컨테이너 없음 모드를 지원하는 커넥터를 찾으려면 the section called “AWS에서 제공한 Greengrass 커넥터” 단원을 참조하십시오. 이러한 커넥터 중 일부에는 컨테이너 없음으로 설정해야 하는 격리 모드 파라미터가 있습니다.
- 로컬 디바이스 및 볼륨 리소스. Docker 컨테이너에서 실행되는 사용자 정의 Lambda 함수는 코어의 디바이스 및 볼륨에 직접 액세스해야 합니다.

AWS IoT Greengrass에서 제공하는 도커 이미지 구성

IDT 테스트를 실행하도록 AWS IoT Greengrass 도커 이미지를 구성하려면 다음 단계를 수행합니다.

사전 조건

이 자습서를 시작하기 전에 다음 작업을 수행해야 합니다.

- 선택한 AWS Command Line Interface(AWS CLI) 버전에 따라 호스트 컴퓨터에 다음 소프트웨어 및 버전을 설치해야 합니다.

  AWS CLI version 2

  - Docker, 버전 18.09 이상. 이전 버전도 작동할 수 있지만 18.09 이상을 사용하는 것이 좋습니다.
  - AWS CLI 버전 2.6.30 이상
    - AWS CLI 버전 2를 설치하려면 AWS CLI 버전 2 설치를 참조하십시오.
    - AWS CLI을(를) 구성하려면 AWS CLI 구성을 참조하십시오.

    > **Note**
    >
    > Windows 컴퓨터에서 최신 AWS CLI 버전 2로 업그레이드하려면 MSI 설치 프로세스를 반복해야 합니다.

AWS CLI version 1

- Docker, 버전 18.09 이상. 이전 버전도 작동할 수 있지만 18.09 이상을 사용하는 것이 좋습니다.
- Python, 버전 3.6 이상.
- pip 버전 18.1 이상
- AWS CLI 버전 1.11.63 이상
  - AWS CLI 버전 1을 설치하려면 AWS CLI 버전 1 설치를 참조하십시오.
  - AWS CLI을(를) 구성하려면 AWS CLI 구성을 참조하십시오.
  - AWS CLI 버전 1의 최신 버전으로 업데이트하려면 다음 명령을 실행합니다.

```
pip install awscli --upgrade --user
```

> **Note**
>
> Windows에서 AWS CLI 버전 1의 MSI 설치를 사용하는 경우 다음 사항에 유의하십시오.
>
> - 이 AWS CLI 버전 1 설치에서 botocore를 설치하지 못하는 경우 Python 및 pip 설치를 사용해 봅니다.
> - 최신 AWS CLI 버전으로 업그레이드하려면 MSI 설치 프로세스를 반복해야 합니다.

- Amazon Elastic Container Registry(Amazon ECR) 리소스에 액세스하려면 다음 권한을 부여해야 합니다.
  - Amazon ECR의 요구 사항에 따라 사용자가 레지스트리에 대해 인증하고 Amazon ECR 리포지토리에서 이미지를 푸시 또는 풀하기 전에 AWS Identity and Access Management (IAM) 정책을 통해 `ecr:GetAuthorizationToken` 권한을 부여해야 합니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 Amazon ECR 리포지토리 정책 예시 및 One Amazon ECR 리포지토리 액세스를 참조하십시오.

1. 도커 이미지를 다운로드하고 컨테이너를 구성합니다. Docker Hub 또는 Amazon Elastic Container Registry(Amazon ECR)에서 사전 작성 이미지를 다운로드하고 Windows, macOS 및 Linux(x86_64) 플랫폼에서 이 이미지를 실행할 수 있습니다.

the section called “Amazon ECR에서 AWS IoT Greengrass 컨테이너 이미지 가져오기”에서 Docker 이미지를 다운로드하려면 의 모든 단계를 완료합니다. 그런 다음 이 항목으로 돌아와 구성을 계속합니다.

2. Linux 사용자만 해당: IDT를 실행하는 사용자에게 도커 명령을 실행할 권한이 있는지 확인하십시오. 자세한 내용은 도커 설명서의 도커를 루트가 아닌 사용자로 관리를 참조하십시오.

3. AWS IoT Greengrass 컨테이너를 실행하려면 운영 체제에 해당하는 명령을 사용합니다.

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

   - *<host-path-to-kernel-config-file>*을 호스트의 커널 구성 파일 경로로 대체하고 *<container-path>*를 볼륨이 컨테이너에 탑재된 경로로 대체합니다.

     호스트의 커널 구성 파일은 일반적으로 /proc/config.gz 또는 /boot/config-*<kernel-release-date>*에 있습니다. uname -r을 실행하여 *<kernel-release-date>* 값을 확인할 수 있습니다.

     예: /boot/config-*<kernel-release-date>*에서 구성 파일을 탑재하려면 다음 명령을 실행합니다.

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     예: proc/config.gz에서 구성 파일을 탑재하려면 다음 명령을 실행합니다.

     ```
     -v /proc/config.gz:/proc/config.gz \
     ```

   - 명령에서 *<image-repository>*:*<tag>*를 대상 이미지의 리포지토리 및 태그 이름으로 대체합니다.

     예: AWS IoT Greengrass 코어 소프트웨어의 최신 버전을 가리키려면 다음 명령을 실행합니다.

     ```
     216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
     ```

AWS IoT Greengrass Docker 이미지 목록을 얻으려면 다음 명령을 실행합니다.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 명령에서 *<image-repository>*:*<tag>*를 대상 이미지의 리포지토리 및 태그 이름으로 대체합니다.

  예: AWS IoT Greengrass 코어 소프트웨어의 최신 버전을 가리키려면 다음 명령을 실행합니다.

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  AWS IoT Greengrass Docker 이미지 목록을 얻으려면 다음 명령을 실행합니다.

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 명령에서 *<image-repository>*:*<tag>*를 대상 이미지의 리포지토리 및 태그 이름으로 대체합니다.

  예: AWS IoT Greengrass 코어 소프트웨어의 최신 버전을 가리키려면 다음 명령을 실행합니다.

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

AWS IoT Greengrass Docker 이미지 목록을 얻으려면 다음 명령을 실행합니다.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

> **Important**
>
> IDT로 테스트할 때는 일반적인 AWS IoT Greengrass 용도로 이미지를 실행하는 데 사용되는 `--entrypoint /greengrass-entrypoint.sh \` 인수를 포함하지 마십시오.

4. 다음 단계: AWS 보안 인증 및 `device.json` 파일을 구성합니다.

AWS IoT Greengrass에서 제공하는 도커 파일 구성

IDT 테스트를 실행하도록 AWS IoT Greengrass 도커 파일에서 작성된 도커 이미지를 구성하려면 다음 단계를 수행합니다.

1. the section called “AWS IoT Greengrass Docker 소프트웨어”에서 도커 파일 패키지를 호스트 컴퓨터에 다운로드하고 압축을 풉니다.
2. `README.md`을 엽니다. 다음 세 단계에서는 이 파일의 섹션을 참조합니다.
3. Prerequisites(사전 조건) 섹션의 요구 사항을 충족하는지 확인합니다.
4. Linux 사용자만 해당: Enable Symlink and Hardlink Protection(symlink 및 hardlink 보호 활성화) 및 Enable IPv4 Network Forwarding(IPv4 네트워크 전달 활성화) 단계를 완료합니다.
5. Docker 이미지를 작성하려면 1단계. AWS IoT Greengrass Docker 이미지를 구축합니다. 그런 다음 이 항목으로 돌아와 구성을 계속합니다.
6. AWS IoT Greengrass 컨테이너를 실행하려면 운영 체제에 해당하는 명령을 사용합니다.

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

- *<host-path-to-kernel-config-file>*을 호스트의 커널 구성 파일 경로로 대체하고 *<container-path>*를 볼륨이 컨테이너에 탑재된 경로로 대체합니다.

  호스트의 커널 구성 파일은 일반적으로 /proc/config.gz 또는 /boot/config-*<kernel-release-date>*에 있습니다. uname -r을 실행하여 *<kernel-release-date>* 값을 확인할 수 있습니다.

  예: /boot/config-*<kernel-release-date>*에서 구성 파일을 탑재하려면 다음 명령을 실행합니다.

  ```
  -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
  ```

  예: proc/config.gz에서 구성 파일을 탑재하려면 다음 명령을 실행합니다.

  ```
  -v /proc/config.gz:/proc/config.gz \
  ```

- 명령에서 *<image-repository>*:*<tag>*를 대상 이미지의 리포지토리 및 태그 이름으로 대체합니다.

  예: AWS IoT Greengrass 코어 소프트웨어의 최신 버전을 가리키려면 다음 명령을 실행합니다.

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  AWS IoT Greengrass Docker 이미지 목록을 얻으려면 다음 명령을 실행합니다.

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 명령에서 *<image-repository>*:*<tag>*를 대상 이미지의 리포지토리 및 태그 이름으로 대체합니다.

예: AWS IoT Greengrass 코어 소프트웨어의 최신 버전을 가리키려면 다음 명령을 실행합니다.

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

AWS IoT Greengrass Docker 이미지 목록을 얻으려면 다음 명령을 실행합니다.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 명령에서 *<image-repository>*:*<tag>*를 대상 이미지의 리포지토리 및 태그 이름으로 대체합니다.

  예: AWS IoT Greengrass 코어 소프트웨어의 최신 버전을 가리키려면 다음 명령을 실행합니다.

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  AWS IoT Greengrass Docker 이미지 목록을 얻으려면 다음 명령을 실행합니다.

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

> **Important**
>
> IDT로 테스트할 때는 일반적인 AWS IoT Greengrass 용도로 이미지를 실행하는 데 사용되는 `--entrypoint /greengrass-entrypoint.sh \` 인수를 포함하지 마십시오.

7. 다음 단계: AWS 보안 인증 및 `device.json` 파일을 구성합니다.

AWS IoT Greengrass에 IDT용 도커 컨테이너를 설정하는 것과 관련된 문제 해결

다음 정보를 사용하면 AWS IoT Greengrass 테스트를 위해 IDT용 도커 컨테이너를 실행하는 중에 발생하는 문제를 해결하는 데 도움이 됩니다.

WARNING: Error loading config file:/home/user/.docker/config.json - stat /home/<user>/.docker/config.json: permission denied

Linux에서 `docker` 명령을 실행할 때 이 오류가 발생하면 다음 명령을 실행합니다. 다음 명령에서 *<user>*를 IDT를 실행하는 사용자로 대체합니다.

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rwx /home/<user>/.docker -R
```

선택 사항: ML 검증을 위해 디바이스 구성

AWS IoT Greengrass용 IDT는 디바이스가 클라우드 학습 모델을 사용하여 로컬에서 ML 추론을 수행할 수 있는지 검증하도록 기계 학습(ML) 검증 테스트를 제공합니다.

ML 검증 테스트를 실행하려면 먼저 the section called “IDT 테스트를 실행하도록 디바이스 구성”에 설명된 대로 디바이스를 구성해야 합니다. 그런 다음 이 주제의 단계를 수행하여 실행할 ML 프레임워크에 대한 종속 항목을 설치합니다.

ML 검증 테스트를 실행하려면 IDT v3.1.0 이상이 필요합니다.

ML 프레임워크 종속 항목 설치

모든 ML 프레임워크 종속 항목은 `/usr/local/lib/python3.x/site-packages` 디렉터리 아래에 설치해야 합니다. 올바른 디렉터리에 설치되도록 하려면 종속 항목을 설치할 때 `sudo` 루트 권한을 사용하는 것이 좋습니다. 가상 환경은 검증 테스트에서 지원되지 않습니다.

> **Note**
>
> 컨테이너화(Greengrass 컨테이너 모드)로 실행되는 Lambda 함수를 테스트하는 경우 `/usr/local/lib/python3.x` 아래에 Python 라이브러리에 대한 심볼릭 링크를 생성하는 것은 지원되지 않습니다. 오류를 방지하려면 올바른 디렉터리 아래에 종속 항목을 설치해야 합니다.

대상 프레임워크에 대한 종속 항목을 설치하는 단계를 따르세요.

- MXNet 종속성 설치

- the section called “TensorFlow 종속 항목 설치”
- DLR 종속 항목 설치

Apache MXNet 종속 항목 설치

이 프레임워크에 대한 IDT 검증 테스트에는 다음과 같은 종속 항목이 있습니다.

- Python 3.6 또는 Python 3.7.

> **Note**
>
> Python 3.6을 사용하고 있다면 Python 3.7에서 Python 3.6 바이너리로의 심볼 링크를 생성해야 합니다. 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다. 예:
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- Apache MXNet v1.2.1 이상.
- NumPy. 버전은 MXNet 버전과 호환되어야 합니다.

MXnet 설치

MXNet 설명서의 지침에 따라 MXNet을 설치합니다.

> **Note**
>
> Python 2.x와 Python 3.x가 모두 디바이스에 설치되어 있는 경우, 종속 항목을 설치하기 위해 실행하는 명령에서 Python 3.x를 사용합니다.

MXNet 설치 검증

다음 옵션 중 하나를 선택하여 MXNet 설치를 검증합니다.

옵션 1: 디바이스에 SSH 및 스크립트 실행

1. 디바이스에 SSH합니다.

2. 종속 항목이 올바르게 설치되었는지 확인하려면 다음 스크립트를 실행합니다.

```
sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

출력은 버전 번호를 인쇄하고 스크립트는 오류 없이 종료되어야 합니다.

옵션 2: IDT 종속 항목 테스트 실행

1. `device.json`이 ML 검증에 대해 구성되어 있는지 확인합니다. 자세한 내용은 the section called "ML 검증을 위해 device.json 구성" 섹션을 참조하세요.
2. 프레임워크에 대한 종속 항목 테스트를 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id mxnet_dependency_check
```

테스트 요약에 `mldependencies`에 대한 `PASSED` 결과가 표시됩니다.

TensorFlow 종속 항목 설치

이 프레임워크에 대한 IDT 검증 테스트에는 다음과 같은 종속 항목이 있습니다.

- Python 3.6 또는 Python 3.7.

> **Note**
>
> Python 3.6을 사용하고 있다면 Python 3.7에서 Python 3.6 바이너리로의 심볼 링크를 생성해야 합니다. 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다. 예:
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- TensorFlow 1.x.

TensorFlow 설치

TensorFlow 설명서의 지침에 따라 pip를 통해 또는 소스에서 TensorFlow 1.x를 설치합니다.

> **Note**
>
> Python 2.x와 Python 3.x가 모두 디바이스에 설치되어 있는 경우, 종속 항목을 설치하기 위해 실행하는 명령에서 Python 3.x를 사용합니다.

TensorFlow 설치 검증

다음 옵션 중 하나를 선택하여 TensorFlow 설치를 검증합니다.

옵션 1: 디바이스에 SSH 및 스크립트 실행

1. 디바이스에 SSH합니다.
2. 종속 항목이 올바르게 설치되었는지 확인하려면 다음 스크립트를 실행합니다.

   ```
   sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
   ```

   출력은 버전 번호를 인쇄하고 스크립트는 오류 없이 종료되어야 합니다.

옵션 2: IDT 종속 항목 테스트 실행

1. `device.json`이 ML 검증에 대해 구성되어 있는지 확인합니다. 자세한 내용은 the section called "ML 검증을 위해 device.json 구성" 섹션을 참조하세요.
2. 프레임워크에 대한 종속 항목 테스트를 실행합니다.

   ```
   devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-
   id tensorflow_dependency_check
   ```

   테스트 요약에 `mldependencies`에 대한 `PASSED` 결과가 표시됩니다.

Amazon SageMaker Neo 딥 러닝 런타임(DLR) 종속 항목 설치

이 프레임워크에 대한 IDT 검증 테스트에는 다음과 같은 종속 항목이 있습니다.

- Python 3.6 또는 Python 3.7.

> **Note**
>
> Python 3.6을 사용하고 있다면 Python 3.7에서 Python 3.6 바이너리로의 심볼 링크를 생성해야 합니다. 이렇게 하면 AWS IoT Greengrass에 대한 Python 요구 사항을 충족하도록 디바이스가 구성됩니다. 예:
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- SageMaker Neo DLR.
- numpy.

DLR 테스트 종속 항목을 설치한 후에는 모델을 컴파일해야 합니다.

DLR 설치

MXNet 설명서의 지침에 따라 Neo DLR을 설치합니다.

> **Note**
>
> Python 2.x와 Python 3.x가 모두 디바이스에 설치되어 있는 경우, 종속 항목을 설치하기 위해 실행하는 명령에서 Python 3.x를 사용합니다.

DLR 설치 검증

다음 옵션 중 하나를 선택하여 DLR 설치를 검증합니다.

옵션 1: 디바이스에 SSH 및 스크립트 실행

1. 디바이스에 SSH합니다.
2. 종속 항목이 올바르게 설치되었는지 확인하려면 다음 스크립트를 실행합니다.

```
sudo python3.7 -c "import dlr; print(dlr.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

출력은 버전 번호를 인쇄하고 스크립트는 오류 없이 종료되어야 합니다.

옵션 2: IDT 종속 항목 테스트 실행

1. `device.json`이 ML 검증에 대해 구성되어 있는지 확인합니다. 자세한 내용은 the section called "ML 검증을 위해 device.json 구성" 섹션을 참조하세요.
2. 프레임워크에 대한 종속 항목 테스트를 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id dlr_dependency_check
```

테스트 요약에 `mldependencies`에 대한 `PASSED` 결과가 표시됩니다.

DLR 모델 컴파일

ML 검증 테스트에 DLR 모델을 사용하려면 먼저 DLR 모델을 컴파일해야 합니다. 단계에서 다음 옵션 중 하나를 선택합니다.

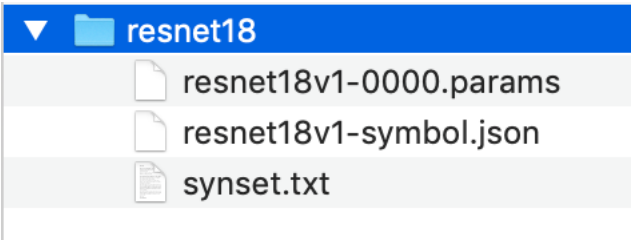
옵션 1: Amazon SageMaker를 사용하여 모델 컴파일

다음 단계에 따라 SageMaker를 사용하여 IDT에서 제공하는 ML 모델을 컴파일합니다. 이 모델은 Apache MXNet을 사용하여 사전 교육되어 있습니다.

1. SageMaker에서 지원되는 디바이스 유형인지 확인합니다. 자세한 내용은 Amazon SageMaker API 참조의 대상 디바이스 옵션을 참조하십시오. 디바이스 유형이 SageMaker에서 현재 지원되지 않는 경우 the section called "옵션 2: TVM을 사용하여 DLR 모델 컴파일"의 단계를 따르십시오.

> **Note**
>
> SageMaker에서 컴파일한 모델로 DLR 테스트를 실행하려면 4분에서 5분 정도 걸릴 수 있습니다. 이 시간 동안 IDT를 중지하지 마시기 바랍니다.

2. DLR용 컴파일되지 않은 사전 교육된 MXNet 모델이 포함된 tarball 파일을 다운로드합니다.
   - dlr-noncompiled-model-1.0.tar.gz
3. tarball의 압축을 풉니다. 이 명령은 다음과 같은 디렉터리 구조를 생성합니다.



4. `resnet18` 디렉터리에서 `synset.txt`를 이동합니다. 새 위치를 기록해 둡니다. 나중에 컴파일된 모델 디렉터리에 이 파일을 복사합니다.
5. `resnet18` 디렉터리의 내용을 압축합니다.

   ```
   tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
   ```

6. 압축된 파일을 AWS 계정의 Amazon S3 버킷에 업로드한 다음 모델 컴파일(콘솔)의 단계에 따라 컴파일 작업을 생성합니다.

   a. 입력 구성에 다음 값을 사용합니다.

   - 데이터 입력 구성에 `{"data": [1, 3, 224, 224]}`를 입력합니다.
   - 기계 학습 프레임워크에서 `MXNet`을 선택합니다.

   b. 출력 구성에 다음 값을 사용합니다.

   - S3 출력 위치에 컴파일된 모델을 저장할 Amazon S3 버킷 또는 폴더의 경로를 입력합니다.
   - 대상 디바이스에서 디바이스 유형을 선택합니다.

7. 지정한 출력 위치에서 컴파일된 모델을 다운로드한 다음 파일의 압축을 풉니다.
8. `synset.txt`를 컴파일된 모델 디렉터리에 복사합니다.
9. 컴파일된 모델 디렉터리의 이름을 `resnet18`로 변경합니다.

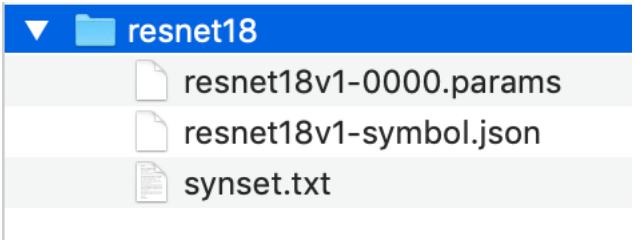
   컴파일된 모델 디렉터리는 디렉터리 구조가 다음과 같아야 합니다.

옵션 2: TVM을 사용하여 DLR 모델 컴파일

다음 단계에 따라 TVM을 사용하여 IDT에서 제공하는 ML 모델을 컴파일합니다. 이 모델은 Apache MXNet을 사용하여 사전 교육되어 있으므로 모델을 컴파일하는 컴퓨터나 디바이스에 MXNet을 설치해야 합니다. MXNet을 설치하려면 MXNet 설명서의 지침을 따르세요.

> **Note**
>
> 대상 디바이스에서 모델을 컴파일하는 것이 좋습니다. 이 방법은 선택 사항이지만 호환성을 보장하고 잠재적인 문제를 완화하는 데 도움이 될 수 있습니다.

1. DLR용 컴파일되지 않은 사전 교육된 MXNet 모델이 포함된 tarball 파일을 다운로드합니다.
   - dlr-noncompiled-model-1.0.tar.gz
2. tarball의 압축을 풉니다. 이 명령은 다음과 같은 디렉터리 구조를 생성합니다.



3. TVM 설명서의 지침에 따라 플랫폼에 대한 소스에서 TVM을 빌드하고 설치합니다.
4. TVM이 빌드된 후 resnet18 모델에 대한 TVM 컴파일을 실행합니다. 다음 단계는 TVM 설명서의 딥 러닝 모델 컴파일을 위한 빠른 시작 자습서를 기반으로 합니다.
   a. 복제된 TVM 리포지토리에서 `relay_quick_start.py` 파일을 엽니다.
   b. 릴레이의 신경망을 정의하는 코드를 업데이트합니다. 다음 옵션 중 하나를 사용할 수 있습니다.
      - 옵션 1: `mxnet.gluon.model_zoo.vision.get_model`을 사용하여 릴레이 모듈 및 파라미터를 가져옵니다.

```
from mxnet.gluon.model_zoo.vision import get_model
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

- 옵션 2: 1단계에서 다운로드한 컴파일되지 않은 모델에서 `relay_quick_start.py` 파일과 동일한 디렉터리에 다음 파일을 복사합니다. 이러한 파일에는 릴레이 모듈 및 파라미터가 포함되어 있습니다.
  - `resnet18v1-symbol.json`
  - `resnet18v1-0000.params`

c. 다음 코드를 사용하도록 컴파일된 모듈을 저장하고 로드하는 코드를 업데이트합니다.

```
from tvm.contrib import util
path_lib = "deploy_lib.so"
#  Export the model library based on your device architecture
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

d. 모델을 빌드합니다.

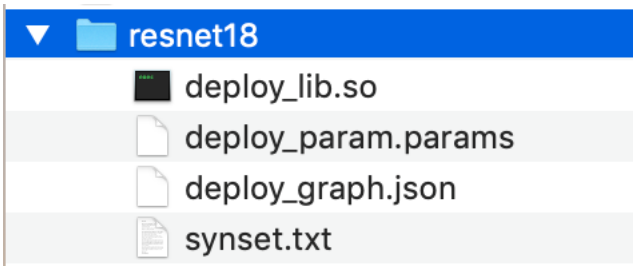
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

이 명령은 다음과 같은 파일을 생성합니다.

- `deploy_graph.json`
- `deploy_lib.so`
- `deploy_param.params`

5. 생성된 모델 파일을 `resnet18`이라는 디렉터리에 복사합니다. 이 디렉터리는 컴파일된 모델 디렉터리입니다.
6. 컴파일된 모델 디렉터리를 호스트 컴퓨터에 복사합니다. 그런 다음 1단계에서 다운로드한 컴파일되지 않은 모델에서 `synset.txt`를 컴파일된 모델 디렉터리에 복사합니다.

   컴파일된 모델 디렉터리는 디렉터리 구조가 다음과 같아야 합니다.



그런 다음 AWS 자격 증명 및 `device.json` 파일을 구성합니다.

AWS IoT Greengrass 검증 도구 모음을 실행하기 위한 IDT 설정 구성.

테스트를 실행하기 전에 호스트 컴퓨터에서 AWS 보안 인증 및 디바이스에 대한 설정을 구성해야 합니다.

AWS 보안 인증 구성

<device-tester-extract-location> `/configs/config.json` 파일에서 IAM 사용자 보안 인증을 구성해야 합니다. the section called “생성 및 구성 AWS 계정”에서 생성된 AWS IoT Greengrass용 IDT 사용자에 대해 자격 증명을 사용합니다. 두 가지 방법 중 하나로 자격 증명을 지정할 수 있습니다.

- 보안 인증 파일
- 환경 변수

보안 인증 파일을 사용하여 AWS 보안 인증 구성

IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 구성 및 자격 증명 파일을 참조하십시오.

자격 증명 파일의 위치는 사용하는 운영 체제에 따라 달라집니다.

- macOS, Linux의 경우: `~/.aws/credentials`
- Windows: `C:\Users\`*`UserName`*`\.aws\credentials`

`credentials` 파일에 AWS 보안 인증을 다음 형식으로 추가합니다.

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

AWS IoT Greengrass용 IDT에서 AWS 파일의 `credentials` 보안 인증을 사용하도록 구성하려면 `config.json` 파일을 다음과 같이 편집합니다.

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "file",
```

```
  "credentials": {
   "profile": "default"
  }
 }
}
```

> **Note**
>
> `default` AWS 프로필을 사용하지 않는 경우 `config.json` 파일에서 프로필 이름을 변경해야 합니다. 자세한 내용은 명명된 프로필을 참조하십시오.

환경 변수를 사용하여 AWS 보안 인증 구성

환경 변수는 운영 체제에서 유지 관리하고 시스템 명령에서 사용하는 변수입니다. 이들은 SSH 세션을 닫으면 저장되지 않습니다. AWS IoT Greengrass용 IDT는 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY` 환경 변수를 사용하여 AWS 보안 인증을 저장할 수 있습니다.

Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 export를 사용합니다.

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Windows에서 이러한 변수를 설정하려면 set을 사용합니다.

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

환경 변수를 사용하도록 IDT를 구성하려면 `config.json` 파일에서 `auth` 섹션을 편집합니다. 예:

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "environment"
 }
}
```

device.json 구성

AWS IoT Greengrass용 IDT에는 AWS 보안 인증 외에도 테스트가 실행되는 디바이스에 대한 정보가 필요합니다(예: IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처).

<device_tester_extract_location>/configs/device.json에 있는 device.json 템플릿을 사용하여 이 정보를 제공해야 합니다.

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "yes | no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
 **************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "container | process | both"
```

```
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*****************************************************************************************
   ],
   ********** Remove the section below if the device is not qualifying for HSI
***************
   "hsm": {
     "p11Provider": "/path/to/pkcs11ProviderLibrary",
     "slotLabel": "<slot_label>",
     "slotUserPin": "<slot_pin>",
     "privateKeyLabel": "<key_label>",
     "openSSLEngine": "/path/to/openssl/engine"
   },

*****************************************************************************************
   ********** Remove the section below if the device is not qualifying for ML
***************
   "machineLearning": {
     "dlrModelPath": "/path/to/compiled/dlr/model",
     "environmentVariables": [
       {
         "key": "<environment-variable-name>",
         "value": "<Path:$PATH>"
       }
     ],
     "deviceResources": [
       {
         "name": "<resource-name>",
         "path": "<resource-path>",
         "type": "device | volume"
       }
     ]
   },

*****************************************************************************************
   "kernelConfigLocation": "",
   "greengrassLocation": "",
   "devices": [
     {
       "id": "<device-id>",
```

```
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> `method`가 `pki`로 설정된 경우에만 `privKeyPath`를 지정합니다.
> `method`가 `password`로 설정된 경우에만 `password`를 지정합니다.

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
```

```
      },
      {
        "name": "docker",
        "value": "no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
**************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "process"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*********************************************************************************************
    ],
    ********** Remove the section below if the device is not qualifying for ML
***************
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ],
      "deviceResources": [
        {
          "name": "<resource-name>",
          "path": "<resource-path>",
```

```
          "type": "device | volume"
        }
      ]
    },

 ********************************************************************************************
    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "docker",
          "containerId": "<container-name | container-id>",
          "containerUser": "<user>"
        }
      }
    ]
  }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

디바이스 풀이라고 하는 디바이스 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 디바이스의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 디바이스가 사용됩니다.

`sku`

테스트 대상 디바이스를 고유하게 식별하는 영숫자 값입니다. SKU는 정규화된 보드를 추적하는 데 사용됩니다.

> **Note**
>
> AWS Partner 디바이스 카탈로그에 보드를 등록하려면 여기서 지정하는 SKU가 목록 등록 프로세스에 사용하는 SKU와 일치해야 합니다.

`features`

디바이스의 지원되는 기능이 포함된 배열입니다. 모든 기능이 필요합니다.

`os` 및 `arch`

지원되는 운영 체제(OS) 및 아키텍처 조합:

- `linux`, `x86_64`
- `linux`, `armv6l`
- `linux`, `armv7l`
- `linux`, `aarch64`
- `ubuntu`, `x86_64`
- `openwrt`, `armv7l`
- `openwrt`, `aarch64`

> **Note**
>
> IDT를 사용하여 Docker 컨테이너에서 실행되는 AWS IoT Greengrass을(를) 테스트하는 경우 x86_64 DockerDocker 아키텍처만 지원됩니다.

`container`

디바이스가 Greengrass 코어에서 컨테이너 모드로 Lambda 함수를 실행하기 위한 모든 소프트웨어 및 하드웨어 요구 사항을 충족하는지 여부를 검증합니다.

유효한 값은 `yes` 또는 `no`입니다.

`docker`

디바이스가 Greengrass Docker 애플리케이션 배포 커넥터를 사용하여 컨테이너를 실행하는 데 필요한 모든 기술 종속성을 충족하는지 검증합니다.

유효한 값은 `yes` 또는 `no`입니다.

`streamManagement`

디바이스가 AWS IoT Greengrass 스트림 관리자를 실행하는 데 필요한 모든 기술적 종속성을 충족하는지 검증합니다.

유효한 값은 yes 또는 no입니다.

`hsi`

제공된 HSI 공유 라이브러리가 하드웨어 보안 모듈(HSM)과 인터페이스할 수 있고 필요한 PKCS#11 API를 올바로 구현하는지 확인합니다. HSM 및 공유 라이브러리가 CSR에 서명하고 TLS 작업을 수행하며 올바른 키 길이와 퍼블릭 키 알고리즘을 제공할 수 있어야 합니다.

유효한 값은 yes 또는 no입니다.

`ml`

디바이스가 로컬로 ML 추론을 실행하는 데 필요한 모든 기술적 종속성을 충족하는지 검증합니다.

유효한 값은 `mxnet`, `tensorflow`, `dlr`, no(예: `mxnet`, `mxnet,tensorflow`, `mxnet,tensorflow,dlr`, no)의 모든 조합일 수 있습니다.

`mlLambdaContainerizationMode`

Greengrass 디바이스가 로컬로 ML 추론을 실행하는 데 필요한 모든 기술적 종속성을 충족하는지 검증합니다.

유효한 값은 `container`, `process`, 또는 `both`의 값입니다.

`processor`

디바이스가 지정된 프로세서 유형에 대한 모든 하드웨어 요구 사항을 충족하는지 확인합니다.

유효한 값은 `cpu` 또는 `gpu`입니다.

> **Note**
>
> `container`, `docker`, `streamManager`, `hsi`, `ml` 기능을 사용하지 않으려면 해당하는 `value`을(를) no(으)로 설정할 수 있습니다.
> Docker는 `streamManagement` 및 `ml`에 대한 기능 검증만 지원합니다.

`machineLearning`

선택 사항. ML 검증 테스트를 위한 구성 정보입니다. 자세한 내용은 the section called “ML 검증을 위해 device.json 구성” 섹션을 참조하세요.

`hsm`

선택 사항. AWS IoT Greengrass 하드웨어 보안 모듈(HSM)을 사용하여 테스트하기 위한 구성 정보입니다. 그렇지 않으면 `hsm` 속성을 생략해야 합니다. 자세한 내용은 하드웨어 보안 통합 섹션을 참조하세요.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`hsm.p11Provider`

PKCS#11 구현의 libdl 로드 가능 라이브러리에 대한 절대 경로입니다.

`hsm.slotLabel`

하드웨어 모듈을 식별하는 데 사용되는 슬롯 레이블입니다.

`hsm.slotUserPin`

모듈에 대해 AWS IoT Greengrass 코어를 인증하는 데 사용되는 사용자 PIN입니다.

`hsm.privateKeyLabel`

하드웨어 모듈에서 키를 식별하는 데 사용되는 레이블입니다.

`hsm.openSSLEngine`

OpenSSL에서 PKCS#11을 지원할 수 있게 해주는 OpenSSL 엔진 `.so` 파일의 절대 경로입니다. AWS IoT Greengrass OTA 업데이트 에이전트가 사용합니다.

`devices.id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`connectivity.protocol`

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 현재 지원되는 값은 `ssh`(물리적 디바이스의 경우) 및 `docker`(도커 컨테이너의 경우)입니다.

`connectivity.ip`

테스트 대상 디바이스의 IP 입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.containerId`

테스트 대상 Docker 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 connectivity.protocol이 docker로 설정된 경우에만 적용됩니다.

connectivity.auth

연결에 대한 인증 정보입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

connectivity.auth.method

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

connectivity.auth.credentials

인증에 사용되는 자격 증명입니다.

connectivity.auth.credentials.password

테스트 대상 디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

connectivity.auth.credentials.privKeyPath

테스트 대상 디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

connectivity.auth.credentials.user

테스트 대상 디바이스에 로그인하기 위한 사용자 이름입니다.

connectivity.auth.credentials.privKeyPath

테스트 대상 디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

connectivity.port

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 22입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

`greengrassLocation`

디바이스에서 AWS IoT Greengrass 코어 소프트웨어의 위치입니다.

물리적 디바이스의 경우 이 값은 AWS IoT Greengrass의 기존 설치를 사용할 때에만 사용됩니다. 이 속성을 사용하여 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전을 사용하도록 IDT에 알립니다.

AWS IoT Greengrass에서 제공하는 도커 이미지 또는 도커 파일을 통해 도커 컨테이너에서 테스트를 실행하는 경우 이 값을 `/greengrass`로 설정합니다.

`kernelConfigLocation`

선택 사항. 커널 구성 파일의 경로입니다. AWS IoT Device Tester는 이 파일을 사용하여 디바이스에서 필요한 커널 기능이 활성화되어 있는지 확인합니다. 지정되지 않은 경우 IDT는 다음 경로를 사용하여 커널 구성 파일을 검색합니다. `/proc/config.gz` 및 `/boot/config-`*`<kernel-version>`*. AWS IoT 디바이스 테스터는 검색된 첫 번째 경로를 사용합니다.

ML 검증을 위해 device.json 구성

이 단원에서는 ML 검증에 적용되는 디바이스 구성 파일의 선택적 속성에 대해 설명합니다. ML 검증에 대한 테스트를 실행하려면 사용 사례에 적용되는 속성을 정의해야 합니다.

`device-ml.json` 템플릿을 사용하여 디바이스의 구성 설정을 정의할 수 있습니다. 이 템플릿에는 선택적 ML 속성이 포함되어 있습니다. 또한 `device.json`을 사용하고 ML 검증 속성을 추가할 수 있습니다. 이러한 파일은 *`<device-tester-extract-location>`*`/configs`에 있으며 ML 검증 속성을 포함합니다. `device-ml.json`을 사용하는 경우 IDT 테스트를 실행하기 전에 파일 이름을 `device.json`으로 변경해야 합니다.

ML 검증에 적용되지 않는 디바이스 구성 속성에 대한 자세한 내용은 the section called “device.json 구성” 단원을 참조하십시오.

`features` 배열의 `ml`

보드에서 지원하는 ML 프레임워크입니다. 이 속성에는 IDT v3.1.0 이상이 필요합니다.

- 보드에서 하나의 프레임워크만 지원하는 경우 프레임워크를 지정하세요. 예:

```
{
```

```
    "name": "ml",
    "value": "mxnet"
}
```

- 보드에서 여러 프레임워크를 지원하는 경우 프레임워크를 쉼표로 구분된 목록으로 지정하세요. 예:

```
{
    "name": "ml",
    "value": "mxnet,tensorflow"
}
```

`features` 배열의 `mlLambdaContainerizationMode`

테스트하는 데 사용할 컨테이너화 모드입니다. 이 속성에는 IDT v3.1.0 이상이 필요합니다.

- 컨테이너화되지 않은 Lambda 함수로 ML 추론 코드를 실행하려면 `process`를 선택합니다. 이 옵션을 사용하려면 AWS IoT Greengrass v1.10.x 이상이 필요합니다.
- 컨테이너화된 Lambda 함수로 ML 추론 코드를 실행하려면 `container`를 선택합니다.
- 두 모드로 ML 추론 코드를 실행하려면 `both`를 선택합니다. 이 옵션을 사용하려면 AWS IoT Greengrass v1.10.x 이상이 필요합니다.

`features` 배열의 `processor`

보드에서 지원하는 하드웨어 액셀러레이터를 나타냅니다. 이 속성에는 IDT v3.1.0 이상이 필요합니다.

- 보드에서 CPU를 프로세서로 사용하는 경우 `cpu`를 선택합니다.
- 보드에서 GPU를 프로세서로 사용하는 경우 `gpu`를 선택합니다.

`machineLearning`

선택 사항. ML 검증 테스트를 위한 구성 정보입니다. 이 속성에는 IDT v3.1.0 이상이 필요합니다.

`dlrModelPath`

`dlr` 프레임워크를 사용하는 데 필요합니다. DLR 컴파일된 모델 디렉터리의 절대 경로로, 이름을 `resnet18`로 지정해야 합니다. 자세한 내용은 the section called “DLR 모델 컴파일” 섹션을 참조하세요.

> **Note**
>
> `/Users/<user>/Downloads/resnet18`은 macOS의 경로 예입니다.

`environmentVariables`

설정을 ML 추론 테스트에 동적으로 전달할 수 있는 키-값 페어입니다. CPU 디바이스의 경우 선택 사항입니다. 이 단원을 사용하여 디바이스 유형에 필요한 프레임워크별 환경 변수를 추가할 수 있습니다. 이러한 요구 사항에 대한 자세한 내용은 프레임워크 또는 디바이스의 공식 웹 사이트를 참조하십시오. 예를 들어, 일부 디바이스에서 MXNet 추론 테스트를 실행하려면 다음 환경 변수가 필요할 수 있습니다.

```
"environmentVariables": [
    ...
    {
        "key": "PYTHONPATH",
        "value": "$MXNET_HOME/python:$PYTHONPATH"
    },
    {
        "key": "MXNET_HOME",
        "value": "$HOME/mxnet/"
    },
    ...
]
```

> **Note**
>
> `value` 필드는 MXNet 설치에 따라 다를 수 있습니다.

GPU 디바이스에서 컨테이너화와 함께 실행되는 Lambda 함수를 테스트하는 경우 GPU 라이브러리용 환경 변수를 추가하십시오. 이렇게 하면 GPU가 계산을 수행할 수 있습니다. 다른 GPU 라이브러리를 사용하려면 라이브러리 또는 디바이스의 공식 설명서를 참조하십시오.

> **Note**
>
> `mlLambdaContainerizationMode` 기능이 `container` 또는 `both`로 설정된 경우 다음 키를 구성하세요.

```
"environmentVariables": [
    {
        "key": "PATH",
```

```
        "value": "<path/to/software/bin>:$PATH"
    },
    {
        "key": "LD_LIBRARY_PATH",
        "value": "<path/to/ld/lib>"
    },
    ...
]
```

`deviceResources`

GPU 디바이스에 필요합니다. Lambda 함수로 액세스할 수 있는 로컬 리소스를 포함합니다. 이 섹션을 사용하여 로컬 디바이스 및 볼륨 리소스를 추가합니다.

- 디바이스 리소스의 경우 `"type": "device"`를 지정합니다. GPU 디바이스의 경우 디바이스 리소스는 `/dev` 아래의 GPU 관련 디바이스 파일이어야 합니다.

  > **Note**
  >
  > `/dev/shm` 디렉터리는 예외입니다. 볼륨 리소스로만 구성할 수 있습니다.

- 볼륨 리소스의 경우 `"type": "volume"`을 지정합니다.

AWS IoT Greengrass 검증 제품군 실행

필수 구성을 설정한 후 테스트를 시작할 수 있습니다. 전체 테스트 제품군의 실행 시간은 하드웨어에 따라 다릅니다. 참조를 위해, Raspberry Pi 3B에서 전체 테스트 제품군을 완료하는 데 약 30분이 걸립니다.

다음 `run-suite` 명령 예제는 디바이스 풀에 대한 자격 테스트를 실행하는 방법을 보여 줍니다. 디바이스 풀은 동일한 디바이스의 집합입니다.

IDT v3.0.0 and later

지정된 테스트 제품군에 있는 모든 테스트 그룹을 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-
id <pool-id>
```

`list-suites` 명령을 사용하여 `tests` 폴더에 있는 테스트 제품군을 나열합니다.

테스트 제품군에서 특정 테스트 그룹을 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-
id <group-id> --pool-id <pool-id>
```

`list-groups` 명령을 사용하여 테스트 제품군의 테스트 그룹을 나열합니다.

테스트 그룹에서 특정 테스트 케이스를 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-
id <test-id>
```

테스트 그룹에서 여러 테스트 사례를 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-
id <test-id1>,<test-id2>
```

테스트 그룹의 테스트 사례를 나열합니다.

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

`run-suite` 명령에 대한 옵션은 선택 사항입니다. 예를 들어 `device.json` 파일에 하나의 디바이스 풀만 정의되어 있는 경우에는 `pool-id`를 생략할 수 있습니다. 또는 `tests` 폴더에서 최신 테스트 제품군 버전을 실행하려면 `suite-id`를 생략할 수 있습니다.

> **Note**
>
> 상위 테스트 제품군 버전이 온라인으로 제공되는 경우 IDT가 메시지를 표시합니다. 자세한 내용은 the section called “기본 업데이트 동작 설정” 섹션을 참조하세요.

`run-suite` 및 기타 IDT 명령에 대한 자세한 내용은 the section called “IDT 명령” 단원을 참조하십시오.

IDT v2.3.0 and earlier

지정된 제품군의 모든 테스트 그룹을 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-
id <pool-id>
```

특정 테스트 그룹을 실행합니다.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-
id <group-id> --pool-id <pool-id>
```

단일 디바이스 풀에서 단일 테스트 제품군을 실행 중인 경우 `suite-id` 및 `pool-id`는 선택 사항입니다. 즉, `device.json` 파일에 하나의 디바이스 풀만 정의되어 있습니다.

Greengrass 종속성 확인하기

관련 테스트 그룹을 실행하기 전에 종속성 확인 프로그램 테스트 그룹을 실행하여 모든 Greengrass 종속성이 설치되어 있는지 확인하는 것이 좋습니다. 예:

- 코어 자격 테스트 그룹을 실행하기 전에 `ggcdependencies`를 실행합니다.
- 컨테이너별 테스트 그룹을 실행하기 전에 `containerdependencies`를 실행하십시오.
- 도커별 테스트 그룹을 실행하기 전에 `dockerdependencies`를 실행하십시오.
- 스트림 관리자별 테스트 그룹을 실행하기 전에 `ggcstreammanagementdependencies`를 실행합니다.

기본 업데이트 동작 설정

테스트 실행을 시작하면 IDT가 최신 테스트 제품군 버전을 온라인으로 확인합니다. 사용 가능한 버전이 있으면 IDT가 사용 가능한 최신 버전으로 업데이트하라는 메시지를 표시합니다. `upgrade-test-suite`(또는 u) 플래그를 설정하여 기본 업데이트 동작을 제어할 수 있습니다. 유효한 값은 다음과 같습니다.

- y. IDT는 사용 가능한 최신 버전을 다운로드하고 사용합니다.
- n (default). IDT는 `suite-id` 옵션에 지정된 버전을 사용합니다. `suite-id`를 지정하지 않으면 IDT가 `tests` 폴더의 최신 버전을 사용합니다.

`upgrade-test-suite` 플래그를 포함하지 않으면 업데이트를 사용할 수 있을 때 IDT가 메시지를 표시하고 30초 동안 입력(y 또는 n)을 기다립니다. 입력이 되지 않으면 기본적으로 n으로 설정되고 테스트가 계속 실행됩니다.

다음 예는 이 기능의 일반적인 사용 사례를 보여줍니다.

테스트 그룹에 사용할 수 있는 최신 테스트를 자동으로 사용합니다.

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

특정 테스트 제품군 버전에서 테스트를 실행합니다.

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id
 DevicePool1
```

런타임에 업데이트하라는 메시지를 표시합니다.

```
devicetester_linux run-suite --pool-id DevicePool1
```

AWS IoT Greengrass용 IDT 명령

IDT 명령은 *<device-tester-extract-location>*/bin 디렉터리에 있습니다. 다음 작업에 사용합니다.

IDT v3.0.0 and later

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 테스트 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 테스트 제품군을 나열합니다.

`list-supported-products`

지원되는 제품(이 경우 AWS IoT Greengrass 버전)과 현재 IDT 버전에 대한 테스트 제품군 버전을 나열합니다.

`list-test-cases`

주어진 테스트 그룹의 테스트 케이스를 나열합니다. 다음 옵션이 지원됩니다.

- `group-id`. 검색할 테스트 그룹입니다. 이 옵션은 필수이며 단일 그룹을 지정해야 합니다.

`run-suite`

디바이스의 풀에 대해 테스트 제품군을 실행합니다. 지원되는 몇 가지 옵션은 다음과 같습니다.

- `suite-id`. 실행할 테스트 제품군 버전입니다. 지정하지 않으면 IDT는 `tests` 폴더의 최신 버전을 사용합니다.
- `group-id`. 실행할 테스트 그룹(쉼표로 구분된 목록). 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.
- `test-id`. 실행할 테스트 케이스(쉼표로 구분된 목록). 지정된 경우 `group-id`는 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 디바이스 풀. `device.json` 파일에 여러 디바이스 풀이 정의되어 있는 경우 하나의 풀을 지정해야 합니다.
- `upgrade-test-suite`. 테스트 제품군 버전 업데이트가 처리되는 방식을 제어합니다. IDT v3.0.0부터는 IDT가 업데이트된 테스트 제품군 버전을 온라인으로 확인합니다. 자세한 내용은 the section called “테스트 제품군 버전” 섹션을 참조하세요.
- `stop-on-first-failure`. 첫 번째 실패 시 실행을 중지하도록 IDT를 구성합니다. 이 옵션은 지정된 테스트 그룹을 디버깅하는 데 `group-id`와 함께 사용해야 합니다. 전체 테스트 제품군을 실행하여 검증 보고서를 생성할 때는 이 옵션을 사용하지 마시기 바랍니다.
- `update-idt`. IDT를 업데이트하라는 프롬프트에 대한 응답을 설정합니다. 입력이 `Y`일 경우 IDT가 최신 버전을 감지하면 테스트 실행이 중지됩니다. 입력이 `N`일 경우 테스트 실행이 계속됩니다.
- 입력이 `update-managed-policy`. `Y`일 경우 IDT가 사용자의 관리형 정책이 업데이트되지 않았음을 감지하면 테스트 실행이 중지됩니다. 입력이 `N`일 경우 테스트 실행이 계속됩니다.

`run-suite` 옵션에 대한 자세한 내용은 다음 `help` 옵션을 사용하십시오.

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v2.3.0 and earlier

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 테스트 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 테스트 제품군을 나열합니다.

`run-suite`

디바이스의 풀에 대해 테스트 제품군을 실행합니다.

`run-suite` 옵션에 대한 자세한 내용은 다음 `help` 옵션을 사용하십시오.

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

결과 및 로그 이해

이 단원에서는 IDT 결과 보고서 및 로그를 보고 해석하는 방법을 설명합니다.

결과 보기

실행하는 동안 IDT는 콘솔, 로그 파일 및 테스트 보고서에 오류를 작성합니다. IDT는 자격 테스트 제품군을 완료한 후 두 개의 테스트 보고서를 생성합니다. 이러한 보고서는 *<device-tester-extract-location>*`/results/`*<execution-id>*`/`에서 확인할 수 있습니다. 두 보고서 모두 검증 테스트 세트의 실행 결과를 캡처합니다.

`awsiotdevicetester_report.xml`은 AWS Partner Device Catalog에 디바이스를 등록하기 위해 AWS에 제출하는 자격 테스트 보고서입니다. 보고서에는 다음 요소가 포함됩니다.

- IDT 버전
- 테스트한 AWS IoT Greengrass 버전
- `device.json` 파일에 지정된 SKU 및 디바이스 풀 이름
- `device.json` 파일에 지정된 디바이스 풀의 기능
- 테스트 결과의 집계 요약
- 디바이스 기능(예: 로컬 리소스 액세스, 섀도우, MQTT 등)을 기반으로 테스트된 라이브러리별 테스트 결과의 분석

`GGQ_Result.xml` 보고서는 JUnit XML 형식입니다. Jenkins, Bamboo 등과 같은 지속적 통합 및 배포 플랫폼에 이 보고서를 통합할 수 있습니다. 보고서에는 다음 요소가 포함됩니다.

- 테스트 결과의 집계 요약
- 테스트한 AWS IoT Greengrass 기능별 테스트 결과의 분석

IDT 보고서 해석

awsiotdevicetester_report.xml 또는 awsiotdevicetester_report.xml의 보고서 섹션에는 실행된 테스트 및 결과가 나열됩니다.

첫 번째 XML 태그 <testsuites>에는 테스트 실행의 요약이 포함됩니다. 예:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 태그에 사용되는 속성

name

테스트 제품군의 이름입니다.

time

검증 세트를 실행하는 데 걸린 시간(초)

tests

실행된 테스트의 수입니다.

failures

실행되었지만 통과하지 못한 테스트의 수입니다.

errors

IDT에서 실행하지 못한 테스트의 수입니다.

disabled

이 속성은 사용되지 않으므로 무시해도 좋습니다.

awsiotdevicetester_report.xml 파일에는 테스트하는 제품에 대한 정보와 테스트 제품군을 실행한 후 확인된 제품 기능에 대한 정보를 포함하는 <awsproduct> 태그가 포함되어 있습니다.

<awsproduct> 태그에 사용되는 속성

name

테스트하는 제품의 이름입니다.

`version`

테스트하는 제품의 버전입니다.

`features`

확인된 기능입니다. `required`로 표시된 기능은 자격에 대한 보드를 제출하는 데 필요합니다. 다음 코드 조각은 `awsiotdevicetester_report.xml` 파일에 이 정보가 나타나는 방식을 보여 줍니다.

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></
feature>
```

`optional`로 표시된 기능은 자격에 필수 기능이 아닙니다. 다음 코드 조각은 선택적 기능을 보여 줍니다.

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></
feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></
feature>
```

필수 기능에 대한 테스트 실패 또는 오류가 없는 경우 디바이스는 AWS IoT Greengrass를 실행하기 위한 기술 요구 사항을 충족하며 AWS IoT 서비스와 상호 작용할 수 있습니다. AWS Partner Device Catalog에 디바이스를 나열하려는 경우 이 보고서를 자격 증거로 사용할 수 있습니다.

테스트 실패 또는 오류의 경우 <testsuites> XML 태그를 검토하여 실패한 테스트를 식별할 수 있습니다. <testsuites> 태그 내부의 <testsuite> XML 태그는 테스트 그룹에 대한 테스트 결과 요약을 보여 줍니다. 예:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

형식은 <testsuites> 태그와 비슷하지만, 사용되지 않고 무시할 수 있는 `skipped` 속성이 있습니다. 각 <testsuite> XML 태그 내부에는 테스트 그룹에 실행된 각 테스트에 대한 <testcase> 태그가 있습니다. 예:

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security
 Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled
```

```
 and following changes are made:Add CIS conn info and Add another CIS conn info"
 attempts="1"></testcase>>
```

<testcase> 태그에 사용되는 속성

name

테스트의 이름입니다.

attempts

IDT에서 테스트 사례를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

로그 보기

IDT는 *<devicetester-extract-location>*/results/*<execution-id>*/logs에서 테스트 실행 로그를 생성합니다. 두 개의 로그 세트가 생성됩니다.

test_manager.log

AWS IoT Device Tester의 Test Manager 구성 요소(예: 구성 관련 로그, 테스트 시퀀싱 및 보고서 생성)에서 생성된 로그입니다.

<test_case_id>.log (for example, ota.log)

테스트 대상 디바이스의 로그를 포함한 테스트 그룹의 로그입니다. 테스트가 실패하면 테스트에 대한 테스트 대상 디바이스의 로그가 포함된 tar.gz 파일이 생성됩니다(예: ota_prod_test_1_ggc_logs.tar.gz).

자세한 내용은 AWS IoT Greengrass용 IDT 문제 해결 섹션을 참조하세요.

IDT를 사용하여 자체 테스트 도구 모음을 개발하고 실행하십시오.

IDT v4.0.0부터 AWS IoT Greengrass용 IDT는 표준화된 구성 설정 및 결과 형식을 디바이스 및 디바이스 소프트웨어에 대한 사용자 지정 테스트 도구 모음을 개발할 수 있는 테스트 도구 모음 환경과 결합합니다. 자체 내부 검증을 위한 사용자 지정 테스트를 추가하거나 디바이스 검증을 위해 고객에게 제공할 수 있습니다.

IDT를 사용하여 다음과 같이 사용자 지정 테스트 도구 모음을 개발하고 실행하십시오.

사용자 지정 테스트 도구 모음을 개발하려면

- 테스트하려는 Greengrass 디바이스에 대해 사용자 지정 테스트 로직이 포함된 테스트 도구 모음을 만드세요.
- 테스트 러너에게 사용자 지정 테스트 도구 모음과 IDT를 제공하세요. 테스트 도구 모음의 특정 설정 구성에 대한 정보를 포함해야 합니다.

사용자 지정 테스트 도구 모음을 실행하려면

- 테스트하려는 디바이스를 설정합니다.
- 사용하려는 테스트 도구 모음의 필요에 따라 설정 구성을 구현하십시오.
- IDT를 사용하여 사용자 지정 테스트 도구 모음을 실행하세요.
- IDT에서 실행한 테스트의 테스트 결과 및 실행 로그를 확인하세요.

최신 버전의 AWS IoT Greengrass용 AWS IoT 디바이스 테스터 다운로드

최신 버전의 IDT를 다운로드하고 읽기 및 쓰기 권한이 있는 파일 시스템의 위치에 소프트웨어의 압축을 폽니다.

> **Note**
>
> 여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하는 것은 지원되지 않습니다. 로컬 드라이브에 IDT 패키지의 압축을 풀고 로컬 워크스테이션에서 IDT 바이너리를 실행하는 것이 좋습니다.
> Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용 중인 경우 경로를 260자 제한 아래로 유지하도록 IDT 압축을 `C:\` 또는 `D:\` 같은 루트 디렉터리에 폽니다.

테스트 도구 모음 생성 워크플로

테스트 도구 모음은 세 가지 유형의 파일로 구성됩니다.

- 테스트 도구 모음 실행 방법에 대한 정보를 IDT에 제공하는 JSON 구성 파일.
- IDT가 테스트 사례를 실행하는 데 사용하는 테스트 실행 파일.
- 테스트를 실행하는 데 필요한 추가 파일.

다음 기본 단계를 완료하여 사용자 지정 IDT 테스트를 생성하세요.

1. 테스트 도구 모음의 JSON 구성 파일을 만드세요.
2. 테스트 도구 모음의 테스트 로직이 포함된 테스트 사례 실행 파일을 만드세요.
3. 테스트 도구 모음을 실행하는 데 테스트 러너에게 필요한 구성 정보를 확인하고 문서화하세요.
4. IDT가 예상대로 테스트 도구 모음을 실행하고 테스트 결과를 생성할 수 있는지 확인하십시오.

샘플 사용자 지정 도구 모음을 빠르게 구축하고 실행하려면 튜토리얼: 샘플 IDT 테스트 도구 모음 구축 및 실행의 지침을 따르십시오.

Python으로 사용자 지정 테스트 도구 모음 생성을 시작하려면 튜토리얼: 간단한 IDT 테스트 도구 모음 개발을(를) 참조하십시오.

튜토리얼: 샘플 IDT 테스트 도구 모음 구축 및 실행

AWS IoT Device Tester 다운로드에는 샘플 테스트 도구 모음의 소스 코드가 포함되어 있습니다. 이 튜토리얼을 완료하여 샘플 테스트 도구 모음을 구축하고 실행하여 AWS IoT Greengrass용 AWS IoT Device Tester를 사용하여 사용자 지정 테스트 도구 모음을 실행하는 방법을 이해할 수 있습니다.

이 자습서에서는 다음 단계를 완료합니다.

1. 샘플 테스트 도구 모음 구축
2. IDT를 사용하여 샘플 테스트 도구 모음을 실행하세요.

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - 최신 버전의 AWS IoT용 디바이스 테스터 다운로드
  - Python 3.7 이상

    컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

    ```
    python3 --version
    ```

    Windows에서 이 명령 사용시 오류가 반환되면 `python --version`을(를) 대신 사용하십시오. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을(를) 설정합니다.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    버전 정보가 반환되지 않았거나 버전 번호가 3.7 미만이면 Python 다운로드의 지침에 따라 Python 3.7 이상을 설치합니다. 자세한 내용은 Python 설명서를 참조하세요.
  - urllib3

    `urllib3`이 제대로 설치되었는지 확인하려면 다음 명령을 실행합니다.

    ```
    python3 -c 'import urllib3'
    ```

    `urllib3`가 설치되지 않은 경우에는 다음 명령을 실행하여 설치합니다.

    ```
    python3 -m pip install urllib3
    ```
- 디바이스 요구 사항
  - Linux 운영 체제를 사용하고 호스트 컴퓨터와 동일한 네트워크에 네트워크로 연결된 디바이스.

    Raspberry Pi OS와 함께 Raspberry Pi를 사용하는 것이 좋습니다. Raspberry Pi에 원격으로 연결하려면 Pi에서 SSH를 설정해야 합니다.

IDT용 디바이스 정보 구성

IDT가 테스트를 실행할 수 있도록 디바이스 정보를 구성하십시오. *<device-tester-extract-location>*`/configs` 폴더에 있는 `device.json` 템플릿을 다음 정보로 업데이트해야 합니다.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

`devices` 개체에 다음 정보를 제공하시기 바랍니다.

`id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`connectivity.ip`

디바이스의 IP 주소입니다.

`connectivity.port`

선택 사항. 디바이스에 SSH 연결에 사용할 포트 번호입니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.user`

디바이스에 로그인하는 데 사용되는 사용자 이름.

`connectivity.auth.credentials.privKeyPath`

디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`devices.connectivity.auth.credentials.password`

디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

> **Note**
>
> `method`가 `pki`로 설정된 경우에만 `privKeyPath`를 지정합니다.
> `method`가 `password`로 설정된 경우에만 `password`를 지정합니다.

샘플 테스트 도구 모음 구축

`<device-tester-extract-location>``/samples/python` 폴더에는 제공된 구축 스크립트를 사용하여 테스트 도구 모음에 결합할 수 있는 샘플 구성 파일, 소스 코드 및 IDT Client SDK가 포함되어 있습니다. 다음 디렉터리 트리는 이러한 샘플 파일의 위치를 보여줍니다.

```
<device-tester-extract-location>
```

```
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

테스트 도구 모음을 구축하려면 호스트 컴퓨터에서 다음 명령을 실행합니다.

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

그러면 `IDTSampleSuitePython_1.0.0` 폴더 내 *<device-tester-extract-location>*`/tests` 폴더에 샘플 테스트 도구 모음이 만들어집니다. `IDTSampleSuitePython_1.0.0` 폴더의 파일을 검토하여 샘플 테스트 도구 모음이 어떻게 구성되어 있는지 이해하고 테스트 사례 실행 파일 및 테스트 구성 JSON 파일의 다양한 예제를 확인하세요.

다음 단계: IDT를 사용하여 생성한 샘플 테스트 도구 모음을 실행하십시오.

IDT를 사용하여 샘플 테스트 도구 모음을 실행하세요.

샘플 테스트 도구 모음을 실행하려면 호스트 컴퓨터에서 다음 명령을 실행하세요.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT는 샘플 테스트 도구 모음을 실행하고 결과를 콘솔로 스트리밍합니다. 테스트 실행이 완료되면 다음 정보가 표시됩니다.

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    sample_group:       PASSED
----------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

문제 해결

다음 정보를 사용하면 튜토리얼 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않습니다

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 이 튜토리얼의 모든 사전 요구 사항을 충족하는지 확인하세요.

테스트 중인 디바이스에 연결할 수 없습니다.

다음을 확인합니다.

- `device.json` 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 디바이스에 연결할 수 있습니다.

튜토리얼: 간단한 IDT 테스트 도구 모음 개발

테스트 도구 모음은 다음을 결합합니다.

- 테스트 로직이 포함된 테스트 실행 파일

- 테스트 도구 모음을 설명하는 JSON 구성 파일

이 튜토리얼에서는 AWS IoT Greengrass용 IDT를 사용하여 단일 테스트 사례가 포함된 Python 테스트 도구 모음을 개발하는 방법을 보여줍니다. 이 자습서에서는 다음 단계를 완료합니다.

1. 테스트 도구 모음 디렉터리 생성
2. JSON 구성 파일 생성
3. 테스트 사례 실행 파일 생성
4. 테스트 도구 모음 실행

필수 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - 최신 버전의 AWS IoT용 디바이스 테스터 다운로드
  - Python 3.7 이상

    컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

    ```
    python3 --version
    ```

    Windows에서 이 명령 사용시 오류가 반환되면 `python --version`을(를) 대신 사용하십시오. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을(를) 설정합니다.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    버전 정보가 반환되지 않았거나 버전 번호가 3.7 미만이면 Python 다운로드의 지침에 따라 Python 3.7 이상을 설치합니다. 자세한 내용은 Python 설명서를 참조하세요.
  - urllib3

    `urllib3`이 제대로 설치되었는지 확인하려면 다음 명령을 실행합니다.

    ```
    python3 -c 'import urllib3'
    ```

`urllib3`가 설치되지 않은 경우에는 다음 명령을 실행하여 설치합니다.

```
python3 -m pip install urllib3
```

- 디바이스 요구 사항
  - Linux 운영 체제를 사용하고 호스트 컴퓨터와 동일한 네트워크에 네트워크로 연결된 디바이스.

    Raspberry Pi OS와 함께 Raspberry Pi를 사용하는 것이 좋습니다. Raspberry Pi에 원격으로 연결하려면 Pi에서 SSH를 설정해야 합니다.

테스트 도구 모음 디렉터리 생성

IDT는 각 테스트 도구 모음 내의 테스트 그룹에 테스트 사례를 논리적으로 분리합니다. 각 테스트 사례는 테스트 그룹 내에 있어야 합니다. 이 튜토리얼을 위해 `MyTestSuite_1.0.0`(이)라는 폴더를 만들고 이 폴더 내에 다음 디렉터리 트리를 생성하십시오.

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

JSON 구성 파일 생성

테스트 도구 모음에는 다음과 같은 필수 JSON 구성 파일이 포함되어야 합니다.

필수 JSON 파일

`suite.json`

테스트 제품군 정보가 포함되어 있습니다. suite.json을 구성하십시오. 섹션을 참조하세요.

`group.json`

테스트 그룹에 대한 정보를 포함합니다. 테스트 도구 모음의 각 테스트 그룹에 대한 `group.json` 파일을 만들어야 합니다. group.json을 구성하십시오. 섹션을 참조하세요.

`test.json`

테스트 케이스에 대한 정보가 들어 있습니다. 테스트 도구 모음의 각 테스트 사례에 대한 `test.json` 파일을 만들어야 합니다. test.json을 구성하십시오. 섹션을 참조하세요.

1. MyTestSuite_1.0.0/suite 폴더에서 다음 폴더 구조로 suite.json 파일을 안에 생성합니다.

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. MyTestSuite_1.0.0/myTestGroup 폴더에서 다음 폴더 구조로 group.json 파일을 안에 생성합니다.

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. MyTestSuite_1.0.0/myTestGroup/myTestCase 폴더에서 다음 폴더 구조로 test.json 파일을 안에 생성합니다.

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
```

```
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
}
```

이제 MyTestSuite_1.0.0 폴더의 디렉터리 트리가 다음과 같이 표시되어야 합니다.

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

IDT 클라이언트 SDK 다운로드

IDT 클라이언트 SDK를 사용하여 IDT가 테스트 대상 디바이스와 상호 작용하고 테스트 결과를 보고할 수 있도록 합니다. 이 튜토리얼에서는 Python 버전의 SDK를 사용합니다.

<device-tester-extract-location>/sdks/python/ 폴더에서 idt_client 폴더를 MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 폴더로 복사합니다.

SDK가 성공적으로 복사되었는지 확인하려면 다음 명령을 실행합니다.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

테스트 사례 실행 파일 생성

테스트 사례 실행 파일에는 실행하려는 테스트 로직이 포함되어 있습니다. 테스트 도구 모음에는 여러 테스트 사례 실행 파일이 포함될 수 있습니다. 이 튜토리얼에서는 하나의 테스트 사례 실행 파일을 생성합니다.

1. 테스트 도구 모음 파일을 만드세요.

   MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 폴더 안에 다음 내용으로 myTestCase.py라는 파일을 만듭니다.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

if __name__ == "__main__":
    main()
```

2. 클라이언트 SDK 함수를 사용하여 myTestCase.py 파일에 다음 테스트 로직을 추가합니다.

   a. 테스트 중인 디바이스에서 SSH 명령어를 실행합니다.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

   b. 테스트 결과를 IDT로 전송합니다.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
```

```
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

IDT용 디바이스 정보 구성

IDT가 테스트를 실행할 수 있도록 디바이스 정보를 구성하십시오. *<device-tester-extract-location>*/configs 폴더에 있는 device.json 템플릿을 다음 정보로 업데이트해야 합니다.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
```

```
        }
      }
    ]
  }
]
```

devices 개체에 다음 정보를 제공하시기 바랍니다.

id

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

connectivity.ip

디바이스의 IP 주소입니다.

connectivity.port

선택 사항. 디바이스에 SSH 연결에 사용할 포트 번호입니다.

connectivity.auth

연결에 대한 인증 정보입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

connectivity.auth.method

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

connectivity.auth.credentials

인증에 사용되는 자격 증명입니다.

connectivity.auth.credentials.user

디바이스에 로그인하는 데 사용되는 사용자 이름.

connectivity.auth.credentials.privKeyPath

디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

devices.connectivity.auth.credentials.password

디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

> **Note**
>
> method가 pki로 설정된 경우에만 privKeyPath를 지정합니다.
> method가 password로 설정된 경우에만 password를 지정합니다.

테스트 도구 모음 실행

테스트 도구 모음을 만든 후에는 예상대로 작동하는지 확인해야 합니다. 이를 위해 기존 디바이스 풀로 테스트 도구 모음을 실행하려면 다음 단계를 완료하세요.

1. MyTestSuite_1.0.0 폴더를 *<device-tester-extract-location>*/tests에 복사하세요.
2. 다음 명령을 실행합니다:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT는 테스트 도구 모음을 실행하고 결과를 콘솔로 스트리밍합니다. 테스트 실행이 완료되면 다음 정보가 표시됩니다.

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=
```

```
========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    myTestGroup:        PASSED
----------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

문제 해결

다음 정보를 사용하면 튜토리얼 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않습니다

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 오류 로그를 확인하기 전에 다음 사항을 확인합니다.

- IDT 클라이언트 SDK는 이 단계에서 설명한 대로 올바른 폴더에 있습니다.
- 이 튜토리얼의 모든 사전 요구 사항을 충족합니다.

테스트 중인 디바이스에 연결할 수 없습니다.

다음을 확인합니다.

- `device.json` 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 디바이스에 연결할 수 있습니다.

IDT 테스트 도구 모음 구성 파일 만들기

이 섹션에서는 사용자 지정 테스트 도구 모음을 작성할 때 포함하는 JSON 구성 파일을 만드는 형식을 설명합니다.

필수 JSON 파일

`suite.json`

테스트 제품군 정보가 포함되어 있습니다. suite.json을 구성하십시오. 섹션을 참조하세요.

`group.json`

테스트 그룹에 대한 정보를 포함합니다. 테스트 도구 모음의 각 테스트 그룹에 대한 `group.json` 파일을 만들어야 합니다. group.json을 구성하십시오. 섹션을 참조하세요.

`test.json`

테스트 케이스에 대한 정보가 들어 있습니다. 테스트 도구 모음의 각 테스트 케이스에 대한 `test.json` 파일을 만들어야 합니다. test.json을 구성하십시오. 섹션을 참조하세요.

JSON 파일(선택 사항)

`state_machine.json`

IDT가 테스트 도구 모음을 실행할 때 테스트가 실행되는 방법을 정의합니다. state_machine.json을 구성합니다. 섹션을 참조하세요.

`userdata_schema.json`

테스트 실행기가 설정 구성에 포함할 수 있는 `userdata.json` 파일의 스키마를 정의합니다. 이 `userdata.json` 파일은 테스트를 실행하는 데 필요하지만 `device.json` 파일에는 없는 추가 구성 정보에 사용됩니다. userdata_schema.json을 구성합니다. 섹션을 참조하세요.

JSON 구성 파일은 다음과 같이 *<custom-test-suite-folder>* 파일에 저장됩니다.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### state_machine.json
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

suite.json을 구성하십시오.

`suite.json` 파일은 환경 변수를 설정하고 테스트 도구 모음을 실행하는 데 사용자 데이터가 필요한지 여부를 결정합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/`suite.json` 파일을 구성하십시오.

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 도구 모음의 고유한 사용자 정의 ID. `id`의 값은 `suite.json` 파일이 있는 테스트 도구 모음 폴더의 이름과 일치해야 합니다. 세트 이름과 세트 버전도 다음 요구 사항을 충족해야 합니다.

- *<suite-name>*은(는) 언더바를 포함할 수 없습니다.
- *<suite-version>*은(는) 다음과 같이 *x.x.x*(으)로 표시됩니다. 여기서 x은(는) 숫자입니다.

ID는 IDT에서 생성한 테스트 보고서에 표시됩니다.

`title`

이 테스트 도구 모음에서 테스트 중인 제품 또는 기능의 사용자 정의 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 도구 모음의 용도에 대한 짧은 설명입니다.

`userDataRequired`

테스트 실행기가 `userdata.json` 파일에 사용자 지정 정보를 포함해야 하는지 여부를 정의합니다. 이 값을 `true`으(로) 설정하는 경우, 테스트 도구 모음 폴더에도 `userdata_schema.json` 파일을 포함해야 합니다.

`environmentVariables`

선택 사항. 이 테스트 도구 모음에 설정할 환경 변수 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

group.json을 구성하십시오.

`group.json` 파일은 테스트 그룹이 필수인지 옵션인지 여부를 정의합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/*<test-group>*/group.json 파일을 구성하십시오.

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 그룹에 대한 고유한 사용자 정의 ID. `id`의 값은 `group.json` 파일이 있는 테스트 그룹 폴더의 이름과 일치해야 하며 언더바(_)를 포함할 수 없습니다. ID는 IDT에서 생성한 테스트 보고서에 사용됩니다.

`title`

테스트 그룹을 나타내는 서술형 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 그룹의 용도에 대한 짧은 설명입니다.

`optional`

선택 사항. IDT에서 필수 테스트 실행을 완료한 후 이 테스트 그룹을 선택적 그룹으로 표시하도록 `true`(으)로 설정합니다. 기본값은 `false`입니다.

test.json을 구성하십시오.

`test.json` 파일은 테스트 케이스 실행 파일 및 테스트 케이스에서 사용되는 환경 변수를 결정합니다. 테스트 케이스 실행 파일 생성에 대한 자세한 내용은 IDT 테스트 케이스 실행 파일 생성 단원을 참조하세요.

다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json 파일을 구성하십시오.

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
                }
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
```

```
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 케이스의 고유한 사용자 정의 ID. `id`의 값은 `test.json` 파일이 있는 테스트 케이스 폴더의 이름과 일치해야 하며 언더바(_)를 포함할 수 없습니다. ID는 IDT에서 생성한 테스트 보고서에 사용됩니다.

`title`

테스트 케이스를 나타내는 서술형 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 케이스의 용도에 대한 짧은 설명입니다.

`requireDUT`

선택 사항. 이 테스트를 실행하는 데 기기가 필요한 경우, `true`(으)로 설정하고, 그렇지 않으면 `false`(으)로 설정합니다. 기본값은 `true`입니다. 테스트 실행기는 `device.json` 파일에서 테스트를 실행하는 데 사용할 기기를 구성합니다.

`requiredResources`

선택 사항. 이 테스트를 실행하는 데 필요한 리소스 기기에 대한 정보를 제공하는 배열입니다.

`requiredResources.name`

이 테스트를 실행할 때 리소스 기기에 부여하는 고유한 이름입니다.

`requiredResources.features`

사용자 정의 리소스 디바이스 기능의 배열.

`requiredResources.features.name`

기능의 이름입니다. 이 디바이스를 사용하려는 디바이스 기능. 이 이름은 테스트 실행기가 `resource.json` 파일에 제공한 기능 이름과 일치합니다.

`requiredResources.features.version`

선택 사항. 기능의 버전. 이 값은 테스트 실행기가 `resource.json` 파일에서 제공한 기능 버전과 일치합니다. 버전이 제공되지 않으면 기능이 확인되지 않습니다. 기능에 버전 번호가 필요하지 않은 경우, 이 필드를 비워 둡니다.

`requiredResources.features.jobSlots`

선택 사항. 이 기능이 지원할 수 있는 동시 테스트 수입니다. 기본값은 1입니다. IDT에서 개별 기능에 대해 서로 다른 기기를 사용하도록 하려면 이 값을 1(으)로 설정하는 것이 좋습니다.

`execution.timeout`

IDT가 테스트 실행이 완료될 때까지 기다리는 시간(밀리초) 입니다. 이 값 설정에 대한 자세한 내용을 알아보려면 IDT 테스트 케이스 실행 파일 생성 섹션을 참조하세요.

`execution.os`

IDT를 실행하는 호스트 컴퓨터의 운영 체제에 따라 실행할 테스트 케이스 실행 파일입니다. 지원되는 값은 `linux`, `mac` 및 `win`입니다.

`execution.os.cmd`

지정된 운영 체제에서 실행하려는 테스트 케이스 실행 파일의 경로입니다. 이 위치는 시스템 경로에 있어야 합니다.

`execution.os.args`

선택 사항. 테스트 케이스 실행 파일을 실행하기 위해 제공할 인수입니다.

`environmentVariables`

선택 사항. 이 테스트 케이스에 설정된 환경 변수 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

> **Note**
>
> `test.json` 파일과 `suite.json` 파일에서 동일한 환경 변수를 지정하는 경우, `test.json` 파일의 값이 우선합니다.

state_machine.json을 구성합니다.

상태 머신은 테스트 제품군 실행 흐름을 제어하는 구조입니다. 테스트 도구 모음의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 최종 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 제품군에 사용자 정의 상태 머신이 포함되지 않은 경우 IDT가 상태 머신을 자동으로 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 실행기에 전체 테스트 제품군 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않은 경우, 테스트 제품군의 모든 테스트 그룹을 무작위 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례의 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 상태 머신이 작동하는 자세한 방법은 IDT 상태 머신 구성을(를) 참조하십시오.

userdata_schema.json을 구성합니다.

`userdata_schema.json` 파일은 테스트 실행기가 사용자 데이터를 제공하는 스키마를 결정합니다. 테스트 도구 모음에 `device.json` 파일에 없는 정보가 필요한 경우, 사용자 데이터가 필요합니다. 예를 들어 테스트에는 Wi-Fi 네트워크 자격 증명, 특정 오픈 포트 또는 사용자가 제공해야 하는 인증서가 필요할 수 있습니다. 이 정보는 `userdata`(이)라는 입력 파라미터로 IDT에 제공될 수 있습니다. 이때

값은 *<device-tester-extract-location>*/config 폴더에 생성한 userdata.json 파일입니다. userdata.json 파일 형식은 테스트 도구 모음에 포함된 userdata_schema.json 파일을 기반으로 합니다.

테스트 실행기가 userdata.json 파일을 제공해야 함을 나타내려면:

1. suite.json 파일에서 userDataRequired을(를) true(으)로 설정합니다.
2. *<custom-test-suite-folder>*에서 userdata_schema.json 파일을 생성하십시오.
3. userdata_schema.json 파일을 편집하여 유효한 IETF 초안 v4 JSON 스키마를 생성하십시오.

IDT는 테스트 도구 모음을 실행하면 자동으로 스키마를 읽고 이를 사용하여 테스트 실행기가 제공한 userdata.json 파일을 검증합니다. 유효한 경우, userdata.json 파일의 내용은 IDT 컨텍스트와 스테이트 머신 컨텍스트 모두에서 사용할 수 있습니다.

IDT 상태 머신 구성

상태 머신은 테스트 제품군 실행 흐름을 제어하는 구조입니다. 테스트 도구 모음의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 최종 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 제품군에 사용자 정의 상태 머신이 포함되지 않은 경우 IDT가 상태 머신을 자동으로 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 실행기에 전체 테스트 제품군 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않은 경우, 테스트 제품군의 모든 테스트 그룹을 무작위 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례의 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 테스트 제품군의 상태 머신은 다음 기준을 충족해야 합니다.

- 각 상태는 테스트 그룹 실행 또는 보고서 파일 생성과 같이 IDT가 취할 수 있는 작업에 해당합니다.
- 상태로 전환하면 상태와 관련된 작업이 실행됩니다.
- 각 상태는 다음 상태의 전환 규칙을 정의합니다.
- 종료 상태는 Succeed 또는 Fail 중 하나여야 합니다.

상태 머신 형식

다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/state_machine.json 파일을 직접 구성할 수 있습니다.

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Comment`

상태 머신의 설명입니다.

`StartAt`

IDT가 테스트 제품군을 실행하기 시작하는 상태의 이름. `StartAt`의 값은 `States` 객체에 나열된 상태 중 하나로 설정해야 합니다.

`States`

사용자 정의 상태 이름을 유효한 IDT 상태에 매핑하는 객체입니다. 각 상태.*state-name* 객체에는 상태 *state-name*에 매핑된 유효한 상태의 정의가 포함됩니다.

`States` 객체에는 `Succeed` 및 `Fail` 상태가 포함되어야 합니다. 유효한 상태에 대한 자세한 내용은 유효한 상태 및 상태 정의 섹션을 참조하세요.

유효한 상태 및 상태 정의

이 섹션에서는 IDT 상태 머신에서 사용할 수 있는 모든 유효한 상태의 상태 정의를 설명합니다. 다음 상태 중 일부는 테스트 케이스 수준의 구성을 지원합니다. 하지만 꼭 필요한 경우가 아니면 테스트 케이스 수준 대신 테스트 그룹 수준에서 상태 전환 규칙을 구성하는 것이 좋습니다.

상태 정의

- RunTask
- Choice
- Parallel
- AddProductFeatures
- 보고서
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

RunTask 상태는 테스트 제품군에 정의된 테스트 그룹에서 테스트 케이스를 실행합니다.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`TestGroup`

선택 사항. 실행할 테스트 그룹의 ID. 이 값을 지정하지 않으면 IDT는 테스트 실행기가 선택한 테스트 그룹을 실행합니다.

`TestCases`

선택 사항. `TestGroup`에서 지정한 그룹의 테스트 케이스 ID 배열. IDT는 `TestGroup` 및 `TestCases` 값을 기반으로 다음과 같이 테스트 실행 동작을 결정합니다.

- `TestGroup`과 `TestCases`가 모두 지정된 경우 IDT는 테스트 그룹에서 지정된 테스트 케이스를 실행합니다.
- `TestCases`가 지정되었지만 `TestGroup`이 지정되지 않은 경우 IDT는 지정된 테스트 케이스를 실행합니다.
- `TestGroup`이 지정되었지만 `TestCases`가 지정되지 않은 경우 IDT는 지정된 테스트 그룹 내의 모든 테스트 케이스를 실행합니다.
- `TestGroup` 또는 `TestCases` 둘 다 지정되지 않은 경우 IDT는 테스트 실행기가 IDT CLI에서 선택한 테스트 그룹에서 모든 테스트 케이스를 실행합니다. 테스트 실행기에 대한 그룹 선택을 활성화하려면 `statemachine.json` 파일에 `RunTask` 및 `Choice` 상태를 모두 포함해야 합니다. 작동 방식에 대한 예제는 상태 시스템 예시: 사용자가 선택한 테스트 그룹 실행을 참조하십시오.

  테스트 실행기의 IDT CLI 명령 활성화에 대한 자세한 내용은 the section called “IDT CLI 명령을 활성화합니다.” 섹션을 참조하세요.

`ResultVar`

테스트 실행 결과와 함께 설정할 컨텍스트 변수의 이름. `TestGroup`에 대한 값을 지정하지 않은 경우 이 값을 지정하지 마십시오. IDT는 다음에 따라 `ResultVar`에서 `true` 또는 `false`로 정의한 변수 값을 설정합니다.

- 변수 이름의 *`text`*`_`*`text`*`_passed` 형식인 경우 값은 첫 번째 테스트 그룹의 모든 테스트를 통과했는지 아니면 건너뛰었는지로 설정됩니다.
- 다른 모든 경우에는 값이 모든 테스트 그룹의 모든 테스트를 통과했는지 아니면 건너뛰었는지로 설정됩니다.

일반적으로 `RunTask` 상태를 사용하여 개별 테스트 케이스 ID를 지정하지 않고 테스트 그룹 ID를 지정하면 IDT가 지정된 테스트 그룹의 모든 테스트 케이스를 실행하게 됩니다. 이 상태에서 실행되는 모든 테스트 케이스는 무작위 순서로 병렬로 실행됩니다. 그러나 모든 테스트 케이스를 실행하는 데 디바이스가 필요하고 사용 가능한 디바이스가 하나뿐인 경우에는 테스트 케이스가 대신 순차적으로 실행됩니다.

오류 처리

지정된 테스트 그룹 또는 테스트 케이스 ID가 유효하지 않은 경우 이 상태에서는 RunTaskError 실행 오류가 발생합니다. 상태에서 실행 오류가 발생하는 경우 상태 머신 컨텍스트의 hasExecutionError 변수도 true로 설정됩니다.

Choice

Choice 상태를 사용하면 사용자 정의 조건에 따라 전환할 다음 상태를 동적으로 설정할 수 있습니다.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Default

Choices에 정의된 표현식을 true로 평가할 수 없는 경우 전환할 기본 상태입니다.

FallthroughOnError

선택 사항. 상태에서 표현식을 평가할 때 오류가 발생할 경우의 동작을 지정합니다. 평가 결과 오류가 발생할 경우 표현식을 건너뛰려면 true로 설정합니다. 일치하는 표현식이 없는 경우 상태 머신은 해당 Default 상태로 전환됩니다. false 값이 지정하지 않은 경우 기본값은 FallthroughOnError입니다.

Choices

현재 상태에서 동작을 실행한 후 전환할 상태를 결정하는 표현식 및 상태의 배열입니다.

Choices.Expression

부울 값으로 평가되어야 하는 표현식 문자열입니다. 표현식이 true로 평가되면 상태 머신은 Choices.Next에 정의된 상태로 전환됩니다. 표현식 문자열은 상태 시스템 컨텍스트에서 값을

검색한 다음 해당 값에 대한 연산을 수행하여 부울 값에 도달합니다. 상태 머신 컨텍스트 액세스에 대한 자세한 내용은 상태 머신 컨텍스트 섹션을 참조하세요.

`Choices.Next`

`Choices.Expression`에 정의된 표현식이 `true`로 평가되면 전환할 상태의 이름.

오류 처리

`Choice` 상태는 다음과 같은 경우에 오류 처리를 요구할 수 있습니다.

- 선택 표현식의 일부 변수는 상태 머신 컨텍스트에 존재하지 않습니다.
- 표현식의 결과는 부울 값이 아닙니다.
- JSON 조회 결과는 문자열, 숫자 또는 부울이 아닙니다.

이 상태에서는 `Catch` 블록을 사용하여 오류를 처리할 수 없습니다. 오류가 발생했을 때 상태 머신 실행을 중지하려면 `FallthroughOnError`를 `false`로 설정해야 합니다. 하지만 사용 사례에 따라 다음 중 하나를 수행하도록 `FallthroughOnError`를 `true`로 설정하는 것이 좋습니다.

- 액세스하는 변수가 존재하지 않을 것으로 예상되는 경우가 있다면, `Default`의 값과 추가 `Choices` 블록을 사용하여 다음 상태를 지정하십시오.
- 액세스 중인 변수가 항상 존재해야 하는 경우 `Default` 상태를 `Fail`로 설정하십시오.

Parallel

`Parallel` 상태를 사용하면 새 상태 머신을 서로 병렬로 정의하고 실행할 수 있습니다.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`Branches`

실행할 상태 머신 정의의 배열입니다. 각 스테이트 머신 정의에는 고유한 `StartAt`, `Succeed`, `Fail` 상태가 포함되어야 합니다. 이 배열의 상태 머신 정의는 자체 정의 이외의 상태를 참조할 수 없습니다.

> **Note**
>
> 각 브랜치 상태 머신은 동일한 상태 머신 컨텍스트를 공유하므로 한 브랜치에서 변수를 설정한 다음 다른 브랜치에서 해당 변수를 읽으면 예상치 못한 동작이 발생할 수 있습니다.

브랜치 상태 머신을 모두 실행한 후에만 `Parallel` 상태가 다음 상태로 이동합니다. 디바이스가 필요한 각 상태는 디바이스를 사용할 수 있을 때까지 실행을 대기합니다. 여러 디바이스를 사용할 수 있는 경우 이 상태는 여러 그룹의 테스트 케이스를 병렬로 실행합니다. 사용할 수 있는 디바이스가 충분하지 않으면 테스트 케이스가 순차적으로 실행됩니다. 테스트 케이스는 병렬로 실행될 때 무작위 순서로 실행되므로 동일한 테스트 그룹에서 테스트를 실행하는 데 여러 디바이스가 사용될 수 있습니다.

오류 처리

실행 오류를 처리하려면 브랜치 상태 머신과 부모 상태 머신이 모두 해당 `Fail` 상태로 전환되는지 확인하십시오.

브랜치 상태 머신은 부모 상태 머신에 실행 오류를 전송하지 않으므로 `Catch` 블록을 사용하여 브랜치 상태 머신의 실행 오류를 처리할 수 없습니다. 대신 공유 상태 머신 컨텍스트의 `hasExecutionErrors` 값을 사용하십시오. 이렇게 하는 방법의 예는 상태 머신 예제: 두 테스트 그룹을 병렬로 실행 섹션을 참조하세요.

AddProductFeatures

`AddProductFeatures` 상태에서는 IDT에서 생성한 `awsiotdevicetester_report.xml` 파일에 제품 기능을 추가할 수 있습니다.

제품 기능은 디바이스가 충족할 수 있는 특정 기준에 대한 사용자 정의 정보입니다. 예를 들어, `MQTT` 제품 기능은 디바이스가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. 보고서에서 제품 기능은 지정된 테스트의 통과 여부에 따라 `supported`, `not-supported` 또는 사용자 지정 값으로 설정됩니다.

> **Note**
>
> AddProductFeatures 상태는 자체적으로 보고서를 생성하지 않습니다. 보고서를 생성하려면 이 상태를 Report 상태로 전환해야 합니다.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`Features`

awsiotdevicetester_report.xml 파일에 표시할 제품 기능의 배열.

`Feature`

기능의 이름입니다.

`FeatureValue`

선택 사항. 보고서에서 `supported` 대신 사용할 사용자 지정 값입니다. 이 값을 지정하지 않으면 테스트 결과에 따라 기능 값이 `supported` 또는 `not-supported`로 설정됩니다.

`FeatureValue`에 사용자 지정 값을 사용하면 동일한 기능을 다른 조건으로 테스트할 수 있으며, IDT는 지원되는 조건에 대한 기능 값을 연결합니다. 예를 들어, 다음 발췌문은 두 개의 개별 기능 값이 있는 `MyFeature` 기능을 보여줍니다.

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

두 테스트 그룹이 모두 통과하면 기능 값이 `first-feature-supported, second-feature-supported`로 설정됩니다.

`Groups`

선택 사항. 테스트 그룹 ID의 배열입니다. 기능을 지원하려면 지정된 각 테스트 그룹 내의 모든 테스트를 통과해야 합니다.

`OneOfGroups`

선택 사항. 테스트 그룹 ID의 배열입니다. 기능이 지원되려면 지정된 테스트 그룹 중 하나 이상의 모든 테스트를 통과해야 합니다.

`TestCases`

선택 사항. 테스트 케이스 ID의 배열입니다. 이 값을 지정하면 다음이 적용됩니다.

- 기능이 지원되려면 지정된 모든 테스트 케이스를 통과해야 합니다.
- `Groups`은 테스트 그룹 ID는 하나만 포함해야 합니다.
- `OneOfGroups`은 지정하지 않아야 합니다.

`IsRequired`

선택 사항. 보고서에서 이 기능을 선택적 기능으로 표시하려면 `false`로 설정합니다. 기본 값은 `true`입니다.

`ExecutionMethods`

선택 사항. `device.json` 파일에 지정된 `protocol` 값과 일치하는 실행 메서드의 배열. 이 값을 지정하는 경우 테스트 실행기는 이 배열의 값 중 하나와 일치하는 `protocol` 값을 지정하여 보고서에 기능을 포함해야 합니다. 이 값을 지정하지 않으면 기능이 항상 보고서에 포함됩니다.

`AddProductFeatures` 상태를 사용하려면 `RunTask` 상태의 `ResultVar` 값을 다음 값 중 하나로 설정해야 합니다.

- 개별 테스트 케이스 ID를 지정한 경우 `ResultVar`을(를) *`group-id_test-id`*`_passed`(으)로 설정합니다.
- 개별 테스트 케이스 ID를 지정하지 않은 경우 `ResultVar`을(를) *`group-id`*`_passed`(으)로 설정합니다.

`AddProductFeatures` 상태에서는 다음과 같은 방식으로 테스트 결과를 확인합니다.

- 테스트 케이스 ID를 지정하지 않은 경우 각 테스트 그룹의 결과는 상태 머신 컨텍스트의 *`group-id`*`_passed` 변수 값에 따라 결정됩니다.
- 테스트 케이스 ID를 지정한 경우 각 테스트의 결과는 상태 머신 컨텍스트의 *`group-id_test-id`*`_passed` 변수 값을 기반으로 결정됩니다.

오류 처리

이 상태에서 제공된 그룹 ID가 유효한 그룹 ID가 아닌 경우 이 상태로 인해 `AddProductFeaturesError` 실행 오류가 발생합니다. 상태에서 실행 오류가 발생하는 경우 상태 머신 컨텍스트의 `hasExecutionErrors` 변수도 `true`로 설정됩니다.

보고서

`Report` 상태는 *`suite-name`*`_Report.xml` 및 `awsiotdevicetester_report.xml` 파일을 생성합니다. 또한 이 상태는 보고서를 콘솔로 스트리밍합니다.

```
{
```

```
    "Type": "Report",
    "Next": "<state-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

테스트 실행기가 테스트 결과를 볼 수 있도록 테스트 실행 흐름이 끝날 무렵에는 항상 `Report` 상태로 전환해야 합니다. 일반적으로 이 상태 이후의 다음 상태는 `Succeed`입니다.

오류 처리

이 상태에서 보고서를 생성하는 데 문제가 발생하면 `ReportError` 실행 오류가 발생합니다.

LogMessage

`LogMessage` 상태는 `test_manager.log` 파일을 생성하고 로그 메시지를 콘솔로 스트리밍합니다.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`Level`

로그 메시지를 생성할 때의 오류 수준입니다. 유효하지 않은 수준을 지정하면 이 상태가 오류 메시지를 생성하고 삭제합니다.

`Message`

로그할 메시지.

SelectGroup

`SelectGroup` 상태는 상태 머신 컨텍스트를 업데이트하여 선택된 그룹을 나타냅니다. 이 상태에서 설정된 값은 이후의 모든 `Choice` 상태에서 사용됩니다.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

`TestGroups`

선택된 것으로 표시될 테스트 그룹 배열입니다. 이 배열의 각 테스트 그룹 ID에 대해 *group-id*`_selected` 변수는 컨텍스트에서 `true`로 설정됩니다. IDT는 지정된 그룹이 존재하는지 여부를 확인하지 않으므로 유효한 테스트 그룹 ID를 제공해야 합니다.

Fail

`Fail` 상태는 상태 머신이 제대로 실행되지 않았음을 나타냅니다. 이는 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
    "Type": "Fail"
}
```

Succeed

`Succeed` 상태는 상태 머신이 올바르게 실행되었음을 나타냅니다. 이는 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
    "Type": "Succeed"
```

```
}
```

상태 머신 컨텍스트

상태 머신 컨텍스트는 실행 중에 상태 머신이 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 상태 머신 컨텍스트는 상태 머신에서만 액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어 `userdata.json` 파일에서 테스트 실행기가 구성한 정보를 사용하여 특정 테스트 실행이 필요한지 여부를 결정할 수 있습니다.

상태 머신 컨텍스트는 다음 형식을 사용합니다.

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

`pool`

테스트 실행을 위해 선택한 디바이스 풀에 대한 정보. 선택한 디바이스 풀의 경우 이 정보는 `device.json` 파일에 정의된 해당 최상위 디바이스 풀 배열 요소에서 검색됩니다.

`userData`

`userdata.json` 파일의 정보.

`config`

정보는 `config.json` 파일을 정의합니다.

`suiteFailed`

값은 상태 머신이 시작될 때 `false`로 설정됩니다. 테스트 그룹이 `RunTask` 상태로 실패하는 경우 이 값은 상태 머신의 남은 실행 기간 동안으로 `true`로 설정됩니다.

`specificTestGroups`

테스트 실행기가 전체 테스트 제품군 대신 실행할 특정 테스트 그룹을 선택하면 이 키가 생성되고 여기에는 특정 테스트 그룹 ID 목록이 포함됩니다.

`specificTestCases`

테스트 실행기가 전체 테스트 제품군 대신 실행할 특정 테스트 케이스를 선택하면 이 키가 생성되고 여기에는 특정 테스트 케이스 ID 목록이 포함됩니다.

`hasExecutionErrors`

상태 머신이 시작될 때 종료되지 않습니다. 어떤 상태에서든 실행 오류가 발생하는 경우 이 변수가 생성되고 남은 상태 머신 실행 기간 동안 `true`로 설정됩니다.

JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의의 JSONPath 쿼리 구문은 `{{$.query}}`입니다. 일부 상태에서는 JSONPath 쿼리를 자리 표시자 문자열로 사용할 수 있습니다. IDT는 자리 표시자 문자열을 컨텍스트에서 평가된 JSONPath 쿼리의 값으로 대체합니다. 다음 값에 자리 표시자를 사용할 수 있습니다.

- `RunTask` 상태의 `TestCases` 값.
- `Expression` 값 `Choice` 상태.

상태 머신 컨텍스트에서 데이터에 액세스할 때 다음 조건이 충족되는지 확인하십시오.

- JSON 경로는 `$.`로 시작해야 합니다.
- 각 값은 문자열, 숫자 또는 부울로 평가되어야 합니다.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 IDT 컨텍스트 사용 섹션을 참조하십시오.

실행 오류

실행 오류는 상태 머신이 상태를 실행할 때 발생하는 상태 머신 정의의 오류입니다. IDT는 `test_manager.log` 파일의 각 오류에 대한 정보를 기록하고 로그 메시지를 콘솔로 스트리밍합니다.

다음 방법을 사용하여 실행 오류를 처리할 수 있습니다.

- 상태 정의에 Catch 블록을 추가합니다.
- 상태 머신 컨텍스트에서 hasExecutionErrors 값의 값을 확인합니다.

Catch

Catch를 사용하려면 상태 정의에 다음을 추가하십시오.

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Catch.ErrorEquals`

캐치할 오류 유형의 배열입니다. 실행 오류가 지정된 값 중 하나와 일치하면 상태 머신이 `Catch.Next`에서 지정된 상태로 전환됩니다. 발생하는 오류 유형에 대한 자세한 내용은 각 상태 정의를 참조하십시오.

`Catch.Next`

현재 상태에서 `Catch.ErrorEquals`에서 지정한 값 중 하나와 일치하는 실행 오류가 발생하는 경우 전환할 다음 상태입니다.

Catch 블록은 둘 중 하나가 일치할 때까지 순차적으로 처리됩니다. Catch 블록에 나열된 오류와 일치하는 오류가 없으면 상태 머신이 계속 실행됩니다. 실행 오류는 잘못된 상태 정의로 인해 발생하므로 상태에 실행 오류가 발생하면 Fail 상태로 전환하는 것이 좋습니다.

hasExecutionError

일부 상태에서는 실행 오류가 발생하는 경우 오류가 발생하는 것 외에도 상태 시스템 컨텍스트에서 `hasExecutionError` 값을 `true`로 설정합니다. 이 값을 사용하여 오류 발생 시기를 감지한 다음 `Choice` 상태를 사용하여 상태 머신을 해당 `Fail` 상태로 전환할 수 있습니다.

이 메서드는 다음과 특징이 있습니다.

- 상태 머신은 `hasExecutionError`에 할당된 어떤 값으로도 시작되지 않으며 특정 상태가 값을 설정하기 전까지는 이 값을 사용할 수 없습니다. 즉, 실행 오류가 발생하지 않을 경우 상태 머신이 중지되지 않도록 이 값에 액세스하는 `Choice` 상태에 대해 명시적으로 `FallthroughOnError`를 `false`로 설정해야 합니다.
- `true`로 설정한 후에는 `hasExecutionError`가 false로 설정되거나 컨텍스트에서 제거되지 않습니다. 즉, 이 값은 `true`로 처음 설정할 때만 유용하며 이후의 모든 상태에서는 의미 있는 값을 제공하지 않습니다.
- `hasExecutionError` 값은 `Parallel` 상태의 모든 브랜치 상태 머신과 공유되므로 액세스 순서에 따라 예상치 못한 결과가 발생할 수 있습니다.

이러한 특성 때문에 Catch 블록을 대신 사용할 수 있는 경우에는 이 방법을 사용하지 않는 것이 좋습니다.

상태 머신 예제

이 섹션에서는 몇 가지 상태 시스템 구성의 예제를 제공합니다.

예제

- 상태 머신 예제: 단일 테스트 그룹 실행
- 상태 머신 예제: 사용자가 선택한 테스트 그룹 실행
- 상태 머신 예제: 제품 기능이 포함된 단일 테스트 그룹 실행
- 상태 머신 예제: 두 테스트 그룹을 병렬로 실행

상태 머신 예제: 단일 테스트 그룹 실행

이 상태 머신:

- ID `GroupA`를 사용하여 테스트 그룹을 실행하며 이 ID는 `group.json` 파일에 있어야 합니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, `Fail`로 전환합니다.
- 보고서를 생성하고 오류가 없는 경우 `Succeed`로 전환하고 그렇지 않으면 `Fail`로 전환합니다.

```
{
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
```

```
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

상태 머신 예제: 사용자가 선택한 테스트 그룹 실행

이 상태 머신:

- 테스트 실행기가 특정 테스트 그룹을 선택했는지 확인합니다. 테스트 실행기가 테스트 그룹을 선택하지 않으면 테스트 케이스를 선택할 수 없기 때문에 상태 머신은 특정 테스트 케이스를 확인하지 않습니다.
- 테스트 그룹을 선택한 경우:

  - 선택한 테스트 그룹 내에서 테스트 케이스를 실행합니다. 이를 위해 상태 머신은 RunTask 상태의 테스트 그룹이나 테스트 케이스를 명시적으로 지정하지 않습니다.
  - 모든 테스트를 실행한 후 보고서를 생성하고 종료합니다.
- 테스트 그룹을 선택하지 않은 경우:
  - 테스트 그룹 GroupA에서 테스트를 실행합니다.
  - 보고서를 생성하고 종료합니다.

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
```

```
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

상태 머신 예제: 제품 기능이 포함된 단일 테스트 그룹 실행

이 상태 머신:

- 테스트 그룹 GroupA를 실행합니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다.
- awsiotdevicetester_report.xml 파일에 FeatureThatDependsOnGroupA 기능을 추가합니다.
  - GroupA가 통과하면 기능이 supported로 설정됩니다.
  - 이 기능은 보고서에서 선택 사항으로 표시되어 있지 않습니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
```

```
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

상태 머신 예제: 두 테스트 그룹을 병렬로 실행

이 상태 머신:

- GroupA및 GroupB 테스트 그룹을 병렬로 실행합니다. 브랜치 상태 머신의 RunTask 상태에 의해 컨텍스트에 저장된 ResultVar 변수는 AddProductFeatures 상태에서 사용할 수 있습니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다. 이 상태 머신은 Catch 블록을 사용하지 않습니다. 해당 메서드는 브랜치 상태 머신의 실행 오류를 감지하지 못하기 때문입니다.
- 통과한 그룹을 기반으로 awsiotdevicetester_report.xml 파일에 기능을 추가합니다.
  - GroupA가 통과하면 기능이 supported로 설정됩니다.
  - 이 기능은 보고서에서 선택 사항으로 표시되어 있지 않습니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

디바이스 풀에 두 개의 디바이스가 구성된 경우 GroupA 및 GroupB 둘 다 동시에 실행할 수 있습니다. 그러나 GroupA 또는 GroupB 둘 중 하나에 여러 테스트가 포함된 경우에는 두 디바이스 모두 해당 테스트에 할당될 수 있습니다. 디바이스가 하나만 구성된 경우 테스트 그룹은 순차적으로 실행됩니다.

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
```

```
                    "Next": "Succeed",
                    "TestGroup": "GroupA",
                    "ResultVar": "GroupA_passed",
                    "Catch": [
                        {
                            "ErrorEquals": [
                                "RunTaskError"
                            ],
                            "Next": "Fail"
                        }
                    ]
                },
                "Succeed": {
                    "Type": "Succeed"
                },
                "Fail": {
                    "Type": "Fail"
                }
            }
        },
        {
            "Comment": "Run GroupB state machine",
            "StartAt": "RunGroupB",
            "States": {
                "RunGroupA": {
                    "Type": "RunTask",
                    "Next": "Succeed",
                    "TestGroup": "GroupB",
                    "ResultVar": "GroupB_passed",
                    "Catch": [
                        {
                            "ErrorEquals": [
                                "RunTaskError"
                            ],
                            "Next": "Fail"
                        }
                    ]
                },
                "Succeed": {
                    "Type": "Succeed"
                },
                "Fail": {
                    "Type": "Fail"
                }
```

```
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
            }
        ]
    },
    "AddProductFeatures": {
        "Type": "AddProductFeatures",
        "Next": "Report",
        "Features": [
            {
                "Feature": "FeatureThatDependsOnGroupA",
                "Groups": [
                    "GroupA"
                ],
                "IsRequired": true
            },
            {
                "Feature": "FeatureThatDependsOnGroupB",
                "Groups": [
                    "GroupB"
                ],
                "IsRequired": true
            }
        ]
    },
    "Report": {
        "Type": "Report",
        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Fail"
```

```
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

IDT 테스트 케이스 실행 파일 생성

다음과 같은 방법으로 테스트 세트 폴더에 테스트 케이스 실행 파일을 만들고 배치할 수 있습니다.

- `test.json` 파일의 인수나 환경 변수를 사용하여 실행할 테스트를 결정하는 테스트 세트의 경우, 전체 테스트 세트에 대해 단일 테스트 사례 실행 파일을 만들거나 테스트 세트의 각 테스트 그룹에 대해 테스트 실행 파일을 만들 수 있습니다.
- 지정된 명령을 기반으로 특정 테스트를 실행하려는 테스트 세트의 경우, 테스트 세트의 각 테스트 사례에 대해 테스트 사례 실행 파일을 하나씩 만듭니다.

테스트 작성자는 사용 사례에 적합한 접근 방식을 결정하고 그에 따라 테스트 사례 실행 파일을 구성할 수 있습니다. 각 `test.json` 파일에 올바른 테스트 케이스 실행 파일 경로를 제공하고 지정된 실행 파일이 올바르게 실행되는지 확인합니다.

모든 디바이스에서 테스트 케이스를 실행할 준비가 되면 IDT는 다음 파일을 읽습니다.

- 선택한 테스트 사례에 대한 `test.json`에 따라 시작할 프로세스와 설정할 환경 변수가 결정됩니다.
- 테스트 세트용 `suite.json`에 따라 설정할 환경 변수가 결정됩니다.

IDT는 `test.json` 파일에 지정된 명령과 인수를 기반으로 필수 테스트 실행 파일 프로세스를 시작하고 필요한 환경 변수를 프로세스에 전달합니다.

IDT 클라이언트 SDK 사용

IDT 클라이언트 SDK를 사용하면 IDT 및 테스트 대상 디바이스와 상호 작용하는 데 사용할 수 있는 API 명령을 사용하여 테스트 실행 파일에 테스트 로직을 작성하는 방법을 간소화할 수 있습니다. IDT는 현재 다음과 같은 SDK를 제공합니다.

- Python용 IDT 클라이언트 SDK
- Go용 IDT 클라이언트 SDK

이 SDK는 *<device-tester-extract-location>*/sdks 폴더에 있습니다. 새 테스트 케이스 실행 파일을 만들 때는 사용할 SDK를 테스트 케이스 실행 파일이 들어 있는 폴더에 복사하고 코드에서 SDK를 참조해야 합니다. 이 섹션에서는 테스트 케이스 실행 파일에서 사용할 수 있는 사용 가능한 API 명령에 대한 간략한 설명을 제공합니다.

이 섹션의 내용

- 디바이스 상호작용
- IDT 상호 작용
- 호스트 상호작용

디바이스 상호작용

다음 명령을 사용하면 추가 디바이스 상호 작용 및 연결 관리 기능을 구현하지 않고도 테스트 중인 디바이스와 통신할 수 있습니다.

`ExecuteOnDevice`

테스트 세트가 SSH 또는 Docker 셸 연결을 지원하는 디바이스에서 셸 명령을 실행할 수 있습니다.

`CopyToDevice`

테스트 세트가 IDT를 실행하는 호스트 컴퓨터의 로컬 파일을 SSH 또는 Docker 셸 연결을 지원하는 디바이스의 지정된 위치로 복사할 수 있습니다.

`ReadFromDevice`

테스트 세트가 UART 연결을 지원하는 디바이스의 직렬 포트에서 읽을 수 있도록 허용합니다.

> **Note**
>
> IDT는 컨텍스트의 디바이스 액세스 정보를 사용하여 만든 디바이스에 대한 직접 연결을 관리하지 않으므로 테스트 사례 실행 파일에서 이러한 디바이스 상호 작용 API 명령을 사용하는 것이 좋습니다. 하지만 이러한 명령이 테스트 사례 요구 사항을 충족하지 않는 경우, IDT 컨텍스트에서 디바이스 액세스 정보를 검색하고 이를 사용하여 테스트 세트에서 디바이스에 직접 연결할 수 있습니다.

직접 연결하려면 테스트 중인 디바이스와 리소스 디바이스에 대해 각각 `device.connectivity` 및 `resource.devices.connectivity` 필드에서 정보를 검색합니다. IDT 컨텍스트 사용에 관한 자세한 내용은 IDT 컨텍스트 사용 섹션을 참조합니다.

IDT 상호 작용

다음 명령을 사용하면 테스트 세트가 IDT와 통신할 수 있습니다.

`PollForNotifications`

테스트 세트에서 IDT의 알림을 확인할 수 있습니다.

`GetContextValue` 및 `GetContextString`

테스트 세트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 내용은 IDT 컨텍스트 사용 섹션을 참조하세요.

`SendResult`

테스트 세트에서 테스트 사례 결과를 IDT에 보고할 수 있습니다. 이 명령은 테스트 세트의 각 테스트 케이스 끝에서 직접적으로 호출해야 합니다.

호스트 상호작용

다음 명령을 사용하면 테스트 세트가 호스트 머신과 통신할 수 있습니다.

`PollForNotifications`

테스트 세트에서 IDT의 알림을 확인할 수 있습니다.

`GetContextValue` 및 `GetContextString`

테스트 세트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 내용은 IDT 컨텍스트 사용 섹션을 참조하세요.

`ExecuteOnHost`

테스트 세트가 로컬 머신에서 명령을 실행할 수 있도록 하고 IDT가 테스트 케이스 실행 수명 주기를 관리할 수 있도록 합니다.

IDT CLI 명령을 활성화합니다.

`run-suite` 명령 IDT CLI는 테스트 실행기가 테스트 실행을 사용자 지정할 수 있는 몇 가지 옵션을 제공합니다. 테스트 실행기가 이러한 옵션을 사용하여 사용자 지정 테스트 세트를 실행할 수 있도록 하려면 IDT CLI에 대한 지원을 구현해야 합니다. 지원을 구현하지 않는 경우, 테스트 실행기는 여전히 테스트를 실행할 수 있지만 일부 CLI 옵션은 제대로 작동하지 않습니다. 이상적인 고객 경험을 제공하려면 IDT CLI에서 `run-suite` 명령에 대한 다음 인수에 대한 지원을 구현하는 것이 좋습니다.

`timeout-multiplier`

테스트를 실행하는 동안 모든 제한 시간에 적용할 1.0보다 큰 값을 지정합니다.

테스트 실행기는 이 인수를 사용하여 실행하려는 테스트 사례의 제한 시간을 늘릴 수 있습니다. 테스트 실행기가 `run-suite` 명령에서 이 인수를 지정하면 IDT는 이 인수를 사용하여 IDT_TEST_TIMEOUT 환경 변수의 값을 계산하고 IDT 컨텍스트에서 `config.timeoutMultiplier` 필드를 설정합니다. 이 인수를 뒷받침하려면 다음을 수행하여야 합니다.

- `test.json` 파일의 제한 시간 값을 직접 사용하는 대신 IDT_TEST_TIMEOUT 환경 변수를 읽고 올바르게 계산된 제한 시간 값을 구합니다.
- IDT 컨텍스트에서 `config.timeoutMultiplier` 값을 검색하여 장기 실행 제한 시간에 적용합니다.

제한 시간 이벤트로 인한 조기 종료에 대한 자세한 내용은 종료 동작을 지정합니다.을(를) 참조하세요.

`stop-on-first-failure`

장애가 발생할 경우, IDT에서 모든 테스트 실행을 중지하도록 지정합니다.

테스트 실행기가 `run-suite` 명령에 이 인수를 지정하면 IDT는 장애가 발생하는 즉시 테스트 실행을 중단합니다. 그러나 테스트 케이스가 병렬로 실행되는 경우, 예상치 못한 결과가 발생할 수 있습니다. 지원을 구현하려면 IDT에서 이 이벤트가 발생할 경우, 테스트 로직에서 실행 중인 모든 테스트 사례를 중지하고, 임시 리소스를 정리하고, 테스트 결과를 IDT에 보고하도록 지시해야 합니다. 실패 시 조기 종료에 대한 자세한 내용은 종료 동작을 지정합니다. 섹션을 참조하세요.

`group-id` 및 `test-id`

IDT가 선택한 테스트 그룹 또는 테스트 케이스만 실행하도록 지정합니다.

테스트 실행기는 `run-suite` 명령과 함께 이러한 인수를 사용하여 다음과 같은 테스트 실행 동작을 지정할 수 있습니다.

- 지정된 테스트 제품군에 있는 모든 테스트 그룹을 실행합니다.
- 지정된 테스트 그룹 내에서 엄선된 테스트를 실행합니다.

이러한 인수를 지원하려면 테스트 세트의 상태 머신이 상태 머신의 특정 RunTask 및 Choice 상태 세트를 포함해야 합니다. 사용자 지정 상태 머신을 사용하지 않는 경우, 기본 IDT 상태 머신에 필요한 상태가 포함되므로 추가 조치를 취할 필요가 없습니다. 하지만 사용자 지정 상태 머신을 사용하는 경우에는 상태 머신 예제: 사용자가 선택한 테스트 그룹 실행을(를) 샘플로 사용하여 상태 머신에 필요한 상태를 추가합니다.

IDT CLI 명령에 대한 자세한 내용은 사용자 지정 테스트 도구 모음 디버그 및 실행 단원을 참조합니다.

이벤트 로그 작성

테스트가 실행되는 동안 stdout 및 stderr에 데이터를 보내 콘솔에 이벤트 로그와 오류 메시지를 기록합니다. 콘솔 메시지의 형식에 대한 자세한 정보는 콘솔 메시지 형식에서 확인하세요.

IDT가 테스트 세트 실행을 마치면 *<devicetester-extract-location>*/results/*<execution-id>*/logs 폴더에 있는 test_manager.log 파일에서도 이 정보를 확인할 수 있습니다.

테스트 대상 디바이스의 로그를 포함하여 테스트 실행의 로그를 *<device-tester-extract-location>*/results/*execution-id*/logs 폴더에 있는 *<group-id>*_*<test-id>* 파일에 기록하도록 각 테스트 사례를 구성할 수 있습니다. 이렇게 하려면 testData.logFilePath 쿼리를 사용하여 IDT 컨텍스트에서 로그 파일의 경로를 검색하고 해당 경로에 파일을 만든 다음 원하는 콘텐츠를 작성해야 합니다. IDT는 실행 중인 테스트 케이스를 기반으로 경로를 자동으로 업데이트합니다. 테스트 사례에 대한 로그 파일을 만들지 않기로 선택하면 해당 테스트 사례에 대한 파일이 생성되지 않습니다.

필요에 따라 *<device-tester-extract-location>*/logs 폴더에 추가 로그 파일을 생성하도록 테스트 실행 파일을 설정할 수도 있습니다. 파일을 덮어쓰지 않도록 로그 파일 이름에 고유한 접두사를 지정하는 것이 좋습니다.

결과를 IDT에 보고합니다.

IDT는 테스트 결과를 awsiotdevicetester_report.xml 및 *suite-name*_report.xml 파일에 기록합니다. 이 보고서 파일은 *<device-tester-extract-location>*/results/*<execution-id>*/에 위치합니다. 두 보고서 모두 테스트 세트의 실행 결과를 캡처합니다. IDT에서 이러한 보고서에 사용하는 스키마에 대한 자세한 내용은 IDT 테스트 결과 및 로그 검토을(를) 참조합니다.

*suite-name*_report.xml 파일 내용을 채우려면 테스트 실행이 완료되기 전에 SendResult 명령을 사용하여 테스트 결과를 IDT에 보고해야 합니다. IDT에서 테스트 결과를 찾을 수 없는 경우, 테스트 사례에 오류가 발생합니다. 다음 Python 발췌문은 테스트 결과를 IDT로 보내는 명령을 보여줍니다.

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

API를 통해 결과를 보고하지 않는 경우, IDT는 테스트 아티팩트 폴더에서 테스트 결과를 찾습니다. 이 폴더의 경로는 IDT 컨텍스트의 testData.testArtifactsPath 파일에 저장됩니다. 이 폴더에서 IDT는 찾은 첫 번째 알파벳순으로 정렬된 XML 파일을 테스트 결과로 사용합니다.

테스트 로직이 JUnit XML 결과를 생성하는 경우, 결과를 파싱한 다음 API를 사용하여 IDT에 제출하는 대신 아티팩트 폴더의 XML 파일에 테스트 결과를 기록하여 결과를 IDT에 직접 제공할 수 있습니다.

이 방법을 사용하는 경우, 테스트 로직이 테스트 결과를 정확하게 요약하고 결과 파일의 형식을 *suite-name*_report.xml 파일과 동일한 형식으로 지정해야 합니다. IDT는 제공된 데이터에 대한 검증을 수행하지 않습니다. 단, 다음과 같은 경우는 예외입니다.

- IDT는 testsuites 태그의 모든 속성을 무시합니다. 대신 보고된 다른 테스트 그룹 결과에서 태그 속성을 계산합니다.
- 하나 이상의 testsuite 태그가 testsuites 내에 있어야 합니다.

IDT는 모든 테스트 사례에 동일한 아티팩트 폴더를 사용하고 테스트 실행 사이에 결과 파일을 삭제하지 않기 때문에 IDT에서 잘못된 파일을 읽을 경우, 이 방법을 사용하면 잘못된 보고가 발생할 수도 있습니다. 생성된 XML 결과 파일은 모든 테스트 사례에서 동일한 이름을 사용하여 각 테스트 사례의 결과를 덮어쓰고 IDT에서 사용할 수 있는 올바른 결과가 있는지 확인하는 것이 좋습니다. 테스트 세트의 보고에는 혼합된 접근 방식을 사용할 수 있습니다. 즉, 일부 테스트 사례에는 XML 결과 파일을 사용하고 다른 테스트 사례에는 API를 통해 결과를 제출하는 등 혼합된 접근 방식을 사용할 수 있지만 이 방법은 권장되지 않습니다.

종료 동작을 지정합니다.

테스트 케이스에서 실패 또는 오류 결과가 보고되더라도 텍스트 실행 파일이 항상 종료 코드 0으로 종료되도록 구성합니다. 0이 아닌 종료 코드는 테스트 케이스가 실행되지 않았음을 나타내거나 테스트 케이스 실행 파일이 IDT에 결과를 전달할 수 없는 경우에만 사용합니다. IDT가 0이 아닌 종료 코드를 받으면 테스트 케이스에 오류가 발생하여 테스트 케이스를 실행할 수 없게 된 것으로 표시합니다.

IDT는 다음 이벤트에서 테스트 케이스가 완료되기 전에 테스트 케이스의 실행을 중단하도록 요청하거나 예상할 수 있습니다. 이 정보를 사용하여 테스트 케이스에서 이러한 각 이벤트를 감지하도록 테스트 케이스 실행 파일을 구성합니다.

제한 시간

테스트 케이스가 `test.json` 파일에 지정된 제한 시간 값보다 오래 실행될 때 발생합니다. 테스트 실행기가 `timeout-multiplier` 인수를 사용하여 제한 시간 승수를 지정한 경우, IDT는 승수로 제한 시간 값을 계산합니다.

이 이벤트를 탐지하려면 IDT_TEST_TIMEOUT 환경 변수를 사용합니다. 테스트 실행기가 테스트를 시작하면 IDT는 IDT_TEST_TIMEOUT 환경 변수의 값을 계산된 제한 시간 값(초)으로 설정하고 변수를 테스트 케이스 실행 파일로 전달합니다. 변수 값을 읽어 적절한 타이머를 설정할 수 있습니다.

인터럽트

테스트 러너가 IDT를 인터럽트할 때 발생합니다. 예를 들어, Ctrl+C 키를 누르면 됩니다.

터미널은 신호를 모든 하위 프로세스에 전파하므로 인터럽트 신호를 감지하도록 테스트 케이스에 신호 처리기를 구성하기만 하면 됩니다.

또는 정기적으로 API를 폴링하여 `PollForNotifications` API 응답의 `CancellationRequested` 부울 값을 확인할 수도 있습니다. IDT는 인터럽트 신호를 수신하면 `CancellationRequested` 부울 값을 `true`(으)로 설정합니다.

첫 번째 실패 시 중지

현재 테스트 케이스와 병렬로 실행 중인 테스트 케이스가 실패하고 테스트 실행기가 `stop-on-first-failure` 인수를 사용하여 IDT에 오류가 발생할 경우, 중지하도록 지정하는 경우, 발생합니다.

이 이벤트를 탐지하기 위해 주기적으로 API를 폴링하여 `PollForNotifications` API 응답의 `CancellationRequested` 부울 값을 확인할 수 있습니다. IDT에서 장애가 발생하고 첫 번째 실패 시 중지되도록 구성된 경우, IDT는 `CancellationRequested` 부울 값을 `true`(으)로 설정합니다.

이러한 이벤트가 발생하면 IDT는 현재 실행 중인 테스트 케이스의 실행이 완료될 때까지 5분 동안 기다립니다. 실행 중인 모든 테스트 케이스가 5분 내에 종료되지 않으면 IDT는 각 프로세스를 강제로 중지합니다. 프로세스가 종료되기 전에 IDT에서 테스트 결과를 받지 못한 경우, 테스트 케이스가 제한 시간이 초과된 것으로 표시됩니다. 가장 좋은 방법은 테스트 케이스에서 이벤트 중 하나가 발생할 때 다음 작업을 수행하도록 하는 것입니다.

1. 일반 테스트 로직 실행을 중단합니다.
2. 테스트 대상 디바이스의 테스트 아티팩트와 같은 임시 리소스를 모두 정리합니다.
3. 테스트 실패 또는 오류와 같은 테스트 결과를 IDT에 보고합니다.
4. 종료

IDT 컨텍스트 사용

IDT가 테스트 도구 모음을 실행할 때, 테스트 도구 모음은 각 테스트 실행 방식을 결정하는 데 사용할 수 있는 데이터 세트에 액세스할 수 있습니다. 이 데이터를 IDT 컨텍스트라고 합니다. 예를 들어 테스트 러너가 `userdata.json` 파일로 제공한 사용자 데이터 구성은 IDT 컨텍스트의 테스트 도구 모음에서 사용할 수 있도록 만들어졌습니다.

IDT 컨텍스트는 읽기 전용 JSON 문서로 간주할 수 있습니다. 테스트 도구 모음은 객체, 배열, 숫자 등과 같은 표준 JSON 데이터 유형을 사용하여 컨텍스트에서 데이터를 검색하고, 컨텍스트에 데이터를 쓸 수 있습니다.

컨텍스트 스키마

IDT 컨텍스트는 다음 형식을 사용합니다.

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
```

```
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

config.json 파일의 정보. config 필드에는 다음과 같은 추가 필드도 포함됩니다.

`config.timeoutMultiplier`

테스트 도구 모음에서 사용하는 모든 시간 제한 값의 승수입니다. 이 값은 IDT CLI에서 테스트 러너에 의해 지정됩니다. 기본값은 1입니다.

`device`

테스트 실행을 위해 선택한 디바이스에 대한 정보. 이 정보는 선택한 디바이스의 device.json 파일에 있는 devices 배열 요소와 동일합니다.

`devicePool`

테스트 실행을 위해 선택한 디바이스 풀에 대한 정보. 이 정보는 선택한 디바이스 풀에 대해 device.json 파일에 정의된 최상위 디바이스 풀 배열 요소와 동일합니다.

`resource`

resource.json 파일의 리소스 디바이스에 대한 정보.

`resource.devices`

이 정보는 resource.json 파일에 정의된 devices 배열과 동일합니다. 각 devices 요소에는 다음과 같은 추가 필드가 포함됩니다.

`resource.device.name`

리소스 디바이스의 이름입니다. 이 값은 test.json 파일의 requiredResource.name 값으로 설정됩니다.

`testData.awsCredentials`

테스트에서 AWS 클라우드에 연결하는 데 사용하는 AWS 보안 인증입니다. 이 정보는 `config.json` 파일에서 가져옵니다.

`testData.logFilePath`

테스트 사례가 로그 메시지를 기록하는 로그 파일의 경로입니다. 이 파일이 존재하지 않는 경우 테스트 도구 모음에서 해당 파일을 생성합니다.

`userData`

테스트 러너가 `userdata.json` 파일에서 제공한 정보.

컨텍스트에서 데이터 액세스

JSON 파일 및 `GetContextValue` 및 `GetContextString` API로 실행 가능한 텍스트 파일에서 JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. IDT 컨텍스트에 액세스하기 위한 JSONPath 문자열의 구문은 다음과 같이 다양합니다.

- `suite.json`와(과) `test.json`에서는 `{{`*`query`*`}}`을(를) 사용합니다. 즉, 루트 요소 `$.`을(를) 사용하여 표현식을 시작하지 마십시오.
- `statemachine.json`에서는 `{{$.`*`query`*`}}`을(를) 사용합니다.
- API 명령에서는 명령에 따라 *`query`* 또는 `{{$.`*`query`*`}}`을(를) 사용합니다. 자세한 내용을 알아보려면 SDK에서 인라인 설명서를 참조하세요.

다음 표에서는 일반적인 JSONPath 표현식의 연산자를 설명합니다.

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.childName`	Accesses the child element with name `childName` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to

Operator	Description
	access the `awsRegion` value in the `config` object is `$.config.awsRegion` .
`[start:end]`	Filters elements from an array, retrieving items beginning from the ## index and going up to the `end` index, both inclusive.
`[index1, index2, ... , indexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[?(expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

필터 표현식을 만들려면 다음 구문을 사용하십시오.

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

이 구문에서:

- `jsonpath`은(는) 표준 JSON 구문을 사용하는 JSONPath입니다.
- `value`은(는) 표준 JSON 구문을 사용하는 모든 사용자 지정 값입니다.
- `operator`는 다음 작업 중 하나를 호출합니다.
  - < (미만)
  - <= (이하)
  - == (같음)

    표현식의 JSONPath 또는 값이 배열, 부울 또는 객체 값인 경우 이것이 사용할 수 있는 유일하게 지원되는 바이너리 연산자입니다.
  - >= (이상)
  - > (초과)
  - =~(정규 표현식 일치). 필터 표현식에서 이 연산자를 사용하려면 표현식 왼쪽의 JSONPath 또는 값이 문자열로 평가되어야 하고, 오른쪽은 RE2 구문을 따르는 패턴 값이어야 합니다.

{{*query*}} 형식의 JSONPath 쿼리를 `test.json` 파일의 `args` 및 `environmentVariables` 필드, `suite.json` 파일의 `environmentVariables` 필드 내에서 자리 표시자 문자열로 사용할 수 있습니다. IDT는 컨텍스트 조회를 수행하고 쿼리의 평가된 값으로 필드를 채웁니다. 예를 들어 `suite.json` 파일에서 자리 표시자 문자열을 사용하여 각 테스트 사례에 따라 변경되는 환경 변수 값을 지정할 수 있으며, IDT는 환경 변수를 각 테스트 사례에 맞는 값으로 채웁니다. 하지만 `test.json` 및 `suite.json` 파일에서 자리 표시자 문자열을 사용하는 경우 쿼리에 다음 사항을 고려해야 합니다.

- 쿼리에서 나타나는 `devicePool` 키는 모두 소문자로 입력해야 합니다. 즉, `devicepool`을(를) 대신 사용하십시오.
- 배열의 경우 문자열 배열만 사용할 수 있습니다. 또한 배열은 비표준 `item1, item2,...,itemN` 형식을 사용합니다. 배열에 요소가 하나만 포함된 경우 이 배열은 `item`(으)로 직렬화되어 문자열 필드와 구분할 수 없게 됩니다.
- 자리 표시자를 사용하여 컨텍스트에서 객체를 검색할 수 없습니다.

이러한 고려 사항 때문에 가능하면 `test.json` 및 `suite.json` 파일의 자리 표시자 문자열 대신 API를 사용하여 테스트 로직의 컨텍스트에 액세스하는 것이 좋습니다. 하지만 경우에 따라 JSONPath 자리 표시자를 사용하여 단일 문자열을 가져와서 환경 변수로 설정하는 것이 더 편리할 수 있습니다.

테스트 러너를 위한 설정 구성

사용자 지정 테스트 도구 모음을 실행하려면 테스트 러너가 실행하려는 테스트 도구 모음을 기반으로 설정을 구성해야 합니다. 설정은 *<device-tester-extract-location>*`/configs/` 폴더에 있는 JSON 구성 파일 템플릿을 기반으로 지정됩니다. 필요한 경우 테스트 러너는 IDT가 AWS 클라우드에 연결하는 데 사용할 AWS 보안 인증도 설정해야 합니다.

테스트 작성자는 테스트 도구 모음을 디버깅하도록 이러한 파일을 구성해야 합니다. 테스트 러너가 테스트 도구 모음을 실행하는 데 필요한 대로 다음 설정을 구성할 수 있도록 지침을 제공해야 합니다.

device.json 구성

`device.json` 파일은 테스트가 실행되는 디바이스에 대한 정보가 필요합니다(예: IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처).

테스트 러너는 *<device-tester-extract-location>*`/configs/` 폴더에 있는 다음 템플릿 `device.json` 파일을 사용하여 이 정보를 제공할 수 있습니다.

```
[
    {
        "id": "<pool-id>",
```

```
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
```

```
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

디바이스 풀이라고 하는 디바이스 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 디바이스의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 디바이스가 사용됩니다.

`sku`

테스트 대상 디바이스를 고유하게 식별하는 영숫자 값입니다. SKU는 정규화된 디바이스를 추적하는 데 사용됩니다.

> **Note**
>
> AWS Partner 디바이스 카탈로그에 보드를 등록하려면 여기서 지정하는 SKU가 목록 등록 프로세스에 사용하는 SKU와 일치해야 합니다.

`features`

선택 사항. 디바이스의 지원되는 기능이 포함된 배열입니다. 디바이스 기능은 테스트 도구 모음에서 구성한 사용자 정의 값입니다. `device.json` 파일에 포함할 기능 이름 및 값에 대한 정보를 테스트 러너에게 제공해야 합니다. 예를 들어, 다른 디바이스의 MQTT 서버 역할을 하는 디바이스를 테스트하려는 경우 이름이 `MQTT_QOS`(으)로 지정된 기능에 대해 지원되는 특정 수준을 검증하도록 테스트 로직을 구성할 수 있습니다. 테스트 러너는 이 기능 이름을 제공하고 기능 값을 디바이스에서 지원하는 QOS 수준으로 설정합니다. 제공된 정보는 `devicePool.features` 쿼리를 사용하여 IDT 컨텍스트에서 검색하거나 `pool.features` 쿼리를 사용하여 상태 머신 컨텍스트에서 검색할 수 있습니다.

`features.name`

기능의 이름입니다.

`features.value`

지원되는 기능 값.

`features.configs`

필요한 경우 기능에 대한 구성 설정.

`features.config.name`

구성 설정의 이름입니다.

`features.config.value`

지원되는 설정값.

`devices`

테스트할 풀의 디바이스 배열입니다. 최소 1개 이상의 디바이스가 필요합니다.

`devices.id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`connectivity.protocol`

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 디바이스는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 값은 `uart` 물리적 디바이스의 경우 및, `docker` Docker 컨테이너의 경우 `ssh` 입니다.

`connectivity.ip`

테스트 대상 디바이스의 IP 입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.password`

테스트 대상 디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.privKeyPath`

테스트 대상 디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.user`

테스트 대상 디바이스에 로그인하기 위한 사용자 이름입니다.

`connectivity.serialPort`

선택 사항. 디바이스가 연결된 직렬 포트입니다.

이 속성은 `connectivity.protocol`이 `uart`로 설정된 경우에만 적용됩니다.

`connectivity.containerId`

테스트 대상 Docker 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

`connectivity.containerUser`

선택 사항. 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 4입니다.

이 속성은 connectivity.protocol이 docker로 설정된 경우에만 적용됩니다.

> **Note**
>
> 테스트 러너가 테스트를 위해 잘못된 디바이스 연결을 구성했는지 확인하기 위해 상태 머신 컨텍스트에서 pool.Devices[0].Connectivity.Protocol을(를) 검색하고 이를 Choice 상태에서 예상한 값과 비교할 수 있습니다. 잘못된 프로토콜이 사용된 경우 LogMessage 상태를 사용하여 메시지를 인쇄하고 Fail 상태로 전환하십시오.
> 또는 오류 처리 코드를 사용하여 잘못된 디바이스 유형에 대한 테스트 실패를 보고할 수 있습니다.

(선택 사항) userdata.json 구성

userdata.json 파일에는 테스트 도구 모음에 필요하지만 device.json 파일에 지정되지 않은 모든 추가 정보가 들어 있습니다. 이 파일의 형식은 테스트 도구 모음에 정의된 userdata_scheme.json 파일에 따라 달라집니다. 테스트 작성자인 경우, 이 정보를 작성한 테스트 도구 모음을 실행할 사용자에게 제공해야 합니다.

(선택 사항) resource.json 구성

resource.json 파일에는 리소스 디바이스로 사용될 모든 디바이스에 대한 정보가 들어 있습니다. 리소스 디바이스는 테스트 대상 디바이스의 특정 기능을 테스트하는 데 필요한 디바이스입니다. 예를 들어 디바이스의 Bluetooth 기능을 테스트하려면 리소스 디바이스를 사용하여 디바이스가 제대로 연결될 수 있는지 테스트할 수 있습니다. 리소스 디바이스는 선택 사항이며 필요한 만큼 리소스 디바이스를 요청할 수 있습니다. 테스트 작성자는 test.json 파일을 사용하여 테스트에 필요한 리소스 디바이스 기능을 정의합니다. 그런 다음 테스트 러너는 resource.json 파일을 사용하여 필수 기능을 갖춘 리소스 디바이스 풀을 제공합니다. 이 정보를 작성한 테스트 도구 모음을 실행할 사용자에게 제공해야 합니다.

테스트 러너는 *<device-tester-extract-location>*/configs/ 폴더에 있는 다음 템플릿 resource.json 파일을 사용하여 이 정보를 제공할 수 있습니다.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
```

```
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

디바이스 풀이라고 하는 디바이스 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 디바이스의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 디바이스는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 디바이스가 사용됩니다.

`features`

선택 사항. 디바이스의 지원되는 기능이 포함된 배열입니다. 이 필드에 필요한 정보는 테스트 도구 모음의 test.json 파일에 정의되어 있으며, 실행할 테스트와 이 테스트를 실행하는 방법을 결정합니다. 테스트 도구 모음에 기능이 필요하지 않은 경우에는 이 필드가 필요하지 않습니다.

`features.name`

기능의 이름입니다.

`features.version`

기능 버전.

`features.jobSlots`

디바이스를 동시에 사용할 수 있는 테스트 수를 나타내는 설정입니다. 기본값은 1입니다.

`devices`

테스트할 풀의 디바이스 배열입니다. 최소 1개 이상의 디바이스가 필요합니다.

`devices.id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`connectivity.protocol`

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 디바이스는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 값은 `uart` 물리적 디바이스의 경우 및, `docker` Docker 컨테이너의 경우 `ssh` 입니다.

`connectivity.ip`

테스트 대상 디바이스의 IP 입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.password`

테스트 대상 디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.privKeyPath`

테스트 대상 디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.user`

테스트 대상 디바이스에 로그인하기 위한 사용자 이름입니다.

`connectivity.serialPort`

선택 사항. 디바이스가 연결된 직렬 포트입니다.

이 속성은 `connectivity.protocol`이 `uart`로 설정된 경우에만 적용됩니다.

`connectivity.containerId`

테스트 대상 Docker 컨테이너의 컨테이너 ID 또는 이름입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

`connectivity.containerUser`

선택 사항. 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `docker`로 설정된 경우에만 적용됩니다.

(선택 사항) config.json 구성

`config.json` 파일 형식의 IDT용 구성 정보가 들어 있습니다. 일반적으로 테스트 러너는 IDT에 대한 AWS 사용자 보안 인증 및 선택적으로 AWS 리전을 제공하는 경우를 제외하고는 이 파일을 수정할 필요가 없습니다. 필요한 권한이 있는 AWS 보안 인증이 제공되면 AWS IoT Device Tester는 사용량 지표를 수집하여 AWS에 제출합니다. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. 자세한 내용은 IDT 사용량 지표 섹션을 참조하세요.

테스트 러너는 다음 방법 중 하나로 AWS 보안 인증을 구성할 수 있습니다.

- 보안 인증 파일

  IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 구성 및 자격 증명 파일을 참조하십시오.

  자격 증명 파일의 위치는 사용하는 운영 체제에 따라 달라집니다.

  - macOS, Linux의 경우: `~/.aws/credentials`
  - Windows: `C:\Users\UserName\.aws\credentials`
- 환경 변수

  환경 변수는 운영 체제에서 유지 관리하고 시스템 명령에서 사용하는 변수입니다. SSH 세션 중에 정의된 변수는 해당 세션이 종료된 후에는 사용할 수 없습니다. IDT는 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY` 환경 변수를 사용하여 AWS 보안 인증을 저장할 수 있습니다.

  Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 export를 사용합니다.

  ```
  export AWS_ACCESS_KEY_ID=<your_access_key_id>
  export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
  ```

  Windows에서 이러한 변수를 설정하려면 set을 사용합니다.

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

IDT용 AWS 보안 인증을 구성하기 위해 테스트 러너는 *<device-tester-extract-location>*/configs/ 폴더에 있는 config.json 파일에서 auth 섹션을 편집합니다.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

> **Note**
>
> 이 파일의 모든 경로는 *<device-tester-extract-location>*을 기준으로 정의됩니다.

`log.location`

*<device-tester-extract-location>*에 있는 로그 폴더의 경로.

`configFiles.root`

구성 파일이 포함된 폴더 경로입니다.

`configFiles.device`

device.json 파일 경로입니다.

`testPath`

테스트 도구 모음이 들어 있는 폴더의 경로.

`reportPath`

IDT에서 테스트 도구 모음을 실행한 후 테스트 결과를 포함할 폴더의 경로입니다.

`awsRegion`

선택 사항. 테스트 도구 모음에서 사용할 AWS 리전. 설정하지 않으면 테스트 도구 모음은 각 테스트 도구 모음에 지정된 기본 리전을 사용합니다.

`auth.method`

IDT가 AWS 보안 인증을 검색하는 데 사용하는 메서드입니다. 지원되는 값은 보안 인증 파일에서 보안 인증을 검색하는 `file`와(과) 환경 변수를 사용하여 보안 인증을 검색하는 `environment`입니다.

`auth.credentials.profile`

보안 인증 파일에서 사용할 보안 인증 프로필. 이 속성은 `auth.method`이 `file`로 설정된 경우에만 적용됩니다.

사용자 지정 테스트 도구 모음 디버그 및 실행

필요한 구성이 설정되면 IDT에서 테스트 도구 모음을 실행할 수 있습니다. 전체 테스트 도구 모음의 런타임은 하드웨어와 테스트 도구 모음의 구성에 따라 달라집니다. 참조를 위해, Raspberry Pi 3B에서 전체 AWS IoT Greengrass 자격 테스트 도구 모음을 완료하는 데 약 30분이 걸립니다.

테스트 도구 모음을 작성할 때 IDT를 사용하여 테스트 도구 모음을 디버그 모드에서 실행하여 코드를 실행하기 전에 검사하거나 테스트 실행기에 제공할 수 있습니다.

IDT를 디버그 모드에서 실행합니다.

테스트 도구 모음은 디바이스와 상호 작용하고, 컨텍스트를 제공하고, 결과를 받기 위해 IDT를 사용하기 때문에 IDT 상호 작용 없이 IDE에서 테스트 도구 모음을 간단히 디버깅할 수는 없습니다. 이를 위해 IDT CLI는 IDT를 디버그 모드에서 실행할 수 있는 `debug-test-suite` 명령을 제공합니다. 다음 명령을 실행하여 `debug-test-suite`에 대해 사용 가능한 옵션을 봅니다.

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

IDT를 디버그 모드에서 실행할 때 IDT는 실제로 테스트 도구 모음을 시작하거나 상태 머신을 실행하지 않습니다. 대신 IDE와 상호 작용하여 IDE에서 실행 중인 테스트 도구 모음의 요청에 응답하고 콘솔에 로그를 인쇄합니다. IDT는 타임아웃이 발생하지 않으며 수동으로 중단될 때까지 종료를 기다립니다. 디버그 모드에서도 IDT는 상태 머신을 실행하지 않으며 보고서 파일을 생성하지 않습니다. 테스트 도구 모음을 디버깅하려면 IDE를 사용하여 IDT가 일반적으로 구성 JSON 파일에서 얻는 일부 정보를 제공해야 합니다. 다음 정보를 제공하는지 확인합니다.

- 각 테스트의 환경 변수 및 인수 IDT는 `test.json` 또는 `suite.json`에서 이 정보를 읽지 않습니다.
- 리소스 디바이스를 선택하기 위한 인수. IDT는 `test.json`에서 이 정보를 읽지 않습니다.

테스트 도구 모음을 디버깅하려면 다음 단계를 완료하세요.

1. 테스트 도구 모음을 실행하는 데 필요한 설정 구성 파일을 생성합니다. 예를 들어, 테스트 도구 모음에 `device.json`, `resource.json`, 및 `user data.json`이(가) 필요한 경우, 필요에 따라 모두 구성해야 합니다.
2. 다음 명령을 실행하여 IDT를 디버그 모드로 설정하고 테스트를 실행하는 데 필요한 디바이스를 선택합니다.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   이 명령을 실행하면 IDT는 테스트 도구 모음의 요청을 기다린 다음 요청에 응답합니다. 또한 IDT는 IDT Client SDK의 케이스 프로세스에 필요한 환경 변수를 생성합니다.
3. IDE에서 `run` 또는 `debug` 구성을 사용하여 다음을 수행합니다.
   a. IDT에서 생성한 환경 변수의 값을 설정합니다.
   b. `test.json` 및 `suite.json` 파일에 지정한 모든 환경 변수 또는 인수의 값을 설정합니다.
   c. 필요에 따라 중단점을 설정합니다.
4. IDE에서 테스트 도구 모음을 실행합니다.

   필요한 횟수만큼 테스트 도구 모음을 디버깅하고 다시 실행할 수 있습니다. 디버그 모드에서는 IDT가 타임아웃되지 않습니다.
5. 디버깅을 완료한 후 IDT를 중단하여 디버그 모드를 종료하십시오.

테스트를 실행하기 위한 IDT CLI 명령

다음 단원에서는 IDT CLI 명령에 대해 설명합니다.

IDT v4.0.0

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 테스트 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 테스트 제품군을 나열합니다.

`list-supported-products`

IDT 버전에 대해 지원되는 제품(이 경우, AWS IoT Greengrass 버전)과 현재 IDT 버전에 대한 AWS IoT Greengrass 자격 테스트 제품군 버전을 나열합니다.

`list-test-cases`

주어진 테스트 그룹의 테스트 케이스를 나열합니다. 다음 옵션이 지원됩니다.

- `group-id`. 검색할 테스트 그룹입니다. 이 옵션은 필수이며 단일 그룹을 지정해야 합니다.

`run-suite`

디바이스의 풀에 대해 테스트 제품군을 실행합니다. 다음은 몇 가지 일반적으로 사용되는 옵션입니다:

- `suite-id`. 실행할 테스트 제품군 버전입니다. 지정하지 않으면 IDT는 `tests` 폴더의 최신 버전을 사용합니다.
- `group-id`. 실행할 테스트 그룹(쉼표로 구분된 목록). 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.
- `test-id`. 실행할 테스트 케이스(쉼표로 구분된 목록). 지정된 경우, `group-id`은(는) 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 디바이스 풀. `device.json` 파일에 여러 디바이스 풀이 정의되어 있는 경우, 테스트 실행기는 하나의 풀을 지정해야 합니다.
- `timeout-multiplier`. 사용자 정의 승수를 사용하여 테스트에 대해 `test.json` 파일에 지정된 테스트 실행 제한 시간을 수정하도록 IDT를 구성합니다.

- `stop-on-first-failure`. 첫 번째 실패 시 실행을 중지하도록 IDT를 구성합니다. 이 옵션은 지정된 테스트 그룹을 디버깅하는 데 `group-id`와(과) 함께 사용해야 합니다.
- `userdata`. 테스트 도구 모음을 실행하는 데 필요한 사용자 데이터 정보가 포함된 파일을 설정합니다. 이는 테스트 도구 모음 `suite.json` 파일에서 `userdataRequired`이(가) true로 설정된 경우에만 필요합니다.

`run-suite` 옵션에 대한 자세한 내용은 다음 `help` 옵션을 사용하십시오.

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

테스트 도구 모음을 디버그 모드에서 실행합니다. 자세한 내용은 IDT를 디버그 모드에서 실행합니다. 섹션을 참조하세요.

IDT 테스트 결과 및 로그 검토

이 섹션에서는 IDT가 콘솔 로그와 테스트 보고서를 생성하는 형식을 설명합니다.

콘솔 메시지 형식

AWS IoT 디바이스 테스터는 테스트 도구 모음을 시작할 때 콘솔에 메시지를 인쇄하는 데 표준 형식을 사용합니다. 다음 발췌문은 IDT에서 생성한 콘솔 메시지의 한 가지 예를 보여줍니다.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

대부분의 콘솔 메시지는 다음 필드로 구성되어 있습니다.

`time`

로깅된 이벤트의 전체 ISO 8601 타임스탬프.

`level`

로깅된 이벤트의 메시지 수준. 일반적으로 로깅된 메시지 수준은 `info`, `warn` 또는 `error`중 하나입니다. IDT는 예상 이벤트가 발생하여 이벤트가 일찍 종료되는 경우 `fatal` 또는 `panic` 메시지를 표시합니다.

`msg`

로깅된 메시지.

`executionId`

현재 IDT 프로세스의 고유 ID 문자열입니다. 이 ID는 개별 IDT 실행을 구분하는 데 사용됩니다.

테스트 제품군에서 생성된 콘솔 메시지는 테스트 대상 디바이스와 테스트 제품군, 테스트 그룹 및 IDT가 실행하는 테스트 케이스에 대한 추가 정보를 제공합니다. 다음 발췌문은 테스트 제품군에서 생성한 콘솔 메시지의 한 가지 예를 보여줍니다:

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

콘솔 메시지의 테스트 제품군 관련 부분에는 다음 필드가 포함됩니다.

`suiteId`

현재 실행 중인 테스트 제품군의 이름.

`groupId`

현재 실행 중인 테스트 그룹의 ID.

`testCaseId`

현재 실행 중인 테스트 케이스의 ID.

`deviceId`

현재 테스트 케이스에서 사용 중인 테스트 대상 디바이스의 ID.

IDT에서 테스트 실행을 완료했을 때 콘솔에 테스트 요약을 인쇄하려면 상태 시스템에 Report 상태를 포함해야 합니다. 테스트 요약에는 테스트 제품군, 실행된 각 그룹의 테스트 결과, 생성된 로그 및 보고서 파일의 위치에 대한 정보가 포함됩니다. 다음 예제에서는 테스트 요약 메시지를 보여줍니다.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
----------------------------------
Test Groups:
```

```
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
---------------------------------
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT 디바이스 테스터 보고서 스키마

`awsiotdevicetester_report.xml`은 다음 정보가 포함된 서명된 보고서입니다.

- IDT 버전
- 테스트 제품군 버전입니다.
- 보고서에 서명하는 데 사용된 보고서 서명 및 키.
- `device.json` 파일에 지정된 SKU 및 디바이스 풀 이름.
- 테스트된 제품 버전 및 디바이스 특성.
- 테스트 결과의 집계 요약 이 정보는 *suite-name*`_report.xml` 파일에 포함된 정보와 동일합니다.

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
```

```
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

awsiotdevicetester_report.xml 파일에는 테스트하는 제품에 대한 정보와 테스트 제품군을 실행한 후 확인된 제품 기능에 대한 정보를 포함하는 <awsproduct> 태그가 포함되어 있습니다.

<awsproduct> 태그에 사용되는 속성

name

테스트하는 제품의 이름입니다.

version

테스트하는 제품의 버전입니다.

features

확인된 기능입니다. 필수로 required 표시된 특성은 테스트 제품군에서 디바이스를 검증하는 데 필요합니다. 다음 코드 조각은 awsiotdevicetester_report.xml 파일에 이 정보가 나타나는 방식을 보여 줍니다.

```
<feature name="ssh" value="supported" type="required"></feature>
```

optional 표시된 특성은 검증에 필요하지 않습니다. 다음 코드 조각은 선택적 기능을 보여 줍니다.

```
<feature name="hsi" value="supported" type="optional"></feature>
```

```
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

테스트 제품군 보고서 스키마

*suite-name*_Result.xml 보고서는 JUnit XML 형식입니다. Jenkins, Bamboo 등과 같은 지속적 통합 및 배포 플랫폼에 이 보고서를 통합할 수 있습니다. 보고서는 테스트 결과의 집계 요약을 포함합니다.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

awsiotdevicetester_report.xml 또는 *suite-name*_report.xml의 보고서 섹션에는 실행된 테스트 및 결과가 나열됩니다.

첫 번째 XML 태그 <testsuites>에는 테스트 실행의 요약이 포함됩니다. 예:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 태그에 사용되는 속성

name

테스트 제품군의 이름입니다.

time

테스트 제품군를 실행하는 데 걸린 시간(초).

tests

실행된 테스트의 수입니다.

failures

실행되었지만 통과하지 못한 테스트의 수입니다.

errors

IDT에서 실행하지 못한 테스트의 수입니다.

disabled

이 속성은 사용되지 않으므로 무시해도 좋습니다.

테스트 실패 또는 오류의 경우 <testsuites> XML 태그를 검토하여 실패한 테스트를 식별할 수 있습니다. <testsuites> 태그 내부의 <testsuite> XML 태그는 테스트 그룹에 대한 테스트 결과 요약을 보여 줍니다. 예:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

형식은 <testsuites> 태그와 비슷하지만, 사용되지 않고 무시할 수 있는 skipped 속성이 있습니다. 각 <testsuite> XML 태그 내부에는 테스트 그룹에 실행된 각 테스트에 대한 <testcase> 태그가 있습니다. 예:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

<testcase> 태그에 사용되는 속성

`name`

테스트의 이름입니다.

`attempts`

IDT에서 테스트 사례를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

IDT 사용량 지표

필요한 권한이 있는 AWS 자격 증명을 제공하면 AWS IoT Device Tester가 사용량 지표를 수집하여 에 AWS제출합니다. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. IDT는 다음과 같은 정보를 수집합니다.

- AWS 계정 IDT를 실행하는 데 사용된 ID
- 테스트 실행에 사용된 IDT CLI 명령
- 실행되는 테스트 제품군
- *< device-tester-extract-location >* 폴더의 테스트 스위트
- 장치 풀에 구성된 장치 수
- 테스트 케이스 이름 및 런타임
- 테스트 결과 정보(예: 테스트 통과, 실패, 오류 발생 또는 건너뛰었는지 여부)
- 제품 기능 테스트
- IDT 종료 행동(예: 예상치 못한 종료 또는 조기 종료)

IDT가 전송하는 모든 정보는 *<device-tester-extract-location>*/results/*<execution-id>*/ 폴더의 metrics.log 파일에도 기록됩니다. 로그 파일을 보면 테스트 실행 중에 수집된 정보를 볼 수 있습니다. 이 파일은 사용량 지표를 수집하도록 선택한 경우에만 생성됩니다.

지표 수집을 비활성화하기 위해 추가 조치를 취할 필요가 없습니다. AWS 자격 증명을 저장하지 말고, 저장된 AWS 자격 증명이 있는 경우 `config.json` 파일에 액세스하도록 구성하지 마십시오.

AWS 자격 증명을 구성하십시오.

아직 계정이 AWS 계정없다면 새로 만들어야 합니다. 이미 계정이 AWS 계정있는 경우 IDT가 사용자 대신 사용량 지표를 전송할 수 있도록 계정에 필요한 권한을 구성하기만 하면 됩니다. AWS

1단계: 생성 AWS 계정

이 단계에는 AWS 계정을 생성하고 구성합니다. 이미 가지고 있다면 AWS 계정다음으로 건너뛰세요. the section called “2단계: IDT에 대한 권한 구성”

가입해 보세요 AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 여세요.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중에는 전화를 받고 키패드로 인증 코드를 입력하는 과정이 있습니다.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스 액세스 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 가서 내 계정(My Account)을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요. AWS 계정 루트 사용자

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

루트 사용자를 사용하여 로그인하는 데 도움이 필요하면AWS 로그인 사용 설명서의 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 참조하십시오.

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 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

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 포털 로그인을 참조하십시오.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.

   지침은AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

2단계: IDT에 대한 권한 구성

이 단계에서는 IDT가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. AWS Management Console 또는 AWS Command Line Interface (AWS CLI) 를 사용하여 IAM 정책과 IDT용 사용자를 생성한 다음 정책을 사용자에게 연결할 수 있습니다.

- IDT에 대한 권한 구성(콘솔)
- IDT에 대한 권한 구성(AWS CLI)

IDT에 대한 권한을 구성하려면(콘솔)

콘솔을 사용하여 AWS IoT Greengrass용 IDT에 대한 권한을 구성하려면 다음 단계를 수행하십시오.

1. IAM 콘솔에 로그인합니다.
2. 특정 권한으로 역할을 생성하는 권한을 부여하는 고객 관리형 정책을 만듭니다.
   a. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
   b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

   c. 다음: 태그를 선택합니다.
   d. 다음: 검토를 선택합니다.
   e. 이름에 **`IDTUsageMetricsIAMPermissions`**를 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.
   f. 정책 생성(Create policy)을 선택합니다.
3. IAM 사용자를 생성하고 사용자에 권한을 연결합니다.

a. IAM 사용자를 생성합니다. IAM 사용 설명서에서 IAM 사용자 생성(콘솔)의 1~5단계를 따르십시오. 이미 IAM 사용자를 생성한 경우 다음 단계로 건너뛰십시오.

b. IAM 사용자에게 권한을 연결합니다.

   i. 권한 설정 페이지에서 기존 정책 직접 연결을 선택합니다.

   ii. 이전 단계에서 생성한 IDT UsageMetrics IAM 권한 정책을 검색하십시오. 확인란을 선택합니다.

c. 다음: 태그를 선택합니다.

d. Next: Review(다음: 검토)를 선택하여 선택 사항의 요약을 봅니다.

e. 사용자 생성을 선택합니다.

f. 사용자의 액세스 키(액세스 키 ID와 비밀 액세스 키)를 보려면 암호와 액세스 키 옆에 있는 Show(표시)를 선택합니다. 액세스 키를 저장하려면 Download .csv(csv 다운로드)를 선택한 후 안전한 위치에 파일을 저장합니다. 나중에 이 정보를 사용하여 자격 증명 파일을 구성합니다. AWS

IDT에 대한 권한을 구성하려면(AWS CLI)

다음 단계에 따라 를 사용하여 IDT에 대한 권한을 구성하십시오. AWS CLI AWS IoT Greengrass콘솔에서 권한을 이미 구성한 경우 the section called “IDT 테스트를 실행하도록 디바이스 구성” 또는 the section called “선택 사항: 도커 컨테이너 구성” 단계로 건너뜁니다.

1. 컴퓨터에 를 설치하고 구성하십시오 (아직 설치되지 않은 AWS CLI 경우). AWS Command Line Interface 사용 설명서 AWS CLI설치 단계를 따르십시오.

   > **Note**
   >
   > 명령줄 셸에서 AWS 서비스와 상호 작용하는 데 사용할 수 있는 오픈 소스 도구입니다. AWS CLI

2. IDT 및 역할을 관리할 권한을 부여하는 다음과 같은 고객 관리형 정책을 생성하십시오. AWS IoT Greengrass

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document
                                          '{\"Version\": \"2012-10-17\",
 \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
tester:SendMetrics\"], \"Resource\": \"*\"}]}'
```

> **Note**
>
> Linux, macOS 또는 Unix 터미널 명령과 다른 JSON 구문을 사용하기 때문에 이 단계에는 Windows 명령 프롬프트 예제가 포함되어 있습니다.

3. IAM 사용자를 만들고 AWS IoT Greengrass용 IDT에 필요한 권한을 연결합니다.

   a. IAM 사용자를 생성합니다.

   ```
   aws iam create-user --user-name user-name
   ```

   b. 생성한 `IDTUsageMetricsIAMPermissions` 정책을 새 IAM 사용자에 연결합니다. *user-name*을 IAM 사용자 이름으로 바꾸고 명령에서 *<account-id>*를 자신이 사용하는 AWS 계정계정의 ID로 바꾸십시오.

```
aws iam attach-user-policy --user-name user-name --policy-arn
 arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. 사용자에 대한 보안 액세스 키를 만듭니다.

   ```
   aws iam create-access-key --user-name user-name
   ```

   출력을 안전한 위치에 저장합니다. 나중에 이 정보를 사용하여 AWS 자격 증명 파일을 구성할 수 있습니다.

IDT에 AWS 자격 증명을 제공하십시오.

IDT가 AWS 자격 증명에 액세스하고 지표를 제출하도록 AWS허용하려면 다음을 수행하십시오.

1. IAM 사용자의 AWS 자격 증명을 환경 변수 또는 자격 증명 파일로 저장합니다.

   a. 다음 명령을 실행하여 환경 변수를 설정합니다.

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   b. 자격 증명 파일을 사용하려면 다음 정보를 `.aws/credentials file:`에 추가합니다.

   ```
   [profile-name]
   aws_access_key_id=access-key
   aws_secret_access_key=secret-access-key
   ```

2. `config.json` 파일의 `auth` 섹션을 구성합니다. 자세한 내용은 (선택 사항) config.json 구성을 (를) 참조하세요.

AWS IoT Greengrass용 IDT 문제 해결

AWS IoT Greengrass용 IDT는 이러한 오류를 해당 오류 유형에 따라 여러 위치에 작성합니다. 오류는 콘솔, 로그 파일 및 테스트 보고서에 작성됩니다.

오류 코드

다음 표에는 AWS IoT Greengrass용 IDT에서 생성되는 오류 코드가 나와 있습니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
101	InternalError	내부 오류가 발생했습니다.	*<device-tester-extract-location>* `/results` 디렉터리에서 로그를 확인합니다. 이 문제를 디버깅할 수 없을 경우 AWS 개발자 지원에 문의하십시오.
102	TimeoutError	제한된 시간 범위 내에 테스트를 완료할 수 없습니다. 이 문제는 다음 경우에 발생할 수 있습니다. • 테스트 시스템과 디바이스 간에 네트워크 연결이 느립니다(예: VPN 네트워크를 사용하는 경우). • 느린 네트워크로 인해 디바이스와 클라우드 간의 통신이 지연됩니다. • 테스트 구성 파일(`test.json`)의 `timeout` 필드가 잘못 수정되었습니다.	• 네트워크 연결과 속도를 확인합니다. • `/test` 디렉터리에 있는 파일을 수정하지 않았는지 확인합니다. • `"--group-id"` 플래그를 사용하여 실패한 테스트 그룹을 수동으로 실행해 봅니다. • 테스트 제한 시간을 늘려 테스트 제품군을 실행해 봅니다. 자세한 내용은 제한 시간 오류 섹션을 참조하세요.
103	PlatformNotSupportError	`device.json` 에 지정된 OS/아키텍처 조합이 잘못되었습니다.	지원되는 조합 중 하나로 구성을 변경합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			• Linux, x86_64 • Linux, ARMv6l • Linux, ARMv7l • Linux, AArch64 • Ubuntu, x86_64 • OpenWRT, ARMv7l • OpenWRT, AArch64 자세한 내용은 device.json 구성 섹션을 참조하세요.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
104	VersionNotSupportError	사용 중인 IDT 버전에서 AWS IoT Greengrass 코어 소프트웨어 버전이 지원되지 않습니다.	device_tester_bin version 명령을 사용하여 지원되는 AWS IoT Greengrass 코어 소프트웨어 버전을 찾습니다. 예를 들어, macOS를 사용하는 경우 ./devicetester_mac_x86_64 version를 사용합니다. 사용 중인 AWS IoT Greengrass 코어 소프트웨어 버전을 찾으려면: • 사전 설치된 AWS IoT Greengrass 코어 소프트웨어로 테스트를 실행 중인 경우 SSH를 사용하여 AWS IoT Greengrass 코어 디바이스에 연결하고 ***<path-to-preinstalled-greengrass-location>*** /greengrass/ggc/core/greengrassd --version을 실행합니다. • 다른 버전의 AWS IoT Greengrass 코

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			어 소프트웨어로 테스트를 실행하는 경우 `devicetester_greengrass_<os>/products/greengrass/gcc` 디렉터리로 이동합니다. AWS IoT Greengrass 코어 소프트웨어 버전이 .zip 파일 이름의 일부입니다. 다른 버전의 AWS IoT Greengrass 코어 소프트웨어를 테스트할 수 있습니다. 자세한 내용은 시작하기 AWS IoT Greengrass 섹션을 참조하세요.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
105	LanguageNotSupportError	IDT는 AWS IoT Greengrass용 Python 라이브러리 및 SDK만 지원합니다.	다음을 확인하십시오. • `devicetester_greengrass_ <os>/products/greengrass/ggsdk` 아래의 SDK 패키지가 Python SDK여야 합니다. • `devicetester_greengrass_ <os> /tests/GGQ_1.0.0/suite/resources/run.runtimefarm/bin` 아래의 bin 폴더 내용이 변경되지 않았는지 확인합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
106	ValidationError	`device.json` 또는 `config.json` 의 일부 필드가 잘못되었습니다.	보고서에서 오류 코드 오른쪽에 표시되는 오류 메시지를 확인합니다. • 디바이스에 대한 인증 유형이 잘못되었습니다. 디바이스에 연결할 올바른 방법을 지정하십시오. 자세한 내용은 the section called “device.json 구성” 섹션을 참조하세요. • 잘못된 프라이빗 키 경로: 프라이빗 키에 올바른 경로를 지정합니다. 자세한 내용은 device.json 구성 섹션을 참조하세요. • 유효하지 않은 AWS 리전: `config.json` 파일에 유효한 AWS 리전을 지정하십시오. 자세한 내용은 AWS 서비스 엔드포인트를 참조하세요. • AWS 자격 증명: 환경 변수 또는 `credentials` 파

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			일을 사용하여 테스트 시스템에서 유효한 AWS 자격 증명을 설정합니다. `auth` 필드가 제대로 구성되었는지 확인합니다. 자세한 내용은 the section called “생성 및 구성 AWS 계정” 섹션을 참조하세요. • HSM 입력이 잘못되었습니다. `device.json` 의 `p11Provider` , `privateKeyLabel` , `slotLabel` , `slotUserPin` 및 `openSSLEngine` 필드를 확인하십시오.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
107	SSHConnectionFailed	테스트 시스템이 구성된 디바이스에 연결할 수 없습니다.	`device.json` 파일에서 다음 필드가 올바른지 확인합니다. • `ip` • `user` • `privKeyPath` • `password` 자세한 내용은 device.json 구성 섹션을 참조하세요.
108	RunCommandError	테스트가 테스트 대상 디바이스에서 명령을 실행하지 못했습니다.	`device.json` 에서 구성된 사용자의 루트 액세스가 허용되는지 확인합니다. 일부 디바이스에서는 루트 액세스로 명령을 실행할 때 암호가 필요합니다. 암호 없이 루트 액세스가 허용되는지 확인합니다. 자세한 내용은 해당 디바이스에 대한 설명서를 참조하십시오. 디바이스에서 실패하는 명령을 수동으로 실행하여 오류가 발생하는지 확인합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
109	PermissionDeniedError	루트 액세스가 없습니다.	디바이스에서 구성된 사용자의 루트 액세스를 설정합니다.
110	CreateFileError	파일을 만들 수 없습니다.	디바이스의 디스크 공간과 디렉터리 권한을 확인합니다.
111	CreateDirError	디렉터리를 만들 수 없습니다.	디바이스의 디스크 공간과 디렉터리 권한을 확인합니다.
112	InvalidPathError	AWS IoT Greengrass 코어 소프트웨어 경로가 잘못되었습니다.	오류 메시지에 있는 경로가 유효한지 확인합니다. `devicetester_greengrass_` *<os>* 디렉터리에 있는 파일을 편집하지 마십시오.
113	InvalidFileError	파일이 잘못되었습니다.	오류 메시지에 있는 파일이 유효한지 확인합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
114	ReadFileError	지정된 로그 파일을 읽을 수 없습니다.	다음을 확인합니다. • 파일 권한이 올바릅니다. • `limits.conf` 가 충분한 파일을 열 수 있도록 허용합니다. • 오류 메시지에 지정된 파일이 존재하며 유효합니다. macOS에서 테스트하는 경우 열린 파일 제한을 늘리십시오. 기본 제한은 256이며, 이 값은 테스트에 충분합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
115	FileNotFoundError	필요한 파일을 찾을 수 없습니다.	다음을 확인합니다. • 압축된 Greengrass 파일은 `devicetester_greengrass_<os>/products/greengrass/ggc` 아래에 존재합니다. AWS IoT Greengrass 코어 소프트웨어 다운로드 페이지에서 AWS IoT Greengrass 코어 tar 파일을 다운로드할 수 있습니다. • SDK 패키지가 `devicetester_greengrass_<os>/products/greengrass/ggsdk`에 있습니다. • `devicetester_greengrass_<os>/tests`에 있는 파일이 수정되지 않았습니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
116	OpenFileFailed	지정된 파일을 열 수 없습니다.	다음을 확인합니다. • 오류 메시지에 지정된 파일이 존재하며 유효합니다. • `limits.conf` 가 충분한 파일을 열 수 있도록 허용합니다. macOS에서 테스트하는 경우 열린 파일 제한을 늘리십시오. 기본 제한은 256이며, 이 값은 테스트에 충분합니다.
117	WriteFileFailed	파일에 쓰지 못했습니다(테스트 중인 디바이스 또는 테스트 시스템일 수 있음).	오류 메시지에 지정된 디렉터리가 존재하며 쓰기 권한이 있는지 확인합니다.
118	FileCleanUpError	테스트가 지정된 파일 또는 디렉터리를 제거하지 못했거나 원격 디바이스에서 지정된 파일을 마운트 해제하지 못했습니다.	이진 파일이 아직 실행 중이면 파일이 잠겨 있을 수 있습니다. 프로세스를 종료하고 지정된 파일을 삭제합니다.
119	InvalidInputError	구성이 잘못되었습니다.	`suite.json` 파일이 유효한지 확인합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
120	InvalidCredentialError	유효하지 않은 AWS 자격 증명.	• AWS 자격 증명을 확인합니다. 자세한 내용은 the section called “AWS 보안 인증 구성” 섹션을 참조하세요. • 네트워크 연결을 확인하고 테스트 그룹을 다시 실행합니다. 네트워크 문제로 인해 이 오류가 발생할 수도 있습니다.
121	AWSSessionError	AWS 세션을 생성하지 못했습니다.	이 오류는 AWS 자격 증명이 잘못되었거나 인터넷 연결이 불안정한 경우에 발생할 수 있습니다. AWS CLI를 사용하여 AWS API 작업을 호출해 봅니다.
122	AWSApiCallError	AWS API 오류가 발생했습니다.	이 오류는 네트워크 문제가 원인일 수 있습니다. 네트워크를 확인하고 테스트 그룹을 다시 시도합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
123	IpNotExistError	연결 정보에 IP 주소가 포함되지 않습니다.	인터넷 연결을 확인하십시오. AWS IoT Greengrass 콘솔을 사용하여 테스트에 사용 중인 AWS IoT Greengrass 코어 사물에 대한 연결 정보를 확인할 수 있습니다. 연결 정보에 엔드포인트 10개가 포함되어 있으면 일부 또는 전부를 제거하고 테스트를 다시 실행할 수 있습니다. 자세한 내용은 연결 정보를 참조하십시오.
124	OTAJobNotCompleteError	OTA 작업이 완료되지 않았습니다.	인터넷 연결을 확인하고 OTA 테스트 그룹을 다시 시도합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
125	CreateGreengrassServiceRoleError	다음 중 하나가 발생했습니다. • 역할을 생성하는 동안 오류가 발생했습니다. • 정책을 AWS IoT Greengrass 서비스 역할에 연결하는 동안 오류가 발생했습니다. • 서비스 역할에 연결된 정책이 잘못되었습니다. • 역할을 AWS 계정과 연결할 때 오류가 발생했습니다.	AWS IoT Greengrass 서비스 역할을 구성합니다. 자세한 내용은 the section called “Greengrass 서비스 역할” 섹션을 참조하세요.
126	DependenciesNotPresentError	해당 테스트에 필요한 하나 이상의 종속성이 디바이스에 없습니다.	테스트 로그 (*<device-tester-extract-location>* /results/ *<execution-id>*/logs/*<test-case-name.log>*) 를 확인하여 디바이스에서 누락된 종속성이 무엇인지 파악합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
127	InvalidHSMConfiguration	제공된 HSM/PKCS 구성 파일이 올바르지 않습니다.	`device.json` 파일에서 PKCS#11을 사용하여 HSM과 상호 작용하는 데 필요한 구성을 제공합니다.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결

128	OTAJobNotSuccededError	OTA 작업이 성공하지 못했습니다.	• ota 테스트 그룹을 개별적으로 실행한 경우 ggcdependencies 테스트 그룹을 실행하여 모든 종속성(예: wget)이 있는지 확인합니다. 그런 다음 ota 테스트 그룹을 다시 시도하십시오.
			• 문제 해결 및 오류 정보는 *<device-tester-extract-location>* / results/ *<execution-id>*/logs/에서 자세한 로그를 확인하십시오. 특히 다음 로그를 확인하십시오.
			  • 콘솔 로그 (test_manager.log)
			  • OTA 테스트 케이스 로그 (ota_test.log)
			  •

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			GGC 대몬 로그(`ota_test_ggc_logs.tar.gz`) • OTA 에이전트 로그(`ota_test_ota_logs.tar.gz`) • 인터넷 연결을 확인하고 `ota` 테스트 그룹을 다시 시도하십시오. • 문제가 지속될 경우 AWS 개발자 지원에 문의하십시오.
129	NoConnectivityError	호스트 에이전트가 인터넷에 연결하지 못합니다.	네트워크 연결 및 방화벽 설정을 확인하십시오. 연결 문제가 해결되면 테스트 그룹을 다시 시도하십시오.
130	NoPermissionError	AWS IoT Greengrass용 IDT를 실행하는 데 사용하는 IAM 사용자가 IDT를 실행하는 데 필요한 AWS 리소스를 생성할 권한이 없습니다.	AWS IoT Greengrass용 IDT를 실행하는 데 필요한 권한을 부여하는 정책 템플릿은 권한 정책 템플릿을 참조하십시오.

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
131	LeftoverAgentExistError	AWS IoT Greengrass용 IDT를 시작하려고 시도할 때 디바이스가 AWS IoT Greengrass 프로세스를 실행합니다.	디바이스에서 실행 중인 기존 Greengrass 대몬(daemon)이 없는지 확인합니다. • sudo ./*<absolute-path-to-greengrass-daemon>* / greengrassd stop 명령을 사용하여 대몬(daemon)을 중지할 수 있습니다. • PID를 기준으로 Greengrass 대몬(daemon)을 종료할 수도 있습니다. **Note** 재부팅 후 자동으로 시작하도록 구성된 AWS IoT Greengrass의 기존 설치를 사용하는 경우 재부팅 후 테스트 제품군을 실행하기 전에 대몬(daemon

오류 코드	오류 코드 이름	가능한 근본 원인	문제 해결
			)을 중지해야 합니다.
132	DeviceTimeOffsetError	디바이스의 시간이 잘못되었습니다.	디바이스를 올바른 시간으로 설정하세요.
133	InvalidMLConfiguration	제공된 ML 구성이 잘못되었습니다.	`device.json` 파일에서 ML 추론 테스트를 실행하는 데 필요한 올바른 구성을 제공하세요. 자세한 내용은 the section called “선택 사항: ML 검증을 위해 디바이스 구성” 섹션을 참조하세요.

AWS IoT Greengrass용 IDT 오류 해결

IDT를 사용할 때는 AWS IoT Greengrass용 IDT를 실행하기 전에 올바른 구성 파일을 준비해야 합니다. 구문 분석 및 구성 오류가 발생할 경우 첫 번째 단계는 환경에 적합한 구성 템플릿을 찾아서 사용하는 것입니다.

그래도 문제가 발생할 경우 다음 디버깅 프로세스를 참조하십시오.

주제

- 오류는 어디서 찾을 수 있나요?
- 구문 분석 오류
- 필수 파라미터 누락 오류
- 테스트를 시작할 수 없음 오류
- 리소스 액세스 권한 없음 오류
- 권한 거부 오류
- SSH 연결 오류
- 제한 시간 오류

- 테스트 도중 명령을 찾을 수 없음 오류
- macOS에서의 보안 예외

오류는 어디서 찾을 수 있나요?

상위 수준의 오류는 실행하는 동안 콘솔에 표시되고, 오류와 함께 실패한 테스트 요약은 모든 테스트가 완료될 때 표시됩니다. `awsiotdevicetester_report.xml`에 테스트 실패의 원인이 되는 모든 오류에 대한 요약이 들어 있습니다. 각 테스트 실행에 대한 로그 파일은 테스트를 실행하는 동안 콘솔에 표시된 테스트 실행에 대한 이름이 UUID인 디렉터리에 저장됩니다.

테스트 로그 디렉터리는 *<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`에 있습니다. 이 디렉터리에는 디버깅에 유용한 다음 파일이 포함되어 있습니다.

파일	설명
`test_manager.log`	테스트를 실행하는 동안 콘솔에 작성된 모든 로그입니다. 결과 요약은 실패한 테스트 목록을 포함하는 이 파일의 끝에 있습니다. 이 파일의 경고 및 오류 로그에서 실패에 대한 일부 정보를 확인할 수 있습니다.
<test-group-id> `__`*<test-name>* `.log`	특정 테스트에 대한 상세 로그입니다.
<test-name> _ggc_logs.tar.gz	AWS IoT Greengrass 코어 대몬(daemon)이 테스트 중 생성한 모든 로그의 압축 모음입니다. 자세한 내용은 문제 해결AWS IoT Greengrass을 참조하세요.
<test-name> `_ota_logs.tar.gz`	테스트 중 AWS IoT Greengrass OTA 에이전트에 의해 생성된 로그의 압축 모음입니다. OTA 테스트만 해당됩니다.
<test-name> `_basic_assertion_publisher_ggad_logs.tar.gz`	테스트 중 AWS IoT 게시자 디바이스에 의해 생성된 로그의 압축 모음입니다.
<test-name> `_basic_assertion_subscriber_ggad_logs.tar.gz`	테스트 중 AWS IoT 구독자 디바이스에 의해 생성된 로그의 압축 모음입니다.

구문 분석 오류

경우에 따라 JSON 구성의 오타로 인해 구문 분석 오류가 발생할 수 있습니다. 대부분의 경우 문제는 JSON 파일에서 대괄호, 쉼표 또는 따옴표를 생략한 결과입니다. IDT는 JSON 확인을 수행하고 디버깅 정보를 인쇄합니다. IDT는 오류가 발생한 줄, 줄 번호, 구문 오류의 열 번호를 인쇄합니다. 이 정보만 있으면 오류를 수정할 수 있지만, 여전히 오류를 찾을 수 없는 경우 IDE나 텍스트 편집기(예: Atom 또는 Sublime)에서 또는 온라인 도구(예: JSONLint)를 통해 수동으로 확인을 수행할 수 있습니다.

필수 파라미터 누락 오류

IDT에 새로운 기능이 추가되고 있으므로 구성 파일에 대한 변경 사항이 도입될 수 있습니다. 기존 구성 파일을 사용하면 구성이 손상될 수 있습니다. 이 문제가 발생할 경우 `/results/<execution-id>/logs` 아래의 `<test_case_id>.log` 파일에 누락된 모든 파라미터가 명시적으로 나열됩니다. 또한 IDT는 JSON 구성 파일 스키마를 검사하여 지원되는 최신 버전이 사용되었는지 확인합니다.

테스트를 시작할 수 없음 오류

테스트 시작 중에 실패를 가리키는 오류가 발생할 수 있습니다. 몇 가지 원인이 있을 수 있으므로 다음을 수행하십시오.

- 실행 명령에 포함한 풀 이름이 실제로 존재하는지 확인합니다. 풀 이름은 `device.json` 파일에서 직접 참조됩니다.
- 풀에 있는 디바이스에 올바른 구성 파라미터가 있는지 확인합니다.

리소스 액세스 권한 없음 오류

터미널 출력 또는 `/results/<execution-id>/logs`아래 `test_manager.log` 파일에 `<user or role> is not authorized to access this resource` 오류 메시지가 나타날 수 있습니다. 이 문제를 해결하려면 `AWSIoTDeviceTesterForGreengrassFullAccess` 관리형 정책을 테스트 사용자에게 연결합니다. 자세한 내용은 the section called “생성 및 구성 AWS 계정” 섹션을 참조하세요.

권한 거부 오류

IDT는 테스트 대상 디바이스에서 다양한 디렉터리와 파일에 대해 작업을 수행합니다. 이러한 작업 일부에는 루트 액세스가 필요합니다. 이러한 작업을 자동화하기 위해서는 IDT가 암호 입력 없이 sudo를 사용하여 명령을 실행할 수 있어야 합니다.

암호 입력 없이 sudo 액세스를 허용하려면 다음 단계를 수행합니다.

> **Note**
>
> `user` 및 `username`은 IDT가 테스트 대상 디바이스에 액세스하는 데 사용하는 SSH 사용자를 나타냅니다.

1. sudo 그룹에 SSH 사용자를 추가하려면 sudo usermod -aG sudo ***<ssh-username>***을 사용하십시오.
2. 변경 사항을 적용하려면 로그아웃했다가 로그인하십시오.
3. `/etc/sudoers` 파일을 열고 파일 끝에 다음 줄을 추가합니다. *<ssh-username>* `ALL=(ALL) NOPASSWD: ALL`

   > **Note**
   >
   > 모범 사례로, `/etc/sudoers`를 편집할 때는 sudo visudo를 사용하는 것이 좋습니다.

SSH 연결 오류

IDT가 테스트 대상 디바이스에 연결할 수 없으면 연결 실패가 `/results/`*<execution-id>*`/logs/`*<test-case-id>*`.log`에 기록됩니다. 테스트 대상 디바이스에 대한 연결은 IDT가 수행하는 첫 번째 작업 중 하나이므로 SSH 실패 메시지는 이 로그 파일의 맨 위에 표시됩니다.

대부분의 Windows 설정은 PuTTy 터미널 애플리케이션을 사용하여 Linux 호스트에 연결합니다. 이 애플리케이션에서는 표준 PEM 프라이빗 키 파일을 PPK라는 전용 Windows 형식으로 변환해야 합니다. `device.json` 파일에서 IDT가 구성되면 PEM 파일만 사용합니다. PPK 파일을 사용하면 IDT가 AWS IoT Greengrass 디바이스에 대한 SSH 연결을 생성할 수 없으며 테스트를 실행할 수 없습니다.

제한 시간 오류

각 테스트의 제한 시간 기본값에 적용되는 제한 시간 승수를 지정하여 각 테스트에 대한 제한 시간을 늘릴 수 있습니다. 이 플래그에 대해 구성된 값은 1.0보다 크거나 같아야 합니다.

제한 시간 승수를 사용하려면 테스트를 실행할 때 `--timeout-multiplier` 플래그를 사용합니다. 예:

```
./devicetester_linux run-suite --suite-id GGQ_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

자세한 내용을 보려면 `run-suite --help`를 실행하십시오.

테스트 도중 명령을 찾을 수 없음 오류

AWS IoT Greengrass 디바이스에서 테스트를 실행하려면 이전 버전의 OpenSSL 라이브러리(libssl1.0.0)가 필요합니다. 대부분의 최신 Linux 배포는 libssl 버전 1.0.2 이상(v1.1.0)을 사용합니다.

예를 들어 Raspberry Pi에서 다음 명령을 실행하여 필요한 libssl 버전을 설치합니다.

1.
```
wget http://ftp.us.debian.org/debian/pool/main/o/openssl/
libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```
2.
```
sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```

macOS에서의 보안 예외

macOS 10.15를 사용하는 호스트 컴퓨터에서 IDT를 실행하면 IDT에 대한 공증 티켓이 제대로 감지되지 않고 IDT 실행이 차단됩니다. IDT를 실행하려면 `devicetester_mac_x86-64` 실행 파일에 보안 예외를 부여해야 합니다.

IDT 실행 파일에 보안 예외를 부여하려면

1. Apple 메뉴에서 시스템 기본 설정을 실행합니다.
2. 보안 및 개인 정보 보호를 선택한 다음 일반 탭에서 잠금 아이콘을 클릭하여 보안 설정을 변경합니다.
3. `"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.` 메시지를 찾아 모두 허용을 선택합니다.
4. 보안 경고를 수락합니다.

IDT 지원 정책에 대해 궁금한 점이 있는 경우 AWS 고객 지원 센터에 문의하십시오.

AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터에 대한 지원 정책

AWS IoT AWS IoT Greengrass에 대한 디바이스 테스터(IDT)는 다운로드 가능한 테스트 프레임워크로, AWS Partner 장치 카탈로그에 포함시킬 AWS IoT Greengrass 장치를 검증하고 자격을 부여할 수

있습니다. 최신 버전의 AWS IoT Greengrass 및 IDT를 사용하여 디바이스를 테스트하고 자격을 부여하는 것이 좋습니다. 자세한 내용은 AWS IoT Greengrass Version 2 개발자 안내서의 지원되는 AWS IoT Greengrass V2 IDT 버전을 참조하세요.

AWS IoT Greengrass 및 IDT의 지원되는 버전 중 하나를 사용하여 디바이스를 테스트하고 자격을 부여할 수 있습니다. IDT의 지원되지 않는 버전을 계속 사용할 수 있지만, 이러한 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다.

> **Important**
>
> 2022년 4월 4일부터 AWS IoT Greengrass V1용 AWS IoT 디바이스 테스터(IDT)는 더 이상 서명된 자격 보고서를 생성하지 않습니다. 더 이상 AWS 디바이스 검증 프로그램을 통해 AWS Partner 디바이스 카탈로그에 새 AWS IoT Greengrass V1 디바이스를 등록할 자격을 부여할 수 없습니다. Greengrass V1 디바이스를 검증할 수는 없지만 계속해서 AWS IoT Greengrass V1용 IDT를 사용하여 Greengrass V1 디바이스를 테스트할 수 있습니다. AWS Partner 디바이스 카탈로그에서 Greengrass 디바이스를 검증하고 등재하려면 AWS IoT Greengrass V2용 IDT를 사용하는 것이 좋습니다.

지원 정책에 대해 궁금한 점이 있는 경우 AWS 고객 지원 센터에 문의하십시오.

AWS IoT Greengrass 문제 해결

이 단원에서는 AWS IoT Greengrass 관련 문제를 해결하는 데 도움이 되는 문제 해결 정보와 가능한 해결책을 제공합니다.

AWS IoT Greengrass 할당량(한도)에 대한 자세한 내용은 Amazon Web Services 일반 참조의 서비스 할당량을 참조하세요.

AWS IoT Greengrass 코어 문제

AWS IoT Greengrass 코어 소프트웨어가 시작되지 않는 경우, 다음과 같은 일반 문제 해결 단계를 시도합니다.

- 아키텍처에 적합한 바이너리를 설치했는지 확인합니다. 자세한 내용은 AWS IoT Greengrass 코어 소프트웨어를 참조하십시오.
- 코어 디바이스가 로컬 스토리지를 사용할 수 있는지 확인합니다. 자세한 설명은 the section called “스토리지 문제 해결” 섹션을 참조하세요.
- `runtime.log` 및 `crash.log`에서 오류 메시지가 있는지 확인합니다. 자세한 설명은 the section called “로그 문제 해결” 섹션을 참조하세요.

다음 증상과 오류를 검색해 AWS IoT Greengrass 코어로 관련 문제 해결에 도움이 되는 정보를 찾습니다.

문제

- 오류: 구성 파일에 CaPath, CertPath 또는 가 없습니다 KeyPath. The Greengrass daemon process with [pid = <pid>] died.
- 오류: Failed to parse /<greengrass-root>/config/config.json.
- 오류: TLS 구성을 생성하는 동안 오류가 발생했습니다: URIScheme ErrUnknown
- 오류: Runtime failed to start: unable to start workers: container test timed out.
- <address>오류: 로컬 Cloudwatch, LogGroup:/GreengrassSystem/connection_manager, 오류: 전송 요청 실패 원인: 포스트 http://<path>/cloudwatch/logs/ RequestError: 다이얼 tcp: PutLogEvents getsockopt: 연결 거부, 응답: {}.
- 오류: 다음 이유로 인해 서버를 생성할 수 없습니다. 그룹을 로드하지 못했습니다: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: no such file or directory.

- 컨테이너화 없이 실행하는 방식에서 Greengrass 컨테이너에서 실행하는 방식으로 변경한 후 AWS IoT Greengrass 코어 소프트웨어가 시작되지 않습니다.
- 오류: Spool size should be at least 262144 bytes.
- 오류: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"}
- 오류: container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"".
- 오류: Greengrass daemon running with PID: <process-id>. Some system components failed to start. Check 'runtime.log' for errors.
- 디바이스 섀도우가 클라우드와 동기화하지 않습니다.
- 오류: unable to accept TCP connection. accept tcp [::]:8000: accept4: too many open files.
- 오류: Runtime execution error: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".
- 경고: [WARN] - [5] GK Remote: 공개 키 데이터를 검색하는 중 오류가 발생했습니다.: 개인 키가 설정되지 않았습니다. ErrPrincipalNotConfigured MqttCertificate
- 오류: 역할을 사용하려고 할 때 권한이 거부되었습니다.<arn:aws:iam::<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.
- AWS IoT Greengrass 코어가 네트워크 프록시를 사용하도록 구성되었으므로 Lambda 함수는 나가는 연결을 만들 수 없습니다.
- 코어가 무한 연결-연결 해제 루프에 있습니다. runtime.log 파일에 연속적인 연결 및 연결 해제 항목 시리즈가 포함되어 있습니다.
- 오류: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \\\"proc\\\" to rootfs \\\"
- 오류: 런타임 실행 오류: Lambda 컨테이너를 시작할 수 없습니다. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}
- [ERROR]-배포 실패. {"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed: container process state: exit status 1"}

- 오류: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}
- Error: [DEBUG]-Failed to get routes. Discarding message.
- 오류: [Errno 24] <lambda-function>이 너무 많이 열려 있습니다.[Errno 24] 열린 파일이 너무 많습니다.
- Error: ds server failed to start listening to socket: listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: bind: invalid argument
- [정보] (복사기) aws.greengrass. StreamManager: 스타드아웃. 원인: com.fasterxml.jackson.databind. JsonMappingException: 인스턴트가 최소 또는 최대 인스턴트 시간을 초과했습니다.
- GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

오류: 구성 파일에 CaPath, CertPath 또는 가 없습니다 KeyPath. The Greengrass daemon process with [pid = <pid>] died.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 `crash.log`에서 이 오류를 볼 수 있습니다. 이는 v1.6 이전 버전을 실행하는 경우에 발생할 수 있습니다. 다음 중 하나를 수행합니다.

- v1.7 이상으로 업그레이드. 항상 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 실행하는 것이 좋습니다. 다운로드 정보는 AWS IoT Greengrass 코어 소프트웨어를 참조하십시오.
- AWS IoT Greengrass 코어 소프트웨어 버전에 대해 올바른 `config.json` 형식을 사용합니다. 자세한 설명은 the section called “AWS IoT Greengrass 코어 구성 파일” 섹션을 참조하세요.

> **Note**
>
> 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전이 무엇인지 알아보려면 디바이스 터미널에서 다음 명령을 실행합니다.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd --version
```

오류: Failed to parse /<greengrass-root>/config/config.json.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. Greengrass 구성 파일이 올바른 JSON 형식을 사용 중인지 확인합니다.

`config.json`(*/greengrass-root*`/config`에 위치)을 열고 JSON 형식을 검증합니다. 예를 들어, 쉼표가 올바르게 사용되었는지 확인합니다.

오류: TLS 구성을 생성하는 동안 오류가 발생했습니다: URIScheme ErrUnknown

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. Greengrass 구성 파일의 crypto 섹션에 있는 속성이 유효한지 확인합니다. 오류 메시지는 더 많은 정보를 제공합니다.

`config.json`(*/greengrass-root*`/config`에 위치)을 열고 `crypto` 섹션을 확인합니다. 예를 들어 인증서와 키 경로는 올바른 URI 형식을 사용하여 올바른 위치를 가리켜야 합니다.

오류: Runtime failed to start: unable to start workers: container test timed out.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. Greengrass 구성 파일에서 `postStartHealthCheckTimeout` 속성을 설정합니다. 이 선택적 속성은 Greengrass 데몬이 상태 확인이 시작 후 완료될 때까지 대기하는 시간(밀리초)을 구성합니다. 기본값은 30초(30000 밀리초)입니다.

`config.json`(*/greengrass-root*`/config`에 위치)을 엽니다. `runtime` 객체에서 `postStartHealthCheckTimeout` 속성을 추가하고 해당 값을 30000을 초과하는 값으로 설정합니다. 유효한 JSON 문서를 생성하는 데 필요한 경우 쉼표를 추가합니다. 예:

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

<address>오류: 로컬 Cloudwatch, LogGroup:/GreengrassSystem/connection_manager, 오류: 전송 요청 실패 원인: 포스트 http://<path>/cloudwatch/logs/ RequestError: 다이얼 tcp: PutLogEvents getsockopt: 연결 거부, 응답: {}.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. 이는 Raspberry Pi에서 AWS IoT Greengrass를 실행하고 필요한 메모리 설정이 완료되지 않은 경우 발생할 수 있습니다. 자세한 정보는 이 단계를 참조하십시오.

오류: 다음 이유로 인해 서버를 생성할 수 없습니다. 그룹을 로드하지 못했습니다: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: no such file or directory.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. 실행 가능한 Lambda를 코어에 배포했다면 `group.json` 파일 (/*greengrass-root*/ggc/deployment/group에 위치)에서 함수의 `Handler` 속성을 확인합니다. 핸들러가 컴파일된 실행 파일의 정확한 이름이 아닌 경우에는 `group.json` 파일의 내용을 빈 JSON 객체(`{}`)로 바꾸고, 다음 명령을 실행해 AWS IoT Greengrass를 시작합니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

그런 다음,AWS Lambda API를 사용하여 구성의 `handler` 파라미터를 업데이트하고 새 함수 버전을 게시하고 별칭을 업데이트합니다. 자세한 내용은 AWS Lambda 함수 버전 및 별칭을 참조하십시오.

별칭을 기준으로 Greengrass 그룹에 함수를 추가했다고 가정한다면(권장 사항) 이제 그룹을 다시 배포할 수 있습니다 (그렇지 않은 경우에는 그룹을 배포하기 전에 그룹 정의 및 구독에서 새 함수 버전이나 별칭을 가리켜야 함).

컨테이너화 없이 실행하는 방식에서 Greengrass 컨테이너에서 실행하는 방식으로 변경한 후 AWS IoT Greengrass 코어 소프트웨어가 시작되지 않습니다.

해결책: 누락된 컨테이너 종속성이 없는지 확인합니다.

오류: Spool size should be at least 262144 bytes.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. `group.json` 파일(*`greengrass-root`*`/ggc/deployment/group`에 위치)을 열고, 파일 내용을 빈 JSON 객체(`{}`)로 바꾸고, 다음 명령을 실행해 AWS IoT Greengrass를 시작합니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

그런 다음 the section called “로컬 스토리지의 캐시에 메시지를 저장하는 방법” 절차에서 해당 단계를 따릅니다. `GGCloudSpooler` 함수의 경우 262144보다 크거나 같은 `GG_CONFIG_MAX_SIZE_BYTES` 값을 지정해야 합니다.

오류: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"}

해결책: Greengrass 코어가 MQTT 메시지를 AWS IoT Core에 보낼 수 없을 경우 `GGCloudSpooler.log`에 이 오류가 표시될 수 있습니다. 이 문제는 코어 환경의 대역폭이 제한되고 지연 시간이 긴 경우에 발생할 수 있습니다. AWS IoT Greengrass v1.10.2 이상을 실행하는 경우

config.json 파일에서 mqttOperationTimeout 값을 늘려 보십시오. 속성이 없으면 coreThing 객체에 추가합니다. 예:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

기본값은 5이고 최소값은 5입니다.

오류: container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"".

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 runtime.log에서 이 오류를 볼 수 있습니다. 이 오류는 umask가 0022보다 큰 경우에 발생합니다. 이 문제를 해결하려면 umask를 0022 이하로 설정해야 합니다. 0022 값을 설정하면 모든 사람에게 새 파일에 대한 읽기 권한이 기본적으로 부여됩니다.

오류: Greengrass daemon running with PID: <process-id>. Some system components failed to start. Check 'runtime.log' for errors.

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. runtime.log 및 crash.log에서 특정 오류 정보를 확인합니다. 자세한 설명은 the section called "로그 문제 해결" 섹션을 참조하세요.

디바이스 섀도우가 클라우드와 동기화하지 않습니다.

해결 방법: AWS IoT Greengrass에 Greengrass 서비스 역할의 `iot:UpdateThingShadow` 및 `iot:GetThingShadow` 작업에 대한 권한이 있는지 확인하십시오. 서비스 역할이 `AWSGreengrassResourceAccessRolePolicy` 관리형 정책을 사용하는 경우 이러한 권한은 기본적으로 포함됩니다.

섀도우 동기화 제한 시간 문제 해결 섹션을 참조하세요.

오류: unable to accept TCP connection. accept tcp [::]:8000: accept4: too many open files.

해결책: `greengrassd` 스크립트 출력에서 이 오류가 표시될 수 있습니다. 이는 AWS IoT Greengrass 코어 소프트웨어에 대한 파일 서술자 한도가 임계값에 도달했기 때문에 이 한도를 높여야 하는 경우에 발생할 수 있습니다.

다음 명령을 사용하고 AWS IoT Greengrass 소프트웨어를 다시 시작합니다.

```
ulimit -n 2048
```

> **Note**
>
> 이 예제에서 한도는 2048로 증가합니다. 사용 사례에 적합한 하나의 값을 선택합니다.

오류: Runtime execution error: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".

해결책: 루트 디렉터리 아래에 AWS IoT Greengrass를 직접 설치하거나 AWS IoT Greengrass 코어 소프트웨어가 설치된 디렉터리와 그 상위 디렉터리에 모두에 대한 `execute` 권한이 있는지 확인해야 합니다.

경고: [WARN] - [5] GK Remote: 공개 키 데이터를 검색하는 중 오류가 발생했습니다.: 개인 키가 설정되지 않았습니다. ErrPrincipalNotConfigured MqttCertificate

해결책: AWS IoT Greengrass는 일반 핸들러를 사용하여 모든 보안 주체의 속성을 검증합니다. 로컬 MQTT 서버에 대한 사용자 지정 프라이빗 키를 지정하지 않은 경우 `runtime.log`에서 이러한 경고가 발생합니다. 자세한 설명은 the section called “보안 주체” 섹션을 참조하세요.

오류: 역할을 사용하려고 할 때 권한이 거부되었습니다.<arn:aws:iam::<account-id>:role/<role-name> to access s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz.

해결 방법: over-the-air (OTA) 업데이트 실패 시 이 오류가 표시될 수 있습니다. 서명자 역할 정책에서 대상 AWS 리전을 `Resource`로서 추가합니다. 이 서명자 역할은 AWS IoT Greengrass 소프트웨어 업데이트의 S3 URL을 미리 서명하는 데 사용됩니다. 자세한 내용은 S3 URL 서명자 역할을 참조하십시오.

AWS IoT Greengrass 코어가 네트워크 프록시를 사용하도록 구성되었으므로 Lambda 함수는 나가는 연결을 만들 수 없습니다.

해결책: Lambda 함수가 연결을 생성하는 데 사용하는 런타임 및 실행 파일에 따라 연결 시간 초과 오류가 표시될 수도 있습니다. Lambda 함수가 적절한 프록시 구성을 사용하여 네트워크 프록시를 통해 연결되는지 확인하십시오. AWS IoT Greengrass는 `http_proxy`, `https_proxy`, 및 `no_proxy` 환경 변수를 통해 사용자 정의 Lambda 함수에 프록시 구성을 전달합니다. 이러한 환경 변수는 다음 Python 코드와 같이 액세스할 수 있습니다.

```
import os
print(os.environ['http_proxy'])
```

환경에 정의된 변수와 동일한 대/소문자를 사용합니다(예: 모두 소문자 `http_proxy` 또는 모두 대문자 `HTTP_PROXY`). 이러한 변수의 경우 AWS IoT Greengrass는 둘 다 지원합니다.

> **Note**
>
> boto3 또는 cURL 및 python `requests` 패키지와 같은 연결을 만드는 데 사용하는 대부분의 공통 라이브러리는 이러한 환경 변수를 기본값으로 사용합니다.

코어가 무한 연결-연결 해제 루프에 있습니다. runtime.log 파일에 연속적인 연결 및 연결 해제 항목 시리즈가 포함되어 있습니다.

해결책: 다른 디바이스가 코어 사물 이름을 AWS IoT에 대한 MQTT 연결에 클라이언트 ID로 사용하도록 하드코딩되는 경우 이 상태가 발생할 수 있습니다. 동일한 AWS 리전 및 AWS 계정에서 동시에 연결하려면 고유의 클라이언트 ID를 사용해야 합니다. 기본적으로 코어는 코어 사물 이름을 이러한 연결에 클라이언트 ID로 사용합니다.

이 문제를 해결하려면 다른 디바이스에서 연결에 사용하는 클라이언트 ID를 변경(권장)하거나 코어의 기본값을 재정의할 수 있습니다.

코어 디바이스의 기본 클라이언트 ID를 재정의하려면

1. 다음 명령을 실행해 Greengrass 대몬(daemon)을 중단합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd stop
   ```

2. su 사용자로서 편집을 하려면 *greengrass-root*`/config/config.json`을 엽니다.
3. `coreThing` 객체에서 `coreClientId` 속성을 추가하고 이 값을 사용자 지정 클라이언트 ID로 설정합니다. 값은 1~128자 이내로 작성해야 합니다. AWS 계정의 현재 AWS 리전에서는 고유해야 합니다.

   ```
   "coreClientId": "MyCustomClientId"
   ```

4. 대몬(daemon)을 시작합니다.

   ```
   cd /greengrass-root/ggc/core/
   sudo ./greengrassd start
   ```

오류: unable to start lambda container. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \\\"proc\\\" to rootfs \\\"

해결책: 일부 플랫폼에서 AWS IoT Greengrass가 `/proc` 파일 시스템을 탑재하여 Lambda 컨테이너를 생성하도록 시도할 때 `runtime.log`에 이 오류가 표시될 수 있습니다. 또는 `operation not permitted`나 `EPERM`과 비슷한 오류가 표시될 수 있습니다. 플랫폼에서 실행되는 테스트가 종속성 확인 스크립트에서 통과되는 경우에도 이러한 오류가 발생할 수 있습니다.

다음과 같은 가능한 해결 방안 중 하나를 수행하십시오.

- Linux 커널에서 `CONFIG_DEVPTS_MULTIPLE_INSTANCES` 옵션을 활성화합니다.
- 호스트에 대한 `/proc` 탑재 옵션을 `rw,relatim` 전용으로 설정합니다.
- Linux 커널을 4.9 이상으로 업그레이드합니다.

> **Note**
>
> 이 문제는 로컬 리소스 액세스를 위한 `/proc` 탑재와는 관련이 없습니다.

오류: 런타임 실행 오류: Lambda 컨테이너를 시작할 수 없습니다. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}

해결책: 배포에 실패하면 runtime.log에서 이 오류가 발생할 수 있습니다. AWS IoT Greengrass 그룹의 Lambda 함수가 코어의 파일 시스템에 있는 `/usr` 디렉터리에 액세스할 수 없는 경우 이 오류가 발생합니다.

이 문제를 해결 하려면 그룹에 로컬 볼륨 리소스를 추가하고 그룹을 배포합니다. 리소스는 다음 조건을 충족해야 합니다.

- `/usr`을 소스 경로 및 대상 경로로 지정
- 리소스를 소유한 Linux 그룹의 OS 그룹 권한을 자동으로 추가
- Lambda 함수와 연결하고 읽기 전용 액세스 허용

[ERROR]-배포 실패. {"deploymentId": "<deployment-id>", "errorString": "container test process with pid <pid> failed: container process state: exit status 1"}

해결책: 배포에 실패하면 runtime.log에서 이 오류가 발생할 수 있습니다. AWS IoT Greengrass 그룹의 Lambda 함수가 코어의 파일 시스템에 있는 `/usr` 디렉터리에 액세스할 수 없는 경우 이 오류가 발생합니다.

`GGCanary.log`에 추가 오류가 있는지 확인하여 이러한 사실을 확인할 수 있습니다. Lambda 함수가 `/usr` 디렉터리에 액세스할 수 없는 경우 `GGCanary.log`에는 다음과 같은 오류가 포함되어 있습니다.

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or
 directory"
```

이 문제를 해결 하려면 그룹에 로컬 볼륨 리소스를 추가하고 그룹을 배포합니다. 리소스는 다음 조건을 충족해야 합니다.

- `/usr`을 소스 경로 및 대상 경로로 지정
- 리소스를 소유한 Linux 그룹의 OS 그룹 권한을 자동으로 추가
- Lambda 함수와 연결하고 읽기 전용 액세스 허용

오류: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 `runtime.log` 파일에서 이 오류를 볼 수 있습니다. 이 문제는 Debian 운영 체제에서 더 일반적으로 발생할 수 있습니다.

이 문제를 해결하려면 다음과 같이 실행합니다.

1. AWS IoT Greengrass Core 소프트웨어를 v1.9.3 이상으로 업그레이드하십시오. 문제가 자동으로 해결됩니다.
2. AWS IoT Greengrass Core 소프트웨어를 업그레이드한 후에도 이 오류가 계속 발생하는 경우 config.json 파일에서 `system.useOverlayWithTmpfs` 속성을 `true`(으)로 설정하십시오.

   Example 예

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

> **Note**
>
> 오류 메시지에 AWS IoT Greengrass 코어 소프트웨어 버전이 표시됩니다. Linux 커널 버전을 찾으려면 `uname -r`을 실행합니다.

Error: [DEBUG]-Failed to get routes. Discarding message.

해결책: 그룹에서 구독을 확인하고, `[DEBUG]` 메시지에 나열된 구독이 존재하는지 확인합니다.

오류: [Errno 24] <lambda-function>이 너무 많이 열려 있습니다.[Errno 24] 열린 파일이 너무 많습니다.

해결책: 함수가 함수 핸들러에서 를 인스턴스화되면 Lambda 함수 로그 파일에 `StreamManagerClient` 이 오류가 표시될 수 있습니다. 처리기 외부에서 클라이언트를 만드는 것이 좋습니다. 자세한 설명은 the section called “StreamManagerClient를 사용하여 스트림 작업” 섹션을 참조하세요.

Error: ds server failed to start listening to socket: listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: bind: invalid argument

해결책: AWS IoT Greengrass 코어 소프트웨어가 시작되지 않을 때 이 오류가 발생할 수 있습니다. 이 오류는 파일 경로가 긴 폴더에 AWS IoT Greengrass Core 소프트웨어를 설치했을 때 발생합니다. 쓰기 디렉터리를 사용하지 않는 경우 파일 경로가 79바이트 미만이고, 쓰기 디렉터리를 사용하는 경우에는 83바이트 미만인 폴더에 AWS IoT Greengrass Core 소프트웨어를 다시 설치하시기 바랍니다.

[정보] (복사기) aws.greengrass. StreamManager: 스타드아웃. 원인: com.fasterxml.jackson.databind. JsonMappingException: 인스턴트가 최소 또는 최대 인스턴트 시간을 초과했습니다.

AWS IoT Greengrass 코어 소프트웨어를 v1.11.3으로 업그레이드할 때 스트림 관리자가 시작되지 않으면 스트림 관리자 로그에 다음 오류가 표시될 수 있습니다.

```
2021-07-16T00:54:58.568Z [INFO] (Copier) aws.greengrass.StreamManager:
 stdout. Caused by: com.fasterxml.jackson.databind.JsonMappingException:
 Instant exceeds minimum or maximum instant (through reference chain:
 com.amazonaws.iot.greengrass.streammanager.export.PersistedSuccessExportStatesV1["lastExportTi
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
2021-07-16T00:54:58.579Z [INFO] (Copier) aws.greengrass.StreamManager: stdout.
 Caused by: java.time.DateTimeException: Instant exceeds minimum or maximum instant.
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
```

v1.11.3 이전의 AWS IoT Greengrass 코어 소프트웨어 버전을 사용 중이고 이를 이후 버전으로 업그레이드하려면 OTA 업데이트를 사용하여 v1.11.4로 업그레이드하십시오.

GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

APT 리포지토리에서 AWS IoT Greengrass 코어 소프트웨어를 설치한 디바이스에서 `apt update`을(를) 실행하면 다음 오류가 표시될 수 있습니다.

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
  The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
 Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
 signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```

이 오류는 AWS IoT Greengrass이(가) APT 저장소에서 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 업데이트하는 옵션을 더 이상 제공하지 않기 때문에 발생합니다. `apt update`을(를) 성공적으로 실행하려면 디바이스의 소스 목록에서 AWS IoT Greengrass 리포지토리를 제거하세요.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

배포 관련 문제

다음 정보를 사용하면 배포 문제를 해결하는 데 도움이 됩니다.

문제

- 현재 배포가 작동하지 않으며 이전의 작동하는 배포로 되돌려야 합니다.
- 배포 시 403 Forbidden 오류가 로그에 나타납니다.
- create-deployment 명령을 처음 실행하면 ConcurrentDeployment 오류가 발생합니다.
- 오류: Greengrass is not authorized to assume the Service Role associated with this account 또는 오류: Failed: TES service role is not associated with this account.
- 오류: unable to execute download step in deployment. error while downloading: error while downloading the Group definition file: ... x509: certificate has expired or is not yet valid
- 배포가 완료되지 않습니다.
- 오류: java 또는 java8 실행 파일을 찾을 수 없음 또는 오류: <deployment-id>그룹 유형 NewDeployment 배포 <group-id>실패 오류: 작업자가 <worker-id>이유 때문에 초기화에 실패했습니다. 설치된 Java 버전이 8보다 크거나 같아야 합니다.
- 배포가 완료되지 않고 runtime.log에 여러 개의 "wait 1s for container to stop" 입력이 포함됩니다.
- 배포가 완료되지 않고 runtime.log에 "[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"가 포함되어 있습니다.
- <path>오류: <deployment-id>그룹 유형 배포 <group-id>실패 오류: 처리 중 오류가 발생했습니다. 그룹 구성이 잘못되었습니다. 112 또는 [119 0] 에 파일에 NewDeployment 대한 rw 권한이 없습니다.
- 오류: < list-of-function-arns >는 루트로 실행되도록 구성되었지만 Greengrass는 루트 권한으로 Lambda 함수를 실행하도록 구성되지 않았습니다.
- 오류: <deployment-id>그룹 유형 NewDeployment 배포 <group-id>실패 오류: Greengrass 배포 오류: 배포의 다운로드 단계를 실행할 수 없습니다. 처리 중 오류: 다운로드한 그룹 파일을 로드할 수 없음: 사용자 이름을 기반으로 UID를 찾을 수 없음, 사용자 이름: ggc_user: 사용자 이름: ggc_user: 사용자 이름: 알 수 없는 사용자 ggc_user.
- 오류: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}
- 오류: <deployment-id>그룹 유형 NewDeployment 배포 <group-id>실패 오류: 프로세스 시작 실패: container_linux.go:259: 컨테이너 프로세스 시작시 “process_linux.go:250: init에 대한 exec setns 프로세스 실행 중 발생함\" 대기: 하위 프로세스 없음\ "”.

- 오류: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: no such host ... [오류]-Greengrass 배포 오류: 클라우드에 배포 상태 보고 실패 ... net/http: 연결을 기다리는 동안 요청이 취소됨(헤더를 기다리는 동안 Client.Timeout 초과됨)

현재 배포가 작동하지 않으며 이전의 작동하는 배포로 되돌려야 합니다.

해결책: AWS IoT 콘솔 또는 AWS IoT Greengrass API를 사용하여 이전의 작동하는 배포를 재배포합니다. 그러면 코어 디바이스에 해당 그룹 버전이 배포됩니다.

배포를 재배포하려면(콘솔)

1. 그룹 구성 페이지에서 배포 탭을 선택합니다. 이 페이지에는 각 배포 시도의 날짜 및 시간, 그룹 버전, 상태를 포함하여 그룹의 배포 기록이 표시됩니다.
2. 다시 배포하려는 배포가 포함된 행을 찾습니다. 재배포하려는 배포를 선택하고 재배포를 선택합니다.

배포를 재배포하려면(CLI)

1. 재배포하려는 배포의 ID를 찾는 데 사용합니다 ListDeployments. 예:

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

이 명령은 그룹의 배포 목록을 반환합니다.

```
{
    "Deployments": [
        {
            "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
```

```
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/8dd1d899-4ac9-4f5d-
afe4-22de086efc62",
            "CreatedAt": "2019-07-01T20:56:49.641Z"
        },
        {
            "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
            "CreatedAt": "2019-07-01T20:41:47.048Z"
        },
        {
            "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
            "CreatedAt": "2019-06-18T15:16:02.965Z"
        }
    ]
}
```

> **Note**
>
> 다음 AWS CLI 명령은 그룹 및 배포 ID에 대한 예제 값을 사용합니다. 명령을 실행할 때 이들 예제 값을 올바른 값으로 바꾸십시오.

2. 대상 CreateDeployment배포를 재배포하는 데 사용합니다. 배포 유형을 `Redeployment`로 설정합니다. 예:

```
aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596
```

이 명령은 새 배포의 ARN 및 ID를 반환합니다.

```
{
    "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
```

```
    "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
}
```

3. 배포 상태를 가져오는 GetDeploymentStatus데 사용합니다.

배포 시 403 Forbidden 오류가 로그에 나타납니다.

해결책: 클라우드 내 AWS IoT Greengrass 코어의 정책에 `"greengrass:*"`가 허용된 작업으로 포함되어 있는지 확인합니다.

create-deployment 명령을 처음 실행하면 ConcurrentDeployment 오류가 발생합니다.

해결책: 배포가 진행 중일 수 있습니다. get-deployment-status를 실행하면 배포가 생성되었는지 확인할 수 있습니다. 그렇지 않으면 배포 생성을 다시 시도하십시오.

오류: Greengrass is not authorized to assume the Service Role associated with this account 또는 오류: Failed: TES service role is not associated with this account.

해결책: 배포에 실패하면 이 오류가 발생할 수 있습니다. Greengrass 서비스 역할이 현재 AWS 리전의 AWS 계정 계정과 연결되어 있는지 확인합니다. 자세한 내용은 the section called “서비스 역할 관리(CLI)” 또는 the section called “서비스 역할 관리(콘솔)” 섹션을 참조하세요.

오류: unable to execute download step in deployment. error while downloading: error while downloading the Group definition file: ... x509: certificate has expired or is not yet valid

해결책: 배포에 실패하면 `runtime.log`에서 이 오류가 발생할 수 있습니다. `x509: certificate has expired or is not yet valid` 메시지가 포함된 `Deployment failed` 오류가 발생하면 디바이스 클록을 확인합니다. TLS 및 X.509 인증서를 사용하는 경우 안전하게 IoT 시스템을 구축할 수 있지만 서버와 클라이언트의 시간이 정확해야 합니다. IoT 디바이스가 AWS IoT Greengrass 또는 서버 인증서를 사용하는 다른 TLS 서비스에 연결하려면 시간이 정확해야(오차 15분 이내) 합니다. 자세한 내용은 AWS 공식 블로그의 사물 인터넷에서 디바이스 시간을 사용한 AWS IoT 서버 인증서 유효성 검사를 참조하십시오.

배포가 완료되지 않습니다.

해결책: 다음을 수행합니다.

- 사용자의 코어 디바이스에서 AWS IoT Greengrass 데몬이 실행 중인지 확인합니다. 코어 디바이스 터미널에서 다음 명령들을 실행하여 대몬(daemon)이 실행 중인지 확인하고 필요한 경우 시작하십시오.

  1. 대몬(daemon)이 실행 중인지 확인하려면:

     ```
     ps aux | grep -E 'greengrass.*daemon'
     ```

     출력에 `root`에 대한 `/greengrass/ggc/packages/1.11.6/bin/daemon` 입력이 포함되어 있는 경우에는 대몬(daemon)이 실행 중인 것입니다.

     경로의 버전은 코어 디바이스에 설치된 AWS IoT Greengrass 코어 소프트웨어 버전에 따라 다릅니다.

  2. 대몬(daemon)을 시작하려면:

     ```
     cd /greengrass/ggc/core/
     sudo ./greengrassd start
     ```

- 코어 디바이스가 연결되어 있고 코어 연결 엔드포인트가 올바르게 구성되어 있는지 확인합니다.

오류: java 또는 java8 실행 파일을 찾을 수 없음 또는 오류: <deployment-id> 그룹 유형 NewDeployment 배포 <group-id>실패 오류: 작업자가 <worker-id> 이유 때문에 초기화에 실패했습니다. 설치된 Java 버전이 8보다 크거나 같아야 합니다.

해결 방법: AWS IoT Greengrass 코어에 대해 스트림 관리자가 활성화된 경우 그룹을 배포하기 전에 코어 디바이스에 Java 8 런타임을 설치해야 합니다. 자세한 내용은 스트림 관리자에 대한 요구 사항을 참조하십시오. AWS IoT 콘솔의 기본 그룹 생성 워크플로를 사용하여 그룹을 생성하면 스트림 관리자가 기본적으로 활성화됩니다.

또는 스트림 관리자를 비활성화한 다음 그룹을 배포합니다. 자세한 설명은 the section called “설정 구성(콘솔)” 섹션을 참조하세요.

배포가 완료되지 않고 runtime.log에 여러 개의 "wait 1s for container to stop" 입력이 포함됩니다.

해결책: 코어 디바이스 터미널에서 다음 명령을 실행하여 AWS IoT Greengrass 데몬을 다시 시작합니다.

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
sudo ./greengrassd start
```

배포가 완료되지 않고 **`runtime.log`**에 "[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"가 포함되어 있습니다.

해결 방법: Greengrass 코어가 HTTPS 프록시 연결을 사용하도록 구성되어 있고 프록시 서버 인증서 체인이 시스템에서 신뢰되지 않으면 `runtime.log`에 이 오류가 표시될 수 있습니다. 이 문제를 해결

하려면 루트 CA 인증서에 인증서 체인을 추가합니다. Greengrass 코어는 이 파일의 인증서를 AWS IoT Greengrass와의 HTTPS 및 MQTT 연결에서 TLS 인증에 사용되는 인증서 풀에 추가합니다.

다음 예에서는 루트 CA 인증서 파일에 추가된 프록시 서버 CA 인증서를 보여줍니다.

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

기본적으로 루트 CA 인증서 파일은 */greengrass-root*/certs/root.ca.pem에 있습니다. 코어 디바이스에서 위치를 찾으려면 config.json에서 `crypto.caPath` 속성을 확인하세요.

> **Note**
>
> *greengrass-root*는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 설치된 경로를 나타냅니다. 일반적으로 이는 /greengrass 디렉터리입니다.

<path>오류: <deployment-id>그룹 유형 배포 <group-id>실패 오류: 처리 중 오류가 발생했습니다. 그룹 구성이 잘못되었습니다. 112 또는 [119 0] 에 파일에 NewDeployment 대한 rw 권한이 없습니다.

해결책: *<path>* 디렉터리의 소유자 그룹이 디렉터리에 대한 읽기 및 쓰기 권한을 보유하고 있는지 확인합니다.

오류: < list-of-function-arns >는 루트로 실행되도록 구성되었지만 Greengrass는 루트 권한으로 Lambda 함수를 실행하도록 구성되지 않았습니다.

해결책: 배포에 실패하면 `runtime.log`에서 이 오류가 발생할 수 있습니다. AWS IoT Greengrass를 구성해 Lambda 함수가 루트 권한으로 실행되도록 해야 합니다. `greengrass_root/config/config.json`에서 `allowFunctionsToRunAsRoot`의 값을 `yes`로 바꾸거나 Lambda 함수를 변경해 다른 사용자/그룹으로서 실행합니다. 자세한 설명은 the section called “루트로서의 Lambda 함수 실행” 섹션을 참조하세요.

오류: <deployment-id>그룹 유형 NewDeployment 배포 <group-id>실패 오류: Greengrass 배포 오류: 배포의 다운로드 단계를 실행할 수 없습니다. 처리 중 오류: 다운로드한 그룹 파일을 로드할 수 없음: 사용자 이름을 기반으로 UID를 찾을 수 없음, 사용자 이름: ggc_user: 사용자 이름: ggc_user: 사용자 이름: 알 수 없는 사용자 ggc_user.

해결 방법: AWS IoT Greengrass 그룹의 기본 액세스 ID가 표준 시스템 계정을 사용하는 경우 `ggc_user` 사용자 및 `ggc_group` 그룹이 디바이스에 있어야 합니다. 사용자 및 그룹을 추가하는 방법에 대한 지침은 이 단계를 참조하십시오. 이름을 표시된 대로 정확히 입력해야 합니다.

오류: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"}

해결책: 배포에 실패하면 `runtime.log`에서 이 오류가 발생할 수 있습니다. 그룹의 Lambda 함수가 코어의 파일 시스템에 있는 `/usr` 디렉터리에 액세스할 수 없는 경우 이 오류가 발생합니다. 이 문제를 해결 하려면 그룹에 로컬 볼륨 리소스를 추가하고 그룹을 배포합니다. 리소스는 다음 조건을 충족해야 합니다.

- `/usr`을 소스 경로 및 대상 경로로 지정
- 리소스를 소유한 Linux 그룹의 OS 그룹 권한을 자동으로 추가
- Lambda 함수와 연결하고 읽기 전용 액세스 허용

오류: <deployment-id>그룹 유형 NewDeployment 배포 <group-id>실패 오류: 프로세스 시작 실패: container_linux.go:259: 컨테이너 프로세스 시작시 “process_linux.go:250: init에 대한 exec setns 프로세스 실행 중 발생함\" 대기: 하위 프로세스 없음\ "”.

해결책: 배포에 실패하면 이 오류가 발생할 수 있습니다. 배포를 다시 시도하십시오.

오류: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: no such host ... [오류]-Greengrass 배포 오류: 클라우드에 배포 상태 보고 실패 ... net/http: 연결을 기다리는 동안 요청이 취소됨(헤더를 기다리는 동안 Client.Timeout 초과됨)

해결책: `systemd-resolved`를 사용하는 경우 이 오류가 발생할 수 있습니다. 이 파라미터는 기본적으로 `DNSSEC` 설정을 활성화합니다. 그 결과 많은 퍼블릭 도메인이 인식되지 않습니다. AWS IoT Greengrass 엔드포인트에 도달하려는 시도가 호스트를 찾지 못하고, 따라서 배포가 `In Progress` 상태를 지속합니다.

다음 명령 및 출력을 사용하여 이 문제를 테스트할 수 있습니다. AWS 리전 엔드포인트의 엔드포인트 자리표시자를 *##*으로 바꾸십시오.

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
 failed-auxiliary
```

한 가지 가능한 해결책은 DNSSEC를 비활성화하는 것입니다. DNSSEC가 `false`일 경우 DNS 조회가 DNSSEC 검증되지 않습니다. 자세한 내용은 `systemd`에 대한 알려진 문제를 참조하십시오.

1. `DNSSEC=false`을 `/etc/systemd/resolved.conf`에 추가합니다.
2. `systemd-resolved`을 다시 시작합니다.

`resolved.conf` 및 DNSSEC에 대한 자세한 내용은 터미널에서 `man resolved.conf`를 실행하십시오.

그룹 생성/함수 생성 관련 문제

다음 정보를 사용하면 AWS IoT Greengrass 그룹 또는 Greengrass Lambda 함수 생성과 관련된 문제 해결에 도움이 됩니다.

문제

- 오류: 그룹의 " 구성이 잘못되었습니다. IsolationMode
- 오류: arn 함수의 IsolationMode " <function-arn>구성이 잘못되었습니다.
- 오류: MemorySize <function-arn>=에서는 IsolationMode arn을 사용한 함수 구성이 허용되지 않습니다. NoContainer
- 오류: =에서는 arn이 있는 함수에 대한 Access Sysfs 구성이 <function-arn>허용되지 않습니다. IsolationMode NoContainer
- <function-arn>오류: =에서 arn을 사용하는 함수의 MemorySize 구성이 필요합니다. IsolationMode GreengrassContainer

- 오류: 함수는 <function-arn><resource-type> IsolationModeNoContainer=에서 허용되지 않는 유형의 리소스를 나타냅니다.
- 오류: Execution configuration for function with arn <function-arn> is not allowed.

오류: 그룹의 " 구성이 잘못되었습니다. IsolationMode

해결책: `function-definition-version`의 `DefaultConfig` 내의 `IsolationMode` 값이 지원되지 않는 경우 이 오류가 발생합니다. 지원되는 값은 `GreengrassContainer` 및 `NoContainer`입니다.

오류: arn 함수의 IsolationMode " <function-arn>구성이 잘못되었습니다.

해결책: `function-definition-version`의 <function-arn> 내의 `IsolationMode` 값이 지원되지 않는 경우 이 오류가 발생합니다. 지원되는 값은 `GreengrassContainer` 및 `NoContainer`입니다.

오류: MemorySize <function-arn>=에서는 IsolationMode arn을 사용한 함수 구성이 허용되지 않습니다. NoContainer

해결 방법: `MemorySize` 값을 지정하고 컨테이너화 없이 실행하도록 선택하면 이 오류가 발생합니다. 컨테이너화 없이 실행되는 Lambda 함수에는 메모리 제한이 있을 수 없습니다. 제한을 제거하거나 Lambda 함수가 AWS IoT Greengrass 컨테이너에서 실행되도록 변경할 수 있습니다.

오류: =에서는 arn이 있는 함수에 대한 Access Sysfs 구성이 <function-arn> 허용되지 않습니다. IsolationMode NoContainer

해결 방법: `AccessSysfs`에 `true`을(를) 지정하고 컨테이너화 없이 실행하도록 선택하면 이 오류가 발생합니다. 컨테이너화 없이 실행되는 Lambda 함수는 파일 시스템에 직접 액세스하도록 코드를 업데이트해야 하며 `AccessSysfs`을(를) 사용할 수 없습니다. `AccessSysfs`를 `false` 값으로 지정하거나 Lambda 함수가 AWS IoT Greengrass 컨테이너에서 실행되도록 변경할 수 있습니다.

<function-arn>오류: =에서 arn을 사용하는 함수의 MemorySize 구성이 필요합니다. IsolationMode GreengrassContainer

해결책: 이 오류는 AWS IoT Greengrass 컨테이너에서 실행 중인 Lambda 함수에 대해 `MemorySize` 제한을 지정하지 않았을 때 발생합니다. `MemorySize` 값을 지정해 오류를 해결하십시오.

오류: 함수는 <function-arn><resource-type> IsolationModeNoContainer=에서 허용되지 않는 유형의 리소스를 나타냅니다.

해결책: 컨테이너화하지 않고 Lambda 함수를 실행하는 경우 `Local.Device`, `Local.Volume`, `ML_Model.SageMaker.Job`, `ML_Model.S3_Object`, `S3_Object.Generic_Archive` 리소스 유형에 액세스할 수 없습니다. 해당 리소스 타입이 필요한 경우 AWS IoT Greengrass 컨테이너에서 실행해야 합니다. 또한 Lambda 함수의 코드를 변경함으로써 컨테이너화 없이 실행 시 로컬 디바이스에 곧바로 액세스할 수도 있습니다.

오류: Execution configuration for function with arn <function-arn> is not allowed.

해결책: 이 오류는 `GGIPDetector` 또는 `GGCloudSpooler`를 사용하여 시스템 Lambda 함수을 작성하거나 `IsolationMode` 또는 `RunAs` 구성을 지정했을 때 발생합니다. 이 시스템 Lambda 함수의 `Execution` 파라미터를 생략해야 합니다.

검색 문제

다음 정보를 사용하면 AWS IoT Greengrass 검색 서비스의 문제 해결에 도움이 됩니다.

문제

- 오류: 디바이스가 너무 많은 그룹의 멤버입니다. 디바이스는 최대 10개 그룹에 속할 수 있습니다.

오류: 디바이스가 너무 많은 그룹의 멤버입니다. 디바이스는 최대 10개 그룹에 속할 수 있습니다.

해결책: 이것은 알려진 문제입니다. 한 클라이언트 디바이스는 최대 10개 그룹의 멤버일 수 있습니다.

기계 학습 리소스 문제

다음 정보를 사용하여 기계 학습 리소스의 문제를 해결할 수 있습니다.

문제

- InvalidML ModelOwner - GroupOwnerSetting ML 모델 리소스에 제공되었지만 존재하지 GroupOwner 않음 GroupPermission
- NoContainer 함수는 Machine Learning 리소스를 연결할 때 권한을 구성할 수 없습니다. <function-arn>리소스 액세스 정책에 <resource-id>권한이 <ro/rw> 있는 기계 학습 리소스를 말합니다.
- 함수는 <function-arn><resource-id> ResourceAccessPolicy 및 리소스 모두에서 권한이 누락된 Machine Learning 리소스를 가리킵니다 OwnerSetting.
- 함수는 <function-arn><resource-id>\ "rw\" 권한이 있는 Machine Learning 리소스를 가리키는 반면, 리소스 소유자 설정은\ "ro\ GroupPermission "만 허용합니다.
- NoContainer 함수는 <function-arn>중첩된 대상 경로의 리소스를 말합니다.
- Lambda <function-arn>은 동일한 그룹 소유자 ID를 공유하여 리소스 <resource-id>에 대한 액세스 권한을 획득합니다.

InvalidML ModelOwner - GroupOwnerSetting ML 모델 리소스에 제공되었지만 존재하지 GroupOwner 않음 GroupPermission

해결 방법: 기계 학습 리소스에 ResourceDownloadOwnerSetting개체가 포함되어 있지만 필수 `GroupOwner` 또는 `GroupPermission` 속성이 정의되지 않은 경우 이 오류가 발생합니다. 이 문제를 해결하려면 누락된 속성을 정의합니다.

NoContainer 함수는 Machine Learning 리소스를 연결할 때 권한을 구성할 수 없습니다. <function-arn>리소스 액세스 정책에 <resource-id>권한이 <ro/rw> 있는 기계 학습 리소스를 말합니다.

해결 방법: 컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 대한 함수 수준 권한을 지정하는 경우 이 오류가 발생합니다. 컨테이너화되지 않은 함수는 기계 학습 리소스에 정의된 리소스 소유자 권한으로부터 권한을 상속해야 합니다. 이 문제를 해결하려면 리소스 소유자 권한(콘솔)을 상속하거나, Lambda 함수의 리소스 액세스 정책(API)에서 권한을 제거하도록 선택하십시오.

함수는 <function-arn><resource-id> ResourceAccessPolicy 및 리소스 모두에서 권한이 누락된 Machine Learning 리소스를 가리킵니다 OwnerSetting.

해결 방법: 기계 학습 리소스에 대한 권한이 연결된 Lambda 함수 또는 리소스에 대해 구성되지 않은 경우 이 오류가 발생합니다. 이 문제를 해결하려면 Lambda 함수의 ResourceAccessPolicy속성 또는 리소스의 속성에서 권한을 OwnerSetting구성하십시오.

함수는 <function-arn><resource-id>\ "rw\" 권한이 있는 Machine Learning 리소스를 가리키는 반면, 리소스 소유자 설정은\ "ro\ GroupPermission "만 허용합니다.

해결 방법: 연결된 Lambda 함수에 대해 정의된 액세스 권한이 기계 학습 리소스에 대해 정의된 리소스 소유자 권한을 초과하는 경우 이 오류가 발생합니다. 이 문제를 해결하려면 Lambda 함수에 대해 더 제한적인 권한을 설정하거나, 리소스 소유자에 대해 덜 제한적인 권한을 설정합니다.

NoContainer 함수는 <function-arn>중첩된 대상 경로의 리소스를 말합니다.

해결 방법: 컨테이너화되지 않은 Lambda 함수에 연결된 여러 개의 기계 학습 리소스가 동일한 대상 경로 또는 중첩된 대상 경로를 사용하는 경우 이 오류가 발생합니다. 이 문제를 해결하려면 리소스에 대해 별도의 대상 경로를 지정합니다.

Lambda <function-arn>은 동일한 그룹 소유자 ID를 공유하여 리소스 <resource-id>에 대한 액세스 권한을 획득합니다.

해결 방법: 동일한 OS 그룹이 Lambda 함수의 다음으로 실행 ID와 기계 학습 리소스의 리소스 소유자로 지정되었지만 리소스가 Lambda 함수에 연결되지 않은 경우 `runtime.log`에 이 오류가 발생합니다. 이 구성은 Lambda 함수에 AWS IoT Greengrass 인증 없이 리소스에 액세스하는 데 사용할 수 있는 암시적 권한을 부여합니다.

이 문제를 해결하려면 속성 중 하나에 대해 다른 OS 그룹을 사용하거나 기계 학습 리소스를 Lambda 함수에 연결합니다.

Docker 내 AWS IoT Greengrass 코어 문제

다음 정보를 사용하면 Docker 컨테이너에서 AWS IoT Greengrass 실행 중 발생하는 문제를 해결하는 데 도움이 됩니다.

문제

- 오류: 알 수 없는 옵션: -no-include-email.
- 경고: IPv4가 비활성화되어 있습니다. 네트워킹이 작동하지 않습니다.
- 오류: 방화벽이 Windows와 컨테이너 간의 파일 공유를 차단하고 있습니다.
- 오류: GetAuthorizationToken 작업을 호출하는 동안 오류가 발생했습니다 (AccessDeniedException). 사용자: arn:aws:iam: <account-id>::user/는 리소스에서 ecr: *를 수행할 <user-name>권한이 없습니다. GetAuthorizationToken
- 오류: Cannot create container for the service greengrass: Conflict. 컨테이너 이름 "/"는 이미 사용 중입니다. aws-iot-greengrass
- 오류: [FATAL]-Failed to reset thread's mount namespace due to an unexpected error: "operation not permitted". To maintain consistency, GGC will crash and need to be manually restarted.

오류: 알 수 없는 옵션: -no-include-email.

해결책: `aws ecr get-login` 명령을 실행할 때 이 오류가 발생할 수 있습니다. 최신 AWS CLI 버전이 설치되어 있는지 확인합니다(예를 들어, `pip install awscli --upgrade --user` 실행). Windows를 사용하고 있고 MSI 설치 관리자를 사용하여 CLI를 설치한 경우 설치 프로세스를 반복해

야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 Microsoft Windows에 AWS Command Line Interface 설치를 참조하십시오.

경고: IPv4가 비활성화되어 있습니다. 네트워킹이 작동하지 않습니다.

해결책: Linux 컴퓨터에서 AWS IoT Greengrass를 실행할 때 이 경고 또는 비슷한 메시지를 받을 수 있습니다. 이 단계의 설명에 따라 IPv4 네트워크 전달을 활성화합니다. IPv4 전달을 활성화하지 않으면 AWS IoT Greengrass 클라우드 배포 및 MQTT 통신이 작동하지 않습니다. 자세한 내용은 Docker 설명서의 런타임에 네임스페이스 커널 파라미터(sysctls) 구성을 참조하십시오.

오류: 방화벽이 Windows와 컨테이너 간의 파일 공유를 차단하고 있습니다.

해결책: Windows 컴퓨터에서 Docker를 실행할 때 이 오류 또는 `Firewall Detected` 메시지를 받을 수 있습니다. 이 오류는 VPN(가상 프라이빗 네트워크)에 로그인되어 있고 네트워크 설정 때문에 공유 드라이브가 탑재되지 않는 경우에도 발생할 수 있습니다. 이러한 경우에는 VPN을 끄고 Docker 컨테이너를 재실행합니다.

오류: GetAuthorizationToken 작업을 호출하는 동안 오류가 발생했습니다 (AccessDeniedException). 사용자: arn:aws:iam: <account-id>::user/는 리소스에서 ecr: *를 수행할 <user-name>권한이 없습니다. GetAuthorizationToken

Amazon ECR 리포지토리에 액세스할 충분한 권한이 없는데 `aws ecr get-login-password` 명령을 실행할 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 Amazon ECR 사용 설명서의 Amazon ECR 리포지토리 정책 예제 및 하나의 Amazon ECR 리포지토리에 액세스를 참조하십시오.

오류: Cannot create container for the service greengrass: Conflict. 컨테이너 이름 “/”는 이미 사용 중입니다. aws-iot-greengrass

해결책: 이 문제는 이전 컨테이너에서 컨테이너 이름을 사용하는 경우 발생할 수 있습니다. 이 문제를 해결하려면 다음 명령을 실행하여 이전 Docker 컨테이너를 제거하십시오.

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

오류: [FATAL]-Failed to reset thread's mount namespace due to an unexpected error: "operation not permitted". To maintain consistency, GGC will crash and need to be manually restarted.

해결책: `runtime.log`의 이 오류는 Docker 컨테이너에서 실행 중인 AWS IoT Greengrass에 `GreengrassContainer` Lambda 함수를 배포하려고 할 때 발생할 수 있습니다. 현재, Greengrass Docker 컨테이너에는 `NoContainer` Lambda함수만 배포할 수 있습니다.

이 문제를 해결하려면 모든 Lambda 함수 `NoContainer` 모드에 있는지 확인하고 새 배포를 시작하십시오. 그런 다음 컨테이너를 시작할 때 기존 `deployment` 디렉터리를 AWS IoT Greengrass Docker 컨테이너에 바인드 마운트하지 마십시오. 대신 Docker 컨테이너에 빈 `deployment` 디렉터리를 만들고 바인드 마운트하십시오. 이를 통해 새로운 Docker 컨테이너는 `NoContainer` 모드로 실행되는 Lambda 함수로 최신 배포를 수신할 수 있습니다.

자세한 설명은 the section called “Docker 컨테이너에서 AWS IoT Greengrass를 실행하십시오.” 섹션을 참조하세요.

로그 문제 해결

로그를 로그로 보낼지, 로컬 파일 시스템에 로그를 저장할지 또는 둘 다와 같은 Greengrass 그룹의 로깅 설정을 구성할 수 있습니다. CloudWatch 문제를 해결할 때 자세한 정보를 얻으려면 일시적으로 로깅 수준을 `DEBUG`로 변경할 수 있습니다. 로깅 설정에 대한 변경 사항은 그룹을 배포할 때 적용됩니다. 자세한 설명은 the section called “AWS IoT Greengrass의 로깅 구성” 섹션을 참조하세요.

로컬 파일 시스템에서 AWS IoT Greengrass는 다음 위치에 로그를 저장합니다. 파일 시스템에서 로그를 읽으려면 루트 권한이 필요합니다.

`greengrass-root``/ggc/var/log/crash.log`

AWS IoT Greengrass 코어가 충돌할 때 생성되는 메시지들을 표시합니다.

`greengrass-root``/ggc/var/log/system/runtime.log`

어떤 구성요소가 실패했는지에 관한 메시지를 표시합니다.

greengrass-root/ggc/var/log/system/

인증서 관리자 및 연결 관리자와 같은 AWS IoT Greengrass 시스템 구성 요소의 모든 로그가 포함됩니다. ggc/var/log/system/ 및 ggc/var/log/system/runtime.log의 메시지를 사용하여 AWS IoT Greengrass 시스템 구성 요소에서 어떤 오류가 발생했는지를 확인할 수 있어야 합니다.

greengrass-root/ggc/var/log/system/localwatch/

Greengrass 로그를 로그에 업로드하는 작업을 처리하는 AWS IoT Greengrass 구성 요소의 로그를 CloudWatch 포함합니다. Greengrass 로그인 정보를 볼 수 없는 경우 이러한 로그를 사용하여 문제를 해결할 수 있습니다. CloudWatch

greengrass-root/ggc/var/log/user/

사용자 정의 Lambda 함수의 모든 로그가 포함됩니다. 이 폴더를 확인하여 로컬 Lambda 함수의 오류 메시지를 찾습니다.

> **Note**
>
> 기본적으로 *greengrass-root*는 /greengrass 디렉터리입니다. 쓰기 디렉터리가 구성되어 있는 경우에는 로그가 이 디렉터리 아래 있습니다.

로그가 클라우드에 저장되도록 구성된 경우 CloudWatch 로그를 사용하여 로그 메시지를 확인하십시오. crash.logAWS IoT Greengrass코어 디바이스의 파일 시스템 로그에서만 찾을 수 있습니다.

에 로그를 기록하도록 구성된 경우 시스템 구성 요소가 연결을 시도할 때 연결 오류가 발생하는지 확인하십시오AWS IoT. AWS IoT CloudWatch

AWS IoT Greengrass 로깅에 대한 자세한 내용은 the section called “AWS IoT Greengrass 로그를 사용하여 모니터링”을 참조하십시오.

> **Note**
>
> AWS IoT Greengrass 코어 소프트웨어 v1.0에 대한 로그는 *greengrass-root*/var/log 디렉터리 아래에 저장됩니다.

스토리지 문제 해결

로컬 파일 스토리지가 가득 차면 다음과 같이 일부 구성 요소를 시작하지 못할 수도 있습니다.

- 로컬 섀도우 업데이트가 수행되지 않습니다.
- 새 AWS IoT Greengrass 코어 MQTT 서버 인증서를 로컬로 다운로드할 수 없습니다.
- 배포에 실패합니다.

로컬에서 사용 가능한 여유 공간을 항상 파악하고 있어야 합니다. 이는 배포된 Lambda 함수의 크기, 로깅 구성(the section called “로그 문제 해결” 참조) 및 로컬로 저장된 섀도우의 수를 기준으로 하여 여유 공간을 계산할 수 있습니다.

메시지 문제 해결

AWS IoT Greengrass에서 로컬로 보내는 모든 메시지는 QoS 0과 함께 전송됩니다. 기본적으로 AWS IoT Greengrass은(는) 인 메모리 대기열에 메시지를 저장합니다. 따라서 Greengrass 코어가 재시작되면(예: 그룹 배포 또는 디바이스 재부팅 이후) 처리되지 않은 메시지가 사라집니다. 한편, 파일 시스템의 캐시에 메시지를 저장하여 코어 재시작에도 유지되도록 AWS IoT Greengrass (v1.6 이상)을 구성할 수 있습니다. 또한 대기열 크기를 구성할 수 있습니다. 대기열 크기를 구성하는 경우에는 262144 바이트(256 KB) 이상인지 확인합니다. 이 값보다 작으면 AWS IoT Greengrass이(가) 적절하게 시작될 수 없습니다. 자세한 설명은 the section called “MQTT 메시지 대기열” 섹션을 참조하세요.

> **Note**
>
> 기본 인 메모리 대기열을 사용하는 경우에는 서비스 중단이 최소화될 때 그룹을 배포하거나 디바이스를 재시작하는 것이 좋습니다.

AWS IoT에서 영구 세션을 설정하도록 코어를 구성할 수도 있습니다. 이렇게 하면 코어가 오프라인 상태에서도 AWS 클라우드에서 전송된 메시지를 수신할 수 있습니다. 자세한 설명은 the section called “AWS IoT Core를 사용하는 MQTT 영구 세션” 섹션을 참조하세요.

섀도우 동기화 제한 시간 문제 해결

Greengrass 코어 디바이스와 클라우드 간 통신에 상당한 시간 지연이 발생할 경우, 제한 시간으로 인해 섀도우 동기화에 실패할 수 있습니다. 이런 경우 다음과 유사한 로그 항목이 표시됩니다.

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
 client error: unable to get cloud shadow what_the_thing_is_named for
 synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
 copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
 {what_the_thing_is_named VersionDiscontinued []}"
```

가능한 해결 방법은 코어 디바이스가 호스트 응답을 기다리는 시간을 구성하는 것입니다. *greengrass-root*/config에서 config.json 파일을 열고 제한 시간 값이 초 단위인 system.shadowSyncTimeout 필드를 추가합니다. 예:

```
{
  "system": {
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

config.json에 지정된 shadowSyncTimeout 값이 없는 경우 기본값은 5초입니다.

> **Note**
>
> AWS IoT Greengrass 코어 소프트웨어 v1.6 및 이전 버전의 경우 기본 shadowSyncTimeout은 1초입니다.

AWS re:Post 확인

이 주제의 문제 해결 정보를 사용해 문제를 해결할 수 없는 경우에는 관련 문제에 대해 문제 해결를 검색하거나 관련 이슈에 대한 AWS IoT GreengrassAWS re:Post 태그를 확인하거나 새 포럼 스레드를 게시할 수 있습니다. AWS IoT Greengrass 팀 구성원은 적극적으로 AWS re:Post를 모니터링합니다.

에 대한 문서 기록 AWS IoT Greengrass

다음 표에는 2018년 6월 이후 AWS IoT Greengrass 개발자 안내서의 중요한 변경 사항이 설명되어 있습니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
v1.11.x Snap에 대한 지원 종료를 위한 업데이트	snapcraft.io의 AWS IoT Greengrass 코어 v 1.11.x 스냅에 대한 지원 종료 정보를 업데이트했습니다.	2023년 9월 22일
v1.11.x Snap에 대한 지원 종료를 위한 업데이트	snapcraft.io의 AWS IoT Greengrass 코어 v 1.11.x 스냅에 대한 지원 종료 정보가 추가되었습니다.	2023년 9월 19일
v1.11.6용 도커 이미지 AWS IoT Greengrass	AWS IoT Greengrass 코어 소프트웨어 v1.11.6용 도커 이미지는 아마존 엘라스틱 컨테이너 레지스트리 (Amazon ECR) 및 Docker Hub에서 사용할 수 있습니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2022년 4월 12일
AWS IoT 사용 중단을 위한 디바이스 테스터 (IDT) AWS IoT Greengrass V1	IDT for AWS IoT Greengrass V1은 더 이상 서명된 검증 보고서를 생성하지 않습니다.	2022년 4월 4일
에 대한 AWS IoT 장치 테스터 지원 업데이트 AWS IoT Greengrass	AWS IoT Greengrass 버전 4.4.1용 IDT는 이제 장치 검증을 위한 AWS IoT Greengrass 코어 소프트웨어 버전 v1.11.6을 사용할 수 있습니다.	2022년 3월 24일
AWS IoT Greengrass 버전 1.11.6이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.11.6을 사용할 수 있습니다. 이 버전에는 성능	2022년 3월 24일

	개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	
IoT SiteWise 커넥터 버전 12 출시	IoT SiteWise 커넥터 버전 12를 사용할 수 있습니다. 이 릴리스에는 버그 수정이 포함되어 있습니다.	2021년 12월 23일
v1.11.5 및 AWS IoT Greengrass v1.10.5용 도커 이미지	AWS IoT Greengrass 코어 소프트웨어 v1.11.5 및 v1.10.5용 도커 이미지는 아마존 엘라스틱 컨테이너 레지스트리(Amazon ECR) 및 도커 허브에서 사용할 수 있습니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2022년 12월 22일
AWS IoT Greengrass V1 유지 관리 정책	AWS IoT Greengrass V1 유지 관리 정책은 AWS IoT Greengrass V1 서비스 및 AWS IoT Greengrass 핵심 소프트웨어 v1.x에 대한 다양한 수준의 유지 관리 및 업데이트를 정의합니다.	2021년 12월 20일
AWS IoT 디바이스 테스터 버전 4.4.1 출시	이제 AWS IoT Greengrass 버전 4.4.1용 IDT를 사용할 수 있습니다. 이 릴리스에는 GGQ(AWS IoT Greengrass 검증 제품군) v1.3.1이 포함되며 장치 검증을 위한 AWS IoT Greengrass 코어 소프트웨어 버전 v1.11.5 및 v1.10.5 사용을 지원합니다.	2021년 12월 20일

AWS IoT Greengrass 버전 1.11.5 및 1.10.5가 출시되었습니다.	코어 소프트웨어 버전 1.11.5 및 1.10.5를 사용할 수 있습니다. AWS IoT Greengrass 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 12월 12일
v1.11.4 및 v1.10.4 도커 이미지를 다시 게시했습니다 AWS IoT Greengrass .	AWS IoT Greengrass 코어 소프트웨어 버전 1.11.4 및 1.10.4용 도커 이미지가 버그 수정을 해결하기 위해 Amazon Elastic Container Registry (Amazon ECR) 및 Docker Hub에 다시 게시되었습니다. BusyBox 최신 Docker 이미지를 사용하려면 `1.11.4-1` 또는 `1.10.4-1` 태그를 사용하십시오. 사용 가능한 태그에 대한 자세한 내용은 Docker Hub의 amazon/aws-iot-greengrass을 참조하십시오.	2021년 12월 8일
CloudWatch 메트릭 커넥터는 입력 데이터의 중복 타임스탬프를 지원합니다.	이제 중복된 타임스탬프와 입력 데이터를 이 커넥터로 보낼 수 있습니다.	2021년 11월 19일
교차 서비스 혼동된 대리자 예방 업데이트	AWS IoT Greengrass IAM 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하여 혼동되는 부정 문제를 방지할 수 있습니다.	2021년 11월 1일

v1.11.4용 도커 이미지 AWS IoT Greengrass	AWS IoT Greengrass 코어 소프트웨어 v1.11.4용 도커 이미지는 아마존 엘라스틱 컨테이너 레지스트리 (Amazon ECR) 및 Docker Hub에서 사용할 수 있습니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 8월 24일
v1.11.4 스냅을 AWS IoT Greengrass 게시했습니다.	스냅 버전 1.11.4를 사용할 수 있습니다. AWS IoT Greengrass 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 8월 20일
에 대한 AWS IoT 장치 테스터 지원 업데이트 AWS IoT Greengrass	AWS IoT Greengrass 버전 4.1.0용 IDT는 이제 장치 검증을 위한 AWS IoT Greengrass 코어 소프트웨어 버전 v1.11.4를 사용할 수 있습니다.	2021년 8월 18일
AWS IoT Greengrass 버전 1.11.4가 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.11.4를 사용할 수 있습니다. 이 릴리스에서는 이전 버전의 Core 소프트웨어에서 v1.11.3으로 업그레이드할 수 없었던 스트림 관리자 문제가 수정되었습니다. AWS IoT Greengrass 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 8월 17일
VPC 엔드포인트(AWS PrivateLink)	AWS IoT Greengrass 이제 컨트롤 플레인의 인터페이스 VPC 엔드포인트 (AWS PrivateLink) 를 AWS IoT Greengrass 지원합니다. 이제 VPC와 AWS IoT Greengrass 컨트롤 플레인 간에 프라이빗 연결을 설정할 수 있습니다.	2021년 8월 16일

AWS IoT 디바이스 테스터 버전 4.1.0 출시	AWS IoT 디바이스 테스터 버전 4.1.0을 사용할 수 있습니다 AWS IoT Greengrass . 이 버전은 장치 검증을 위해 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.3 및 1.10.4를 사용할 수 있도록 지원합니다.	2021년 6월 23일
AWS IoT Greengrass v1.11.3 스냅을 게시했습니다.	AWS IoT Greengrass 스냅 버전 1.11.3에는 성능 개선 및 버그 수정이 포함되어 있습니다. 모범 사례로, 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 6월 15일
v1.11.3 및 AWS IoT Greengrass v1.10.4용 도커 이미지가 출시되었습니다.	AWS IoT Greengrass 코어 소프트웨어 v1.11.3 및 v1.10.4용 도커 이미지는 아마존 엘라스틱 컨테이너 레지스트리(Amazon ECR) 및 도커 허브에서 사용할 수 있습니다. 이러한 버전의 Core에는 성능 개선 및 버그 수정이 포함되어 있습니다. AWS IoT Greengrass 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 6월 15일
AWS IoT Greengrass 버전 1.11.3이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.11.3을 사용할 수 있습니다. 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 6월 14일

AWS IoT Greengrass 버전 1.10.4가 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.10.4를 사용할 수 있습니다. 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 6월 14일
Modbus-TCP Protocol Adapter 버전 2 출시	Modbus-TCP 프로토콜 어댑터 커넥터 버전 2를 사용할 수 있습니다. 이 릴리스에서는 ASCII, UTF8 및 ISO8859 인코딩 소스 문자열에 대한 지원이 추가되었습니다.	2021년 5월 24일
Docker 애플리케이션 배포 커넥터 버전 7 릴리스	Greengrass Docker 애플리케이션 배포 커넥터 버전 7을 사용할 수 있습니다.	2021년 4월 5일
AWS IoT Greengrass 버전 1.11.1이 출시되었습니다.	AWS IoT Greengrass 코어 소프트웨어 버전 1.11.1을 사용할 수 있습니다. 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 3월 29일
AWS IoT 디바이스 테스터 버전 4.0.2 출시	AWS IoT 디바이스 테스터 버전 4.0.2를 사용할 수 있습니다 AWS IoT Greengrass . 이 버전은 IDT v4.0.0을 대체하며 코어 소프트웨어 버전 1.11.1에 대한 지원을 추가합니다. AWS IoT Greengrass 또한 IDT가 하드웨어 보안 통합(HSI) 오류를 마스킹하도록 했던 문제도 수정되었습니다.	2021년 3월 29일

IoT SiteWise 커넥터 버전 11 출시	IoT SiteWise 커넥터 버전 11을 사용할 수 있습니다. 숨겨지거나 인쇄할 수 없는 문자가 포함된 문자열에 대한 지원이 시작됩니다. 이번 릴리스에는 일반적인 성능 개선 및 버그 수정도 포함되어 있습니다.	2021년 3월 24일
AWS IoT Greengrass v1.11.0 스냅을 다시 게시했습니다.	AWS IoT Greengrass Python 인터프리터를 사용할 때 발생할 수 있는 애플리케이션 충돌 및 버그 수정 문제를 해결하기 위해 스냅 버전 1.11.0이 Snapcraft에 다시 게시되었습니다. AWS IoT Greengrass 소프트웨어 버전 1.10 및 1.9에 대한 스냅은 제공하지 않습니다.	2021년 3월 19일
에 대한 AWS IoT 장치 테스터 지원 업데이트 AWS IoT Greengrass	AWS IoT Greengrass 버전 4.0.0용 IDT는 이제 장치 검증을 위한 AWS IoT Greengrass 코어 소프트웨어 버전 v1.10.3 사용을 지원합니다.	2021년 3월 18일
v1.8.4 스냅을 다시 게시했습니다. AWS IoT Greengrass	AWS IoT Greengrass Python 인터프리터를 사용할 때 발생할 수 있는 애플리케이션 충돌 및 버그 수정 문제를 해결하기 위해 스냅 버전 1.8.4가 Snapcraft에 다시 게시되었습니다.	2021년 3월 15일

ARMv7L용 v1.11.0 도커 이미지를 다시 게시했습니다 AWS IoT Greengrass .	ArmV7L 플랫폼용 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.0의 도커 이미지가 버그 수정과 Python 인터프리터 사용 시 발생할 수 있는 애플리케이션 충돌 문제를 해결하기 위해 Amazon Elastic Container Registry (Amazon ECR) 및 Docker Hub에 다시 게시되었습니다.	2021년 3월 8일
AWS IoT Greengrass v1.10.3 도커 이미지 공개	AWS IoT Greengrass 코어 소프트웨어 버전 1.10.3용 도커 이미지는 이제 Amazon Elastic Container Registry (Amazon ECR) 및 Docker Hub에서 사용할 수 있습니다.	2021년 3월 8일
v1.11.0 및 v1.9.4 도커 이미지를 다시 게시했습니다 AWS IoT Greengrass .	Python 인터프리터를 사용할 때 발생할 수 있는 애플리케이션 충돌 및 버그 수정 문제를 해결하기 위해 AWS IoT Greengrass 코어 소프트웨어 버전 1.11.0 및 1.9.4용 도커 이미지가 Amazon Elastic Container Registry (Amazon ECR) 및 Docker Hub에 다시 게시되었습니다. ARMv7L용 Docker 이미지는 현재 다시 게시되지 않았습니다.	2021년 2월 26일

AWS IoT Greengrass 버전 1.10.3이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.10.3을 사용할 수 있습니다. 이 버전은 `systemComponentAuthTimeout` 코어 구성 속성을 추가하고 성능 개선 및 버그 수정을 포함합니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2021년 2월 24일
IoT SiteWise 커넥터 버전 10 출시	IoT SiteWise 커넥터 버전 10을 사용할 수 있습니다. 이 릴리스는 연결이 끊긴 경우 StreamManager 클라이언트와 관련된 안정성 문제를 해결하고 a가 없을 때 OPC-UA 값 처리를 개선합니다. `SourceTimestamp` IoT SiteWise 커넥터를 사용하여 로컬 장치 및 장비 데이터를 IoT의 자산 자산으로 전송할 수 SiteWise 있습니다.	2021년 1월 22일
IoT SiteWise 커넥터 버전 9 출시	IoT SiteWise 커넥터 버전 9를 사용할 수 있습니다. 이로써 사용자 지정 Greengrass `StreamManager` 스트림 대상, OPC-UA 데드밴딩, 사용자 지정 스캔 모드 및 사용자 지정 스캔 속도에 대한 지원이 시작됩니다. 여기에는 IoT SiteWise 게이트웨이에서 구성을 업데이트하는 동안 개선된 성능도 포함됩니다. IoT SiteWise 커넥터를 사용하여 로컬 장치 및 장비 데이터를 IoT의 자산 자산으로 전송할 수 SiteWise 있습니다.	2020년 12월 15일

AWS IoT 디바이스 테스터 버전 4.0.0 출시	AWS IoT 디바이스 테스터 버전 4.0.0을 사용할 수 있습니다. AWS IoT Greengrass 이 버전을 사용하면 IDT를 사용하여 디바이스 검증을 위한 사용자 지정 테스트 제품군을 개발하고 실행할 수 있습니다. 여기에는 macOS 및 Windows용 코드 서명 IDT 애플리케이션도 포함됩니다.	2020년 12월 15일
AWS IoT Greengrass 스냅 v1.11	AWS IoT Greengrass 스냅 버전 1.11.0은 컨테이너화되지 않은 Lambda 함수를 지원합니다. 모범 사례로, 항상 최신 버전을 실행하는 것이 좋습니다.	2020년 12월 6일
IoT SiteWise 커넥터 버전 8 출시	IoT SiteWise 커넥터 버전 8을 사용할 수 있습니다. 이 릴리스에서는 커넥터에 간헐적인 네트워크 연결이 발생할 때의 안정성이 개선되었습니다. IoT SiteWise 커넥터를 사용하여 로컬 장치 및 장비 데이터를 IoT의 자산 자산으로 전송할 수 SiteWise 있습니다.	2020년 11월 19일
Kinesis Firehose 커넥터는 컨테이너 없음 모드를 지원합니다.	커넥터에 대한 컨테이너화 모드를 구성하는 `Isolation Mode` 파라미터가 추가되었습니다.	2020년 10월 19일
Docker 애플리케이션 배포 커넥터 버전 6 릴리스	Greengrass Docker 애플리케이션 배포 커넥터 버전 6을 사용할 수 있습니다.	2020년 9월 18일

AWS IoT Greengrass 버전 1.11.0 출시	AWS IoT Greengrass Core 소프트웨어 버전 1.11.0을 사용할 수 있습니다. 이 버전에는 시스템 상태 원격 측정 기능과 로컬 상태 점검 API가 추가되었습니다. 스트림 관리자는 이제 Amazon 심플 스토리지 서비스 (Amazon S3) 및 IoT로 데이터를 내보낼 수 있습니다. SiteWise 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2020년 9월 16일
IoT SiteWise 커넥터 버전 7 출시	IoT SiteWise 커넥터 버전 7을 사용할 수 있습니다. 이 릴리스는 게이트웨이 지표 관련 문제가 해결되었습니다. IoT SiteWise 커넥터를 사용하여 로컬 장치 및 장비 데이터를 IoT의 자산 자산으로 전송할 수 SiteWise 있습니다.	2020년 8월 14일
ServiceNow MetricBase 통합, Splunk 통합 및 Twilio 알림 커넥터는 컨테이너 없음 모드를 지원합니다.	`IsolationMode` 파라미터를 사용하여 커넥터에 대한 컨테이너화 모드를 구성할 수 있습니다.	2020년 7월 30일
SNS 커넥터는 컨테이너 없음 모드를 지원합니다.	커넥터에 대한 컨테이너화 모드를 구성하는 `IsolationMode` 파라미터를 사용할 수 있습니다.	2020년 7월 6일
CloudWatch 메트릭 커넥터는 컨테이너 없음 모드를 지원합니다.	`IsolationMode` 파라미터를 사용하여 커넥터에 대한 컨테이너화 모드를 구성할 수 있습니다.	2020년 6월 17일

AWS IoT Greengrass 버전 1.10.2가 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.10.2를 사용할 수 있습니다. 이 버전은 `mqttOperationTimeout` 코어 구성 속성을 추가하고 성능 개선 및 버그 수정을 포함합니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2020년 6월 8일
TensorFlow 기계 학습 설치 프로그램은 사용되지 않음	AWS IoT Greengrass Tensorflow 사전 패키징된 기계 학습 설치 프로그램은 더 이상 사용되지 않습니다. 기계 학습 샘플이 Python 3.7로 업그레이드되었습니다.	2020년 5월 29일
Chainer 프레임워크 지원 및 Greengrass 기계 학습 설치 프로그램 사용 중지	AWS IoT Greengrass 사전 패키징된 MXnet 및 DLR용 기계 학습 설치 프로그램 및 다운로드는 더 이상 사용되지 않습니다. Chainer 프레임워크 지원 및 관련 다운로드는 더 이상 사용되지 않습니다.	2020년 5월 4일

IoT SiteWise 커넥터 버전 6 출시	IoT SiteWise 커넥터 버전 6을 사용할 수 있습니다. 이번 릴리스에는 새로운 OPC-UA 태그의 CloudWatch 메트릭 및 자동 검색에 대한 지원이 추가되었습니다. 즉, OPC-UA 소스에 대한 태그가 변경되면 게이트웨이를 다시 시작할 필요가 없습니다. 이 버전의 커넥터에는 스트림 관리자와 AWS IoT Greengrass Core 소프트웨어 v1.10.0 이상이 필요합니다. IoT SiteWise 커넥터를 사용하여 로컬 장치 및 장비 데이터를 IoT의 자산 자산으로 전송할 수 SiteWise 있습니다.	2020년 4월 29일
Python 3.7로 업그레이드된 커넥터	Python 런타임을 지원하는 커넥터가 Python 3.7로 업그레이드되었습니다. 커넥터 버전을 Python 2.7에서 Python 3.7로 업그레이드하는 것이 좋습니다.	2020년 4월 29일
Greengrass 디바이스 설정을 무음 모드에서 실행 가능	스크립트가 값을 입력하라는 메시지를 표시하지 않도록 자동 모드에서 Greengrass 디바이스 설정을 실행할 수 있습니다.	2020년 4월 27일
새로운 Docker 기본 이미지	알파인 리눅스 (x86_64, ARMv7L 또는 AArch64) 기반 이미지를 기반으로 구축된 AWS IoT Greengrass 도커 이미지를 다운로드할 수 있습니다.	2020년 4월 23일

AWS IoT Greengrass 버전 1.10.1이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.10.1을 사용할 수 있습니다. 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 항상 최신 버전을 실행하는 것이 좋습니다.	2020년 4월 16일
새로운 보안 장	AWS IoT Greengrass 새 정보가 추가되어 보안 콘텐츠가 재구성되었습니다.	2020년 3월 30일
APT 패키지 관리자를 사용하여 설치하십시오. AWS IoT Greengrass	지원되는 데비안 기반 Linux 배포판에서는 AWS IoT Greengrass Core 소프트웨어를 장치에 설치하는 `apt` 데 사용할 수 있습니다.	2020년 2월 26일
IoT SiteWise 커넥터 버전 5 출시	IoT SiteWise 커넥터 버전 5를 사용할 수 있습니다. 이 릴리스는 AWS IoT Greengrass Core 소프트웨어 v1.9.4와의 호환성 문제를 수정합니다. IoT SiteWise 커넥터를 사용하여 로컬 장치 및 장비 데이터를 IoT의 자산 자산으로 전송할 수 SiteWise 있습니다.	2020년 2월 12일
코어 디바이스를 빠르게 설정하는 새로운 스크립트	Greengrass 디바이스 설정을 사용하여 몇 분 안에 코어 디바이스를 구성할 수 있습니다. 또한 AWS IoT Greengrass 이제 Node.js 12.x Lambda 함수를 지원합니다.	2019년 12월 20일

AWS IoT Greengrass 버전 1.10.0이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.10.0을 사용할 수 있습니다. 이 버전의 새로운 기능은 스트림 매니저, Docker 애플리케이션 배포를 통한 컨테이너 지원, 기계 학습 리소스에 액세스 할 수 있는 컨테이너화되지 않은 Lambda 함수, AWS IoT을 사용한 MQTT 영구 세션 지원, 지정된 포트를 통한 로컬 MQTT 트래픽 지원을 포함합니다.	2019년 11월 25일
배포 알림에 대한 콘솔 지원	Amazon EventBridge 콘솔을 사용하여 Greengrass 그룹 배포 상태가 변경될 때 트리거되는 이벤트 규칙을 생성할 수 있습니다.	2019년 11월 14일
AWS IoT Greengrass 버전 1.9.4가 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.9.4를 사용할 수 있습니다. 이 버전에는 성능 개선과 버그 수정이 포함됩니다. 모범 사례로, 항상 최신 버전을 실행하는 것이 좋습니다.	2019년 10월 17일
Greengrass 서비스 역할을 관리하기 위한 콘솔 지원	AWS IoT 콘솔의 새롭고 개선된 기능을 사용하여 Greengrass 서비스 역할을 관리하세요.	2019년 10월 4일
그룹 레벨 태그를 관리하기 위한 콘솔 지원	AWS IoT 콘솔에서 Greengrass 그룹에 대한 태그를 생성, 보기 및 관리할 수 있습니다.	2019년 9월 23일

새로운 기계 학습 커넥터	모델 입력 및 예측을 게시하려면 ML 피드백 커넥터를 사용하고, 로컬 객체 감지 추론 서비스를 실행하려면 ML Object Detection 커넥터를 사용합니다.	2019년 9월 19일
AWS IoT Greengrass 버전 1.9.3이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.9.3을 사용할 수 있습니다. 이 버전을 사용하면 ARMv6L 아키텍처의 Raspbian 배포판에 AWS IoT Greengrass Core 소프트웨어를 설치하고, ALPN을 사용하여 포트 443에서 OTA 업데이트를 지원하며, Python 2.7 Lambda 함수에서 다른 Lambda 함수로 전송되는 바이너리 페이로드에 대한 버그 수정이 포함되어 있습니다.	2019년 9월 12일
AWS IoT Greengrass 버전 1.8.4가 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.8.4를 사용할 수 있습니다. 이 버전에는 성능 개선과 버그 수정이 포함됩니다. v1.8.x를 실행 중인 경우, v1.8.4 또는 v1.9.3으로 업그레이드하는 것이 좋습니다. 이전 버전의 경우, v1.9.3으로 업그레이드하는 것이 좋습니다.	2019년 8월 30일

AWS IoT Greengrass 다음 지원을 포함하여 버전 1.9.2가 출시되었습니다. OpenWrt	AWS IoT Greengrass Core 소프트웨어 버전 1.9.2를 사용할 수 있습니다. 이 버전을 사용하면 Armv8 (AArch64) 및 ARMv7L OpenWrt 아키텍처를 사용하는 배포판에 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다.	2019년 6월 20일
AWS IoT Greengrass 버전 1.8.3이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.8.3을 사용할 수 있습니다. 이 버전에는 일반적인 성능 향상과 버그 수정이 포함됩니다. v1.8.x를 실행 중인 경우, v1.8.3 또는 v1.9.2로 업그레이드하는 것이 좋습니다. 이전 버전의 경우, v1.9.2로 업그레이드하는 것이 좋습니다.	2019년 6월 20일
AWS IoT Greengrass 버전 1.9.1이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.9.1을 사용할 수 있습니다. 이 버전에는 항목에 와일드카드 AWS IoT 문자가 포함된 메시지에 대한 버그 수정이 포함되어 있습니다.	2019년 5월 10일
AWS IoT Greengrass 버전 1.8.2가 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.8.2를 사용할 수 있습니다. 이 버전에는 일반적인 성능 향상과 버그 수정이 포함됩니다. v1.8.x를 실행 중인 경우, v1.8.2 또는 v1.9.0으로 업그레이드하는 것이 좋습니다. 이전 버전의 경우, v1.9.0으로 업그레이드하는 것이 좋습니다.	2019년 5월 2일

AWS IoT Greengrass 버전 1.9.0이 출시되었습니다.	새로운 기능: Python 3.7 및 Node.js 8.10 Lambda 런타임에 대한 지원, 최적화된 MQTT 연결 및 로컬 MQTT 서버에 대한 EC(Elliptic Curve) 키 지원	2019년 5월 1일
AWS IoT Greengrass 버전 1.8.1이 출시되었습니다.	AWS IoT Greengrass Core 소프트웨어 버전 1.8.1을 사용할 수 있습니다. 이 버전에는 일반적인 성능 향상과 버그 수정이 포함됩니다. 모범 사례로, 항상 최신 버전을 실행하는 것이 좋습니다.	2019년 4월 18일
AWS IoT Greengrass 스냅크래프트에서 스냅을 사용할 수 있습니다.	AWS IoT Greengrass Snap Store 앱을 사용하여 Linux 디바이스에서 소프트웨어를 빠르게 설계, 테스트 및 배포할 수 있습니다. AWS IoT Greengrass	2019년 4월 1일
태그 기반 권한을 사용하여 추가 액세스 제어 지원	태그 인 AWS Identity and Access Management (IAM) 정책을 사용하여 AWS IoT Greengrass 리소스에 대한 액세스를 제어할 수 있습니다.	2019년 3월 29일
IoT Analytics 커넥터 출시	IoT Analytics 커넥터를 사용하여 로컬 디바이스 데이터를 AWS IoT Analytics 채널에 전송합니다.	2019년 3월 15일
Kinesis Firehose 커넥터의 Batch 지원	Kinesis Firehose 커넥터는 지정된 간격으로 Amazon Data Firehose에 일괄 데이터 레코드를 전송할 수 있도록 지원합니다.	2019년 3월 15일

AWS CloudFormationAWS IoT Greengrass 리소스 지원	AWS CloudFormation 템플릿을 사용하여 AWS IoT Greengrass 리소스를 만들고 관리합니다.	2019년 3월 15일
AWS IoT Greengrass 버전 1.8.0이 출시되었습니다.	새로운 기능: Lambda 함수의 구성 가능한 기본 액세스 ID, 포트 443을 통한 HTTPS 트래픽 지원, MQTT 연결을 위한 예측 가능한 이름이 지정된 클라이언트 ID. AWS IoT	2019년 3월 7일
AWS IoT Greengrass 버전 1.7.1 및 1.6.1이 출시되었습니다.	코어 소프트웨어 버전 1.7.1 및 1.6.1을 사용할 수 있습니다. AWS IoT Greengrass 이러한 버전에는 Linux 커널 버전 3.17 이상이 필요합니다. Greengrass 코어 소프트웨어 버전을 실행 중인 고객은 1.7.1 버전으로 즉시 업그레이드하는 것이 좋습니다.	2019년 2월 11일
SageMaker 네오 딥러닝 런타임	SageMaker Neo 딥러닝 런타임은 SageMaker Neo 딥러닝 컴파일러로 최적화된 머신 러닝 모델을 지원합니다.	2018년 11월 28일
Docker AWS IoT Greengrass 컨테이너에서 실행	Greengrass 그룹이 컨테이너화 없이 실행되도록 구성하여 Docker AWS IoT Greengrass 컨테이너에서 실행할 수 있습니다.	2018년 11월 26일

AWS IoT Greengrass 버전 1.7.0이 출시되었습니다.	새로운 기능: Greengrass 커넥터, 로컬 암호 관리자, Lambda 함수의 격리 및 권한 설정, 신뢰 보안의 하드웨어 루트, ALPN 또는 네트워크 프록시를 사용한 연결, Raspbian Stretch 지원.	2018년 11월 26일
AWS IoT Greengrass 소프트웨어 다운로드	AWS IoT Greengrass 코어 소프트웨어, AWS IoT Greengrass 코어 SDK 및 AWS IoT Greengrass Machine Learning SDK 패키지는 Amazon을 통해 다운로드할 수 있습니다. CloudFront	2018년 11월 26일
AWS IoT 디바이스 테스터 전용 AWS IoT Greengrass	AWS IoT 디바이스 테스터 AWS IoT Greengrass 용을 사용하여 CPU 아키텍처, 커널 구성 및 드라이버가 호환되는지 확인하십시오 AWS IoT Greengrass.	2018년 11월 26일
AWS CloudTrailAWS IoT Greengrass API 호출 로깅	AWS IoT Greengrass 에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업의 기록을 제공하는 AWS 서비스와 통합됩니다 AWS IoT Greengrass.	2018년 10월 29일
엔비디아 젯슨 TX2에서의 TensorFlow v1.10.1 지원	제공하는 NVIDIA Jetson TX2 용 TensorFlow 사전 컴파일된 라이브러리는 이제 v1.10.1 을 사용합니다. AWS IoT Greengrass TensorFlow 이것은 Jetpack 3.3 및 CUDA Toolkit 9.0을 지원합니다.	2018년 10월 18일

MXNet v1.2.1 기계 학습 리소스 지원	AWS IoT Greengrass MXNet v1.2.1을 사용하여 학습된 기계 학습 모델을 지원합니다.	2018년 8월 29일
AWS IoT Greengrass 버전 1.6.0이 출시되었습니다.	새로운 기능: Lambda 실행 파일, 구성 가능한 메시지 대기열, 구성 가능한 재연결 시도 간격, /proc의 볼륨 리소스, 구성 가능한 쓰기 디렉터리.	2018년 7월 26일

이전 업데이트

다음 표에는 2018년 7월 이전에 AWS IoT Greengrass 개발자 안내서의 중요한 변경 사항이 설명되어 있습니다.

변경 사항	설명	날짜
AWS IoT Greengrass 버전 1.5.0 출시	새로운 기능: • 클라우드 학습 모델을 사용하는 로컬 머신 러닝 추론. 자세한 설명은 기계 학습 추론 수행 섹션을 참조하세요. • Greengrass Lambda 함수는 이제 JSON 외에 이진 입력 데이터도 지원합니다. 자세한 내용은 AWS IoT Greengrass 코어 버전을 참조하십시오.	2018년 3월 29일
AWS IoT Greengrass 버전 1.3.0이 출시되었습니다.	새로운 기능: • 클라우드에 배포된 Greengrass 업데이트 작업을 처리할 수 있는 O ver-the-air (OTA) 업데이트 에이전트. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트 섹션을 참조하세요. • Greengrass Lambda 함수에서 로컬 주변 장치 및 리소스에 액세스합니다. 자세한 설명은 Lambda 함수와 커넥터를 사용하여 로컬 리소스에 액세스 섹션을 참조하세요.	2017년 11월 27일

변경 사항	설명	날짜
AWS IoT Greengrass 버전 1.1.0 출시	새로운 기능: • 배포된 AWS IoT Greengrass 그룹을 재설정합니다. 자세한 설명은 배포 재설정 섹션을 참조하세요. • Python 2.7 외에 Node.js 6.10 및 Java 8 Lambda 런타임 지원	2017년 9월 20일
AWS IoT Greengrass 버전 1.0.0 출시	AWS IoT Greengrass 일반적으로 사용할 수 있습니다.	2017년 6월 7일