aws

개발자 가이드, 버전 2

AWS IoT Greengrass

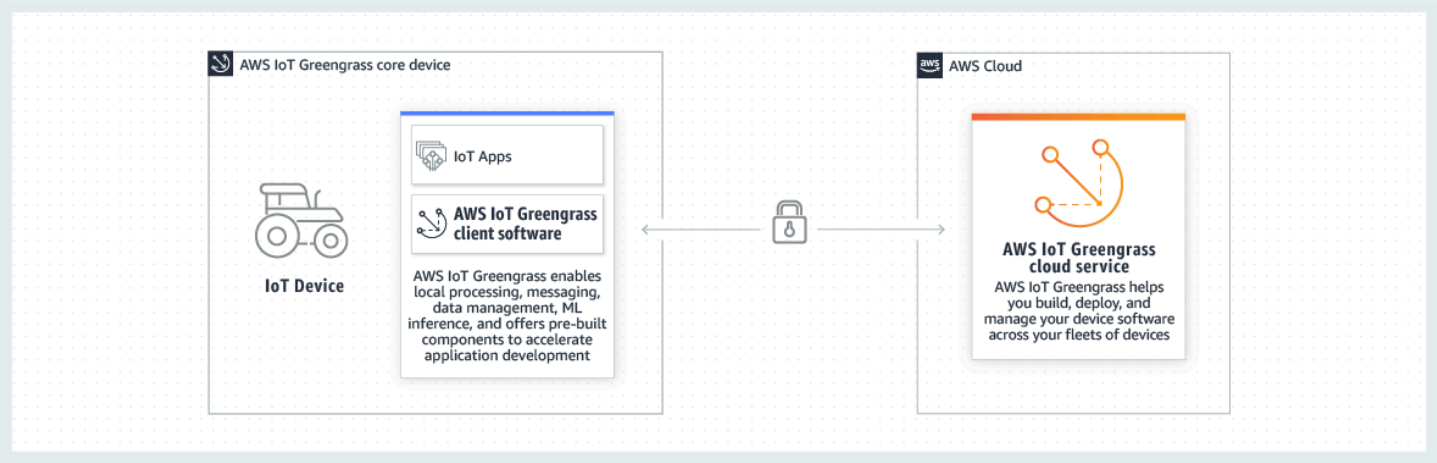
AWS IoT Greengrass: 개발자 가이드, 버전 2

Table of Contents

AWS IoT Greengrass(이)란 무엇인가요?

AWS IoT Greengrass는 디바이스에서 IoT 애플리케이션을 구축, 배포 및 관리하는 데 도움이 되는 오픈 소스 IoT (사물 인터넷) 에지 런타임 및 클라우드 서비스입니다. 이를 AWS IoT Greengrass 사용하여 디바이스에서 생성되는 데이터에 대해 로컬에서 조치를 취하고, 머신 러닝 모델을 기반으로 예측을 실행하고, 디바이스 데이터를 필터링 및 집계할 수 있도록 하는 소프트웨어를 구축할 수 있습니다. AWS IoT Greengrass장치가 데이터가 생성되는 위치와 더 가까운 곳에서 데이터를 수집 및 분석하고, 로컬 이벤트에 자율적으로 대응하고, 로컬 네트워크의 다른 장치와 안전하게 통신할 수 있도록 합니다. 또한 Greengrass 장치는 IoT 데이터와 안전하게 AWS IoT Core 통신하고 IoT 데이터를 로 내보낼 수 있습니다. AWS 클라우드 구성 요소라고 하는 사전 구축된 소프트웨어 모듈을 사용하여 엣지 애플리케이션을 구축하는 데 AWS IoT Greengrass를 사용할 수 있습니다. 구성 요소는 엣지 디바이스를 AWS 서비스 또는 타사 서비스에 연결할 수 있습니다. 또한 Lambda 함수, Docker 컨테이너, 네이티브 운영 체제 프로세스 또는 원하는 사용자 지정 런타임을 사용하여 소프트웨어를 패키징하고 실행할 수 있습니다. AWS IoT Greengrass

다음 예제는 AWS IoT Greengrass 디바이스가 과 상호 작용하는 방식을 보여줍니다. AWS 클라우드



새로운 기능

AWS IoT Greengrass V2새로운 기능 및 개선 사항을 소개합니다. 다음은 버전 2에서 제공되는 새로운 기능에 대한 자세한 내용입니다.

- 새로워진 내용 AWS IoT Greengrass Version 2

를 처음 사용하는 경우 AWS IoT Greengrass

를 처음 사용하는 경우 다음 섹션을 검토하는 것이 좋습니다. AWS IoT Greengrass

- AWS IoT Greengrass 작동 방식

그런 다음 시작 자습서를 따라 의 기본 기능을 사용해 보십시오AWS IoT Greengrass. 이 자습서에서는 장치에 AWS IoT Greengrass Core 소프트웨어를 설치하고 Hello World 구성 요소를 개발하고 배포를 위해 해당 구성 요소를 패키징합니다.

기존 사용자의 경우 AWS IoT Greengrass

의 AWS IoT Greengrass V1 현재 사용자에게는 Greengrass 버전 1과 Greengrass 버전 2의 차이점을 이해하고 버전 1에서 버전 2로 이동하는 방법을 배우는 데 도움이 되는 다음 항목을 권장합니다.

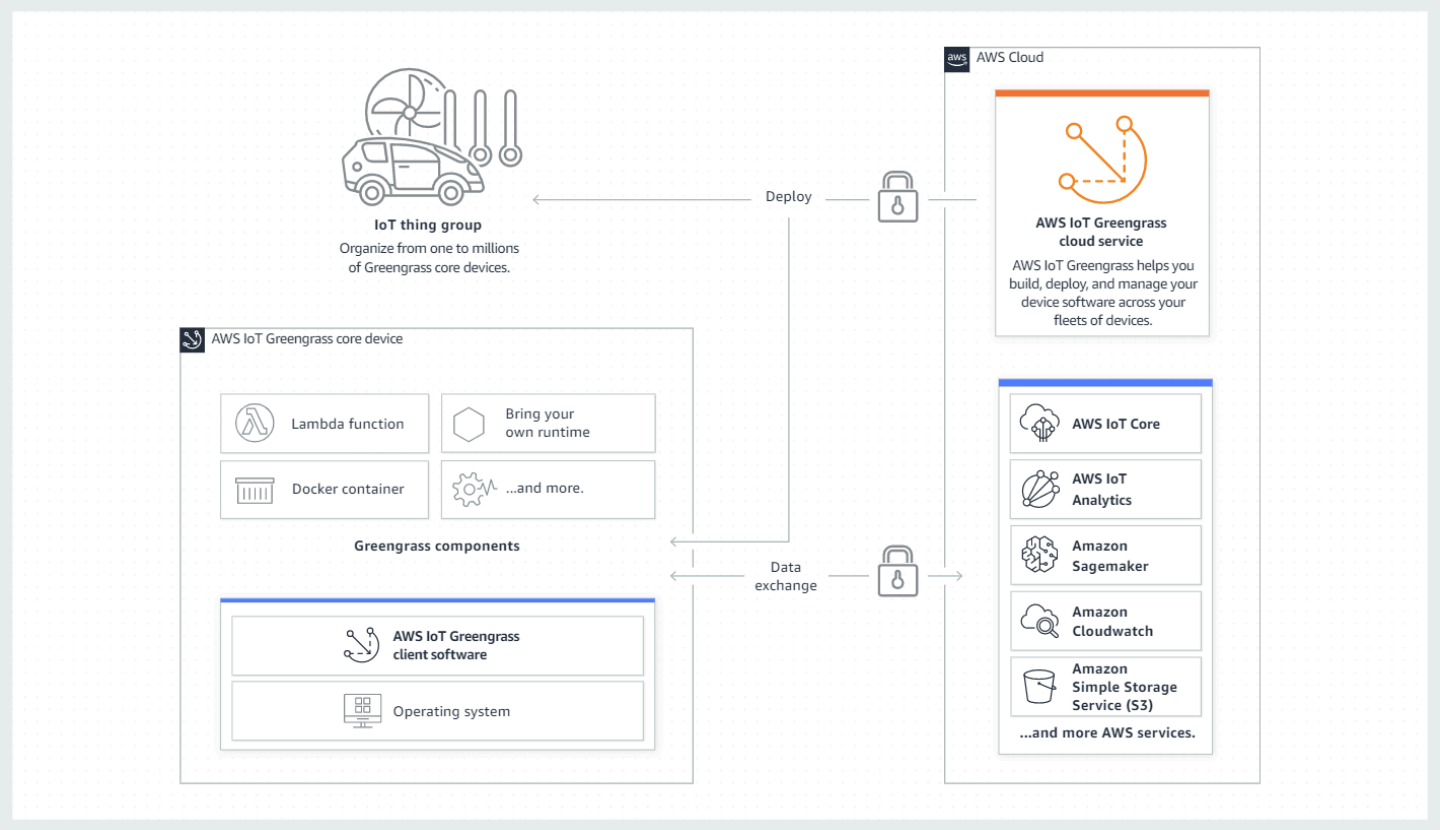
- AWS IoT Greengrass버전 1에서 마이그레이션

AWS IoT Greengrass 작동 방식

AWS IoT GreengrassCore 소프트웨어라고도 하는 이 AWS IoT Greengrass 클라이언트 소프트웨어는 Windows 및 Linux 기반 배포판 (예: 우분투 또는 라즈베리 파이 OS) 에서 실행되며 ARM 또는 x86 아키텍처를 사용하는 장치의 경우 사용할 수 있습니다. 를 사용하면 생성되는 데이터에 대해 로컬에서 작동하도록 장치를 프로그래밍하고AWS IoT Greengrass, 기계 학습 모델을 기반으로 예측을 실행하고, 장치 데이터를 필터링 및 집계할 수 있습니다. AWS IoT GreengrassAWS Lambda함수, Docker 컨테이너, 네이티브 OS 프로세스 또는 원하는 사용자 지정 런타임을 로컬에서 실행할 수 있습니다.

AWS IoT Greengrass에지 디바이스 기능을 쉽게 확장할 수 있도록 구성 요소라고 하는 사전 빌드된 소프트웨어 모듈을 제공합니다. AWS IoT Greengrass구성 요소를 사용하면 엣지에서 AWS 서비스 및 타사 애플리케이션에 연결할 수 있습니다. IoT 애플리케이션을 개발한 후 현장의 AWS IoT Greengrass 여러 디바이스에서 해당 애플리케이션을 원격으로 배포, 구성 및 관리할 수 있습니다.

다음 예는 AWS IoT Greengrass 장치가 AWS IoT Greengrass 클라우드 서비스 및 의 다른 AWS 서비스와 상호 작용하는 방식을 보여줍니다. AWS 클라우드



AWS IoT Greengrass의 주요 개념

이해하고 사용하기 AWS IoT Greengrass 위한 필수 개념은 다음과 같습니다.

AWS IoT물건

AWS IoT사물은 특정 장치 또는 논리적 개체의 표현입니다. 사물에 대한 정보는 AWS IoT 레지스트리에 저장됩니다.

그린그래스 코어 디바이스

AWS IoT GreengrassCore 소프트웨어를 실행하는 기기. 그린그래스 코어 디바이스는 AWS IoT와 같습니다. AWS IoT사물 그룹에 여러 코어 디바이스를 추가하여 Greengrass 코어 디바이스 그룹을 생성하고 관리할 수 있습니다. 자세한 설명은 AWS IoT Greengrass코어 디바이스 설정 섹션을 참조하세요.

그린그래스 클라이언트 디바이스

MQTT를 통해 Greengrass 코어 디바이스에 연결하고 통신하는 디바이스입니다. Greengrass 클라이언트 기기는 정말 중요합니다. AWS IoT 코어 디바이스는 연결된 클라이언트 디바이스의 데이터를 처리, 필터링 및 집계할 수 있습니다. 클라이언트 장치, AWS IoT Core 클라우드 서비스 및

Greengrass 구성 요소 간에 MQTT 메시지를 릴레이하도록 코어 장치를 구성할 수 있습니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

클라이언트 장치는 FreeRTOS를 실행하거나 AWS IoT Device SDK또는 Greengrass 검색 API를 사용하여 연결할 수 있는 핵심 장치에 대한 정보를 가져올 수 있습니다.

Greengrass 구성 요소

Greengrass 코어 디바이스에 배포되고 Greengrass 코어 디바이스에서 실행되는 소프트웨어 모듈입니다. 와 함께 개발 및 배포되는 모든 소프트웨어는 AWS IoT Greengrass 구성 요소로 모델링됩니다. AWS IoT Greengrass애플리케이션에서 사용할 수 있는 기능을 제공하는 사전 구축된 공용 구성 요소를 제공합니다. 또한 로컬 장치나 클라우드에서 사용자 지정 구성 요소를 개발할 수 있습니다. 사용자 지정 구성 요소를 개발한 후에는 AWS IoT Greengrass 클라우드 서비스를 사용하여 단일 또는 다중 코어 장치에 배포할 수 있습니다. 사용자 지정 구성 요소를 만들고 해당 구성 요소를 코어 장치에 배포할 수 있습니다. 이렇게 하면 코어 기기는 다음 리소스를 다운로드하여 구성 요소를 실행합니다.

- 레시피: 구성 요소 세부 정보, 구성 및 매개 변수를 정의하여 소프트웨어 모듈을 설명하는 JSON 또는 YAML 파일입니다.
- Artifact: 기기에서 실행되는 소프트웨어를 정의하는 소스 코드, 바이너리 또는 스크립트. 처음부터 아티팩트를 생성하거나 Lambda 함수, Docker 컨테이너 또는 사용자 지정 런타임을 사용하여 구성 요소를 생성할 수 있습니다.
- 종속성: 구성 요소 간의 관계로, 종속 구성 요소의 자동 업데이트 또는 재시작을 적용할 수 있습니다. 예를 들어, 암호화 구성 요소에 종속된 보안 메시지 처리 구성 요소를 가질 수 있습니다. 이렇게 하면 암호화 구성 요소를 업데이트하면 메시지 처리 구성 요소가 자동으로 업데이트되고 다시 시작됩니다.

자세한 내용은 AWS-제공된 구성 요소 및 AWS IoT Greengrass구성 요소 개발 섹션을 참조하세요.

배포

구성 요소를 전송하고 원하는 구성 요소 구성을 대상 장치 (단일 Greengrass 코어 장치 또는 Greengrass 코어 장치 그룹일 수 있음) 에 적용하는 프로세스입니다. 배포는 업데이트된 구성 요소 구성을 대상에 자동으로 적용하고 종속성으로 정의된 다른 구성 요소를 모두 포함합니다. 기존 배포를 복제하여 동일한 구성 요소를 사용하지만 다른 대상에 배포되는 새 배포를 만들 수도 있습니다. 배포는 지속적이므로 배포의 구성 요소 또는 구성 요소 구성에 대한 모든 업데이트가 자동으로 모든 대상에 전송됩니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

AWS IoT Greengrass코어 소프트웨어

코어 디바이스에 설치하는 모든 AWS IoT Greengrass 소프트웨어 세트. AWS IoT Greengrass 핵심 소프트웨어는 다음과 같이 구성됩니다.

- Nucleus: 이 필수 구성 요소는 AWS IoT Greengrass Core 소프트웨어의 최소 기능을 제공합니다. NUCLEUS는 다른 구성 요소의 배포, 오케스트레이션 및 라이프사이클 관리를 관리합니다. 또한 개별 장치의 로컬 AWS IoT Greengrass 구성 요소 간 통신을 용이하게 합니다. 자세한 설명은 그린그래스 핵 섹션을 참조하세요.
- 옵션 구성 요소: 이러한 구성 가능한 구성 요소는 에지 장치에서 제공되며 에지 장치에서 추가 기능을 사용할 수 있습니다. AWS IoT Greengrass 요구 사항에 따라 데이터 스트리밍, 로컬 기계 학습 추론 또는 로컬 명령줄 인터페이스와 같이 장치에 배포할 선택적 구성 요소를 선택할 수 있습니다. 자세한 설명은 AWS-제공된 구성 요소 섹션을 참조하세요.

새 버전의 구성 요소를 디바이스에 배포하여 AWS IoT Greengrass Core 소프트웨어를 업그레이드할 수 있습니다.

AWS IoT Greengrass의 기능

AWS IoT Greengrass Version 2다음 요소로 구성됩니다.

- 소프트웨어 배포판
  - Greengrass 핵 구성 요소는 코어 소프트웨어의 최소 설치입니다. AWS IoT Greengrass 이 구성 요소는 Greengrass 구성 요소의 배포, 오케스트레이션 및 라이프사이클 관리를 관리합니다.
  - 서비스, 프로토콜 AWS및 소프트웨어와 통합되는 추가 옵션 제공 구성 요소.
  - Greengrass 개발 도구: 사용자 지정 Greengrass 구성 요소를 만들고, 테스트하고, 빌드하고, 게시하고, 배포하는 데 사용할 수 있습니다.
  - 에는 사용자 AWS IoT Device SDK 지정 Greengrass 구성 요소에 대한 IPC (프로세스 간 통신) 라이브러리와 클라이언트 장치용 Greengrass 검색 라이브러리가 포함되어 있습니다.
  - 코어 디바이스의 데이터 스트림을 관리하는 데 사용할 수 있는 스트림 관리자 SDK.
- 클라우드 서비스
  - AWS IoT Greengrass V2 API
  - AWS IoT Greengrass V2 콘솔

AWS IoT Greengrass 코어 소프트웨어

에지 디바이스에서 실행되는 AWS IoT Greengrass Core 소프트웨어를 사용하여 다음 작업을 수행할 수 있습니다.

- AWS클라우드로 자동 내보내기를 통해 로컬 장치의 데이터 스트림을 처리합니다. 자세한 설명은 Greengrass 코어 디바이스의 데이터 스트림 관리 섹션을 참조하세요.
- AWS IoT및 구성 요소 간의 MQTT 메시징을 지원합니다. 자세한 설명은 MQTT 메시지 게시/구독 AWS IoT Core 섹션을 참조하세요.
- MQTT를 통해 연결 및 통신하는 로컬 장치와 상호 작용할 수 있습니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.
- 구성 요소 간 로컬 게시 및 구독 메시지를 지원합니다. 자세한 설명은 로컬 메시지 게시/구독 섹션을 참조하세요.
- 구성 요소 및 Lambda 함수를 배포하고 호출합니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.
- 설치 및 실행 스크립트 지원 등 구성 요소 수명 주기를 관리합니다. 자세한 설명은 AWS IoT Greengrass컴포넌트 레시피 참조 섹션을 참조하세요.
- AWS IoT GreengrassCore 소프트웨어 및 사용자 지정 구성 요소의 보안 over-the-air (OTA) 소프트웨어 업데이트를 수행합니다. 자세한 내용은 AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트 및 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.
- 로컬 비밀의 안전하고 암호화된 스토리지를 제공하고 구성 요소별로 액세스를 제어합니다. 자세한 설명은 시크릿 매니저 섹션을 참조하세요.
- 디바이스 인증 및 권한 부여를 통해 디바이스와 AWS 클라우드 간의 연결을 보호합니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.

지속적인 소프트웨어 배포를 생성하는 AWS IoT Greengrass API를 통해 Greengrass 코어 디바이스를 구성하고 관리합니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

일부 기능은 특정 플랫폼에서만 지원됩니다. 자세한 설명은 운영 체제별 Greengrass 기능 호환성 섹션을 참조하세요.

지원되는 플랫폼, 요구 사항 및 다운로드에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass 코어 디바이스 설정.

이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

운영 체제별 Greengrass 기능 호환성

AWS IoT Greengrass 다양한 운영 체제를 실행하는 장치를 지원합니다. 일부 기능은 특정 운영 체제에서만 지원됩니다. 다음 표를 참조하여 지원되는 각 운영 체제에서 사용할 수 있는 기능을 알아보십시오. 지원되는 운영 체제, 요구 사항 및 Greengrass 코어 장치 설정 방법에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass코어 디바이스 설정

메시지 전송

기능	Linux	Windows
및 구성 요소 간에 AWS IoT MQTT 메시지를 교환합니다.	예.	네
구성 요소 간에 로컬 게시/구독 메시지를 교환합니다.	예	네
MQTT를 통해 로컬 IoT 디바이스와 상호 작용	예.	네
Modbus-RTU 컴포넌트를 사용하여 로컬 Modbus-RTU 장치와 상호 작용하십시오.	네.	아니요

보안

기능	Linux	Windows
장치 인증 및 권한 부여를 통한 보안 연결	예	네

기능	Linux	Windows
에서 안전하고 암호화된 비밀을 배포하고 액세스할 수 있습니다. AWS Secrets Manager	예	네
하드웨어 보안 모듈 (HSM) 을 사용하여 기기의 개인 키와 인증서를 안전하게 저장하세요.	예	아니요
다음과 같은 핵심 장치를 감사하십시오. AWS IoT Device Defender	예	네
AWS 자격 증명을 사용하여 AWS 서비스와 상호작용하세요	예	네

설치

기능	Linux	Windows
자동 AWS IoT Greengrass 프로비저닝으로 설치	예	네
수동 AWS IoT Greengrass 프로비저닝으로 설치	예	네
AWS IoT 플릿 AWS IoT Greengrass 프로비저닝으로 설치	예	네

기능	Linux	Windows
사용자 지정 프로비저닝 AWS IoT Greengrass 플러그인으로 설치	 예	 네
사전 빌드된 Docker 이미지를 사용하여 Docker AWS IoT Greengrass 컨테이너에서 실행합니다.	 예	 아니요

원격 유지 관리 및 업데이트

기능	Linux	Windows
보안 over-the-air (OTA) 소프트웨어 업데이트 수행	 예	 네
다음을 사용하여 핵심 장치를 관리합니다. AWS Systems Manager	 예	 아니요
AWS IoT 보안 터널링으로 코어 디바이스에 연결	 예	 아니요

기계 학습

기능	Linux	Windows
Amazon SageMaker Edge Manager를 사용하여 기계 학습 추론 수행	 예	 네

기능	Linux	Windows
Amazon Lookout for Vision을 사용하여 기계 학습 추론 수행	네.	아니요
DLR을 사용하여 머신 러닝 추론 수행	예.	네
다음을 사용하여 기계 학습 추론을 수행합니다. TensorFlow	예	네

구성 요소 기능

기능	Linux	Windows
Lambda 함수 배포 및 호출	예.	아니요
구성 요소에서 Docker 컨테이너 실행	예	네
스트림 관리자를 사용하여 대용량 데이터 스트림을 처리하고 내보냅니다.	예	네
라이프사이클 스크립트로 구성 요소 라이프사이클을 관리합니다.	예	네

기능	Linux	Windows
디바이스 섀도우와 상호작용하세요.	예.	네
Amazon 로그에 CloudWatch 로그 업로드	예.	네
CloudWatch 지표 구성 요소를 사용하여 Amazon CloudWatch 지표에 데이터 업로드	예	네
Amazon SNS 구성 요소를 사용하여 Amazon 단순 알림 서비스에 메시지를 게시합니다.	예.	아니요
스트림 관리자를 사용하여 Amazon Data Firehose 전송 스트림에 데이터를 게시합니다.	예.	네
Firehose 구성 요소를 사용하여 Amazon Data Firehose 전송 스트림에 데이터를 게시합니다.	예.	아니요
실시간 시스템 텔레메트리 메트릭을 수집하고 이에 따라 조치를 취하십시오.	예	네
구성 요소 프로세스에 대한 시스템 리소스 제한을 구성합니다.	예	아니요

기능	Linux	Windows
구성 요소 프로세스 일시 중지 및 재개	 예	 아니요
AWS IoT SiteWise 구성 요소를 AWS IoT SiteWise 사용하여 통합하십시오.	 예	 네
Kinesis Video Streams 구성 요소용 에지 커넥터를 사용하여 Amazon Kinesis Video Streams에 비디오 스트림을 게시하십시오.	 예.	 아니요

컴포넌트 개발

기능	Linux	Windows
코어 디바이스에서 로컬로 구성 요소 개발	 예.	 네
AWS IoT Greengrass CLI를 사용하여 코어 디바이스와 상호작용	 예	 네
로컬 디버그 콘솔을 사용하여 코어 기기와 상호작용하세요.	 예	 네
사용자 지정 구성 요소에서 AWS IoT Device SDK Python 용 사용	 네	 네

기능	Linux	Windows
사용자 지정 구성 요소에서 AWS IoT Device SDK C++용 사용	 예	 네
사용자 지정 구성 요소에서 AWS IoT Device SDK Java용 사용	 예	 네

디바이스 인증

기능	Linux	Windows
IoT 장치 AWS IoT Device Tester AWS IoT Greengrass V2 검증에 사용	 예	 네

새로워진 내용 AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 는 다음 기능을 AWS IoT Greengrass 도입한 주요 버전입니다.

- 게시자 지원 구성 요소 — AWS IoT Greengrass 이제 게시자 지원 구성 요소를 제공합니다. 이러한 구성 요소는 타사 공급업체에서 개발, 제공 및 서비스를 제공합니다. 자세한 설명은 게시자 지원 구성 요소 섹션을 참조하세요.
- VPC에서 Greengrass 디바이스 운영 — 이제 VPC에서 Greengrass 코어 디바이스를 운영할 수 있습니다. 이를 통해 공용 인터넷 액세스 없이 VPC에서 배포를 수행할 수 있습니다. 자세한 설명은 VPC에서 AWS IoT Greengrass 코어 디바이스 운영 섹션을 참조하세요.
- Greengrass 테스트 프레임워크 (GTF) — GTF를 이제 사용할 AWS IoT Greengrass Version 2 수 있습니다. GTF는 자동화를 지원하는 빌딩 블록 모음입니다. end-to-end 이를 통해 AWS IoT Greengrass Version 2 내부 고객은 서비스 팀이 소프트웨어 변경 검증, 자동 승인 및 품질 보증 목적으로 사용하는 것과 동일한 테스트 프레임워크를 사용할 수 있습니다. 자세한 내용은 Github의 Greengrass 테스트 프레임워크를 참조하십시오.
- PSA 인증 — AWS IoT Greengrass nucleus 버전 2.7.0 이상은 이제 플랫폼 보안 아키텍처 (PSA) 인증을 받았습니다. 자세한 내용은 PSA 인증을 참조하십시오. AWS IoT Greengrass

AWS IoT Greengrass 릴리스 노트에서는 AWS IoT Greengrass 새 기능, 업데이트 및 개선 사항, 일반 수정 사항 등 릴리스에 대한 세부 정보를 제공합니다. AWS IoT Greengrass 에는 다음과 같은 유형의 릴리스가 있습니다.

- 에 대한 새 기능 출시 AWS IoT Greengrass
- AWS IoT Greengrass 코어 소프트웨어 업데이트

이 섹션에는 최신 AWS IoT Greengrass V2 릴리즈 노트와 주요 기능 변경 사항 및 중요한 버그 수정이 모두 포함되어 있습니다. 추가적인 사소한 수정에 대한 자세한 내용은 의 aws-greengrass 조직을 참조하십시오. GitHub

릴리스 정보

- 릴리스: AWS IoT Greengrass 2024년 2월 15일 코어 v2.12.2 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 12월 8일 코어 v2.12.1 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 11월 7일 코어 v2.12.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 10월 18일 코어 v2.11.3 소프트웨어 업데이트

- 릴리스: AWS IoT Greengrass 2023년 8월 9일 코어 v2.11.2 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 7월 21일 코어 v2.11.1 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 6월 28일에 코어 v2.11.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 6월 21일 코어 v2.10.3 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 6월 5일 코어 v2.10.2 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 5월 11일 코어 v2.10.1 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 5월 9일 코어 v2.10.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 4월 20일 코어 v2.9.6 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 3월 30일 코어 v2.9.5 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 2월 24일 코어 v2.9.4 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2023년 2월 1일 코어 v2.9.3 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 12월 22일 코어 v2.9.2 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 11월 18일 코어 v2.9.1 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 11월 15일 코어 v2.9.0 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 10월 13일 코어 v2.8.1 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 10월 7일 코어 v2.8.0 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 7월 28일 코어 v2.7.0 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 6월 27일 코어 v2.6.0 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 5월 31일 코어 v2.5.6 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 4월 6일 코어 v2.5.5 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 3월 23일 코어 v2.5.4 소프트웨어 업데이트
- 출시: AWS IoT Greengrass 2022년 1월 6일 코어 v2.5.3 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 12월 3일 코어 v2.5.2 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 11월 23일 코어 v2.5.1 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 11월 12일에 코어 v2.5.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 8월 3일 코어 v2.4.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 6월 29일 코어 v2.3.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 6월 18일 코어 v2.2.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 4월 26일 코어 v2.1.0 소프트웨어 업데이트
- 릴리스: AWS IoT Greengrass 2021년 3월 9일 코어 v2.0.5 소프트웨어 업데이트

- 릴리스: AWS IoT Greengrass 2021년 2월 4일 코어 v2.0.4 소프트웨어 업데이트

릴리스: AWS IoT Greengrass 2024년 2월 15일 코어 v2.12.2 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.2와 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2024년 2월 15일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass Core 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.12.2를 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 • 이전 로그가 제대로 정리되지 않던 문제를 수정합니다. • 일반적인 버그 수정 및 개선입니다.
섀도우 매니저	섀도우 매니저 구성 요소 버전 2.3.6을 사용할 수 있습니다. 버그 수정 및 개선 장치가 오프라인 상태일 때 AWS 클라우드 업데이트를 통해 삭제된 섀도우 속성이 다시 연결되더라도 로컬 섀도우에 계속 남아 있는 문제를 수정합니다.
람다 런처	람다 런처 구성 요소 버전 2.0.13을 사용할 수 있습니다. 버그 수정 및 개선 일반적인 버그 수정 및 개선입니다.
디스크 스풀러	디스크 스풀러 구성 요소 버전 1.0.3을 사용할 수 있습니다. 버그 수정 및 개선 데이터베이스 연결을 재사용하여 성능을 개선합니다.

릴리스: AWS IoT Greengrass 2023년 12월 8일 코어 v2.12.1 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.1과 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2023년 12월 8일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.12.1을 사용할 수 있습니다. 버그 수정 및 개선 • NUCLEUS가 배포 주제에 대한 MQTT 구독을 중복하여 추가 로깅 및 MQTT 게시로 이어질 수 있는 문제를 수정합니다.
클라이언트 장치 인증	클라이언트 장치 인증 구성 요소 버전 2.4.5를 사용할 수 있습니다. 새로운 기능 매개변수로 사물 이름을 선택하기 위한 와일드카드 접두사 지원을 추가합니다. `selectionRule` 버그 수정 및 개선 특정 경우에 인증서가 새 연결 정보로 업데이트되지 않는 문제를 수정합니다.
디스크 스풀러	디스크 스풀러 구성 요소 버전 1.0.2를 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 특정 경우에 MQTT 메시지 형식 필드가 유지되지 않는 문제를 수정합니다.
MQTT 브리지	디스크 스풀러 구성 요소 버전 2.3.1을 사용할 수 있습니다. 버그 수정 및 개선 로컬 MQTT 클라이언트가 연결 해제 루프에 빠지는 문제를 수정합니다.
스트림 매니저	스트림 관리자 구성 요소 버전 2.1.12를 사용할 수 있습니다. 버그 수정 및 개선 Greengrass 자격 증명이 AWS 서비스 요청 시 선호되도록 자격 증명이 사용되는 순서를 업데이트합니다.

릴리스: AWS IoT Greengrass 2023년 11월 7일 코어 v2.12.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.0과 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2023년 11월 7일

출시 하이라이트

- 롤백 시 부트스트랩 — AWS IoT Greengrass 이제 라는 Greengrass 핵 구성 매개변수를 제공합니다. `BootstrapOnRollback` 이 기능을 사용하면 롤백 배포의 일부로 부트스트랩 수명 주기 단계를 실행할 수 있습니다.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.12.0을 사용할 수 있습니다. 새로운 기능 • 롤백 배포의 일부로 부트스트랩 수명 주기 단계를 실행할 수 있습니다.

릴리스: AWS IoT Greengrass 2023년 10월 18일 코어 v2.11.3 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.11.3을 제공합니다.

릴리스 날짜: 2023년 10월 18일

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.11.3을 사용할 수 있습니다. 버그 수정 및 개선 • 종속성이 실패할 경우 구성 요소가 제대로 시작되지 않을 수 있는 Nucleus 문제를 수정합니다. 새로운 기능 • 구성 가능한 s3 엔드포인트 유형을 추가합니다.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.3.1을 사용할 수 있습니다. 버그 수정 및 개선 • 특정 오류의 로그 수준을 조정합니다.
로컬 디버그 콘솔	Lambda 관리자 구성 요소 버전 2.4.0을 사용할 수 있습니다. 새로운 기능 • 스트림 관리자 디버깅 콘솔을 추가합니다.

구성 요소	세부 정보
로그 관리자	로그 관리자 구성 요소 버전 2.3.6을 사용할 수 있습니다. 버그 수정 및 개선 • 특정 오류의 로그 수준을 조정합니다.
섀도우 매니저	섀도우 매니저 컴포넌트 버전 2.3.4를 사용할 수 있습니다. 버그 수정 및 개선 • null 및 빈 섀도우 상태 문서에 대한 지원을 추가합니다.

릴리스: AWS IoT Greengrass 2023년 8월 9일 코어 v2.11.2 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.11.2를 제공합니다.

출시일: 2023년 8월 9일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT

GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.11.2를 사용할 수 있습니다. 버그 수정 및 개선 • Nucleus MQTT 5 클라이언트에서 많은 수 (50개 이상) 의 구독을 사용 중인 경우 오프라인으로 표시될 수 있는 문제를 수정합니다. • docker 다이얼 TCP 실패에 대한 재시도를 추가합니다.

릴리스: AWS IoT Greengrass 2023년 7월 21일 코어 v2.11.1 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.11.1을 제공합니다.

출시일: 2023년 7월 21일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.11.1을 사용할 수 있습니다. 버그 수정 및 개선 • 부트스트랩 작업이 실패하고 배포 메타데이터 파일이 손상된 경우 Nucleus가 시작되지 않는 문제를 수정합니다. • 온디맨드 Lambda 구성 요소가 배포 상태 업데이트에 보고되지 않는 문제를 수정합니다. • 중복 권한 부여 정책 ID에 대한 지원을 추가합니다.
람다 매니저	Lambda 관리자 버전 2.2.11을 사용할 수 있습니다. 버그 수정 및 개선 • Lambda LegacySubscriptionRouter 구성이 변경될 때 구성이 업데이트되지 않는 문제를 수정합니다.

릴리스: AWS IoT Greengrass 2023년 6월 28일에 코어 v2.11.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.11.0을 제공합니다.

출시일: 2023년 6월 28일

출시 하이라이트

- 영구 디스크 스풀러 — AWS IoT Greengrass 이제 Greengrass 코어 디바이스에서 스풀링된 메시지에 대한 영구 스풀러 구현을 제공합니다. AWS IoT Core 이 구성 요소는 이러한 아웃바운드 메시지를 디스크에 저장합니다. 자세한 설명은 디스크 스풀러 섹션을 참조하세요.

- 로컬 배포 개선 — 이제 로컬 배포를 취소하고, 배포 실패 처리 정책을 설정하고, 자세한 배포 상태를 확인할 수 있습니다.
- 로깅 속도 개선 — Log Manager 구성 요소의 로그 업로드 속도가 개선되었습니다.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.11.0을 사용할 수 있습니다. 새로운 기능 • 로컬 배포를 취소할 수 있습니다. • 로컬 배포를 위한 장애 처리 정책을 구성할 수 있습니다. • 디스크 스풀러 플러그인에 대한 지원을 추가합니다.
그린그래스 CLI	그린그래스 CLI의 버전 2.11.0을 사용할 수 있습니다.

구성 요소	세부 정보
	새로운 기능 • 로컬 배포를 취소할 수 있습니다. • 로컬 배포를 위한 장애 처리 정책을 구성할 수 있습니다. • 자세한 배포 상태 보고를 개선합니다.
디스크 스풀러	디스크 스풀러 구성 요소 버전 1.0.0을 사용할 수 있습니다. • 디스크 스풀러 구성 요소는 Greengrass 코어 디바이스에서 Greengrass 코어 디바이스로 전송된 메시지를 영구적으로 저장합니다. AWS IoT Core
로그 관리자	로그 관리자 구성 요소 버전 2.3.5를 사용할 수 있습니다. 개선 사항 로그 업로드 속도를 개선합니다.

릴리스: AWS IoT Greengrass 2023년 6월 21일 코어 v2.10.3 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.10.3을 제공합니다.

출시일: 2023년 6월 21일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하

는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.10.3을 사용할 수 있습니다. 버그 수정 및 개선 • Greengrass가 PKCS #11 제공자를 사용할 때 배포 알림을 구독하지 않는 문제를 수정합니다.

릴리스: AWS IoT Greengrass 2023년 6월 5일 코어 v2.10.2 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.10.2를 제공합니다.

출시일: 2023년 6월 5일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하

는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.10.2를 사용할 수 있습니다. 버그 수정 및 개선 • 구성 요소 수명 주기의 대소문자를 구분하지 않는 구문 분석을 허용합니다. • 환경 PATH 변수가 올바르게 다시 생성되지 않던 문제를 수정합니다. • 특수 문자가 있는 사용자 이름의 스트림 관리자를 비롯한 구성 요소의 프록시 URI 인코딩을 수정합니다.
클라이언트 장치 인증	클라이언트 장치 인증 구성 요소 버전 2.4.2를 사용할 수 있습니다. 새로운 기능 새 `startupTimeoutSeconds` 구성 옵션을 추가합니다.
Lambda 관리자	Lambda 관리자 구성 요소 버전 2.2.9를 사용할 수 있습니다. 버그 수정 및 개선 시계가 왜곡되어 포트 번호가 손상되는 문제를 수정합니다.
로그 관리자	로그 관리자 구성 요소 버전 2.3.4를 사용할 수 있습니다. 버그 수정 및 개선 • `periodicUploadIntervalSec` 파라미터를 소수 값으로 설정하기 위한 서포트를 추가합니다. 최소값은 1마이크로초입니다.

구성 요소	세부 정보
	• 로그 관리자가 CloudWatch `putLogEvents` 제한을 준수하지 않는 문제를 수정합니다.
MQTT 3.1 브로커 (모켓)	MQTT 3.1 브로커 (모켓) 구성 요소 버전 2.3.3을 사용할 수 있습니다. 새로운 기능 새 구성 옵션을 추가합니다. `startupTimeoutSeconds`
MQTT 브리지	MQTT 브리지 구성 요소 버전 2.2.6을 사용할 수 있습니다. 새로운 기능 새 `startupTimeoutSeconds` 구성 옵션을 추가합니다.
스트림 매니저	스트림 관리자 구성 요소 버전 2.1.7을 사용할 수 있습니다. 버그 수정 및 개선 스트림 관리자가 프록시 구성을 제대로 읽지 못하는 문제를 수정합니다.

릴리스: AWS IoT Greengrass 2023년 5월 11일 코어 v2.10.1 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.10.1을 제공합니다.

출시일: 2023년 5월 11일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.10.1을 사용할 수 있습니다. 버그 수정 및 개선 • Jetson Nano를 비롯한 특정 ARMv8 프로세서에서 시작 시 충돌이 발생할 수 있는 문제를 수정합니다. • Greengrass는 더 이상 구성 요소의 표준을 닫지 않으므로 동작이 2.10.0 이전 동작으로 되돌아갑니다.
스트림 매니저	새 스트림 관리자 버전 2.1.6을 사용할 수 있습니다. 버그 수정 및 개선 Jetson Nano를 비롯한 특정 ARMv8 프로세서에서 시작 시 충돌이 발생할 수 있는 문제를 수정합니다.

릴리스: AWS IoT Greengrass 2023년 5월 9일 코어 v2.10.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.10.0과 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2023년 5월 9일

출시 하이라이트

- MQTT5 지원 — AWS IoT Greengrass 이제 MQTT5 AWS IoT Core 사용을 통한 메시지 송수신을 지원합니다. 자세한 내용은 AWS IoT CoreMQTT 메시지 게시를 참조하십시오.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.10.0을 사용할 수 있습니다. 새로운 기능 • 빈 정규 `interpolateComponentConfiguration` 표현식에 대한 지원을 추가합니다. Greengrass는 이제 루트 구성 객체에서 보간합니다. • MQTT5 지원을 추가합니다.

구성 요소	세부 정보
	• 플러그인 구성 요소를 스캔하지 않고 빠르게 로드할 수 있는 메커니즘을 추가합니다. • Greengrass가 사용하지 않는 Docker 이미지를 삭제하여 디스크 공간을 절약할 수 있도록 합니다. 버그 수정 및 개선 • 롤백으로 인해 배포의 특정 구성 값이 그대로 유지되는 문제를 수정합니다. • Greengrass Nucleus가 사용자 지정 AWS 비자격 증명 및 데이터 엔드포인트에서 AWS 도메인 시퀀스를 검증하는 문제를 수정합니다. • 활성 버전으로 잠그는 대신 AWS 클라우드 협상을 통해 모든 그룹 종속성을 다시 해결하도록 다중 그룹 종속성 해결을 업데이트합니다. 또한 이 업데이트는 배포 오류 코드도 제거합니다. `INSTALLED_COMPONENT_NOT_FOUND` • Docker 이미지가 이미 로컬에 있는 경우 다운로드를 건너뛰도록 Greengrass 핵을 업데이트합니다. • 제한 시간이 만료되기 전에 구성 요소 설치 단계를 다시 시작하도록 Greengrass Nucleus를 업데이트합니다. • 추가 사소한 수정 및 개선
섀도우 매니저	새 섀도우 매니저 버전 2.3.2를 사용할 수 있습니다. 버그 수정 및 개선 로컬 섀도우 데이터베이스가 손상되면 섀도우 관리자가 `BROKEN` 상태로 전환되는 문제를 수정합니다.

릴리스: AWS IoT Greengrass 2023년 4월 20일 코어 v2.9.6 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소 버전 2.9.6을 제공합니다.

릴리스 날짜: 2023년 4월 20일

릴리스 세부 정보

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.6을 사용할 수 있습니다. 버그 수정 및 개선 • LAUNCH_DIRECTORY_CORRTED 오류와 함께 그린그래스 배포가 실패하고 이후 디바이스 재부팅 시 Greengrass가 시작되지 않는 문제를 수정합니다. 이 오류는 Greengrass를 다시 시작해야 하는 배포가 있는 여러 사물 그룹 간에 Greengrass 디바이스를 이동할 때 발생할 수 있습니다.

릴리스: AWS IoT Greengrass 2023년 3월 30일 코어 v2.9.5 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.9.5를 제공합니다.

출시일: 2023년 3월 30일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.5를 사용할 수 있습니다. 새로운 기능 • Greengrass 핵 소프트웨어 서명 검증에 대한 지원을 추가합니다. 버그 수정 및 개선 • 로컬 레시피 메타데이터 영역이 Greengrass nucleus 시작 지역과 일치하지 않는 경우 배포가 실패하는 문제를 수정합니다. Greengrass 핵은 이제 이런 일이 발생하면 클라우드와 재협상합니다. • MQTT 메시지 스풀러가 꽉 차서 메시지를 제거하지 않는 문제를 수정합니다. • 추가 사소한 수정 및 개선.

릴리스: AWS IoT Greengrass 2023년 2월 24일 코어 v2.9.4 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.9.4를 제공합니다.

출시일: 2023년 2월 24일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.4를 사용할 수 있습니다. 버그 수정 및 개선 • QOS 0 메시지를 삭제하기 전에 null 메시지가 있는지 확인합니다. • 작업 상태 세부 정보 값이 1024자 제한을 초과하는 경우 해당 값을 잘라냅니다.

구성 요소	세부 정보
	• Greengrass 루트 경로에 공백이 포함된 경우 Greengrass 루트 경로를 올바르게 읽도록 Windows용 부트스트랩 스크립트를 업데이트합니다. • 구독 응답이 전송되지 않은 경우 클라이언트 메시지가 AWS IoT Core 삭제되도록 구독을 업데이트합니다. • 기본 구성 파일이 손상되거나 누락된 경우 Nucleus가 백업 파일에서 구성을 로드하도록 합니다.

릴리스: AWS IoT Greengrass 2023년 2월 1일 코어 v2.9.3 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소 버전 2.9.3을 제공합니다.

출시일: 2023년 2월 1일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하여 에서 AWS 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.3을 사용할 수 있습니다. 버그 수정 및 개선 • MQTT 클라이언트 ID가 중복되지 않도록 합니다. • 손상을 방지하고 복구할 수 있도록 보다 강력한 파일 읽기 및 쓰기 기능을 추가합니다. • 특정 네트워크 관련 오류가 발생하면 docker image pull을 재시도합니다. • MQTT 연결 `noProxyAddresses` 옵션을 추가합니다.

출시: AWS IoT Greengrass 2022년 12월 22일 코어 v2.9.2 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.9.2를 제공합니다.

출시일: 2022년 12월 22일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT

GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.2를 사용할 수 있습니다. 버그 수정 및 개선 • 구성이 진행 중인 배포에 적용되지 `interpolateComponentConfiguration` 않는 문제를 수정합니다. • OSHI를 사용하여 모든 하위 프로세스를 나열합니다.

출시: AWS IoT Greengrass 2022년 11월 18일 코어 v2.9.1 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.1과 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

릴리스 날짜: 2022년 11월 18일

릴리스 하이라이트

- 로그 관리자 — 이제 로그 관리자가 새 파일이 교체될 때까지 기다리지 않고 활성 로그 파일을 처리하고 직접 업로드합니다. 이 개선으로 로그 지연이 크게 줄어듭니다. 자세한 내용은 로그 관리자를 참조하십시오.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.1을 사용할 수 있습니다. 버그 수정 및 개선 • 디플로이먼트에서 플러그인 컴포넌트가 제거되면 Greengrass가 다시 시작되는 문제를 수정했습니다.
로그 매니저	새 로그 관리자 버전 2.3.0을 사용할 수 있습니다. **Note** 로그 관리자 2.3.0으로 업그레이드할 때는 Greengrass nucleus 2.9.1로 업그레이드하는 것이 좋습니다. 새로운 기능 • 새 파일이 교체될 때까지 기다리지 않고 활성 로그 파일을 처리하고 직접 업로드하여 로그 지연을 줄입니다. 버그 수정 및 개선 • 고유한 이름을 가진 파일을 회전할 때 로그 로테이션 지원을 개선합니다. • 추가 사소한 수정 및 개선

출시: AWS IoT Greengrass 2022년 11월 15일 코어 v2.9.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.0과 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2022년 11월 15일

출시 하이라이트

- 오프라인 인증 — AWS IoT Greengrass 이제 오프라인 인증을 지원합니다. AWS IoT Greengrass코어 디바이스가 클라우드에 연결되어 있지 않아도 클라이언트 디바이스를 코어 디바이스에 연결할 수 있도록 코어 디바이스를 구성할 수 있습니다. 자세한 내용은 오프라인 인증을 참조하십시오.
- 하위 배포 — 이제 하위 배포를 만들 수 있습니다. 하위 배포를 사용하여 실패한 배포를 해결할 수 있습니다. 각 하위 배포는 소수의 기기 하위 집합에서 실패한 배포의 다양한 구성을 테스트할 수 있습니다. 자세한 내용은 하위 배포 만들기를 참조하십시오.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.9.0을 사용할 수 있습니다. 새로운 기능 • 더 적은 수의 장치로 배포를 다시 시도하는 하위 배포를 생성하는 기능을 추가합니다. 이 기능을 사용하면 실패한 배포를 보다 효율적으로 테스트하고 해결할 수 있습니다. 버그 수정 및 개선 • `useraddgroupadd`, 및 가 없는 시스템에 대한 지원을 `usermod` 개선합니다. • 기타 사소한 수정 및 개선
클라이언트 장치 인증	클라이언트 장치 인증 구성 요소 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 • 클라이언트 장치의 오프라인 인증에 대한 지원을 추가합니다. 이 기능을 사용하면 코어 장치가 인터넷에 연결되어 있지 않아도 클라이언트 장치를 코어 장치에 계속 연결할 수 있습니다. • 고객이 제공한 인증 기관 (CA) 에 대한 지원을 추가합니다. 코어 디바이스는 고객이 제공한 CA를 루트 인증서로 사용하여 MQTT 브로커 인증서를 생성합니다.
MQTT 5 브로커 (EMQX)	구성 요소 버전 1.2.0을 맷 5 브로커 (EMQX) 사용할 수 있습니다. 새로운 기능 인증서 체인에 대한 지원을 추가합니다.
모켓 MQTT 브로커	새로운 모켓 MQTT 브로커 컴포넌트의 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 인증서 체인에 대한 지원을 추가합니다.
시크릿 매니저	새 시크릿 매니저 버전 2.1.4를 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 시크릿 매니저가 배포되고 Greengrass nucleus가 다시 시작될 때 캐시된 비밀이 제거되던 문제를 수정합니다.
스트림 매니저	새 스트림 관리자 버전 2.1.2를 사용할 수 있습니다. 버그 수정 및 개선 영어가 아닌 언어를 사용하는 Windows OS의 문제를 수정합니다.

출시: AWS IoT Greengrass 2022년 10월 13일 코어 v2.8.1 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소의 버전 2.8.1을 제공합니다.

출시일: 2022년 10월 13일

> **Note**
> 그린그래스 핵 버전 2.8.0을 사용하는 경우 그린그래스 핵 버전 2.8.1로 업그레이드하는 것이 좋습니다.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로

배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.8.1을 사용할 수 있습니다. 버그 수정 및 개선 • Greengrass API 오류에서 배포 오류 코드가 올바르게 생성되지 않던 문제를 수정합니다. • 구성 요소가 배포 중에 상태에 도달하면 플릿 상태 업데이트가 부정확한 정보를 전송하는 문제를 수정합니다. ERRORED • Greengrass의 기존 구독이 50개 이상인 경우 배포를 완료할 수 없었던 문제를 수정합니다.

출시: AWS IoT Greengrass 2022년 10월 7일 코어 v2.8.0 소프트웨어 업데이트

이 릴리스는 그린그래스 핵 구성 요소 버전 2.8.0과 MQTT 5 브로커 (EMQX) 구성 요소 버전 1.1.0을 제공합니다.

출시일: 2022년 10월 7일

출시 하이라이트

- 배포 오류 코드 — Greengrass nucleus는 이제 구성 요소 배포를 완료할 수 없는 경우 자세한 오류 코드가 포함된 배포 상태 응답을 보고합니다. 자세한 설명은 세부 배포 오류 코드 섹션을 참조하세요.
- 구성 요소 오류 상태 — 이제 Greengrass 핵은 구성 요소가 또는 상태에 들어갈 때 자세한 오류 상태를 포함하는 구성 요소 상태 응답을 보고합니다. BROKEN ERRORED 자세한 설명은 세부 구성 요소 상태 코드 섹션을 참조하세요.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.8.0을 사용할 수 있습니다. 새로운 기능 • Greengrass nucleus를 업데이트하여 핵심 장치에 구성 요소를 배포하는 데 문제가 발생할 경우 자세한 오류 코드가 포함된 배포 상태 응답을 보고합니다. 자세한 설명은 세부 배포 오류 코드 섹션을 참조하세요. • 구성 요소가 또는 상태에 들어갈 때 자세한 오류 코드가 포함된 구성 요소 상태 응답을 보고하도록 Greengrass Nucleus를 업데이트합니다BROKEN. ERRORED 자세한 설명은 세부 구성 요소 상태 코드 섹션을 참조하세요. • 상태 메시지 필드를 확장하여 디바이스의 클라우드 가용성 정보를 개선합니다. • 폴링 상태 서비스의 견고성을 개선합니다.

구성 요소	세부 정보
	버그 수정 및 개선 • 구성이 변경될 때 손상된 구성 요소를 다시 설치할 수 있도록 합니다. • 부트스트랩 배포 중 Nucleus 재시작으로 인해 배포가 실패하는 문제를 수정합니다. • Windows에서 루트 경로에 공백이 있는 경우 설치가 실패하는 문제를 수정합니다. • 배포 중에 종료된 구성 요소가 새 버전의 종료 스크립트를 사용하는 문제를 수정합니다. • 다양한 종료 개선 사항. • 추가 사소한 수정 및 개선
MQTT 5 브로커 (EMQX)	구성 요소 버전 1.1.0을 맷 5 브로커 (EMQX) 사용할 수 있습니다. 새로운 기능 • 브로커 옵션 및 플러그인을 포함한 EMQX 구성에 대한 지원을 추가합니다. 버그 수정 및 개선 • EMQX를 버전 4.4.9로 업데이트합니다.

출시: AWS IoT Greengrass 2022년 7월 28일 코어 v2.7.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.7.0, 스트림 관리자 구성 요소 버전 2.1.0, Lambda 관리자 구성 요소 버전 2.2.5를 제공합니다.

릴리스 날짜: 2022년 7월 28일

릴리스 하이라이트

- 스트림 관리자 텔레메트리 지표 — 이제 Stream Manager가 자동으로 EventBridge Amazon에 텔레메트리 지표를 전송하므로 사용자는 코어 디바이스가 업로드하는 데이터의 양을 모니터링하고 분석하는 클라우드 애플리케이션을 만들 수 있습니다. 자세한 설명은 AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집 섹션을 참조하세요.

- 사용자 지정 인증 기관 (CA) — CA가 등록되지 않은 사용자 지정 인증서 CA에서 서명한 클라이언트 인증서가 이제 지원됩니다. AWS IoT 자세한 설명은 사설 CA에서 서명한 장치 인증서를 사용하십시오. 섹션을 참조하세요.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.7.0을 사용할 수 있습니다. 새로운 기능 • 코어 디바이스가 로컬 배포를 적용할 때 상태 업데이트를 AWS IoT Greengrass 클라우드로 전송하도록 Greengrass Nucleus를 업데이트합니다. • CA가 등록되지 않은 사용자 지정 인증 기관 (CA) 에서 서명한 클라이언트 인증서에 대한 지원을 추가합니다. AWS IoT 이 기능을 사용하려면 새 `greengrassDataPlaneEndpoint` 구성 옵션을 로 설정하면

구성 요소	세부 정보
	`iotdata` 됩니다. 자세한 설명은 사설 CA에서 서명한 장치 인증서를 사용하십시오. 섹션을 참조하세요. 버그 수정 및 개선 • 특정 시나리오에서 핵이 중지되거나 다시 시작될 때 Greengrass nucleus가 배포를 롤백하는 문제를 수정합니다. 이제 Nucleus가 재시작된 후 Nucleus는 배포를 재개합니다. • 소프트웨어를 시스템 서비스로 설정하도록 지정할 때 `--start` 인수를 준수하도록 Greengrass 설치 프로그램을 업데이트합니다. • nucleus가 구성 SubscribeToComponentUpdates 요소를 업데이트한 이벤트에서 배포 ID를 설정하는 동작을 업데이트합니다. • 추가 사소한 수정 및 개선
스트림 매니저	스트림 관리자 구성 요소 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • EventBridgeAmazon에 텔레메트리 지표를 자동으로 전송하도록 이 구성 요소를 업데이트합니다. 자세한 설명은 AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집 섹션을 참조하세요. 이 기능을 사용하려면 v2.7.0 이상의 Greengrass 핵 구성 요소가 필요합니다. • Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.2.5를 사용할 수 있습니다. 새로운 기능 • 로컬 게시/구독 메시지를 구독하는 이벤트 소스에 MQTT 주제 와일드카드 지원을 추가합니다. 이 기능을 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소가 필요합니다. • Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.

출시: AWS IoT Greengrass 2022년 6월 27일 코어 v2.6.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.6.0, 새로 제공된 구성 요소 및 AWS 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2022년 6월 27일

출시 하이라이트

- 로컬 게시/구독 주제의 와일드카드 — 이제 로컬 게시/구독 주제를 구독할 때 MQTT 와일드카드를 사용할 수 있습니다. 자세한 내용은 로컬 메시지 게시/구독 및 SubscribeToTopic 섹션을 참조하세요.
- 클라이언트 디바이스 섀도 지원 — 이제 사용자 지정 구성 요소의 클라이언트 디바이스 섀도와 상호 작용하고 클라이언트 디바이스 섀도우를 동기화할 수 있습니다. AWS IoT Core 자세한 설명은 클라이언트 디바이스 섀도우와 상호 작용 및 동기화 섹션을 참조하세요.
- 클라이언트 디바이스에 대한 로컬 MQTT 5 지원 — 이제 EMQX MQTT 5 브로커를 배포하여 클라이언트 디바이스와 코어 디바이스 간 통신에 MQTT 5 기능을 사용할 수 있습니다. 자세한 내용은 맷 5 브로커 (EMQX) 및 클라이언트 장치를 코어 장치에 연결 섹션을 참조하세요.
- 구성 요소 구성의 레시피 변수 — 이제 구성 요소 구성에서 특정 레시피 변수를 사용할 수 있습니다. 레시피에서 구성 요소의 기본 구성을 정의하거나 배포에서 구성 요소를 구성할 때 이러한 레시피 변수를 사용할 수 있습니다. 자세한 내용은 레시피 변수 및 병합 업데이트에 레시피 변수를 사용하십시오. 섹션을 참조하세요.
- IPC 권한 부여 정책의 와일드카드 — 이제 * 와일드카드를 사용하여 IPC (프로세스 간 통신) 권한 부여 정책의 모든 문자 조합을 일치시킬 수 있습니다. 이 와일드카드를 사용하면 단일 권한 부여 정책으로 여러 리소스에 대한 액세스를 허용할 수 있습니다. 자세한 설명은 권한 부여 정책의 와일드카드 섹션을 참조하세요.
- 로컬 배포 및 구성 요소를 관리하는 IPC 작업 - 이제 로컬 배포를 관리하고 구성 요소 세부 정보를 보는 사용자 지정 구성 요소를 개발할 수 있습니다. 자세한 내용은 IPC: 로컬 배포 및 구성 요소 관리를 참조하십시오.
- 클라이언트 장치를 인증하고 권한을 부여하는 IPC 작업 - 이제 이러한 작업을 사용하여 사용자 지정 로컬 브로커 구성 요소를 만들 수 있습니다. 자세한 내용은 IPC: 클라이언트 장치 인증 및 권한 부여를 참조하십시오.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.6.0을 사용할 수 있습니다. 새로운 기능 • 로컬 게시/구독 주제를 구독할 때 MQTT 와일드카드 지원을 추가합니다. 자세한 내용은 로컬 메시지 게시/구독 및 SubscribeToTopic 섹션을 참조하세요. • 레시피 변수 이외의 구성 요소 구성의 레시피 변수에 대한 지원을 추가합니다. *component_dependency_name* `:configuration:` *json_pointer* 레시피에 구성 요소를 정의하거나 `DefaultConfiguration` 배포에서 구성 요소를 구성할 때 이러한 레시피 변수를 사용할 수 있습니다. 이 기능을 활성화하려면 interpolateComponentConfiguration구성 옵션을 로 설정합니다`true`. 자세한 내용은 레시피 변수 및 병합 업데이트에 레시피 변수를 사용하십시오. 섹션을 참조하세요. • IPC (프로세스 간 통신) 권한 부여 정책에 * 와일드카드에 대한 전체 지원을 추가합니다. 이제 리소스 문자열의 문자를 모든 * 문자 조합과 일치하도록 지정할 수 있습니다. 자세한 설명은 권한 부여 정책의 와일드카드 섹션을 참조하세요.

구성 요소	세부 정보
	• Greengrass CLI에서 사용하는 IPC 작업을 호출하는 사용자 지정 구성 요소에 대한 지원을 추가합니다. 이러한 IPC 작업을 사용하여 로컬 배포를 관리하고, 구성 요소 세부 정보를 보고, 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 암호를 생성할 수 있습니다. 자세한 내용은 IPC: 로컬 배포 및 구성 요소 관리를 참조하십시오. 버그 수정 및 개선 • 특정 시나리오에서 종속 구성 요소의 하드 종속성이 다시 시작되거나 상태가 변경될 때 종속 구성 요소가 반응하지 않는 문제를 수정합니다. • 배포 실패 시 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 보고하는 오류 메시지를 개선합니다. • 특정 시나리오에서 핵이 재시작될 때 Greengrass 핵이 사물 배포를 두 번 적용한 문제를 수정합니다. • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
MQTT 5 브로커 (EMQX)	새로운 EMQX MQTT 5 브로커 구성 요소의 버전 1.0.0을 사용할 수 있습니다. 새로운 기능 • 로컬 EMQX MQTT 5 브로커에 대한 지원을 추가합니다. 클라이언트 디바이스는 이 MQTT 브로커에 연결하여 MQTT 5 기능을 사용하여 코어 디바이스와 통신할 수 있습니다.

구성 요소	세부 정보
섀도우 매니저	섀도우 매니저 구성 요소 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 • 로컬 게시/구독 인터페이스를 통해 로컬 섀도우 서비스에 대한 지원을 추가합니다. 이제 섀도우 MQTT 주제에 대해 로컬 게시/구독 메시지 브로커와 통신하여 코어 디바이스에서 섀도우를 가져오고, 업데이트하고, 삭제할 수 있습니다. 이 기능을 사용하면 MQTT 브리지를 사용하여 클라이언트 장치와 로컬 게시/구독 인터페이스 간에 섀도우 주제에 대한 메시지를 릴레이함으로써 클라이언트 장치를 로컬 섀도우 서비스에 연결할 수 있습니다. 이 기능을 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소가 필요합니다. 클라이언트 디바이스를 로컬 섀도우 서비스에 연결하려면 MQTT 브리지 구성 요소 v2.2.0 이상도 사용해야 합니다. • 로컬 섀도우 서비스와 에서 섀도우를 동기화하는 방향을 사용자 지정하도록 구성할 수 있는 `direction` 옵션을 추가합니다. AWS 클라우드 이 옵션을 구성하여 에 대한 대역폭과 연결을 줄일 수 AWS 클라우드 있습니다.
클라이언트 장치 인증	클라이언트 장치 인증 구성 요소 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 • IPC (프로세스 간 통신) 작업을 호출하여 클라이언트 장치를 인증하고 권한을 부여하는 사용자 지정 구성 요소에 대한 지원을 추가합니다. 예를 들어 사용자 지정 MQTT 브로커 구성 요소에서 이러한 작업을 사용할 수 있습니다. 자세한 내용은 IPC: 클라이언트 장치 인증 및 권한 부여를 참조하십시오. • 이 구성 요소의 작동 방식을 조정하기 위해 구성할 수 있는 `maxActiveAuthTokens` `cloudQueueSize` , 및 `threadPoolSize` 옵션을 추가합니다.

구성 요소	세부 정보
MQTT 브리지	MQTT 브리지 구성 요소 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 • 로컬 게시/구독을 소스 메시지 브로커로 지정할 때 MQTT 주제 와일드카드 (#및+) 에 대한 지원을 추가합니다. 이 기능을 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소가 필요합니다. • 메시지를 릴레이할 때 대상 주제에 접두사를 추가하도록 MQTT 브리지를 구성하도록 지정할 수 있는 `targetTopicPrefix` 옵션을 추가합니다.
그린그래스 CLI	그린그래스 CLI의 버전 2.6.0을 사용할 수 있습니다. 새로운 기능 • Greengrass CLI에서 사용하는 IPC (프로세스 간 통신) 작업을 호출하는 사용자 지정 구성 요소에 대한 지원을 추가합니다. 이러한 IPC 작업을 사용하여 로컬 배포를 관리하고, 구성 요소 세부 정보를 보고, 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 암호를 생성할 수 있습니다. 자세한 내용은 IPC: 로컬 배포 및 구성 요소 관리를 참조하십시오. 버그 수정 및 개선 • 추가 사소한 수정 및 개선

출시: AWS IoT Greengrass 2022년 5월 31일 코어 v2.5.6 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.5.6과 로그 관리자 구성 요소 버전 2.2.4를 제공합니다.

출시일: 2022년 5월 31일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.6을 사용할 수 있습니다. 새로운 기능 • ECC 키를 사용하는 하드웨어 보안 모듈에 대한 지원을 추가합니다. 하드웨어 보안 모듈 (HSM) 을 사용하여 디바이스의 개인 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요. 버그 수정 및 개선 • 특정 시나리오에서 손상된 설치 스크립트가 포함된 구성 요소를 배포할 때 배포가 완료되지 않는 문제를 수정합니다. • 시작 시 성능을 개선합니다. • 추가 사소한 수정 및 개선
로그 관리자	로그 관리자 구성 요소 버전 2.2.4를 사용할 수 있습니다. 버그 수정 및 개선 • 잘못된 구성을 처리할 때의 안정성을 개선합니다.

구성 요소	세부 정보
	• 추가 사소한 수정 및 개선

출시: AWS IoT Greengrass 2022년 4월 6일 코어 v2.5.5 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소의 버전 2.5.5를 제공합니다.

출시일: 2022년 4월 6일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.5를 사용할 수 있습니다.

구성 요소	세부 정보
	새로운 기능 • 사용자 지정 구성 요소의 루트 CA (인증 기관) 인증서에 액세스할 수 있도록 구성 요소의 `GG_ROOT_CA_PATH` 환경 변수를 추가합니다. 버그 수정 및 개선 • 영어가 아닌 표시 언어를 사용하는 Windows 장치에 대한 지원을 추가합니다. • Greengrass Nucleus가 부울 설치 프로그램 인수를 파싱하는 방식을 업데이트하여 부울 값 없이 부울 인수를 지정하여 값을 지정할 수 있도록 합니다. `true` 예를 들어 이제 자동 리소스 프로비저닝으로 설치하는 `--provision` 대신 `--provision true` 지정하도록 지정할 수 있습니다. • 특정 시나리오에서 코어 디바이스가 프로비저닝한 후 AWS IoT Greengrass 클라우드 서비스에 상태를 보고하지 않던 문제를 수정합니다. • 추가 사소한 수정 및 개선

출시: AWS IoT Greengrass 2022년 3월 23일 코어 v2.5.4 소프트웨어 업데이트

이번 릴리스는 그린그래스 핵 구성 요소 버전 2.5.4와 Lambda 런처 구성 요소 버전 2.0.10을 제공합니다.

출시일: 2022년 3월 23일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.4를 사용할 수 있습니다. 버그 수정 및 개선 • 일반적인 버그 수정 및 개선입니다.
람다 런처	Lambda 런처 구성 요소 버전 2.0.10을 사용할 수 있습니다. 버그 수정 및 개선 • 일반적인 버그 수정 및 개선입니다.

출시: AWS IoT Greengrass 2022년 1월 6일 코어 v2.5.3 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소 버전 2.5.3과 새로운 PKCS #11 공급자 구성 요소를 제공합니다.

출시일: 2022년 1월 6일

출시 하이라이트

- 하드웨어 보안 통합 —이제 하드웨어 보안 모듈 (HSM) 에 안전하게 저장하는 개인 키와 인증서를 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.3을 사용할 수 있습니다. 새로운 기능 • 하드웨어 보안 통합에 대한 지원을 추가합니다. 하드웨어 보안 모듈 (HSM) 을 사용하여 기기의 개인 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요. 버그 수정 및 개선 • Nucleus가 MQTT 연결을 설정하는 동안 발생하는 런타임 예외 문제를 수정합니다. AWS IoT Core
PKCS #11 제공자	PKCS #11 공급자 구성 요소 버전 2.0.0을 사용할 수 있습니다.

구성 요소	세부 정보
	새로운 기능 • 하드웨어 보안 통합에 대한 지원을 추가합니다. 하드웨어 보안 모듈(HSM)을 사용하여 기기의 개인 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 내용은 하드웨어 보안 통합을(를) 참조하세요.

릴리스: AWS IoT Greengrass 2021년 12월 3일 코어 v2.5.2 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소 버전 2.5.2를 제공합니다.

출시일: 2021년 12월 3일

출시 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.2를 사용할 수 있습니다. 버그 수정 및 개선 • Greengrass nucleus 업데이트 후 Windows 서비스를 중지하거나 디바이스를 재부팅한 후 Windows 서비스가 다시 시작되지 않는 문제를 수정합니다.
AWS IoT Device Defender	AWS IoT Device Defender구성 요소 버전 3.0.1을 사용할 수 있습니다. 이 버전의 AWS IoT Device Defender 구성 요소에는 버전 2.x와 다른 구성 매개 변수가 필요합니다. 버전 2.x에 대해 기본이 아닌 구성을 사용하고 v2.x에서 v3.x로 업그레이드하려면 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 구성 요소 구성을 참조하십시오. AWS IoT Device Defender 새로운 기능 • Windows를 실행하는 핵심 장치에 대한 지원을 추가합니다. • 구성 요소 유형을 Lambda 구성 요소에서 일반 구성 요소로 변경합니다. 이제 이 구성 요소는 구독을 생성할 때 더 이상 기존 구독 라우터 구성 요소에 의존하지 않습니다. • `UseInstaller` 구성 요소 종속성을 설치하는 설치 스크립트를 선택적으로 비활성화할 수 있는 새 구성 매개 변수를 추가합니다.

릴리스: AWS IoT Greengrass 2021년 11월 23일 코어 v2.5.1 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소의 버전 2.5.1을 제공합니다.

릴리스 날짜: 2021년 11월 23일

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.1을 사용할 수 있습니다. 버그 수정 및 개선 • Windows용 자바 런타임 환경 (JRE) 의 32비트 버전에 대한 지원을 추가합니다. • AWS IoT정책이 권한을 부여하지 않는 핵심 장치에 대한 사물 그룹 제거 동작을 변경합니다. `greengrass:ListThingGroupsForCoreDevice` 이 버전에서는 배포가 계속되고 경고가 기록되며 사물 그룹에서 코어 장치를 제거해도 구성 요소가 제거되지 않습니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요. • Greengrass 핵이 Greengrass 구성 요소 프로세스에 사용할 수 있도록 하는 시스템 환경 변수 관련 문제를 수정합니다. 이제 최신 시스템 환경 변수를 사용하도록 구성 요소를 다시 시작할 수 있습니다.

릴리스: AWS IoT Greengrass 2021년 11월 12일에 코어 v2.5.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.5.0, 새로 제공된 구성 요소 및 AWS 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2021년 11월 12일

출시 하이라이트

- Windows 장치 지원 —이제 Windows 운영 체제를 실행하는 장치에서 AWS IoT Greengrass Core 소프트웨어를 실행할 수 있습니다. 자세한 내용은 지원되는 플랫폼 및 요구 사항 및 운영 체제별 Greengrass 기능 호환성 섹션을 참조하세요.
- 새 사물 그룹 제거 동작 - 이제 사물 그룹에서 핵심 장치를 제거하여 다음에 해당 장치에 배포할 때 해당 사물 그룹의 구성 요소를 제거할 수 있습니다.

> **Important**
>
> 이 변경으로 인해 핵심 장치의 AWS IoT 정책에는 `greengrass:ListThingGroupsForCoreDevice` 권한이 있어야 합니다. AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 기본 AWS IoT 정책이 `greengrass:*` 허용되며 여기에는 이 권한이 포함됩니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.

- 하드웨어 보안 지원 - 이제 하드웨어 보안 모듈 (HSM) 을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성하여 장치의 개인 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요.
- HTTPS 프록시 지원 - 이제 HTTPS 프록시를 통해 연결하도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

릴리스 세부 정보

- 플랫폼 지원 업데이트
- 공개 구성 요소 업데이트

플랫폼 지원 업데이트

플랫폼	세부 정보
Windows	AWS IoT Greengrass이제 다음 버전의 Windows에서 AWS IoT Greengrass Core 소프트웨어를 실행할 수 있습니다. • Windows 10 • Windows Server 2019 자세한 내용은 지원되는 플랫폼 및 요구 사항 및 운영 체제별 Greengrass 기능 호환성 섹션을 참조하세요.

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.5.0을 사용할 수 있습니다. 새로운 기능 • Windows를 실행하는 코어 디바이스에 대한 지원을 추가합니다.

구성 요소	세부 정보
	• 사물 그룹 제거 동작을 변경합니다. 이 버전에서는 사물 그룹에서 코어 장치를 제거하여 다음 배포 시 해당 사물 그룹의 구성 요소를 제거할 수 있습니다. 이 변경으로 인해 코어 디바이스의 AWS IoT 정책에 `greengrass:ListThingGroupsForCoreDevice` 권한이 있어야 합니다. AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 기본 AWS IoT 정책이 `greengrass:*` 허용되며 여기에는 이 권한이 포함됩니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요. • HTTPS 프록시 구성에 대한 지원을 추가합니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요. • 새 `windowsUser` 구성 매개변수를 추가합니다. 이 매개 변수를 사용하여 Windows 코어 장치에서 구성 요소를 실행하는 데 사용할 기본 사용자를 지정할 수 있습니다. 자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요. • HTTP 요청 제한 시간을 사용자 지정하여 속도가 느린 네트워크에서 성능을 향상시키는 데 사용할 수 있는 새 `httpClient` 구성 옵션을 추가합니다. 자세한 내용은 HttpClient 구성 매개 변수를 참조하십시오. 버그 수정 및 개선 • 구성 요소에서 코어 장치를 다시 시작하는 부트스트랩 수명 주기 옵션을 수정합니다. • 레시피 변수에 하이픈에 대한 지원을 추가합니다. • 온디맨드 Lambda 함수 구성 요소에 대한 IPC 인증을 수정합니다. • 로그 메시지를 개선하고 중요하지 않은 로그를 레벨에서 `INFO` `DEBUG` 레벨로 변경하여 로그를 더 유용하게 사용할 수 있도록 합니다. • 자동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치할 때 Greengrass nucleus가 생성하는 기본 토큰 교환 역할에서 `iot:DescribeCertificate` 권한을 제거합니다. 이 권한은 Greengrass 핵에서 사용되지 않습니다. • 동일한 정책에 사용할 수 있는 경우 `iam:CreatePolicy` 자동 프로비저닝 스크립트에 `iam:GetPolicy` 권한이 필요하지 않도록 문제를 수정합니다.

구성 요소	세부 정보
	• 추가 사소한 수정 및 개선
그린그래스 CLI	그린그래스 CLI의 버전 2.5.0을 사용할 수 있습니다. 새로운 기능 • Windows를 실행하는 코어 디바이스에 대한 지원을 추가합니다. • Windows 디바이스에서 Greengrass CLI를 사용할 수 있도록 시스템 그룹을 승인하도록 지정할 수 있는 새 `AuthorizedWindowsGroups` 구성 매개변수를 추가합니다. • 로컬 배포를 위한 `windowsUser` 파라미터를 추가합니다. 이 매개 변수를 사용하여 Windows 코어 장치에서 구성 요소를 실행하는 데 사용할 사용자를 지정할 수 있습니다.

구성 요소	세부 정보
CloudWatch 메트릭	CloudWatch메트릭 구성 요소 버전 3.0.0을 사용할 수 있습니다. 이 버전의 CloudWatch 메트릭 구성 요소에는 버전 2.x와 다른 구성 매개 변수가 필요합니다. 버전 2.x에 대해 기본이 아닌 구성을 사용하고 v2.x에서 v3.x로 업그레이드하려면 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 메트릭 구성 요소 구성을 참조하십시오. CloudWatch 새로운 기능 • Windows를 실행하는 핵심 장치에 대한 지원을 추가합니다. • 구성 요소 유형을 Lambda 구성 요소에서 일반 구성 요소로 변경합니다. 이제 이 구성 요소는 구독을 생성할 때 더 이상 기존 구독 라우터 구성 요소에 의존하지 않습니다. • `InputTopic` 구성 요소가 메시지를 수신하기 위해 구독하는 주제를 지정하는 새 구성 매개 변수를 추가합니다. • `OutputTopic` 구성 요소가 상태 응답을 게시하는 주제를 지정하는 새 구성 매개 변수를 추가합니다. • AWS IoT CoreMQTT 주제를 게시하고 구독할지 여부를 지정하는 새 `PubSubToIoTCore` 구성 매개 변수를 추가합니다. • `UseInstaller` 구성 요소 종속성을 설치하는 설치 스크립트를 선택적으로 비활성화할 수 있는 새 구성 매개 변수를 추가합니다. 버그 수정 및 개선 입력 데이터의 중복 타임스탬프에 대한 지원을 추가합니다.
Lambda 관리자	Lambda 관리자 구성 요소 버전 2.2.0을 사용할 수 있습니다. 버그 수정 및 개선 • Lambda 함수가 재시작 후 로그를 기록하지 못하던 문제를 수정합니다. • 주제에 와일드카드가 있는 경우 레거시 구독 라우터가 중복 메시지를 보내는 문제를 수정합니다. • 고정되지 않은 Lambda 함수가 에서 Greengrass IPC (프로세스 간 통신) 라이브러리를 사용할 수 없었던 문제를 수정합니다. AWS IoT Device SDK

릴리스: AWS IoT Greengrass 2021년 8월 3일 코어 v2.4.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.4.0, AWS 새로 제공된 구성 요소 및 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2021년 8월 3일

출시 하이라이트

- 시스템 리소스 제한 —Greengrass nucleus 구성 요소가 이제 시스템 리소스 제한을 지원합니다. 각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 설명은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요.
- 구성 요소 일시 중지/재개 —Greengrass 핵은 이제 구성 요소 일시 중지 및 재개를 지원합니다. IPC (프로세스 간 통신) 라이브러리를 사용하여 다른 구성 요소의 프로세스를 일시 중지하고 재개하는 사용자 지정 구성 요소를 개발할 수 있습니다. 자세한 내용은 PauseComponent 및 ResumeComponent 섹션을 참조하세요.
- AWS IoT플릿 프로비저닝으로 설치 - 새로운 AWS IoT 플릿 프로비저닝 플러그인을 사용하여 필요한 리소스를 프로비저닝하기 위해 연결하는 기기에 AWS IoT Greengrass Core 소프트웨어를 설치합니다. AWS IoT AWS 디바이스는 클레임 인증서를 사용하여 프로비저닝합니다. 제조 과정에서 클레임 인증서를 디바이스에 내장할 수 있으므로 온라인 상태가 되는 즉시 각 디바이스를 프로비전할 수 있습니다. 자세한 설명은 AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.
- 사용자 지정 프로비저닝으로 설치 AWS IoT Greengrass —Core 소프트웨어를 디바이스에 설치할 때 필요한 AWS 리소스를 프로비저닝할 수 있는 사용자 지정 프로비저닝 플러그인을 개발하세요. 설치 중에 실행되는 Java 애플리케이션을 생성하여 사용자 지정 사용 사례에 맞게 Greengrass 코어 디바이스를 설정할 수 있습니다. 자세한 설명은 사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.4.0을 사용할 수 있습니다. 새로운 기능 • 시스템 리소스 제한에 대한 지원을 추가합니다. 각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 설명은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요. • 구성 요소를 일시 중지 및 재개하기 위한 IPC 작업을 추가합니다. 자세한 내용은 PauseComponent 및 ResumeComponent 섹션을 참조하세요. • 프로비저닝 플러그인에 대한 지원을 추가합니다. 설치 중에 실행할 JAR 파일을 지정하여 Greengrass 코어 장치에 필요한 AWS 리소스를 프로비저닝할 수 있습니다. Greengrass Nucleus에는 사용자 지정 프로비저닝 플러그인을 개발하기 위해 구현할 수 있는 인터페이스가 포함되어 있습니다. 자세한 설명은 사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요. • AWS IoT GreengrassCore 소프트웨어 설치 `thing-name-policy` 프로그램에 선택적 인수를 추가합니다. 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치할 때 이 옵션을 사용하여 기존 또는 사용자 지정 AWS IoT 정책을 지정할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 • 시작 시 로깅 구성을 업데이트합니다. 이렇게 하면 시작 시 로깅 구성이 적용되지 않던 문제가 해결됩니다. • 설치 중에 Greengrass 루트 폴더의 구성 요소 저장소를 가리키도록 Nucleus 로더 심볼릭 링크를 업데이트합니다. 이 업데이트를 사용하면 Core 소프트웨어를 설치할 때 다운로드한 JAR 파일 및 기타 Nucleus 아티팩트를 삭제할 수 있습니다. AWS IoT Greengrass • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
그린그래스 CLI	그린그래스 CLI의 버전 2.4.0을 사용할 수 있습니다. 새로운 기능 • 시스템 리소스 제한에 대한 지원을 추가합니다. 로컬 배포를 생성할 때 각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 및 디플로이먼트 create 명령을 참조하십시오.
AWS IoT클레임별 플릿 프로비저닝	이제 클레임 플러그인을 통한 AWS IoT 플릿 프로비저닝을 사용할 수 있습니다. 자세한 설명은 AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요. 새로운 기능 • AWS IoT플릿 프로비저닝과 함께 AWS IoT Greengrass Core 소프트웨어를 설치하기 위한 지원을 추가합니다. 설치 중에 디바이스를 AWS IoT 연결하여 필요한 AWS 리소스를 프로비저닝하고 정기적인 작업에 사용할 디바이스 인증서를 다운로드합니다.

릴리스: AWS IoT Greengrass 2021년 6월 29일 코어 v2.3.0 소프트웨어 업데이트

이 릴리스는 Greengrass 핵 구성 요소의 버전 2.3.0을 제공합니다.

출시일: 2021년 6월 29일

출시 하이라이트

- 대규모 구성 지원 —Greengrass nucleus 구성 요소는 이제 최대 10MB의 배포 문서를 지원합니다. 이제 Greengrass 구성 요소에 대규모 구성 업데이트를 배포할 수 있습니다.

> **Note**
>
> 이 기능을 사용하려면 코어 디바이스의 AWS IoT 정책에서 `greengrass:GetDeploymentConfiguration` 권한을 허용해야 합니다. AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 코어 장치 AWS IoT 정책에서 `greengrass:*` 허용하며 여기에는 이 권한이 포함됩니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 • 7KB (사물을 대상으로 하는 배포의 경우) 또는 31KB (사물 그룹을 대상으로 하는 배포의 경우) 에서 최대 10MB의 배포 구성 문서에 대한 지원을 추가합니다. 이 기능을 사용하려면 코어 디바이스의 AWS IoT 정책에서 권한을 허용해야 합니다. `greengrass:GetDeploymentConfiguration` AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 코어 장치 AWS IoT 정책에서 `greengrass:*` 허용하며 여기에는 이 권한이 포함됩니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요. • `iot:thingName` 레시피 변수를 추가합니다. 이 레시피 변수를 사용하여 레시피에 있는 핵심 장치 사물의 이름을 가져올 수 있습니다. AWS IoT 자세한 설명은 레시피 변수 섹션을 참조하세요. 버그 수정 및 개선 • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.

릴리스: AWS IoT Greengrass 2021년 6월 18일 코어 v2.2.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.2.0, AWS 새로 제공된 구성 요소 및 제공된 구성 요소에 대한 업데이트를 제공합니다. AWS

출시일: 2021년 6월 18일

출시 하이라이트

- 클라이언트 장치 지원 AWS —새로 제공되는 클라이언트 장치 구성 요소를 사용하면 클라우드 검색을 사용하여 클라이언트 장치를 코어 장치에 연결할 수 있습니다. Greengrass 구성 요소에서 클라이언트 장치를 클라이언트 장치와 AWS IoT Core 동기화하고 클라이언트 장치와 상호 작용할 수 있습니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

- 로컬 섀도우 서비스 - 새 섀도우 관리자 구성 요소를 사용하면 코어 디바이스에서 로컬 섀도우 서비스를 사용할 수 있습니다. 이 섀도우 서비스를 사용하면 의 Greengrass IPC (프로세스 간 통신) 라이브러리를 사용하여 오프라인 상태에서 로컬 섀도우와 상호 작용할 수 있습니다. AWS IoT Device SDK 섀도우 관리자 구성 요소를 사용하여 로컬 섀도우 상태를 동기화할 수도 있습니다. AWS IoT Core 자세한 설명은 디바이스 섀도우와 상호작용 섹션을 참조하세요.

릴리스 세부 정보

- 공개 구성 요소 업데이트

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 • 로컬 섀도우 관리를 위한 IPC 작업을 추가합니다. 버그 수정 및 개선 • JAR 파일의 크기를 줄입니다. • 메모리 사용량을 줄입니다.

구성 요소	세부 정보
	• 특정 경우에 로그 구성이 업데이트되지 않던 문제를 수정합니다. • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
섀도우 매니저	새 섀도우 관리자 구성 요소 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 클래식 섀도우 및 네임드 섀도우에 대한 지원을 추가합니다. • IPC를 사용한 로컬 섀도우 관리에 대한 지원을 추가합니다. • 섀도우 AWS IoT Core 동기화에 대한 지원을 추가합니다.
클라이언트 장치 인증	새 클라이언트 장치 인증 구성 요소 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • MQTT를 통해 코어 디바이스에 연결되는 로컬 IoT 디바이스인 Greengrass 클라이언트 디바이스에 대한 지원을 추가합니다. • 클라이언트 디바이스의 인증 및 권한 부여와 MQTT 작업에 대한 지원을 추가합니다.
모켓 MQTT 브로커	새로운 모켓 MQTT 브로커 컴포넌트 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 클라이언트 장치와의 통신을 처리하는 로컬 Moquette MQTT 브로커에 대한 지원을 추가합니다.
MQTT 브리지	새 MQTT 브리지 구성 요소의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 로컬 MQTT 브로커, 로컬 Greengrass 게시/구독 브로커 및 MQTT 브로커 간의 릴레이 메시지 지원을 추가합니다. AWS IoT Core

구성 요소	세부 정보
IP 감지기	새 IP 탐지기 구성 요소 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 클라이언트 장치가 연결할 수 있도록 코어 장치의 로컬 MQTT 브로커 엔드포인트를 AWS IoT Greengrass 클라우드 서비스에 보고하는 지원을 추가합니다.
로그 관리자	로그 관리자 구성 요소 버전 2.1.1을 사용할 수 있습니다. 버그 수정 및 개선 • 특정 경우에 시스템 로그 구성이 업데이트되지 않던 문제를 수정합니다.
DLR 개체 감지	DLR 객체 감지 버전 2.1.2를 사용할 수 있습니다. 버그 수정 및 개선 • 샘플 DLR 개체 감지 추론 결과의 경계 상자가 부정확했던 이미지 스케일링 문제를 수정합니다.
TensorFlow 라이트 객체 감지	TensorFlow Lite 객체 감지 버전 2.1.1을 사용할 수 있습니다. 버그 수정 및 개선 • 샘플 TensorFlow Lite 객체 감지 추론 결과의 경계 상자가 부정확하게 표시되는 이미지 스케일링 문제를 수정합니다.

릴리스: AWS IoT Greengrass 2021년 4월 26일 코어 v2.1.0 소프트웨어 업데이트

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.1.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS

출시일: 2021년 4월 26일

출시 하이라이트

- Docker Hub와 Amazon Elastic Container 레지스트리 (Amazon ECR) 통합 —새로운 Docker 애플리케이션 관리자 구성 요소를 사용하면 Amazon ECR에서 퍼블릭 또는 프라이빗 이미지를 다운로드할

수 있습니다. 또한 이 구성 요소를 사용하여 Docker Hub 및 에서 공개 이미지를 다운로드할 수 있습니다. AWS Marketplace 자세한 설명은 도커 컨테이너 실행 섹션을 참조하세요.

- AWS IoT Greengrass코어 소프트웨어용 Dockerfile 및 Docker 이미지 - Greengrass Docker 이미지를 사용하여 Amazon Linux 2를 기본 운영 체제로 사용하는 Docker 컨테이너에서 AWS IoT Greengrass 실행할 수 있습니다. AWS IoT GreengrassDockerfile을 사용하여 자신만의 그린그래스 이미지를 만들 수도 있습니다. 자세한 설명은 Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어 실행 섹션을 참조하세요.
- 추가 기계 학습 프레임워크 및 플랫폼 지원 TensorFlow —Lite 2.5.0 및 DLR 1.6.0을 사용하여 사전 학습된 모델을 사용하여 샘플 이미지 분류 및 객체 감지를 수행하는 샘플 기계 학습 추론 구성 요소를 배포할 수 있습니다. 또한 이번 릴리스는 Armv8 (AArch64) 기기에 대한 샘플 머신 러닝 지원을 확장합니다. 자세한 설명은 기계 학습 추론 수행 섹션을 참조하세요.

릴리스 세부 정보

- 플랫폼 지원 업데이트
- 퍼블릭 구성 요소 업데이트

플랫폼 지원 업데이트

플랫폼	세부 정보
Docker	에 대한 AWS IoT Greengrass Dockerfile 및 Docker 이미지를 이제 사용할 수 있습니다. Dockerfile AWS IoT GreengrassAmazon Linux 2 (x86_64) 기본 이미지에 AWS IoT Greengrass 코어 소프트웨어 및 종속 항목이 설치된 컨테이너 이미지를 빌드하기 위한 Dockerfile을 제공합니다. 다른 플랫폼 아키텍처에서 실행되도록 Dockerfile의 기본 이미지를 수정할 수 있습니다. AWS IoT Greengrass 도커 이미지 AWS IoT GreengrassAmazon Linux 2 (x86_64) 기본 이미지에 AWS IoT Greengrass 코어 소프트웨어 및 종속 항목이 설치되어 있는 사전 빌드된 Docker 이미지를 제공합니다.

플랫폼	세부 정보
	자세한 설명은 Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어 실행 섹션을 참조하세요.

퍼블릭 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보

그린그래스의 핵	그린그래스 핵 버전 2.1.0을 사용할 수 있습니다.
	
	새로운 기능
	• Amazon ECR의 프라이빗 리포지토리에서 Docker 이미지를 다운로드할 수 있습니다.
	• 코어 디바이스의 MQTT 구성을 사용자 지정하기 위해 다음 파라미터를 추가합니다.
	  • `maxInFlightPublishes` — 동시에 전송할 수 있는 승인되지 않은 MQTT QoS 1 메시지의 최대 수입니다.
	  • `maxPublishRetry` — 게시하지 못한 메시지를 재시도할 수 있는 최대 횟수입니다.

구성 요소	세부 정보
	• `fleetstatusservice` 구성 매개 변수를 추가하여 코어 장치가 장치 상태를 에 게시하는 간격을 구성합니다. AWS 클라우드 • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub. 버그 수정 및 개선 • Nucleus가 다시 시작될 때 섀도우 디플로이먼트가 중복되던 문제를 수정합니다. • 서비스 로드 예외가 발생했을 때 Nucleus가 충돌하던 문제를 수정합니다. • 순환 종속성을 포함하는 배포가 실패하도록 구성 요소 종속성 해결을 개선합니다. • 플러그인 구성 요소가 이전에 코어 장치에서 제거된 경우 플러그인 구성 요소가 재배포되지 않던 문제를 수정합니다. • `HOME`환경 변수가 Lambda 구성 요소 또는 루트로 실행되는 구성 요소의 */greengrass/v2* `/work` 디렉토리로 설정되는 문제를 수정했습니다. 이제 구성 요소를 실행하는 사용자의 홈 디렉터리에 `HOME` 변수가 올바르게 설정되었습니다. • 기타 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
Docker 애플리케이션 관리자	새 Docker 애플리케이션 관리자 구성 요소 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • Amazon ECR의 프라이빗 리포지토리에서 이미지를 다운로드하기 위한 자격 증명을 관리합니다. • Amazon ECR, 도커 허브 등에서 공개 이미지를 다운로드합니다. AWS Marketplace
람다 런처	Lambda 런처 구성 요소 버전 2.0.4를 사용할 수 있습니다. 버그 수정 및 개선 • 구성 요소가 Lambda 함수 `AddGroupOwner` 컨테이너로 올바르게 전달되지 않는 문제를 수정합니다.

구성 요소	세부 정보
레거시 구독 라우터	레거시 구독 라우터 구성 요소 버전 2.1.0을 사용할 수 있습니다. 버그 수정 및 개선 • `source`및 `target` 의 ARN 대신 구성 요소 이름을 지정하는 서포트를 추가합니다. 구독의 구성 요소 이름을 지정하면 Lambda 함수 버전이 변경될 때마다 구독을 재구성할 필요가 없습니다.
로컬 디버그 콘솔	로컬 디버그 콘솔 구성 요소 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • HTTPS를 사용하여 로컬 디버그 콘솔에 대한 연결을 보호합니다. HTTPS는 기본적으로 활성화되어 있습니다. 버그 수정 및 개선 • 구성 편집기에서 플래시바 메시지를 무시할 수 있습니다.
로그 관리자	로그 관리자 구성 요소 버전 2.1.0을 사용할 수 있습니다. 버그 수정 및 개선 • 표준 출력 (stdout) `logFileDirectoryPath` 및 `logFileRegex` 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에 대한 기본값과 작동하는 기본값을 사용하십시오. • 로그를 Logs에 업로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. CloudWatch • 로그 스트림 이름의 콜론 문자 (`:`) 를 올바르게 처리하십시오. CloudWatch 로그 로그 스트림 이름은 콜론을 지원하지 않습니다. • 로그 스트림에서 사물 그룹 이름을 제거하여 로그 스트림 이름을 단순화합니다. • 정상 동작 중에 인쇄되는 오류 로그 메시지를 제거합니다.

구성 요소	세부 정보
DLR 이미지 분류	DLR 이미지 분류 구성 요소 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • 딥러닝 런타임 v1.6.0을 사용하십시오. • Armv8 (AArch64) 플랫폼에서 샘플 이미지 분류에 대한 지원을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다. • 샘플 추론을 위한 카메라 통합을 활성화하십시오. 새 `UseCamera` 구성 매개변수를 사용하여 샘플 추론 코드를 활성화하여 Greengrass 코어 장치의 카메라에 액세스하고 캡처한 이미지에서 로컬로 추론을 실행할 수 있습니다. • 추론 결과를 게시하기 위한 지원을 에 추가합니다. AWS 클라우드 새 `PublishResultsOnTopic` 구성 매개 변수를 사용하여 결과를 게시하려는 주제를 지정합니다. • 추론을 수행하려는 이미지의 사용자 지정 디렉터리를 지정할 수 있는 새 `ImageDirectory` 구성 매개 변수를 추가합니다. 버그 수정 및 개선 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과를 기록합니다. • AWS IoT GreengrassCore 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 기록합니다. • AWS IoT Device SDK를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용할 수 있습니다.

구성 요소	세부 정보
DLR 객체 감지	DLR 개체 감지 구성 요소 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • 딥러닝 런타임 v1.6.0을 사용하십시오. • Armv8 (AArch64) 플랫폼에서 샘플 객체 감지에 대한 지원을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다. • 샘플 추론을 위한 카메라 통합을 활성화하십시오. 새 `UseCamera` 구성 매개변수를 사용하여 샘플 추론 코드를 활성화하여 Greengrass 코어 장치의 카메라에 액세스하고 캡처한 이미지에서 로컬로 추론을 실행할 수 있습니다. • 추론 결과를 게시하기 위한 지원을 에 추가합니다. AWS 클라우드 새 `PublishResultsOnTopic` 구성 매개 변수를 사용하여 결과를 게시하려는 주제를 지정합니다. • 추론을 수행하려는 이미지의 사용자 지정 디렉터리를 지정할 수 있는 새 `ImageDirectory` 구성 매개 변수를 추가합니다. 버그 수정 및 개선 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과를 기록합니다. • AWS IoT GreengrassCore 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 기록합니다. • AWS IoT Device SDK를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용할 수 있습니다.
DLR 이미지 분류 모델 스토어	DLR 이미지 분류 모델 스토어 구성 요소 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • Armv8 (AArch64) 플랫폼을 위한 샘플 ResNet -50 이미지 분류 모델을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다.

구성 요소	세부 정보
DLR 객체 감지 모델 스토어	DLR 개체 감지 모델 저장소 구성 요소 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • Armv8 (AArch64) 플랫폼용 샘플 YoloV3 개체 감지 모델을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다.
DLR 인스톨러	DLR 구성 요소 버전 1.6.1을 사용할 수 있습니다. 새로운 기능 • 딥러닝 런타임 v1.6.0 및 해당 종속 항목을 설치합니다. • Armv8 (AArch64) 플랫폼에 DLR을 설치하기 위한 지원을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다. 버그 수정 및 개선 • 가상 환경에 를 AWS IoT Device SDK 설치하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다. • 추가 사소한 버그 수정 및 개선
TensorFlow 라이트 이미지 분류	새로운 TensorFlow Lite 이미지 분류 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • Lite를 사용한 TensorFlow 샘플 이미지 분류 추론에 대한 지원을 추가합니다.
TensorFlow 라이트 객체 감지	새로운 TensorFlow Lite 객체 감지 구성 요소 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • Lite를 사용한 TensorFlow 샘플 객체 감지 추론에 대한 지원을 추가합니다.

구성 요소	세부 정보
TensorFlow Lite 이미지 분류 모델 스토어	새로운 TensorFlow Lite 이미지 분류 모델 스토어 구성 요소 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • Lite를 사용한 샘플 이미지 분류 추론을 위해 사전 학습된 MobileNet v1 양자화 모델을 제공하십시오. TensorFlow
TensorFlow Lite 객체 감지 모델 스토어	새로운 TensorFlow Lite 객체 감지 모델 스토어 컴포넌트 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • Lite를 사용한 샘플 객체 감지 추론을 위해 COCO 데이터세트에서 학습된 사전 학습된 싱글 샷 감지 (SSD) MobileNet 모델을 제공합니다. TensorFlow
TensorFlow Lite	새 TensorFlow Lite 구성 요소 버전 2.5.0을 사용할 수 있습니다. 새로운 기능 • Armv7, Armv8 (AArch64) 및 x86_64 플랫폼의 가상 환경에 TensorFlow 라이트 v1.6.0 및 해당 종속 프로그램을 설치합니다.

릴리스: AWS IoT Greengrass 2021년 3월 9일 코어 v2.0.5 소프트웨어 업데이트

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.0.5를 제공하고 제공된 구성 요소를 업데이트합니다. AWS 네트워크 프록시 지원 문제 및 AWS 중국 지역의 Greengrass 데이터 플레인 엔드포인트 관련 문제를 수정합니다.

출시일: 2021년 3월 9일

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.0.5를 사용할 수 있습니다. 버그 수정 및 개선 • AWS제공된 구성 요소를 다운로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. • AWS중국 지역에서 올바른 Greengrass 데이터 플레인 엔드포인트를 사용하십시오.

릴리스: AWS IoT Greengrass 2021년 2월 4일 코어 v2.0.4 소프트웨어 업데이트

이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.0.4를 제공합니다. 여기에는 포트 443을 통한 HTTPS 통신을 구성하고 버그를 수정하기 위한 새 `greengrassDataPlanePort` 매개변수가 포함되어 있습니다. 이제 최소 IAM 정책에 따라 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 실행할 `sts:GetCallerIdentity` 때 `iam:GetPolicy` 및 가 필요합니다. `--provision true`

출시일: 2021년 2월 4일

공개 구성 요소 업데이트

다음 표에는 새 기능 및 업데이트된 기능을 포함하는 AWS 제공된 구성 요소가 나열되어 있습니다.

> **Important**
>
> 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

구성 요소	세부 정보
그린그래스 핵	그린그래스 핵 버전 2.0.4를 사용할 수 있습니다. 새로운 기능 • 포트 443을 통한 HTTPS 트래픽을 활성화합니다. nucleus 구성 요소 버전 2.0.4의 새 `greengrassDataPlanePort` 구성 매개변수를 사용하여 기본 포트 8443이 아닌 포트 443을 통해 이동하도록 HTTPS 통신을 구성할 수 있습니다. 자세한 설명은 포트 443을 통해 HTTPS를 구성합니다. 섹션을 참조하세요. • 작업 경로 레시피 변수를 추가합니다. 이 레시피 변수를 사용하여 구성 요소의 작업 폴더 경로를 가져올 수 있습니다. 이 경로를 사용하여 구성 요소와 해당 종속성 간에 파일을 공유할 수 있습니다. 자세한 내용은 작업 경로 레시피 변수를 참조하십시오. 버그 수정 및 개선 • 역할 정책이 이미 존재하는 경우 토큰 교환 AWS Identity and Access Management (IAM) 역할 정책이 생성되지 않도록 합니다. 이 변경으로 인해 이제 설치 프로그램을 실행할 `sts:GetCallerIdentity` 때 `iam:GetPolicy` 및 가 필요합니다. `--provision true` 자세한 설명은 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책 섹션을 참조하세요. • 아직 성공적으로 등록되지 않은 배포의 취소를 올바르게 처리합니다.

구성 요소	세부 정보
	• 배포를 롤백할 때 최신 타임스탬프가 있는 이전 항목을 제거하도록 구성을 업데이트합니다. • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.

AWS IoT Greengrass버전 1에서 마이그레이션

AWS IoT Greengrass Version 2AWS IoT GreengrassCore 소프트웨어, API 및 콘솔의 메이저 버전 릴리스입니다. AWS IoT Greengrass V2모듈식 애플리케이션AWS IoT Greengrass V1, 대규모 장치로의 배포, 추가 플랫폼 지원 등 몇 가지 개선 사항을 소개합니다.

> **Note**
>
> 2023년 6월 30일 이후에는 더 AWS IoT Greengrass Version 1 이상 기능 업데이트, 개선 사항, 버그 수정 또는 보안 패치를 받을 수 없습니다. AWS IoT Greengrass V1 관리형 정책에 대한 자세한 정보는 섹션을 참조하세요. 를 사용하는 AWS IoT Greengrass V1 경우 로 마이그레이션하는 것이 좋습니다. AWS IoT Greengrass V2

에서 AWS IoT Greengrass V1 마이그레이션하려면 이 가이드의 지침을 따르십시오AWS IoT Greengrass V2.

V1 애플리케이션을 V2에서 실행할 수 있나요?

대부분의 V1 애플리케이션은 애플리케이션 코드를 변경할 필요 없이 V2 코어 디바이스에서 실행할 수 있습니다. V1 애플리케이션이 다음 기능을 사용하는 경우 V2에서 해당 애플리케이션을 실행할 수 없습니다.

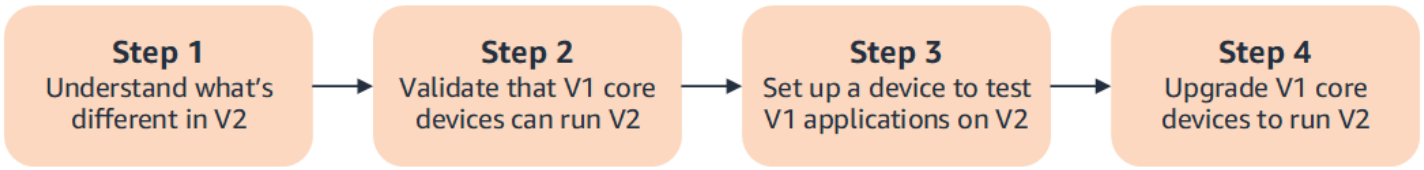
- C 및 C++ Lambda 함수 런타임

V1 애플리케이션이 다음 기능 중 하나를 사용하는 경우 AWS IoT Device SDK V2를 사용하여 애플리케이션을 실행하도록 애플리케이션 코드를 수정해야 합니다. AWS IoT Greengrass V2

- 로컬 섀도우 서비스와 상호 작용하십시오.
- 로컬 연결 장치 (Greengrass 장치) 에 메시지 게시

마이그레이션 개요

상위 수준에서 다음 절차를 사용하여 코어 디바이스를 에서 AWS IoT Greengrass V1 로 업그레이드할 수 있습니다. AWS IoT Greengrass V2 따라야 하는 정확한 절차는 사용자 환경의 특정 요구 사항에 따라 달라집니다.



1. V1과 V2의 차이점을 이해하십시오.

   AWS IoT Greengrass V2디바이스 플릿 및 배포 가능한 소프트웨어에 대한 새로운 기본 개념을 소개하고 V2는 V1의 여러 개념을 단순화합니다.

   AWS IoT Greengrass V2클라우드 서비스 및 AWS IoT Greengrass Core 소프트웨어 v2.x는 클라우드 서비스 및 Core 소프트웨어 v1.x와 역호환되지 않습니다. AWS IoT Greengrass V1 AWS IoT Greengrass 따라서 AWS IoT Greengrass V1 over-the-air (OTA) 업데이트는 코어 디바이스를 V1에서 V2로 업그레이드할 수 없습니다.

2. V1 코어 기기가 V2를 실행할 수 있는지 확인

   V1 코어 디바이스가 Core 소프트웨어 v2.x 및 기능을 AWS IoT Greengrass 실행할 수 있는지 확인합니다. AWS IoT Greengrass V2 AWS IoT Greengrass V2장치 요구 사항이 다릅니다. AWS IoT Greengrass V1

3. 새 기기를 설정하여 V2에서 V1 애플리케이션을 테스트하세요.

   프로덕션 환경에서 디바이스에 대한 위험을 최소화하려면 V2에서 V1 애플리케이션을 테스트할 새 디바이스를 만드십시오. AWS IoT GreengrassCore 소프트웨어 v2.x를 설치한 후에는 애플리케이션을 마이그레이션하고 테스트하기 위한 AWS IoT Greengrass V2 구성 요소를 만들고 배포할 수 있습니다. AWS IoT Greengrass V1

4. V1 코어 디바이스를 업그레이드하여 V2를 실행합니다.

   기존 V1 코어 디바이스를 업그레이드하여 Core 소프트웨어 v2.x AWS IoT Greengrass 및 구성 요소를 실행합니다. AWS IoT Greengrass V2 디바이스 플릿을 V1에서 V2로 마이그레이션하려면 플릿의 각 디바이스에 대해 이 단계를 반복합니다.

AWS IoT Greengrass V1 과 사이의 차이점 AWS IoT Greengrass V2

AWS IoT Greengrass V2 장치, 플릿 및 배포 가능한 소프트웨어에 대한 새로운 기본 개념을 소개합니다. 이 섹션에서는 V2와 다른 V1 개념에 대해 설명합니다.

그린그래스 개념 및 용어

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
애플리케이션 코드	에서 AWS IoT Greengrass V1 Lambda 함수는 코어 디바이스에서 실행되는 소프트웨어를 정의합니다. 각 Greengrass 그룹에서 함수가 사용하는 구독과 로컬 리소스를 정의합니다. Core 소프트웨어가 컨테이너화된 Lambda 런타임 환경에서 실행하는 Lambda 함수의 경우 메모리 제한과 같은 컨테이너 파라미터를 정의합니다. AWS IoT Greengrass	에서 AWS IoT Greengrass V2 구성 요소는 코어 디바이스에서 실행되는 소프트웨어 모듈입니다. • 각 구성 요소에는 구성 요소 수명 주기의 각 단계에서 실행할 구성 요소의 메타데이터, 매개 변수, 종속성 및 스크립트를 정의하는 레시피가 있습니다. • 레시피는 또한 스크립트, 컴파일된 코드, 정적 리소스와 같은 바이너리 파일인 구성 요소의 아티팩트를 정의합니다. • 구성 요소를 코어 기기에 배포하면 코어 기기는 구성 요소 레시피와 아티팩트를 다운로드하여 구성 요소를 실행합니다. V1 Lambda 함수를 Lambda 런타임 환경에서 실행되는 구성 요소로 가져올 수 있습니다. AWS IoT Greengrass V2 Lambda 함수를 가져올 때 함수에 대한 구독, 로컬 리소스 및 컨테이너 파라미터를 지정합니다. 자세한 설명은 2단계: 애플리케이션을 마이그레이션하기 위한 AWS IoT Greengrass V2 구성 요소 생성 및 배포 AWS

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		IoT Greengrass V1 섹션을 참조하세요. 사용자 지정 구성 요소를 생성하는 방법에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass구성 요소 개발

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 그룹 및 배포	에서 AWS IoT Greengrass V1 그룹은 코어 장치, 해당 핵심 장치의 설정 및 소프트웨어, 해당 핵심 장치에 연결할 수 있는 항목 목록을 정의합니다. AWS IoT 배포를 생성하여 그룹 구성을 코어 장치에 전송합니다.	AWS IoT Greengrass V2에서는 배포를 사용하여 코어 장치에서 실행되는 소프트웨어 구성 요소 및 구성을 정의합니다. • 각 배포는 단일 코어 디바이스 (AWS IoT 사물) 또는 여러 코어 디바이스를 포함할 수 있는 AWS IoT 사물 그룹을 대상으로 합니다. • 사물 그룹으로의 배포는 연속적이므로 사물 그룹에 코어 장치를 추가하면 해당 그룹에 대한 소프트웨어 구성이 제공됩니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요. AWS IoT Greengrass V2에서는 Greengrass CLI를 사용하여 로컬 배포를 생성하여 사용자 지정 소프트웨어 구성 요소를 개발하는 디바이스에서 사용자 지정 소프트웨어 구성 요소를 테스트할 수도 있습니다. 자세한 설명은 AWS IoT Greengrass 구성 요소 생성 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 코어 소프트웨어	에서 AWS IoT Greengrass V1 AWS IoT Greengrass 코어 소프트웨어는 소프트웨어와 모든 기능을 포함하는 단일 패키지입니다. AWS IoT Greengrass 코어 소프트웨어를 설치하는 에지 디바이스를 Greengrass 코어라고 합니다.	에서 AWS IoT Greengrass V2 AWS IoT Greengrass Core 소프트웨어는 모듈식이므로 메모리 사용량을 제어하기 위해 설치할 대상을 선택할 수 있습니다. • Greengrass nucleus 구성 요소는 코어 소프트웨어의 최소 필수 설치입니다. AWS IoT Greengrass 핵을 설치하는 에지 디바이스를 Greengrass 코어 디바이스라고 합니다. • NUCLEUS는 코어 디바이스의 다른 구성 요소에 대한 배포, 오케스트레이션 및 라이프사이클 관리를 처리합니다. • 스트림 관리자, 보안 관리자, 로그 관리자와 같은 기능은 해당 기능이 필요할 때만 배포하는 구성 요소입니다. 자세한 설명은 AWS-제공된 구성 요소 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
커넥터	에서 커넥터는 로컬 인프라 AWS IoT Greengrass V1, 장치 프로토콜 및 기타 클라우드 서비스와 상호 작용하기 위해 AWS IoT Greengrass V1 코어 장치에 배포하는 사전 빌드된 모듈입니다. AWS	AWS IoT Greengrass V2에서는 AWS V1의 커넥터가 제공하는 기능을 구현하는 Greengrass 구성 요소를 제공합니다. 다음 AWS IoT Greengrass V2 구성 요소는 Greengrass V1 커넥터 기능을 제공합니다. • CloudWatch 메트릭 구성 요소 • AWS IoT Device Defender 구성 요소 • Firehose 컴포넌트 • 모드버스-RTU 프로토콜 어댑터 컴포넌트 • 아마존 SNS 컴포넌트 자세한 설명은 AWS-제공된 구성 요소 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
커넥티드 디바이스 (그린그래스 디바이스)	에서 AWS IoT Greengrass V1 커넥티드 디바이스는 AWS IoT Greengrass 그룹에 추가하여 해당 그룹의 코어 디바이스에 연결하고 MQTT를 통해 통신하는 것입니다. 연결된 장치를 추가하거나 제거할 때마다 해당 그룹을 배포해야 합니다. 구독을 사용하면 연결된 장치와 코어 장치의 응용 프로그램 간에 메시지를 릴레이할 수 있습니다. AWS IoT Core	AWS IoT Greengrass V2 에서는 연결된 디바이스를 Greengrass 클라이언트 디바이스라고 합니다. • 클라이언트 디바이스를 코어 디바이스에 연결하여 연결하고 MQTT를 통해 통신합니다. • 클라이언트 장치의 연결을 승인하려면 클라이언트 장치 그룹에 적용할 수 있는 권한 부여 정책을 정의하므로 클라이언트 장치를 추가하거나 제거하기 위해 배포를 만들 필요가 없습니다. • 클라이언트 장치와 Greengrass 구성 요소 간에 메시지를 릴레이하기 위해 선택적 MQTT 브리지 구성 요소를 구성할 수 있습니다. AWS IoT Core AWS IoT Greengrass V1 및 AWS IoT Greengrass V2 모두에서 장치는 FreeRTOS를 실행하거나 AWS IoT Device SDK또는 Greengrass 검색 API를 사용하여 연결할 수 있는 핵심 장치에 대한 정보를 얻을 수 있습니다. Greengrass 검색 API는 이전 버전과 호환되므로 V1 코어 장치에 연결하는 클라이언트 장치가 있는 경우

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		코드를 변경하지 않고 V2 코어 장치에 연결할 수 있습니다. 클라이언트 디바이스에 대한 자세한 내용은 을 참조하십시오. 로컬 IoT 기기와 상호작용
로컬 리소스	에서 AWS IoT Greengrass V1 컨테이너에서 실행되는 Lambda 함수는 코어 디바이스의 파일 시스템에 있는 볼륨과 디바이스에 액세스하도록 구성할 수 있습니다. 이러한 파일 시스템 리소스를 로컬 리소스라고 합니다.	에서는 Lambda 함수 AWS IoT Greengrass V2, Docker 컨테이너 또는 네이티브 운영 체제 프로세스 또는 사용자 지정 런타임인 구성 요소를 실행할 수 있습니다. • 컨테이너화된 Lambda 함수를 구성 요소로 가져올 때는 함수가 사용하는 로컬 리소스를 지정해야 합니다. • 컨테이너화되지 않은 Lambda 함수 및 Lambda 이외의 구성 요소는 코어 디바이스의 로컬 리소스와 직접 작동할 수 있으므로 구성 요소가 사용하는 로컬 리소스를 지정할 필요가 없습니다.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
로컬 섀도우 서비스	AWS IoT Greengrass V1에서는 로컬 섀도우 서비스가 기본적으로 활성화되며 이름이 지정되지 않은 클래식 섀도만 지원합니다. Lambda 함수의 AWS IoT Greengrass Core SDK를 사용하여 디바이스의 섀도우와 상호 작용할 수 있습니다.	AWS IoT Greengrass V2에서는 섀도우 관리자 구성 요소를 배포하여 로컬 섀도우 서비스를 활성화합니다. • Lambda 함수의 AWS IoT Device SDK V2와 사용자 지정 구성 요소를 사용하여 디바이스의 섀도우와 상호 작용할 수 있습니다. • 로컬 섀도우 서비스는 명명된 섀도우를 지원합니다. • 로컬 섀도우 서비스를 사용하면 섀도우를 삭제하고 삭제된 섀도우를 동기화할 수 있습니다 AWS IoT Core. 자세한 설명은 디바이스 섀도우와 상호작용 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2

구독	에서는 AWS IoT Greengrass V1 Greengrass 그룹에 대한 구독을 정의하여 Lambda 함수, 커넥터, 연결된 디바이스, MQTT 브로커 및 로컬 섀도우 서비스 간의 통신 AWS IoT Core 채널을 지정합니다. 구독은 Lambda 함수가 이벤트 메시지를 수신하여 함수 페이로드로 사용할 위치를 지정합니다.	AWS IoT Greengrass V2에서는 구독을 사용하지 않고 통신 채널을 지정합니다.
		• 구성 요소는 자체 통신 채널을 관리하여 로컬 게시/구독 메시지, AWS IoT Core MQTT 메시지 및 로컬 섀도우 서비스와 상호 작용합니다.
		  • 다른 구성 요소나 AWS IoT Core MQTT 브로커의 메시지에 반응하는 구성 요소를 개발하려면 로컬 게시/구독 메시징 및 MQTT 메시징을 위한 IPC(프로세스 간 통신) 인터페이스를 사용할 수 있습니다. AWS IoT Core
		  • 로컬 섀도우 서비스와 상호 작용하는 구성 요소를 개발하려면 로컬 섀도우 서비스용 IPC 인터페이스를 사용할 수 있습니다.
		  • 구성 요소 구성에서는 구성 요소에 사용 권한이 있는 주제 및 로컬 섀도우를 지정하는 권한 부여 정책을 정의합니다.
		• 클라이언트 장치, 로컬 게시/구독 브로커 및 MQTT 브로커 간의 통신 채널을 구성하려면 AWS IoT Core MQTT 브리지 구성 요소를 구성하

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		고 배포합니다. MQTT 브리지 구성 요소를 사용하면 구성 요소의 클라이언트 장치와 상호 작용하고 클라이언트 장치 간에 메시지를 릴레이할 수 있습니다. AWS IoT Core
다른 사람 액세스 AWS 서비스	AWS IoT Greengrass V1에서는 그룹 역할이라고 하는 AWS Identity and Access Management (IAM) 역할을 Greengrass 그룹에 연결합니다. 그룹 역할은 해당 그룹의 핵심 디바이스에서 Lambda 함수 AWS IoT Greengrass 및 기능이 액세스하는 데 사용하는 권한을 정의합니다. AWS 서비스	AWS IoT Greengrass V2에서는 Greengrass 코어 장치에 AWS IoT 역할 별칭을 연결합니다. 역할 별칭은 토큰 교환 역할이라는 IAM 역할을 가리킵니다. 토큰 교환 역할은 코어 디바이스의 Greengrass 구성 요소가 액세스하는 데 사용하는 권한을 정의합니다. AWS 서비스 자세한 내용은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스을(를) 참조하세요.

V1 코어 디바이스에서 V2 소프트웨어를 실행할 수 있는지 확인

AWS IoT GreengrassCore 소프트웨어 v2.x는AWS IoT Greengrass Core 소프트웨어 v1.x와 요구 사항이 다릅니다. V1 코어 장치를 V2로 업그레이드하기 전에 에 대한 장치 요구 사항을AWS IoT Greengrass V2 검토하십시오. AWS IoT Greengrass V2는 현재 Yocto Project를 사용하는 사용자 지정 Linux 기반 시스템의 마이그레이션을 지원하지 않습니다.

AWS IoT Device Tester(IDT) 를 사용하여 장치가AWS IoT Greengrass Core 소프트웨어 v2.x를AWS IoT Greengrass V2 실행하기 위한 요구 사항을 충족하는지 확인할 수 있습니다. IDT는 호스트 컴퓨터에서 실행되고 검증할 장치에 연결하는 다운로드 가능한 테스트 프레임워크입니다. 지침에 따라 IDT를 사용하여AWS IoT Greengrass 검증 제품군을 실행하십시오. IDT를 구성할 때 장치가 Docker, 머신 러닝 (ML), 데이터 스트림 관리 및 하드웨어 보안 통합과 같은 선택적 기능을 지원하는지 여부를 검증하도록 선택할 수 있습니다.

IDT에서 V1 코어 장치에 대한 V2 테스트 실패 또는 오류를 보고하는 경우 해당 장치를 V1에서 V2로 업그레이드할 수 없습니다.

새 V2 코어 디바이스를 설정하여 V1 애플리케이션을 테스트합니다.

새 AWS IoT Greengrass V2 코어 디바이스를 설정하여 AWS IoT Greengrass V1 애플리케이션에 AWS 제공된 구성 요소 및 AWS Lambda 기능을 배포하고 테스트하십시오. 또한 이 V2 코어 디바이스를 사용하여 코어 디바이스에서 네이티브 프로세스를 실행하는 추가 사용자 지정 Greengrass 구성 요소를 개발하고 테스트할 수 있습니다. V2 코어 디바이스에서 애플리케이션을 테스트한 후 기존 V1 코어 디바이스를 V2로 업그레이드하고 V1 기능을 제공하는 V2 구성 요소를 배포할 수 있습니다.

1단계: 새 AWS IoT Greengrass V2 기기에 설치

새 장치에 AWS IoT Greengrass Core 소프트웨어 v2.x를 설치합니다. 시작 자습서를 따라 장치를 설정하고 구성 요소를 개발하고 배포하는 방법을 배울 수 있습니다. 이 자습서에서는 자동 프로비저닝을 사용하여 장치를 빠르게 설정합니다. AWS IoT Greengrass Core 소프트웨어 v2.x를 설치할 때 Greengrass CLI를 배포할 `--deploy-dev-tools` 인수를 지정하여 디바이스에서 직접 구성 요소를 개발, 테스트 및 디버그할 수 있습니다. 프록시를 사용하거나 하드웨어 보안 모듈 (HSM) 을 사용하여 AWS IoT Greengrass Core 소프트웨어를 설치하는 방법을 비롯한 기타 설치 옵션에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass 코어 소프트웨어 설치

(선택 사항) Amazon CloudWatch Logs에 로깅을 활성화합니다.

V2 코어 디바이스가 Amazon CloudWatch Logs에 로그를 업로드할 수 있게 하려면 AWS 제공된 로그 관리자 구성 요소를 배포하면 됩니다. CloudWatch 로그를 사용하여 구성 요소 로그를 볼 수 있으므로 코어 디바이스의 파일 시스템에 액세스하지 않고도 디버깅하고 문제를 해결할 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

2단계: 애플리케이션을 마이그레이션하기 위한 AWS IoT Greengrass V2 구성 요소 생성 및 배포 AWS IoT Greengrass V1

에서 대부분의 AWS IoT Greengrass V1 애플리케이션을 실행할 AWS IoT Greengrass V2 수 있습니다. Lambda 함수를 AWS IoT Greengrass V2 에서 실행되는 구성 요소로 가져올 수 있으며 커넥터와 동일한 기능을 제공하는 AWS -provide 구성 요소를 사용할 수 있습니다. AWS IoT Greengrass

또한 사용자 지정 구성 요소를 개발하여 Greengrass 코어 장치에서 실행할 기능 또는 런타임을 빌드할 수 있습니다. 로컬에서 구성 요소를 개발하고 테스트하는 방법에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass 구성 요소 생성.

주제

- V1 람다 함수 가져오기
- V1 커넥터 사용
- 도커 컨테이너 실행
- 기계 학습 추론을 실행합니다.
- Connect V1 그린그래스 디바이스
- 로컬 섀도우 서비스를 활성화합니다.
- 다음과 통합하십시오. AWS IoT SiteWise

V1 람다 함수 가져오기

Lambda 함수를 구성 요소로 가져올 수 있습니다. AWS IoT Greengrass V2 다음 접근 방식 중에서 선택하십시오.

- V1 Lambda 함수를 Greengrass 구성 요소로 직접 가져올 수 있습니다.
- v2에서 AWS IoT Device SDK Greengrass 라이브러리를 사용하도록 Lambda 함수를 업데이트한 다음 Lambda 함수를 Greengrass 구성 요소로 가져옵니다.
- Lambda 함수와 동일한 기능을 구현하기 위해 비 Lambda 코드와 AWS IoT Device SDK v2를 사용하는 사용자 지정 구성 요소를 생성합니다.

Lambda 함수가 스트림 관리자 또는 로컬 암호와 같은 기능을 사용하는 경우, 이러한 기능을 패키징하는 제공된 구성 요소에 대한 AWS 종속성을 정의해야 합니다. Lambda 함수 구성 요소를 배포할 때 배포에는 종속성으로 정의한 각 기능에 대한 구성 요소도 포함됩니다. 배포 시 코어 디바이스에 배포할 암호와 같은 파라미터를 구성할 수 있습니다. 모든 V1 기능에 V2의 Lambda 함수에 대한 구성 요소 종속성이 필요한 것은 아닙니다. 다음 목록은 V2 Lambda 함수 구성 요소에서 V1 기능을 사용하는 방법을 설명합니다.

- 다른 서비스에 액세스 AWS

  Lambda 함수가 자격 증명을 AWS 사용하여 AWS 다른 서비스에 요청하는 경우, 코어 디바이스의 토큰 교환 역할은 코어 디바이스가 Lambda 함수가 사용하는 작업을 수행할 AWS 수 있도록 허용해야 합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.
- 스트림 매니저

Lambda 함수가 스트림 관리자를 사용하는 경우 함수를 가져올 때 구성 요소 종속성으로 `aws.greengrass.StreamManager` 지정하십시오. 스트림 관리자 구성 요소를 배포할 때 대상 코어 디바이스에 설정할 스트림 관리자 파라미터를 지정하십시오. 코어 디바이스의 토큰 교환 역할은 코어 디바이스가 스트림 관리자와 함께 사용하는 AWS 클라우드 대상에 액세스할 수 있도록 허용해야 합니다. 자세한 설명은 스트림 관리자 섹션을 참조하세요.

- 로컬 시크릿

  Lambda 함수가 로컬 암호를 사용하는 경우 함수를 가져올 때 구성 요소 종속성으로 `aws.greengrass.SecretManager` 지정하십시오. Secret Manager 구성 요소를 배포할 때는 대상 코어 디바이스에 배포할 비밀 리소스를 지정하십시오. 코어 디바이스의 토큰 교환 역할은 코어 디바이스가 배포할 비밀 리소스를 검색할 수 있도록 허용해야 합니다. 자세한 설명은 시크릿 매니저 섹션을 참조하세요.

  Lambda 함수 구성 요소를 배포할 때 V2에서 IPC 작업을 사용할 권한을 부여하는 IPC 권한 부여 정책을 갖도록 GetSecretValue 구성하십시오. AWS IoT Device SDK

- 로컬 섀도우

  Lambda 함수가 로컬 섀도우와 상호 작용하는 경우 V2를 사용하도록 Lambda 함수 코드를 업데이트해야 합니다. AWS IoT Device SDK 또한 함수를 가져올 때 구성 요소 종속성으로 `aws.greengrass.ShadowManager` 지정해야 합니다. 자세한 설명은 디바이스 섀도우와 상호작용 섹션을 참조하세요.

  Lambda 함수 구성 요소를 배포할 때 V2에서 섀도우 IPC 작업을 사용할 권한을 부여하는 IPC 권한 부여 정책을 포함하도록 구성하십시오. AWS IoT Device SDK

- 서브스크립션
  - Lambda 함수가 클라우드 소스의 메시지를 구독하는 경우 함수를 가져올 때 해당 구독을 이벤트 소스로 지정하십시오.
  - Lambda 함수가 다른 Lambda 함수의 메시지를 구독하거나 Lambda 함수가 AWS IoT Core 메시지를 또는 다른 Lambda 함수에 게시하는 경우, Lambda 함수를 배포할 때 기존 구독 라우터 구성 요소를 구성하고 배포하십시오. 레거시 구독 라우터 구성 요소를 배포할 때 Lambda 함수가 사용하는 구독을 지정하십시오.

    > **Note**
    >
    > Lambda 함수가 Core SDK의 함수를 사용하는 `publish()` 경우에만 기존 구독 라우터 구성 요소가 필요합니다. AWS IoT Greengrass V2의 IPC (프로세스 간 통신) 인터페이스를

사용하도록 Lambda 함수 코드를 업데이트하면 기존 구독 라우터 구성 요소를 배포할 필요가 없습니다. AWS IoT Device SDK 자세한 내용은 다음 프로세스 간 통신 서비스를 참조하십시오.

- 로컬 메시지 게시/구독
- MQTT 메시지 게시/구독 AWS IoT Core

- Lambda 함수가 로컬 연결 디바이스의 메시지를 구독하는 경우 함수를 가져올 때 해당 구독을 이벤트 소스로 지정하십시오. 또한 연결된 디바이스의 메시지를 이벤트 소스로 지정한 로컬 게시/구독 주제로 릴레이하도록 MQTT 브리지 구성 요소를 구성하고 배포해야 합니다.
- Lambda 함수가 로컬 연결 디바이스에 메시지를 게시하는 경우 V2를 사용하여 로컬 게시/구독 메시지를 게시하도록 Lambda 함수 코드를 업데이트해야 합니다. AWS IoT Device SDK 또한 로컬 게시/구독 메시지 브로커의 메시지를 연결된 디바이스로 릴레이하도록 MQTT 브리지 구성 요소를 구성하고 배포해야 합니다.

- 로컬 볼륨 및 디바이스

  컨테이너화된 Lambda 함수가 로컬 볼륨 또는 디바이스에 액세스하는 경우 Lambda 함수를 가져올 때 해당 볼륨과 디바이스를 지정하십시오. 이 기능에는 구성 요소 종속성이 필요하지 않습니다.

자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

V1 커넥터 사용

일부 AWS IoT Greengrass 커넥터와 동일한 기능을 AWS 제공하는 제공 구성 요소를 배포할 수 있습니다. 배포를 생성할 때 커넥터의 매개 변수를 구성할 수 있습니다.

다음 AWS IoT Greengrass V2 구성 요소는 Greengrass V1 커넥터 기능을 제공합니다.

- CloudWatch 메트릭 구성 요소
- AWS IoT Device Defender 구성 요소
- Firehose 컴포넌트
- 모드버스-RTU 프로토콜 어댑터 컴포넌트
- 아마존 SNS 컴포넌트

도커 컨테이너 실행

AWS IoT Greengrass V2 V1 Docker 애플리케이션 배포 커넥터를 직접 대체할 구성 요소를 제공하지 않습니다. 하지만 Docker 애플리케이션 관리자 구성 요소를 사용하여 Docker 이미지를 다운로드한 다음 다운로드한 이미지에서 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 만들 수 있습니다. 자세한 내용은 도커 컨테이너 실행 및 Docker 애플리케이션 관리자 섹션을 참조하세요.

기계 학습 추론을 실행합니다.

AWS IoT Greengrass V2 Amazon SageMaker Edge Manager 에이전트를 설치하고 Greengrass 코어 디바이스에서 SageMaker NEO 컴파일된 모델을 모델 구성 요소로 사용할 수 있도록 하는 Amazon SageMaker Edge Manager 구성 요소를 제공합니다. AWS IoT Greengrass V2 디바이스에 딥 러닝 런타임 및 TensorFlow Lite를 설치하는 구성 요소도 제공합니다. 해당 DLR 및 TensorFlow Lite 모델 및 추론 구성 요소를 사용하여 샘플 이미지 분류 및 객체 감지 추론을 수행할 수 있습니다. TensorFlowMXNet과 같은 다른 기계 학습 프레임워크를 사용하려면 이러한 프레임워크를 사용하는 사용자 지정 구성 요소를 직접 개발할 수 있습니다.

Connect V1 그린그래스 디바이스

에서는 AWS IoT Greengrass V1 연결된 디바이스를 클라이언트 디바이스라고 합니다. AWS IoT Greengrass V2 AWS IoT Greengrass V2 클라이언트 디바이스에 대한 지원은 이전 버전과 AWS IoT Greengrass V1 호환되므로 애플리케이션 코드를 변경하지 않고도 V1 클라이언트 디바이스를 V2 코어 디바이스에 연결할 수 있습니다. 클라이언트 디바이스를 V2 코어 디바이스에 연결할 수 있게 하려면 클라이언트 디바이스 지원을 지원하는 Greengrass 구성 요소를 배포하고 클라이언트 디바이스를 코어 디바이스에 연결합니다. 클라이언트 디바이스, AWS IoT Core 클라우드 서비스 및 Greengrass 구성 요소 (Lambda 함수 포함) 간에 메시지를 릴레이하려면 MQTT 브리지 구성 요소를 배포하고 구성하십시오. IP 탐지기 구성 요소를 배포하여 연결 정보를 자동으로 탐지하거나 엔드포인트를 수동으로 관리할 수 있습니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

로컬 섀도우 서비스를 활성화합니다.

에서 AWS IoT Greengrass V2 로컬 섀도우 서비스는 AWS 제공된 섀도우 관리자 구성 요소에 의해 구현됩니다. AWS IoT Greengrass V2 명명된 섀도우에 대한 지원도 포함됩니다. 구성 요소가 로컬 섀도우와 상호 작용하고 섀도우 상태를 동기화하고 섀도우 관리자 구성 요소를 구성 및 배포하고 구성 요소 코드에서 섀도우 IPC 작업을 사용할 수 있도록 하려면 AWS IoT Core 자세한 설명은 디바이스 섀도우와 상호작용 섹션을 참조하세요.

다음과 통합하십시오. AWS IoT SiteWise

V1 코어 디바이스를 AWS IoT SiteWise 게이트웨이로 사용하는 경우 지침에 따라 새 V2 코어 디바이스를 AWS IoT SiteWise 게이트웨이로 설정하십시오. AWS IoT SiteWise AWS IoT SiteWise 구성 요소를 자동으로 배포하는 설치 스크립트를 제공합니다.

3단계: 애플리케이션 AWS IoT Greengrass V2 테스트

V2 구성 요소를 만들어 새 V2 코어 기기에 배포한 후 애플리케이션이 기대치를 충족하는지 확인하십시오. 디바이스의 로그를 확인하여 구성 요소의 표준 출력 (stdout) 및 표준 오류 (stderr) 메시지를 볼 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

Greengrass CLI를 코어 디바이스에 배포한 경우 이를 사용하여 구성 요소 및 해당 구성을 디버깅할 수 있습니다. 자세한 설명은 그린그래스 CLI 명령 섹션을 참조하세요.

애플리케이션이 V2 코어 디바이스에서 작동하는지 확인한 후 애플리케이션의 Greengrass 구성 요소를 다른 코어 디바이스에 배포할 수 있습니다. 네이티브 프로세스나 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 개발한 경우 먼저 해당 구성 요소를 AWS IoT Greengrass 서비스에 게시하여 다른 핵심 장치에 배포해야 합니다.

그린그래스 V1 코어 디바이스를 그린그래스 V2로 업그레이드

애플리케이션과 구성 요소가 AWS IoT Greengrass V2 코어 장치에서 작동하는지 확인한 후 현재 v1.x를 실행하는 장치 (예: 프로덕션 장치) 에 AWS IoT Greengrass Core 소프트웨어 v2.x를 설치할 수 있습니다. 그런 다음 Greengrass V2 구성 요소를 배포하여 장치에서 Greengrass 애플리케이션을 실행하십시오.

여러 디바이스를 V1에서 V2로 업그레이드하려면 업그레이드할 각 디바이스에 대해 다음 단계를 완료하세요. 사물 그룹을 사용하여 V2 구성 요소를 코어 디바이스 플릿에 배포할 수 있습니다.

> **Tip**
>
> 여러 디바이스의 업그레이드 프로세스를 자동화하는 스크립트를 생성하는 것이 좋습니다. AWS Systems Manager플릿을 관리하는 데 사용하는 경우 Systems Manager를 사용하여 각 디바이스에서 해당 스크립트를 실행하여 플릿을 V1에서 V2로 업그레이드할 수 있습니다. 업그레이드 프로세스를 가장 잘 자동화하는 방법에 대한 질문은 AWS Enterprise Support 담당자에게 문의할 수 있습니다.

1단계: AWS IoT Greengrass Core 소프트웨어 v2.x 설치

다음 옵션 중에서 선택하여 V1 AWS IoT Greengrass 코어 장치에 Core 소프트웨어 v2.x를 설치합니다.

- 더 적은 단계로 업그레이드할 수 있습니다.

  더 적은 단계로 업그레이드하려면 v2.x 소프트웨어를 설치하기 전에 v1.x 소프트웨어를 제거하면 됩니다.
- 다운타임을 최소화하면서 업그레이드하세요

  가동 중지 시간을 최소화하면서 업그레이드하려면 두 버전의 AWS IoT Greengrass Core 소프트웨어를 동시에 설치할 수 있습니다. AWS IoT GreengrassCore 소프트웨어 v2.x를 설치하고 V2 애플리케이션이 제대로 작동하는지 확인한 후 Core 소프트웨어 v1.x를 제거합니다. AWS IoT Greengrass 이 옵션을 선택하기 전에 두 버전의 AWS IoT Greengrass Core 소프트웨어를 동시에 실행하는 데 필요한 추가 RAM을 고려하십시오.

v2.x를 설치하기 전에 AWS IoT Greengrass Core v1.x를 제거하십시오.

순차적으로 업그레이드하려면 디바이스에 v2.x를 설치하기 전에 AWS IoT Greengrass Core 소프트웨어 v1.x를 제거해야 합니다.

Core 소프트웨어 v1.x를 제거하려면 AWS IoT Greengrass

1. AWS IoT GreengrassCore 소프트웨어 v1.x가 서비스로 실행되는 경우 서비스를 중지, 비활성화 및 제거해야 합니다.

   a. 실행 중인 AWS IoT Greengrass Core 소프트웨어 v1.x 서비스를 중지합니다.

   ```
   sudo systemctl stop greengrass
   ```

   b. 서비스가 중지될 때까지 기다리십시오. `list`명령을 사용하여 서비스 상태를 확인할 수 있습니다.

   ```
   sudo systemctl list-units --type=service | grep greengrass
   ```

   c. 서비스를 비활성화합니다.

   ```
   sudo systemctl disable greengrass
   ```

d. 서비스를 제거합니다.

```
sudo rm /etc/systemd/system/greengrass.service
```

2. AWS IoT GreengrassCore 소프트웨어 v1.x가 서비스로 실행되지 않는 경우 다음 명령을 사용하여 데몬을 중지하십시오. *#####-### ##### ##* 폴더의 이름으로 바꾸십시오. 기본 위치는 `/greengrass`입니다.

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

3. (선택 사항) Greengrass 루트 폴더와 해당하는 경우 사용자 지정 쓰기 폴더를 디바이스의 다른 폴더에 백업합니다.

a. 다음 명령을 사용하여 현재 Greengrass 루트 폴더를 다른 폴더에 복사한 다음 루트 폴더를 제거합니다.

```
sudo cp -r /greengrass-root /path/to/greengrass-backup
rm -rf /greengrass-root
```

b. 다음 명령을 사용하여 쓰기 폴더를 다른 폴더로 이동한 다음 쓰기 폴더를 제거합니다.

```
sudo cp -r /write-directory /path/to/write-directory-backup
rm -rf /write-directory
```

그런 다음 의 설치 지침을 사용하여 디바이스에 소프트웨어를 설치할 수 있습니다. AWS IoT Greengrass V2

> **Tip**
>
> V1에서 V2로 마이그레이션할 때 코어 장치 ID를 재사용하려면 지침에 따라 수동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치하십시오. 먼저 장치에서 V1 코어 소프트웨어를 제거한 다음 V1 코어 장치의 사물과 인증서를 재사용하고 인증서 AWS IoT 정책을 업데이트하여 v2.x 소프트웨어에 필요한 권한을 부여하십시오. AWS IoT

이미 v1.x를 AWS IoT Greengrass 실행 중인 장치에 Core 소프트웨어 v2.x를 설치합니다.

AWS IoT GreengrassCore 소프트웨어 v1.x를 이미 실행 중인 장치에 Core v2.x 소프트웨어를 설치하는 경우 다음 AWS IoT Greengrass 사항에 유의하십시오.

- V2 코어 디바이스의 AWS IoT 사물 이름은 고유해야 합니다. V1 코어 디바이스와 같은 사물 이름을 사용하지 마세요.
- AWS IoT GreengrassCore 소프트웨어 v2.x에 사용하는 포트는 v1.x에 사용하는 포트와 달라야 합니다.
  - 8088이 아닌 포트를 사용하도록 V1 스트림 관리자를 구성하십시오. 자세한 내용은 스트림 관리자 구성을 참조하십시오.
  - 8883 이외의 포트를 사용하도록 V1 MQTT 브로커를 구성합니다. 자세한 내용은 로컬 메시징을 위한 MQTT 포트 구성을 참조하십시오.
- AWS IoT Greengrass V2Greengrass 시스템 서비스의 이름을 바꾸는 옵션을 제공하지 않습니다. Greengrass를 시스템 서비스로 실행하는 경우 시스템 서비스 이름이 충돌하지 않도록 다음 중 하나를 수행해야 합니다.
  - v2.x를 설치하기 전에 v1.x용 Greengrass 서비스 이름을 변경하십시오.
  - 시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어 v2.x를 설치한 다음 소프트웨어를 이름이 아닌 시스템 서비스로 수동으로 구성합니다. `greengrass`

  v1.x용 Greengrass 서비스 이름을 변경하려면

  1. AWS IoT Greengrass코어 소프트웨어 v1.x 서비스를 중지합니다.

     ```
     sudo systemctl stop greengrass
     ```

  2. 서비스가 중지될 때까지 기다리세요. 서비스를 중지하는 데 몇 분 정도 걸릴 수 있습니다. `list-units`명령을 사용하여 서비스가 중지되었는지 확인할 수 있습니다.

     ```
     sudo systemctl list-units --type=service | grep greengrass
     ```

  3. 서비스를 비활성화합니다.

     ```
     sudo systemctl disable greengrass
     ```

  4. 서비스 이름을 변경합니다.

```
sudo mv /etc/systemd/system/greengrass.service /etc/systemd/system/greengrass-v1.service
```

5. 서비스를 다시 로드하고 시작합니다.

```
sudo systemctl daemon-reload
sudo systemctl reset-failed
sudo systemctl enable greengrass-v1
sudo systemctl start greengrass-v1
```

그런 다음 의 설치 지침을 사용하여 장치에 소프트웨어를 설치할 수 있습니다. AWS IoT Greengrass V2

> **Tip**
>
> V1에서 V2로 마이그레이션할 때 코어 장치 ID를 재사용하려면 지침에 따라 수동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치하십시오. 먼저 장치에서 V1 코어 소프트웨어를 제거한 다음 V1 코어 장치의 사물과 인증서를 재사용하고 인증서 AWS IoT 정책을 업데이트하여 v2.x 소프트웨어에 필요한 권한을 부여하십시오. AWS IoT

2단계: 코어 디바이스에 구성 요소 배포 AWS IoT Greengrass V2

AWS IoT GreengrassCore 소프트웨어 v2.x를 디바이스에 설치한 후 다음 리소스가 포함된 배포를 생성하십시오. 유사한 장치 집합에 구성 요소를 배포하려면 해당 장치가 포함된 사물 그룹에 대한 배포를 생성하십시오.

- V1 Lambda 함수에서 생성한 Lambda 함수 구성 요소. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.
- V1 구독을 사용하는 경우, 레거시 구독 라우터 구성 요소입니다.
- 스트림 관리자를 사용하는 경우, 스트림 관리자 구성 요소입니다. 자세한 설명은 Greengrass 코어 디바이스의 데이터 스트림 관리 섹션을 참조하세요.
- 로컬 시크릿을 사용하는 경우 시크릿 매니저 컴포넌트입니다.
- V1 커넥터를 사용하는 경우 AWS-제공된 커넥터 구성 요소
- Docker 컨테이너를 사용하는 경우 Docker 애플리케이션 관리자 구성 요소입니다. 자세한 설명은 도커 컨테이너 실행 섹션을 참조하세요.

- 기계 학습 추론을 사용하는 경우 기계 학습을 위한 구성 요소가 지원됩니다. 자세한 설명은 기계 학습 추론 수행 섹션을 참조하세요.
- 커넥티드 디바이스를 사용하는 경우 클라이언트 디바이스용 구성 요소가 지원됩니다. 또한 클라이언트 장치 지원을 활성화하고 클라이언트 장치를 코어 장치에 연결해야 합니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.
- 디바이스 섀도를 사용하는 경우 섀도우 관리자 구성 요소입니다. 자세한 설명은 디바이스 섀도우와 상호작용 섹션을 참조하세요.
- Greengrass 코어 디바이스에서 로그 관리자 구성 요소인 Amazon Logs로 CloudWatch 로그를 업로드하는 경우 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.
- 와 통합하는 AWS IoT SiteWise 경우 지침에 따라 V2 코어 디바이스를 AWS IoT SiteWise 게이트웨이로 설정하십시오. AWS IoT SiteWiseAWS IoT SiteWise구성 요소를 자동으로 배포하는 설치 스크립트를 제공합니다.
- 사용자 지정 기능을 구현하기 위해 개발한 사용자 정의 구성 요소입니다.

배포 생성 및 수정에 대한 자세한 내용은 을 참조하십시오. 디바이스에 AWS IoT Greengrass 구성 요소 배포

자습서: AWS IoT Greengrass V2 시작하기

이 시작 자습서를 완료하여 의 AWS IoT Greengrass V2 기본 기능을 배울 수 있습니다. 이 자습서에서는 다음 작업을 수행합니다.

1. 라즈베리파이와 같은 리눅스 디바이스 또는 윈도우 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하고 구성하세요. 이 장치는 Greengrass 코어 디바이스입니다.
2. 그린그래스 코어 디바이스에서 Hello World 구성 요소를 개발하십시오. 구성 요소는 Greengrass 코어 장치에서 실행되는 소프트웨어 모듈입니다.
3. 해당 구성 요소를 AWS IoT Greengrass V2 in에 AWS 클라우드 업로드하십시오.
4. 해당 구성 요소를 Greengrass 코어 AWS 클라우드 장치에 배포하십시오.

> **Note**
>
> 이 자습서에서는 개발 환경을 설정하고 의 AWS IoT Greengrass 기능을 탐색하는 방법을 설명합니다. 프로덕션 디바이스를 설정하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.
>
> - AWS IoT Greengrass코어 디바이스 설정
> - AWS IoT Greengrass 코어 소프트웨어 설치

이 자습서에는 20~30분이 소요될 것으로 예상됩니다.

주제

사전 조건

이 시작하기 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 계정이 없는 경우 1단계: AWS 계정 설정 섹션을 참조하십시오.
- AWS 리전지원하는 팬의 사용AWS IoT Greengrass V2. 지원되는 리전 목록은 AWS 일반 참조에서 AWS IoT Greengrass V2 엔드포인트 및 할당량을 참조하십시오.
- 관리자 권한이 있는 AWS Identity and Access Management (IAM) 사용자.
- 라즈베리 파이 OS가 설치된 라즈베리 파이 (이전에는 라즈비안이라고 함) 또는 윈도우 10 디바이스와 같이 그린그래스 코어 디바이스로 설정하기 위한 디바이스입니다. 이 디바이스에 대한 관리자 권한이 있거나 관리자 권한을 획득할 수 있는 능력 (예:) 이 있어야 합니다. `sudo` 이 장치는 인터넷에 연결되어 있어야 합니다.

  AWS IoT GreengrassCore 소프트웨어를 설치하고 실행하기 위한 요구 사항을 충족하는 다른 장치를 사용하도록 선택할 수도 있습니다. 자세한 설명은 지원되는 플랫폼 및 요구 사항 섹션을 참조하세요.

  개발 컴퓨터가 이러한 요구 사항을 충족하는 경우 이 자습서에서 Greengrass 코어 장치로 설정할 수 있습니다.
- Python 3.5 이상이 기기의 모든 사용자를 위해 설치되고 `PATH` 환경 변수에 추가되었습니다. 윈도우에서는 모든 사용자를 위한 윈도우용 Python 런처도 설치되어 있어야 합니다.

  > **Important**
  >
  > Windows에서 Python은 기본적으로 모든 사용자에게 설치되지 않습니다. Python을 설치할 때 AWS IoT Greengrass Core 소프트웨어가 Python 스크립트를 실행하도록 구성하도록 설치를 사용자 정의해야 합니다. 예를 들어, 그래픽 Python 설치 프로그램을 사용하는 경우 다음을 수행하십시오.
  >
  > 1. 모든 사용자를 위한 런처 설치를 선택합니다 (권장).
  > 2. Customize installation를 선택합니다.
  > 3. Next를 선택합니다.
  > 4. Install for all users을(를) 선택합니다.
  > 5. Add Python to environment variables을(를) 선택합니다.
  > 6. 설치를 선택합니다.

자세한 내용은 Python 3 설명서의 윈도우에서 Python 사용을 참조하십시오.

- AWS Command Line Interface(AWS CLI) 개발 컴퓨터와 장치에 자격 증명을 사용하여 설치 및 구성되었습니다. 개발 컴퓨터와 장치에서 동일한 AWS 리전 방법을 사용하여 구성해야 합니다. AWS CLI AWS IoT Greengrass V2와 함께 사용하려면 다음 버전 중 하나 이상이 있어야 합니다. AWS CLI
  - AWS CLIV1 최소 버전: v1.18.197
  - 최소 V2 버전: v2.1.11 AWS CLI

> **Tip**
>
> 다음 명령을 실행하여 사용 중인 버전을 확인할 수 있습니다. AWS CLI
>
> ```
> aws --version
> ```

자세한 내용은 AWS Command Line Interface사용 설명서의 설치, 업데이트 AWS CLI 및 제거 및 구성을 참조하십시오. AWS CLI

> **Note**
>
> 32비트 운영 체제의 Raspberry Pi와 같은 32비트 ARM 장치를 사용하는 경우 V1을 설치하십시오. AWS CLI AWS CLI V2는 32비트 ARM 디바이스에서 사용할 수 없습니다. 자세한 내용은 버전 1 설치, 업데이트 및 제거를 AWS CLI 참조하십시오.

1단계: AWS 계정 설정

AWS 계정에 등록

AWS 계정 항목이 없으면 다음 절차에 따라 생성하십시오.

AWS 계정에 가입하려면

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

가입 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

AWS 계정에 가입하면 AWS 계정 루트 사용자 항목이 생성됩니다. 루트 사용자에게 계정의 모든 AWS 서비스 및 리소스에 대한 액세스 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자를 보안하며 AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 AWS Management Console에 계정 소유자로 로그인합니다. 다음 페이지에서 암호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하십시오.
2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자용 가상 MFA 디바이스 활성화(콘솔) 섹션을 참조하십시오.

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서에서 Enabling AWS IAM Identity Center를 참조하세요.
2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하십시오.

2단계: 환경 설정

이 섹션의 단계에 따라 AWS IoT Greengrass 핵심 장치로 사용할 Linux 또는 Windows 장치를 설정합니다.

리눅스 디바이스 (라즈베리파이) 설정

이 단계에서는 라즈베리 파이 OS와 함께 라즈베리 파이를 사용한다고 가정합니다. 다른 장치나 운영 체제를 사용하는 경우 해당 장치의 관련 설명서를 참조하십시오.

라즈베리 파이를 설정하려면 AWS IoT Greengrass V2

1. 라즈베리파이에서 SSH를 활성화하여 원격으로 연결합니다. 자세한 내용은 라즈베리 파이 설명서의 SSH (보안 셸) 를 참조하십시오.
2. 라즈베리파이의 IP 주소를 찾아 SSH로 연결하세요. 이렇게 하려면 라즈베리파이에서 다음 명령을 실행하면 됩니다.

   ```
   hostname -I
   ```

3. SSH로 라즈베리파이에 연결하세요.

   개발 컴퓨터에서 다음 명령을 실행합니다. *###* 이름을 로그인할 사용자 이름으로 바꾸고 이전 단계에서 찾은 IP 주소로 *pi-ip-address*바꾸십시오.

   ```
   ssh username@pi-ip-address
   ```

   > **Important**
   >
   > 개발 컴퓨터에서 이전 버전의 Windows를 사용하는 경우 ssh 명령이 없거나 Raspberry Pi에 ssh 연결했지만 연결하지 못할 수 있습니다. 라즈베리 파이에 연결하려면 무료 오픈 소

스 SSH 클라이언트인 PuTTY를 설치하고 구성하면 됩니다. PuTTY 설명서를 참조하여 라즈베리 파이에 연결하세요.

4. AWS IoT GreengrassCore 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. 라즈베리파이에서 다음 명령을 사용하여 Java 11을 설치합니다.

```
sudo apt install default-jdk
```

설치가 완료되면 다음 명령을 실행하여 자바가 라즈베리 파이에서 실행되는지 확인합니다.

```
java -version
```

이 명령은 기기에서 실행되는 Java 버전을 인쇄합니다. 출력은 다음 예제와 비슷할 수 있습니다.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

팁: 라즈베리 파이에서 커널 파라미터 설정하기

기기가 Raspberry Pi인 경우 다음 단계를 완료하여 Linux 커널 매개변수를 확인하고 업데이트할 수 있습니다.

1. `/boot/cmdline.txt` 파일을 엽니다. 이 파일은 Raspberry Pi 부팅 시 적용할 Linux 커널 매개변수를 지정합니다.

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 열 수 있습니다.

   ```
   sudo nano /boot/cmdline.txt
   ```

2. `/boot/cmdline.txt`파일에 다음 커널 파라미터가 포함되어 있는지 확인하십시오. 이 `systemd.unified_cgroup_hierarchy=0` 파라미터는 cgroups v2 대신 cgroups v1을 사용하도록 지정합니다.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

/boot/cmdline.txt파일에 이러한 매개변수가 없거나 값이 다른 매개변수를 포함하는 경우 이러한 매개변수와 값을 포함하도록 파일을 업데이트하십시오.

3. /boot/cmdline.txt파일을 업데이트한 경우 Raspberry Pi를 재부팅하여 변경 사항을 적용하십시오.

```
sudo reboot
```

Linux 디바이스 설정 (기타)

Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT GreengrassCore 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 장치에 OpenJDK를 설치하는 방법을 보여줍니다.

   - Debian 기반 또는 Ubuntu 기반 배포판의 경우:

     ```
     sudo apt install default-jdk
     ```

   - Red Hat 기반 배포판의 경우:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - 대상 Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 아마존 리눅스 2023의 경우:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

   ```
   java -version
   ```

이 명령은 장치에서 실행되는 Java 버전을 인쇄합니다. 예를 들어 Debian 기반 배포의 경우 출력은 다음 샘플과 비슷할 수 있습니다.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (선택 사항) 기기에서 구성 요소를 실행하는 기본 시스템 사용자 및 그룹을 생성합니다. 설치 중에 설치 프로그램 인수를 사용하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 이 사용자 및 그룹을 만들도록 선택할 수도 있습니다. `--component-default-user` 자세한 설명은 인스톨러 인수 섹션을 참조하세요.

   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```

3. AWS IoT GreengrassCore 소프트웨어를 실행하는 사용자 (일반적으로`root`) 에게 모든 사용자 및 그룹과 `sudo` 함께 실행할 수 있는 권한이 있는지 확인하십시오.

   a. 다음 명령을 실행하여 `/etc/sudoers` 파일을 엽니다.

      ```
      sudo visudo
      ```

   b. 사용자의 권한이 다음 예와 같은지 확인합니다.

      ```
      root    ALL=(ALL:ALL) ALL
      ```

4. (선택 사항) 컨테이너화된 Lambda 함수를 실행하려면 cgroups v1을 활성화하고 메모리 및 디바이스 cgroups를 활성화하고 마운트해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없다면 이 단계를 건너뛰어도 됩니다.

   이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터를 사용하여 디바이스를 부팅하십시오.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   디바이스의 커널 파라미터 보기 및 설정에 대한 자세한 내용은 운영 체제 및 부트로더 설명서를 참조하십시오. 지침에 따라 커널 파라미터를 영구적으로 설정하십시오.

5. 의 요구 사항 목록에 나와 있는 대로 다른 모든 필수 종속 항목을 장치에 설치하십시오. 디바이스 요구 사항

Windows 디바이스 설정

Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT GreengrassCore 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
2. PATH 시스템 변수에서 Java를 사용할 수 있는지 확인하고 사용할 수 없는 경우 추가하십시오. 이 LocalSystem 계정은 AWS IoT Greengrass Core 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 다음을 따릅니다.

   a. Windows 키를 눌러 시작 메뉴를 엽니다.

   b. 시작 메뉴에서 시스템 옵션을 **environment variables** 검색하려면 입력합니다.

   c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.

   d. 환경 변수 선택... 환경 변수 창을 열려면

   e. 시스템 변수에서 경로를 선택한 다음 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.

   f. Java 설치 `bin` 폴더 경로가 있는지 확인합니다. 경로는 다음 예와 비슷할 수 있습니다.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Java 설치 `bin` 폴더가 경로에 없는 경우 [새로 만들기] 를 선택하여 추가한 다음 [확인] 을 선택합니다.

3. Windows 명령 프롬프트 (`cmd.exe`) 를 관리자 권한으로 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. *### ## ###* 바꾸십시오.

   ```
   net user /add ggc_user password
   ```

   > **Tip**
   >
   > Windows 구성에 따라 사용자 암호가 향후 날짜에 만료되도록 설정될 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 업데이트하십시오. 사용자 비밀번호가 만료되지 않도록 설정할 수도 있습니다.
   >
   > - 사용자 및 암호가 언제 만료되는지 확인하려면 다음 명령을 실행합니다.

```
net user ggc_user | findstr /C:expires
```

- 사용자 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 이 명령이 더 이상 사용되지 않는 Windows 10 이상을 사용하는 경우 wmic 다음 명령을 실행하세요. PowerShell

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Microsoft에서 PsExec유틸리티를 다운로드하여 장치에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. ### 이전에 설정한 사용자 암호로 바꾸십시오.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

가 PsExec License Agreement열리면 라이선스에 Accept동의하도록 선택하고 명령을 실행합니다.

> **Note**
>
> Windows 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 계정에 기본 사용자 정보를 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

3단계:AWS IoT Greengrass 핵심 소프트웨어 설치

이 섹션의 단계를 따라 Raspberry Pi를 로컬 개발에 사용할 수 있는AWS IoT Greengrass 코어 디바이스로 설정하세요. 이 섹션에서는 다음 작업을 수행하는 설치 프로그램을 다운로드하여 실행하여 장치의AWS IoT Greengrass Core 소프트웨어를 구성합니다.

- Greengrass 핵 구성 요소를 설치합니다. 핵은 필수 구성 요소이며 장치에서AWS IoT Greengrass Core 소프트웨어를 실행하기 위한 최소 요구 사항입니다. 자세한 내용은 Greengrass 핵 구성 요소를 참조하세요.
- 디바이스를 사물로AWS IoT 등록하고 디바이스를 연결할 수 있는 디지털 인증서를AWS 다운로드합니다. 자세한 정보는 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여을 참조하세요.
- 장치의 사물을AWS IoT 사물 그룹 (사물 그룹 또는 사물 집합인AWS IoT 사물 그룹) 에 추가합니다. 사물 그룹을 사용하면 Greengrass 코어 디바이스 플릿을 관리할 수 있습니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹에 배포하도록 선택할 수 있습니다. 자세한 내용은 AWS IoT Core개발자 안내서의 장치 관리를 참조하십시오.AWS IoT
- Greengrass 코어 디바이스가AWS 서비스와 상호 작용할 수 있도록 하는 IAM 역할을 생성합니다. 기본적으로 이 역할을 사용하면 디바이스가 Amazon Logs와 상호AWS IoT 작용하고 AmazonCloudWatch Logs로 로그를 전송할 수 있습니다. 자세한 정보는 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스을 참조하세요.
- 코어 장치에서 개발한 사용자 지정 구성 요소를 테스트하는 데 사용할 수 있는AWS IoT Greengrass 명령줄 인터페이스 (`greengrass-cli`) 를 설치합니다. 자세한 정보는 그린그래스 커맨드 라인 인터페이스을 참조하세요.

AWS IoT GreengrassCore 소프트웨어 설치 (콘솔)

1. AWS IoT Greengrass 콘솔에 로그인합니다.
2. Greengrass 시작하기에서 코어 디바이스 1개 설정을 선택합니다.
3. 1단계: Greengrass 코어 장치 등록에서 Core 장치 이름에 Greengrass 코어 장치의 AWS IoT 사물 이름을 입력합니다. 해당 항목이 존재하지 않는 경우 설치 프로그램이 생성합니다.
4. 2단계: 연속 배포를 적용할 사물 그룹에 추가에서 사물 그룹의 경우 코어 장치를 추가할 AWS IoT 사물 그룹을 선택합니다.

   - 새 그룹 이름 입력을 선택한 경우 사물 그룹 이름에 생성할 새 그룹의 이름을 입력합니다. 설치 프로그램이 새 그룹을 자동으로 생성합니다.
   - 기존 그룹 선택을 선택한 경우 사물 그룹 이름에서 사용하려는 기존 그룹을 선택합니다.
   - 그룹 없음을 선택하면 설치 프로그램이 사물 그룹에 코어 장치를 추가하지 않습니다.
5. 3단계: Greengrass Core 소프트웨어 설치에서 다음 단계를 완료하십시오.

   a. 코어 디바이스의 운영 체제를 리눅스 또는 윈도우 중에서 선택하세요.

   b. 디바이스에 AWS 자격 증명을 제공하면 설치 프로그램이 코어 디바이스용 AWS IoT 및 IAM 리소스를 프로비저닝할 수 있습니다. 보안을 강화하려면 프로비저닝에 필요한 최소 권한만

허용하는 IAM 역할에 대한 임시 자격 증명을 확보하는 것이 좋습니다. 자세한 설명은 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책 섹션을 참조하세요.

> **Note**
>
> 설치 프로그램은 자격 증명을 저장하거나 저장하지 않습니다.

디바이스에서 다음 중 하나를 수행하여 자격 증명을 검색하고 AWS IoT Greengrass Core 소프트웨어 설치 프로그램에서 사용할 수 있도록 하세요.

- (권장) 에서 임시 자격 증명을 사용하십시오. AWS IAM Identity Center

  i. IAM ID 센터의 액세스 키 ID, 보안 액세스 키 및 세션 토큰을 제공하십시오. 자세한 내용은 IAM Identity Center 사용 설명서의 임시 자격 증명 가져오기 및 새로 고침에서 수동 자격 증명 새로 고침을 참조하십시오.

  ii. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어에 자격 증명을 제공하십시오.

  Linux or Unix

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

  Windows Command Prompt (CMD)

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

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- IAM 역할의 임시 보안 자격 증명을 사용하십시오.

i. 수임한 IAM 역할의 액세스 키 ID, 보안 액세스 키, 세션 토큰을 제공하십시오. 이러한 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 임시 보안 자격 증명 요청을 참조하십시오.

ii. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어에 자격 증명을 제공하십시오.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- IAM 사용자의 장기 자격 증명을 사용하십시오.

i. IAM 사용자의 액세스 키 ID와 비밀 액세스 키를 제공하십시오. 프로비저닝용 IAM 사용자를 생성하고 나중에 삭제할 수 있습니다. 사용자에게 제공하는 IAM 정책은 을 참조하십시오. 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

ii. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어에 자격 증명을 제공하십시오.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
```

iii. (선택 사항) Greengrass 디바이스를 프로비저닝하기 위해 IAM 사용자를 생성한 경우 해당 사용자를 삭제하십시오.

iv. (선택 사항) 기존 IAM 사용자의 액세스 키 ID와 비밀 액세스 키를 사용한 경우 해당 사용자의 키를 업데이트하여 더 이상 유효하지 않도록 하십시오. 자세한 내용은 AWS Identity and Access Management사용 설명서의 액세스 키 업데이트를 참조하십시오.

c. 설치 프로그램 실행에서 다음 단계를 완료하십시오.

i. 설치 프로그램 다운로드에서 복사를 선택하고 복사한 명령을 코어 장치에서 실행합니다. 이 명령은 최신 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드하고 장치에서 압축을 풉니다.

ii. 설치 프로그램 실행에서 복사를 선택하고 복사한 명령을 코어 장치에서 실행합니다. 이 명령은 이전에 지정한 AWS IoT 사물 및 사물 그룹 이름을 사용하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 실행하고 코어 장치의 AWS 리소스를 설정합니다.

이 명령은 다음 작업도 수행합니다.

- AWS IoT Greengrass코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 장치에서는 Systemd init 시스템이 필요합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

- 코어 디바이스에서 사용자 지정 Greengrass 구성 요소를 개발할 수 있는 명령줄 도구인 AWS IoT Greengrass CLI 구성 요소를 배포합니다.
- `ggc_user`시스템 사용자를 사용하여 코어 디바이스에서 소프트웨어 구성 요소를 실행하도록 지정합니다. Linux 장치에서 이 명령은 또한 `ggc_group` 시스템 그룹을 사용하도록 지정하며, 설치 프로그램은 시스템 사용자 및 그룹을 자동으로 생성합니다.

이 명령을 실행하면 설치 프로그램이 성공했음을 나타내는 다음 메시지가 표시됩니다.

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> Linux 장치가 있고 systemd가 없는 경우 설치 프로그램은 소프트웨어를 시스템 서비스로 설정하지 않으며 nucleus를 시스템 서비스로 설정하는 데 대한 성공 메시지도 표시되지 않습니다.

AWS IoT Greengrass코어 소프트웨어 (CLI) 설치

AWS IoT GreengrassCore 소프트웨어 설치 및 구성하기

1. Greengrass 코어 디바이스에서 다음 명령을 실행하여 홈 디렉터리로 전환합니다.

   Linux or Unix

   ```
   cd ~
   ```

Windows Command Prompt (CMD)

```
cd %USERPROFILE%
```

PowerShell

```
cd ~
```

2. 코어 디바이스에서 AWS IoT Greengrass Core 소프트웨어를 라는 `greengrass-nucleus-latest.zip` 파일로 다운로드합니다.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

3. AWS IoT GreengrassCore 소프트웨어를 디바이스의 폴더에 압축을 풉니다. 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 시작합니다. 이 명령은 다음 작업을 수행합니다.

   - 코어 장치가 작동하는 데 필요한 AWS 리소스를 생성합니다.
   - AWS IoT Greengrass코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 장치에서는 Systemd init 시스템이 필요합니다.

   > **Important**
   >
   > Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

   - 코어 디바이스에서 사용자 지정 Greengrass 구성 요소를 개발할 수 있는 명령줄 도구인 AWS IoT Greengrass CLI 구성 요소를 배포합니다.
   - `ggc_user`시스템 사용자를 사용하여 코어 디바이스에서 소프트웨어 구성 요소를 실행하도록 지정합니다. Linux 장치에서 이 명령은 또한 `ggc_group` 시스템 그룹을 사용하도록 지정하며, 설치 프로그램은 시스템 사용자 및 그룹을 자동으로 생성합니다.

   다음과 같이 명령의 인수 값을 바꿉니다.

   a. *`/greengrass/v2`*또는 *`C:\greengrass\v2`*: AWS IoT Greengrass Core 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.

   b. *`GreengrassInstaller`*. AWS IoT GreengrassCore 소프트웨어 설치 프로그램의 압축을 푼 폴더의 경로입니다.

   c. *`##.`* 리소스를 찾거나 생성하는 위치. AWS 리전

d. *MyGreengrassCore*. Greengrass 코어 디바이스의 AWS IoT 이름입니다. 해당 사물이 존재하지 않는 경우 설치 프로그램이 생성합니다. 설치 프로그램은 인증서를 다운로드하여 사물로 인증합니다. AWS IoT 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.

> **Note**
>
> 사물 이름에는 콜론 () : 문자를 포함할 수 없습니다.

e. *MyGreengrassCoreGroup*. Greengrass 코어 디바이스의 AWS IoT 사물 그룹 이름입니다. 사물 그룹이 없는 경우 설치 프로그램은 사물 그룹을 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 배포가 활성화되어 있는 경우 코어 장치는 배포에서 지정하는 소프트웨어를 다운로드하고 실행합니다.

> **Note**
>
> 사물 그룹 이름에는 콜론 (:) 문자를 포함할 수 없습니다.

f. *ThingPolicy#####* V2IoT. Greengrass 코어 디바이스가 및 와 AWS IoT 통신할 수 있도록 허용하는 AWS IoT 정책의 이름입니다. AWS IoT Greengrass AWS IoT정책이 존재하지 않는 경우 설치 프로그램은 이 이름으로 허용 AWS IoT 정책을 생성합니다. 사용 사례에 맞게 이 정책의 권한을 제한할 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

g. *##### v2 TokenExchangeRole*. Greengrass 코어 디바이스가 임시 자격 증명을 받을 AWS 수 있도록 허용하는 IAM 역할의 이름입니다. 역할이 존재하지 않는 경우 설치 프로그램은 역할을 생성하고 이름이 지정된 정책을 생성하여 연결합니다. *GreengrassV2TokenExchangeRole*Access 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

h. *GreengrassCoreTokenExchangeRoleAlias*. Greengrass 코어 디바이스가 나중에 임시 자격 증명을 받을 수 있도록 허용하는 IAM 역할의 별칭입니다. 역할 별칭이 없는 경우 설치 프로그램은 역할 별칭을 생성하여 지정한 IAM 역할을 가리킵니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
```

```
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true \
  --deploy-dev-tools true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true ^
  --deploy-dev-tools true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true `
  --deploy-dev-tools true
```

> **Note**
>
> 메모리가 제한된 AWS IoT Greengrass 디바이스에서 실행하는 경우 AWS IoT Greengrass Core 소프트웨어가 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하려면 nucleus 구성 요소의 `jvmOptions` 구성 매개변수에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.

이 명령을 실행하면 설치 프로그램이 성공했음을 나타내는 다음 메시지가 표시됩니다.

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> Linux 장치가 있고 systemd가 없는 경우 설치 프로그램은 소프트웨어를 시스템 서비스로 설정하지 않으며 nucleus를 시스템 서비스로 설정하는 데 대한 성공 메시지도 표시되지 않습니다.

(선택 사항) 그린그래스 소프트웨어 실행 (리눅스)

소프트웨어를 시스템 서비스로 설치한 경우 설치 프로그램이 소프트웨어를 자동으로 실행합니다. 그렇지 않으면 소프트웨어를 실행해야 합니다. 설치 프로그램이 소프트웨어를 시스템 서비스로 설정했는지 확인하려면 설치 프로그램 출력에서 다음 줄을 확인하십시오.

```
Successfully set up Nucleus as a system service
```

이 메시지가 표시되지 않으면 다음과 같이 소프트웨어를 실행하십시오.

1. 다음 명령을 실행하여 소프트웨어를 실행합니다.

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

소프트웨어가 성공적으로 시작되면 다음 메시지를 인쇄합니다.

```
Launched Nucleus successfully.
```

2. AWS IoT GreengrassCore 소프트웨어를 계속 실행하려면 현재 명령 셸을 열어 두어야 합니다. SSH를 사용하여 코어 장치에 연결하는 경우 개발 컴퓨터에서 다음 명령을 실행하여 코어 장치에서 추가 명령을 실행하는 데 사용할 수 있는 두 번째 SSH 세션을 엽니다. *###* 이름을 로그인할 사용자 이름으로 바꾸고, 장치의 IP 주소로 *pi-ip-address*바꾸십시오.

```
ssh username@pi-ip-address
```

Greengrass 시스템 서비스와 상호 작용하는 방법에 대한 자세한 내용은 을 참조하십시오. Greengrass 핵을 시스템 서비스로 구성

디바이스에 그린그래스 CLI가 설치되어 있는지 확인합니다.

Greengrass CLI는 배포하는 데 최대 1분이 걸릴 수 있습니다. 다음 명령을 실행하여 배포 상태를 확인합니다. 코어 디바이스의 *MyGreengrassCore*이름으로 바꾸십시오.

```
aws greengrassv2 list-effective-deployments --core-device-thing-name MyGreengrassCore
```

는 코어 장치에 대한 배포 상태를 `coreDeviceExecutionStatus` 나타냅니다. 상태가 `SUCCEEDED` 표시되면 다음 명령을 실행하여 Greengrass CLI가 설치되고 실행되는지 확인합니다. 루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli help
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli help
```

이 명령은 Greengrass CLI에 대한 도움말 정보를 출력합니다. 찾을 수 `greengrass-cli` 없는 경우 배포 시 Greengrass CLI를 설치하지 못한 것일 수 있습니다. 자세한 설명은 문제 해결 AWS IoT Greengrass V2 섹션을 참조하세요.

다음 명령을 실행하여 AWS IoT Greengrass CLI를 디바이스에 수동으로 배포할 수도 있습니다.

- *### AWS ## #### ####* 바꾸십시오. AWS CLI 디바이스에서 구성할 때 사용한 것과 동일한 AWS 리전 것을 사용해야 합니다.
- *## ID# ID#* 바꾸십시오. AWS 계정
- 코어 *MyGreengrassCore*디바이스의 이름으로 바꾸십시오.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.2"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.2\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.2\"}}'
```

> **Tip**
>
> PATH환경 변수에 */greengrass/v2*/bin (Linux) 또는 *C:\greengrass\v2*\bin (Windows) 를 추가하여 절대 경로 `greengrass-cli` 없이 실행할 수 있습니다.

AWS IoT GreengrassCore 소프트웨어 및 로컬 개발 도구가 장치에서 실행됩니다. 다음으로 기기에서 Hello WorldAWS IoT Greengrass 구성 요소를 개발할 수 있습니다.

4단계: 기기의 구성 요소 개발 및 테스트

구성 요소는 AWS IoT Greengrass 코어 기기에서 실행되는 소프트웨어 모듈입니다. 구성 요소를 사용하면 복잡한 애플리케이션을 하나의 Greengrass 코어 장치에서 다른 Greengrass 코어 장치로 재사용할 수 있는 개별 구성 요소로 만들고 관리할 수 있습니다. 모든 구성 요소는 레시피와 아티팩트로 구성됩니다.

- 레시피

  모든 구성 요소에는 메타데이터를 정의하는 레시피 파일이 포함되어 있습니다. 레시피는 또한 구성 요소의 구성 매개 변수, 구성 요소 종속성, 수명 주기 및 플랫폼 호환성을 지정합니다. 구성 요소 수명 주기는 구성 요소를 설치, 실행 및 종료하는 명령을 정의합니다. 자세한 설명은 AWS IoT Greengrass컴포넌트 레시피 참조 섹션을 참조하세요.

  레시피는 JSON 또는 YAML 형식으로 정의할 수 있습니다.
- 아티팩트

  구성 요소에는 구성 요소 바이너리인 아티팩트가 여러 개 있을 수 있습니다. 아티팩트에는 스크립트, 컴파일된 코드, 정적 리소스 및 구성 요소가 사용하는 기타 모든 파일이 포함될 수 있습니다. 구성 요소는 구성 요소 종속성의 아티팩트를 사용할 수도 있습니다.

를 사용하면 AWS IoT Greengrass Greengrass CLI를 사용하여 클라우드와 상호 작용하지 않고도 Greengrass 코어 디바이스에서 로컬로 구성 요소를 개발하고 테스트할 수 있습니다. AWS 로컬 구성 요소를 완성하면 구성 요소 레시피와 아티팩트를 사용하여 AWS 클라우드의 AWS IoT Greengrass 서비스에 해당 구성 요소를 만든 다음 모든 Greengrass 코어 장치에 배포할 수 있습니다. 구성 요소에 대한 자세한 내용은 AWS IoT Greengrass구성 요소 개발를 참조하세요.

이 섹션에서는 코어 디바이스에서 로컬로 기본 Hello World 구성 요소를 만들고 실행하는 방법을 알아봅니다.

기기에서 Hello World 구성 요소를 개발하려면

1. 레시피와 아티팩트를 위한 하위 폴더가 있는 컴포넌트 폴더를 만드세요. Greengrass 코어 디바이스에서 다음 명령을 실행하여 이러한 폴더를 생성하고 구성 요소 폴더로 변경합니다. *~/greengrassv2* 또는 *%USERPROFILE%\ greengrassv2#* 로컬 개발에 사용할 폴더 경로로 바꾸십시오.

Linux or Unix

```
mkdir -p ~/greengrassv2/{recipes,artifacts}
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. 텍스트 편집기를 사용하여 구성요소의 메타데이터, 매개변수, 종속성, 수명 주기 및 플랫폼 기능을 정의하는 레시피 파일을 만드십시오. 레시피 파일 이름에 구성 요소 버전을 포함하면 어떤 레시피가 어떤 구성 요소 버전을 반영하는지 식별할 수 있습니다. 레시피에 YAML 또는 JSON 형식을 선택할 수 있습니다.

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   JSON

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   nano recipes/com.example.HelloWorld-1.0.0.yaml
   ```

   > **Note**
   >
   > AWS IoT Greengrass 구성 요소에 시맨틱 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 버전은 구성 `1.0.0` 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

3. 다음 레시피를 파일에 붙여넣습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
```

```
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

이 레시피의 `ComponentConfiguration` 섹션에서는 기본값이 `Message` 인 매개변수를 정의합니다. `world` 이 `Manifests` 섹션에서는 플랫폼에 대한 라이프사이클 지침 및 아티팩트 집합인 매니페스트를 정의합니다. 예를 들어 여러 매니페스트를 정의하여 다양한 플랫폼에 대해 서로 다른 설치 지침을 지정할 수 있습니다. 매니페스트에서 `Lifecycle` 섹션은 Greengrass 코어 디바이스에 `Message` 매개변수 값을 인수로 사용하여 Hello World 스크립트를 실행하도록 지시합니다.

4. 다음 명령을 실행하여 구성 요소 아티팩트를 위한 폴더를 생성합니다.

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir artifacts\com.example.HelloWorld\1.0.0
   ```

   PowerShell

   ```
   mkdir artifacts\com.example.HelloWorld\1.0.0
   ```

> **Important**
>
> 아티팩트 폴더 경로에는 다음 형식을 사용해야 합니다. 레시피에 지정한 구성 요소 이름과 버전을 포함하십시오.
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. 텍스트 편집기를 사용하여 Hello World 구성 요소에 대한 Python 스크립트 아티팩트 파일을 만드십시오.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   다음 Python 스크립트를 복사하여 파일에 붙여넣습니다.

   ```
   import sys

   message = "Hello, %s!" % sys.argv[1]

   # Print the message to stdout, which Greengrass saves in a log file.
   print(message)
   ```

6. 로컬 AWS IoT Greengrass CLI를 사용하여 Greengrass 코어 디바이스의 구성 요소를 관리합니다.

   다음 명령을 실행하여 구성 요소를 코어에 배포합니다. AWS IoT Greengrass */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass V2 루트 폴더로 바꾸고 *~/greengrassv2 ## %USERPROFILE%\ greengrassv2# ## ## ##* 폴더로 바꾸십시오.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

이 명령은 에서 `recipes` 레시피를 사용하고 Python 스크립트를 사용하는 구성 요소를 추가합니다`artifacts`. `--merge`옵션은 지정한 구성 요소 및 버전을 추가하거나 업데이트합니다.

7. AWS IoT Greengrass Core 소프트웨어는 구성 요소 프로세스의 stdout을 폴더의 `logs` 로그 파일에 저장합니다. 다음 명령을 실행하여 Hello World 구성 요소가 실행되고 메시지를 인쇄하는지 확인합니다.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.HelloWorld.log
   ```

   이 `type` 명령은 파일 내용을 터미널에 씁니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
   ```

   다음 예와 비슷한 메시지가 표시될 것입니다.

```
Hello, world!
```

> **Note**
>
> 파일이 없는 경우 로컬 배포가 아직 완료되지 않았을 수 있습니다. 15초 이내에 파일이 존재하지 않으면 배포가 실패했을 수 있습니다. 예를 들어 레시피가 유효하지 않은 경우 이런 일이 발생할 수 있습니다. 다음 명령을 실행하여 AWS IoT Greengrass 코어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\logs\greengrass.log
> ```
>
> type명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
> ```

8. 로컬 구성 요소를 수정하여 코드를 반복하고 테스트하십시오. 텍스트 `hello_world.py` 편집기에서 열고 4행에 다음 코드를 추가하여 AWS IoT Greengrass 코어가 기록하는 메시지를 편집합니다.

```
message += " Greetings from your first Greengrass component."
```

이제 `hello_world.py` 스크립트에 다음과 같은 내용이 포함되어야 합니다.

```
import sys

message = "Hello, %s!" % sys.argv[1]
message += " Greetings from your first Greengrass component."
```

```
# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

9. 다음 명령을 실행하여 구성 요소를 변경 내용으로 업데이트합니다.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --recipeDir ~/greengrassv2/recipes `
     --artifactDir ~/greengrassv2/artifacts `
     --merge "com.example.HelloWorld=1.0.0"
   ```

   이 명령은 최신 Hello World 아티팩트로 `com.example.HelloWorld` 구성 요소를 업데이트합니다.

10. 다음 명령을 실행하여 구성 요소를 다시 시작합니다. 구성 요소를 다시 시작하면 코어 기기는 최신 변경 내용을 사용합니다.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli component restart \
      --names "com.example.HelloWorld"
    ```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component restart ^
  --names "com.example.HelloWorld"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component restart `
  --names "com.example.HelloWorld"
```

11. 로그를 다시 확인하여 Hello World 구성 요소가 새 메시지를 인쇄하는지 확인하십시오.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\logs\com.example.HelloWorld.log
    ```

    이 `type` 명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.

    PowerShell

    ```
    gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
    ```

    다음 예와 비슷한 메시지가 표시될 것입니다.

    ```
    Hello, world! Greetings from your first Greengrass component.
    ```

12. 구성 요소의 구성 매개변수를 업데이트하여 다양한 구성을 테스트할 수 있습니다. 구성 요소를 배포할 때 구성 업데이트를 지정할 수 있습니다. 구성 업데이트는 코어 장치에서 구성 요소의 구성을 수정하는 방법을 정의합니다. 기본값으로 재설정할 구성 값과 코어 장치에 병합할 새 구성 값을 지정할 수 있습니다. 자세한 설명은 구성 요소 구성 업데이트 섹션을 참조하세요.

    다음을 따릅니다.

a. 텍스트 편집기를 사용하여 구성 업데이트를 포함하는 `hello-world-config-update.json` 파일을 생성하십시오.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano hello-world-config-update.json
   ```

b. 다음 JSON 객체를 복사하여 파일에 붙여넣습니다. 이 JSON 객체는 값을 `Message` 매개변수에 `friend` 병합하여 값을 업데이트하는 구성 업데이트를 정의합니다. 이 구성 업데이트에서는 재설정할 값을 지정하지 않습니다. 병합 업데이트는 기존 값을 대체하므로 `Message` 파라미터를 재설정할 필요가 없습니다.

   ```
   {
     "com.example.HelloWorld": {
       "MERGE": {
         "Message": "friend"
       }
     }
   }
   ```

c. 다음 명령을 실행하여 Hello World 구성 요소에 구성 업데이트를 배포합니다.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --merge "com.example.HelloWorld=1.0.0" \
     --update-config hello-world-config-update.json
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --merge "com.example.HelloWorld=1.0.0" ^
     --update-config hello-world-config-update.json
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --merge "com.example.HelloWorld=1.0.0" `
   ```

```
  --update-config hello-world-config-update.json
```

d. 로그를 다시 확인하여 Hello World 구성 요소가 새 메시지를 출력하는지 확인합니다.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

이 type 명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

다음 예와 비슷한 메시지가 표시될 것입니다.

```
Hello, friend! Greetings from your first Greengrass component.
```

13. 구성 요소 테스트를 마친 후 핵심 장치에서 해당 구성 요소를 제거합니다. 다음 명령을 실행합니다.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

> **Important**
>
> 구성 요소를 업로드한 후 코어 장치에 다시 배포하려면 이 단계가 필요합니다 AWS IoT Greengrass. 그렇지 않으면 로컬 배포에서 구성 요소의 다른 버전을 지정하기 때문에 버전 호환성 오류가 발생하여 배포가 실패합니다.

다음 명령어를 실행하고 `com.example.HelloWorld` 구성 요소가 장치의 구성 요소 목록에 나타나지 않는지 확인합니다.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component list
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component list
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component list
```

Hello World 구성 요소가 완성되었으므로 이제 AWS IoT Greengrass 클라우드 서비스에 업로드할 수 있습니다. 그런 다음 Greengrass 코어 디바이스에 컴포넌트를 배포할 수 있습니다.

5단계: AWS IoT Greengrass 서비스에서 구성 요소 생성

코어 디바이스에서 구성 요소 개발을 마치면 의 AWS IoT Greengrass 서비스에 업로드할 수 AWS 클라우드 있습니다. AWS IoT Greengrass콘솔에서 구성 요소를 직접 만들 수도 있습니다. AWS IoT Greengrass개별 장치 또는 장치 집합에 배포할 수 있도록 구성 요소를 호스팅하는 구성 요소 관리 서

비스를 제공합니다. AWS IoT Greengrass서비스에 구성 요소를 업로드하려면 다음 단계를 완료하십시오.

- 구성 요소 아티팩트를 S3 버킷에 업로드합니다.
- 각 아티팩트의 아마존 심플 스토리지 서비스 (Amazon S3) URI를 구성 요소 레시피에 추가합니다.
- 구성 요소 AWS IoT Greengrass 레시피에서 구성 요소를 생성합니다.

이 섹션에서는 Greengrass 코어 디바이스에서 다음 단계를 완료하여 Hello World 구성 요소를 서비스에 업로드합니다AWS IoT Greengrass.

AWS IoT Greengrass(콘솔) 에서 구성 요소를 생성합니다.

1. AWS계정의 S3 버킷을 사용하여 AWS IoT Greengrass 구성 요소 아티팩트를 호스팅하십시오. 구성 요소를 코어 디바이스에 배포하면 디바이스가 버킷에서 구성 요소의 아티팩트를 다운로드합니다.

   기존 S3 버킷을 사용하거나 새 버킷을 생성할 수 있습니다.

   a. Amazon S3 콘솔의 버킷에서 버킷 생성을 선택합니다.

   b. 버킷 이름에 고유한 버킷 이름을 입력합니다. 예를 들어 **greengrass-component-artifacts-*region*-*123456789012***를 사용할 수 있습니다. *123456789012#* 이 자습서에서 사용하는 AWS 계정 ID 및 *####* 바꾸십시오. AWS 리전

   c. AWS지역의 경우 이 자습서에 사용할 AWS 지역을 선택하십시오.

   d. 버킷 생성을 선택합니다.

   e. 버킷에서 생성한 버킷을 선택하고 버킷의 **artifacts/com.example.HelloWorld/1.0.0** 폴더에 **hello_world.py** 스크립트를 업로드합니다. S3 버킷에 객체를 업로드하는 방법에 대한 자세한 내용은 Amazon 심플 스토리지 서비스 사용 설명서의 객체 업로드를 참조하십시오.

   f. S3 버킷에 있는 hello_world.py 객체의 S3 URI를 복사합니다. 이 URI는 다음 예제와 비슷해야 합니다. *DOC-EXAMPLE-BUCKET# S3 ###* 이름으로 바꾸십시오.

   ```
   s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

2. 코어 디바이스가 S3 버킷의 구성 요소 아티팩트에 액세스할 수 있도록 허용합니다.

   각 코어 디바이스에는 클라우드와 상호 AWS IoT 작용하고 클라우드로 로그를 전송할 수 있는 코어 디바이스 IAM 역할이 있습니다. AWS 이 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스

를 허용하지 않으므로 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색하도록 허용하는 정책을 생성하고 연결해야 합니다.

디바이스 역할이 이미 S3 버킷에 대한 액세스를 허용하고 있다면 이 단계를 건너뛰어도 됩니다. 그렇지 않으면 다음과 같이 액세스를 허용하는 IAM 정책을 생성하여 역할에 연결하십시오.

a. IAM 콘솔 탐색 메뉴에서 [Policies] 를 선택한 다음 [Create policy] 를 선택합니다.

b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다. *DOC-EXAMPLE-BUCKET#* 다운로드할 코어 디바이스의 구성 요소 아티팩트가 포함된 S3 버킷의 이름으로 바꾸십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

c. 다음을 선택합니다.

d. 정책 세부 정보 섹션의 이름에 을 입력합니다.
MyGreengrassV2ComponentArtifactPolicy

e. 정책 생성을 선택합니다.

f. IAM 콘솔 탐색 메뉴에서 [Role] 을 선택한 다음 코어 디바이스의 역할 이름을 선택합니다. AWS IoT GreengrassCore 소프트웨어를 설치할 때 이 역할 이름을 지정했습니다. 이름을 지정하지 않은 경우 기본값은 입니다`GreengrassV2TokenExchangeRole`.

g. 권한에서 권한 추가를 선택한 다음 정책 연결을 선택합니다.

h. 권한 추가 페이지에서 생성한 `MyGreengrassV2ComponentArtifactPolicy` 정책 옆의 확인란을 선택한 다음 권한 추가를 선택합니다.

3. 구성 요소 레시피를 사용하여 AWS IoT Greengrass콘솔에서 구성 요소를 만들 수 있습니다.

a. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택한 다음 구성 요소 생성을 선택합니다.

b. 구성 요소 정보에서 레시피를 JSON으로 입력을 선택합니다. 플레이스홀더 레시피는 다음 예와 비슷해야 합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
```

```
      ]
    }
  ]
}
```

c. 각 `Artifacts` 섹션의 플레이스홀더 URI를 객체의 S3 URI로 바꾸십시오. `hello_world.py`

d. 구성 요소 생성을 선택합니다.

e. com.example에서 HelloWorld구성 요소 페이지에서 구성 요소 상태가 배포 가능인지 확인합니다.

() 에서 AWS IoT Greengrass 구성 요소를 생성합니다. AWS CLI

헬로 월드 컴포넌트를 업로드하려면

1. 내 S3 버킷을 사용하여 AWS IoT Greengrass 구성 요소 아티팩트를 호스팅하십시오AWS 계정. 구성 요소를 코어 디바이스에 배포하면 디바이스가 버킷에서 구성 요소의 아티팩트를 다운로드합니다.

   기존 S3 버킷을 사용하거나 다음 명령을 실행하여 버킷을 생성할 수 있습니다. 이 명령은 AWS 계정 ID를 사용하여 버킷을 생성하고 고유한 버킷 이름을 형성합니다. AWS 리전 *123456789012#* 이 자습서에서 사용하는 AWS 계정 ID와 *####* 바꾸십시오. AWS 리전

   ```
   aws s3 mb s3://greengrass-component-artifacts-123456789012-region
   ```

   이 명령은 요청이 성공하면 다음 정보를 출력합니다.

   ```
   make_bucket: greengrass-component-artifacts-123456789012-region
   ```

2. 코어 디바이스가 S3 버킷의 구성 요소 아티팩트에 액세스할 수 있도록 허용합니다.

   각 코어 디바이스에는 코어 디바이스와 상호 AWS IoT 작용하고 로그를 전송할 수 있는 코어 디바이스 IAM 역할이 있습니다. AWS 클라우드 이 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않으므로 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색하도록 허용하는 정책을 생성하고 연결해야 합니다.

   코어 디바이스의 역할이 이미 S3 버킷에 대한 액세스를 허용한 경우에는 이 단계를 건너뛰어도 됩니다. 그렇지 않으면 다음과 같이 액세스를 허용하는 IAM 정책을 생성하여 역할에 연결하십시오.

a. 라는 component-artifact-policy.json 파일을 생성하고 다음 JSON을 파일에 복사합니다. 이 정책은 S3 버킷의 모든 파일에 대한 액세스를 허용합니다. *DOC-EXAMPLE-BUCKET# S3 ###* 이름으로 바꾸십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

b. 다음 명령을 실행하여 의 정책 문서에서 정책을 생성합니다. component-artifact-policy.json

Linux or Unix

```
aws iam create-policy \\
  --policy-name MyGreengrassV2ComponentArtifactPolicy \\
  --policy-document file://component-artifact-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name MyGreengrassV2ComponentArtifactPolicy ^
  --policy-document file://component-artifact-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name MyGreengrassV2ComponentArtifactPolicy `
  --policy-document file://component-artifact-policy.json
```

출력의 정책 메타데이터에서 Amazon 리소스 이름 (ARN) 정책을 복사합니다. 다음 단계에서 이 ARN을 사용하여 이 정책을 핵심 장치 역할에 연결합니다.

c. 다음 명령을 실행하여 정책을 핵심 장치 역할에 연결합니다. *GreenGrassV2#* 코어 디바이스의 역할 이름으로 TokenExchangeRole 바꾸십시오. Core 소프트웨어를 설치할 때 이 역할 이름을 지정했습니다AWS IoT Greengrass. 정책 ARN을 이전 단계의 ARN으로 교체합니다.

Linux or Unix

```
aws iam attach-role-policy \\
  --role-name GreengrassV2TokenExchangeRole \\
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

명령 출력이 없으면 성공한 것입니다. 이제 코어 디바이스는 이 S3 버킷에 업로드한 아티팩트에 액세스할 수 있습니다.

3. Hello World Python 스크립트 아티팩트를 S3 버킷에 업로드합니다.

다음 명령을 실행하여 스크립트가 AWS IoT Greengrass 코어에 있는 버킷의 동일한 경로에 스크립트를 업로드합니다. *DOC-EXAMPLE-BUCKET# S3 ###* 이름으로 바꾸십시오.

Linux or Unix

```
aws s3 cp \
```

```
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py \
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Windows Command Prompt (CMD)

```
aws s3 cp ^
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py ^
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

PowerShell

```
aws s3 cp `
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py `
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

이 명령은 요청이 성공하면 로 `upload:` 시작하는 줄을 출력합니다.

4. 아티팩트의 Amazon S3 URI를 구성 요소 레시피에 추가합니다.

   Amazon S3 URI는 버킷 이름과 버킷 내 아티팩트 객체 경로로 구성됩니다. 스크립트 아티팩트의 Amazon S3 URI는 이전 단계에서 아티팩트를 업로드한 URI입니다. 이 URI는 다음 예제와 비슷해야 합니다. *DOC-EXAMPLE-BUCKET# S3 ###* 이름으로 바꾸십시오.

   ```
   s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   레시피에 아티팩트를 추가하려면 Amazon S3 URI의 구조가 `Artifacts` 포함된 목록을 추가하십시오.

   JSON

   ```
   "Artifacts": [
     {
       "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
   hello_world.py"
     }
   ]
   ```

   텍스트 편집기에서 레시피 파일을 엽니다.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

레시피에 아티팩트를 추가합니다. 레시피 파일은 다음 예제와 비슷해야 합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
```

```
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
Artifacts:
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

텍스트 편집기에서 레시피 파일을 엽니다.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

레시피에 아티팩트를 추가합니다. 레시피 파일은 다음 예제와 비슷해야 합니다.

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

5. AWS IoT Greengrass레시피에서 구성 요소 리소스를 생성합니다. 다음 명령을 실행하여 레시피에서 구성 요소를 생성합니다. 레시피는 바이너리 파일로 제공됩니다.

   JSON

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.yaml
   ```

   요청이 성공하면 응답은 다음 예제와 비슷해 보입니다.

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
     "componentName": "com.example.HelloWorld",
     "componentVersion": "1.0.0",
     "creationTimestamp": "Mon Nov 30 09:04:05 UTC 2020",
     "status": {
       "componentState": "REQUESTED",
       "message": "NONE",
       "errors": {}
     }
   }
   ```

   arn출력에서 를 복사하여 다음 단계에서 구성 요소의 상태를 확인합니다.

> **Note**
>
> AWS IoT Greengrass콘솔의 구성 요소 페이지에서도 Hello World 구성 요소를 볼 수 있습니다.

6. 구성 요소가 생성되고 배포할 준비가 되었는지 확인합니다. 구성 요소를 만들 때 구성 요소의 상태는 입니다REQUESTED. 그런 다음 구성 요소를 배포할 수 있는지 AWS IoT Greengrass 확인합니다. 다음 명령을 실행하여 구성 요소 상태를 쿼리하고 구성 요소가 배포 가능한지 확인할 수 있습니다. 를 이전 단계의 arn ARN으로 교체합니다.

```
aws greengrassv2 describe-component --arn
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0"
```

구성 요소가 검증되면 응답은 구성 요소 상태가 유효하다는 것을 나타냅니다. DEPLOYABLE

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
  "componentName": "com.example.HelloWorld",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-11-30T18:04:05.823Z",
  "publisher": "Amazon",
  "description": "My first Greengrass component.",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "os": "linux",
      "architecture": "all"
    }
  ]
}
```

이제 Hello World 구성 요소를 에서 사용할 수 있습니다. AWS IoT Greengrass 이 Greengrass 코어 디바이스 또는 다른 코어 디바이스에 다시 배포할 수 있습니다.

6단계: 구성 요소 배포

를 사용하여 AWS IoT Greengrass 개별 장치 또는 장치 그룹에 구성 요소를 배포할 수 있습니다. 구성 요소를 배포할 때 각 대상 장치에 해당 구성 요소의 소프트웨어를 AWS IoT Greengrass 설치하고 실행합니다. 배포할 구성 요소와 각 구성 요소에 배포할 구성 업데이트를 지정합니다. 또한 배포 대상 장치에 배포가 배포되는 방식을 제어할 수 있습니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

이 섹션에서는 Hello World 구성 요소를 Greengrass 코어 장치에 다시 배포합니다.

구성 요소 배포 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 내 구성 요소 탭에서 을 선택합니다 com.example.HelloWorld.
3. com.example.HelloWorld 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 새 배포 생성을 선택한 후 다음을 선택합니다.
5. 대상 지정 페이지에서 다음 작업을 수행합니다.

   a. 이름(Name) 상자에 **`Deployment for MyGreengrassCore`**를 입력합니다.

   b. 배포 대상으로 코어 디바이스를 선택하고 코어 디바이스의 AWS IoT 사물 이름을 선택합니다. 이 튜토리얼의 기본값은 입니다 *`MyGreengrassCore`*.

   c. 다음을 선택합니다.
6. 구성 요소 선택 페이지의 내 구성 요소에서 com.example.HelloWorld구성 요소가 선택되어 있는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 구성 요소를 선택하고 com.example.HelloWorld다음을 수행합니다.

   a. 구성 요소 구성을 선택합니다.

   b. 구성 업데이트 아래에 있는 병합할 구성에 다음 구성을 입력합니다.

   ```
   {
     "Message": "universe"
   }
   ```

   이 구성 업데이트는 이 배포에서 기기의 Hello World `Message` 매개변수를 로 `universe` 설정합니다.

   c. 확인을 선택합니다.

d. 다음을 선택합니다.

8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 배포를 선택합니다.
10. 배포가 성공적으로 완료되었는지 확인하세요. 배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. Hello World 로그를 확인하여 변경 사항을 확인하세요. Greengrass 코어 디바이스에서 다음 명령을 실행합니다.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
    ```

    PowerShell

    ```
    gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
    ```

    다음 예와 비슷한 메시지가 표시되어야 합니다.

    ```
    Hello, universe! Greetings from your first Greengrass component.
    ```

    > **Note**
    >
    > 로그 메시지가 변경되지 않으면 배포가 실패했거나 코어 기기에 도달하지 못한 것입니다. 이는 코어 디바이스가 인터넷에 연결되어 있지 않거나 S3 버킷에서 아티팩트를 검색할 권한이 없는 경우 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.
    >
    > Linux or Unix
    >
    > ```
    > sudo tail -f /greengrass/v2/logs/greengrass.log
    > ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\greengrass.log
```

이 type 명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.

PowerShell

```
gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
```

자세한 설명은 문제 해결 AWS IoT Greengrass V2 섹션을 참조하세요.

구성 요소 배포 (AWS CLI)

헬로 월드 컴포넌트를 배포하려면

1. 개발 컴퓨터에서 이라는 hello-world-deployment.json 파일을 만들고 다음 JSON을 파일에 복사합니다. 이 파일은 배포할 구성 요소 및 구성을 정의합니다.

```
{
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"Message\":\"universe\"}"
      }
    }
  }
}
```

이 구성 파일은 이전 절차에서 개발하고 게시한 Hello World 구성 1.0.0 요소의 버전을 배포하도록 지정합니다. 는 구성 요소 구성을 JSON으로 인코딩된 문자열로 병합하도록 configurationUpdate 지정합니다. 이 구성 업데이트는 이 배포에서 기기의 Hello World Message 매개변수를 로 universe 설정합니다.

2. 다음 명령을 실행하여 Greengrass 코어 디바이스에 구성 요소를 배포합니다. 개별 장치인 사물 또는 장치 그룹인 사물 그룹에 배포할 수 있습니다. 코어 디바이스의 AWS IoT 사물 *MyGreengrassCore*이름으로 바꾸십시오.

   Linux or Unix

   ```
   aws greengrassv2 create-deployment \
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
     --cli-input-json file://hello-world-deployment.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 create-deployment ^
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
     --cli-input-json file://hello-world-deployment.json
   ```

   PowerShell

   ```
   aws greengrassv2 create-deployment `
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
     --cli-input-json file://hello-world-deployment.json
   ```

   이 명령은 다음 예와 비슷한 응답을 출력합니다.

   ```
   {
     "deploymentId": "deb69c37-314a-4369-a6a1-3dff9fce73a9",
     "iotJobId": "b5d92151-6348-4941-8603-bdbfb3e02b75",
     "iotJobArn": "arn:aws:iot:region:account-id:job/b5d92151-6348-4941-8603-
   bdbfb3e02b75"
   }
   ```

3. 배포가 성공적으로 완료되었는지 확인합니다. 배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. Hello World 로그를 확인하여 변경 사항을 확인하세요. Greengrass 코어 디바이스에서 다음 명령을 실행합니다.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
```

다음 예와 비슷한 메시지가 표시되어야 합니다.

```
Hello, universe! Greetings from your first Greengrass component.
```

> **Note**
>
> 로그 메시지가 변경되지 않으면 배포가 실패했거나 코어 기기에 도달하지 못한 것입니다. 이는 코어 디바이스가 인터넷에 연결되어 있지 않거나 S3 버킷에서 아티팩트를 검색할 권한이 없는 경우 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> 이 type 명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```

자세한 내용은 문제 해결 AWS IoT Greengrass V2을(를) 참조하세요.

다음 단계

이 튜토리얼을 완료했습니다. AWS IoT Greengrass코어 소프트웨어 및 Hello World 구성 요소가 기기에서 실행됩니다. 또한 Hello World 구성 요소를AWS IoT Greengrass 클라우드 서비스에서 사용하여 다른 장치에 배포할 수 있습니다. 이 자습서에서 사용되는 주제에 대한 자세한 내용은 다음을 참조하십시오.

- AWS IoT Greengrass 구성 요소 생성
- 코어 디바이스에 배포할 구성 요소를 게시하세요.
- 디바이스에 AWS IoT Greengrass 구성 요소 배포

AWS IoT Greengrass코어 디바이스 설정

이 섹션의 작업을 완료하여 AWS IoT Greengrass Core 소프트웨어를 설치, 구성 및 실행하십시오.

> **Note**
>
> 이 섹션에서는 AWS IoT Greengrass Core 소프트웨어의 고급 설치 및 구성에 대해 설명합니다. 를 처음 사용하는 경우 먼저 시작 자습서를 완료하여 코어 장치를 설정하고 의 AWS IoT Greengrass 기능을 살펴보는 것이 좋습니다. AWS IoT Greengrass V2

지원되는 플랫폼 및 요구 사항

시작하기 전에 AWS IoT Greengrass Core 소프트웨어를 설치하고 실행하기 위한 다음 요구 사항을 충족하는지 확인하십시오.

> **Tip**
>
> AWS파트너 장치 AWS IoT Greengrass V2 카탈로그에서 적합한 장치를 검색할 수 있습니다.

주제

- 지원하는 플랫폼
- 디바이스 요구 사항
- Lambda 함수 요구 사항

지원하는 플랫폼

AWS IoT Greengrass다음 플랫폼을 실행하는 장치를 공식적으로 지원합니다. 이 목록에 포함되지 않은 플랫폼의 기기는 작동할 수 있지만 AWS IoT Greengrass 테스트는 지정된 플랫폼에서만 가능합니다.

Linux

아키텍처:

- Armv7l
- ARMv8(AArch64)
- x86_64

Windows

아키텍처:

- x86_64

버전:

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

> **Note**
>
> 일부 AWS IoT Greengrass 기능은 현재 Windows 장치에서 지원되지 않습니다. 자세한 내용은 운영 체제별 Greengrass 기능 호환성 및 Windows 디바이스의 기능 고려 사항 섹션을 참조하세요.

Linux 플랫폼은 Docker AWS IoT Greengrass V2 컨테이너에서도 실행할 수 있습니다. 자세한 설명은 Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어 실행 섹션을 참조하세요.

사용자 지정 Linux 기반 운영 체제를 구축하려면 프로젝트의 BitBake 레시피를 사용할 AWS IoT Greengrass V2 수 있습니다. `meta-aws` 이 `meta-aws` 프로젝트는 Yocto Project 빌드 프레임워크로 OpenEmbedded구축된 임베디드 Linux 시스템에서 AWS 엣지 소프트웨어 기능을 구축하는 데 사용할 수 있는 레시피를 제공합니다. Yocto Project는 하드웨어 아키텍처에 관계없이 임베디드 애플리케이션을 위한 사용자 지정 Linux 기반 시스템을 구축할 수 있도록 지원하는 오픈 소스 협업 프로젝트입니다. 장치에 Core AWS IoT Greengrass V2 소프트웨어를 설치, 구성 및 자동으로 실행하는 BitBake 레시피입니다. AWS IoT Greengrass

디바이스 요구 사항

AWS IoT GreengrassCore 소프트웨어 v2.x를 설치하고 실행하려면 장치가 다음 요구 사항을 충족해야 합니다.

> **Note**
>
> AWS IoT Device Testerfor를 AWS IoT Greengrass 사용하여 장치가 Core 소프트웨어를 실행하고 AWS IoT Greengrass Core 소프트웨어와 통신할 수 있는지 확인할 수 있습니다. AWS 클라우드 자세한 설명은 AWS IoT Device TesterAWS IoT GreengrassV2에 사용하기 섹션을 참조하세요.

Linux

- AWS 리전지원하는 제품의 사용AWS IoT Greengrass V2. 지원되는 리전 목록은 AWS 일반 참조에서 AWS IoT Greengrass V2 엔드포인트 및 할당량을 참조하십시오.
- AWS IoT GreengrassCore 소프트웨어에 사용할 수 있는 최소 256MB의 디스크 공간. 이 요구 사항에는 코어 장치에 배포된 구성 요소는 포함되지 않습니다.
- AWS IoT Greengrass코어 소프트웨어에 최소 96MB RAM이 할당됩니다. 이 요구 사항에는 코어 기기에서 실행되는 구성 요소는 포함되지 않습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.
- 자바 런타임 환경 (JRE) 버전 8 이상. 장치의 PATH 환경 변수에서 Java를 사용할 수 있어야 합니다. Java를 사용하여 사용자 정의 구성 요소를 개발하려면 JDP(Java Devopment Kit)를 설치해야 합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
- GNU C 라이브러리 (glibc) 버전 2.25 이상
- AWS IoT GreengrassCore 소프트웨어를 루트 사용자로 실행해야 합니다. 예를 sudo 들어 사용하세요.
- AWS IoT GreengrassCore 소프트웨어 (예:) 를 실행하는 루트 사용자는 모든 사용자 및 그룹과 root sudo 함께 실행할 수 있는 권한이 있어야 합니다. /etc/sudoers파일은 이 사용자에게 다른 sudo 그룹으로 실행할 수 있는 권한을 부여해야 합니다. 에서 사용자의 권한은 다음 예와 /etc/sudoers 같아야 합니다.

```
root    ALL=(ALL:ALL) ALL
```

- 코어 디바이스는 엔드포인트 및 포트 세트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.
- /tmp디렉터리는 권한을 가지고 exec 마운트되어야 합니다.
- 다음 셸 명령 모두:
  - ps -ax -o pid,ppid
  - sudo
  - sh
  - kill
  - cp
  - chmod
  - rm
  - ln
  - echo
  - exit
  - id
  - uname
  - grep
- 기기에 다음과 같은 선택적 셸 명령이 필요할 수도 있습니다.
  - (선택 사항) systemctl. 이 명령은 AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설정하는 데 사용됩니다.
  - (선택 사항) useraddgroupadd, 및usermod. 이 명령은 ggc_user 시스템 사용자 및 ggc_group 시스템 그룹을 설정하는 데 사용됩니다.
  - (선택 사항) mkfifo. 이 명령은 Lambda 함수를 구성 요소로 실행하는 데 사용됩니다.
- 구성 요소 프로세스에 대한 시스템 리소스 제한을 구성하려면 디바이스에서 Linux 커널 버전 2.6.24 이상을 실행해야 합니다.
- Lambda 함수를 실행하려면 디바이스가 추가 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.

Windows

- AWS 리전지원하는 제품의 사용. AWS IoT Greengrass V2 지원되는 리전 목록은 AWS 일반 참조에서 AWS IoT Greengrass V2 엔드포인트 및 할당량을 참조하십시오.

- AWS IoT GreengrassCore 소프트웨어에 사용할 수 있는 최소 256MB의 디스크 공간. 이 요구 사항에는 코어 장치에 배포된 구성 요소는 포함되지 않습니다.
- AWS IoT Greengrass코어 소프트웨어에 최소 160MB RAM이 할당됩니다. 이 요구 사항에는 코어 기기에서 실행되는 구성 요소는 포함되지 않습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.
- 자바 런타임 환경 (JRE) 버전 8 이상. 장치의 PATH 시스템 변수에서 Java를 사용할 수 있어야 합니다. Java를 사용하여 사용자 정의 구성 요소를 개발하려면 JDP(Java Devopment Kit)를 설치해야 합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.

  > **Note**
  >
  > 그린그래스 핵 버전 2.5.0을 사용하려면 64비트 버전의 자바 런타임 환경 (JRE) 을 사용해야 합니다. 그린그래스 뉴클리어스 버전 2.5.1은 32비트 및 64비트 JRE를 지원합니다.

- Core 소프트웨어를 설치하는 사용자는 관리자여야 합니다. AWS IoT Greengrass
- AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 설치해야 합니다. 소프트웨어 설치 `--setup-system-service true` 시기를 지정하십시오.
- 구성 요소 프로세스를 실행하는 각 사용자가 LocalSystem 계정에 있어야 하고 사용자 이름과 암호가 해당 LocalSystem 계정의 Credential Manager 인스턴스에 있어야 합니다. 지침에 따라 AWS IoT GreengrassCore 소프트웨어를 설치할 때 이 사용자를 설정할 수 있습니다.
- 코어 기기는 엔드포인트 및 포트 세트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

Lambda 함수 요구 사항

Lambda 함수를 실행하려면 디바이스가 다음 요구 사항을 충족해야 합니다.

- Linux 기반 운영 체제.
- 장치에 `mkfifo` 셸 명령이 있어야 합니다.
- 디바이스는 Lambda 함수에 필요한 프로그래밍 언어 라이브러리를 실행해야 합니다. 필수 라이브러리를 디바이스에 설치하고 PATH 환경 변수에 추가해야 합니다. Greengrass는 Lambda가 지원하는 모든 버전의 Python, Node.js 및 Java 런타임을 지원합니다. Greengrass는 더 이상 사용되지 않는 Lambda 런타임 버전에 추가 제한을 적용하지 않습니다. Lambda 런타임용 AWS IoT Greengrass 지원에 대한 자세한 내용은 AWS Lambda함수 실행 섹션을 참조하세요.

- 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 다음 요구 사항을 충족해야 합니다.
  - Linux 커널 버전 4.4 이상.
  - 커널은 cgroups v1을 지원해야 하며 다음 cgroup을 활성화하고 마운트해야 합니다.
    - 컨테이너화된 Lambda 함수의 메모리 제한을 설정하는 AWS IoT Greengrass 데 사용되는 메모리 cgroup입니다.
    - 디바이스 cgroup은 시스템 디바이스 또는 볼륨에 액세스하는 컨테이너식 Lambda 함수를 위한 것입니다.

    AWS IoT Greengrass코어 소프트웨어는 cgroups v2를 지원하지 않습니다.

    이 요구 사항을 충족하려면 다음 Linux 커널 파라미터를 사용하여 기기를 부팅하십시오.

    ```
    cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
    ```

    > **Tip**
    >
    > Raspberry Pi에서는 `/boot/cmdline.txt` 파일을 편집하여 기기의 커널 매개변수를 설정합니다.

  - 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`
    - 기타:
      - `CONFIG_POSIX_MQUEUE`
      - `CONFIG_OVERLAY_FS`
      - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`
      - `CONFIG_SECCOMP_FILTER`

- `CONFIG_KEYS`
- `CONFIG_SECCOMP`
- `CONFIG_SHMEM`

> **Tip**
>
> Linux 커널 파라미터를 확인하고 설정하는 방법을 알아보려면 Linux 배포판 설명서를 참조하십시오. for를 사용하여 AWS IoT Device Tester 장치가 이러한 요구 사항을 충족하는지 확인할 수도 있습니다. AWS IoT Greengrass 자세한 설명은 AWS IoT Device TesterAWS IoT GreengrassV2에 사용하기 섹션을 참조하세요.

Windows 디바이스의 기능 고려 사항

일부 AWS IoT Greengrass 기능은 현재 Windows 장치에서 지원되지 않습니다. 기능 차이점을 검토하여 Windows 장치가 요구 사항을 충족하는지 확인하세요. 자세한 설명은 운영 체제별 Greengrass 기능 호환성 섹션을 참조하세요.

AWS 계정 설정

AWS 계정가 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입하려면

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

   가입 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정에 가입하면 AWS 계정 루트 사용자 항목이 생성됩니다. 루트 사용자에게 계정의 모든 AWS 서비스 및 리소스에 대한 액세스 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

다음 옵션 중 하나를 선택하여 관리 사용자를 생성합니다.

관리자를 관리하는 방법 한 가지 선택	목적	By	다른 방법
IAM Identity Center에서 (권장)	단기 보안 인증 정보를 사용하여 AWS에 액세스합니다. 이는 보안 모범 사례와 일치합니다. 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.	AWS IAM Identity Center 사용 설명서의 시작하기 지침을 따르세요.	AWS Command Line Interface 사용 설명서의 AWS IAM Identity Center 사용할 AWS CLI 구성을 통해 프로그래밍 방식의 액세스를 구성합니다.
IAM에서 (권장되지 않음)	장기 보안 인증 정보를 사용하여 AWS에 액세스합니다.	IAM 사용 설명서의 첫 IAM 관리 사용자 및 사용자 그룹 만들기에 나온 지침을 따릅니다.	IAM 사용 설명서에 나온 IAM 사용자의 액세스 키 관리에서 프로그래밍 방식 액세스를 구성합니다.

AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass에지 AWS 디바이스까지 확장하여 생성된 데이터를 기반으로 조치를 취하고 관리, 분석 및 내구성 있는 스토리지에 사용할 수 있습니다. AWS 클라우드 엣지 디바이스에 AWS IoT Greengrass Core 소프트웨어를 설치하여 AWS IoT Greengrass 및 디바이스와 통합할 수 있습니다. AWS 클라우드

> **Important**
>
> Core 소프트웨어를 다운로드하고 설치하기 전에 AWS IoT Greengrass 코어 장치가 Core 소프트웨어 v2.0을 설치하고 실행하기 위한 요구 사항을 충족하는지 확인하십시오. AWS IoT Greengrass

AWS IoT GreengrassCore 소프트웨어에는 장치를 Greengrass 코어 장치로 설정하는 설치 프로그램이 포함되어 있습니다. 설치 프로그램을 실행하면 루트 폴더 및 사용할 폴더와 같은 옵션을 구성할 수 있습니다. AWS 리전 설치 프로그램이 필요한 AWS IoT 리소스와 IAM 리소스를 자동으로 생성하도록 선택할 수 있습니다. 또한 로컬 개발 도구를 배포하여 사용자 지정 구성 요소 개발에 사용할 디바이스를 구성할 수도 있습니다.

AWS IoT GreengrassCore 소프트웨어를 에 AWS 클라우드 연결하고 작동하려면 다음과 같은 AWS IoT IAM 리소스가 필요합니다.

- AWS IoT 사물. 디바이스를 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다. AWS IoT AWS 이 인증서를 사용하면 장치가 AWS IoT 및 AWS IoT Greengrass 와 통신할 수 있습니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.
- (선택 사항) AWS IoT 사물 그룹. 사물 그룹을 사용하여 수많은 Greengrass 코어 디바이스를 관리합니다. 소프트웨어 구성 요소를 장치에 배포할 때 개별 장치 또는 장치 그룹에 배포하도록 선택할 수 있습니다. 사물 그룹에 장치를 추가하여 해당 사물 그룹의 소프트웨어 구성 요소를 장치에 배포할 수 있습니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.
- IAM 역할. Greengrass 코어 디바이스는 AWS IoT Core 자격 증명 공급자를 사용하여 IAM 역할을 가진 AWS 서비스에 대한 호출을 승인합니다. 이 역할을 통해 디바이스는 Amazon Logs와 상호 작용하고AWS IoT, 로그를 Amazon Logs로 전송하고, Amazon CloudWatch Simple Storage Service (Amazon S3) 에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.
- AWS IoT역할 별칭. Greengrass 코어 디바이스는 역할 별칭을 사용하여 사용할 IAM 역할을 식별합니다. 역할 별칭을 사용하면 IAM 역할을 변경하면서도 기기 구성을 동일하게 유지할 수 있습니다. 자세한 내용은 개발자 안내서의 AWS서비스에 대한 다이렉트 콜 승인을 참조하십시오. AWS IoT Core

다음 옵션 중 하나를 선택하여 AWS IoT Greengrass 코어 장치에 Core 소프트웨어를 설치합니다.

- 빠른 설치

  가능한 한 적은 단계로 Greengrass 코어 디바이스를 설정하려면 이 옵션을 선택하십시오. 설치 프로그램이 필요한 AWS IoT 리소스와 IAM 리소스를 자동으로 생성합니다. 이 옵션을 사용하려면 설치 프로그램에 AWS 자격 증명을 제공해야 에서 리소스를 생성할 수 있습니다. AWS 계정

  이 옵션을 사용하여 방화벽 또는 네트워크 프록시 뒤에 설치할 수 없습니다. 장치가 방화벽 또는 네트워크 프록시 뒤에 있는 경우 수동 설치를 고려해 보세요.

자세한 설명은 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

- 수동 설치

  필요한 AWS 리소스를 수동으로 생성하거나 방화벽 또는 네트워크 프록시 뒤에 설치하려면 이 옵션을 선택합니다. 수동 설치를 사용하면 필수 AWS IoT 및 IAM 리소스를 생성하므로 설치 관리자에게 에서 리소스를 AWS 계정 생성할 권한을 부여할 필요가 없습니다. 포트 443이나 네트워크 프록시를 통해 연결하도록 디바이스를 구성할 수도 있습니다. 또한 하드웨어 보안 모듈 (HSM), TPM (신뢰할 수 있는 플랫폼 모듈) 또는 다른 암호화 요소에 저장한 개인 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다.

  자세한 설명은 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

- 플릿 프로비저닝을 사용한 설치 AWS IoT

  AWS IoT플릿 프로비저닝 템플릿에서 필요한 AWS 리소스를 생성하려면 이 옵션을 선택합니다. 비슷한 디바이스를 하나의 플릿으로 만들거나 고객이 나중에 활성화하는 디바이스 (예: 차량 또는 스마트 홈 디바이스) 를 제조하는 경우 이 옵션을 선택할 수 있습니다. 디바이스는 클레임 인증서를 사용하여 AWS 리소스를 인증하고 프로비저닝합니다. 여기에는 디바이스가 정상 작동을 위해 연결하는 데 사용하는 X.509 클라이언트 인증서가 포함됩니다. AWS 클라우드 제조 과정에서 클레임 인증서를 디바이스의 하드웨어에 내장하거나 플래시할 수 있으며, 동일한 클레임 인증서와 키를 사용하여 여러 디바이스를 프로비전할 수 있습니다. 포트 443이나 네트워크 프록시를 통해 연결하도록 장치를 구성할 수도 있습니다.

  자세한 설명은 AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

- 사용자 지정 프로비저닝을 사용한 설치

  필요한 AWS 리소스를 프로비저닝하는 사용자 정의 Java 애플리케이션을 개발하려면 이 옵션을 선택합니다. 자체 X.509 클라이언트 인증서를 생성하거나 프로비저닝 프로세스를 보다 세밀하게 제어하려는 경우 이 옵션을 선택할 수 있습니다. AWS IoT Greengrass사용자 지정 프로비저닝 애플리케이션과 Core 소프트웨어 설치 프로그램 간에 정보를 교환하기 위해 구현할 수 있는 인터페이스를 제공합니다. AWS IoT Greengrass

  자세한 설명은 사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

또한 AWS IoT Greengrass는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 컨테이너화된 환경을 제공합니다. Dockerfile을 사용하여 Docker 컨테이너에서 실행할 AWS IoT Greengrass 수 있습니다.

주제

- 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치
- 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치
- AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치
- 사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치
- 인스톨러 인수

자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치

AWS IoT GreengrassCore 소프트웨어에는 장치를 Greengrass 코어 장치로 설정하는 설치 프로그램이 포함되어 있습니다. 장치를 빠르게 설정하기 위해 설치 프로그램은 핵심 장치가 작동하는 데 필요한 AWS IoT 사물, AWS IoT 사물 그룹, IAM 역할 및 AWS IoT 역할 별칭을 프로비저닝할 수 있습니다. 또한 설치 프로그램은 로컬 개발 도구를 코어 장치에 배포할 수 있으므로 장치를 사용하여 사용자 지정 소프트웨어 구성 요소를 개발하고 테스트할 수 있습니다. 설치 프로그램에서 이러한 리소스를 제공하고 배포를 생성하려면 AWS 자격 증명이 필요합니다.

디바이스에 AWS 자격 증명을 제공할 수 없는 경우 코어 디바이스가 작동하는 데 필요한 AWS 리소스를 프로비저닝할 수 있습니다. 개발 도구를 코어 장치에 배포하여 개발 장치로 사용할 수도 있습니다. 이렇게 하면 설치 프로그램을 실행할 때 장치에 제공하는 권한을 줄일 수 있습니다. 자세한 설명은 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

> **Important**
>
> Core 소프트웨어를 다운로드하기 전에 AWS IoT Greengrass 코어 장치가 Core 소프트웨어 v2.0을 설치하고 실행하기 위한 요구 사항을 충족하는지 확인하십시오. AWS IoT Greengrass

주제

- 장치 환경을 설정합니다.
- 디바이스에 AWS 자격 증명을 제공합니다.
- AWS IoT GreengrassCore 소프트웨어 다운로드
- AWS IoT Greengrass 코어 소프트웨어 설치

장치 환경을 설정합니다.

이 섹션의 단계에 따라 AWS IoT Greengrass 핵심 장치로 사용할 Linux 또는 Windows 장치를 설정합니다.

Linux 디바이스 설정

Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT GreengrassCore 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 장치에 OpenJDK를 설치하는 방법을 보여줍니다.

   - Debian 기반 또는 Ubuntu 기반 배포판의 경우:

     ```
     sudo apt install default-jdk
     ```

   - Red Hat 기반 배포판의 경우:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - 대상 Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 아마존 리눅스 2023의 경우:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

   ```
   java -version
   ```

   이 명령은 장치에서 실행되는 Java 버전을 인쇄합니다. 예를 들어 Debian 기반 배포의 경우 출력은 다음 샘플과 비슷할 수 있습니다.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (선택 사항) 기기에서 구성 요소를 실행하는 기본 시스템 사용자 및 그룹을 생성합니다. 설치 중에 설치 프로그램 인수를 사용하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 이 사용자 및 그룹을 만들도록 선택할 수도 있습니다. `--component-default-user` 자세한 설명은 인스톨러 인수 섹션을 참조하세요.

   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```

3. AWS IoT GreengrassCore 소프트웨어를 실행하는 사용자 (일반적으로`root`) 에게 모든 사용자 및 그룹과 `sudo` 함께 실행할 수 있는 권한이 있는지 확인하십시오.

   a. 다음 명령을 실행하여 `/etc/sudoers` 파일을 엽니다.

      ```
      sudo visudo
      ```

   b. 사용자의 권한이 다음 예와 같은지 확인합니다.

      ```
      root    ALL=(ALL:ALL) ALL
      ```

4. (선택 사항) 컨테이너화된 Lambda 함수를 실행하려면 cgroups v1을 활성화하고 메모리 및 디바이스 cgroups를 활성화하고 마운트해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없다면 이 단계를 건너뛰어도 됩니다.

   이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터를 사용하여 디바이스를 부팅하십시오.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   디바이스의 커널 파라미터 보기 및 설정에 대한 자세한 내용은 운영 체제 및 부트로더 설명서를 참조하십시오. 지침에 따라 커널 파라미터를 영구적으로 설정하십시오.

5. 의 요구 사항 목록에 나와 있는 대로 다른 모든 필수 종속 항목을 장치에 설치하십시오. 디바이스 요구 사항

Windows 디바이스 설정

> **Note**
>
> 이 기능은 v2.5.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT GreengrassCore 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
2. PATH 시스템 변수에서 Java를 사용할 수 있는지 확인하고 사용할 수 없는 경우 추가하십시오. 이 LocalSystem 계정은 AWS IoT Greengrass Core 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 다음을 따릅니다.

   a. Windows 키를 눌러 시작 메뉴를 엽니다.

   b. 시작 메뉴에서 시스템 옵션을 **environment variables** 검색하려면 입력합니다.

   c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.

   d. 환경 변수 선택... 환경 변수 창을 열려면

   e. 시스템 변수에서 경로를 선택한 다음 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.

   f. Java 설치 `bin` 폴더 경로가 있는지 확인합니다. 경로는 다음 예와 비슷할 수 있습니다.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Java 설치 `bin` 폴더가 경로에 없는 경우 [새로 만들기] 를 선택하여 추가한 다음 [확인] 을 선택합니다.

3. Windows 명령 프롬프트 (`cmd.exe`) 를 관리자 권한으로 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. *### ## ###* 바꾸십시오.

```
net user /add ggc_user password
```

> **Tip**
>
> Windows 구성에 따라 사용자 암호가 향후 날짜에 만료되도록 설정될 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 업데이트하십시오. 사용자 비밀번호가 만료되지 않도록 설정할 수도 있습니다.
>
> - 사용자 및 암호가 언제 만료되는지 확인하려면 다음 명령을 실행합니다.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 사용자 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 이 명령이 더 이상 사용되지 않는 Windows 10 이상을 사용하는 경우 wmic 다음 명령을 실행하세요. PowerShell

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Microsoft에서 PsExec유틸리티를 다운로드하여 장치에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. ### 이전에 설정한 사용자 암호로 바꾸십시오.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

가 PsExec License Agreement열리면 라이선스에 Accept동의하도록 선택하고 명령을 실행합니다.

> **Note**
>
> Windows 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 계정에 기본 사용자 정보를 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

디바이스에 AWS 자격 증명을 제공합니다.

설치 프로그램이 필요한 AWS 리소스를 프로비저닝할 수 있도록 디바이스에 AWS 자격 증명을 제공하십시오. 필요한 권한에 대한 자세한 내용은 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책(을)를 참조하십시오.

디바이스에 AWS 자격 증명을 제공하려면

- 디바이스에 AWS 자격 증명을 제공하면 설치 프로그램이 코어 디바이스에 AWS IoT 및 IAM 리소스를 프로비저닝할 수 있습니다. 보안을 강화하려면 프로비저닝에 필요한 최소 권한만 허용하는 IAM 역할에 대한 임시 자격 증명을 확보하는 것이 좋습니다. 자세한 설명은 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책 섹션을 참조하세요.

> **Note**
>
> 설치 프로그램은 자격 증명을 저장하거나 저장하지 않습니다.

디바이스에서 다음 중 하나를 수행하여 자격 증명을 검색하고 AWS IoT Greengrass Core 소프트웨어 설치 프로그램에서 사용할 수 있도록 하세요.

- (권장) 에서 임시 자격 증명을 사용하십시오. AWS IAM Identity Center

  a. IAM ID 센터의 액세스 키 ID, 보안 액세스 키 및 세션 토큰을 제공하십시오. 자세한 내용은 IAM Identity Center 사용 설명서의 임시 자격 증명 가져오기 및 새로 고침에서 수동 자격 증명 새로 고침을 참조하십시오.

  b. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어에 자격 증명을 제공하십시오.

  Linux or Unix

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

  Windows Command Prompt (CMD)

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

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- IAM 역할의 임시 보안 자격 증명을 사용하십시오.

  a. 수임한 IAM 역할의 액세스 키 ID, 보안 액세스 키, 세션 토큰을 제공하십시오. 이러한 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 임시 보안 자격 증명 요청을 참조하십시오.

b. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어에 자격 증명을 제공하십시오.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- IAM 사용자의 장기 자격 증명을 사용하십시오.

a. IAM 사용자의 액세스 키 ID와 비밀 액세스 키를 제공하십시오. 프로비저닝용 IAM 사용자를 생성하고 나중에 삭제할 수 있습니다. 사용자에게 제공하는 IAM 정책은 을 참조하십시오. 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

b. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어에 자격 증명을 제공하십시오.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

c. (선택 사항) Greengrass 디바이스를 프로비저닝하기 위해 IAM 사용자를 생성한 경우 해당 사용자를 삭제하십시오.

d. (선택 사항) 기존 IAM 사용자의 액세스 키 ID와 비밀 액세스 키를 사용한 경우 해당 사용자의 키를 업데이트하여 더 이상 유효하지 않도록 하십시오. 자세한 내용은 AWS Identity and Access Management사용 설명서의 액세스 키 업데이트를 참조하십시오.

AWS IoT GreengrassCore 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다.

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 다음 위치에서 특정 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다. *###* 다운로드할 버전으로 교체하십시오.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

AWS IoT GreengrassCore 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 이름이 지정된 파일에 AWS IoT Greengrass Core 소프트웨어를 `greengrass-nucleus-latest.zip` 다운로드합니다.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

2. (선택 사항) Greengrass 핵 소프트웨어 서명을 확인하려면

> **Note**
>
> 이 기능은 Greengrass 핵 버전 2.9.5 이상에서 사용할 수 있습니다.

a. 다음 명령을 사용하여 Greengrass 핵 아티팩트의 서명을 확인하십시오.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`호출 시 확인 결과를 나타내는 출력이 출력됩니다.

   i. Greengrass nucleus zip 파일에 서명된 경우 출력에는 다음 명령문이 포함됩니다.

   ```
   jar verified.
   ```

   ii. Greengrass nucleus zip 파일에 서명되지 않은 경우 출력에는 다음 명령문이 포함됩니다.

   ```
   jar is unsigned.
   ```

c. Jarsigner `-certs` 옵션을 `-verify -verbose` 옵션과 함께 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. AWS IoT GreengrassCore 소프트웨어를 디바이스의 폴더에 압축을 풉니다. 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (선택 사항) 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 버전을 확인합니다.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

> **Important**
>
> v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 Core 소프트웨어를 설치한 후에 이 폴더를 제거하지 마십시오. AWS IoT Greengrass AWS IoT GreengrassCore 소프트웨어는 이 폴더의 파일을 사용하여 실행합니다.
> 최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치해야 하며, AWS IoT Greengrass Core 소프트웨어를 설치한 후 이 폴더를 제거할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어 설치

다음을 수행하도록 지정하는 인수를 사용하여 설치 프로그램을 실행합니다.

- 코어 기기 작동에 필요한 AWS 리소스를 생성합니다.
- `ggc_user`시스템 사용자를 사용하여 코어 장치에서 소프트웨어 구성 요소를 실행하도록 지정합니다. Linux 장치에서 이 명령은 또한 `ggc_group` 시스템 그룹을 사용하도록 지정하며, 설치 프로그램은 시스템 사용자 및 그룹을 자동으로 생성합니다.
- AWS IoT Greengrass코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 장치에서는 Systemd init 시스템이 필요합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

로컬 개발 도구를 사용하여 개발 디바이스를 설정하려면 `--deploy-dev-tools true` 인수를 지정하십시오. 로컬 개발 도구는 설치가 완료된 후 배포하는 데 최대 1분이 걸릴 수 있습니다.

지정할 수 있는 인수에 대한 자세한 내용은 을 참조하십시오인스톨러 인수.

> **Note**
>
> 메모리가 제한된 AWS IoT Greengrass 장치에서 실행하는 경우 AWS IoT Greengrass Core 소프트웨어가 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하려면 nucleus 구성 요소의 `jvmOptions` 구성 매개변수에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. Core 설치 프로그램을 실행합니다. AWS IoT Greengrass 명령의 인수 값을 다음과 같이 바꿉니다.

   a. */greengrass/v2*또는 *C:\greengrass\v2*: AWS IoT Greengrass Core 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.

   b. *GreengrassInstaller*. AWS IoT GreengrassCore 소프트웨어 설치 프로그램의 압축을 푼 폴더의 경로입니다.

   c. *##.* 리소스를 찾거나 생성하는 위치. AWS 리전

   d. *MyGreengrassCore*. Greengrass 코어 디바이스의 AWS IoT 이름입니다. 해당 사물이 존재하지 않는 경우 설치 프로그램이 생성합니다. 설치 프로그램은 인증서를 다운로드하여 사물로 인증합니다. AWS IoT 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.

      > **Note**
      >
      > 사물 이름에는 콜론 () : 문자를 포함할 수 없습니다.

   e. *MyGreengrassCoreGroup*. Greengrass 코어 디바이스의 AWS IoT 사물 그룹 이름입니다. 사물 그룹이 없는 경우 설치 프로그램은 사물 그룹을 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 배포가 활성화되어 있는 경우 코어 장치는 배포에서 지정하는 소프트웨어를 다운로드하고 실행합니다.

      > **Note**
      >
      > 사물 그룹 이름에는 콜론 (:) 문자를 포함할 수 없습니다.

   f. *ThingPolicy#####* V2IoT. Greengrass 코어 디바이스가 및 와 AWS IoT 통신할 수 있도록 허용하는 AWS IoT 정책의 이름입니다. AWS IoT Greengrass AWS IoT정책이 존재하지 않는 경우 설치 프로그램은 이 이름으로 허용 AWS IoT 정책을 생성합니다. 사용 사례에 맞게 이 정책의 권한을 제한할 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

   g. *##### v2 TokenExchangeRole*. Greengrass 코어 디바이스가 임시 자격 증명을 받을 AWS 수 있도록 허용하는 IAM 역할의 이름입니다. 역할이 존재하지 않는 경우 설치 프로그램은 역할을 생성하고 이름이 지정된 정책을 생성하여 연결합니다. *GreengrassV2TokenExchangeRole*Access 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

h. *GreengrassCoreTokenExchangeRoleAlias*. Greengrass 코어 디바이스가 나중에 임시 자격 증명을 받을 수 있도록 허용하는 IAM 역할의 별칭입니다. 역할 별칭이 없는 경우 설치 프로그램은 역할 별칭을 생성하여 지정한 IAM 역할을 가리킵니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
```

```
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true
```

> **Important**
>
> Windows 코어 디바이스에서는 Core 소프트웨어를 `--setup-system-service true` 시스템 서비스로 설정하도록 지정해야 합니다. AWS IoT Greengrass

설치 프로그램이 성공하면 다음 메시지를 인쇄합니다.

- 지정한 `--provision` 경우 설치 프로그램은 리소스가 성공적으로 구성되었는지 `Successfully configured Nucleus with provisioned resource details` 여부를 인쇄합니다.
- 지정한 `--deploy-dev-tools` 경우 설치 프로그램은 배포가 성공적으로 `Configured Nucleus to deploy aws.greengrass.Cli component` 생성되었는지 여부를 인쇄합니다.
- 지정한 `--setup-system-service true` 경우 설치 프로그램은 소프트웨어를 서비스로 설정하고 실행했는지 `Successfully set up Nucleus as a system service` 여부를 인쇄합니다.
- `--setup-system-service true`지정하지 않으면 설치 프로그램이 성공하고 소프트웨어를 실행했는지 `Launched Nucleus successfully` 여부를 인쇄합니다.

2. 그린그래스 핵v2.0.4 이상을 설치한 경우 이 단계를 건너뛰십시오. 최신 버전의 소프트웨어를 다운로드한 경우 v2.0.4 이상을 설치한 것입니다.

   다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 루트 폴더에 필요한 파일 권한을 설정합니다. 설치 명령에 지정한 루트 */greengrass/v2* 폴더로 바꾸고 */greengrass#* 루트 폴더의 상위 폴더로 바꾸십시오.

   ```
   sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
   ```

AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 설치한 경우 설치 프로그램이 소프트웨어를 자동으로 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 설명은 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.

> **Note**
>
> 기본적으로 설치 관리자가 생성하는 IAM 역할은 S3 버킷의 구성 요소 아티팩트에 대한 액세스를 허용하지 않습니다. Amazon S3에 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 설명은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용 섹션을 참조하세요. 구성 요소 아티팩트를 위한 S3 버킷이 아직 없는 경우, 버킷을 생성한 후 나중에 이러한 권한을 추가할 수 있습니다.

소프트웨어 구성 및 사용 방법에 대한 자세한 내용은 AWS IoT Greengrass 다음을 참조하십시오.

- AWS IoT Greengrass Core 소프트웨어 구성
- AWS IoT Greengrass구성 요소 개발
- 디바이스에 AWS IoT Greengrass 구성 요소 배포
- 그린그래스 커맨드 라인 인터페이스

수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치

AWS IoT Greengrass Core 소프트웨어에는 장치를 Greengrass 코어 장치로 설정하는 설치 프로그램이 포함되어 있습니다. 디바이스를 수동으로 설정하려면 디바이스에서 사용할 필수 AWS IoT 리소스와 IAM 리소스를 생성할 수 있습니다. 이러한 리소스를 수동으로 생성하는 경우 설치 프로그램에 AWS 자격 증명을 제공할 필요가 없습니다.

AWS IoT Greengrass Core 소프트웨어를 수동으로 설치하는 경우 네트워크 프록시를 사용하거나 포트 AWS 443에 연결하도록 장치를 구성할 수도 있습니다. 예를 들어 장치가 방화벽이나 네트워크 프록시 뒤에서 실행되는 경우 이러한 구성 옵션을 지정해야 할 수 있습니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

PKCS #11 인터페이스를 통해 하드웨어 보안 모듈 (HSM) 을 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수도 있습니다. 이 기능을 사용하면 개인 키와 인증서 파일이 소프트웨어에서 노출되거나 중복되지 않도록 안전하게 저장할 수 있습니다. HSM, TPM (신뢰할 수 있는 플랫폼 모듈) 또는 다른 암호화 요소와 같은 하드웨어 모듈에 개인 키와 인증서를 저장할 수 있습니다. 이 기능은 Linux

장치에서만 사용할 수 있습니다. 하드웨어 보안 및 사용 요구 사항에 대한 자세한 내용은 을 참조하십시오하드웨어 보안 통합.

> **Important**
>
> AWS IoT Greengrass Core 소프트웨어를 다운로드하기 전에 코어 장치가 Core 소프트웨어 v2.0을 설치하고 실행하기 위한 요구 사항을 충족하는지 확인하십시오. AWS IoT Greengrass

주제

- 엔드포인트 검색 AWS IoT
- 사물 생성하기 AWS IoT
- 사물 인증서를 생성하세요.
- 사물 인증서를 구성합니다.
- 토큰 교환 역할 생성
- 인증서를 디바이스에 다운로드합니다.
- 디바이스 환경 설정
- AWS IoT Greengrass 코어 소프트웨어 다운로드
- Core 소프트웨어를 설치합니다. AWS IoT Greengrass

엔드포인트 검색 AWS IoT

AWS IoT 엔드포인트를 가져와서 나중에 사용할 수 있도록 저장하세요 AWS 계정. 장치는 이러한 엔드포인트를 사용하여 연결합니다. AWS IoT 다음을 따릅니다.

1. 사용자의 AWS IoT 데이터 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 사용자의 AWS IoT 자격 증명 엔드포인트를 가져오세요. AWS 계정

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

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

사물 생성하기 AWS IoT

AWS IoT 사물은 연결된 장치 및 논리적 개체를 나타냅니다 AWS IoT. Greengrass 코어 디바이스는 AWS IoT 사물입니다. 장치를 사물로 등록하면 해당 장치가 디지털 인증서를 사용하여 인증할 수 있습니다. AWS IoT AWS

이 섹션에서는 디바이스를 나타내는 AWS IoT 사물을 생성합니다.

사물을 만들려면 AWS IoT

1. 디바이스에 AWS IoT 맞는 사물을 만드세요. 개발 컴퓨터에서 다음 명령을 실행합니다.

   - 사용할 사물 *MyGreengrassCore*이름으로 바꿉니다. 이 이름은 Greengrass 코어 디바이스의 이름이기도 합니다.

     > **Note**
     >
     > 사물 이름에는 콜론 (:) 문자를 포함할 수 없습니다.

```
aws iot create-thing --thing-name MyGreengrassCore
```

요청이 성공하면 응답은 다음 예제와 비슷합니다.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
```

```
}
```

2. (선택 사항) 새 AWS IoT 사물 그룹 또는 기존 사물 그룹에 사물을 추가합니다. 사물 그룹을 사용하여 수많은 Greengrass 코어 디바이스를 관리합니다. 소프트웨어 구성 요소를 장치에 배포할 때 개별 장치 또는 장치 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포가 있는 사물 그룹에 장치를 추가하여 해당 사물 그룹의 소프트웨어 구성 요소를 장치에 배포할 수 있습니다. 다음을 따릅니다.

   a. (선택 사항) AWS IoT 사물 그룹을 생성합니다.

      - 생성할 사물 그룹의 *MyGreengrassCoreGroup*이름으로 바꿉니다.

      > **Note**
      >
      > 사물 그룹 이름에는 콜론 (:) 문자를 포함할 수 없습니다.

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      요청이 성공하면 응답은 다음 예와 비슷합니다.

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

   b. AWS IoT 사물 그룹에 사물을 추가합니다.

      - 사물 *MyGreengrassCore*이름으로 바꾸세요 AWS IoT .
      - 사물 그룹의 *MyGreengrassCoreGroup*이름으로 바꾸십시오.

      ```
      aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
      name MyGreengrassCoreGroup
      ```

      요청이 성공하면 명령이 출력되지 않습니다.

사물 인증서를 생성하세요.

장치를 사물로 등록하면 해당 장치가 디지털 인증서를 사용하여 인증할 수 있습니다. AWS IoT AWS 이 인증서를 사용하면 장치가 AWS IoT 및 AWS IoT Greengrass 와 통신할 수 있습니다.

이 섹션에서는 장치를 연결하는 데 사용할 수 있는 인증서를 만들고 AWS 다운로드합니다.

하드웨어 보안 모듈 (HSM) 을 사용하여 개인 키와 인증서를 안전하게 저장하도록 AWS IoT Greengrass 코어 소프트웨어를 구성하려면 HSM의 개인 키에서 인증서를 생성하는 단계를 따르십시오. 그렇지 않으면 단계에 따라 서비스에 인증서와 개인 키를 생성하십시오. AWS IoT 하드웨어 보안 기능은 Linux 디바이스에서만 사용할 수 있습니다. 하드웨어 보안 및 사용 요구 사항에 대한 자세한 내용은 을 참조하십시오하드웨어 보안 통합.

AWS IoT 서비스에 인증서 및 개인 키를 생성하십시오.

사물 인증서를 만들려면

1. 사물에 대한 인증서를 다운로드할 폴더를 AWS IoT 생성합니다.

   ```
   mkdir greengrass-v2-certs
   ```

2. 사물에 대한 인증서를 생성하고 AWS IoT 다운로드합니다.

   ```
   aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
    greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
   public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
   ```

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificateId":
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificatePem": "-----BEGIN CERTIFICATE-----
   MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
    0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
    WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
    EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
    jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
    MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
    WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
   ```

```
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

나중에 인증서를 구성하는 데 사용할 인증서의 Amazon 리소스 이름 (ARN) 을 저장합니다.

HSM의 프라이빗 키로 인증서를 생성합니다.

> **Note**
>
> 이 기능은 v2.5.3 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

사물 인증서를 만들려면

1. 코어 디바이스에서 HSM에서 PKCS #11 토큰을 초기화하고 개인 키를 생성합니다. 개인 키는 키 크기가 RSA-2048 (또는 그 이상) 인 RSA 키이거나 ECC 키여야 합니다.

> **Note**
>
> ECC 키가 있는 하드웨어 보안 모듈을 사용하려면 Greengrass nucleus v2.5.6 이상을 사용해야 합니다.
> 하드웨어 보안 모듈과 시크릿 관리자를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

토큰을 초기화하고 개인 키를 생성하는 방법을 알아보려면 HSM 설명서를 참조하십시오. HSM이 개체 ID를 지원하는 경우 개인 키를 생성할 때 개체 ID를 지정하십시오. 토큰을 초기화하고 개인 키를 생성할 때 지정하는 슬롯 ID, 사용자 PIN, 개체 레이블, 개체 ID (HSM에서 사용하는 경우) 를 저장합니다. 이 값은 나중에 사물 인증서를 HSM으로 가져오고 Core 소프트웨어를 구성할 때 사용합니다. AWS IoT Greengrass

2. 개인 키에서 인증서 서명 요청 (CSR) 을 생성합니다. AWS IoT 이 CSR을 사용하여 HSM에서 생성한 개인 키에 대한 사물 인증서를 생성합니다. 프라이빗 키에서 CSR을 생성하는 방법에 대한 자세한 내용은 HSM 설명서를 참조하십시오. CSR은 파일입니다 (예:). `iotdevicekey.csr`
3. 디바이스에서 개발 컴퓨터로 CSR을 복사합니다. 개발 컴퓨터와 장치에서 SSH와 SCP가 활성화된 경우 개발 컴퓨터의 `scp` 명령을 사용하여 CSR을 전송할 수 있습니다. 디바이스의 IP 주소로 바꾸고 *~/iotdevicekey.csr#* 디바이스의 CSR 파일 *device-ip-address*경로로 바꾸십시오.

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 개발 컴퓨터에 사물에 대한 인증서를 다운로드할 폴더를 만드십시오. AWS IoT

```
mkdir greengrass-v2-certs
```

5. CSR 파일을 사용하여 사물에 대한 인증서를 생성하고 개발 AWS IoT 컴퓨터에 다운로드합니다.

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-
request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
device.pem.crt
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
```

```
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

나중에 인증서를 구성하는 데 사용할 인증서의 ARN을 저장합니다.

사물 인증서를 구성합니다.

이전에 만든 사물에 AWS IoT 사물 인증서를 연결하고 핵심 장치에 대한 AWS IoT 권한을 정의하는 AWS IoT 정책을 인증서에 추가합니다.

사물 인증서를 구성하려면

1. 인증서를 사물에 AWS IoT 첨부합니다.

   - AWS IoT 물건 *MyGreengrassCore*이름으로 바꾸세요.
   - 인증서 Amazon 리소스 이름 (ARN) 을 이전 단계에서 생성한 인증서의 ARN으로 교체합니다.

```
aws iot attach-thing-principal --thing-name MyGreengrassCore
 --principal arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청이 성공하면 명령이 출력되지 않습니다.

2. Greengrass 코어 AWS IoT 디바이스에 대한 권한을 정의하는 AWS IoT 정책을 생성하여 첨부하십시오. 다음 정책은 모든 MQTT 주제 및 Greengrass 작업에 대한 액세스를 허용하므로 장치가 사용자 지정 애플리케이션 및 새로운 Greengrass 작업이 필요한 향후 변경 사항과 함께 작동하도록 할 수 있습니다. 사용 사례에 따라 이 정책을 제한할 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

   이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 해당 AWS IoT 정책을 연결할 수 있습니다.

   다음을 따릅니다.

   a. Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

      ```
      nano greengrass-v2-iot-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "iot:Publish",
              "iot:Subscribe",
              "iot:Receive",
              "iot:Connect",
              "greengrass:*"
            ],
            "Resource": [
              "*"
            ]
          }
        ]
      }
      ```

b. AWS IoT 정책 문서에서 정책을 생성합니다.

- *GreenGrassV2IoT# ThingPolicy* 생성할 정책의 이름으로 바꾸십시오.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. AWS IoT 정책을 사물의 인증서에 연결합니다. AWS IoT

- *GreenGrassV2IoT# ThingPolicy ###* 정책 이름으로 바꾸십시오.
- 대상 ARN을 사내 인증서의 ARN으로 바꾸십시오. AWS IoT

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청이 성공하면 명령이 출력되지 않습니다.

토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라고 하는 IAM 서비스 역할을 사용하여 서비스 호출을 승인합니다. AWS 디바이스는 AWS IoT 자격 증명 공급자를 사용하여 이 역할에 대한 임시 AWS 자격 증명을 가져오고, 이를 통해 디바이스가 Amazon Logs와 상호 작용하고 AWS IoT, Amazon Logs에 로그를 전송하고, Amazon CloudWatch S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

AWS IoT 역할 별칭을 사용하여 Greengrass 코어 디바이스의 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 장치의 토큰 교환 역할을 변경하면서도 장치 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 AWS 서비스에 대한 다이렉트 콜 승인을 참조하십시오.

이 섹션에서는 토큰 교환 IAM 역할과 해당 역할을 가리키는 AWS IoT 역할 별칭을 생성합니다. Greengrass 코어 디바이스를 이미 설정한 경우 새 디바이스를 생성하는 대신 토큰 교환 역할 및 역할 별칭을 사용할 수 있습니다. 그런 다음 해당 역할과 별칭을 사용하도록 디바이스의 AWS IoT 사물을 구성합니다.

토큰 교환 IAM 역할을 만들려면

1. 디바이스에서 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 다음을 따릅니다.

   a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 들어 있는 파일을 생성하십시오.

      예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

      ```
      nano device-role-trust-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

      ```
      {
        "Version": "2012-10-17",
      ```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 신뢰 정책 문서를 사용하여 토큰 교환 역할을 생성합니다.

- *GreenGrassV2# TokenExchangeRole* 생성할 IAM 역할의 이름으로 바꾸십시오.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. 토큰 교환 역할에 필요한 액세스 정책 문서가 포함된 파일을 생성하십시오.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 이 액세스 정책은 S3 버킷의 구성 요소 아티팩트에 대한 액세스를 허용하지 않습니다. Amazon S3에 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 설명은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용 섹션을 참조하세요.
>
> 구성 요소 아티팩트를 위한 S3 버킷이 아직 없는 경우, 버킷을 생성한 후 나중에 이러한 권한을 추가할 수 있습니다.

d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreenGrassV2# TokenExchangeRoleAccess* 생성할 IAM 정책의 이름으로 바꾸십시오.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. IAM 정책을 토큰 교환 역할에 연결합니다.

- *GreenGrassV2# TokenExchangeRole IAM* 역할 이름으로 바꾸십시오.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 교체합니다.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

요청이 성공하면 명령이 출력되지 않습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

- 생성할 역할 별칭의 *GreengrassCoreTokenExchangeRoleAlias*이름으로 바꾸십시오.
- 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 교체합니다.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 역할 별칭을 생성하려면 토큰 교환 IAM 역할을 전달할 권한이 있어야 합니다. AWS IoT 역할 별칭을 만들려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 이 권한이 있는지 확인하세요. 자세한 내용은 사용 설명서의 AWS 서비스에 역할을 전달할 수 있는 권한 부여를 AWS Identity and Access Management 참조하십시오.

3. Greengrass 코어 디바이스가 역할 별칭을 사용하여 토큰 교환 역할을 맡도록 허용하는 AWS IoT 정책을 만들고 첨부하십시오. 이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 역할 별칭 AWS IoT 정책을 연결할 수 있습니다. 다음을 따릅니다.

   a. (선택 사항) 역할 별칭에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

      - 리소스 ARN을 역할 별칭의 ARN으로 대체합니다.

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
          {
      ```

```
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. AWS IoT 정책 문서에서 정책을 생성합니다.

- 생성할 AWS IoT 정책의 *GreengrassCoreTokenExchangeRoleAliasPolicy*이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. AWS IoT 정책을 사물의 인증서에 연결합니다. AWS IoT

- 역할 별칭 AWS IoT 정책의 *GreengrassCoreTokenExchangeRoleAliasPolicy*이름으로 바꾸십시오.
- 대상 ARN을 사내 인증서의 ARN으로 바꾸십시오. AWS IoT

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청이 성공하면 명령이 출력되지 않습니다.

인증서를 디바이스에 다운로드합니다.

이전에 디바이스의 인증서를 개발 컴퓨터에 다운로드했습니다. 이 섹션에서는 인증서를 코어 장치에 복사하여 연결에 사용하는 인증서로 장치를 설정합니다 AWS IoT. Amazon 루트 인증 기관 (CA) 인증서도 다운로드할 수 있습니다. HSM을 사용하는 경우 이 섹션의 HSM으로 인증서 파일도 가져올 수 있습니다.

- 이전에 AWS IoT 서비스에서 사물 인증서와 개인 키를 생성한 경우 단계에 따라 개인 키와 인증서 파일이 포함된 인증서를 다운로드하십시오.
- 이전에 하드웨어 보안 모듈 (HSM) 의 개인 키에서 사물 인증서를 생성한 경우 단계에 따라 HSM에 개인 키 및 인증서가 포함된 인증서를 다운로드합니다.

개인 키가 포함된 인증서 및 인증서 파일 다운로드

인증서를 디바이스에 다운로드하려면

1. 개발 컴퓨터의 AWS IoT 사물 인증서를 디바이스에 복사합니다. 개발 컴퓨터와 디바이스에서 SSH와 SCP가 활성화된 경우 개발 컴퓨터의 `scp` 명령을 사용하여 인증서를 전송할 수 있습니다. 디바이스의 IP 주소로 *device-ip-address*바꾸십시오.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중에 이 폴더에 AWS IoT Greengrass Core 소프트웨어를 설치하게 됩니다.

   Linux or Unix

   - 사용할 */greengrass/v2* 폴더로 바꾸십시오.

   ```
   sudo mkdir -p /greengrass/v2
   ```

Windows Command Prompt

- *C:\greengrass\v2* 를 사용할 폴더로 바꾸십시오.

```
mkdir C:\greengrass\v2
```

PowerShell

- *C:\greengrass\v2* 를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 폴더에 대한 권한을 설정합니다.

   - */greengrass# ## ###* 상위 폴더로 바꾸십시오.

```
sudo chmod 755 /greengrass
```

4. AWS IoT 사물 인증서를 Greengrass 루트 폴더에 복사합니다.

   Linux or Unix

   - Greengrass 루트 */greengrass/v2* 폴더로 교체합니다.

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

   Windows Command Prompt

   - *C:\greengrass\v2* 를 사용할 폴더로 바꾸십시오.

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

   PowerShell

   - *C:\greengrass\v2* 를 사용할 폴더로 바꿉니다.

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. Amazon 루트 인증 기관 (CA) 인증서를 다운로드하십시오. AWS IoT 인증서는 기본적으로 Amazon의 루트 CA 인증서와 연결됩니다.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:\greengrass\v2\\AmazonRootCA1.pem
   ```

개인 키와 인증서가 포함된 인증서를 HSM에 다운로드하십시오.

> **Note**
>
> 이 기능은 v2.5.3 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

인증서를 장치에 다운로드하려면

1. 개발 컴퓨터의 AWS IoT 사물 인증서를 디바이스에 복사합니다. 개발 컴퓨터와 디바이스에서 SSH와 SCP가 활성화된 경우 개발 컴퓨터의 `scp` 명령을 사용하여 인증서를 전송할 수 있습니다. 디바이스의 IP 주소로 *`device-ip-address`*바꾸십시오.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중에 이 폴더에 AWS IoT Greengrass Core 소프트웨어를 설치하게 됩니다.

   Linux or Unix

   - 사용할 */greengrass/v2* 폴더로 바꾸십시오.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - *C:\greengrass\v2* 를 사용할 폴더로 바꾸십시오.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - *C:\greengrass\v2* 를 사용할 폴더로 바꿉니다.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 폴더에 대한 권한을 설정합니다.

   - */greengrass# ## ###* 상위 폴더로 바꾸십시오.

   ```
   sudo chmod 755 /greengrass
   ```

4. 사물 인증서 파일을 ~/greengrass-v2-certs/device.pem.crt HSM으로 가져옵니다. 인증서를 가져오는 방법을 알아보려면 HSM 설명서를 참조하십시오. 이전에 HSM에서 개인 키를 생성한 것과 동일한 토큰, 슬롯 ID, 사용자 PIN, 개체 레이블, 개체 ID (HSM에서 사용하는 경우) 를 사용하여 인증서를 가져옵니다.

> **Note**
>
> 이전에 개체 ID 없이 개인 키를 생성했고 인증서에 개체 ID가 있는 경우 개인 키의 개체 ID를 인증서와 같은 값으로 설정하십시오. 개인 키 객체의 객체 ID를 설정하는 방법을 알아보려면 HSM 설명서를 참조하십시오.

5. (선택 사항) HSM에만 존재하도록 사물 인증서 파일을 삭제합니다.

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. Amazon 루트 인증 기관 (CA) 인증서를 다운로드하십시오. AWS IoT 인증서는 기본적으로 Amazon의 루트 CA 인증서와 연결됩니다.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:\greengrass\v2\\AmazonRootCA1.pem
   ```

디바이스 환경 설정

이 섹션의 단계에 따라 AWS IoT Greengrass 핵심 장치로 사용할 Linux 또는 Windows 장치를 설정합니다.

Linux 디바이스 설정

Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass Core 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 장치에 OpenJDK를 설치하는 방법을 보여줍니다.

   - Debian 기반 또는 Ubuntu 기반 배포판의 경우:

     ```
     sudo apt install default-jdk
     ```

   - Red Hat 기반 배포판의 경우:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - 대상 Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 아마존 리눅스 2023의 경우:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

   ```
   java -version
   ```

   이 명령은 장치에서 실행되는 Java 버전을 인쇄합니다. 예를 들어 Debian 기반 배포의 경우 출력은 다음 샘플과 비슷할 수 있습니다.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (선택 사항) 기기에서 구성 요소를 실행하는 기본 시스템 사용자 및 그룹을 생성합니다. 설치 중에 설치 프로그램 인수를 사용하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 이 사용자 및 그룹을 만들도록 선택할 수도 있습니다. `--component-default-user` 자세한 설명은 인스톨러 인수 섹션을 참조하세요.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. AWS IoT Greengrass Core 소프트웨어를 실행하는 사용자 (일반적으로root) 에게 모든 사용자 및 그룹과 sudo 함께 실행할 수 있는 권한이 있는지 확인하십시오.

   a. 다음 명령을 실행하여 /etc/sudoers 파일을 엽니다.

   ```
   sudo visudo
   ```

   b. 사용자의 권한이 다음 예와 같은지 확인합니다.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (선택 사항) 컨테이너화된 Lambda 함수를 실행하려면 cgroups v1을 활성화하고 메모리 및 디바이스 cgroups를 활성화하고 마운트해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없다면 이 단계를 건너뛰어도 됩니다.

   이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터를 사용하여 디바이스를 부팅하십시오.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   디바이스의 커널 파라미터 보기 및 설정에 대한 자세한 내용은 운영 체제 및 부트로더 설명서를 참조하십시오. 지침에 따라 커널 파라미터를 영구적으로 설정하십시오.

5. 의 요구 사항 목록에 나와 있는 대로 다른 모든 필수 종속 항목을 장치에 설치하십시오. 디바이스 요구 사항

Windows 디바이스 설정

> **Note**
>
> 이 기능은 v2.5.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass Core 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
2. PATH 시스템 변수에서 Java를 사용할 수 있는지 확인하고 사용할 수 없는 경우 추가하십시오. 이 LocalSystem 계정은 AWS IoT Greengrass Core 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 다음을 따릅니다.

   a. Windows 키를 눌러 시작 메뉴를 엽니다.

   b. 시작 메뉴에서 시스템 옵션을 **environment variables** 검색하려면 입력합니다.

   c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.

   d. 환경 변수 선택... 환경 변수 창을 열려면

   e. 시스템 변수에서 경로를 선택한 다음 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.

   f. Java 설치 `bin` 폴더 경로가 있는지 확인합니다. 경로는 다음 예와 비슷할 수 있습니다.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Java 설치 `bin` 폴더가 경로에 없는 경우 [새로 만들기] 를 선택하여 추가한 다음 [확인] 을 선택합니다.
3. Windows 명령 프롬프트 (`cmd.exe`) 를 관리자 권한으로 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. *### ## ###* 바꾸십시오.

```
net user /add ggc_user password
```

> **Tip**
>
> Windows 구성에 따라 사용자 암호가 향후 날짜에 만료되도록 설정될 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 업데이트하십시오. 사용자 비밀번호가 만료되지 않도록 설정할 수도 있습니다.
>
> - 사용자 및 암호가 언제 만료되는지 확인하려면 다음 명령을 실행합니다.
>
> ```
> net user ggc_user | findstr /C:expires
> ```

- 사용자 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 이 명령이 더 이상 사용되지 않는 Windows 10 이상을 사용하는 경우 wmic 다음 명령을 실행하세요. PowerShell

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Microsoft에서 PsExec유틸리티를 다운로드하여 장치에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. ### 이전에 설정한 사용자 암호로 바꾸십시오.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

가 PsExec License Agreement열리면 라이선스에 Accept동의하도록 선택하고 명령을 실행합니다.

> **Note**
>
> Windows 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 계정에 기본 사용자 정보를 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다.

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 다음 위치에서 특정 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다. ### 다운로드할 버전으로 교체하십시오.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

AWS IoT Greengrass Core 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 이름이 지정된 파일에 AWS IoT Greengrass Core 소프트웨어를 `greengrass-nucleus-latest.zip` 다운로드합니다.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.
2. (선택 사항) Greengrass 핵 소프트웨어 서명을 확인하려면

   > **Note**
   >
   > 이 기능은 Greengrass 핵 버전 2.9.5 이상에서 사용할 수 있습니다.

   a. 다음 명령을 사용하여 Greengrass 핵 아티팩트의 서명을 확인하십시오.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`호출 시 확인 결과를 나타내는 출력이 출력됩니다.

   i. Greengrass nucleus zip 파일에 서명된 경우 출력에는 다음 명령문이 포함됩니다.

   ```
   jar verified.
   ```

   ii. Greengrass nucleus zip 파일에 서명되지 않은 경우 출력에는 다음 명령문이 포함됩니다.

   ```
   jar is unsigned.
   ```

c. Jarsigner `-certs` 옵션을 `-verify -verbose` 옵션과 함께 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. AWS IoT Greengrass Core 소프트웨어를 디바이스의 폴더에 압축을 풉니다. 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (선택 사항) 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 Core 소프트웨어를 설치한 후에 이 폴더를 제거하지 마십시오. AWS IoT Greengrass AWS IoT Greengrass Core 소프트웨어는 이 폴더의 파일을 사용하여 실행합니다.
> 최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치해야 하며, AWS IoT Greengrass Core 소프트웨어를 설치한 후 이 폴더를 제거할 수 있습니다.

Core 소프트웨어를 설치합니다. AWS IoT Greengrass

다음 작업을 지정하는 인수를 사용하여 설치 프로그램을 실행합니다.

- 이전에 만든 AWS 리소스와 인증서를 사용하도록 지정하는 부분 구성 파일에서 설치합니다. AWS IoT Greengrass Core 소프트웨어는 장치에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 프로그램은 사용자가 제공한 부분 구성 파일을 사용하여 전체 구성 파일을 만듭니다.
- `ggc_user`시스템 사용자를 사용하여 코어 장치에서 소프트웨어 구성 요소를 실행하도록 지정합니다. Linux 장치에서 이 명령은 또한 `ggc_group` 시스템 그룹을 사용하도록 지정하며, 설치 프로그램은 시스템 사용자 및 그룹을 자동으로 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 장치에서는 Systemd init 시스템이 필요합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 을 참조하십시오인스톨러 인수.

> **Note**
>
> 메모리가 제한된 AWS IoT Greengrass 장치에서 실행하는 경우 AWS IoT Greengrass Core 소프트웨어가 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하려면 nucleus 구성 요소의 `jvmOptions` 구성 매개변수에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.

- 이전에 AWS IoT 서비스에서 사물 인증서와 개인 키를 생성한 경우 단계에 따라 개인 키와 인증서 파일을 사용하여 AWS IoT Greengrass Core 소프트웨어를 설치하십시오.
- 이전에 하드웨어 보안 모듈 (HSM) 의 개인 키에서 사물 인증서를 만든 경우 단계에 따라 개인 키와 인증서를 사용하여 HSM에 AWS IoT Greengrass Core 소프트웨어를 설치하십시오.

개인 키와 인증서 파일을 사용하여 AWS IoT Greengrass Core 소프트웨어를 설치합니다.

AWS IoT Greengrass Core 소프트웨어를 설치하려면

1. AWS IoT Greengrass Core 소프트웨어 버전을 확인하십시오.

   - 소프트웨어가 들어 있는 폴더의 *GreengrassInstaller*경로로 바꾸십시오.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 텍스트 편집기를 사용하여 설치 프로그램에 제공할 이름을 `config.yaml` 지정하는 구성 파일을 생성합니다.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 콘텐츠를 파일에 복사합니다. 이 부분 구성 파일은 시스템 매개변수와 Greengrass 핵 매개변수를 지정합니다.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.2"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
```

이어서 다음을 수행합니다.

- 의 각 인스턴스를 Greengrass 루트 */greengrass/v2* 폴더로 교체합니다.
- 사물의 *MyGreengrassCore*이름으로 바꾸십시오 AWS IoT .
- *2.12.2#* AWS IoT Greengrass Core 소프트웨어 버전으로 교체하십시오.
- *us-west-2# #### ##* AWS 리전 곳으로 바꾸십시오.
- 토큰 교환 *GreengrassCoreTokenExchangeRoleAlias*역할 별칭의 이름으로 바꾸십시오.
- 를 `iotDataEndpoint` AWS IoT 데이터 엔드포인트로 바꾸십시오.
- 를 `iotCredEndpoint` AWS IoT 자격 증명 엔드포인트로 바꾸십시오.

> **Note**
>
> 이 구성 파일에서 다음 예와 같이 사용할 포트 및 네트워크 프록시와 같은 다른 nucleus 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 Greengrass 핵 구성을 참조하십시오.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.2"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. 설치 프로그램을 실행하고 구성 파일을 `--init-config` 제공하도록 지정합니다.

   - */greengrass/v2*또는 *C:\greengrass\v2* 를 Greengrass 루트 폴더로 바꾸십시오.
   - 의 각 인스턴스를 설치 *GreengrassInstaller*프로그램의 압축을 푼 폴더로 바꾸십시오.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 `--setup-system-service true` 설정하도록 지정해야 합니다. AWS IoT Greengrass

지정한 `--setup-system-service true` 경우 설치 프로그램은 소프트웨어를 시스템 서비스로 설정하고 실행했는지 `Successfully set up Nucleus as a system service` 여부를 인쇄합니다. 그렇지 않으면 소프트웨어를 성공적으로 설치해도 설치 프로그램이 메시지를 출력하지 않습니다.

> **Note**
>
> `deploy-dev-tools`인수 없이 설치 프로그램을 실행할 때는 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. `--provision true` Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 을 참조하십시오. 그린그래스 커맨드 라인 인터페이스

4. 루트 폴더의 파일을 확인하여 설치를 확인합니다.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   설치에 성공하면 루트 폴더에는, `config`packages, 등의 여러 폴더가 포함됩니다`logs`.

개인 키와 인증서를 사용하여 HSM에 AWS IoT Greengrass Core 소프트웨어를 설치합니다.

> **Note**
>
> 이 기능은 v2.5.3 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

AWS IoT Greengrass Core 소프트웨어를 설치하려면

1. AWS IoT Greengrass Core 소프트웨어 버전을 확인하십시오.

   - 소프트웨어가 들어 있는 폴더의 *`GreengrassInstaller`*경로로 바꾸십시오.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. AWS IoT Greengrass Core 소프트웨어가 HSM의 개인 키와 인증서를 사용할 수 있게 하려면 Core 소프트웨어를 설치할 때 PKCS #11 공급자 구성 요소를 설치하십시오. AWS IoT Greengrass PKCS #11 공급자 구성 요소는 설치 중에 구성할 수 있는 플러그인입니다. 다음 위치에서 최신 버전의 PKCS #11 공급자 구성 요소를 다운로드할 수 있습니다.

   - https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar

   라는 이름의 파일에 PKCS #11 공급자 플러그인을 다운로드합니다. `aws.greengrass.crypto.Pkcs11Provider.jar` 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
   aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
   aws.greengrass.crypto.Pkcs11Provider.jar
   ```

   이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

3. 텍스트 편집기를 사용하여 설치 프로그램에 제공할 이름을 `config.yaml` 지정하는 구성 파일을 생성합니다.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   다음 YAML 콘텐츠를 파일에 복사합니다. 이 부분 구성 파일은 시스템 매개변수, Greengrass 핵 매개변수 및 PKCS #11 제공자 매개변수를 지정합니다.

   ```
   ---
   system:
     certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
     privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
   ```

```
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.2"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

이어서 다음을 수행합니다.

- PKCS #11 *URI# ## iotdevicekey#* 각 인스턴스를 개인 키를 생성하고 인증서를 가져온 개체 레이블로 교체합니다.
- 의 각 인스턴스를 Greengrass 루트 */greengrass/v2* 폴더로 교체합니다.
- 사물의 *MyGreengrassCore*이름으로 바꾸십시오 AWS IoT .
- *2.12.2#* AWS IoT Greengrass Core 소프트웨어 버전으로 교체하십시오.
- *us-west-2# #### ##* AWS 리전 곳으로 바꾸십시오.
- 토큰 교환 *GreengrassCoreTokenExchangeRoleAlias*역할 별칭의 이름으로 바꾸십시오.
- 를 `iotDataEndpoint` AWS IoT 데이터 엔드포인트로 바꾸십시오.
- 를 `iotCredEndpoint` AWS IoT 자격 증명 엔드포인트로 바꾸십시오.
- 구성 요소의 구성 매개변수를 코어 디바이스의 HSM 구성 값으로 교체합니다. `aws.greengrass.crypto.Pkcs11Provider`

> **Note**
>
> 이 구성 파일에서 다음 예와 같이 사용할 포트 및 네트워크 프록시와 같은 다른 nucleus 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 Greengrass 핵 구성을 참조하십시오.

```
---
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.2"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

4. 설치 프로그램을 실행하고 구성 파일을 `--init-config` 제공하도록 지정합니다.

   - Greengrass 루트 */greengrass/v2* 폴더로 교체합니다.
   - 의 각 인스턴스를 설치 *GreengrassInstaller*프로그램의 압축을 푼 폴더로 바꾸십시오.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
```

```
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 `--setup-system-service true` 설정하도록 지정해야 합니다. AWS IoT Greengrass

지정한 `--setup-system-service true` 경우 설치 프로그램은 소프트웨어를 시스템 서비스로 설정하고 실행했는지 `Successfully set up Nucleus as a system service` 여부를 인쇄합니다. 그렇지 않으면 소프트웨어를 성공적으로 설치해도 설치 프로그램이 메시지를 출력하지 않습니다.

> **Note**
>
> `deploy-dev-tools`인수 없이 설치 프로그램을 실행할 때는 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. `--provision true` Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 을 참조하십시오. 그린그래스 커맨드 라인 인터페이스

5. 루트 폴더의 파일을 확인하여 설치를 확인합니다.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   설치에 성공하면 루트 폴더에는, `config`, `packages`, 등의 여러 폴더가 포함됩니다`logs`.

AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설치한 경우 설치 프로그램이 소프트웨어를 자동으로 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 설명은 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.

소프트웨어 구성 및 사용 방법에 대한 자세한 내용은 AWS IoT Greengrass 다음을 참조하십시오.

- AWS IoT Greengrass Core 소프트웨어 구성
- AWS IoT Greengrass구성 요소 개발
- 디바이스에 AWS IoT Greengrass 구성 요소 배포
- 그린그래스 커맨드 라인 인터페이스

AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다.

AWS IoT 플릿 프로비저닝을 사용하면 X.509 디바이스 인증서와 프라이빗 키를 생성하여 디바이스에 처음 연결할 때 디바이스에 안전하게 AWS IoT 전달하도록 구성할 수 있습니다. AWS IoT AWS IoT Amazon 루트 인증 기관 (CA) 에서 서명한 클라이언트 인증서를 제공합니다. 플릿 프로비저닝으로 프로비저닝하는 Greengrass 코어 디바이스에 대한 사물 그룹, 사물 유형 및 권한을 AWS IoT 지정하도록 구성할 수도 있습니다. 프로비저닝 템플릿을 정의하여 각 디바이스를 프로비저닝하는 방법을 AWS IoT 정의합니다. 프로비저닝 템플릿은 프로비저닝 시 디바이스용으로 생성할 사물, 정책 및 인증서 리소스를 지정합니다. 자세한 내용은 개발자 안내서의 프로비전 템플릿을 참조하십시오. AWS IoT Core

AWS IoT Greengrass AWS IoT 플릿 프로비저닝으로 생성된 AWS 리소스를 사용하여 AWS IoT Greengrass Core 소프트웨어를 설치하는 데 사용할 수 있는 AWS IoT 플릿 프로비저닝 플러그인을 제공합니다. 플릿 프로비저닝 플러그인은 클레임을 통한 프로비저닝을 사용합니다. 기기는 프로비저닝 클레임 인증서와 개인 키를 사용하여 일반 작업에 사용할 수 있는 고유한 X.509 장치 인증서와 개인 키를 얻습니다. 제조 과정에서 각 장치에 클레임 인증서와 개인 키를 내장할 수 있으므로 고객이 나중에 각 장치가 온라인 상태가 될 때 장치를 활성화할 수 있습니다. 여러 장치에 동일한 클레임 인증서와 개인 키를 사용할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 클레임별 프로비저닝을 참조하십시오.

> **Note**
>
> 플릿 프로비저닝 플러그인은 현재 하드웨어 보안 모듈 (HSM) 에 개인 키와 인증서 파일을 저장하는 것을 지원하지 않습니다. HSM을 사용하려면 수동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치하세요.

AWS IoT 플릿 프로비저닝과 함께 AWS IoT Greengrass Core 소프트웨어를 설치하려면 Greengrass 코어 디바이스를 AWS 계정 프로비저닝하는 데 AWS IoT 사용되는 리소스를 설정해야 합니다. 이러한 리소스에는 프로비저닝 템플릿, 클레임 인증서, 토큰 교환 IAM 역할이 포함됩니다. 이러한 리소스를 생성한 후에는 이를 재사용하여 플릿의 여러 코어 디바이스를 프로비저닝할 수 있습니다. 자세한 설명은 Greengrass 코어 디바이스를 위한 AWS IoT 플릿 프로비저닝 설정 섹션을 참조하세요.

> **Important**
>
> Core 소프트웨어를 다운로드하기 전에 AWS IoT Greengrass 코어 장치가 Core 소프트웨어 v2.0을 설치하고 실행하기 위한 요구 사항을 충족하는지 확인하십시오. AWS IoT Greengrass

주제

필수 조건

AWS IoT 플릿 프로비저닝과 함께 AWS IoT Greengrass Core 소프트웨어를 설치하려면 먼저 Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝을 설정해야 합니다. 이 단계를 한 번 완료하면 플릿 프로비저닝을 사용하여 원하는 수의 디바이스에 AWS IoT Greengrass Core 소프트웨어를 설치할 수 있습니다.

엔드포인트 검색 AWS IoT

AWS IoT 엔드포인트를 가져와서 나중에 사용할 수 있도록 저장하세요 AWS 계정. 장치는 이러한 엔드포인트를 사용하여 연결합니다. AWS IoT 다음을 따릅니다.

1. 사용자의 AWS IoT 데이터 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 사용자의 AWS IoT 자격 증명 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

인증서를 디바이스에 다운로드합니다.

디바이스는 클레임 인증서와 개인 키를 사용하여 AWS 리소스 프로비저닝 요청을 인증하고 X.509 디바이스 인증서를 획득합니다. 제조 과정에서 클레임 인증서와 개인 키를 디바이스에 내장하거나 설치 중에 인증서와 키를 디바이스에 복사할 수 있습니다. 이 섹션에서는 클레임 인증서와 개인 키를 디바이스에 복사합니다. 또한 Amazon 루트 인증 기관 (CA) 인증서를 디바이스에 다운로드합니다.

> **Important**
>
> 프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스를 포함하여 항상 보호되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 청구 인증서의 무단 사용 등 오용의 징후가 있는지 모니터링하는 것이 좋습니다. 오용을 감지하면 프로비저닝 클레임 인증서를 비활성화하여 디바이스 프로비저닝에 사용할 수 없도록 하십시오. 자세한 내용을 알아보려면 AWS IoT Core 개발자 안내서의 AWS IoT 모니터링를 참조하세요.
> 디바이스 수와 등록되는 디바이스를 더 잘 관리할 수 있도록 플릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. AWS 계정 사전 프로비저닝 후크는 등록 시 디바이스가 제공하는 템플릿 파라미터의 유효성을 검사하는 AWS Lambda 함수입니다. 예를 들

어, 데이터베이스를 기준으로 기기 ID를 검사하여 기기에 프로비전 권한이 있는지 확인하는 사전 프로비저닝 후크를 만들 수 있습니다. 자세한 내용은 개발자 안내서의 사전 프로비저닝 후크를 참조하십시오. AWS IoT Core

클레임 인증서를 디바이스에 다운로드하려면

1. 클레임 인증서와 개인 키를 장치에 복사합니다. 개발 컴퓨터와 장치에서 SSH와 SCP가 활성화된 경우 개발 컴퓨터의 `scp` 명령을 사용하여 클레임 인증서와 개인 키를 전송할 수 있습니다. 다음 예제 명령은 이러한 파일을 개발 `claim-certs` 컴퓨터에 이름이 지정된 폴더를 장치로 전송합니다. 디바이스의 IP 주소로 *`device-ip-address`*바꾸십시오.

   ```
   scp -r claim-certs/ device-ip-address:~
   ```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중에 이 폴더에 AWS IoT Greengrass Core 소프트웨어를 설치하게 됩니다.

   Linux or Unix

   - 사용할 */greengrass/v2* 폴더로 바꾸십시오.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - *C:\greengrass\v2* 를 사용할 폴더로 바꾸십시오.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - *C:\greengrass\v2* 를 사용할 폴더로 바꿉니다.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 폴더에 대한 권한을 설정합니다.

- */greengrass# ## ###* 상위 폴더로 바꾸십시오.

```
sudo chmod 755 /greengrass
```

4. 클레임 인증서를 Greengrass 루트 폴더로 이동합니다.

   - */greengrass/v2*또는 *C:\greengrass\v2* 를 Greengrass 루트 폴더로 바꾸십시오.

   Linux or Unix

   ```
   sudo mv ~/claim-certs /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   move %USERPROFILE%\claim-certs C:\greengrass\v2
   ```

   PowerShell

   ```
   mv -Path ~\claim-certs -Destination C:\greengrass\v2
   ```

5. Amazon 루트 인증 기관 (CA) 인증서를 다운로드하십시오. AWS IoT 인증서는 기본적으로 Amazon의 루트 CA 인증서와 연결됩니다.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

디바이스 환경 설정

이 섹션의 단계에 따라 AWS IoT Greengrass 핵심 장치로 사용할 Linux 또는 Windows 장치를 설정합니다.

Linux 디바이스 설정

Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass Core 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 장치에 OpenJDK를 설치하는 방법을 보여줍니다.

   - Debian 기반 또는 Ubuntu 기반 배포판의 경우:

     ```
     sudo apt install default-jdk
     ```

   - Red Hat 기반 배포판의 경우:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - 대상 Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 아마존 리눅스 2023의 경우:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

   ```
   java -version
   ```

   이 명령은 장치에서 실행되는 Java 버전을 인쇄합니다. 예를 들어 Debian 기반 배포의 경우 출력은 다음 샘플과 비슷할 수 있습니다.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (선택 사항) 기기에서 구성 요소를 실행하는 기본 시스템 사용자 및 그룹을 생성합니다. 설치 중에 설치 프로그램 인수를 사용하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 이 사용자 및 그룹을 만들도록 선택할 수도 있습니다. `--component-default-user` 자세한 설명은 인스톨러 인수 섹션을 참조하세요.

   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```

3. AWS IoT Greengrass Core 소프트웨어를 실행하는 사용자 (일반적으로`root`) 에게 모든 사용자 및 그룹과 `sudo` 함께 실행할 수 있는 권한이 있는지 확인하십시오.

   a. 다음 명령을 실행하여 `/etc/sudoers` 파일을 엽니다.

      ```
      sudo visudo
      ```

   b. 사용자의 권한이 다음 예와 같은지 확인합니다.

      ```
      root    ALL=(ALL:ALL) ALL
      ```

4. (선택 사항) 컨테이너화된 Lambda 함수를 실행하려면 cgroups v1을 활성화하고 메모리 및 디바이스 cgroups를 활성화하고 마운트해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없다면 이 단계를 건너뛰어도 됩니다.

   이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터를 사용하여 디바이스를 부팅하십시오.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   디바이스의 커널 파라미터 보기 및 설정에 대한 자세한 내용은 운영 체제 및 부트로더 설명서를 참조하십시오. 지침에 따라 커널 파라미터를 영구적으로 설정하십시오.

5. 의 요구 사항 목록에 나와 있는 대로 다른 모든 필수 종속 항목을 장치에 설치하십시오. 디바이스 요구 사항

Windows 디바이스 설정

> **Note**
>
> 이 기능은 v2.5.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

Windows 디바이스를 설정하려면 다음을 수행하십시오. AWS IoT Greengrass V2

1. AWS IoT Greengrass Core 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
2. PATH 시스템 변수에서 Java를 사용할 수 있는지 확인하고 사용할 수 없는 경우 추가하십시오. 이 LocalSystem 계정은 AWS IoT Greengrass Core 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 다음을 따릅니다.

   a. Windows 키를 눌러 시작 메뉴를 엽니다.

   b. 시작 메뉴에서 시스템 옵션을 **environment variables** 검색하려면 입력합니다.

   c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.

   d. 환경 변수 선택... 환경 변수 창을 열려면

   e. 시스템 변수에서 경로를 선택한 다음 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.

   f. Java 설치 `bin` 폴더 경로가 있는지 확인합니다. 경로는 다음 예와 비슷할 수 있습니다.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Java 설치 `bin` 폴더가 경로에 없는 경우 [새로 만들기] 를 선택하여 추가한 다음 [확인] 을 선택합니다.
3. Windows 명령 프롬프트 (`cmd.exe`) 를 관리자 권한으로 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. *### ## ###* 바꾸십시오.

```
net user /add ggc_user password
```

> **Tip**
>
> Windows 구성에 따라 사용자 암호가 향후 날짜에 만료되도록 설정될 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 업데이트하십시오. 사용자 비밀번호가 만료되지 않도록 설정할 수도 있습니다.
>
> - 사용자 및 암호가 언제 만료되는지 확인하려면 다음 명령을 실행합니다.
>
> ```
> net user ggc_user | findstr /C:expires
> ```

- 사용자 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 이 명령이 더 이상 사용되지 않는 Windows 10 이상을 사용하는 경우 wmic 다음 명령을 실행하세요. PowerShell

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Microsoft에서 PsExec유틸리티를 다운로드하여 장치에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. ### 이전에 설정한 사용자 암호로 바꾸십시오.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

가 PsExec License Agreement열리면 라이선스에 Accept동의하도록 선택하고 명령을 실행합니다.

> **Note**
>
> Windows 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 계정에 기본 사용자 정보를 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다.

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 다음 위치에서 특정 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다. ### 다운로드할 버전으로 교체하십시오.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

AWS IoT Greengrass Core 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 이름이 지정된 파일에 AWS IoT Greengrass Core 소프트웨어를 `greengrass-nucleus-latest.zip` 다운로드합니다.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.
2. (선택 사항) Greengrass 핵 소프트웨어 서명을 확인하려면

   > **Note**
   >
   > 이 기능은 Greengrass 핵 버전 2.9.5 이상에서 사용할 수 있습니다.

   a. 다음 명령을 사용하여 Greengrass 핵 아티팩트의 서명을 확인하십시오.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`호출 시 확인 결과를 나타내는 출력이 출력됩니다.

   i. Greengrass nucleus zip 파일에 서명된 경우 출력에는 다음 명령문이 포함됩니다.

   ```
   jar verified.
   ```

   ii. Greengrass nucleus zip 파일에 서명되지 않은 경우 출력에는 다음 명령문이 포함됩니다.

   ```
   jar is unsigned.
   ```

c. Jarsigner `-certs` 옵션을 `-verify -verbose` 옵션과 함께 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. AWS IoT Greengrass Core 소프트웨어를 디바이스의 폴더에 압축을 풉니다. 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (선택 사항) 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 Core 소프트웨어를 설치한 후에 이 폴더를 제거하지 마십시오. AWS IoT Greengrass AWS IoT Greengrass Core 소프트웨어는 이 폴더의 파일을 사용하여 실행합니다.
최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치해야 하며, AWS IoT Greengrass Core 소프트웨어를 설치한 후 이 폴더를 제거할 수 있습니다.

AWS IoT 플릿 프로비저닝 플러그인 다운로드

다음 위치에서 AWS IoT 플릿 프로비저닝 플러그인의 최신 버전을 다운로드할 수 있습니다.

- https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass - /fleetprovisioningbyclaim-latest.jar FleetProvisioningByClaim

Note

다음 위치에서 AWS IoT 플릿 프로비저닝 플러그인의 특정 버전을 다운로드할 수 있습니다. *### ##### ####* 교체하십시오. 플릿 프로비저닝 플러그인의 각 버전에 대한 자세한 내용은 을 참조하십시오AWS IoT플릿 프로비저닝 플러그인 변경 로그.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

플릿 프로비저닝 플러그인은 오픈소스입니다. 소스 코드를 보려면 에서 AWS IoT 플릿 프로비저닝 플러그인을 참조하십시오. GitHub

AWS IoT 플릿 프로비저닝 플러그인을 다운로드하려면

- 디바이스에서 AWS IoT 플릿 프로비저닝 플러그인을 이름이 지정된 파일에 다운로드하세요. `aws.greengrass.FleetProvisioningByClaim.jar` 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  PowerShell

  ```
  iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -
  OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

AWS IoT Greengrass Core 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 프로그램을 실행합니다.

- 플릿 프로비저닝 플러그인을 사용하여 리소스를 프로비저닝하도록 지정하는 부분 구성 파일에서 설치합니다. AWS AWS IoT Greengrass Core 소프트웨어는 장치에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 프로그램은 사용자가 제공하는 부분 구성 파일과 플릿 프로비저닝 플러그인이 생성하는 AWS 리소스로부터 전체 구성 파일을 생성합니다.
- `ggc_user`시스템 사용자를 사용하여 코어 디바이스에서 소프트웨어 구성 요소를 실행하도록 지정합니다. Linux 장치에서 이 명령은 또한 `ggc_group` 시스템 그룹을 사용하도록 지정하며, 설치 프로그램은 시스템 사용자 및 그룹을 자동으로 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 장치에서는 Systemd init 시스템이 필요합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 을 참조하십시오인스톨러 인수.

> **Note**
>
> 메모리가 제한된 AWS IoT Greengrass 장치에서 실행하는 경우 AWS IoT Greengrass Core 소프트웨어가 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하려면 nucleus 구성 요소의 `jvmOptions` 구성 매개변수에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.

Core 소프트웨어를 설치하려면 AWS IoT Greengrass

1. AWS IoT Greengrass Core 소프트웨어 버전을 확인하십시오.

   - 소프트웨어가 들어 있는 폴더의 *`GreengrassInstaller`*경로로 바꾸십시오.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 텍스트 편집기를 사용하여 설치 프로그램에 제공할 이름을 `config.yaml` 지정하는 구성 파일을 생성합니다.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 콘텐츠를 파일에 복사합니다. 이 부분 구성 파일은 플릿 프로비저닝 플러그인의 파라미터를 지정합니다. 지정할 수 있는 옵션에 대한 자세한 내용은 을 참조하십시오AWS IoT플릿 프로비저닝 플러그인 구성.

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.2"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.2"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
```

```
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

이어서 다음을 수행합니다.

- *2.12.2#* AWS IoT Greengrass Core 소프트웨어 버전으로 교체하십시오.
- */greengrass/v2*또는 *C:\greengrass\v2* 의 각 인스턴스를 Greengrass 루트 폴더로 바꿉니다.

  > **Note**
  >
  > Windows 디바이스에서는 경로 구분자를 이중 백슬래시 () 로 지정해야 합니다 (예:)\\. C:\\greengrass\\v2

- *us-west-2# ##### ###* 및 기타 리소스를 만든 AWS 지역으로 바꾸세요.
- 를 데이터 `iotDataEndpoint` 엔드포인트로 교체하세요. AWS IoT
- 를 `iotCredentialEndpoint` AWS IoT 자격 증명 엔드포인트로 바꾸십시오.
- 토큰 교환 역할 별칭의 *GreengrassCoreTokenExchangeRoleAlias*이름으로 바꾸십시오.
- 플릿 프로비저닝 템플릿의 *GreengrassFleetProvisioningTemplate*이름으로 바꾸십시오.
- 를 디바이스의 클레임 인증서 `claimCertificatePath` 경로로 바꾸십시오.
- 를 디바이스의 클레임 인증서 개인 키 `claimCertificatePrivateKeyPath` 경로로 바꾸십시오.
- 템플릿 매개 변수 (`templateParameters`) 를 장치를 프로비전하는 데 사용할 값으로 바꾸십시오. 이 예제에서는 `ThingGroupName` 매개 변수를 정의하는 ThingName예제 템플릿을 참조합니다.

Note

이 구성 파일에서 다음 예와 같이 사용할 포트 및 네트워크 프록시와 같은 다른 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 Greengrass 핵 구성을 참조하십시오.

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.2"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
```

```
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.2"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\
\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

HTTPS 프록시를 사용하려면 플릿 프로비저닝 플러그인 버전 1.1.0 이상을 사용해야 합니다. 다음 예와 같이 `rootCaPath` `system` 언더를 추가로 지정해야 합니다.

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. 설치 관리자를 실행합니다. 플릿 프로비저닝 플러그인을 `--trusted-plugin` 제공하도록 지정하고, 구성 파일을 `--init-config` 제공하도록 지정합니다.

   - Greengrass 루트 */greengrass/v2* 폴더로 교체합니다.
   - 의 각 인스턴스를 설치 *GreengrassInstaller*프로그램의 압축을 푼 폴더로 바꾸십시오.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
   ```

```
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 `--setup-system-service true` 설정하도록 지정해야 합니다. AWS IoT Greengrass

지정한 `--setup-system-service true` 경우 설치 프로그램은 소프트웨어를 시스템 서비스로 설정하고 실행했는지 `Successfully set up Nucleus as a system service` 여부를 인쇄합니다. 그렇지 않으면 소프트웨어를 성공적으로 설치해도 설치 프로그램이 메시지를 출력하지 않습니다.

> **Note**
>
> `deploy-dev-tools`인수 없이 설치 프로그램을 실행할 때는 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. `--provision true` Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 을 참조하십시오. 그린그래스 커맨드 라인 인터페이스

4. 루트 폴더의 파일을 확인하여 설치를 확인합니다.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

설치에 성공하면 루트 폴더에는, `config`, `packages`, 등의 여러 폴더가 포함됩니다`logs`.

AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설치한 경우 설치 프로그램이 소프트웨어를 자동으로 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 설명은 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.

소프트웨어 구성 및 사용 방법에 대한 자세한 내용은 AWS IoT Greengrass 다음을 참조하십시오.

- AWS IoT Greengrass Core 소프트웨어 구성
- AWS IoT Greengrass구성 요소 개발
- 디바이스에 AWS IoT Greengrass 구성 요소 배포
- 그린그래스 커맨드 라인 인터페이스

Greengrass 코어 디바이스를 위한 AWS IoT 플릿 프로비저닝 설정

플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 먼저 컴퓨터에 다음 리소스를 설정해야 합니다. AWS 계정 이러한 리소스를 통해 디바이스는 Greengrass 코어 디바이스에 AWS IoT 등록되고 Greengrass 코어 디바이스로 작동할 수 있습니다. 이 섹션의 단계를 한 번 따라 해당 리소스를 만들고 구성하십시오. AWS 계정

- 코어 디바이스가 서비스 호출을 승인하는 데 사용하는 토큰 교환 IAM 역할. AWS
- 토큰 AWS IoT 교환 역할을 가리키는 역할 별칭입니다.

- (선택 사항) 핵심 장치가 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 호출을 승인하는 데 사용하는 AWS IoT 정책입니다. 이 AWS IoT 정책은 토큰 교환 역할을 가리키는 AWS IoT 역할 별칭에 대한 `iot:AssumeRoleWithCertificate` 권한을 허용해야 합니다.

  플릿의 모든 코어 디바이스에 대해 단일 AWS IoT 정책을 사용하거나 플릿 프로비저닝 템플릿을 구성하여 각 코어 디바이스에 대한 AWS IoT 정책을 생성할 수 있습니다.
- AWS IoT플릿 프로비저닝 템플릿. 이 템플릿은 다음을 지정해야 합니다.
  - AWS IoT사물 리소스. 기존 사물 그룹 목록을 지정하여 온라인 상태가 되면 각 장치에 구성 요소를 배포할 수 있습니다.
  - AWS IoT정책 리소스. 이 리소스는 다음 속성 중 하나를 정의할 수 있습니다.
    - 기존 AWS IoT 정책의 이름. 이 옵션을 선택하면 이 템플릿에서 생성한 코어 디바이스가 동일한 AWS IoT 정책을 사용하며 해당 권한을 플릿으로 관리할 수 있습니다.
    - AWS IoT정책 문서. 이 옵션을 선택하면 이 템플릿으로 만든 각 코어 장치는 고유한 AWS IoT 정책을 사용하며 각 개별 코어 장치에 대한 권한을 관리할 수 있습니다.
  - AWS IoT인증서 리소스. 이 인증서 리소스는 `AWS::IoT::Certificate::Id` 매개변수를 사용하여 인증서를 코어 장치에 연결해야 합니다. 자세한 내용은 AWS IoT개발자 안내서의 J ust-in-time 프로비저닝을 참조하십시오.
- 플릿 AWS IoT 프로비저닝 템플릿의 프로비저닝 클레임 인증서 및 개인 키. 제조 과정에서 이 인증서와 개인 키를 디바이스에 내장할 수 있으므로 디바이스가 온라인 상태가 되면 스스로 등록하고 프로비저닝할 수 있습니다.

> **Important**
>
> 프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스를 포함하여 항상 보호되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 청구 인증서의 무단 사용 등 오용의 징후가 있는지 모니터링하는 것이 좋습니다. 오용을 감지하면 프로비저닝 클레임 인증서를 비활성화하여 디바이스 프로비저닝에 사용할 수 없도록 하십시오. 자세한 내용을 알아보려면 AWS IoT Core 개발자 안내서의 AWS IoT 모니터링를 참조하세요.
>
> 디바이스 수와 등록되는 디바이스를 더 잘 관리할 수 있도록 플릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. AWS 계정 사전 프로비저닝 후크는 등록 시 디바이스가 제공하는 템플릿 파라미터의 유효성을 검사하는 AWS Lambda 함수입니다. 예를 들어, 데이터베이스를 기준으로 기기 ID를 검사하여 기기에 프로비전 권한이 있는지 확인하는 사전 프로비저닝 후크를 만들 수 있습니다. 자세한 내용은 개발자 안내서의 사전 프로비저닝 후크를 참조하십시오. AWS IoT Core

- 디바이스가 플릿 프로비저닝 템플릿을 등록하고 사용할 수 있도록 하기 위해 프로비저닝 클레임 인증서에 연결하는 AWS IoT 정책입니다.

주제

- 토큰 교환 역할 생성
- AWS IoT 정책 생성
- 플릿 프로비저닝 템플릿을 생성하세요.
- 프로비저닝 클레임 인증서 및 개인 키를 생성합니다.

토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라고 하는 IAM 서비스 역할을 사용하여 서비스 호출을 승인합니다. AWS 디바이스는 AWS IoT 자격 증명 공급자를 사용하여 이 역할에 대한 임시 AWS 자격 증명을 가져오고, 이를 통해 디바이스가 Amazon Logs와 상호 작용하고AWS IoT, Amazon Logs에 로그를 전송하고, Amazon CloudWatch S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

AWS IoT역할 별칭을 사용하여 Greengrass 코어 디바이스의 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 장치의 토큰 교환 역할을 변경하면서도 장치 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core개발자 안내서의 AWS서비스에 대한 다이렉트 콜 승인을 참조하십시오.

이 섹션에서는 토큰 교환 IAM 역할과 해당 역할을 가리키는 AWS IoT 역할 별칭을 생성합니다. Greengrass 코어 디바이스를 이미 설정한 경우 새 디바이스를 생성하는 대신 토큰 교환 역할 및 역할 별칭을 사용할 수 있습니다.

토큰 교환 IAM 역할을 만들려면

1. 디바이스에서 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 다음을 따릅니다.

   a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 들어 있는 파일을 생성하십시오.

      예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

      ```
      nano device-role-trust-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 신뢰 정책 문서를 사용하여 토큰 교환 역할을 생성합니다.

- *GreenGrassV2# TokenExchangeRole* 생성할 IAM 역할의 이름으로 바꾸십시오.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
```

```
  }
```

c. 토큰 교환 역할에 필요한 액세스 정책 문서가 포함된 파일을 생성하십시오.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 이 액세스 정책은 S3 버킷의 구성 요소 아티팩트에 대한 액세스를 허용하지 않습니다. Amazon S3에 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 설명은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용 섹션을 참조하세요.
>
> 구성 요소 아티팩트를 위한 S3 버킷이 아직 없는 경우, 버킷을 생성한 후 나중에 이러한 권한을 추가할 수 있습니다.

d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreenGrassV2# TokenExchangeRoleAccess* 생성할 IAM 정책의 이름으로 바꾸십시오.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. IAM 정책을 토큰 교환 역할에 연결합니다.

- *GreenGrassV2# TokenExchangeRole IAM* 역할 이름으로 바꾸십시오.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 교체합니다.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

요청이 성공하면 명령이 출력되지 않습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

- 생성할 역할 별칭의 *GreengrassCoreTokenExchangeRoleAlias*이름으로 바꾸십시오.
- 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 교체합니다.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 역할 별칭을 생성하려면 토큰 교환 IAM 역할을 전달할 권한이 있어야 합니다. AWS IoT 역할 별칭을 만들려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 이 권한이 있는지 확인하세요. 자세한 내용은 사용 설명서의 AWS 서비스에 역할을 전달할 수 있는 권한 부여를 AWS Identity and Access Management 참조하십시오.

AWS IoT 정책 생성

장치를 사물로 등록한 후 해당 장치는 디지털 인증서를 사용하여 인증할 수 있습니다. AWS IoT AWS 이 인증서에는 장치가 인증서와 함께 사용할 수 있는 권한을 정의하는 하나 이상의 AWS IoT 정책이 포함되어 있습니다. 이러한 정책을 통해 장치는 AWS IoT 및 와 통신할 수 AWS IoT Greengrass 있습니다.

AWS IoT플릿 프로비저닝을 사용하면 디바이스를 AWS IoT 연결하여 디바이스 인증서를 생성하고 다운로드합니다. 다음 섹션에서 생성하는 플릿 프로비저닝 템플릿에서 모든 디바이스의 인증서에 동일한 AWS IoT 정책을 AWS IoT 연결할지 아니면 각 디바이스에 대해 새 정책을 생성할지 지정할 수 있습니다.

이 섹션에서는 모든 기기의 인증서에 AWS IoT 연결하는 AWS IoT 정책을 생성합니다. 이 접근 방식을 사용하면 모든 디바이스의 권한을 플릿으로 관리할 수 있습니다. 각 디바이스에 대해 새 AWS IoT 정책을 생성하려는 경우 이 섹션을 건너뛰고 플릿 템플릿을 정의할 때 해당 섹션의 정책을 참조할 수 있습니다.

AWS IoT 정책 생성

- Greengrass 코어 디바이스 플릿에 대한 AWS IoT 권한을 정의하는 AWS IoT 정책을 생성하십시오. 다음 정책은 모든 MQTT 주제 및 Greengrass 작업에 대한 액세스를 허용하므로 장치가 사용자 지정 애플리케이션 및 새로운 Greengrass 작업이 필요한 향후 변경 사항과 함께 작동하도록 할 수 있습니다. 이 정책은 또한 `iot:AssumeRoleWithCertificate` 권한을 허용하여 이전 섹션에서 생성한 토큰 교환 역할을 장치가 사용할 수 있도록 합니다. 사용 사례에 따라 이 정책을 제한할 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

  다음을 따릅니다.

  a. Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

     예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

     ```
     nano greengrass-v2-iot-policy.json
     ```

     다음 JSON을 파일에 복사합니다.

     - `iot:AssumeRoleWithCertificate`리소스를 이전 섹션에서 생성한 AWS IoT 역할 별칭의 ARN으로 교체합니다.

     ```
     {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Action": [
             "iot:Publish",
             "iot:Subscribe",
             "iot:Receive",
             "iot:Connect",
             "greengrass:*"
           ],
           "Resource": [
             "*"
           ]
         },
     ```

```
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. AWS IoT정책 문서에서 정책을 생성합니다.

- *GreenGrassV2IoT# ThingPolicy* 생성할 정책의 이름으로 바꾸십시오.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
```

```
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

플릿 프로비저닝 템플릿을 생성하세요.

AWS IoT플릿 프로비저닝 템플릿은 AWS IoT 사물, 정책 및 인증서를 프로비저닝하는 방법을 정의합니다. 플릿 프로비저닝 플러그인으로 Greengrass 코어 디바이스를 프로비저닝하려면 다음을 지정하는 템플릿을 생성해야 합니다.

- AWS IoT사물 리소스. 기존 사물 그룹 목록을 지정하여 온라인 상태가 되면 각 장치에 구성 요소를 배포할 수 있습니다.
- AWS IoT정책 리소스. 이 리소스는 다음 속성 중 하나를 정의할 수 있습니다.
  - 기존 AWS IoT 정책의 이름. 이 옵션을 선택하면 이 템플릿에서 생성한 코어 디바이스가 동일한 AWS IoT 정책을 사용하며 해당 권한을 플릿으로 관리할 수 있습니다.
  - AWS IoT정책 문서. 이 옵션을 선택하면 이 템플릿으로 만든 각 코어 장치는 고유한 AWS IoT 정책을 사용하며 각 개별 코어 장치에 대한 권한을 관리할 수 있습니다.
- AWS IoT인증서 리소스. 이 인증서 리소스는 `AWS::IoT::Certificate::Id` 매개변수를 사용하여 인증서를 코어 장치에 연결해야 합니다. 자세한 내용은 AWS IoT개발자 안내서의 J ust-in-time 프로비저닝을 참조하십시오.

템플릿에서 기존 AWS IoT 사물 그룹 목록에 사물을 추가하도록 지정할 수 있습니다. 코어 디바이스가 AWS IoT Greengrass 처음으로 연결되면 구성원인 각 사물 그룹에 대해 Greengrass 배포를 수신합니다. 사물 그룹을 사용하면 온라인 상태가 되는 즉시 각 장치에 최신 소프트웨어를 배포할 수 있습니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

AWS IoT서비스를 이용하려면 디바이스를 프로비저닝할 AWS 계정 때 AWS IoT 리소스를 생성하고 업데이트할 수 있는 권한이 필요합니다. AWS IoT서비스에 액세스 권한을 부여하려면 IAM 역할을 생성하고 템플릿을 생성할 때 제공해야 합니다. AWS IoT디바이스를 프로비저닝할 때 사용할 AWS IoT 수 있는 모든 권한에 대한 액세스를 허용하는 관리형 정책을 제공합니다. AWSIoTThingsRegistration 이 관리형 정책을 사용하거나 사용 사례에 맞게 관리형 정책의 권한 범위를 좁히는 사용자 지정 정책을 만들 수 있습니다.

이 섹션에서는 디바이스용 리소스를 AWS IoT 프로비저닝할 수 있는 IAM 역할을 생성하고 해당 IAM 역할을 사용하는 플릿 프로비저닝 템플릿을 생성합니다.

플릿 프로비저닝 템플릿을 만들려면

1. 내 리소스를 프로비저닝하는 역할을 맡을 AWS IoT 수 있는 IAM 역할을 생성하십시오. AWS 계정 다음을 따릅니다.

   a. 역할을 AWS IoT 수임할 수 있는 신뢰 정책 문서가 들어 있는 파일을 생성하십시오.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

      ```
      nano aws-iot-trust-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. 신뢰 정책 문서로 IAM 역할을 생성합니다.

      - 생성할 IAM 역할의 *GreengrassFleetProvisioningRole*이름으로 바꾸십시오.

      ```
      aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-policy-document file://aws-iot-trust-policy.json
      ```

      요청이 성공하면 응답은 다음 예와 비슷합니다.

      ```
      {
      ```

```
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassFleetProvisioningRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassFleetProvisioningRole",
    "CreateDate": "2021-07-26T00:15:12+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

c. 기기를 프로비저닝할 때 사용할 AWS IoT 수 있는 모든 권한에 대한 액세스를 허용하는 AWSIoTThingsRegistration정책을 검토하세요. 이 관리형 정책을 사용하거나 사용 사례에 맞게 범위 축소된 권한을 정의하는 사용자 지정 정책을 만들 수 있습니다. 사용자 지정 정책을 생성하기로 결정했다면 지금 생성하세요.

d. IAM 정책을 플릿 프로비저닝 역할에 연결합니다.

- *GreengrassFleetProvisioningRole*을 IAM 역할의 이름으로 바꿉니다.
- 이전 단계에서 사용자 지정 정책을 생성한 경우 정책 ARN을 사용할 IAM 정책의 ARN으로 교체하십시오.

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

요청이 성공하면 명령이 출력되지 않습니다.

2. (선택 사항) 사전 프로비저닝 후크를 만드세요. 사전 프로비저닝 후크는 등록 중에 기기가 제공하는 템플릿 매개변수의 유효성을 검사하는 AWS Lambda 기능입니다. 사전 프로비저닝 후크를 사용하면 어떤 디바이스를 얼마나 많이 온보딩할지 더 세밀하게 제어할 수 있습니다. AWS 계정 자세한 내용은 개발자 안내서의 사전 프로비저닝 후크를 참조하십시오. AWS IoT Core

3. 플릿 프로비저닝 템플릿을 만드세요. 다음을 따릅니다.

   a. 프로비저닝 템플릿 문서를 포함할 파일을 생성하십시오.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

      ```
      nano greengrass-fleet-provisioning-template.json
      ```

      프로비저닝 템플릿 문서를 작성하세요. 다음 예제 프로비저닝 템플릿에서 시작할 수 있습니다. 이 템플릿은 다음과 같은 속성을 가진 AWS IoT 사물을 생성하도록 지정합니다.

      - 사물의 이름은 `ThingName` 템플릿 파라미터에 지정하는 값입니다.
      - 사물은 `ThingGroupName` 템플릿 매개변수에 지정하는 사물 그룹의 멤버입니다. 사물 그룹은 사용자 내에 있어야 합니다AWS 계정.
      - 사물의 인증서에는 이름이 지정된 AWS IoT 정책이 `GreengrassV2IoTThingPolicy` 첨부되어 있습니다.

      자세한 내용은 AWS IoT Core개발자 안내서의 프로비저닝 템플릿을 참조하십시오.

      ```
      {
        "Parameters": {
          "ThingName": {
            "Type": "String"
          },
          "ThingGroupName": {
            "Type": "String"
          },
          "AWS::IoT::Certificate::Id": {
            "Type": "String"
          }
        },
        "Resources": {
          "MyThing": {
            "OverrideSettings": {
              "AttributePayload": "REPLACE",
              "ThingGroups": "REPLACE",
              "ThingTypeName": "REPLACE"
            },
            "Properties": {
      ```

```
        "AttributePayload": {},
        "ThingGroups": [
          {
            "Ref": "ThingGroupName"
          }
        ],
        "ThingName": {
          "Ref": "ThingName"
        }
      },
      "Type": "AWS::IoT::Thing"
    },
    "MyPolicy": {
      "Properties": {
        "PolicyName": "GreengrassV2IoTThingPolicy"
      },
      "Type": "AWS::IoT::Policy"
    },
    "MyCertificate": {
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "Active"
      },
      "Type": "AWS::IoT::Certificate"
    }
  }
}
```

> **Note**
>
> *MyThingMyPolicy*, 및 *MyCertificate*는 플릿 프로비저닝 템플릿의 각 리소스 사양을 식별하는 임의의 이름입니다. AWS IoT템플릿에서 생성하는 리소스에는 이러한 이름을 사용하지 않습니다. 이러한 이름을 사용하거나 템플릿에서 각 리소스를 식별하는 데 도움이 되는 값으로 바꿀 수 있습니다.

b. 프로비저닝 템플릿 문서에서 플릿 프로비저닝 템플릿을 생성합니다.

- 생성할 템플릿의 *GreengrassFleetProvisioningTemplate*이름으로 바꾸십시오.
- 템플릿 설명을 템플릿에 대한 설명으로 바꾸십시오.

- 프로비저닝 역할 ARN을 이전에 만든 역할의 ARN으로 교체합니다.

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

> **Note**
>
> 사전 프로비저닝 후크를 생성한 경우 인수를 사용하여 사전 프로비저닝 후크의 Lambda 함수의 ARN을 지정하십시오. `--pre-provisioning-hook`

```
--pre-provisioning-hook targetArn=arn:aws:lambda:us-
west-2:123456789012:function:GreengrassPreProvisioningHook
```

요청이 성공하면 응답은 다음 예제와 비슷합니다.

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

프로비저닝 클레임 인증서 및 개인 키를 생성합니다.

클레임 인증서는 디바이스를 AWS IoT 사물로 등록하고 일반적인 작업에 사용할 고유한 X.509 디바이스 인증서를 검색할 수 있는 X.509 인증서입니다. 클레임 인증서를 생성한 후에는 디바이스에서 이를 사용하여 고유한 디바이스 인증서를 생성하고 플릿 프로비저닝 템플릿으로 프로비저닝할 수 있도록 허용하는 AWS IoT 정책을 연결합니다. 클레임 인증서가 있는 장치는 정책에서 허용하는 프로비저닝 템플릿만 사용하여 프로비저닝할 수 있습니다. AWS IoT

이 섹션에서는 클레임 인증서를 생성하고 이전 섹션에서 만든 플릿 프로비저닝 템플릿과 함께 사용할 디바이스에 맞게 구성합니다.

Important

프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스를 포함하여 항상 보호되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 청구 인증서의 무단 사용 등 오용의 징후가 있는지 모니터링하는 것이 좋습니다. 오용을 감지하면 프로비저닝 클레임 인증서를 비활성화하여 디바이스 프로비저닝에 사용할 수 없도록 하십시오. 자세한 내용을 알아보려면 AWS IoT Core 개발자 안내서의 AWS IoT 모니터링를 참조하세요.
디바이스 수와 등록되는 디바이스를 더 잘 관리할 수 있도록 플릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. AWS 계정 사전 프로비저닝 후크는 등록 시 디바이스가 제공하는 템플릿 파라미터의 유효성을 검사하는 AWS Lambda 함수입니다. 예를 들어, 데이터베이스를 기준으로 기기 ID를 검사하여 기기에 프로비전 권한이 있는지 확인하는 사

전 프로비저닝 후크를 만들 수 있습니다. 자세한 내용은 개발자 안내서의 사전 프로비저닝 후크를 참조하십시오. AWS IoT Core

프로비저닝 클레임 인증서 및 개인 키를 만들려면

1. 클레임 인증서와 개인 키를 다운로드할 폴더를 만드십시오.

```
mkdir claim-certs
```

2. 프로비저닝에 사용할 인증서와 개인 키를 만들고 저장합니다. AWS IoTAmazon 루트 인증 기관(CA) 에서 서명한 클라이언트 인증서를 제공합니다.

   Linux or Unix

```
aws iot create-keys-and-certificate \
  --certificate-pem-outfile "claim-certs/claim.pem.crt" \
  --public-key-outfile "claim-certs/claim.public.pem.key" \
  --private-key-outfile "claim-certs/claim.private.pem.key" \
  --set-as-active
```

   Windows Command Prompt (CMD)

```
aws iot create-keys-and-certificate ^
  --certificate-pem-outfile "claim-certs/claim.pem.crt" ^
  --public-key-outfile "claim-certs/claim.public.pem.key" ^
  --private-key-outfile "claim-certs/claim.private.pem.key" ^
  --set-as-active
```

   PowerShell

```
aws iot create-keys-and-certificate `
  --certificate-pem-outfile "claim-certs/claim.pem.crt" `
  --public-key-outfile "claim-certs/claim.public.pem.key" `
  --private-key-outfile "claim-certs/claim.private.pem.key" `
  --set-as-active
```

   응답에는 요청이 성공할 경우 인증서에 대한 정보가 포함됩니다. 나중에 사용할 수 있도록 인증서의 ARN을 저장합니다.

3. 디바이스가 인증서를 사용하여 고유한 디바이스 인증서를 생성하고 플릿 프로비저닝 템플릿으로 프로비저닝할 수 있도록 허용하는 AWS IoT 정책을 생성하고 첨부하십시오. 다음 정책은 기기 프로비저닝 MQTT API에 대한 액세스를 허용합니다. 자세한 내용은 개발자 안내서의 디바이스 프로비저닝 MQTT API를 참조하십시오. AWS IoT Core

   다음을 따릅니다.

   a. Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

      ```
      nano greengrass-provisioning-claim-iot-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

      - *###* 각 인스턴스를 플릿 프로비저닝을 설정한 AWS 리전 곳으로 바꾸십시오.
      - *account-id# # ##### ID#* 바꾸십시오. AWS 계정
      - 의 각 인스턴스를 이전 섹션에서 생성한 플릿 프로비저닝 템플릿의 이름으로 *GreengrassFleetProvisioningTemplate*바꾸십시오.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "*"
          },
          {
            "Effect": "Allow",
            "Action": [
              "iot:Publish",
              "iot:Receive"
            ],
            "Resource": [
              "arn:aws:iot:region:account-id:topic/$aws/certificates/create/*",
              "arn:aws:iot:region:account-id:topic/$aws/provisioning-
      templates/GreengrassFleetProvisioningTemplate/provision/*"
            ]
      ```

```
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

b. AWS IoT정책 문서에서 정책을 생성하십시오.

- 생성할 정책의 *GreengrassProvisioningClaimPolicy*이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topic/$aws/certificates/create/*\",
```

```
          \"arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/
*\",
          \"arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

4. AWS IoT정책을 프로비저닝 클레임 인증서에 연결합니다.

   - 첨부할 정책의 *GreengrassProvisioningClaimPolicy*이름으로 바꾸십시오.
   - 대상 ARN을 프로비저닝 클레임 인증서의 ARN으로 교체합니다.

   ```
   aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
   target arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   요청이 성공하면 명령이 출력되지 않습니다.

이제 디바이스가 Greengrass 코어 디바이스에 등록하고 스스로를 프로비저닝하는 데 사용할 수 있는 AWS IoT 프로비저닝 클레임 인증서와 개인 키가 생겼습니다. 제조 과정에서 클레임 인증서와 개인 키를 장치에 내장하거나 Core 소프트웨어를 설치하기 전에 인증서와 키를 장치에 복사할 수 있습니다. AWS IoT Greengrass 자세한 내용은 AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치을(를) 참조하세요.

AWS IoT플릿 프로비저닝 플러그인 구성

AWS IoT플릿 프로비저닝 플러그인은 플릿 프로비저닝으로 AWS IoT GreengrassCore 소프트웨어를 설치할 때 사용자 지정할 수 있는 다음과 같은 구성 매개변수를 제공합니다.

`rootPath`

AWS IoT GreengrassCore 소프트웨어의 루트로 사용할 폴더의 경로입니다.

`awsRegion`

플릿 프로비저닝 플러그인이 AWS 리소스를 프로비저닝하는 데 사용하는 파일입니다. AWS 리전

`iotDataEndpoint`

사용자의 AWS IoT AWS 계정 데이터 엔드포인트.

`iotCredentialEndpoint`

사용자의 AWS IoT 자격 증명 엔드포인트AWS 계정.

`iotRoleAlias`

토큰 교환 IAM 역할을 가리키는 AWS IoT 역할 별칭입니다. AWS IoT자격 증명 공급자는 이 역할을 맡아 Greengrass 코어 디바이스가 서비스와 상호 작용할 AWS 수 있도록 합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

`provisioningTemplate`

리소스를 프로비저닝하는 데 사용할 AWS IoT 플릿 프로비저닝 템플릿입니다. AWS 이 템플릿은 다음을 지정해야 합니다.

- AWS IoT사물 리소스. 기존 사물 그룹 목록을 지정하여 온라인 상태가 되면 각 장치에 구성 요소를 배포할 수 있습니다.
- AWS IoT정책 리소스. 이 리소스는 다음 속성 중 하나를 정의할 수 있습니다.
  - 기존 AWS IoT 정책의 이름. 이 옵션을 선택하면 이 템플릿에서 생성한 코어 디바이스가 동일한 AWS IoT 정책을 사용하며 해당 권한을 플릿으로 관리할 수 있습니다.
  - AWS IoT정책 문서. 이 옵션을 선택하면 이 템플릿으로 만든 각 코어 장치는 고유한 AWS IoT 정책을 사용하며 각 개별 코어 장치에 대한 권한을 관리할 수 있습니다.
- AWS IoT인증서 리소스. 이 인증서 리소스는 `AWS::IoT::Certificate::Id` 매개변수를 사용하여 인증서를 코어 장치에 연결해야 합니다. 자세한 내용은 AWS IoT개발자 안내서의 J ust-in-time 프로비저닝을 참조하십시오.

자세한 내용은 AWS IoT Core개발자 안내서의 프로비저닝 템플릿을 참조하십시오.

`claimCertificatePath`

에서 지정하는 프로비전 템플릿의 프로비전 클레임 인증서 경로입니다. `provisioningTemplate` 자세한 내용을 알아보려면 AWS IoT Core API 참조의 CreateProvisioningClaim 섹션을 참조하십시오.

`claimCertificatePrivateKeyPath`

에서 지정하는 프로비전 템플릿의 프로비전 클레임 인증서 개인 키 경로입니다. `provisioningTemplate` 자세한 내용을 알아보려면 AWS IoT Core API 참조의 CreateProvisioningClaim 섹션을 참조하십시오.

> **Important**
>
> 프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스를 포함하여 항상 보호되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 청구 인증서의 무단 사용 등 오용의 징후가 있는지 모니터링하는 것이 좋습니다. 오용을 감지하면 프로비저닝 클레임 인증서를 비활성화하여 디바이스 프로비저닝에 사용할 수 없도록 하십시오. 자세한 내용을 알아보려면 AWS IoT Core 개발자 안내서의 AWS IoT 모니터링를 참조하세요.
>
> 디바이스 수와 등록되는 디바이스를 더 잘 관리할 수 있도록 플릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. AWS 계정 사전 프로비저닝 후크는 등록 시 디바이스가 제공하는 템플릿 파라미터의 유효성을 검사하는 AWS Lambda 함수입니다. 예를 들어, 데이터베이스를 기준으로 기기 ID를 검사하여 기기에 프로비전 권한이 있는지 확인하는 사전 프로비저닝 후크를 만들 수 있습니다. 자세한 내용은 개발자 안내서의 사전 프로비저닝 후크를 참조하십시오. AWS IoT Core

`rootCaPath`

Amazon 루트 인증 기관 (CA) 인증서의 경로.

`templateParameters`

(선택 사항) 플릿 프로비저닝 템플릿에 제공할 파라미터 맵. 자세한 내용은 개발자 안내서의 프로비저닝 템플릿의 매개변수 섹션을 참조하십시오. AWS IoT Core

`deviceId`

(선택 사항) 플릿 프로비저닝 플러그인이 MQTT 연결을 생성할 때 클라이언트 ID로 사용할 기기 식별자입니다. AWS IoT

기본값: 임의의 UUID.

`mqttPort`

(선택 사항) MQTT 연결에 사용할 포트입니다.

기본값: 8883

`proxyUrl`

(선택 사항) 형식의 `scheme://userinfo@host:port` 프록시 서버 URL. HTTPS 프록시를 사용하려면 플릿 프로비저닝 플러그인 버전 1.1.0 이상을 사용해야 합니다.

- `scheme`— 스킴은 또는이어야 합니다. `http` `https`

> **Important**
>
> HTTPS 프록시를 사용하려면 그린그래스 코어 디바이스에서 그린그래스 핵 v2.5.0 이상을 실행해야 합니다.
> HTTPS 프록시를 구성하는 경우 코어 디바이스의 Amazon 루트 CA 인증서에 프록시 서버 CA 인증서를 추가해야 합니다. 자세한 설명은 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.

- `userinfo`— (선택 사항) 사용자 이름 및 암호 정보. 에서 이 정보를 지정하는 경우 Greengrass 코어 디바이스는 및 필드를 무시합니다. `url` `username` `password`
- `host`— 프록시 서버의 호스트 이름 또는 IP 주소.
- `port`— (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 기기는 다음과 같은 기본값을 사용합니다.
  - `http`— 80
  - `https`— 443

`proxyUserName`

(선택 사항) 프록시 서버를 인증하는 사용자 이름.

`proxyPassword`

(선택 사항) 프록시 서버를 인증하는 사용자 이름.

CSRPath

(선택 사항) CSR에서 디바이스 인증서를 생성하는 데 사용할 인증서 서명 요청 (CSR) 파일의 경로입니다. 자세한 내용은 개발자 가이드의 AWS IoT Core클레임별 프로비저닝을 참조하십시오.

csrPrivateKey경로

(선택 사항, 선언된 경우 필수) CSR을 생성하는 데 사용된 개인 키의 `csrPath` 경로입니다. CSR을 생성하는 데 개인 키를 사용했어야 합니다. 자세한 내용은 AWS IoT Core개발자 가이드의 클레임에 의한 프로비저닝을 참조하십시오.

AWS IoT플릿 프로비저닝 플러그인 변경 로그

다음 표는 claim plugin (`aws.greengrass.FleetProvisioningByClaim`) 에 의한AWS IoT 플릿 프로비저닝의 각 버전 변경 사항을 설명합니다.

버전	변경
1.2.0	버그 수정 및 향상 • 구성 가능한 개인 키 경로를 사용하여 인증서 서명 요청을 통한 장치 프로비저닝에 대한 지원을 추가합니다. • 사소한 수정 및 개선 사항.
1.1.0	버그 수정 및 향상 • Windows 장치에서 플러그인을 구성할 때 추가 파일 경로 형식에 대한 지원을 추가합니다. • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 단원을 참조하세요.
1.0.0	초기 버전.

사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다.

AWS IoT Greengrass Core 소프트웨어 설치 프로그램은 필수 리소스를 프로비저닝하는 사용자 지정 플러그인으로 구현할 수 있는 Java 인터페이스를 제공합니다. AWS 사용자 지정 X.509 클라이언트 인증서를 사용하거나 다른 설치 프로세스에서 지원하지 않는 복잡한 프로비저닝 단계를 실행하도록 프로비저닝 플러그인을 개발할 수 있습니다. 자세한 내용은 개발자 안내서의 자체 클라이언트 인증서 만들기를 참조하십시오. AWS IoT Core

AWS IoT Greengrass Core 소프트웨어를 설치할 때 사용자 지정 프로비저닝 플러그인을 실행하려면 설치 프로그램에 제공하는 JAR 파일을 만들어야 합니다. 설치 프로그램은 플러그인을 실행하고 플러그인은 Greengrass 코어 장치의 AWS 리소스를 정의하는 프로비저닝 구성을 반환합니다. 설치 프로그램은 이 정보를 사용하여 기기에 AWS IoT Greengrass Core 소프트웨어를 구성합니다. 자세한 설명은 커스텀 프로비저닝 플러그인 개발 섹션을 참조하세요.

> **Important**
>
> Core 소프트웨어를 다운로드하기 전에 AWS IoT Greengrass 코어 장치가 Core 소프트웨어 v2.0을 설치하고 실행하기 위한 요구 사항을 충족하는지 확인하십시오. AWS IoT Greengrass

주제

필수 조건

사용자 지정 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치하려면 다음이 있어야 합니다.

- 를 구현하는 사용자 지정 프로비저닝 플러그인용 JAR 파일입니다. `DeviceIdentityInterface` 사용자 지정 프로비저닝 플러그인은 각 시스템 및 nucleus 구성 매개변수의 값을 반환해야 합니다. 그렇지 않으면 설치 중에 구성 파일에 해당 값을 제공해야 합니다. 자세한 설명은 커스텀 프로비저닝 플러그인 개발 섹션을 참조하세요.

디바이스 환경을 설정합니다.

이 섹션의 단계에 따라 AWS IoT Greengrass 핵심 장치로 사용할 Linux 또는 Windows 장치를 설정합니다.

Linux 디바이스 설정

Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass Core 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 장치에 OpenJDK를 설치하는 방법을 보여줍니다.

   - Debian 기반 또는 Ubuntu 기반 배포판의 경우:

```
sudo apt install default-jdk
```

- Red Hat 기반 배포판의 경우:

```
sudo yum install java-11-openjdk-devel
```

- 대상 Amazon Linux 2:

```
sudo amazon-linux-extras install java-openjdk11
```

- 아마존 리눅스 2023의 경우:

```
sudo dnf install java-11-amazon-corretto -y
```

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

이 명령은 장치에서 실행되는 Java 버전을 인쇄합니다. 예를 들어 Debian 기반 배포의 경우 출력은 다음 샘플과 비슷할 수 있습니다.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (선택 사항) 기기에서 구성 요소를 실행하는 기본 시스템 사용자 및 그룹을 생성합니다. 설치 중에 설치 프로그램 인수를 사용하여 AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 이 사용자 및 그룹을 만들도록 선택할 수도 있습니다. `--component-default-user` 자세한 설명은 인스톨러 인수 섹션을 참조하세요.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. AWS IoT Greengrass Core 소프트웨어를 실행하는 사용자 (일반적으로`root`) 에게 모든 사용자 및 그룹과 sudo 함께 실행할 수 있는 권한이 있는지 확인하십시오.

   a. 다음 명령을 실행하여 `/etc/sudoers` 파일을 엽니다.

```
sudo visudo
```

b. 사용자의 권한이 다음 예와 같은지 확인합니다.

```
root    ALL=(ALL:ALL) ALL
```

4. (선택 사항) 컨테이너화된 Lambda 함수를 실행하려면 cgroups v1을 활성화하고 메모리 및 디바이스 cgroups를 활성화하고 마운트해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없다면 이 단계를 건너뛰어도 됩니다.

   이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터를 사용하여 디바이스를 부팅하십시오.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

   디바이스의 커널 파라미터 보기 및 설정에 대한 자세한 내용은 운영 체제 및 부트로더 설명서를 참조하십시오. 지침에 따라 커널 파라미터를 영구적으로 설정하십시오.

5. 의 요구 사항 목록에 나와 있는 대로 다른 모든 필수 종속 항목을 장치에 설치하십시오. 디바이스 요구 사항

Windows 디바이스 설정

> **Note**
>
> 이 기능은 v2.5.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass Core 소프트웨어를 실행하는 데 필요한 Java 런타임을 설치합니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
2. PATH 시스템 변수에서 Java를 사용할 수 있는지 확인하고 사용할 수 없는 경우 추가하십시오. 이 LocalSystem 계정은 AWS IoT Greengrass Core 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 다음을 따릅니다.

   a. Windows 키를 눌러 시작 메뉴를 엽니다.

b. **environment variables**를 입력하여 시작 메뉴에서 시스템 옵션을 검색합니다.

c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.

d. 환경 변수 선택... 환경 변수 창을 열려면

e. 시스템 변수에서 경로를 선택한 다음 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.

f. Java 설치 `bin` 폴더 경로가 있는지 확인합니다. 경로는 다음 예와 비슷할 수 있습니다.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Java 설치 `bin` 폴더가 경로에 없는 경우 [새로 만들기] 를 선택하여 추가한 다음 [확인] 을 선택합니다.

3. Windows 명령 프롬프트 (`cmd.exe`) 를 관리자 권한으로 엽니다.

4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. *### ## ###* 바꾸십시오.

```
net user /add ggc_user password
```

> **Tip**
>
> Windows 구성에 따라 사용자 암호가 향후 날짜에 만료되도록 설정될 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 업데이트하십시오. 사용자 비밀번호가 만료되지 않도록 설정할 수도 있습니다.
>
> - 사용자 및 암호가 언제 만료되는지 확인하려면 다음 명령을 실행합니다.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 사용자 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - 이 명령이 더 이상 사용되지 않는 Windows 10 이상을 사용하는 경우 `wmic` 다음 명령을 실행하세요. PowerShell
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Microsoft에서 PsExec유틸리티를 다운로드하여 장치에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. *###* 이전에 설정한 사용자 암호로 바꾸십시오.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   가 PsExec License Agreement열리면 라이선스에 Accept동의하도록 선택하고 명령을 실행합니다.

   > **Note**
   >
   > Windows 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 계정에 기본 사용자 정보를 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다.

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 다음 위치에서 특정 버전의 AWS IoT Greengrass Core 소프트웨어를 다운로드할 수 있습니다. *###* 다운로드할 버전으로 교체하십시오.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

AWS IoT Greengrass Core 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 이름이 지정된 파일에 AWS IoT Greengrass Core 소프트웨어를 `greengrass-nucleus-latest.zip` 다운로드합니다.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
```

이 소프트웨어를 다운로드하면 Greengrass 코어 소프트웨어 라이선스 계약에 동의하는 것입니다.

2. (선택 사항) Greengrass 핵 소프트웨어 서명을 확인하려면

> **Note**
>
> 이 기능은 Greengrass 핵 버전 2.9.5 이상에서 사용할 수 있습니다.

a. 다음 명령을 사용하여 Greengrass 핵 아티팩트의 서명을 확인하십시오.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

설치한 JDK 버전에 따라 파일 이름이 다르게 보일 수 있습니다. 설치한 JDK 버전으로 *jdk17.0.6_10*바꾸십시오.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. `jarsigner`호출 시 확인 결과를 나타내는 출력이 출력됩니다.

i. Greengrass nucleus zip 파일에 서명된 경우 출력에는 다음 명령문이 포함됩니다.

```
jar verified.
```

ii. Greengrass nucleus zip 파일에 서명되지 않은 경우 출력에는 다음 명령문이 포함됩니다.

```
jar is unsigned.
```

c. Jarsigner `-certs` 옵션을 `-verify -verbose` 옵션과 함께 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. AWS IoT Greengrass Core 소프트웨어를 디바이스의 폴더에 압축을 풉니다. 사용하려는 *GreengrassInstaller*폴더로 바꾸십시오.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (선택 사항) 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 Core 소프트웨어를 설치한 후에 이 폴더를 제거하지 마십시오. AWS IoT Greengrass AWS IoT Greengrass Core 소프트웨어는 이 폴더의 파일을 사용하여 실행합니다.
> 최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치해야 하며, AWS IoT Greengrass Core 소프트웨어를 설치한 후 이 폴더를 제거할 수 있습니다.

Core 소프트웨어를 설치합니다. AWS IoT Greengrass

다음 작업을 지정하는 인수를 사용하여 설치 프로그램을 실행합니다.

- 사용자 지정 프로비저닝 플러그인을 사용하여 리소스를 프로비저닝하도록 지정하는 부분 구성 파일에서 설치합니다. AWS AWS IoT Greengrass Core 소프트웨어는 장치에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 프로그램은 사용자가 제공하는 부분 구성 파일과 사용자 지정 프로비저닝 플러그인이 생성하는 AWS 리소스로부터 전체 구성 파일을 생성합니다.
- `ggc_user`시스템 사용자를 사용하여 코어 기기에서 소프트웨어 구성 요소를 실행하도록 지정합니다. Linux 장치에서 이 명령은 또한 `ggc_group` 시스템 그룹을 사용하도록 지정하며, 설치 프로그램은 시스템 사용자 및 그룹을 자동으로 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 장치에서는 Systemd init 시스템이 필요합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 을 참조하십시오인스톨러 인수.

> **Note**
>
> 메모리가 제한된 AWS IoT Greengrass 장치에서 실행하는 경우 AWS IoT Greengrass Core 소프트웨어가 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하려면 nucleus 구성 요소의 `jvmOptions` 구성 매개변수에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.

AWS IoT Greengrass Core 소프트웨어를 설치하려면 (Linux)

1. AWS IoT Greengrass Core 소프트웨어 버전을 확인하십시오.

   - 소프트웨어가 들어 있는 폴더의 *GreengrassInstaller*경로로 바꾸십시오.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 텍스트 편집기를 사용하여 설치 프로그램에 제공할 이름을 `config.yaml` 지정하는 구성 파일을 생성합니다.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   다음 YAML 콘텐츠를 파일에 복사합니다.

   ```
   ---
   system:
     rootpath: "/greengrass/v2"
     # The following values are optional. Return them from the provisioning plugin or
    set them here.
     # certificateFilePath: ""
     # privateKeyPath: ""
     # rootCaPath: ""
     # thingName: ""
   services:
     aws.greengrass.Nucleus:
       version: "2.12.2"
       configuration:
   ```

```
      # The following values are optional. Return them from the provisioning plugin
or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

이어서 다음을 수행합니다.

- *2.12.2#* Core 소프트웨어 버전으로 교체합니다. AWS IoT Greengrass
- 의 각 인스턴스를 Greengrass 루트 */greengrass/v2* 폴더로 교체합니다.
- (선택 사항) 시스템 및 핵 구성 값을 지정합니다. 프로비저닝 플러그인에서 이 값을 제공하지 않는 경우 이 값을 설정해야 합니다.
- (선택 사항) 프로비저닝 플러그인에 제공할 구성 매개변수를 지정합니다.

> **Note**
>
> 다음 예와 같이 이 구성 파일에서 사용할 포트 및 네트워크 프록시와 같은 다른 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 Greengrass 핵 구성을 참조하십시오.
>
> ```
> ---
> system:
>   rootpath: "/greengrass/v2"
>   # The following values are optional. Return them from the provisioning
>  plugin or set them here.
>   # certificateFilePath: ""
>   # privateKeyPath: ""
>   # rootCaPath: ""
>   # thingName: ""
> services:
>   aws.greengrass.Nucleus:
>     version: "2.12.2"
>     configuration:
>       mqtt:
>         port: 443
>       greengrassDataPlanePort: 443
> ```

```
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
  plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
   com.example.CustomProvisioning:
     configuration:
       # You can specify configuration parameters to provide to your plugin.
       # pluginParameter: ""
```

3. 설치 관리자를 실행합니다. 사용자 지정 프로비저닝 플러그인을 `--trusted-plugin` 제공하도록 지정하고 구성 파일을 `--init-config` 제공하도록 지정합니다.

   - */greengrass/v2*또는 *C:\greengrass\v2* 를 Greengrass 루트 폴더로 바꾸십시오.
   - 의 각 인스턴스를 설치 *GreengrassInstaller*프로그램의 압축을 푼 폴더로 바꾸십시오.
   - 사용자 지정 프로비저닝 플러그인 JAR 파일의 경로를 플러그인의 JAR 파일 경로로 바꾸십시오.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin /path/to/com.example.CustomProvisioning.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
     -jar ./GreengrassInstaller/lib/Greengrass.jar ^
     --trusted-plugin /path/to/com.example.CustomProvisioning.jar ^
     --init-config ./GreengrassInstaller/config.yaml ^
   ```

```
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> Windows 코어 디바이스에서는 Core 소프트웨어를 시스템 서비스로 `--setup-system-service true` 설정하도록 지정해야 합니다. AWS IoT Greengrass

지정한 `--setup-system-service true` 경우 설치 프로그램은 소프트웨어를 시스템 서비스로 설정하고 실행했는지 `Successfully set up Nucleus as a system service` 여부를 인쇄합니다. 그렇지 않으면 소프트웨어를 성공적으로 설치해도 설치 프로그램이 메시지를 출력하지 않습니다.

> **Note**
>
> `deploy-dev-tools`인수 없이 설치 프로그램을 실행할 때는 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. `--provision true` Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 을 참조하십시오. 그린그래스 커맨드 라인 인터페이스

4. 루트 폴더의 파일을 확인하여 설치를 확인합니다.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

설치에 성공하면 루트 폴더에는, `config`, `packages`, `logs` 등의 여러 폴더가 포함됩니다.

AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설치한 경우 설치 프로그램이 소프트웨어를 자동으로 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 설명은 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.

소프트웨어 구성 및 사용 방법에 대한 자세한 내용은 AWS IoT Greengrass 다음을 참조하십시오.

- AWS IoT Greengrass Core 소프트웨어 구성
- AWS IoT Greengrass구성 요소 개발
- 디바이스에 AWS IoT Greengrass 구성 요소 배포
- 그린그래스 커맨드 라인 인터페이스

커스텀 프로비저닝 플러그인 개발

사용자 지정 프로비저닝 플러그인을 개발하려면 다음을 구현하는 Java 클래스를 만듭니다 `com.aws.greengrass.provisioning.DeviceIdentityInterface` 인터페이스를 구현해야 합니다. 프로젝트에 Greengrass nucleus JAR 파일을 포함시켜 이 인터페이스와 해당 클래스에 액세스할 수 있습니다. 이 인터페이스는 플러그인 구성을 입력하고 프로비저닝 구성을 출력하는 방법을 정의합니다. 프로비저닝 구성은 시스템에 대한 구성을 정의하고Greengrass 핵 성분. 이AWS IoT Greengrass핵심 소프트웨어 설치 관리자는 이 프로비저닝 구성을 사용하여AWS IoT Greengrass디바이스의 핵심 소프트웨어.

사용자 지정 프로비저닝 플러그인을 개발한 후 이 플러그인을 JAR 파일로 빌드하여AWS IoT Greengrass설치 중에 플러그인을 실행하는 핵심 소프트웨어 설치 프로그램입니다. 설치 프로그램은 설치 관리자가 사용하는 것과 동일한 JVM에서 사용자 지정 프로비저닝 플러그인을 실행하므로 플러그인 코드만 포함된 JAR를 만들 수 있습니다.

> **Note**
>
> 이AWS IoT플릿 프로비저닝 플러그인을 구현합니다 `DeviceIdentityInterface`를 사용하여 설치 중에 플릿 프로비저닝을 사용합니다. 플릿 프로비저닝 플러그인은 오픈 소스이므로 소

스 코드를 탐색하여 프로비저닝 플러그인 인터페이스를 사용하는 방법의 예를 볼 수 있습니다. 자세한 내용은 단원을 참조하십시오.AWS IoT플릿 프로비저닝 플러그인(GitHub에 있음).

주제

- 요구 사항
- 을 구현하려면 DeviceIdentityInterface 인터페이스

요구 사항

사용자 지정 프로비저닝 플러그인을 개발하려면 다음 요구 사항을 충족하는 Java 클래스를 만들어야 합니다.

- 를 사용합니다.`com.aws.greengrass`패키지 또는 패키지 내 패키지`com.aws.greengrass`패키지
- 인수가 없는 생성자가 있습니다.
- 을 구현합니다.`DeviceIdentityInterface`인터페이스를 구현해야 합니다. 자세한 정보는 을 구현하려면 DeviceIdentityInterface 인터페이스을 참조하십시오.

을 구현하려면 DeviceIdentityInterface 인터페이스

이`com.aws.greengrass.provisioning.DeviceIdentityInterface`사용자 정의 플러그인에 인터페이스하고 Greengrass 핵을 프로젝트에 종속성으로 추가하십시오.

이 DeviceIdentityInterface 사용자 지정 프로비저닝 플러그인 프로젝트

- Greengrass 핵 JAR 파일을 라이브러리로 추가하거나 Greengrass 핵을 Maven 종속성으로 추가할 수 있습니다. 다음 중 하나를 수행하세요.
  - Greengrass 핵 JAR 파일을 라이브러리로 추가하려면AWS IoT Greengrass코어 소프트웨어, Greengrass 핵 JAR이 포함되어 있습니다. 의 최신 버전을 다운로드할 수 있습니다.AWS IoT Greengrass다음 위치의 코어 소프트웨어
    - https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

    당신은 Greengrass 핵 JAR 파일을 찾을 수 있습니다 (`Greengrass.jar`) 에서`lib`ZIP 파일의 폴더 프로젝트에 이 JAR 파일을 추가합니다.

- Maven 프로젝트에서 Greengrass 핵을 소비하려면nucleus의 아티팩트com.aws.greengrass그룹 을 추가해야 합니다.greengrass-common저장소입니다. Greengrass 핵은 메이븐 센트럴 리포지토리에서 사용할 수 없기 때문입니다.

```
<project ...>
    ...
    <repositories>
        <repository>
            <id>greengrass-common</id>
            <name>greengrass common</name>
            <url>https://d2jrmugq4soldf.cloudfront.net/snapshots</url>
        </repository>
    </repositories>
    ...
    <dependencies>
        <dependency>
            <groupId>com.aws.greengrass</groupId>
            <artifactId>nucleus</artifactId>
            <version>2.5.0-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

디바이스아이덴티인터페이스

이com.aws.greengrass.provisioning.DeviceIdentityInterface인터페이스의 모양은 다음과 같습니다.

> **Note**
>
> 에서 이러한 클래스를 탐색할 수도 있습니다.com.aws.그린그래스 프로비저닝 패키지의Greengrass 핵 소스 코드(GitHub에 있음).

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
            throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
```

```
    String name();
}

com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}

com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy;  // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

의 각 구성 값`SystemConfiguration`과`NucleusConfiguration`를 설치하는 데 필요합니다.AWS IoT Greengrass핵심 소프트웨어이지만 반환할 수 있습니다.`null`. 사용자 지정 프로비저닝 플러그인이 반환되는 경우`null`모든 구성 값에 대해 다음을 생성할 때 시스템 또는 핵 구성에 해당 값을 제공해야 합니다.`config.yaml`제공할 파일AWS IoT Greengrass코어 소프트웨어 설치 사용자 지정 프로비저닝 플러그인이 에서 정의하는 옵션에 대해 null이 아닌 값을 반환하는 경우`config.yaml`그러면 설치 프로그램이 의 값을 대체합니다.`config.yaml`플러그인에서 반환한 값을 사용합니다.

인스톨러 인수

AWS IoT GreengrassCore 소프트웨어에는 소프트웨어를 설정하고 Greengrass 코어 장치 실행에 필요한 AWS 리소스를 프로비저닝하는 설치 프로그램이 포함되어 있습니다. 설치 프로그램에는 설치 구성을 위해 지정할 수 있는 다음과 같은 인수가 포함되어 있습니다.

`-h, --help`

(선택 사항) 설치 프로그램의 도움말 정보를 표시합니다.

`--version`

(선택 사항) AWS IoT Greengrass Core 소프트웨어 버전을 표시합니다.

`-Droot`

(선택 사항) AWS IoT Greengrass Core 소프트웨어의 루트로 사용할 폴더의 경로입니다.

> **Note**
>
> 이 인수는 JVM 속성을 설정하므로 설치 프로그램을 실행할 `-jar` 때 먼저 JVM 속성을 지정해야 합니다. 예를 들어, `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar`를 지정합니다.

기본값:

- Linux: `~/.greengrass`
- Windows: `%USERPROFILE%/.greengrass`

`-ar, --aws-region`

AWS IoT GreengrassCore 소프트웨어가 필수 리소스를 검색하거나 생성하는 데 사용하는 AWS 항목입니다. AWS 리전

`-p, --provision`

(선택 사항) 이 장치를 AWS IoT 사물로 등록하고 코어 장치에 필요한 AWS 리소스를 제공할 수 있습니다. 지정하는 `true` 경우 AWS IoT Greengrass Core 소프트웨어는 AWS IoT 사물 (선택 사항), AWS IoT 사물 그룹, IAM 역할 및 AWS IoT 역할 별칭을 프로비전합니다.

기본값: `false`

`-tn, --thing-name`

(선택 사항) 이 코어 디바이스로 AWS IoT 등록한 사물의 이름. 해당 이름을 가진 항목이 사용자 AWS 계정 내부에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 해당 항목을 생성합니다.

> **Note**
>
> 사물 이름에는 콜론 (`:`) 문자를 포함할 수 없습니다.

이 인수를 `--provision true` 적용하려면 지정해야 합니다.

기본값: 임의 UUID `GreengrassV2IotThing_` 추가

`-tgn, --thing-group-name`

(선택 사항) 이 코어 디바이스의 AWS IoT 사물을 추가하는 AWS IoT 사물 그룹의 이름. 배포가 이 사물 그룹을 대상으로 하는 경우 이 코어 장치는 연결될 때 해당 배포를 AWS IoT Greengrass 수신합니다. 이 이름을 가진 사물 그룹이 사용자 AWS 계정 내에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 해당 사물 그룹을 생성합니다.

> **Note**
>
> 사물 그룹 이름에는 콜론 (`:`) 문자를 포함할 수 없습니다.

이 인수를 `--provision true` 적용하도록 지정해야 합니다.

`-tpn, --thing-policy-name`

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다.

(선택 사항) 이 코어 디바이스의 사물 인증서에 첨부할 AWS IoT 정책의 이름. AWS IoT 이 이름을 가진 AWS IoT 정책이 AWS 계정 귀사에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 해당 정책을 생성합니다.

AWS IoT GreengrassCore 소프트웨어는 기본적으로 허용 AWS IoT 정책을 생성합니다. 이 정책의 범위를 좁히거나 사용 사례에 대한 권한을 제한하는 사용자 지정 정책을 만들 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

이 인수를 `--provision true` 적용하도록 지정해야 합니다.

기본값: `GreengrassV2IoTThingPolicy`

`-trn, --tes-role-name`

(선택 사항) 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 하는 AWS 자격 증명을 획득하는 데 사용할 IAM 역할의 이름입니다. 이 이름을 가진 역할이 사용자 AWS 계정 내에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 `GreengrassV2TokenExchangeRoleAccess` 정책을 사용하여 해당 역할을 생성합니다. 이 역할은 구성 요소 아티팩트를 호스팅하는 S3 버킷에 액세스할 수 없습니다. 따라서 구성 요소를 생성할 때 아티팩트의 S3 버킷 및 객체에 권한을 추가해야

합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

이 인수를 `--provision true` 적용하도록 지정해야 합니다.

기본값: `GreengrassV2TokenExchangeRole`

`-tra, --tes-role-alias-name`

(선택 사항) 이 코어 디바이스의 AWS 자격 증명을 제공하는 IAM 역할을 가리키는 AWS IoT 역할 별칭의 이름입니다. 이 이름의 역할 별칭이 사용자 AWS 계정 계정에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 이를 생성하여 지정한 IAM 역할을 가리킵니다.

이 인수를 `--provision true` 적용하도록 지정해야 합니다.

기본값: `GreengrassV2TokenExchangeRoleAlias`

`-ss, --setup-system-service`

(선택 사항) AWS IoT Greengrass Core 소프트웨어를 이 기기가 부팅될 때 실행되는 시스템 서비스로 설정할 수 있습니다. 시스템 서비스 이름은 입니다`greengrass`. 자세한 설명은 Greengrass 핵을 시스템 서비스로 구성 섹션을 참조하세요.

Linux 운영 체제에서 이 인수를 사용하려면 기기에서 systemd init 시스템을 사용할 수 있어야 합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

기본값: `false`

`-u, --component-default-user`

AWS IoT GreengrassCore 소프트웨어가 구성 요소를 실행하는 데 사용하는 사용자의 이름 또는 ID. 예를 들어, **`ggc_user`**를 지정할 수 있습니다. 이 값은 Windows 운영 체제에서 설치 프로그램을 실행할 때 필요합니다.

Linux 운영 체제에서는 그룹을 선택적으로 지정할 수도 있습니다. 사용자와 그룹을 콜론으로 구분하여 지정합니다. 예: **`ggc_user:ggc_group`**.

Linux 운영 체제에는 다음과 같은 추가 고려 사항이 적용됩니다.

- 루트로 실행하는 경우 기본 구성 요소 사용자는 구성 파일에 정의된 사용자입니다. 구성 파일이 사용자를 정의하지 않는 경우 기본값은 입니다. `ggc_user:ggc_group` `ggc_user`존재하거나 존재하지 `ggc_group` 않는 경우 소프트웨어에서 생성합니다.
- 루트가 아닌 사용자로 실행하는 경우 AWS IoT Greengrass Core 소프트웨어는 해당 사용자를 사용하여 구성 요소를 실행합니다.
- 그룹을 지정하지 않으면 AWS IoT Greengrass Core 소프트웨어는 시스템 사용자의 기본 그룹을 사용합니다.

자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.

`-d, --deploy-dev-tools`

(선택 사항) Greengrass CLI 구성 요소를 다운로드하여 이 코어 디바이스에 배포할 수 있습니다. 이 도구를 사용하여 이 코어 디바이스에서 구성 요소를 개발하고 디버그할 수 있습니다.

> **Important**
>
> 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

이 인수를 `--provision true` 적용하도록 지정해야 합니다.

기본값: `false`

`-init, --init-config`

(선택 사항) AWS IoT Greengrass Core 소프트웨어를 설치하는 데 사용할 구성 파일의 경로입니다. 예를 들어 이 옵션을 사용하여 특정 Nucleus 구성으로 새 코어 장치를 설정할 수 있습니다.

> **Important**
>
> 지정한 구성 파일은 코어 장치의 기존 구성 파일과 병합됩니다. 여기에는 코어 장치의 구성 요소 및 구성 요소 구성이 포함됩니다. 구성 파일에는 변경하려는 구성만 나열하는 것이 좋습니다.

`-tp, --trusted-plugin`

(선택 사항) 신뢰할 수 있는 플러그인으로 로드할 JAR 파일의 경로입니다. 이 옵션을 사용하여 프로비저닝 플러그인 JAR 파일을 제공할 수 있습니다. 예를 들어 플릿 프로비저닝 또는 사용자 지정 프로비저닝으로 설치하거나 하드웨어 보안 모듈에서 개인 키 및 인증서를 사용하여 설치할 수 있습니다.

`-s, --start`

(선택 사항) Core 소프트웨어를 설치한 후 AWS IoT Greengrass Core 소프트웨어를 시작하고 선택적으로 리소스를 프로비저닝할 수 있습니다.

기본값: `true`

AWS IoT GreengrassCore 소프트웨어 실행

AWS IoT GreengrassCore 소프트웨어를 설치한 후 실행하여 장치를 연결합니다AWS IoT Greengrass.

AWS IoT GreengrassCore 소프트웨어를 설치할 때 systemd와 함께 시스템 서비스로 설치할지 여부를 지정할 수 있습니다. 이 옵션을 선택하면 설치 프로그램이 자동으로 소프트웨어를 실행하고 장치 부팅 시 실행되도록 구성합니다.

> **Important**
>
> Windows 코어 장치에서는 Core 소프트웨어를 시스템 서비스로 AWS IoT Greengrass 설정해야 합니다.

주제

- AWS IoT GreengrassCore 소프트웨어가 시스템 서비스로 실행되는지 확인하세요.
- AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 실행합니다.
- 시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어 실행

AWS IoT GreengrassCore 소프트웨어가 시스템 서비스로 실행되는지 확인하세요.

AWS IoT GreengrassCore 소프트웨어를 설치할 때 `--setup-system-service true` 인수를 지정하여 AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설치할 수 있습니다. Linux 디바이스에서 AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설정하려면 systemd init 시스템이 필요합니다. 이 옵션을 사용하면 설치 프로그램이 소프트웨어를 자동으로 실행하고 장치 부팅 시 실행되도록 구성합니다. AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 성공적으로 설치하면 설치 프로그램이 다음 메시지를 출력합니다.

```
Successfully set up Nucleus as a system service
```

이전에 AWS IoT Greengrass Core 소프트웨어를 설치했는데 설치 프로그램 출력이 없는 경우 소프트웨어가 시스템 서비스로 설치되었는지 확인할 수 있습니다.

AWS IoT GreengrassCore 소프트웨어가 시스템 서비스로 설치되었는지 확인하려면

- 다음 명령을 실행하여 Greengrass 시스템 서비스의 상태를 확인합니다.

  Linux or Unix (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

  AWS IoT GreengrassCore 소프트웨어가 시스템 서비스로 설치되고 활성화된 경우 응답은 다음 예와 비슷합니다.

  ```
  # greengrass.service - Greengrass Core
     Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
   preset: disabled)
     Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
   Main PID: 16107 (sh)
     CGroup: /system.slice/greengrass.service
             ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
             ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
  v2/alts/current/distro/lib/Greengrass...
  ```

  찾을 `systemctl` 수 없거나 `greengrass.service` 찾을 수 없는 경우 AWS IoT Greengrass Core 소프트웨어는 시스템 서비스로 설치되지 않습니다. 소프트웨어를 실행하려면 을 참조하십시오시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어 실행.

Windows Command Prompt (CMD)

```
sc query greengrass
```

AWS IoT GreengrassCore 소프트웨어가 Windows 서비스로 설치되고 활성화된 경우 응답은 다음 예와 비슷합니다.

```
SERVICE_NAME: greengrass
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

PowerShell

```
Get-Service greengrass
```

AWS IoT GreengrassCore 소프트웨어가 Windows 서비스로 설치되고 활성 상태인 경우 응답은 다음 예와 비슷합니다.

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 실행합니다.

AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 설치한 경우 시스템 서비스 관리자를 사용하여 소프트웨어를 시작, 중지 및 관리할 수 있습니다. 자세한 설명은 Greengrass 핵을 시스템 서비스로 구성 섹션을 참조하세요.

AWS IoT GreengrassCore 소프트웨어를 실행하려면

- 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어를 시작합니다.

Linux or Unix (systemd)

```
sudo systemctl start greengrass.service
```

Windows Command Prompt (CMD)

```
sc start greengrass
```

PowerShell

```
Start-Service greengrass
```

시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어 실행

Linux 코어 장치에서 AWS IoT Greengrass Core 소프트웨어가 시스템 서비스로 설치되지 않은 경우 소프트웨어의 로더 스크립트를 실행하여 소프트웨어를 실행할 수 있습니다.

시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어를 실행하려면

- 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어를 시작합니다. 터미널에서 이 명령을 실행하는 경우 AWS IoT Greengrass Core 소프트웨어가 계속 실행되도록 터미널 세션을 열어 두어야 합니다.
  - */greengrass/v2*또는 *C:\greengrass\v2* 를 사용 중인 Greengrass 루트 폴더로 바꾸십시오.

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

소프트웨어가 성공적으로 시작되면 다음 메시지를 인쇄합니다.

```
Launched Nucleus successfully.
```

Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어 실행

AWS IoT Greengrass Docker 컨테이너에서 실행되도록 구성할 수 있습니다. Docker는 Linux 컨테이너 기반 애플리케이션을 빌드, 실행, 테스트 및 배포할 수 있는 도구를 제공하는 플랫폼입니다. AWS IoT Greengrass Docker 이미지를 실행할 때 Docker 컨테이너에 AWS 자격 증명을 제공할지 여부를 선택하고 AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 Greengrass 코어 장치가 작동하는 데 AWS 필요한 리소스를 자동으로 프로비저닝하도록 허용할 수 있습니다. AWS 자격 증명을 제공하지 않으려면 Docker 컨테이너에서 AWS 리소스를 수동으로 프로비저닝하고 AWS IoT Greengrass Core 소프트웨어를 실행할 수 있습니다.

주제

지원되는 플랫폼 및 요구 사항

Docker 컨테이너에 AWS IoT Greengrass Core 소프트웨어를 설치하고 실행하려면 호스트 컴퓨터가 다음과 같은 최소 요구 사항을 충족해야 합니다.

- 인터넷에 연결된 Linux 기반 운영 체제.
- 도커 엔진 버전 18.09 이상
- (선택 사항) 도커 컴포지션 버전 1.22 이상. Docker Compose는 Docker Compose CLI를 사용하여 Docker 이미지를 실행하려는 경우에만 필요합니다.

Docker 컨테이너 내에서 Lambda 함수 구성 요소를 실행하려면 추가 요구 사항을 충족하도록 컨테이너를 구성해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.

구성 요소를 프로세스 모드에서 실행합니다.

AWS IoT Greengrass Docker 컨테이너 내의 격리된 런타임 환경에서 Lambda 함수 AWS 또는 제공된 구성 요소를 실행하는 것을 지원하지 않습니다. AWS IoT Greengrass 이러한 구성 요소는 격리 없이 프로세스 모드에서 실행해야 합니다.

Lambda 함수 구성 요소를 구성할 때 격리 모드를 컨테이너 없음으로 설정합니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

AWS제공된 다음 구성 요소를 배포할 때는 파라미터를 로 설정하도록 각 구성 요소의 구성을 업데이트하십시오`containerMode`. `NoContainer` 구성 업데이트에 대한 자세한 내용은 을 참조하십시오구성 요소 구성 업데이트.

- CloudWatch 메트릭
- Device Defender
- Firehose
- 모드버스-RTU 프로토콜 어댑터
- Amazon SNS

AWS IoT Greengrass 도커 소프트웨어 다운로드

AWS IoT Greengrass Amazon Linux 2 (x86_64) 기본 이미지에 AWS IoT Greengrass 코어 소프트웨어 및 종속 항목이 설치된 컨테이너 이미지를 빌드하기 위한 Dockerfile을 제공합니다. 다른 플랫폼 아키텍처에서 실행되도록 Dockerfile의 기본 이미지를 수정할 수 있습니다. AWS IoT Greengrass

에서 Dockerfile 패키지를 다운로드하십시오. GitHub

도커파일은 이전 버전의 그린그래스를 사용합니다. 원하는 Greengrass 버전을 사용하려면 파일을 업데이트해야 합니다. Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지를 빌드하는 방법에 대한 자세한 내용은 을 참조하십시오. AWS IoT GreengrassDockerfile에서 컨테이너 이미지 빌드

리소스 프로비저닝 방법을 선택하세요. AWS

Docker 컨테이너에 AWS IoT Greengrass Core 소프트웨어를 설치할 때 Greengrass 코어 장치가 작동하는 데 필요한 AWS 리소스를 자동으로 프로비저닝할지 아니면 수동으로 프로비저닝한 리소스를 사용할지를 선택할 수 있습니다.

- 자동 리소스 프로비저닝 —컨테이너 이미지를 처음 실행할 때 설치 프로그램이 AWS IoT AWS IoT 사물, 사물 그룹, IAM 역할 및 AWS IoT 역할 별칭을 프로비저닝합니다 AWS IoT Greengrass . 설치

프로그램은 로컬 개발 도구를 코어 장치에 배포할 수도 있으므로 장치를 사용하여 사용자 지정 소프트웨어 구성 요소를 개발하고 테스트할 수 있습니다. 이러한 리소스를 자동으로 프로비저닝하려면 AWS 자격 증명을 Docker 이미지에 환경 변수로 제공해야 합니다.

자동 프로비저닝을 사용하려면 Docker 환경 변수를 `PROVISION=true` 설정하고 자격 증명 파일을 마운트하여 컨테이너에 AWS 자격 증명을 제공해야 합니다.

- 수동 리소스 프로비저닝 - 컨테이너에 AWS 자격 증명을 제공하지 않으려면 컨테이너 이미지를 실행하기 전에 AWS 리소스를 수동으로 프로비저닝할 수 있습니다. AWS IoT Greengrass Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램에 이러한 리소스에 대한 정보를 제공하는 구성 파일을 만들어야 합니다.

  수동 프로비저닝을 사용하려면 Docker 환경 변수를 설정해야 합니다. `PROVISION=false` 수동 프로비저닝은 기본 옵션입니다.

자세한 내용은 AWS IoT GreengrassDockerfile에서 컨테이너 이미지 빌드을(를) 참조하세요.

AWS IoT GreengrassDockerfile에서 컨테이너 이미지 빌드

AWS다운로드하여 Docker 컨테이너에서 Core 소프트웨어를 실행하는 데 사용할 수 있는 Dockerfile을 제공합니다. AWS IoT Greengrass Dockerfile에는 컨테이너 이미지를 빌드하기 위한 소스 코드가 포함되어 있습니다. AWS IoT Greengrass

AWS IoT Greengrass컨테이너 이미지를 빌드하기 전에 설치하려는 AWS IoT Greengrass Core 소프트웨어 버전을 선택하도록 Dockerfile을 구성해야 합니다. 또한 환경 변수를 구성하여 설치 중에 리소스를 프로비저닝하는 방법을 선택하고 다른 설치 옵션을 사용자 지정할 수 있습니다. 이 섹션에서는 Dockerfile에서 AWS IoT Greengrass Docker 이미지를 구성하고 빌드하는 방법을 설명합니다.

Dockerfile 패키지를 다운로드하십시오.

AWS IoT GreengrassDockerfile 패키지는 다음에서 다운로드할 수 있습니다. GitHub

AWS 그린그래스 도커 리포지토리

패키지를 다운로드한 후 컴퓨터의 *download-directory/*`aws-greengrass-docker-`*nucleus-version* 폴더에 콘텐츠를 추출합니다. 도커파일은 이전 버전의 그린그래스를 사용합니다. 원하는 Greengrass 버전을 사용하려면 파일을 업데이트해야 합니다.

AWS IoT GreengrassCore 소프트웨어 버전을 지정하십시오.

Dockerfile의 다음 빌드 인수를 사용하여 Docker 이미지에서 사용할 AWS IoT Greengrass Core 소프트웨어 버전을 지정합니다. AWS IoT Greengrass 기본적으로 Dockerfile은 최신 버전의 Core 소프트웨어를 사용합니다. AWS IoT Greengrass

`GREENGRASS_RELEASE_VERSION`

Core 소프트웨어 버전. AWS IoT Greengrass 기본적으로 Dockerfile은 사용 가능한 최신 버전의 Greengrass 핵을 다운로드합니다. 값을 다운로드하려는 뉴클리어스 버전으로 설정합니다.

환경 변수 설정

환경 변수를 사용하면 Docker 컨테이너에 AWS IoT Greengrass Core 소프트웨어를 설치하는 방법을 사용자 지정할 수 있습니다. AWS IoT GreengrassDocker 이미지의 환경 변수를 다양한 방법으로 설정할 수 있습니다.

- 동일한 환경 변수를 사용하여 여러 이미지를 만들려면 Dockerfile에서 직접 환경 변수를 설정하십시오.
- 를 `docker run` 사용하여 컨테이너를 시작하는 경우 명령에서 환경 변수를 인수로 전달하거나 환경 변수 파일에 환경 변수를 설정한 다음 파일을 인수로 전달하십시오. Docker에서 환경 변수를 설정하는 방법에 대한 자세한 내용은 Docker 설명서의 환경 변수를 참조하십시오.
- 를 `docker-compose up` 사용하여 컨테이너를 시작하는 경우 환경 변수 파일에 환경 변수를 설정한 다음 파일을 인수로 전달하십시오. Compose에서 환경 변수를 설정하는 방법에 대한 자세한 내용은 Docker 설명서를 참조하십시오.

AWS IoT GreengrassDocker 이미지에 대해 다음과 같은 환경 변수를 구성할 수 있습니다.

> **Note**
>
> Dockerfile에서 `TINI_KILL_PROCESS_GROUP` 변수를 수정하지 마세요. 이 변수를 사용하면 PID 그룹의 모든 PID에 `SIGTERM` 전달할 수 있으므로 Docker 컨테이너가 중지될 때 AWS IoT Greengrass Core 소프트웨어가 올바르게 종료될 수 있습니다.

`GGC_ROOT_PATH`

(선택 사항) Core 소프트웨어의 루트로 사용할 컨테이너 내 폴더의 경로입니다. AWS IoT Greengrass

기본값: `/greengrass/v2`

`PROVISION`

(선택 사항) AWS IoT Greengrass Core에서 AWS 리소스를 프로비저닝할지 여부를 결정합니다.

- 지정하는 `true` 경우 AWS IoT Greengrass Core 소프트웨어는 컨테이너 이미지를 사물로 등록하고 Greengrass 코어 장치에 AWS 필요한 리소스를 제공합니다. AWS IoT AWS IoT GreengrassCore 소프트웨어는 AWS IoT 사물 (선택 사항), AWS IoT 사물 그룹, IAM 역할 및 역할 별칭을 AWS IoT 프로비전합니다. 자세한 설명은 자동 리소스 프로비저닝으로 Docker AWS IoT Greengrass 컨테이너에서 실행 섹션을 참조하세요.
- 지정하는 `false` 경우 수동으로 생성한 AWS 리소스와 인증서를 사용하도록 지정하는 구성 파일을 생성하여 AWS IoT Greengrass Core 설치 프로그램에 제공해야 합니다. 자세한 설명은 수동 리소스 프로비저닝으로 Docker AWS IoT Greengrass 컨테이너에서 실행 섹션을 참조하세요.

기본값: `false`

`AWS_REGION`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 필요한 AWS 리소스를 검색하거나 생성하는 데 사용하는 파일입니다. AWS 리전

기본값: `us-east-1`.

`THING_NAME`

(선택 사항) 이 코어 디바이스로 AWS IoT 등록한 사물의 이름. 이 이름을 가진 사물이 사용자 AWS 계정 내부에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 해당 항목을 생성합니다.

이 인수를 `PROVISION=true` 적용하도록 지정해야 합니다.

기본값: 임의 UUID `GreengrassV2IotThing_` 추가.

`THING_GROUP_NAME`

(선택 사항) 이 코어 디바이스를 추가하는 AWS IoT 사물 그룹의 이름. 배포가 이 사물 그룹을 대상으로 하는 AWS IoT 경우 이 디바이스 및 해당 그룹의 다른 코어 디바이스가 연결될 때 해당 배포를 수신합니다. AWS IoT Greengrass 이 이름을 가진 사물 그룹이 사용자 AWS 계정 내에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 해당 사물 그룹을 생성합니다.

이 인수를 PROVISION=true 적용하도록 지정해야 합니다.

TES_ROLE_NAME

(선택 사항) Greengrass 코어 디바이스가 서비스와 상호 작용할 AWS 수 있도록 하는 AWS 자격 증명을 획득하는 데 사용할 IAM 역할의 이름입니다. 이 이름을 가진 역할이 사용자 AWS 계정 내에 없는 경우 AWS IoT Greengrass Core 소프트웨어에서 정책을 사용하여 해당 역할을 생성합니다. GreengrassV2TokenExchangeRoleAccess 이 역할은 구성 요소 아티팩트를 호스팅하는 S3 버킷에 액세스할 수 없습니다. 따라서 구성 요소를 생성할 때 아티팩트의 S3 버킷 및 객체에 권한을 추가해야 합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

기본값: GreengrassV2TokenExchangeRole

TES_ROLE_ALIAS_NAME

(선택 사항) Greengrass 코어 디바이스의 AWS 자격 증명을 제공하는 IAM 역할을 가리키는 AWS IoT 역할 별칭의 이름. 이 이름의 역할 별칭이 사용자 AWS 계정 계정에 없는 경우 AWS IoT Greengrass Core 소프트웨어가 이를 생성하여 지정한 IAM 역할을 가리킵니다.

기본값: GreengrassV2TokenExchangeRoleAlias

COMPONENT_DEFAULT_USER

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 구성 요소를 실행하는 데 사용하는 시스템 사용자 및 그룹의 이름 또는 ID. 콜론으로 구분하여 사용자와 그룹을 지정합니다. 그룹은 선택 사항입니다. 예를 들어, **ggc_user:ggc_group** 또는 **ggc_user** 를 지정할 수 있습니다.

- 루트로 실행하는 경우 구성 파일에 정의된 사용자 및 그룹이 기본값으로 설정됩니다. 구성 파일이 사용자 및 그룹을 정의하지 않는 경우 기본값은 입니다. ggc_user:ggc_group ggc_user존재하거나 존재하지 ggc_group 않는 경우 소프트웨어에서 생성합니다.
- 루트가 아닌 사용자로 실행하는 경우 AWS IoT Greengrass Core 소프트웨어는 해당 사용자를 사용하여 구성 요소를 실행합니다.
- 그룹을 지정하지 않으면 AWS IoT Greengrass Core 소프트웨어는 시스템 사용자의 기본 그룹을 사용합니다.

자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.

DEPLOY_DEV_TOOLS

컨테이너 이미지에 Greengrass CLI 구성 요소를 다운로드하고 배포할지 여부를 정의합니다. Greengrass CLI를 사용하여 로컬에서 구성 요소를 개발하고 디버그할 수 있습니다.

> **Important**
>
> 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

기본값: `false`

`INIT_CONFIG`

(선택 사항) AWS IoT Greengrass Core 소프트웨어를 설치하는 데 사용할 구성 파일의 경로입니다. 예를 들어 이 옵션을 사용하여 특정 핵 구성으로 새 Greengrass 코어 디바이스를 설정하거나 수동으로 프로비저닝된 리소스를 지정할 수 있습니다. 이 인수에 지정한 경로에 구성 파일을 마운트해야 합니다.

`TRUSTED_PLUGIN`

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다.

(선택 사항) 신뢰할 수 있는 플러그인으로 로드할 JAR 파일의 경로. 이 옵션을 사용하면 플릿 프로비저닝 또는 사용자 지정 프로비저닝으로 설치하는 등의 프로비저닝 플러그인 JAR 파일을 제공할 수 있습니다.

`THING_POLICY_NAME`

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다.

(선택 사항) 이 코어 디바이스의 사물 인증서에 첨부할 AWS IoT 정책의 이름. AWS IoT 이 이름을 가진 AWS IoT 정책이 사용자 내에 없는 경우 AWS IoT Greengrass 코어 소프트웨어에서 AWS 계정 해당 정책을 생성합니다.

이 인수를 `PROVISION=true` 적용하도록 지정해야 합니다.

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어는 기본적으로 허용 AWS IoT 정책을 생성합니다. 이 정책의 범위를 좁히거나 사용 사례에 대한 권한을 제한하는 사용자 지정 정책을 만들 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

설치할 종속성을 지정합니다.

AWS IoT GreengrassDockerfile의 RUN 명령은 Core 소프트웨어 설치 프로그램을 실행하기 위한 컨테이너 환경을 준비합니다. AWS IoT Greengrass Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 실행하기 전에 설치되는 종속성을 사용자 지정할 수 있습니다.

이미지 빌드 AWS IoT Greengrass

AWS IoT GreengrassDockerfile을 사용하여 컨테이너 이미지를 AWS IoT Greengrass 빌드합니다. Docker CLI 또는 Docker Compose CLI를 사용하여 이미지를 빌드하고 컨테이너를 시작할 수 있습니다. Docker CLI를 사용하여 이미지를 빌드한 다음 Docker Compose를 사용하여 해당 이미지에서 컨테이너를 시작할 수도 있습니다.

Docker

1. 호스트 시스템에서 다음 명령을 실행하여 구성된 Dockerfile이 포함된 디렉토리로 전환합니다.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. 다음 명령을 실행하여 Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지를 빌드합니다.

   ```
   sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
   ```

Docker Compose

1. 호스트 시스템에서 다음 명령을 실행하여 Dockerfile과 Compose 파일이 포함된 디렉터리로 전환합니다.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. 다음 명령을 실행하여 Compose 파일을 사용하여 컨테이너 이미지를 빌드합니다. AWS IoT Greengrass

   ```
   docker-compose -f docker-compose.yml build
   ```

AWS IoT Greengrass컨테이너 이미지를 성공적으로 생성했습니다. Docker 이미지에는 AWS IoT Greengrass Core 소프트웨어가 설치되어 있습니다. 이제 Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어를 실행할 수 있습니다.

자동 리소스 프로비저닝으로 Docker AWS IoT Greengrass 컨테이너에서 실행

이 자습서에서는 자동으로 AWS 프로비저닝된 리소스 및 로컬 개발 도구를 사용하여 Docker 컨테이너에 AWS IoT Greengrass Core 소프트웨어를 설치하고 실행하는 방법을 보여줍니다. 이 개발 환경을 사용하여 Docker 컨테이너의 AWS IoT Greengrass 기능을 탐색할 수 있습니다. 소프트웨어에서 이러한 리소스를 프로비저닝하고 로컬 개발 도구를 배포하려면 AWS 자격 증명이 필요합니다.

컨테이너에 AWS 자격 증명을 제공할 수 없는 경우 코어 장치가 작동하는 데 필요한 AWS 리소스를 프로비저닝할 수 있습니다. 개발 도구를 코어 디바이스에 배포하여 개발 디바이스로 사용할 수도 있습니다. 이렇게 하면 컨테이너를 실행할 때 디바이스에 제공하는 권한을 줄일 수 있습니다. 자세한 설명은 수동 리소스 프로비저닝으로 Docker AWS IoT Greengrass 컨테이너에서 실행 섹션을 참조하세요.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 계정이 없는 경우 AWS 계정 설정 섹션을 참조하십시오.
- Greengrass 코어 디바이스를 위한 AWS IAM 리소스 AWS IoT 및 IAM 리소스를 프로비저닝할 수 있는 권한을 가진 IAM 사용자입니다. AWS IoT GreengrassCore 소프트웨어 설치 프로그램은 AWS 사용자 자격 증명을 사용하여 이러한 리소스를 자동으로 프로비저닝합니다. 리소스를 자동으로 프로비저닝하기 위한 최소 IAM 정책에 대한 자세한 내용은 을 참조하십시오. 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책
- AWS IoT GreengrassDocker 이미지. AWS IoT GreengrassDockerfile에서 이미지를 빌드할 수 있습니다.
- Docker 컨테이너를 실행하는 호스트 컴퓨터는 다음 요구 사항을 충족해야 합니다.
  - 인터넷에 연결된 Linux 기반 운영 체제.
  - 도커 엔진 버전 18.09 이상
  - (선택 사항) 도커 컴포지션 버전 1.22 이상 Docker Compose는 Docker Compose CLI를 사용하여 Docker 이미지를 실행하려는 경우에만 필요합니다.

AWS 보안 인증 구성

이 단계에서는 호스트 컴퓨터에 보안 자격 증명이 포함된 자격 증명 파일을 생성합니다. AWS AWS IoT GreengrassDocker 이미지를 실행할 때는 이 자격 증명 파일이 포함된 폴더를 Docker `/root/.aws/` 컨테이너에 마운트해야 합니다. AWS IoT Greengrass설치 프로그램은 이러한 자격 증명

을 사용하여 사용자의 리소스를 프로비저닝합니다. AWS 계정 설치 프로그램이 리소스를 자동으로 프로비저닝하는 데 필요한 최소 IAM 정책에 대한 자세한 내용은 을 참조하십시오. 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책

1. 다음 중 하나를 검색하십시오.

   - IAM 사용자의 장기 자격 증명 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.
   - (권장) IAM 역할의 임시 자격 증명. 임시 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 임시 보안 자격 증명 사용을 참조하십시오. AWS CLI
2. 자격 증명 파일을 저장할 폴더를 생성하십시오.

   ```
   mkdir ./greengrass-v2-credentials
   ```
3. 텍스트 편집기를 사용하여 `credentials` `./greengrass-v2-credentials` 폴더에 이름이 지정된 구성 파일을 생성합니다.

   예를 들어, 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다. `credentials`

   ```
   nano ./greengrass-v2-credentials/credentials
   ```
4. 다음 형식으로 `credentials` 파일에 AWS 자격 증명을 추가합니다.

   ```
   [default]
   aws_access_key_id = AKIAIOSFODNN7EXAMPLE
   aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   aws_session_token
    = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
   ```

   임시 `aws_session_token` 자격 증명에만 포함하십시오.

> **Important**
>
> AWS IoT Greengrass컨테이너를 시작한 후 호스트 컴퓨터에서 자격 증명 파일을 제거합니다. 자격 증명 파일을 제거하지 않으면 AWS 자격 증명은 컨테이너 내부에 마운트된 상태로 유지됩니다. 자세한 설명은 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행합니다. 섹션을 참조하세요.

환경 파일 생성

이 자습서에서는 환경 파일을 사용하여 Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램으로 전달될 환경 변수를 설정합니다. 또한 docker run명령의 -e or --env 인수를 사용하여 Docker 컨테이너의 환경 변수를 설정하거나 파일의 environment블록에 변수를 설정할 수 있습니다. docker-compose.yml

1. 텍스트 편집기를 사용하여 라는 .env 환경 파일을 생성합니다.

   예를 들어, Linux 기반 .env 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 현재 디렉토리에 를 생성할 수 있습니다.

   ```
   nano .env
   ```

2. 다음 내용을 파일에 복사합니다.

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=true
   THING_NAME=MyGreengrassCore
   THING_GROUP_NAME=MyGreengrassCoreGroup
   TES_ROLE_NAME=GreengrassV2TokenExchangeRole
   TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   ```

   그런 다음 다음 값을 바꿉니다.

   - */greengrass/v2*. 설치에 사용하려는 Greengrass 루트 폴더입니다. GGC_ROOT환경 변수를 사용하여 이 값을 설정합니다.
   - *##*. 리소스를 생성한 AWS 리전 위치.
   - *MyGreengrassCore*. AWS IoT 사물의 이름입니다. 존재하지 않는 물건은 인스톨러가 생성합니다. 설치 프로그램은 인증서를 다운로드하여 사물로 인증합니다. AWS IoT
   - *MyGreengrassCoreGroup*. AWS IoT사물 그룹의 이름. 사물 그룹이 없는 경우 설치 프로그램은 사물 그룹을 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 배포가 활성화되어 있는 경우 코어 장치는 배포에서 지정하는 소프트웨어를 다운로드하고 실행합니다.
   - *#####v2 TokenExchangeRole*. Greengrass 코어 디바이스가 임시 자격 증명을 받을 AWS 수 있도록 허용하는 IAM 토큰 교환 역할의 이름으로 바꾸십시오. *### #### ## ## ## #### # ### #### GreenGrassv2 Access## ### #### #####. TokenExchangeRole* 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

- *GreengrassCoreTokenExchangeRoleAlias*. 토큰 교환 역할 별칭. 역할 별칭이 없는 경우 설치 프로그램은 역할 별칭을 생성하고 지정한 IAM 토큰 교환 역할을 가리킵니다. 자세한 내용을 알아보려면 다음 섹션을 참조하세요.

> **Note**
>
> DEPLOY_DEV_TOOLS환경 변수를 로 true 설정하여 Greengrass CLI 구성 요소를 배포할 수 있습니다. 이렇게 하면 Docker 컨테이너 내에서 사용자 지정 구성 요소를 개발할 수 있습니다. 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행합니다.

이 자습서에서는 Docker 컨테이너에 구축한 Docker 이미지를 시작하는 방법을 보여줍니다. Docker CLI 또는 Docker Compose CLI를 사용하여 Docker 컨테이너에서 코어 소프트웨어 이미지를 AWS IoT Greengrass 실행할 수 있습니다.

Docker

1. 다음 명령을 실행하여 Docker 컨테이너를 시작합니다.

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-credentials:/root/.aws/:ro \
 --env-file .env \
 -p 8883 \
 your-container-image:version
```

   이 예제 명령은 docker run에 다음 인수를 사용합니다.

   - --rm. 컨테이너가 나올 때 컨테이너를 청소합니다.
   - --init. 컨테이너에서 init 프로세스를 사용합니다.

> **Note**
>
> Docker 컨테이너를 중지할 때 AWS IoT Greengrass Core 소프트웨어를 종료하려면 `--init` 인수가 필요합니다.

- `-it`. (선택 사항) 포그라운드에서 Docker 컨테이너를 대화형 프로세스로 실행합니다. 이 `-d` 인수를 인수로 대체하여 대신 Docker 컨테이너를 분리 모드에서 실행할 수 있습니다. 자세한 내용은 Docker 설명서의 분리형 vs. 포그라운드를 참조하십시오.
- `--name`. 라는 이름의 컨테이너를 실행합니다. `aws-iot-greengrass`
- `-v`. Docker 컨테이너에 볼륨을 마운트하여 컨테이너 내에서 구성 파일과 인증서 파일을 AWS IoT Greengrass 실행할 수 있도록 합니다.
- `--env-file`. (선택 사항) Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램으로 전달될 환경 변수를 설정하는 환경 파일을 지정합니다. 이 인수는 환경 변수를 설정하기 위해 환경 파일을 만든 경우에만 필요합니다. 환경 파일을 만들지 않은 경우 Docker run 명령에서 `--env` 인수를 사용하여 환경 변수를 직접 설정할 수 있습니다.
- `-p`. (선택 사항) 8883 컨테이너 포트를 호스트 시스템에 게시합니다. MQTT 트래픽에 포트 8883을 AWS IoT Greengrass 사용하기 때문에 MQTT를 통해 연결하고 통신하려는 경우 이 인수가 필요합니다. 다른 포트를 열려면 추가 인수를 사용하십시오. `-p`

> **Note**
>
> 보안을 강화하여 Docker 컨테이너를 실행하려면 `--cap-drop` 및 `--cap-add` 인수를 사용하여 컨테이너의 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 런타임 권한 및 Linux 기능을 참조하십시오.

2. 호스트 `./greengrass-v2-credentials` 디바이스에서 자격 증명을 제거합니다.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> 이러한 자격 증명은 코어 디바이스가 설정 중에만 필요한 광범위한 권한을 제공하므로 이러한 자격 증명을 제거하는 것입니다. 이러한 자격 증명을 제거하지 않으면 컨테이너에서 실행 중인 Greengrass 구성 요소 및 기타 프로세스가 해당 자격 증명에 액세스할 수 있습니다. Greengrass 구성 요소에 AWS 자격 증명을 제공해야 하는 경우 토큰

교환 서비스를 사용하십시오. 자세한 설명은 AWS서비스와 상호작용 섹션을 참조하세요.

Docker Compose

1. 텍스트 편집기를 사용하여 이름이 지정된 Docker Compose 파일을 생성합니다. `docker-compose.yml`

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 현재 디렉터리에 파일을 생성할 수 있습니다. `docker-compose.yml`

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > 에서 제공하는 Compose 파일의 최신 버전을 다운로드하여 사용할 수도 있습니다 AWS. GitHub

2. Compose 파일에 다음 콘텐츠를 추가합니다. 파일은 다음 예제와 비슷할 것입니다. *## #### ## ####* 이름으로 바꾸세요.

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       container_name: aws-iot-greengrass
       image: docker-image
       volumes:
         - ./greengrass-v2-credentials:/root/.aws/:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

   이 예제 Compose 파일의 다음 매개변수는 선택사항입니다.

- `ports`—8883 컨테이너 포트를 호스트 컴퓨터에 게시합니다. MQTT 트래픽에 포트 8883을 AWS IoT Greengrass 사용하기 때문에 MQTT를 통해 연결하고 통신하려는 경우 이 매개 변수가 필요합니다.
- `env_file`—Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램으로 전달될 환경 변수를 설정하는 환경 파일을 지정합니다. 이 매개 변수는 환경 변수를 설정하기 위해 환경 파일을 만든 경우에만 필요합니다. 환경 파일을 만들지 않은 경우 환경 매개변수를 사용하여 Compose 파일에서 직접 변수를 설정할 수 있습니다.

> **Note**
>
> 보안을 강화하여 Docker 컨테이너를 실행하려면 Compose 파일에서 `cap_drop` 및 `cap_add` 를 사용하여 컨테이너의 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 런타임 권한 및 Linux 기능을 참조하십시오.

3. 다음 명령을 실행하여 Docker 컨테이너를 시작합니다.

```
docker-compose -f docker-compose.yml up
```

4. 호스트 `./greengrass-v2-credentials` 디바이스에서 자격 증명을 제거합니다.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> 이러한 자격 증명은 코어 디바이스가 설정 중에만 필요한 광범위한 권한을 제공하므로 이러한 자격 증명을 제거하는 것입니다. 이러한 자격 증명을 제거하지 않으면 컨테이너에서 실행 중인 Greengrass 구성 요소 및 기타 프로세스가 해당 자격 증명에 액세스할 수 있습니다. Greengrass 구성 요소에 AWS 자격 증명을 제공해야 하는 경우 토큰 교환 서비스를 사용하십시오. 자세한 설명은 AWS서비스와 상호작용 섹션을 참조하세요.

다음 단계

AWS IoT Greengrass코어 소프트웨어는 이제 Docker 컨테이너에서 실행되고 있습니다. 다음 명령을 실행하여 현재 실행 중인 컨테이너의 컨테이너 ID를 검색합니다.

```
docker ps
```

그런 다음 다음 명령을 실행하여 컨테이너에 액세스하고 컨테이너 내에서 실행되는 AWS IoT Greengrass Core 소프트웨어를 탐색할 수 있습니다.

```
docker exec -it container-id /bin/bash
```

간단한 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오4단계: 기기의 구성 요소 개발 및 테스트. 자습서: AWS IoT Greengrass V2 시작하기

> **Note**
>
> 를 `docker exec` 사용하여 Docker 컨테이너 내에서 명령을 실행하면 해당 명령이 Docker 로그에 기록되지 않습니다. Docker 로그에 명령을 기록하려면 대화형 셸을 Docker 컨테이너에 연결하세요. 자세한 설명은 대화형 셸을 Docker 컨테이너에 연결합니다. 섹션을 참조하세요.

AWS IoT GreengrassCore 로그 파일이 `greengrass.log` 호출되고 위치에 있습니다. */greengrass/v2*`/logs` 구성 요소 로그 파일도 같은 디렉터리에 있습니다. Greengrass 로그를 호스트의 임시 디렉터리에 복사하려면 다음 명령을 실행합니다.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

컨테이너가 종료되거나 제거된 후에도 로그를 유지하려면 전체 Greengrass */greengrass/v2*`/logs` 디렉터리를 마운트하는 대신 디렉터리만 호스트의 임시 로그 디렉터리에 바인드 마운트하는 것이 좋습니다. 자세한 설명은 Docker 컨테이너 외부에서 Greengrass 로그 유지 섹션을 참조하세요.

실행 중인 AWS IoT Greengrass Docker 컨테이너를 중지하려면 또는 를 실행하십시오. `docker stop` `docker-compose -f docker-compose.yml stop` 이 작업은 Greengrass `SIGTERM` 프로세스로 전송되고 컨테이너에서 시작된 모든 관련 프로세스를 종료합니다. Docker 컨테이너는 `docker-init` 실행 파일을 프로세스 PID 1로 초기화하므로 남아 있는 좀비 프로세스를 제거하는 데 도움이 됩니다. 자세한 내용은 Docker 설명서의 초기화 프로세스 지정을 참조하십시오.

Docker AWS IoT Greengrass 컨테이너에서 실행할 때 발생하는 문제 해결에 대한 자세한 내용은 을 참조하십시오. 도커 컨테이너에서 AWS IoT Greengrass 문제 해결

수동 리소스 프로비저닝으로 Docker AWS IoT Greengrass 컨테이너에서 실행

이 자습서에서는 수동으로 프로비저닝된 리소스를 사용하여 Docker 컨테이너에 AWS IoT Greengrass Core 소프트웨어를 설치하고 실행하는 방법을 보여줍니다. AWS

주제

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- AWS 계정. 계정이 없는 경우 AWS 계정 설정 섹션을 참조하십시오.
- AWS IoT GreengrassDocker 이미지. AWS IoT GreengrassDockerfile에서 이미지를 빌드할 수 있습니다.
- Docker 컨테이너를 실행하는 호스트 컴퓨터는 다음 요구 사항을 충족해야 합니다.
  - 인터넷에 연결된 Linux 기반 운영 체제.
  - 도커 엔진 버전 18.09 이상
  - (선택 사항) 도커 컴포지션 버전 1.22 이상 Docker Compose는 Docker Compose CLI를 사용하여 Docker 이미지를 실행하려는 경우에만 필요합니다.

AWS IoT엔드포인트를 검색합니다.

AWS IoT엔드포인트를 가져와서 나중에 사용할 수 있도록 저장하세요AWS 계정. 장치는 이러한 엔드포인트를 사용하여 연결합니다. AWS IoT 다음을 따릅니다.

1. 사용자의 AWS IoT 데이터 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 사용자의 AWS IoT 자격 증명 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

AWS IoT 사물 생성

AWS IoT사물은 연결된 장치 및 논리적 개체를 나타냅니다. AWS IoT Greengrass 코어 디바이스는 AWS IoT 사물입니다. 장치를 사물로 등록하면 해당 장치가 디지털 인증서를 사용하여 인증할 수 있습니다. AWS IoT AWS

이 섹션에서는 디바이스를 나타내는 AWS IoT 사물을 생성합니다.

AWS IoT 사물을 생성하려면

1. AWS IoT디바이스용 사물을 만드세요. 개발 컴퓨터에서 다음 명령을 실행합니다.

   - 사용할 사물 *MyGreengrassCore*이름으로 바꿉니다. 이 이름은 Greengrass 코어 디바이스의 이름이기도 합니다.

> **Note**
>
> 사물 이름에는 콜론 (:) 문자를 포함할 수 없습니다.

```
aws iot create-thing --thing-name MyGreengrassCore
```

요청이 성공하면 응답은 다음 예제와 비슷합니다.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (선택 사항) 새 AWS IoT 사물 그룹 또는 기존 사물 그룹에 사물을 추가합니다. 사물 그룹을 사용하여 수많은 Greengrass 코어 디바이스를 관리합니다. 소프트웨어 구성 요소를 장치에 배포할 때 개별 장치 또는 장치 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포가 있는 사물 그룹에 장치를 추가하여 해당 사물 그룹의 소프트웨어 구성 요소를 장치에 배포할 수 있습니다. 다음을 따릅니다.

   a. (선택 사항) AWS IoT 사물 그룹을 생성합니다.

      - 생성할 사물 그룹의 *MyGreengrassCoreGroup*이름으로 바꿉니다.

      > **Note**
      >
      > 사물 그룹 이름에는 콜론 (:) 문자를 포함할 수 없습니다.

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      요청이 성공하면 응답은 다음 예와 비슷합니다.

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
      ```

```
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. AWS IoT사물 그룹에 사물을 추가합니다.

- 사물 *MyGreengrassCore*이름으로 바꾸세요AWS IoT.
- 사물 그룹의 *MyGreengrassCoreGroup*이름으로 바꾸십시오.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

요청이 성공하면 명령이 출력되지 않습니다.

사물 인증서를 생성하세요.

장치를 사물로 등록하면 해당 장치가 디지털 인증서를 사용하여 인증할 수 있습니다. AWS IoT AWS 이 인증서를 사용하면 장치가 AWS IoT 및 AWS IoT Greengrass 와 통신할 수 있습니다.

이 섹션에서는 장치를 연결하는 데 사용할 수 있는 인증서를 만들고 AWS 다운로드합니다.

사물 인증서를 만들려면

1. 사물에 대한 인증서를 다운로드할 폴더를 AWS IoT 생성합니다.

```
mkdir greengrass-v2-certs
```

2. 사물에 대한 인증서를 생성하고 AWS IoT 다운로드합니다.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
```

```
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

나중에 인증서를 구성하는 데 사용할 인증서의 Amazon 리소스 이름 (ARN) 을 저장합니다.

사물 인증서를 구성합니다.

이전에 만든 사물에 AWS IoT 사물 인증서를 연결하고 핵심 장치에 대한 AWS IoT 권한을 정의하는 AWS IoT 정책을 인증서에 추가합니다.

사물 인증서를 구성하려면

1. 인증서를 사물에 AWS IoT 첨부합니다.

   - AWS IoT물건 *MyGreengrassCore*이름으로 바꾸세요.
   - 인증서 Amazon 리소스 이름 (ARN) 을 이전 단계에서 생성한 인증서의 ARN으로 교체합니다.

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   요청이 성공하면 명령이 출력되지 않습니다.

2. Greengrass 코어 AWS IoT 디바이스에 대한 권한을 정의하는 AWS IoT 정책을 생성하여 첨부하십시오. 다음 정책은 모든 MQTT 주제 및 Greengrass 작업에 대한 액세스를 허용하므로 장치가 사용자 지정 애플리케이션 및 새로운 Greengrass 작업이 필요한 향후 변경 사항과 함께 작동하도록 할 수 있습니다. 사용 사례에 따라 이 정책을 제한할 수 있습니다. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

   이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 해당 AWS IoT 정책을 연결할 수 있습니다.

   다음을 따릅니다.

   a. Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

      ```
      nano greengrass-v2-iot-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "iot:Publish",
      ```

```
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

b. AWS IoT정책 문서에서 정책을 생성합니다.

- *GreenGrassV2IoT# ThingPolicy* 생성할 정책의 이름으로 바꾸십시오.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
```

```
  }",
  "policyVersionId": "1"
}
```

c. AWS IoT정책을 사물의 인증서에 연결합니다. AWS IoT

- *GreenGrassV2IoT# ThingPolicy ###* 정책 이름으로 바꾸십시오.
- 대상 ARN을 사내 인증서의 ARN으로 바꾸십시오. AWS IoT

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청이 성공하면 명령이 출력되지 않습니다.

토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라고 하는 IAM 서비스 역할을 사용하여 서비스 호출을 승인합니다. AWS 디바이스는 AWS IoT 자격 증명 공급자를 사용하여 이 역할에 대한 임시 AWS 자격 증명을 가져오고, 이를 통해 디바이스가 Amazon Logs와 상호 작용하고AWS IoT, Amazon Logs에 로그를 전송하고, Amazon CloudWatch S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

AWS IoT역할 별칭을 사용하여 Greengrass 코어 디바이스의 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 장치의 토큰 교환 역할을 변경하면서도 장치 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core개발자 안내서의 AWS서비스에 대한 직접 호출 승인을 참조하십시오.

이 섹션에서는 토큰 교환 IAM 역할과 해당 역할을 가리키는 AWS IoT 역할 별칭을 생성합니다. Greengrass 코어 디바이스를 이미 설정한 경우 새 디바이스를 생성하는 대신 토큰 교환 역할 및 역할 별칭을 사용할 수 있습니다. 그런 다음 해당 역할과 별칭을 사용하도록 디바이스의 AWS IoT 사물을 구성합니다.

토큰 교환 IAM 역할을 만들려면

1. 디바이스에서 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 다음을 따릅니다.

   a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 들어 있는 파일을 생성하십시오.

예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

```
nano device-role-trust-policy.json
```

다음 JSON을 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 신뢰 정책 문서를 사용하여 토큰 교환 역할을 생성합니다.

- *GreenGrassV2# TokenExchangeRole* 생성할 IAM 역할의 이름으로 바꾸십시오.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
```

```
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. 토큰 교환 역할에 필요한 액세스 정책 문서가 포함된 파일을 생성하십시오.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 이 액세스 정책은 S3 버킷의 구성 요소 아티팩트에 대한 액세스를 허용하지 않습니다. Amazon S3에 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자

세한 설명은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용 섹션을 참조하세요.
구성 요소 아티팩트를 위한 S3 버킷이 아직 없는 경우, 버킷을 생성한 후 나중에 이러한 권한을 추가할 수 있습니다.

d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreenGrassV2# TokenExchangeRoleAccess* 생성할 IAM 정책의 이름으로 바꾸십시오.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. IAM 정책을 토큰 교환 역할에 연결합니다.

- *GreenGrassV2# TokenExchangeRole IAM* 역할 이름으로 바꾸십시오.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 교체합니다.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

요청이 성공하면 명령이 출력되지 않습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

   - 생성할 역할 별칭의 *GreengrassCoreTokenExchangeRoleAlias*이름으로 바꾸십시오.
   - 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 교체합니다.

   ```
   aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
   arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
   ```

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
     "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
   GreengrassCoreTokenExchangeRoleAlias"
   }
   ```

   > **Note**
   >
   > 역할 별칭을 생성하려면 토큰 교환 IAM 역할을 전달할 권한이 있어야 합니다. AWS IoT 역할 별칭을 만들려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 이 권한이 있는지 확인하세요. 자세한 내용은 사용 설명서의 AWS 서비스에 역할을 전달할 수 있는 권한 부여를 AWS Identity and Access Management 참조하십시오.

3. Greengrass 코어 디바이스가 역할 별칭을 사용하여 토큰 교환 역할을 맡도록 허용하는 AWS IoT 정책을 만들고 첨부하십시오. 이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 역할 별칭 AWS IoT 정책을 연결할 수 있습니다. 다음을 따릅니다.

   a. (선택 사항) 역할 별칭에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

      예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      다음 JSON을 파일에 복사합니다.

- 리소스 ARN을 역할 별칭의 ARN으로 대체합니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. AWS IoT정책 문서에서 정책을 생성합니다.

- 생성할 AWS IoT 정책의 *GreengrassCoreTokenExchangeRoleAliasPolicy*이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
```

```
}
```

c. AWS IoT정책을 사물의 인증서에 연결합니다. AWS IoT

- 역할 별칭 AWS IoT 정책의 *GreengrassCoreTokenExchangeRoleAliasPolicy*이름으로 바꾸십시오.
- 대상 ARN을 사내 인증서의 ARN으로 바꾸십시오. AWS IoT

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청이 성공하면 명령이 출력되지 않습니다.

인증서를 디바이스에 다운로드합니다.

이전에 디바이스의 인증서를 개발 컴퓨터에 다운로드했습니다. 이 섹션에서는 Amazon 루트 인증 기관 (CA) 인증서를 다운로드합니다. 그런 다음 개발 컴퓨터가 아닌 다른 컴퓨터에서 Docker의 AWS IoT Greengrass Core 소프트웨어를 실행하려는 경우 인증서를 해당 호스트 컴퓨터에 복사합니다. AWS IoT GreengrassCore 소프트웨어는 이러한 인증서를 사용하여 AWS IoT 클라우드 서비스에 연결합니다.

인증서를 디바이스에 다운로드하려면

1. 개발 컴퓨터에서 Amazon 루트 인증 기관 (CA) 인증서를 다운로드합니다. AWS IoT인증서는 기본적으로 Amazon의 루트 CA 인증서와 연결됩니다.

   Linux or Unix

   ```
   sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://
   www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .
\greengrass-v2-certs\AmazonRootCA1.pem
```

2. 개발 컴퓨터와 다른 디바이스에서 Docker의 AWS IoT Greengrass Core 소프트웨어를 실행하려는 경우 인증서를 호스트 컴퓨터에 복사하십시오. 개발 컴퓨터와 호스트 컴퓨터에서 SSH와 SCP를 사용하도록 설정한 경우 개발 컴퓨터의 `scp` 명령을 사용하여 인증서를 전송할 수 있습니다. 호스트 컴퓨터의 IP 주소로 *device-ip-address*바꾸십시오.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

구성 파일 생성

1. 호스트 컴퓨터에서 구성 파일을 저장할 폴더를 생성합니다.

```
mkdir ./greengrass-v2-config
```

2. 텍스트 편집기를 사용하여 `config.yaml` `./greengrass-v2-config` 폴더에 이름이 지정된 구성 파일을 생성합니다.

   예를 들어, 다음 명령을 실행하여 GNU nano를 사용하여 생성할 수 있습니다. `config.yaml`

```
nano ./greengrass-v2-config/config.yaml
```

3. 다음 YAML 콘텐츠를 파일에 복사합니다. 이 부분 구성 파일은 시스템 매개변수와 Greengrass 핵 매개변수를 지정합니다.

```
---
system:
  certificateFilePath: "/tmp/certs/device.pem.crt"
  privateKeyPath: "/tmp/certs/private.pem.key"
  rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "nucleus-version"
```

```
    configuration:
      awsRegion: "region"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
```

그런 다음 다음 값을 바꾸십시오.

- */tmp/certs.* 컨테이너를 시작할 때 다운로드한 인증서를 마운트하는 Docker 컨테이너의 디렉터리입니다.
- */greengrass/v2*. 설치에 사용하려는 Greengrass 루트 폴더입니다. GGC_ROOT환경 변수를 사용하여 이 값을 설정합니다.
- *MyGreengrassCore*. AWS IoT 사물의 이름입니다.
- *# ##.* 설치할 AWS IoT Greengrass Core 소프트웨어 버전입니다. 이 값은 다운로드한 Docker 이미지 또는 Dockerfile의 버전과 일치해야 합니다. latest태그와 함께 Greengrass Docker 이미지를 다운로드한 경우 **docker inspect *image-id*** 를 사용하여 이미지 버전을 확인하세요.
- *##.* AWS IoT리소스를 생성한 AWS 리전 곳. 또한 환경 파일의 AWS_REGION 환경 변수에도 동일한 값을 지정해야 합니다.
- *GreengrassCoreTokenExchangeRoleAlias*. 토큰 교환 역할 별칭.
- *device-data-prefix*. AWS IoT데이터 엔드포인트의 접두사.
- *device-credentials-prefix*. AWS IoT자격 증명 엔드포인트의 접두사.

환경 파일 생성

이 자습서에서는 환경 파일을 사용하여 Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램으로 전달될 환경 변수를 설정합니다. 또한 docker run명령의 -e or --env 인수를 사용하여 Docker 컨테이너의 환경 변수를 설정하거나 파일의 environment블록에 변수를 설정할 수 있습니다. docker-compose.yml

1. 텍스트 편집기를 사용하여 라는 .env 환경 파일을 생성합니다.

   예를 들어, Linux 기반 .env 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 현재 디렉토리에 를 생성할 수 있습니다.

   ```
   nano .env
   ```

2. 다음 내용을 파일에 복사합니다.

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=false
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
INIT_CONFIG=/tmp/config/config.yaml
```

그런 다음 다음 값을 바꿉니다.

- */greengrass/v2*. AWS IoT GreengrassCore 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.
- *##*. AWS IoT리소스를 생성한 AWS 리전 곳. 구성 파일의 awsRegion 구성 매개변수에도 동일한 값을 지정해야 합니다.
- */tmp/config/.* Docker 컨테이너를 시작할 때 구성 파일을 마운트하는 폴더입니다.

> **Note**
>
> DEPLOY_DEV_TOOLS환경 변수를 로 true 설정하여 Greengrass CLI 구성 요소를 배포할 수 있습니다. 이렇게 하면 Docker 컨테이너 내에서 사용자 지정 구성 요소를 개발할 수 있습니다. 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행합니다.

이 자습서에서는 Docker 컨테이너에 구축한 Docker 이미지를 시작하는 방법을 보여줍니다. Docker CLI 또는 Docker Compose CLI를 사용하여 Docker 컨테이너에서 코어 소프트웨어 이미지를 AWS IoT Greengrass 실행할 수 있습니다.

Docker

- 이 가이드에서는 Docker 컨테이너에 구축한 Docker 이미지를 시작하는 방법을 보여줍니다.

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-config:/tmp/config/:ro \
```

```
-v path/to/greengrass-v2-certs:/tmp/certs:ro \
--env-file .env \
-p 8883 \
your-container-image:version
```

이 예제 명령은 docker run에 다음 인수를 사용합니다.

- `--rm`. 컨테이너가 나올 때 컨테이너를 청소합니다.
- `--init`. 컨테이너에서 init 프로세스를 사용합니다.

  > **Note**
  >
  > Docker 컨테이너를 중지할 때 AWS IoT Greengrass Core 소프트웨어를 종료하려면 `--init` 인수가 필요합니다.

- `-it`. (선택 사항) 포그라운드에서 대화형 프로세스로 Docker 컨테이너를 실행합니다. 이 `-d` 인수를 인수로 대체하여 대신 Docker 컨테이너를 분리 모드에서 실행할 수 있습니다. 자세한 내용은 Docker 설명서의 분리형 vs. 포그라운드를 참조하십시오.
- `--name`. 라는 이름의 컨테이너를 실행합니다. `aws-iot-greengrass`
- `-v`. Docker 컨테이너에 볼륨을 마운트하여 컨테이너 내에서 구성 파일과 인증서 파일을 AWS IoT Greengrass 실행할 수 있도록 합니다.
- `--env-file`. (선택 사항) Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램으로 전달될 환경 변수를 설정하는 환경 파일을 지정합니다. 이 인수는 환경 변수를 설정하기 위해 환경 파일을 만든 경우에만 필요합니다. 환경 파일을 만들지 않은 경우 Docker run 명령에서 `--env` 인수를 사용하여 환경 변수를 직접 설정할 수 있습니다.
- `-p`. (선택 사항) 8883 컨테이너 포트를 호스트 시스템에 게시합니다. MQTT 트래픽에 포트 8883을 AWS IoT Greengrass 사용하기 때문에 MQTT를 통해 연결하고 통신하려는 경우 이 인수가 필요합니다. 다른 포트를 열려면 추가 인수를 사용하십시오. `-p`

> **Note**
>
> 보안을 강화하여 Docker 컨테이너를 실행하려면 `--cap-drop` 및 `--cap-add` 인수를 사용하여 컨테이너의 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 런타임 권한 및 Linux 기능을 참조하십시오.

Docker Compose

1. 텍스트 편집기를 사용하여 라는 이름의 Docker Compose 파일을 만드십시오. `docker-compose.yml`

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 현재 디렉터리에 파일을 생성할 수 있습니다. `docker-compose.yml`

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > 에서 제공하는 Compose 파일의 최신 버전을 다운로드하여 사용할 수도 있습니다 AWS. GitHub

2. Compose 파일에 다음 콘텐츠를 추가합니다. 파일은 다음 예제와 비슷할 것입니다. :version을 *your-container-nameDocker* 이미지 이름으로 바꾸세요.

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       build:
         context: .
       container_name: aws-iot-greengrass
       image: your-container-name:version
       volumes:
         - /path/to/greengrass-v2-config:/tmp/config/:ro
         - /path/to/greengrass-v2-certs:/tmp/certs:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

   이 예제 작성 파일의 다음 매개변수는 선택사항입니다.

   - `ports`—8883 컨테이너 포트를 호스트 컴퓨터에 게시합니다. MQTT 트래픽에 포트 8883을 AWS IoT Greengrass 사용하기 때문에 MQTT를 통해 연결하고 통신하려는 경우 이 매개 변수가 필요합니다.

- `env_file`—Docker 컨테이너 내의 AWS IoT Greengrass Core 소프트웨어 설치 프로그램으로 전달될 환경 변수를 설정하는 환경 파일을 지정합니다. 이 매개 변수는 환경 변수를 설정하기 위해 환경 파일을 만든 경우에만 필요합니다. 환경 파일을 만들지 않은 경우 환경 매개변수를 사용하여 Compose 파일에서 직접 변수를 설정할 수 있습니다.

> **Note**
>
> 보안을 강화하여 Docker 컨테이너를 실행하려면 Compose 파일에서 `cap_drop` 및 `cap_add` 를 사용하여 컨테이너의 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 런타임 권한 및 Linux 기능을 참조하십시오.

3. 다음 명령을 실행하여 컨테이너를 시작합니다.

```
docker-compose -f docker-compose.yml up
```

다음 단계

AWS IoT Greengrass이제 Core 소프트웨어가 Docker 컨테이너에서 실행되고 있습니다. 다음 명령을 실행하여 현재 실행 중인 컨테이너의 컨테이너 ID를 검색합니다.

```
docker ps
```

그런 다음 다음 명령을 실행하여 컨테이너에 액세스하고 컨테이너 내에서 실행되는 AWS IoT Greengrass Core 소프트웨어를 탐색할 수 있습니다.

```
docker exec -it container-id /bin/bash
```

간단한 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오4단계: 기기의 구성 요소 개발 및 테스트. 자습서: AWS IoT Greengrass V2 시작하기

> **Note**
>
> 를 `docker exec` 사용하여 Docker 컨테이너 내에서 명령을 실행하면 해당 명령이 Docker 로그에 기록되지 않습니다. Docker 로그에 명령을 기록하려면 대화형 셸을 Docker 컨테이너에 연결하세요. 자세한 설명은 대화형 셸을 Docker 컨테이너에 연결합니다. 섹션을 참조하세요.

AWS IoT GreengrassCore 로그 파일이 `greengrass.log` 호출되고 위치에 있습니다. */greengrass/v2*/logs 구성 요소 로그 파일도 같은 디렉터리에 있습니다. Greengrass 로그를 호스트의 임시 디렉터리에 복사하려면 다음 명령을 실행합니다.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

컨테이너가 종료되거나 제거된 후에도 로그를 유지하려면 전체 Greengrass */greengrass/v2*/logs 디렉터리를 마운트하는 대신 디렉터리만 호스트의 임시 로그 디렉터리에 바인드 마운트하는 것이 좋습니다. 자세한 설명은 Docker 컨테이너 외부에서 Greengrass 로그 유지 섹션을 참조하세요.

실행 중인 AWS IoT Greengrass Docker 컨테이너를 중지하려면 또는 를 실행하십시오. `docker stop docker-compose -f docker-compose.yml stop` 이 작업은 Greengrass `SIGTERM` 프로세스로 전송되고 컨테이너에서 시작된 모든 관련 프로세스를 종료합니다. Docker 컨테이너는 `docker-init` 실행 파일을 프로세스 PID 1로 초기화하므로 남아 있는 좀비 프로세스를 제거하는 데 도움이 됩니다. 자세한 내용은 Docker 설명서의 초기화 프로세스 지정을 참조하십시오.

Docker AWS IoT Greengrass 컨테이너에서 실행할 때 발생하는 문제 해결에 대한 자세한 내용은 을 참조하십시오. 도커 컨테이너에서 AWS IoT Greengrass 문제 해결

도커 컨테이너에서 AWS IoT Greengrass 문제 해결

다음 정보를 사용하면 Docker AWS IoT Greengrass 컨테이너에서의 실행 문제를 해결하고 Docker 컨테이너에서의 문제를 디버깅하는 데 도움이 됩니다. AWS IoT Greengrass

주제

- Docker 컨테이너 실행 관련 문제 해결
- 도커 컨테이너에서 AWS IoT Greengrass 디버깅

Docker 컨테이너 실행 관련 문제 해결

다음 정보를 사용하면 도커 컨테이너에서 AWS IoT Greengrass 실행 중 발생하는 문제를 해결하는 데 도움이 됩니다.

주제

- 오류: TTY가 아닌 장치에서 대화형 로그인을 수행할 수 없습니다.
- 오류: 알 수 없는 옵션: - no-include-email
- 오류: 방화벽이 Windows와 컨테이너 간의 파일 공유를 차단하고 있습니다.

- <user-name>오류: GetAuthorizationToken 작업을 호출하는 동안 오류가 발생했습니다 (AccessDeniedException). 사용자: arn:aws:iam:: account-id:user/가 수행할 권한이 없음: ecr: 리소스: * GetAuthorizationToken
- 오류: 풀 레이트 한도에 도달했습니다.

오류: TTY가 아닌 장치에서 대화형 로그인을 수행할 수 없습니다.

이 오류는 `aws ecr get-login-password` 명령을 실행할 때 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 참조하십시오.

오류: 알 수 없는 옵션: - no-include-email

이 오류는 `aws ecr get-login` 명령을 실행할 때 발생할 수 있습니다. 최신 AWS CLI 버전이 설치되어 있는지 확인합니다(예를 들어, `pip install awscli --upgrade --user` 실행). 자세한 내용은 AWS Command Line Interface 사용 설명서의 Microsoft Windows에 AWS Command Line Interface 설치를 참조하십시오.

오류: 방화벽이 Windows와 컨테이너 간의 파일 공유를 차단하고 있습니다.

Windows 컴퓨터에서 Docker를 실행할 때 이 오류 또는 `Firewall Detected` 메시지가 표시될 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.

<user-name>오류: GetAuthorizationToken 작업을 호출하는 동안 오류가 발생했습니다 (AccessDeniedException). 사용자: arn:aws:iam:: ***account-id:user/#*** 수행할 권한이 없음: ecr: 리소스: * 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: 풀 레이트 한도에 도달했습니다.

Docker Hub는 익명 및 무료 Docker Hub 사용자가 만들 수 있는 풀 요청 수를 제한합니다. 익명 또는 무료 사용자 풀 요청의 속도 한도를 초과하면 다음 오류 중 하나가 발생합니다.

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

이러한 오류를 해결하려면 몇 시간 정도 기다린 후 다른 풀 리퀘스트를 시도할 수 있습니다. 지속적으로 많은 수의 풀 리퀘스트를 제출할 계획이라면 Docker Hub 웹 사이트에서 속도 제한에 대한 정보와 Docker 계정 인증 및 업그레이드 옵션을 참조하십시오.

도커 컨테이너에서 AWS IoT Greengrass 디버깅

도커 컨테이너로 문제를 디버그하려면 Greengrass 런타임 로그를 유지하거나 대화형 셸을 도커 컨테이너에 연결할 수 있습니다.

Docker 컨테이너 외부에서 Greengrass 로그 유지

AWS IoT Greengrass컨테이너를 중지한 후 다음 `docker cp` 명령을 사용하여 Docker 컨테이너의 Greengrass 로그를 임시 로그 디렉터리로 복사할 수 있습니다.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

컨테이너가 종료되거나 제거된 후에도 로그를 유지하려면 디렉터리를 바인드 마운트한 후 AWS IoT Greengrass Docker 컨테이너를 실행해야 합니다. */greengrass/v2*/logs

/greengrass/v2/logs디렉터리를 바인드 마운트하려면 새 Docker 컨테이너를 실행할 때 다음 중 하나를 수행하십시오. AWS IoT Greengrass

- 명령에 `-v /tmp/logs:/greengrass/v2/logs:ro` 포함시키십시오. `docker run`

  `docker-compose up`명령을 실행하기 전에 Compose 파일의 `volumes` 블록을 수정하여 다음 줄을 포함하세요.

  ```
  volumes:
   - /tmp/logs:/greengrass/v2/logs:ro
  ```

그런 다음 호스트의 로그를 확인하여 Docker 컨테이너 내에서 실행 AWS IoT Greengrass 중인 Greengrass 로그를 확인할 수 있습니다. */tmp/logs*

Greengrass Docker 컨테이너 실행에 대한 자세한 내용은 및 을 참조하십시오. 수동 프로비저닝으로 AWS IoT Greengrass Docker에서 실행 자동 프로비저닝으로 AWS IoT Greengrass Docker에서 실행

대화형 셸을 Docker 컨테이너에 연결합니다.

를 `docker exec` 사용하여 Docker 컨테이너 내에서 명령을 실행하는 경우 해당 명령은 Docker 로그에 캡처되지 않습니다. Docker 로그에 명령을 기록하면 Greengrass Docker 컨테이너의 상태를 조사하는 데 도움이 될 수 있습니다. 다음 중 하나를 수행합니다.

- 별도의 터미널에서 다음 명령을 실행하여 터미널의 표준 입력, 출력 및 오류를 실행 중인 컨테이너에 연결합니다. 이렇게 하면 현재 터미널에서 Docker 컨테이너를 보고 제어할 수 있습니다.

  ```
  docker attach container-id
  ```

- 별도의 터미널에서 다음 명령을 실행합니다. 이렇게 하면 컨테이너가 연결되어 있지 않아도 대화형 모드에서 명령을 실행할 수 있습니다.

  ```
  docker exec -it container-id sh -c "command > /proc/1/fd/1"
  ```

일반적인 AWS IoT Greengrass 문제 해결은 을 참조하십시오문제 해결.

AWS IoT Greengrass Core 소프트웨어 구성

AWS IoT Greengrass Core 소프트웨어는 소프트웨어를 구성하는 데 사용할 수 있는 옵션을 제공합니다. 배포를 생성하여 각 AWS IoT Greengrass 코어 장치에서 Core 소프트웨어를 구성할 수 있습니다.

주제

- Greengrass 핵 구성 요소 배포하기
- Greengrass 핵을 시스템 서비스로 구성
- JVM 옵션으로 메모리 할당을 제어하세요.
- 구성 요소를 실행하는 사용자를 구성하십시오.
- 구성 요소에 대한 시스템 리소스 제한을 구성합니다.
- 포트 443에서 또는 네트워크 프록시를 통해 연결
- 사설 CA에서 서명한 장치 인증서를 사용하십시오.
- MQTT 타임아웃 및 캐시 설정을 구성합니다.

Greengrass 핵 구성 요소 배포하기

AWS IoT Greengrass 는 Greengrass AWS IoT Greengrass 코어 장치에 배포할 수 있는 구성 요소로서 Core 소프트웨어를 제공합니다. 배포를 생성하여 여러 Greengrass 코어 디바이스에 동일한 구성을 적용할 수 있습니다. 자세한 내용은 그린그래스 핵 및 AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트 섹션을 참조하세요.

Greengrass 핵을 시스템 서비스로 구성

다음을 수행하려면 기기의 init 시스템에서 AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 구성해야 합니다.

- 기기가 부팅되면 AWS IoT Greengrass Core 소프트웨어를 시작합니다. 대량의 디바이스를 관리하는 경우 이 방법을 사용하는 것이 좋습니다.
- 플러그인 구성 요소를 설치하고 실행합니다. AWS제공되는 몇 가지 구성 요소는 Greengrass 핵과 직접 인터페이스할 수 있는 플러그인 구성 요소입니다. 구성 요소 유형에 대한 자세한 정보는 구성 요소 유형을(를) 참조하세요.
- 코어 디바이스의 AWS IoT Greengrass Core 소프트웨어에 업데이트 적용 over-the-air (OTA) 자세한 설명은 AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트 섹션을 참조하세요.
- 배포에서 AWS IoT Greengrass 구성 요소를 새 버전으로 업데이트하거나 특정 구성 매개 변수를 업데이트할 때 구성 요소가 Core 소프트웨어 또는 코어 장치를 다시 시작할 수 있도록 합니다. 자세한 내용은 부트스트랩 수명 주기 단계를 참조하십시오.

> **Important**
>
> Windows 코어 장치에서는 AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 설정해야 합니다.

주제

- Nucleus를 시스템 서비스로 구성 (Linux)
- Nucleus를 시스템 서비스로 구성 (Windows)

Nucleus를 시스템 서비스로 구성 (Linux)

리눅스 기기는 initd, systemd, Systemd 및 SystemV와 같은 다양한 초기화 시스템을 지원합니다. AWS IoT Greengrass Core 소프트웨어를 설치할 때 `--setup-system-service true` 인수를 사용하여 Nucleus를 시스템 서비스로 시작하고 장치 부팅 시 시작되도록 구성합니다. 설치 프로그램은 AWS IoT Greengrass Core 소프트웨어를 systemd의 시스템 서비스로 구성합니다.

Nucleus가 시스템 서비스로 실행되도록 수동으로 구성할 수도 있습니다. 다음은 systemd를 위한 서비스 파일의 예입니다.

```
[Unit]
Description=Greengrass Core

[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

시스템 서비스를 구성한 후 다음 명령을 실행하여 부팅 시 장치 시작을 구성하고 AWS IoT Greengrass Core 소프트웨어를 시작 또는 중지할 수 있습니다.

- 서비스 상태를 확인하려면 (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

- 장치 부팅 시 Nucleus가 시작되도록 하기 위함입니다.

  ```
  sudo systemctl enable greengrass.service
  ```

- 장치 부팅 시 핵이 시작되지 않도록 하기 위함입니다.

  ```
  sudo systemctl disable greengrass.service
  ```

- AWS IoT Greengrass Core 소프트웨어를 시작하려면

```
sudo systemctl start greengrass.service
```

- AWS IoT Greengrass Core 소프트웨어를 중지하려면

```
sudo systemctl stop greengrass.service
```

Nucleus를 시스템 서비스로 구성 (Windows)

AWS IoT Greengrass Core 소프트웨어를 설치할 때 `--setup-system-service true` 인수를 사용하여 Nucleus를 Windows 서비스로 시작하고 장치 부팅 시 시작되도록 구성합니다.

서비스를 구성한 후 다음 명령을 실행하여 부팅 시 디바이스 시작 및 AWS IoT Greengrass Core 소프트웨어 시작 또는 중지를 구성할 수 있습니다. 이러한 명령을 실행하려면 명령 프롬프트를 실행하거나 관리자 PowerShell 권한으로 실행해야 합니다.

Windows Command Prompt (CMD)

- 서비스 상태를 확인하려면

```
sc query "greengrass"
```

- 장치 부팅 시 Nucleus가 시작되도록 하기 위함입니다.

```
sc config "greengrass" start=auto
```

- 장치 부팅 시 핵이 시작되지 않도록 하기 위함입니다.

```
sc config "greengrass" start=disabled
```

- AWS IoT Greengrass Core 소프트웨어를 시작하려면

```
sc start "greengrass"
```

- AWS IoT Greengrass Core 소프트웨어를 중지하려면

```
sc stop "greengrass"
```

> **Note**
>
> Windows 장치에서 AWS IoT Greengrass Core 소프트웨어는 Greengrass 구성 요소 프로세스를 종료하는 동안 이 종료 신호를 무시합니다. 이 명령을 실행할 때 AWS IoT Greengrass Core 소프트웨어가 종료 신호를 무시하는 경우 몇 초 정도 기다린 후 다시 시도하십시오.

PowerShell

- 서비스 상태를 확인하려면

```
Get-Service -Name "greengrass"
```

- 장치 부팅 시 Nucleus가 시작되도록 하기 위함입니다.

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- 장치 부팅 시 핵이 시작되지 않도록 하기 위함입니다.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- AWS IoT Greengrass Core 소프트웨어를 시작하려면

```
Start-Service -Name "greengrass"
```

- AWS IoT Greengrass Core 소프트웨어를 중지하려면

```
Stop-Service -Name "greengrass"
```

> **Note**
>
> Windows 장치에서 AWS IoT Greengrass Core 소프트웨어는 Greengrass 구성 요소 프로세스를 종료하는 동안 이 종료 신호를 무시합니다. 이 명령을 실행할 때 AWS IoT Greengrass Core 소프트웨어가 종료 신호를 무시하는 경우 몇 초 정도 기다린 후 다시 시도하십시오.

JVM 옵션으로 메모리 할당을 제어하세요.

메모리가 제한된 AWS IoT Greengrass 기기에서 실행하는 경우 Java Virtual Machine (JVM) 옵션을 사용하여 최대 힙 크기, 가비지 컬렉션 모드 및 코어 소프트웨어에서 사용하는 메모리 양을 제어하는 컴파일러 옵션을 제어할 수 있습니다. AWS IoT Greengrass JVM의 힙 크기에 따라 가비지 컬렉션이 발생하기 전이나 애플리케이션의 메모리가 부족해지기 전에 애플리케이션이 사용할 수 있는 메모리 양이 결정됩니다. 최대 힙 크기는 로드가 많이 발생하는 작업 중 힙을 확장하는 경우 JVM에서 할당할 수 있는 최대 메모리 양을 지정합니다.

메모리 할당을 제어하려면 nucleus 구성 요소가 포함된 새 배포를 만들거나 기존 배포를 수정하고 nucleus 구성 요소 구성의 `jvmOptions` 구성 매개 변수에 JVM 옵션을 지정하십시오.

요구 사항에 따라 메모리 할당을 줄이거나 최소 메모리 할당으로 AWS IoT Greengrass Core 소프트웨어를 실행할 수 있습니다.

메모리 할당 감소

메모리 할당을 줄여 AWS IoT Greengrass Core 소프트웨어를 실행하려면 다음 예제 구성 병합 업데이트를 사용하여 nucleus 구성에서 JVM 옵션을 설정하는 것이 좋습니다.

```
{
  "jvmOptions": "-Xmx64m -XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

최소 메모리 할당

최소 메모리 할당으로 AWS IoT Greengrass Core 소프트웨어를 실행하려면 다음 예제 구성 병합 업데이트를 사용하여 nucleus 구성에서 JVM 옵션을 설정하는 것이 좋습니다.

```
{
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
}
```

이 예제 구성 병합 업데이트는 다음 JVM 옵션을 사용합니다.

`-Xmx`*NN*`m`

최대 JVM 힙 크기를 설정합니다.

메모리 할당을 줄이려면 시작 `-Xmx64m` 값으로 사용하여 힙 크기를 64MB로 제한하십시오. 최소 메모리 할당의 경우 시작 `-Xmx32m` 값으로 사용하여 힙 크기를 32MB로 제한하십시오.

실제 요구 사항에 따라 `-Xmx` 값을 늘리거나 줄일 수 있지만 최대 힙 크기를 16MB 미만으로 설정하지 않는 것이 좋습니다. 환경에 비해 최대 힙 크기가 너무 작으면 메모리 부족으로 인해 AWS IoT Greengrass Core 소프트웨어에서 예상치 못한 오류가 발생할 수 있습니다.

`-XX:+UseSerialGC`

JVM 힙 공간에 직렬 가비지 컬렉션을 사용하도록 지정합니다. 직렬 가비지 컬렉터는 속도가 느리지만 다른 JVM 가비지 컬렉션 구현보다 메모리를 덜 사용합니다.

`-XX:TieredStopAtLevel=1`

JVM에 Java just-in-time (JIT) 컴파일러를 한 번 사용하도록 지시합니다. JIT 컴파일된 코드는 기기 메모리의 공간을 사용하므로 JIT 컴파일러를 두 번 이상 사용하면 단일 컴파일보다 더 많은 메모리를 소비합니다.

`-Xint`

JVM에 (JIT) 컴파일러를 사용하지 않도록 지시합니다. just-in-time 대신 JVM은 해석된 전용 모드로 실행됩니다. 이 모드는 JIT 컴파일된 코드를 실행하는 것보다 느리지만 컴파일된 코드는 메모리 공간을 전혀 사용하지 않습니다.

구성 병합 업데이트를 만드는 방법에 대한 자세한 내용은 을 참조하십시오. 구성 요소 구성 업데이트

구성 요소를 실행하는 사용자를 구성하십시오.

AWS IoT Greengrass Core 소프트웨어는 소프트웨어를 실행하는 사용자와는 다른 시스템 사용자 및 그룹으로 구성 요소 프로세스를 실행할 수 있습니다. 이렇게 하면 AWS IoT Greengrass 코어 장치에서 실행되는 구성 요소에 해당 권한을 부여하지 않고도 루트 또는 관리자 권한으로 Core 소프트웨어를 실행할 수 있으므로 보안이 향상됩니다.

다음 표에는 지정한 사용자로 AWS IoT Greengrass Core 소프트웨어가 실행할 수 있는 구성 요소 유형이 나와 있습니다. 자세한 설명은 구성 요소 유형 섹션을 참조하세요.

구성 요소 유형	구성 요소 사용자 구성
뉴클리어스	 아니요

구성 요소 유형	구성 요소 사용자 구성
플러그인	 아니요
제네릭	 네
Lambda (비컨테이너식)	 예
람다 (컨테이너식)	 네

배포 구성에서 구성 요소 사용자를 지정하려면 먼저 구성 요소 사용자를 만들어야 합니다. Windows 기반 장치에서는 계정의 자격 증명 관리자 인스턴스에 사용자의 사용자 이름과 암호도 저장해야 합니다. LocalSystem 자세한 설명은 Windows 장치에서 구성 요소 사용자 설정 섹션을 참조하세요.

Linux 기반 장치에서 구성 요소 사용자를 구성할 때 선택적으로 그룹을 지정할 수도 있습니다. 다음과 같은 형식으로 사용자와 그룹을 콜론 (`:`) 으로 구분하여 지정합니다. *`user`*`:`*`group`* 그룹을 지정하지 않는 경우 AWS IoT Greengrass Core 소프트웨어는 기본적으로 사용자의 기본 그룹을 사용합니다. 이름 또는 ID를 사용하여 사용자와 그룹을 식별할 수 있습니다.

Linux 기반 장치에서는 존재하지 않는 시스템 사용자 (알 수 없는 사용자라고도 함) 로 구성 요소를 실행하여 보안을 강화할 수도 있습니다. Linux 프로세스는 동일한 사용자가 실행하는 다른 모든 프로세스에 신호를 보낼 수 있습니다. 알 수 없는 사용자는 다른 프로세스를 실행하지 않으므로 구성 요소를 알 수 없는 사용자로 실행하여 구성 요소가 코어 장치의 다른 구성 요소에 신호를 보내는 것을 방지할 수 있습니다. 알 수 없는 사용자로 구성 요소를 실행하려면 코어 장치에 없는 사용자 ID를 지정하십시오. 존재하지 않는 그룹 ID를 알 수 없는 그룹으로 실행하도록 지정할 수도 있습니다.

각 구성 요소 및 각 코어 장치에 대해 사용자를 구성할 수 있습니다.

- 구성 요소 구성

해당 구성 요소별 사용자와 함께 실행되도록 각 구성 요소를 구성할 수 있습니다. 배포를 만들 때 해당 구성 요소 `runWith` 구성의 각 구성 요소에 대해 사용자를 지정할 수 있습니다. AWS IoT Greengrass Core 소프트웨어는 구성 요소를 구성한 경우 지정된 사용자로 구성 요소를 실행합니다. 그렇지 않으면 코어 장치용으로 구성한 기본 사용자로 구성 요소를 실행하는 것이 기본값입니다. 배포 구성에서 구성 요소 사용자를 지정하는 방법에 대한 자세한 내용은 의 `runWith`구성 매개 변수를 참조하십시오. 배포 만들기

- 코어 디바이스의 기본 사용자 구성

  AWS IoT Greengrass Core 소프트웨어가 구성 요소를 실행하는 데 사용하는 기본 사용자를 구성할 수 있습니다. AWS IoT Greengrass Core 소프트웨어는 구성 요소를 실행할 때 해당 구성 요소에 사용자를 지정했는지 확인하고 이를 사용하여 구성 요소를 실행합니다. 구성 요소가 사용자를 지정하지 않는 경우 AWS IoT Greengrass Core 소프트웨어는 코어 장치용으로 구성한 기본 사용자로 구성 요소를 실행합니다. 자세한 설명은 기본 구성 요소 사용자 구성 섹션을 참조하세요.

> **Note**
>
> Windows 기반 장치에서는 구성 요소를 실행할 기본 사용자를 최소 한 명 이상 지정해야 합니다.
>
> Linux 기반 장치에서 구성 요소를 실행하도록 사용자를 구성하지 않는 경우 다음 고려 사항이 적용됩니다.
>
> - AWS IoT Greengrass Core 소프트웨어를 루트로 실행하면 소프트웨어가 구성 요소를 실행하지 않습니다. 루트로 실행하는 경우 구성 요소를 실행할 기본 사용자를 지정해야 합니다.
> - 루트 사용자가 아닌 사용자로 AWS IoT Greengrass Core 소프트웨어를 실행하는 경우 소프트웨어는 해당 사용자로 구성 요소를 실행합니다.

주제

- Windows 장치에서 구성 요소 사용자 설정
- 기본 구성 요소 사용자 구성

Windows 장치에서 구성 요소 사용자 설정

Windows 기반 장치에서 구성 요소 사용자를 설정하려면

1. 장치의 LocalSystem 계정에서 구성 요소 사용자를 생성합니다.

   ```
   net user /add component-user password
   ```

2. Microsoft의 PsExec 유틸리티를 사용하여 구성 요소 사용자의 사용자 이름과 암호를 LocalSystem 계정의 자격 증명 관리자 인스턴스에 저장합니다.

   ```
   psexec -s cmd /c cmdkey /generic:component-user /user:component-user /pass:password
   ```

   > **Note**
   >
   > Windows 기반 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 구성 요소 사용자 정보를 계정에 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

기본 구성 요소 사용자 구성

배포를 사용하여 코어 디바이스의 기본 사용자를 구성할 수 있습니다. 이 배포에서는 nucleus 구성 요소 구성을 업데이트합니다.

> **Note**
>
> `--component-default-user`옵션을 사용하여 AWS IoT Greengrass Core 소프트웨어를 설치할 때 기본 사용자를 설정할 수도 있습니다. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어 설치 섹션을 참조하세요.

구성 `aws.greengrass.Nucleus` 요소에 대한 다음 구성 업데이트를 지정하는 배포를 생성합니다.

Linux

```
{
  "runWithDefault": {
```

```
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

> **Note**
>
> 지정하는 사용자가 존재해야 하며 이 사용자의 사용자 이름과 암호는 Windows 디바이스 LocalSystem 계정의 자격 증명 관리자 인스턴스에 저장되어 있어야 합니다. 자세한 설명은 Windows 장치에서 구성 요소 사용자 설정 섹션을 참조하세요.

다음 예에서는 기본 사용자 및 ggc_group 기본 그룹으로 구성된 ggc_user Linux 기반 장치에 대한 배포를 정의합니다. merge구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.2",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}
```

구성 요소에 대한 시스템 리소스 제한을 구성합니다.

> **Note**
>
> 이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다.

다음 표에는 시스템 리소스 제한을 지원하는 구성 요소 유형이 나와 있습니다. 자세한 설명은 구성 요소 유형 섹션을 참조하세요.

구성 요소 유형	시스템 리소스 제한을 구성합니다.
뉴클리어스	 아니요
플러그인	 아니요
제네릭	 네
Lambda (비컨테이너식)	 예
람다 (컨테이너식)	 아니요

Important

Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어를 실행할 때는 시스템 리소스 제한이 지원되지 않습니다.

각 구성 요소 및 각 코어 장치에 대한 시스템 리소스 제한을 구성할 수 있습니다.

- 구성 요소 구성

  해당 구성 요소에만 적용되는 시스템 리소스 제한으로 각 구성 요소를 구성할 수 있습니다. 배포를 만들 때 배포의 각 구성 요소에 대한 시스템 리소스 제한을 지정할 수 있습니다. 구성 요소가 시스템 리소스 제한을 지원하는 경우 AWS IoT Greengrass Core 소프트웨어는 구성 요소의 프로세스에 제한을 적용합니다. 구성 요소에 대한 시스템 리소스 제한을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 사용자가 코어 장치에 구성한 모든 기본값을 사용합니다. 자세한 설명은 배포 만들기 섹션을 참조하세요.
- 코어 디바이스의 기본값을 구성합니다.

  AWS IoT Greengrass Core 소프트웨어가 이러한 제한을 지원하는 구성 요소에 적용하는 기본 시스템 리소스 제한을 구성할 수 있습니다. AWS IoT Greengrass Core 소프트웨어는 구성 요소를 실행할 때 해당 구성 요소에 지정된 시스템 리소스 제한을 적용합니다. 해당 구성 요소가 시스템 리소스 제한을 지정하지 않는 경우 AWS IoT Greengrass Core 소프트웨어는 사용자가 코어 장치에 구성한 기본 시스템 리소스 제한을 적용합니다. 기본 시스템 리소스 제한을 지정하지 않으면 AWS IoT Greengrass Core 소프트웨어는 기본적으로 시스템 리소스 제한을 적용하지 않습니다. 자세한 설명은 기본 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요.

기본 시스템 리소스 제한을 구성합니다.

Greengrass nucleus 구성 요소를 배포하여 코어 디바이스의 기본 시스템 리소스 제한을 구성할 수 있습니다. 기본 시스템 리소스 제한을 구성하려면 구성 요소에 대한 다음 구성 업데이트를 지정하는 배포를 생성하십시오. `aws.greengrass.Nucleus`

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

다음 예시에서는 CPU 시간 제한을 CPU 코어 4개가 있는 기기의 50% 사용량에 해당하는 수준으로 구성하는 배포를 정의합니다. 2 또한 이 예시에서는 메모리 사용량을 100MB로 구성합니다.

```
{
```

```
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.2",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":\"cpus\":2,\"memory
\":102400}}}"
      }
    }
  }
}
```

포트 443에서 또는 네트워크 프록시를 통해 연결

AWS IoT Greengrass 코어 디바이스는 TLS 클라이언트 인증과 함께 MQTT 메시징 프로토콜을 AWS IoT Core 사용하여 통신합니다. 일반적으로 MQTT over TLS는 포트 8883을 사용합니다. 그러나 보안 조치로서 제한적 환경 하에서 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. 기타 제한적인 환경에서는 인터넷에 연결하기 전에 모든 트래픽이 프록시를 거쳐야 할 수 있습니다.

> **Note**
>
> Greengrass 핵 구성 요소 v2.0.3 및 이전 버전을 실행하는 Greengrass 코어 디바이스는 포트 8443을 사용하여 데이터 플레인 엔드포인트에 연결합니다. AWS IoT Greengrass 이러한 디바이스는 포트 8443에서 이 엔드포인트에 연결할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

이러한 시나리오에서 통신을 활성화하기 위해 다음과 같은 구성 옵션을 AWS IoT Greengrass 제공합니다.

- 포트 443을 통한 MQTT 통신. 네트워크에서 포트 443에 대한 연결을 허용하는 경우 기본 포트 8883 대신 MQTT 트래픽에 포트 443을 사용하도록 Greengrass 코어 디바이스를 구성할 수 있습니다. 이는 포트 443으로 직접 연결되거나 네트워크 프록시 서버를 통해 연결되는 것일 수 있습니다. 인증서 기반 클라이언트 인증을 사용하는 기본 구성과 달리 포트 443의 MQTT는 인증에 장치 서비스 역할을 사용합니다.

  자세한 설명은 포트 443을 통해 MQTT를 구성합니다. 섹션을 참조하세요.

- 포트 443을 통한 HTTPS 통신. AWS IoT Greengrass Core 소프트웨어는 기본적으로 포트 8443을 통해 HTTPS 트래픽을 전송하지만 포트 443을 사용하도록 구성할 수 있습니다. AWS IoT Greengrass 애플리케이션 계층 프로토콜 네트워크 (ALPN) TLS 확장을 사용하여 이 연결을 활성화합니다. 기본 구성과 마찬가지로 포트 443의 HTTPS는 인증서 기반 클라이언트 인증을 사용합니다.

  > **Important**
  >
  > ALPN을 사용하고 포트 443을 통해 HTTPS 통신을 활성화하려면 코어 디바이스에서 Java 8 업데이트 252 이상을 실행해야 합니다. Java 버전 9 이상의 모든 업데이트는 ALPN도 지원합니다.

  자세한 설명은 포트 443을 통해 HTTPS를 구성합니다. 섹션을 참조하세요.

- 네트워크 프록시를 통한 연결. Greengrass 코어 장치에 연결하기 위한 중개자 역할을 하도록 네트워크 프록시 서버를 구성할 수 있습니다. AWS IoT Greengrass HTTP 및 HTTPS 프록시에 대한 기본 인증을 지원합니다.

  HTTPS 프록시를 사용하려면 그린그래스 코어 디바이스에서 그린그래스 핵 v2.5.0 이상을 실행해야 합니다.

  AWS IoT Greengrass Core 소프트웨어는,, 및 환경 변수를 통해 프록시 구성을 구성 요소에 전달합니다. `ALL_PROXY` `HTTP_PROXY` `HTTPS_PROXY` `NO_PROXY` 구성 요소가 프록시를 통해 연결하려면 이러한 설정을 사용해야 합니다. 구성 요소는 일반적으로 이러한 환경 변수를 기본적으로 사용하여 연결하는 공통 라이브러리 (예: boto3, cURL 및 python `requests` 패키지) 를 사용합니다. 구성 요소가 이러한 환경 변수도 지정하는 경우 해당 변수를 재정의하지 AWS IoT Greengrass 않습니다.

  자세한 설명은 네트워크 프록시를 구성하십시오. 섹션을 참조하세요.

포트 443을 통해 MQTT를 구성합니다.

기존 코어 장치의 포트 443을 통해 또는 새 코어 장치에 Core 소프트웨어를 설치할 때 MQTT를 구성할 수 있습니다. AWS IoT Greengrass

주제

- 기존 코어 디바이스의 포트 443을 통해 MQTT를 구성합니다.
- 설치 중에 포트 443을 통해 MQTT를 구성합니다.

기존 코어 디바이스의 포트 443을 통해 MQTT를 구성합니다.

배포를 사용하여 단일 코어 장치 또는 코어 장치 그룹의 포트 443을 통해 MQTT를 구성할 수 있습니다. 이 배포에서는 nucleus 구성 요소 구성을 업데이트합니다. 구성을 업데이트하면 Nucleus가 다시 시작됩니다. `mqtt`

포트 443을 통해 MQTT를 구성하려면 구성 요소에 대한 다음 구성 업데이트를 지정하는 배포를 생성하십시오. `aws.greengrass.Nucleus`

```
{
  "mqtt": {
    "port": 443
  }
}
```

다음 예시에서는 포트 443을 통해 MQTT를 구성하는 배포를 정의합니다. `merge`구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.2",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

설치 중에 포트 443을 통해 MQTT를 구성합니다.

AWS IoT Greengrass 코어 디바이스에 Core 소프트웨어를 설치할 때 포트 443을 통해 MQTT를 구성할 수 있습니다. `--init-config`설치 프로그램 인수를 사용하여 포트 443을 통해 MQTT를 구성하십시오. 수동 프로비저닝, 플릿 프로비저닝 또는 사용자 지정 프로비저닝으로 설치할 때 이 인수를 지정할 수 있습니다.

포트 443을 통해 HTTPS를 구성합니다.

이 기능을 사용하려면 그린그래스 핵 v2.0.4 이상이 필요합니다.

기존 코어 디바이스에서 포트 443을 통해 HTTPS를 구성하거나 새 코어 디바이스에 AWS IoT Greengrass Core 소프트웨어를 설치할 때 구성할 수 있습니다.

주제

- 기존 코어 디바이스의 포트 443을 통해 HTTPS를 구성합니다.
- 설치 중에 포트 443을 통해 HTTPS를 구성합니다.

기존 코어 디바이스의 포트 443을 통해 HTTPS를 구성합니다.

배포를 사용하여 단일 코어 장치 또는 핵심 장치 그룹에서 포트 443을 통해 HTTPS를 구성할 수 있습니다. 이 배포에서는 nucleus 구성 요소 구성을 업데이트합니다.

포트 443을 통해 HTTPS를 구성하려면 구성 요소에 대한 다음 구성 업데이트를 지정하는 배포를 생성하십시오. `aws.greengrass.Nucleus`

```
{
  "greengrassDataPlanePort": 443
}
```

다음 예시에서는 포트 443을 통해 HTTPS를 구성하는 배포를 정의합니다. `merge`구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.2",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

설치 중에 포트 443을 통해 HTTPS를 구성합니다.

AWS IoT Greengrass 코어 디바이스에 Core 소프트웨어를 설치할 때 포트 443을 통해 HTTPS를 구성할 수 있습니다. `--init-config`설치 프로그램 인수를 사용하여 포트 443을 통해 HTTPS를 구성하십시오. 수동 프로비저닝, 플릿 프로비저닝 또는 사용자 지정 프로비저닝으로 설치할 때 이 인수를 지정할 수 있습니다.

네트워크 프록시를 구성하십시오.

이 섹션의 절차에 따라 Greengrass 코어 디바이스가 HTTP 또는 HTTPS 네트워크 프록시를 통해 인터넷에 연결되도록 구성합니다. 코어 디바이스가 사용하는 엔드포인트 및 포트에 대한 자세한 내용은 을 참조하십시오. 프록시 또는 방화벽을 통한 장치 트래픽 허용

> **Important**
>
> 코어 디바이스에서 v2.4.0 이전의 Greengrass nucleus 버전을 실행하는 경우, 디바이스의 역할은 네트워크 프록시를 사용할 수 있는 다음 권한을 허용해야 합니다.
>
> - `iot:Connect`
> - `iot:Publish`
> - `iot:Receive`
> - `iot:Subscribe`
>
> 이는 디바이스가 토큰 교환 서비스의 AWS 자격 증명을 사용하여 MQTT 연결을 인증하기 때문에 필요합니다. AWS IoT 장치는 MQTT를 사용하여 에서 배포를 수신하고 설치하므로 역할에 이러한 권한을 정의하지 않으면 장치가 작동하지 않습니다. AWS 클라우드 기기는 일반적으로 X.509 인증서를 사용하여 MQTT 연결을 인증하지만, 프록시를 사용할 때는 이를 인증할 수 없습니다.
> 장치 역할을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

주제

- 기존 코어 디바이스에 네트워크 프록시를 구성합니다.
- 설치 중에 네트워크 프록시를 구성하십시오.
- 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요.
- 네트워크프록시 오브젝트

기존 코어 디바이스에 네트워크 프록시를 구성합니다.

배포를 사용하여 단일 코어 장치 또는 코어 장치 그룹에 네트워크 프록시를 구성할 수 있습니다. 이 배포에서는 nucleus 구성 요소 구성을 업데이트합니다. 구성을 업데이트하면 Nucleus가 다시 시작됩니다. `networkProxy`

네트워크 프록시를 구성하려면 다음 구성 업데이트를 병합하는 aws.greengrass.Nucleus 구성 요소에 대한 배포를 생성하십시오. 이 구성 업데이트에는 NetworkProxy 객체가 포함되어 있습니다.

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

다음 예제는 네트워크 프록시를 구성하는 배포를 정의합니다. merge구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.2",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
}
```

설치 중에 네트워크 프록시를 구성하십시오.

AWS IoT Greengrass 코어 디바이스에 Core 소프트웨어를 설치할 때 네트워크 프록시를 구성할 수 있습니다. --init-configinstaller 인수를 사용하여 네트워크 프록시를 구성합니다. 수동 프로비저닝, 플릿 프로비저닝 또는 사용자 지정 프로비저닝으로 설치할 때 이 인수를 지정할 수 있습니다.

코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요.

HTTPS 프록시를 사용하도록 핵심 장치를 구성하는 경우 핵심 장치에 프록시 서버 인증서 체인을 추가하여 핵심 장치가 HTTPS 프록시를 신뢰할 수 있도록 해야 합니다. 그렇지 않으면 코어 디바이스가 프록시를 통해 트래픽을 라우팅하려고 할 때 오류가 발생할 수 있습니다. 프록시 서버 CA 인증서를 코어 디바이스의 Amazon 루트 CA 인증서 파일에 추가합니다.

코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하려면

1. 코어 디바이스에서 Amazon 루트 CA 인증서 파일을 찾으십시오.

   - 자동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치한 경우 Amazon 루트 CA 인증서 파일은 에 있습니다. */greengrass/v2*/rootCA.pem
   - 수동 또는 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치한 경우 Amazon 루트 CA 인증서 파일이 에 있을 수 있습니다. */greengrass/v2*/AmazonRootCA1.pem

   Amazon 루트 CA 인증서가 이러한 위치에 없는 경우 `system.rootCaPath` 속성을 확인하여 위치를 찾으십시오. */greengrass/v2*/config/effectiveConfig.yaml
2. 프록시 서버 CA 인증서 파일의 내용을 Amazon 루트 CA 인증서 파일에 추가합니다.

   다음 예는 Amazon 루트 CA 인증서 파일에 추가된 프록시 서버 CA 인증서를 보여줍니다.

```
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

네트워크프록시 오브젝트

networkProxy 객체를 사용하여 네트워크 프록시 관련 정보를 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`noProxyAddresses`

(선택 사항) 프록시에서 제외되는 IP 주소 또는 호스트 이름을 쉼표로 구분한 목록입니다.

`proxy`

연결할 프록시입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`url`

형식의 프록시 서버 URL`scheme://userinfo@host:port`.

- `scheme`— 스킴은 또는이어야 `http` 합니다`https`.

  > **Important**
  >
  > HTTPS 프록시를 사용하려면 그린그래스 코어 디바이스에서 그린그래스 핵 v2.5.0 이상을 실행해야 합니다.
  > HTTPS 프록시를 구성하는 경우 코어 디바이스의 Amazon 루트 CA 인증서에 프록시 서버 CA 인증서를 추가해야 합니다. 자세한 설명은 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.

- `userinfo`— (선택 사항) 사용자 이름 및 암호 정보. 에서 이 정보를 지정하는 경우 Greengrass 코어 디바이스는 및 필드를 무시합니다. `url` `username` `password`
- `host`— 프록시 서버의 호스트 이름 또는 IP 주소.
- `port`— (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 기기는 다음과 같은 기본값을 사용합니다.
  - `http`— 80
  - `https`— 443

`username`

(선택 사항) 프록시 서버를 인증하는 사용자 이름.

`password`

(선택 사항) 프록시 서버를 인증하는 암호.

사설 CA에서 서명한 장치 인증서를 사용하십시오.

사용자 지정 사설 인증 기관 (CA) 을 사용하는 경우 Greengrass **greengrassDataPlaneEndpoint** nucleus를 '로 설정해야 합니다. **iotdata** 설치 또는 설치 중에 installer 인수를 **--init-config** 사용하여 이 옵션을 설정할 수 있습니다.

기기가 연결되는 Greengrass 데이터 플레인 엔드포인트를 사용자 지정할 수 있습니다. 이 구성 옵션을 **iotdata** 설정하여 Greengrass 데이터 플레인 엔드포인트를 IoT 데이터 엔드포인트와 동일한 엔드포인트로 설정할 수 있습니다. 이 엔드포인트는 에서 지정할 수 있습니다. **iotDataEndpoint**

MQTT 타임아웃 및 캐시 설정을 구성합니다.

AWS IoT Greengrass 환경에서 구성 요소는 MQTT를 사용하여 통신할 수 있습니다. AWS IoT Core AWS IoT Greengrass Core 소프트웨어는 구성 요소의 MQTT 메시지를 관리합니다. 코어 디바이스와의 연결이 끊어지면 소프트웨어가 MQTT 메시지를 캐시하여 나중에 연결이 복원되면 다시 시도합니다. AWS 클라우드 메시지 제한 시간 및 캐시 크기와 같은 설정을 구성할 수 있습니다. 자세한 내용은 Greengrass 핵 `mqtt.spooler` 구성 요소의 `mqtt` 및 구성 매개 변수를 참조하십시오.

AWS IoT Core MQTT 메시지 브로커에 서비스 할당량을 부과합니다. 이러한 할당량은 코어 기기와 간에 전송하는 메시지에 적용될 수 있습니다. AWS IoT Core 자세한 내용은 의 AWS IoT Core 메시지 브로커 서비스 할당량을 참조하십시오. AWS 일반 참조

AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트

AWS IoT GreengrassCore 소프트웨어는 Greengrass nucleus 구성 요소와 소프트웨어의 OTA over-the-air (업데이트를 수행하기 위해 장치에 배포할 수 있는) 기타 선택적 구성 요소로 구성됩니다. 이 기능은 AWS IoT Greengrass Core 소프트웨어에 내장되어 있습니다.

OTA 업데이트를 통해 다음을 보다 효율적으로 수행할 수 있습니다.

- 보안 취약성을 수정합니다.
- 소프트웨어 안정성 문제를 해결합니다.
- 새 기능 또는 향상된 기능을 배포합니다.

주제

- 요구 사항
- 코어 기기 고려사항

- Greengrass 핵 업데이트 동작
- OTA 업데이트 수행

요구 사항

AWS IoT GreengrassCore 소프트웨어의 OTA 업데이트를 배포하려면 다음 요구 사항이 적용됩니다.

- 배포를 받으려면 Greengrass 코어 디바이스가 AWS 클라우드 에 연결되어 있어야 합니다.
- Greengrass 코어 디바이스를 올바르게 구성하고 및 를 사용한 인증을 위한 인증서 및 키를 제공해야 합니다. AWS IoT Core AWS IoT Greengrass
- AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 설정하고 실행해야 합니다. JAR 파일에서 nucleus를 실행하면 OTA 업데이트가 작동하지 않습니다. `Greengrass.jar` 자세한 설명은 Greengrass 핵을 시스템 서비스로 구성 섹션을 참조하세요.

코어 기기 고려사항

OTA 업데이트를 수행하기 전에 업데이트하는 핵심 기기와 연결된 클라이언트 기기에 미치는 영향을 파악하세요.

- Greengrass 핵이 꺼집니다.
- 코어 디바이스에서 실행 중인 모든 구성 요소도 종료되었습니다. 이러한 구성 요소가 로컬 리소스에 기록하는 경우 제대로 종료하지 않으면 해당 리소스가 잘못된 상태로 남을 수 있습니다. 구성 요소는 프로세스 간 통신을 사용하여 사용하는 리소스를 정리할 때까지 업데이트를 연기하도록 nucleus 구성 요소에 지시할 수 있습니다.
- Nucleus 구성 요소가 종료되는 동안 코어 디바이스는 및 로컬 디바이스와의 연결이 끊어집니다. AWS 클라우드 코어 디바이스는 종료된 동안 클라이언트 디바이스의 메시지를 라우팅하지 않습니다.
- 구성 요소로 실행되는 수명이 긴 Lambda 함수는 동적 상태 정보를 잃고 보류 중인 모든 작업을 중단합니다.

Greengrass 핵 업데이트 동작

구성 요소를 배포할 때 해당 구성 요소의 모든 종속 항목에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습

니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.

Greengrass nucleus 구성 요소의 버전이 변경되면 AWS IoT Greengrass Core 소프트웨어 (핵심 및 장치의 다른 모든 구성 요소가 포함됨) 가 다시 시작되어 변경 사항을 적용합니다. nucleus 구성 요소가 업데이트되면 코어 디바이스에 미치는 영향이 크기 때문에 새 Nucleus 패치 버전을 디바이스에 배포하는 시기를 제어하는 것이 좋습니다. 이렇게 하려면 배포에 Greengrass nucleus 구성 요소를 직접 포함해야 합니다. 구성 요소를 직접 포함한다는 것은 해당 구성 요소의 특정 버전을 배포 구성에 포함한다는 의미이며 구성 요소 종속성에 의존하지 않고 해당 구성 요소를 장치에 배포할 수 있습니다. 구성 요소 레시피의 종속성 정의에 대한 자세한 내용은 을 참조하십시오. 레시피 형식

다음 표를 검토하여 작업 및 배포 구성을 기반으로 하는 Greengrass nucleus 구성 요소의 업데이트 동작을 이해하십시오.

작업	배포 구성	Nucleus 업데이트 동작
배포를 수정하지 않고 기존 배포의 대상이 되는 사물 그룹에 새 디바이스를 추가합니다.	배포에는 Greengrass 핵이 직접 포함되지 않습니다. 배포에는 제공된 구성 요소가 하나 이상 직접 포함되거나 AWS 제공된 구성 요소 또는 Greengrass 핵에 의존하는 사용자 지정 구성 요소가 포함됩니다. AWS	새 디바이스에는 모든 구성 요소 종속성 요구 사항을 충족하는 Nucleus의 최신 패치 버전을 설치합니다. 기존 디바이스에서는 설치된 Nucleus 버전을 업데이트하지 않습니다.
배포를 수정하지 않고 기존 배포의 대상이 되는 사물 그룹에 새 장치를 추가합니다.	배포에는 특정 버전의 Greengrass 핵이 직접 포함됩니다.	새 디바이스에 지정된 Nucleus 버전을 설치합니다. 기존 디바이스에서는 설치된 Nucleus 버전을 업데이트하지 않습니다.
새 배포를 만들거나 기존 배포를 수정하십시오.	배포에는 Greengrass 핵이 직접 포함되지 않습니다. 배포에는 제공된 구성 요소가 하나 이상 직접 포함되거나	대상 사물 그룹에 추가하는 새 장치를 포함하여 모든 구성 요소 종속성 요구 사항을 충족하는 Nucleus의 최신 패치 버전

작업	배포 구성	Nucleus 업데이트 동작
	AWS 제공된 구성 요소 또는 Greengrass 핵에 의존하는 사용자 지정 구성 요소가 포함됩니다. AWS	을 모든 대상 장치에 설치합니다.
새 배포를 생성하거나 기존 배포를 수정하십시오.	배포에는 특정 버전의 Greengrass 핵이 직접 포함됩니다.	대상 사물 그룹에 추가하는 새 장치를 포함하여 모든 대상 장치에 지정된 Nucleus 버전을 설치합니다.

OTA 업데이트 수행

OTA 업데이트를 수행하려면 nucleus 구성 요소와 설치할 버전이 포함된 배포를 생성하십시오.

AWS IoT GreengrassCore 소프트웨어 제거

Greengrass AWS IoT Greengrass 코어 장치로 사용하고 싶지 않은 장치에서 Core 소프트웨어를 제거하여 제거할 수 있습니다. 다음 단계를 사용하여 실패한 설치를 정리할 수도 있습니다.

AWS IoT GreengrassCore 소프트웨어를 제거하려면

1. 소프트웨어를 시스템 서비스로 실행하는 경우 서비스를 중지, 비활성화 및 제거해야 합니다. 운영 체제에 맞게 다음 명령을 실행합니다.

   Linux

   1. 서비스를 중단합니다.

      ```
      sudo systemctl stop greengrass.service
      ```

   2. 서비스를 비활성화합니다.

      ```
      sudo systemctl disable greengrass.service
      ```

   3. 서비스를 제거합니다.

```
sudo rm /etc/systemd/system/greengrass.service
```

4. 서비스가 삭제되었는지 확인합니다.

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

> **Note**
> 이 명령을 실행하려면 관리자 권한으로 명령 프롬프트를 실행해야 합니다.

1. 서비스를 중단합니다.

```
sc stop "greengrass"
```

2. 서비스를 비활성화합니다.

```
sc config "greengrass" start=disabled
```

3. 서비스를 제거합니다.

```
sc delete "greengrass"
```

4. 디바이스를 다시 시작합니다.

Windows (PowerShell)

> **Note**
> 이 명령을 실행하려면 관리자 PowerShell 권한으로 실행해야 합니다.

1. 서비스를 중단합니다.

```
Stop-Service -Name "greengrass"
```

2. 서비스를 비활성화합니다.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. 서비스를 제거합니다.

   - PowerShell 6.0 이상 버전:

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

   - PowerShell 6.0 이전 버전의 경우:

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item
 -Force -Verbose
```

4. 디바이스를 다시 시작합니다.

2. 디바이스에서 루트 폴더를 제거합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 루트 폴더 경로로 바꿉니다.

   Linux

```
sudo rm -rf /greengrass/v2
```

   Windows (Command Prompt)

```
rmdir /s /q C:\greengrass\v2
```

   Windows (PowerShell)

```
cmd.exe /c "rmdir /s /q C:\greengrass\v2"
```

3. AWS IoT Greengrass서비스에서 코어 디바이스를 삭제합니다. 이 단계는 코어 디바이스의 상태 정보를 에서 제거합니다AWS 클라우드. 이름이 같은 AWS IoT Greengrass 코어 장치에 Core 소프트웨어를 다시 설치하려면 이 단계를 완료해야 합니다.

   - AWS IoT Greengrass콘솔에서 코어 디바이스를 삭제하려면 다음과 같이 하십시오.

     a. AWS IoT Greengrass 콘솔로 이동합니다.

     b. 코어 디바이스를 선택합니다.

     c. 삭제할 코어 디바이스를 선택합니다.

d. Delete를 선택합니다.

e. 확인 모달에서 삭제를 선택합니다.

- 를 사용하여 코어 디바이스를 AWS Command Line Interface 삭제하려면 DeleteCoreDevice작업을 사용하십시오. 다음 명령을 실행하고 코어 디바이스의 이름으로 *MyGreengrassCore*바꿉니다.

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 자습서

다음 자습서를 완료하여 기능에 대해 AWS IoT Greengrass V2 알아볼 수 있습니다.

주제

튜토리얼: 구성 요소 업데이트를 연기하는 Greengrass 구성 요소 개발

이 자습서를 완료하여 over-the-air 배포 업데이트를 연기하는 구성 요소를 개발할 수 있습니다. 기기에 업데이트를 배포할 때 다음과 같은 조건에 따라 업데이트를 연기하고 싶을 수 있습니다.

- 디바이스의 배터리 잔량이 부족합니다.
- 기기에서 중단할 수 없는 프로세스나 작업이 실행 중입니다.
- 장치의 인터넷 연결이 제한적이거나 비용이 많이 듭니다.

> **Note**
>
> 구성 요소는 AWS IoT Greengrass 코어 장치에서 실행되는 소프트웨어 모듈입니다. 구성 요소를 사용하면 복잡한 애플리케이션을 하나의 Greengrass 코어 장치에서 다른 Greengrass 코어 장치로 재사용할 수 있는 개별 구성 요소로 만들고 관리할 수 있습니다.

이 자습서에서는 다음 작업을 수행합니다.

1. 개발 컴퓨터에 그린그래스 개발 키트 CLI (GDK CLI) 를 설치합니다. GDK CLI는 사용자 지정 Greengrass 구성 요소를 개발하는 데 도움이 되는 기능을 제공합니다.
2. 코어 디바이스의 배터리 잔량이 임계값 미만일 때 구성 요소 업데이트를 연기하는 Hello World 구성 요소를 개발하십시오. 이 구성 요소는 SubscribeToComponentUpdatesIPC 작업을 사용하여 업데이

트 알림을 구독합니다. 알림을 받으면 배터리 잔량이 사용자 지정 가능한 임계값보다 낮은지 확인합니다. 배터리 잔량이 임계값 미만인 경우 DeferComponentUpdateIPC 작업을 사용하여 30초 동안 업데이트를 연기합니다. GDK CLI를 사용하여 개발 컴퓨터에서 이 구성 요소를 개발합니다.

> **Note**
>
> 이 구성 요소는 코어 기기에서 생성한 파일에서 배터리 잔량을 읽어 실제 배터리를 모방하므로 배터리 없이 코어 기기에서 이 자습서를 완료할 수 있습니다.

3. 해당 구성 요소를 AWS IoT Greengrass 서비스에 게시하십시오.
4. 해당 구성 요소를 Greengrass 코어 장치에 배포하여 테스트합니다. AWS 클라우드 그런 다음 코어 디바이스의 가상 배터리 잔량을 수정하고 추가 배포를 생성하여 배터리 잔량이 부족할 때 코어 디바이스가 업데이트를 연기하는 방식을 확인합니다.

이 자습서에는 20~30분이 소요될 것으로 예상됩니다.

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- AWS 계정. 계정이 없는 경우 AWS 계정 설정 섹션을 참조하십시오.
- 관리자 권한이 있는 AWS Identity and Access Management (IAM) 사용자
- 인터넷에 연결된 Greengrass 코어 장치. 코어 디바이스 설정 방법에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 디바이스 설정.
  - Python 3.6 이상이 모든 사용자를 위해 코어 기기에 설치되고 PATH 환경 변수에 추가되었습니다. 윈도우에서는 모든 사용자를 위한 윈도우용 Python 런처도 설치되어 있어야 합니다.

> **Important**
>
> Windows에서 Python은 기본적으로 모든 사용자에게 설치되지 않습니다. Python을 설치할 때 AWS IoT Greengrass Core 소프트웨어가 Python 스크립트를 실행하도록 구성하도록 설치를 사용자 정의해야 합니다. 예를 들어, 그래픽 Python 설치 프로그램을 사용하는 경우 다음을 수행하십시오.
>
> 1. 모든 사용자를 위한 런처 설치를 선택합니다 (권장).
> 2. Customize installation를 선택합니다.
> 3. Next를 선택합니다.

4. Install for all users을(를) 선택합니다.
5. Add Python to environment variables을(를) 선택합니다.
6. 설치를 선택합니다.

자세한 내용은 Python 3 설명서의 윈도우에서 Python 사용을 참조하십시오.

- 인터넷에 연결된 윈도우, macOS 또는 유닉스 계열 개발 컴퓨터.
  - Python 3.6 이상이 개발 컴퓨터에 설치되어 있어야 합니다.
  - Git이 개발 컴퓨터에 설치되었습니다.
  - AWS Command Line Interface(AWS CLI) 개발 컴퓨터에 자격 증명을 사용하여 설치 및 구성되었습니다. 자세한 내용은 AWS Command Line Interface사용 설명서의 설치, 업데이트, 제거 AWS CLI 및 구성을 참조하십시오. AWS CLI

> **Note**
>
> 라즈베리 파이 또는 다른 32비트 ARM 디바이스를 사용하는 경우 V1을 설치하세요. AWS CLI AWS CLI V2는 32비트 ARM 디바이스에서 사용할 수 없습니다. 자세한 내용은 버전 1 설치, 업데이트 및 제거를 AWS CLI 참조하십시오.

1단계: 그린그래스 개발 키트 CLI 설치

그린그래스 개발 키트 CLI (GDK CLI) 는 사용자 지정 그린그래스 구성 요소를 개발하는 데 도움이 되는 기능을 제공합니다. GDK CLI를 사용하여 사용자 지정 구성 요소를 만들고 빌드하고 게시할 수 있습니다.

개발 컴퓨터에 GDK CLI를 설치하지 않은 경우 다음 단계를 완료하여 설치하십시오.

최신 버전의 GDK CLI를 설치하려면

1. 개발 컴퓨터에서 다음 명령을 실행하여 리포지토리에서 GitHub 최신 버전의 GDK CLI를 설치합니다.

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

2. 다음 명령을 실행하여 GDK CLI가 성공적으로 설치되었는지 확인합니다.

```
gdk --help
```

gdk명령을 찾을 수 없는 경우 해당 폴더를 PATH에 추가합니다.

- Linux 디바이스에서는 /home/*MyUser*/.local/bin PATH에 추가하고 사용자 *MyUser*이름으로 바꾸십시오.
- Windows 장치에서는 *PythonPath*\\Scripts PATH에 추가하고 장치의 Python 폴더 *PythonPath*경로로 바꾸십시오.

2단계: 업데이트를 연기하는 구성 요소 개발

이 섹션에서는 코어 기기의 배터리 잔량이 구성 요소를 배포할 때 구성한 임계값 미만일 때 구성 요소 업데이트를 연기하는 Hello World 구성 요소를 Python으로 개발합니다. 이 구성 요소에서는 Python용 AWS IoT Device SDK v2의 프로세스 간 통신 (IPC) 인터페이스를 사용합니다. SubscribeToComponentUpdatesIPC 작업을 사용하면 코어 기기가 배포를 수신할 때 알림을 받을 수 있습니다. 그런 다음 DeferComponentUpdateIPC 작업을 사용하여 디바이스의 배터리 잔량을 기반으로 업데이트를 연기하거나 승인합니다.

업데이트를 연기하는 Hello World 구성 요소 개발

1. 개발 컴퓨터에서 구성 요소 소스 코드를 저장할 폴더를 생성합니다.

   ```
   mkdir com.example.BatteryAwareHelloWorld
   cd com.example.BatteryAwareHelloWorld
   ```

2. 텍스트 편집기를 사용하여 이름이 지정된 파일을 생성합니다gdk-config.json. GDK CLI는 gdk-config.json 이름이 지정된 GDK CLI 구성 파일을 읽고 구성 요소를 빌드하고 게시합니다. 이 구성 파일은 구성 요소 폴더의 루트에 있습니다.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano gdk-config.json
   ```

   다음 JSON을 파일에 복사합니다.

   - *Amazon#* 사용자 이름으로 바꾸십시오.

- *us-west-2#* 코어 AWS 리전 디바이스가 작동하는 곳으로 바꾸십시오. GDK CLI는 여기에 구성 요소를 게시합니다. AWS 리전
- 사용할 S3 버킷 *greengrass-component-artifacts*접두사로 바꾸십시오. GDK CLI를 사용하여 구성 요소를 게시하면 GDK CLI는 이 값, 및 AWS 계정 ID로 구성된 S3 버킷에 구성 요소의 아티팩트를 AWS 리전 다음 형식으로 업로드합니다. *bucketPrefix*-*region*-*accountId*

  예를 들어**us-west-2**, **greengrass-component-artifacts** 및 를 지정하고 AWS 계정 ID가 인 경우 GDK CLI는 **123456789012** 이름이 지정된 S3 버킷을 사용합니다. `greengrass-component-artifacts-us-west-2-123456789012`

```
{
  "component": {
    "com.example.BatteryAwareHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

구성 파일은 다음을 지정합니다.

- GDK CLI가 Greengrass 구성 요소를 클라우드 서비스에 게시할 때 사용할 버전입니다. AWS IoT Greengrass `NEXT_PATCH`클라우드 서비스에서 사용할 수 있는 최신 버전 다음에 다음 패치 버전을 선택하도록 지정합니다. AWS IoT Greengrass 구성 요소에 아직 AWS IoT Greengrass 클라우드 서비스 버전이 없는 경우 GDK `1.0.0` CLI에서 버전을 사용합니다.
- 컴포넌트의 빌드 시스템. `zip`빌드 시스템을 사용할 때 GDK CLI는 구성 요소의 소스를 구성 요소의 단일 아티팩트가 되는 ZIP 파일로 패키징합니다.
- GDK CLI가 그린그래스 구성 요소를 게시하는 AWS 리전 곳입니다.
- GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷의 접두사입니다.

3. 텍스트 편집기를 사용하여 라는 파일에 구성 요소 소스 코드를 생성합니다. `main.py`

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano main.py
```

다음 Python 코드를 파일에 복사합니다.

```
import json
import os
import sys
import time
import traceback

from pathlib import Path

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15  # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000  # Milliseconds


class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
 Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                          event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                          event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
```

```
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
            # SubscribeToComponentUpdates returns a tuple with the response and the
operation.
            _, self.subscription_operation =
self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
        if self.subscription_operation is not None:
            self.subscription_operation.close()
            self.subscription_operation = None

    def defer_update(self, deployment_id):
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id,
recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

    def acknowledge_update(self, deployment_id):
        # Specify recheck_after_ms=0 to acknowledge a component update.
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id, recheck_after_ms=0)

    def is_battery_below_threshold(self):
        return self.get_battery_level() < self.battery_threshold

    def get_battery_level(self):
        # Read the battery level from the virtual battery level file.
        with self.battery_file_path.open('r') as f:
            data = json.load(f)
            return float(data['battery_level'])

    def print_message(self):
        message = 'Hello, World!'
        if self.is_battery_below_threshold():
            message += ' Battery level (%d) is below threshold (%d), so the
component will defer updates' % (
                self.get_battery_level(), self.battery_threshold)
        else:
```

```
            message += ' Battery level (%d) is above threshold (%d), so the
 component will acknowledge updates' % (
                self.get_battery_level(), self.battery_threshold)
        print(message)


def main():
    # Read the battery threshold and virtual battery file path from command-line
 args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))

    try:
        # Create an IPC client and a Hello World printer that defers component
 updates.
        ipc_client = GreengrassCoreIPCClientV2()
        hello_world_printer = BatteryAwareHelloWorldPrinter(
            ipc_client, battery_file_path, battery_threshold)
        hello_world_printer.subscribe_to_component_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                hello_world_printer.print_message()
                time.sleep(HELLO_WORLD_PRINT_INTERVAL)
        except InterruptedError:
            print('Subscription interrupted')
        hello_world_printer.close_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

이 Python 응용 프로그램은 다음을 수행합니다.

- 나중에 코어 기기에 생성할 가상 배터리 잔량 파일에서 코어 기기의 배터리 잔량을 읽습니다. 이 가상 배터리 잔량 파일은 실제 배터리를 모방하므로 배터리가 없는 코어 장치에서도 이 자습서를 완료할 수 있습니다.
- 배터리 임계값 및 가상 배터리 잔량 파일 경로에 대한 명령줄 인수를 읽습니다. 구성 요소 레시피는 구성 매개변수를 기반으로 이러한 명령줄 인수를 설정하므로 구성 요소를 배포할 때 이러한 값을 사용자 지정할 수 있습니다.
- Python용 v2의 IPC 클라이언트 AWS IoT Device SDK V2를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 통신합니다. IPC 클라이언트 V2는 원래 IPC 클라이언트와 비교하여 사용자 지정 구성 요소에서 IPC를 사용하기 위해 작성해야 하는 코드의 양을 줄여줍니다.
- IPC 작업을 사용하여 업데이트 알림을 구독합니다. SubscribeToComponentUpdates AWS IoT GreengrassCore 소프트웨어는 각 배포 전후에 알림을 보냅니다. 구성 요소는 알림을 받을 때마다 다음 함수를 호출합니다. 향후 배포에 대한 알림인 경우 구성 요소는 배터리 잔량이 임계값보다 낮은지 확인합니다. 배터리 잔량이 임계값 미만인 경우 구성 요소는 DeferComponentUpdateIPC 작업을 사용하여 업데이트를 30초 동안 연기합니다. 그렇지 않으면 배터리 잔량이 임계값 미만이 아닌 경우 구성 요소가 업데이트를 승인하므로 업데이트를 계속할 수 있습니다.

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어는 로컬 배포에 대한 업데이트 알림을 보내지 않으므로 AWS IoT Greengrass 클라우드 서비스를 사용하여 이 구성 요소를 배포하여 테스트하십시오.

4. 텍스트 편집기를 사용하여 or 라는 `recipe.json` 파일에 구성 요소 레시피를 만드십시오. `recipe.yaml` 구성 요소 레시피는 구성 요소의 메타데이터, 기본 구성 매개 변수 및 플랫폼별 수명 주기 스크립트를 정의합니다.

   JSON

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano recipe.json
   ```

   다음 JSON을 파일에 복사합니다.

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "COMPONENT_NAME",
     "ComponentVersion": "COMPONENT_VERSION",
     "ComponentDescription": "This Hello World component defers updates when the
    battery level is below a threshold.",
     "ComponentPublisher": "COMPONENT_AUTHOR",
     "ComponentConfiguration": {
       "DefaultConfiguration": {
         "BatteryThreshold": 50,
         "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
         "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
       }
     },
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": "python3 -m pip install --user awsiotsdk --upgrade",
   ```

```
        "run": "python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/LinuxBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk --upgrade",
        "run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/WindowsBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

YAML

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano recipe.yaml
```

다음 YAML을 파일에 복사합니다.

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
 battery level is below a threshold."
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade
      run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk --upgrade
      run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/WindowsBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
```

이 레시피는 다음을 지정합니다.

- 배터리 임계값, Linux 코어 디바이스의 가상 배터리 파일 경로, Windows 코어 디바이스의 가상 배터리 파일 경로에 대한 기본 구성 파라미터
- Python용 AWS IoT Device SDK v2의 최신 버전을 설치하는 `install` 라이프사이클입니다.

- 에서 Python 애플리케이션을 실행하는 `run` 라이프사이클`main.py`.
- 구성 요소 레시피를 빌드할 때 `COMPONENT_VERSION` GDK CLI가 정보를 대체하는 자리 표시자(예: `COMPONENT_NAME` 및)

구성 요소 레시피에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass컴포넌트 레시피 참조

3단계: AWS IoT Greengrass 서비스에 구성 요소 게시

이 섹션에서는 Hello World 구성 요소를 AWS IoT Greengrass 클라우드 서비스에 게시합니다. AWS IoT Greengrass클라우드 서비스에서 구성 요소를 사용할 수 있게 되면 핵심 장치에 배포할 수 있습니다. GDK CLI를 사용하여 개발 컴퓨터의 구성 요소를 클라우드 서비스에 게시합니다AWS IoT Greengrass. GDK CLI는 컴포넌트의 레시피와 아티팩트를 자동으로 업로드합니다.

Hello World 구성 요소를 서비스에 게시하려면 AWS IoT Greengrass

1. 다음 명령을 실행하여 GDK CLI를 사용하여 구성 요소를 빌드합니다. 구성 요소 빌드 명령은 GDK CLI 구성 파일을 기반으로 레시피와 아티팩트를 생성합니다. 이 프로세스에서 GDK CLI는 구성 요소의 소스 코드가 포함된 ZIP 파일을 생성합니다.

   ```
   gdk component build
   ```

   다음 예와 비슷한 메시지가 표시되어야 합니다.

   ```
   [2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:16] INFO - Building the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
   [2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
    system. If this is incorrect, please exit and specify custom build command in the
    'gdk-config.json'.
   [2022-04-28 11:20:16] INFO - Zipping source code files of the component.
   [2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
    component artifacts build folder.
   [2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
   ```

```
[2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
\greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
```

2. 다음 명령을 실행하여 구성 요소를 AWS IoT Greengrass 클라우드 서비스에 게시합니다. 구성 요소 게시 명령은 구성 요소의 ZIP 파일 아티팩트를 S3 버킷에 업로드합니다. 그런 다음 구성 요소 레시피에서 ZIP 파일의 S3 URI를 업데이트하고 레시피를 서비스에 업로드합니다. AWS IoT Greengrass 이 프로세스에서 GDK CLI는 클라우드 서비스에서 이미 사용 가능한 Hello World 구성 요소 버전을 확인하여 해당 버전 이후의 다음 패치 버전을 선택할 수 있습니다. AWS IoT Greengrass 구성 요소가 아직 없는 경우 GDK CLI는 버전을 사용합니다. `1.0.0`

```
gdk component publish
```

다음 예와 비슷한 메시지가 표시될 것입니다. 출력은 GDK CLI에서 생성한 구성 요소의 버전을 알려줍니다.

```
[2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2022-04-28 11:20:30] INFO - No private version of the component
 'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
 next version to create.
[2022-04-28 11:20:30] INFO - Publishing the component
 'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
[2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
 bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
 first time using this bucket, add the 's3:GetObject' permission to each core
 device's token exchange role to allow it to download the component artifacts. For
 more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
device-service-role.html.
[2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
[2022-04-28 11:20:30] INFO - Updating the component recipe
 com.example.BatteryAwareHelloWorld-1.0.0.
[2022-04-28 11:20:31] INFO - Creating a new greengrass component
 com.example.BatteryAwareHelloWorld-1.0.0
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
 the account.'com.example.BatteryAwareHelloWorld'.
```

3. 출력에서 S3 버킷 이름을 복사합니다. 나중에 버킷 이름을 사용하여 코어 디바이스가 이 버킷에서 구성 요소 아티팩트를 다운로드하도록 허용할 수 있습니다.
4. (선택 사항) AWS IoT Greengrass 콘솔에서 구성 요소를 보고 성공적으로 업로드되었는지 확인합니다. 다음을 따릅니다.

   a. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.

   b. 구성 요소 페이지에서 내 구성 요소 탭을 선택한 다음 선택합니다 com.example.BatteryAwareHelloWorld.

      이 페이지에서 구성 요소의 레시피와 구성 요소에 대한 기타 정보를 볼 수 있습니다.
5. 코어 디바이스가 S3 버킷의 구성 요소 아티팩트에 액세스할 수 있도록 허용합니다.

   각 코어 디바이스에는 클라우드와 상호 AWS IoT 작용하고 클라우드로 로그를 전송할 수 있는 코어 디바이스 IAM 역할이 있습니다. AWS 이 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않으므로 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색하도록 허용하는 정책을 생성하고 연결해야 합니다.

   디바이스 역할이 이미 S3 버킷에 대한 액세스를 허용하고 있다면 이 단계를 건너뛰어도 됩니다. 그렇지 않으면 다음과 같이 액세스를 허용하는 IAM 정책을 생성하여 역할에 연결하십시오.

   a. IAM 콘솔 탐색 메뉴에서 [Policies] 를 선택한 다음 [Create policy] 를 선택합니다.

   b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다. *greengrass-component-artifacts-us-west-2-123456789012#* GDK CLI가 구성 요소의 아티팩트를 업로드한 S3 버킷의 이름으로 바꾸십시오.

      예를 들어, GDK CLI 구성 파일에서 AWS 계정 and를 **greengrass-component-artifacts** 지정하고 **us-west-2** ID가 인 경우 GDK **123456789012** CLI는 이름이 지정된 S3 버킷을 사용합니다. `greengrass-component-artifacts-us-west-2-123456789012`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
```

```
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-
west-2-123456789012/*"
    }
  ]
}
```

c. 다음을 선택합니다.

d. 정책 세부 정보 섹션의 이름에 을 입력합니다. **MyGreengrassV2ComponentArtifactPolicy**

e. 정책 생성을 선택합니다.

f. IAM 콘솔 탐색 메뉴에서 [Role] 을 선택한 다음 코어 디바이스의 역할 이름을 선택합니다. AWS IoT GreengrassCore 소프트웨어를 설치할 때 이 역할 이름을 지정했습니다. 이름을 지정하지 않은 경우 기본값은 입니다`GreengrassV2TokenExchangeRole`.

g. 권한에서 권한 추가를 선택한 다음 정책 연결을 선택합니다.

h. 권한 추가 페이지에서 생성한 `MyGreengrassV2ComponentArtifactPolicy` 정책 옆의 확인란을 선택한 다음 권한 추가를 선택합니다.

4단계: 코어 디바이스에 구성 요소 배포 및 테스트

이 섹션에서는 구성 요소를 코어 장치에 배포하여 기능을 테스트합니다. 코어 디바이스에서 가상 배터리 잔량 파일을 생성하여 실제 배터리를 모방합니다. 그런 다음 추가 배포를 생성하고 코어 장치의 구성 요소 로그 파일을 관찰하여 구성 요소가 업데이트를 연기하고 승인하는지 확인합니다.

업데이트를 연기하는 Hello World 구성 요소를 배포하고 테스트하려면

1. 텍스트 편집기를 사용하여 가상 배터리 잔량 파일을 생성합니다. 이 파일은 실제 배터리를 모방합니다.

   - Linux 코어 디바이스에서는 라는 파일을 생성합니다. `/home/ggc_user/virtual_battery.json` `sudo`권한을 가지고 텍스트 편집기를 실행합니다.
   - Windows 코어 디바이스에서는 라는 이름의 파일을 생성합니다`C:\Users\ggc_user\virtual_battery.json`. 텍스트 편집기를 관리자 권한으로 실행합니다.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

다음 JSON을 파일에 복사합니다.

```
{
  "battery_level": 50
}
```

2. Hello World 구성 요소를 코어 디바이스에 배포합니다. 다음을 따릅니다.

   a. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.

   b. 구성 요소 페이지에서 내 구성 요소 탭을 선택한 다음 선택합니다 com.example.BatteryAwareHelloWorld.

   c. com.example.BatteryAwareHelloWorld 페이지에서 배포를 선택합니다.

   d. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포를 만들도록 선택한 후 다음을 선택합니다.

   e. 새 배포를 생성하기로 선택한 경우 배포할 대상 코어 장치 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 장치 또는 사물 그룹을 선택하고 다음을 선택합니다.

   f. 구성 요소 선택 페이지에서 구성 com.example.BatteryAwareHelloWorld요소가 선택되었는지 확인하고 다음을 선택합니다.

   g. 구성 요소 구성 페이지에서 구성 요소를 선택하고 com.example.BatteryAwareHelloWorld다음을 수행합니다.

      i. 구성 요소 구성을 선택합니다.

      ii. 구성 com.example.BatteryAwareHelloWorld 모달의 구성 업데이트에 있는 병합할 구성에 다음 구성 업데이트를 입력합니다.

      ```
      {
        "BatteryThreshold": 70
      }
      ```

      iii. 확인을 선택하여 모달을 닫고 다음을 선택합니다.

   h. 고급 설정 확인 페이지의 배포 정책 섹션의 구성 요소 업데이트 정책에서 구성 요소 알림이 선택되어 있는지 확인합니다. 새 배포를 만들 때 구성 요소 알림이 기본적으로 선택됩니다.

   i. 검토 페이지에서 배포를 선택합니다.

   배포를 완료하는 데 최대 1분이 걸릴 수 있습니다.

3. AWS IoT GreengrassCore 소프트웨어는 구성 요소 프로세스의 stdout을 폴더의 `logs` 로그 파일에 저장합니다. 다음 명령을 실행하여 Hello World 구성 요소가 실행되고 상태 메시지를 인쇄하는지 확인합니다.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   다음 예와 비슷한 메시지가 표시될 것입니다.

   ```
   Hello, World! Battery level (50) is below threshold (70), so the component will
    defer updates.
   ```

   > **Note**
   >
   > 파일이 없는 경우 배포가 아직 완료되지 않았을 수 있습니다. 30초 이내에 파일이 존재하지 않으면 배포가 실패했을 수 있습니다. 예를 들어 코어 디바이스에 S3 버킷에서 구성 요소의 아티팩트를 다운로드할 권한이 없는 경우 이 문제가 발생할 수 있습니다. 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.
   >
   > Linux or Unix
   >
   > ```
   > sudo tail -f /greengrass/v2/logs/greengrass.log
   > ```
   >
   > Windows Command Prompt (CMD)
   >
   > ```
   > type C:\greengrass\v2\logs\greengrass.log
   > ```

이 type 명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

4. 코어 디바이스에 새 배포를 생성하여 구성 요소가 업데이트를 연기하는지 확인합니다. 다음을 따릅니다.

   a. AWS IoT Greengrass콘솔 탐색 메뉴에서 배포를 선택합니다.

   b. 이전에 생성하거나 수정한 배포를 선택합니다.

   c. 배포 페이지에서 수정을 선택합니다.

   d. 배포 수정 모달에서 배포 수정을 선택합니다.

   e. 각 단계에서 다음을 선택한 다음 배포를 선택합니다.

5. 다음 명령을 실행하여 구성 요소의 로그를 다시 확인하고 업데이트가 지연되는지 확인합니다.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   다음 예와 비슷한 메시지가 표시되어야 합니다. 구성 요소가 업데이트를 30초 동안 연기하므로 구성 요소는 이 메시지를 반복해서 인쇄합니다.

   ```
   Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
   ```

6. 텍스트 편집기를 사용하여 가상 배터리 잔량 파일을 편집하고 배터리 잔량을 임계값 이상의 값으로 변경하여 배포를 계속할 수 있습니다.

- Linux 코어 디바이스에서는 이름이 지정된 파일을 /home/ggc_user/virtual_battery.json 편집합니다. sudo권한을 가지고 텍스트 편집기를 실행합니다.
- Windows 코어 디바이스에서는 이름이 지정된 파일을 C:\Users\ggc_user\virtual_battery.json 편집합니다. 텍스트 편집기를 관리자 권한으로 실행합니다.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
sudo nano /home/ggc_user/virtual_battery.json
```

배터리 잔량을 로 변경합니다. 80

```
{
  "battery_level": 80
}
```

7. 다음 명령을 실행하여 구성 요소의 로그를 다시 보고 업데이트가 승인되는지 확인합니다.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
```

다음 예와 비슷한 메시지가 표시되어야 합니다.

```
Hello, World! Battery level (80) is above threshold (70), so the component will
 acknowledge updates.
Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
```

이 튜토리얼을 완료했습니다. Hello World 구성 요소는 코어 장치의 배터리 잔량에 따라 업데이트를 연기하거나 승인합니다. 이 자습서에서 다루는 주제에 대한 자세한 내용은 다음을 참조하십시오.

- AWS IoT Greengrass구성 요소 개발
- 디바이스에 AWS IoT Greengrass 구성 요소 배포
- AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core
- AWS IoT Greengrass개발 키트 명령줄 인터페이스

튜토리얼: MQTT를 통해 로컬 IoT 디바이스와 상호 작용

이 자습서를 완료하여 MQTT를 통해 코어 장치에 연결되는 로컬 IoT 장치 (클라이언트 장치라고 함) 와 상호 작용하도록 코어 장치를 구성할 수 있습니다. 이 자습서에서는 클라우드 검색을 사용하여 코어 장치를 클라이언트 장치로 연결하도록 AWS IoT 사물을 구성합니다. 클라우드 검색을 구성하면 클라이언트 장치가 AWS IoT Greengrass 클라우드 서비스에 요청을 보내 핵심 장치를 검색할 수 있습니다. 응답의 응답에는 검색하도록 클라이언트 장치를 구성한 핵심 장치에 대한 연결 정보 및 인증서가 AWS IoT Greengrass 포함됩니다. 그러면 클라이언트 장치는 이 정보를 사용하여 MQTT를 통해 통신할 수 있는 사용 가능한 코어 장치에 연결할 수 있습니다.

이 자습서에서는 다음 작업을 수행합니다.

1. 필요한 경우 코어 디바이스의 권한을 검토하고 업데이트하십시오.
2. 클라이언트 디바이스를 코어 디바이스에 연결하여 클라우드 검색을 사용하여 코어 디바이스를 검색할 수 있도록 합니다.
3. Greengrass 구성 요소를 코어 장치에 배포하여 클라이언트 장치 지원을 활성화합니다.
4. 클라이언트 디바이스를 코어 디바이스에 연결하고 AWS IoT Core 클라우드 서비스와의 통신을 테스트합니다.
5. 클라이언트 장치와 통신하는 사용자 지정 Greengrass 구성 요소를 개발하십시오.
6. 클라이언트 디바이스의 디바이스 AWS IoT 섀도우와 상호 작용하는 사용자 지정 구성 요소를 개발하십시오.

이 자습서에서는 단일 코어 기기와 단일 클라이언트 기기를 사용합니다. 자습서를 따라 여러 클라이언트 장치를 연결하고 테스트할 수도 있습니다.

이 자습서에는 30~60분이 소요될 것으로 예상됩니다.

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- AWS 계정. 계정이 없는 경우 AWS 계정 설정 섹션을 참조하십시오.
- 관리자 권한이 있는 AWS Identity and Access Management (IAM) 사용자.
- 그린그래스 코어 디바이스. 코어 디바이스 설정 방법에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 디바이스 설정.
  - 코어 디바이스는 그린그래스 핵 v2.6.0 이상을 실행해야 합니다. 이 버전에는 로컬 게시/구독 커뮤니케이션의 와일드카드 지원 및 클라이언트 디바이스 섀도 지원이 포함됩니다.

    > **Note**
    >
    > 클라이언트 장치 지원을 위해서는 그린그래스 핵 v2.2.0 이상이 필요합니다. 하지만 이 자습서에서는 로컬 퍼블리시/서브스크립션의 MQTT 와일드카드 지원 및 클라이언트 디바이스 섀도 지원과 같은 새로운 기능을 살펴봅니다. 이러한 기능을 사용하려면 그린그래스 핵 v2.6.0 이상이 필요합니다.

  - 코어 디바이스는 연결할 클라이언트 디바이스와 동일한 네트워크에 있어야 합니다.
  - (선택 사항) 사용자 지정 Greengrass 구성 요소를 개발하는 모듈을 완료하려면 코어 디바이스에서 Greengrass CLI를 실행해야 합니다. 자세한 설명은 그린그래스 CLI 설치 섹션을 참조하세요.
- 이 AWS IoT 튜토리얼에서는 클라이언트 디바이스로 연결하는 방법에 대해 설명합니다. 자세한 내용은 AWS IoT Core개발자 안내서에서 AWS IoT 리소스 만들기를 참조하십시오.
  - 클라이언트 장치의 AWS IoT 정책에서 `greengrass:Discover` 권한을 허용해야 합니다. 자세한 설명은 클라이언트 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.
  - 클라이언트 장치는 코어 장치와 동일한 네트워크에 있어야 합니다.
  - 클라이언트 기기는 Python 3을 실행해야 합니다.
  - 클라이언트 디바이스는 Git을 실행해야 합니다.

1단계: 코어 디바이스 AWS IoT 정책 검토 및 업데이트

클라이언트 장치를 지원하려면 코어 장치 AWS IoT 정책에서 다음 권한을 허용해야 합니다.

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`

- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (선택 사항) 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 IP Detector 구성 요소를 사용하려면 이 권한이 필요합니다.

핵심 장치에 대한 이러한 권한 및 AWS IoT 정책에 대한 자세한 내용은 데이터 영역 작업에 대한 AWS IoT 정책 및 을 참조하십시오클라이언트 AWS IoT 장치를 지원하기 위한 최소 정책.

이 섹션에서는 코어 장치에 대한 AWS IoT 정책을 검토하고 누락된 필수 권한을 추가합니다. AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 코어 장치에는 모든 AWS IoT Greengrass 작업에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다 (`greengrass:*`). 이 경우 섀도우 관리자 구성 요소가 장치 섀도우와 AWS IoT Core 동기화되도록 구성하려는 경우에만 AWS IoT 정책을 업데이트해야 합니다. 그렇지 않으면 이 섹션을 건너뛰어도 됩니다.

핵심 장치 정책을 검토 및 업데이트하려면 AWS IoT

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 코어 기기를 선택합니다.
2. 코어 디바이스 페이지에서 업데이트할 코어 디바이스를 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 코어 디바이스의 사물로 연결되는 링크를 선택합니다. 이 링크를 클릭하면 AWS IoT 콘솔에서 사물 세부 정보 페이지가 열립니다.
4. 사물 세부 정보 페이지에서 인증서를 선택합니다.
5. 인증서 탭에서 사물의 활성 인증서를 선택합니다.
6. 인증서 세부 정보 페이지에서 정책을 선택합니다.
7. 정책 탭에서 검토 및 업데이트할 AWS IoT 정책을 선택합니다. 코어 디바이스의 활성 인증서에 연결된 모든 정책에 필요한 권한을 추가할 수 있습니다.

> **Note**
>
> AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 지정된 정책을 선택하는 것이 좋습니다 (있는 GreengrassV2IoTThingPolicy경우). 빠른 설치 프로그램으로 만든 코어 디바이스는 기본적으로 이 정책 이름을 사용합니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 핵심 장치에도 이러한 권한이 부여됩니다.

8. 정책 개요에서 활성 버전 편집을 선택합니다.
9. 정책에서 필요한 권한을 검토하고 누락된 필수 권한을 추가하십시오.
10. 새 정책 버전을 활성 버전으로 설정하려면 정책 버전 상태에서 편집한 버전을 이 정책의 활성 버전으로 설정을 선택합니다.
11. 새 버전으로 저장을 선택합니다.

2단계: 클라이언트 장치 지원 활성화

클라이언트 장치가 클라우드 검색을 사용하여 코어 장치에 연결하려면 장치를 연결해야 합니다. 클라이언트 장치를 코어 장치에 연결하면 해당 클라이언트 장치가 연결에 사용할 핵심 장치의 IP 주소 및 인증서를 검색할 수 있도록 합니다.

클라이언트 장치를 코어 장치에 안전하게 연결하고 Greengrass 구성 요소와 통신할 수 있도록 하려면 다음 Greengrass 구성 요소를 코어 장치에 배포합니다. AWS IoT Core

- 클라이언트 장치 인증 (`aws.greengrass.clientdevices.Auth`)

  클라이언트 장치 인증 구성 요소를 배포하여 클라이언트 장치를 인증하고 클라이언트 장치 작업을 승인합니다. 이 구성 요소를 사용하면 AWS IoT 사물을 코어 장치에 연결할 수 있습니다.

  이 구성 요소를 사용하려면 몇 가지 구성이 필요합니다. 클라이언트 장치 그룹과 각 그룹이 수행할 권한이 있는 작업 (예: MQTT를 통한 연결 및 통신) 을 지정해야 합니다. 자세한 내용은 클라이언트 장치 인증 구성 요소 구성을 참조하십시오.
- MQTT 3.1.1 브로커 (모켓) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Mocquette MQTT 브로커 구성 요소를 배포하여 경량 MQTT 브로커를 실행하십시오. Moquette MQTT 브로커는 MQTT 3.1.1과 호환되며 QoS 0, QoS 1, QoS 2, 보존 메시지, 라스트 윌 메시지 및 영구 구독에 대한 로컬 지원을 포함합니다.

  이 구성 요소를 사용하기 위해 구성하지 않아도 됩니다. 하지만 이 구성 요소가 MQTT 브로커를 작동하는 포트를 구성할 수 있습니다. 기본적으로 포트 8883을 사용합니다.
- MQTT 브리지 (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (선택 사항) MQTT 브리지 구성 요소를 배포하여 클라이언트 장치 (로컬 MQTT), 로컬 게시/구독 및 MQTT 간에 메시지를 릴레이합니다. AWS IoT Core 클라이언트 장치를 Greengrass 구성 요소의 클라이언트 장치와 AWS IoT Core 동기화하고 클라이언트 장치와 상호 작용하도록 이 구성 요소를 구성합니다.

이 구성 요소를 사용하려면 구성이 필요합니다. 이 구성 요소가 메시지를 릴레이하는 주제 매핑을 지정해야 합니다. 자세한 내용은 MQTT 브리지 구성 요소 구성을 참조하십시오.

- IP 감지기 (`aws.greengrass.clientdevices.IPDetector`)

  (선택 사항) 코어 디바이스의 MQTT 브로커 엔드포인트를 클라우드 서비스에 자동으로 보고하도록 IP 탐지기 구성 요소를 배포하십시오AWS IoT Greengrass. 라우터가 MQTT 브로커 포트를 코어 디바이스에 전달하는 것과 같이 복잡한 네트워크 설정이 있는 경우에는 이 구성 요소를 사용할 수 없습니다.

  이 구성 요소를 사용하도록 구성하지 않아도 됩니다.

이 섹션에서는 AWS IoT Greengrass 콘솔을 사용하여 클라이언트 장치를 연결하고 클라이언트 장치 구성 요소를 코어 장치에 배포합니다.

클라이언트 장치 지원을 활성화하려면

1. AWS IoT Greengrass 콘솔로 이동합니다.
2. 왼쪽 탐색 메뉴에서 코어 기기를 선택합니다.
3. 코어 디바이스 페이지에서 클라이언트 디바이스 지원을 활성화하려는 코어 디바이스를 선택합니다.
4. 코어 장치 세부 정보 페이지에서 클라이언트 장치 탭을 선택합니다.
5. 클라이언트 디바이스 탭에서 클라우드 검색 구성을 선택합니다.

   코어 디바이스 검색 구성 페이지가 열립니다. 이 페이지에서 클라이언트 장치를 코어 장치에 연결하고 클라이언트 장치 구성 요소를 배포할 수 있습니다. 이 페이지에서는 1단계: 대상 코어 장치 선택에서 자동으로 코어 장치를 선택합니다.

   > **Note**
   >
   > 또한 이 페이지를 사용하여 사물 그룹에 대한 코어 장치 검색을 구성할 수 있습니다. 이 옵션을 선택하면 사물 그룹의 모든 핵심 장치에 클라이언트 장치 구성 요소를 배포할 수 있습니다. 하지만 이 옵션을 선택하면 나중에 배포를 생성한 후 클라이언트 디바이스를 각 코어 디바이스에 수동으로 연결해야 합니다. 이 자습서에서는 싱글 코어 디바이스를 구성합니다.

6. 2단계: 클라이언트 장치 연결에서 클라이언트 장치의 AWS IoT 사물을 코어 장치에 연결합니다. 이렇게 하면 클라이언트 장치가 클라우드 검색을 사용하여 코어 장치의 연결 정보 및 인증서를 검색할 수 있습니다. 다음을 따릅니다.

   a. [클라이언트 장치 연결] 을 선택합니다.

   b. 클라이언트 장치를 코어 장치와 연결 모달에 연결할 사물의 AWS IoT 이름을 입력합니다.

   c. Add(추가)를 선택합니다.

   d. Associate(연결)를 선택합니다.

7. 3단계: Greengrass 구성 요소 구성 및 배포에서 구성 요소를 배포하여 클라이언트 장치 지원을 활성화합니다. 대상 코어 장치에 이전 배포가 있는 경우 이 페이지는 해당 배포를 수정합니다. 그렇지 않으면 이 페이지에서 코어 장치에 대한 새 배포를 생성합니다. 클라이언트 장치 구성 요소를 구성하고 배포하려면 다음과 같이 하십시오.

   a. 이 튜토리얼을 완료하려면 코어 디바이스에서 Greengrass nucleus v2.6.0 이상을 실행해야 합니다. 코어 디바이스가 이전 버전을 실행하는 경우 다음을 수행하십시오.

      i. 상자를 선택하여 aws.greengrass.Nucleus구성 요소를 배포합니다.

      ii. aws.greengrass.Nucleus구성 요소의 경우 구성 편집을 선택합니다.

      iii. 구성 요소 버전의 경우 버전 2.6.0 이상을 선택합니다.

      iv. 확인을 선택합니다.

      > **Note**
      >
      > Greengrass nucleus를 이전 마이너 버전에서 업그레이드하고 코어 디바이스가 핵에 종속된 AWS-제공 구성 요소를 실행하는 경우 AWS -제공 구성 요소도 최신 버전으로 업데이트해야 합니다. 이 자습서의 뒷부분에서 배포를 검토할 때 이러한 구성 요소의 버전을 구성할 수 있습니다. 자세한 설명은 AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트 섹션을 참조하세요.

   b. aws.greengrass.clientdevices.Auth구성 요소의 경우 구성 편집을 선택합니다.

   c. 클라이언트 장치 인증 구성 요소의 구성 편집 모달에서 클라이언트 장치가 코어 장치의 MQTT 브로커를 게시하고 구독하도록 허용하는 권한 부여 정책을 구성합니다. 다음을 따릅니다.

i. 구성에서 코드를 병합할 구성 블록에 클라이언트 장치 권한 부여 정책이 포함된 다음 구성을 입력합니다. 각 장치 그룹 권한 부여 정책은 일련의 작업과 클라이언트 장치가 해당 작업을 수행할 수 있는 리소스를 지정합니다.

- 이 정책은 이름이 로 MyClientDevice 시작하는 클라이언트 장치가 모든 MQTT 주제에 연결하고 통신할 수 있도록 허용합니다. *MyClientDevice*#* 클라이언트 장치로 연결할 AWS IoT 사물의 이름으로 바꾸십시오. *와일드카드로 클라이언트 장치 이름과 일치하는 이름을 지정할 수도 있습니다. *와일드카드는 이름 끝에 있어야 합니다.

  연결할 두 번째 클라이언트 장치가 있는 경우, *MyOtherClientDevice*#* 해당 클라이언트 장치의 이름 또는 해당 클라이언트 장치의 이름과 일치하는 와일드카드 패턴으로 바꾸십시오. 그렇지 않으면 이름이 일치하는 MyOtherClientDevice* 클라이언트 장치가 연결 및 통신할 수 있도록 허용하는 선택 규칙의 이 섹션을 제거하거나 유지할 수 있습니다.
- 이 정책은 OR 운영자를 사용하여 이름이 로 MyOtherClientDevice 시작하는 클라이언트 장치가 모든 MQTT 주제에 대해 연결 및 통신할 수 있도록 허용합니다. 선택 규칙에서 이 조항을 제거하거나 연결할 클라이언트 장치에 맞게 수정할 수 있습니다.
- 이 정책을 통해 클라이언트 장치는 모든 MQTT 주제를 게시하고 구독할 수 있습니다. 최상의 보안 관행을 따르려면 mqtt:publish 및 mqtt:subscribe 작업을 클라이언트 장치가 통신하는 데 사용하는 최소 주제 집합으로 제한하십시오.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice* OR
 thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
```

```
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish to all
 topics.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to all
 topics.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

자세한 내용은 클라이언트 장치 인증 구성 요소 구성을 참조하십시오.

ii. 확인을 선택합니다.

d. aws.greengrass.clientdevices.mqtt.Bridge구성 요소의 경우 구성 편집을 선택합니다.

e. MQTT 브리지 구성 요소의 구성 편집 모달에서 클라이언트 장치의 MQTT 메시지를 릴레이하는 주제 매핑을 구성합니다. AWS IoT Core 다음을 따릅니다.

i. 구성의 코드 병합을 위한 구성 블록에 다음 구성을 입력합니다. 이 구성은 `clients/+/hello/world` 주제 필터에 대한 MQTT 메시지를 클라이언트 장치에서 AWS IoT Core 클라우드 서비스로 릴레이하도록 지정합니다. 예를 들어, 이 주제 필터는 주제와 일치합니다. `clients/MyClientDevice1/hello/world`

```
{
```

```
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

자세한 내용은 MQTT 브리지 구성 요소 구성을 참조하십시오.

ii. 확인을 선택합니다.

8. 검토 및 배포를 선택하여 이 페이지에서 생성한 배포를 검토하십시오.
9. 이 지역에서 Greengrass 서비스 역할을 이전에 설정하지 않은 경우 콘솔에서 서비스 역할을 설정하는 모드가 열립니다. 클라이언트 장치 인증 구성 요소는 이 서비스 역할을 사용하여 클라이언트 장치의 ID를 확인하고 IP 탐지기 구성 요소는 이 서비스 역할을 사용하여 핵심 장치 연결 정보를 관리합니다. 권한 부여를 선택합니다.
10. 검토 페이지에서 [Deploy] 를 선택하여 코어 장치에 대한 배포를 시작합니다.
11. 배포가 성공했는지 확인하려면 배포 상태를 확인하고 코어 장치의 로그를 확인하십시오. 코어 디바이스의 배포 상태를 확인하려면 배포 개요에서 타겟을 선택하면 됩니다. 자세한 내용은 다음 자료를 참조하세요.

    - 배포 상태를 확인합니다 의 상태를 확인하세요 의
    - 모니터 AWS IoT Greengrass 로그

3단계: 클라이언트 장치 연결

클라이언트 장치는 AWS IoT Device SDK 를 사용하여 코어 장치를 검색, 연결 및 통신할 수 있습니다. 클라이언트 디바이스는 AWS IoT 사물이어야 합니다. 자세한 내용은 AWS IoT Core개발자 안내서의 사물 객체 만들기를 참조하십시오.

이 섹션에서는 Python용 AWS IoT Device SDK v2를 설치하고 에서 Greengrass 검색 샘플 애플리케이션을 실행합니다. AWS IoT Device SDK

> **Note**
>
> 다른 프로그래밍 언어로도 사용할 수 있습니다. AWS IoT Device SDK 이 자습서에서는 Python용 AWS IoT Device SDK v2를 사용하지만 사용 사례에 맞는 다른 SDK도 살펴볼 수 있

습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 AWS IoT 디바이스 SDK 섹션을 참조하세요.

클라이언트 디바이스를 코어 디바이스에 연결하려면

1. AWS IoT Device SDKv2용 Python을 AWS IoT 다운로드하여 클라이언트 장치로 연결할 사물에 설치합니다.

   클라이언트 기기에서 다음을 수행하십시오.

   a. AWS IoT Device SDKv2 for Python 저장소를 복제하여 다운로드하십시오.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Python용 AWS IoT Device SDK v2를 설치합니다.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Python용 AWS IoT Device SDK v2의 샘플 폴더로 변경합니다.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. 샘플 Greengrass 검색 애플리케이션을 실행합니다. 이 애플리케이션에는 클라이언트 디바이스 사물 이름, 사용할 MQTT 주제 및 메시지, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다. 다음 예제에서는 주제에 Hello World 메시지를 보냅니다. `clients/MyClientDevice1/hello/world`

   > **Note**
   >
   > 이 항목은 메시지를 AWS IoT Core 이전으로 릴레이하도록 MQTT 브리지를 구성한 주제와 일치합니다.

   - *MyClientDevice1#* 클라이언트 디바이스의 사물 이름으로 바꾸십시오.
   - *~/certs/ AmazonRoot ca1.pem# ##### ##### Amazon ## CA ### ###* 바꾸십시오.
   - *~/certs/device.pem.crt#* 클라이언트 디바이스의 디바이스 인증서 경로로 바꾸십시오.
   - *~/certs/private.pem.key#* 클라이언트 장치의 개인 키 파일 경로로 바꾸십시오.

- *us-east-1#* 클라이언트 디바이스 및 코어 디바이스가 운영되는 지역으로 바꾸십시오. AWS

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

검색 샘플 애플리케이션은 메시지를 10번 전송하고 연결을 끊습니다. 또한 메시지를 게시하는 동일한 주제를 구독합니다. 애플리케이션에서 해당 주제에 대한 MQTT 메시지를 수신했다고 출력에 표시되면 클라이언트 기기는 코어 기기와 성공적으로 통신할 수 있습니다.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'
```

```
...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

애플리케이션이 대신 오류를 출력하는 경우 Greengrass 검색 문제 해결을 참조하십시오.

또한 코어 디바이스의 Greengrass 로그를 확인하여 클라이언트 디바이스가 성공적으로 연결하고 메시지를 전송하는지 확인할 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

4. MQTT 브리지가 클라이언트 장치의 메시지를 중계하는지 확인하십시오. AWS IoT Core AWS IoT Core콘솔의 MQTT 테스트 클라이언트를 사용하여 MQTT 주제 필터를 구독할 수 있습니다. 다음을 따릅니다.

   a. AWS IoT 콘솔로 이동합니다.

   b. 왼쪽 탐색 메뉴의 테스트에서 MQTT 테스트 클라이언트를 선택합니다.

   c. 주제 구독 탭의 주제 필터에 코어 기기의 클라이언트 장치 메시지를 `clients/+/hello/world` 구독하도록 입력합니다.

   d. 구독을 선택합니다.

   e. 클라이언트 장치에서 게시/구독 애플리케이션을 다시 실행합니다.

      MQTT 테스트 클라이언트는 이 주제 필터와 일치하는 주제에 대해 클라이언트 장치에서 보내는 메시지를 표시합니다.

4단계: 클라이언트 기기와 통신하는 구성 요소 개발

클라이언트 장치와 통신하는 Greengrass 구성 요소를 개발할 수 있습니다. 구성 요소는 IPC (프로세스 간 통신) 및 로컬 게시/구독 인터페이스를 사용하여 코어 장치에서 통신합니다. 클라이언트 장치와 상호 작용하려면 클라이언트 장치와 로컬 게시/구독 인터페이스 간에 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성하십시오.

이 섹션에서는 클라이언트 장치의 메시지를 로컬 게시/구독 인터페이스로 릴레이하도록 MQTT 브리지 구성 요소를 업데이트합니다. 그런 다음 이러한 메시지를 구독하고 메시지를 수신하면 메시지를 인쇄하는 구성 요소를 개발합니다.

클라이언트 장치와 통신하는 구성 요소 개발

1. 배포를 코어 장치로 수정하고 클라이언트 장치의 메시지를 로컬 게시/구독으로 릴레이하도록 MQTT 브리지 구성 요소를 구성합니다. 다음을 따릅니다.

   a. AWS IoT Greengrass 콘솔로 이동합니다.

   b. 왼쪽 탐색 메뉴에서 Core devices를 선택합니다.

   c. 코어 디바이스 페이지에서 이 튜토리얼에 사용 중인 코어 디바이스를 선택합니다.

   d. 코어 장치 세부 정보 페이지에서 클라이언트 장치 탭을 선택합니다.

   e. 클라이언트 디바이스 탭에서 클라우드 검색 구성을 선택합니다.

      코어 디바이스 검색 구성 페이지가 열립니다. 이 페이지에서 코어 장치에 배포할 클라이언트 장치 구성 요소를 변경하거나 구성할 수 있습니다.

   f. 3단계에서 aws.greengrass.clientdevices.mqtt.Bridge구성 요소에 대해 구성 편집을 선택합니다.

   g. MQTT 브리지 구성 요소의 구성 편집 모달에서 클라이언트 장치의 MQTT 메시지를 로컬 게시/구독 인터페이스로 릴레이하는 주제 매핑을 구성합니다. 다음을 따릅니다.

      i. 구성의 코드 병합 구성 블록에 다음 구성을 입력합니다. 이 구성은 주제 필터와 일치하는 주제에 대한 MQTT 메시지를 클라이언트 장치에서 AWS IoT Core 클라우드 서비스 및 로컬 Greengrass 게시/구독 브로커로 중계하도록 지정합니다. `clients/+/hello/world`

      ```
      {
        "mqttTopicMapping": {
          "HelloWorldIotCoreMapping": {
            "topic": "clients/+/hello/world",
            "source": "LocalMqtt",
            "target": "IotCore"
          },
          "HelloWorldPubsubMapping": {
            "topic": "clients/+/hello/world",
            "source": "LocalMqtt",
            "target": "Pubsub"
          }
        }
      }
      ```

      자세한 내용은 MQTT 브리지 구성 요소 구성을 참조하십시오.

ii. 확인을 선택합니다.

h. 검토 및 배포를 선택하여 이 페이지에서 생성한 배포를 검토하십시오.

i. 검토 페이지에서 [Deploy] 를 선택하여 코어 장치에 대한 배포를 시작합니다.

j. 배포가 성공했는지 확인하려면 배포 상태를 확인하고 코어 장치의 로그를 확인하십시오. 코어 디바이스의 배포 상태를 확인하려면 배포 개요에서 타겟을 선택하면 됩니다. 자세한 내용은 다음 자료를 참조하세요.

- 배포 상태를 확인합니다 의 상태를 확인하세요 의
- 모니터 AWS IoT Greengrass 로그

2. 클라이언트 장치의 Hello World 메시지를 구독하는 Greengrass 구성 요소를 개발하고 배포합니다. 다음을 따릅니다.

a. 코어 디바이스에 레시피와 아티팩트를 저장할 폴더를 만드세요.

Linux or Unix

```
mkdir recipes
mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

PowerShell

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

> **Important**
>
> 아티팩트 폴더 경로에는 다음 형식을 사용해야 합니다. 레시피에 지정한 구성 요소 이름과 버전을 포함하십시오.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. 텍스트 편집기를 사용하여 다음 내용이 포함된 구성 요소 레시피를 생성합니다. 이 레시피는 Python용 AWS IoT Device SDK v2를 설치하고 주제를 구독하고 메시지를 인쇄하는 스크립트를 실행하도록 지정합니다.

예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

```
nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
```

다음 레시피를 파일에 복사합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to Hello World messages
 from client devices.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
```

```
        "run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    }
  ]
}
```

c. 텍스트 편집기를 사용하여 다음 `hello_world_subscriber.py` 내용으로 이름이 지정된 Python 스크립트 아티팩트를 만드십시오. 이 응용 프로그램은 게시/구독 IPC 서비스를 사용하여 주제를 구독하고 `clients/+/hello/world` 주제를 수신한 메시지를 인쇄합니다.

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

   다음 Python 코드를 파일에 복사합니다.

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients/+/hello/world'
TIMEOUT = 10


def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
```

```
        traceback.print_exc()


try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
 on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')

    operation.close()
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

> **Note**
>
> 이 구성 요소는 Python용 v2의 IPC 클라이언트 AWS IoT Device SDK V2를 사용하여 AWS IoT Greengrass Core 소프트웨어와 통신합니다. IPC 클라이언트 V2는 원래 IPC 클라이언트와 비교하여 사용자 지정 구성 요소에서 IPC를 사용하기 위해 작성해야 하는 코드의 양을 줄여줍니다.

d. Greengrass CLI를 사용하여 구성 요소를 배포합니다.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

3. 구성 요소 로그를 보고 구성 요소가 성공적으로 설치되고 주제를 구독하는지 확인하십시오.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MyHelloWorldSubscriber.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
Tail 10 -Wait
```

로그 피드를 열어 두어 코어 장치가 메시지를 수신하는지 확인할 수 있습니다.

4. 클라이언트 디바이스에서 샘플 Greengrass 검색 애플리케이션을 다시 실행하여 코어 디바이스로 메시지를 전송합니다.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

5. 구성 요소 로그를 다시 보고 구성 요소가 클라이언트 장치로부터 메시지를 수신하고 인쇄하는지 확인하십시오.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MyHelloWorldSubscriber.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
   Tail 10 -Wait
   ```

5단계: 클라이언트 장치 섀도우와 상호 작용하는 구성 요소 개발

클라이언트 장치의 장치 섀도우와 상호 작용하는 Greengrass 구성 요소를 개발할 수 있습니다. AWS IoT 섀도우는 클라이언트 장치와 같은 사물에 대한 현재 또는 원하는 상태 정보를 저장하는 JSON 문서입니다. AWS IoT 사용자 지정 구성 요소는 클라이언트 장치가 연결되어 있지 않은 경우에도 클라이언트 장치의 섀도에 액세스하여 상태를 관리할 수 있습니다. AWS IoT 각 AWS IoT 오브젝트에는 이름이 지정되지 않은 섀도우가 있으며, 각 오브젝트에 대해 이름이 지정된 섀도우를 여러 개 만들 수도 있습니다.

이 섹션에서는 섀도우 관리자 컴포넌트를 배포하여 코어 디바이스의 섀도우를 관리합니다. 또한 MQTT 브리지 구성 요소를 업데이트하여 클라이언트 장치와 섀도우 관리자 구성 요소 간에 섀도우 메시지를 릴레이할 수 있습니다. 그런 다음 클라이언트 장치의 섀도를 업데이트하는 구성 요소를 개발하고 구성 요소의 섀도우 업데이트에 응답하는 샘플 응용 프로그램을 클라이언트 장치에서 실행합니다. 이 구성 요소는 핵심 장치가 클라이언트 장치로 연결되는 스마트 조명의 색상 상태를 관리하는 스마트 조명 관리 응용 프로그램을 나타냅니다.

클라이언트 장치 섀도우와 상호 작용하는 구성 요소 개발

1. 배포를 코어 장치로 수정하여 섀도우 관리자 구성 요소를 배포하고, 클라이언트 장치와 섀도우 관리자가 통신하는 로컬 게시/구독 간에 섀도우 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성합니다. 다음을 따릅니다.

   a. AWS IoT Greengrass 콘솔로 이동합니다.
   b. 왼쪽 탐색 메뉴에서 Core devices를 선택합니다.
   c. 코어 디바이스 페이지에서 이 튜토리얼에 사용 중인 코어 디바이스를 선택합니다.

d. 코어 장치 세부 정보 페이지에서 클라이언트 장치 탭을 선택합니다.

e. 클라이언트 디바이스 탭에서 클라우드 검색 구성을 선택합니다.

   코어 디바이스 검색 구성 페이지가 열립니다. 이 페이지에서 코어 장치에 배포할 클라이언트 장치 구성 요소를 변경하거나 구성할 수 있습니다.

f. 3단계에서 aws.greengrass.clientdevices.mqtt.Bridge구성 요소에 대해 구성 편집을 선택합니다.

g. MQTT 브리지 구성 요소의 구성 편집 모달에서 클라이언트 장치와 로컬 게시/구독 인터페이스 간에 장치 섀도우 주제에 대한 MQTT 메시지를 릴레이하는 주제 매핑을 구성합니다. 또한 배포에 호환되는 MQTT 브리지 버전이 지정되어 있는지 확인합니다. 클라이언트 디바이스 섀도를 지원하려면 MQTT 브리지 v2.2.0 이상이 필요합니다. 다음을 따릅니다.

   i. 구성 요소 버전의 경우 버전 2.2.0 이상을 선택하십시오.

   ii. 구성의 코드를 병합할 구성 블록에 다음 구성을 입력합니다. 이 구성은 섀도우 주제에 대해 MQTT 메시지를 릴레이하도록 지정합니다.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

자세한 내용은 MQTT 브리지 구성 요소 구성을 참조하십시오.

iii. 확인을 선택합니다.

h. 3단계에서 배포할 aws.greengrass.ShadowManager구성 요소를 선택합니다.

i. 검토 및 배포를 선택하여 이 페이지에서 생성한 배포를 검토하십시오.

j. 검토 페이지에서 [Deploy] 를 선택하여 코어 장치에 대한 배포를 시작합니다.

k. 배포가 성공했는지 확인하려면 배포 상태를 확인하고 코어 장치의 로그를 확인하십시오. 코어 디바이스의 배포 상태를 확인하려면 배포 개요에서 타겟을 선택하면 됩니다. 자세한 내용은 다음 자료를 참조하세요.

- 배포 상태를 확인합니다 의 상태를 확인하세요 의
- 모니터 AWS IoT Greengrass 로그

2. 스마트 조명 클라이언트 장치를 관리하는 Greengrass 구성 요소를 개발하고 배포하십시오. 다음을 따릅니다.

a. 코어 디바이스에 컴포넌트의 아티팩트가 들어 있는 폴더를 생성합니다.

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

> **Important**
>
> 아티팩트 폴더 경로에는 다음 형식을 사용해야 합니다. 레시피에 지정한 구성 요소 이름과 버전을 포함하십시오.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. 텍스트 편집기를 사용하여 다음 내용이 포함된 구성 요소 레시피를 생성합니다. 이 레시피는 Python용 AWS IoT Device SDK v2를 설치하고 스마트 조명 클라이언트 장치의 그림자와 상호 작용하여 색상을 관리하는 스크립트를 실행하도록 지정합니다.

예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 만들 수 있습니다.

```
nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
```

다음 레시피를 파일에 복사합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MySmartLightManager",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that interacts with smart light client
 devices.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.Nucleus": {
      "VersionRequirement": "^2.6.0"
    },
    "aws.greengrass.ShadowManager": {
      "VersionRequirement": "^2.2.0"
    },
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "VersionRequirement": "^2.2.0"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "smartLightDeviceNames": [],
      "accessControl": {
        "aws.greengrass.ShadowManager": {
          "com.example.clientdevices.MySmartLightManager:shadow:1": {
            "policyDescription": "Allows access to client devices' unnamed
 shadows",
            "operations": [
              "aws.greengrass#GetThingShadow",
              "aws.greengrass#UpdateThingShadow"
            ],
            "resources": [
```

```
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        },
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MySmartLightManager:pubsub:1": {
            "policyDescription": "Allows access to client devices' unnamed
 shadow updates",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "$aws/things/+/shadow/update/accepted"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/smart_light_manager.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/smart_light_manager.py"
      }
    }
  ]
}
```

c. 텍스트 편집기를 사용하여 다음 `smart_light_manager.py` 내용으로 이름이 지정된 Python 스크립트 아티팩트를 만드십시오. 이 애플리케이션은 섀도우 IPC 서비스를 사용하여

클라이언트 디바이스 섀도를 가져오고 업데이트하고 로컬 게시/구독 IPC 서비스를 사용하여 보고된 섀도우 업데이트를 수신합니다.

예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py
```

다음 Python 코드를 파일에 복사합니다.

```
import json
import random
import sys
import time
import traceback
from uuid import uuid4

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import ResourceNotFoundError

SHADOW_COLOR_PROPERTY = 'color'
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'
SET_COLOR_INTERVAL = 15


class SmartLightDevice():
    def __init__(self, client_device_name: str, reported_color: str = None):
        self.name = client_device_name
        self.reported_color = reported_color
        self.desired_color = None


class SmartLightDeviceManager():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):
        self.ipc_client = ipc_client
        self.devices = {}
        self.client_tokens = set()
        self.shadow_update_accepted_subscription_operation = None
        self.client_device_names_configuration_subscription_operation = None
```

```
        self.update_smart_light_device_list()

    def update_smart_light_device_list(self):
        # Update the device list from the component configuration.
        response = self.ipc_client.get_configuration(
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
        # Identify the difference between the configuration and the currently
tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
set(current_device_names)
        removed_device_names = set(current_device_names) -
set(updated_device_names)
        # Stop tracking any smart light devices that are no longer in the
configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)
        # Start tracking any new smart light devices that are in the
configuration.
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
device.reported_color))

    def get_device_reported_color(self, smart_light_device):
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
```

```
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color
                }
            },
            'clientToken': client_token
        }
        payload = bytes(json.dumps(payload_json), 'utf-8')
        self.ipc_client.update_thing_shadow(
            thing_name=smart_light_device.name, shadow_name='',
payload=payload)
        smart_light_device.desired_color = color

    def subscribe_to_shadow_update_accepted_events(self):
        if self.shadow_update_accepted_subscription_operation == None:
            # SubscribeToTopic returns a tuple with the response and the
operation.
            _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
                topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
            print('Successfully subscribed to shadow update accepted topic')

    def close_shadow_update_accepted_subscription(self):
        if self.shadow_update_accepted_subscription_operation is not None:
            self.shadow_update_accepted_subscription_operation.close()

    def on_shadow_update_accepted_event(self, event):
        try:
            message = str(event.binary_message.message, 'utf-8')
            accepted_payload = json.loads(message)
            # Check for reported states from smart light devices and ignore
desired states from components.
            if 'reported' in accepted_payload['state']:
                # Process this update only if it uses a client token created by
this component.
                client_token = accepted_payload.get('clientToken')
                if client_token is not None and client_token in
self.client_tokens:
                    self.client_tokens.remove(client_token)
                    shadow_state = accepted_payload['state']['reported']
```

```
                    if SHADOW_COLOR_PROPERTY in shadow_state:
                        reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                        topic = event.binary_message.context.topic
                        client_device_name = topic.split('/')[2]
                        if client_device_name in self.devices:
                            # Set the reported color for the smart light
device.
                            self.devices[client_device_name].reported_color =
reported_color
                            print(
                                'Received shadow update confirmation from
client device: %s' % client_device_name)
                    else:
                        print("Shadow update doesn't specify color")
        except:
            traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
            _, self.client_device_names_configuration_subscription_operation =
self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
light device names')

    def close_client_device_names_configuration_subscription(self):
        if self.client_device_names_configuration_subscription_operation is not
None:

self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
event.configuration_update_event.key_path:
                print('Received configuration update for list of client
devices')
                self.update_smart_light_device_list()
        except:
```

```
            traceback.print_exc()


def choose_random_color():
    return random.choice(COLORS)

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

 smart_light_manager.subscribe_to_client_device_name_configuration_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                # Set each smart light device to a random color at a regular
 interval.
                for device_name in smart_light_manager.devices:
                    device = smart_light_manager.devices[device_name]
                    desired_color = choose_random_color()
                    print('Chose random color (%s) for %s' %
                          (desired_color, device_name))
                    if desired_color == device.desired_color:
                        print('Desired color for %s is already %s' %
                              (device_name, desired_color))
                    elif desired_color == device.reported_color:
                        print('Reported color for %s is already %s' %
                              (device_name, desired_color))
                    else:
                        smart_light_manager.request_device_color_change(
                            device, desired_color)
                        print('Requested color change for %s to %s' %
                              (device_name, desired_color))
                time.sleep(SET_COLOR_INTERVAL)
        except InterruptedError:
            print('Application interrupted')
        smart_light_manager.close_shadow_update_accepted_subscription()

 smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
```

```
        exit(1)


if __name__ == '__main__':
    main()
```

이 Python 응용 프로그램은 다음을 수행합니다.

- 구성 요소의 구성을 읽고 관리할 스마트 조명 클라이언트 장치 목록을 가져옵니다.
- SubscribeToConfigurationUpdateIPC 작업을 사용하여 구성 업데이트 알림을 구독합니다. AWS IoT GreengrassCore 소프트웨어는 구성 요소의 구성이 변경될 때마다 알림을 보냅니다. 구성 요소가 구성 업데이트 알림을 받으면 구성 요소가 관리하는 스마트 조명 클라이언트 장치 목록을 업데이트합니다.
- 각 스마트 조명 클라이언트 장치의 섀도우를 가져와 초기 색상 상태를 가져옵니다.
- 각 스마트 조명 클라이언트 장치의 색상을 15초마다 임의의 색상으로 설정합니다. 컴포넌트는 클라이언트 디바이스의 사물 섀도우를 업데이트하여 색상을 변경합니다. 이 작업은 MQTT를 통해 클라이언트 디바이스에 섀도우 델타 이벤트를 전송합니다.
- IPC 작업을 사용하여 로컬 게시/구독 인터페이스에서 섀도우 업데이트 허용 메시지를 구독합니다. SubscribeToTopic 이 구성 요소는 이러한 메시지를 수신하여 각 스마트 조명 클라이언트 장치의 색상을 추적합니다. 스마트 조명 클라이언트 장치가 섀도우 업데이트를 수신하면 MQTT 메시지를 보내 업데이트를 받았는지 확인합니다. MQTT 브리지는 이 메시지를 로컬 게시/구독 인터페이스로 전달합니다.

d. Greengrass CLI를 사용하여 구성 요소를 배포합니다. 이 구성 요소를 배포할 때는 섀도우를 관리하는 클라이언트 장치 목록을 지정합니다. `smartLightDeviceNames` *MyClientDevice1#* 클라이언트 디바이스의 사물 이름으로 바꾸십시오.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
```

```
    }
  }'
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
  --update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

3. 구성 요소 로그를 보고 구성 요소가 성공적으로 설치되고 실행되는지 확인합니다.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MySmartLightManager.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
 10 -Wait
```

구성 요소가 스마트 조명 클라이언트 장치의 색상 변경 요청을 보냅니다. 섀도우 관리자는 요청을 수신하고 섀도우의 desired 상태를 설정합니다. 하지만 스마트 라이트 클라이언트 장치가 아직 실행되고 있지 않으므로 섀도우의 reported 상태는 변경되지 않습니다. 구성 요소 로그에는 다음 메시지가 포함됩니다.

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

로그 피드를 열어 두고 구성 요소가 메시지를 인쇄하는 시기를 확인할 수 있습니다.

4. Greengrass Discovery를 사용하고 디바이스 섀도 업데이트를 구독하는 샘플 애플리케이션을 다운로드하여 실행하십시오. 클라이언트 디바이스에서 다음을 수행하십시오.

   a. Python용 AWS IoT Device SDK v2의 샘플 폴더로 변경합니다. 이 샘플 애플리케이션은 samples 폴더의 명령줄 파싱 모듈을 사용합니다.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

   b. 텍스트 편집기를 사용하여 다음 basic_discovery_shadow.py 내용으로 이름이 지정된 Python 스크립트를 작성하십시오. 이 애플리케이션은 Greengrass 검색 및 섀도를 사용하여 클라이언트 디바이스와 코어 디바이스 간에 속성을 동기화된 상태로 유지합니다.

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   ```
   nano basic_discovery_shadow.py
   ```

   다음 Python 코드를 파일에 복사합니다.

   ```
   # Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   # SPDX-License-Identifier: Apache-2.0.
   ```

```
from awscrt import io
from awscrt import mqtt
from awsiot import iotshadow
from awsiot.greengrass_discovery import DiscoveryClient
from awsiot import mqtt_connection_builder
from concurrent.futures import Future
import sys
import threading
import traceback
from uuid import uuid4

# Parse arguments
import utils.command_line_utils;
cmdUtils = utils.command_line_utils.CommandLineUtils("Basic Discovery -
 Greengrass discovery example with device shadows.")
cmdUtils.add_common_mqtt_commands()
cmdUtils.add_common_topic_message_commands()
cmdUtils.add_common_logging_commands()
cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.",
 True, str)
cmdUtils.register_command("cert", "<path>", "Path to your client certificate in
 PEM format.", True, str)
cmdUtils.remove_command("endpoint")
cmdUtils.register_command("thing_name", "<str>", "The name assigned to your IoT
 Thing", required=True)
cmdUtils.register_command("region", "<str>", "The region to connect through.",
 required=True)
cmdUtils.register_command("shadow_property", "<str>", "The name of the shadow
 property you want to change (optional, default='color'", default="color")
# Needs to be called so the command utils parse the commands
cmdUtils.get_args()

# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = cmdUtils.get_command_required("thing_name")
shadow_property = cmdUtils.get_command("shadow_property")

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
```

```
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()


def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=cmdUtils.get_command_required("cert"),
                        pri_key_filepath=cmdUtils.get_command_required("key"),

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=cmdUtils.get_command_required("thing_name"),
                        clean_session=False,
                        keep_alive_secs=30)

                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection

                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
```

```
                    continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
                return

        if response.state:
```

```
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
                    change_shadow_value(value)
                    return

            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)
```

```
def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
```

```
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
                else:
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value

def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return

        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value
```

```
        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
None
        # for both reported and desired to clear the shadow document
completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
                client_token=token,
            )

        future = shadow_client.publish_update_shadow(request,
mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)
```

```
if __name__ == '__main__':
    tls_options =
 io.TlsContextOptions.create_client_with_mtls_from_path(cmdUtils.get_command_required("
 cmdUtils.get_command_required("key"))
    if cmdUtils.get_command(cmdUtils.m_cmd_ca_file):
        tls_options.override_default_trust_store_from_path(None,
 cmdUtils.get_command(cmdUtils.m_cmd_ca_file))
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
 DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
 socket_options, tls_context, cmdUtils.get_command_required("region"))
    resp_future =
 discovery_client.discover(cmdUtils.get_command_required("thing_name"))
    discover_response = resp_future.result()

    print(discover_response)
    if cmdUtils.get_command("print_discover_resp_only"):
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.
        # Note that is **is** important to wait for "accepted/rejected"
 subscriptions
        # to succeed before publishing the corresponding "request".
        print("Subscribing to Update responses...")
        update_accepted_subscribed_future, _ =
 shadow_client.subscribe_to_update_shadow_accepted(

 request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_accepted)

        update_rejected_subscribed_future, _ =
 shadow_client.subscribe_to_update_shadow_rejected(
```

```
request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_rejected)

        # Wait for subscriptions to succeed
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")
```

```
        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

이 Python 응용 프로그램은 다음을 수행합니다.

- Greengrass 디스커버리를 사용하여 코어 디바이스를 검색하고 연결합니다.
- 코어 디바이스에 섀도우 문서를 요청하여 속성의 초기 상태를 가져옵니다.
- 섀도우 델타 이벤트를 구독합니다. 섀도우 델타 이벤트는 속성 `desired` 값이 해당 값과 다를 때 코어 디바이스가 전송합니다`reported`. 애플리케이션은 섀도우 델타 이벤트를 수신하면 속성 값을 변경하고 코어 디바이스에 업데이트를 전송하여 새 값을 해당 `reported` 값으로 설정합니다.

이 애플리케이션은 Greengrass 디스커버리와 v2의 섀도우 샘플을 결합합니다. AWS IoT Device SDK

c. 샘플 애플리케이션을 실행합니다. 이 응용 프로그램에는 클라이언트 장치 사물 이름, 사용할 섀도우 속성, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다.

- *MyClientDevice1#* 클라이언트 디바이스의 사물 이름으로 바꾸십시오.

- *~/certs/ AmazonRoot ca1.pem# ##### ##### Amazon ## CA ### ###* 바꾸십시오.
- *~/certs/device.pem.crt#* 클라이언트 디바이스의 디바이스 인증서 경로로 바꾸십시오.
- *~/certs/private.pem.key#* 클라이언트 장치의 개인 키 파일 경로로 바꾸십시오.
- *us-east-1#* 클라이언트 디바이스 및 코어 디바이스가 운영되는 지역으로 바꾸십시오. AWS

```
python3 basic_discovery_shadow.py \
  --thing_name MyClientDevice1 \
  --shadow_property color \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbosity Warn
```

샘플 애플리케이션은 섀도우 토픽을 구독하고 코어 디바이스로부터 섀도우 델타 이벤트를 수신할 때까지 기다립니다. 애플리케이션이 섀도 델타 이벤트를 수신하고 이에 응답한다는 결과가 출력에 표시되면 클라이언트 디바이스는 코어 디바이스의 섀도우와 성공적으로 상호 작용할 수 있습니다.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGG
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
Subscribing to Delta events...
Requesting current shadow state...
Received shadow delta event.
```

```
  Delta reports that desired value is 'purple'. Changing local value...
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

애플리케이션이 대신 오류를 출력하는 경우 Greengrass 검색 문제 해결을 참조하십시오.

또한 코어 디바이스의 Greengrass 로그를 확인하여 클라이언트 디바이스가 성공적으로 연결하고 메시지를 전송하는지 확인할 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

5. 구성 요소 로그를 다시 보고 구성 요소가 스마트 조명 클라이언트 장치로부터 섀도우 업데이트 확인을 수신하는지 확인하십시오.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MySmartLightManager.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
 10 -Wait
```

구성 요소는 메시지를 기록하여 스마트 조명 클라이언트 장치의 색상이 변경되었음을 확인합니다.

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.959Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Received
 shadow update confirmation from client device: MyClientDevice1.
```

```
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

> **Note**
>
> 클라이언트 디바이스의 섀도우는 코어 디바이스와 클라이언트 디바이스 간에 동기화됩니다. 하지만 코어 디바이스는 클라이언트 디바이스의 섀도우를 동기화하지 않습니다AWS IoT Core. 예를 들어 AWS IoT Core 섀도우를 동기화하여 플릿에 있는 모든 디바이스의 상태를 보거나 수정할 수 있습니다. 섀도우를 동기화하도록 섀도우 관리자 컴포넌트를 구성하는 방법에 대한 자세한 내용은 을 AWS IoT Core 참조하십시오로컬 장치 섀도우를 다음과 동기화합니다. AWS IoT Core.

이 튜토리얼을 완료했습니다. 클라이언트 디바이스는 코어 디바이스에 연결하고, Greengrass 컴포넌트에 MQTT 메시지를 AWS IoT Core 전송하고, 코어 디바이스로부터 섀도우 업데이트를 수신합니다. 이 자습서에서 다루는 주제에 대한 자세한 내용은 다음을 참조하십시오.

- 클라이언트 장치 연결
- 코어 디바이스 엔드포인트 관리
- 클라이언트 장치 통신 테스트
- 그린그래스 디스커버리 RESTful API
- 클라이언트 장치 간에 MQTT 메시지를 중계하고 AWS IoT Core
- 구성 요소에서 클라이언트 장치와 상호 작용
- 디바이스 섀도우와 상호작용
- 클라이언트 디바이스 섀도우와 상호 작용 및 동기화

튜토리얼: SageMaker 엣지 매니저 시작하기

> **Important**
>
> SageMaker 엣지 매니저는 2024년 4월 26일에 단종됩니다. 에지 장치에 모델을 계속 배포하는 방법에 대한 자세한 내용은 SageMaker Edge Manager 수명 종료를 참조하십시오.

Amazon SageMaker Edge Manager는 엣지 디바이스에서 실행되는 소프트웨어 에이전트입니다. SageMaker Edge Manager는 엣지 디바이스에 대한 모델 관리를 제공하므로 Amazon SageMaker NEO로 컴파일된 모델을 Greengrass 코어 디바이스에서 직접 패키징하고 사용할 수 있습니다. SageMaker Edge Manager를 사용하면 코어 디바이스에서 모델 입력 및 출력 데이터를 샘플링하고 모니터링 및 분석을 AWS 클라우드 위해 해당 데이터를 로 전송할 수도 있습니다. Greengrass 코어 디바이스에서 SageMaker Edge Manager가 작동하는 방식에 대한 자세한 내용은 을 참조하십시오. 그린그래스 코어 디바이스에서 Amazon SageMaker Edge Manager 사용

이 자습서에서는 기존 코어 장치에서 AWS 제공된 샘플 구성 요소와 함께 SageMaker Edge Manager를 사용하여 시작하는 방법을 보여줍니다. 이러한 샘플 구성 요소는 SageMaker Edge Manager 구성 요소를 종속 항목으로 사용하여 Edge Manager 에이전트를 배포하고 Neo를 사용하여 컴파일된 사전 학습된 모델을 사용하여 추론을 수행합니다. SageMaker SageMaker Edge Manager 에이전트에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 SageMaker Edge Manager를 참조하십시오.

기존 Greengrass 코어 디바이스에서 SageMaker Edge Manager 에이전트를 설정하고 사용하기 위해 다음과 같은 샘플 추론 및 모델 구성 요소를 생성하는 데 사용할 수 있는 예제 코드를 AWS 제공합니다.

- 이미지 분류
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- 물체 감지
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

이 자습서에서는 샘플 구성 요소와 SageMaker Edge Manager 에이전트를 배포하는 방법을 보여줍니다.

주제

사전 조건

이 자습서를 완료하려면 다음 사전 요구 사항을 충족해야 합니다.

- 아마존 리눅스 2, 데비안 기반 리눅스 플랫폼 (x86_64 또는 Armv8) 또는 윈도우 (x86_64) 에서 실행되는 그린그래스 코어 디바이스. 계정이 없는 경우 자습서: AWS IoT Greengrass V2 시작하기 단원을 참조하십시오.
- Python 3.6 이상 (사용 중인 Python `pip` 버전용 포함) 이 코어 기기에 설치되어 있어야 합니다.
- 코어 디바이스에 설치된 OpenGL API GLX 런타임 `libgl1-mesa-glx` ().
- 관리자 권한이 있는 AWS Identity and Access Management (IAM) 사용자.
- 다음 요구 사항을 충족하는 인터넷 지원 Windows, Mac 또는 Unix 계열 개발 컴퓨터:
  - Python 3.6 이상이 설치되었습니다.
  - AWS CLIIAM 관리자 사용자 자격 증명으로 설치 및 구성되었습니다. 자세한 내용은 설치 AWS CLI 및 구성을 참조하십시오. AWS CLI
- Greengrass 코어 AWS 계정 디바이스와 동일한 디바이스에서 생성된 S3 버킷은 다음과 AWS 리전 같습니다.
  - 샘플 추론 및 모델 구성 요소에 포함된 아티팩트를 저장하는 S3 버킷입니다. 이 자습서에서는 *DOC-EXAMPLE-BUCKET1* 을 사용하여 이 버킷을 참조합니다.
  - SageMaker 에지 디바이스 플릿과 연결하는 S3 버킷. SageMaker Edge Manager에는 에지 디바이스 플릿을 생성하고 디바이스에서 추론을 실행하여 얻은 샘플 데이터를 저장하려면 S3 버킷이 필요합니다. 이 자습서에서는 *DOC-EXAMPLE-BUCKET2* 을 사용하여 이 버킷을 참조합니다.

  S3 버킷 생성에 대한 자세한 내용은 Amazon S3 시작하기를 참조하십시오.
- Greengrass 장치 역할은 다음과 같이 구성되었습니다.
  - 다음 IAM 정책 예제와 같이 역할을 `credentials.iot.amazonaws.com` 허용하고 `sagemaker.amazonaws.com` 역할을 맡을 수 있는 신뢰 관계입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- AmazonSageMakerEdgeDeviceFleetPolicyIAM 관리형 정책.
- AmazonSageMakerFullAccessIAM 관리형 정책.
- 구성 요소 아티팩트가 포함된 S3 버킷에 대한 `s3:GetObject` 작업 (다음 IAM 정책 예시 참조).

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
    ],
    "Effect": "Allow"
  }
  ]
}
```

엣지 매니저에서 SageMaker Greengrass 코어 디바이스를 설정합니다.

SageMaker Edge Manager의 에지 디바이스 플릿은 논리적으로 그룹화된 디바이스 모음입니다. 에서 SageMaker Edge Manager를 사용하려면 Edge Manager 에이전트를 SageMaker 배포하는 Greengrass 코어 장치와 동일한 AWS IoT 역할 별칭을 사용하는 에지 장치 집합을 만들어야 합니다. AWS IoT Greengrass 그런 다음 코어 디바이스를 해당 플릿의 일부로 등록해야 합니다.

주제

- 에지 디바이스 플릿 생성
- 그린그래스 코어 디바이스 등록

에지 디바이스 플릿 생성

에지 디바이스 플릿을 만들려면 (콘솔)

1. Amazon SageMaker 콘솔에서 Edge Manager를 선택한 다음 Edge 디바이스 플릿을 선택합니다.
2. 디바이스 플릿 페이지에서 디바이스 플릿 생성을 선택합니다.
3. 디바이스 플릿 속성에서 다음을 수행하십시오.
   - 디바이스 플릿 이름에 디바이스 플릿의 이름을 입력합니다.
   - IAM 역할의 경우 Greengrass 코어 디바이스를 설정할 때 지정한 역할 별칭의 AWS IoT Amazon 리소스 이름 (ARN) 을 입력합니다.
   - IAM 역할 별칭 생성 토글을 비활성화합니다.
4. 다음을 선택합니다.
5. 출력 구성에서 S3 버킷 URI에 디바이스 플릿과 연결할 S3 버킷의 URI를 입력합니다.
6. 제출을 선택합니다.

그린그래스 코어 디바이스 등록

Greengrass 코어 디바이스를 엣지 디바이스 (콘솔) 로 등록하려면

1. Amazon SageMaker 콘솔에서 Edge Manager를 선택한 다음 Edge 디바이스를 선택합니다.
2. 디바이스 페이지에서 디바이스 등록을 선택합니다.
3. 디바이스 속성에서 디바이스 플릿 이름에 생성한 디바이스 플릿의 이름을 입력하고 다음을 선택합니다.
4. 다음을 선택합니다.
5. 디바이스 소스에서 디바이스 이름에 Greengrass 코어 디바이스의 AWS IoT 사물 이름을 입력합니다.
6. 제출을 선택합니다.

샘플 구성 요소를 생성합니다.

SageMaker Edge Manager 구성 요소 사용을 시작하는 데 도움이 되도록 샘플 추론 및 모델 구성 요소를 만들고 이를 자동으로 업로드하는 Python 스크립트를 AWS 제공합니다. GitHub AWS 클라우드 개발 컴퓨터에서 다음 단계를 완료하세요.

샘플 구성 요소를 만들려면

1. AWS IoT Greengrass구성 요소 예제 저장소를 개발 컴퓨터에 다운로드하십시오. GitHub
2. 다운로드한 `/machine-learning/sagemaker-edge-manager` 폴더로 이동합니다.

   ```
   cd download-directory/machine-learning/sagemaker-edge-manager
   ```

3. 다음 명령을 실행하여 샘플 구성 요소를 만들고 에 AWS 클라우드 업로드합니다.

   ```
   python3 create_components.py -r region -b DOC-EXAMPLE-BUCKET
   ```

   ### Greengrass 코어 디바이스를 생성한 AWS 리전 곳으로 바꾸고, *DOC-EXAMPLE-BUCKET1* 이름을 S3 버킷 이름으로 바꾸어 구성 요소 아티팩트를 저장합니다.

   > **Note**
   >
   > 기본적으로 스크립트는 이미지 분류와 객체 감지 추론을 위한 샘플 구성 요소를 생성합니다. 특정 유형의 추론에만 사용할 구성 요소를 만들려면 인수를 지정하십시오. `-i ImageClassification | ObjectDetection`

이제 SageMaker Edge Manager와 함께 사용할 샘플 추론 및 모델 구성 요소가 에서 생성되었습니다. AWS 계정 AWS IoT Greengrass콘솔에서 샘플 구성 요소를 보려면 구성 요소를 선택한 다음 내 구성 요소에서 다음 구성 요소를 검색하십시오.

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

샘플 이미지 분류 추론 실행

AWS제공된 샘플 구성 요소와 SageMaker Edge Manager 에이전트를 사용하여 이미지 분류 추론을 실행하려면 이러한 구성 요소를 코어 장치에 배포해야 합니다. 이러한 구성 요소를 배포하면 SageMaker NEO로 컴파일된 사전 학습된 Resnet-50 모델이 다운로드되고 장치에 Edge Manager 에이전트가 설치됩니다. SageMaker SageMaker Edge Manager 에이전트는 모델을 로드하고 주제에 대한 추론 결과

를 게시합니다. gg/sageMakerEdgeManager/image-classification 이러한 추론 결과를 보려면 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트를 사용하여 이 주제를 구독하십시오.

주제

- 알림 주제를 구독하세요.
- 샘플 구성 요소 배포
- 추론 결과 보기

알림 주제를 구독하세요.

이 단계에서는 샘플 추론 구성 요소가 게시한 AWS IoT MQTT 메시지를 감시하도록 AWS IoT 콘솔에서 MQTT 클라이언트를 구성합니다. 기본적으로 구성 요소는 주제에 대한 추론 결과를 게시합니다. gg/sageMakerEdgeManager/image-classification Greengrass 코어 디바이스에 구성 요소를 배포하기 전에 이 항목을 구독하여 구성 요소가 처음 실행될 때의 추론 결과를 확인하십시오.

기본 알림 주제를 구독하려면

1. AWS IoT콘솔 탐색 메뉴에서 Test, MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독의 주제 이름 상자에 를 입력합니다**gg/sageMakerEdgeManager/image-classification**.
3. 구독을 선택합니다.

샘플 구성 요소 배포

이 단계에서는 다음 구성 요소를 구성하고 코어 장치에 배포합니다.

- aws.greengrass.SageMakerEdgeManager
- com.greengrass.SageMakerEdgeManager.ImageClassification
- com.greengrass.SageMakerEdgeManager.ImageClassification.Model

구성 요소를 배포하려면(콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 [Deployments] 를 선택한 다음 수정하려는 대상 장치의 배포를 선택합니다.
2. 배포 페이지에서 수정을 선택한 다음 배포 개정을 선택합니다.

3. 대상 지정 페이지에서 다음을 선택합니다.
4. 구성 요소 선택 페이지에서 다음을 수행합니다.
   a. 내 구성 요소에서 다음 구성 요소를 선택합니다.
      - `com.greengrass.SageMakerEdgeManager.ImageClassification`
      - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
   b. 공용 구성 요소에서 선택한 구성 요소만 표시 토글을 끄고 `aws.greengrass.SageMakerEdgeManager` 구성 요소를 선택합니다.
   c. 다음을 선택합니다.
5. 구성 요소 구성 페이지에서 구성 `aws.greengrass.SageMakerEdgeManager` 요소를 선택하고 다음을 수행합니다.
   a. 구성 요소 구성을 선택합니다.
   b. 구성 업데이트 아래에 있는 병합할 구성에 다음 구성을 입력합니다.

      ```
      {
          "DeviceFleetName": "device-fleet-name",
          "BucketName": "DOC-EXAMPLE-BUCKET"
      }
      ```

      생성한 에지 디바이스 플릿의 이름으로 바꾸고 *device-fleet-name*, *DOC-EXAMPLE-BUCKET# #### ###* 연결된 S3 버킷의 이름으로 바꾸십시오.
   c. 확인을 선택하고 다음을 선택합니다.
6. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
7. 검토 페이지에서 [Deploy] 를 선택합니다.

구성 요소를 배포하려면(AWS CLI)

1. 개발 컴퓨터에서 SageMaker Edge Manager 구성 요소의 배포 구성을 정의하는 `deployment.json` 파일을 만듭니다. 이 파일은 다음 예제와 비슷합니다.

   ```
   {
     "targetArn":"targetArn",
     "components": {
       "aws.greengrass.SageMakerEdgeManager": {
         "componentVersion": "1.0.x",
   ```

```
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\":\"device-fleet-name\",\"BucketName\":\"DOC-
EXAMPLE-BUCKET2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
  }
}
```

- targetArn 필드에서 *targetArn*을(를) 다음 형식으로 배포 대상으로 지정할 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)으로 바꿉니다.
  - 사물: arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 사물 그룹: arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- merge필드에서 생성한 에지 디바이스 플릿의 이름으로 *device-fleet-name*바꿉니다. 그런 다음 *DOC-EXAMPLE-BUCKET2* 이름을 디바이스 플릿과 연결된 S3 버킷의 이름으로 바꾸십시오.
- 각 구성 요소의 구성 요소 버전을 사용 가능한 최신 버전으로 바꿉니다.

2. 다음 명령을 실행하여 디바이스에 구성 요소를 배포합니다.

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. 다음 단계에서는 구성 요소 로그를 확인하여 배포가 정상적으로 완료되었는지 확인하고 추론 결과를 확인합니다.

추론 결과 보기

구성 요소를 배포한 후에는 Greengrass 코어 디바이스의 구성 요소 로그와 콘솔의 MQTT 클라이언트에서 추론 결과를 볼 수 있습니다. AWS IoT AWS IoT 구성 요소가 추론 결과를 게시하는 주제를 구독하려면 을 참조하십시오. 알림 주제를 구독하세요.

- AWS IoTMQTT 클라이언트 - 추론 구성 요소가 기본 알림 주제에 게시하는 결과를 보려면 다음 단계를 완료하십시오.

  1. AWS IoT콘솔 탐색 메뉴에서 Test, MQTT 테스트 클라이언트를 선택합니다.
  2. 구독에서 선택합니다. **gg/sageMakerEdgeManager/image-classification**

- 구성 요소 로그 - 구성 요소 로그에서 추론 결과를 보려면 Greengrass 코어 장치에서 다음 명령을 실행합니다.

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

구성 요소 로그나 MQTT 클라이언트에서 추론 결과를 볼 수 없는 경우 배포가 실패했거나 코어 디바이스에 도달하지 못한 것입니다. 이는 코어 기기가 인터넷에 연결되어 있지 않거나 구성 요소를 실행할 수 있는 적절한 권한이 없는 경우 발생할 수 있습니다. 코어 장치에서 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

자세한 내용은 머신 러닝 추론 문제 해결을(를) 참조하세요.

자습서: Lite를 사용하여 TensorFlow 샘플 이미지 분류 추론 수행

이 자습서에서는 TensorFlow Lite 이미지 분류 추론 구성 요소를 사용하여 Greengrass 코어 기기에서 샘플 이미지 분류 추론을 수행하는 방법을 보여줍니다. 이 구성 요소에는 다음과 같은 구성 요소 종속성이 포함됩니다.

- TensorFlow 라이트 이미지 분류 모델 스토어 컴포넌트

- TensorFlow 라이트 런타임 컴포넌트

이 구성 요소를 배포하면 사전 학습된 MobileNet v1 모델을 다운로드하고 TensorFlow Lite 런타임과 해당 종속 항목을 설치합니다. 이 구성 요소는 주제에 대한 추론 결과를 게시합니다. `ml/tflite/image-classification` 이러한 추론 결과를 보려면 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트를 사용하여 이 주제를 구독하십시오.

이 자습서에서는 샘플 추론 구성 요소를 배포하여 에서 제공하는 샘플 이미지에 대해 이미지 분류를 수행합니다. AWS IoT Greengrass 이 튜토리얼을 완료한 후에는 Greengrass 코어 디바이스의 로컬 카메라 이미지에 대해 이미지 분류를 수행하도록 샘플 추론 구성 요소를 수정하는 방법을 보여주는 튜토리얼을 튜토리얼: Lite를 사용하여 TensorFlow 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행 완료할 수 있습니다.

Greengrass 기기에서의 기계 학습에 대한 자세한 내용은 을 참조하십시오. 기계 학습 추론 수행

주제

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- 리눅스 그린그래스 코어 디바이스. 계정이 없는 경우 자습서: AWS IoT Greengrass V2 시작하기 단원을 참조하십시오. 코어 디바이스는 다음 요구 사항을 충족해야 합니다.
  - Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
  - 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

    ```
    sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
     libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
    ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

1단계: 기본 알림 주제 구독

이 단계에서는 TensorFlow Lite 이미지 분류 구성요소가 게시한 AWS IoT MQTT 메시지를 감시하도록 AWS IoT 콘솔에서 MQTT 클라이언트를 구성합니다. 기본적으로 구성 요소는 주제에 대한 추론 결과를 게시합니다. `ml/tflite/image-classification` Greengrass 코어 디바이스에 구성 요소를 배포하기 전에 이 항목을 구독하여 구성 요소가 처음 실행될 때의 추론 결과를 확인하십시오.

기본 알림 주제를 구독하려면

1. AWS IoT콘솔 탐색 메뉴에서 Test, MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독의 주제 이름 상자에 를 입력합니다**`ml/tflite/image-classification`**.
3. 구독을 선택합니다.

2단계: TensorFlow Lite 이미지 분류 구성 요소 배포

이 단계에서는 TensorFlow Lite 이미지 분류 구성요소를 코어 기기에 배포합니다.

TensorFlow Lite 이미지 분류 구성 요소를 배포하려면 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 퍼블릭 구성 요소 탭에서 `aws.greengrass.TensorFlowLiteImageClassification`을(를) 선택합니다.
3. aws.greengrass.TensorFlowLiteImageClassification 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 다음 중 하나를 선택합니다.
   a. 이 구성 요소를 대상 디바이스의 기존 배포에 병합하려면 기존 배포에 추가를 선택한 다음 수정하려는 배포를 선택합니다.
   b. 대상 디바이스에서 새 배포를 생성하려면 새 배포 생성을 선택합니다. 디바이스에 기존 배포가 있는 경우 이 단계를 선택하면 기존 배포가 대체됩니다.
5. 대상 지정 페이지에서 다음 작업을 수행합니다.
   a. 배포 정보 아래에서 친숙한 배포 이름을 입력하거나 수정합니다.
   b. 배포 대상 아래에서 배포 대상을 선택하고 다음을 선택합니다. 기존 배포 수정 시 배포 대상을 변경할 수 없습니다.
6. 구성 요소 선택 페이지의 공용 구성 요소에서 `aws.greengrass.TensorFlowLiteImageClassification` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 [Deploy] 를 선택합니다.

TensorFlow Lite 이미지 분류 구성 요소를 배포하려면 (AWS CLI)

1. `deployment.json`파일을 생성하여 TensorFlow Lite 이미지 분류 구성 요소의 배포 구성을 정의합니다. 이 파일은 다음과 같아야 합니다.

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
```

```
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```

- targetArn 필드에서 *targetArn*을(를) 다음 형식으로 배포 대상으로 지정할 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)으로 바꿉니다.
  - 사물: arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 사물 그룹: arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 이 자습서에서는 구성 요소 버전 2.1.0을 사용합니다. aws.greengrass.TensorFlowLiteObjectDetection구성 요소 개체에서 *2.1.0#* 다른 버전의 TensorFlow Lite 개체 감지 구성 요소를 사용하도록 바꾸십시오.

2. 다음 명령을 실행하여 TensorFlow Lite 이미지 분류 구성 요소를 기기에 배포합니다.

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. 다음 단계에서는 구성 요소 로그를 확인하여 배포가 정상적으로 완료되었는지 확인하고 추론 결과를 확인합니다.

3단계: 추론 결과 보기

구성 요소를 배포한 후에는 Greengrass 코어 디바이스의 구성 요소 로그와 콘솔의 MQTT 클라이언트에서 추론 결과를 볼 수 있습니다. AWS IoT AWS IoT 구성 요소가 추론 결과를 게시하는 주제를 구독하려면 을 참조하십시오. 1단계: 기본 알림 주제 구독

- AWS IoTMQTT 클라이언트 - 추론 구성 요소가 기본 알림 주제에 게시하는 결과를 보려면 다음 단계를 완료하십시오.

  1. AWS IoT콘솔 탐색 메뉴에서 Test, MQTT 테스트 클라이언트를 선택합니다.
  2. 구독에서 원하는 항목을 선택합니다. **ml/tflite/image-classification**

     다음 예와 비슷한 메시지가 표시되어야 합니다.

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
```

```
  "inference-type": "image-classification",
  "inference-description": "Top 5 predictions with score 0.3 or above ",
  "inference-results": [
    {
      "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "Persian cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "tiger cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "dalmatian, coach dog, carriage dog",
      "Score": "0.5607843137254902"
    },
    {
      "Label": "malamute, malemute, Alaskan malamute",
      "Score": "0.5450980392156862"
    }
  ]
}
```

- 구성 요소 로그 - 구성 요소 로그에서 추론 결과를 보려면 Greengrass 코어 장치에서 다음 명령을 실행합니다.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
  ```

  다음 예와 비슷한 결과가 표시될 것입니다.

  ```
  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
   IoT core....
   {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
   serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
   "2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-
  ```

```
description": "Top 5 predictions with score 0.3 or above ", "inference-results":
 [{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":
 "0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},
 {"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},
 {"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]}.
 {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
 serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
```

구성 요소 로그나 MQTT 클라이언트에서 추론 결과를 볼 수 없다면 배포가 실패했거나 코어 디바이스에 도달하지 못한 것입니다. 이는 코어 기기가 인터넷에 연결되어 있지 않거나 구성 요소를 실행할 수 있는 적절한 권한이 없는 경우 발생할 수 있습니다. 코어 장치에서 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

자세한 설명은 머신 러닝 추론 문제 해결 섹션을 참조하세요.

다음 단계

카메라 인터페이스가 지원되는 Greengrass 코어 디바이스가 있는 경우 카메라 이미지에 대한 이미지 분류를 수행하도록 샘플 추론 구성 요소를 수정하는 방법을 보여주는 작업을 튜토리얼: Lite를 사용하여 TensorFlow 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행 완료할 수 있습니다.

샘플 TensorFlow Lite 이미지 분류 추론 구성 요소의 구성을 더 자세히 알아보려면 다음을 시도해 보십시오.

- `InferenceInterval`구성 파라미터를 수정하여 추론 코드의 실행 빈도를 변경하십시오.
- 추론 `ImageDirectory` 구성 요소 구성의 `ImageName` 및 구성 매개 변수를 수정하여 추론에 사용할 사용자 지정 이미지를 지정합니다.

공용 구성 요소의 구성을 사용자 지정하거나 사용자 지정 기계 학습 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오. 머신 러닝 구성 요소 사용자 지정

튜토리얼: Lite를 사용하여 TensorFlow 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행

이 자습서에서는 TensorFlow Lite 이미지 분류 추론 구성 요소를 사용하여 Greengrass 코어 장치의 로컬 카메라 이미지에 대한 샘플 이미지 분류 추론을 수행하는 방법을 보여줍니다. 이 구성 요소에는 다음과 같은 구성 요소 종속성이 포함됩니다.

- TensorFlow 라이트 이미지 분류 모델 스토어 컴포넌트
- TensorFlow 라이트 런타임 컴포넌트

> **Note**
>
> 이 가이드에서는 라즈베리 파이 또는 NVIDIA Jetson Nano 디바이스용 카메라 모듈에 액세스하지만 ARMv7L, Armv8 또는 AWS IoT Greengrass x86_64 플랫폼의 다른 디바이스도 지원합니다. 다른 기기용으로 카메라를 설정하려면 해당 장치의 관련 설명서를 참조하십시오.

Greengrass 기기에서의 기계 학습에 대한 자세한 내용은 을 참조하십시오. 기계 학습 추론 수행

주제

사전 조건

이 자습서를 완료하려면 먼저 자습서: Lite를 사용하여 TensorFlow 샘플 이미지 분류 추론 수행 완료해야 합니다.

또한 다음 항목이 필요합니다.

- 카메라 인터페이스가 있는 Linux Greengrass 코어 디바이스. 이 가이드에서는 지원되는 다음 장치 중 하나의 카메라 모듈에 액세스합니다.

- 라즈베리파이 OS (이전에는 라즈비안이라고 함) 를 실행하는 라즈베리파이
- 엔비디아 젯슨 나노

Greengrass 코어 장치 설정에 대한 자세한 내용은 을 참조하십시오. 자습서: AWS IoT Greengrass V2 시작하기

코어 디바이스는 다음 요구 사항을 충족해야 합니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

- 라즈베리 파이 또는 엔비디아 젯슨 나노 디바이스의 경우 라즈베리 파이 카메라 모듈 V2 - 8메가픽셀, 1080p. 카메라 설정 방법을 알아보려면 Raspberry Pi 설명서의 Connecting the camera를 참조하십시오.

1단계: 디바이스의 카메라 모듈 구성

이 단계에서는 장치의 카메라 모듈을 설치하고 활성화합니다. 장치에서 다음 명령을 실행합니다.

Raspberry Pi (Armv7l)

1. 카메라 모듈용 picamera 인터페이스를 설치합니다. 다음 명령을 실행하여 이 자습서에 필요한 카메라 모듈과 기타 Python 라이브러리를 설치합니다.

   ```
   sudo apt-get install -y python3-picamera
   ```

2. Picamera가 성공적으로 설치되었는지 확인하세요.

   ```
   sudo -u ggc_user bash -c 'python3 -c "import picamera"'
   ```

   출력에 오류가 포함되어 있지 않으면 검사가 성공한 것입니다.

   > **Note**
   >
   > 기기에 설치된 Python 실행 파일이 다음과 python3.7 같은 경우 이 자습서의 명령에 python3.7 대신 python3 를 사용하십시오. 종속성 오류를 피하기 위해 pip 설치가 올바른 python3.7 또는 python3 버전에 매핑되는지 확인하십시오.

3. 디바이스를 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 명령어를 실행하여 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.

7. 다음 명령을 실행하여 런타임 구성요소가 만든 가상 환경에서 추론 구성요소가 카메라에 액세스할 수 있도록 하는 심볼릭 링크를 생성합니다.

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

이 자습서의 *ML RootPath* 기본값은 입니다. */greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml 이 위치의 greengrass_ml_tflite_venv 폴더는 에서 자습서: Lite를 사용하여 TensorFlow 샘플 이미지 분류 추론 수행 처음으로 추론 구성 요소를 배포할 때 생성됩니다.

Jetson Nano (Armv8)

1. 다음 명령을 실행하여 카메라 설정을 테스트합니다.

```
gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),
 width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink
 location=test.jpg
```

이렇게 하면 현재 `test.jpg` 디렉터리에 이름이 지정된 이미지가 캡처되고 저장됩니다.

2. (선택 사항) 장치를 재부팅합니다. 이전 단계에서 `gst-launch` 명령을 실행할 때 문제가 발생하는 경우 기기를 재부팅하면 문제가 해결될 수 있습니다.

```
sudo reboot
```

> **Note**
>
> Jetson Nano와 같은 Armv8 (AArch64) 기기의 경우 런타임 구성요소로 만든 가상 환경에서 추론 구성요소가 카메라에 액세스할 수 있도록 심볼릭 링크를 만들지 않아도 됩니다.

2단계: 기본 알림 주제 구독 확인

에서 자습서: Lite를 사용하여 TensorFlow 샘플 이미지 분류 추론 수행 AWS IoT MQTT 클라이언트가 해당 주제에 대해 TensorFlow Lite 이미지 분류 구성 요소가 게시한 MQTT 메시지를 감시하도록 AWS IoT 콘솔에서 구성하도록 구성했습니다. `ml/tflite/image-classification` AWS IoT콘솔에서 이 구독이 존재하는지 확인하십시오. 그렇지 않은 경우 Greengrass 코어 1단계: 기본 알림 주제 구독 디바이스에 구성 요소를 배포하기 전에 다음 단계에 따라 이 주제를 구독하십시오.

3단계: TensorFlow Lite 이미지 분류 구성 요소 구성을 수정하고 배포합니다.

이 단계에서는 TensorFlow Lite 이미지 분류 구성 요소를 구성하고 코어 장치에 배포합니다.

TensorFlow Lite 이미지 분류 구성 요소 구성 및 배포하기 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 퍼블릭 구성 요소 탭에서 `aws.greengrass.TensorFlowLiteImageClassification`을(를) 선택합니다.
3. aws.greengrass.TensorFlowLiteImageClassification 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 다음 중 하나를 선택합니다.
   a. 이 구성 요소를 대상 디바이스의 기존 배포에 병합하려면 기존 배포에 추가를 선택한 다음 수정하려는 배포를 선택합니다.
   b. 대상 디바이스에서 새 배포를 생성하려면 새 배포 생성을 선택합니다. 디바이스에 기존 배포가 있는 경우 이 단계를 선택하면 기존 배포가 대체됩니다.
5. 대상 지정 페이지에서 다음 작업을 수행합니다.
   a. 배포 정보 아래에서 친숙한 배포 이름을 입력하거나 수정합니다.
   b. 배포 대상 아래에서 배포 대상을 선택하고 다음을 선택합니다. 기존 배포 수정 시 배포 대상을 변경할 수 없습니다.
6. 구성 요소 선택 페이지의 공용 구성 요소에서 `aws.greengrass.TensorFlowLiteImageClassification` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 다음을 수행하십시오.
   a. 추론 구성 요소를 선택하고 구성 요소 구성을 선택합니다.
   b. 구성 업데이트에서 병합할 구성 상자에 다음 구성 업데이트를 입력합니다.

```
{
  "InferenceInterval": "60",
  "UseCamera": "true"
}
```

이 구성 업데이트를 통해 구성 요소는 장치의 카메라 모듈에 액세스하여 카메라로 촬영한 이미지를 추론합니다. 추론 코드는 60초마다 실행됩니다.

c. 확인을 선택하고 다음을 선택합니다.

8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 [Deploy] 를 선택합니다.

TensorFlow Lite 이미지 분류 구성 요소를 구성 및 배포하려면 (AWS CLI)

1. `deployment.json`파일을 생성하여 TensorFlow Lite 이미지 분류 구성 요소의 배포 구성을 정의합니다. 이 파일은 다음과 같아야 합니다.

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
    }
  }
}
```

- `targetArn` 필드에서 *targetArn*을(를) 다음 형식으로 배포 대상으로 지정할 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)으로 바꿉니다.
  - 사물: `arn:aws:iot:`*region*`:`*account-id*`:thing/`*thingName*
  - 사물 그룹: `arn:aws:iot:`*region*`:`*account-id*`:thinggroup/`*thingGroupName*
- 이 자습서에서는 구성 요소 버전 2.1.0을 사용합니다. `aws.greengrass.TensorFlowLiteImageClassification`구성 요소 객체에서 *2.1.0#* 다른 버전의 TensorFlow Lite 이미지 분류 구성 요소를 사용하도록 바꾸십시오.

이번 구성 업데이트를 통해 구성 요소는 장치의 카메라 모듈에 액세스하여 카메라로 촬영한 이미지를 추론합니다. 추론 코드는 60초마다 실행됩니다. 다음 값을 바꾸십시오.

2. 다음 명령을 실행하여 TensorFlow Lite 이미지 분류 구성 요소를 기기에 배포합니다.

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. 다음 단계에서는 구성 요소 로그를 확인하여 배포가 정상적으로 완료되었는지 확인하고 추론 결과를 확인합니다.

4단계: 추론 결과 보기

구성 요소를 배포한 후에는 Greengrass 코어 디바이스의 구성 요소 로그와 콘솔의 MQTT 클라이언트에서 추론 결과를 볼 수 있습니다. AWS IoT AWS IoT 구성 요소가 추론 결과를 게시하는 주제를 구독하려면 을 참조하십시오. 2단계: 기본 알림 주제 구독 확인

- AWS IoTMQTT 클라이언트 - 추론 구성 요소가 기본 알림 주제에 게시하는 결과를 보려면 다음 단계를 완료하십시오.

  1. AWS IoT콘솔 탐색 메뉴에서 Test, MQTT 테스트 클라이언트를 선택합니다.
  2. 구독에서 원하는 항목을 선택합니다. **ml/tflite/image-classification**

- 구성 요소 로그 - 구성 요소 로그에서 추론 결과를 보려면 Greengrass 코어 장치에서 다음 명령을 실행합니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

구성 요소 로그나 MQTT 클라이언트에서 추론 결과를 볼 수 없는 경우 배포가 실패했거나 코어 디바이스에 도달하지 못한 것입니다. 이는 코어 기기가 인터넷에 연결되어 있지 않거나 구성 요소를 실행하는 데 필요한 권한이 없는 경우 발생할 수 있습니다. 코어 장치에서 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 기기 배포 서비스의 로그가 포함되어 있습니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

자세한 설명은 머신 러닝 추론 문제 해결 섹션을 참조하세요.

다음 단계

이 자습서에서는 카메라로 촬영한 이미지에 대한 샘플 이미지 분류를 수행하는 사용자 지정 구성 옵션과 함께 TensorFlow Lite 이미지 분류 구성 요소를 사용하는 방법을 보여줍니다.

공용 구성 요소의 구성을 사용자 지정하거나 사용자 지정 기계 학습 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오머신 러닝 구성 요소 사용자 지정.

구성 요소

AWS IoT Greengrass구성 요소는 Greengrass 코어 장치에 배포하는 소프트웨어 모듈입니다. 구성 요소는 애플리케이션, 런타임 설치 프로그램, 라이브러리 또는 기기에서 실행하는 모든 코드를 나타낼 수 있습니다. 다른 구성 요소에 종속되는 구성 요소를 정의할 수 있습니다. 예를 들어 Python을 설치하는 구성 요소를 정의한 다음 해당 구성 요소를 Python 응용 프로그램을 실행하는 구성 요소의 종속성으로 정의할 수 있습니다. 장치 집합에 구성 요소를 배포할 때 Greengrass는 장치에 필요한 소프트웨어 모듈만 배포합니다.

주제

AWS-제공된 구성 요소

AWS IoT Greengrass 장치에 배포할 수 있는 사전 빌드된 구성 요소를 제공하고 유지 관리합니다. 이러한 구성 요소에는 기능 (예: 스트림 관리자), AWS IoT Greengrass V1 커넥터 (예: CloudWatch 메트릭) 및 로컬 개발 도구 (예: AWS IoT Greengrass CLI) 가 포함됩니다. 독립형 기능을 위해 이러한 구성 요소를 장치에 배포하거나 사용자 지정 Greengrass 구성 요소의 종속 항목으로 사용할 수 있습니다.

> **Note**
>
> AWS제공되는 몇 가지 구성 요소는 Greengrass 핵의 특정 마이너 버전에 따라 다릅니다. 이러한 종속성 때문에 Greengrass 핵을 새 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 의존하는 핵의 특정 버전에 대한 자세한 내용은 해당 구성 요소 항목을 참조하십시오. 핵 업데이트에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트

구성 요소의 구성 요소 유형이 제네릭과 Lambda인 경우 구성 요소의 현재 버전은 제네릭 유형이고 구성 요소의 이전 버전은 Lambda 유형입니다.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
그린그래스 핵	AWS IoT Greengrass Core 소프트웨어의 핵심. 이 구성 요소를 사용하여 코어 장치의 소프트웨어를 구성하고 업데이트할 수 있습니다.	Nucleus	Linux, Windows	예
클라이언트 장치 인증	클라이언트 디바이스라고 하는 로컬 IoT 디바이스를 코어 디바이스에 연결할 수 있도록 합니다.	플러그인	Linux, Windows	예
CloudWatch 측정 항목	CloudWatchAmazon에 사용자 지정 지표를 게시합니다.	일반, Lambda	Linux, Windows	예
AWS IoT Device Defender	관리자에게 Greengrass 코어 장치 상태의 변경 사항을 알려 비정상적인 동작을 식별합니다.	일반, Lambda	Linux, Windows	예
디스크 스풀러	Greengrass 코어 디바이스에	플러그인	Linux, Windows	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
	서 스풀링된 메시지에 대한 영구 저장 옵션을 활성화합니다. AWS IoT Core 이 구성 요소는 이러한 아웃바운드 메시지를 디스크에 저장합니다.			
Docker 애플리케이션 관리자	도커 허브 및 아마존 Elastic AWS IoT Greengrass Container 레지스트리 (Amazon ECR)에서 도커 이미지를 다운로드할 수 있습니다.	일반	Linux, Windows	아니요
Kinesis Video Streams용 에지 커넥터	로컬 카메라에서 비디오 피드를 읽고 Kinesis Video Streams에 스트림을 게시하고 Grafana 대시보드에 스트림을 표시합니다. AWS IoT TwinMaker	일반	Linux	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
그린그래스 CLI	로컬 배포를 생성하고 Greengrass 코어 장치 및 해당 구성 요소와 상호 작용하는 데 사용할 수 있는 명령줄 인터페이스를 제공합니다.	플러그인	Linux, Windows	예
IP 감지기	MQTT 브로커 연결 정보를 에 보고하여 AWS IoT Greengrass 클라이언트 장치가 연결 방법을 찾을 수 있도록 합니다.	플러그인	Linux, Windows	예
Firehose	Amazon Data Firehose 전송 스트림을 통해 내 목적지로 데이터를 게시합니다. AWS 클라우드	Lambda	Linux	아니요
람다 런처	Lambda 함수의 프로세스 및 환경 구성을 처리합니다.	제네릭	Linux	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
Lambda 관리자	Lambda 함수의 프로세스 간 통신 및 크기 조정을 처리합니다.	플러그인	Linux	아니요
Lambda 런타임	각 Lambda 런타임에 아티팩트를 제공합니다.	제네릭	Linux	아니요
레거시 서브스크립션 라우터	V1에서 실행되는 Lambda 함수의 구독을 관리합니다. AWS IoT Greengrass	제네릭	Linux	아니요
로컬 디버그 콘솔	Greengrass 코어 장치 및 해당 구성 요소를 디버깅 및 관리하는 데 사용할 수 있는 로컬 콘솔을 제공합니다.	플러그인	Linux, Windows	예
로그 매니저	Greengrass 코어 디바이스에서 로그를 수집하고 업로드합니다.	플러그인	Linux, Windows	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
기계 학습 구성 요소	Greengrass 코어 기기에서 기계 학습 추론을 수행하는 데 사용할 수 있는 기계 학습 모델 및 샘플 추론 코드를 제공합니다.	기계 학습 구성 요소을 참조하십시오.		
모드버스-RTU 프로토콜 어댑터	로컬 Modbus RTU 장치에서 정보를 폴링합니다.	Lambda	Linux	아니요
뉴클리어스 텔레메트리 이미터	핵에서 수집한 시스템 상태 원격 측정 데이터를 로컬 주제 또는 MQTT 주제에 게시합니다. AWS IoT Core	플러그인	Linux, Windows	예
MQTT 브리지	클라이언트 장치, 로컬 게시/구독 및 간에 MQTT 메시지를 릴레이합니다. AWS IoT Greengrass AWS IoT Core	플러그인	Linux, Windows	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
MQTT 3.1.1 브로커 (모켓)	클라이언트 디바이스와 코어 디바이스 간의 메시지를 처리하는 MQTT 3.1.1 브로커를 실행합니다.	플러그인	Linux, Windows	예
맷 5 브로커 (EMQX)	클라이언트 디바이스와 코어 디바이스 간의 메시지를 처리하는 MQTT 5 브로커를 실행합니다.	일반	Linux, Windows	아니요
PKCS #11 제공업체	Greengrass 구성요소가 하드웨어 보안 모듈 (HSM) 에 안전하게 저장한 개인 키와 인증서에 액세스할 수 있도록 합니다.	플러그인	Linux	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
시크릿 매니저	Greengrass 코어 디바이스의 사용자 지정 구성 요소에서 암호와 같은 자격 증명을 안전하게 사용할 수 있도록 비밀번호의 AWS Secrets Manager 비밀을 배포합니다.	플러그인	Linux, Windows	예
보안 터널링	제한된 방화벽 뒤에 있는 Greengrass 코어 디바이스와 양방향 통신을 설정하는 데 사용할 수 있는 AWS IoT 보안 터널링 연결을 활성화합니다.	일반	Linux	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
섀도우 매니저	코어 디바이스의 섀도우와 상호 작용할 수 있습니다. 섀도우 문서 스토리지를 관리하고 로컬 섀도우 상태를 AWS IoT Device Shadow 서비스와 동기화하는 작업도 관리합니다.	플러그인	Linux, Windows	예
Amazon SNS	Amazon SNS 주제에 메시지를 게시합니다.	Lambda	Linux	아니요
스트림 관리자	로컬 소스에서로 대용량 데이터를 스트리밍합니다. AWS 클라우드	일반	Linux, Windows	아니요
Systems Manager 에이전트	를 사용하여 코어 디바이스를 AWS Systems Manager 관리하면 디바이스를 패치하고 명령을 실행하는 등의 작업을 수행할 수 있습니다.	일반	Linux	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
토큰 교환 서비스	AWS 서비스와 상호 작용하는 데 사용할 수 있는 AWS 자격 증명을 제공합니다.	일반	Linux, Windows	아니요
IoT SiteWise OPC-UA 컬렉터	OPC-UA 서버에서 데이터를 수집합니다.	일반	Linux, Windows	아니요
IoT SiteWise OPC-UA 데이터 소스 시뮬레이터	샘플 데이터를 생성하는 로컬 OPC-UA 서버를 실행합니다.	일반	Linux, Windows	아니요
IoT SiteWise 퍼블리셔	데이터를 AWS 클라우드에 게시합니다.	일반	Linux, Windows	아니요
IoT SiteWise 프로세서	Greengrass 코어 디바이스에서 데이터를 처리합니다.	일반	Linux, Windows	아니요

그린그래스 핵

Greengrass nucleus 구성 요소 (`aws.greengrass.Nucleus`) 는 필수 구성 요소이며 장치에서 AWS IoT Greengrass Core 소프트웨어를 실행하기 위한 최소 요구 사항입니다. AWS IoT Greengrass Core 소프트웨어를 원격으로 사용자 지정하고 업데이트하도록 이 구성 요소를 구성할 수 있습니다. 이 구성 요소를 배포하여 코어 장치의 프록시, 장치 역할 및 AWS IoT 사물 구성과 같은 설정을 구성하십시오.

> **Important**
>
> nucleus 구성 요소의 버전이 변경되거나 특정 구성 매개변수를 변경하면 Nucleus 및 디바이스의 다른 모든 구성 요소를 포함하는 AWS IoT Greengrass Core 소프트웨어가 다시 시작되어 변경 사항을 적용합니다.
> 구성 요소를 배포하면 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
> 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass Core 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x

- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 장치에 설치할 수 있습니다.

- Linux
- Windows

자세한 설명은 지원하는 플랫폼 섹션을 참조하세요.

요구 사항

Greengrass nucleus 및 Core 소프트웨어를 설치하고 실행하려면 장치가 특정 요구 사항을 충족해야 합니다 AWS IoT Greengrass . 자세한 설명은 디바이스 요구 사항 섹션을 참조하세요.

Greengrass 핵 구성 요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.

- Greengrass 핵 구성 요소는 AWS IoT 자격 증명 및 Amazon S3에 AWS IoT data 연결되어야 합니다.

의존성

Greengrass 핵에는 구성 요소 종속성이 포함되어 있지 않습니다. 그러나 AWS 제공된 일부 구성 요소에는 핵이 종속성으로 포함됩니다. 자세한 설명은 AWS-제공된 구성 요소 섹션을 참조하세요.

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

다운로드 및 설치

Greengrass nucleus 구성 요소를 설정하는 설치 프로그램을 장치에 다운로드할 수 있습니다. 이 설치 프로그램은 장치를 Greengrass 코어 장치로 설정합니다. 수행할 수 있는 설치 유형에는 두 가지가 있습니다. 하나는 필요한 AWS 리소스를 생성하는 빠른 설치이고 다른 하나는 리소스를 직접 생성하는 수동 설치입니다. AWS 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어 설치 섹션을 참조하세요.

튜토리얼을 따라 Greengrass 핵을 설치하고 Greengrass 구성 요소 개발을 살펴볼 수도 있습니다. 자세한 설명은 자습서: AWS IoT Greengrass V2 시작하기 섹션을 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개변수를 제공합니다. 일부 매개 변수를 적용하려면 AWS IoT Greengrass Core 소프트웨어를 다시 시작해야 합니다. 이 구성 요소를 구성하는 이유와 방법에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass Core 소프트웨어 구성.

`iotRoleAlias`

토큰 교환 IAM 역할을 가리키는 AWS IoT 역할 별칭입니다. AWS IoT 자격 증명 공급자는 이 역할을 맡아 Greengrass 코어 디바이스가 서비스와 상호 작용할 AWS 수 있도록 합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

`--provision true`옵션과 함께 AWS IoT Greengrass Core 소프트웨어를 실행하면 소프트웨어가 역할 별칭을 프로비저닝하고 Nucleus 구성 요소에 해당 값을 설정합니다.

`interpolateComponentConfiguration`

(선택 사항) Greengrass nucleus를 사용하여 구성 요소 구성의 구성 요소 레시피 변수를 보간하고 구성 업데이트를 병합할 수 있습니다. 구성에서 레시피 변수를 사용하는 Greengrass 구성 요소를 코어 디바이스에서 실행할 수 `true` 있도록 이 옵션을 로 설정하는 것이 좋습니다.

이 기능은 이 구성 요소의 v2.6.0 이상에서 사용할 수 있습니다.

기본값: `false`

`networkProxy`

(선택 사항) 모든 연결에 사용할 네트워크 프록시. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

> **Important**
> 이 구성 매개 변수에 변경 내용을 배포하면 AWS IoT Greengrass Core 소프트웨어가 다시 시작되어 변경 내용이 적용됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`noProxyAddresses`

(선택 사항) 프록시에서 제외되는 IP 주소 또는 호스트 이름을 쉼표로 구분한 목록입니다.

`proxy`

연결할 프록시. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`url`

형식의 프록시 서버 URL`scheme://userinfo@host:port`.

- `scheme`— 스킴은 또는이어야 `http` 합니다`https`.

  > **Important**
  > HTTPS 프록시를 사용하려면 그린그래스 코어 디바이스에서 그린그래스 핵 v2.5.0 이상을 실행해야 합니다.
  > HTTPS 프록시를 구성하는 경우 코어 디바이스의 Amazon 루트 CA 인증서에 프록시 서버 CA 인증서를 추가해야 합니다. 자세한 설명은 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.

- `userinfo`— (선택 사항) 사용자 이름 및 암호 정보. 에서 이 정보를 지정하는 경우 Greengrass 코어 디바이스는 및 필드를 무시합니다. `url` `username` `password`
- `host`— 프록시 서버의 호스트 이름 또는 IP 주소.
- `port`— (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 기기는 다음 기본값을 사용합니다.
  - `http`— 80
  - `https`— 443

`username`

(선택 사항) 프록시 서버를 인증하는 사용자 이름.

`password`

(선택 사항) 프록시 서버를 인증하는 암호.

`mqtt`

(선택 사항) 그린그래스 코어 디바이스의 MQTT 구성. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

> **Important**
>
> 이 구성 매개변수에 변경 내용을 배포하면 AWS IoT Greengrass Core 소프트웨어가 다시 시작되어 변경 내용이 적용됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`port`

(선택 사항) MQTT 연결에 사용할 포트입니다.

기본값: `8883`

`keepAliveTimeoutMs`

(선택 사항) 클라이언트가 MQTT 연결을 유지하기 위해 전송하는 각 `PING` 메시지 사이의 시간(밀리초). 이 값은 보다 커야 합니다. `pingTimeoutMs`

기본값: `60000` (60초)

`pingTimeoutMs`

(선택 사항) 클라이언트가 서버로부터 `PINGACK` 메시지를 수신할 때까지 기다리는 시간 (밀리초). 대기 시간이 제한 시간을 초과하면 코어 디바이스가 닫히고 MQTT 연결이 다시 열립니다. 이 값은 보다 작아야 합니다. `keepAliveTimeoutMs`

기본값: `30000` (30초)

`operationTimeoutMs`

(선택 사항) 클라이언트가 MQTT 작업 (예: `CONNECT` 또는) 이 완료될 때까지 기다리는 시간 (밀리초`PUBLISH`) 입니다. 이 옵션은 MQTT `PING` 또는 킵 얼라이브 메시지에는 적용되지 않습니다.

기본값: `30000` (30초)

`maxInFlightPublishes`

(선택 사항) 동시에 전송할 수 있는 승인되지 않은 MQTT QoS 1 메시지의 최대 수입니다.

이 기능은 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

기본값: `5`

유효 범위: 최대값 100

`maxMessageSizeInBytes`

(선택 사항) MQTT 메시지의 최대 크기. 메시지가 이 크기를 초과하면 Greengrass 핵은 메시지를 거부하고 오류가 발생합니다.

이 기능은 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

기본값: `131072` (128KB)

유효 범위: 최대값 `2621440` (2.5MB)

`maxPublishRetry`

(선택 사항) 게시하지 못한 메시지를 다시 시도할 수 있는 최대 횟수입니다. 횟수 제한 없이 `-1` 재시도하도록 지정할 수 있습니다.

이 기능은 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

기본값: `100`

`spooler`

(선택 사항) 그린그래스 코어 디바이스의 MQTT 스풀러 구성. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`storageType`

메시지를 저장하기 위한 스토리지 유형. 로 `storageType` 설정된 `Disk` 경우 를 구성할 `pluginName` 수 있습니다. `Memory` 또는 `Disk`를 지정할 수 있습니다.

이 기능은 v2.11.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

> **Important**
>
> MQTT 스풀러가 로 `storageType` 설정되어 `Disk` 있고 Greengrass nucleus를 버전 2.11.x에서 이전 버전으로 다운그레이드하려면 구성을 다시 로 변경해야 합니다.

Memory 이에 대한 storageType 유일한 컨피그레이션은 Greengrass nucleus 버전 2.10.x 및 이전 버전에서 지원됩니다. Memory 이 지침을 따르지 않으면 스풀러가 손상될 수 있습니다. 이로 인해 Greengrass 코어 디바이스에서 MQTT 메시지를 에 보낼 수 없게 됩니다. AWS 클라우드

기본값: Memory

pluginName

(선택 사항) 플러그인 구성 요소 이름. 이 구성 요소는 로 설정된 경우에만 storageType 사용됩니다Disk. 이 옵션은 기본적으로 디스크 스풀러 Greengrass에서 aws.greengrass.DiskSpooler 제공하는 것을 사용하며 이 옵션을 사용합니다.

이 기능은 v2.11.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: "aws.greengrass.DiskSpooler"

maxSizeInBytes

(선택 사항) 코어 디바이스가 처리되지 않은 MQTT 메시지를 메모리에 저장하는 최대 캐시 크기입니다. 캐시가 가득 차면 새 메시지는 거부됩니다.

기본값: 2621440 (2.5MB)

keepQos0WhenOffline

(선택 사항) 코어 디바이스가 오프라인일 때 수신하는 MQTT QoS 0 메시지를 스풀링할 수 있습니다. 이 옵션을 로 설정하면 코어 디바이스가 오프라인 true 상태에서는 전송할 수 없는 QoS 0 메시지를 스풀링합니다. 이 옵션을 로 false 설정하면 코어 디바이스가 이러한 메시지를 삭제합니다. 코어 디바이스는 스풀이 꽉 차지 않는 한 항상 QoS 1 메시지를 스풀링합니다.

기본값: false

version

(선택 사항) MQTT 버전. mqtt3 또는 mqtt5를 지정할 수 있습니다.

이 기능은 v2.10.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: mqtt5

`receiveMaximum`

(선택 사항) 브로커가 전송할 수 있는 승인되지 않은 QoS1 패킷의 최대 수입니다.

이 기능은 v2.10.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: `100`

`sessionExpirySeconds`

(선택 사항) IoT Core에서 세션을 지속하도록 요청할 수 있는 시간 (초) 입니다. 기본값은 에서 지원하는 최대 시간입니다 AWS IoT Core.

이 기능은 v2.10.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: `604800 (7 days)`

`minimumReconnectDelaySeconds`

(선택 사항) 재연결 동작을 위한 옵션입니다. MQTT가 다시 연결되는 데 걸리는 최소 시간 (초).

이 기능은 v2.10.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: `1`

`maximumReconnectDelaySeconds`

(선택 사항) 재연결 동작을 위한 옵션입니다. MQTT가 다시 연결하는 데 걸리는 최대 시간 (초).

이 기능은 v2.10.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: `120`

`minimumConnectedTimeBeforeRetryResetSeconds`

(선택 사항) 재연결 동작을 위한 옵션입니다. 재시도 지연이 최소로 재설정되기 전에 연결이 활성 상태여야 하는 시간 (초) 입니다.

이 기능은 v2.10.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

기본값: `30`

`jvmOptions`

(선택 사항) Core 소프트웨어를 실행하는 데 사용할 JVM 옵션. AWS IoT Greengrass AWS IoT Greengrass Core 소프트웨어 실행을 위한 권장 JVM 옵션에 대한 자세한 내용은 을 참조하십시오. JVM 옵션으로 메모리 할당을 제어하세요.

> **Important**
>
> 이 구성 매개변수에 변경 내용을 배포하면 AWS IoT Greengrass Core 소프트웨어가 다시 시작되어 변경 내용이 적용됩니다.

`iotDataEndpoint`

사용자의 AWS IoT AWS 계정 데이터 엔드포인트.

`--provision true`옵션을 사용하여 AWS IoT Greengrass Core 소프트웨어를 실행하면 소프트웨어가 Nucleus 구성 요소에서 데이터 및 자격 증명 엔드포인트를 AWS IoT 가져와 이를 설정합니다.

`iotCredEndpoint`

사용자의 AWS IoT 자격 증명 엔드포인트. AWS 계정

`--provision true`옵션을 사용하여 AWS IoT Greengrass Core 소프트웨어를 실행하면 소프트웨어가 Nucleus 구성 요소에서 데이터 및 자격 증명 엔드포인트를 AWS IoT 가져와 이를 설정합니다.

`greengrassDataPlaneEndpoint`

이 기능은 이 구성 요소의 v2.7.0 이상에서 사용할 수 있습니다.

자세한 설명은 사설 CA에서 서명한 장치 인증서를 사용하십시오. 섹션을 참조하세요.

`greengrassDataPlanePort`

이 기능은 이 구성 요소의 v2.0.4 이상에서 사용할 수 있습니다.

(선택 사항) 데이터 플레인 연결에 사용할 포트입니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요.

> **Important**
>
> 장치가 아웃바운드 연결을 할 수 있는 포트를 지정해야 합니다. 차단된 포트를 지정하면 장치를 AWS IoT Greengrass 연결하여 배포를 받을 수 없습니다.

다음 옵션 중 하나를 선택합니다.

- 443
- 8443

기본값: 8443

`awsRegion`

사용 AWS 리전 방법.

`runWithDefault`

구성 요소를 실행하는 데 사용할 시스템 사용자입니다.

> **Important**
>
> 이 구성 매개변수에 변경 내용을 배포하면 AWS IoT Greengrass Core 소프트웨어가 다시 시작되어 변경 내용이 적용됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`posixUser`

시스템 사용자의 이름 또는 ID, 선택적으로 코어 디바이스가 일반 및 Lambda 구성 요소를 실행하는 데 사용하는 시스템 그룹. `user:group` 형식으로 사용자와 그룹을 콜론(`:`)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass Core 소프트웨어는 사용자의 기본 그룹을 사용합니다. 예를 들어 `ggc_user` 또는 를 지정할 수 있습니다`ggc_user:ggc_group`. 자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.

`--component-default-user` *`ggc_user`*`:`*`ggc_group`*옵션과 함께 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 실행하면 소프트웨어가 nucleus 구성 요소에 이 매개변수를 설정합니다.

`windowsUser`

이 기능은 이 구성 요소의 v2.5.0 이상에서 사용할 수 있습니다.

Windows 코어 디바이스에서 이 구성 요소를 실행하는 데 사용할 Windows 사용자의 이름입니다. 사용자는 각 Windows 코어 장치에 존재해야 하며 사용자 이름과 암호는 LocalSystem 계정의 자격 증명 관리자 인스턴스에 저장되어 있어야 합니다. 자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.

`--component-default-user` *`ggc_user`*옵션과 함께 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 실행하면 소프트웨어가 nucleus 구성 요소에 이 매개 변수를 설정합니다.

`systemResourceLimits`

이 기능은 이 구성 요소의 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 코어 장치에서는 이 기능을 지원하지 않습니다.

기본적으로 일반 및 비컨테이너식 Lambda 구성 요소 프로세스에 적용할 시스템 리소스 제한입니다. 배포를 생성할 때 개별 구성 요소에 대한 시스템 리소스 제한을 재정의할 수 있습니다. 자세한 설명은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`cpus`

각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 장치의 경우 이 값을 2 설정하여 각 구성 요소의 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 기기에서는 이 값을 `0.25` 설정하여 각 구성 요소의 프로세스를 CPU 사용량의 25% 로 제한할 수 있습니다. 이 값을 CPU AWS IoT Greengrass 코어 수보다 큰 수로 설정하는 경우 Core 소프트웨어는 구성 요소의 CPU 사용량을 제한하지 않습니다.

`memory`

각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 RAM 크기 (KB).

`s3EndpointType`

(선택 사항) S3 엔드포인트 유형. 이 파라미터는 미국 동부 (버지니아 북부) (`us-east-1`) 지역에만 적용됩니다. 다른 지역에서 이 매개변수를 설정하면 무시됩니다. 다음 옵션 중 하나를 선택합니다.

- `REGIONAL`— S3 클라이언트 및 사전 서명된 URL은 리전 엔드포인트를 사용합니다.
- `GLOBAL`— S3 클라이언트 및 미리 서명된 URL은 레거시 엔드포인트를 사용합니다.

기본값: `GLOBAL`

`logging`

(선택 사항) 코어 디바이스의 로깅 구성. Greengrass 로그를 구성하고 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 모니터 AWS IoT Greengrass 로그

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`level`

(선택 사항) 출력할 로그 메시지의 최소 수준.

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`format`

(선택 사항) 로그의 데이터 형식. 다음 옵션 중 하나를 선택합니다.

- `TEXT`— 로그를 텍스트 형식으로 보려면 이 옵션을 선택합니다.
- `JSON`— Greengrass CLI logs 명령으로 로그를 보거나 프로그래밍 방식으로 로그와 상호 작용하려면 이 옵션을 선택하십시오.

기본값: `TEXT`

`outputType`

(선택 사항) 로그의 출력 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `FILE`— AWS IoT Greengrass Core 소프트웨어는 지정한 디렉터리의 파일에 로그를 `outputDirectory` 출력합니다.
- `CONSOLE`— AWS IoT Greengrass Core 소프트웨어는 로그를 에 `stdout` 인쇄합니다. 코어 디바이스에서 로그를 인쇄할 때 로그를 보려면 이 옵션을 선택합니다.

기본값: `FILE`

`fileSizeKB`

(선택 사항) 각 로그 파일의 최대 크기 (KB). 로그 파일이 이 최대 파일 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어에서 새 로그 파일을 생성합니다.

이 매개 변수는 `FILE` 를 지정하는 경우에만 적용됩니다`outputType`.

기본값: `1024`

`totalLogsSizeKB`

(선택 사항) Greengrass 핵을 포함한 각 구성 요소에 대한 로그 파일의 최대 총 크기 (킬로바이트). Greengrass nucleus의 로그 파일에는 플러그인 구성 요소의 로그도 포함됩니다. 구성 요소의 전체 로그 파일 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 해당 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 매개 변수는 Greengrass nucleus (시스템`diskSpaceLimit`) 및 각 구성 요소에 대해 지정할 수 있는 로그 관리자 구성 요소의 디스크 공간 제한 매개 변수 () 와 동일합니다. AWS IoT Greengrass Core 소프트웨어는 두 값 중 최소값을 Greengrass 핵과 각 구성 요소의 최대 총 로그 크기로 사용합니다.

이 매개변수는 를 지정하는 `FILE` 경우에만 적용됩니다. `outputType`

기본값: `10240`

`outputDirectory`

(선택 사항) 로그 파일의 출력 디렉터리입니다.

이 매개 변수는 `FILE` 를 지정하는 경우에만 적용됩니다`outputType`.

기본값:*`/greengrass/v2`*`/logs`, 여기서 *`/greengrass/v2`* 는 AWS IoT Greengrass 루트 폴더입니다.

`fleetstatus`

이 매개 변수는 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

(선택 사항) 코어 디바이스의 플릿 상태 구성.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`periodicStatusPublishIntervalSeconds`

(선택 사항) 코어 디바이스가 디바이스 상태를 에 게시하는 데 걸리는 시간 (초). AWS 클라우드

최소: `86400` (24시간)

기본값: `86400` (24시간)

`telemetry`

(선택 사항) 코어 장치의 시스템 상태 원격 측정 구성. 원격 분석 메트릭 및 원격 분석 데이터에 대한 조치 방법에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`enabled`

(선택 사항) 원격 분석을 활성화하거나 비활성화할 수 있습니다.

기본값: `true`

`periodicAggregateMetricsIntervalSeconds`

(선택 사항) 코어 디바이스가 지표를 집계하는 간격 (초).

이 값을 지원되는 최소값보다 낮게 설정하면 NUCLEUS는 기본값을 대신 사용합니다.

최소: `3600`

기본값: `3600`

`periodicPublishMetricsIntervalSeconds`

(선택 사항) 코어 장치가 원격 분석 메트릭을 에 게시하는 데 걸리는 시간 (초) 입니다. AWS 클라우드

이 값을 지원되는 최소값보다 낮게 설정하면 Nucleus는 기본값을 대신 사용합니다.

최소: `86400`

기본값: `86400`

`deploymentPollingFrequencySeconds`

(선택 사항) 배포 알림을 폴링하는 기간 (초) 입니다.

기본값: `15`

`componentStoreMaxSizeBytes`

(선택 사항) 구성 요소 저장소의 최대 디스크 크기로, 구성 요소 레시피와 아티팩트로 구성됩니다.

기본값: `10000000000` (10GB)

`platformOverride`

(선택 사항) 코어 기기의 플랫폼을 식별하는 속성 사전입니다. 이를 사용하여 구성 요소 레시피가 구성 요소의 올바른 수명 주기 및 아티팩트를 식별하는 데 사용할 수 있는 사용자 지정 플랫폼 속성

을 정의할 수 있습니다. 예를 들어, 실행할 구성 요소의 최소 아티팩트 집합만 배포하도록 하드웨어 기능 속성을 정의할 수 있습니다. 자세한 내용은 구성 요소 레시피의 매니페스트 플랫폼 매개 변수를 참조하십시오.

이 파라미터를 사용하여 코어 디바이스의 `os` 및 `architecture` 플랫폼 속성을 오버라이드할 수도 있습니다.

`httpClient`

이 매개변수는 이 구성 요소의 v2.5.0 이상에서 사용할 수 있습니다.

(선택 사항) 코어 디바이스의 HTTP 클라이언트 구성. 이러한 구성 옵션은 이 구성 요소가 수행하는 모든 HTTP 요청에 적용됩니다. 코어 디바이스가 느린 네트워크에서 실행되는 경우 이러한 제한 시간을 늘려 HTTP 요청 제한 시간이 초과되지 않도록 할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`connectionTimeoutMs`

(선택 사항) 연결 요청 제한 시간이 초과되기 전에 연결이 열릴 때까지 기다리는 시간 (밀리초) 입니다.

기본값: `2000` (2초)

`socketTimeoutMs`

(선택 사항) 연결 제한 시간이 초과되기 전에 열린 연결을 통해 데이터가 전송될 때까지 기다리는 시간 (밀리초) 입니다.

기본값: `30000` (30초)

Example 예: 구성 병합 업데이트

```
{
  "iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "http://my-proxy-server:1100",
      "username": "Mary_Major",
      "password": "pass@word1357"
    }
  },
```

```
  "mqtt": {
    "port": 443
  },
  "greengrassDataPlanePort": 443,
  "jvmOptions": "-Xmx64m",
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.12.2	버그 수정 및 개선 • 이전 로그가 제대로 정리되지 않던 문제를 수정합니다. • 일반적인 버그 수정 및 개선입니다.
2.12.1	버그 수정 및 개선 • NUCLEUS가 배포 주제에 대한 MQTT 구독을 중복하여 추가 로깅 및 MQTT 게시로 이어질 수 있는 문제를 수정합니다.
2.12.0	새로운 기능 • 롤백 배포의 일부로 부트스트랩 라이프사이클 단계를 실행할 수 있습니다.
2.11.3	버그 수정 및 개선 • 종속성이 실패할 경우 구성 요소가 제대로 시작되지 않을 수 있는 Nucleus 문제를 수정합니다. 새로운 기능 • 구성 가능한 s3 엔드포인트 유형을 추가합니다.
2.11.2	버그 수정 및 개선 • Nucleus MQTT 5 클라이언트에서 많은 수 (50개 이상) 의 구독을 사용 중인 경우 오프라인으로 표시될 수 있는 문제를 수정합니다. • docker 다이얼 TCP 실패에 대한 재시도를 추가합니다.
2.11.1	버그 수정 및 개선 • 부트스트랩 작업이 실패하고 배포 메타데이터 파일이 손상된 경우 Nucleus가 시작되지 않는 문제를 수정합니다. • 온디맨드 Lambda 구성 요소가 배포 상태 업데이트에 보고되지 않는 문제를 수정합니다. • 중복 권한 부여 정책 ID에 대한 지원을 추가합니다.
2.11.0	새로운 기능 • 로컬 배포를 취소할 수 있습니다. • 로컬 배포를 위한 장애 처리 정책을 구성할 수 있습니다.

버전	변경
	• 디스크 스풀러 플러그인에 대한 지원을 추가합니다.
2.10.3	버그 수정 및 개선 • Greengrass가 PKCS #11 제공자를 사용할 때 배포 알림을 구독하지 않는 문제를 수정합니다.
2.10.2	버그 수정 및 개선 • 구성 요소 수명 주기의 대소문자를 구분하지 않는 구문 분석을 허용합니다. • 환경 PATH 변수가 올바르게 다시 생성되지 않던 문제를 수정합니다. • 특수 문자가 있는 사용자 이름의 스트림 관리자를 비롯한 구성 요소의 프록시 URI 인코딩을 수정합니다.
2.10.1	버그 수정 및 개선 • Jetson Nano를 비롯한 특정 ARMv8 프로세서에서 시작 시 충돌이 발생할 수 있는 문제를 수정합니다. • Greengrass는 더 이상 구성 요소의 표준을 닫지 않으므로 동작이 2.10.0 이전 동작으로 되돌아갑니다.

버전	변경
2.10.0	새로운 기능 • 빈 정규 `interpolateComponentConfiguration` 표현식에 대한 지원을 추가합니다. Greengrass는 이제 루트 구성 객체에서 보간합니다. • MQTT5 지원을 추가합니다. • 플러그인 구성 요소를 스캔하지 않고 빠르게 로드할 수 있는 메커니즘을 추가합니다. • Greengrass가 사용하지 않는 Docker 이미지를 삭제하여 디스크 공간을 절약할 수 있도록 합니다. 버그 수정 및 개선 • 롤백으로 인해 배포의 특정 구성 값이 그대로 유지되는 문제를 수정합니다. • Greengrass nucleus가 사용자 지정 AWS 비자격 증명 및 데이터 엔드포인트에서 AWS 도메인 시퀀스를 검증하는 문제를 수정합니다. • 활성 버전으로 잠그는 대신 AWS 클라우드 협상을 통해 모든 그룹 종속성을 다시 해결하도록 다중 그룹 종속성 해결을 업데이트합니다. 또한 이 업데이트는 배포 오류 코드도 제거합니다. `INSTALLED_COMPONENT_NOT_FOUND` • Docker 이미지가 이미 로컬에 있는 경우 다운로드를 건너뛰도록 Greengrass 핵을 업데이트합니다. • 제한 시간이 만료되기 전에 구성 요소 설치 단계를 다시 시작하도록 Greengrass Nucleus를 업데이트합니다. • 추가 사소한 수정 및 개선.
2.9.6	버그 수정 및 개선 • LAUNCH_DIRECTORY_CORRTED 오류와 함께 그린그래스 배포가 실패하고 이후 디바이스 재부팅 시 Greengrass가 시작되지 않는 문제를 수정합니다. 이 오류는 Greengrass를 다시 시작해야 하는 배포가 있는 여러 사물 그룹 간에 Greengrass 디바이스를 이동할 때 발생할 수 있습니다.

버전	변경
2.9.5	새로운 기능 • Greengrass 핵 소프트웨어 서명 검증에 대한 지원을 추가합니다. 버그 수정 및 개선 • 로컬 레시피 메타데이터 영역이 Greengrass nucleus 시작 지역과 일치하지 않는 경우 배포가 실패하는 문제를 수정합니다. Greengrass 핵은 이제 이런 일이 발생하면 클라우드와 재협상합니다. • MQTT 메시지 스풀러가 꽉 차서 메시지를 제거하지 않는 문제를 수정합니다. • 추가 사소한 수정 및 개선.
2.9.4	버그 수정 및 개선 • QOS 0 메시지를 삭제하기 전에 null 메시지가 있는지 확인합니다. • 작업 상태 세부 정보 값이 1024자 제한을 초과하는 경우 해당 값을 잘라냅니다. • Greengrass 루트 경로에 공백이 포함된 경우 Greengrass 루트 경로를 올바르게 읽도록 Windows용 부트스트랩 스크립트를 업데이트합니다. • 구독 응답이 전송되지 않은 경우 클라이언트 메시지가 AWS IoT Core 삭제되도록 구독을 업데이트합니다. • 기본 구성 파일이 손상되거나 누락된 경우 Nucleus가 백업 파일에서 구성을 로드하도록 합니다.
2.9.3	버그 수정 및 개선 • MQTT 클라이언트 ID가 중복되지 않도록 합니다. • 손상을 방지하고 복구할 수 있도록 보다 강력한 파일 읽기 및 쓰기 기능을 추가합니다. • 특정 네트워크 관련 오류가 발생하면 docker image pull을 재시도합니다. • MQTT 연결 `noProxyAddresses` 옵션을 추가합니다.
2.9.2	버그 수정 및 개선 • 구성이 진행 중인 배포에 적용되지 `interpolateComponentConfiguration` 않는 문제를 수정합니다. • OSHI를 사용하여 모든 하위 프로세스를 나열합니다.

버전	변경
2.9.1	버그 수정 및 개선 • 디플로이먼트에서 플러그인 컴포넌트가 제거되면 Greengrass가 다시 시작되는 문제를 수정했습니다.
2.9.0	새로운 기능 • 더 적은 수의 하위 장치로 배포를 다시 시도하는 하위 배포를 생성하는 기능을 추가합니다. 이 기능을 사용하면 실패한 배포를 보다 효율적으로 테스트하고 해결할 수 있습니다. 버그 수정 및 개선 • `useraddgroupadd`, 및 가 없는 시스템에 대한 지원을 `usermod` 개선합니다. • 기타 사소한 수정 및 개선.
2.8.1	버그 수정 및 개선 • Greengrass API 오류에서 배포 오류 코드가 올바르게 생성되지 않던 문제를 수정합니다. • 구성 요소가 배포 중에 상태에 도달하면 플릿 상태 업데이트가 부정확한 정보를 전송하는 문제를 수정합니다. `ERRORED` • Greengrass의 기존 구독이 50개 이상인 경우 배포를 완료할 수 없었던 문제를 수정합니다.

버전	변경
2.8.0	새로운 기능 • Greengrass nucleus를 업데이트하여 핵심 장치에 구성 요소를 배포하는 데 문제가 발생할 경우 자세한 오류 코드가 포함된 배포 상태 응답을 보고합니다. 자세한 설명은 세부 배포 오류 코드 섹션을 참조하세요. • 구성 요소가 또는 상태에 들어갈 때 자세한 오류 코드가 포함된 구성 요소 상태 응답을 보고하도록 Greengrass Nucleus를 업데이트합니다BROKEN. ERRORED 자세한 설명은 세부 구성 요소 상태 코드 섹션을 참조하세요. • 상태 메시지 필드를 확장하여 디바이스의 클라우드 가용성 정보를 개선합니다. • 플릿 상태 서비스 안정성을 개선합니다. 버그 수정 및 개선 • 구성이 변경될 때 손상된 구성 요소를 다시 설치할 수 있도록 합니다. • 부트스트랩 배포 중 Nucleus 재시작으로 인해 배포가 실패하는 문제를 수정합니다. • Windows에서 루트 경로에 공백이 있는 경우 설치가 실패하는 문제를 수정합니다. • 배포 중에 종료된 구성 요소가 새 버전의 종료 스크립트를 사용하는 문제를 수정합니다. • 다양한 종료 개선 사항. • 추가 사소한 수정 및 개선

버전	변경
2.7.0	새로운 기능 • 코어 디바이스가 로컬 배포를 적용할 때 상태 업데이트를 AWS IoT Greengrass 클라우드로 전송하도록 Greengrass Nucleus를 업데이트합니다. • CA가 등록되지 않은 사용자 지정 인증 기관 (CA) 에서 서명한 클라이언트 인증서에 대한 지원을 추가합니다. AWS IoT 이 기능을 사용하려면 새 `greengrassDataPlaneEndpoint` 구성 옵션을 로 설정하면 `iotdata` 됩니다. 자세한 설명은 사설 CA에서 서명한 장치 인증서를 사용하십시오. 섹션을 참조하세요. 버그 수정 및 개선 • 특정 시나리오에서 핵이 중지되거나 다시 시작될 때 Greengrass nucleus가 배포를 롤백하는 문제를 수정합니다. 이제 Nucleus가 재시작된 후 Nucleus는 배포를 재개합니다. • 소프트웨어를 시스템 서비스로 설정하도록 지정할 때 `--start` 인수를 준수하도록 Greengrass 설치 프로그램을 업데이트합니다. • nucleus가 구성 SubscribeToComponentUpdates 요소를 업데이트한 이벤트에서 배포 ID를 설정하는 동작을 업데이트합니다. • 추가 사소한 수정 및 개선

버전	변경
2.6.0	새로운 기능 • 로컬 게시/구독 주제를 구독할 때 MQTT 와일드카드 지원을 추가합니다. 자세한 내용은 로컬 메시지 게시/구독 및 SubscribeToTopic 섹션을 참조하세요. • 레시피 변수 이외의 구성 요소 구성의 레시피 변수에 대한 지원을 추가합니다. *component_dependency_name* :configuration: *json_pointer* 레시피에 구성 요소를 정의하거나 DefaultConfiguration 배포에서 구성 요소를 구성할 때 이러한 레시피 변수를 사용할 수 있습니다. 이 기능을 활성화하려면 interpolateComponentConfiguration구성 옵션을 로 설정합니다true. 자세한 내용은 레시피 변수 및 병합 업데이트에 레시피 변수를 사용하십시오. 섹션을 참조하세요. • IPC (프로세스 간 통신) 권한 부여 정책에 * 와일드카드에 대한 전체 지원을 추가합니다. 이제 리소스 문자열의 문자를 모든 * 문자 조합과 일치하도록 지정할 수 있습니다. 자세한 설명은 권한 부여 정책의 와일드카드 섹션을 참조하세요. • Greengrass CLI에서 사용하는 IPC 작업을 호출하는 사용자 지정 구성 요소에 대한 지원을 추가합니다. 이러한 IPC 작업을 사용하여 로컬 배포를 관리하고, 구성 요소 세부 정보를 보고, 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 암호를 생성할 수 있습니다. 자세한 내용은 IPC: 로컬 배포 및 구성 요소 관리를 참조하십시오. 버그 수정 및 개선 • 특정 시나리오에서 종속 구성 요소의 하드 종속성이 다시 시작되거나 상태가 변경될 때 종속 구성 요소가 반응하지 않는 문제를 수정합니다. • 배포 실패 시 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 보고하는 오류 메시지를 개선합니다. • 특정 시나리오에서 핵이 재시작될 때 Greengrass 핵이 사물 배포를 두 번 적용한 문제를 수정합니다. • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.

버전	변경
2.5.6	새로운 기능 • ECC 키를 사용하는 하드웨어 보안 모듈에 대한 지원을 추가합니다. 하드웨어 보안 모듈 (HSM) 을 사용하여 디바이스의 개인 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요. 버그 수정 및 개선 • 특정 시나리오에서 손상된 설치 스크립트가 포함된 구성 요소를 배포할 때 배포가 완료되지 않는 문제를 수정합니다. • 시작 시 성능을 개선합니다. • 추가 사소한 수정 및 개선
2.5.5	새로운 기능 • 사용자 지정 구성 요소의 루트 CA (인증 기관) 인증서에 액세스할 수 있도록 구성 요소의 `GG_ROOT_CA_PATH` 환경 변수를 추가합니다. 버그 수정 및 개선 • 영어가 아닌 표시 언어를 사용하는 Windows 장치에 대한 지원을 추가합니다. • Greengrass nucleus가 부울 설치 프로그램 인수를 파싱하는 방식을 업데이트하여 부울 값 없이 부울 인수를 지정하여 값을 지정할 수 있도록 합니다. `true` 예를 들어 이제 자동 리소스 프로비저닝으로 설치하는 `--provision` 대신 `--provision true` 지정하도록 지정할 수 있습니다. • 특정 시나리오에서 코어 디바이스가 프로비저닝한 후 AWS IoT Greengrass 클라우드 서비스에 상태를 보고하지 않던 문제를 수정합니다. • 추가 사소한 수정 및 개선
2.5.4	버그 수정 및 개선 • 일반적인 버그 수정 및 개선입니다.

버전	변경
2.5.3	새로운 기능 • 하드웨어 보안 통합에 대한 지원을 추가합니다. 하드웨어 보안 모듈 (HSM) 을 사용하여 기기의 개인 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요. 버그 수정 및 개선 • Nucleus가 MQTT 연결을 설정하는 동안 발생하는 런타임 예외 문제를 수정합니다. AWS IoT Core
2.5.2	버그 수정 및 개선 • Greengrass nucleus 업데이트 후 Windows 서비스를 중지하거나 디바이스를 재부팅한 후 Windows 서비스가 다시 시작되지 않는 문제를 수정합니다.
2.5.1	**Warning** 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 • Windows용 자바 런타임 환경 (JRE) 의 32비트 버전에 대한 지원을 추가합니다. • AWS IoT 정책이 권한을 부여하지 않는 핵심 장치에 대한 사물 그룹 제거 동작을 변경합니다. `greengrass:ListThingGroupsForCoreDevice` 이 버전에서는 배포가 계속되고 경고가 기록되며 사물 그룹에서 코어 장치를 제거해도 구성 요소가 제거되지 않습니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요. • Greengrass 핵이 Greengrass 구성 요소 프로세스에 사용할 수 있도록 하는 시스템 환경 변수 관련 문제를 수정합니다. 이제 최신 시스템 환경 변수를 사용하도록 구성 요소를 다시 시작할 수 있습니다.

버전	변경
2.5.0	새로운 기능 • Windows를 실행하는 핵심 장치에 대한 지원을 추가합니다. • 사물 그룹 제거 동작을 변경합니다. 이 버전에서는 사물 그룹에서 코어 장치를 제거하여 다음 배포 시 해당 사물 그룹의 구성 요소를 제거할 수 있습니다. 이 변경으로 인해 코어 디바이스의 AWS IoT 정책에 `greengrass:ListThingGroupsForCoreDevice` 권한이 있어야 합니다. AWS IoT Greengrass 코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 기본 AWS IoT 정책이 `greengrass:*` 허용되며 여기에는 이 권한이 포함됩니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요. • HTTPS 프록시 구성에 대한 지원을 추가합니다. 자세한 설명은 포트 443에서 또는 네트워크 프록시를 통해 연결 섹션을 참조하세요. • 새 `windowsUser` 구성 매개변수를 추가합니다. 이 매개 변수를 사용하여 Windows 코어 장치에서 구성 요소를 실행하는 데 사용할 기본 사용자를 지정할 수 있습니다. 자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요. • HTTP 요청 제한 시간을 사용자 지정하여 속도가 느린 네트워크에서 성능을 향상시키는 데 사용할 수 있는 새 `httpClient` 구성 옵션을 추가합니다. 자세한 내용은 HttpClient 구성 매개 변수를 참조하십시오. 버그 수정 및 개선 • 구성 요소에서 코어 장치를 다시 시작하는 부트스트랩 수명 주기 옵션을 수정합니다. • 레시피 변수에 하이픈 지원을 추가합니다. • 온디맨드 Lambda 함수 구성 요소에 대한 IPC 인증을 수정합니다. • 로그 메시지를 개선하고 중요하지 않은 로그를 레벨에서 `INFO` `DEBUG` 레벨로 변경하여 로그를 더 유용하게 사용할 수 있도록 합니다. • 자동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치할 때 Greengrass nucleus가 생성하는 기본 토큰 교환 역할에서 `iot:DescribeCertificate` 권한을 제거합니다. 이 권한은 Greengrass 핵에서 사용되지 않습니다.

버전	변경
	• 동일한 정책에 사용할 수 있는 경우 `iam:CreatePolicy` 자동 프로비저닝 스크립트에 `iam:GetPolicy` 권한이 필요하지 않도록 문제를 수정합니다. • 추가 사소한 수정 및 개선
2.4.0	새로운 기능 • 시스템 리소스 제한에 대한 지원을 추가합니다. 각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 설명은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요. • 구성 요소를 일시 중지 및 재개하기 위한 IPC 작업을 추가합니다. 자세한 내용은 PauseComponent 및 ResumeComponent 섹션을 참조하세요. • 프로비저닝 플러그인에 대한 지원을 추가합니다. 설치 중에 실행할 JAR 파일을 지정하여 Greengrass 코어 장치에 필요한 AWS 리소스를 프로비저닝할 수 있습니다. Greengrass Nucleus에는 사용자 지정 프로비저닝 플러그인을 개발하기 위해 구현할 수 있는 인터페이스가 포함되어 있습니다. 자세한 설명은 사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요. • AWS IoT Greengrass Core 소프트웨어 설치 `thing-name-policy` 프로그램에 선택적 인수를 추가합니다. 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치할 때 이 옵션을 사용하여 기존 또는 사용자 지정 AWS IoT 정책을 지정할 수 있습니다. 버그 수정 및 개선 • 시작 시 로깅 구성을 업데이트합니다. 이렇게 하면 시작 시 로깅 구성이 적용되지 않던 문제가 해결됩니다. • 설치 중에 Greengrass 루트 폴더의 구성 요소 저장소를 가리키도록 Nucleus 로더 심볼릭 링크를 업데이트합니다. 이 업데이트를 사용하면 Core 소프트웨어를 설치할 때 다운로드한 JAR 파일 및 기타 Nucleus 아티팩트를 삭제할 수 있습니다. AWS IoT Greengrass • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.

버전	변경
2.3.0	새로운 기능 • 7KB (사물을 대상으로 하는 배포의 경우) 또는 31KB (사물 그룹을 대상으로 하는 배포의 경우) 에서 최대 10MB까지 배포 구성 문서에 대한 지원을 추가합니다. 이 기능을 사용하려면 코어 디바이스의 AWS IoT 정책에서 권한을 허용해야 합니다. `greengrass:GetDeploymentConfiguration` AWS IoT Greengrass 코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 코어 장치 AWS IoT 정책에서 `greengrass:*` 허용하며 여기에는 이 권한이 포함됩니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요. • `iot:thingName` 레시피 변수를 추가합니다. 이 레시피 변수를 사용하여 레시피에 있는 핵심 장치 사물의 이름을 가져올 수 있습니다. AWS IoT 자세한 설명은 레시피 변수 섹션을 참조하세요. 버그 수정 및 개선 • 추가 사소한 수정 및 개선. 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
2.2.0	새로운 기능 • 로컬 섀도우 관리를 위한 IPC 작업을 추가합니다. 버그 수정 및 개선 • JAR 파일의 크기를 줄입니다. • 메모리 사용량을 줄입니다. • 특정 경우에 로그 구성이 업데이트되지 않던 문제를 수정합니다. • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.

버전	변경

2.1.0	새로운 기능
	• Amazon ECR의 프라이빗 리포지토리에서 Docker 이미지를 다운로드할 수 있습니다.
	• 코어 디바이스의 MQTT 구성을 사용자 지정하기 위해 다음 파라미터를 추가합니다.
	  • `maxInFlightPublishes` — 동시에 전송될 수 있는 승인되지 않은 MQTT QoS 1 메시지의 최대 수입니다.
	  • `maxPublishRetry` — 게시하지 못한 메시지를 재시도할 수 있는 최대 횟수입니다.
	• `fleetstatusservice` 구성 매개 변수를 추가하여 코어 장치가 장치 상태를 에 게시하는 간격을 구성합니다. AWS 클라우드
	• 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
	버그 수정 및 개선
	• Nucleus가 다시 시작될 때 섀도우 디플로이먼트가 중복되던 문제를 수정합니다.
	• 서비스 로드 예외가 발생했을 때 Nucleus가 충돌하던 문제를 수정합니다.
	• 순환 종속성을 포함하는 배포가 실패하도록 구성 요소 종속성 해결을 개선합니다.
	• 플러그인 구성 요소가 이전에 코어 장치에서 제거된 경우 플러그인 구성 요소가 재배포되지 않던 문제를 수정합니다.
	• `HOME`환경 변수가 Lambda 구성 요소 또는 루트로 실행되는 구성 요소의 */greengrass/v2* `/work` 디렉토리로 설정되는 문제를 수정했습니다. 이제 구성 요소를 실행하는 사용자의 홈 디렉터리에 `HOME` 변수가 올바르게 설정되었습니다.
	• 기타 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.

버전	변경
2.0.5	버그 수정 및 개선 • AWS제공된 구성 요소를 다운로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. • AWS 중국 지역에서 올바른 Greengrass 데이터 플레인 엔드포인트를 사용하십시오.
2.0.4	새로운 기능 • 포트 443을 통한 HTTPS 트래픽을 활성화합니다. nucleus 구성 요소 버전 2.0.4의 새 `greengrassDataPlanePort` 구성 매개변수를 사용하여 기본 포트 8443이 아닌 포트 443을 통해 이동하도록 HTTPS 통신을 구성할 수 있습니다. 자세한 설명은 포트 443을 통해 HTTPS를 구성합니다. 섹션을 참조하세요. • 작업 경로 레시피 변수를 추가합니다. 이 레시피 변수를 사용하여 구성 요소의 작업 폴더 경로를 가져올 수 있습니다. 이 경로를 사용하여 구성 요소와 해당 종속성 간에 파일을 공유할 수 있습니다. 자세한 내용은 작업 경로 레시피 변수를 참조하십시오. 버그 수정 및 개선 • 역할 정책이 이미 존재하는 경우 토큰 교환 AWS Identity and Access Management (IAM) 역할 정책이 생성되지 않도록 합니다. 이 변경으로 인해 이제 설치 프로그램을 실행할 `sts:GetCallerIdentity` 때 `iam:GetPolicy` 및 가 필요합니다. `--provision true` 자세한 설명은 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책 섹션을 참조하세요. • 아직 성공적으로 등록되지 않은 배포의 취소를 올바르게 처리합니다. • 배포를 롤백할 때 최신 타임스탬프가 있는 이전 항목을 제거하도록 구성을 업데이트합니다. • 추가 사소한 수정 및 개선 자세한 내용은 의 릴리스를 참조하십시오 GitHub.
2.0.3	초기 버전

클라이언트 장치 인증

클라이언트 장치 인증 구성 요소 (`aws.greengrass.clientdevices.Auth`) 는 클라이언트 장치를 인증하고 클라이언트 장치 작업을 승인합니다.

> **Note**
>
> 클라이언트 디바이스는 Greengrass 코어 디바이스에 연결하여 MQTT 메시지와 데이터를 전송하여 처리하는 로컬 IoT 디바이스입니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

주제

버전

> **Note**
>
> 클라이언트 장치 인증 버전 2.3.0이 중단되었습니다. 클라이언트 장치 인증 버전 2.3.1 이상으로 업그레이드하는 것이 좋습니다.

이 구성 요소의 버전은 다음과 같습니다.

- 2.4.x
- 2.3.x

- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 서비스 역할은 사용자와 연결되어 있어야 AWS 계정 하며 권한을 허용해야 합니다. `iot:DescribeCertificate`
- 코어 디바이스의 AWS IoT 정책은 다음 권한을 허용해야 합니다.
  - `greengrass:GetConnectivityInfo`, 리소스에는 이 구성 요소를 실행하는 코어 디바이스의 ARN이 포함됩니다.
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`여기서 리소스에는 코어 디바이스에 연결되는 각 클라이언트 디바이스의 Amazon 리소스 이름 (ARN) 이 포함됩니다.
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:PutCertificateAuthorities`
  - `iot:Publish`, 여기서 리소스에는 다음 MQTT 주제의 ARN이 포함됩니다.

    - `$aws/things/coreDeviceThingName*-gci/shadow/get`
  - `iot:Subscribe`, 여기서 리소스에는 다음 MQTT 주제 필터의 ARN이 포함됩니다.
    - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
    - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
  - `iot:Receive`, 리소스에는 다음 MQTT 주제의 ARN이 포함됩니다.
    - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
    - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

  자세한 내용은 데이터 영역 작업에 대한 AWS IoT 정책 및 클라이언트 AWS IoT 장치를 지원하기 위한 최소 정책 섹션을 참조하세요.
- (선택 사항) 오프라인 인증을 사용하려면 AWS IoT Greengrass 서비스에서 사용하는 AWS Identity and Access Management (IAM) 역할에 다음 권한이 포함되어야 합니다.
  - `greengrass:ListClientDevicesAssociatedWithCoreDevice`코어 디바이스에서 오프라인 인증을 위한 클라이언트를 나열할 수 있도록 합니다.
- 클라이언트 장치 인증 구성 요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.
  - 클라이언트 디바이스 인증 구성 요소는AWS IoT data, AWS IoT 자격 증명 및 Amazon S3에 연결되어 있어야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`iot.region.amazonaws.com`	443	예	사물 인증서에 대한 AWS IoT 정보를 가져오는 데 사용됩니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.4.4

다음 표에는 이 구성 요소의 버전 2.4.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.13.0	소프트

2.4.3

다음 표에는 이 구성 요소의 버전 2.4.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.12.0	소프트

2.4.1 and 2.4.2

다음 표에는 이 구성 요소의 버전 2.4.1 및 2.4.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.11.0	소프트

2.3.0 – 2.4.0

다음 표에는 이 구성 요소의 버전 2.3.0 ~ 2.4.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.10.0	소프트

2.3.0

다음 표에는 이 구성 요소의 버전 2.3.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.10.0	소프트

2.2.3

다음 표에는 이 구성 요소의 버전 2.2.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <=2.9.0	소프트

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <=2.8.0	소프트

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.8.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.6.0 <2.7.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.7.0	소프트

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.6.0	소프트

2.0.2 and 2.0.3

다음 표에는 이 구성 요소의 버전 2.0.2 및 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.5.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.4.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.3.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

v2.4.5

`deviceGroups`

장치 그룹은 핵심 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용

할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *groupNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의 구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름의 처음과 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하거나 끝나는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (`:`) 를 사용하십시오. \ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 MyClientDevice1 또는 인 클라이언트 장치와 MyClientDevice2 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 MyClientDevice 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 MyClientDevice 끝나는 클라이언트 장치와 일치합니다.

```
thingName: *MyClientDevice
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *policyNameKey*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *statementNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- mqtt:connect— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

  이 작업은 다음 리소스를 지원합니다.

  - mqtt:clientId:*deviceClientId*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *deviceClientId*ID로 바꾸십시오.
- mqtt:publish— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - mqtt:topic:*mqttTopic*— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *MQTTtopic# ### ###* 바꾸십시오.

    이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.
- mqtt:subscribe— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - mqtt:topicfilter:*mqttTopicFilter*— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *mqttTopicFilter*필터로 바꾸십시오.

이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 `mqtt:topicfilter:client/+/status` 리소스를 구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

*와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:`*`mqttTopic`*) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

`certificates`

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`serverCertificateValiditySeconds`

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간 (초). 이 옵션을 구성하여 클라이언트 장치의 연결을 끊었다가 코어 장치에 다시 연결하는 빈도를 사용자 지정할 수 있습니다.

이 구성 요소는 만료되기 24시간 전에 로컬 MQTT 서버 인증서를 교체합니다. MQTT 브로커 (예: Moquette MQTT 브로커 구성 요소) 가 새 인증서를 생성하고 다시 시작합니다. 이 경우 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 잠시 후 클라이언트 장치를 코어 장치에 다시 연결할 수 있습니다.

기본값: `604800` (7일)

최소값: `172800` (2일)

최대치: 864000 (10일)

`performance`

(선택 사항) 이 코어 디바이스의 성능 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`maxActiveAuthTokens`

(선택 사항) 활성 클라이언트 장치 인증 토큰의 최대 수. 이 수를 늘리면 더 많은 수의 클라이언트 장치를 재인증하지 않고도 단일 코어 장치에 연결할 수 있습니다.

기본값: 2500

`cloudRequestQueueSize`

(선택 사항) 이 구성 요소가 AWS 클라우드 요청을 거부하기 전에 대기열에 추가할 최대 요청 수입니다.

기본값: 100

`maxConcurrentCloudRequests`

(선택 사항) 에 전송할 최대 동시 요청 수입니다. AWS 클라우드 이 수를 늘리면 많은 수의 클라이언트 장치를 연결하는 핵심 장치의 인증 성능을 향상시킬 수 있습니다.

기본값: 1

`certificateAuthority`

(선택 사항) 핵심 장치 중간 기관을 자체 중간 인증 기관으로 대체하기 위한 인증 기관 구성 옵션.

> **Note**
>
> 사용자 지정 인증 기관 (CA) 으로 Greengrass 코어 디바이스를 구성하고 동일한 CA를 사용하여 클라이언트 디바이스 인증서를 발급하는 경우 Greengrass는 클라이언트 디바이스 MQTT 작업에 대한 권한 부여 정책 검사를 우회합니다. 클라이언트 장치 인증 구성 요소는 사용하도록 구성된 CA에서 서명한 인증서를 사용하여 클라이언트를 완전히 신뢰합니다.
> 사용자 지정 CA를 사용할 때 이 동작을 제한하려면 다른 CA 또는 중간 CA를 사용하여 클라이언트 장치를 만들고 서명한 다음 올바른 중간 CA를 가리키도록 `certificateUri` 및 `certificateChainUri` 필드를 조정하십시오.

이 개체에는 다음 정보가 들어 있습니다.

인증서 (URI)

인증서의 위치. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서를 가리키는 URI일 수 있습니다.

`certificateChainUri`

코어 디바이스 CA의 인증서 체인 위치. 이는 루트 CA로 돌아가는 완전한 인증서 체인이어야 합니다. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 URI 일 수 있습니다.

`privateKeyUri`

코어 디바이스의 개인 키 위치. 이는 파일 시스템 URI 또는 하드웨어 보안 모듈에 저장된 인증서 개인 키를 가리키는 URI일 수 있습니다.

`security`

(선택 사항) 이 코어 디바이스의 보안 구성 옵션. 이 개체에는 다음 정보가 들어 있습니다.

`clientDeviceTrustDurationMinutes`

코어 장치를 사용하여 재인증해야 하기 전까지 클라이언트 장치의 인증 정보를 신뢰할 수 있는 기간 (분). 기본값은 1입니다.

`metrics`

(선택 사항) 이 코어 장치에 대한 메트릭 옵션. 오류 지표는 클라이언트 장치 인증에 오류가 있는 경우에만 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`disableMetrics`

`disableMetrics`필드가 로 `true` 설정된 경우 클라이언트 장치 인증은 지표를 수집하지 않습니다.

기본값: `false`

`aggregatePeriodSeconds`

집계 기간 (초) 은 클라이언트 장치 인증이 메트릭을 집계하여 원격 분석 에이전트로 보내는 빈도를 결정합니다. 원격 분석 에이전트가 여전히 하루에 한 번 메트릭을 게시하므로 메트릭이 게시되는 빈도는 변경되지 않습니다.

기본값: 3600

startupTimeoutSeconds

(선택 사항) 구성 요소를 시작하는 데 걸리는 최대 시간 (초). 이 제한 시간을 BROKEN 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: 120

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 MyClientDevice 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
```

```
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
```

```
}
```

v2.4.2 - v2.4.4

`deviceGroups`

장치 그룹은 코어 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`groupNameKey`

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *`groupNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의 구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구

문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (:) 를 사용하십시오. \\ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 `MyClientDevice1` 또는 인 클라이언트 장치와 `MyClientDevice2` 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 `MyClientDevice` 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`policyNameKey`

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *`policyNameKey`*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`statementNameKey`

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *`statementNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`operations`

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- `mqtt:connect`— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:clientId:`*`deviceClientId`*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *`deviceClientId`*ID로 바꾸십시오.
- `mqtt:publish`— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

- `mqtt:topic:mqttTopic`— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *MQTTtopic# ### ###* 바꾸십시오.

  이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.

- `mqtt:subscribe`— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topicfilter:mqttTopicFilter`— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *mqttTopicFilter*필터로 바꾸십시오.

    이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

    클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 `mqtt:topicfilter:client/+/status` 리소스를 구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

*와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:mqttTopic`) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

`certificates`

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`serverCertificateValiditySeconds`

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간 (초). 이 옵션을 구성하여 클라이언트 장치의 연결을 끊었다가 코어 장치에 다시 연결하는 빈도를 사용자 지정할 수 있습니다.

이 구성 요소는 만료되기 24시간 전에 로컬 MQTT 서버 인증서를 교체합니다. MQTT 브로커 (예: Moquette MQTT 브로커 구성 요소) 가 새 인증서를 생성하고 다시 시작합니다. 이 경우 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 잠시 후 클라이언트 장치를 코어 장치에 다시 연결할 수 있습니다.

기본값: `604800` (7일)

최소값: `172800` (2일)

최대치: `864000` (10일)

`performance`

(선택 사항) 이 코어 디바이스의 성능 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`maxActiveAuthTokens`

(선택 사항) 활성 클라이언트 장치 인증 토큰의 최대 수. 이 수를 늘리면 더 많은 수의 클라이언트 장치를 재인증하지 않고도 단일 코어 장치에 연결할 수 있습니다.

기본값: `2500`

`cloudRequestQueueSize`

(선택 사항) 이 구성 요소가 AWS 클라우드 요청을 거부하기 전에 대기열에 추가할 최대 요청 수입니다.

기본값: `100`

`maxConcurrentCloudRequests`

(선택 사항) 에 전송할 최대 동시 요청 수입니다. AWS 클라우드 이 수를 늘리면 많은 수의 클라이언트 장치를 연결하는 핵심 장치의 인증 성능을 향상시킬 수 있습니다.

기본값: `1`

`certificateAuthority`

(선택 사항) 핵심 장치 중간 기관을 자체 중간 인증 기관으로 대체하기 위한 인증 기관 구성 옵션.

> **Note**
>
> 사용자 지정 인증 기관 (CA) 으로 Greengrass 코어 디바이스를 구성하고 동일한 CA를 사용하여 클라이언트 디바이스 인증서를 발급하는 경우 Greengrass는 클라이언트 디바이스 MQTT 작업에 대한 권한 부여 정책 검사를 우회합니다. 클라이언트 장치 인증 구성 요소는 사용하도록 구성된 CA에서 서명한 인증서를 사용하여 클라이언트를 완전히 신뢰합니다.
> 사용자 지정 CA를 사용할 때 이 동작을 제한하려면 다른 CA 또는 중간 CA를 사용하여 클라이언트 장치를 만들고 서명한 다음 올바른 중간 CA를 가리키도록 `certificateUri` 및 `certificateChainUri` 필드를 조정하십시오.

이 개체에는 다음 정보가 들어 있습니다.

인증서 (URI)

인증서의 위치. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서를 가리키는 URI일 수 있습니다.

`certificateChainUri`

코어 디바이스 CA의 인증서 체인 위치. 이는 루트 CA로 돌아가는 완전한 인증서 체인이어야 합니다. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 URI일 수 있습니다.

`privateKeyUri`

코어 디바이스의 개인 키 위치. 이는 파일 시스템 URI 또는 하드웨어 보안 모듈에 저장된 인증서 개인 키를 가리키는 URI일 수 있습니다.

`security`

(선택 사항) 이 코어 디바이스의 보안 구성 옵션. 이 개체에는 다음 정보가 들어 있습니다.

`clientDeviceTrustDurationMinutes`

코어 장치를 사용하여 재인증해야 하기 전까지 클라이언트 장치의 인증 정보를 신뢰할 수 있는 기간 (분). 기본값은 1입니다.

`metrics`

(선택 사항) 이 코어 장치에 대한 메트릭 옵션. 오류 지표는 클라이언트 장치 인증에 오류가 있는 경우에만 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`disableMetrics`

`disableMetrics`필드가 로 `true` 설정된 경우 클라이언트 장치 인증은 지표를 수집하지 않습니다.

기본값: `false`

`aggregatePeriodSeconds`

집계 기간 (초) 은 클라이언트 장치 인증이 메트릭을 집계하여 원격 분석 에이전트로 보내는 빈도를 결정합니다. 원격 분석 에이전트가 여전히 하루에 한 번 메트릭을 게시하므로 메트릭이 게시되는 빈도는 변경되지 않습니다.

기본값: `3600`

startupTimeoutSeconds

(선택 사항) 구성 요소를 시작하는 데 걸리는 최대 시간 (초). 이 제한 시간을 `BROKEN` 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: `120`

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 `MyClientDevice` 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
```

```
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
```

```
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

`deviceGroups`

장치 그룹은 코어 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용

할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *groupNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의 구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (:) 를 사용하십시오. \\ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 MyClientDevice1 또는 인 클라이언트 장치와 MyClientDevice2 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 MyClientDevice 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`policyNameKey`

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *`policyNameKey`*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *statementNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- `mqtt:connect`— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:clientId:`*deviceClientId*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *deviceClientId*ID로 바꾸십시오.
- `mqtt:publish`— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topic:`*mqttTopic*— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *MQTTtopic# ### ###* 바꾸십시오.

    이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.
- `mqtt:subscribe`— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topicfilter:`*mqttTopicFilter*— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *mqttTopicFilter*필터로 바꾸십시오.

    이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

    클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 `mqtt:topicfilter:client/+/status` 리소스를

구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

*와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:`*`mqttTopic`*) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

`certificates`

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`serverCertificateValiditySeconds`

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간 (초). 이 옵션을 구성하여 클라이언트 장치의 연결을 끊었다가 코어 장치에 다시 연결하는 빈도를 사용자 지정할 수 있습니다.

이 구성 요소는 만료되기 24시간 전에 로컬 MQTT 서버 인증서를 교체합니다. MQTT 브로커 (예: Moquette MQTT 브로커 구성 요소) 가 새 인증서를 생성하고 다시 시작합니다. 이 경우 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 잠시 후 클라이언트 장치를 코어 장치에 다시 연결할 수 있습니다.

기본값: `604800` (7일)

최소값: `172800` (2일)

최대치: `864000` (10일)

`performance`

(선택 사항) 이 코어 디바이스의 성능 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`maxActiveAuthTokens`

(선택 사항) 활성 클라이언트 장치 인증 토큰의 최대 수. 이 수를 늘리면 더 많은 수의 클라이언트 장치를 재인증하지 않고도 단일 코어 장치에 연결할 수 있습니다.

기본값: `2500`

`cloudRequestQueueSize`

(선택 사항) 이 구성 요소가 AWS 클라우드 요청을 거부하기 전에 대기열에 추가할 최대 요청 수입니다.

기본값: `100`

`maxConcurrentCloudRequests`

(선택 사항) 에 전송할 최대 동시 요청 수입니다. AWS 클라우드 이 수를 늘리면 많은 수의 클라이언트 장치를 연결하는 핵심 장치의 인증 성능을 향상시킬 수 있습니다.

기본값: `1`

`certificateAuthority`

(선택 사항) 핵심 장치 중간 기관을 자체 중간 인증 기관으로 대체하기 위한 인증 기관 구성 옵션. 이 개체에는 다음 정보가 들어 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

인증서 (URI)

인증서의 위치. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서를 가리키는 URI일 수 있습니다.

`certificateChainUri`

코어 디바이스 CA의 인증서 체인 위치. 이는 루트 CA로 돌아가는 완전한 인증서 체인이어야 합니다. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 URI일 수 있습니다.

`privateKeyUri`

코어 디바이스의 개인 키 위치. 이는 파일 시스템 URI 또는 하드웨어 보안 모듈에 저장된 인증서 개인 키를 가리키는 URI일 수 있습니다.

`security`

(선택 사항) 이 코어 디바이스의 보안 구성 옵션. 이 개체에는 다음 정보가 들어 있습니다.

`clientDeviceTrustDurationMinutes`

코어 장치를 사용하여 재인증해야 하기 전까지 클라이언트 장치의 인증 정보를 신뢰할 수 있는 기간 (분). 기본값은 1입니다.

`metrics`

(선택 사항) 이 코어 장치에 대한 메트릭 옵션. 오류 지표는 클라이언트 장치 인증에 오류가 있는 경우에만 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`disableMetrics`

`disableMetrics`필드가 로 `true` 설정된 경우 클라이언트 장치 인증은 지표를 수집하지 않습니다.

기본값: `false`

`aggregatePeriodSeconds`

집계 기간 (초) 은 클라이언트 장치 인증이 메트릭을 집계하여 원격 분석 에이전트로 보내는 빈도를 결정합니다. 원격 분석 에이전트가 여전히 하루에 한 번 메트릭을 게시하므로 메트릭이 게시되는 빈도는 변경되지 않습니다.

기본값: `3600`

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 `MyClientDevice` 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
```

```
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
```

```
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.3.x

`deviceGroups`

장치 그룹은 코어 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용

할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *groupNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의 구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (:) 를 사용하십시오. \\ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 MyClientDevice1 또는 인 클라이언트 장치와 MyClientDevice2 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 MyClientDevice 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`policyNameKey`

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *`policyNameKey`*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *statementNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- mqtt:connect— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

  이 작업은 다음 리소스를 지원합니다.

  - mqtt:clientId:*deviceClientId*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *deviceClientId*ID로 바꾸십시오.
- mqtt:publish— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - mqtt:topic:*mqttTopic*— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *MQTTtopic# ### ###* 바꾸십시오.

    이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.
- mqtt:subscribe— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - mqtt:topicfilter:*mqttTopicFilter*— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *mqttTopicFilter*필터로 바꾸십시오.

    이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

    클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 mqtt:topicfilter:client/+/status 리소스를

구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

*와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:`*`mqttTopic`*) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

`certificates`

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`serverCertificateValiditySeconds`

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간 (초). 이 옵션을 구성하여 클라이언트 장치의 연결을 끊었다가 코어 장치에 다시 연결하는 빈도를 사용자 지정할 수 있습니다.

이 구성 요소는 만료되기 24시간 전에 로컬 MQTT 서버 인증서를 교체합니다. MQTT 브로커 (예: Moquette MQTT 브로커 구성 요소) 가 새 인증서를 생성하고 다시 시작합니다. 이 경우 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 잠시 후 클라이언트 장치를 코어 장치에 다시 연결할 수 있습니다.

기본값: `604800` (7일)

최소값: `172800` (2일)

최대치: `864000` (10일)

`performance`

(선택 사항) 이 코어 디바이스의 성능 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`maxActiveAuthTokens`

(선택 사항) 활성 클라이언트 장치 인증 토큰의 최대 수. 이 수를 늘리면 더 많은 수의 클라이언트 장치를 재인증하지 않고도 단일 코어 장치에 연결할 수 있습니다.

기본값: `2500`

`cloudRequestQueueSize`

(선택 사항) 이 구성 요소가 AWS 클라우드 요청을 거부하기 전에 대기열에 추가할 최대 요청 수입니다.

기본값: `100`

`maxConcurrentCloudRequests`

(선택 사항) 에 전송할 최대 동시 요청 수입니다. AWS 클라우드 이 수를 늘리면 많은 수의 클라이언트 장치를 연결하는 핵심 장치의 인증 성능을 향상시킬 수 있습니다.

기본값: `1`

`certificateAuthority`

(선택 사항) 핵심 장치 중간 기관을 자체 중간 인증 기관으로 대체하기 위한 인증 기관 구성 옵션. 이 개체에는 다음 정보가 들어 있습니다.

인증서 (URI)

인증서의 위치. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서를 가리키는 URI일 수 있습니다.

`certificateChainUri`

코어 디바이스 CA의 인증서 체인 위치. 이는 루트 CA로 돌아가는 완전한 인증서 체인이어야 합니다. 파일 시스템 URI이거나 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 URI일 수 있습니다.

`privateKeyUri`

코어 디바이스의 개인 키 위치. 이는 파일 시스템 URI 또는 하드웨어 보안 모듈에 저장된 인증서 개인 키를 가리키는 URI일 수 있습니다.

`security`

(선택 사항) 이 코어 디바이스의 보안 구성 옵션. 이 개체에는 다음 정보가 들어 있습니다.

`clientDeviceTrustDurationMinutes`

코어 장치를 사용하여 재인증해야 하기 전까지 클라이언트 장치의 인증 정보를 신뢰할 수 있는 기간 (분). 기본값은 1입니다.

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 `MyClientDevice` 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
```

```
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.2.x

`deviceGroups`

장치 그룹은 코어 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`groupNameKey`

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *`groupNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의 구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구

문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (:) 를 사용하십시오. \\ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 `MyClientDevice1` 또는 인 클라이언트 장치와 `MyClientDevice2` 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 `MyClientDevice` 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`policyNameKey`

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *`policyNameKey`*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`statementNameKey`

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *`statementNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`operations`

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- `mqtt:connect`— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:clientId:`*`deviceClientId`*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *`deviceClientId`*ID로 바꾸십시오.
- `mqtt:publish`— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

- `mqtt:topic:`*`mqttTopic`*— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *`MQTTtopic# ### ###`* 바꾸십시오.

  이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.

- `mqtt:subscribe`— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *`mqttTopicFilter`*필터로 바꾸십시오.

    이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

    클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 `mqtt:topicfilter:client/+/status` 리소스를 구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

*와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:`*`mqttTopic`*) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

`certificates`

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`serverCertificateValiditySeconds`

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간 (초). 이 옵션을 구성하여 클라이언트 장치의 연결을 끊었다가 코어 장치에 다시 연결하는 빈도를 사용자 지정할 수 있습니다.

이 구성 요소는 만료되기 24시간 전에 로컬 MQTT 서버 인증서를 교체합니다. MQTT 브로커 (예: Moquette MQTT 브로커 구성 요소) 가 새 인증서를 생성하고 다시 시작합니다. 이 경우 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 잠시 후 클라이언트 장치를 코어 장치에 다시 연결할 수 있습니다.

기본값: `604800` (7일)

최소값: `172800` (2일)

최대치: `864000` (10일)

`performance`

(선택 사항) 이 코어 디바이스의 성능 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`maxActiveAuthTokens`

(선택 사항) 활성 클라이언트 장치 인증 토큰의 최대 수. 이 수를 늘리면 더 많은 수의 클라이언트 장치를 재인증하지 않고도 단일 코어 장치에 연결할 수 있습니다.

기본값: `2500`

`cloudRequestQueueSize`

(선택 사항) 이 구성 요소가 AWS 클라우드 요청을 거부하기 전에 대기열에 추가할 최대 요청 수입니다.

기본값: `100`

`maxConcurrentCloudRequests`

(선택 사항) 에 전송할 최대 동시 요청 수입니다. AWS 클라우드 이 수를 늘리면 많은 수의 클라이언트 장치를 연결하는 핵심 장치의 인증 성능을 향상시킬 수 있습니다.

기본값: `1`

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 `MyClientDevice` 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
```

```
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.1.x

`deviceGroups`

장치 그룹은 코어 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`groupNameKey`

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *`groupNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의 구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (:) 를 사용하십시오. \\ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 `MyClientDevice1` 또는 인 클라이언트 장치와 `MyClientDevice2` 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 `MyClientDevice` 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *policyNameKey*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *statementNameKey*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`operations`

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- `mqtt:connect`— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:clientId:`*deviceClientId*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *deviceClientId*ID로 바꾸십시오.
- `mqtt:publish`— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topic:`*mqttTopic*— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *MQTTtopic# ### ###* 바꾸십시오.

    이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.

- `mqtt:subscribe`— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *`mqttTopicFilter`*필터로 바꾸십시오.

    이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

    클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 `mqtt:topicfilter:client/+/status` 리소스를 구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

  *와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:`*`mqttTopic`*) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

`certificates`

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`serverCertificateValiditySeconds`

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간 (초). 이 옵션을 구성하여 클라이언트 장치의 연결을 끊었다가 코어 장치에 다시 연결하는 빈도를 사용자 지정할 수 있습니다.

이 구성 요소는 만료되기 24시간 전에 로컬 MQTT 서버 인증서를 교체합니다. MQTT 브로커(예: Moquette MQTT 브로커 구성 요소)가 새 인증서를 생성하고 다시 시작합니다. 이 경우 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 잠시 후 클라이언트 장치를 코어 장치에 다시 연결할 수 있습니다.

기본값: `604800` (7일)

최소값: `172800` (2일)

최대치: `864000` (10일)

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 `MyClientDevice` 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
```

```
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
```

```
      }
    }
  }
}
```

v2.0.x

`deviceGroups`

장치 그룹은 코어 장치에 연결하고 통신할 수 있는 권한이 있는 클라이언트 장치 그룹입니다. 선택 규칙을 사용하여 클라이언트 장치 그룹을 식별하고 각 장치 그룹에 대한 권한을 지정하는 클라이언트 장치 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`formatVersion`

이 구성 개체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- `2021-03-05`

`definitions`

이 코어 디바이스의 디바이스 그룹. 각 정의는 클라이언트 장치가 그룹의 구성원인지 평가하는 선택 규칙을 지정합니다. 또한 각 정의는 선택 규칙과 일치하는 클라이언트 장치에 적용할 권한 정책을 지정합니다. 클라이언트 장치가 여러 장치 그룹의 구성원인 경우 장치 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`groupNameKey`

이 장치 그룹의 이름. 이 장치 그룹을 식별하는 데 도움이 되는 *`groupNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`selectionRule`

이 장치 그룹의 구성원인 클라이언트 장치를 지정하는 쿼리입니다. 클라이언트 장치가 연결되면 코어 장치는 이 선택 규칙을 평가하여 클라이언트 장치가 이 장치 그룹의

구성원인지 확인합니다. 클라이언트 장치가 구성원인 경우 코어 장치는 이 장치 그룹의 정책을 사용하여 클라이언트 장치의 동작을 승인합니다.

각 선택 규칙은 클라이언트 장치와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 조항으로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 플릿 인덱싱 쿼리 구문을 참조하십시오.

*와일드카드를 사용하면 하나의 선택 규칙 조항으로 여러 클라이언트 장치를 일치시킬 수 있습니다. 사물 이름 끝에 이 와일드카드를 사용하여 이름이 지정한 문자열로 시작하는 클라이언트 장치를 일치시킬 수 있습니다. 이 와일드카드를 사용하여 모든 클라이언트 디바이스를 일치시킬 수도 있습니다.

> **Note**
>
> 콜론 문자 () 가 포함된 값을 선택하려면 콜론을 이스케이프하여 백슬래시 문자 (:) 를 사용하십시오. \\ JSON과 같은 형식에서는 백슬래시 문자를 이스케이프 처리해야 하므로 콜론 문자 앞에 백슬래시 문자 두 개를 입력합니다. 예를 들어, 이름이 다음과 같은 항목을 `thingName: MyTeam\\\\:ClientDevice1` 선택하도록 지정합니다. `MyTeam:ClientDevice1`

다음 선택기를 지정할 수 있습니다.

- `thingName`— 클라이언트 장치 사물의 이름AWS IoT.

Example 선택 규칙 예시

다음 선택 규칙은 이름이 `MyClientDevice1` 또는 인 클라이언트 장치와 `MyClientDevice2` 일치합니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 선택 규칙 예시 (와일드카드 사용)

다음 선택 규칙은 이름이 로 시작하는 클라이언트 디바이스와 `MyClientDevice` 일치합니다.

```
thingName: MyClientDevice*
```

Example 선택 규칙 예시 (모든 장치 매칭)

다음 선택 규칙은 모든 클라이언트 장치와 일치합니다.

```
thingName: *
```

`policyName`

이 장치 그룹의 클라이언트 장치에 적용되는 권한 정책. `policies`개체에 정의한 정책 이름을 지정합니다.

`policies`

코어 장치에 연결하는 클라이언트 장치에 대한 클라이언트 장치 권한 부여 정책. 각 권한 부여 정책은 클라이언트 장치가 이러한 작업을 수행할 수 있는 일련의 작업과 리소스를 지정합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`policyNameKey`

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 *`policyNameKey`*이름으로 바꾸십시오. 이 정책 이름을 사용하여 장치 그룹에 적용할 정책을 정의할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`statementNameKey`

이 정책 설명의 이름. 이 정책 설명을 식별하는 데 도움이 되는 *`statementNameKey`*이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`operations`

이 정책에서 리소스를 허용하기 위한 작업 목록.

다음 작업 중 하나를 포함할 수 있습니다.

- `mqtt:connect`— 코어 디바이스에 연결할 수 있는 권한을 부여합니다. 클라이언트 장치에 코어 장치에 연결하려면 이 권한이 있어야 합니다.

이 작업은 다음 리소스를 지원합니다.

- `mqtt:clientId:`*`deviceClientId`*— 클라이언트 장치가 코어 장치의 MQTT 브로커에 연결하는 데 사용하는 클라이언트 ID를 기반으로 액세스를 제한합니다. 사용할 클라이언트 *`deviceClientId`*ID로 바꾸십시오.

- `mqtt:publish`— MQTT 메시지를 주제에 게시할 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topic:`*`mqttTopic`*— 클라이언트 장치가 메시지를 게시하는 MQTT 주제에 따라 액세스를 제한합니다. *`MQTTtopic# ### ###`* 바꾸십시오.

    이 리소스는 MQTT 주제 와일드카드를 지원하지 않습니다.

- `mqtt:subscribe`— MQTT 주제 필터를 구독하여 메시지를 받을 수 있는 권한을 부여합니다.

  이 작업은 다음 리소스를 지원합니다.

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 클라이언트 장치가 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 *`mqttTopicFilter`*필터로 바꾸십시오.

    이 리소스는 + 및 # MQTT 주제 와일드카드를 지원합니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

    클라이언트 장치는 허용한 정확한 주제 필터를 구독할 수 있습니다. 예를 들어 클라이언트 장치가 `mqtt:topicfilter:client/+/status` 리소스를 구독하도록 허용하면 클라이언트 장치는 구독할 수 `client/+/status` 있지만 구독할 수는 없습니다`client/client1/status`.

*와일드카드를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

`resources`

이 정책에서 작업을 허용할 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정하십시오. 예를 들어 `mqtt:publish` 작업을 지정하는 정책에 MQTT 주제 리소스 목록 (`mqtt:topic:`*`mqttTopic`*) 을 지정할 수 있습니다.

*와일드카드를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. *와일드카드를 사용하여 부분 리소스 식별자를 일치시킬 수는 없습니다. 예를 들어 지정할 수는 있지만 지정할 **`"resources": "*"`** 수는 없습니다. **`"resources": "mqtt:clientId:*"`**

`statementDescription`

(선택 사항) 이 정책 설명에 대한 설명.

Example 예: 구성 병합 업데이트 (제한적 정책 사용)

다음 예제 구성에서는 이름이 로 시작하는 클라이언트 장치가 모든 주제에 대해 연결 및 `MyClientDevice` 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
```

```
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (허용 정책 사용)

다음 예제 구성은 모든 클라이언트 장치가 모든 주제에 대해 연결 및 게시/구독할 수 있도록 지정합니다.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.4.5	새로운 기능 매개변수로 사물 이름을 선택하기 위한 와일드카드 접두사 지원을 추가합니다. `selectionRule`

버전	변경
	버그 수정 및 개선 특정 경우에 인증서가 새 연결 정보로 업데이트되지 않는 문제를 수정합니다.
2.4.4	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.4.3	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.4.2	새로운 기능 새 구성 옵션을 추가합니다. `startupTimeoutSeconds`
2.4.1	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.4.0	새로운 기능 • 원격 분석 에이전트가 게시할 운영 지표를 내보내는 클라이언트 장치 인증 지원을 추가합니다. 버그 수정 및 개선 • 클라이언트 장치 인증이 클라이언트 장치의 ID를 확인하는 데 10초 이상 걸리는 문제를 수정합니다. • 추가 사소한 수정 및 개선
2.3.2	버그 수정 및 개선 • 구성 요소가 오프라인일 때 다시 시작할 때 인증서 주체를 올바르게 생성하도록 호스트 이름 정보를 캐싱하는 지원을 추가합니다.
2.3.1	버그 수정 및 개선 • 메모리 누수를 수정합니다.

버전	변경
2.3.0	**Warning** 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 새로운 기능 • 코어 장치가 인터넷에 연결되어 있지 않아도 핵심 장치에 계속 연결할 수 있도록 클라이언트 장치의 오프라인 인증 지원을 추가합니다. • 코어 디바이스가 루트 인증서로 사용하여 MQTT 브로커 인증서를 생성하는 고객 제공 인증 기관에 대한 지원을 추가합니다.
2.2.3	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.2	버그 수정 및 개선 • 특정 시나리오에서 로컬 MQTT 서버 인증서가 의도한 것보다 더 자주 교체되는 문제를 수정합니다.
2.2.1	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.0	새로운 기능 • IPC (프로세스 간 통신) 작업을 호출하여 클라이언트 장치를 인증하고 권한을 부여하는 사용자 지정 구성 요소에 대한 지원을 추가합니다. 예를 들어 사용자 지정 MQTT 브로커 구성 요소에서 이러한 작업을 사용할 수 있습니다. 자세한 내용은 IPC: 클라이언트 장치 인증 및 권한 부여를 참조하십시오. • 이 구성 요소의 작동 방식을 조정하기 위해 구성할 수 있는 `maxActiveAuthTokens` `cloudQueueSize` , 및 `threadPoolSize` 옵션을 추가합니다.

버전	변경
2.1.0	새로운 기능 • MQTT Broker 서버 인증서 만료 시기를 사용자 지정하도록 구성할 수 있는 `serverCertificateValiditySeconds` 옵션을 추가합니다. 2~10일 후에 만료되도록 서버 인증서를 구성할 수 있습니다. 버그 수정 및 개선 • 이 구성 요소가 구성 재설정 업데이트를 처리하는 방식과 관련된 문제를 수정합니다. • 특정 시나리오에서 로컬 MQTT 서버 인증서가 의도한 것보다 더 자주 교체되는 문제를 수정합니다. 이 수정 사항을 적용하려면 Moquette MQTT 브로커 구성 요소 v2.1.0 이상도 사용해야 합니다. • 이 구성 요소가 인증서를 교체할 때 기록하는 메시지를 개선합니다. • Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.3	버그 수정 및 개선 • 이제 코어 디바이스의 개인 키를 교체하면 자격 증명이 새로 고쳐집니다. • 로그 메시지를 더 명확하게 표시하도록 업데이트되었습니다.
2.0.2	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.1	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.0	초기 버전

CloudWatch 측정 항목

Amazon CloudWatch 지표 구성 요소 (`aws.greengrass.Cloudwatch`) 는 Greengrass 코어 디바이스의 사용자 지정 지표를 Amazon에 게시합니다. CloudWatch 구성 요소를 통해 구성 요소는 CloudWatch 지표를 게시할 수 있으며, 이를 사용하여 Greengrass 코어 장치 환경을 모니터링하고 분

석할 수 있습니다. 자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 측정치 사용을 참조하십시오.

이 구성 요소가 포함된 CloudWatch 지표를 게시하려면 이 구성 요소가 구독하는 주제에 메시지를 게시하십시오. 기본적으로 이 구성 요소는 `cloudwatch/metric/put` 로컬 게시/구독 주제를 구독합니다. 이 구성 요소를 배포할 때 AWS IoT Core MQTT 주제를 비롯한 다른 주제를 지정할 수 있습니다.

이 구성 요소는 동일한 네임스페이스에 있는 메트릭을 일괄 처리하여 정기적으로 게시합니다. CloudWatch

> **Note**
>
> 이 구성 요소는 V1의 CloudWatch 메트릭 커넥터와 유사한 기능을 제공합니다. AWS IoT Greengrass 자세한 내용은 AWS IoT Greengrass V1 개발자 CloudWatch 안내서의 메트릭 커넥터를 참조하십시오.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

각 구성 요소 버전의 변경 사항에 대한 자세한 내용은 변경 로그를 참조하십시오.

유형

v3.x

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

v2.x

이 구성 요소는 Lambda 구성 요소 () 입니다. `aws.greengrass.lambda` Greengrass 핵은 Lambda 런처 구성 요소를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

v3.x

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

v2.x

이 구성 요소는 Linux 코어 장치에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

3.x

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Greengrass 장치 역할은 다음 예시 IAM 정책에 나와 있는 것처럼 `cloudwatch:PutMetricData` 작업을 허용해야 합니다.

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

자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 권한 참조를 참조하십시오.

2.x

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Greengrass 장치 역할은 다음 예시 IAM 정책에 나와 있는 것처럼 `cloudwatch:PutMetricData` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
```

```
    }
  ]
}
```

자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 권한 참조를 참조하십시오.

- 이 구성 요소로부터 출력 데이터를 받으려면 이 구성 요소를 배포할 때 기존 구독 라우터 구성 요소 (aws.greengrass.LegacySubscriptionRouter) 에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성은 이 구성 요소가 응답을 게시하는 주제를 지정합니다.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-cloudwatch": {
        "id": "aws-greengrass-cloudwatch",
        "source": "component:aws.greengrass.Cloudwatch",
        "subject": "cloudwatch/metric/put/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-cloudwatch": {
        "id": "aws-greengrass-cloudwatch",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
  cloudwatch:version",
        "subject": "cloudwatch/metric/put/status",
        "target": "cloud"
      }
    }
  }
  ```

  - *### AWS ## #### ####* 바꾸십시오.
  - *###* 이 구성 요소가 실행하는 Lambda 함수 버전으로 교체하십시오. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소 버전의 레시피를 확인해야 합니다. AWS IoT

Greengrass 콘솔에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 쌍을 찾으십시오. 이 키-값 쌍에는 Lambda 함수의 이름과 버전이 들어 있습니다.

> **Important**
>
> 이 구성 요소를 배포할 때마다 레거시 구독 라우터에서 Lambda 함수 버전을 업데이트해야 합니다. 이렇게 하면 배포하는 구성 요소 버전에 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 설명은 배포 만들기 섹션을 참조하세요.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`monitoring. region .amazonaws.com`	443	예	지표 업로드. CloudWatch

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

3.0.0 - 3.1.0

다음 표에는 이 구성 요소의 버전 3.0.0~3.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <3.0.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.2 and 2.1.3

다음 표에는 이 구성 요소의 버전 2.1.2 및 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.8 - 2.1.0

다음 표에는 이 구성 요소의 버전 2.0.8~2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	하드
람다 런처	>=1.0.0	하드
Lambda 런타임	>=1.0.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	>=1.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

v3.x

`PublishInterval`

(선택 사항) 구성 요소가 지정된 네임스페이스에 대해 일괄 처리된 메트릭을 게시할 때까지 기다려야 하는 최대 시간 (초). 메트릭을 수신할 때 (즉, 일괄 처리 없이) 게시하도록 구성 요소를 구성하려면 다음을 지정하십시오. `0`

구성 요소는 동일한 네임스페이스에서 20개의 지표를 수신한 후 또는 지정한 간격 후에 게시합니다. CloudWatch

> **Note**
> 구성 요소는 이벤트가 게시되는 순서를 지정하지 않습니다.

이 값은 최대 900초일 수 있습니다.

기본값: 10초

`MaxMetricsToRetain`

(선택 사항) 구성 요소가 새 메트릭으로 교체하기 전에 메모리에 저장할 모든 네임스페이스의 최대 메트릭 수입니다.

이 제한은 코어 기기가 인터넷에 연결되어 있지 않은 경우에 적용되므로 구성 요소가 나중에 게시하기 위해 메트릭을 버퍼링합니다. 버퍼가 가득 차면 구성 요소는 가장 오래된 메트릭을 최신 메트릭으로 바꿉니다. 지정된 네임스페이스의 메트릭은 동일한 네임스페이스의 메트릭만 대체합니다.

> **Note**
>
> 구성 요소의 호스트 프로세스가 중단되는 경우 구성 요소는 메트릭을 저장하지 않습니다. 예를 들어 배포 중이나 코어 디바이스가 다시 시작될 때 이런 일이 발생할 수 있습니다.

이 값은 최소 2,000개 지표여야 합니다.

기본값: 5,000개 지표

`InputTopic`

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. 를 지정하는 경우 이 `true` `PubSubToIoTCore` 항목에서 MQTT 와일드카드 (+ 및 #) 를 사용할 수 있습니다.

기본값: `cloudwatch/metric/put`

`OutputTopic`

(선택 사항) 구성 요소가 상태 응답을 게시하는 주제입니다.

기본값: `cloudwatch/metric/put/status`

`PubSubToIoTCore`

(선택 사항) AWS IoT Core MQTT 주제를 게시하고 구독할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

기본값: `false`

`UseInstaller`

(선택 사항) 이 구성 요소의 SDK 종속성을 설치하기 위해 이 구성 요소의 설치 프로그램 스크립트를 사용할지 여부를 정의하는 부울 값입니다.

사용자 지정 스크립트를 사용하여 종속성을 설치하거나 미리 빌드된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 로 설정합니다. `false` 이 구성 요소를 사용하려면 종속성을 포함하여 다음 라이브러리를 설치하고 기본 Greengrass 시스템 사용자가 사용할 수 있도록 해야 합니다.

- AWS IoT Device SDK Python용 v2
- AWS SDK for Python (Boto3)

기본값: `true`

`PublishRegion`

(선택 사항) CloudWatch 메트릭을 게시할 대상. AWS 리전 이 값은 코어 장치의 기본 지역보다 우선합니다. 이 매개변수는 지역 간 지표에만 필요합니다.

`accessControl`

(선택 사항) 구성 요소가 지정된 주제를 게시하고 구독할 수 있도록 허용하는 권한 부여 정책이 포함된 객체입니다. `InputTopic`및 `OutputTopic` 에 대해 사용자 지정 값을 지정하는 경우 이 개체의 리소스 값을 업데이트해야 합니다.

기본값:

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:pubsub:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
```

```
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  }
}
```

Example 예: 구성 병합 업데이트

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

> **Note**
>
> 이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함됩니다. 디바이스에서 이 구성 요소를 구성하려면 다음 파라미터만 편집하는 것이 좋습니다.

`lambdaParams`

이 구성 요소의 Lambda 함수에 대한 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`EnvironmentVariables`

Lambda 함수의 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`PUBLISH_INTERVAL`

(선택 사항) 구성 요소가 지정된 네임스페이스에 대한 일괄 측정치를 게시할 때까지 기다려야 하는 최대 시간 (초). 메트릭을 수신할 때 (즉, 일괄 처리 없이) 게시하도록 구성 요소를 구성하려면 다음을 지정하십시오. `0`

구성 요소는 동일한 네임스페이스에서 20개의 지표를 수신한 후 또는 지정한 간격 후에 게시합니다. CloudWatch

> **Note**
>
> 구성 요소는 이벤트가 게시되는 순서를 보장하지 않습니다.

이 값은 최대 900초일 수 있습니다.

기본값: 10초

`MAX_METRICS_TO_RETAIN`

(선택 사항) 구성 요소가 새 메트릭으로 교체하기 전에 메모리에 저장할 모든 네임스페이스의 최대 메트릭 수입니다.

이 제한은 코어 기기가 인터넷에 연결되어 있지 않은 경우에 적용되므로 구성 요소가 나중에 게시하기 위해 메트릭을 버퍼링합니다. 버퍼가 가득 차면 구성 요소는 가장 오래된 메트릭을 최신 메트릭으로 바꿉니다. 지정된 네임스페이스의 메트릭은 동일한 네임스페이스의 메트릭만 대체합니다.

> **Note**
>
> 구성 요소의 호스트 프로세스가 중단되는 경우 구성 요소는 메트릭을 저장하지 않습니다. 예를 들어 배포 중이나 코어 디바이스가 다시 시작될 때 이런 일이 발생할 수 있습니다.

이 값은 최소 2,000개 지표여야 합니다.

기본값: 5,000개 지표

`PUBLISH_REGION`

(선택 사항) CloudWatch 지표를 AWS 리전 게시할 대상. 이 값은 코어 장치의 기본 지역보다 우선합니다. 이 매개변수는 지역 간 지표에만 필요합니다.

`containerMode`

(선택 사항) 이 구성요소의 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- `NoContainer`— 구성 요소는 격리된 런타임 환경에서 실행되지 않습니다.

- `GreengrassContainer`— 구성 요소는 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소의 컨테이너 매개 변수를 포함하는 개체입니다. 를 `GreengrassContainer` 지정하면 구성 요소에서 이러한 매개 변수를 사용합니다`containerMode`.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당할 메모리 양 (KB).

기본값은 64MB (65,535KB) 입니다.

`pubsubTopics`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제를 포함하는 객체입니다. 각 주제를 지정하고 구성 요소가 MQTT 주제를 구독하는지 AWS IoT Core 아니면 로컬 게시/구독 주제의 MQTT 주제를 구독할지를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열 형식의 배열 인덱스입니다.

다음 정보가 포함된 객체:

`type`

(선택 사항) 이 구성 요소가 메시지를 구독하는 데 사용하는 게시/구독 메시지의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `PUB_SUB` – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- `IOT_CORE`— AWS IoT Core MQTT 메시지를 구독하십시오. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 있습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본값: `PUB_SUB`

`topic`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제입니다. 를 지정하는 경우 이 `IotCore` `type` 항목에서 MQTT 와일드카드 (+및#) 를 사용할 수 있습니다.

Example 예: 구성 병합 업데이트 (컨테이너 모드)

```
{
  "containerMode": "GreengrassContainer"
}
```

Example 예: 구성 병합 업데이트 (컨테이너 모드 없음)

```
{
  "containerMode": "NoContainer"
}
```

입력 데이터

이 구성 요소는 다음 주제에 대한 지표를 수락하고 CloudWatch 지표를 게시합니다. 기본적으로 이 구성 요소는 로컬 게시/구독 메시지를 구독합니다. 사용자 정의 구성 요소에서 이 구성 요소에 메시지를 게시하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독

구성 요소 버전 v3.0.0부터 구성 매개 변수를 로 설정하여 MQTT 주제를 구독하도록 이 구성 요소를 구성할 수도 있습니다. `PubSubToIoTCore` `true` 사용자 지정 구성 요소의 MQTT 주제에 메시지를 게시하는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본 주제: `cloudwatch/metric/put`

메시지는 다음 속성을 허용합니다. 입력 메시지는 JSON 형식이어야 합니다.

`request`

이 메시지의 메트릭입니다.

요청 객체에는 CloudWatch에 게시할 지표 데이터가 포함되어 있습니다. 지표 값은 `PutMetricData`작업 사양을 충족해야 합니다.

다음 정보가 `object` 포함된 유형:

`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.

유형: `string`

유효한 패턴: `[^:].*`

`metricData`

지표에 대한 데이터.

다음 정보가 `object` 포함된 유형:

`metricName`

지표의 이름.

유형: `string`

`value`

지표에 대한 값.

> **Note**
>
> CloudWatch 너무 작거나 너무 큰 값은 거부합니다. 값은 ~ `1.174271e+108` (밑수 10) 또는 `2e-360` ~ (기수 2) `8.515920e-109` 사이여야 합니다. `2e360` CloudWatch `NaN`+`Infinity`, 및 같은 특수 값은 지원하지 않습니다`-Infinity`.

유형: `double`

`dimensions`

(선택 사항) 지표의 크기. 측정기준은 측정항목 및 해당 데이터에 대한 추가 정보를 제공합니다. 지표는 최대 10개 차원을 정의할 수 있습니다.

이 구성 요소에는 이름이 지정된 차원이 자동으로 포함되며coreName, 여기서 값은 코어 장치의 이름입니다.

유형: array 각각 다음 정보가 들어 있는 개체 유형:

name

(선택 사항) 차원 이름.

유형: string

value

(선택 사항) 차원 값입니다.

유형: string

timestamp

(선택 사항) 지표 데이터가 수신된 시간으로, Unix epoch 시간을 기준으로 초 단위로 표시됩니다.

구성 요소가 메시지를 수신하는 시간이 기본값입니다.

유형: double

> **Note**
>
> 이 구성 요소를 버전 2.0.3에서 2.0.7 사이에서 사용하는 경우 단일 소스에서 여러 지표를 전송할 때 각 지표에 대해 개별적으로 타임스탬프를 검색하는 것이 좋습니다. 변수를 사용하여 타임스탬프를 저장하지 마세요.

unit

(선택 사항) 지표의 단위.

유형: string

유효한 값: SecondsMicroseconds, MillisecondsBytes,Kilobytes,Megabytes, GigabytesTerabytes,Bits,Kilobits, MegabitsGigabits,Terabits,Percent, CountBytes/Second,Kilobytes/Second,Megabytes/Second, Gigabytes/SecondTerabytes/Second,Bits/Second,Kilobits/Second, Megabits/SecondGigabits/Second,Terabits/Second,Count/Second, None

기본값은 None입니다.

> **Note**
>
> CloudWatch PutMetricDataAPI에 적용되는 모든 할당량은 이 구성 요소를 사용하여 게시한 지표에 적용됩니다. 다음과 같은 할당량이 특히 중요합니다.
>
> - API 페이로드의 40KB 제한
> - API 요청당 20개 지표
> - PutMetricData API에 대한 150개의 초당 트랜잭션(TPS)
>
> 자세한 내용은 사용 CloudWatch 설명서의 서비스 할당량을 참조하십시오. CloudWatch

Example 입력 예

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
  }
}
```

출력 데이터

이 구성 요소는 기본적으로 응답을 다음 로컬 게시/구독 주제에 출력 데이터로 게시합니다. 사용자 지정 구성 요소에서 이 주제에 대한 메시지를 구독하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독

구성 매개 변수를 로 설정하여 MQTT 주제에 게시하도록 이 PubSubToIoTCore 구성 요소를 구성할 수도 있습니다. true 사용자 지정 구성 요소의 MQTT 주제에 대한 메시지 구독에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

> **Note**
>
> 구성 요소 버전 2.0.x는 기본적으로 응답을 MQTT 주제에 대한 출력 데이터로 게시합니다. 주제를 레거시 구독 라우터 구성 요소의 subject 구성에서와 같이 지정해야 합니다.

기본 주제: cloudwatch/metric/put/status

Example 출력 예: 성공

응답에는 메트릭 데이터의 네임스페이스와 응답의 RequestId 필드가 포함됩니다. CloudWatch

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

Example 출력 예: 실패

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

> **Note**
>
> 구성 요소가 연결 오류와 같이 재시도할 수 있는 오류를 감지하면 다음 배치에서 게시를 다시 시도합니다.

라이선스

이 구성 요소에는 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 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Cloudwatch.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

v3.x

버전	변경
3.1.0	버그 수정 및 개선 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.
3.0.0	이 버전의 CloudWatch 메트릭 구성 요소에는 버전 2.x와 다른 구성 매개 변수가 필요합니다. 버전 2.x에 대해 기본이 아닌 구성을 사용하고 v2.x에서 v3.x로 업그레이드하려면 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 메트릭 구성 요소 구성을 참조하십시오. CloudWatch 새로운 기능 • Windows를 실행하는 핵심 장치에 대한 지원을 추가합니다. • 구성 요소 유형을 Lambda 구성 요소에서 일반 구성 요소로 변경합니다. 이제 이 구성 요소는 구독을 생성할 때 더 이상 기존 구독 라우터 구성 요소에 의존하지 않습니다. • `InputTopic` 구성 요소가 메시지를 수신하기 위해 구독하는 주제를 지정하는 새 구성 매개 변수를 추가합니다. • `OutputTopic` 구성 요소가 상태 응답을 게시하는 주제를 지정하는 새 구성 매개 변수를 추가합니다. • AWS IoT Core MQTT 주제를 게시하고 구독할지 여부를 지정하는 새 `PubSubToIoTCore` 구성 매개 변수를 추가합니다. • `UseInstaller` 구성 요소 종속성을 설치하는 설치 스크립트를 선택적으로 비활성화할 수 있는 새 구성 매개 변수를 추가합니다.

버전	변경
	버그 수정 및 개선 입력 데이터의 중복 타임스탬프에 대한 지원을 추가합니다.

v2.x

버전	변경
2.1.3	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.
2.0.8	버그 수정 및 개선 • 입력 데이터의 중복 타임스탬프에 대한 지원을 추가합니다. • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.0.4	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

다음 사항도 참조하십시오.

- Amazon 사용 CloudWatch 설명서에서 Amazon CloudWatch 지표 사용
- PutMetricData아마존 CloudWatch API 레퍼런스에서

AWS IoT Device Defender

AWS IoT Device Defender 구성 요소 (`aws.greengrass.DeviceDefender`) 는 관리자에게 Greengrass 코어 장치의 상태 변경 사항을 알립니다. 그러면 손상된 디바이스를 나타낼 수 있는 비정상적인 동작을 식별할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 AWS IoT Device Defender 섹션을 참조하십시오.

이 구성 요소는 코어 장치에서 시스템 메트릭을 읽습니다. 그런 다음 메트릭을 에 게시합니다. AWS IoT Device Defender 이 구성 요소가 보고하는 메트릭을 읽고 해석하는 방법에 대한 자세한 내용은 AWS IoT Core 개발자 가이드의 기기 메트릭 문서 사양을 참조하십시오.

> **Note**
>
> 이 구성 요소는 의 Device Defender 커넥터와 유사한 기능을 제공합니다. AWS IoT Greengrass V1 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 장치 디펜더 커넥터를 참조하십시오.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성

- 구성
- 입력 데이터
- 출력 데이터
- 로컬 로그 파일
- 라이선스
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 3.1.x
- 3.0.x
- 2.0.x

각 구성 요소 버전의 변경 사항에 대한 자세한 내용은 변경 로그를 참조하십시오.

유형

v3.x

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

v2.x

이 구성 요소는 Lambda 구성 요소 () 입니다. `aws.greengrass.lambda` Greengrass 핵은 Lambda 런처 구성 요소를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

v3.x

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux

- Windows

v2.x

이 구성 요소는 Linux 코어 장치에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

v3.x

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- AWS IoT Device Defender 탐지 기능을 사용하여 위반을 모니터링하도록 구성되었습니다. 자세한 정보는 AWS IoT Core 개발자 안내서의 감지 사용 방법을 참조하세요.

v2.x

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- AWS IoT Device Defender 탐지 기능을 사용하여 위반을 모니터링하도록 구성되었습니다. 자세한 정보는 AWS IoT Core 개발자 안내서의 감지 사용 방법을 참조하세요.
- 코어 디바이스에 설치된 psutil 라이브러리 버전 5.7.0은 구성 요소와 호환이 확인된 최신 버전입니다.
- 코어 기기에 설치된 cbor 라이브러리 버전 1.0.0은 구성 요소와 함께 사용할 수 있는 것으로 확인된 최신 버전입니다.
- 이 구성 요소로부터 출력 데이터를 받으려면 이 구성 요소를 배포할 때 기존 구독 라우터 구성 요소 (`aws.greengrass.LegacySubscriptionRouter`) 에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성은 이 구성 요소가 응답을 게시하는 주제를 지정합니다.

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
```

```
      "id": "aws-greengrass-device-defender",
      "source": "component:aws.greengrass.DeviceDefender",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
      "id": "aws-greengrass-device-defender",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-
defender:version",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

- *### AWS ## #### ####* 바꾸십시오.
- *###* 이 구성 요소가 실행하는 Lambda 함수 버전으로 교체하십시오. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소 버전의 레시피를 확인해야 합니다. AWS IoT Greengrass 콘솔에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 쌍을 찾으십시오. 이 키-값 쌍에는 Lambda 함수의 이름과 버전이 들어 있습니다.

> **Important**
>
> 이 구성 요소를 배포할 때마다 레거시 구독 라우터에서 Lambda 함수 버전을 업데이트해야 합니다. 이렇게 하면 배포하는 구성 요소 버전에 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 설명은 배포 만들기 섹션을 참조하세요.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

3.1.1

다음 표에는 이 구성 요소의 버전 3.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <3.0.0	소프트
토큰 교환 서비스	>=0.0.0	하드

3.0.0 - 3.0.2

다음 표에는 이 구성 요소의 버전 3.0.0~3.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <3.0.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.10 and 2.0.11

다음 표에는 이 구성 요소의 버전 2.0.10 및 2.0.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	하드
람다 런처	^2.0.0	하드

종속성	호환되는 버전	종속성 유형
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	하드
람다 런처	>=1.0.0	하드
Lambda 런타임	>=1.0.0	소프트
토큰 교환 서비스	>=1.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

v3.x

`PublishRetryCount`

게시가 재시도되는 횟수입니다. 이 기능은 버전 3.1.1에서 사용할 수 있습니다.

최소값은 0입니다.

최대값은 72입니다.

기본값: 5

`SampleIntervalSeconds`

(선택 사항) 구성 요소가 지표를 수집하고 보고하는 각 주기 사이의 시간 (초) 입니다.

최소값은 300초(5분)입니다.

기본값: 300초

`UseInstaller`

(선택 사항) 이 구성 요소의 종속 항목을 설치하기 위해 이 구성 요소의 설치 프로그램 스크립트를 사용할지 여부를 정의하는 부울 값입니다.

사용자 지정 스크립트를 사용하여 종속성을 설치하거나 미리 빌드된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 로 설정하십시오. `false` 이 구성 요소를 사용하려면 종속성을 포함하여 다음 라이브러리를 설치하고 기본 Greengrass 시스템 사용자가 사용할 수 있도록 해야 합니다.

- AWS IoT Device SDK 파이썬용 v2
- cbor 라이브러리. 버전 1.0.0은 구성 요소와 함께 사용할 수 있는 것으로 확인된 최신 버전입니다.
- psutil 라이브러리. 버전 5.7.0은 구성 요소와 호환이 확인된 최신 버전입니다.

> **Note**
>
> HTTPS 프록시를 사용하도록 구성한 코어 디바이스에서 이 구성 요소의 버전 3.0.0 또는 3.0.1을 사용하는 경우 이 값을 로 설정해야 합니다. `false` 설치 프로그램 스크립트는 이 구성 요소의 이러한 버전에서 HTTPS 프록시 기반 작업을 지원하지 않습니다.

기본값: `true`

v2.x

> **Note**
>
> 이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함됩니다. 디바이스에서 이 구성 요소를 구성하려면 다음 파라미터만 편집하는 것이 좋습니다.

`lambdaParams`

이 구성 요소의 Lambda 함수에 대한 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`EnvironmentVariables`

Lambda 함수의 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`PROCFS_PATH`

(선택 사항) 폴더 경로. `/proc`

- 컨테이너에서 이 구성 요소를 실행하려면 기본값인 을 사용합니다`/host-proc`. 구성 요소는 기본적으로 컨테이너에서 실행됩니다.
- 이 구성 요소를 컨테이너 없음 모드에서 실행하려면 이 매개 변수를 지정하십시오`/proc`.

기본값: `/host-proc`. 이 경로가 이 구성 요소가 컨테이너에 `/proc` 폴더를 마운트하는 기본 경로입니다.

> **Note**
>
> 이 컴포넌트는 이 폴더에 대한 읽기 전용 액세스 권한을 가집니다.

`SAMPLE_INTERVAL_SECONDS`

(선택 사항) 구성 요소가 지표를 수집하고 보고하는 각 주기 사이의 시간 (초) 입니다.

최소값은 300초(5분)입니다.

기본값: 300초

`containerMode`

(선택 사항) 이 컴포넌트의 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- `GreengrassContainer`— 구성 요소는 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다. AWS IoT Greengrass
- `NoContainer`— 구성 요소는 격리된 런타임 환경에서 실행되지 않습니다.

  이 옵션을 지정하는 경우 `PROCFS_PATH` 환경 변수 매개 변수에 `/proc` 대해 지정해야 합니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소의 컨테이너 매개 변수를 포함하는 개체입니다. 를 `GreengrassContainer` 지정하면 구성 요소에서 이러한 매개 변수를 사용합니다`containerMode`.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당할 메모리 양 (KB) 입니다.

기본값은 50,000KB입니다.

`pubsubTopics`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제를 포함하는 객체입니다. 각 주제를 지정하고 구성 요소가 MQTT 주제를 구독하는지 AWS IoT Core 아니면 로컬 게시/구독 주제의 MQTT 주제를 구독할지를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열 형식의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

`type`

(선택 사항) 이 구성 요소가 메시지를 구독하는 데 사용하는 게시/구독 메시지의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- IOT_CORE— AWS IoT Core MQTT 메시지를 구독하십시오. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 있습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본값: PUB_SUB

`topic`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제입니다. 를 지정하는 경우 이 `IotCore type` 항목에서 MQTT 와일드카드 (+및#) 를 사용할 수 있습니다.

Example 예: 구성 병합 업데이트 (컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 예: 구성 병합 업데이트 (컨테이너 모드 없음)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

입력 데이터

이 구성 요소는 메시지를 입력 데이터로 받아들이지 않습니다.

출력 데이터

이 구성 요소는 다음과 같은 전용 주제에 보안 메트릭을 게시합니다. AWS IoT Device Defender 이 구성 요소는 *coreDeviceName*메트릭을 게시할 때 코어 장치의 이름으로 대체됩니다.

주제 (AWS IoT Core MQTT): `$aws/things/`*coreDeviceName*`/defender/metrics/json`

Example 출력 예시

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

이 구성 요소가 보고하는 메트릭에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 기기 메트릭 문서 사양을 참조하십시오.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DeviceDefender.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -
  Wait
  ```

라이선스

이 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

v3.x

버전	변경
3.1.1	버그 수정 및 개선 • 네트워크 중단 후 연결이 복구되지 않을 때 클라이언트 연결 재시도를 추가합니다. • 메트릭 게시를 위한 구성 가능한 재시도를 추가합니다.
3.1.0	버그 수정 및 개선 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.
3.0.1	구성 요소가 지표의 델타 값을 계산하는 방법과 관련된 문제를 수정합니다.

버전	변경
3.0.0	**Warning** 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 초기 버전

v2.x

버전	변경
2.0.11	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.10	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.9	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.8	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	그린그래스 뉴클리어스 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.

버전	변경
2.0.3	초기 버전

디스크 스풀러

디스크 스풀러 구성 요소 (`aws.greengrass.DiskSpooler`) 는 Greengrass 코어 디바이스에서 Greengrass 코어 디바이스로 스풀링된 메시지를 위한 영구 저장 옵션을 제공합니다. AWS IoT Core 이 구성 요소는 이러한 아웃바운드 메시지를 디스크에 저장합니다.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.0.x

유형

이 컴포넌트는 플러그인 컴포넌트 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- `storageType`이 구성 요소를 사용하도록 `Disk` 설정해야 합니다. Greengrass 핵 구성에서 이를 설정할 수 있습니다.
- `maxSizeInBytes`디바이스의 사용 가능한 공간보다 크게 구성해서는 안 됩니다. Greengrass 핵 구성에서 이를 설정할 수 있습니다.
- 디스크 스풀러 구성요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포하면 호환되는 버전의 AWS IoT Greengrass 종속 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

1.0.1 – 1.0.3

다음 표에는 이 구성 요소의 버전 1.0.1~1.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.11.0 <2.13.0	하드

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.11.0 <2.12.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

사용량

디스크 스풀러 구성 요소를 사용하려면 를 `aws.greengrass.DiskSpooler` 배포해야 합니다.

이 구성 요소를 구성하고 사용하려면 를 로 설정해야 합니다. `pluginName` `aws.greengrass.DiskSpooler`

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.0.3	버그 수정 및 개선 데이터베이스 연결을 재사용하여 성능을 개선합니다.
1.0.2	버그 수정 및 개선 특정 경우에 MQTT 메시지 형식 필드가 유지되지 않는 문제를 수정합니다.
1.0.1	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.0	초기 버전

Docker 애플리케이션 관리자

Docker 애플리케이션 관리자 구성 요소 (`aws.greengrass.DockerApplicationManager`) 를 사용하면 AWS IoT Greengrass Amazon Elastic Container Registry (Amazon ECR) 에 호스팅된 퍼블릭 이미지 레지스트리 및 프라이빗 레지스트리에서 Docker 이미지를 다운로드할 수 있습니다. 또한 자격 증명을 자동으로 AWS IoT Greengrass 관리하여 Amazon ECR의 프라이빗 리포지토리에서 이미지를 안전하게 다운로드할 수 있습니다.

Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 개발하는 경우 구성 요소에서 아티팩트로 지정된 Docker 이미지를 다운로드할 수 있도록 Docker 애플리케이션 관리자를 종속 항목으로 포함시키십시오. 자세한 설명은 도커 컨테이너 실행 섹션을 참조하세요.

주제

- 구성
- 로컬 로그 파일
- Changelog
- 다음 사항도 참조하십시오.

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 도커 엔진 1.9.1 이상이 그린그래스 코어 디바이스에 설치되었습니다. 버전 20.10은 Core 소프트웨어와 호환이 검증된 최신 버전입니다. AWS IoT Greengrass Docker 컨테이너를 실행하는 구성 요소를 배포하려면 먼저 코어 디바이스에 Docker를 직접 설치해야 합니다.
- Docker 데몬은 이 구성 요소를 배포하기 전에 코어 디바이스에서 시작되어 실행되었습니다.
- 지원되는 다음 이미지 소스 중 하나에 저장된 Docker 이미지:
  - 아마존 Elastic 컨테이너 레지스트리 (Amazon ECR) 의 퍼블릭 및 프라이빗 이미지 리포지토리
  - 퍼블릭 도커 허브 리포지토리
  - 퍼블릭 도커 신뢰할 수 있는 레지스트리

- Docker 이미지는 사용자 지정 Docker 컨테이너 구성 요소에 아티팩트로 포함됩니다. 다음 URI 형식을 사용하여 Docker 이미지를 지정하세요.
  - 프라이빗 아마존 ECR 이미지: `docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
  - 퍼블릭 아마존 ECR 이미지: `docker:public.ecr.aws/repository/image[:tag|@digest]`
  - 퍼블릭 도커 허브 이미지: `docker:name[:tag|@digest]`

  자세한 설명은 도커 컨테이너 실행 섹션을 참조하세요.

  > **Note**
  >
  > 이미지의 아티팩트 URI에 이미지 태그 또는 이미지 다이제스트를 지정하지 않으면 사용자 지정 Docker 컨테이너 구성 요소를 배포할 때 Docker 애플리케이션 관리자가 해당 이미지의 사용 가능한 최신 버전을 가져옵니다. 모든 코어 디바이스에서 동일한 버전의 이미지를 실행하도록 하려면 아티팩트 URI에 이미지 태그 또는 이미지 다이제스트를 포함하는 것이 좋습니다.

- Docker 컨테이너 구성 요소를 실행하는 시스템 사용자에게 루트 또는 관리자 권한이 있거나 루트 또는 관리자가 아닌 사용자로 실행하도록 Docker를 구성해야 합니다.
  - Linux 디바이스에서는 그룹에 사용자를 추가하여 `docker` 그룹 없이 `docker` 명령을 호출할 수 있습니다. `sudo`
  - Windows 장치에서는 관리자 권한 없이 `docker` 명령을 호출할 사용자를 `docker-users` 그룹에 추가할 수 있습니다.

  Linux or Unix

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 루트가 아닌 사용자를 `docker` 그룹에 추가하려면 `ggc_user` 다음 명령어를 실행합니다.

  ```
  sudo usermod -aG docker ggc_user
  ```

  자세한 내용은 루트가 아닌 사용자로 Docker 관리를 참조하십시오.

  Windows Command Prompt (CMD)

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 사용자를 `docker-users` 그룹에 추가하려면 `ggc_user` 관리자로 다음 명령을 실행합니다.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 사용자를 `docker-users` 그룹에 추가하려면 `ggc_user` 관리자로 다음 명령을 실행합니다.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- 네트워크 프록시를 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성하는 경우 동일한 프록시 서버를 사용하도록 Docker를 구성해야 합니다.
- Docker 이미지가 Amazon ECR 사설 레지스트리에 저장되어 있는 경우 토큰 교환 서비스 구성 요소를 Docker 컨테이너 구성 요소에 종속 항목으로 포함해야 합니다. 또한 Greengrass 장치 역할은 다음 예제 IAM `ecr:GetAuthorizationToken` 정책에 `ecr:BatchGetImage` 나와 있는 것처럼,, 및 `ecr:GetDownloadUrlForLayer` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- docker 애플리케이션 관리자 구성 요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.
  - docker 애플리케이션 관리자 구성요소에는 이미지를 다운로드할 수 있는 연결이 있어야 합니다. 예를 들어 ECR을 사용하는 경우 다음 엔드포인트에 연결되어 있어야 합니다.
    - `*.dkr.ecr.region.amazonaws.com`(VPC 엔드포인트)
      `com.amazonaws.region.ecr.dkr`

- `api.ecr.region.amazonaws.com`(VPC 엔드포인트) `com.amazonaws.region.ecr.api`

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`ecr.region.amazonaws.com`	443	아니요	Amazon ECR에서 Docker 이미지를 다운로드하는 경우 필수입니다.

`hub.docker.com`
 	
`registry.hub.docker.com/v1`
	443	아니요	Docker Hub에서 Docker 이미지를 다운로드하는 경우 필요합니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.0.11

다음 표에는 이 구성 요소의 버전 2.0.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.13.0	소프트

2.0.10

다음 표에는 이 구성 요소의 버전 2.0.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.12.0	소프트

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.11.0	소프트

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.10.0	소프트

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.9.0	소프트

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.8.0	소프트

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.7.0	소프트

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.6.0	소프트

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.5.0	소프트

2.0.2

다음 표에는 이 구성 요소의 버전 2.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.4.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.3.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.2.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.0.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.9	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.0.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.0	초기 버전

다음 사항도 참조하십시오.

- 도커 컨테이너 실행

Kinesis Video Streams용 에지 커넥터

Kinesis Video Streams 구성 요소 `aws.iot.EdgeConnectorForKVS` () 용 에지 커넥터는 로컬 카메라에서 비디오 피드를 읽고 Kinesis Video Streams에 스트림을 게시합니다. 실시간 스트리밍 프로토콜 (RTSP) 을 사용하여 인터넷 프로토콜 (IP) 카메라에서 비디오 피드를 읽도록 이 구성 요소를 구성할 수 있습니다. 그런 다음 Amazon Managed Grafana 또는 로컬 Grafana 서버에 대시보드를 설정하여 비디오 스트림을 모니터링하고 상호 작용할 수 있습니다.

이 구성 요소를 통합하여 AWS IoT TwinMaker Grafana 대시보드에서 비디오 스트림을 표시하고 제어할 수 있습니다. AWS IoT TwinMaker 물리적 시스템의 운영 디지털 트윈을 구축할 수 있게 해주는 AWS 서비스입니다. AWS IoT TwinMaker 이를 사용하여 센서, 카메라 및 엔터프라이즈 애플리케이션의 데이터를 시각화하여 실제 공장, 건물 또는 산업 플랜트를 추적할 수 있습니다. 또한 이 데이터를 사용하여 작업을 모니터링하고, 오류를 진단하고, 오류를 수정할 수 있습니다. 자세한 내용은 AWS IoT TwinMaker 사용 설명서의(AWS IoT TwinMaker 이)란 무엇입니까? 섹션을 참조하십시오.

이 구성 요소는 산업 데이터를 모델링하고 저장하는 AWS IoT SiteWise AWS 서비스인 구성을 저장합니다. 에서 AWS IoT SiteWise 자산은 장치, 장비 또는 기타 개체 그룹과 같은 개체를 나타냅니다. 이 구성 요소를 구성하고 사용하려면 각 Greengrass 코어 장치 및 각 코어 장치에 연결된 각 IP 카메라에 대한 AWS IoT SiteWise 자산을 생성합니다. 각 자산에는 라이브 스트리밍, 온디맨드 업로드, 로컬 캐싱과 같은 기능을 제어하도록 구성하는 속성이 있습니다. 각 카메라의 URL을 지정하려면 카메라의 URL이 AWS Secrets Manager 포함된 시크릿을 생성해야 합니다. 카메라에 인증이 필요한 경우 URL에 사용자 이름과 암호도 지정합니다. 그런 다음 IP 카메라의 자산 속성에 해당 암호를 지정합니다.

이 구성 요소는 각 카메라의 비디오 스트림을 Kinesis 비디오 스트림에 업로드합니다. 각 카메라의 AWS IoT SiteWise 에셋 구성에서 대상 Kinesis 비디오 스트림의 이름을 지정합니다. Kinesis 비디오 스트림이 없는 경우 이 구성 요소가 자동으로 생성합니다.

AWS IoT TwinMaker 는 이러한 AWS IoT SiteWise 자산과 Secrets Manager 암호를 만들기 위해 실행할 수 있는 스크립트를 제공합니다. 이러한 리소스를 만드는 방법과 이 구성 요소를 설치, 구성 및 사용하는 방법에 대한 자세한 내용은 사용 AWS IoT TwinMaker 설명서의 AWS IoT TwinMaker 비디오 통합을 참조하십시오.

> **Note**
>
> Kinesis Video Streams 구성 요소용 에지 커넥터는 다음 AWS 리전 제품에서만 사용할 수 있습니다.
>
> - 미국 동부(버지니아 북부)
> - 미국 서부(오레곤)
> - 유럽(프랑크푸르트)
> - 유럽(아일랜드)
> - 아시아 태평양(싱가포르)

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 구성 요소 구성이 각 코어 장치마다 고유해야 하므로 이 구성 요소는 단일 코어 장치에만 배포할 수 있습니다. 핵심 장치 그룹에는 이 구성 요소를 배포할 수 없습니다.
- GStreamer 1.18.4 이상이 코어 기기에 설치되었습니다. 자세한 내용은 GStreamer 설치를 참조하십시오.

  가 설치된 `apt` 기기에서 다음 명령을 실행하여 GStreamer를 설치할 수 있습니다.

  ```
  sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
   gstreamer1.0-plugins-base-apps
  sudo apt install -y gstreamer1.0-libav
  sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
  plugins-ugly gstreamer1.0-tools
  ```
- 각 코어 AWS IoT SiteWise 디바이스의 에셋. 이 AWS IoT SiteWise 자산은 코어 디바이스를 나타냅니다. 이 자산을 만드는 방법에 대한 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 AWS IoT TwinMaker 비디오 통합을 참조하십시오.
- 각 코어 장치에 연결하는 각 IP 카메라의 AWS IoT SiteWise 자산. 이러한 AWS IoT SiteWise 자산은 비디오를 각 코어 장치로 스트리밍하는 카메라를 나타냅니다. 각 카메라의 에셋은 카메라에 연결되

는 코어 디바이스의 에셋과 연결되어야 합니다. 카메라 자산에는 Kinesis 비디오 스트림, 인증 암호 및 비디오 스트리밍 매개변수를 지정하도록 구성할 수 있는 속성이 있습니다. 카메라 자산을 생성하고 구성하는 방법에 대한 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 AWS IoT TwinMaker 비디오 통합을 참조하십시오.

- 각 IP 카메라의 AWS Secrets Manager 비밀. 이 비밀번호는 키-값 쌍을 정의해야 합니다. 여기서 키는 있고 값은 카메라의 URL입니다. `RTSPStreamUrl` 카메라 인증이 필요한 경우 이 URL에 사용자 이름과 암호를 포함하십시오. 이 구성 요소에 필요한 리소스를 생성할 때 스크립트를 사용하여 암호를 만들 수 있습니다. 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 AWS IoT TwinMaker 비디오 통합을 참조하십시오.

  Secrets Manager 콘솔 및 API를 사용하여 추가 시크릿을 생성할 수도 있습니다. 자세한 내용은 AWS Secrets Manager사용 설명서의 암호 만들기를 참조하십시오.

- Greengrass 토큰 교환 역할은 다음 예시 IAM 정책에 표시된 것처럼 다음 AWS Secrets Manager 및 Kinesis Video Streams 작업을 허용해야 합니다. AWS IoT SiteWise

> **Note**
>
> 이 예제 정책은 디바이스가 및 라는 이름의 비밀 값을 가져오도록 허용합니다. **`IPCamera1Url`** **`IPCamera2Url`** 각 IP 카메라를 구성할 때 해당 카메라의 URL이 포함된 암호를 지정합니다. 카메라에 인증이 필요한 경우 URL에 사용자 이름과 암호도 지정합니다. 코어 디바이스의 토큰 교환 역할은 연결할 각 IP 카메라의 비밀에 대한 액세스를 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera1Url",
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
```

```
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:PutMedia",
        "kinesisvideo:TagStream"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

> **Note**
>
> 고객 관리 AWS Key Management Service 키를 사용하여 비밀을 암호화하는 경우 장치 역할도 해당 `kms:Decrypt` 작업을 허용해야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`kinesisvideo.` *`region`*`.amazonaws.com`	443	예	Kinesis Video Streams에 데이터를

엔드포인트	포트	필수	설명
			업로드합니다.
data.iotsitewise. *region*.amazonaws.com	443	예	비디오 스트림 메타데이터를 에 AWS IoT SiteWise 게시하십시오.
secretsmanager. *region*.amazonaws.com	443	예	카메라 URL 시크릿을 코어 디바이스에 다운로드합니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

다음 표에는 이 구성 요소의 버전 1.0.0~1.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	>=2.0.3	하드
스트림 매니저	>=2.0.9	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`SiteWiseAssetIdForHub`

이 코어 디바이스를 나타내는 AWS IoT SiteWise 자산의 ID. 이 자산을 만들고 이를 사용하여 이 구성 요소와 상호 작용하는 방법에 대한 자세한 내용은 사용 AWS IoT TwinMaker 설명서의 AWS IoT TwinMaker 비디오 통합을 참조하십시오.

Example 예: 구성 병합 업데이트

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

라이선스

이 구성 요소에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- 퀴츠 잡 스케줄러/아파치 라이선스 2.0
- GStreamer 1.x용 자바 바인딩/GNU 레서 제너럴 퍼블릭 라이선스 v3.0용

사용량

이 구성 요소를 구성하고 상호 작용하기 위해 코어 장치를 나타내는 AWS IoT SiteWise 자산과 해당 장치가 연결되는 IP 카메라에 속성을 설정할 수 있습니다. Grafana 대시보드에서 비디오 스트림을 시각화하고 상호 작용할 수도 있습니다. AWS IoT TwinMaker 자세한 내용은 사용 설명서의 AWS IoT TwinMaker 비디오 통합을 참조하십시오. AWS IoT TwinMaker

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

```
sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.0.4	버그 수정 및 개선 • 실시간 업로드가 중지되는 문제를 수정합니다.
1.0.3	일반적인 버그 수정 및 개선입니다.
1.0.1	일반적인 버그 수정 및 개선입니다.
1.0.0	초기 버전

다음 사항도 참조하십시오.

- AWS IoT TwinMaker 사용 설명서의 AWS IoT TwinMaker란 무엇입니까?
- AWS IoT TwinMaker AWS IoT TwinMaker 사용 설명서의 동영상 통합
- AWS IoT SiteWise 사용 설명서의 AWS IoT SiteWise란 무엇입니까?
- AWS IoT SiteWise사용 설명서의 속성 값 업데이트
- AWS Secrets Manager 사용 설명서의 AWS Secrets Manager란 무엇입니까?
- AWS Secrets Manager사용 설명서에서 시크릿 생성 및 관리

그린그래스 CLI

Greengrass CLI 구성 요소 (`aws.greengrass.Cli`) 는 코어 디바이스에서 로컬에서 구성 요소를 개발하고 디버깅하는 데 사용할 수 있는 로컬 명령줄 인터페이스를 제공합니다. Greengrass CLI를 사용하면 예를 들어 로컬 배포를 생성하고 코어 디바이스에서 구성 요소를 다시 시작할 수 있습니다.

Core 소프트웨어를 설치할 때 이 구성 요소를 설치할 수 있습니다. AWS IoT Greengrass 자세한 설명은 자습서: AWS IoT Greengrass V2 시작하기 섹션을 참조하세요.

> **Important**
>
> 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

이 구성 요소를 설치한 후 다음 명령을 실행하여 도움말 설명서를 확인하십시오. 이 구성 요소를 설치하면 폴더에 심볼 링크가 추가됩니다`greengrass-cli`. */greengrass/v2*`/bin` 이 경로에서 Greengrass CLI를 실행하거나 `PATH` 환경 변수에 추가하여 절대 경로 `greengrass-cli` 없이 실행할 수 있습니다.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

다음 명령은 예를 `com.example.HelloWorld` 들어 라는 이름의 구성 요소를 다시 시작합니다.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli component restart --names
 "com.example.HelloWorld"
```

자세한 설명은 그린그래스 커맨드 라인 인터페이스 섹션을 참조하세요.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 소프트웨어와 상호 작용하려면 Greengrass CLI를 사용할 권한이 있어야 합니다. AWS IoT Greengrass Greengrass CLI를 사용하려면 다음 중 하나를 수행하십시오.
  - AWS IoT Greengrass Core 소프트웨어를 실행하는 시스템 사용자를 사용하십시오.
  - 루트 또는 관리자 권한이 있는 사용자를 사용하십시오. Linux 코어 디바이스에서는 를 사용하여 루트 `sudo` 권한을 얻을 수 있습니다.
  - 구성 요소를 배포할 때 `AuthorizedPosixGroups` 또는 `AuthorizedWindowsGroups` 구성 매개 변수에 지정한 그룹에 속한 시스템 사용자를 사용하십시오. 자세한 내용은 Greengrass CLI 구성 요소 구성을 참조하십시오.
- Greengrass CLI 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포하면 호환되는 버전의 AWS IoT Greengrass 종속 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다.AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.12.0 and 2.12.1

다음 표에는 이 구성 요소의 버전 2.12.0 및 2.12.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.12.0 <2.13.0	소프트

2.11.0 – 2.11.3

다음 표에는 이 구성 요소의 버전 2.11.0~2.11.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.11.0 <2.12.0	소프트

2.10.0 – 2.10.3

다음 표에는 이 구성 요소의 버전 2.10.0~2.10.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.11.0	소프트

2.9.0 – 2.9.6

다음 표에는 이 구성 요소의 버전 2.9.0~2.9.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.10.0	소프트

2.8.0 – 2.8.1

다음 표에는 이 구성 요소의 버전 2.8.0 및 2.8.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.9.0	소프트

2.7.0

다음 표에는 이 구성 요소의 버전 2.7.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.8.0	소프트

2.6.0

다음 표에는 이 구성 요소의 버전 2.6.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.7.0	소프트

2.5.0 – 2.5.6

다음 표에는 이 구성 요소의 버전 2.5.0~2.5.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.6.0	소프트

2.4.0

다음 표에는 이 구성 요소의 버전 2.4.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.5.0	소프트

2.3.0

다음 표에는 이 구성 요소의 버전 2.3.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.4.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.1.0	소프트

> **Note**
>
> 그린그래스 핵의 최소 호환 버전은 그린그래스 CLI 구성 요소의 패치 버전에 해당합니다.

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

2.5.x

`AuthorizedPosixGroups`

(선택 사항) 쉼표로 구분된 시스템 그룹 목록을 포함하는 문자열입니다. 이러한 시스템 그룹이 Greengrass CLI를 사용하여 Core 소프트웨어와 상호 작용할 AWS IoT Greengrass 수 있도록 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어, `group1,1002,group3` 는 세 개의 시스템 그룹 (`group1`,`1002`, 및`group3`) 이 Greengrass CLI를 사용할 수 있도록 승인합니다.

승인할 그룹을 지정하지 않은 경우 Greengrass CLI를 루트 sudo 사용자 () 또는 Core 소프트웨어를 실행하는 AWS IoT Greengrass 시스템 사용자로 사용할 수 있습니다.

`AuthorizedWindowsGroups`

(선택 사항) 쉼표로 구분된 시스템 그룹 목록을 포함하는 문자열입니다. 이러한 시스템 그룹이 Greengrass CLI를 사용하여 Core 소프트웨어와 상호 작용할 AWS IoT Greengrass 수 있도록 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어, `group1,1002,group3` 는 세 개의 시스템 그룹 (`group1`,`1002`, 및`group3`) 이 Greengrass CLI를 사용할 수 있도록 승인합니다.

승인할 그룹을 지정하지 않은 경우 관리자 또는 Core 소프트웨어를 실행하는 시스템 사용자로 Greengrass CLI를 사용할 수 있습니다. AWS IoT Greengrass

Example 예: 구성 병합 업데이트

다음 예제 컨피그레이션은 세 개의 POSIX 시스템 그룹 (`group1`1002, 및`group3`) 과 두 개의 Windows 사용자 그룹 (`Device Operators`및`QA Engineers`) 이 Greengrass CLI를 사용할 수 있도록 권한을 부여하도록 지정합니다.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

`AuthorizedPosixGroups`

(선택 사항) 쉼표로 구분된 시스템 그룹 목록을 포함하는 문자열입니다. 이러한 시스템 그룹이 Greengrass CLI를 사용하여 Core 소프트웨어와 상호 작용할 AWS IoT Greengrass 수 있도록 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어, `group1,1002,group3` 는 세 개의 시스템 그룹 (`group1`,`1002`, 및`group3`) 이 Greengrass CLI를 사용할 수 있도록 승인합니다.

승인할 그룹을 지정하지 않은 경우 Greengrass CLI를 루트 sudo 사용자 () 또는 Core 소프트웨어를 실행하는 AWS IoT Greengrass 시스템 사용자로 사용할 수 있습니다.

Example 예: 구성 병합 업데이트

다음 예제 구성은 Greengrass CLI를 사용할 수 있는 세 개의 시스템 그룹 (`group11002`,, 및`group3`) 을 승인하도록 지정합니다.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.12.1	그린그래스 뉴클리어스 버전 2.12.1 릴리스를 위해 버전이 업데이트되었습니다.
2.12.0	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.11.3	그린그래스 뉴클리어스 버전 2.11.3 릴리스를 위해 버전이 업데이트되었습니다.
2.11.2	그린그래스 뉴클리어스 버전 2.11.2 릴리스를 위해 버전이 업데이트되었습니다.
2.11.1	그린그래스 뉴클리어스 버전 2.11.1 릴리스를 위해 버전이 업데이트되었습니다.
2.11.0	새로운 기능 • 로컬 배포를 취소할 수 있습니다. • 로컬 배포를 위한 장애 처리 정책을 구성할 수 있습니다.

버전	변경
	• 자세한 배포 상태 보고를 개선합니다.
2.10.3	그린그래스 뉴클리어스 버전 2.10.3 릴리스를 위해 버전이 업데이트되었습니다.
2.10.2	그린그래스 뉴클리어스 버전 2.10.2 릴리스를 위해 버전이 업데이트되었습니다.
2.10.1	그린그래스 뉴클리어스 버전 2.10.1 릴리스를 위해 버전이 업데이트되었습니다.
2.10.0	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.9.6	Greengrass 뉴클리어스 버전 2.9.6 릴리스를 위해 버전이 업데이트되었습니다.
2.9.5	Greengrass 뉴클리어스 버전 2.9.5 릴리스를 위해 버전이 업데이트되었습니다.
2.9.4	Greengrass 뉴클리어스 버전 2.9.4 릴리스를 위해 버전이 업데이트되었습니다.
2.9.3	Greengrass 뉴클리어스 버전 2.9.3 릴리스를 위해 버전이 업데이트되었습니다.
2.9.2	Greengrass 뉴클리어스 버전 2.9.2 릴리스를 위해 버전이 업데이트되었습니다.
2.9.1	Greengrass 뉴클리어스 버전 2.9.1 릴리스를 위해 버전이 업데이트되었습니다.
2.9.0	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.8.1	그린그래스 뉴클리어스 버전 2.8.1 릴리스에 대한 버전이 업데이트되었습니다.
2.8.0	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.7.0	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.6.0	새로운 기능 • Greengrass CLI에서 사용하는 IPC (프로세스 간 통신) 작업을 호출하는 사용자 지정 구성 요소에 대한 지원을 추가합니다. 이러한 IPC 작업을 사용하여 로컬 배포를 관리하고, 구성 요소 세부 정보를 보고, 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 암호를 생성할 수 있습니다. 자세한 내용은 IPC: 로컬 배포 및 구성 요소 관리를 참조하십시오. 버그 수정 및 개선 • 추가 사소한 수정 및 개선.
2.5.6	Greengrass 뉴클리어스 버전 2.5.6 릴리스를 위해 버전이 업데이트되었습니다.
2.5.5	Greengrass 뉴클리어스 버전 2.5.5 릴리스에 대한 버전이 업데이트되었습니다.
2.5.4	Greengrass 뉴클리어스 버전 2.5.4 릴리스를 위해 버전이 업데이트되었습니다.
2.5.3	Greengrass 뉴클리어스 버전 2.5.3 릴리스를 위해 버전이 업데이트되었습니다.
2.5.2	Greengrass 뉴클리어스 버전 2.5.2 릴리스를 위해 버전이 업데이트되었습니다.
2.5.1	그린그래스 뉴클리어스 버전 2.5.1 릴리스에 대한 버전이 업데이트되었습니다.
2.5.0	새로운 기능 • Windows를 실행하는 코어 디바이스에 대한 지원을 추가합니다. • Windows 디바이스에서 Greengrass CLI를 사용하도록 시스템 그룹을 승인하도록 지정할 수 있는 새 `AuthorizedWindowsGroups` 구성 매개변수를 추가합니다. • 로컬 배포를 위한 `windowsUser` 파라미터를 추가합니다. 이 매개 변수를 사용하여 Windows 코어 장치에서 구성 요소를 실행하는 데 사용할 사용자를 지정할 수 있습니다.
2.4.0	새로운 기능 • 시스템 리소스 제한에 대한 지원을 추가합니다. 로컬 배포를 생성할 때 각 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 및 배포 create 명령을 참조하십시오.

버전	변경
2.3.0	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.0	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.0.5 릴리스에 대한 버전이 업데이트되었습니다.
2.0.4	Greengrass 뉴클리어스 버전 2.0.4 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

IP 감지기

IP 탐지기 구성 요소 (`aws.greengrass.clientdevices.IPDetector`) 는 다음을 수행합니다.

- Greengrass 코어 디바이스의 네트워크 연결 정보를 모니터링합니다. 이 정보에는 코어 디바이스의 네트워크 엔드포인트와 MQTT 브로커가 작동하는 포트가 포함됩니다.
- AWS IoT Greengrass클라우드 서비스에서 코어 디바이스의 연결 정보를 업데이트합니다.

클라이언트 디바이스는 Greengrass 클라우드 디스커버리를 사용하여 관련 코어 디바이스의 연결 정보를 검색할 수 있습니다. 그러면 클라이언트 디바이스가 성공적으로 연결될 때까지 각 코어 디바이스에 연결을 시도할 수 있습니다.

> **Note**
>
> 클라이언트 디바이스는 Greengrass 코어 디바이스에 연결하여 MQTT 메시지와 데이터를 전송하여 처리하는 로컬 IoT 디바이스입니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

IP 탐지기 구성요소는 코어 장치의 기존 연결 정보를 탐지된 정보로 대체합니다. 이 구성 요소는 기존 정보를 제거하므로 IP 감지기 구성 요소를 사용하거나 연결 정보를 수동으로 관리할 수 있습니다.

> **Note**
>
> IP 탐지기 구성요소는 IPv4 주소만 탐지합니다.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 서비스 역할은 사용자 AWS 계정 및 권한에 연결되고 허용되어야 합니다. `iot:GetThingShadow` `iot:UpdateThingShadow`
- 코어 디바이스의 AWS IoT 정책에서 `greengrass:UpdateConnectivityInfo` 권한을 허용해야 합니다. 자세한 내용은 데이터 영역 작업에 대한 AWS IoT 정책 및 클라이언트 AWS IoT 장치를 지원하기 위한 최소 정책 섹션을 참조하세요.
- 기본 포트 8883이 아닌 포트를 사용하도록 코어 디바이스의 MQTT 브로커 구성 요소를 구성하는 경우 IP 탐지기 v2.1.0 이상을 사용해야 합니다. 브로커가 작동하는 포트를 보고하도록 구성하십시오.
- 복잡한 네트워크 설정을 사용하는 경우 IP 탐지기 구성 요소가 클라이언트 장치가 코어 장치에 연결할 수 있는 엔드포인트를 식별하지 못할 수 있습니다. IP 탐지기 구성 요소가 엔드포인트를 관리할 수 없는 경우 대신 코어 장치 엔드포인트를 수동으로 관리해야 합니다. 예를 들어 MQTT 브로커 포트를 코어 디바이스에 전달하는 라우터 뒤에 코어 디바이스가 있는 경우 라우터의 IP 주소를 코어 디바이스의 엔드포인트로 지정해야 합니다. 자세한 설명은 코어 디바이스 엔드포인트 관리 섹션을 참조하세요.
- IP 탐지기 구성요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.13.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.12.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.11.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.10.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.9.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.8.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.7.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.6.0	소프트

2.1.0 and 2.0.2

다음 표에는 이 구성 요소의 버전 2.1.0 및 2.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.5.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.4.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.3.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

2.1.x

`defaultPort`

(선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다.

기본값: 8883

`includeIPv4LoopbackAddrs`

(선택 사항) 이 옵션을 사용하여 IPv4 루프백 주소를 검색하고 보고할 수 있습니다. 이러한 IP 주소는 예를 들어 장치가 자체적으로 `localhost` 통신할 수 있는 IP 주소입니다. 코어 기기와 클라이언트 기기가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용하십시오.

기본값: `false`

`includeIPv4LinkLocalAddrs`

(선택 사항) 이 옵션을 사용하여 IPv4 링크-로컬 주소를 검색하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 동적 호스트 구성 프로토콜 (DHCP) 이 없거나 정적으로 할당된 IP 주소가 없는 경우 이 옵션을 사용하십시오.

기본값: `false`

2.0.x

`includeIPv4LoopbackAddrs`

(선택 사항) 이 옵션을 사용하여 IPv4 루프백 주소를 검색하고 보고할 수 있습니다. 이러한 IP 주소는 예를 들어 장치가 자체적으로 `localhost` 통신할 수 있는 IP 주소입니다. 코어 기기와 클라이언트 기기가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용하십시오.

기본값: `false`

`includeIPv4LinkLocalAddrs`

(선택 사항) 이 옵션을 사용하여 IPv4 링크-로컬 주소를 검색하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 동적 호스트 구성 프로토콜 (DHCP) 이 없거나 정적으로 할당된 IP 주소가 없는 경우 이 옵션을 사용하십시오.

기본값: `false`

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.8	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	버그 수정 및 개선 • 특정 시나리오에서 이 구성 요소가 기록하는 오류 메시지를 개선합니다.

버전	변경
	• Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	개선 사항 • 기본이 아닌 MQTT 브로커 포트를 사용할 수 있는 `defaultPort` 파라미터를 추가합니다. • 로그 메시지를 보다 명확하게 표시하도록 업데이트되었습니다.
2.0.2	Greengrass 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.1	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.0	초기 버전

Firehose

Firehose 구성 요소 (`aws.greengrass.KinesisFirehose`) 는 Amazon Data Firehose 전송 스트림을 통해 Amazon S3, Amazon Redshift 및 Amazon 서비스와 같은 대상에 데이터를 게시합니다. OpenSearch 자세한 내용은 Amazon Data Firehose란 무엇입니까? 를 참조하십시오. Amazon Data Firehose 개발자 가이드에서 확인할 수 있습니다.

이 구성 요소를 사용하여 Kinesis 전송 스트림에 게시하려면 이 구성 요소가 구독하는 주제에 메시지를 게시하십시오. 기본적으로 이 구성 요소는 `kinesisfirehose/message` 및 `kinesisfirehose/message/binary/#` 로컬 게시/구독 주제를 구독합니다. 이 구성 요소를 배포할 때 AWS IoT Core MQTT 주제를 비롯한 다른 주제를 지정할 수 있습니다.

> **Note**
>
> 이 구성 요소는 V1의 Firehose 커넥터와 유사한 기능을 제공합니다. AWS IoT Greengrass 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 Firehose 커넥터를 참조하십시오.

주제

- 버전

- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 입력 데이터
- 출력 데이터
- 로컬 로그 파일
- 라이선스
- Changelog
- 다음 사항도 참조하십시오.

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 Lambda 구성 요소 () 입니다. `aws.greengrass.lambda` Greengrass 핵은 Lambda 런처 구성 요소를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.

- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Greengrass 장치 역할은 다음 예제 IAM 정책에 나와 있는 것처럼 `firehose:PutRecord` 및 `firehose:PutRecordBatch` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

  이 구성 요소의 입력 메시지 페이로드에서 기본 전송 스트림을 동적으로 재정의할 수 있습니다. 애플리케이션에서 이 기능을 사용하는 경우 IAM 정책은 모든 대상 스트림을 리소스로 포함해야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해).

- 이 구성 요소로부터 출력 데이터를 받으려면 이 구성 요소를 배포할 때 기존 구독 라우터 구성 요소(aws.greengrass.LegacySubscriptionRouter) 에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성은 이 구성 요소가 응답을 게시하는 주제를 지정합니다.

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "component:aws.greengrass.KinesisFirehose",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
kinesisfirehose:version",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

- *### AWS ## #### ####* 바꾸십시오.
- *###* 이 구성 요소가 실행하는 Lambda 함수 버전으로 교체하십시오. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소 버전의 레시피를 확인해야 합니다. AWS IoT Greengrass 콘솔에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 쌍을 찾으십시오. 이 키-값 쌍에는 Lambda 함수의 이름과 버전이 들어 있습니다.

> **Important**
>
> 이 구성 요소를 배포할 때마다 레거시 구독 라우터에서 Lambda 함수 버전을 업데이트해야 합니다. 이렇게 하면 배포하는 구성 요소 버전에 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 설명은 배포 만들기 섹션을 참조하세요.

- Firehose 구성 요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.
  - Firehose 구성 요소에는 VPC 엔드포인트가 `firehose.region.amazonaws.com` 인 연결이 있어야 합니다. `com.amazonaws.region.kinesis-firehose`

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`firehose. region.amazonaws.com`	443	예	Firehose에 데이터를 업로드합니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	하드
람다 런처	^2.0.0	하드

종속성	호환되는 버전	종속성 유형
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.8 - 2.1.0

다음 표에는 이 구성 요소의 버전 2.0.8 및 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	하드
람다 런처	>=1.0.0	하드
Lambda 런타임	>=1.0.0	소프트
토큰 교환 서비스	>=1.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

> **Note**
>
> 이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함됩니다. 디바이스에서 이 구성 요소를 구성하려면 다음 파라미터만 편집하는 것이 좋습니다.

`lambdaParams`

이 구성 요소의 Lambda 함수에 대한 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`EnvironmentVariables`

Lambda 함수의 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`DEFAULT_DELIVERY_STREAM_ARN`

구성 요소가 데이터를 보내는 기본 Firehose 전송 스트림의 ARN입니다. 입력 메시지 페이로드의 `delivery_stream_arn` 속성으로 대상 스트림을 재정의할 수 있습니다.

> **Note**
>
> 핵심 장치 역할은 모든 대상 전송 스트림에서 필요한 작업을 허용해야 합니다. 자세한 설명은 요구 사항 섹션을 참조하세요.

`PUBLISH_INTERVAL`

(선택 사항) 구성 요소가 일괄 처리된 데이터를 Firehose에 게시할 때까지 대기할 최대 시간(초) 입니다. 메트릭을 수신할 때 (즉, 일괄 처리 없이) 게시하도록 구성 요소를 구성하려면 다음을 지정하세요. `0`

이 값은 최대 900초일 수 있습니다.

기본값: 10초

`DELIVERY_STREAM_QUEUE_SIZE`

(선택 사항) 구성 요소가 동일한 전송 스트림에 대한 새 레코드를 거부하기 전에 메모리에 보관할 최대 레코드 수입니다.

이 값은 최소 2,000개 레코드여야 합니다.

기본값: 5,000개 레코드

`containerMode`

(선택 사항) 이 컴포넌트의 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- `NoContainer`— 구성 요소는 격리된 런타임 환경에서 실행되지 않습니다.
- `GreengrassContainer`— 구성 요소는 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소의 컨테이너 매개 변수를 포함하는 개체입니다. 를 `GreengrassContainer` 지정하면 구성 요소에서 이러한 매개 변수를 사용합니다`containerMode`.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당할 메모리 양 (KB) 입니다.

기본값은 64MB (65,535KB) 입니다.

`pubsubTopics`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제를 포함하는 객체입니다. 각 주제를 지정하고 구성 요소가 MQTT 주제를 구독하는지 AWS IoT Core 아니면 로컬 게시/구독 주제의 MQTT 주제를 구독할지를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열 형식의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

`type`

(선택 사항) 이 구성 요소가 메시지를 구독하는 데 사용하는 게시/구독 메시지의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `PUB_SUB` – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- `IOT_CORE`— AWS IoT Core MQTT 메시지를 구독하십시오. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 있습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본값: `PUB_SUB`

`topic`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제입니다. 를 지정하는 경우 이 `IotCore` `type` 항목에서 MQTT 와일드카드 (+및#) 를 사용할 수 있습니다.

Example 예: 구성 병합 업데이트 (컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
```

```
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 예: 구성 병합 업데이트 (컨테이너 모드 없음)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "NoContainer"
}
```

입력 데이터

이 구성 요소는 다음 주제의 스트림 콘텐츠를 수락하고 대상 전송 스트림으로 콘텐츠를 전송합니다. 구성 요소는 두 가지 유형의 입력 데이터를 받아들입니다.

- `kinesisfirehose/message` 주제에 대한 JSON 데이터.
- `kinesisfirehose/message/binary/#` 주제에 대한 이진 데이터.

JSON 데이터의 기본 주제 (로컬 게시/구독): `kinesisfirehose/message`

메시지는 다음 속성을 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

`request`

기본 스트림과 다른 경우 전송 스트림과 대상 전송 스트림으로 보낸 데이터입니다.

다음 정보가 `object` 포함된 유형:

`data`

전송 스트림으로 보낸 데이터입니다.

유형: `string`

`delivery_stream_arn`

(선택 사항) 대상 Firehose 전송 스트림의 ARN입니다. 기본 전송 스트림을 재정의하려면 이 속성을 지정하세요.

유형: `string`

`id`

요청에 대한 임의의 ID입니다. 이 속성을 사용하여 입력 요청을 출력 응답에 매핑할 수 있습니다. 이 속성을 지정하면 구성 요소가 응답 개체의 `id` 속성을 이 값으로 설정합니다.

유형: `string`

Example 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"
}
```

바이너리 데이터의 기본 주제 (로컬 게시/구독): `kinesisfirehose/message/binary/#`

이 주제는 이진 데이터가 포함된 메시지를 보내는데 사용합니다. 구성 요소는 바이너리 데이터를 파싱하지 않습니다. 구성 요소는 데이터를 있는 그대로 스트리밍합니다.

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. 뒤에 슬래시 () / 를 포함해야 합니다.

출력 데이터

이 구성 요소는 기본적으로 응답을 다음 MQTT 주제에 출력 데이터로 게시합니다. 이 주제를 레거시 구독 라우터 구성 요소의 `subject` 구성에서 로 지정해야 합니다. 사용자 지정 구성 요소에서 이 항목

의 메시지를 구독하는 방법에 대한 자세한 내용은 을 참조하십시오MQTT 메시지 게시/구독 AWS IoT Core.

기본 주제 (AWS IoT Core MQTT): `kinesisfirehose/message/status`

Example 출력 예시

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**
>
> 구성 요소가 연결 오류와 같은 재시도할 수 있는 오류를 감지하면 다음 배치에서 게시를 다시 시도합니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass 루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

라이선스

이 구성 요소에는 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 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.7	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.4	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.
2.0.8	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	Greengrass 뉴클리어스 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

다음 사항도 참조하십시오.

- Amazon 데이터 파이어호스란 무엇입니까? Amazon Data Firehose 개발자 가이드에서

람다 런처

Lambda 런처 구성 요소 `aws.greengrass.LambdaLauncher` () 는 코어 디바이스에서 함수를 시작하고 AWS Lambda 중지합니다. AWS IoT Greengrass 또한 이 구성 요소는 컨테이너화를 설정하고 지정한 사용자로 프로세스를 실행합니다.

> **Note**
>
> Lambda 함수 구성 요소를 코어 디바이스에 배포하는 경우 배포에는 이 구성 요소도 포함됩니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- Lambda 런처 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환 가능한 버전의 AWS IoT Greengrass 종속성도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.0.11 – 2.0.13

다음 표에는 이 구성 요소의 버전 2.0.11~2.0.13에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
Lambda 관리자	>=2.0.0 <2.4.0	하드

2.0.9 – 2.0.10

다음 표에는 이 구성 요소의 버전 2.0.9~2.0.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
Lambda 관리자	>=2.0.0 <2.3.0	하드

2.0.4 - 2.0.8

다음 표에는 이 구성 요소의 버전 2.0.4~2.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
Lambda 관리자	>=2.0.0 <2.2.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
Lambda 관리자	>=2.0.3 <2.1.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass 루트 폴더 */greengrass/v2* 경로로 바꾸고 Name은 이 구성 요소가 시작하는 Lambda 함수 구성 요소의 *lambdaFunctionComponent####* 대체합니다.

  ```
  sudo tail -f /greengrass/v2/logs/lambdaFunctionComponentName.log
  ```

Changelog

다음 표는 각 구성 요소 버전의 변경 사항을 설명합니다.

버전	변경
2.0.13	버그 수정 및 개선 일반적인 버그 수정 및 개선입니다.

버전	변경
2.0.12	버그 수정 및 개선 이전 프로세스가 제대로 중지되지 않은 경우 Lambda Launcher에서 오류가 발생할 수 있는 문제를 수정합니다.
2.0.11	람다 매니저 2.3.0 지원
2.0.10	버그 수정 및 개선 • 일반적인 버그 수정 및 개선입니다.
2.0.9	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.8	Greengrass 뉴클리어스 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.7	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	일반 성능 향상 및 버그 수정.
2.0.4	버그 수정 및 개선 • 구성 요소가 Lambda 함수 `AddGroupOwner` 컨테이너로 올바르게 전달되지 않는 문제를 수정합니다.
2.0.3	초기 버전

Lambda 관리자

Lambda 관리자 구성 요소 `aws.greengrass.LambdaManager` () 는 Greengrass 코어 디바이스에서 실행되는 함수에 AWS Lambda 대한 작업 항목 및 프로세스 간 통신을 관리합니다.

> **Note**
>
> Lambda 함수 구성 요소를 코어 디바이스에 배포하는 경우 배포에는 이 구성 요소도 포함됩니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

주제

- 버전

- 운영 체제
- 유형
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

운영 체제

이 구성 요소는 Linux 코어 장치에만 설치할 수 있습니다.

유형

이 구성 요소는 플러그인 구성 요소 (`aws.greengrass.plugin`) 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- Lambda 관리자 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환 가능한 버전의 AWS IoT Greengrass 종속성도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.3.2

다음 표에는 이 구성 요소의 버전 2.3.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.2.10 and 2.3.1

다음 표에는 이 구성 요소의 버전 2.2.10 및 2.3.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.2.8 and 2.2.9

다음 표에는 이 구성 요소의 버전 2.2.8 및 2.2.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.2.5

다음 표에는 이 구성 요소의 버전 2.2.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.2.4

다음 표에는 이 구성 요소의 버전 2.2.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.2.1 - 2.2.3

다음 표에는 이 구성 요소의 버전 2.2.1 ~ 2.2.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.6.0	소프트

2.1.3 and 2.1.4

다음 표에는 이 구성 요소의 버전 2.1.3 및 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`logHandlerMode`

> **Note**
>
> Lambda 관리자 버전 2.3.0 이상에만 해당

사용할 Lambda 로그 관리자의 구현을 선택하는 데 사용됩니다. Lambda 로그를 읽는 `optimized` 데 사용할 스레드 수를 줄이려면 값을 로 설정합니다.

`getResultTimeoutInSecond`

(선택 사항) 제한 시간이 초과되기 전에 Lambda 함수를 실행할 수 있는 최대 시간 (초).

기본값: `60`

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

```
/greengrass/v2/logs/greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.3.2	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.1	버그 수정 및 개선 • 특정 오류의 로그 수준을 조정합니다.
2.3.0	새로운 기능 • 로그 핸들러는 CPU 부하를 줄이도록 최적화되었습니다. 구성 옵션을 로 설정하여 이 기능을 사용하십시오`logHandlerMode` `optimized` 버그 수정 및 개선 • 더 이상 전체 스택 트레이스를 로깅하지 않으므로 로그와 성능이 향상됩니다. `WorkQueueFullException` • 종료 시간 초과를 방지하기 위해 람다 종료 제한 시간을 15초에서 300초로 설정합니다. • 구성을 변경한 후 온디맨드 람다가 다시 시작되지 않는 문제를 수정합니다.

버전	변경
2.2.11	버그 수정 및 개선 • Lambda LegacySubscriptionRouter 구성이 변경될 때 구성이 업데이트되지 않는 문제를 수정합니다.
2.2.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.9	버그 수정 및 개선 시계가 왜곡되어 포트 번호가 손상되는 문제를 수정합니다.
2.2.8	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.7	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.6	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.5	새로운 기능 • 로컬 게시/구독 메시지를 구독하는 이벤트 소스에 MQTT 주제 와일드카드 지원을 추가합니다. 이 기능을 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소가 필요합니다. • Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.4	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.3	버그 수정 및 개선 • Lambda 함수의 여러 인스턴스가 단일 cgroup을 공유하는 문제를 해결합니다. 이 구성 요소는 cgroups를 사용하여 Lambda 함수의 리소스 사용을 관리합니다.

버전	변경
2.2.2	버그 수정 및 개선 • 특정 시나리오에서 고정된 Lambda 함수 구성 요소가 예기치 않게 다시 시작되는 문제를 수정합니다.
2.2.1	버그 수정 및 개선 • 이 구성 요소의 Greengrass nucleus 종속성 버전 제약 조건을 변경하여 종속성 해결 문제를 해결합니다.
2.2.0	버그 수정 및 개선 • Lambda 함수가 재시작 후 로그를 기록하지 못하던 문제를 수정합니다. • 주제에 와일드카드가 있는 경우 레거시 구독 라우터가 중복 메시지를 보내는 문제를 수정합니다. • 고정되지 않은 Lambda 함수가 의 Greengrass IPC (프로세스 간 통신) 라이브러리를 사용할 수 없었던 문제를 수정합니다. AWS IoT Device SDK
2.1.4	버그 수정 및 개선 • NodeJS 런타임을 사용하는 Lambda 함수가 메시지를 하나만 처리하던 문제를 수정합니다. • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

Lambda 런타임

Lambda 런타임 구성 요소 `aws.greengrass.LambdaRuntimes` () 는 Greengrass 코어 디바이스가 함수를 실행하는 데 사용하는 런타임을 제공합니다. AWS Lambda

> **Note**
>
> Lambda 함수 구성 요소를 코어 디바이스에 배포하는 경우 배포에는 이 구성 요소도 포함됩니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- Lambda 런타임 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

이 구성 요소에는 종속성이 없습니다.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.0.8	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

레거시 서브스크립션 라우터

레거시 구독 라우터 (`aws.greengrass.LegacySubscriptionRouter`) 는 Greengrass 코어 디바이스의 구독을 관리합니다. 구독은 Lambda 함수가 코어 디바이스에서 MQTT 메시징에 사용할 수

있는 주제를 정의하는 AWS IoT Greengrass V1의 기능입니다. 자세한 내용은 V1 개발자 안내서의 MQTT 메시징 워크플로의 관리형 구독을 참조하십시오. AWS IoT Greengrass

이 구성 요소를 사용하여 Core SDK를 사용하는 커넥터 구성 요소 및 Lambda 함수 구성 요소에 대한 구독을 활성화할 수 있습니다. AWS IoT Greengrass

> **Note**
>
> Lambda 함수가 Core SDK의 함수를 사용하는 `publish()` 경우에만 기존 구독 라우터 구성 요소가 필요합니다. AWS IoT Greengrass V2의 IPC (프로세스 간 통신) 인터페이스를 사용하도록 Lambda 함수 코드를 업데이트하면 기존 구독 라우터 구성 요소를 배포할 필요가 없습니다. AWS IoT Device SDK 자세한 내용은 다음 프로세스 간 통신 서비스를 참조하십시오.
>
> - 로컬 메시지 게시/구독
> - MQTT 메시지 게시/구독 AWS IoT Core

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 레거시 구독 라우터는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

v2.1.x

`subscriptions`

(선택 사항) 코어 기기에서 활성화할 구독. 이 객체는 각 키가 고유한 ID이고 각 값은 해당 커넥터의 구독을 정의하는 객체입니다. Core SDK를 사용하는 V1 커넥터 구성 요소 또는 Lambda 함수를 배포할 때 구독을 구성해야 합니다. AWS IoT Greengrass

각 구독 객체에는 다음 정보가 포함됩니다.

`id`

이 구독의 고유 ID. 이 ID는 이 구독 객체의 키와 일치해야 합니다.

`source`

Core SDK를 사용하여 AWS IoT Greengrass 지정한 주제에 대한 MQTT 메시지를 게시하는 Lambda 함수입니다. `subject` 다음 중 하나를 지정하십시오.

- 코어 디바이스의 Lambda 함수 구성 요소 이름. `component:`접두사를 사용하여 구성 요소 이름을 지정합니다 (예:). **`component:com.example.HelloWorldLambda`**
- 코어 디바이스에 있는 Lambda 함수의 Amazon 리소스 이름 (ARN).

> **Important**
>
> Lambda 함수 버전이 변경되면 새 버전의 함수로 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.
> 가져올 함수 버전이 포함된 Amazon 리소스 이름 (ARN) 을 지정해야 합니다. `$LATEST`같은 버전 별칭을 사용할 수 없습니다.

V1 커넥터 구성 요소에 대한 구독을 배포하려면 구성 요소의 이름 또는 커넥터 구성 요소의 Lambda 함수의 ARN을 지정하십시오.

`subject`

소스 및 타겟이 메시지를 게시하고 수신할 수 있는 MQTT 주제 또는 주제 필터. 이 값은 + 및 # 주제 와일드카드를 지원합니다.

`target`

에서 지정한 주제에 대한 MQTT 메시지를 수신하는 대상입니다. `subject` 구독은 함수가 코어 디바이스의 Lambda `source` 함수에 MQTT 메시지를 AWS IoT Core 게시하거나 코어 디바이스에 Lambda 함수에 게시하도록 지정합니다. 다음 중 하나를 지정하십시오.

- `cloud`. `source`함수는 MQTT 메시지를 에 게시합니다. AWS IoT Core
- 코어 디바이스의 Lambda 함수 구성 요소 이름. `component:`접두사를 사용하여 구성 요소 이름을 지정합니다 (예:). **`component:com.example.HelloWorldLambda`**
- 코어 디바이스에 있는 Lambda 함수의 Amazon 리소스 이름 (ARN).

> **Important**
>
> Lambda 함수 버전이 변경되면 새 버전의 함수로 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.
> 가져올 함수 버전이 포함된 Amazon 리소스 이름 (ARN) 을 지정해야 합니다. `$LATEST`같은 버전 별칭을 사용할 수 없습니다.

기본값: 구독 없음

Example 구성 업데이트 예시 (구독 정의) AWS IoT Core

다음 예제는 `com.example.HelloWorldLambda` Lambda 함수 구성 요소가 주제에 MQTT 메시지를 게시하도록 지정합니다. AWS IoT Core `hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example 구성 업데이트 예제 (다른 Lambda 함수에 대한 구독 정의)

다음 예제는 `com.example.HelloWorldLambda` Lambda 함수 구성 요소가 주제의 Lambda 함수 구성 요소에 MQTT 메시지를 게시하도록 `com.example.MessageRelay` 지정합니다. `hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

(선택 사항) 코어 디바이스에서 활성화할 서브스크립션. 이 객체는 각 키가 고유한 ID이고 각 값은 해당 커넥터의 구독을 정의하는 객체입니다. Core SDK를 사용하는 V1 커넥터 구성 요소 또는 Lambda 함수를 배포할 때 구독을 구성해야 합니다. AWS IoT Greengrass

각 구독 객체에는 다음 정보가 포함됩니다.

`id`

이 구독의 고유 ID. 이 ID는 이 구독 객체의 키와 일치해야 합니다.

`source`

Core SDK를 사용하여 AWS IoT Greengrass 지정한 주제에 대한 MQTT 메시지를 게시하는 Lambda 함수입니다. `subject` 다음을 지정합니다.

- 코어 디바이스에 있는 Lambda 함수의 Amazon 리소스 이름 (ARN).

> **Important**
>
> Lambda 함수 버전이 변경되면 새 버전의 함수로 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.
> 가져올 함수 버전이 포함된 Amazon 리소스 이름 (ARN) 을 지정해야 합니다. `$LATEST`같은 버전 별칭을 사용할 수 없습니다.

V1 커넥터 구성 요소에 대한 구독을 배포하려면 커넥터 구성 요소의 Lambda 함수의 ARN을 지정하십시오.

`subject`

소스 및 대상이 메시지를 게시하고 수신할 수 있는 MQTT 주제 또는 주제 필터. 이 값은 + 및 # 주제 와일드카드를 지원합니다.

`target`

에서 지정한 주제에 대한 MQTT 메시지를 수신하는 대상입니다. `subject` 구독은 함수가 코어 디바이스의 Lambda `source` 함수에 MQTT 메시지를 AWS IoT Core 게시하거나 코어 디바이스에 Lambda 함수에 게시하도록 지정합니다. 다음 중 하나를 지정하십시오.

- `cloud`. `source`함수는 MQTT 메시지를 에 게시합니다. AWS IoT Core
- 코어 디바이스에 있는 Lambda 함수의 Amazon 리소스 이름 (ARN).

> **Important**
>
> Lambda 함수 버전이 변경되면 새 버전의 함수로 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.

가져올 함수 버전이 포함된 Amazon 리소스 이름 (ARN) 을 지정해야 합니다. `$LATEST`같은 버전 별칭을 사용할 수 없습니다.

기본값: 구독 없음

Example 구성 업데이트 예시 (구독 정의) AWS IoT Core

다음 예제에서는 `Greengrass_HelloWorld` 함수가 주제에 MQTT 메시지를 게시하도록 AWS IoT Core 지정합니다. `hello/world`

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
  }
}
```

Example 구성 업데이트 예제 (다른 Lambda 함수에 대한 구독 정의)

다음 예제는 `Greengrass_HelloWorld` 함수가 주제에 MQTT 메시지를 게시하도록 지정합니다. `Greengrass_MessageRelay` `hello/world`

```
"subscriptions": {
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	버그 수정 및 개선 • `source`및 `target` 의 ARN 대신 구성 요소 이름을 지정하는 서포트를 추가합니다. 구독의 구성 요소 이름을 지정하면 Lambda 함수 버전이 변경될 때마다 구독을 재구성할 필요가 없습니다.
2.0.3	초기 버전

로컬 디버그 콘솔

로컬 디버그 콘솔 구성 요소 (`aws.greengrass.LocalDebugConsole`) 는 AWS IoT Greengrass 핵심 장치 및 해당 구성 요소에 대한 정보를 표시하는 로컬 대시보드를 제공합니다. 이 대시보드를 사용하여 핵심 장치를 디버깅하고 로컬 구성 요소를 관리할 수 있습니다.

> **Important**
>
> 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 사용자 이름과 암호를 사용하여 대시보드에 로그인합니다. 사용자 이름 () 이 제공됩니다. `debug` AWS IoT GreengrassCLI를 사용하여 코어 디바이스의 대시보드를 통해 사용자를 인증하는 임시 비밀번호를 생성해야 합니다. AWS IoT GreengrassCLI를 사용하여 로컬 디버그 콘솔을 사용할 수 있어야 합니다. 자세한 내용은 Greengrass CLI 요구 사항을 참조하십시오. 암호 생성 및 로그인 방법에 대한 자세한 내용은 로컬 디버그 콘솔 구성 요소 사용을 참조하십시오.
- 로컬 디버그 콘솔 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.4.1

다음 표에는 이 구성 요소의 버전 2.4.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.10.0 <2.13.0	하드
그린그래스 CLI	>=2.10.0 <2.13.0	하드

2.4.0

다음 표에는 이 구성 요소의 버전 2.4.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.10.0 <2.12.0	하드
그린그래스 CLI	>=2.10.0 <2.12.0	하드

2.3.0 and 2.3.1

다음 표에는 이 구성 요소의 버전 2.3.0 및 2.3.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.10.0 <2.12.0	하드
그린그래스 CLI	>=2.10.0 <2.12.0	하드

2.2.9

다음 표에는 이 구성 요소의 버전 2.2.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.12.0	하드

종속성	호환되는 버전	종속성 유형
그린그래스 CLI	>=2.1.0 <2.12.0	하드

2.2.8

다음 표에는 이 구성 요소의 버전 2.2.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.11.0	하드
그린그래스 CLI	>=2.1.0 <2.11.0	하드

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.10.0	하드
그린그래스 CLI	>=2.1.0 <2.10.0	하드

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.9.0	하드
그린그래스 CLI	>=2.1.0 <2.9.0	하드

2.2.5

다음 표에는 이 구성 요소의 버전 2.2.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.8.0	하드
그린그래스 CLI	>=2.1.0 <2.8.0	하드

2.2.4

다음 표에는 이 구성 요소의 버전 2.2.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.7.0	하드
그린그래스 CLI	>=2.1.0 <2.7.0	하드

2.2.3

다음 표에는 이 구성 요소의 버전 2.2.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.6.0	하드
그린그래스 CLI	>=2.1.0 <2.6.0	하드

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.5.0	하드
그린그래스 CLI	>=2.1.0 <2.5.0	하드

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.4.0	하드
그린그래스 CLI	>=2.1.0 <2.4.0	하드

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.3.0	하드
그린그래스 CLI	>=2.1.0 <2.3.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.2.0	하드
그린그래스 CLI	>=2.1.0 <2.2.0	하드

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트

종속성	호환되는 버전	종속성 유형
그린그래스 CLI	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

v2.1.x - v2.4.x

`httpsEnabled`

(선택 사항) 로컬 디버그 콘솔에서 HTTPS 통신을 활성화할 수 있습니다. HTTPS 통신을 활성화하면 로컬 디버그 콘솔이 자체 서명된 인증서를 만듭니다. 웹 브라우저는 자체 서명된 인증서를 사용하는 웹 사이트에 대한 보안 경고를 표시하므로 인증서를 수동으로 확인해야 합니다. 그러면 경고를 우회할 수 있습니다. 자세한 설명은 사용량 섹션을 참조하세요.

기본값: `true`

`port`

(선택 사항) 로컬 디버그 콘솔을 제공할 포트입니다.

기본값: `1441`

`websocketPort`

(선택 사항) 로컬 디버그 콘솔에 사용할 웹 소켓 포트입니다.

기본값: `1442`

`bindHostname`

(선택 사항) 로컬 디버그 콘솔에 사용할 호스트 이름.

Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어를 실행하는 경우 이 매개변수를 로 `0.0.0.0` 설정하면 Docker 컨테이너 외부에서 로컬 디버그 콘솔을 열 수 있습니다.

기본값: `localhost`

Example 예: 구성 병합 업데이트

다음 예제 구성은 기본 포트가 아닌 포트에서 로컬 디버그 콘솔을 열고 HTTPS를 비활성화하도록 지정합니다.

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

`port`

(선택 사항) 로컬 디버그 콘솔을 제공할 포트입니다.

기본값: `1441`

`websocketPort`

(선택 사항) 로컬 디버그 콘솔에 사용할 웹 소켓 포트입니다.

기본값: `1442`

`bindHostname`

(선택 사항) 로컬 디버그 콘솔에 사용할 호스트 이름.

Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어를 실행하는 경우 이 매개변수를 `0.0.0.0`로 설정하면 Docker 컨테이너 외부에서 로컬 디버그 콘솔을 열 수 있습니다.

기본값: `localhost`

Example 예: 구성 병합 업데이트

다음 예제 구성은 기본 포트가 아닌 포트에서 로컬 디버그 콘솔을 열도록 지정합니다.

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

사용량

로컬 디버그 콘솔을 사용하려면 Greengrass CLI에서 세션을 생성하십시오. 세션을 생성하면 Greengrass CLI는 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 사용자 이름과 임시 비밀번호를 제공합니다.

다음 지침에 따라 코어 디바이스 또는 개발 컴퓨터에서 로컬 디버그 콘솔을 여십시오.

v2.1.x - v2.4.x

버전 2.1.0 이상에서는 로컬 디버그 콘솔이 기본적으로 HTTPS를 사용합니다. HTTPS가 활성화되면 로컬 디버그 콘솔은 연결을 보호하기 위해 자체 서명된 인증서를 만듭니다. 이 자체 서명된 인증서 때문에 로컬 디버그 콘솔을 열면 웹 브라우저에 보안 경고가 표시됩니다. Greengrass CLI로 세션을 생성하면 출력에 인증서의 지문이 포함되므로 인증서가 합법적이고 연결이 안전한지 확인할 수 있습니다.

HTTPS를 비활성화할 수 있습니다. 자세한 내용은 로컬 디버그 콘솔 구성을 참조하십시오.

로컬 디버그 콘솔을 열려면

1. (선택 사항) 개발 컴퓨터에서 로컬 디버그 콘솔을 보려면 SSH를 통해 콘솔 포트를 전달하면 됩니다. 하지만 먼저 코어 디바이스의 SSH 구성 파일에서 `AllowTcpForwarding` 옵션을 활성화해야 합니다. 이 옵션은 기본적으로 활성화되어 있습니다. 개발 컴퓨터에서 다음 명령을 실행하여 개발 컴퓨터의 대시보드를 보십시오`localhost:1441`.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > 기본 포트는 `1441` 및 에서 변경할 수 `1442` 있습니다. 자세한 내용은 로컬 디버그 콘솔 구성을 참조하십시오.

2. 세션을 생성하여 로컬 디버그 콘솔을 사용하십시오. 세션을 만들 때 인증에 사용할 암호를 생성합니다. 로컬 디버그 콘솔에는 보안을 강화하기 위해 암호가 필요합니다. 이 구성 요소를 사용하여 중요한 정보를 보고 코어 장치에서 작업을 수행할 수 있기 때문입니다. 또한 구성 요소 구성에서 HTTPS를 사용하도록 설정하는 경우 로컬 디버그 콘솔은 연결 보안을 위한 인증서를 생성합니다. HTTPS는 기본적으로 활성화되어 있습니다.

AWS IoT GreengrassCLI를 사용하여 세션을 생성합니다. 이 명령은 8시간 후에 만료되는 임의의 43자 암호를 생성합니다. */greengrass/v2*또는 *C:\greengrass\v2 # ##* 폴더 경로로 바꾸십시오. AWS IoT Greengrass V2

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

HTTPS를 사용하도록 로컬 디버그 콘솔을 구성한 경우 명령 출력은 다음 예와 같습니다. 로컬 디버그 콘솔을 열 때 인증서 지문을 사용하여 연결이 안전한지 확인합니다.

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is
 self-signed so you will need to bypass your web browser's security warnings to
 open the console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
 96 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

디버그 뷰 구성 요소는 8시간 동안 지속되는 세션을 만듭니다. 그런 다음 로컬 디버그 콘솔을 다시 보려면 새 암호를 생성해야 합니다.

3. 대시보드를 열고 로그인합니다. Greengrass 코어 장치 또는 SSH를 통해 포트를 전달하는 경우 개발 컴퓨터에서 대시보드를 볼 수 있습니다. 다음 중 하나를 수행합니다.

   - 로컬 디버그 콘솔에서 HTTPS를 활성화한 경우 (기본 설정), 다음을 수행하십시오.

     a. 코어 `https://localhost:1441` 디바이스에서 열거나, SSH를 통해 포트를 전달한 경우 개발 컴퓨터에서 엽니다.

        브라우저에 잘못된 보안 인증서에 대한 보안 경고가 표시될 수 있습니다.

b. 브라우저에 보안 경고가 표시되면 인증서가 합법적인지 확인하고 보안 경고를 우회하십시오. 다음을 따릅니다.

   i. 인증서의 SHA-256 또는 SHA-1 지문을 찾아 `get-debug-password` 명령에서 이전에 인쇄한 SHA-256 또는 SHA-1 지문과 일치하는지 확인하십시오. 브라우저가 하나 또는 두 개의 지문을 모두 제공할 수 있습니다. 인증서 및 지문을 보려면 브라우저 설명서를 참조하십시오. 일부 브라우저에서는 인증서 지문을 지문이라고 합니다.

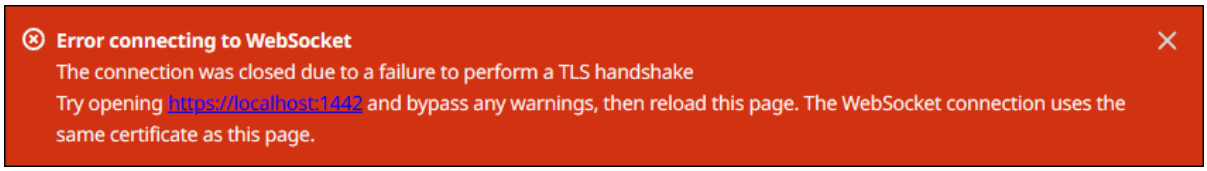
   > **Note**
   >
   > 인증서 지문이 일치하지 않는 경우 로 Step 2 이동하여 새 세션을 생성하십시오. 인증서 지문이 여전히 일치하지 않으면 연결이 안전하지 않을 수 있습니다.

   ii. 인증서 지문이 일치하면 브라우저의 보안 경고를 우회하여 로컬 디버그 콘솔을 여십시오. 브라우저 보안 경고를 우회하려면 브라우저 설명서를 참조하십시오.

c. `get-debug-password`명령에서 이전에 인쇄한 사용자 이름과 암호를 사용하여 웹 사이트에 로그인합니다.

   로컬 디버그 콘솔이 열립니다.

d. 로컬 디버그 콘솔에 TLS 핸드셰이크 WebSocket 실패로 인해 연결할 수 없다는 오류가 표시되면 URL에 대해 자체 서명된 보안 경고를 우회해야 합니다. WebSocket



   다음을 따릅니다.

   i. 로컬 `https://localhost:1442` 디버그 콘솔을 연 브라우저와 동일한 브라우저에서 엽니다.

   ii. 인증서를 확인하고 보안 경고를 우회하십시오.

   경고를 우회한 후 브라우저에 HTTP 404 페이지가 표시될 수 있습니다.

   iii. 다시 엽니다`https://localhost:1441`.

   로컬 디버그 콘솔에는 코어 디바이스에 대한 정보가 표시됩니다.

- 로컬 디버그 콘솔에서 HTTPS를 비활성화한 경우 다음을 수행하십시오.

  a. 코어 `http://localhost:1441` 디바이스에서 열고, SSH를 통해 포트를 전달한 경우 개발 컴퓨터에서 여십시오.

  b. `get-debug-password`명령에서 이전에 인쇄한 사용자 이름과 암호를 사용하여 웹 사이트에 로그인합니다.

  로컬 디버그 콘솔이 열립니다.

v2.0.x

로컬 디버그 콘솔을 열려면

1. (선택 사항) 개발 컴퓨터에서 로컬 디버그 콘솔을 보려면 SSH를 통해 콘솔 포트를 전달하면 됩니다. 하지만 먼저 코어 디바이스의 SSH 구성 파일에서 `AllowTcpForwarding` 옵션을 활성화해야 합니다. 이 옵션은 기본적으로 활성화되어 있습니다. 개발 컴퓨터에서 다음 명령을 실행하여 개발 컴퓨터의 대시보드를 보십시오`localhost:1441`.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > 기본 포트는 1441 및 에서 변경할 수 1442 있습니다. 자세한 내용은 로컬 디버그 콘솔 구성을 참조하십시오.

2. 세션을 생성하여 로컬 디버그 콘솔을 사용하십시오. 세션을 만들 때 인증에 사용할 암호를 생성합니다. 로컬 디버그 콘솔에는 보안을 강화하기 위해 암호가 필요합니다. 이 구성 요소를 사용하여 중요한 정보를 보고 코어 장치에서 작업을 수행할 수 있기 때문입니다.

   AWS IoT GreengrassCLI를 사용하여 세션을 생성합니다. 이 명령은 8시간 후에 만료되는 임의의 43자 암호를 생성합니다. */greengrass/v2*또는 *C:\greengrass\v2 # ##* 폴더 경로로 바꾸십시오. AWS IoT Greengrass V2

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

명령 출력은 다음 예와 같습니다.

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

디버그 뷰 구성 요소는 4시간 동안 지속되는 세션을 만든 다음 새 암호를 생성해야 로컬 디버그 콘솔을 다시 볼 수 있습니다.

3. 코어 `http://localhost:1441` 디바이스에서 여십시오. SSH를 통해 포트를 전달한 경우에는 개발 컴퓨터에서 여십시오.
4. `get-debug-password`명령에서 이전에 인쇄한 사용자 이름과 암호를 사용하여 웹 사이트에 로그인합니다.

   로컬 디버그 콘솔이 열립니다.

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.4.1	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.4.0	새로운 기능 • 스트림 매니저 디버깅 콘솔을 추가합니다.
2.3.1	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.0	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다. 새로운 기능 • MQTT 디버그 클라이언트가 포함되어 PubSub 있습니다. AWS IoT Core
2.2.7	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.6	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.5	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.4	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.2.3	버그 수정 및 개선 • 구성 요소가 SSL 개인 키를 보유한 키스토어를 해독하지 못할 때 시작되지 않던 문제를 수정합니다. • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.2	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.1	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.0	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • HTTPS를 사용하여 로컬 디버그 콘솔에 대한 연결을 보호합니다. HTTPS는 기본적으로 활성화되어 있습니다. 버그 수정 및 개선 • 구성 편집기에서 플래시바 메시지를 무시할 수 있습니다.
2.0.3	초기 버전

로그 매니저

로그 관리자 구성 요소 (`aws.greengrass.LogManager`) 는 AWS IoT Greengrass 코어 디바이스에서 Amazon Logs로 CloudWatch 로그를 업로드합니다. Greengrass 코어, 기타 Greengrass 구성 요소, Greengrass 구성 요소가 아닌 기타 애플리케이션 및 서비스에서 로그를 업로드할 수 있습니다. 로그 및 로컬 파일 시스템의 로그를 모니터링하는 방법에 대한 자세한 내용은 을 참조하십시오. CloudWatch 모니터 AWS IoT Greengrass 로그

로그 관리자 구성 요소를 사용하여 로그에 기록할 CloudWatch 때는 다음 고려 사항이 적용됩니다.

- 로그 지연

> **Note**
>
> 순환 및 활성 로그 파일의 로그 지연을 줄이는 로그 관리자 버전 2.3.0으로 업그레이드하는 것이 좋습니다. 로그 관리자 2.3.0으로 업그레이드할 때는 Greengrass nucleus 2.9.1으로도 업그레이드하는 것이 좋습니다.

로그 관리자 구성 요소 버전 2.2.8 (이하) 은 순환된 로그 파일의 로그만 처리하고 업로드합니다. 기본적으로 AWS IoT Greengrass Core 소프트웨어는 1시간마다 또는 1,024KB가 된 이후에 로그 파일을 교체합니다. 따라서 로그 관리자 구성 요소는 AWS IoT Greengrass Core 소프트웨어 또는 Greengrass 구성 요소가 1,024KB 이상의 로그를 기록한 후에만 로그를 업로드합니다. 로그 파일 크기를 낮게 제한하여 로그 파일이 더 자주 회전하도록 구성할 수 있습니다. 이로 인해 로그 관리자 구성 요소가 로그를 로그에 CloudWatch 더 자주 업로드합니다.

로그 관리자 구성 요소 버전 2.3.0 (이상) 은 모든 로그를 처리하고 업로드합니다. 새 로그를 작성하면 로그 관리자 버전 2.3.0 (이상) 은 활성 로그 파일이 회전될 때까지 기다리지 않고 해당 활성 로그 파일을 처리하여 직접 업로드합니다. 즉, 5분 이내에 새 로그를 볼 수 있습니다.

로그 관리자 구성 요소는 새 로그를 주기적으로 업로드합니다. 기본적으로 로그 관리자 구성 요소는 5분마다 새 로그를 업로드합니다. 업로드 간격을 낮게 구성하여 로그 관리자 구성 요소가 로그를 로그에 더 자주 업로드하도록 CloudWatch 구성할 수 있습니다. `periodicUploadIntervalSec` 이 주기적 간격을 구성하는 방법에 대한 자세한 내용은 구성을 참조하십시오.

동일한 Greengrass 파일 시스템에서 거의 실시간으로 로그를 업로드할 수 있습니다. 로그를 실시간으로 관찰해야 하는 경우 파일 시스템 로그를 사용하는 것이 좋습니다.

> **Note**
>
> 다른 파일 시스템을 사용하여 로그를 기록하는 경우 로그 관리자는 로그 관리자 구성 요소 버전 2.2.8 및 이전 버전의 동작으로 되돌아갑니다. 파일 시스템 로그에 액세스하는 방법에 대한 자세한 내용은 파일 시스템 로그 액세스를 참조하십시오.

- 클럭 스큐

  로그 관리자 구성요소는 표준 서명 버전 4 서명 프로세스를 사용하여 CloudWatch 로그에 대한 API 요청을 생성합니다. 코어 디바이스의 시스템 시간이 15분 이상 동기화되지 않으면 CloudWatch Logs는 요청을 거부합니다. 자세한 정보는 AWS 일반 참조의 서명 버전 4 서명 프로세스를 참조하십시오.

이 구성 요소가 로그를 업로드하는 로그 그룹 및 로그 스트림에 대한 자세한 내용은 을 참조하십시오. 사용량

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 사용량
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 장치 역할은 다음 예제 IAM `logs:CreateLogStream` 정책에 `logs:PutLogEvents` 나와 있는 것처럼,, 및 `logs:DescribeLogStreams` 작업을 허용해야 합니다. `logs:CreateLogGroup`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어를 설치할 때 생성하는 Greengrass 장치 역할에는 기본적으로 이 예제 정책의 권한이 포함됩니다.

자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 CloudWatch 로그에 대한 ID 기반 정책(IAM 정책) 사용을 참조하십시오.

- 로그 관리자 구성요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.
  - 로그 관리자 구성 요소에는 VPC 엔드포인트가 `logs.region.amazonaws.com` 인 연결이 있어야 합니다. `com.amazonaws.us-east-1.logs`

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`logs.region.amazonaws.com`	443	아니요	로그에 로그를 기록하는 경우 필요합니다. CloudWatch

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.3.7

다음 표에는 이 구성 요소의 버전 2.3.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.13.0	소프트

2.3.5 and 2.3.6

다음 표에는 이 구성 요소의 버전 2.3.5 및 2.3.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.12.0	소프트

2.3.3 – 2.3.4

다음 표에는 이 구성 요소의 버전 2.3.3~2.3.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.11.0	소프트

2.2.8 – 2.3.2

다음 표에는 이 구성 요소의 버전 2.2.8 ~ 2.3.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.10.0	소프트

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.9.0	소프트

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.8.0	소프트

2.2.5

다음 표에는 이 구성 요소의 버전 2.2.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.7.0	소프트

2.2.1 - 2.2.4

다음 표에는 이 구성 요소의 버전 2.2.1 - 2.2.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.6.0	소프트

2.1.3 and 2.2.0

다음 표에는 이 구성 요소의 버전 2.1.3 및 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.1.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

v2.3.6 – v2.3.7

`logsUploaderConfiguration`

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그의 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`systemLogsConfiguration`

(선택 사항) Greengrass 핵 및 플러그인 구성 요소의 로그를 포함하는 AWS IoT Greengrass Core 소프트웨어 시스템 로그의 구성 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정하십시오. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`uploadToCloudWatch`

(선택 사항) 시스템 로그를 Logs에 업로드할 CloudWatch 수 있습니다.

기본값: `false`

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 Greengrass nucleus 구성요소가 JSON 형식 로그를 출력하도록 구성한 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 로깅 형식 매개 변수 () JSON 를 지정하십시오. `logging.format`

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) Greengrass 시스템 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` Greengrass 시스템 로그 파일의 총 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 동일합니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 단위입니다. `diskSpaceLimit` 다음 옵션 중 하나를 선택합니다.

- KB— 킬로바이트
- MB— 메가바이트
- GB— 기가바이트

기본값: KB

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`componentLogsConfigurationMap`

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵. 이 맵의 각 `componentName` 개체는 구성 요소 또는 응용 프로그램의 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 Logs에 CloudWatch 업로드합니다.

> **Important**
>
> 구성 요소당 단일 구성 키를 사용하는 것이 좋습니다. 를 사용할 때 활발히 기록되는 로그 파일이 하나뿐인 파일 그룹만 대상으로 지정해야 `logFileRegex` 합니다. 이 권장 사항을 따르지 않으면 로그가 중복되어 업로드될 수 CloudWatch 있습니다. 단일 정규식을 사용하여 여러 활성 로그 파일을 대상으로 하는 경우 log manager v2.3.1 이상으로 업그레이드하고 예제 구성을 사용하여 구성을 변경하는 것이 좋습니다.

> **Note**
>
> v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 대신 목록을 계속 사용할 수 있습니다. `componentLogsConfiguration` `componentLogsConfigurationMap` 하지만 병합 및 재설정 업데이트를 사용하여 특정 구성 요소의 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration`매개변수에 대한 자세한 내용은 이 구성 요소의 v2.1.x용 구성 매개변수를 참조하십시오.

componentName

이 로그 구성을 위한 구성 *componentName*요소 또는 응용 프로그램의 로그 구성입니다. Greengrass 구성 요소의 이름 또는 다른 값을 지정하여 이 로그 그룹을 식별할 수 있습니다.

각 개체에는 다음 정보가 포함됩니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 특정 JSON 형식을 사용하는 경우에만 적용됩니다. 이 형식은 AWS IoT Greengrass로깅 모듈 리포지토리에서 찾을 수 있습니다. GitHub

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` 이 구성 요소 로그 파일의 전체 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 관련이 있습니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 `diskSpaceLimit` 단위입니다. 다음 옵션 중 하나를 선택합니다.

- `KB`— 킬로바이트
- `MB`— 메가바이트
- `GB`— 기가바이트

기본값: `KB`

`logFileDirectoryPath`

(선택 사항) 이 구성 요소의 로그 파일이 들어 있는 폴더의 경로입니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

기본값: */greengrass/v2*/logs.

`logFileRegex`

(선택 사항) 구성 요소나 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규 표현식입니다. 로그 관리자 구성 요소는 이 정규 표현식을 사용하여 의 폴더에 있는 로그 파일을 `logFileDirectoryPath` 식별합니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

구성 요소나 애플리케이션이 로그 파일을 회전시키는 경우 회전된 로그 파일 이름과 일치하는 정규식을 지정하십시오. 예를 들어 Hello World 애플리케이션의 로그를 **`hello_world\\\\w*.log`** 업로드하도록 지정할 수 있습니다. `\\\\w*`패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일을 일치시킵니다. 이 예제에서 로그 관리자는 다음 로그 파일을 업로드합니다.

- `hello_world.log`— Hello World 애플리케이션의 최신 로그 파일입니다.
- `hello_world_2020_12_15_17_0.log`— 헬로 월드 애플리케이션의 이전 로그 파일입니다.

기본값:*componentName*`\\\\w*.log`, 여기서 *ComponentName#* 이 로그 구성의 구성 요소 이름입니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규 표현식입니다. 정규 표현식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 이전 줄의 로그 메시지에 새 줄을 추가합니다.

기본적으로 로그 관리자 구성요소는 줄이 탭이나 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자가 해당 줄을 새 로그 메시지로 처리합니다. 그렇지 않으면 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성요소가 스택 트레이스와 같이 여러 줄에 걸쳐있는 메시지를 분할하지 않도록 합니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간 (초).

기본값: `300` (5분)

최소: `0.000001` (1마이크로초)

`deprecatedVersionSupport`

로그 관리자가 로그 관리자 v2.3.5에 도입된 향상된 로깅 속도를 사용해야 하는지 여부를 나타냅니다. 개선 기능을 `false` 사용하려면 값을 로 설정합니다.

log manager v2.3.1 또는 이전 버전에서 업그레이드할 `false` 때 이 값을 로 설정하면 중복된 로그 항목이 업로드될 수 있습니다.

기본값은 `true`입니다.

Example 예: 구성 병합 업데이트

다음 예제 구성은 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 로그에 업로드하도록 CloudWatch 지정합니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
```

```
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example 예: log manager v2.3.1을 사용하여 여러 활성 로그 파일을 업로드하도록 구성

다음 예제 구성은 여러 활성 로그 파일을 대상으로 하려는 경우 권장되는 예제입니다. 이 예제 구성은 업로드하려는 활성 로그 파일을 지정합니다 CloudWatch. 이 구성 예제 구성을 사용하면 다음과 일치하는 회전된 파일도 업로드됩니다. `logFileRegex` 이 예제 구성은 로그 관리자 v2.3.1에서 지원됩니다.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.x

`logsUploaderConfiguration`

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그의 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`systemLogsConfiguration`

(선택 사항) Greengrass 핵 및 플러그인 구성 요소의 로그를 포함하는 AWS IoT Greengrass Core 소프트웨어 시스템 로그의 구성 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정하십시오. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`uploadToCloudWatch`

(선택 사항) 시스템 로그를 Logs에 업로드할 CloudWatch 수 있습니다.

기본값: `false`

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 Greengrass nucleus 구성요소가 JSON 형식 로그를 출력하도록 구성한 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 로깅 형식 매개 변수 () `JSON` 를 지정하십시오. `logging.format`

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) Greengrass 시스템 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` Greengrass 시스템 로그 파일의 총 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 동일합니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 단위입니다. `diskSpaceLimit` 다음 옵션 중 하나를 선택합니다.

- `KB`— 킬로바이트
- `MB`— 메가바이트
- `GB`— 기가바이트

기본값: `KB`

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`componentLogsConfigurationMap`

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵. 이 맵의 각 `componentName` 개체는 구성 요소 또는 응용 프로그램의 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 Logs에 CloudWatch 업로드합니다.

> **Important**
>
> 구성 요소당 단일 구성 키를 사용하는 것이 좋습니다. 를 사용할 때 활발히 기록되는 로그 파일이 하나뿐인 파일 그룹만 대상으로 지정해야 `logFileRegex` 합니다. 이 권장 사항을 따르지 않으면 로그가 중복되어 업로드될 수 CloudWatch 있습니다. 단일 정규식을 사용하여 여러 활성 로그 파일을 대상으로 하는 경우 log manager v2.3.1로 업그레이드하고 예제 구성을 사용하여 구성을 변경하는 것이 좋습니다.

> **Note**
>
> v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 대신 목록을 계속 사용할 수 있습니다. `componentLogsConfiguration` `componentLogsConfigurationMap` 하지만 병합 및 재설정 업데이트를 사용하여 특정 구성 요소의 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration`매개변수에 대한 자세한 내용은 이 구성 요소의 v2.1.x용 구성 매개변수를 참조하십시오.

`componentName`

이 로그 구성을 위한 구성 *`componentName`*요소 또는 응용 프로그램의 로그 구성입니다. Greengrass 구성 요소의 이름 또는 다른 값을 지정하여 이 로그 그룹을 식별할 수 있습니다.

각 개체에는 다음 정보가 포함됩니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 특정 JSON 형식을 사용하는 경우에만 적용됩니다. 이 형식은 AWS IoT Greengrass로깅 모듈 리포지토리에서 찾을 수 있습니다. GitHub

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` 이 구성 요소 로그 파일의 전체 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 관련이 있습니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 `diskSpaceLimit` 단위입니다. 다음 옵션 중 하나를 선택합니다.

- `KB`— 킬로바이트
- `MB`— 메가바이트
- `GB`— 기가바이트

기본값: `KB`

`logFileDirectoryPath`

(선택 사항) 이 구성 요소의 로그 파일이 들어 있는 폴더의 경로입니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

기본값: */greengrass/v2*/logs.

`logFileRegex`

(선택 사항) 구성 요소나 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규 표현식입니다. 로그 관리자 구성 요소는 이 정규 표현식을 사용하여 의 폴더에 있는 로그 파일을 `logFileDirectoryPath` 식별합니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

구성 요소나 애플리케이션이 로그 파일을 회전시키는 경우 회전된 로그 파일 이름과 일치하는 정규식을 지정하십시오. 예를 들어 Hello World 애플리케이션의 로그를 **`hello_world\\\\w*.log`** 업로드하도록 지정할 수 있습니다. `\\\\w*`패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일을 일치시킵니다. 이 예제에서 로그 관리자는 다음 로그 파일을 업로드합니다.

- `hello_world.log`— Hello World 애플리케이션의 최신 로그 파일입니다.
- `hello_world_2020_12_15_17_0.log`— 헬로 월드 애플리케이션의 이전 로그 파일입니다.

기본값:*componentName*`\\\\w*.log`, 여기서 *ComponentName#* 이 로그 구성의 구성 요소 이름입니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규 표현식입니다. 정규 표현식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 이전 줄의 로그 메시지에 새 줄을 추가합니다.

기본적으로 로그 관리자 구성요소는 줄이 탭이나 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자가 해당 줄을 새 로그 메시지로 처리합니다. 그렇지 않으면 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성요소가 스택 트레이스와 같이 여러 줄에 걸쳐있는 메시지를 분할하지 않도록 합니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간 (초).

기본값: `300` (5분)

최소: `0.000001` (1마이크로초)

Example 예: 구성 병합 업데이트

다음 예제 구성은 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 로그에 업로드하도록 CloudWatch 지정합니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

Example 예: log manager v2.3.1을 사용하여 여러 활성 로그 파일을 업로드하도록 구성

다음 예제 구성은 여러 활성 로그 파일을 대상으로 하려는 경우 권장되는 예제입니다. 이 예제 구성은 업로드하려는 활성 로그 파일을 지정합니다 CloudWatch. 이 구성 예제 구성을 사용하면 다음과 일치하는 회전된 파일도 업로드됩니다. `logFileRegex` 이 예제 구성은 로그 관리자 v2.3.1에서 지원됩니다.

```
{
```

```
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.2.x

`logsUploaderConfiguration`

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그의 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`systemLogsConfiguration`

(선택 사항) Greengrass 핵 및 플러그인 구성 요소의 로그를 포함하는 AWS IoT Greengrass Core 소프트웨어 시스템 로그의 구성 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정하십시오. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`uploadToCloudWatch`

(선택 사항) 시스템 로그를 Logs에 업로드할 CloudWatch 수 있습니다.

기본값: `false`

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 Greengrass nucleus 구성요소가 JSON 형식 로그를 출력하도록 구성한 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 로깅 형식 매개 변수 () `JSON` 를 지정하십시오. `logging.format`

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`

- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) Greengrass 시스템 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` Greengrass 시스템 로그 파일의 총 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 동일합니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 단위입니다. `diskSpaceLimit` 다음 옵션 중 하나를 선택합니다.

- `KB`— 킬로바이트
- `MB`— 메가바이트
- `GB`— 기가바이트

기본값: `KB`

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`componentLogsConfigurationMap`

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵. 이 맵의 각 `componentName` 개체는 구성 요소 또는 응용 프로그램의 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 Logs에 CloudWatch 업로드합니다.

> **Note**
>
> v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 대신 `componentLogsConfiguration` 목록을 계속 사용할 수 있습니다. `componentLogsConfigurationMap` 하지만 병합 및 재설정 업데이트를 사용하

여 특정 구성 요소의 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration`매개변수에 대한 자세한 내용은 이 구성 요소의 v2.1.x용 구성 매개변수를 참조하십시오.

`componentName`

이 로그 구성을 위한 구성 *`componentName`*요소 또는 응용 프로그램의 로그 구성입니다. Greengrass 구성 요소의 이름 또는 다른 값을 지정하여 이 로그 그룹을 식별할 수 있습니다.

각 개체에는 다음 정보가 포함됩니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 특정 JSON 형식을 사용하는 경우에만 적용됩니다. 이 형식은 AWS IoT Greengrass로깅 모듈 리포지토리에서 찾을 수 있습니다. GitHub

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` 이 구성 요소 로그 파일의 전체 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 관련이 있습니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 `diskSpaceLimit` 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB— 킬로바이트
- MB— 메가바이트
- GB— 기가바이트

기본값: KB

`logFileDirectoryPath`

(선택 사항) 이 구성 요소의 로그 파일이 들어 있는 폴더의 경로입니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

기본값: */greengrass/v2*/logs.

`logFileRegex`

(선택 사항) 구성 요소나 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규 표현식입니다. 로그 관리자 구성 요소는 이 정규 표현식을 사용하여 의 폴더에 있는 로그 파일을 `logFileDirectoryPath` 식별합니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

구성 요소나 애플리케이션이 로그 파일을 회전시키는 경우 회전된 로그 파일 이름과 일치하는 정규식을 지정하십시오. 예를 들어 Hello World 애플리케이션의 로그를 **`hello_world\\\\w*.log`** 업로드하도록 지정할 수 있습니다. `\\\\w*`패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일을 일치시킵니다. 이 예제에서 로그 관리자는 다음 로그 파일을 업로드합니다.

- `hello_world.log`— Hello World 애플리케이션의 최신 로그 파일입니다.
- `hello_world_2020_12_15_17_0.log`— 헬로 월드 애플리케이션의 이전 로그 파일입니다.

기본값:*componentName*`\\\\w*.log`, 여기서 *ComponentName#* 이 로그 구성의 구성 요소 이름입니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규 표현식입니다. 정규 표현식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 이전 줄의 로그 메시지에 새 줄을 추가합니다.

기본적으로 로그 관리자 구성요소는 줄이 탭이나 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자가 해당 줄을 새 로그 메시지로 처리합니다. 그렇지 않으면 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성요소가 스택 트레이스와 같이 여러 줄에 걸쳐있는 메시지를 분할하지 않도록 합니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간 (초).

기본값: `300` (5분)

최소: `0.000001` (1마이크로초)

Example 예: 구성 병합 업데이트

다음 예제 구성은 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 로그에 업로드하도록 CloudWatch 지정합니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
```

```
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

`logsUploaderConfiguration`

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그의 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`systemLogsConfiguration`

(선택 사항) Greengrass 핵 및 플러그인 구성 요소의 로그를 포함하는 AWS IoT Greengrass Core 소프트웨어 시스템 로그의 구성 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정하십시오. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`uploadToCloudWatch`

(선택 사항) 시스템 로그를 Logs에 업로드할 CloudWatch 수 있습니다.

기본값: `false`

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 Greengrass nucleus 구성요소가 JSON 형식 로그를 출력하도록 구성한 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 로깅 형식 매개 변수 () `JSON` 를 지정하십시오. `logging.format`

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) Greengrass 시스템 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` Greengrass 시스템 로그 파일의 총 크기가 이 최대 크기를 초과

하면 AWS IoT Greengrass Core 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 동일합니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 단위입니다. `diskSpaceLimit` 다음 옵션 중 하나를 선택합니다.

- `KB`— 킬로바이트
- `MB`— 메가바이트
- `GB`— 기가바이트

기본값: `KB`

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`componentLogsConfiguration`

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 목록입니다. 이 목록의 각 구성은 구성 요소 또는 응용 프로그램의 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 Logs에 CloudWatch 업로드합니다.

각 개체에는 다음 정보가 들어 있습니다.

`componentName`

이 로그 구성의 구성 요소 또는 응용 프로그램 이름. Greengrass 구성 요소의 이름 또는 다른 값을 지정하여 이 로그 그룹을 식별할 수 있습니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 특정 JSON 형식을 사용하는 경우에만 적용됩니다. 이 형식은 AWS IoT Greengrass 로깅 모듈 리포지토리에서 찾을 수 있습니다. GitHub

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

`diskSpaceLimit`

(선택 사항) 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` 이 구성 요소 로그 파일의 전체 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 관련이 있습니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 `diskSpaceLimit` 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB— 킬로바이트
- MB— 메가바이트
- GB— 기가바이트

기본값: KB

`logFileDirectoryPath`

(선택 사항) 이 구성 요소의 로그 파일이 들어 있는 폴더의 경로입니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개변수를 지정할 필요가 없습니다.

기본값: */greengrass/v2*/logs.

`logFileRegex`

(선택 사항) 구성 요소나 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규 표현식입니다. 로그 관리자 구성 요소는 이 정규 표현식을 사용하여 의 폴더에 있는 로그 파일을 `logFileDirectoryPath` 식별합니다.

표준 출력 (stdout) 및 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에는 이 매개 변수를 지정할 필요가 없습니다.

구성 요소나 애플리케이션이 로그 파일을 회전시키는 경우 회전된 로그 파일 이름과 일치하는 정규식을 지정하십시오. 예를 들어 Hello World 애플리케이션의 로그를 **`hello_world\\\\w*.log`** 업로드하도록 지정할 수 있습니다. `\\\\w*`패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일을 일치시킵니다. 이 예제에서 로그 관리자는 다음 로그 파일을 업로드합니다.

- `hello_world.log`— Hello World 애플리케이션의 최신 로그 파일입니다.
- `hello_world_2020_12_15_17_0.log`— 헬로 월드 애플리케이션의 이전 로그 파일입니다.

기본값:*`componentName`*`\\\\w*.log`, 여기서 *`ComponentName#`* 이 로그 구성의 구성 요소 이름입니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규 표현식입니다. 정규 표현식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 이전 줄의 로그 메시지에 새 줄을 추가합니다.

기본적으로 로그 관리자 구성요소는 줄이 탭이나 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자가 해당 줄을 새 로그 메시지로 처리합니다. 그렇지 않으면 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성요소가 스택 트레이스와 같이 여러 줄에 걸쳐있는 메시지를 분할하지 않도록 합니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간 (초).

기본값: `300` (5분)

최소: `0.000001` (1마이크로초)

Example 예: 구성 병합 업데이트

다음 예제 구성은 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 로그에 업로드하도록 CloudWatch 지정합니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.0.x

`logsUploaderConfiguration`

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그의 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`systemLogsConfiguration`

(선택 사항) AWS IoT Greengrass 코어 소프트웨어 시스템 로그의 구성. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정하십시오. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`uploadToCloudWatch`

(선택 사항) 시스템 로그를 Logs에 업로드할 CloudWatch 수 있습니다.

기본값: `false`

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 Greengrass nucleus 구성요소가 JSON 형식 로그를 출력하도록 구성한 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 로깅 형식 매개 변수 () `JSON` 를 지정하십시오. `logging.format`

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) Greengrass 시스템 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` Greengrass 시스템 로그 파일의 총 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 동일합니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 단위입니다. `diskSpaceLimit` 다음 옵션 중 하나를 선택합니다.

- `KB`— 킬로바이트
- `MB`— 메가바이트
- `GB`— 기가바이트

기본값: `KB`

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`componentLogsConfiguration`

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 목록입니다. 이 목록의 각 구성은 구성 요소 또는 응용 프로그램의 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 Logs에 CloudWatch 업로드합니다.

각 개체에는 다음 정보가 들어 있습니다.

`componentName`

이 로그 구성의 구성 요소 또는 응용 프로그램 이름. Greengrass 구성 요소의 이름 또는 다른 값을 지정하여 이 로그 그룹을 식별할 수 있습니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 특정 JSON 형식을 사용하는 경우에만 적용됩니다. 이 형식은 AWS IoT Greengrass 로깅 모듈 리포지토리에서 찾을 수 있습니다. GitHub

여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`diskSpaceLimit`

(선택 사항) 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기 (지정한 단위) `diskSpaceLimitUnit` 이 구성 요소 로그 파일의 전체 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 매개변수는 Greengrass 핵 구성요소의 로그 크기 제한 매개변수 (`totalLogsSizeKB`) 와 관련이 있습니다. AWS IoT GreengrassCore 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) 의 `diskSpaceLimit` 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB— 킬로바이트
- MB— 메가바이트
- GB— 기가바이트

기본값: KB

`logFileDirectoryPath`

이 구성 요소의 로그 파일이 들어 있는 폴더의 경로입니다.

Greengrass 구성 요소의 로그를 업로드하려면 지정한 ***/greengrass/v2*/logs** 후 Greengrass 루트 */greengrass/v2* 폴더로 대체하십시오.

`logFileRegex`

구성 요소나 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규 표현식입니다. 로그 관리자 구성 요소는 이 정규 표현식을 사용하여 의 폴더에 있는 로그 파일을 `logFileDirectoryPath` 식별합니다.

Greengrass 구성 요소의 로그를 업로드하려면 회전된 로그 파일 이름과 일치하는 정규식을 지정하십시오. 예를 들어 Hello World 구성 요소의 로그를 **com.example.HelloWorld\\w*.log** 업로드하도록 지정할 수 있습니다. \\w*패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일을 일치시킵니다. 이 예제에서 로그 관리자는 다음 로그 파일을 업로드합니다.

- `com.example.HelloWorld.log`— Hello World 구성 요소의 최신 로그 파일입니다.
- `com.example.HelloWorld_2020_12_15_17_0.log`— 헬로 월드 구성 요소의 이전 로그 파일입니다. Greengrass 핵은 로그 파일에 회전하는 타임스탬프를 추가합니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다. CloudWatch

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규 표현식입니다. 정규 표현식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 이전 줄의 로그 메시지에 새 줄을 추가합니다.

기본적으로 로그 관리자 구성요소는 줄이 탭이나 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자가 해당 줄을 새 로그 메시지로 처리합니다. 그렇지 않으면 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성요소가 스택 트레이스와 같이 여러 줄에 걸쳐있는 메시지를 분할하지 않도록 합니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간 (초).

기본값: `300` (5분)

최소: `0.000001` (1마이크로초)

Example 예: 구성 병합 업데이트

다음 예제 구성은 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 로그에 업로드하도록 CloudWatch 지정합니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

사용량

로그 관리자 구성 요소는 다음 로그 그룹 및 로그 스트림에 업로드합니다.

2.1.0 and later

로그 그룹 이름

```
/aws/greengrass/componentType/region/componentName
```

로그 그룹 이름은 다음 변수를 사용합니다.

- `componentType`— 구성 요소의 유형으로, 다음 중 하나일 수 있습니다.
  - `GreengrassSystemComponent`— 이 로그 그룹에는 Greengrass 핵과 동일한 JVM에서 실행되는 핵 및 플러그인 구성 요소에 대한 로그가 포함됩니다. 구성 요소는 Greengrass 핵의 일부입니다.
  - `UserComponent`— 이 로그 그룹에는 일반 구성 요소, Lambda 구성 요소 및 디바이스의 기타 애플리케이션에 대한 로그가 포함됩니다. 구성 요소는 Greengrass 핵의 일부가 아닙니다.

  자세한 설명은 구성 요소 유형 섹션을 참조하세요.
- `region`— 코어 디바이스가 사용하는 AWS 지역.
- `componentName`— 구성 요소 이름. 시스템 로그의 경우 이 값은 입니다`System`.

로그 스트림 이름

```
/date/thing/thingName
```

로그 스트림 이름은 다음 변수를 사용합니다.

- `date`— 로그 날짜 (예:`2020/12/15`. 로그 관리자 구성 요소는 `yyyy/MM/dd` 형식을 사용합니다.
- `thingName`— 코어 디바이스의 이름.

> **Note**
>
> 사물 이름에 콜론 (`:`) 이 포함된 경우 로그 관리자는 콜론을 더하기 () 로 바꿉니다. +

2.0.x

로그 그룹 이름

```
/aws/greengrass/componentType/region/componentName
```

로그 그룹 이름은 다음 변수를 사용합니다.

- componentType— 구성 요소의 유형으로, 다음 중 하나일 수 있습니다.
  - GreengrassSystemComponent— 성분은 Greengrass 핵의 일부입니다.
  - UserComponent— 성분은 Greengrass 핵의 일부가 아닙니다. 로그 관리자는 장치의 Greengrass 구성 요소 및 기타 애플리케이션에 이 유형을 사용합니다.
- region— 코어 디바이스가 사용하는 AWS 지역.
- componentName— 구성 요소 이름. 시스템 로그의 경우 이 값은 입니다System.

로그 스트림 이름

```
/date/deploymentTargets/thingName
```

로그 스트림 이름은 다음 변수를 사용합니다.

- date— 로그 날짜 (예:2020/12/15. 로그 관리자 구성 요소는 yyyy/MM/dd 형식을 사용합니다.
- deploymentTargets— 배포 대상 항목에 구성 요소가 포함됩니다. 로그 관리자 구성 요소는 각 대상을 슬래시로 구분합니다. 로컬 배포의 결과로 구성 요소가 코어 장치에서 실행되는 경우 이 값은 입니다. LOCAL_DEPLOYMENT

  이름이 지정된 MyGreengrassCore 코어 디바이스가 있고 코어 디바이스에 두 개의 배포가 있는 경우를 예로 들어 보겠습니다.

  - 코어 기기를 대상으로 하는 배포,. MyGreengrassCore
  - 코어 디바이스를 포함하는 이름이 지정된 MyGreengrassCoreGroup 사물 그룹을 대상으로 하는 배포입니다.

  이 코어 디바이스의 deploymentTargets 용도는 다음과 같습니다thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup.
- thingName— 코어 디바이스의 이름.

로그 항목 형식.

Greengrass 핵은 문자열 또는 JSON 형식으로 로그 파일을 작성합니다. 시스템 로그의 경우 항목 `format` 필드를 설정하여 형식을 제어합니다. `logging` Greengrass nucleus 컴포넌트의 구성 파일에서 `logging` 항목을 찾을 수 있습니다. 자세한 내용은 Greengrass 핵 구성을 참조하십시오.

텍스트 형식은 자유 형식이며 모든 문자열을 사용할 수 있습니다. 다음 플릿 상태 서비스 메시지는 문자열 형식 로깅의 예입니다.

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

Greengrass CLI logs 명령으로 로그를 보거나 프로그래밍 방식으로 로그와 상호 작용하려면 JSON 형식을 사용해야 합니다. 다음 예제에서는 JSON 형태를 간략하게 설명합니다.

```
{
   "loggerName": <string>,
   "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
   "eventType": <string, optional>,
   "cause": <string, optional>,
   "contexts": {},
   "thread": <string>,
   "message": <string>,
   "timestamp": <epoch time> # Needs to be epoch time
}
```

구성 요소 로그의 출력을 제어하려면 구성 옵션을 사용할 수 있습니다. `minimumLogLevel` 이 옵션을 사용하려면 구성 요소가 로그 항목을 JSON 형식으로 작성해야 합니다. 시스템 로그 파일과 같은 형식을 사용해야 합니다.

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.3.7	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.6	버그 수정 및 개선 • 특정 오류의 로그 수준을 조정합니다.
2.3.5	개선 사항 로그 업로드 속도를 개선합니다. 그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.3.4	버그 수정 및 개선 • `periodicUploadIntervalSec` 파라미터를 소수 값으로 설정하는 서포트를 추가합니다. 최소값은 1마이크로초입니다. • 로그 관리자가 CloudWatch `putLogEvents` 제한을 준수하지 않는 문제를 수정합니다.
2.3.3	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.2	버그 수정 및 개선 • 로그 파일이 업로드되기 전에 삭제되지 않도록 공간 관리를 개선합니다. • 캐시 관리 문제를 수정합니다. • 추가 사소한 버그 수정 및 개선
2.3.1	버그 수정 및 개선 • 활성 로그 파일이 여러 개 있는 대상 파일 그룹이 중복 항목을 업로드하는 CloudWatch 문제를 수정합니다. • 추가적인 사소한 버그 수정 및 개선
2.3.0	**Note** 로그 관리자 2.3.0으로 업그레이드할 때는 Greengrass nucleus 2.9.1로 업그레이드하는 것이 좋습니다. 새로운 기능 새 파일이 교체될 때까지 기다리지 않고 활성 로그 파일을 처리하고 직접 업로드하여 로그 지연을 줄입니다. 버그 수정 및 개선 • 고유한 이름을 가진 파일을 회전할 때 로그 로테이션 지원을 개선합니다. • 추가 사소한 버그 수정 및 개선

버전	변경
2.2.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.4	버그 수정 및 개선 • 잘못된 구성을 처리할 때의 안정성을 개선합니다. • 추가 사소한 수정 및 개선.
2.2.3	버그 수정 및 개선 • 구성 요소가 다시 시작되거나 오류가 발생하는 특정 시나리오의 안정성을 개선합니다. • 특정 시나리오에서 대용량 로그 메시지와 대용량 로그 파일이 업로드되지 않는 문제를 수정합니다. • 이 구성 요소가 구성 재설정 업데이트를 처리하는 방식과 관련된 문제를 수정합니다. • `nulldiskSpaceLimit` 구성 값으로 인해 구성 요소를 배포할 수 없었던 문제를 수정합니다.
2.2.2	버그 수정 및 개선 • 256킬로바이트보다 큰 로그 메시지에 대한 지원을 추가합니다. 로그 관리자 구성요소는 이러한 대용량 로그 메시지를 로그 이벤트 타임스탬프가 동일한 여러 메시지로 분할합니다.
2.2.1	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.0	새 기능 • `componentLogsConfigurationMap` 구성 요소 로그 구성을 위한 맵 형식을 지원하는 구성 매개변수를 추가합니다. 맵의 각 `componentName` 개체는 구성 요소 또는 응용 프로그램의 로그 구성을 정의합니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	버그 수정 및 개선 • 특정 경우에 시스템 로그 구성이 업데이트되지 않던 문제를 수정합니다.
2.1.0	버그 수정 및 개선 • 표준 출력 (stdout) `logFileDirectoryPath` 및 `logFileRegex` 표준 오류 (stderr) 로 인쇄되는 Greengrass 구성 요소에 대한 기본값과 작동하는 기본값을 사용하십시오. • 로그를 Logs에 업로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. CloudWatch • 로그 스트림 이름의 콜론 문자 (`:`) 를 올바르게 처리하십시오. CloudWatch 로그 로그 스트림 이름은 콜론을 지원하지 않습니다. • 로그 스트림에서 사물 그룹 이름을 제거하여 로그 스트림 이름을 단순화합니다. • 정상 동작 중에 인쇄되는 오류 로그 메시지를 제거합니다.
2.0.x	초기 버전

기계 학습 구성 요소

AWS IoT GreengrassAmazon에서 학습한 모델 SageMaker 또는 Amazon S3에 저장된 사전 학습된 자체 모델을 사용하여 기계 학습 추론을 수행하기 위해 지원되는 디바이스에 배포할 수 있는 다음과 같은 기계 학습 구성 요소를 제공합니다.

AWS다음과 같은 범주의 기계 학습 구성 요소를 제공합니다.

- 모델 구성 요소 —기계 학습 모델을 Greengrass 아티팩트로 포함합니다.
- 런타임 구성 요소 - Greengrass 코어 장치에 기계 학습 프레임워크 및 해당 종속성을 설치하는 스크립트가 들어 있습니다.
- 추론 구성 요소 - 추론 코드를 포함하며 기계 학습 프레임워크를 설치하고 사전 학습된 기계 학습 모델을 다운로드하기 위한 구성 요소 종속성을 포함합니다.

AWS제공된 기계 학습 구성 요소의 샘플 추론 코드와 사전 학습된 모델을 사용하여 DLR 및 Lite를 사용하여 이미지 분류 및 객체 감지를 수행할 수 있습니다. TensorFlow Amazon S3에 저장된 자체 모델로 사용자 지정 기계 학습 추론을 수행하거나 다른 기계 학습 프레임워크를 사용하려면 이러한 공개 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 기계 학습 구성 요소를 생성할 수 있습니다. 자세한 설명은 머신 러닝 구성 요소 사용자 지정 섹션을 참조하세요.

AWS IoT Greengrass또한 Greengrass AWS 코어 장치에서 SageMaker Edge Manager 에이전트의 설치 및 수명 주기를 관리하기 위해 제공된 구성 요소가 포함되어 있습니다. SageMaker Edge Manager를 사용하면 Amazon SageMaker NEO로 컴파일된 모델을 코어 디바이스에서 직접 사용할 수 있습니다. 자세한 설명은 그린그래스 코어 디바이스에서 Amazon SageMaker Edge Manager 사용 섹션을 참조하세요.

다음 표에는 에서 사용할 수 있는 기계 학습 구성 요소가 나와 있습니다. AWS IoT Greengrass

> **Note**
>
> AWS제공되는 몇 가지 구성 요소는 Greengrass 핵의 특정 마이너 버전에 따라 다릅니다. 이러한 종속성 때문에 Greengrass 핵을 새 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 의존하는 핵의 특정 버전에 대한 자세한 내용은 해당 구성 요소 항목을 참조하십시오. 핵 업데이트에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트

구성 요소의 구성 요소 유형이 제네릭과 Lambda인 경우 구성 요소의 현재 버전은 제네릭 유형이고 구성 요소의 이전 버전은 Lambda 유형입니다.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
Lookout for Vision Edge Agent를 찾아보세요	Greengrass 코어 디바이스에 Amazon Lookout for Vision 런타임을 배포하므로 컴퓨터 비전을 사용하여 산업용 제품의 결함	일반	Linux	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
	을 찾을 수 있습니다.			
SageMaker 엣지 매니저	Greengrass 코어 디바이스에 Amazon SageMaker Edge Manager 에이전트를 배포합니다.	일반	Linux, Windows	아니요
DLR 이미지 분류	DLR 이미지 분류 모델 스토어와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 기기에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
DLR 객체 감지	DLR 개체 탐지 모델 저장소와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 개체 감지 모델을 다운로드하고, 지원되는 장치에서 개체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 이미지 분류 모델 스토어	샘플 ResNet-50개의 이미지 분류 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 객체 감지 모델 스토어	샘플 YoloV3 객체 감지 모델을 Greengrass 아티팩트로 포함하는 모델 컴포넌트입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
DLR 런타임	Greengrass 코어 디바이스에 DLR 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트가 포함된 런타임 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow 라이트 이미지 분류	TensorFlow Lite 이미지 분류 모델 저장소와 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 기기에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow 라이트 오브젝트 감지	TensorFlow Lite 개체 감지 모델 저장소 및 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 개체 감지 모델을 다운로드하고, 지원되는 장치에서 개체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow 라이트 이미지 분류 모델 스토어	샘플 MobileNet v1 모델을 Greengrass 아티팩트로 포함하는 모델 컴포넌트입니다.	일반	Linux, Windows	아니요
TensorFlow Lite 객체 감지 모델 스토어	샘플 싱글 샷 감지 (SSD) MobileNet 모델을 Greengrass 아티팩트로 포함하는 모델 컴포넌트입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow 라이트 런타임	Greengrass 코어 기기에 TensorFlow Lite 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트가 포함된 런타임 구성 요소입니다.	일반	Linux, Windows	아니요

Lookout for Vision Edge Agent를 찾아보세요

Lookout for Vision Edge Agent 구성 요소 `aws.iot.lookoutvision.EdgeAgent` () 는 컴퓨터 비전을 사용하여 산업용 제품의 시각적 결함을 찾아내는 로컬 Amazon Lookout for Vision 런타임 서버를 설치합니다.

이 구성 요소를 사용하려면 Lookout for Vision 머신 러닝 모델 구성 요소를 만들고 배포하세요. 이러한 기계 학습 모델은 모델 학습에 사용하는 이미지의 패턴을 찾아 이미지에 이상이 있는지 예측합니다. 그런 다음 이 런타임 구성 요소에 이미지와 비디오 스트림을 제공하여 기계 학습 모델을 사용하여 이상을 탐지하는 클라이언트 애플리케이션 구성 요소라고 하는 사용자 지정 Greengrass 구성 요소를 개발하고 배포할 수 있습니다.

Lookout for Vision Edge Agent API를 사용하여 다른 Greengrass 구성 요소의 이 구성 요소와 상호 작용할 수 있습니다. 이 API는 원격 프로시저 호출을 위한 프로토콜인 gRPC를 사용하여 구현됩니다. 자세한 내용은 Amazon Lookout for Vision 개발자 안내서의 클라이언트 애플리케이션 구성 요소 작성 및 Lookout for Vision Edge Agent API 참조를 참조하십시오.

이 구성 요소를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- Amazon Lookout for Vision
- Amazon Lookout for Vision이란 무엇입니까? Amazon Lookout for Vision 개발자 가이드에서
- Amazon Lookout for Vision 개발자 가이드에서 Lookout for Vision 모델 생성

- Amazon Lookout for Vision 개발자 가이드에서 엣지 디바이스에서 Lookout for Vision 모델을 사용하는 방법

> **Note**
>
> Lookout for Vision Edge Agent 구성 요소는 다음과 같은 경우에만 사용할 수 있습니다. AWS 리전
>
> - 미국 동부(오하이오)
> - 미국 동부(버지니아 북부)
> - 미국 서부(오리건)
> - 유럽(프랑크푸르트)
> - 유럽(아일랜드)
> - 아시아 태평양(도쿄)
> - 아시아 태평양(서울)

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.2.x
- 1.1.x

- 1.0.x
- 0.1.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 그린그래스 코어 디바이스는 Armv8 (AArch64) 또는 x86_64 아키텍처를 사용해야 합니다.
- 이 구성 요소의 버전 1.0.0 이상을 사용하는 경우 Python 3.8 또는 Python 3.9 (포함) 는 `pip` Greengrass 코어 장치에 설치됩니다.

  이 컴포넌트의 버전 0.1.x를 사용하는 경우, Python 3.7 (포함`pip`) 은 Greengrass 코어 디바이스에 설치됩니다.

  > **Important**
  >
  > 기기에는 다음과 같은 Python 버전 중 하나가 있어야 합니다. 이 구성 요소는 이후 버전의 Python을 지원하지 않습니다.

- 그래픽 처리 장치 (GPU) 추론을 사용하려면 코어 기기가 다음 요구 사항을 충족해야 합니다. 이 구성 요소 버전 1.1.0 이상에서는 GPU 추론이 선택 사항입니다.
  - CUDA를 지원하는 그래픽 처리 장치 (GPU). 자세한 내용은 CUDA 툴킷 설명서에서 CUDA 지원 GPU가 있는지 확인을 참조하십시오.
  - CuDNN, CUDA, TensorRT가 그린그래스 코어 디바이스에 설치되었습니다.
    - 젯슨 나노 또는 젯슨 자비에르, cuDNN, CUDA 및 TensorRT와 같은 NVIDIA 젯슨 디바이스에는 NVIDIA와 함께 설치되어 있습니다. JetPack 변경할 필요는 없습니다. 이 구성 요소는 JetPack 4.4, JetPack4.5, JetPack 4.5.1 및 JetPack 4.6.1을 지원합니다.

> **Important**
>
> 다음 버전 중 하나를 설치하고 다른 버전은 설치하지 않아야 합니다. JetPack Lookout for Vision 서비스는 이러한 플랫폼을 위한 JetPack 컴퓨터 비전 모델을 컴파일합니다.

- NVIDIA Ampere 마이크로아키텍처가 탑재된 GPU (또는 GPU의 컴퓨팅 파워가 8.0인 경우) 가 설치된 x86 디바이스에서는 다음을 수행하십시오.
  - NVIDIA cuDNN 설치 가이드의 지침에 따라 cuDNN을 설치합니다.
  - 리눅스용 NVIDIA CUDA 설치 가이드의 지침에 따라 CUDA 버전 11.2를 설치합니다.
  - 엔비디아 텐서RT 문서의 지침에 따라 텐서RT 버전 8.2.0을 설치합니다.
- 암페어 이전의 NVIDIA 아키텍처를 사용하는 GPU를 사용하는 x86 기기 (또는 GPU의 컴퓨팅 용량이 8.0 미만인 경우) 에서는 다음을 수행하십시오.
  - NVIDIA cuDNN 설치 가이드의 지침에 따라 cuDNN을 설치합니다.
  - 리눅스용 NVIDIA CUDA 설치 가이드의 지침에 따라 CUDA 버전 10.2를 설치합니다.
  - NVIDIA TensorRT 설명서의 지침에 따라 TensorRT 버전 7.1.3 이상 (버전 8.0.0 이전 버전) 을 설치합니다.
- 이 구성 요소를 실행하는 시스템 사용자는 기기의 GPU에 액세스할 수 있는 시스템 그룹의 구성원이어야 합니다. 이 그룹의 이름은 운영 체제에 따라 다릅니다. 운영 체제 및 GPU 설명서를 참조하여 이 시스템 그룹의 이름을 확인하십시오.

  예를 들어 NVIDIA Jetson 디바이스에서 이 그룹의 이름은 이며video, 다음 명령을 실행하여 이 그룹에 시스템 사용자를 추가할 수 있습니다. *ggc_user# ### ###* 이름으로 바꾸십시오.

  ```
  sudo usermod -aG video ggc_user
  ```

의존성

이 구성 요소에는 종속성이 없습니다.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`Socket`

(선택 사항) Edge Agent가 작동하는 파일 소켓. Lookout for Vision 모델 구성 요소는 이 파일 소켓을 사용하여 Edge Agent와 통신합니다. 이 매개변수를 변경하는 경우 Lookout for Vision 모델 구성 요소를 배포할 때 동일한 값을 지정해야 합니다.

기본값: `unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock`

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass 루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.2.0	일반적인 버그 수정 및 개선입니다.
1.1.9	일반적인 버그 수정 및 개선입니다.
1.1.8	일반적인 버그 수정 및 개선입니다.
1.1.7	새로운 기능 • Lookout for Vision Edge Agent 가상 환경에 `opencv-python-headless` 패키지를 설치합니다. 버그 수정 및 개선 • 신뢰도 점수 계산을 개선합니다.

버전	변경
	• 히트맵 모델 마스크의 크기를 원본 파일 크기로 조정합니다. • 일반적인 버그 수정 및 개선입니다.
1.1.6	새로운 기능 결과에 새 값을 추가했습니다. `DetectAnomalies` • `anomaly_score` — 이미지의 변칙성을 나타내는 0.0에서 1.0 사이의 숫자입니다. • `anomaly_threshold` — 모델 훈련 중에 설정된 임계값으로, 변칙 영상과 일반 영상의 경계를 결정합니다. 일반적인 버그 수정 및 개선입니다.
1.1.4	새로운 기능 사용 가능한 경우 이미지 크기 조정을 위한 OpenCV 지원이 추가되었습니다. Edge 에이전트는 OpenCV를 사용할 수 없을 때 필로우를 사용합니다. 버그 수정 및 개선 일반적인 버그 수정 및 개선입니다.
1.1.3	일반적인 버그 수정 및 개선입니다.
1.1.1	일반적인 버그 수정 및 개선입니다.
1.1.0	새로운 기능 • 이미지의 이상을 식별하는 이미지 분할 모델에 대한 지원을 추가합니다. • CPU 추론 지원이 추가되어 GPU가 없는 코어 디바이스에서도 Lookout for Vision 모델을 사용할 수 있습니다. 버그 수정 및 개선 • 일반적인 버그 수정 및 개선입니다.

버전	변경
1.0.0	이 버전의 Lookout for Vision Edge Agent 구성 요소를 사용하려면 버전 0.1.x와 다른 버전의 Python이 필요합니다. v0.1.x에서 v1.x로 업그레이드하려면 코어 기기에 설치된 Python을 업그레이드해야 합니다. 버그 수정 및 개선 • 일반적인 버그 수정 및 개선입니다.
0.1.37	일반적인 버그 수정 및 개선입니다.
0.1.36	초기 버전

SageMaker 엣지 매니저

> **Important**
>
> SageMaker 엣지 매니저는 2024년 4월 26일에 단종됩니다. 에지 장치에 모델을 계속 배포하는 방법에 대한 자세한 내용은 SageMaker Edge Manager 수명 종료를 참조하십시오.

Amazon SageMaker Edge Manager 구성 요소 (`aws.greengrass.SageMakerEdgeManager`) 는 SageMaker 에지 관리자 에이전트 바이너리를 설치합니다.

SageMaker Edge Manager는 에지 디바이스에 대한 모델 관리를 제공하므로 다양한 엣지 디바이스에서 기계 학습 모델을 최적화, 보호, 모니터링 및 유지할 수 있습니다. SageMaker Edge Manager 구성 요소는 코어 장치에 SageMaker Edge Manager 에이전트의 수명 주기를 설치하고 관리합니다. 또한 SageMaker Edge Manager를 사용하여 SageMaker NEO로 컴파일된 모델을 Greengrass 코어 디바이스의 모델 구성 요소로 패키징하고 사용할 수 있습니다. 코어 디바이스에서 SageMaker Edge Manager 에이전트를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 그린그래스 코어 디바이스에서 Amazon SageMaker Edge Manager 사용

SageMaker 에지 관리자 구성 요소 v1.3.x는 에지 관리자 에이전트 바이너리 v1.20220822.836f3023을 설치합니다. Edge Manager 에이전트 바이너리 버전에 대한 자세한 내용은 Edge Manager 에이전트를 참조하십시오.

> **Note**
>
> SageMaker Edge Manager 구성 요소는 AWS 리전 다음과 같은 경우에만 사용할 수 있습니다.
>
> - 미국 동부(오하이오)
> - 미국 동부(버지니아 북부)
> - 미국 서부(오레곤)
> - EU(프랑크푸르트)
> - EU(아일랜드)
> - 아시아 태평양(도쿄)

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 아마존 리눅스 2, 데비안 기반 리눅스 플랫폼 (x86_64 또는 Armv8) 또는 윈도우 (x86_64) 에서 실행되는 그린그래스 코어 디바이스. 계정이 없는 경우 자습서: AWS IoT Greengrass V2 시작하기 단원을 참조하십시오.
- Python 3.6 이상 (사용 중인 Python `pip` 버전용 포함) 이 코어 기기에 설치되어 있어야 합니다.
- Greengrass 장치 역할은 다음과 같이 구성되었습니다.
  - 다음 IAM 정책 예제와 같이 역할을 `credentials.iot.amazonaws.com` 허용하고 `sagemaker.amazonaws.com` 역할을 맡을 수 있는 신뢰 관계입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- AmazonSageMakerEdgeDeviceFleetPolicyIAM 관리형 정책.
- `s3:PutObject`작업은 다음 IAM 정책 예제에 나와 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Greengrass 코어 AWS 계정 디바이스와 AWS 리전 동일한 디바이스에서 생성된 Amazon S3 버킷 SageMaker Edge Manager에는 에지 디바이스 플릿을 생성하고 추론 실행의 샘플 데이터를 디바이스에 저장하기 위한 S3 버킷이 필요합니다. S3 버킷 생성에 대한 자세한 내용은 Amazon S3 시작하기를 참조하십시오.
- Greengrass 코어 디바이스와 동일한 AWS IoT 역할 별칭을 사용하는 SageMaker 에지 디바이스 플릿입니다. 자세한 설명은 에지 디바이스 플릿 생성 섹션을 참조하세요.
- Greengrass 코어 디바이스가 Edge 디바이스 플릿에 에지 SageMaker 디바이스로 등록되었습니다. 에지 디바이스 이름은 코어 디바이스의 AWS IoT 사물 이름과 일치해야 합니다. 자세한 설명은 그린그래스 코어 디바이스 등록 섹션을 참조하세요.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
edge.sagemaker. *region*.amazonaws.com	443	예	디바이스 등록 상태를 확인하고 메트릭을 로 전송하십시오. SageMaker
*.s3.amazonaws.com	443	예	지정한 S3 버킷에 캡처 데이터를 업로드합니다. 데이터를 업로드하는 각 버킷의 * 이름으로 바꿀 수 있습니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

1.3.5

다음 표에는 이 구성 요소의 버전 1.3.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.3.4

다음 표에는 이 구성 요소의 버전 1.3.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.3.3

다음 표에는 이 구성 요소의 버전 1.3.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.3.2

다음 표에는 이 구성 요소의 버전 1.3.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.3.1

다음 표에는 이 구성 요소의 버전 1.3.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.1.1 - 1.3.0

다음 표에는 이 구성 요소의 버전 1.1.1 - 1.3.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.1.0

다음 표에는 이 구성 요소의 버전 1.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.0.3

다음 표에는 이 구성 요소의 버전 1.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	>=0.0.0	하드

1.0.1 and 1.0.2

다음 표에는 이 구성 요소의 버전 1.0.1 및 1.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트
토큰 교환 서비스	>=0.0.0	하드

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트
토큰 교환 서비스	>=0.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

> **Note**
>
> 이 섹션에서는 구성 요소에 설정하는 구성 매개 변수에 대해 설명합니다. 해당 SageMaker Edge Manager 구성에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 Edge Manager 에이전트를 참조하십시오.

`DeviceFleetName`

Greengrass 코어 디바이스가 포함된 SageMaker Edge Manager 디바이스 플릿의 이름입니다.

이 구성 요소를 배포할 때 구성 업데이트에서 이 매개 변수의 값을 지정해야 합니다.

`BucketName`

캡처된 추론 데이터를 업로드할 S3 버킷의 이름. 버킷 이름에는 문자열이 `sagemaker` 포함되어야 합니다.

`CaptureDataDestination`로 설정하거나 로 `Cloud` 설정한 `CaptureDataPeriodicUpload` 경우 이 구성 요소를 배포할 때 구성 업데이트에서 이 파라미터의 값을 지정해야 합니다. `true`

> **Note**
>
> 캡처 데이터는 향후 분석을 위해 추론 입력, 추론 결과 및 추가 추론 데이터를 S3 버킷 또는 로컬 디렉터리에 업로드하는 데 사용하는 SageMaker 기능입니다. SageMaker Edge Manager에서 캡처 데이터를 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 모델 관리를 참조하십시오.

`CaptureDataBatchSize`

(선택 사항) 에이전트가 처리하는 캡처 데이터 요청 배치의 크기. 이 값은 에서 지정한 버퍼 크기보다 작아야 `CaptureDataBufferSize` 합니다. 버퍼 크기의 절반을 초과하지 않는 것이 좋습니다.

에이전트는 버퍼의 요청 수가 해당 수에 도달하거나 `CaptureDataPushPeriodSeconds` 간격이 경과할 때 (둘 중 먼저 발생하는 시점) 요청 일괄 처리를 처리합니다. `CaptureDataBatchSize`

기본값: `10`

`CaptureDataBufferSize`

(선택 사항) 버퍼에 저장된 캡처 데이터 요청의 최대 수입니다.

기본값: `30`

`CaptureDataDestination`

(선택 사항) 캡처한 데이터를 저장하는 대상. 이 매개변수는 다음 값을 가질 수 있습니다.

- `Cloud`—캡처된 데이터를 지정한 S3 버킷에 업로드합니다. `BucketName`

- `Disk`—캡처된 데이터를 구성 요소의 작업 디렉토리에 씁니다.

지정하는 `Disk` 경우 로 `CaptureDataPeriodicUpload` 설정하여 캡처된 데이터를 S3 버킷에 정기적으로 업로드하도록 선택할 수도 있습니다. `true`

기본값: `Cloud`

`CaptureDataPeriodicUpload`

(선택 사항) 캡처된 데이터를 정기적으로 업로드할지 여부를 지정하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

로 설정한 `true` 경우 이 파라미터를 `CaptureDataDestination` 로 `Disk` 설정하고 에이전트가 캡처된 데이터를 S3 버킷에 정기적으로 업로드하도록 하려면 이 파라미터를 로 설정합니다.

기본값: `false`

`CaptureDataPeriodicUploadPeriodSeconds`

(선택 사항) SageMaker Edge Manager 에이전트가 캡처된 데이터를 S3 버킷에 업로드하는 간격(초). 로 설정한 경우 이 파라미터를 사용하십시오`CaptureDataPeriodicUpload`. `true`

기본값: `8`

`CaptureDataPushPeriodSeconds`

(선택 사항) SageMaker Edge Manager 에이전트가 버퍼의 캡처 데이터 일괄 요청을 처리하는 간격(초) 입니다.

에이전트는 버퍼의 요청 수가 해당 수와 일치하거나 `CaptureDataPushPeriodSeconds` 간격이 경과할 때 (둘 중 먼저 발생하는 시점) 요청 일괄 처리를 처리합니다. `CaptureDataBatchSize`

기본값: `4`

`CaptureDataBase64EmbedLimit`

(선택 사항) Edge Manager 에이전트가 업로드하는 SageMaker 캡처된 데이터의 최대 크기 (바이트).

기본값: `3072`

`FolderPrefix`

(선택 사항) 에이전트가 캡처된 데이터를 쓰는 폴더의 이름. `CaptureDataDestination`로 `Disk` 설정하면 에이전트는 지정된 디렉터리에 폴더를 만듭니다`CaptureDataDiskPath`.

`CaptureDataDestination`로 설정하거나 로 `Cloud` 설정하면 `CaptureDataPeriodicUpload` 에이전트가 S3 버킷에 폴더를 생성합니다. `true`

기본값: `sme-capture`

`CaptureDataDiskPath`

이 기능은 v1.1.0 이상 버전의 SageMaker Edge Manager 구성 요소에서 사용할 수 있습니다.

(선택 사항) 에이전트가 캡처된 데이터 폴더를 만드는 폴더의 경로입니다. `CaptureDataDestination`로 `Disk` 설정하면 에이전트는 캡처된 데이터 폴더를 이 디렉터리에 만듭니다. 이 값을 지정하지 않으면 에이전트는 구성 요소의 작업 디렉터리에 캡처된 데이터 폴더를 만듭니다. `FolderPrefix`매개 변수를 사용하여 캡처된 데이터 폴더의 이름을 지정합니다.

기본값: */greengrass/v2*`/work/aws.greengrass.SageMakerEdgeManager/capture`

`LocalDataRootPath`

이 기능은 v1.2.0 이상 버전의 SageMaker Edge Manager 구성 요소에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소가 코어 장치에 다음 데이터를 저장하는 경로:

- `DbEnable`로 설정한 경우의 런타임 데이터용 로컬 데이터베이스`true`.
- SageMaker 설정 `DeploymentEnable` 시 이 구성 요소가 자동으로 다운로드하는 NEO 컴파일된 모델입니다. `true`

기본값: */greengrass/v2*`/work/aws.greengrass.SageMakerEdgeManager`

`DbEnable`

(선택 사항) 구성 요소에 장애가 발생하거나 장치 전원이 꺼지는 경우 이 구성 요소가 런타임 데이터를 로컬 데이터베이스에 저장하여 데이터를 보존하도록 할 수 있습니다.

이 데이터베이스에는 코어 디바이스의 파일 시스템에 5MB의 스토리지가 필요합니다.

기본값: `false`

`DeploymentEnable`

이 기능은 v1.2.0 이상 버전의 SageMaker Edge Manager 구성 요소에서 사용할 수 있습니다.

(선택 사항) Amazon S3에 업로드한 모델로부터 SageMaker NEO 컴파일된 모델을 자동으로 검색하도록 이 구성 요소를 활성화할 수 있습니다. Amazon S3에 새 모델을 업로드한 후 SageMaker Studio 또는 SageMaker API를 사용하여 새 모델을 이 코어 디바이스에 배포합니다. 이 기능을 활성

화하면 배포를 생성할 필요 없이 새 모델을 코어 디바이스에 AWS IoT Greengrass 배포할 수 있습니다.

> **Important**
>
> 이 기능을 사용하려면 로 `DbEnable` 설정해야 `true` 합니다. 이 기능은 로컬 데이터베이스를 사용하여 에서 검색한 모델을 추적합니다. AWS 클라우드

기본값: `false`

`DeploymentPollInterval`

이 기능은 v1.2.0 이상 버전의 SageMaker Edge Manager 구성 요소에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소가 다운로드할 새 모델을 확인하는 데 걸리는 시간 (분). 로 설정하면 이 옵션이 `DeploymentEnable` 적용됩니다`true`.

기본값: `1440` (1일)

`DLRBackendOptions`

이 기능은 v1.2.0 이상 버전의 SageMaker Edge Manager 구성 요소에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소가 사용하는 DLR 런타임에 설정할 DLR 런타임 플래그입니다. 다음 플래그를 설정할 수 있습니다.

- `TVM_TENSORRT_CACHE_DIR`— TensorRT 모델 캐싱을 활성화합니다. 읽기/쓰기 권한이 있는 기존 폴더의 절대 경로를 지정합니다.
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`— TensorRT 모델 캐시 폴더의 상한을 지정합니다. 디렉터리 크기가 이 한도를 초과하면 가장 적게 사용되는 캐시된 엔진이 삭제됩니다. 기본값은 512MB입니다.

예를 들어 이 매개변수를 다음 값으로 설정하여 TensorRT 모델 캐싱을 활성화하고 캐시 크기를 800MB로 제한할 수 있습니다.

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;
 TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

(선택 사항) 디버그 로깅을 활성화할지 여부를 지정하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

기본값: `false`

`UnixSocketName`

(선택 사항) 코어 디바이스의 SageMaker Edge Manager 소켓 파일 디스크립터 위치.

기본값: `/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example 예: 구성 병합 업데이트

다음 예제 구성은 코어 디바이스가 의 일부이고 에이전트가 *MyEdgeDeviceFleet*디바이스와 S3 버킷 모두에 캡처 데이터를 기록하도록 지정합니다. 이 구성은 디버그 로깅도 활성화합니다.

```
{
    "DeviceFleetName": "MyEdgeDeviceFleet",
    "BucketName": "DOC-EXAMPLE-BUCKET",
    "CaptureDataDestination": "Disk",
    "CaptureDataPeriodicUpload": "true",
    "SagemakerEdgeLogVerbose": "true"
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail
 10 -Wait
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.3.5	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
1.3.4	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
1.3.3	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
1.3.2	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
1.3.1	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
1.3.0	새로운 기능 • TensorRT 캐시 디스크 크기 관리에 대한 지원을 추가합니다. • 디스크에 캐시된 모델의 크기 제한을 설정하는 선택적 TVM_TENSORRT_CACHE_DISK_SIZE_MB 플래그를 DLR BackendOptions 매개변수에 추가합니다. 개선 사항 • 향상된 예측 동시성을 제공합니다. 이를 통해 GPU와 같은 장치 가속기 엔진을 더 잘 사용할 수 있습니다.

버전	변경
1.2.0	새로운 기능 • Amazon S3에 업로드한 SageMaker NEO 컴파일 모델을 자동으로 검색하도록 이 구성 요소에 대한 지원을 추가합니다. 이 기능을 활성화하면 배포를 생성할 필요 없이 새 모델을 코어 디바이스에 배포할 수 있습니다. AWS IoT Greengrass • 구성 요소에 장애가 발생하거나 장치 전원이 꺼지는 경우 이 구성 요소가 런타임 데이터를 보존하는 데 사용하는 백업 데이터베이스에 대한 지원을 추가합니다. • 이 구성 요소를 구성할 때 DLR 런타임 플래그를 구성할 수 있도록 지원을 추가합니다.
1.1.1	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
1.1.0	새로운 기능 • Amazon Linux 2를 실행하는 Greengrass 코어 디바이스에 대한 지원을 추가합니다. • 새 `CaptureDataDiskPath` 구성 매개변수를 추가합니다. 이 매개 변수를 사용하여 장치에서 캡처된 데이터 폴더의 경로를 지정할 수 있습니다. 버그 수정 및 개선 • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.2	버그 수정 및 개선 구성 요소 수명 주기에서 설치 스크립트를 업데이트합니다. 이제 이 구성 요소를 배포하기 전에 코어 기기에 Python 3.6 이상 버전 (사용 중인 Python 버전 포함`pip`) 이 설치되어 있어야 합니다.
1.0.1	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.0	초기 버전

DLR 이미지 분류

DLR 이미지 분류 구성 요소 (`aws.greengrass.DLRImageClassification`) 에는 딥러닝 런타임 및 resnet-50 모델을 사용하여 이미지 분류 추론을 수행하기 위한 샘플 추론 코드가 포함되어 있습니다. 이 구성 요소는 DLR 이미지 분류 모델 스토어 변형과 구성 요소를 종속성으로 사용하여 DLR 런타임 DLR 및 샘플 모델을 다운로드합니다.

이 추론 구성 요소를 사용자 지정 학습된 DLR 모델과 함께 사용하려면 종속 모델 저장소 구성 요소의 사용자 지정 버전을 만드십시오. 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 추론 구성 요소를 만들 수 있습니다.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.13

다음 표에는 이 구성 요소의 버전 2.1.13에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

종속성	호환되는 버전	종속성 유형
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

종속성	호환되는 버전	종속성 유형
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.4 - 2.1.5

다음 표에는 이 구성 요소의 버전 2.1.4 ~ 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

종속성	호환되는 버전	종속성 유형
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

종속성	호환되는 버전	종속성 유형
DLR 이미지 분류 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	~2.0.0	소프트
DLR 이미지 분류 모델 스토어	~2.0.0	하드
DLR	~1.3.0	소프트

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

2.1.x

`accessControl`

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 하는 권한 부여 정책이 포함된 객체입니다.

기본값:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
```

```
            ],
            "resources": [
               "ml/dlr/image-classification"
            ]
         }
      }
   }
```

`PublishResultsOnTopic`

(선택 사항) 추론 결과를 게시하려는 주제입니다. 이 값을 수정하는 경우 `accessControl` 매개 변수의 값도 사용자 지정 주제 이름과 일치하도록 수정해야 합니다. `resources`

기본값: `ml/dlr/image-classification`

`Accelerator`

사용하려는 액셀러레이터. 지원되는 값은 `cpu` 및 `gpu`입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 가속만 지원합니다. 다른 사용자 지정 모델에서 GPU 가속을 사용하려면 사용자 지정 모델 구성 요소를 만들어 공개 모델 구성 요소를 재정의하십시오.

기본값: `cpu`

`ImageDirectory`

(선택 사항) 추론 구성 요소가 이미지를 읽는 기기의 폴더 경로입니다. 읽기/쓰기 권한이 있는 장치의 모든 위치로 이 값을 수정할 수 있습니다.

기본값: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/image_classification/sample_images/`

> **Note**
>
> 값을 `UseCamera` ~로 `true` 설정하면 이 구성 매개변수는 무시됩니다.

`ImageName`

(선택 사항) 추론 구성 요소가 메이크 예측의 입력으로 사용하는 이미지의 이름입니다. 구성요소는 에 지정된 폴더에서 이미지를 찾습니다. `ImageDirectory` 기본적으로 구성 요소는 기본 이

미지 디렉터리의 샘플 이미지를 사용합니다. AWS IoT Greengrassjpeg, jpgpng, 및 같은 이미지 형식을 지원합니다npy.

기본값: cat.jpeg

> **Note**
>
> 값을 UseCamera 로 설정하는 true 경우 이 구성 매개변수는 무시됩니다.

`InferenceInterval`

(선택 사항) 추론 코드로 예측한 각 예측 사이의 시간 (초). 샘플 추론 코드는 무한정 실행되며 지정된 시간 간격으로 예측을 반복합니다. 예를 들어 카메라로 촬영한 이미지를 실시간 예측에 사용하려는 경우 이 간격을 더 짧게 변경할 수 있습니다.

기본값: 3600

`ModelResourceKey`

(선택 사항) 종속 공개 모델 구성 요소에 사용되는 모델. 공개 모델 구성 요소를 사용자 지정 구성 요소로 재정의하는 경우에만 이 매개 변수를 수정하십시오.

기본값:

```
{
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "windows": "DLR-resnet50-win-cpu-ImageClassification"
}
```

`UseCamera`

(선택 사항) Greengrass 코어 장치에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 true 및 false입니다.

이 값을 로 true 설정하면 샘플 추론 코드가 기기의 카메라에 액세스하여 캡처한 이미지에 대해 로컬에서 추론을 실행합니다. ImageName및 ImageDirectory 매개변수의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 캡처한 이미지를 저장하는 위치에 대한 읽기/쓰기 권한이 있는지 확인하십시오.

기본값: `false`

> **Note**
>
> 이 구성 요소의 레시피를 보면 구성 매개변수가 기본 `UseCamera` 구성에 표시되지 않습니다. 하지만 구성 요소를 배포할 때 구성 병합 업데이트에서 이 매개 변수의 값을 수정할 수 있습니다.
>
> `UseCamera`로 `true` 설정하면 런타임 구성 요소가 만든 가상 환경에서 추론 구성 요소가 카메라에 액세스할 수 있도록 심볼릭 링크도 만들어야 합니다. 샘플 추론 컴포넌트와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 구성 요소 구성 업데이트

2.0.x

`MLRootPath`

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값을 이 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 장치의 모든 위치로 수정할 수 있습니다.

기본값: *`/greengrass/v2`*`/work/variant.DLR/greengrass_ml`

기본값: *`/greengrass/v2`*`/work/variant.TensorFlowLite/greengrass_ml`

`Accelerator`

사용하려는 액셀러레이터. 지원되는 값은 `cpu` 및 `gpu`입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 가속만 지원합니다. 다른 사용자 지정 모델에서 GPU 가속을 사용하려면 사용자 지정 모델 구성 요소를 만들어 공개 모델 구성 요소를 재정의하십시오.

기본값: `cpu`

`ImageName`

(선택 사항) 추론 구성요소가 예측의 입력으로 사용하는 이미지의 이름. 구성요소는 에 지정된 폴더에서 이미지를 찾습니다. `ImageDirectory` 기본 위치는 입니다*`MLRootPath`*`/images`. AWS IoT Greengrass`jpeg`, `jpg`, `png`, 및 같은 이미지 형식을 지원합니다`npy`.

기본값: `cat.jpeg`

`InferenceInterval`

(선택 사항) 추론 코드로 예측한 각 예측 사이의 시간 (초). 샘플 추론 코드는 무한정 실행되며 지정된 시간 간격으로 예측을 반복합니다. 예를 들어 카메라로 촬영한 이미지를 실시간 예측에 사용하려는 경우 이 간격을 더 짧게 변경할 수 있습니다.

기본값: `3600`

`ModelResourceKey`

(선택 사항) 종속 공개 모델 구성 요소에 사용되는 모델. 공개 모델 구성 요소를 사용자 지정 구성 요소로 재정의하는 경우에만 이 매개 변수를 수정하십시오.

기본값:

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log -
Tail 10 -Wait
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.13	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.12	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.11	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.8	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	구성 요소가 모두 출시되었습니다. AWS 리전
2.1.4	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다. 이 버전은 유럽 (런던) 에서 사용할 수 없습니다 (). `eu-west-2`
2.1.3	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.1	새로운 기능 • 딥러닝 런타임 v1.6.0을 사용하십시오. • Armv8 (AArch64) 플랫폼에서 샘플 이미지 분류에 대한 지원을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다. • 샘플 추론을 위한 카메라 통합을 활성화하십시오. 새 `UseCamera` 구성 매개변수를 사용하여 샘플 추론 코드를 활성화하여 Greengrass 코어 장치의 카메라에 액세스하고 캡처한 이미지에서 로컬로 추론을 실행할 수 있습니다. • 추론 결과를 게시하기 위한 지원을 에 추가합니다. AWS 클라우드 새 `PublishResultsOnTopic` 구성 매개 변수를 사용하여 결과를 게시하려는 주제를 지정합니다. • 추론을 수행하려는 이미지의 사용자 지정 디렉터리를 지정할 수 있는 새 `ImageDirectory` 구성 매개 변수를 추가합니다. 버그 수정 및 개선 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과를 기록합니다. • AWS IoT GreengrassCore 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 기록합니다. • AWS IoT Device SDK를 사용하여 구성 요소 구성을 읽고 구성 변경 내용을 적용합니다.
2.0.4	초기 버전

DLR 객체 감지

DLR 객체 감지 구성 요소 (`aws.greengrass.DLRObjectDetection`) 에는 Deep Learning Runtime을 사용하여 객체 감지 추론을 수행하는 샘플 추론 코드와 사전 학습된 모델 샘플이 포함되어 있습니다. 이 구성 요소는 DLR 객체 감지 모델 스토어 변형과 구성 DLR 런타임 요소를 종속 항목으로 사용하여 DLR 및 샘플 모델을 다운로드합니다.

이 추론 구성 요소를 사용자 지정 학습된 DLR 모델과 함께 사용하려면 종속 모델 저장소 구성 요소의 사용자 지정 버전을 만드십시오. 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 추론 구성 요소를 만들 수 있습니다.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.

- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.13

다음 표에는 이 구성 요소의 버전 2.1.13에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.4 - 2.1.5

다음 표에는 이 구성 요소의 버전 2.1.4 ~ 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트
DLR 객체 감지 모델 스토어	~2.1.0	하드
DLR	~1.6.0	하드

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	~2.0.0	소프트
DLR 객체 감지 모델 스토어	~2.0.0	하드
DLR	~1.3.0	소프트

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

2.1.x

`accessControl`

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 하는 권한 부여 정책이 포함된 객체입니다.

기본값:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(선택 사항) 추론 결과를 게시하려는 주제입니다. 이 값을 수정하는 경우 `accessControl` 매개 변수의 값도 사용자 지정 주제 이름과 일치하도록 수정해야 합니다. `resources`

기본값: `ml/dlr/object-detection`

`Accelerator`

사용하려는 액셀러레이터. 지원되는 값은 `cpu` 및 `gpu`입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 가속만 지원합니다. 다른 사용자 지정 모델에서 GPU 가속을 사용하려면 사용자 지정 모델 구성 요소를 만들어 공개 모델 구성 요소를 재정의하십시오.

기본값: `cpu`

`ImageDirectory`

(선택 사항) 추론 구성 요소가 이미지를 읽는 기기의 폴더 경로입니다. 읽기/쓰기 권한이 있는 장치의 모든 위치로 이 값을 수정할 수 있습니다.

기본값: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/object_detection/sample_images/`

> **Note**
>
> 값을 `UseCamera` ~로 `true` 설정하면 이 구성 매개변수는 무시됩니다.

`ImageName`

(선택 사항) 추론 구성 요소가 메이크 예측의 입력으로 사용하는 이미지의 이름입니다. 구성요소는 에 지정된 폴더에서 이미지를 찾습니다. `ImageDirectory` 기본적으로 구성 요소는 기본 이미지 디렉터리의 샘플 이미지를 사용합니다. AWS IoT Greengrass`jpeg`, `jpg`png, 및 같은 이미지 형식을 지원합니다npy.

기본값: `objects.jpg`

> **Note**
>
> 값을 `UseCamera` 로 설정하는 `true` 경우 이 구성 매개변수는 무시됩니다.

`InferenceInterval`

(선택 사항) 추론 코드로 예측한 각 예측 사이의 시간 (초). 샘플 추론 코드는 무한정 실행되며 지정된 시간 간격으로 예측을 반복합니다. 예를 들어 카메라로 촬영한 이미지를 실시간 예측에 사용하려는 경우 이 간격을 더 짧게 변경할 수 있습니다.

기본값: `3600`

`ModelResourceKey`

(선택 사항) 종속 공개 모델 구성 요소에 사용되는 모델. 공개 모델 구성 요소를 사용자 지정 구성 요소로 재정의하는 경우에만 이 매개 변수를 수정하십시오.

기본값:

```
{
    "armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",
    "aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",
    "x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",
    "windows": "DLR-resnet50-win-cpu-ObjectDetection"
}
```

`UseCamera`

(선택 사항) Greengrass 코어 장치에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

이 값을 로 `true` 설정하면 샘플 추론 코드가 기기의 카메라에 액세스하여 캡처한 이미지에 대해 로컬에서 추론을 실행합니다. `ImageName`및 `ImageDirectory` 매개변수의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 캡처한 이미지를 저장하는 위치에 대한 읽기/쓰기 권한이 있는지 확인하십시오.

기본값: `false`

> **Note**
>
> 이 구성 요소의 레시피를 보면 구성 매개변수가 기본 `UseCamera` 구성에 표시되지 않습니다. 하지만 구성 요소를 배포할 때 구성 병합 업데이트에서 이 매개 변수의 값을 수정할 수 있습니다.
> `UseCamera`로 `true` 설정하면 런타임 구성 요소가 만든 가상 환경에서 추론 구성 요소가 카메라에 액세스할 수 있도록 심볼릭 링크도 만들어야 합니다. 샘플 추론 컴포넌트와

함께 카메라를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 구성 요소 구성 업데이트

2.0.x

`MLRootPath`

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값을 이 구성 요소를 실행하는 사용자에게 읽기/쓰기 액세스 권한이 있는 장치의 모든 위치로 수정할 수 있습니다.

기본값: */greengrass/v2*/work/variant.DLR/greengrass_ml

기본값: */greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

`Accelerator`

수정하지 마세요. 종속 모델 구성 요소의 모델은 CPU 가속기용으로만 컴파일되므로 현재 가속기에 지원되는 값은 뿐입니다. `cpu`

`ImageName`

(선택 사항) 추론 구성 요소가 메이크 예측의 입력으로 사용하는 이미지의 이름입니다. 구성요소는 에 지정된 폴더에서 이미지를 찾습니다. `ImageDirectory` 기본 위치는 입니다*MLRootPath*/images. AWS IoT Greengrassjpeg, jpgpng, 및 같은 이미지 형식을 지원합니다npy.

기본값: `objects.jpg`

`InferenceInterval`

(선택 사항) 추론 코드로 예측한 각 예측 사이의 시간 (초). 샘플 추론 코드는 무한정 실행되며 지정된 시간 간격으로 예측을 반복합니다. 예를 들어 카메라로 촬영한 이미지를 실시간 예측에 사용하려는 경우 이 간격을 더 짧게 변경할 수 있습니다.

기본값: `3600`

`ModelResourceKey`

(선택 사항) 종속 공개 모델 구성 요소에 사용되는 모델. 공개 모델 구성 요소를 사용자 지정 구성 요소로 재정의하는 경우에만 이 매개 변수를 수정하십시오.

기본값:

```
{
    armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
    x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10
   -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.13	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.12	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.11	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.8	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	구성 요소가 모두 출시되었습니다. AWS 리전
2.1.4	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다. 이 버전은 유럽 (런던) 에서 사용할 수 없습니다 (). `eu-west-2`
2.1.3	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	버그 수정 및 개선 • 샘플 DLR 개체 감지 추론 결과의 경계 상자가 부정확했던 이미지 스케일링 문제를 수정합니다.
2.1.1	새로운 기능 • 딥러닝 런타임 v1.6.0을 사용하십시오. • Armv8 (AArch64) 플랫폼에서 샘플 객체 감지에 대한 지원을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다.

버전	변경
	• 샘플 추론을 위한 카메라 통합을 활성화하십시오. 새 `UseCamera` 구성 매개변수를 사용하여 샘플 추론 코드를 활성화하여 Greengrass 코어 장치의 카메라에 액세스하고 캡처한 이미지에서 로컬로 추론을 실행할 수 있습니다. • 추론 결과를 게시하기 위한 지원을 에 추가합니다. AWS 클라우드 새 `PublishResultsOnTopic` 구성 매개 변수를 사용하여 결과를 게시하려는 주제를 지정합니다. • 추론을 수행하려는 이미지의 사용자 지정 디렉터리를 지정할 수 있는 새 `ImageDirectory` 구성 매개 변수를 추가합니다. 버그 수정 및 개선 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과를 기록합니다. • AWS IoT GreengrassCore 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 기록합니다. • AWS IoT Device SDK를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용할 수 있습니다.
2.0.4	초기 버전

DLR 이미지 분류 모델 스토어

DLR 이미지 분류 모델 저장소는 사전 학습된 ResNet -50개 모델을 Greengrass 아티팩트로 포함하는 기계 학습 모델 구성 요소입니다. 이 컴포넌트에 사용되는 사전 학습된 모델은 GluonCV Model Zoo에서 가져온 것으로, Neo 딥 러닝 런타임을 사용하여 컴파일됩니다. SageMaker

DLR 이미지 분류 추론 구성 요소는 이 구성 요소를 모델 소스의 종속 항목으로 사용합니다. 사용자 지정 학습된 DLR 모델을 사용하려면 이 모델 구성 요소의 사용자 지정 버전을 만들고 사용자 지정 모델을 구성 요소 아티팩트로 포함해야 합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 만들 수 있습니다.

> **Note**
>
> DLR 이미지 분류 모델 스토어 구성 요소의 이름은 버전에 따라 다릅니다. 버전 2.1.x 이상 버전의 구성 요소 이름은 입니다. `variant.DLR.ImageClassification.ModelStore` 버전 2.0.x의 구성 요소 이름은 입니다. `variant.ImageClassification.ModelStore`

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x () `variant.DLR.ImageClassification.ModelStore`
- 2.0.x () `variant.ImageClassification.ModelStore`

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.

- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	~2.0.0	소프트

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.12	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.11	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.8	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	새로운 기능 • Windows 코어 디바이스의 샘플 이미지 분류 모델을 추가합니다. • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	새로운 기능 • Armv8 (AArch64) 플랫폼을 위한 샘플 ResNet -50 이미지 분류 모델을 추가하세요. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다.
2.0.4	초기 버전

DLR 객체 감지 모델 스토어

DLR 객체 감지 모델 저장소는 사전 학습된 YoloV3 모델을 Greengrass 아티팩트로 포함하는 기계 학습 모델 구성 요소입니다. 이 컴포넌트에 사용된 샘플 모델은 GluonCV Model Zoo에서 가져온 후 네오 딥러닝 런타임을 사용하여 컴파일됩니다. SageMaker

DLR 개체 탐지 추론 구성 요소는 이 구성 요소를 모델 소스의 종속 항목으로 사용합니다. 사용자 지정 학습된 DLR 모델을 사용하려면 이 모델 구성 요소의 사용자 지정 버전을 만들고 사용자 지정 모델을 구성 요소 아티팩트로 포함해야 합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 만들 수 있습니다.

> **Note**
>
> DLR 개체 감지 모델 스토어 구성 요소의 이름은 버전에 따라 다릅니다. 버전 2.1.x 이상 버전의 구성 요소 이름은 입니다. `variant.DLR.ObjectDetection.ModelStore` 버전 2.0.x의 구성 요소 이름은 입니다. `variant.ObjectDetection.ModelStore`

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS 불스아이를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

레거시 카메라 스택을 활성화하려면

1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

```
sudo raspi-config
```

2. 인터페이스 옵션을 선택합니다.
3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.13

다음 표에는 이 구성 요소의 버전 2.1.13에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.1.5 and 2.1.6

다음 표에는 이 구성 요소의 버전 2.1.5 및 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	~2.0.0	소프트

구성

이 컴포넌트에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.13	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.12	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.11	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.8	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Armv8 (AArch64) 장치의 문제를 해결하기 위한 CPU 모델을 추가합니다.
2.1.5	새로운 기능 • Windows 코어 디바이스용 샘플 객체 감지 모델을 추가합니다. 버그 수정 및 개선 • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	새로운 기능 • Armv8 (AArch64) 플랫폼용 샘플 YoloV3 객체 감지 모델을 추가하세요. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다.
2.0.4	초기 버전

DLR 런타임

DLR 런타임 구성 요소 (`variant.DLR`) 에는 디바이스의 가상 환경에 DLR (딥 러닝 런타임) 및 해당 종속성을 설치하는 스크립트가 포함되어 있습니다. DLR 이미지 분류 및 DLR 객체 감지 구성 요소는 이 구성 요소를 DLR 설치를 위한 종속 항목으로 사용합니다. 구성 요소 버전 1.6.x는 DLR v1.6.0을 설치하고 구성 요소 버전 1.3.x는 DLR v1.3.0을 설치합니다.

다른 런타임을 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 기계 학습 구성 요소를 만들 수 있습니다.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.6.x
- 1.3.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.

- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

엔드포인트 및 포트

기본적으로 이 구성 요소는 코어 기기에서 사용하는 플랫폼에 따라 설치 프로그램 스크립트를 사용하여 `apt` `yumbrew`,, 및 `pip` 명령을 사용하여 패키지를 설치합니다. 이 구성 요소는 설치 스크립트를 실행하기 위해 다양한 패키지 인덱스 및 리포지토리에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 프록시 또는 방화벽을 통해 이 구성 요소의 아웃바운드 트래픽을 허용하려면 코어 장치가 설치에 연결되는 패키지 색인 및 리포지토리의 엔드포인트를 식별해야 합니다.

이 구성 요소의 설치 스크립트에 필요한 엔드포인트를 식별할 때는 다음 사항을 고려하십시오.

- 엔드포인트는 코어 디바이스의 플랫폼에 따라 다릅니다. 예를 들어, Ubuntu를 실행하는 코어 기기는 또는 `apt` 대신 사용합니다. `yum` `brew` 또한 동일한 패키지 색인을 사용하는 기기는 소스 목록이 다를 수 있으므로 서로 다른 저장소에서 패키지를 가져올 수 있습니다.
- 각 기기에는 패키지를 검색할 위치를 정의하는 자체 소스 목록이 있기 때문에 동일한 패키지 색인을 사용하는 여러 장치 간에 엔드포인트가 다를 수 있습니다.
- 엔드포인트는 시간이 지남에 따라 변경될 수 있습니다. 각 패키지 색인은 패키지를 다운로드하는 저장소의 URL을 제공하며, 패키지 소유자는 패키지 색인이 제공하는 URL을 변경할 수 있습니다.

이 구성 요소가 설치하는 종속성 및 설치 프로그램 스크립트를 사용하지 않도록 설정하는 방법에 대한 자세한 내용은 구성 매개 변수를 참조하십시오. UseInstaller

기본 작동에 필요한 엔드포인트 및 포트에 대한 자세한 내용은 을 참조하십시오. 프록시 또는 방화벽을 통한 장치 트래픽 허용

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

1.6.11 and 1.6.12

다음 표에는 이 구성 요소의 버전 1.6.11 및 1.6.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <3.0.0	소프트

1.6.10

다음 표에는 이 구성 요소의 버전 1.6.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

1.6.9

다음 표에는 이 구성 요소의 버전 1.6.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

1.6.8

다음 표에는 이 구성 요소의 버전 1.6.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

1.6.6 and 1.6.7

다음 표에는 이 구성 요소의 버전 1.6.6 및 1.6.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

1.6.4 and 1.6.5

다음 표에는 이 구성 요소의 버전 1.6.4 및 1.6.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

1.6.3

다음 표에는 이 구성 요소의 버전 1.6.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

1.6.2

다음 표에는 이 구성 요소의 버전 1.6.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

1.6.1

다음 표에는 이 구성 요소의 버전 1.6.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

1.3.x

다음 표에는 이 구성 요소의 버전 1.3.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	~2.0.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`MLRootPath`

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값을 이 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 장치의 모든 위치로 수정할 수 있습니다.

기본값: */greengrass/v2*/work/variant.DLR/greengrass_ml

`WindowsMLRootPath`

이 기능은 이 구성 요소의 v1.6.6 이상에서 사용할 수 있습니다.

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Windows 코어 장치의 폴더 경로입니다. 이 값을 이 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 장치의 모든 위치로 수정할 수 있습니다.

기본값: *C:\greengrass\v2*\\work\\variant.DLR\\greengrass_ml

`UseInstaller`

(선택 사항) 이 구성 요소의 설치 스크립트를 사용하여 DLR 및 해당 종속성을 설치할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

DLR 설치에 `false` 사용자 지정 스크립트를 사용하거나 미리 빌드된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 로 설정합니다. 이 구성 요소를 AWS -제공된 DLR 추론 구성 요소와 함께 사용하려면 종속성을 비롯한 다음 라이브러리를 설치하고 ML 구성 요소를 실행하는 시스템 사용자 등 `ggc_user` 시스템 사용자가 사용할 수 있도록 해야 합니다.

- Python 3.7 이상 (사용 중인 Python `pip` 버전용 포함).
- 딥러닝 런타임 v1.6.0
- NumPy.
- 오픈CV-파이썬.
- AWS IoT Device SDK파이썬을 위한 v2.
- AWS커먼 런타임 (CRT) 파이썬.
- 피카메라 (라즈베리 파이 기기에만 해당).
- `awscam`모듈 (AWS DeepLens기기용).
- libGL (리눅스 디바이스용)

기본값: `true`

사용량

구성 매개 변수를 로 설정한 상태로 이 `UseInstaller` 구성 요소를 사용하면 `true` 디바이스에 DLR 및 해당 종속 항목을 설치할 수 있습니다. 구성 요소는 DLR에 필요한 OpenCV NumPy 및 라이브러리가 포함된 가상 환경을 디바이스에 설정합니다.

> **Note**
>
> 또한 이 구성 요소의 설치 스크립트는 장치에 가상 환경을 구성하고 설치된 기계 학습 프레임워크를 사용하는 데 필요한 추가 시스템 라이브러리의 최신 버전을 설치합니다. 이렇게 하면 기기의 기존 시스템 라이브러리가 업그레이드될 수 있습니다. 이 구성 요소가 지원되는 각 운영 체제에 대해 설치하는 라이브러리 목록은 다음 표를 참조하십시오. 이 설치 프로세스를 사용자 지정하려면 `UseInstaller` 구성 매개 변수를 로 `false` 설정하고 자체 설치 스크립트를 개발하십시오.

플랫폼	장치 시스템에 설치된 라이브러리	가상 환경에 설치된 라이브러리
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	None
Ubuntu	`wget`	None

추론 구성 요소를 배포할 때 이 런타임 구성 요소는 먼저 장치에 DLR 및 해당 종속성이 이미 설치되어 있는지 확인하고, 설치되어 있지 않은 경우 자동으로 설치합니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.DLR.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.6.12	버그 수정 및 개선 • Windows OS 사용자를 위한 설치 스크립트를 수정합니다.
1.6.11	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.10	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.9	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.8	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.7	버그 수정 및 개선 • 특정 Linux 플랫폼에서 기본적으로 사용할 수 없는 LibGL을 설치하도록 `UseInstaller` 설치 스크립트를 업데이트합니다.

버전	변경
	• 이 컴포넌트의 가상 환경에서 항상 Python 3.9를 사용하도록 `UseInstaller` 설치 스크립트를 업데이트합니다. 이 변경은 다른 라이브러리와의 호환성을 보장하는 데 도움이 됩니다.
1.6.6	새로운 기능 • Windows를 실행하는 코어 디바이스에 대한 지원을 추가합니다. • Windows 핵심 장치에서 추론 결과 폴더를 구성하는 데 사용할 수 있는 새 `WindowsMLRootPath` 구성 매개 변수를 추가합니다.
1.6.5	새로운 기능 • 이 `UseInstaller` 구성 요소에서 설치 스크립트를 사용하지 않도록 설정하는 데 사용할 수 있는 새 구성 매개 변수를 추가합니다.
1.6.4	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.3	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.2	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
1.6.1	새로운 기능 • 딥러닝 런타임 v1.6.0 및 종속 항목을 설치합니다. • Armv8 (AArch64) 플랫폼에 DLR을 설치하기 위한 지원을 추가합니다. 이는 젯슨 나노와 같은 NVIDIA Jetson을 실행하는 Greengrass 코어 디바이스에 대한 머신 러닝 지원을 확대합니다. 버그 수정 및 개선 • 가상 환경에 를 AWS IoT Device SDK 설치하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다. • 추가 사소한 버그 수정 및 개선
1.3.2	초기 버전 DLR v1.3.0을 설치합니다.

TensorFlow 라이트 이미지 분류

TensorFlow Lite 이미지 분류 구성 요소 (`aws.greengrass.TensorFlowLiteImageClassification`) 에는 TensorFlow Lite 런타임을

사용하여 이미지 분류 추론을 수행하는 샘플 추론 코드와 사전 학습된 MobileNet 1.0 양자화 모델 샘플이 포함되어 있습니다. 이 구성 요소는 TensorFlow 라이트 이미지 분류 모델 스토어 변형과 구성 TensorFlow 라이트 런타임 요소를 종속성으로 사용하여 TensorFlow Lite 런타임과 샘플 모델을 다운로드합니다.

이 추론 구성 요소를 사용자 지정 학습된 TensorFlow Lite 모델과 함께 사용하려면 종속 모델 저장소 구성 요소의 사용자 지정 버전을 만드십시오. 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 추론 구성 요소를 만들 수 있습니다.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux

- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹

션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

종속성	호환되는 버전	종속성 유형
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`accessControl`

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 하는 권한 부여 정책이 포함된 객체입니다.

기본값:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(선택 사항) 추론 결과를 게시하려는 주제입니다. 이 값을 수정하는 경우 `accessControl` 매개 변수의 값도 사용자 지정 주제 이름과 일치하도록 수정해야 합니다. `resources`

기본값: `ml/tflite/image-classification`

`Accelerator`

사용하려는 액셀러레이터. 지원되는 값은 `cpu` 및 `gpu`입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 가속만 지원합니다. 다른 사용자 지정 모델에서 GPU 가속을 사용하려면 사용자 지정 모델 구성 요소를 만들어 공개 모델 구성 요소를 재정의하십시오.

기본값: `cpu`

`ImageDirectory`

(선택 사항) 추론 구성 요소가 이미지를 읽는 기기의 폴더 경로입니다. 읽기/쓰기 권한이 있는 장치의 모든 위치로 이 값을 수정할 수 있습니다.

기본값: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/image_classification/sample_images/`

> **Note**
>
> 값을 `UseCamera` ~로 `true` 설정하면 이 구성 매개변수는 무시됩니다.

`ImageName`

(선택 사항) 추론 구성 요소가 메이크 예측의 입력으로 사용하는 이미지의 이름. 구성요소는 에 지정된 폴더에서 이미지를 찾습니다. `ImageDirectory` 기본적으로 구성 요소는 기본 이미지 디렉터리의 샘플 이미지를 사용합니다. AWS IoT Greengrass`jpeg`, `jpg`png, 및 같은 이미지 형식을 지원합니다npy.

기본값: `cat.jpeg`

> **Note**
>
> 값을 `UseCamera` 로 설정하는 `true` 경우 이 구성 매개변수는 무시됩니다.

`InferenceInterval`

(선택 사항) 추론 코드로 예측한 각 예측 사이의 시간 (초). 샘플 추론 코드는 무한정 실행되며 지정된 시간 간격으로 예측을 반복합니다. 예를 들어 카메라로 촬영한 이미지를 실시간 예측에 사용하려는 경우 이 간격을 더 짧게 변경할 수 있습니다.

기본값: `3600`

`ModelResourceKey`

(선택 사항) 종속 공개 모델 구성 요소에 사용되는 모델. 공개 모델 구성 요소를 사용자 지정 구성 요소로 재정의하는 경우에만 이 매개 변수를 수정하십시오.

기본값:

```
{
    "model": "TensorFlowLite-Mobilenet"
}
```

`UseCamera`

(선택 사항) Greengrass 코어 장치에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

이 값을 로 `true` 설정하면 샘플 추론 코드가 기기의 카메라에 액세스하여 캡처한 이미지에 대해 로컬에서 추론을 실행합니다. `ImageName`및 `ImageDirectory` 매개변수의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 캡처한 이미지를 저장하는 위치에 대한 읽기/쓰기 권한이 있는지 확인하십시오.

기본값: `false`

> **Note**
>
> 이 구성 요소의 레시피를 보면 구성 매개변수가 기본 `UseCamera` 구성에 표시되지 않습니다. 하지만 구성 요소를 배포할 때 구성 병합 업데이트에서 이 매개 변수의 값을 수정할 수 있습니다.
> `UseCamera`로 `true` 설정하면 런타임 구성 요소가 만든 가상 환경에서 추론 구성 요소가 카메라에 액세스할 수 있도록 심볼릭 링크도 만들어야 합니다. 샘플 추론 컴포넌트와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 구성 요소 구성 업데이트

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	초기 버전

TensorFlow 라이트 오브젝트 감지

TensorFlow Lite 객체 감지 구성 요소 (`aws.greengrass.TensorFlowLiteObjectDetection`) 에는 TensorFlow Lite를 사용하여 객체 감지 추론을 수행하는 샘플 추론 코드와 사전 학습된 단일 샷 탐지 (SSD) MobileNet 1.0 모델 샘플이 포함되어 있습니다. 이 구성 요소는 TensorFlow Lite 객체 감지 모델 스토어 변형과 구성 TensorFlow 라이트 런타임 요소를 종속 항목으로 사용하여 TensorFlow Lite 및 샘플 모델을 다운로드합니다.

사용자 지정 학습된 TensorFlow Lite 모델과 함께 이 추론 구성 요소를 사용하려면 종속 모델 저장소 구성 요소의 사용자 지정 버전을 만들면 됩니다. 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 추론 구성 요소를 만드십시오.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성

- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.

- NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

  ```
  pip3 install --upgrade numpy
  ```

- 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

  레거시 카메라 스택을 활성화하려면

  1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

     ```
     sudo raspi-config
     ```

  2. 인터페이스 옵션을 선택합니다.
  3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
  4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드

종속성	호환되는 버전	종속성 유형
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

종속성	호환되는 버전	종속성 유형
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트
TensorFlow 라이트 이미지 분류 모델 스토어	>=2.1.0 <2.2.0	하드
TensorFlow 라이트	>=2.5.0 <2.6.0	하드

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`accessControl`

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 하는 권한 부여 정책이 포함된 객체입니다.

기본값:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(선택 사항) 추론 결과를 게시하려는 주제입니다. 이 값을 수정하는 경우 `accessControl` 매개 변수의 값도 사용자 지정 주제 이름과 일치하도록 수정해야 합니다. `resources`

기본값: `ml/tflite/object-detection`

`Accelerator`

사용하려는 액셀러레이터. 지원되는 값은 `cpu` 및 `gpu`입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 가속만 지원합니다. 다른 사용자 지정 모델에서 GPU 가속을 사용하려면 사용자 지정 모델 구성 요소를 만들어 공개 모델 구성 요소를 재정의하십시오.

기본값: `cpu`

`ImageDirectory`

(선택 사항) 추론 구성 요소가 이미지를 읽는 기기의 폴더 경로입니다. 읽기/쓰기 권한이 있는 장치의 모든 위치로 이 값을 수정할 수 있습니다.

기본값: */greengrass/v2*/packages/artifacts-unarchived/*component-name*/object_detection/sample_images/

> **Note**
>
> 값을 `UseCamera` ~로 `true` 설정하면 이 구성 매개변수는 무시됩니다.

`ImageName`

(선택 사항) 추론 구성 요소가 메이크 예측의 입력으로 사용하는 이미지의 이름입니다. 구성요소는 에 지정된 폴더에서 이미지를 찾습니다. `ImageDirectory` 기본적으로 구성 요소는 기본 이미지 디렉터리의 샘플 이미지를 사용합니다. AWS IoT Greengrass`jpeg`, `jpg`png, 및 같은 이미지 형식을 지원합니다npy.

기본값: `objects.jpg`

> **Note**
>
> 값을 `UseCamera` 로 설정하는 `true` 경우 이 구성 매개변수는 무시됩니다.

`InferenceInterval`

(선택 사항) 추론 코드로 예측한 각 예측 사이의 시간 (초). 샘플 추론 코드는 무한정 실행되며 지정된 시간 간격으로 예측을 반복합니다. 예를 들어 카메라로 촬영한 이미지를 실시간 예측에 사용하려는 경우 이 간격을 더 짧게 변경할 수 있습니다.

기본값: `3600`

`ModelResourceKey`

(선택 사항) 종속 공개 모델 구성 요소에 사용되는 모델. 공개 모델 구성 요소를 사용자 지정 구성 요소로 재정의하는 경우에만 이 매개 변수를 수정하십시오.

기본값:

```
{
    "model": "TensorFlowLite-SSD"
}
```

`UseCamera`

(선택 사항) Greengrass 코어 장치에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

이 값을 로 `true` 설정하면 샘플 추론 코드가 기기의 카메라에 액세스하여 캡처한 이미지에 대해 로컬에서 추론을 실행합니다. `ImageName`및 `ImageDirectory` 매개변수의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 캡처한 이미지를 저장하는 위치에 대한 읽기/쓰기 권한이 있는지 확인하십시오.

기본값: `false`

> **Note**
>
> 이 구성 요소의 레시피를 보면 구성 매개변수가 기본 `UseCamera` 구성에 표시되지 않습니다. 하지만 구성 요소를 배포할 때 구성 병합 업데이트에서 이 매개 변수의 값을 수정할 수 있습니다.
> `UseCamera`로 `true` 설정하면 런타임 구성 요소가 만든 가상 환경에서 추론 구성 요소가 카메라에 액세스할 수 있도록 심볼릭 링크도 만들어야 합니다. 샘플 추론 컴포넌트와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 구성 요소 구성 업데이트

> **Note**
>
> 이 구성요소의 레시피를 보면 `UseCamera` 구성 매개변수가 기본 구성에 표시되지 않습니다. 하지만 구성 요소를 배포할 때 구성 병합 업데이트에서 이 매개 변수의 값을 수정할 수 있습니다.
> `UseCamera`로 `true` 설정하면 런타임 구성 요소가 만든 가상 환경에서 추론 구성 요소가 카메라에 액세스할 수 있도록 심볼릭 링크도 만들어야 합니다. 샘플 추론 컴포넌트와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 구성 요소 구성 업데이트

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	버그 수정 및 개선 • 샘플 TensorFlow Lite 객체 감지 추론 결과의 경계 상자가 부정확하게 표시되는 이미지 스케일링 문제를 수정합니다.
2.1.0	초기 버전

TensorFlow 라이트 이미지 분류 모델 스토어

TensorFlow Lite 이미지 분류 모델 저장소 (`variant.TensorFlowLite.ImageClassification.ModelStore`) 는 사전 학습된 MobileNet v1 모델을 Greengrass 아티팩트로 포함하는 기계 학습 모델 구성 요소입니다. 이 구성 요소에 사용된 샘플 모델은 Hub에서 가져와서 Lite를 사용하여 구현됩니다. TensorFlow TensorFlow

TensorFlow 라이트 이미지 분류추론 구성 요소는 이 구성 요소를 모델 소스의 종속 항목으로 사용합니다. 사용자 지정 학습된 TensorFlow Lite 모델을 사용하려면 이 모델 구성 요소의 사용자 지정 버전을 만들고 사용자 지정 모델을 구성 요소 아티팩트로 포함해야 합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 만들 수 있습니다.

주제

- 버전
- 유형
- 운영 체제

- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 컴포넌트의 버전은 다음과 같습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	초기 버전

TensorFlow Lite 객체 감지 모델 스토어

TensorFlow Lite 객체 감지 모델 저장소 (`variant.TensorFlowLite.ObjectDetection.ModelStore`) 는 사전 학습된 싱글 샷 감지 (SSD) 모델을 Greengrass 아티팩트로 포함하는 기계 학습 MobileNet 모델 구성 요소입니다. 이 구성 요소에 사용된 샘플 모델은 TensorFlow 허브에서 가져와서 Lite를 사용하여 구현됩니다. TensorFlow

TensorFlow Lite 개체 감지 추론 구성 요소는 이 구성 요소를 모델 소스의 종속 항목으로 사용합니다. 사용자 지정 학습된 TensorFlow Lite 모델을 사용하려면 이 모델 구성 요소의 사용자 지정 버전을 만들고 사용자 지정 모델을 구성 요소 아티팩트로 포함해야 합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 만들 수 있습니다.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성

- 구성
- 로컬 로그 파일
- Changelog

버전

이 컴포넌트의 버전은 다음과 같습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.

- NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

  ```
  pip3 install --upgrade numpy
  ```

- 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

  레거시 카메라 스택을 활성화하려면

  1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

     ```
     sudo raspi-config
     ```

  2. 인터페이스 옵션을 선택합니다.
  3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
  4. Raspberry Pi를 재부팅합니다.

의존성

구성 요소를 배포하면 해당 종속 항목의 호환 AWS IoT Greengrass 버전도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.9	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.8	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	초기 버전

TensorFlow 라이트 런타임

TensorFlow Lite 런타임 구성 요소 (`variant.TensorFlowLite`) 에는 장치의 가상 환경에 TensorFlow Lite 버전 2.5.0과 해당 종속성을 설치하는 스크립트가 포함되어 있습니다. TensorFlow Lite 이미지 분류 및 TensorFlow Lite 개체 감지 구성 요소는 이 런타임 구성 요소를 Lite 설치를 위한 종속 항목으로 사용합니다. TensorFlow

> **Note**
>
> TensorFlow Lite 런타임 구성 요소 v2.5.6 이상은 Lite 런타임의 기존 설치와 해당 종속성을 다시 설치합니다. TensorFlow 이렇게 다시 설치하면 코어 기기가 호환되는 버전의 Lite 및 해당 종속 항목을 실행하도록 하는 데 도움이 됩니다. TensorFlow

다른 런타임을 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 기계 학습 구성 요소를 만들 수 있습니다.

주제

- 버전

- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 사용량
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.5.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```
  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```
    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

엔드포인트 및 포트

기본적으로 이 구성 요소는 코어 기기에서 사용하는 플랫폼에 따라 설치 프로그램 스크립트를 사용하여 `apt` `yum``brew`,, 및 `pip` 명령을 사용하여 패키지를 설치합니다. 이 구성 요소는 설치 스크립트를 실행하기 위해 다양한 패키지 인덱스 및 리포지토리에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 프록시 또는 방화벽을 통해 이 구성 요소의 아웃바운드 트래픽을 허용하려면 코어 장치가 설치에 연결되는 패키지 색인 및 리포지토리의 엔드포인트를 식별해야 합니다.

이 구성 요소의 설치 스크립트에 필요한 엔드포인트를 식별할 때는 다음 사항을 고려하십시오.

- 엔드포인트는 코어 디바이스의 플랫폼에 따라 다릅니다. 예를 들어, Ubuntu를 실행하는 코어 기기는 또는 `apt` 대신 사용합니다. `yum` `brew` 또한 동일한 패키지 색인을 사용하는 기기는 소스 목록이 다를 수 있으므로 서로 다른 저장소에서 패키지를 가져올 수 있습니다.

- 각 기기에는 패키지를 검색할 위치를 정의하는 자체 소스 목록이 있기 때문에 동일한 패키지 색인을 사용하는 여러 장치 간에 엔드포인트가 다를 수 있습니다.
- 엔드포인트는 시간이 지남에 따라 변경될 수 있습니다. 각 패키지 색인은 패키지를 다운로드하는 저장소의 URL을 제공하며, 패키지 소유자는 패키지 색인이 제공하는 URL을 변경할 수 있습니다.

이 구성 요소가 설치하는 종속성 및 설치 프로그램 스크립트를 사용하지 않도록 설정하는 방법에 대한 자세한 내용은 구성 매개 변수를 참조하십시오. UseInstaller

기본 작동에 필요한 엔드포인트 및 포트에 대한 자세한 내용은 을 참조하십시오. 프록시 또는 방화벽을 통한 장치 트래픽 허용

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.5.14

다음 표에는 이 구성 요소의 버전 2.5.14에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

2.5.13

다음 표에는 이 구성 요소의 버전 2.5.13에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

2.5.12

다음 표에는 이 구성 요소의 버전 2.5.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

2.5.11

다음 표에는 이 구성 요소의 버전 2.5.11에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

2.5.10

다음 표에는 이 구성 요소의 버전 2.5.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

2.5.9

다음 표에는 이 구성 요소의 버전 2.5.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

2.5.8

다음 표에는 이 구성 요소의 버전 2.5.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

2.5.5 - 2.5.7

다음 표에는 이 구성 요소의 버전 2.5.5~2.5.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

2.5.3 and 2.5.4

다음 표에는 이 구성 요소의 버전 2.5.3 및 2.5.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.5.2

다음 표에는 이 구성 요소의 버전 2.5.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.5.1

다음 표에는 이 구성 요소의 버전 2.5.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.5.0

다음 표에는 이 구성 요소의 버전 2.5.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`MLRootPath`

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값을 이 구성 요소를 실행하는 사용자에게 읽기/쓰기 액세스 권한이 있는 장치의 모든 위치로 수정할 수 있습니다.

기본값: */greengrass/v2*`/work/variant.TensorFlowLite/greengrass_ml`

`WindowsMLRootPath`

이 기능은 이 구성 요소의 v1.6.6 이상에서 사용할 수 있습니다.

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Windows 코어 장치의 폴더 경로입니다. 이 값을 이 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 장치의 모든 위치로 수정할 수 있습니다.

기본값: *C:\greengrass\v2*`\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(선택 사항) 이 구성 요소의 설치 스크립트를 사용하여 TensorFlow Lite 및 해당 종속 항목을 설치할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

TensorFlow Lite 설치에 `false` 사용자 지정 스크립트를 사용하거나 미리 빌드된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 로 설정하십시오. 이 구성 요소를 AWS -제공된 TensorFlow Lite 추론 구성 요소와 함께 사용하려면 모든 종속성을 포함한 다음 라이브러리를 설치하고 ML 구성 요소를 실행하는 시스템 사용자 등 `ggc_user` 시스템 사용자가 사용할 수 있도록 해야 합니다.

- 파이썬 3.8 이상 (사용 중인 파이썬 `pip` 버전용 포함)

- TensorFlow 라이트 v2.5.0
- NumPy
- 오픈CV-파이썬
- AWS IoT Device SDK파이썬용 v2
- AWS커먼 런타임 (CRT) Python
- 피카메라 (라즈베리 파이 기기용)
- `awscam`모듈 (기기용AWS DeepLens)
- libGL (리눅스 디바이스용)

기본값: `true`

사용량

`UseInstaller`구성 매개 변수를 로 설정한 상태에서 이 컴포넌트를 사용하면 TensorFlow Lite 및 해당 종속 항목을 `true` 디바이스에 설치할 수 있습니다. 구성 요소는 Lite에 필요한 OpenCV NumPy 및 라이브러리가 포함된 가상 환경을 장치에 설정합니다. TensorFlow

> **Note**
>
> 또한 이 구성 요소의 설치 스크립트는 장치에 가상 환경을 구성하고 설치된 기계 학습 프레임워크를 사용하는 데 필요한 추가 시스템 라이브러리의 최신 버전을 설치합니다. 이렇게 하면 기기의 기존 시스템 라이브러리가 업그레이드될 수 있습니다. 이 구성 요소가 지원되는 각 운영 체제에 대해 설치하는 라이브러리 목록은 다음 표를 참조하십시오. 이 설치 프로세스를 사용자 지정하려면 `UseInstaller` 구성 매개 변수를 로 `false` 설정하고 자체 설치 스크립트를 개발하십시오.

플랫폼	장치 시스템에 설치된 라이브러리	가상 환경에 설치된 라이브러리
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	None
Ubuntu	`wget`	None

추론 구성 요소를 배포할 때 이 런타임 구성 요소는 먼저 기기에 TensorFlow Lite와 해당 종속 항목이 이미 설치되어 있는지 확인합니다. 그렇지 않은 경우 런타임 구성 요소가 자동으로 설치합니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.TensorFlowLite.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.5.14	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.5.13	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.12	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.11	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.10	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.9	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.8	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.7	버그 수정 및 개선 • 특정 Linux 플랫폼에서 기본적으로 사용할 수 없는 LibGL을 설치하도록 `UseInstaller` 설치 스크립트를 업데이트합니다. • 이 컴포넌트의 가상 환경에서 항상 Python 3.9를 사용하도록 `UseInstaller` 설치 스크립트를 업데이트합니다. 이 변경은 다른 라이브러리와의 호환성을 보장하는 데 도움이 됩니다.
2.5.6	버그 수정 및 개선 • 이 구성 요소를 업데이트하여 TensorFlow Lite 2.5.0 (`tflite-runtime-2.5.0.post1`) 의 최신 패치를 설치하므로 Python 3.9에서 이 구성 요소를 사용할 수 있습니다. 이 구성 요소가 해당 패치를 설치하지 못하면 대신 설치됩니다. `tflite-runtime-2.5.0` • 이 구성 요소를 업데이트하여 기존에 설치된 TensorFlow Lite 및 해당 종속 항목을 다시 설치합니다. 이번 변경은 코어 기기가 호환되는 버전의 TensorFlow Lite 및 해당 종속 항목을 실행하도록 하는 데 도움이 됩니다.
2.5.5	새로운 기능 • Windows를 실행하는 핵심 장치에 대한 지원을 추가합니다. • Windows 핵심 장치에서 추론 결과 폴더를 구성하는 데 사용할 수 있는 새 `WindowsMLRootPath` 구성 매개 변수를 추가합니다.

버전	변경
2.5.4	새로운 기능 • 이 `UseInstaller` 구성 요소에서 설치 스크립트를 비활성화할 수 있는 새 구성 매개 변수를 추가합니다.
2.5.3	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.2	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.1	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.5.0	초기 버전

모드버스-RTU 프로토콜 어댑터

Modbus-RTU 프로토콜 어댑터 컴포넌트 (`aws.greengrass.Modbus`) 는 로컬 Modbus RTU 장치의 정보를 폴링합니다.

이 컴포넌트를 사용하여 로컬 Modbus RTU 장치에 정보를 요청하려면 이 컴포넌트가 구독하는 주제에 메시지를 게시하십시오. 메시지에서 장치에 전송할 Modbus RTU 요청을 지정하십시오. 그런 다음 이 구성 요소는 Modbus RTU 요청 결과가 포함된 응답을 게시합니다.

> **Note**
>
> 이 구성 요소는 V1의 Modbus RTU 프로토콜 어댑터 커넥터와 유사한 기능을 제공합니다. AWS IoT Greengrass 자세한 내용은 V1 개발자 안내서의 Modbus RTU 프로토콜 어댑터 커넥터를 참조하십시오. AWS IoT Greengrass

주제

- 구성
- 입력 데이터
- 출력 데이터
- Modbus RTU 요청 및 응답
- 로컬 로그 파일
- 라이선스
- Changelog

버전

이 컴포넌트의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 Lambda 구성 요소 () 입니다. `aws.greengrass.lambda` Greengrass 핵은 Lambda 런처 구성 요소를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- AWS IoT Greengrass 코어 디바이스와 Modbus 디바이스 간의 물리적 연결. 코어 장치는 USB 포트와 같은 직렬 포트를 통해 Modbus RTU 네트워크에 물리적으로 연결되어야 합니다.

- 이 컴포넌트로부터 출력 데이터를 받으려면 이 컴포넌트를 배포할 때 레거시 서브스크립션 라우터 컴포넌트 (`aws.greengrass.LegacySubscriptionRouter`) 에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성은 이 구성 요소가 응답을 게시하는 주제를 지정합니다.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "component:aws.greengrass.Modbus",
        "subject": "modbus/adapter/response",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
  modbus:version",
        "subject": "modbus/adapter/response",
        "target": "cloud"
      }
    }
  }
  ```

  - *### AWS ## #### ####* 바꾸십시오.
  - *###* 이 구성 요소가 실행하는 Lambda 함수 버전으로 교체하십시오. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소 버전의 레시피를 확인해야 합니다. AWS IoT Greengrass 콘솔에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 쌍을 찾으십시오. 이 키-값 쌍에는 Lambda 함수의 이름과 버전이 들어 있습니다.

> **Important**
>
> 이 구성 요소를 배포할 때마다 레거시 구독 라우터에서 Lambda 함수 버전을 업데이트해야 합니다. 이렇게 하면 배포하는 구성 요소 버전에 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 설명은 배포 만들기 섹션을 참조하세요.

- Modbus-RTU 프로토콜 어댑터는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.4 and 2.1.5

다음 표에는 이 구성 요소의 버전 2.1.4 및 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.8 and 2.1.0

다음 표에는 이 구성 요소의 버전 2.0.8 및 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	하드
람다 런처	>=1.0.0	하드
Lambda 런타임	>=1.0.0	소프트
토큰 교환 서비스	>=1.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

> **Note**
>
> 이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함됩니다. 디바이스에서 이 구성 요소를 구성하려면 다음 파라미터만 편집하는 것이 좋습니다.

v2.1.x

`lambdaParams`

이 구성 요소의 Lambda 함수에 대한 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`EnvironmentVariables`

Lambda 함수의 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`ModbusLocalPort`

코어 디바이스의 물리적 Modbus 직렬 포트에 대한 절대 경로 (예:. `/dev/ttyS2`

컨테이너에서 이 컴포넌트를 실행하려면 이 경로를 컴포넌트가 액세스할 수 있는 시스템 장치 (incontainerParams.devices) 로 정의해야 합니다. 이 구성 요소는 기본적으로 컨테이너에서 실행됩니다.

> **Note**
>
> 이 구성 요소에는 기기에 대한 읽기/쓰기 권한이 있어야 합니다.

`ModbusBaudRate`

(선택 사항) 로컬 Modbus TCP 장치와의 직렬 통신을 위한 전송 속도를 지정하는 문자열 값입니다.

기본값: `9600`

`ModbusByteSize`

(선택 사항) 로컬 Modbus TCP 장치와의 직렬 통신에서 바이트 크기를 지정하는 문자열 값입니다. 5,6, 7 또는 비트를 선택합니다. 8

기본값: 8

`ModbusParity`

(선택 사항) 로컬 Modbus TCP 장치와의 직렬 통신에서 데이터 무결성을 확인하는 데 사용할 패리티 모드입니다.

- `E`— 균일한 패리티로 데이터 무결성을 확인합니다.
- `O`— 홀수 패리티로 데이터 무결성을 확인합니다.
- `N`— 데이터 무결성을 확인하지 마세요.

기본값: `N`

`ModbusStopBits`

(선택 사항) 로컬 Modbus TCP 장치와의 직렬 통신에서 바이트 끝을 나타내는 비트 수를 지정하는 문자열 값입니다.

기본값: `1`

`containerMode`

(선택 사항) 이 컴포넌트의 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- `GreengrassContainer`— 구성 요소는 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다. AWS IoT Greengrass

  이 옵션을 지정하는 경우 시스템 장치 (in`containerParams.devices`) 를 지정하여 컨테이너에 Modbus 장치에 대한 액세스 권한을 부여해야 합니다.
- `NoContainer`— 컴포넌트는 격리된 런타임 환경에서 실행되지 않습니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소의 컨테이너 매개 변수를 포함하는 개체입니다. 를 `GreengrassContainer` 지정하면 구성 요소에서 이러한 매개 변수를 사용합니다`containerMode`.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당할 메모리 양 (KB) 입니다.

기본값은 512MB (525,312KB) 입니다.

`devices`

(선택 사항) 구성 요소가 컨테이너에서 액세스할 수 있는 시스템 장치를 지정하는 개체입니다.

> **Important**
>
> 컨테이너에서 이 구성 요소를 실행하려면 `ModbusLocalPort` 환경 변수에 구성하는 시스템 장치를 지정해야 합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열로서의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

`path`

코어 디바이스에 있는 시스템 디바이스의 경로. 이 값은 구성한 값과 같아야 `ModbusLocalPort` 합니다.

`permission`

(선택 사항) 컨테이너에서 시스템 장치에 액세스할 수 있는 권한. 이 값은 구성 요소가 시스템 장치에 대한 읽기/쓰기 권한을 가짐을 지정하는 값이어야 `rw` 합니다.

기본값: `rw`

`addGroupOwner`

(선택 사항) 구성 요소를 실행하는 시스템 그룹을 시스템 장치의 소유자로 추가할지 여부.

기본값: `true`

`pubsubTopics`

(선택 사항) 구성 요소가 메시지 수신을 구독하는 주제를 포함하는 개체입니다. 각 주제를 지정하고 구성 요소가 MQTT 주제를 구독하는지 AWS IoT Core 아니면 로컬 게시/구독 주제의 MQTT 주제를 구독할지를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열 형식의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

`type`

(선택 사항) 이 구성 요소가 메시지를 구독하는 데 사용하는 게시/구독 메시지의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- `IOT_CORE`— AWS IoT Core MQTT 메시지를 구독하십시오. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 있습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본값: PUB_SUB

`topic`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제입니다. 를 지정하는 경우 이 `IotCore` `type` 항목에서 MQTT 와일드카드 (+및#) 를 사용할 수 있습니다.

Example 예: 구성 병합 업데이트 (컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (컨테이너 모드 없음)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

`lambdaParams`

이 구성 요소의 Lambda 함수에 대한 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`EnvironmentVariables`

Lambda 함수의 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`ModbusLocalPort`

코어 디바이스의 물리적 Modbus 직렬 포트에 대한 절대 경로 (예:. `/dev/ttyS2`

컨테이너에서 이 컴포넌트를 실행하려면 이 경로를 컴포넌트가 액세스할 수 있는 시스템 장치 (`incontainerParams.devices`) 로 정의해야 합니다. 이 구성 요소는 기본적으로 컨테이너에서 실행됩니다.

> **Note**
>
> 이 구성 요소에는 기기에 대한 읽기/쓰기 권한이 있어야 합니다.

`containerMode`

(선택 사항) 이 컴포넌트의 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- `GreengrassContainer`— 구성 요소는 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다. AWS IoT Greengrass

  이 옵션을 지정하는 경우 시스템 장치 (`incontainerParams.devices`) 를 지정하여 컨테이너에 Modbus 장치에 대한 액세스 권한을 부여해야 합니다.
- `NoContainer`— 컴포넌트는 격리된 런타임 환경에서 실행되지 않습니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소의 컨테이너 매개 변수를 포함하는 개체입니다. 를 `GreengrassContainer` 지정하면 구성 요소에서 이러한 매개 변수를 사용합니다`containerMode`.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당할 메모리 양 (KB) 입니다.

기본값은 512MB (525,312KB) 입니다.

`devices`

(선택 사항) 구성 요소가 컨테이너에서 액세스할 수 있는 시스템 장치를 지정하는 개체입니다.

> **Important**
>
> 컨테이너에서 이 구성 요소를 실행하려면 `ModbusLocalPort` 환경 변수에 구성하는 시스템 장치를 지정해야 합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열로서의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

`path`

코어 디바이스에 있는 시스템 디바이스의 경로. 이 값은 구성한 값과 같아야 `ModbusLocalPort` 합니다.

`permission`

(선택 사항) 컨테이너에서 시스템 장치에 액세스할 수 있는 권한. 이 값은 구성 요소가 시스템 장치에 대한 읽기/쓰기 권한을 가짐을 지정하는 값이어야 `rw` 합니다.

기본값: `rw`

`addGroupOwner`

(선택 사항) 구성 요소를 실행하는 시스템 그룹을 시스템 장치의 소유자로 추가할지 여부.

기본값: `true`

`pubsubTopics`

(선택 사항) 구성 요소가 메시지 수신을 구독하는 주제를 포함하는 개체입니다. 각 주제를 지정하고 구성 요소가 MQTT 주제를 구독하는지 AWS IoT Core 아니면 로컬 게시/구독 주제의 MQTT 주제를 구독할지를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0`— 문자열 형식의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

`type`

(선택 사항) 이 구성 요소가 메시지를 구독하는 데 사용하는 게시/구독 메시지의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- IOT_CORE— AWS IoT Core MQTT 메시지를 구독하십시오. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 있습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본값: PUB_SUB

`topic`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제입니다. 를 지정하는 경우 이 `IotCore type` 항목에서 MQTT 와일드카드 (+및#) 를 사용할 수 있습니다.

Example 예: 구성 병합 업데이트 (컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 예: 구성 병합 업데이트 (컨테이너 모드 없음)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
```

```
    }
  },
  "containerMode": "NoContainer"
}
```

입력 데이터

이 구성 요소는 다음 주제에 대한 Modbus RTU 요청 파라미터를 수락하고 Modbus RTU 요청을 장치에 보냅니다. 기본적으로 이 컴포넌트는 로컬 게시/구독 메시지를 구독합니다. 사용자 정의 구성 요소에서 이 구성 요소에 메시지를 게시하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독

기본 주제 (로컬 게시/구독): `modbus/adapter/request`

메시지는 다음 속성을 허용합니다. 입력 메시지는 JSON 형식이어야 합니다.

`request`

전송할 Modbus RTU 요청에 대한 파라미터입니다.

요청 메시지의 모양은 해당 메시지가 나타내는 Modbus RTU 요청 유형에 따라 달라집니다. 모든 요청에는 다음 속성이 필요합니다.

다음 정보가 `object` 포함된 유형:

`operation`

실행할 작업의 이름. 예를 들어, Modbus RTU 장치에서 코일을 `ReadCoilsRequest` 읽도록 지정합니다. 지원되는 작업에 대한 자세한 내용은 을 참조하십시오. Modbus RTU 요청 및 응답

유형: `string`

`device`

요청의 대상 디바이스입니다.

이 값은 `0` ~와 `247` 사이의 정수여야 합니다.

유형: `integer`

요청에 포함할 다른 파라미터는 작업에 따라 다릅니다. 이 구성 요소는 순환 중복 검사 (CRC) 를 처리하여 데이터 요청을 확인합니다.

> **Note**
>
> 속성 포함을 요청하는 경우 `address` 속성 값을 정수로 지정해야 합니다. 예를 들어 `"address": 1`입니다.

`id`

요청에 대한 임의의 ID입니다. 이 속성을 사용하여 입력 요청을 출력 응답에 매핑할 수 있습니다. 이 속성을 지정하면 구성 요소가 응답 개체의 `id` 속성을 이 값으로 설정합니다.

유형: `string`

Example 입력 예: 코일 요청 읽기

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

출력 데이터

이 구성 요소는 기본적으로 응답을 다음 MQTT 주제에 출력 데이터로 게시합니다. 이 주제를 레거시 구독 라우터 구성 요소의 `subject` 구성에서 로 지정해야 합니다. 사용자 지정 구성 요소에서 이 항목의 메시지를 구독하는 방법에 대한 자세한 내용은 을 참조하십시오MQTT 메시지 게시/구독 AWS IoT Core.

기본 주제 (AWS IoT Core MQTT): `modbus/adapter/response`

응답 메시지의 모양은 요청 작업과 응답 상태에 따라 달라집니다. 예제는 예제 요청 및 응답 섹션을 참조하세요.

모든 응답에는 다음 속성이 포함됩니다.

`response`

모드버스 RTU 장치의 응답.

다음 정보가 `object` 포함된 유형:

`status`

요청 상태입니다. 상태는 다음 값 중 하나일 수 있습니다.

- `Success`— 요청이 유효했고, 컴포넌트가 Modbus RTU 네트워크에 요청을 보냈고, Modbus RTU 네트워크가 응답을 반환했습니다.
- `Exception`— 요청이 유효했고, 컴포넌트가 Modbus RTU 네트워크에 요청을 보냈고, Modbus RTU 네트워크에서 예외를 반환했습니다. 자세한 설명은 응답 상태: 예외 섹션을 참조하세요.
- `No Response`— 요청이 유효하지 않아 구성 요소가 Modbus RTU 네트워크에 요청을 보내기 전에 오류가 발생했습니다. 자세한 설명은 응답 상태: 응답 없음 섹션을 참조하세요.

`operation`

구성 요소가 요청한 작업입니다.

`device`

구성 요소가 요청을 보낸 장치.

`payload`

모드버스 RTU 장치의 응답. 인 `status` 경우 이 객체에는 오류에 대한 설명이 있는 `error` 속성 (예:) 만 포함됩니다. `No Response [Input/Output] No Response received from the remote unit`

`id`

요청의 ID로, 어떤 응답이 어떤 요청에 해당하는지 식별하는 데 사용할 수 있습니다.

> **Note**
>
> 쓰기 작업에 대한 응답은 단순히 요청의 에코일 뿐입니다. 쓰기 응답에는 의미 있는 정보가 포함되지 않지만 응답 상태를 확인하여 요청의 성공 또는 실패 여부를 확인하는 것이 좋습니다.

Example 출력 예: 성공

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example 출력 예: 실패

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "MyRequest"
}
```

더 많은 예제는 예제 요청 및 응답 단원을 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.

읽기 코일

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"
}
```

홀딩 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,
    "count": 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 작성

요청 예:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
```

```
}
```

여러 개의 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 작성

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"
}
```

마스크 쓰기 레지스터

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 읽기, 쓰기

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**
>
> 응답에는 컴포넌트가 읽는 레지스터가 포함됩니다.

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 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"
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass 루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
```

라이선스

이 구성 요소에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- 파이모드버스/BSD 라이선스
- 페이시리얼/BSD 라이선스

이 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.8	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.7	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	버그 수정 및 개선 • `ReadDiscreteInput` 작업 관련 문제를 수정합니다.
2.1.4	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.2	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • Modbus RTU 장치와의 직렬 통신을 구성하기 위해 지정할 수 있는 `ModbusBaudRate` `ModbusByteSize` `ModbusParity` ,, 및 `ModbusStopBits` 옵션을 추가합니다.
2.0.8	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

MQTT 브리지

MQTT 브리지 구성 요소 (`aws.greengrass.clientdevices.mqtt.Bridge`) 는 클라이언트 장치, 로컬 Greengrass 게시/구독 및 간에 MQTT 메시지를 중계합니다. AWS IoT Core 이 구성 요소를 사용하여 사용자 지정 구성 요소에서 클라이언트 장치의 MQTT 메시지를 처리하고 클라이언트 장치를 와 동기화할 수 있습니다. AWS 클라우드

> **Note**
>
> 클라이언트 디바이스는 Greengrass 코어 디바이스에 연결하여 MQTT 메시지와 데이터를 전송하여 처리하는 로컬 IoT 디바이스입니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

이 구성 요소를 사용하여 다음 메시지 브로커 간에 메시지를 릴레이할 수 있습니다.

- 로컬 MQTT — 로컬 MQTT 브로커는 클라이언트 디바이스와 코어 디바이스 간의 메시지를 처리합니다.
- 로컬 게시/구독 — 로컬 Greengrass 메시지 브로커는 코어 디바이스의 구성 요소 간 메시지를 처리합니다. Greengrass 구성 요소에서 이러한 메시지와 상호 작용하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- AWS IoT Core— AWS IoT Core MQTT 브로커는 IoT 장치와 AWS 클라우드 대상 간의 메시지를 처리합니다. Greengrass 구성 요소에서 이러한 메시지와 상호 작용하는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.3.x
- 2.2.x

- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 기본 포트 8883 이외의 포트를 사용하도록 코어 디바이스의 MQTT 브로커 구성 요소를 구성하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. 브로커가 작동하는 포트에 연결하도록 구성하십시오.
- MQTT 브리지 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.3.0

다음 표에는 이 구성 요소의 버전 2.3.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.5.0	하드

2.2.5 and 2.2.6

다음 표에는 이 구성 요소의 버전 2.2.5 및 2.2.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.5.0	하드

2.2.3 and 2.2.4

다음 표에는 이 구성 요소의 버전 2.2.3 및 2.2.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.4.0	하드

2.2.0 – 2.2.2

다음 표에는 이 구성 요소의 버전 2.2.0~2.2.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.3.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.0.0 <2.2.0	하드

2.0.0 to 2.1.0

다음 표에는 이 구성 요소의 버전 2.0.0~2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.0.0 <2.1.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

2.3.0

`mqttTopicMapping`

연결하려는 주제 매핑. 이 구성 요소는 소스 주제의 메시지를 구독하고 수신한 메시지를 대상 주제에 게시합니다. 각 주제 매핑은 주제, 원본 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topicMappingNameKey`

이 주제 매핑의 이름. *`topicMappingName##`* 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topic`

소스 브로커와 대상 브로커 사이를 연결하는 주제 또는 주제 필터.

+및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

> **Note**
>
> Pubsub소스 브로커에서 MQTT 주제 와일드카드를 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소를 사용해야 합니다.

`targetTopicPrefix`

이 구성 요소가 메시지를 릴레이할 때 대상 주제에 추가할 접두사입니다.

`source`

소스 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

`target`

대상 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

mqtt5 RouteOptions

(선택 사항) 소스 주제에서 대상 주제로 메시지를 브리징하기 위한 주제 매핑을 구성하는 옵션을 제공합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`mqtt5 RouteOptionsNameKey`

주제 매핑을 위한 경로 옵션의 이름. *`RouteOptionsNameKeymqtt5#`* 필드에 정의된 일치하는 *`topicMappingName##`* 바꿉니다. `mqttTopicMapping`

이 객체에는 다음 정보가 포함되어 있어야 합니다.

노로컬

(선택 사항) 활성화된 경우 브리지는 브리지 자체에서 게시한 주제에 대한 메시지를 전달하지 않습니다. 이 방법을 사용하면 다음과 같이 루프를 방지할 수 있습니다.

```
{
    "mqtt5RouteOptions": {
        "toIoTCore": {
            "noLocal": true
        }
    },
    "mqttTopicMapping": {
        "toIoTCore": {
            "topic": "device",
            "source": "LocalMqtt",
            "target": "IotCore"
        },
```

```
        "toLocal": {
            "topic": "device",
            "source": "IotCore",
            "target": "LocalMqtt"
        }
    }
}
```

noLocal가 있는 라우트에만 지원됩니다LocalMqtt. source

기본값: false

retainAsPublished

(선택 사항) 활성화된 경우 브리지에서 전달된 메시지는 해당 경로에 대해 브로커에 게시된 메시지와 동일한 retain 플래그를 갖습니다.

retainAsPublished가 있는 라우트에서만 지원됩니다. source LocalMqtt

기본값: false

mqtt

(선택 사항) 로컬 브로커와 통신하기 위한 MQTT 프로토콜 설정

version

(선택 사항) 브리지에서 로컬 브로커와 통신하는 데 사용하는 MQTT 프로토콜 버전입니다. 핵 구성에서 선택한 MQTT 버전과 동일해야 합니다.

다음 중에서 선택합니다.

- mqtt3
- mqtt5

mqttTopicMapping객체의 source 또는 target 필드가 로 설정된 경우 MQTT 브로커를 배포해야 합니다. LocalMqtt mqtt5옵션을 선택한 경우 를 사용해야 합니다. 맷 5 브로커 (EMQX)

기본값: mqtt3

ackTimeoutSeconds

(선택 사항) 작업이 실패하기 전에 PUBACK, SUBACK 또는 UNSUBACK 패킷을 기다리는 시간 간격.

기본값: 60

connAckTimeout밀리초

(선택 사항) 연결을 종료하기 전에 CONNACK 패킷을 기다리는 시간 간격.

기본값: 20000 (20초)

pingTimeoutMs

(선택 사항) 브리지가 로컬 브로커로부터 PINGACK 메시지를 수신할 때까지 기다리는 시간 (밀리초) 입니다. 대기 시간이 제한 시간을 초과하면 브리지가 닫히고 MQTT 연결이 다시 열립니다. 이 값은 다음보다 작아야 합니다. `keepAliveTimeoutSeconds`

기본값: 30000 (30초)

keepAliveTimeout초

(선택 사항) MQTT 연결을 유지하기 위해 브리지가 전송하는 각 PING 메시지 사이의 시간 (초) 입니다. 이 값은 보다 `pingTimeoutMs` 커야 합니다.

기본값: 60

maxReconnectDelayMs

(선택 사항) MQTT가 다시 연결되는 데 걸리는 최대 시간 (초) 입니다.

기본값: 30000 (30초)

minReconnectDelayMs

(선택 사항) MQTT가 다시 연결되는 데 걸리는 최소 시간 (초) 입니다.

최대 수신

(선택 사항) 브리지가 전송할 수 있는 승인되지 않은 QoS1 패킷의 최대 수입니다.

기본값: 100

maximumPacketSize

클라이언트가 MQTT 패킷에 허용하는 최대 바이트 수입니다.

기본값: null (제한 없음)

sessionExpiryInterval

(선택 사항) 브리지와 로컬 브로커 간에 세션이 지속되도록 요청할 수 있는 시간 (초).

기본값: 4294967295 (세션은 만료되지 않음)

`brokerUri`

(선택 사항) 로컬 MQTT 브로커의 URI입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다. 다음 형식을 사용하고 *###* MQTT 브로커가 작동하는 포트로 교체하십시오. `ssl://localhost:`*`port`*

기본값: `ssl://localhost:8883`

startupTimeoutSeconds

(선택 사항) 구성 요소를 시작하는 데 걸리는 최대 시간 (초). 이 제한 시간을 BROKEN 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: `120`

Example 예: 구성 병합 업데이트

다음 예제 구성 업데이트는 다음을 지정합니다.

- 클라이언트 장치의 메시지를 `clients/+/hello/world` 주제 필터와 일치하는 AWS IoT Core 주제로 릴레이합니다.
- 주제 필터와 일치하는 주제에 대해 클라이언트 장치의 메시지를 로컬 게시/구독으로 중계하고 대상 `clients/+/detections` 주제에 `events/input/` 접두사를 추가합니다. 결과 대상 주제는 주제 필터와 일치합니다. `events/input/clients/+/detections`
- 클라이언트 장치의 메시지를 주제 필터와 일치하는 주제로 전달하고 대상 `clients/+/status` 주제에 `$aws/rules/StatusUpdateRule/` 접두사를 추가합니다. AWS IoT Core 이 예제에서는 Basic Ingest를 사용하여 비용을 절감하기 `StatusUpdateRule` 위해 이름이 지정된 AWS IoT규칙에 이러한 메시지를 직접 전달합니다.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
```

```
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example 예: MQTT 5 구성

다음 예제 컨피그레이션은 다음을 업데이트합니다.

- 브리지가 로컬 브로커와 함께 MQTT 5 프로토콜을 사용할 수 있도록 합니다.
- MQTT retain을 주제 매핑을 위한 게시된 설정으로 구성합니다. `ClientDeviceHelloWorld`

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

`mqttTopicMapping`

연결하려는 주제 매핑. 이 구성 요소는 소스 주제의 메시지를 구독하고 수신한 메시지를 대상 주제에 게시합니다. 각 주제 매핑은 주제, 원본 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름. *topicMappingName##* 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topic`

소스 브로커와 대상 브로커 사이를 연결하는 주제 또는 주제 필터.

+및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

> **Note**
>
> Pubsub소스 브로커에서 MQTT 주제 와일드카드를 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소를 사용해야 합니다.

`targetTopicPrefix`

이 구성 요소가 메시지를 릴레이할 때 대상 주제에 추가할 접두사입니다.

`source`

소스 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이

스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

`target`

대상 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

`brokerUri`

(선택 사항) 로컬 MQTT 브로커의 URI입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다. 다음 형식을 사용하고 *###* MQTT 브로커가 작동하는 포트로 교체하십시오. `ssl://localhost:`*`port`*

기본값: `ssl://localhost:8883`

startupTimeoutSeconds

(선택 사항) 구성 요소를 시작하는 데 걸리는 최대 시간 (초). 이 제한 시간을 `BROKEN` 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: `120`

Example 예: 구성 병합 업데이트

다음 예제 구성 업데이트는 다음을 지정합니다.

- 클라이언트 장치의 메시지를 `clients/+/hello/world` 주제 필터와 일치하는 AWS IoT Core 주제로 릴레이합니다.
- 주제 필터와 일치하는 주제에 대해 클라이언트 장치의 메시지를 로컬 게시/구독으로 중계하고 대상 `clients/+/detections` 주제에 `events/input/` 접두사를 추가합니다. 결과 대상 주제는 주제 필터와 일치합니다. `events/input/clients/+/detections`
- 클라이언트 장치의 메시지를 주제 필터와 일치하는 주제로 전달하고 대상 `clients/+/status` 주제에 `$aws/rules/StatusUpdateRule/` 접두사를 추가합니다. AWS IoT Core 이 예제에서는 Basic Ingest를 사용하여 비용을 절감하기 `StatusUpdateRule` 위해 이름이 지정된 AWS IoT규칙에 이러한 메시지를 직접 전달합니다.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.2.0 - 2.2.5

`mqttTopicMapping`

연결하려는 주제 매핑. 이 구성 요소는 소스 주제의 메시지를 구독하고 수신한 메시지를 대상 주제에 게시합니다. 각 주제 매핑은 주제, 원본 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름. *topicMappingName##* 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topic`

소스 브로커와 대상 브로커 사이를 연결하는 주제 또는 주제 필터.

+및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

> **Note**
>
> Pubsub소스 브로커에서 MQTT 주제 와일드카드를 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소를 사용해야 합니다.

`targetTopicPrefix`

이 구성 요소가 메시지를 릴레이할 때 대상 주제에 추가할 접두사입니다.

`source`

소스 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

source그리고 target 달라야 합니다.

target

대상 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- Pubsub— 현지 Greengrass 게시/구독 메시지 브로커.
- IotCore— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

source그리고 target 달라야 합니다.

brokerUri

(선택 사항) 로컬 MQTT 브로커의 URI입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다. 다음 형식을 사용하고 *###* MQTT 브로커가 작동하는 포트로 교체하십시오. ssl://localhost:*port*

기본값: ssl://localhost:8883

Example 예: 구성 병합 업데이트

다음 예제 구성 업데이트는 다음을 지정합니다.

- 클라이언트 장치의 메시지를 clients/+/hello/world 주제 필터와 일치하는 AWS IoT Core 주제로 릴레이합니다.
- 주제 필터와 일치하는 주제에 대해 클라이언트 장치의 메시지를 로컬 게시/구독으로 중계하고 대상 clients/+/detections 주제에 events/input/ 접두사를 추가합니다. 결과 대상 주제는 주제 필터와 일치합니다. events/input/clients/+/detections
- 클라이언트 장치의 메시지를 주제 필터와 일치하는 주제로 전달하고 대상 clients/+/status 주제에 $aws/rules/StatusUpdateRule/ 접두사를 추가합니다. AWS IoT Core 이 예제에

서는 Basic Ingest를 사용하여 비용을 절감하기 `StatusUpdateRule` 위해 이름이 지정된 AWS IoT규칙에 이러한 메시지를 직접 전달합니다.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.1.x

`mqttTopicMapping`

연결하려는 주제 매핑. 이 구성 요소는 소스 주제의 메시지를 구독하고 수신한 메시지를 대상 주제에 게시합니다. 각 주제 매핑은 주제, 원본 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topicMappingNameKey`

이 주제 매핑의 이름. *`topicMappingName##`* 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topic`

소스 브로커와 대상 브로커 사이를 연결하는 주제 또는 주제 필터.

`LocalMqtt`or `IotCore` 소스 브로커를 지정하는 경우 + 및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

`source`

소스 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

`target`

대상 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

source그리고 target 달라야 합니다.

`brokerUri`

(선택 사항) 로컬 MQTT 브로커의 URI입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다. 다음 형식을 사용하고 *###* MQTT 브로커가 작동하는 포트로 교체하십시오. `ssl://localhost:`*`port`*

기본값: `ssl://localhost:8883`

Example 예: 구성 병합 업데이트

다음 예제 구성 업데이트는 클라이언트 장치의 메시지를 `clients/MyClientDevice1/hello/world` 및 `clients/MyClientDevice2/hello/world` 항목으로 릴레이하도록 AWS IoT Core 지정합니다.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.0.x

`mqttTopicMapping`

연결하려는 주제 매핑. 이 구성 요소는 소스 주제의 메시지를 구독하고 수신한 메시지를 대상 주제에 게시합니다. 각 주제 매핑은 주제, 원본 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름. *topicMappingName##* 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꾸십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topic`

소스 브로커와 대상 브로커 사이를 연결하는 주제 또는 주제 필터.

`LocalMqttor IotCore` 소스 브로커를 지정하는 경우 + 및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

`source`

소스 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

`target`

대상 메시지 브로커. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt`— 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커.
- `Pubsub`— 현지 Greengrass 게시/구독 메시지 브로커.
- `IotCore`— AWS IoT Core MQTT 메시지 브로커.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

`source`그리고 `target` 달라야 합니다.

Example 예: 구성 병합 업데이트

다음 예제 구성 업데이트는 클라이언트 장치의 메시지를 `clients/MyClientDevice1/hello/world` 및 `clients/MyClientDevice2/hello/world` 항목으로 릴레이하도록 AWS IoT Core 지정합니다.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.3.1	버그 수정 및 개선 로컬 MQTT 클라이언트가 연결 해제 루프에 빠지는 문제를 수정합니다.
2.3.0	새로운 기능 로컬 MQTT AWS IoT Core 소스와 로컬 MQTT 소스 간의 브리징을 위한 MQTT5 지원을 추가합니다.

버전	변경
2.2.6	새로운 기능 새 `startupTimeoutSeconds` 구성 옵션을 추가합니다.
2.2.5	클라이언트 장치 인증 버전 2.4.0 릴리스용으로 버전이 업데이트되었습니다.
2.2.4	Greengrass 클라이언트 장치 인증 버전 2.3.0 릴리스에 대한 버전이 업데이트되었습니다.
2.2.3	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.
2.2.2	버그 수정 및 개선 • 로깅 조정.
2.2.1	버그 수정 및 개선 MQTT 브리지에서 MQTT 주제를 구독하지 못하는 문제가 수정되었습니다.
2.2.0	새로운 기능 • 로컬 게시/구독을 소스 메시지 브로커로 지정할 때 MQTT 주제 와일드카드 (#및+) 에 대한 지원을 추가합니다. 이 기능을 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소가 필요합니다. • 메시지를 릴레이할 때 대상 주제에 접두사를 추가하도록 MQTT 브리지를 구성하도록 지정할 수 있는 `targetTopicPrefix` 옵션을 추가합니다.
2.1.1	버그 수정 및 개선 • 이 구성 요소가 구성 재설정 업데이트를 처리하는 방식과 관련된 문제를 수정합니다. • 인증서가 교체될 때 MQTT 클라이언트 연결이 끊기는 빈도를 줄입니다.
2.1.0	새로운 기능 • 기본이 아닌 MQTT 브로커 포트를 사용할 수 있는 `brokerUri` 매개 변수를 추가합니다.
2.0.1	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.

버전	변경
2.0.0	초기 버전

MQTT 3.1.1 브로커 (모켓)

Mocquette MQTT 브로커 구성 요소 (`aws.greengrass.clientdevices.mqtt.Moquette`) 는 클라이언트 장치와 Greengrass 코어 장치 간의 MQTT 메시지를 처리합니다. 이 구성 요소는 Moquette MQTT 브로커의 수정된 버전을 제공합니다. 이 MQTT 브로커를 배포하여 경량 MQTT 브로커를 실행하십시오. MQTT 브로커를 선택하는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 브로커를 선택하세요

이 브로커는 MQTT 3.1.1 프로토콜을 구현합니다. 여기에는 QoS 0, QoS 1, QoS 2 보존 메시지, 라스트 윌 메시지 및 영구 세션에 대한 지원이 포함됩니다.

> **Note**
>
> 클라이언트 디바이스는 Greengrass 코어 디바이스에 연결하여 MQTT 메시지와 데이터를 전송하여 처리하는 로컬 IoT 디바이스입니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 MQTT 브로커가 작동하는 포트에서 연결을 수락할 수 있어야 합니다. 이 구성 요소는 기본적으로 포트 8883에서 MQTT 브로커를 실행합니다. 이 구성 요소를 구성할 때 다른 포트를 지정할 수 있습니다.

  다른 포트를 지정하고 MQTT 브리지 구성 요소를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 사용하도록 구성하십시오.

  다른 포트를 지정하고 IP 탐지기 구성 요소를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 탐지기 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 보고하도록 구성하십시오.

- 모켓 MQTT 브로커 컴포넌트는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 AWS IoT Greengrass 종속 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.3.2 – 2.3.5

다음 표에는 이 구성 요소의 버전 2.3.2~2.3.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.5.0	하드

2.3.0 and 2.3.1

다음 표에는 이 구성 요소의 버전 2.3.0 및 2.3.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.4.0	하드

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.3.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.0.0 <2.2.0	하드

2.0.0 - 2.0.2

다음 표에는 이 구성 요소의 버전 2.0.0~2.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.0.0 <2.1.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`moquette`

(선택 사항) 사용할 Moquette MQTT 브로커 구성. 이 구성 요소에서 Moqeutte 구성 옵션의 하위 세트를 구성할 수 있습니다. 자세한 내용은 Moquette 구성 파일의 인라인 설명을 참조하십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`ssl_port`

(선택 사항) MQTT 브로커가 작동하는 포트입니다.

> **Note**
>
> 다른 포트를 지정하고 MQTT 브리지 구성 요소를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 사용하도록 구성하십시오.
> 다른 포트를 지정하고 IP 탐지기 구성 요소를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 탐지기 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 보고하도록 구성하십시오.

기본값: 8883

`host`

(선택 사항) MQTT 브로커가 바인딩되는 인터페이스입니다. 예를 들어 MQTT 브로커가 특정 로컬 네트워크에만 바인딩되도록 이 매개변수를 변경할 수 있습니다.

기본값: `0.0.0.0` (모든 네트워크 인터페이스에 바인딩)

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간 (초). 이 제한 시간을 BROKEN 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: `120`

Example 예: 구성 병합 업데이트

다음 예제 컨피그레이션은 포트 443에서 MQTT 브로커를 작동하도록 지정합니다.

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.3.5	버그 수정 및 개선 • 모켓을 버전 0.17로 업데이트했습니다.
2.3.4	버그 수정 및 개선 • 클라이언트에서 메시지를 보내거나 받을 때 중복된 클라이언트 ID로 인해 잘못된 세션 오류가 발생할 수 있는 문제를 수정합니다. 이 문제로 인해 클라이언트 세션이 종료되었습니다.
2.3.3	새로운 기능 새 `startupTimeoutSeconds` 구성 옵션을 추가합니다.
2.3.2	클라이언트 장치 인증 버전 2.4.0 릴리스용으로 버전이 업데이트되었습니다.
2.3.1	버그 수정 및 개선 • 재연결을 시도한 후 잘못된 세션으로 인해 클라이언트 연결이 끊길 수 있는 경합 상태를 수정합니다.

버전	변경
2.3.0	인증서 체인에 대한 지원을 추가합니다.
2.2.0	클라이언트 장치 인증 버전 2.2.0 릴리스의 버전이 업데이트되었습니다.
2.1.0	버그 수정 및 개선 • Moquette 버전 0.16을 사용하도록 이 구성 요소를 업데이트하여 성능을 개선하고 기타 여러 개선 사항을 포함합니다. • 특정 시나리오에서 로컬 MQTT 서버 인증서가 의도한 것보다 더 자주 교체되는 문제를 수정합니다. 이 수정 사항을 적용하려면 v2.1.0 이상의 클라이언트 장치 인증 구성 요소도 사용해야 합니다.
2.0.2	버그 수정 및 개선 • 최대 MQTT 메시지 크기를 8,092바이트에서 128킬로바이트로 늘립니다. 메시지 크기 제한에는 메시지 헤더가 포함되므로 유효 MQTT 메시지 페이로드 한도는 약간 작습니다. • 파라미터에 정수 값에 대한 지원을 추가합니다. `ssl_port`
2.0.1	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.0	초기 버전

맷 5 브로커 (EMQX)

EMQX MQTT 브로커 구성 요소 (`aws.greengrass.clientdevices.mqtt.EMQX`) 는 클라이언트 디바이스와 그린그래스 코어 디바이스 간의 MQTT 메시지를 처리합니다. 이 구성 요소는 EMQX MQTT 5.0 브로커의 수정된 버전을 제공합니다. 이 MQTT 브로커를 배포하여 클라이언트 디바이스와 코어 디바이스 간 통신에 MQTT 5 기능을 사용할 수 있습니다. MQTT 브로커를 선택하는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 브로커를 선택하세요

이 브로커는 MQTT 5.0 프로토콜을 구현합니다. 여기에는 세션 및 메시지 만료 간격, 사용자 속성, 공유 구독, 주제 별칭 등에 대한 지원이 포함됩니다. MQTT 5는 MQTT 3.1.1과 역호환되므로 Moquette MQTT 3.1.1 브로커를 실행하는 경우 이를 EMQX MQTT 5 브로커로 교체할 수 있으며 클라이언트 디바이스는 평소와 같이 계속 연결되고 작동할 수 있습니다.

> **Note**
>
> 클라이언트 디바이스는 Greengrass 코어 디바이스에 연결하여 MQTT 메시지와 데이터를 전송하여 처리하는 로컬 IoT 디바이스입니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.0.x
- 1.2.x
- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 MQTT 브로커가 작동하는 포트에서 연결을 수락할 수 있어야 합니다. 이 구성 요소는 기본적으로 포트 8883에서 MQTT 브로커를 실행합니다. 이 구성 요소를 구성할 때 다른 포트를 지정할 수 있습니다.

  다른 포트를 지정하고 MQTT 브리지 구성 요소를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 사용하도록 구성하십시오.

  다른 포트를 지정하고 IP 탐지기 구성 요소를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 탐지기 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 보고하도록 구성하십시오.
- Linux 코어 디바이스에서 Docker는 코어 디바이스에 설치 및 구성되었습니다.
  - 도커 엔진 1.9.1 이상이 그린그래스 코어 디바이스에 설치되었습니다. 버전 20.10은 Core 소프트웨어와 함께 작동하는 것으로 검증된 최신 버전입니다. AWS IoT Greengrass Docker 컨테이너를 실행하는 구성 요소를 배포하려면 먼저 코어 디바이스에 Docker를 직접 설치해야 합니다.
  - Docker 데몬은 이 구성 요소를 배포하기 전에 코어 디바이스에서 시작되어 실행되었습니다.
  - 이 구성 요소를 실행하는 시스템 사용자에게는 루트 또는 관리자 권한이 있어야 합니다. 또는 `docker` 그룹의 시스템 사용자로 이 구성 요소를 실행하고 권한 없이 EQMX MQTT 브로커를 `false` 실행하도록 이 구성 요소의 `requiresPrivileges` 옵션을 구성할 수 있습니다.
- EMQX MQTT 브로커 구성 요소는 VPC에서 실행되도록 지원됩니다.
- EMQX MQTT 브로커 구성 요소는 플랫폼에서 지원되지 않습니다. `armv7`

의존성

구성 요소를 배포하면 호환되는 버전의 AWS IoT Greengrass 종속 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹

션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.5.0	하드

1.2.2 – 1.2.3

다음 표에는 이 구성 요소의 버전 1.2.2 ~ 1.2.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.5.0	하드

1.2.0 and 1.2.1

다음 표에는 이 구성 요소의 버전 1.2.0 및 1.2.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.4.0	하드

1.0.0 and 1.1.0

다음 표에는 이 구성 요소의 버전 1.0.0 및 1.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.3.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

2.0.0

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

> **Important**
>
> MQTT 5 브로커 (EMQX) 구성 요소 버전 2를 사용하는 경우 구성 파일을 업데이트해야 합니다. 버전 1 구성 파일은 버전 2에서 작동하지 않습니다.

EmqxConfig

(선택 사항) 사용할 EMQX MQTT 브로커 구성입니다. 이 구성 요소에서 EMQX 구성 옵션을 설정할 수 있습니다.

EMQX 브로커를 사용하는 경우 Greengrass는 기본 구성을 사용합니다. 이 컨피그레이션은 이 필드를 사용하여 수정하지 않는 한 사용됩니다.

다음 구성 설정을 수정하면 EMQX Broker 구성 요소가 다시 시작됩니다. 기타 구성 변경 사항은 구성 요소를 다시 시작하지 않아도 적용됩니다.

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`보안에 민감한 옵션을 구성할 수 있습니다. 여기에는 TLS 설정, 인증 및 권한 부여 공급자가 포함됩니다. 상호 TLS 인증을 사용하는 기본 구성과 Greengrass 클라이언트 장치 인증 공급자를 사용하는 것이 좋습니다.

Example 예: 기본 구성

다음 예는 MQTT 5 (EMQX) 브로커에 설정된 기본값을 보여줍니다. 구성 설정을 사용하여 이러한 설정을 재정의할 수 있습니다. `emqxConfig`

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
  },
  "node": {
    "cookie": "<placeholder>"
  },
  "listeners": {
     "ssl": {
       "default": {
         "ssl_options": {
           "keyfile": "{work:path}\\data\\key.pem",
           "certfile": "{work:path}\\data\\cert.pem",
           "cacertfile": null,
           "verify": "verify_peer",
           "versions": ["tlsv1.3", "tlsv1.2"],
           "fail_if_no_peer_cert": true
         }
       }
     },
     "tcp": {
       "default": {
         "enabled": false
       }
     },
     "ws": {
       "default": {
         "enabled": false
       }
     },
     "wss": {
       "default": {
         "enabled": false
       }
     }
  },
  "plugins": {
    "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
    "install_dir": "plugins"
  }
}
```

인증 모드

(선택 사항) 브로커의 권한 제공자를 설정합니다. 다음 값 중 하나일 수 있습니다.

- `enabled`— (기본값) Greengrass 인증 및 권한 부여 공급자를 사용합니다.
- `bypass_on_failure`— Greengrass 인증 공급자를 사용하고, Greengrass가 인증 또는 권한 부여를 거부하는 경우 EMQX 제공자 체인에 남아 있는 인증 공급자를 사용하십시오.
- `bypass`— Greengrass 공급자가 비활성화되었습니다. 인증 및 권한 부여는 EMQX 공급자 체인에 의해 처리됩니다.

`requiresPrivilege`

(선택 사항) Linux 코어 디바이스에서는 루트 또는 관리자 권한 없이 EMQX MQTT 브로커를 실행하도록 지정할 수 있습니다. 이 옵션을 로 `false` 설정하는 경우 이 구성 요소를 실행하는 시스템 사용자가 그룹의 구성원이어야 합니다. `docker`

기본값: `true`

`startupTimeoutSeconds`

(선택 사항) EMQX MQTT 브로커가 시작되는 최대 시간 (초) 입니다. 이 제한 시간을 `BROKEN` 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: `90`

`ipcTimeoutSeconds`

(선택 사항) Greengrass 핵이 IPC (프로세스 간 통신) 요청에 응답할 때까지 구성 요소가 대기하는 최대 시간 (초) 입니다. 이 구성 요소가 클라이언트 장치가 인증되었는지 확인할 때 시간 초과 오류를 보고하면 이 숫자를 늘리십시오.

기본값: `5`

`crtLogLevel`

(선택 사항) CRT (AWS공용 런타임) 라이브러리의 로그 수준.

EMQX MQTT 브로커 로그 레벨 (in) 이 기본값입니다. `log.level` `emqx`

`restartIdentifier`

(선택 사항) EMQX MQTT 브로커를 다시 시작하려면 이 옵션을 구성합니다. 이 구성 값이 변경되면 이 구성 요소는 MQTT 브로커를 다시 시작합니다. 이 옵션을 사용하여 클라이언트 디바이스의 연결을 강제로 끊을 수 있습니다.

`dockerOptions`

(선택 사항) Linux 운영 체제에서만 Docker 명령줄에 매개 변수를 추가하도록 이 옵션을 구성합니다. 예를 들어, 추가 포트를 매핑하려면 `-p` Docker 매개변수를 사용하십시오.

```
"-p 1883:1883"
```

Example 예: v1.x 구성 파일을 v2.x로 업데이트

다음 예제는 v1.x 구성 파일을 버전 2.x로 업데이트하는 데 필요한 변경 사항을 보여줍니다.

버전 1.x 구성 파일:

```
{
    "emqx": {
        "listener.ssl.external": "443",
        "listener.ssl.external.max_connections": "1024000",
        "listener.ssl.external.max_conn_rate": "500",
        "listener.ssl.external.rate_limit": "50KB,5s",
        "listener.ssl.external.handshake_timeout": "15s",
        "log.level": "warning"
    },
    "mergeConfigurationFiles": {
        "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
    }
}
```

v2에 해당하는 구성 파일:

```
{
    "emqxConfig": {
        "listeners": {
            "ssl": {
                "default": {
                   "bind": "8883",
                   "max_connections": "1024000",
                   "max_conn_rate": "500",
                   "handshake_timeout": "15s",
                }
            }
```

```
        },
        "log": {
            "console": {
              "enable": true,
              "level": "warning"
            }
        }
    },
    "authMode": "enabled"
}
```

listener.ssl.external.rate_limit구성 항목에 해당하는 항목은 없습니다. use_greengrass_managed_certificates구성 옵션이 제거되었습니다.

Example 예: 브로커의 새 포트 설정

다음 예에서는 MQTT 브로커가 작동하는 포트를 기본 8883에서 포트 1234로 변경합니다. Linux를 사용하는 경우 필드를 포함하십시오. dockerOptions

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example 예: MQTT 브로커의 로그 수준 조정

다음 예제는 MQTT 브로커의 로그 레벨을 로 변경합니다. debug 다음 로그 수준 중에서 선택할 수 있습니다.

- debug
- info
- notice
- warning

- error
- critical
- alert
- emergency

기본 로그 수준은 입니다warning.

```
{
  "emqxConfig": {
    "log": {
      "console": {
         "level": "debug"
      }
    }
  }
}
```

Example 예: EMQX 대시보드 활성화

다음 예제는 브로커를 모니터링하고 관리할 수 있도록 EMQX 대시보드를 활성화합니다. Linux를 사용하는 경우 필드를 포함하십시오. dockerOptions

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

emqx

(선택 사항) 사용할 EMQX MQTT 브로커 구성. 이 구성 요소에서 일부 EMQX 구성 옵션을 구성할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`listener.ssl.external`

(선택 사항) MQTT 브로커가 작동하는 포트입니다.

> **Note**
>
> 다른 포트를 지정하고 MQTT 브리지 구성 요소를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 사용하도록 구성하십시오.
> 다른 포트를 지정하고 IP 탐지기 구성 요소를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 탐지기 v2.1.0 이상을 사용해야 합니다. MQTT 브로커가 작동하는 포트를 보고하도록 구성하십시오.

기본값: `8883`

`listener.ssl.external.max_connections`

(선택 사항) MQTT 브로커가 지원하는 최대 동시 연결 수입니다.

기본값: `1024000`

`listener.ssl.external.max_conn_rate`

(선택 사항) MQTT 브로커가 수신할 수 있는 초당 최대 새 연결 수입니다.

기본값: `500`

`listener.ssl.external.rate_limit`

(선택 사항) MQTT 브로커에 대한 모든 연결의 대역폭 제한. 해당 대역폭의 대역폭과 지속 시간을 쉼표 (`,`) 로 구분하여 다음 형식으로 지정합니다. `bandwidth,duration` 예를 들어 MQTT 브로커를 5초마다 50킬로바이트 (KB) 의 데이터로 `50KB,5s` 제한하도록 지정할 수 있습니다.

`listener.ssl.external.handshake_timeout`

(선택 사항) MQTT 브로커가 새 연결 인증을 완료할 때까지 기다리는 시간입니다.

기본값: 15s

`mqtt.max_packet_size`

(선택 사항) MQTT 메시지의 최대 크기.

기본값: 268435455 (256MB에서 1을 뺀 값)

`log.level`

(선택 사항) MQTT 브로커의 로그 레벨입니다. 다음 옵션 중 하나를 선택합니다.

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

기본 로그 수준은 입니다. `warning`

`requiresPrivilege`

(선택 사항) Linux 코어 디바이스에서는 루트 또는 관리자 권한 없이 EMQX MQTT 브로커를 실행하도록 지정할 수 있습니다. 이 옵션을 로 `false` 설정하는 경우 이 구성 요소를 실행하는 시스템 사용자가 그룹의 구성원이어야 합니다. `docker`

기본값: `true`

`startupTimeoutSeconds`

(선택 사항) EMQX MQTT 브로커가 시작되는 최대 시간 (초) 입니다. 이 제한 시간을 `BROKEN` 초과하면 구성 요소의 상태가 로 변경됩니다.

기본값: `90`

`ipcTimeoutSeconds`

(선택 사항) Greengrass 핵이 IPC (프로세스 간 통신) 요청에 응답할 때까지 구성 요소가 대기하는 최대 시간 (초) 입니다. 이 구성 요소가 클라이언트 장치가 인증되었는지 확인할 때 시간 초과 오류를 보고하면 이 숫자를 늘리십시오.

기본값: 5

`crtLogLevel`

(선택 사항) CRT (AWS공용 런타임) 라이브러리의 로그 수준.

EMQX MQTT 브로커 로그 레벨 (in) 이 기본값입니다. `log.level` `emqx`

`restartIdentifier`

(선택 사항) EMQX MQTT 브로커를 다시 시작하려면 이 옵션을 구성합니다. 이 구성 값이 변경되면 이 구성 요소는 MQTT 브로커를 다시 시작합니다. 이 옵션을 사용하여 클라이언트 디바이스의 연결을 강제로 끊을 수 있습니다.

`dockerOptions`

(선택 사항) Linux 운영 체제에서만 Docker 명령줄에 매개 변수를 추가하도록 이 옵션을 구성합니다. 예를 들어, 추가 포트를 매핑하려면 `-p` Docker 매개변수를 사용하십시오.

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

(선택 사항) 지정된 EMQX 구성 파일의 기본값을 추가하거나 재정의하도록 이 옵션을 구성합니다. 구성 파일 및 형식에 대한 자세한 내용은 EMQX 4.0 설명서의 구성을 참조하십시오. 지정한 값은 구성 파일에 추가됩니다.

다음 예제에서는 `etc/emqx.conf` 파일을 업데이트합니다.

```
"mergeConfigurationFiles": {
    "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"
},
```

EMQX에서 지원하는 구성 파일 외에도 Greengrass는 라는 EMQX용 Greengrass 인증 플러그인을 구성하는 파일을 지원합니다. `etc/plugins/aws_greengrass_emqx_auth.conf` `auth_mode`use_greengrass_managed_certificates지원되는 옵션에는 및 두 가지가 있습니다. 다른 인증 제공업체를 사용하려면 `auth_mode` 옵션을 다음 중 하나로 설정합니다.

- `enabled`— (기본값) Greengrass 인증 및 권한 부여 공급자를 사용합니다.
- `bypass_on_failure`— Greengrass 인증 공급자를 사용하고, Greengrass가 인증 또는 권한 부여를 거부하는 경우 EMQX 제공자 체인에 남아 있는 인증 공급자를 사용하십시오.
- `bypass`— Greengrass 공급자가 비활성화되었습니다. 그러면 EMQX 공급자 체인이 인증 및 권한 부여를 처리합니다.

인 use_greengrass_managed_certificates 경우 이 옵션은 Greengrass가 브로커 TLS 인증서를 관리한다는 것을 나타냅니다. true false인 경우 다른 소스를 통해 인증서를 제공했음을 나타냅니다.

다음 예제는 etc/plugins/aws_greengrass_emqx_auth.conf 구성 파일의 기본값을 업데이트합니다.

```
"mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
  },
```

> **Note**
>
> aws.greengrass.clientdevices.mqtt.EMQX보안에 민감한 옵션을 구성할 수 있습니다. 여기에는 TLS 설정, 인증 및 권한 부여 공급자가 포함됩니다. 권장 구성은 상호 TLS 인증 및 Greengrass 클라이언트 장치 인증 공급자를 사용하는 기본 구성입니다.

replaceConfigurationFiles

(선택 사항) 지정된 EMQX 구성 파일을 대체하도록 이 옵션을 구성합니다. 지정한 값은 기존 구성 파일 전체를 대체합니다. 이 섹션에서는 etc/emqx.conf 파일을 지정할 수 없습니다. 를 사용하여 mergeConfigurationFile 수정해야 etc/emqx.conf 합니다.

Example 예: 구성 병합 업데이트

다음 예제 컨피그레이션은 포트 443에서 MQTT 브로커를 작동하도록 지정합니다.

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
  "startupTimeoutSeconds": "90",
```

```
  "ipcTimeoutSeconds": "5"
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -
  Tail 10 -Wait
  ```

라이선스

Windows 운영 체제에서 이 소프트웨어에는 Microsoft 소프트웨어 라이선스 약관에 따라 배포된 코드가 포함되어 있습니다. - Microsoft Visual Studio 커뮤니티 2022. 이 소프트웨어를 다운로드하면 해당 코드의 라이선스 약관에 동의하는 것으로 간주됩니다.

이 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

v2.x

버전	변경
2.0.0	이 버전의 MQTT 5 브로커 (EMQX) 에는 버전 1.x와 다른 구성 매개 변수가 필요합니다. 버전 1.x에 기본 구성이 아닌 구성을 사용하는 경우 2.x용 구성 요소 구성을 업데이트해야 합니다. 자세한 설명은 구성 섹션을 참조하세요. 새로운 기능 • MQTT 브로커를 EMQX 5.1.1로 업그레이드합니다. • 구성 요소를 다시 시작하지 않고도 브로커 구성을 변경할 수 있습니다. 업데이트 • `emqx,mergeConfigurationFiles` , `emqxConfig` 구성 필드를 대체하는 새 `replaceConfigurationFiles` 구성 필드를 추가합니다.

v1.x

버전	변경
1.2.3	버그 수정 및 개선 • 클라이언트의 연결을 끊었다가 다시 인증하여 이전에 인증한 후 클라이언트가 EMQX와 상호 작용할 수 없었던 문제를 수정합니다.
1.2.2	클라이언트 장치 인증 버전 2.4.0 릴리스용으로 버전이 업데이트되었습니다.
1.2.1	버그 수정 및 개선 • Visual C++ 재배포 가능 항목이 아직 없는 경우 Windows에서 구성 요소가 시작되지 않는 문제를 수정합니다. • EMQX를 버전 4.4.14로 업데이트합니다.
1.2.0	인증서 체인에 대한 지원을 추가합니다.

버전	변경
1.1.0	새로운 기능 • 브로커 옵션 및 플러그인을 포함한 EMQX 구성에 대한 지원을 추가합니다. 버그 수정 및 개선 • EMQX를 버전 4.4.9로 업데이트합니다.
1.0.1	TLS 핸드셰이크 중에 일부 MQTT 클라이언트가 연결에 실패하는 문제를 수정합니다.
1.0.0	초기 버전

뉴클리어스 텔레메트리 이미터

핵 원격 측정 이미터 구성 요소 (`aws.greengrass.telemetry.NucleusEmitter`) 는 시스템 상태 원격 측정 데이터를 수집하여 로컬 주제 및 MQTT 주제에 지속적으로 게시합니다. AWS IoT Core 이 구성 요소를 사용하면 Greengrass 코어 장치에서 실시간 시스템 원격 분석을 수집할 수 있습니다. Amazon에 시스템 원격 측정 데이터를 게시하는 Greengrass 원격 분석 에이전트에 대한 자세한 내용은 을 참조하십시오. EventBridge AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집

기본적으로 nucleus 텔레메트리 이미터 구성 요소는 60초마다 다음 로컬 게시/구독 주제에 텔레메트리 데이터를 게시합니다.

```
$local/greengrass/telemetry
```

핵 원격 측정 이미터 구성 요소는 기본적으로 MQTT 주제에 게시하지 않습니다. AWS IoT Core 배포 시 AWS IoT Core MQTT 주제에 게시하도록 이 구성 요소를 구성할 수 있습니다. MQTT 주제를 사용하여 에 데이터를 게시하는 경우 요금이 AWS 클라우드 부과됩니다. AWS IoT Core

AWS IoT GreengrassInfluxDB 및 Grafana를 사용하여 코어 장치에서 로컬로 원격 측정 데이터를 분석하고 시각화하는 데 도움이 되는 여러 커뮤니티 구성 요소를 제공합니다. 이러한 구성 요소는 핵 방사체 구성 요소의 원격 측정 데이터를 사용합니다. 자세한 내용은 InfluxDB 게시자 구성 요소에 대한 README를 참조하십시오.

주제

- 버전
- 유형
- 운영 체제
- 의존성
- 구성
- 출력 데이터
- 사용량
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.0.x

유형

이 컴포넌트는 플러그인 컴포넌트 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

1.0.8

다음 표에는 이 구성 요소의 버전 1.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.13.0	하드

1.0.7

다음 표에는 이 구성 요소의 버전 1.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.12.0	하드

1.0.6

다음 표에는 이 구성 요소의 버전 1.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.11.0	하드

1.0.5

다음 표에는 이 구성 요소의 버전 1.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.10.0	하드

1.0.4

다음 표에는 이 구성 요소의 버전 1.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.9.0	하드

1.0.3

다음 표에는 이 구성 요소의 버전 1.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.8.0	하드

1.0.2

다음 표에는 이 구성 요소의 버전 1.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.7.0	하드

1.0.1

다음 표에는 이 구성 요소의 버전 1.0.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.6.0	하드

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.4.0 <2.5.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`pubSubPublish`

(선택 사항) 원격 분석 데이터를 `$local/greengrass/telemetry` 주제에 게시할지 여부를 정의합니다. 지원되는 값은 `true` 및 `false`입니다.

기본값: `true`

`mqttTopic`

(선택 사항) 이 구성 요소가 원격 분석 데이터를 게시하는 AWS IoT Core MQTT 주제입니다.

이 값을 원격 분석 데이터를 AWS IoT Core 게시하려는 MQTT 주제로 설정합니다. 이 값이 비어 있는 경우 Nucleus Emitter는 원격 측정 데이터를 에 게시하지 않습니다. AWS 클라우드

> **Note**
>
> MQTT 주제를 사용하여 에 데이터를 게시하는 경우 요금이 AWS 클라우드 부과됩니다. AWS IoT Core

기본값: `""`

`telemetryPublishIntervalMs`

(선택 사항) 구성 요소가 원격 측정 데이터를 게시하는 데 걸리는 시간 (밀리초). 이 값을 지원되는 최소값보다 낮게 설정하면 구성 요소가 최소값을 대신 사용합니다.

> **Note**
>
> 게시 간격이 짧을수록 코어 장치의 CPU 사용량이 늘어납니다. 기본 게시 간격으로 시작하여 기기의 CPU 사용량에 따라 조정하는 것이 좋습니다.

최소: `500`

기본값: `60000`

Example 예: 구성 병합 업데이트

다음 예제는 5초마다 원격 분석 데이터를 `$local/greengrass/telemetry` 주제 및 `greengrass/myTelemetry` AWS IoT Core MQTT 주제에 게시할 수 있는 샘플 구성 병합 업데이트를 보여줍니다.

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

출력 데이터

이 구성 요소는 다음 주제에 대해 원격 분석 메트릭을 JSON 배열로 게시합니다.

지역 주제: `$local/greengrass/telemetry`

선택적으로 원격 분석 메트릭을 AWS IoT Core MQTT 주제에 게시하도록 선택할 수도 있습니다. 주제에 대한 자세한 내용은 개발자 안내서의 MQTT 주제를 참조하십시오. AWS IoT Core

Example 예시 데이터

```
[
  {
    "A": "Average",
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
```

```
    "A": "Count",
    "N": "TotalNumberOfFDs",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Count",
    "V": 7316
  },
  {
    "A": "Count",
    "N": "SystemMemUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Megabytes",
    "V": 10098
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStarting",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsInstalled",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStateless",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStopping",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
```

```
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsBroken",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsRunning",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 7
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsErrored",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsNew",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsFinished",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 2
  }
```

```
]
```

출력 배열에는 다음과 같은 속성을 가진 메트릭 목록이 포함됩니다.

A

지표의 집계 유형입니다.

CpuUsage지표의 경우 이 속성이 로 설정된 이유는 지표의 게시된 값이 마지막 게시 이벤트 이후 평균 CPU 사용량이기 Average 때문입니다.

다른 모든 지표의 경우 nucleus emitter는 지표 값을 집계하지 않으며 이 속성은 로 설정됩니다. Count

N

지표의 이름.

NS

메트릭 네임스페이스.

TS

데이터가 수집된 시점의 타임스탬프.

U

지표 값의 단위입니다.

V

지표 값.

핵 방사체는 다음과 같은 지표를 게시합니다.

명칭	설명	
시스템		
SystemMemUsage	운영 체제를 포함하여 Greengrass 코어 디바이스의 모든 애플리케이션에서 현재 사용 중인 메모리의 양입니다.	

명칭	설명	
CpuUsage	운영 체제를 포함하여 Greengrass 코어 디바이스의 모든 애플리케이션에서 현재 사용 중인 CPU의 양입니다.	
TotalNumberOfFDs	Greengrass 코어 디바이스의 운영 체제에 저장된 파일 디스크립터 수입니다. 하나의 파일 디스크립터는 열려 있는 파일 하나를 고유하게 식별합니다.	
그린그래스 핵		
NumberOfComponentsRunning	Greengrass 코어 디바이스에서 실행 중인 구성 요소 수입니다.	
NumberOfComponentsErrored	Greengrass 코어 디바이스에서 오류 상태에 있는 구성 요소의 수입니다.	
NumberOfComponentsInstalled	Greengrass 코어 디바이스에 설치된 구성 요소 수입니다.	
NumberOfComponentsStarting	Greengrass 코어 디바이스에서 시작되는 구성 요소 수입니다.	
NumberOfComponentsNew	Greengrass 코어 디바이스에 새로 추가된 구성 요소의 수.	
NumberOfComponentsStopping	Greengrass 코어 디바이스에서 중지되는 구성 요소의 수입니다.	

명칭	설명	
NumberOfComponents Finished	Greengrass 코어 디바이스에서 완성된 구성 요소의 수입니다.	
NumberOfComponents Broken	Greengrass 코어 디바이스에서 고장난 구성 요소의 수입니다.	
NumberOfComponents Stateless	Greengrass 코어 디바이스에서 스테이트리스 상태인 구성 요소의 수.	

사용량

시스템 상태 원격 측정 데이터를 사용하려면 Nucleus Emitter가 원격 측정 데이터를 게시하는 주제를 구독하고 필요에 따라 해당 데이터에 반응하는 사용자 지정 구성 요소를 만들 수 있습니다. Nucleus emitter 구성 요소는 원격 측정 데이터를 로컬 주제에 게시하는 옵션을 제공하므로 해당 주제를 구독하고 게시된 데이터를 사용하여 코어 장치에서 로컬로 작업할 수 있습니다. 그러면 코어 기기는 클라우드와의 연결이 제한되더라도 텔레메트리 데이터에 반응할 수 있습니다.

예를 들어, `$local/greengrass/telemetry` 주제에 대해 원격 분석 데이터를 수신하는 구성 요소를 구성하고 이 데이터를 스트림 관리자 구성 요소로 전송하여 데이터를 로 스트리밍할 수 있습니다. AWS 클라우드 이러한 구성 요소를 만드는 방법에 대한 자세한 내용은 및 을 참조하십시오로컬 메시지 게시/구독. 스트림 관리자를 사용하는 사용자 지정 구성 요소 만들기

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.0.8	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.7	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.6	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.5	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.4	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.3	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.2	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.1	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
1.0.0	초기 버전

PKCS #11 제공업체

PKCS #11 공급자 구성 요소 (`aws.greengrass.crypto.Pkcs11Provider`) 를 사용하면 PKCS #11 인터페이스를 통해 하드웨어 보안 모듈 (HSM) 을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 이 구성 요소를 사용하면 인증서 및 개인 키 파일을 안전하게 저장하여 소프트웨어에서 노출되거나 중복되지 않도록 할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요.

HSM에 인증서와 개인 키를 저장하는 Greengrass 코어 디바이스를 프로비저닝하려면 Core 소프트웨어를 설치할 때 이 구성 요소를 프로비저닝 플러그인으로 지정해야 합니다. AWS IoT Greengrass 자세한 설명은 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

AWS IoT Greengrass이 구성 요소를 JAR 파일로 제공하여 설치 중에 프로비저닝 플러그인으로 지정할 수 있도록 다운로드할 수 있습니다. 구성 요소 JAR 파일의 최신 버전을 다음 URL 로 다운로드할 수 있습니다. https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- PKCS #1 v1.5 서명 체계와 RSA-2048 키 크기 (또는 그 이상) 또는 ECC 키가 있는 RSA 키를 지원하는 하드웨어 보안 모듈입니다.

  > **Note**
  >
  > ECC 키가 있는 하드웨어 보안 모듈을 사용하려면 Greengrass nucleus v2.5.6 이상을 사용해야 합니다.
  > 하드웨어 보안 모듈과 시크릿 관리자를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

- AWS IoT GreengrassCore 소프트웨어가 런타임 시 (libdl 사용) 로드하여 PKCS #11 함수를 호출할 수 있는 PKCS #11 공급자 라이브러리입니다. PKCS #11 공급자 라이브러리는 다음과 같은 PKCS #11 API 작업을 구현해야 합니다.
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`

  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- 하드웨어 모듈은 PKCS#11 사양에 정의된 것처럼 슬롯 레이블로 확인할 수 있어야 합니다.
- HSM에서 개체 ID를 지원하는 경우 개인 키와 인증서를 HSM의 동일한 슬롯에 저장하고 동일한 개체 레이블과 개체 ID를 사용해야 합니다.
- 인증서와 개인 키는 개체 레이블로 확인할 수 있어야 합니다.
- 개인 키에는 다음과 같은 권한이 있어야 합니다.
  - `sign`
  - `decrypt`
- (선택 사항) Secret Manager 구성 요소를 사용하려면 버전 2.1.0 이상을 사용해야 하며, 개인 키에는 다음과 같은 권한이 있어야 합니다.

  - `unwrap`
  - `wrap`
- (선택 사항) TPM2 라이브러리를 사용하고 Greengrass 코어를 서비스로 실행하는 경우 PKCS #11 스토어 위치가 포함된 환경 변수를 제공해야 합니다. 다음 예제는 필수 환경 변수가 포함된 systemd 서비스 파일입니다.

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.13.0	소프트

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.12.0	소프트

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.11.0	소프트

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.10.0	소프트

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.9.0	소프트

2.0.2

다음 표에는 이 구성 요소의 버전 2.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.8.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.7.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.3 <2.6.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`name`

PKCS #11 구성의 이름.

`library`

AWS IoT Greengrass코어 소프트웨어가 libdl로 로드할 수 있는 PKCS #11 구현 라이브러리의 절대 파일 경로입니다.

`slot`

개인 키와 디바이스 인증서가 포함된 슬롯의 ID입니다. 이 값은 슬롯 인덱스 또는 슬롯 레이블과 다릅니다.

`userPin`

슬롯에 액세스하는 데 사용할 사용자 PIN.

Example 예: 구성 병합 업데이트

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.0.7	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.3	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.2	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.1	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.0	초기 버전

시크릿 매니저

시크릿 매니저 컴포넌트 (`aws.greengrass.SecretManager`) 는 Greengrass 코어 디바이스의 AWS Secrets Manager 시크릿을 배포합니다. 이 구성 요소를 사용하면 Greengrass 코어 장치의 사용자 지정 구성 요소에서 암호와 같은 자격 증명을 안전하게 사용할 수 있습니다. Secrets Manager에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager란 무엇입니까?를 참조하세요.

사용자 정의 Greengrass 구성 요소에 있는 이 구성 요소의 암호에 액세스하려면 에서 GetSecretValue작업을 사용하십시오. AWS IoT Device SDK 자세한 내용은 AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core 및 비밀 값 검색 섹션을 참조하세요.

이 구성 요소는 코어 장치의 비밀을 암호화하여 자격 증명과 암호를 사용해야 할 때까지 안전하게 유지합니다. 코어 디바이스의 개인 키를 사용하여 비밀을 암호화하고 해독합니다.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 장치 역할은 다음 예시 IAM 정책에 나와 있는 것처럼 `secretsmanager:GetSecretValue` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:123456789012:secret:MySecret"
      ]
    }
  ]
}
```

> **Note**
>
> 고객 관리 AWS Key Management Service 키를 사용하여 비밀을 암호화하는 경우 장치 역할도 해당 작업을 허용해야 합니다. `kms:Decrypt`

Secrets Manager의 IAM 정책에 대한 자세한 내용은 AWS Secrets Manager사용 설명서의 다음을 참조하십시오.

- 에 대한 인증 및 액세스 제어 AWS Secrets Manager
- IAM 정책 또는 비밀 정책에서 사용할 수 있는 작업, 리소스 및 컨텍스트 키 AWS Secrets Manager

- 사용자 지정 구성 요소는 이 구성 요소와 함께 저장한 암호를 가져올 수 `aws.greengrass#GetSecretValue` 있는 권한 부여 정책을 정의해야 합니다. 이 권한 부여 정책에서는 구성 요소의 액세스를 특정 비밀에 제한할 수 있습니다. 자세한 내용은 시크릿 매니저 IPC 인증을 참조하십시오.
- (선택 사항) 코어 디바이스의 개인 키와 인증서를 하드웨어 보안 모듈 (HSM) 에 저장하는 경우 HSM 은 RSA 키를 지원해야 하고, 개인 키에는 권한이 있어야 하며, 공개 키에는 `unwrap` 권한이 있어야 합니다. `wrap`

엔드포인트와 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`secretsmanager.` *`region`*`.amazonaws.com`	443	예	암호를 코어 디바이스에 다운로드합니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.13.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.12.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.11.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.10.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.9.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.8.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.7.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.6.0	소프트

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

2.0.4 and 2.0.5

다음 표에는 이 구성 요소의 버전 2.0.4 및 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`cloudSecrets`

코어 디바이스에 배포할 Secrets Manager 암호 목록입니다. 레이블을 지정하여 배포할 각 시크릿의 버전을 정의할 수 있습니다. 버전을 지정하지 않으면 이 구성 요소는 스테이징 `AWSCURRENT` 레이블이 첨부된 버전을 배포합니다. 자세한 내용은 사용 설명서의 스테이징 레이블을 참조하십시오. AWS Secrets Manager

시크릿 매니저 컴포넌트는 시크릿을 로컬에 캐시합니다. Secrets Manager에서 보안 값이 변경되는 경우 이 구성 요소는 새 값을 자동으로 검색하지 않습니다. 로컬 복사본을 업데이트하려면 암호에 새 레이블을 지정하고 새 레이블로 식별된 암호를 검색하도록 이 구성 요소를 구성하십시오.

각 개체에는 다음 정보가 들어 있습니다.

`arn`

배포할 비밀의 ARN입니다. 시크릿의 ARN은 전체 ARN 또는 부분 ARN일 수 있습니다. 부분 ARN보다는 전체 ARN을 지정하는 것이 좋습니다. 자세한 내용은 부분 ARN에서 암호 찾기를 참조하십시오. 다음은 전체 ARN과 부분 ARN의 예입니다.

- 풀 ARN: `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-abcdef`
- 부분 ARN: `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName`

`labels`

(선택 사항) 코어 디바이스에 배포할 시크릿 버전을 식별하기 위한 레이블 목록.

각 레이블은 문자열이어야 합니다.

Example 예: 구성 병합 업데이트

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    }
  ]
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.7	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	버그 수정 및 개선 시크릿 매니저가 배포되고 Greengrass nucleus가 다시 시작될 때 캐시된 비밀이 제거되던 문제를 수정합니다.

버전	변경
	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • 하드웨어 보안 통합에 대한 지원을 추가합니다. Secret Manager 구성 요소는 하드웨어 보안 모듈 (HSM) 에 저장한 개인 키를 사용하여 암호를 암호화하고 해독할 수 있습니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요. 버그 수정 및 개선 • Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.9	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.8	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.5	개선 사항 • AWS중국 지역 및 지역에 대한 지원을 추가합니다. AWS GovCloud (US)
2.0.4	초기 버전

보안 터널링

`aws.greengrass.SecureTunneling`구성 요소를 사용하면 제한된 방화벽 뒤에 있는 Greengrass 코어 장치와 안전한 양방향 통신을 설정할 수 있습니다.

예를 들어 들어오는 모든 연결을 차단하는 방화벽 뒤에 Greengrass 코어 디바이스가 있다고 가정해 보겠습니다. 보안 터널링은 MQTT를 사용하여 장치에 액세스 토큰을 전송한 다음 방화벽을 통해 장치에 SSH 연결을 설정하는 WebSockets 데 사용합니다. 이 AWS IoT 관리형 터널을 사용하면 디바이스에 필요한 SSH 연결을 열 수 있습니다. AWS IoT보안 터널링을 사용하여 원격 장치에 연결하는 방법에 대한 자세한 내용은 개발자 안내서의 AWS IoT보안 터널링을 참조하십시오. AWS IoT

이 구성 요소는 해당 `$aws/things/greengrass-core-device/tunnels/notify` 주제에 대한 AWS IoT Core MQTT 메시지 브로커를 구독하여 보안 터널링 알림을 수신합니다.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- 라이선스
- 사용량
- 다음 사항도 참조하십시오.
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

아키텍처:

- Armv71
- ARMv8(AArch64)
- x86_64

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 보안 터널링 구성 요소에 사용할 수 있는 최소 32MB의 디스크 공간. 이 요구 사항에는 Greengrass 코어 소프트웨어 또는 동일한 장치에서 실행되는 기타 구성 요소가 포함되지 않습니다.
- 보안 터널링 구성 요소에 사용할 수 있는 최소 16MB RAM이 있습니다. 이 요구 사항에는 Greengrass 코어 소프트웨어 또는 동일한 장치에서 실행되는 기타 구성 요소가 포함되지 않습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.
- 보안 터널링 구성 요소 버전 1.0.12 이상을 사용하려면 GNU C 라이브러리 (glibc) 버전 2.25 이상, Linux 커널 3.2 이상이 필요합니다. 장기 지원 종료일이 지난 운영 체제 및 라이브러리 버전은 지원되지 않습니다. 장기간 지원되는 운영 체제와 라이브러리를 사용해야 합니다.
- 운영 체제와 Java 런타임을 모두 64비트로 설치해야 합니다.
- Python 3.5 이상이 Greengrass 코어 디바이스에 설치되고 PATH 환경 변수에 추가되었습니다.
- `libcrypto.so.1.1`Greengrass 코어 디바이스에 설치되고 PATH 환경 변수에 추가되었습니다.
- Greengrass 코어 디바이스의 포트 443에서 아웃바운드 트래픽을 엽니다.
- Greengrass 코어 장치와 통신하는 데 사용할 통신 서비스에 대한 지원을 켜십시오. 예를 들어 장치에 대한 SSH 연결을 열려면 해당 장치에서 SSH를 켜야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`data.tunneling.iot. region.amazonaws.com`	443	예	보안 터널을 구축하십시오.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

1.0.18

다음 표에는 이 구성 요소의 버전 1.0.18에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트

1.0.16 – 1.0.17

다음 표에는 이 구성 요소의 버전 1.0.16~1.0.17에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트

1.0.14 – 1.0.15

다음 표에는 이 구성 요소의 버전 1.0.14~1.0.15에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트

1.0.11 – 1.0.13

다음 표에는 이 구성 요소의 버전 1.0.11 — 1.0.13에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트

1.0.10

다음 표에는 이 구성 요소의 버전 1.0.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트

1.0.9

다음 표에는 이 구성 요소의 버전 1.0.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트

1.0.8

다음 표에는 이 구성 요소의 버전 1.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트

1.0.5 - 1.0.7

다음 표에는 이 구성 요소의 버전 1.0.5~1.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

1.0.4

다음 표에는 이 구성 요소의 버전 1.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트

1.0.3

다음 표에는 이 구성 요소의 버전 1.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트

1.0.2

다음 표에는 이 구성 요소의 버전 1.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트

1.0.1

다음 표에는 이 구성 요소의 버전 1.0.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`OS_DIST_INFO`

(선택 사항) 코어 디바이스의 운영 체제. 기본적으로 구성 요소는 코어 장치에서 실행 중인 운영 체제를 자동으로 식별하려고 시도합니다. 구성 요소가 기본값으로 시작하지 않는 경우 이 값을 사용하여 운영 체제를 지정하십시오. 이 구성 요소에 지원되는 운영 체제 목록은 을 참조하십시오디바이스 요구 사항.

이 값은`auto`,, `ubuntuamzn2`, 중 하나일 수 `raspberrypi` 있습니다.

기본값: `auto`

`accessControl`

(선택 사항) 구성 요소가 보안 터널링 알림 주제를 구독하도록 허용하는 권한 부여 정책이 포함된 객체입니다.

> **Note**
>
> 배포가 사물 그룹을 대상으로 하는 경우 이 구성 매개 변수를 수정하지 마십시오. 배포가 개별 코어 디바이스를 대상으로 하고 디바이스 주제로만 구독을 제한하려면 코어 디바이스의

사물 이름을 지정하십시오. 기기 권한 부여 정책의 `resources` 값에서 MQTT 주제 와일드카드를 기기의 사물 이름으로 바꾸십시오.

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Example 예: 구성 병합 업데이트

다음 예제 구성은 이 구성 요소가 Ubuntu를 실행하는 코어 기기에서 보안 터널을 열 수 있도록 지정합니다. **MyGreengrassCore**

```
{
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
  ```

라이선스

이 구성 요소에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- AWS IoT디바이스 클라이언트/Apache 라이선스 2.0
- AWS IoT Device SDK for Java/Apache 라이선스 2.0
- gson/아파치 라이선스 2.0
- log4j /아파치 라이선스 2.0
- slf4j /아파치 라이선스 2.0

사용량

기기에서 보안 터널링 구성 요소를 사용하려면 다음과 같이 하세요.

1. 보안 터널링 구성 요소를 장치에 배포하십시오.
2. AWS IoT 콘솔을 엽니다. 왼쪽 메뉴에서 원격 작업을 선택한 다음 보안 터널을 선택합니다.
3. Greengrass 디바이스에 터널을 만드세요.
4. 소스 액세스 토큰을 다운로드하세요.
5. 로컬 프록시를 소스 액세스 토큰과 함께 사용하여 목적지에 연결합니다. 자세한 내용은 AWS IoT 개발자 안내서의 로컬 프록시 사용 방법을 참조하십시오.

다음 사항도 참조하십시오.

- AWS IoT개발자 안내서의 AWS IoT보안 터널링
- 개발자 안내서의 로컬 프록시 사용 방법 AWS IoT

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.0.18	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.17	버그 수정 및 개선 • 사용자가 터널을 만들지 못하도록 차단하던 스레드 정리 문제를 수정했습니다. 이제 이 구성 요소는 CloseTunnel 신호를 수신한 후 또는 터널이 12시간 후에 만료된 경우 스레드를 정리합니다.
1.0.16	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.15	버그 수정 및 개선 • 장치에 홈 디렉터리가 없는 사용자의 시작 문제를 수정합니다. 이제 섀도우 문서용 디렉터리를 만들지 않아도 보안 터널링 구성 요소가 시작됩니다.
1.0.14	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.13	버그 수정 및 개선 • 분리된 클라이언트 프로세스로 인해 둘 이상의 터널이 디바이스를 타겟팅하지 못하는 문제를 수정합니다.
1.0.12	버그 수정 및 개선 • 라즈베리 파이 OS에서 실행할 때 x86_64 (AMD64) 및 ARMv8 (Aarch64)에 대한 지원을 추가합니다.

버전	변경
1.0.11	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.10	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.9	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.8	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.7	버그 수정 및 개선 • SCP를 통해 대용량 파일을 전송할 때 구성 요소 연결이 끊기는 문제를 수정합니다.
1.0.6	이 버전에는 버그 수정이 포함되어 있습니다.
1.0.5	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.4	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.3	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.2	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
1.0.1	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
1.0.0	초기 버전

섀도우 매니저

섀도우 관리자 구성 요소 (`aws.greengrass.ShadowManager`) 는 코어 디바이스에서 로컬 섀도우 서비스를 활성화합니다. 로컬 섀도우 서비스를 사용하면 구성 요소가 프로세스 간 통신을 사용하여 로컬 섀도우와 상호 작용할 수 있습니다. 섀도우 관리자 구성 요소는 로컬 섀도우 문서의 저장을 관리하고 로컬 섀도우 상태와 AWS IoT Device Shadow 서비스의 동기화도 처리합니다.

Greengrass 코어 장치가 그림자와 상호 작용하는 방식에 대한 자세한 내용은 을 참조하십시오. 디바이스 섀도우와 상호작용

주제

- 버전

- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소 () `aws.greengrass.plugin` 입니다. Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 이 구성 요소를 실행합니다. 코어 디바이스에서 이 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다.

이 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- (선택 사항) 섀도우를 AWS IoT Device Shadow 서비스에 동기화하려면 Greengrass 코어 장치 AWS IoT 정책에서 다음과 같은 AWS IoT Core 섀도우 정책 작업을 허용해야 합니다.
  - `iot:GetThingShadow`
  - `iot:UpdateThingShadow`
  - `iot:DeleteThingShadow`

  이러한 AWS IoT Core 정책에 대한 자세한 내용은 AWS IoT 개발자 안내서의 AWS IoT Core 정책 조치를 참조하십시오.

  최소 AWS IoT 정책에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책
- 섀도우 관리자 구성 요소는 VPC에서 실행되도록 지원됩니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.3.5 – 2.3.6

다음 표에는 이 구성 요소의 버전 2.3.5 및 2.3.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.13.0	소프트

2.3.3 and 2.3.4

다음 표에는 이 구성 요소의 버전 2.3.3 및 2.3.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.12.0	소프트

2.3.2

다음 표에는 이 구성 요소의 버전 2.3.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.11.0	소프트

2.3.0 and 2.3.1

다음 표에는 이 구성 요소의 버전 2.3.0 및 2.3.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.5.0 <2.10.0	소프트

2.2.3 and 2.2.4

다음 표에는 이 구성 요소의 버전 2.2.3 및 2.2.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <3.0.0	소프트

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.9.0	소프트

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.8.0	소프트

2.1.1 and 2.2.0

다음 표에는 이 구성 요소의 버전 2.1.1 및 2.2.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.7.0	소프트

2.0.5 - 2.1.0

다음 표에는 이 구성 요소의 버전 2.0.5~2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.6.0	소프트

2.0.3 and 2.0.4

다음 표에는 이 구성 요소의 버전 2.0.3 및 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.5.0	소프트

2.0.1 and 2.0.2

다음 표에는 이 구성 요소의 버전 2.0.1 및 2.0.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.4.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.2.0 <2.3.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

2.3.x

`strategy`

(선택 사항) 이 구성 요소가 코어 장치 간에 AWS IoT Core 섀도우를 동기화하는 데 사용하는 전략입니다.

이 개체에는 다음과 같은 정보가 들어 있습니다.

`type`

(선택 사항) 이 구성 요소가 코어 장치 간에 AWS IoT Core 섀도우를 동기화하는 데 사용하는 전략 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `realTime`— 섀도우 업데이트가 발생할 AWS IoT Core 때마다 섀도우를 동기화합니다.
- `periodic`— `delay` 구성 매개 변수로 지정한 일정한 간격으로 섀도우를 동기화합니다. AWS IoT Core

기본값: `realTime`

`delay`

(선택 사항) `periodic` 동기화 전략을 지정할 때 이 구성 요소가 섀도우를 동기화하는 간격(초) 입니다. AWS IoT Core

> **Note**
>
> `periodic`동기화 전략을 지정하는 경우 이 매개 변수가 필요합니다.

`synchronize`

(선택 사항) 섀도우와 동기화되는 방식을 결정하는 동기화 설정입니다. AWS 클라우드

> **Note**
>
> 섀도우를 와 동기화하려면 이 속성을 사용하여 구성 업데이트를 생성해야 합니다 AWS 클라우드.

이 객체에는 다음과 같은 정보가 들어 있습니다.

`coreThing`

(선택 사항) 동기화할 코어 디바이스 섀도우. 이 개체에는 다음 정보가 들어 있습니다.

`classic`

(선택 사항) 기본적으로 섀도우 관리자는 코어 디바이스의 클래식 섀도우의 로컬 상태를 와 동기화합니다. AWS 클라우드 클래식 디바이스 섀도를 동기화하지 않으려면 `false` 이 설정을 로 설정하십시오.

기본값: `true`

`namedShadows`

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명명된 섀도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니

다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다 AWS IoT Greengrass . 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

`shadowDocumentsMap`

(선택 사항) 동기화할 추가 장치 섀도. 이 구성 매개 변수를 사용하면 섀도우 문서를 더 쉽게 지정할 수 있습니다. `shadowDocuments`개체 대신 이 매개 변수를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocumentsMap`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocuments`

각 개체에는 다음 정보가 들어 있습니다.

`thingName`

이 섀도우 구성에 대한 *`ThingName#`* 섀도우 구성입니다.

`classic`

(선택 사항) 장치의 클래식 장치 섀도를 동기화하지 않으려면 `false` 이 값을 로 설정하십시오. `thingName`

`namedShadows`

동기화하려는 명명된 섀도우 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`shadowDocuments`

(선택 사항) 동기화할 추가 장치 섀도 목록. 대신 `shadowDocumentsMap` 파라미터를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocuments`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocumentsMap`

이 목록의 각 개체에는 다음 정보가 들어 있습니다.

`thingName`

섀도우를 동기화할 장치의 사물 이름.

`classic`

(선택 사항) 디바이스의 클래식 디바이스 섀도를 동기화하지 않으려면 이 옵션을 로 설정하십시오`false`. `thingName`

기본값: `true`

`namedShadows`

(선택 사항) 동기화하려는 이름이 지정된 장치 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`direction`

(선택 사항) 로컬 섀도우 서비스와 로컬 섀도우 서비스 간에 섀도우를 동기화하는 AWS 클라우드 방향입니다. 이 옵션을 구성하여 에 대한 대역폭과 연결을 줄일 수 AWS 클라우드 있습니다. 다음 옵션 중 하나를 선택합니다.

- `betweenDeviceAndCloud`— 로컬 섀도우 서비스와 의 AWS 클라우드 섀도우를 동기화합니다.
- `deviceToCloud`— 로컬 섀도우 서비스의 섀도우 업데이트를 로 보내고 섀도우 업데이트는 무시합니다 AWS 클라우드. AWS 클라우드
- `cloudToDevice`— 에서 섀도우 업데이트를 받고 로컬 섀도우 서비스에서 섀도우 업데이트는 로 보내지 않습니다 AWS 클라우드. AWS 클라우드

기본값: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(선택 사항) 섀도우 서비스 요청의 속도 제한을 결정하는 설정입니다.

이 개체에는 다음 정보가 들어 있습니다.

`maxOutboundSyncUpdatesPerSecond`

(선택 사항) 장치가 전송하는 초당 최대 동기화 요청 수입니다.

기본값: 초당 요청 100개

`maxTotalLocalRequestsRate`

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수입니다.

기본값: 초당 요청 200개

`maxLocalRequestsPerSecondPerThing`

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 로컬 IPC 요청의 최대 수입니다.

기본값: 각 사물에 대해 초당 요청 20개

> **Note**
>
> 이러한 속도 제한 매개변수는 로컬 섀도우 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT Device Shadow 서비스의 초당 최대 요청 수는 사용자에 따라 다릅니다 AWS 리전. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

`shadowDocumentSizeLimitBytes`

(선택 사항) 로컬 섀도우에 대한 각 JSON 상태 문서의 최대 허용 크기입니다.

이 값을 늘리면 구름 그림자에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

기본값: 8192바이트

최대: 30720바이트

Example 예: 구성 병합 업데이트

다음 예제는 섀도우 관리자 구성 요소에 사용 가능한 모든 구성 매개 변수가 포함된 샘플 구성 병합 업데이트를 보여줍니다.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
```

```
    },
    "synchronize":{
       "shadowDocumentsMap":{
          "MyDevice1":{
             "classic":false,
             "namedShadows":[
                "MyShadowA",
                "MyShadowB"
             ]
          },
          "MyDevice2":{
             "classic":true,
             "namedShadows":[]
          }
       },
       "direction":"betweenDeviceAndCloud"
    },
    "rateLimits":{
       "maxOutboundSyncUpdatesPerSecond":100,
       "maxTotalLocalRequestsRate":200,
       "maxLocalRequestsPerSecondPerThing":20
    },
    "shadowDocumentSizeLimitBytes":8192
}
```

2.2.x

`strategy`

(선택 사항) 이 구성 요소가 코어 장치 간에 AWS IoT Core 섀도우를 동기화하는 데 사용하는 전략입니다.

이 개체에는 다음과 같은 정보가 들어 있습니다.

`type`

(선택 사항) 이 구성 요소가 코어 장치 간에 AWS IoT Core 섀도우를 동기화하는 데 사용하는 전략 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `realTime`— 섀도우 업데이트가 발생할 AWS IoT Core 때마다 섀도우를 동기화합니다.
- `periodic`— `delay` 구성 매개 변수로 지정한 일정한 간격으로 섀도우를 동기화합니다. AWS IoT Core

기본값: `realTime`

`delay`

(선택 사항) `periodic` 동기화 전략을 지정할 때 이 구성 요소가 섀도우를 동기화하는 간격(초) 입니다. AWS IoT Core

> **Note**
>
> `periodic`동기화 전략을 지정하는 경우 이 매개 변수가 필요합니다.

`synchronize`

(선택 사항) 섀도우와 동기화되는 방식을 결정하는 동기화 설정입니다. AWS 클라우드

> **Note**
>
> 섀도우를 와 동기화하려면 이 속성을 사용하여 구성 업데이트를 생성해야 합니다 AWS 클라우드.

이 객체에는 다음과 같은 정보가 들어 있습니다.

`coreThing`

(선택 사항) 동기화할 코어 디바이스 섀도우. 이 개체에는 다음 정보가 들어 있습니다.

`classic`

(선택 사항) 기본적으로 섀도우 관리자는 코어 디바이스의 클래식 섀도우의 로컬 상태를 와 동기화합니다. AWS 클라우드 클래식 디바이스 섀도를 동기화하지 않으려면 `false` 이 설정을 로 설정하십시오.

기본값: `true`

`namedShadows`

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명명된 섀도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니

다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다 AWS IoT Greengrass . 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

`shadowDocumentsMap`

(선택 사항) 동기화할 추가 장치 섀도. 이 구성 매개 변수를 사용하면 섀도우 문서를 더 쉽게 지정할 수 있습니다. `shadowDocuments`개체 대신 이 매개 변수를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocumentsMap`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocuments`

각 개체에는 다음 정보가 들어 있습니다.

`thingName`

이 섀도우 구성에 대한 *`ThingName#`* 섀도우 구성입니다.

`classic`

(선택 사항) 장치의 클래식 장치 섀도를 동기화하지 않으려면 `false` 이 값을 로 설정하십시오. `thingName`

`namedShadows`

동기화하려는 명명된 섀도우 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`shadowDocuments`

(선택 사항) 동기화할 추가 장치 섀도 목록. 대신 `shadowDocumentsMap` 파라미터를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocuments`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocumentsMap`

이 목록의 각 개체에는 다음 정보가 들어 있습니다.

`thingName`

섀도우를 동기화할 장치의 사물 이름.

`classic`

(선택 사항) 디바이스의 클래식 디바이스 섀도를 동기화하지 않으려면 이 옵션을 로 설정하십시오`false`. `thingName`

기본값: `true`

`namedShadows`

(선택 사항) 동기화하려는 이름이 지정된 장치 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`direction`

(선택 사항) 로컬 섀도우 서비스와 로컬 섀도우 서비스 간에 섀도우를 동기화하는 AWS 클라우드 방향입니다. 이 옵션을 구성하여 에 대한 대역폭과 연결을 줄일 수 AWS 클라우드 있습니다. 다음 옵션 중 하나를 선택합니다.

- `betweenDeviceAndCloud`— 로컬 섀도우 서비스와 의 AWS 클라우드 섀도우를 동기화합니다.
- `deviceToCloud`— 로컬 섀도우 서비스의 섀도우 업데이트를 로 보내고 섀도우 업데이트는 무시합니다 AWS 클라우드. AWS 클라우드
- `cloudToDevice`— 에서 섀도우 업데이트를 받고 로컬 섀도우 서비스에서 섀도우 업데이트는 로 보내지 않습니다 AWS 클라우드. AWS 클라우드

기본값: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(선택 사항) 섀도우 서비스 요청의 속도 제한을 결정하는 설정입니다.

이 개체에는 다음과 같은 정보가 들어 있습니다.

`maxOutboundSyncUpdatesPerSecond`

(선택 사항) 장치가 전송하는 초당 최대 동기화 요청 수입니다.

기본값: 초당 요청 100개

`maxTotalLocalRequestsRate`

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수입니다.

기본값: 초당 요청 200개

`maxLocalRequestsPerSecondPerThing`

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 로컬 IPC 요청의 최대 수입니다.

기본값: 각 사물에 대해 초당 요청 20개

> **Note**
>
> 이러한 속도 제한 매개변수는 로컬 섀도우 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT Device Shadow 서비스의 초당 최대 요청 수는 사용자에 따라 다릅니다 AWS 리전. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

`shadowDocumentSizeLimitBytes`

(선택 사항) 로컬 섀도우에 대한 각 JSON 상태 문서의 최대 허용 크기입니다.

이 값을 늘리면 구름 그림자에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

기본값: 8192바이트

최대: 30720바이트

Example 예: 구성 병합 업데이트

다음 예제는 섀도우 관리자 구성 요소에 사용 가능한 모든 구성 매개 변수가 포함된 샘플 구성 병합 업데이트를 보여줍니다.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
```

```
    },
    "synchronize":{
       "shadowDocumentsMap":{
          "MyDevice1":{
             "classic":false,
             "namedShadows":[
                "MyShadowA",
                "MyShadowB"
             ]
          },
          "MyDevice2":{
             "classic":true,
             "namedShadows":[]
          }
       },
       "direction":"betweenDeviceAndCloud"
    },
    "rateLimits":{
       "maxOutboundSyncUpdatesPerSecond":100,
       "maxTotalLocalRequestsRate":200,
       "maxLocalRequestsPerSecondPerThing":20
    },
    "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

`strategy`

(선택 사항) 이 구성 요소가 코어 장치 간에 AWS IoT Core 섀도우를 동기화하는 데 사용하는 전략입니다.

이 개체에는 다음과 같은 정보가 들어 있습니다.

`type`

(선택 사항) 이 구성 요소가 코어 장치 간에 AWS IoT Core 섀도우를 동기화하는 데 사용하는 전략 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `realTime`— 섀도우 업데이트가 발생할 AWS IoT Core 때마다 섀도우를 동기화합니다.
- `periodic`— `delay` 구성 매개 변수로 지정한 일정한 간격으로 섀도우를 동기화합니다. AWS IoT Core

기본값: `realTime`

`delay`

(선택 사항) `periodic` 동기화 전략을 지정할 때 이 구성 요소가 섀도우를 동기화하는 간격(초) 입니다. AWS IoT Core

> **Note**
>
> `periodic`동기화 전략을 지정하는 경우 이 매개 변수가 필요합니다.

`synchronize`

(선택 사항) 섀도우와 동기화되는 방식을 결정하는 동기화 설정입니다. AWS 클라우드

> **Note**
>
> 섀도우를 와 동기화하려면 이 속성을 사용하여 구성 업데이트를 생성해야 합니다 AWS 클라우드.

이 객체에는 다음과 같은 정보가 들어 있습니다.

`coreThing`

(선택 사항) 동기화할 코어 디바이스 섀도우. 이 개체에는 다음 정보가 들어 있습니다.

`classic`

(선택 사항) 기본적으로 섀도우 관리자는 코어 디바이스의 클래식 섀도우의 로컬 상태를 와 동기화합니다. AWS 클라우드 클래식 디바이스 섀도를 동기화하지 않으려면 `false` 이 설정을 로 설정하십시오.

기본값: `true`

`namedShadows`

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명명된 섀도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니

다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다 AWS IoT Greengrass . 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

`shadowDocumentsMap`

(선택 사항) 동기화할 추가 장치 섀도. 이 구성 매개 변수를 사용하면 섀도우 문서를 더 쉽게 지정할 수 있습니다. `shadowDocuments`개체 대신 이 매개 변수를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocumentsMap`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocuments`

각 개체에는 다음 정보가 들어 있습니다.

`thingName`

이 섀도우 구성에 대한 *`ThingName#`* 섀도우 구성입니다.

`classic`

(선택 사항) 장치의 클래식 장치 섀도를 동기화하지 않으려면 `false` 이 값을 로 설정하십시오. `thingName`

`namedShadows`

동기화하려는 명명된 섀도우 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`shadowDocuments`

(선택 사항) 동기화할 추가 장치 섀도 목록. 대신 `shadowDocumentsMap` 파라미터를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocuments`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocumentsMap`

이 목록의 각 개체에는 다음 정보가 들어 있습니다.

`thingName`

섀도우를 동기화할 장치의 사물 이름.

`classic`

(선택 사항) 디바이스의 클래식 디바이스 섀도를 동기화하지 않으려면 이 옵션을 로 설정하십시오`false`. `thingName`

기본값: `true`

`namedShadows`

(선택 사항) 동기화하려는 이름이 지정된 장치 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`rateLimits`

(선택 사항) 섀도우 서비스 요청의 속도 제한을 결정하는 설정입니다.

이 개체에는 다음과 같은 정보가 들어 있습니다.

`maxOutboundSyncUpdatesPerSecond`

(선택 사항) 장치가 전송하는 초당 최대 동기화 요청 수입니다.

기본값: 초당 요청 100개

`maxTotalLocalRequestsRate`

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수입니다.

기본값: 초당 요청 200개

`maxLocalRequestsPerSecondPerThing`

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 로컬 IPC 요청의 최대 수입니다.

기본값: 각 사물에 대해 초당 요청 20개

> **Note**
>
> 이러한 속도 제한 매개변수는 로컬 섀도우 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT Device Shadow 서비스의 초당 최대 요청 수는 사용자에 따라 다릅니다

AWS 리전. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

`shadowDocumentSizeLimitBytes`

(선택 사항) 로컬 섀도우에 대한 각 JSON 상태 문서의 최대 허용 크기입니다.

이 값을 늘리면 구름 그림자에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

기본값: 8192바이트

최대: 30720바이트

Example 예: 구성 병합 업데이트

다음 예제는 섀도우 관리자 구성 요소에 사용 가능한 모든 구성 매개 변수가 포함된 샘플 구성 병합 업데이트를 보여줍니다.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
```

```
    "rateLimits":{
       "maxOutboundSyncUpdatesPerSecond":100,
       "maxTotalLocalRequestsRate":200,
       "maxLocalRequestsPerSecondPerThing":20
    },
    "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

`synchronize`

(선택 사항) 섀도우와 동기화되는 방식을 결정하는 동기화 설정입니다. AWS 클라우드

> **Note**
>
> 섀도우를 와 동기화하려면 이 속성을 사용하여 구성 업데이트를 생성해야 합니다 AWS 클라우드.

이 객체에는 다음과 같은 정보가 들어 있습니다.

`coreThing`

(선택 사항) 동기화할 코어 디바이스 섀도우. 이 개체에는 다음 정보가 들어 있습니다.

`classic`

(선택 사항) 기본적으로 섀도우 관리자는 코어 디바이스의 클래식 섀도우의 로컬 상태를 와 동기화합니다. AWS 클라우드 클래식 디바이스 섀도를 동기화하지 않으려면 `false` 이 설정을 로 설정하십시오.

기본값: `true`

`namedShadows`

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명명된 섀도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니

다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다 AWS IoT Greengrass . 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

`shadowDocumentsMap`

(선택 사항) 동기화할 추가 장치 섀도. 이 구성 매개 변수를 사용하면 섀도우 문서를 더 쉽게 지정할 수 있습니다. `shadowDocuments`개체 대신 이 매개 변수를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocumentsMap`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocuments`

각 개체에는 다음 정보가 들어 있습니다.

`thingName`

이 섀도우 구성에 대한 *`ThingName#`* 섀도우 구성입니다.

`classic`

(선택 사항) 장치의 클래식 장치 섀도를 동기화하지 않으려면 `false` 이 값을 로 설정하십시오. `thingName`

`namedShadows`

동기화하려는 명명된 섀도우 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`shadowDocuments`

(선택 사항) 동기화할 추가 장치 섀도 목록. 대신 `shadowDocumentsMap` 파라미터를 사용하는 것이 좋습니다.

> **Note**
>
> `shadowDocuments`개체를 지정하는 경우 개체를 지정해서는 안 됩니다. `shadowDocumentsMap`

이 목록의 각 개체에는 다음 정보가 들어 있습니다.

`thingName`

섀도우를 동기화할 장치의 사물 이름.

`classic`

(선택 사항) 디바이스의 클래식 디바이스 섀도를 동기화하지 않으려면 이 옵션을 로 설정하십시오`false`. `thingName`

기본값: `true`

`namedShadows`

(선택 사항) 동기화하려는 이름이 지정된 장치 섀도 목록. 섀도우의 정확한 이름을 지정해야 합니다.

`rateLimits`

(선택 사항) 섀도우 서비스 요청의 속도 제한을 결정하는 설정입니다.

이 개체에는 다음과 같은 정보가 들어 있습니다.

`maxOutboundSyncUpdatesPerSecond`

(선택 사항) 장치가 전송하는 초당 최대 동기화 요청 수입니다.

기본값: 초당 요청 100개

`maxTotalLocalRequestsRate`

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수입니다.

기본값: 초당 요청 200개

`maxLocalRequestsPerSecondPerThing`

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 로컬 IPC 요청의 최대 수입니다.

기본값: 각 사물에 대해 초당 요청 20개

> **Note**
>
> 이러한 속도 제한 매개변수는 로컬 섀도우 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT Device Shadow 서비스의 초당 최대 요청 수는 사용자에 따라 다릅니다 AWS 리전. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

`shadowDocumentSizeLimitBytes`

(선택 사항) 로컬 섀도우에 대한 각 JSON 상태 문서의 최대 허용 크기입니다.

이 값을 늘리면 구름 그림자에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 의 AWS IoT Device Shadow Service API에 대한 제한을 참조하십시오 Amazon Web Services 일반 참조.

기본값: 8192바이트

최대: 30720바이트

Example 예: 구성 병합 업데이트

다음 예제는 섀도우 관리자 구성 요소에 사용 가능한 모든 구성 매개 변수가 포함된 샘플 구성 병합 업데이트를 보여줍니다.

```
{
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": []
      }
    ]
  },
```

```
  "rateLimits": {
    "maxOutboundSyncUpdatesPerSecond": 100,
    "maxTotalLocalRequestsRate": 200,
    "maxLocalRequestsPerSecondPerThing": 20
  },
  "shadowDocumentSizeLimitBytes": 8192
}
```

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.3.6	버그 수정 및 개선 • 장치가 오프라인 상태일 때 AWS 클라우드 업데이트를 통해 삭제된 섀도우 속성이 다시 연결되더라도 로컬 섀도우에 계속 남아 있는 문제를 수정합니다.
2.3.5	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.4	버그 수정 및 개선 • null 및 빈 섀도우 상태 문서에 대한 지원을 추가합니다.
2.3.3	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.2	버그 수정 및 개선 • 로컬 섀도우 데이터베이스가 손상되면 섀도우 관리자가 BROKEN 상태로 전환되는 문제를 수정합니다. • 그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.3.1	버그 수정 및 개선 • 클라우드 섀도우 업데이트가 동기화되지 않을 수 있는 상태를 수정합니다. • 명명된 섀도우 동기화 구성의 변경 사항이 명명된 섀도우 하나에만 적용되는 문제를 수정합니다.
2.3.0	버그 수정 및 개선 • Greengrass 장치 개인 키가 하드웨어 보안 모듈에 저장되어 있을 때 섀도우가 동기화되지 않는 문제를 수정합니다.
2.2.4	버그 수정 및 개선 • 로컬 섀도우 문서를 업데이트할 때 섀도우 크기 검증이 클라우드와 일치하지 않던 문제를 수정합니다.

버전	변경
	• 배포가 구성 노드에서 a를 수행하는 경우 섀도우 관리자가 구성 업데이트 수신을 RESET 중지하는 문제를 수정합니다.
2.2.3	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.2	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.1	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.2.0	새로운 기능 • 로컬 게시/구독 인터페이스를 통해 로컬 섀도우 서비스에 대한 지원을 추가합니다. 이제 섀도우 MQTT 주제에 대해 로컬 게시/구독 메시지 브로커와 통신하여 코어 디바이스에서 섀도우를 가져오고, 업데이트하고, 삭제할 수 있습니다. 이 기능을 사용하면 MQTT 브리지를 사용하여 클라이언트 장치와 로컬 게시/구독 인터페이스 간에 섀도우 주제에 대한 메시지를 릴레이함으로써 클라이언트 장치를 로컬 섀도우 서비스에 연결할 수 있습니다. 이 기능을 사용하려면 v2.6.0 이상의 Greengrass 핵 구성 요소가 필요합니다. 클라이언트 디바이스를 로컬 섀도우 서비스에 연결하려면 MQTT 브리지 구성 요소 v2.2.0 이상도 사용해야 합니다. • 로컬 섀도우 서비스와 에서 섀도우를 동기화하는 방향을 사용자 지정하도록 구성할 수 있는 `direction` 옵션을 추가합니다. AWS 클라우드 이 옵션을 구성하여 에 대한 대역폭과 연결을 줄일 수 AWS 클라우드 있습니다.
2.1.1	버그 수정 및 개선 • JSON 장치 섀도우 상태 문서의 `desired` 및 `reported` 섹션의 최대 깊이가 5레벨이 아닌 4레벨이었던 문제를 수정합니다. • Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • 주기적인 섀도우 동기화 간격에 대한 지원을 추가하여 대역폭 사용량과 요금을 줄이도록 코어 디바이스를 구성할 수 있습니다.

버전	변경
2.0.6	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	버그 수정 및 개선 • 섀도우 관리자가 이전에 삭제된 섀도우의 새로 생성된 버전을 삭제하던 문제를 수정합니다. • `DeleteThingShadow` IPC 작업을 업데이트하여 호출 시 섀도우 버전을 증가시킵니다.
2.0.3	Greengrass 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.2	버그 수정 및 개선 • 에서 AWS IoT Core 섀도우 상태를 동기화할 때 섀도우 관리자가 `delta` 속성을 인식하지 못하던 문제를 수정했습니다. • 가끔 섀도우에 대한 동기화 요청이 잘못 병합되는 문제를 수정했습니다.
2.0.1	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.0	초기 버전

Amazon SNS

Amazon SNS 구성 요소 (`aws.greengrass.SNS`) 는 메시지를 Amazon Simple Notification Service (Amazon SNS) 주제에 게시합니다. 이 구성 요소를 사용하여 Greengrass 코어 디바이스에서 웹 서버, 이메일 주소 및 기타 메시지 구독자에게 이벤트를 보낼 수 있습니다. 자세한 정보는 Amazon Simple Notification 개발자 안내서의 What is Amazon SNS?를 참조하십시오.

이 구성 요소를 사용하여 Amazon SNS 주제에 게시하려면 이 구성 요소가 구독하는 주제에 메시지를 게시하십시오. 기본적으로 이 구성 요소는 `sns/message` 로컬 게시/구독 주제를 구독합니다. 이 구성 요소를 배포할 때 AWS IoT Core MQTT 주제를 비롯한 다른 주제를 지정할 수 있습니다.

사용자 지정 구성 요소에서 다른 소스의 메시지를 이 구성 요소에 게시하기 전에 처리하도록 필터링 또는 형식 지정 로직을 구현하고 싶을 수 있습니다. 이렇게 하면 메시지 처리 로직을 단일 구성 요소에 중앙 집중화할 수 있습니다.

> **Note**
>
> 이 구성 요소는 AWS IoT Greengrass V1의 Amazon SNS 커넥터와 유사한 기능을 제공합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 Amazon SNS 커넥터를 참조하십시오.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 Lambda 구성 요소 () 입니다. `aws.greengrass.lambda` Greengrass 핵은 Lambda 런처 구성 요소를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- 코어 디바이스에 설치되고 PATH 환경 변수에 추가된 Python 버전 3.7입니다.
- Amazon SNS 주제. 자세한 설명은 Amazon Simple Notification Service 개발자 안내서에서 Amazon SNS 주제 생성을 참조하세요.
- Greengrass 장치 역할은 다음 예시 IAM 정책에 나와 있는 것처럼 `sns:Publish` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

  이 구성 요소의 입력 메시지 페이로드에서 기본 주제를 동적으로 재정의할 수 있습니다. 애플리케이션에서 이 기능을 사용하는 경우 IAM 정책은 모든 대상 주제를 리소스로 포함해야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해).
- 이 구성 요소로부터 출력 데이터를 받으려면 이 구성 요소를 배포할 때 기존 구독 라우터 구성 요소(aws.greengrass.LegacySubscriptionRouter) 에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성은 이 구성 요소가 응답을 게시하는 주제를 지정합니다.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "component:aws.greengrass.SNS",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

- *### AWS ## #### ####* 바꾸십시오.
- *###* 이 구성 요소가 실행하는 Lambda 함수 버전으로 교체하십시오. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소 버전의 레시피를 확인해야 합니다. AWS IoT Greengrass 콘솔에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 쌍을 찾으십시오. 이 키-값 쌍에는 Lambda 함수의 이름과 버전이 들어 있습니다.

> **Important**
>
> 이 구성 요소를 배포할 때마다 레거시 구독 라우터에서 Lambda 함수 버전을 업데이트해야 합니다. 이렇게 하면 배포하는 구성 요소 버전에 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 설명은 배포 만들기 섹션을 참조하세요.

- Amazon SNS 구성 요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.
  - Amazon SNS 구성 요소에는 VPC 엔드포인트가 다음과 같은 연결이 있어야 합니다. `sns.region.amazonaws.com` `com.amazonaws.us-east-1.sns`

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`sns.region.amazonaws.com`	443	예	Amazon SNS에 메시지 게시

의존성

구성 요소를 배포할 때 는 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.8 - 2.1.0

다음 표에는 이 구성 요소의 버전 2.0.8 및 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	하드
람다 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	하드
람다 런처	>=1.0.0	하드
Lambda 런타임	>=1.0.0	소프트
토큰 교환 서비스	>=1.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

> **Note**
>
> 이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함됩니다. 디바이스에서 이 구성 요소를 구성하려면 다음 파라미터만 편집하는 것이 좋습니다.

`lambdaParams`

이 구성 요소의 Lambda 함수에 대한 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`EnvironmentVariables`

Lambda 함수의 파라미터를 포함하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

DEFAULT_SNS_ARN

이 구성 요소가 메시지를 게시하는 기본 Amazon SNS 주제의 ARN입니다. 입력 메시지 페이로드의 sns_topic_arn 속성으로 대상 주제를 재정의할 수 있습니다.

containerMode

(선택 사항) 이 구성 요소의 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- NoContainer— 구성 요소는 격리된 런타임 환경에서 실행되지 않습니다.
- GreengrassContainer— 구성 요소는 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다.

기본값: GreengrassContainer

containerParams

(선택 사항) 이 구성 요소의 컨테이너 매개 변수를 포함하는 개체입니다. 를 GreengrassContainer 지정하면 구성 요소에서 이러한 매개 변수를 사용합니다containerMode.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

memorySize

(선택 사항) 구성 요소에 할당할 메모리 양 (KB) 입니다.

기본값은 512MB (525,312KB) 입니다.

pubsubTopics

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제를 포함하는 객체입니다. 각 주제를 지정하고 구성 요소가 MQTT 주제를 구독하는지 AWS IoT Core 아니면 로컬 게시/구독 주제의 MQTT 주제를 구독할지를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0— 문자열 형식의 배열 인덱스입니다.

다음 정보가 포함된 객체입니다.

type

(선택 사항) 이 구성 요소가 메시지를 구독하는 데 사용하는 게시/구독 메시지의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
- IOT_CORE— AWS IoT Core MQTT 메시지를 구독하십시오. 이 옵션을 선택하면 주제에 MQTT 와일드카드가 포함될 수 있습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

기본값: PUB_SUB

`topic`

(선택 사항) 구성 요소가 메시지 수신을 위해 구독하는 주제입니다. 를 지정하는 경우 이 `IotCore type` 항목에서 MQTT 와일드카드 (+및#) 를 사용할 수 있습니다.

Example 예: 구성 병합 업데이트 (컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 예: 구성 병합 업데이트 (컨테이너 모드 없음)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

입력 데이터

이 구성 요소는 다음 주제에 대한 메시지를 수락하고 메시지를 대상 Amazon SNS 주제에 있는 그대로 게시합니다. 기본적으로 이 구성 요소는 로컬 게시/구독 메시지를 구독합니다. 사용자 정의 구성 요소

에서 이 구성 요소에 메시지를 게시하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독

기본 주제 (로컬 게시/구독): `sns/message`

메시지는 다음 속성을 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

`request`

Amazon SNS 주제로 전송할 메시지에 대한 정보입니다.

다음 정보가 `object` 포함된 유형:

`message`

메시지의 내용 (문자열).

JSON 객체를 보내려면 객체를 문자열로 직렬화하고 `json` 속성에 `message_structure` 지정하십시오.

유형: `string`

`subject`

(선택 사항) 메시지 제목.

유형: `string`

제목은 ASCII 텍스트이며 최대 100자까지 입력할 수 있습니다. 문자, 숫자 또는 문장 부호로 시작해야 합니다. 줄 바꿈이나 제어 문자는 포함할 수 없습니다.

`sns_topic_arn`

(선택 사항) 이 구성 요소가 메시지를 게시하는 Amazon SNS 주제의 ARN입니다. 기본 Amazon SNS 주제를 재정의하려면 이 속성을 지정하십시오.

유형: `string`

`message_structure`

(선택 사항) 메시지 구조. 속성에서 문자열로 직렬화하는 JSON 메시지를 `json` 보내도록 지정합니다. `content`

유형: `string`

유효값: `json`

`id`

요청에 대한 임의의 ID입니다. 이 속성을 사용하여 입력 요청을 출력 응답에 매핑할 수 있습니다. 이 속성을 지정하면 구성 요소가 응답 개체의 `id` 속성을 이 값으로 설정합니다.

유형: `string`

> **Note**
>
> 메시지 크기는 최대 256KB일 수 있습니다.

Example 입력 예: 문자열 메시지

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example 입력 예: JSON 메시지

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

출력 데이터

이 구성 요소는 기본적으로 응답을 다음 MQTT 주제에 출력 데이터로 게시합니다. 이 주제를 레거시 구독 라우터 구성 요소의 `subject` 구성에서 로 지정해야 합니다. 사용자 지정 구성 요소에서 이 항목의 메시지를 구독하는 방법에 대한 자세한 내용은 을 참조하십시오MQTT 메시지 게시/구독 AWS IoT Core.

기본 주제 (AWS IoT Core MQTT): `sns/message/status`

Example 출력 예: 성공

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example 출력 예: 실패

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass 루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
  ```

라이선스

이 구성 요소에는 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 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.7	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.6	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.5	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.3	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.2	Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.1.0	새로운 기능 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스가 HTTPS 프록시를 신뢰할 수 있도록 하세요. 섹션을 참조하세요.
2.0.8	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.7	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.6	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.5	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.4	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.3	초기 버전

스트림 관리자

스트림 관리자 구성 요소 (`aws.greengrass.StreamManager`) 를 사용하면 Greengrass 코어 AWS 클라우드 디바이스에서 전송할 데이터 스트림을 처리할 수 있습니다.

사용자 지정 구성 요소에서 스트림 관리자를 구성하고 사용하는 방법에 대한 자세한 내용은 을 참조하십시오Greengrass 코어 디바이스의 데이터 스트림 관리.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.1.x
- 2.0.x

> **Note**
>
> 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 스트림 관리자 구성 요소 버전 2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 수 없습니다. 스트림 관리자를 처음 배포하는 경우 스트림 관리자 구성 요소의 최신 버전을 배포하는 것이 좋습니다.

유형

이 구성 요소는 일반 구성 요소 (`aws.greengrass.generic`) 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 토큰 교환 역할은 스트림 관리자와 함께 사용하는 AWS 클라우드 대상에 대한 액세스를 허용해야 합니다. 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”
- 스트림 관리자 구성 요소는 VPC에서 실행되도록 지원됩니다. VPC에 이 구성 요소를 배포하려면 다음이 필요합니다.
  - 스트림 관리자 구성 요소는 데이터를 게시하는 AWS 서비스에 연결되어 있어야 합니다.
    - 아마존 S3: `com.amazonaws.region.s3`
    - 아마존 키네시스 데이터 스트림: `com.amazonaws.region.kinesis-streams`
    - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
  - `us-east-1`리전의 Amazon S3에 데이터를 게시하면 이 구성 요소는 기본적으로 S3 글로벌 엔드포인트를 사용하려고 시도하지만 Amazon S3 VPC 인터페이스 엔드포인트를 통해 이 엔드포인트를 사용할 수 없습니다. 자세한 AWS PrivateLink 내용은 Amazon S3의 제한 및 제한을 참조하십시오. 이 문제를 해결하려면 다음 옵션 중에서 선택할 수 있습니다.
    - `AWS_S3_US_EAST_1_REGIONAL_ENDPOINT=regional`환경 변수를 제공하여 해당 `us-east-1` 지역의 S3 엔드포인트를 사용하도록 스트림 관리자 구성 요소를 구성합니다.
    - Amazon S3 인터페이스 VPC 엔드포인트 대신 Amazon S3 게이트웨이 VPC 엔드포인트를 생성합니다. S3 게이트웨이 엔드포인트는 S3 글로벌 엔드포인트에 대한 액세스를 지원합니다. 자세한 내용은 게이트웨이 엔드포인트 생성을 참조하십시오.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`iotanalytics.region.amazonaws.com`	443	아니요	에 데이터를 게시하는 경우 필수입니다. AWS IoT Analytics
`kinesis.region.amazonaws.com`	443	아니요	Firehose에 데이터를 게시하

엔드포인트	포트	필수	설명
			는 경우 필수입니다.
data.iotsitewise. *region*.amazonaws.com	443	아니요	에 데이터를 게시하는 경우 필수입니다. AWS IoT SiteWise
*.s3.amazonaws.com	443	아니요	S3 버킷에 데이터를 게시하는 경우 필수입니다. 데이터를 게시하는 각 버킷의 * 이름으로 바꿀 수 있습니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11~2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.13.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.9 – 2.1.10

다음 표에는 이 구성 요소의 버전 2.1.9~2.1.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.12.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.5 – 2.1.8

다음 표에는 이 구성 요소의 버전 2.1.5~2.1.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.11.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.2 – 2.1.4

다음 표에는 이 구성 요소의 버전 2.1.2~2.1.4에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.10.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.9.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.8.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.15

다음 표에는 이 구성 요소의 버전 2.0.15에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.7.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.13 and 2.0.14

다음 표에는 이 구성 요소의 버전 2.0.13 및 2.0.14에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.6.0	소프트

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	>=0.0.0	하드

2.0.11 and 2.0.12

다음 표에는 이 구성 요소의 버전 2.0.11 및 2.0.12에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.5.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.10

다음 표에는 이 구성 요소의 버전 2.0.10에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.4.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.3.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.0 <2.2.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.0.3 <2.1.0	소프트
토큰 교환 서비스	>=0.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`STREAM_MANAGER_STORE_ROOT_DIR`

(선택 사항) 스트림을 저장하는 데 사용되는 로컬 디렉토리의 절대 경로입니다. 이 값은 슬래시로 시작해야 합니다(예: `/data`).

기존 폴더를 지정해야 하며 Stream Manager 구성 요소를 실행하는 시스템 사용자에게 이 폴더에 대한 읽기 및 쓰기 권한이 있어야 합니다. 예를 들어 다음 명령을 실행하여 폴더를 만들고 구성할 수 있으며`/var/greengrass/streams`, 이 폴더를 스트림 관리자 루트 폴더로 지정할 수 있습니다. 이 명령을 사용하면 기본 시스템 사용자 () 가 이 폴더를 읽고 쓸 수 있습니다. `ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

기본값: *`/greengrass/v2`*`/work/aws.greengrass.StreamManager`

`STREAM_MANAGER_SERVER_PORT`

(선택 사항) 스트림 관리자와 통신하는 데 사용할 로컬 포트 번호입니다.

사용 가능한 임의 포트를 `0` 사용하도록 지정할 수 있습니다.

기본값: `8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(선택 사항) 클라이언트가 스트림 관리자와 상호 작용하기 전에 인증을 필수로 설정할 수 있습니다. Stream Manager SDK는 클라이언트와 스트림 관리자 간의 상호 작용을 제어합니다. 이 매개변수는 스트림 작업을 위해 Stream Manager SDK를 호출할 수 있는 클라이언트를 결정합니다. 자세한 내용은 스트림 관리자 클라이언트 인증을 참조하십시오.

지정하는 `true` 경우 스트림 관리자 SDK는 Greengrass 구성 요소만 클라이언트로 허용합니다.

지정하는 `false` 경우 Stream Manager SDK를 사용하면 코어 기기의 모든 프로세스를 클라이언트로 사용할 수 있습니다.

기본값: `true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(선택 사항) 스트림 관리자가 데이터를 내보내는 데 사용할 수 있는 평균 최대 대역폭 (초당 킬로비트) 입니다.

기본값: 제한 없음

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(선택 사항) Stream Manager가 데이터를 내보내는 데 사용할 수 있는 최대 활성 스레드 수입니다.

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 제약이 있는 기기의 내보내기 스트림 수를 제한하세요.

기본값: 스레드 5개

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

(선택 사항) 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.

> **Note**
>
> 스트림 관리자는 stream `sizeThresholdForMultipartUploadBytes` 속성을 사용하여 Amazon S3에 단일 업로드로 내보낼지 멀티파트 업로드로 내보낼지 결정합니다. AWS IoT Greengrass 구성 요소는 Amazon S3로 내보내는 스트림을 생성할 때 이 임계값을 설정할 수 있습니다.

기본값: `5242880` (5MB). 이 값은 최소값이기도 합니다.

`LOG_LEVEL`

(선택 사항) 구성 요소의 로깅 수준입니다. 여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

기본값: `INFO`

`JVM_ARGS`

(선택 사항) 시작 시 스트림 관리자에 전달할 사용자 지정 Java 가상 머신 인수. 여러 인수를 공백으로 구분합니다.

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.

Example 예: 구성 병합 업데이트

다음 예제 구성은 기본이 아닌 포트를 사용하도록 지정합니다.

```
{
  "STREAM_MANAGER_SERVER_PORT": "18088"
}
```

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.StreamManager.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 -
  Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.1.12	버그 수정 및 개선 Greengrass 자격 증명이 AWS 서비스 요청 시 선호되도록 자격 증명이 사용되는 순서를 업데이트합니다.
2.1.11	그린그래스 뉴클리어스 버전 2.12.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.10	버그 수정 및 개선 HTTPS 프록시 구성이 Greengrass CA (인증 기관) 인증서 체인을 신뢰하지 않는 문제를 수정합니다.
2.1.9	그린그래스 뉴클리어스 버전 2.11.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.8	버그 수정 및 개선 스트림 관리자가 SiteWise 익스포트를 무한 재시도하다가 실패하는 문제를 수정합니다. `InvalidRequestException`
2.1.7	버그 수정 및 개선 스트림 관리자가 프록시 구성을 제대로 읽지 못하는 문제를 수정합니다.
2.1.6	버그 수정 및 개선 Jetson Nano를 비롯한 특정 ARMv8 프로세서에서 시작 시 충돌이 발생할 수 있는 문제를 수정합니다.
2.1.5	그린그래스 뉴클리어스 버전 2.10.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.4	버그 수정 및 개선 • 단일 배치 내에서 타임스탬프가 동일한 동일한 속성 에셋에 대한 항목이 `ConflictingOperationException` SiteWise API에서 반환되어 스트림 관리자가 계속 재시도하는 문제를 수정합니다. • 기본 연결 제한 시간을 3초에서 1분으로 업데이트합니다.

버전	변경
2.1.3	버그 수정 및 개선 시스템 사용자로 실행할 때 Windows OS에서 시작 문제가 해결되었습니다.
2.1.2	버그 수정 및 개선 • 영어가 아닌 언어를 사용하는 Windows OS의 문제를 수정합니다. • Greengrass 뉴클리어스 버전 2.9.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.1	Greengrass 뉴클리어스 버전 2.8.0 릴리스를 위해 버전이 업데이트되었습니다.
2.1.0	새로운 기능 • EventBridgeAmazon에 텔레메트리 지표를 자동으로 전송하도록 이 구성 요소를 업데이트합니다. 자세한 설명은 AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집 섹션을 참조하세요. 이 기능을 사용하려면 v2.7.0 이상의 Greengrass 핵 구성 요소가 필요합니다. • Greengrass 뉴클리어스 버전 2.7.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.15	Greengrass 뉴클리어스 버전 2.6.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.14	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.
2.0.13	Greengrass 뉴클리어스 버전 2.5.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.12	버그 수정 및 개선 스트림 관리자 v2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드하지 못했던 문제를 수정합니다. 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 이제 v2.0.12로 업그레이드할 수 있습니다.
2.0.11	그린그래스 뉴클리어스 버전 2.4.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.10	그린그래스 뉴클리어스 버전 2.3.0 릴리스를 위해 버전이 업데이트되었습니다.

버전	변경
2.0.9	Greengrass 뉴클리어스 버전 2.2.0 릴리스를 위해 버전이 업데이트되었습니다.
2.0.8	그린그래스 뉴클리어스 버전 2.1.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.7	초기 버전

Systems Manager 에이전트

AWS Systems Manager 에이전트 구성 요소 (`aws.greengrass.SystemsManagerAgent`) 는 Systems Manager 에이전트를 설치하므로 Systems Manager를 사용하여 핵심 장치를 관리할 수 있습니다. Systems Manager는 Amazon EC2 인스턴스 AWS, 온프레미스 서버 및 VM (가상 머신), 에지 디바이스 등의 인프라를 보고 제어하는 데 사용할 수 있는 AWS 서비스입니다. Systems Manager를 사용하면 운영 데이터를 보고, 운영 작업을 자동화하고, 보안 및 규정 준수를 유지할 수 있습니다. 자세한 내용은 AWS Systems Manager무엇입니까를 참조하십시오. 및 AWS Systems Manager 사용 설명서의 Systems Manager 에이전트 정보

Systems Manager 도구 및 기능을 기능이라고 합니다. 그린그래스 코어 디바이스는 모든 Systems Manager 기능을 지원합니다. 이러한 기능 및 Systems Manager를 사용하여 핵심 장치를 관리하는 방법에 대한 자세한 내용은 사용 AWS Systems Manager 설명서의 Systems Manager 기능을 참조하십시오.

주제

- 버전
- 유형
- 운영 체제
- 요구 사항
- 의존성
- 구성
- 로컬 로그 파일
- 다음 사항도 참조하십시오.
- Changelog

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 64비트 리눅스 플랫폼에서 실행되는 그린그래스 코어 디바이스: Armv8 (AArch64) 또는 x86_64.
- Systems Manager가 위임할 수 있는 AWS Identity and Access Management (IAM) 서비스 역할이 있어야 합니다. 이 역할에는 AmazonSSM ManagedInstanceCore 관리형 정책 또는 동등한 권한을 정의하는 사용자 지정 정책이 포함되어야 합니다. 자세한 내용은 사용 설명서의 에지 디바이스용 IAM 서비스 역할 생성을 참조하십시오.AWS Systems Manager

  이 구성 요소를 배포할 때는 `SSMRegistrationRole` 구성 파라미터에 이 역할의 이름을 지정해야 합니다.
- Greengrass 장치 역할은 `ssm:AddTagsToResource` 및 `ssm:RegisterManagedInstance` 작업을 허용해야 합니다. 또한 장치 역할은 이전 요구 사항을 충족하는 IAM 서비스 역할에 대한 `iam:PassRole` 작업을 허용해야 합니다. 다음 예시 IAM 정책은 이러한 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
ec2messages. *region*.amazonaws.com	443	예	에서 Systems Manager 서비스와 통신하십시오 AWS 클라우드.
ssm.*region*.amazonaws.com	443	예	코어 디바이스를 Systems Manager 관리 노드

엔드포인트	포트	필수	설명
			로 등록합니다.
ssmmessages. *region*.amazonaws.com	443	예	에서 Systems Manager의 기능인 세션 관리자와 통신할 수 AWS 클라우드있습니다.

자세한 내용은 사용 설명서의 참조: ec2message, ssmmessages 및 기타 API 호출을 참조하십시오.AWS Systems Manager

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다.AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

다음 표에는 이 구성 요소의 버전 1.0.0~1.1.0에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	^2.0.0	소프트

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음과 같은 구성 매개 변수를 제공합니다.

`SSMRegistrationRole`

Systems Manager가 맡을 수 있는 IAM 서비스 역할이며, 여기에는 AmazonSSM ManagedInstanceCore 관리형 정책 또는 동등한 권한을 정의하는 사용자 지정 정책이 포함됩니다. 자세한 내용은 사용 설명서의 에지 디바이스용 IAM 서비스 역할 생성을 참조하십시오.AWS Systems Manager

`SSMOverrideExistingRegistration`

(선택 사항) 하이브리드 활성화로 등록된 Systems Manager Agent가 코어 디바이스에서 이미 실행되고 있는 경우 디바이스의 기존 Systems Manager Agent 등록을 재정의할 수 있습니다. 이 구성 요소가 제공하는 Systems Manager Agent를 사용하여 코어 디바이스를 관리 노드로 `true` 등록하려면 이 옵션을 설정합니다.

> **Note**
>
> 이 옵션은 하이브리드 활성화로 등록된 장치에만 적용됩니다. 코어 디바이스가 Systems Manager 에이전트가 설치되어 있고 인스턴스 프로필 역할이 구성된 Amazon EC2 인스턴스에서 실행되는 경우 Amazon EC2 인스턴스의 기존 관리형 노드 ID는 로 시작합니다. `i-` Systems Manager Agent 구성 요소를 설치하면 Systems Manager 에이전트는 ID가 `mi-` 대신 로 시작하는 새 관리 노드를 등록합니다. `i-` 그런 다음 ID가 로 `mi-` 시작하는 관리 노드를 사용하여 Systems Manager에서 코어 디바이스를 관리할 수 있습니다.

기본값: `false`

`SSMResourceTags`

(선택 사항) 이 구성 요소가 코어 장치용으로 생성하는 Systems Manager 관리 노드에 추가할 태그입니다. Systems Manager에서 이러한 태그를 사용하여 핵심 장치 그룹을 관리할 수 있습니다. 예를 들어, 지정한 태그가 있는 모든 장치에서 명령을 실행할 수 있습니다.

각 태그가 `Key` a와 a가 있는 객체인 목록을 `Value` 지정하십시오. 예를 들어 다음 값은 이 구성 요소가 핵심 기기의 관리 노드에서 **`Owner`** 태그를 로 `SSMResourceTags` **`richard-roe`** 설정하도록 지시합니다.

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
```

```
]
```

관리 노드가 이미 존재하고 `SSMOverrideExistingRegistration` 있는 경우 이 구성 요소는 이러한 태그를 무시합니다. `false`

Example 예: 구성 병합 업데이트

다음 예제 구성은 코어 장치가 Systems Manager에 등록하고 `SSMServiceRole` 통신할 수 있도록 이름이 지정된 서비스 역할을 사용하도록 지정합니다.

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

로컬 로그 파일

Systems Manager Agent 소프트웨어는 Greengrass 루트 폴더 외부의 폴더에 로그를 기록합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 Systems Manager 에이전트 로그 보기를 참조하십시오.

Systems Manager Agent 구성 요소는 셸 스크립트를 사용하여 Systems Manager 에이전트를 설치, 시작 및 중지합니다. 다음 로그 파일에서 이러한 스크립트의 출력을 찾을 수 있습니다.

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass 루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

다음 사항도 참조하십시오.

- 다음을 사용하여 Greengrass 코어 장치를 관리하십시오. AWS Systems Manager
- AWS Systems Manager 사용 설명서의 AWS Systems Manager란 무엇입니까?
- AWS Systems Manager 사용 설명서의 Systems Manager 에이전트에 대한 정보

Changelog

다음 표는 각 구성 요소 버전의 변경 사항을 설명합니다.

버전	변경
1.1.0	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.
1.0.0	초기 버전

토큰 교환 서비스

토큰 교환 서비스 구성 요소 (`aws.greengrass.TokenExchangeService`) 는 사용자 지정 구성 요소의 AWS 서비스와 상호 작용하는 데 사용할 수 있는 AWS 자격 증명을 제공합니다.

토큰 교환 서비스는 Amazon Elastic Container Service (Amazon ECS) 컨테이너 인스턴스를 로컬 서버로 실행합니다. 이 로컬 서버는 Greengrass 코어 nucleus 구성 요소에서 구성한 AWS IoT 역할 별칭을 사용하여 AWS IoT 자격 증명 공급자에 연결합니다. 구성 요소는 두 개의 환경 변수 및 를 제공합니다. `AWS_CONTAINER_CREDENTIALS_FULL_URI` `AWS_CONTAINER_AUTHORIZATION_TOKEN` `AWS_CONTAINER_CREDENTIALS_FULL_URI`이 로컬 서버의 URI를 정의합니다. 구성 요소가 AWS SDK 클라이언트를 만들면 클라이언트는 이 URI 환경 변수를 인식하고 의 토큰을 사용하여 토큰 교환 서비스에 연결하고 자격 증명을 AWS 검색합니다. `AWS_CONTAINER_AUTHORIZATION_TOKEN` 이를 통해 Greengrass 코어 디바이스는 AWS 서비스 오퍼레이션을 호출할 수 있습니다. 사용자 지정 구성 요소에서 이 구성 요소를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오AWS서비스와 상호작용.

> **Important**
>
> 이러한 방식의 AWS 자격 증명 취득 지원이 2016년 7월 13일에 AWS SDK에 추가되었습니다. 구성 요소는 해당 날짜 또는 그 이후에 생성된 AWS SDK 버전을 사용해야 합니다. 자세한 내용은 Amazon Elastic 컨테이너 서비스 개발자 안내서의 지원되는 AWS SDK 사용을 참조하십시오.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux

- Windows

의존성

이 구성 요소에는 종속성이 없습니다.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 Greengrass 핵 구성 요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.0.3	초기 버전

IoT SiteWise OPC-UA 컬렉터

IoT SiteWise OPC-UA 수집기 구성 요소 (`aws.iot.SiteWiseEdgeCollectorOpcua`) 를 사용하면 AWS IoT SiteWise 게이트웨이가 로컬 OPC-UA 서버에서 데이터를 수집할 수 있습니다.

이 구성 요소를 사용하면 AWS IoT SiteWise 게이트웨이를 여러 OPC-UA 서버에 연결할 수 있습니다. AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 사용 설명서의 AWS IoT SiteWise Edge에서의 사용을 참조하십시오.AWS IoT SiteWise

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 2.4.x
- 2.3.x

- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 코어 디바이스는 다음 플랫폼 중 하나에서 실행되어야 합니다.
  - 운영체제: Ubuntu 18.04 이상

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영체제: Red Hat Enterprise Linux (RHEL) 8

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영체제: Amazon Linux 2

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영체제: Debian 11

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - OS: 윈도우 서버 2019 이상

    아키텍처: x86_64 (AMD64)

- Greengrass 코어 디바이스는 OPC-UA 서버에 대한 아웃바운드 네트워크 연결을 허용해야 합니다.

의존성

구성 요소를 배포할 때 호환되는 버전의 AWS IoT Greengrass 종속 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다.AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

다음 표에는 이 구성 요소의 모든 버전에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.3.0 <3.0.0	하드
스트림 관리자	>=2.3.0 2.0.10<3.0.0	하드
시크릿 매니저	>2.0.10 =2.0.8 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

AWS IoT SiteWise 콘솔 또는 API를 사용하여 IoT SiteWise OPC-UA 수집기 구성요소를 구성할 수 있습니다. 자세한 내용은 사용 AWS IoT SiteWise 설명서의 4단계: 데이터 소스 추가 - 선택 사항을 참조하십시오.

입력 데이터

이 구성 요소는 다음 형식의 데이터만 받아들이고 다른 모든 형식은 무시되고 삭제됩니다. 아래 표는 OPC UA 데이터 유형을 해당 데이터 유형과 매핑합니다. SiteWise

SiteWise 데이터 유형	OPC UA 데이터 유형	설명
`STRING`	`String`	최대 길이가 1024바이트인 문자열입니다.

SiteWise 데이터 유형	OPC UA 데이터 유형	설명
	Guid XmlElement	
INTEGER	SByte Byte Int16 UInt16 Int32 UInt32* Int64*	범위가 1인 부호있는 32비트 정수. -2,147,483,648 to 2,147,483,647
DOUBLE	UInt32* Int64* Float Double	범위가 0이고 배정밀도가 두 배인 부동 소수점 숫자입니다- 10^100 to 10^100. IEEE 754
BOOLEAN	Boolean	true 또는 false

* OPC UA SiteWise 데이터 유형 UInt32 및 Int64 의 경우 해당 값을 나타낼 수 INTEGER 있으면 해당 데이터 유형이 SiteWise 되고 그렇지 않으면 해당 데이터 유형이 됩니다. DOUBLE

출력 데이터

이 구성 요소는 AWS IoT Greengrass 스트림 관리자에 BatchPutAssetPropertyValue 메시지를 기록합니다. 자세한 내용을 알아보려면 AWS IoT SiteWise API 참조의 BatchPutAssetPropertyValue(을)를 참조하세요.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
   10 -Wait
  ```

문제 해결 및 디버깅

이 구성 요소에는 고객이 문제를 식별하고 해결하는 데 도움이 되는 새 이벤트 로그가 포함되어 있습니다. 로그 파일은 로컬 로그 파일과는 별개이며 다음 위치에서 찾을 수 있습니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua/logs/
IotSiteWiseOpcUaCollectorEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgeCollectorOpcua\logs
\IotSiteWiseOpcUaCollectorEvents.log
```

이 로그에는 자세한 정보와 문제 해결 지침이 포함되어 있습니다. 문제 해결 정보는 문제 해결 방법에 대한 설명과 함께 진단과 함께 제공되며 경우에 따라 추가 정보로 연결되는 링크와 함께 제공됩니다. 진단 정보에는 다음이 포함됩니다.

- 심각도 수준
- Timestamp
- 추가 이벤트별 정보

Example 로그 예

```
dataSourceConnectionSuccess:
  Summary: Successfully connected to OpcUa server
  Level: INFO
  Timestamp: '2023-06-15T21:04:16.303Z'
  Description: Successfully connected to the data source.
  AssociatedMetrics:
  - Name: FetchedDataStreams
    Description: The number of fetched data streams for this data source
    Value: 1.0
    Namespace: IoTSiteWise
    Dimensions:
    - Name: SourceName
      Value: SourceName{value=OPC-UA Server}
    - Name: ThingName
      Value: test-core
  AssociatedData:
  - Name: DataSourceTrace
    Description: Name of the data source
    Data:
    - OPC-UA Server
  - Name: EndpointUri
    Description: The endpoint to which the connection was attempted.
    Data:
    - '"opc.tcp://10.0.0.1:1234"'
```

라이선스

이 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
2.4.2	버그 수정 및 개선 • 노드가 여러 번 검색될 수 있는 OPC UA 서버 검색 중 발생하는 문제를 수정합니다. • 각 스냅샷 데이터 포인트의 타임스탬프가 새로워지도록 스냅샷 기능을 수정합니다.
2.4.1	버그 수정 및 개선 • 프록시 지원 관련 문제를 수정합니다. • 스레드 정리가 실패하여 데이터가 차단되는 문제를 수정했습니다.
2.4.0	새로운 기능 • 문제를 쉽게 식별하고 해결할 수 있도록 이벤트 로그를 추가합니다. 버그 수정 및 개선 • OPC-UA 사양 버전 1.05를 사용하는 OPC-UA 서버에 연결할 때 인증서 오류가 발생했던 OPC-UA 클라이언트 관련 문제를 수정합니다.
2.3.0	새로운 기능 • 리눅스에서 Greengrass nucleus HTTP 프록시 컨피그레이션에 대한 지원을 추가합니다. 버그 수정 및 개선 • 보안 문제를 수정합니다 (CVE-2019-19135).
2.2.0	새로운 기능 • Linux ARMv8 아키텍처에 데이터 수집 팩을 설치하기 위한 지원을 추가합니다. • 리눅스 ARMv8의 최소 요구 사항:

버전	변경
	• 메모리: 4기가바이트 • CPU: ARM 코텍스-A72 또는 이에 상응하는 사양 버그 수정 및 개선 • 노드 검색 프로세스의 메트릭 로깅을 개선합니다. • 지원되지 않는 데이터 유형의 처리를 개선합니다. • 데이터 스트림 오류 로깅을 개선합니다.
2.1.3	새로운 기능 • 윈도우 서버 2019 이상에 대한 지원을 추가합니다. 버그 수정 및 개선 • 지원되지 않는 장치에 이 구성 요소를 배포할 때 나타나는 오류 메시지를 개선합니다.

버전	변경
2.1.1	새로운 기능 • 다음 구독 속성을 구성하기 위한 지원을 추가합니다. • DataChangeTrigger- 데이터 변경 알림을 시작하는 조건을 정의할 수 있습니다. • QueueSize- 모니터링된 항목에 대한 알림이 대기열에 추가되는 특정 측정 단위에 대한 OPC-UA 서버의 대기열 깊이 • PublishingIntervalMilliseconds- 구독이 생성될 때 지정된 게시 주기 간격 (밀리초) • SnapshotFrequencyMilliseconds - AWS IoT SiteWise Edge가 꾸준한 데이터 스트림을 수집하도록 스냅샷 빈도 제한 시간 설정을 구성할 수 있습니다. • 이 버전은 품질 데이터 수집을 지원하고 다음 데이터 BAD 품질을 기반으로 데이터를 필터링합니다. • UNCERTAIN 품질 데이터 • BAD품질 데이터 버그 수정 및 개선 • 고객 지표 개선. • 암호화가 활성화된 상태로 서버에 연결할 때 가끔 문제가 발생했던 보안 인코딩을 수정합니다. • 속성 그룹이 업데이트되지 않던 문제를 수정합니다.
2.0.3	버그 수정 및 개선.
2.0.2	Edge와의 자산 우선 순위 동기화에 대한 버그 수정 및 개선.
2.0.1	초기 버전

다음 사항도 참조하십시오.

- 이게 AWS IoT SiteWise뭐야? AWS IoT SiteWise 사용 설명서에서.

IoT SiteWise OPC-UA 데이터 소스 시뮬레이터

IoT SiteWise OPC-UA 데이터 소스 시뮬레이터 구성 요소 (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) 는 샘플 데이터를 생성하는 로컬 OPC-UA 서버를 시작합니다. 이 OPC-UA 서버를 사용하여 게이트웨이의 IoT SiteWise OPC-UA 수집기 구성요소가 읽는 데이터 소스를 시뮬레이션할 수 있습니다. AWS IoT SiteWise 그런 다음 이 샘플 데이터를 사용하여 AWS IoT SiteWise 기능을 탐색할 수 있습니다. AWS IoT SiteWise게이트웨이에 대한 자세한 내용은 사용 설명서의 Using AWS IoT SiteWise at the Edge를 AWS IoT SiteWise 참조하십시오.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 1.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 코어 디바이스는 로컬 호스트의 포트 4840을 사용할 수 있어야 합니다. 이 구성 요소의 로컬 OPC-UA 서버가 이 포트에서 실행됩니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다. AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

다음 표에는 이 구성 요소의 모든 버전에 대한 종속성이 나열되어 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.3.0 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
  ```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
1.0.0	초기 버전 윈도우 서버 2016 이상에 대한 지원을 추가합니다.

다음 사항도 참조하십시오.

- 무엇입니까AWS IoT SiteWise? AWS IoT SiteWise사용 설명서에서.

IoT SiteWise 퍼블리셔

IoT SiteWise 게시자 구성 요소 (`aws.iot.SiteWiseEdgePublisher`) 를 사용하면 AWS IoT SiteWise 게이트웨이가 에지에서 에지로 데이터를 내보낼 수 있습니다. AWS 클라우드

AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 사용 설명서의 AWS IoT SiteWise Edge에서의 사용을 참조하십시오.AWS IoT SiteWise

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

> **Note**
>
> IoT SiteWise 퍼블리셔 버전 3.1.1이 중단되었습니다. 버전 3.1.0을 사용하는 것이 좋습니다.

- 3.1.x
- 3.0.x
- 2.4.x

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 코어 디바이스는 다음 플랫폼 중 하나에서 실행되어야 합니다.
  - 운영체제: Ubuntu 18.04 이상

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영체제: Red Hat Enterprise Linux (RHEL) 8

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영체제: Amazon Linux 2

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영체제: Debian 11

    아키텍처: x86_64 (AMD64) 또는 ARMv8 (Aarch64)
  - 운영 체제: 윈도우 서버 2019 이상

아키텍처: x86_64 (AMD64)

- Greengrass 코어 디바이스는 인터넷에 연결되어야 합니다.
- Greengrass 코어 디바이스는 작업을 수행할 권한이 있어야 합니다. `iotsitewise:BatchPutAssetPropertyValue` 자세한 내용은 코어 장치가 서비스와 상호 AWS 작용하도록 권한 부여를 참조하십시오.

  Example 권한 정책

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

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`data.iotsitewise.`*`region`*`.amazonaws.com`	443	예	에 데이터를 게시합니다. AWS IoT SiteWise

의존성

구성 요소를 배포할 때 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹

션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다.AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

다음 표에는 이 구성 요소의 버전 2.0.x ~ 2.2.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
그린그래스 핵	>=2.3.0<3.0.0	하드
스트림 관리자	>=2.3.0 =2.0.10<3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

AWS IoT SiteWise 콘솔 또는 API를 사용하여 IoT SiteWise 게시자 구성 요소를 구성할 수 있습니다. 자세한 내용은 사용 AWS IoT SiteWise 설명서의 3단계: 게시자 구성 - 선택 사항을 참조하십시오.

입력 데이터

이 구성 요소는 AWS IoT Greengrass 스트림 관리자에서 `PutAssetPropertyValueEntry` 메시지를 읽습니다. 자세한 내용을 알아보려면 AWS IoT SiteWise API 참조의 PutAssetPropertyValueEntry(을)를 참조하세요.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
  Wait
  ```

문제 해결 및 디버깅

이 구성 요소에는 고객이 문제를 식별하고 해결하는 데 도움이 되는 새 이벤트 로그가 포함되어 있습니다. 로그 파일은 로컬 로그 파일과는 별개이며 다음 위치에서 찾을 수 있습니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs
\IotSiteWisePublisherEvents.log
```

이 로그에는 자세한 정보와 문제 해결 지침이 포함되어 있습니다. 문제 해결 정보는 문제 해결 방법에 대한 설명과 함께 진단과 함께 제공되며 경우에 따라 추가 정보로 연결되는 링크와 함께 제공됩니다. 진단 정보에는 다음이 포함됩니다.

- 심각도 수준
- Timestamp
- 추가 이벤트별 정보

Example 로그 예

```
accountBeingThrottled:
  Summary: Data upload speed slowed due to quota limits
  Level: WARN
  Timestamp: '2023-06-09T21:30:24.654Z'
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points
 ingested"
    quota for a customers account. See the associated documentation and associated
    metric for the number of requests that were limited for more information. Note
    that this may be temporary and not require any change, although if the issue
 continues
    you may need to request an increase for the mentioned quota.
  FurtherInformation:
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
  AssociatedMetrics:
  - Name: TotalErrorCount
    Description: The total number of errors of this type that occurred.
    Value: 327724.0
  AssociatedData:
  - Name: AggregatePropertyAliases
    Description: The aggregated property aliases of the throttled data.
    FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

라이선스

이 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
3.1.1	버그 수정 및 개선 • 오류 발생 시 영향을 받는 데이터 별칭을 식별하는 추가 로깅을 추가합니다.

버전	변경
	• 수집된 데이터의 보관 기간에 대한 AWS IoT SiteWise API 제한의 로컬 적용을 추가합니다. • Amazon S3 대상이 여러 개인 경우 Publisher가 StreamManager 스트림의 체크포인트를 혼동하는 문제를 수정합니다. • 게시자가 스트림을 읽는 방식과 관련된 성능 병목 현상을 수정합니다. StreamManager
3.1.0	새로운 기능 • Amazon S3에 데이터를 파켓 파일로 게시하기 위한 지원을 추가합니다. • AWS IoT SiteWise 버퍼 수집에 대한 지원을 추가합니다.
3.0.0	버그 수정 및 개선 • 프록시 지원 관련 문제를 수정합니다. 새로운 기능 • MQTT 브로커의 데이터 통합을 지원할 수 있습니다.
2.4.1	버그 수정 및 개선 • 구성 요소가 Java Corretto 11 버전 11.0.20.8.1 이상에서 작동하도록 활성화합니다. 구성 요소 버전 2.4.0 및 2.3.3은 Java Corretto 버전 11.0.20.8.1과 함께 사용할 때 `"Could not find or load main class"` 오류 메시지를 표시합니다.
2.4.0	새로운 기능 • 문제를 쉽게 식별하고 수정할 수 있도록 새 이벤트 로그를 추가합니다. 버그 수정 및 개선 • 게시자 체크포인트 복구를 개선합니다.
2.3.3	버그 수정 및 개선 • 높은 처리량을 지원하는 기능을 개선합니다.
2.3.2	버그 수정 및 개선 • 게시자 구성을 다운로드할 때 HTTP 프록시 지원이 수정되었습니다.

버전	변경

2.3.1	새로운 기능
	• Linux ARMv8 아키텍처에 데이터 수집 팩을 설치하기 위한 지원을 추가합니다.
	• 리눅스 ARMv8의 최소 요구 사항:
	  • 메모리: 4기가바이트
	  • CPU: ARM 코텍스-A72 또는 이에 상응하는 사양

2.2.3	버그 수정 및 개선 • 재시도 가능한 예외 목록에 없는 일반 예외에 대한 재시도를 제거합니다.
2.2.2	버그 수정 및 개선 • HTTP 프록시 서버를 AWS IoT SiteWise 통한 데이터 업로드 지원을 다시 도입했습니다.
2.2.1	**Note** 이 버전은 HTTP 프록시 구성을 지원하지 않습니다. 버전 2.2.2 이상에서는 이 기능에 대한 지원이 다시 도입되었습니다. 새로운 기능 • 데이터를 업로드할 때 압축을 전환할 수 있도록 이 구성 요소에 지원을 추가합니다. AWS IoT SiteWise

버전	변경

2.2.0	**Note**
	이 버전은 HTTP 프록시 구성을 지원하지 않습니다. 버전 2.2.2 이상에서는 이 기능에 대한 지원이 다시 도입되었습니다.
	
	새로운 기능
	• 데이터를 서비스로 보내기 전에 데이터를 압축하도록 이 구성 요소를 업데이트합니다. AWS IoT SiteWise
	  • 대부분의 경우 이 변경으로 인해 이 구성 요소의 이전 버전에 비해 대역폭 사용량이 75% 감소합니다.
	  • 대부분의 경우 이 변경으로 인해 CPU 사용량이 최대 5% 까지 증가합니다. 대량의 데이터를 처리하는 게이트웨이에서 이러한 변경으로 인해 CPU 사용량이 최대 15% 증가할 수 있습니다.
	  • 이 변경은 AWS IoT SiteWise 서비스 요금이나 서비스 할당량 사용에 영향을 주지 않습니다.
	  • Windows Server 2019 이상에 대한 지원을 추가합니다.
	버그 수정 및 개선
	• 체크포인트 파일이 손상되면 이 구성 요소가 시작되지 않는 문제를 수정합니다.

2.1.4	버그 수정 및 개선 • Java 버전 8과의 호환성을 수정합니다.

버전	변경
2.1.3	**Warning** 이 버전은 미국 동부 (오하이오), 캐나다 (중부) 및 AWS GovCloud (미국 동부) 지역을 제외하고 더 이상 사용할 수 없습니다. 이 구성 요소 버전을 실행하려면 Java 버전 11 이상이 필요합니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 • 지원되지 않는 장치에 이 구성 요소를 배포할 때 나타나는 오류 메시지를 개선합니다. • 데이터 업로드가 실패할 경우 오류를 기록하도록 업데이트합니다.
2.1.2	버그 수정 및 개선 • 데이터가 만료되는 즉시 만료된 데이터 내보내기 기능을 호출하도록 업데이트되었습니다.
2.1.1	버그 수정 및 개선
2.1.0	새로운 기능 • 최신 데이터를 클라우드에 먼저 게시하기 위한 지원을 추가합니다. • 만료된 데이터를 클라우드에 게시하지 않기 위한 지원을 추가합니다. • 만료된 데이터를 로컬에 저장하기 위한 지원을 추가합니다. 버그 수정 및 개선 • 디스크 I/O 및 해당 지연 시간을 줄입니다.
2.0.2	버그 수정 및 개선
2.0.1	초기 버전

다음 사항도 참조하십시오.

- 이게 뭐야 AWS IoT SiteWise? AWS IoT SiteWise 사용 설명서에서

IoT SiteWise 프로세서

IoT SiteWise 프로세서 구성 요소 (`aws.iot.SiteWiseEdgeProcessor`) 를 사용하면 AWS IoT SiteWise 게이트웨이가 에지에서 데이터를 처리할 수 있습니다.

이 구성 요소를 통해 AWS IoT SiteWise 게이트웨이는 자산 모델 및 자산을 사용하여 게이트웨이 장치의 데이터를 처리할 수 있습니다. AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 사용 AWS IoT SiteWise 설명서의 Using AWS IoT SiteWise at the Edge를 참조하십시오.

주제

버전

이 구성 요소의 버전은 다음과 같습니다.

- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소 () `aws.greengrass.generic` 입니다. Greengrass 핵은 구성 요소의 라이프사이클 스크립트를 실행합니다.

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- Greengrass 코어 디바이스는 다음 플랫폼 중 하나에서 실행되어야 합니다.
  - 운영 체제: 우분투 20.04 또는 18.04

    아키텍처: x86_64 (AMD64)
  - 운영체제: Red Hat Enterprise Linux (RHEL) 8

    아키텍처: x86_64 (AMD64)
  - 운영체제: Amazon Linux 2

    아키텍처: x86_64 (AMD64)
  - 운영 체제: 윈도우 서버 2019 이상

    아키텍처: x86_64 (AMD64)
- Greengrass 코어 디바이스는 포트 443에서 인바운드 트래픽을 허용해야 합니다.
- Greengrass 코어 디바이스는 포트 443과 8883에서 아웃바운드 트래픽을 허용해야 합니다.
- 다음 포트는 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8086, 8445, 9000, 9500, 11080, 50010에서 사용하도록 예약되어 있습니다. AWS IoT SiteWise예약된 포트를 트래픽용으로 사용하면 연결이 종료될 수 있습니다.

> **Note**
>
> 포트 8087은 이 구성 요소의 버전 2.0.15 이상에만 필요합니다.

- Greengrass 장치 역할에는 장치의 AWS IoT SiteWise 게이트웨이를 사용할 수 있는 권한이 있어야 합니다. AWS IoT Greengrass V2 자세한 내용은 AWS IoT SiteWise 사용 설명서의 요구 사항을 참조하십시오.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에도 다음 엔드포인트 및 포트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 설명은 프록시 또는 방화벽을 통한 장치 트래픽 허용 섹션을 참조하세요.

엔드포인트	포트	필수	설명
`model.iotsitewise.region.amazonaws.com`	443	예	AWS IoT SiteWise 자산 및 자산 모델에 대한 정보를 얻으십시오.
`edge.iotsitewise.region.amazonaws.com`	443	예	코어 디바이스의 AWS IoT SiteWise 게이트웨이 구성에 대한 정보를 얻을 수 있습니다.
`ecr.region.amazonaws.com`	443	예	Amazon Elastic 컨테이너 레

엔드포인트	포트	필수	설명
			지스트리에서 AWS IoT SiteWise 엣지 게이트웨이 Docker 이미지를 다운로드하십시오.
iot.*region*.amazonaws.com	443	예	사용자를 위한 디바이스 엔드포인트를 확보하십시오. AWS 계정
sts.*region*.amazonaws.com	443	예	ID를 가져오세요. AWS 계정
monitor.iotsitewise. *region*.amazonaws.com	443	아니요	코어 디바이스에서 AWS IoT SiteWise Monitor 포털에 액세스하는 경우 필요합니다.

의존성

구성 요소를 배포하면 호환되는 버전의 종속 AWS IoT Greengrass 항목도 배포됩니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 해당 종속성에 대한 요구 사항을 모두 충족해야 합니다. 이 섹션에는 이 구성 요소의 릴리스된 버전에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 시맨틱 버전 제약 조건이 나열되어 있습니다. 콘솔에서 구성 요소의 각 버전에 대한 종속성을 볼 수도 있습니다.AWS IoT Greengrass 구성 요소 세부 정보 페이지에서 종속성 목록을 찾아보십시오.

다음 표에는 이 구성 요소의 버전 2.0.x ~ 2.1.x에 대한 종속성이 나와 있습니다.

종속성	호환되는 버전	종속성 유형
토큰 교환 서비스	>=2.0.3 <3.0.0	하드
스트림 관리자	>=2.0.3 =2.0.10 <3.0.0	하드
그린그래스 CLI	>=2.3.0 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 구성 요소 레시피 참조를 참조하십시오.

구성

이 구성 요소에는 구성 매개변수가 없습니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

이 구성 요소의 로그를 보려면

- 코어 기기에서 다음 명령을 실행하여 이 구성 요소의 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꿉니다.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -
  Wait
  ```

라이선스

이 구성 요소에는 다음과 같은 타사 소프트웨어/라이선스가 포함됩니다.

- 아파치-2.0
- 메사추세츠공과대학
- BSD-2-조항
- BSD-3-조항
- CDDL-1.0
- CDDL-1.1
- 디스크
- 즐리브
- GPL-3.0-GCC-예외 적용
- 퍼블릭 도메인
- 파이썬-2.0
- 유니코드-DFS-2015
- BSD-1-조항
- OpenSSL
- EPL-1.0

- EPL-2.0
- GPL-2.0- with-classpath-exception
- MPL-2.0
- CC0-1.0
- JSON

이 구성 요소는 Greengrass Core 소프트웨어 라이센스 계약에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경
3.2.0	성능 개선 • 메모리 사용량을 줄이고 설치에 필요한 디스크 공간을 줄이도록 API 서비스를 최적화합니다. • 이렇게 하면 초기 메모리 사용량이 2GB 줄어들고 (현재는 시작 시 7.5GB의 메모리를 사용하지만 여전히 16GB가 권장됨) 전체 구성 요소의 다운로드 크기는 500MB 감소 (현재는 1.4GB 다운로드 필요) 됩니다. 새로운 기능 • `GetAssetPropertyValueAggregates` API는 이제 엣지에서 15분 어그리게이션 윈도우를 지원합니다. • 포트 8081 및 8082를 더 이상 사용할 필요가 없어 이 구성 요소가 올바르게 실행될 수 있습니다. **Note** AWS IoT SiteWise 데이터 플레인 API (예:) 의 로컬 엔드포인트가 에서 로 `get-asset-property-value` 변경되었습니다. `http://localhost:8081` `http://localhost:11080/data` AWS IoT SiteWise 컨트롤 플레인 API (예:) 의 로컬 엔드포인트가 에서 `http://localhost:11080` 로 `list-asset-models` 변경되고 있습니다. `http://lo`

버전	변경
	`calhost:11080/control` AWS 항상 SiteWise 에지 게이트웨이 HTTPS 엔드포인트를 사용할 것을 권장합니다. 이러한 엔드포인트는 변경되지 않았습니다. 버그 수정 및 개선 • 이전 동기화가 중단된 경우 이제 에서 AWS IoT SiteWise 동기화하면 리소스가 유효한 상태로 전환됩니다. 이렇게 하면 강제 재시작 후 일부 리소스가 손상되는 문제가 해결됩니다. • 동기화 중에 리소스를 수정하면 엣지에서 리소스가 손상될 수 있는 드문 문제를 수정합니다. 이제 이 상태가 감지되면 동기화가 실패하고 다음 동기화 시 리소스가 재시도됩니다. • API용 HTTP 엔드포인트를 외부에서 호출할 수 있었던 문제를 수정합니다. 이제는 HTTPS만 사용하여 로컬 루프백 주소 외부의 API를 호출할 수 있습니다. • `ListAssets` 이제 API는 엣지에 저장된 자산의 자산 계층 구조를 표시합니다. • Windows에서 데이터 처리 팩을 다시 시작, 업그레이드 또는 다운그레이드하지 못하는 문제를 수정합니다. • 고객이 자격 증명을 사용하여 MQTT 브로커에 연결할 수 없었던 Windows OS용 데이터 처리 팩의 버그를 수정합니다.
3.1.3	버그 수정 및 개선 • 일부 리소스가 실제로 실패했을 때 데이터 처리 팩이 동기화 성공을 잘못 보고하던 문제를 수정했습니다. • 상위 항목이 동일하지 않는 한 여러 자산의 이름이 같도록 허용하십시오.
3.1.1	버그 수정 및 개선 • 시간대 불일치로 인해 SigV4 요청이 실패하는 문제를 수정했습니다. • 변환 및 지표 속성이 다시 시작한 후 속성에 의존할 때 계산이 중지되는 문제를 수정했습니다. • 사용자 지정 Stream Manager 포트 구성 지원을 활성화합니다. • 엣지에 동기화된 속성의 업데이트가 중단될 수 있는 문제를 수정했습니다.

버전	변경
3.1.0	버그 수정 및 개선 • `ListAssetModels` API가 다음 토큰을 생성하지 못하는 문제를 수정했습니다.
3.0.0	새로운 기능 • MQTT 브로커의 데이터 통합을 지원할 수 있습니다.
2.2.1	버그 수정 및 개선 • 컨트롤 플레인 데이터 스토리지가 클라우드 운영 방식과 더 일관되도록 동기화 프로세스를 조정하세요. 이는 업그레이드에 약간의 영향을 미칩니다. **Note** 버전 2.2.1 이상에서 동기화된 컨트롤 플레인 데이터는 이전 버전과 호환되지 않습니다. 이전 버전으로 다운그레이드하려면 새로 설치해야 합니다. 이는 업그레이드에는 영향을 미치지 않으며, 이전 버전에서 동기화된 데이터는 버전 2.2.1에서 작동합니다. • 자격 증명의 우선 순위를 AWS IoT Greengrass V2 지정하도록 AWS 자격 증명 체인을 추가로 수정했습니다.
2.1.37	버그 수정 및 개선 • dependency-routing-service 프로세스를 더 이상 사용하지 않고 해당 기능을 프로세스로 이동하여 property-state-service 통신 프로세스의 리소스 사용량을 줄이십시오. • 에서 사용하는 한도와 일치하도록 `get-asset-property-value-history` API의 최대 결과 제한을 20,000으로 늘리십시오. AWS IoT SiteWise • 최대 결과 제한이 지정되지 않은 경우 `get-asset-property-value-history` API의 페이지 매김 결과에 다음 토큰이 제공되지 않던 문제를 수정했습니다.

버전	변경
2.1.35	버그 수정 및 개선 • AWS 자격 증명 체인을 수정하여 자격 증명의 우선 순위를 지정합니다 AWS IoT Greengrass . • 사물 그룹의 일부로 배포할 때 발생하는 계정 감지 문제를 수정합니다. AWS IoT
2.1.34	버그 수정 및 개선 • Linux에서 멀티스레딩을 사용하도록 메트릭/변환 계산을 조정합니다. Windows는 호환성을 위해 계속해서 단일 스레드 계산을 실행합니다. • 일부 계산 창에서 메트릭 계산이 누락되는 문제를 수정합니다.
2.1.33	버그 수정 및 개선 • Greengrass 콘솔에 대한 오류 상태 보고 관련 문제를 수정합니다.
2.1.32	버그 수정 및 개선 • 사용자 지정된 사용자 이름 및 그룹에 대한 지원을 추가합니다.
2.1.31	버그 수정 및 개선 • 에서 모델링된 데이터에 대한 시간 가중 평균 및 시간 가중 표준 편차를 계산할 수 있는 지원을 추가합니다. AWS IoT SiteWise
2.1.29	버그 수정 및 개선 • 에지 기능에 자산을 필터링하기 위한 지원을 추가합니다.
2.1.28	버그 수정 및 개선 • 많은 자산을 에지에서 에지로 동기화할 수 있도록 리소스 동기화를 최적화합니다. AWS 클라우드
2.1.24	버그 수정 및 개선 • 리소스를 두 번째로 동기화할 때 대시보드가 사라지던 문제를 수정합니다.

버전	변경
2.1.23	버그 수정 및 개선 • 인터넷 연결이 느린 경우 설치가 실패하지 않도록 설치 프로세스 제한 시간을 추가했습니다. `aws.iot.SiteWiseEdgeProcessor` • 리소스 동기화를 최적화하여 클라우드와 엣지 간의 동기화 효율성을 개선했습니다.
2.1.21	⚠ Warning 2.0.x에서 2.1.x로 업그레이드하면 로컬 데이터가 손실됩니다. 새로운 기능 • 윈도우 서버 2019 이상에 대한 지원을 추가합니다. • 리눅스 기반 운영 체제의 docker를 제거합니다.
2.0.16	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.
2.0.15	버그 수정 및 개선 • 이 구성 요소가 리소스 동기화 API 작업에 사용하는 포트를 8085에서 8087로 변경합니다. 따라서 이제 이 구성 요소를 사용하려면 포트 8087을 사용할 수 있어야 합니다. 이 구성 요소를 사용하려면 여전히 포트 8085가 필요합니다. • 사용자가 API 작업 호출을 시도하는 대신 로그인 중에 인증되지 않은 사용자를 거부하도록 AWS OpsHub 인증을 업데이트합니다.
2.0.14	이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.
2.0.13	버그 수정 및 개선 • 이 구성 요소가 Amazon CloudWatch 지표에 데이터를 보고할 때 이제 모델링되지 않은 데이터를 올바르게 표시하도록 문제를 수정합니다.

버전	변경
2.0.9	버그 수정 및 개선 • 코어 AWS IoT SiteWise 디바이스에서 리소스를 생성하고 업데이트할 수 있는 안정성을 개선합니다. • 코어 장치에 설치된 구성 요소, 각 구성 요소의 버전 및 각 구성 요소의 상태를 모니터링하는 데 사용할 수 있는 추가 로컬 API 작업을 추가합니다. 코어 장치의 AWS IoT SiteWise 응용 AWS OpsHub 프로그램에 있는 설정 탭에서 이 정보를 볼 수 있습니다. • 이 구성 요소가 실행하는 Docker 컨테이너의 상태를 추가합니다. `docker ps`명령을 실행하여 컨테이너의 상태를 볼 수 있습니다.
2.0.7	버그 수정 및 개선 • 코어 기기에서 AWS IoT SiteWise Monitor 포털 보기 지원을 수정했습니다.
2.0.6	버그 수정 및 개선 • 이 구성 요소가 코어 기기에서 계산하는 AWS IoT SiteWise `statetime()` `earliest()` , 및 `latest()` 함수를 수정합니다.
2.0.5	버그 수정 및 개선 • 이 구성 요소가 코어 기기에서 계산하는 변환의 AWS IoT SiteWise `pretrigger()` 함수에 대한 지원을 추가합니다. • 이 구성 요소가 인증을 위한 LDAP (경량 디렉터리 액세스 프로토콜) 구성을 저장하는 경로를 변경합니다.
2.0.2	초기 버전

다음 사항도 참조하십시오.

- 무엇입니까? AWS IoT SiteWise AWS IoT SiteWise 사용 설명서에서.

게시자 지원 구성 요소

퍼블리셔가 지원하는 구성 요소는 프리뷰 릴리스에 AWS IoT Greengrass 있으며 변경될 수 있습니다. 에서는 이러한 구성 요소를 지원하지 않습니다. AWS 각 구성 요소에 문제가 있는 경우 게시자에게 문의해야 합니다.

Greengrass Publisher가 지원하는 구성 요소는 타사 구성 요소 공급업체에서 개발, 제공 및 서비스합니다. 타사 구성 요소 공급업체는 AWS Partner 장치 카탈로그, AWS Heroes 또는 커뮤니티 공급업체의 공급업체입니다. 타사 구성 요소 공급업체에 직접 문의하여 이 카탈로그의 구성 요소를 구입할 수 있습니다.

Greengrass Publisher가 지원하는 구성 요소에는 다음이 포함됩니다.

주제

- AIShield.Edge
- AI 센서 EdgeLabs
- 그린그래스 S3 인제스터

AIShield.Edge

이 구성 요소는 보쉬에서 제공하는 AiShield에서 개발하고 지원합니다. AiShield.Edge를 사용하여 AI 보안을 강화하십시오. 이 구성 요소는 위협에 기반한 맞춤형 방어 체계를 엣지 디바이스에 원활하게 배포하여 AI 공격으로부터 디바이스를 보호하도록 설계되었습니다.

이 구성 요소는 다음과 같은 이점을 제공합니다.

- AiShield AI 보안을 통한 취약성 분석에서 내부 강화된 엣지 방어로 원활하게 전환할 수 있습니다. AWS
- 여러 에지 장치에 맞춤형 방어 수단을 쉽게 배포할 수 있습니다.
- 다양한 모델 유형 및 프레임워크를 지원하는 다양한 AI 설정에 맞게 조정된 광범위한 보호
- Greengrass 워크플로우와의 원활한 통합으로 Amazon SageMaker 최신 정보를 받아보세요.
- 다음으로 직접 전달되는 데이터를 통해 잠재적 위협에 대한 즉각적인 통찰력을 확보하십시오. AWS IoT Core
- 마켓플레이스의 AiShield AI 보안이 제공하는 엣지에서의 방어 배치를 위한 통합 AI 보안 경로 AWS

이 구성 요소는 다음 플랫폼에서 실행되어야 합니다.

- OS: Linux

이 구성 요소 구매에 관심이 있는 경우 보쉬 소프트웨어 및 디지털 솔루션 (<AIShield.Contact@bosch.com>) 으로 문의하십시오.

AI 센서 EdgeLabs

이 구성 요소는 AI가 개발하고 지원합니다 EdgeLabs. AI EdgeLabs 센서는 AI 기반 위협 탐지 및 예방 기능이 포함된 컨테이너 기반 애플리케이션입니다. AI 센서는 Greengrass 구성 요소에 포함되며 다른 Greengrass 구성 요소와 함께 코어 장치에 독립형 컨테이너로 배포됩니다.

현재 구성 요소는 네트워크 통신을 지속적으로 확인하고 Edge Host 또는 IoT 게이트웨이에서 실행되는 소프트웨어에서 위협 패턴을 찾는 컨테이너 기반 에이전트입니다. 이 구성 요소는 eBPF, 프로세스 대역폭의 동작 검증 및 호스트 기반 구성을 사용합니다. 이 구성 요소의 주요 기능은 NDR/IPS 및 EDR 기능을 기반으로 합니다.

이 구성 요소는 다음과 같은 이점을 제공합니다.

- 네트워크 공격 및 멀웨어에 대한 AI 기반 위협 탐지 (EDR/NDR)
- 자동화된 AI 기반 사고 대응 (IPS)
- 외부 데이터 전송을 최소화하는 호스트 로컬 위협 인텔리전스
- Docker 및 Greengrass를 사용한 가벼운 배포

이 구성 요소는 다음 플랫폼 중 하나에서 실행되어야 합니다.

- OS: Linux

이 구성 요소 구매에 관심이 있는 경우 AI EdgeLabs (<contact@edgelabs.ai>) 에 문의하십시오.

그린그래스 S3 인제스터

이 컴포넌트는 네이선 글로버가 개발하고 지원합니다. Greengrass S3 인제스터 구성 요소는 스트림 관리자 구성 요소와 함께 사용하도록 설계되었습니다. 이 구성 요소는 스트림 관리자에서 줄로 구분된 JSON 메시지 스트림을 가져와 GZIP 파일로 일괄 처리합니다. 이 구성 요소를 사용하면 추가 처리 또는 저장을 위해 Amazon S3로 데이터를 효율적으로 수집할 수 있습니다. 이 구성 요소는 실시간 데이터 전송을 지원하지 않습니다AWS 클라우드.

이 구성 요소는 다음 플랫폼 중 하나에서 실행되어야 합니다.

- OS: Linux
- OS: 윈도우

<# ## ## ### ### #### ### ### (nathan@glovers.id.au) ## ######.>

커뮤니티 구성 요소

Greengrass 소프트웨어 카탈로그는 Greengrass 커뮤니티에서 개발한 Greengrass 구성 요소의 색인입니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다. 카탈로그는 다음 링크에서 볼 수 있습니다: https://github.com/aws-greengrass/ aws-greengrass-software-catalog.

각 구성 요소에는 탐색할 수 있는 공용 GitHub 저장소가 있습니다. 커뮤니티 구성 요소의 전체 목록을 GitHub 찾으려면 Greengrass 소프트웨어 카탈로그를 참조하십시오. 예를 들어, 이 카탈로그에는 다음과 같은 구성 요소가 포함되어 있습니다.

- Amazon Kinesis Video Streams

  이 구성 요소는 실시간 스트리밍 프로토콜 (RTSP) 을 사용하는 로컬 카메라에서 오디오 및 비디오 스트림을 수집합니다. 그러면 구성 요소가 오디오 및 비디오 스트림을 Amazon Kinesis Video Streams에 업로드합니다.
- 블루투스 IoT 게이트웨이

  이 구성 요소는 블루투스 LE (저에너지) 기기와 통신할 수 있게 해주는 BluePy라이브러리를 사용하여 Bluetooth LE 클라이언트 인터페이스를 생성합니다.
- 인증서 로테이터

  이 구성 요소는 AWS IoT Greengrass 핵심 장치 인증서와 개인 키를 대규모로 플릿 전체에 순환시키는 수단을 제공합니다.
- 컨테이너식 보안 터널링

  이 구성 요소는 특정 호스트 운영 체제에 의존하지 않는 재사용 가능한 레시피로 모든 종속성 및 일치하는 라이브러리가 포함된 보안 터널링을 위한 Docker 컨테이너를 제공합니다.
- Grafana

이 구성 요소를 사용하면 Greengrass 코어 장치에서 Grafana 서버를 호스팅할 수 있습니다. Grafana 대시보드를 사용하여 코어 장치의 데이터를 시각화하고 관리할 수 있습니다.

- 아마존 룩아웃 포 비전용 GStremer

  이 구성 요소는 사용자 지정 GStreamer 파이프라인에서 Lookout for Vision 이상 탐지를 수행할 수 있도록 GStreamer 플러그인을 제공합니다.

- 홈 어시스턴트

  이 구성 요소를 통해 고객은 홈 어시스턴트를 사용하여 스마트 홈 장치를 로컬로 제어할 수 있습니다. 엣지 및 클라우드의 AWS 서비스와의 통합을 제공하여 홈 어시스턴트를 확장하는 홈 자동화 솔루션을 제공합니다.

- InfluxDBGrafana 대시보드

  이 구성 요소는 InfluxDB 및 Grafana 구성 요소를 설정하는 원클릭 환경을 제공합니다. InfluxDB를 Grafana에 연결하고 실시간으로 원격 측정을 렌더링하는 로컬 Grafana 대시보드 설정을 자동화합니다. AWS IoT Greengrass

- InfluxDB

  이 구성 요소는 Greengrass 코어 장치에 InfluxDB 시계열 데이터베이스를 제공합니다. 이 구성 요소를 사용하여 IoT 센서의 데이터를 처리하고, 실시간으로 데이터를 분석하고, 엣지에서 작업을 모니터링할 수 있습니다.

- InfluxDB 퍼블리셔

  이 구성 요소는 Nucleus 이미터 플러그인의 AWS IoT Greengrass 시스템 상태 원격 측정을 InfluxDB로 전달합니다. 또한 이 구성 요소는 사용자 지정 텔레메트리를 InfluxDB로 전달할 수 있습니다.

- IoT 퍼브서브 프레임워크

  이 프레임워크는 v2 사용자 지정 구성 요소를 사용하여 분산 이벤트 기반 IoT pubsub 애플리케이션의 코드 품질을 개선하는 데 도움이 되는 애플리케이션 아키텍처, 템플릿 코드 및 배포 가능한 예제를 제공합니다. AWS IoT Greengrass 자세한 설명은 AWS IoT Greengrass 구성 요소 생성 섹션을 참조하세요.

- 주피터 랩스

  이 구성 요소는 코어 JupyterLab 디바이스에 배포됩니다. AWS IoT Greengrass Jupyter 환경은 에서 설정한 프로세스 및 환경 변수 리소스에 액세스할 수 있으므로 Python으로 AWS IoT Greengrass 작성된 구성 요소를 테스트하고 개발하는 프로세스가 간소화됩니다.

- 로컬 웹 서버

이 구성 요소를 사용하면 Greengrass 코어 장치에서 로컬 웹 사용자 인터페이스를 만들 수 있습니다. 예를 들어 디바이스 및 애플리케이션 설정을 구성하거나 디바이스를 모니터링할 수 있는 로컬 웹 사용자 인터페이스를 생성할 수 있습니다.

- LoRaWaN 프로토콜 어댑터

  이 구성 요소는 저전력 광역 네트워크 (LPWAN) 프로토콜인 LoRaWa N 프로토콜을 사용하는 로컬 무선 장치에서 데이터를 수집합니다. 구성 요소를 사용하면 클라우드와 통신하지 않고도 로컬에서 데이터를 분석하고 조치를 취할 수 있습니다.

- 모드버스 TCP

  이 구성 요소는 ModbustCP 프로토콜을 사용하여 로컬 장치에서 데이터를 수집하여 선택한 데이터 스트림에 게시합니다.

- 노드 레드

  이 구성 요소는 NPM을 사용하여 코어 디바이스에 Node-RED를 AWS IoT Greengrass 설치합니다. 구성 요소는 명시적으로 배포 및 구성해야 하는 Node-RED 인증 구성 요소에 따라 달라집니다. Greengrass용 Node-RED CLI를 사용하여 Node-RED 플로우를 디바이스에 배포할 수 있습니다. AWS IoT Greengrass

- 노드-레드 도커

  이 구성 요소는 공식 Node-RED 도커 컨테이너를 사용하여 AWS IoT Greengrass 코어 디바이스에 Node-RED를 설치합니다. 구성 요소는 명시적으로 배포 및 구성해야 하는 Node-Red 인증 구성 요소에 따라 달라집니다. Greengrass용 Node-RED CLI를 사용하여 Node-RED 플로우를 디바이스에 배포할 수 있습니다. AWS IoT Greengrass

- 노드-레드 인증

  이 구성 요소는 코어 기기에서 실행되는 Node-RED 인스턴스를 보호하기 위해 사용자 이름과 암호를 구성합니다. AWS IoT Greengrass

- OpenThread경계 라우터

  이 컴포넌트는 OpenThread 보더 라우터 도커 컨테이너를 배포합니다. 컴포넌트는 스레드 보더 라우터가 포함된 Matter 디바이스를 구성하는 데 도움이 됩니다.

- OSI Pi 스트리밍 데이터 커넥터

  이 구성 요소는 OSI Pi 데이터 아카이브에서 최신 데이터 아키텍처로의 스트리밍 실시간 데이터 통합을 제공합니다. AWS 메시징을 통해 중앙에서 관리되는 OSI Pi 자산 프레임워크에 통합됩니다. AWS IoT PubSub

- PostgreSQL DB

  이 구성 요소는 엣지에서 PostgreSQL 관계형 데이터베이스를 지원합니다. 고객은 이 구성 요소를 사용하여 docker 컨테이너 내에서 로컬 PostgreSQL 인스턴스를 프로비저닝하고 관리할 수 있습니다.

- S3 파일 업로더

  이 구성 요소는 디렉터리에서 새 파일을 모니터링하여 Amazon Simple Storage Service (Amazon S3) 에 업로드한 다음 업로드가 성공적으로 완료되면 파일을 삭제합니다.

- 시크릿 매니저 클라이언트

  이 구성 요소는 레시피 수명 주기 스크립트의 Secrets Manager 구성 요소에서 암호를 검색해야 하는 다른 구성 요소에서 사용할 수 있는 CLI 도구를 제공합니다.

- 컨테이너로의 TES 라우팅

  이 구성 요소는 컨테이너와 함께 구성 요소를 사용할 수 있도록 AWS IoT Greengrass 장치에 nftables 또는 iptables를 구성합니다. 토큰 교환 서비스

- WebRTC

  이 구성 요소는 코어 장치에 연결된 RTSP 카메라의 오디오 및 비디오 스트림을 수집합니다. AWS IoT Greengrass 그런 다음 구성 요소는 Amazon Kinesis Video Streams를 통해 오디오 및 비디오 스트림을 peer-to-peer 통신 또는 릴레이로 전환합니다.

기능을 요청하거나 버그를 보고하려면 해당 구성 요소의 리포지토리에서 GitHub 이슈를 여십시오. AWS커뮤니티 구성 요소를 지원하지 않습니다. 자세한 내용은 각 구성 요소의 저장소에 있는 CONTRIBUTING.md파일을 참조하십시오.

AWS제공된 일부 구성 요소도 오픈 소스입니다. 자세한 내용은 오픈소스 AWS IoT Greengrass 코어 소프트웨어을(를) 참조하세요.

AWS IoT Greengrass개발 도구

AWS IoT Greengrass개발 도구를 사용하여 사용자 지정 Greengrass 구성 요소를 만들고, 테스트하고, 빌드하고, 게시하고 배포할 수 있습니다.

- 그린그래스 개발 키트 CLI

로컬 개발 환경에서 AWS IoT Greengrass 개발 키트 명령줄 인터페이스 (GDK CLI) 를 사용하여 Greengrass 소프트웨어 카탈로그의 템플릿과 커뮤니티 구성 요소로 구성 요소를 만들 수 있습니다. GDK CLI를 사용하여 구성 요소를 빌드하고 구성 요소를 서비스에 개인 구성 요소로 게시할 AWS IoT Greengrass 수 있습니다. AWS 계정

- 그린그래스 커맨드 라인 인터페이스

  그린그래스 코어 디바이스에서 그린그래스 명령줄 인터페이스 (Greengrass CLI) 를 사용하여 그린그래스 구성 요소를 배포하고 디버깅할 수 있습니다. Greengrass CLI는 코어 디바이스에 배포하여 로컬 배포를 생성하고, 설치된 구성 요소에 대한 세부 정보를 보고, 로그 파일을 탐색할 수 있는 구성 요소입니다.

- 로컬 디버그 콘솔

  Greengrass 코어 디바이스의 로컬 디버그 콘솔을 사용하여 로컬 대시보드 웹 인터페이스를 사용하여 Greengrass 구성 요소를 배포하고 디버깅할 수 있습니다. 로컬 디버그 콘솔은 코어 디바이스에 배포하여 로컬 배포를 생성하고 설치된 구성 요소에 대한 세부 정보를 볼 수 있는 구성 요소입니다.

AWS IoT Greengrass또한 사용자 지정 Greengrass 구성 요소에서 사용할 수 있는 다음과 같은 SDK를 제공합니다.

- 에는 AWS IoT Device SDK 프로세스 간 통신 (IPC) 라이브러리가 포함되어 있습니다. 자세한 설명은 AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core 섹션을 참조하세요.
- 스트림 관리자 SDK는 데이터 스트림을 로 전송하는 데 사용할 수 있습니다. AWS 클라우드 자세한 내용은 Greengrass 코어 디바이스의 데이터 스트림 관리을(를) 참조하세요.

주제

- AWS IoT Greengrass개발 키트 명령줄 인터페이스
- 그린그래스 커맨드 라인 인터페이스
- AWS IoT Greengrass 테스트 프레임워크 사용

AWS IoT Greengrass개발 키트 명령줄 인터페이스

AWS IoT Greengrass개발 키트 명령줄 인터페이스 (GDK CLI) 는 사용자 지정 Greengrass 구성 요소를 개발하는 데 도움이 되는 기능을 제공합니다. GDK CLI를 사용하여 사용자 지정 구성 요소를 만들고 빌드하고 게시할 수 있습니다. GDK CLI로 구성 요소 리포지토리를 생성할 때 Greengrass 소프트웨어

카탈로그의 템플릿 또는 커뮤니티 구성 요소에서 시작할 수 있습니다. 그런 다음 파일을 ZIP 아카이브로 패키징하거나, Maven 또는 Gradle 빌드 스크립트를 사용하거나, 사용자 지정 빌드 명령을 실행하는 빌드 시스템을 선택할 수 있습니다. 구성 요소를 생성한 후 GDK CLI를 사용하여 서비스에 게시할 수 있으며, 콘솔 또는 API를 사용하여 AWS IoT Greengrass Greengrass 코어 장치에 구성 요소를 배포할 수 있습니다. AWS IoT Greengrass

GDK CLI를 사용하지 않고 Greengrass 구성 요소를 개발하는 경우 구성 요소의 새 버전을 생성할 때마다 구성 요소 레시피 파일의 버전 및 아티팩트 URI를 업데이트해야 합니다. GDK CLI를 사용하면 구성 요소의 새 버전을 게시할 때마다 버전 및 아티팩트 URI가 자동으로 업데이트될 수 있습니다.

GDK CLI는 오픈 소스이며 에서 사용할 수 있습니다. GitHub 구성 요소 개발 요구 사항에 맞게 GDK CLI를 사용자 지정하고 확장할 수 있습니다. 리포지토리에서 이슈를 열고 요청을 가져오도록 초대합니다 GitHub . GDK CLI 소스는 다음 링크에서 찾을 수 있습니다. https://github.com/aws-greengrass/aws-greengrass-gdk-cli

사전 조건

Greengrass 개발 키트 CLI를 설치하고 사용하려면 다음이 필요합니다.

- AWS 계정. 계정이 없는 경우 AWS 계정 설정 섹션을 참조하십시오.
- 인터넷에 연결된 윈도우, macOS 또는 유닉스 계열 개발 컴퓨터.
- GDK CLI 버전 1.1.0 이상의 경우 개발 컴퓨터에 Python 3.6 이상이 설치되어 있어야 합니다.

  GDK CLI 버전 1.0.0의 경우 개발 컴퓨터에 Python 3.8 이상이 설치되어 있어야 합니다.
- Git이 개발 컴퓨터에 설치되었습니다.
- AWS Command Line Interface(AWS CLI) 개발 컴퓨터에 자격 증명을 사용하여 설치 및 구성되었습니다. 자세한 내용은 AWS Command Line Interface사용 설명서의 설치, 업데이트, 제거 AWS CLI 및 구성을 참조하십시오. AWS CLI

  > **Note**
  >
  > 라즈베리 파이 또는 다른 32비트 ARM 디바이스를 사용하는 경우 V1을 설치하세요. AWS CLI AWS CLI V2는 32비트 ARM 디바이스에서 사용할 수 없습니다. 자세한 내용은 버전 1 설치, 업데이트 및 제거를 AWS CLI 참조하십시오.
- GDK CLI를 사용하여 구성 요소를 서비스에 게시하려면 다음 권한이 있어야 합니다. AWS IoT Greengrass
  - `s3:CreateBucket`

  - `s3:GetBucketLocation`
  - `s3:PutObject`
  - `greengrass:CreateComponentVersion`
  - `greengrass:ListComponentVersions`
- GDK CLI를 사용하여 로컬 파일 시스템이 아닌 S3 버킷에 아티팩트가 있는 구성 요소를 빌드하려면 다음 권한이 있어야 합니다.
  - `s3:ListBucket`

  이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

Changelog

다음 표는 GDK CLI의 각 버전에서 변경된 내용을 설명합니다. 자세한 내용은 의 GDK CLI 릴리스 페이지를 참조하십시오. GitHub

버전	변경
1.6.2	버그 수정 및 개선 • 상대 경로로 인해 Windows gradlew.bat 가 작동하지 않는 문제를 수정합니다. • 로깅, 테스트 및 패키징이 약간 개선되었습니다.
1.6.1	버그 수정 및 개선 • CLI 인수 구문 분석에 대한 보안 수정을 추가합니다. • GDK가 최신 그린그래스 테스팅 프레임워크 (GTF) 릴리스 이름을 기본 GTF 버전으로 가져올 수 있도록 합니다. • GDK는 GTF의 이전 버전을 사용하는 고객에게 최신 버전으로 업데이트하도록 추천할 수 있습니다.
1.6.0	새로운 기능 • `component build`및 `component publish` 명령 중에 Greengrass 레시피 스키마에 대한 레시피 검증 검사를 추가합니다. 이 업데이트를 통해 개발자는 구성 요소 생성 프로세스 초기에 구성 요소 레시피 내에서 실행 가능한 문제를 식별할 수 있습니다.

버전	변경
	• 명령으로 풀다운할 수 있는 신뢰도 테스트 세트를 템플릿에 추가합니다. `test-e2e init` 이 신뢰도 테스트 스위트에는 기본 구성 요소 테스트 요구 사항에 맞게 사용 및 확장할 수 있는 8개의 일반 테스트가 포함되어 있습니다. 버그 수정 및 개선 • `test-e2e`명령에서 사용하는 기본 Greengrass 테스트 프레임워크 (GTF) 버전을 버전 1.2.0으로 업데이트합니다.
1.5.0	버그 수정 및 개선 인 경우 `excludes` 빌드 옵션에서 인식되는 패턴을 `zip` 업데이트합니다. `build_system` 이제 이 버전에서는 와일드카드 문자를 기반으로 경로 이름과 일치하는 글로브 패턴을 인식합니다. 이를 통해 제외할 디렉터리를 사용자 지정할 수 있습니다.
1.4.0	새로운 기능 • 기존 GDK 구성 파일 내의 필드를 수정하기 위한 대화형 프롬프트를 시작하는 새 `config` 명령을 추가합니다. • 계속하기 전에 `gdk component build` 및 `gdk component publish` 명령을 수정하여 레시피 크기가 Greengrass 요구 사항 (<=16000바이트) 내에 있는지 확인합니다. 버그 수정 및 개선 • 레시피 구문 오류로 인해 빌드가 인식되지 않는 경우 `gdk component build` 명령 출력에 추가 로깅을 추가합니다. • 오픈 테스트 프레임워크의 `otf-options` 이름이 Greengrass 테스팅 프레임워크로 변경됨에 따라 `gtf-options` 및 의 이름을 `gtf-version` 각각, 로 변경합니다. `otf-version`

버전	변경
1.3.0	새로운 기능 • 오픈 테스트 프레임워크를 사용한 구성 요소 end-to-end 테스트를 지원하는 새 `test-e2e` 명령을 추가합니다. • zip 빌드 시스템에서 구성 가능한 zip 파일 이름을 지원하는 새 구성 옵션을 추가합니다. `zip_name` • GDK 구성 파일의 `region` 속성을 선택 사항으로 만듭니다. 버그 수정 및 개선 • 인수를 사용하여 GDK 프로젝트를 초기화할 때 지정된 템플릿 또는 리포지토리가 없더라도 새 디렉터리가 생성되는 문제를 수정합니다. `--name`
1.2.3	버그 수정 및 개선 • 잘못된 오류 처리로 인해 버킷 생성이 실패하는 문제를 수정합니다. • 구성 요소 레시피의 목록 구조가 제거되는 문제를 수정합니다.
1.2.2	버그 수정 및 개선 • 레시피 키는 더 이상 대소문자를 구분하지 않습니다. • 새 버킷을 만들기 전에 버킷이 에 AWS 리전 존재하고 사용자가 액세스할 수 있는지 확인하는 검사를 추가합니다. 사용자에게 `GetBucketLocation` 권한이 있어야 합니다. • GDK CLI `excludes` 구성 파일의 키워드 문제를 수정합니다.
1.2.1	버그 수정 및 개선 • `gdk-config.json` 파일의 지역 구성 AWS 리전 항목에서 캐나다 (중부`ca-central-1`) () 를 수락합니다. • 명령에 대한 `--region` GDK CLI 인수 관련 문제를 수정합니다`publish`.

버전	변경
1.2.0	새로운 기능 • GDK CLI `build` 구성 파일의 컨피그레이션에 `options` 항목을 추가합니다. 빌드 시스템을 사용할 때 zip 아티팩트에서 특정 파일을 `options` 제외하도록 `excludes` under를 지원합니다. `zip` • Gradle Wrapper를 사용하여 구성 요소를 빌드하도록 `gradlew` 빌드 시스템을 추가합니다. • 빌드 옵션에 Kotlin DSL 빌드 파일 지원을 추가합니다. `gradle` • GDK CLI `publish` 구성 파일의 컨피그레이션에 `options` 항목을 추가합니다. Amazon `options` S3에 파일을 업로드할 때 추가 인수를 제공하도록 `file_upload_args` under를 지원합니다. 버그 수정 및 개선 • 빌드 명령어를 실행하기 전에 Gradle 빌드가 정리되지 않던 문제를 수정합니다. • 빌드 명령이 실패했을 때 빌드가 종료되지 않는 문제를 수정합니다. • `gdk component list`명령의 출력 형식을 개선합니다.

버전	변경
1.1.0	새로운 기능 • Gradle 빌드 시스템에 대한 지원을 추가합니다. • Windows 디바이스에 Maven 빌드 시스템에 대한 지원을 추가합니다. • 구성 요소 게시 명령에 `--bucket` 인수를 추가합니다. 이 인수를 사용하여 GDK CLI가 구성 요소 아티팩트를 업로드하는 정확한 버킷을 지정할 수 있습니다. • 구성 요소 init `--name` 명령에 인수를 추가합니다. 이 옵션을 사용하여 GDK CLI가 구성 요소를 초기화하는 폴더를 지정할 수 있습니다. • S3 버킷에는 있지만 로컬 구성 요소 빌드 폴더에는 없는 구성 요소 아티팩트에 대한 지원을 추가합니다. 이 기능을 사용하여 기계 학습 모델과 같은 대형 구성 요소 아티팩트의 대역폭 비용을 줄일 수 있습니다. 버그 수정 및 개선 • 구성 요소를 게시하기 전에 구성 요소가 빌드되었는지 확인하기 위해 구성 요소 게시 명령을 업데이트합니다. 구성 요소가 빌드되지 않은 경우 이제 이 명령으로 구성 요소가 자동으로 빌드됩니다. • ZIP 파일 이름에 대문자가 포함된 경우 Windows 디바이스에서 zip 빌드 시스템이 빌드되지 않는 문제를 수정합니다. • 로그 메시지 형식을 개선하고 3.8 이전 버전의 Python을 실행하는 `INFO` 장치에서 기본 로그 수준을 로 변경합니다. • 최소 파이썬 버전 요구 사항을 Python 3.6으로 변경합니다.
1.0.0	초기 버전

AWS IoT Greengrass개발 키트 명령줄 인터페이스 설치 또는 업데이트

AWS IoT Greengrass개발 키트 명령줄 인터페이스 (GDK CLI) 는 Python을 기반으로 구축되었으므로 개발 컴퓨터에 설치하는 데 `pip` 사용할 수 있습니다.

> **Tip**
>
> venv와 같은 Python 가상 환경에도 GDK CLI를 설치할 수 있습니다. 자세한 내용은 Python 3 설명서의 가상 환경 및 패키지를 참조하십시오.

GDK CLI를 설치 또는 업데이트하려면

1. 다음 명령을 실행하여 리포지토리에서 GitHub 최신 버전의 GDK CLI를 설치합니다.

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

> **Note**
>
> 특정 버전의 GDK CLI를 설치하려면 versionTag를 설치할 버전 태그로 *#####*. 리포지토리에서 GitHub GDK CLI의 버전 태그를 볼 수 있습니다.
>
> ```
> python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
> ```

2. 다음 명령을 실행하여 GDK CLI가 성공적으로 설치되었는지 확인합니다.

```
gdk --help
```

gdk명령을 찾을 수 없는 경우 해당 폴더를 PATH에 추가합니다.

- Linux 디바이스에서는 /home/*MyUser*/.local/bin PATH에 추가하고 사용자 *MyUser*이름으로 바꾸십시오.
- Windows 장치에서는 *PythonPath*\\Scripts PATH에 추가하고 장치의 Python 폴더 *PythonPath*경로로 바꾸십시오.

이제 GDK CLI를 사용하여 Greengrass 구성 요소를 생성, 빌드 및 게시할 수 있습니다. GDK CLI를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass개발 키트 명령줄 인터페이스 명령

AWS IoT Greengrass개발 키트 명령줄 인터페이스 명령

AWS IoT Greengrass개발 키트 명령줄 인터페이스 (GDK CLI) 는 개발 컴퓨터에서 Greengrass 구성 요소를 생성, 빌드 및 게시하는 데 사용할 수 있는 명령줄 인터페이스를 제공합니다. GDK CLI 명령은 다음 형식을 사용합니다.

```
gdk <command> <subcommand> [arguments]
```

GDK CLI를 설치하면 설치 프로그램이 PATH에 gdk 추가되므로 명령줄에서 GDK CLI를 실행할 수 있습니다.

모든 명령에 다음 인수를 사용할 수 있습니다.

- GDK CLI 명령에 `--help` 대한 정보에는 `-h` 또는 를 사용하십시오.
- `-v`또는 `--version` 를 사용하여 설치된 GDK CLI의 버전을 확인할 수 있습니다.
- `-d`또는 `--debug` 를 사용하여 GDK CLI를 디버깅하는 데 사용할 수 있는 자세한 로그를 출력합니다.

이 섹션에서는 GDK CLI 명령에 대해 설명하고 각 명령에 대한 예를 제공합니다. 각 명령의 개요에는 해당 인수와 사용법이 나와 있습니다. 선택적 인수는 대괄호 안에 표시됩니다.

가용 명령

- 구성 요소
- config
- 테스트-e2e

구성 요소

AWS IoT Greengrass개발 키트 `component` 명령줄 인터페이스 (GDK CLI) 의 명령을 사용하여 사용자 지정 Greengrass 구성 요소를 생성, 빌드 및 게시할 수 있습니다.

하위 명령

- init
- build
- publish
- list

init

구성 요소 템플릿 또는 커뮤니티 구성 요소에서 Greengrass 구성 요소 폴더를 초기화합니다.

GDK CLI는 Greengrass 소프트웨어 카탈로그에서 커뮤니티 구성 요소를 검색하고 의 구성 요소 템플릿 저장소에서 AWS IoT Greengrass 구성 요소 템플릿을 검색합니다. GitHub

Note

GDK CLI v1.0.0을 사용하는 경우 빈 폴더에서 이 명령을 실행해야 합니다. GDK CLI는 템플릿 또는 커뮤니티 구성 요소를 현재 폴더에 다운로드합니다.
GDK CLI v1.1.0 이상을 사용하는 경우 인수를 지정하여 GDK `--name` CLI가 템플릿 또는 커뮤니티 구성 요소를 다운로드하는 폴더를 지정할 수 있습니다. 이 인수를 사용하는 경우 존재하지 않는 폴더를 지정하십시오. GDK CLI는 폴더를 자동으로 생성합니다. 이 인수를 지정하지 않으면 GDK CLI는 비어 있어야 하는 현재 폴더를 사용합니다.
구성 요소가 zip 빌드 시스템을 사용하는 경우 GDK CLI는 구성 요소 폴더의 특정 파일을 구성 요소 폴더와 동일한 이름의 zip 파일로 압축합니다. 예를 들어 구성 요소 폴더 이름이 인 경우 GDK CLI는 `HelloWorld` 라는 zip 파일을 생성합니다. `HelloWorld.zip` 구성 요소 레시피에서 zip 아티팩트 이름은 구성 요소 폴더의 이름과 일치해야 합니다. Windows 디바이스에서 GDK CLI 버전 1.0.0을 사용하는 경우 구성 요소 폴더 및 zip 파일 이름에는 소문자만 포함되어야 합니다.
zip 빌드 시스템을 사용하는 템플릿 또는 커뮤니티 구성 요소를 템플릿 또는 구성 요소와 다른 이름의 폴더로 초기화하는 경우 구성 요소 레시피에서 zip 아티팩트 이름을 변경해야 합니다. zip 파일 이름이 구성 요소 폴더의 이름과 일치하도록 `Artifacts` 및 `Lifecycle` 정의를 업데이트하십시오. 다음 예제에서는 `Artifacts` 및 `Lifecycle` 정의의 zip 파일 이름을 강조 표시합니다.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
```

```
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

시놉시스

```
$ gdk component init
    [--language]
    [--template]
    [--repository]
    [--name]
```

인수 (구성 요소 템플릿에서 초기화)

- `-l`, `--language` — 지정한 템플릿에 사용할 프로그래밍 언어입니다.

  `--repository`또는 `--language` 및 중 하나를 지정해야 `--template` 합니다.
- `-t`, `--template` — 로컬 구성 요소 프로젝트에 사용할 구성 요소 템플릿입니다. 사용 가능한 템플릿을 보려면 list 명령을 사용합니다.

  `--repository`또는 `--language` 및 중 하나를 지정해야 `--template` 합니다.
- `-n`, `--name` — (선택 사항) GDK CLI가 구성 요소를 초기화하는 로컬 폴더의 이름입니다. 존재하지 않는 폴더를 지정하십시오. GDK CLI는 폴더를 자동으로 생성합니다.

  이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

인수 (커뮤니티 구성 요소에서 초기화)

- `-r, --repository` — 로컬 폴더로 체크아웃할 커뮤니티 구성 요소입니다. 사용 가능한 커뮤니티 구성 요소를 보려면 list 명령을 사용합니다.

  `--repository`또는 `--language` 및 중 하나를 지정해야 `--template` 합니다.
- `-n, --name` — (선택 사항) GDK CLI가 구성 요소를 초기화하는 로컬 폴더의 이름입니다. 존재하지 않는 폴더를 지정하십시오. GDK CLI는 폴더를 자동으로 생성합니다.

  이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

출력

다음 예제는 이 명령을 실행하여 Python Hello World 템플릿에서 구성 요소 폴더를 초기화할 때 생성되는 출력을 보여줍니다.

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
 component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
 from Greengrass Software Catalog.
```

다음 예제는 이 명령을 실행하여 커뮤니티 구성 요소에서 구성 요소 폴더를 초기화할 때 생성되는 출력을 보여줍니다.

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
 from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

구성 요소 소스를 레시피와 아티팩트로 빌드하여 서비스에 게시할 수 있습니다. AWS IoT Greengrass GDK CLI는 GDK CLI 구성 파일에 지정된 빌드 시스템을 실행합니다. `gdk-config.json` 파일이 있는 동일한 폴더에서 이 명령을 실행해야 합니다. `gdk-config.json`

이 명령을 실행하면 GDK CLI가 구성 요소 폴더의 폴더에 레시피와 아티팩트를 생성합니다. `greengrass-build` GDK CLI는 레시피를 폴더에 `greengrass-build/recipes` 저장하고 아티팩트를 폴더에 저장합니다. `greengrass-build/artifacts/`*`componentName`*`/`*`componentVersion`*

GDK CLI v1.1.0 이상을 사용하는 경우 구성 요소 레시피는 S3 버킷에는 있지만 로컬 구성 요소 빌드 폴더에는 없는 아티팩트를 지정할 수 있습니다. 이 기능을 사용하면 기계 학습 모델과 같이 아티팩트가 큰 구성 요소를 개발할 때 대역폭 사용량을 줄일 수 있습니다.

구성 요소를 빌드한 후 다음 중 하나를 수행하여 Greengrass 코어 기기에서 테스트할 수 있습니다.

- AWS IoT GreengrassCore 소프트웨어를 실행하는 장치가 아닌 다른 장치에서 개발하는 경우 구성 요소를 게시하여 Greengrass 코어 장치에 배포해야 합니다. 구성 요소를 AWS IoT Greengrass 서비스에 게시하고 Greengrass 코어 장치에 배포합니다. 자세한 내용은 publish 명령 및 배포 만들기 을 참조하십시오.
- AWS IoT GreengrassCore 소프트웨어를 실행하는 동일한 장치에서 개발하는 경우 구성 요소를 AWS IoT Greengrass 서비스에 게시하여 배포하거나 로컬 배포를 만들어 구성 요소를 설치하고 실행할 수 있습니다. 로컬 배포를 생성하려면 Greengrass CLI를 사용하십시오. 자세한 내용은 그린그래스 커맨드 라인 인터페이스 및 로컬 배포를 통한 테스트 AWS IoT Greengrass 구성 요소 섹션을 참조하세요. 로컬 배포를 생성할 때 레시피 폴더와 아티팩트 `greengrass-build/recipes` `greengrass-build/artifacts` 폴더로 지정하십시오.

시놉시스

```
$ gdk component build
```

인수

None

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
```

```
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

publish

이 구성 요소를 AWS IoT Greengrass 서비스에 게시합니다. 이 명령은 빌드 아티팩트를 S3 버킷에 업로드하고, 레시피의 아티팩트 URI를 업데이트하고, 레시피에서 새 버전의 구성 요소를 생성합니다. GDK CLI는 GDK CLI 구성 파일에 지정된 S3 AWS 버킷과 지역을 사용합니다. `gdk-config.json` 파일이 있는 동일한 폴더에서 이 명령을 실행해야 합니다. `gdk-config.json`

GDK CLI v1.1.0 이상을 사용하는 경우 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 `--bucket` 수 있습니다. 이 인수를 지정하지 않으면 GDK *`bucket-region-accountId`* CLI는 이름이 인 S3 버킷에 업로드됩니다. *`### ###`* 지역은 지정된 `gdk-config.json` 값이고 *`accountID#`* ID입니다. AWS 계정 버킷이 없는 경우 GDK CLI에서 버킷을 생성합니다.

GDK CLI v1.2.0 이상을 사용하는 경우 매개변수를 사용하여 GDK CLI 구성 AWS 리전 파일에 지정된 내용을 재정의할 수 있습니다. `--region` 매개변수를 사용하여 추가 옵션을 지정할 수도 있습니다. `--options` 사용 가능한 옵션 목록은 을 참조하십시오그린그래스 개발 키트 CLI 구성 파일.

이 명령을 실행하면 GDK CLI가 레시피에 지정한 버전으로 구성요소를 게시합니다. 지정하는 `NEXT_PATCH` 경우 GDK CLI는 아직 존재하지 않는 다음 패치 버전을 사용합니다. 시맨틱 버전은 메이저를 사용합니다. 마이너. 패치 넘버링 시스템. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

> **Note**
>
> GDK CLI v1.1.0 이상을 사용하는 경우 이 명령을 실행하면 GDK CLI에서 구성 요소가 빌드되었는지 확인합니다. 구성 요소가 빌드되지 않은 경우 GDK CLI는 구성 요소를 게시하기 전에 구성 요소를 빌드합니다.

시놉시스

```
$ gdk component publish
    [--bucket] [--region] [--options]
```

인수

- `-b, --bucket` — (선택 사항) GDK CLI가 구성 요소 아티팩트를 게시하는 S3 버킷의 이름을 지정합니다.

  이 인수를 지정하지 않으면 GDK *`bucket`*-*`region`*-*`accountId`* CLI는 이름이 인 S3 버킷에 업로드됩니다. *`###`* *`###`* 지역은 지정된 `gdk-config.json` 값이고 *`accountID#`* ID입니다. AWS 계정 버킷이 없는 경우 GDK CLI에서 버킷을 생성합니다.

  버킷이 없는 경우 GDK CLI에서 버킷을 생성합니다.

  이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.
- `-r, --region` — (선택 사항) 구성 요소 생성 시 대상 이름을 지정합니다. AWS 리전 이 인수는 GDK CLI 컨피그레이션의 지역 이름을 대체합니다.

  이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.
- `-o, --options` (선택 사항) 구성 요소를 게시하기 위한 옵션 목록을 지정합니다. 인수는 유효한 JSON 문자열이거나 게시 옵션이 포함된 JSON 파일의 파일 경로여야 합니다. 이 인수는 GDK CLI 컨피그레이션의 옵션을 재정의합니다.

  이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:45:30] INFO - No private version of the component
 'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
 version to create.
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
 If this is your first time using this bucket, add the 's3:GetObject' permission
 to each core device's token exchange role to allow it to download the component
 artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developerguide/device-service-role.html.
```

```
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
 com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
 com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
 account.'com.example.PythonHelloWorld'.
```

list

사용 가능한 구성 요소 템플릿 및 커뮤니티 구성 요소 목록을 검색하십시오.

GDK CLI는 Greengrass 소프트웨어 카탈로그에서 커뮤니티 구성 요소를 검색하고 의 구성 요소 템플릿 저장소에서 AWS IoT Greengrass 구성 요소 템플릿을 검색합니다. GitHub

이 명령의 출력을 init 명령에 전달하여 템플릿 및 커뮤니티 구성 요소에서 구성 요소 리포지토리를 초기화할 수 있습니다.

시놉시스

```
$ gdk component list
    [--template]
    [--repository]
```

인수

- `-t`, `--template` — (선택 사항) 사용 가능한 구성 요소 템플릿을 나열하려면 이 인수를 지정합니다. 이 명령은 각 템플릿의 이름과 언어를 형식으로 *`name`*`-`*`language`* 출력합니다. 예를 들어`HelloWorld-python`, 에서 템플릿 이름은 `HelloWorld` 이고 언어는 입니다`python`.
- `-r`, `--repository` — (선택 사항) 사용 가능한 커뮤니티 구성 요소 저장소를 나열하려면 이 인수를 지정합니다.

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
 Greengrass Software Catalog.
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.
1. HelloWorld-python
2. HelloWorld-java
```

config

AWS IoT Greengrass개발 키트 `config` 명령줄 인터페이스 (GDK CLI) 의 명령을 사용하여 구성 파일에서 GDK의 구성을 수정합니다. `gdk-config.json`

하위 명령

- update

update

대화형 프롬프트를 시작하여 기존 GDK 구성 파일 내의 필드를 수정합니다.

시놉시스

```
$ gdk config update
    [--component]
```

인수

- `-c`, `--component` — 파일의 구성 요소 관련 필드를 업데이트합니다. `gdk-config.json` 이 인수는 유일한 옵션이므로 필수입니다.

출력

다음 예제는 이 명령을 실행하여 구성 요소를 구성할 때 생성되는 출력을 보여줍니다.

```
$ gdk config update --component
Current value of the REQUIRED component_name is (default:
 com.example.PythonHelloWorld):
Current value of the REQUIRED author is (default: author):
Current value of the REQUIRED version is (default: NEXT_PATCH):
Do you want to change the build configurations? (y/n)
Do you want to change the publish configurations? (y/n)
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

테스트-e2e

AWS IoT Greengrass개발 키트 `test-e2e` 명령줄 인터페이스 (GDK CLI) 의 명령을 사용하여 GDK 프로젝트에서 테스트 모듈을 초기화, 빌드 및 end-to-end 실행합니다.

하위 명령

- init
- build
- run

init

그린그래스 테스팅 프레임워크 (GTF) 를 사용하는 테스트 모듈로 기존 GDK CLI 프로젝트를 초기화합니다.

기본적으로 GDK CLI는 의 구성 요소 템플릿 저장소에서 maven 모듈 템플릿을 AWS IoT Greengrass 검색합니다. GitHub 이 maven 모듈에는 JAR 파일에 대한 종속성이 있습니다. `aws-greengrass-testing-standalone`

이 명령은 GDK 프로젝트 `gg-e2e-tests` 내부에 라는 새 디렉토리를 만듭니다. 테스트 모듈 디렉터리가 이미 존재하고 비어 있지 않은 경우 명령어는 아무 작업도 하지 않고 종료됩니다. 이 `gg-e2e-tests` 폴더에는 maven 프로젝트로 구성된 Cucumber 기능과 단계 정의가 들어 있습니다.

기본적으로 이 명령은 GTF의 최신 릴리스 버전을 사용하려고 시도합니다.

시놉시스

```
$ gdk test-e2e init
    [--gtf-version]
```

인수

- `-ov, --gtf-version` — (선택 사항) GDK 프로젝트의 end-to-end 테스트 모듈과 함께 사용할 GTF 버전입니다. 이 값은 릴리스의 GTF 버전 중 하나여야 합니다. 이 인수는 GDK CLI `gtf_version` 컨피그레이션의 를 재정의합니다.

출력

다음 예제는 이 명령을 실행하여 테스트 모듈로 GDK 프로젝트를 초기화할 때 생성되는 출력을 보여줍니다.

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
 1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
 gg-e2e-tests directory...
```

build

> **Note**
>
> end-to-end 테스트 모듈을 gdk component build 빌드하기 전에 를 실행하여 구성 요소를 빌드해야 합니다.

end-to-end 테스트 모듈을 빌드하세요. GDK CLI는 속성 아래의 GDK CLI 구성 파일에 gdk-config.json 지정된 빌드 시스템을 사용하여 테스트 모듈을 빌드합니다. test-e2e 파일이 있는 동일한 폴더에서 이 명령을 실행해야 합니다. gdk-config.json

기본적으로 GDK CLI는 maven 빌드 시스템을 사용하여 테스트 모듈을 빌드합니다. 명령을 실행하려면 Maven이 필요합니다. gdk test-e2e build

테스트 기능 파일에 GDK_COMPONENT_NAME 및 와 같은 보간 변수가 있는 경우 테스트 모듈을 gdk-component-build 빌드하기 전에 를 실행하여 구성 요소를 GDK_COMPONENT_RECIPE_FILE 빌드해야 합니다.

이 명령을 실행하면 GDK CLI가 GDK 프로젝트 구성의 모든 변수를 보간하고 모듈을 빌드하여 최종 테스트 gg-e2e-tests JAR 파일을 생성합니다.

시놉시스

```
$ gdk test-e2e build
```

인수

None

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
```

```
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.........
```

run

GDK 구성 파일의 테스트 옵션을 사용하여 테스트 모듈을 실행합니다.

> **Note**
>
> end-to-end 테스트를 gdk test-e2e build 실행하기 전에 를 실행하여 테스트 모듈을 빌드해야 합니다.

시놉시스

```
$ gdk test-e2e run
    [--gtf-options]
```

인수

- `-oo, --gtf-options` — (선택 사항) end-to-end 테스트 실행 옵션 목록을 지정합니다. 인수는 유효한 JSON 문자열이거나 GTF 옵션이 포함된 JSON 파일의 파일 경로여야 합니다. 구성 파일에 제공된 옵션은 명령 인수에 제공된 옵션과 병합됩니다. 옵션이 두 위치에 모두 있는 경우 인수에 있는 옵션이 구성 파일에 있는 옵션보다 우선합니다.

  이 명령에 `tags` 옵션이 지정되지 않은 경우 GDK는 태그를 사용합니다`Sample`. 지정되지 않은 경우, `ggc-archive` GDK는 최신 버전의 Greengrass 핵 아카이브를 다운로드합니다.

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ gdk test-e2e run
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar –ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip –
tags=Sample
```

```
16:35:59.693 [] [] [] [INFO]
 com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
 com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.......
```

그린그래스 개발 키트 CLI 구성 파일

AWS IoT Greengrass개발 키트 명령줄 인터페이스 (GDK CLI) 는 `gdk-config.json` 이름이 지정된 구성 파일을 읽고 구성 요소를 빌드하고 게시합니다. 이 구성 파일은 구성 요소 리포지토리의 루트에 있어야 합니다. GDK CLI init 명령을 사용하여 이 구성 파일로 구성 요소 저장소를 초기화할 수 있습니다.

주제

- GDK CLI 구성 파일 형식
- GDK CLI 구성 파일 예제

GDK CLI 구성 파일 형식

구성 요소에 대한 GDK CLI 구성 파일을 정의할 때 JSON 형식으로 다음 정보를 지정합니다.

`gdk_version`

이 구성 요소와 호환되는 GDK CLI의 최소 버전입니다. 이 값은 릴리스의 GDK CLI 버전 중 하나여야 합니다.

`component`

이 구성 요소의 구성.

`componentName`

`author`

구성 요소의 작성자 또는 게시자.

`version`

구성 요소의 버전입니다. 다음 중 하나를 지정하십시오.

- `NEXT_PATCH`— 이 옵션을 선택하면 구성 요소를 게시할 때 GDK CLI에서 버전을 설정합니다. GDK CLI는 서비스를 AWS IoT Greengrass 쿼리하여 가장 최근에 게시된 구성 요소 버전을 식별합니다. 그런 다음 버전을 해당 버전 이후의 다음 패치 버전으로 설정합니다. 이전에 구성 요소를 게시한 적이 없는 경우 GDK CLI는 버전을 사용합니다. `1.0.0`

  이 옵션을 선택하면 Greengrass CLI를 사용하여 Core 소프트웨어를 실행하는 로컬 개발 컴퓨터에 구성 요소를 로컬로 배포하고 테스트할 수 없습니다. AWS IoT Greengrass 로컬 배포를 활성화하려면 시맨틱 버전을 대신 지정해야 합니다.

- 시맨틱 버전 (예: **`1.0.0`** 시맨틱 버전은 메이저를 사용합니다. 마이너. 패치 넘버링 시스템. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

  구성 요소를 배포하고 테스트하려는 Greengrass 코어 장치에서 구성 요소를 개발하는 경우 이 옵션을 선택합니다. Greengrass CLI로 로컬 배포를 생성하려면 특정 버전으로 구성 요소를 빌드해야 합니다.

`build`

이 컴포넌트의 소스를 아티팩트로 빌드하는 데 사용할 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`build_system`

사용할 빌드 시스템. 다음 옵션 중 하나를 선택합니다.

- `zip`— 구성 요소의 폴더를 ZIP 파일로 패키징하여 구성 요소의 유일한 아티팩트로 정의합니다. 다음 유형의 구성 요소에 대해 이 옵션을 선택합니다.
  - Python 또는 JavaScript 같은 해석된 프로그래밍 언어를 사용하는 컴포넌트
  - 코드가 아닌 파일을 패키징하는 구성 요소 (예: 기계 학습 모델 또는 기타 리소스).

  GDK CLI는 구성 요소 폴더를 구성 요소 폴더와 동일한 이름의 zip 파일로 압축합니다. 예를 들어 구성 요소 폴더 이름이 인 경우 GDK CLI는 `HelloWorld` 라는 zip 파일을 생성합니다. `HelloWorld.zip`

  > **Note**
  >
  > Windows 디바이스에서 GDK CLI 버전 1.0.0을 사용하는 경우 구성 요소 폴더 및 zip 파일 이름에는 소문자만 포함되어야 합니다.

  GDK CLI는 구성 요소 폴더를 zip 파일로 압축할 때 다음 파일을 건너뛰습니다.

  - `gdk-config.json` 파일
  - 레시피 파일 (또는) `recipe.json` `recipe.yaml`
  - 빌드 폴더 (예: `greengrass-build`
- `maven`— `mvn clean package` 명령을 실행하여 구성 요소 소스를 아티팩트에 빌드합니다. Java 구성 요소와 같이 Maven을 사용하는 구성 요소에 대해 이 옵션을 선택합니다.

  Windows 디바이스에서 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.
- `gradle`— `gradle build` 명령을 실행하여 구성 요소의 소스를 아티팩트로 빌드합니다. Gradle을 사용하는 구성 요소에 대해 이 옵션을 선택합니다. 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

  `gradle`빌드 시스템은 Kotlin DSL을 빌드 파일로 지원합니다. 이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.
- `gradlew`— `gradlew` 명령을 실행하여 구성 요소 소스를 아티팩트로 빌드합니다. Gradle 래퍼를 사용하는 구성 요소에 대해 이 옵션을 선택합니다.

  이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.
- `custom`— 사용자 지정 명령을 실행하여 구성 요소의 소스를 레시피와 아티팩트로 빌드합니다. `custom_build_command`파라미터에 사용자 지정 명령을 지정합니다.

`custom_build_command`

(선택 사항) 사용자 지정 빌드 시스템에서 실행할 사용자 지정 빌드 명령입니다. 를 지정하는 경우 이 파라미터를 `custom` 지정해야 `build_system` 합니다.

> **Important**
>
> 이 명령은 구성 요소 폴더 내의 다음 폴더에 레시피와 아티팩트를 생성해야 합니다. GDK CLI는 구성 요소 빌드 명령을 실행할 때 이러한 폴더를 자동으로 생성합니다.
>
> - 레시피 폴더: `greengrass-build/recipes`
> - 아티팩트 폴더: `greengrass-build/artifacts/componentName/componentVersion`
>
>   *ComponentName# ## ## ####* 바꾸고 ComponentVersion을 *## ## ## ## #* 로 바꿉니다. `NEXT_PATCH`

단일 문자열 또는 문자열 목록을 지정할 수 있습니다. 여기서 각 문자열은 명령의 한 단어입니다. 예를 들어 C++ 구성 요소에 대한 사용자 지정 빌드 명령을 실행하려면 **`cmake --build build --config Release`** 또는 **`["cmake", "--build", "build", "--config", "Release"]`** 를 지정할 수 있습니다.

사용자 지정 빌드 시스템의 예를 보려면 aws.greengrass.labs.LocalWebServer community component GitHubon을 참조하십시오.

`options`

(선택 사항) 구성 요소 빌드 프로세스 중에 사용되는 추가 구성 옵션.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

`excludes`

zip 파일을 빌드할 때 구성 요소 디렉터리에서 제외할 파일을 정의하는 글로브 패턴 목록입니다. 인 경우에만 유효합니다. `build_system` `zip`

> **Note**
>
> GDK CLI 버전 1.4.0 이하에서는 제외 목록의 항목과 일치하는 모든 파일이 구성 요소의 모든 하위 디렉토리에서 제외됩니다. GDK CLI 버전 1.5.0 이상에서 동일한 동작을 수행하려면 제외 목록의 `**/` 기존 항목 앞에 추가하십시오. 예를 들어, `*.txt` 는 디렉터리에서만 텍스트 파일을 제외하고, 모든 디렉터리 및 하위 디렉터리에서 텍스트 파일을 제외합니다. `**/*.txt`
> GDK CLI 버전 1.5.0 이상에서는 구성 요소 빌드 `excludes` 중에 GDK 구성 파일에 정의된 경우 경고가 표시될 수 있습니다. 이 경고를 비활성화하려면 환경 변수를 로 설정합니다. `GDK_EXCLUDES_WARN_IGNORE` `true`

GDK CLI는 항상 zip 파일에서 다음 파일을 제외합니다.

- `gdk-config.json` 파일
- 레시피 파일 (또는) `recipe.json` `recipe.yaml`
- 빌드 폴더 (예: `greengrass-build`

다음 파일은 기본적으로 제외됩니다. 그러나 `excludes` 옵션을 사용하여 이러한 파일 중 제외할 파일을 제어할 수 있습니다.

- 접두사 “test" () `test*` 로 시작하는 모든 폴더
- 모든 숨김 파일

- node_modules 폴더

excludes옵션을 지정하는 경우 GDK CLI는 옵션으로 설정한 파일만 제외합니다. excludes excludes옵션을 지정하지 않으면 GDK CLI는 앞서 언급한 기본 파일 및 폴더를 제외합니다.

zip_name

빌드 프로세스 중에 zip 아티팩트를 만들 때 사용할 zip 파일 이름입니다. 인 경우에만 유효합니다. build_system zip build_system가 비어 있는 경우 컴포넌트 이름이 zip 파일 이름으로 사용됩니다.

publish

이 구성 요소를 AWS IoT Greengrass 서비스에 게시하는 데 사용할 구성입니다.

GDK CLI v1.1.0 이상을 사용하는 경우 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 --bucket 수 있습니다. 이 인수를 지정하지 않으면 GDK *bucket*-*region*-*accountId* CLI는 이름이 인 S3 버킷에 업로드됩니다. *### ###* 지역은 지정된 gdk-config.json 값이고 *accountID#* ID입니다. AWS 계정 버킷이 없는 경우 GDK CLI에서 버킷을 생성합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

bucket

구성 요소 아티팩트를 호스팅하는 데 사용할 S3 버킷 이름.

region

GDK CLI가 이 구성 요소를 게시하는 AWS 리전 곳입니다.

GDK CLI v1.3.0 이상을 사용하는 경우 이 속성은 선택 사항입니다.

options

(선택 사항) 구성 요소 버전 생성 중에 사용되는 추가 구성 옵션.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

file_upload_args

파일을 버킷에 업로드하는 동안 Amazon S3로 전송된 인수 (예: 메타데이터 및 암호화 메커니즘) 를 포함하는 JSON 구조입니다. 허용된 인수 목록은 Boto3 설명서의 S3Transfer클래스를 참조하십시오. .

`test-e2e`

(선택 사항) 구성 요소를 end-to-end 테스트하는 동안 사용할 구성입니다. 이 기능은 GDK CLI v1.3.0 이상에서 사용할 수 있습니다.

`build`

`build_system`— 사용할 빌드 시스템. 기본 옵션은 입니다`maven`. 다음 옵션 중 하나를 선택합니다.

- `maven`— `mvn package` 명령을 실행하여 테스트 모듈을 빌드합니다. Maven을 사용하는 테스트 모듈을 빌드하려면 이 옵션을 선택하십시오.
- `gradle`— `gradle build` 명령을 실행하여 테스트 모듈을 빌드합니다. Gradle을 사용하는 테스트 모듈에 대해 이 옵션을 선택합니다.

`gtf_version`

(선택 사항) GTF로 GDK 프로젝트를 초기화할 때 테스트 모듈의 종속 항목으로 사용할 Greengrass end-to-end 테스트 프레임워크 (GTF) 버전입니다. 이 값은 릴리스의 GTF 버전 중 하나여야 합니다. 기본값은 GTF 버전 1.1.0입니다.

`gtf_options`

(선택 사항) 구성 요소 end-to-end 테스트 중에 사용되는 추가 구성 옵션.

다음 목록에는 GTF 버전 1.1.0에서 사용할 수 있는 옵션이 포함되어 있습니다.

- `additional-plugins`— (선택 사항) 추가 오이 플러그인
- `aws-region`— AWS 서비스의 특정 지역 엔드포인트를 대상으로 합니다. AWSSDK가 발견한 내용을 기본값으로 설정합니다.
- `credentials-path`— 선택적 AWS 프로필 자격 증명 경로. 호스트 환경에서 검색된 자격 증명이 기본값입니다.
- `credentials-path-rotation`— AWS 자격 증명의 선택적 순환 기간. 기본값은 15분 또는 `PT15M` 입니다.
- `csr-path`— 디바이스 인증서를 생성하는 데 사용할 CSR의 경로입니다.
- `device-mode`— 테스트 중인 대상 장치. 기본값은 로컬 장치입니다.
- `env-stage`— Greengrass의 배포 환경을 대상으로 합니다. 기본값은 프로덕션입니다.
- `existing-device-cert-arn`— Greengrass의 디바이스 인증서로 사용하려는 기존 인증서의 arn.
- `feature-path`— 추가 기능 파일이 들어 있는 파일 또는 디렉터리. 기본값은 추가 기능 파일이 사용되지 않는 것입니다.

- `gg-cli-version`— 그린그래스 CLI의 버전을 재정의합니다. 에 있는 값이 기본값입니다. `ggc.version`
- `gg-component-bucket`— Greengrass 구성 요소가 들어 있는 기존 Amazon S3 버킷의 이름.
- `gg-component-overrides`— Greengrass 컴포넌트 오버라이드 목록입니다.
- `gg-persist`— 테스트 실행 후 유지되는 테스트 요소 목록. 기본 동작은 아무것도 지속하지 않는 것입니다. 허용되는 값은: `aws.resources`, `installed.software`, 및 `generated.files` 입니다.
- `gg-runtime`— 테스트가 테스트 리소스와 상호 작용하는 방식에 영향을 미치는 값 목록입니다. 이 값이 파라미터를 대체합니다. `gg.persist` 기본값이 비어 있으면 설치된 Greengrass 런타임을 포함하여 모든 테스트 리소스가 테스트 케이스별로 관리된다고 가정합니다. 허용되는 값은`aws.resources`, `installed.software` 및 입니다. `generated.files`
- `ggc-archive`— 보관된 Greengrass 핵 구성 요소에 대한 경로.
- `ggc-install-root`— Greengrass 핵 구성 요소를 설치하기 위한 디렉토리입니다. 기본값은 test.temp.path 및 테스트 실행 폴더입니다.
- `ggc-log-level`— 테스트 실행을 위한 Greengrass 핵 로그 레벨을 설정합니다. 기본값은 “정보”입니다.
- `ggc-tes-rolename`— AWS IoT Greengrass Core가 AWS 서비스에 액세스하기 위해 맡게 되는 IAM 역할입니다. 지정된 이름을 가진 역할이 존재하지 않는 경우 역할이 생성되고 기본 액세스 정책이 적용됩니다.
- `ggc-trusted-plugins`— Greengrass에 추가해야 하는 신뢰할 수 있는 플러그인의 호스트 경로를 쉼표로 구분한 목록입니다. DUT 자체의 경로를 제공하려면 경로 앞에 'dut: '를 접두사로 붙입니다.
- `ggc-user-name`— 그린그래스 핵에 대한 사용자:그룹 POSIX사용자 값입니다. 로그인되어 있는 현재 사용자 이름이 기본값입니다.
- `ggc-version`— 실행 중인 Greengrass 핵 구성 요소의 버전을 재정의합니다. ggc.archive에 있는 값이 기본값입니다.
- `log-level`— 테스트 실행의 로그 수준. 기본값은 “INFO”입니다.
- `parallel-config`— 배치 인덱스 및 배치 수를 JSON 문자열로 설정합니다. 배치 인덱스의 기본값은 0이고 배치 수는 1입니다.
- `proxy-url`— 이 URL을 통해 트래픽을 라우팅하도록 모든 테스트를 구성합니다.
- `tags`— 기능 태그만 실행합니다. '&'와 교차할 수 있습니다.

- `test-id-prefix`— AWS 리소스 이름 및 태그를 포함하여 모든 테스트 관련 리소스에 적용되는 공통 접두사입니다. 기본값은 “gg” 접두사입니다.
- `test-log-path`— 전체 테스트 실행 결과를 포함할 디렉터리입니다. 기본값은 “TestResults”입니다.
- `test-results-json`— 결과 Cucumber JSON 보고서가 디스크에 기록되어 생성되는지 여부를 결정하는 플래그입니다. 기본값은 true입니다.
- `test-results-log`— 콘솔 출력이 생성되어 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 false입니다.
- `test-results-xml`— 결과 JUnit XML 보고서가 생성되어 디스크에 기록되는지 여부를 결정하는 플래그를 지정합니다. 기본값은 true입니다.
- `test-temp-path`— 로컬 테스트 아티팩트를 생성하기 위한 디렉터리입니다. gg-testing 접두사가 붙은 임의의 임시 디렉토리가 기본값입니다.
- `timeout-multiplier`— 모든 테스트 타임아웃에 멀티플라이어가 제공됩니다. 기본값은 1.0.

GDK CLI 구성 파일 예제

다음 GDK CLI 구성 파일 예제를 참조하여 Greengrass 구성 요소 환경을 구성하는 데 도움이 될 수 있습니다.

헬로 월드 (Python)

다음 GDK CLI 구성 파일은 Python 스크립트를 실행하는 Hello World 구성 요소를 지원합니다. 이 구성 파일은 `zip` 빌드 시스템을 사용하여 컴포넌트의 Python 스크립트를 GDK CLI가 아티팩트로 업로드하는 ZIP 파일로 패키징합니다.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
           "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
```

```
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
    },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

헬로 월드 (자바)

다음 GDK CLI 구성 파일은 자바 애플리케이션을 실행하는 Hello World 구성 요소를 지원합니다. 이 구성 파일은 maven 빌드 시스템을 사용하여 구성 요소의 Java 소스 코드를 GDK CLI가 아티팩트로 업로드하는 JAR 파일로 패키징합니다.

```
{
  "component": {
    "com.example.JavaHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
```

```
                "some-key": "some-value"
              }
            }
          }
        }
    },
    "test-e2e":{
      "build":{
          "build_system": "maven"
      },
      "gtf_version": "1.1.0",
      "gtf_options": {
           "tags": "Sample"
       }
    },
    "gdk_version": "1.6.1"
    }
}
```

커뮤니티 구성 요소

Greengrass 소프트웨어 카탈로그의 여러 커뮤니티 구성 요소는 GDK CLI를 사용합니다. 이러한 구성 요소의 리포지토리에서 GDK CLI 구성 파일을 탐색할 수 있습니다.

커뮤니티 구성 요소의 GDK CLI 구성 파일을 보려면

1. 다음 명령을 실행하여 GDK CLI를 사용하는 커뮤니티 구성 요소를 나열합니다.

   ```
   gdk component list --repository
   ```

   응답에는 GDK CLI를 사용하는 각 커뮤니티 구성 요소의 GitHub 저장소 이름이 나열됩니다. 각 리포지토리는 조직 내에 존재합니다. awslabs

   ```
   [2022-02-22 17:27:31] INFO - Listing all the available component repositories from
    Greengrass Software Catalog.
   [2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
   1. aws-greengrass-labs-database-influxdb
   2. aws-greengrass-labs-telemetry-influxdbpublisher
   3. aws-greengrass-labs-dashboard-grafana
   4. aws-greengrass-labs-dashboard-influxdb-grafana
   5. aws-greengrass-labs-local-web-server
   6. aws-greengrass-labs-lookoutvision-gstreamer
   ```

2. 다음 URL에서 커뮤니티 구성 요소의 GitHub 저장소를 엽니다. 이전 단계의 커뮤니티 구성 요소 *community-component-name*이름으로 바꾸십시오.

   ```
   https://github.com/awslabs/community-component-name
   ```

그린그래스 커맨드 라인 인터페이스

Greengrass 명령줄 인터페이스 (CLI) 를 사용하면 디바이스의 AWS IoT Greengrass Core와 상호 작용하여 로컬에서 구성 요소를 개발하고 문제를 디버깅할 수 있습니다. 예를 들어 Greengrass CLI를 사용하여 로컬 배포를 생성하고 코어 디바이스에서 구성 요소를 다시 시작할 수 있습니다.

그린그래스 CLI 구성 요소 (`aws.greengrass.Cli`) 를 배포하여 그린그래스 CLI를 코어 디바이스에 설치합니다.

> **Important**
>
> 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소를 사용하면 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 액세스할 수 있습니다. 이 구성 요소를 필요한 핵심 장치에만 배포하여 최소 권한 원칙을 따르세요.

주제

- 그린그래스 CLI 설치
- 그린그래스 CLI 명령

그린그래스 CLI 설치

다음 방법 중 하나로 Greengrass CLI를 설치할 수 있습니다.

- 디바이스에 AWS IoT Greengrass Core 소프트웨어를 처음 설치할 때 `--deploy-dev-tools` 인수를 사용하십시오. 또한 이 인수를 `--provision true` 적용하도록 지정해야 합니다.
- Greengrass CLI 구성 요소 (`aws.greengrass.Cli`) 를 디바이스에 배포합니다.

이 섹션에서는 Greengrass CLI 구성 요소를 배포하는 단계를 설명합니다. 초기 설정 시 Greengrass CLI를 설치하는 방법에 대한 자세한 내용은 을 참조하십시오. 자습서: AWS IoT Greengrass V2 시작하기

필수 조건

Greengrass CLI 구성 요소를 배포하려면 다음 요구 사항을 충족해야 합니다.

- AWS IoT Greengrass 코어 디바이스에 코어 소프트웨어가 설치 및 구성되었습니다. 자세한 설명은 자습서: AWS IoT Greengrass V2 시작하기 섹션을 참조하세요.
- 를 사용하여 Greengrass CLI를 AWS CLI 배포하려면 를 설치하고 구성해야 합니다. AWS CLI 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 구성을 참조하십시오.
- 코어 소프트웨어와 상호 작용하려면 Greengrass CLI를 사용할 권한이 있어야 합니다. AWS IoT Greengrass Greengrass CLI를 사용하려면 다음 중 하나를 수행하십시오.
  - AWS IoT Greengrass Core 소프트웨어를 실행하는 시스템 사용자를 사용하십시오.
  - 루트 또는 관리자 권한이 있는 사용자를 사용하십시오. Linux 코어 디바이스에서는 를 사용하여 루트 sudo 권한을 얻을 수 있습니다.
  - 구성 요소를 배포할 때 `AuthorizedPosixGroups` 또는 `AuthorizedWindowsGroups` 구성 매개 변수에 지정한 그룹에 속한 시스템 사용자를 사용하십시오. 자세한 내용은 Greengrass CLI 구성 요소 구성을 참조하십시오.

그린그래스 CLI 구성 요소 배포

Greengrass CLI 구성 요소를 코어 디바이스에 배포하려면 다음 단계를 완료하세요.

그린그래스 CLI 구성 요소를 배포하려면 (콘솔)

1. AWS IoT Greengrass 콘솔에 로그인합니다.
2. 탐색 메뉴에서 구성 요소를 선택합니다.
3. 구성 요소 페이지의 퍼블릭 구성 요소 탭에서 `aws.greengrass.Cli`을(를) 선택합니다.
4. aws.greengrass.Cli 페이지에서 배포를 선택합니다.
5. 배포에 추가에서 새 배포 생성을 선택합니다.
6. 대상 지정 페이지의 배포 대상 아래에 있는 대상 이름 목록에서 배포하려는 Greengrass 그룹을 선택하고 다음을 선택합니다.
7. 구성 요소 선택 페이지에서 구성 aws.greengrass.Cli요소가 선택되었는지 확인하고 다음을 선택합니다.
8. 구성 요소 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
10. 검토 페이지에서 배포를 클릭합니다.

그린그래스 CLI 구성 요소를 배포하려면 () AWS CLI

1. 디바이스에서 `deployment.json` 파일을 생성하여 Greengrass CLI 구성 요소의 배포 구성을 정의합니다. 이 파일은 다음과 같아야 합니다.

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.2",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\":\"<group1>,<group2>,...,<groupN>\",
\"AuthorizedWindowsGroups\":\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

- `target` 필드에서 *targetArn*을(를) 다음 형식으로 배포 대상으로 지정할 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)으로 바꿉니다.
  - 사물: `arn:aws:iot:`*region*`:`*account-id*`:thing/`*thingName*
  - 사물 그룹: `arn:aws:iot:`*region*`:`*account-id*`:thinggroup/`*thingGroupName*
- `aws.greengrass.Cli`구성 요소 개체에서 다음과 같이 값을 지정합니다.

  `version`

  그린그래스 CLI 컴포넌트의 버전입니다.

  `configurationUpdate.AuthorizedPosixGroups`

  (선택 사항) 쉼표로 구분된 시스템 그룹 목록을 포함하는 문자열입니다. 이러한 시스템 그룹이 Greengrass CLI를 사용하여 Core 소프트웨어와 상호 작용할 AWS IoT Greengrass 수 있도록 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어, `group1,1002,group3` 는 세 개의 시스템 그룹 (`group1`,`1002`, 및`group3`) 이 Greengrass CLI를 사용할 수 있도록 승인합니다.

  승인할 그룹을 지정하지 않은 경우 Greengrass CLI를 루트 sudo 사용자 () 또는 Core 소프트웨어를 실행하는 AWS IoT Greengrass 시스템 사용자로 사용할 수 있습니다.

`configurationUpdate.AuthorizedWindowsGroups`

(선택 사항) 쉼표로 구분된 시스템 그룹 목록을 포함하는 문자열입니다. 이러한 시스템 그룹이 Greengrass CLI를 사용하여 Core 소프트웨어와 상호 작용할 AWS IoT Greengrass 수 있도록 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어, `group1,1002,group3` 는 세 개의 시스템 그룹 (`group1`,`1002`, 및`group3`) 이 Greengrass CLI를 사용할 수 있도록 승인합니다.

승인할 그룹을 지정하지 않은 경우 관리자 또는 Core 소프트웨어를 실행하는 시스템 사용자로 Greengrass CLI를 사용할 수 있습니다. AWS IoT Greengrass

2. 다음 명령을 실행하여 Greengrass CLI 구성 요소를 디바이스에 배포합니다.

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/
to/deployment.json
```

설치 중에 구성 요소가 디바이스의 */greengrass/v2*/bin 폴더에 심볼릭 링크를 추가하고 이 `greengrass-cli` 경로에서 Greengrass CLI를 실행합니다. 절대 경로 없이 Greengrass CLI를 실행하려면 */greengrass/v2*/bin 폴더를 PATH 변수에 추가하십시오. Greengrass CLI 설치를 확인하려면 다음 명령을 실행합니다.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

다음 결과가 표시됩니다.

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

      --ggcRootPath=<ggcRootPath>
                  The AWS IoT Greengrass V2 root directory.
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
```

```
  help                Show help information for a command.
  component           Retrieve component information and stop or restart
                        components.
  deployment          Create local deployments and retrieve deployment status.
  logs                Analyze Greengrass logs.
  get-debug-password  Generate a password for use with the HTTP debug view
                        component.
```

찾을 수 `greengrass-cli` 없는 경우 배포 시 Greengrass CLI를 설치하지 못한 것일 수 있습니다. 자세한 내용은 문제 해결 AWS IoT Greengrass V2을(를) 참조하세요.

그린그래스 CLI 명령

Greengrass CLI는 코어 디바이스와 로컬에서 상호 작용할 수 있는 명령줄 인터페이스를 제공합니다. AWS IoT Greengrass Greengrass CLI 명령은 다음 형식을 사용합니다.

```
$ greengrass-cli <command> <subcommand> [arguments]
```

기본적으로 폴더의 `greengrass-cli` 실행 파일은 */greengrass/v2*/bin/ 폴더에서 실행 중인 AWS IoT Greengrass Core 소프트웨어 버전과 상호 작용합니다. */greengrass/v2* 이 위치에 있지 않은 실행 파일을 호출하거나 다른 위치에서 AWS IoT Greengrass Core 소프트웨어와 상호 작용하려는 경우 다음 방법 중 하나를 사용하여 상호 작용하려는 AWS IoT Greengrass Core 소프트웨어의 루트 경로를 명시적으로 지정해야 합니다.

- `GGC_ROOT_PATH` 환경 변수를 */greengrass/v2*로 설정합니다.
- 다음 예와 같이 명령에 `--ggcRootPath` */greengrass/v2* 인수를 추가합니다.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

모든 명령에 다음 인수를 사용할 수 있습니다.

- 특정 Greengrass CLI 명령에 `--help` 대한 정보를 제공하는 데 사용합니다.
- Greengrass CLI 버전에 `--version` 대한 정보를 보려면 사용하십시오.

이 섹션에서는 Greengrass CLI 명령에 대해 설명하고 이러한 명령에 대한 예를 제공합니다. 각 명령의 개요에는 해당 인수와 사용법이 나와 있습니다. 선택적 인수는 대괄호 안에 표시됩니다.

가용 명령

- 구성 요소
- 배포
- 로그
- get-debug-password

구성 요소

component명령을 사용하여 코어 기기의 로컬 구성 요소와 상호 작용할 수 있습니다.

하위 명령

- 세부 정보
- list
- 를 다시 시작합니다
- stop

세부 정보

한 구성 요소의 버전, 상태 및 구성을 검색합니다.

시놉시스

```
greengrass-cli component details --name <component-name>
```

인수

--name,-n. 구성 요소 이름.

출력

다음 예제에서는 이 명령을 실행할 때 생성되는 출력을 보여 줍니다.

```
$ sudo greengrass-cli component details --name MyComponent

Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

list

디바이스에 설치된 각 구성 요소의 이름, 버전, 상태 및 구성을 검색합니다.

시놉시스

```
greengrass-cli component list
```

인수

없음

출력

다음 예제에서는 이 명령을 실행할 때 생성되는 출력을 보여 줍니다.

```
$ sudo greengrass-cli component list

Components currently running in Greengrass:
Component Name: FleetStatusService
Version: 0.0.0
State: RUNNING
Configuration: {"periodicUpdateIntervalSec":86400.0}
Component Name: UpdateSystemPolicyService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Nucleus
Version: 2.0.0
State: FINISHED
Configuration: {"awsRegion":"region","runWithDefault":
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}
Component Name: DeploymentService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
```

```
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

를 다시 시작합니다

구성 요소를 다시 시작합니다.

시놉시스

```
greengrass-cli component restart --names <component-name>,...
```

인수

--names,-n. 구성 요소 이름. 구성 요소 이름이 하나 이상 필요합니다. 각 이름을 쉼표로 구분하여 추가 구성 요소 이름을 지정할 수 있습니다.

출력

없음

stop

구성 요소 실행을 중지합니다.

시놉시스

```
greengrass-cli component stop --names <component-name>,...
```

인수

--names,-n. 구성 요소 이름. 구성 요소 이름이 하나 이상 필요합니다. 필요한 경우 각 이름을 쉼표로 구분하여 추가 구성 요소 이름을 지정할 수 있습니다.

출력

없음

배포

deployment명령을 사용하여 코어 디바이스의 로컬 구성 요소와 상호 작용할 수 있습니다.

로컬 배포 진행 상황을 모니터링하려면 status 하위 명령을 사용합니다. 콘솔을 사용하여 로컬 배포 진행 상황을 모니터링할 수 없습니다.

하위 명령

- create
- 취소
- list
- status

create

지정된 구성 요소 레시피, 아티팩트 및 런타임 인수를 사용하여 로컬 배포를 만들거나 업데이트합니다.

시놉시스

```
greengrass-cli deployment create
    --recipeDir path/to/component/recipe
    [--artifactDir path/to/artifact/folder ]
    [--update-config {component-configuration}]
    [--groupId <thing-group>]
    [--merge "<component-name>=<component-version>"]...
    [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"]...
    [--systemLimits "{component-system-resource-limits}]"]...
    [--remove <component-name>,...]
    [--failure-handling-policy <policy name[ROLLBACK, DO_NOTHING]>]
```

인수

- `--recipeDir,-r`. 구성 요소 레시피 파일이 들어 있는 폴더의 전체 경로입니다.
- `--artifactDir,-a`. 배포에 포함하려는 아티팩트 파일이 들어 있는 폴더의 전체 경로입니다. 아티팩트 폴더에는 다음과 같은 디렉터리 구조가 포함되어야 합니다.

  ```
  /path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
  ```

- `--update-config,-c`. 배포를 위한 구성 인수로, JSON 문자열 또는 JSON 파일로 제공됩니다. JSON 문자열은 다음 형식이어야 합니다.

  ```
  { \
    "componentName": { \
      "MERGE": {"config-key": "config-value"}, \
      "RESET": ["path/to/reset/"] \
    } \
  ```

```
}
```

MERGE및 대소문자를 RESET 구분하며 대문자여야 합니다.

- --groupId,. -g 배포를 위한 대상 사물 그룹입니다.
- --merge,-m. 추가 또는 업데이트하려는 대상 구성 요소의 이름 및 버전 구성 요소 정보를 형식으로 제공해야 *<component>*=*<version>* 합니다. 지정할 각 추가 구성 요소에 대해 별도의 인수를 사용하십시오. 필요한 경우 --runWith 인수를 사용하여 구성 요소 실행에 필요한 posixUserposixGroup, 및 windowsUser 정보를 제공하십시오.
- --runWith. 제네릭 또는 Lambda 구성 요소를 실행하기 위한 posixUserposixGroup, 및 windowsUser 정보입니다. 이 정보는 다음 형식으로 제공해야 합니다. *<component>*:{posixUser|windowsUser}=*<user>*[:*<=posixGroup>*] 예를 들어, **HelloWorld:posixUser=ggc_user:ggc_group** 또는 를 지정할 수 **HelloWorld:windowsUser=ggc_user** 있습니다. 지정할 각 추가 옵션에 대해 별도의 인수를 사용하십시오.

  자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.
- --systemLimits. 코어 디바이스의 일반 및 비컨테이너식 Lambda 구성 요소 프로세스에 적용할 시스템 리소스 제한 각 구성 요소의 프로세스가 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 직렬화된 JSON 객체 또는 JSON 파일의 파일 경로를 지정합니다. JSON 객체의 형식은 다음과 같아야 합니다.

  ```
  {  \
    "componentName": { \
      "cpus": cpuTimeLimit, \
      "memory": memoryLimitInKb \
    } \
  }
  ```

  각 구성 요소에 대해 다음과 같은 시스템 리소스 제한을 구성할 수 있습니다.

  - cpus— 이 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 장치의 경우 이 값을 2 설정하여 이 구성 요소의 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 기기에서는 이 값을 0.25 설정하여 이 구성 요소의 프로세스 CPU 사용량을 25% 로 제한할 수 있습니다. 이 값을 CPU AWS IoT Greengrass 코어 수보다 큰 수로 설정하는 경우 Core 소프트웨어는 구성 요소의 CPU 사용량을 제한하지 않습니다.

- `memory`— 이 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 RAM 크기 (KB).

  자세한 설명은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요.

  이 기능은 리눅스 코어 디바이스의 그린그래스 코어 구성 요소 및 그린그래스 CLI의 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

- `--remove`. 로컬 배포에서 제거하려는 대상 구성 요소의 이름. 클라우드 배포에서 병합된 구성 요소를 제거하려면 대상 사물 그룹의 그룹 ID를 다음 형식으로 제공해야 합니다.

  Greengrass nucleus v2.4.0 and later

  ```
  --remove <component-name> --groupId <group-name>
  ```

  Earlier than v2.4.0

  ```
  --remove <component-name> --groupId thinggroup/<group-name>
  ```

- `--failure-handling-policy`. 배포 실패 시 취할 조치를 정의합니다. 지정할 수 있는 작업은 두 가지입니다.

  - `ROLLBACK` –
  - `DO_NOTHING` –

  이 기능은 v2.11.0 이상에서 사용할 수 있습니다. 그린그래스 핵

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ sudo greengrass-cli deployment create \
    --merge MyApp1=1.0.0 \
    --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \
    --remove MyApp3 \
    --recipeDir recipes/ \
    --artifactDir artifacts/

Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-
ad89-5151213dfcbc
```

취소

지정된 배포를 취소합니다.

시놉시스

```
greengrass-cli deployment cancel
    -i <deployment-id>
```

인수

-i. 취소할 배포의 고유 식별자입니다. 배포 ID는 create 명령의 출력에 반환됩니다.

출력

- None

list

최근 10개 로컬 배포의 상태를 검색합니다.

시놉시스

```
greengrass-cli deployment list
```

인수

None

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다. 배포 상태에 따라 출력에는 다음 상태 값 IN_PROGRESSSUCCEEDED, 또는 중 하나가 표시됩니다FAILED.

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

특정 배포의 상태를 검색합니다.

시놉시스

```
greengrass-cli deployment status -i <deployment-id>
```

인수

`-i`. 디플로이먼트의 ID.

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다. 배포 상태에 따라 출력에는 다음 상태 값 `IN_PROGRESS`, `SUCCEEDED`, 또는 중 하나가 표시됩니다`FAILED`.

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

로그

`logs`명령을 사용하여 코어 디바이스의 Greengrass 로그를 분석할 수 있습니다.

하위 명령

- get
- 목록 키워드
- list-log-files

get

Greengrass 로그 파일을 수집, 필터링 및 시각화합니다. 이 명령은 JSON 형식의 로그 파일만 지원합니다. nucleus 구성에서 로깅 형식을 지정할 수 있습니다.

시놉시스

```
greengrass-cli logs get
```

```
    [--log-dir path/to/a/log/folder]
    [--log-file path/to/a/log/file]
    [--follow true | false ]
    [--filter <filter> ]
    [--time-window <start-time>,<end-time> ]
    [--verbose ]
    [--no-color ]
    [--before <value> ]
    [--after <value> ]
    [--syslog ]
    [--max-long-queue-size <value> ]
```

인수

- `--log-dir`,`-ld`. 로그 파일을 확인할 디렉터리 경로 (예:***/greengrass/v2*/logs**. 와 함께 사용하지 마십시오`--syslog`. 지정할 추가 디렉터리마다 별도의 인수를 사용하십시오. `--log-dir`또는 중 하나 이상을 사용해야 `--log-file` 합니다. 단일 명령에 두 인수를 모두 사용할 수도 있습니다.
- `--log-file`,`-lf`. 사용하려는 로그 디렉터리의 경로 지정할 추가 디렉터리마다 별도의 인수를 사용합니다. `--log-dir`또는 중 하나 이상을 사용해야 `--log-file` 합니다. 단일 명령에 두 인수를 모두 사용할 수도 있습니다.
- `--follow`,`-fol`. 로그 업데이트가 발생하는 대로 표시합니다. Greengrass CLI는 계속 실행되며 지정된 로그에서 읽습니다. 기간을 지정하는 경우 Greengrass CLI는 모든 기간이 종료된 후 로그 모니터링을 중지합니다.
- `--filter`,. `-f` 필터로 사용할 키워드, 정규 표현식 또는 키-값 쌍. 이 값을 문자열, 정규 표현식 또는 키-값 쌍으로 제공하십시오. 지정할 각 추가 필터에 대해 별도의 인수를 사용하십시오.

  평가 시 단일 인수에 지정된 여러 필터는 OR 연산자로 구분되고 추가 인수에 지정된 필터는 AND 연산자와 결합됩니다. 예를 들어 명령에 이 포함된 `--filter "installed" --filter "name=alpha,name=beta"` 경우 Greengrass CLI는 키워드와 값이 또는 인 `name` 키를 모두 포함하는 로그 메시지를 `installed` 필터링하고 표시합니다. `alpha beta`
- `--time-window`,. `-t` 로그 정보를 표시할 시간 창입니다. 정확한 타임스탬프와 상대 오프셋을 모두 사용할 수 있습니다. 이 정보는 다음 형식으로 제공해야 합니다. *<begin-time>*,*<end-time>* 시작 시간이나 종료 시간을 지정하지 않는 경우 해당 옵션의 기본값은 현재 시스템 날짜 및 시간입니다. 지정할 추가 시간 창마다 별도의 인수를 사용하십시오.

  Greengrass CLI는 다음과 같은 타임스탬프 형식을 지원합니다.

  - `yyyy-MM-DD`예를 들어,. `2020-06-30` 이 형식을 사용하는 경우 기본 시간은 00:00:00 입니다.

`yyyyMMDD`예를 들어,. `20200630` 이 형식을 사용하는 경우 기본 시간은 00:00:00 입니다.

`HH:mm:ss`예를 들어,. `15:30:45` 이 형식을 사용할 경우 날짜는 현재 시스템 날짜로 기본 설정됩니다.

`HH:mm:ssSSS`예를 들어,`15:30:45`. 이 형식을 사용할 경우 날짜는 현재 시스템 날짜로 기본 설정됩니다.

`YYYY-MM-DD'T'HH:mm:ss'Z'`예를 들어,`2020-06-30T15:30:45Z`.

`YYYY-MM-DD'T'HH:mm:ss`, 예를 들어,`2020-06-30T15:30:45`.

`yyyy-MM-dd'T'HH:mm:ss.SSS`, 예를 들어,`2020-06-30T15:30:45.250`.

상대 오프셋은 현재 시스템 시간으로부터의 기간 오프셋을 지정합니다. Greengrass CLI는 상대 오프셋에 대해 다음 형식을 지원합니다. `+|-[<value>h|hr|hours][value m|min|minutes][value]s|sec|seconds`

예를 들어, 현재 시간보다 1시간에서 2시간 15분 전 사이의 시간 창을 지정하는 다음 인수는 입니다. `--time-window -2h15min,-1hr`

- `--verbose`. 로그 메시지의 모든 필드를 표시합니다. 와 함께 사용하지 마십시오`--syslog`.
- `--no-color,-nc`. 색상 코딩을 제거합니다. 로그 메시지의 기본 색상 코딩은 굵은 빨간색 텍스트를 사용합니다. ANSI 이스케이프 시퀀스를 사용하므로 Unix 계열 터미널만 지원합니다.
- `--before,-b`. 일치하는 로그 항목 앞에 표시할 줄 수입니다. 기본값은 0.
- `--after,-a`. 일치하는 로그 항목 다음에 표시할 줄 수입니다. 기본값은 0.
- `--syslog`. RFC3164 정의된 syslog 프로토콜을 사용하여 모든 로그 파일을 처리합니다. 및 와 함께 `--log-dir` 사용하지 마십시오. `--verbose` syslog 프로토콜은 다음 형식을 사용합니다. `"<$Priority>$Timestamp $Host $Logger ($Class): $Message"` 로그 파일을 지정하지 않으면 Greengrass CLI는, 또는 에서 로그 메시지를 읽습니다. `/var/log/messages /var/log/syslog /var/log/system.log`

  AWS IoT Greengrass현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.
- `--max-log-queue-size,-m`. 메모리에 할당할 최대 로그 항목 수입니다. 이 옵션을 사용하면 메모리 사용을 최적화할 수 있습니다. 기본값은 100입니다.

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ sudo greengrass-cli logs get --verbose \
    --log-file /greengrass/v2/logs/greengrass.log \
    --filter deployment,serviceName=DeploymentService \
    --filter level=INFO \
    --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22

2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.DeploymentService: Current deployment finished.
 {DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,
 currentState=RUNNING}
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted
 deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

목록 키워드

로그 파일을 필터링하는 데 사용할 수 있는 추천 키워드를 표시합니다.

시놉시스

```
greengrass-cli logs list-keywords [arguments]
```

인수

None

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog
```

```
Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

지정된 디렉터리에 있는 로그 파일을 표시합니다.

시놉시스

```
greengrass-cli logs list-log-files [arguments]
```

인수

`--log-dir`,`-ld`. 로그 파일을 확인할 디렉터리 경로입니다.

출력

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
/greengrass/v2/logs/main.log
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

사용`get-debug-password`명령을 사용하여 임의로 생성된 암호를로컬 디버그 콘솔 구성 요소(`aws.greengrass.LocalDebugConsole`). 비밀번호는 생성된 후 8시간 후에 만료됩니다.

시놉시스

```
greengrass-cli get-debug-password
```

인수

없음

출력

다음 예제에서는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
 console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

AWS IoT Greengrass 테스트 프레임워크 사용

Greengrass 테스트 프레임워크 (GTF) 는 고객 관점에서 end-to-end 자동화를 지원하는 빌딩 블록 모음입니다. GTF는 Cucumber를 기능 드라이버로 사용합니다. AWS IoT Greengrass 동일한 구성 요소를 사용하여 다양한 장치의 소프트웨어 변경 사항을 검증합니다. 자세한 내용은 Github의 Greengrass 테스트 프레임워크를 참조하십시오.

GTF는 자동화된 테스트를 실행하는 데 사용되는 도구인 Cucumber를 사용하여 구현되어 구성 요소의 행동 기반 개발 (BDD) 을 장려합니다. Cucumber에서는 이 시스템의 기능이 라는 특수 형식의 파일에 요약되어 있습니다. `feature` 각 기능은 자동화된 테스트로 변환할 수 있는 사양인 시나리오라는 사람이 읽을 수 있는 형식으로 설명됩니다. 각 시나리오는 Gherkin이라는 도메인별 언어를 사용하여 테스트 중인 이 시스템의 상호 작용과 결과를 정의하는 일련의 단계로 요약됩니다. Gherkin 단계는 사양을 테스트 흐름에 고정적으로 연결하는 단계 정의라는 방법을 사용하여 프로그래밍 코드에 연결됩니다. GTF의 단계 정의는 Java로 구현됩니다.

주제

- 작동 방식
- Changelog

- 그린그래스 테스팅 프레임워크 구성 옵션
- 튜토리얼: Greengrass end-to-end 테스트 프레임워크 및 Greengrass 개발 키트를 사용하여 테스트 실행
- 튜토리얼: 신뢰도 테스트 도구 모음의 신뢰도 테스트 사용

작동 방식

AWS IoT Greengrass GTF를 여러 Java 모듈로 구성된 독립형 JAR로 배포합니다. 구성 요소 end-to-end 테스트에 GTF를 사용하려면 Java 프로젝트 내에 테스트를 구현해야 합니다. 테스트 가능한 JAR을 Java 프로젝트에 종속 항목으로 추가하면 GTF의 기존 기능을 사용하고 사용자 지정 테스트 케이스를 작성하여 확장할 수 있습니다. 사용자 지정 테스트 케이스를 실행하려면 Java 프로젝트를 빌드하고 에 설명된 구성 옵션을 사용하여 대상 JAR을 실행하면 됩니다. 그린그래스 테스팅 프레임워크 구성 옵션

GTF 독립형 JAR

Greengrass는 Cloudfront를 메이븐 리포지토리로 사용하여 다양한 버전의 GTF 독립형 JAR을 호스팅합니다. GTF 버전의 전체 목록은 GTF 릴리스를 참조하십시오.

GTF 독립형 JAR에는 다음 모듈이 포함되어 있습니다. 이러한 모듈에만 국한되지 않습니다. 프로젝트에서 이러한 각 종속성을 개별적으로 선택하여 선택하거나 테스트 독립형 JAR 파일에 모든 종속성을 한 번에 포함할 수 있습니다.

- `aws-greengrass-testing-resources`: 이 모듈은 테스트 과정에서 AWS 리소스의 라이프사이클을 관리하기 위한 추상화를 제공합니다. 이를 통해 `ResourceSpec` 추상화를 사용하여 사용자 지정 AWS 리소스를 정의하면 GTF가 해당 리소스의 생성 및 제거를 알아서 처리할 수 있습니다.
- `aws-greengrass-testing-platform`: 이 모듈은 테스트 수명 주기 동안 테스트 대상 기기에 대한 플랫폼 수준의 추상화를 제공합니다. 여기에는 플랫폼과 관계없이 OS와 상호 작용하는 데 사용되는 API가 포함되어 있으며 기기 셸에서 실행되는 명령을 시뮬레이션하는 데 사용할 수 있습니다.
- `aws-greengrass-testing-components`: 이 모듈은 배포, IPC 및 기타 기능과 같은 Greengrass 핵심 기능을 테스트하는 데 사용되는 샘플 구성 요소로 구성되어 있습니다.
- `aws-greengrass-testing-features`: 이 모듈은 Greengrass 환경에서 테스트하는 데 사용되는 재사용 가능한 공통 단계와 정의로 구성되어 있습니다.

주제

- Changelog

- 그린그래스 테스팅 프레임워크 구성 옵션
- 튜토리얼: Greengrass end-to-end 테스트 프레임워크 및 Greengrass 개발 키트를 사용하여 테스트 실행
- 튜토리얼: 신뢰도 테스트 도구 모음의 신뢰도 테스트 사용

Changelog

다음 표에는 GTF의 각 버전에서의 변경 사항이 설명되어 있습니다. 자세한 내용은 의 GTF 릴리스 페이지를 참조하십시오. GitHub

버전	변경
1.2.0	새로운 기능 • 테스트 중에 MQTT 및 인터넷 네트워크 연결을 구성하기 위한 네트워크 관련 단계를 추가합니다. • 장치 RAM 및 CPU 사용을 모니터링하는 시스템 메트릭 단계를 추가합니다. 버그 수정 및 개선 • Greengrass CLI 로컬 배포 단계는 성공할 때까지 재시도합니다. • 테스트는 Greengrass 핵을 죽이는 대신 정상적으로 차단합니다. • 사물 및 역할 별칭에 대한 자격 증명을 검색할 수 있을 때까지 GTF가 AWS IoT 자격 증명 엔드포인트를 폴링하는 방식을 개선했습니다. • 누락된 아티팩트와 레시피 디렉토리를 수정합니다. 이 버전은 누락된 구성 요소 버전도 수정합니다. • docker 이미지가 없는 경우 docker 이미지 정리 중에 GTF가 실패하는 문제를 수정합니다. • CURRENT 키워드를 구성 요소 버전으로 추가합니다.
1.1.0	새로운 기능 • 구성을 사용하여 사용자 지정 구성 요소를 설치하는 기능을 추가합니다. 이를 위해서는 사용자 지정 구성 요소의 레시피가 필요합니다. • 사용자 지정 구성으로 로컬 배포를 업데이트하는 기능을 추가합니다.

버전	변경
	버그 수정 및 개선 • 로그 컨텍스트 GTF 버전 불일치 문제를 수정합니다.
1.0.0	초기 버전

그린그래스 테스팅 프레임워크 구성 옵션

GTF 구성 옵션

그린그래스 테스트 프레임워크 (GTF) 를 사용하면 출시 중에 특정 매개변수를 구성할 수 있습니다. end-to-end 테스트 흐름을 조율하기 위한 테스트 프로세스. 이러한 구성 옵션을 GTF 독립형 JAR의 CLI 인수로 지정할 수 있습니다.

GTF 버전 1.1.0 이상에서는 다음과 같은 구성 옵션을 제공합니다.

- `additional-plugins`— (선택 사항) 추가 오이 플러그인
- `aws-region`— 특정 지역 엔드포인트를 대상으로 합니다.AWS서비스. 기본값은 다음과 같습니다.AWSSDK가 발견한 내용.
- `credentials-path`— 선택 사항AWS프로필 자격 증명 경로. 호스트 환경에서 검색된 자격 증명이 기본값입니다.
- `credentials-path-rotation`— 선택적 회전 기간AWS자격 증명. 기본값은 15분 또는`PT15M`.
- `csr-path`— 디바이스 인증서를 생성하는 데 사용할 CSR의 경로입니다.
- `device-mode`— 테스트 대상 장치. 기본값은 로컬 장치입니다.
- `env-stage`— Greengrass의 배포 환경을 대상으로 합니다. 기본값은 프로덕션입니다.
- `existing-device-cert-arn`— Greengrass의 디바이스 인증서로 사용하려는 기존 인증서의 arn.
- `feature-path`— 추가 기능 파일이 들어 있는 파일 또는 디렉터리 기본값은 추가 기능 파일이 사용되지 않는 것입니다.
- `gg-cli-version`— 그린그래스 CLI의 버전을 재정의합니다. 기본값은 에 있는 값입니다.`ggc.version`.
- `gg-component-bucket`— 그린그래스 구성 요소가 들어 있는 기존 Amazon S3 버킷의 이름.
- `gg-component-overrides`— 그린그래스 구성 요소 재정의 목록.

- `gg-persist`— 테스트 실행 후 유지되는 테스트 요소 목록. 기본 동작은 아무것도 지속하지 않는 것입니다. 허용되는 값은 다음과 같습니다.`aws.resources`,`installed.software`, 그리고`generated.files`.
- `gg-runtime`— 테스트가 테스트 리소스와 상호 작용하는 방식에 영향을 미치는 값 목록. 이 값은 다음을 대체합니다.`gg.persist`파라미터. 기본값이 비어 있으면 설치된 Greengrass 런타임을 포함하여 모든 테스트 리소스가 테스트 케이스별로 관리된다고 가정합니다. 허용되는 값은 다음과 같습니다.`aws.resources`,`installed.software`, 그리고`generated.files`.
- `ggc-archive`— 보관된 그린그래스 핵 구성 요소의 경로.
- `ggc-install-root`— 그린그래스 핵 구성 요소를 설치하기 위한 디렉토리. 기본값은 test.temp.path 및 테스트 실행 폴더입니다.
- `ggc-log-level`— 테스트 실행을 위한 그린그래스 핵 로그 수준을 설정합니다. 기본값은 “정보”입니다.
- `ggc-tes-rolename`— 다음과 같은 IAM 역할AWS IoT Greengrass코어가 액세스를 맡게 됩니다.AWS서비스. 지정된 이름을 가진 역할이 존재하지 않는 경우 역할이 생성되고 기본 액세스 정책이 적용됩니다.
- `ggc-trusted-plugins`— Greengrass에 추가해야 하는 신뢰할 수 있는 플러그인의 호스트 경로를 쉼표로 구분한 목록입니다. DUT 자체의 경로를 제공하려면 경로 앞에 'dut: '를 접두사로 붙입니다.
- `ggc-user-name`— 그린그래스 핵의 사용자:그룹 POSIX사용자 값입니다. 로그인한 현재 사용자 이름이 기본값입니다.
- `ggc-version`— 실행 중인 Greengrass nucleus 구성 요소의 버전을 재정의합니다. ggc.archive에 있는 값이 기본값입니다.
- `log-level`— 테스트 실행의 로그 수준. 기본값은 “INFO”입니다.
- `parallel-config`— 배치 인덱스 및 배치 수를 JSON 문자열로 설정합니다. 배치 인덱스의 기본값은 0이고 배치 수는 1입니다.
- `proxy-url`— 이 URL을 통해 트래픽을 라우팅하도록 모든 테스트를 구성합니다.
- `tags`— 기능 태그만 실행합니다. '&'와 교차할 수 있습니다.
- `test-id-prefix`— 다음을 포함한 모든 테스트 관련 리소스에 적용되는 공통 접두사AWS리소스 이름 및 태그. 기본값은 “gg” 접두사입니다.
- `test-log-path`— 전체 테스트 실행 결과를 포함할 디렉터리입니다. 기본값은 “TestResults”입니다.
- `test-results-json`— 결과 Cucumber JSON 보고서가 디스크에 기록되어 생성되는지 여부를 결정하는 플래그입니다. 기본값은 true입니다.

- `test-results-log`— 콘솔 출력이 생성되어 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 false입니다.
- `test-results-xml`— 결과 JUnit XML 보고서가 생성되어 디스크에 기록되는지 여부를 결정하는 플래그를 지정합니다. 기본값은 true입니다.
- `test-temp-path`— 로컬 테스트 아티팩트를 생성하기 위한 디렉터리입니다. gg-testing 접두사가 붙은 임의의 임시 디렉토리가 기본값입니다.
- `timeout-multiplier`— 모든 테스트 타임아웃에 멀티플라이어가 제공됩니다. 기본값은 1.0.

튜토리얼: Greengrass end-to-end 테스트 프레임워크 및 Greengrass 개발 키트를 사용하여 테스트 실행

AWS IoT Greengrass테스트 프레임워크 (GTF) 와 Greengrass 개발 키트 (GDK) 는 개발자에게 테스트를 실행할 수 있는 방법을 제공합니다. end-to-end 이 자습서를 완료하여 구성 요소로 GDK 프로젝트를 초기화하고, end-to-end 테스트 모듈로 GDK 프로젝트를 초기화하고, 사용자 지정 테스트 케이스를 구축할 수 있습니다. 사용자 지정 테스트 케이스를 빌드한 후 테스트를 실행할 수 있습니다.

이 자습서에서는 다음 작업을 수행합니다.

1. 구성 요소를 사용하여 GDK 프로젝트를 초기화합니다.
2. 테스트 모듈을 사용하여 GDK 프로젝트를 초기화합니다. end-to-end
3. 사용자 지정 테스트 케이스를 만드세요.
4. 새 테스트 케이스에 태그를 추가합니다.
5. 테스트 JAR을 빌드하세요.
6. 테스트를 실행합니다.

주제

- 예: 사용자 지정 테스트 케이스 만들기

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- GDK 버전 1.3.0 이상
- Java
- Maven
- Git

1단계: 구성 요소를 사용하여 GDK 프로젝트 초기화

- GDK 프로젝트로 빈 폴더를 초기화합니다. 다음 명령을 실행하여 Python으로 구현된 `HelloWorld` 구성 요소를 다운로드합니다.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  이 명령은 현재 `HelloWorld` 디렉터리에 이름이 지정된 새 디렉터리를 만듭니다.

2단계: 테스트 모듈로 GDK 프로젝트 초기화 end-to-end

- GDK를 사용하면 기능 및 단계 구현으로 구성된 테스트 모듈 템플릿을 다운로드할 수 있습니다. 다음 명령을 실행하여 `HelloWorld` 디렉터리를 열고 테스트 모듈을 사용하여 기존 GDK 프로젝트를 초기화합니다.

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  이 명령은 디렉터리 `gg-e2e-tests` 내에 이름이 지정된 새 디렉터리를 만듭니다. `HelloWorld` 이 테스트 디렉토리는 Greengrass 테스트 독립형 JAR에 종속되는 Maven 프로젝트입니다.

3단계: 사용자 지정 테스트 케이스 만들기

사용자 지정 테스트 사례 작성은 크게 두 단계로 구성됩니다. 테스트 시나리오가 포함된 기능 파일을 만들고 단계 정의를 구현하는 단계입니다. 사용자 지정 테스트 케이스를 만드는 예제는 을 참조하십시

오예: 사용자 지정 테스트 케이스 만들기. 다음 단계를 사용하여 사용자 지정 테스트 케이스를 빌드하세요.

1. 테스트 시나리오가 포함된 기능 파일 만들기

   기능은 일반적으로 테스트 중인 소프트웨어의 특정 기능을 설명합니다. Cucumber에서 각 기능은 제목, 자세한 설명 및 시나리오라고 하는 특정 사례의 하나 이상의 예가 포함된 개별 기능 파일로 지정됩니다. 각 시나리오는 제목, 자세한 설명, 상호 작용 및 예상 결과를 정의하는 일련의 단계로 구성됩니다. 시나리오는 “주어진”, “언제”, “then” 키워드를 사용하여 구조화된 형식으로 작성됩니다.

2. 단계 정의를 구현하십시오.

   단계 정의는 Gherkin 단계를 일반 언어로 프로그래밍 코드에 연결합니다. Cucumber는 시나리오에서 Gherkin 단계를 식별하면 실행할 일치하는 단계 정의를 찾습니다.

4단계: 새 테스트 케이스에 태그 추가

- 기능 및 시나리오에 태그를 할당하여 테스트 프로세스를 구성할 수 있습니다. 태그를 사용하여 시나리오의 하위 집합을 분류하고 실행할 후크를 조건부로 선택할 수도 있습니다. 기능 및 시나리오에는 공백으로 구분된 여러 개의 태그가 있을 수 있습니다.

  이 예시에서는 `HelloWorld` 컴포넌트를 사용하고 있습니다.

  기능 파일에서 태그 `@HelloWorld` 옆에 이름이 지정된 새 태그를 추가합니다. `@Sample`

```
@Sample @HelloWorld
Scenario: As a developer, I can create a component and deploy it on my device
....
```

5단계: 테스트 JAR 빌드

1. 구성 요소를 빌드합니다. 테스트 모듈을 빌드하기 전에 구성 요소를 빌드해야 합니다.

```
gdk component build
```

2. 다음 명령을 사용하여 테스트 모듈을 빌드합니다. 이 명령은 `greengrass-build` 폴더에 테스트 JAR을 빌드합니다.

```
gdk test-e2e build
```

6단계: 테스트 실행

사용자 지정 테스트 케이스를 실행하면 GTF가 테스트 중에 생성된 리소스를 관리하는 동시에 테스트의 라이프사이클을 자동화합니다. 먼저 테스트 대상 장치 (DUT) 를 AWS IoT 사물로 프로비저닝하고 여기에 Greengrass 코어 소프트웨어를 설치합니다. 그러면 해당 경로에 지정된 레시피를 `HelloWorld` 사용하여 이름이 지정된 새 구성 요소가 생성됩니다. 그런 다음 Greengrass 사물 배포를 통해 `HelloWorld` 구성 요소를 코어 장치에 배포합니다. 그런 다음 배포가 성공했는지 여부를 확인합니다. 배포가 성공하면 배포 상태가 3분 `COMPLETED` 이내로 변경됩니다.

1. 프로젝트 디렉터리의 `gdk-config.json` 파일로 이동하여 `HelloWorld` 태그가 있는 테스트를 대상으로 지정하십시오. 다음 명령어를 사용하여 `test-e2e` 키를 업데이트합니다.

   ```
   "test-e2e":{
     "gtf_options" : {
          "tags":"HelloWorld"
      }
   }
   ```

2. 테스트를 실행하기 전에 호스트 기기에 AWS 자격 증명을 제공해야 합니다. GTF는 테스트 프로세스 중에 이러한 자격 증명을 사용하여 AWS 리소스를 관리합니다. 제공하는 역할에 테스트에 포함된 필수 작업을 자동화할 수 있는 권한이 있는지 확인하세요.

   다음 명령을 실행하여 AWS 자격 증명을 제공하십시오.

   - Linux or Unix

     ```
     export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

3. 다음 명령을 사용하여 테스트를 실행합니다.

```
gdk test-e2e run
```

이 명령은 `greengrass-build` 폴더에 있는 Greengrass 핵의 최신 버전을 다운로드하고 이를 사용하여 테스트를 실행합니다. 또한 이 명령은 `HelloWorld` 태그가 있는 시나리오만 대상으로 하고 해당 시나리오에 대한 보고서를 생성합니다. 이 테스트 중에 생성된 AWS 리소스는 테스트가 끝나면 삭제되는 것을 확인할 수 있습니다.

예: 사용자 지정 테스트 케이스 만들기

Example

GDK 프로젝트에 다운로드한 테스트 모듈은 샘플 기능과 단계 구현 파일로 구성되어 있습니다.

다음 예에서는 Greengrass 소프트웨어의 사물 배포 기능을 테스트하기 위한 기능 파일을 생성합니다. AWS 클라우드Greengrass를 통해 구성 요소를 배포하는 시나리오를 사용하여 이 기능의 기능을 부분적으로 테스트합니다. 이 단계는 이 사용 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다.

1. 기능 파일 만들기

현재 디렉터리의 `gg-e2e-tests/src/main/resources/greengrass/features` 폴더로 이동합니다. 다음 예와 같은 샘플을 `component.feature` 찾을 수 있습니다.

이 기능 파일에서 Greengrass 소프트웨어의 사물 배포 기능을 테스트할 수 있습니다. Greengrass 클라우드를 통해 구성 요소를 배포하는 시나리오를 사용하여 이 기능의 기능을 부분적으로 테스트할 수 있습니다. 시나리오는 이 사용 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다.

```
Feature: Testing features of Greengrassv2 component

Background:
    Given my device is registered as a Thing
```

```
    And my device is running Greengrass

@Sample
Scenario: As a developer, I can create a component and deploy it on my device
    When I create a Greengrass deployment with components
        HelloWorld | /path/to/recipe/file
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    And I call my custom step
```

GTF에는 다음과 같은 단계를 제외한 모든 단계에 대한 단계 정의가 포함되어 있습니다. `And I call my custom step`

2. 단계 정의 구현

   GTF 독립형 JAR에는 한 단계를 제외한 모든 단계의 단계 정의가 포함되어 있습니다. `And I call my custom step` 테스트 모듈에서 이 단계를 구현할 수 있습니다.

   테스트 파일의 소스 코드로 이동합니다. 다음 명령어를 사용하여 단계 정의를 사용하여 사용자 지정 단계를 연결할 수 있습니다.

```
@And("I call my custom step")
public void customStep() {
    System.out.println("My custom step was called ");
}
```

튜토리얼: 신뢰도 테스트 도구 모음의 신뢰도 테스트 사용

AWS IoT Greengrass테스트 프레임워크 (GTF) 와 Greengrass 개발 키트 (GDK) 는 개발자에게 테스트를 실행할 수 있는 방법을 제공합니다. end-to-end 이 튜토리얼을 완료하여 구성 요소로 GDK 프로젝트를 초기화하고, end-to-end 테스트 모듈로 GDK 프로젝트를 초기화하고, 신뢰도 테스트 스위트의 신뢰도 테스트를 사용할 수 있습니다. 사용자 지정 테스트 케이스를 빌드한 후 테스트를 실행할 수 있습니다.

신뢰도 테스트는 기본 구성 요소 동작을 검증하는 Greengrass에서 제공하는 일반 테스트입니다. 이러한 테스트는 보다 구체적인 구성 요소 요구 사항에 맞게 수정하거나 확장할 수 있습니다.

이 자습서에서는 HelloWorld 구성 요소를 사용합니다. 다른 구성 요소를 사용하는 경우 구성 요소를 해당 HelloWorld 구성 요소로 교체하십시오.

이 자습서에서는 다음 작업을 수행합니다.

1. 구성 요소로 GDK 프로젝트를 초기화합니다.
2. 테스트 모듈을 사용하여 GDK 프로젝트를 초기화합니다. end-to-end
3. 신뢰도 테스트 스위트의 테스트를 사용하세요.
4. 새 테스트 케이스에 태그를 추가합니다.
5. 테스트 JAR을 빌드하세요.
6. 테스트를 실행합니다.

주제

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- GDK 버전 1.6.0 이상
- Java
- Maven
- Git

1단계: 구성 요소를 사용하여 GDK 프로젝트 초기화

- GDK 프로젝트로 빈 폴더를 초기화합니다. 다음 명령을 실행하여 Python으로 구현된 `HelloWorld` 구성 요소를 다운로드합니다.

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

이 명령은 현재 `HelloWorld` 디렉터리에 이름이 지정된 새 디렉터리를 만듭니다.

2단계: 테스트 모듈로 GDK 프로젝트 초기화 end-to-end

- GDK를 사용하면 기능 및 단계 구현으로 구성된 테스트 모듈 템플릿을 다운로드할 수 있습니다. 다음 명령을 실행하여 `HelloWorld` 디렉터리를 열고 테스트 모듈을 사용하여 기존 GDK 프로젝트를 초기화합니다.

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  이 명령은 디렉터리 `gg-e2e-tests` 내에 이름이 지정된 새 디렉터리를 만듭니다. `HelloWorld` 이 테스트 디렉토리는 Greengrass 테스트 독립형 JAR에 종속되는 Maven 프로젝트입니다.

3단계: 컨피던스 테스트 스위트의 테스트 사용

신뢰도 테스트 사례 작성은 제공된 기능 파일을 사용하고 필요한 경우 시나리오를 수정하는 것으로 구성됩니다. 신뢰도 테스트 사용의 예는 을 참조하십시오예: 사용자 지정 테스트 케이스 만들기. 신뢰도 테스트를 사용하려면 다음 단계를 따르세요.

- 제공된 기능 파일을 사용하세요.

  현재 디렉터리의 `gg-e2e-tests/src/main/resources/greengrass/features` 폴더로 이동합니다. 샘플 `confidenceTest.feature` 파일을 열어 신뢰도 테스트를 사용하십시오.

4단계: 새 테스트 케이스에 태그 추가

- 기능 및 시나리오에 태그를 할당하여 테스트 프로세스를 구성할 수 있습니다. 태그를 사용하여 시나리오의 하위 집합을 분류하고 실행할 후크를 조건부로 선택할 수도 있습니다. 기능 및 시나리오에는 공백으로 구분된 여러 개의 태그가 있을 수 있습니다.

  이 예시에서는 `HelloWorld` 컴포넌트를 사용하고 있습니다.

  각 시나리오에는 태그가 지정되어 있습니다. `@ConfidenceTest` 테스트 스위트의 일부만 실행하려는 경우 태그를 변경하거나 추가하세요. 각 테스트 시나리오는 각 신뢰도 테스트의 상단에 설명

되어 있습니다. 시나리오는 각 테스트 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다. 자체 단계를 추가하거나 기존 단계를 수정하여 이러한 테스트를 확장할 수 있습니다.

```
@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
....
```

5단계: 테스트 JAR 빌드

1. 구성 요소를 빌드합니다. 테스트 모듈을 빌드하기 전에 구성 요소를 빌드해야 합니다.

```
gdk component build
```

2. 다음 명령을 사용하여 테스트 모듈을 빌드합니다. 이 명령은 `greengrass-build` 폴더에 테스트 JAR을 빌드합니다.

```
gdk test-e2e build
```

6단계: 테스트 실행

신뢰도 테스트를 실행하면 GTF는 테스트 중에 생성된 리소스를 관리하는 동시에 테스트의 라이프사이클을 자동화합니다. 먼저 테스트 대상 장치 (DUT) 를 AWS IoT 사물로 프로비저닝하고 여기에 Greengrass 코어 소프트웨어를 설치합니다. 그러면 해당 경로에 지정된 레시피를 `HelloWorld` 사용하여 이름이 지정된 새 구성 요소가 생성됩니다. 그런 다음 Greengrass 사물 배포를 통해 `HelloWorld` 구성 요소를 코어 장치에 배포합니다. 그런 다음 배포가 성공했는지 여부를 확인합니다. 배포가 성공하면 배포 상태가 3분 `COMPLETED` 이내로 변경됩니다.

1. 프로젝트 디렉터리의 `gdk-config.json` 파일로 이동하여 `ConfidenceTest` 태그 또는 4단계에서 지정한 태그 중 하나를 사용하여 테스트를 대상으로 지정합니다. 다음 명령어를 사용하여 `test-e2e` 키를 업데이트합니다.

```
  "test-e2e":{
    "gtf_options" : {
         "tags":"ConfidenceTest"
     }
```

```
}
```

2. 테스트를 실행하기 전에 호스트 기기에 AWS 자격 증명을 제공해야 합니다. GTF는 테스트 프로세스 중에 이러한 자격 증명을 사용하여 AWS 리소스를 관리합니다. 제공하는 역할에 테스트에 포함된 필수 작업을 자동화할 수 있는 권한이 있는지 확인하세요.

   다음 명령을 실행하여 AWS 자격 증명을 제공하십시오.

   - Linux or Unix

     ```
     export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     PowerShell

     ```
     $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
     $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
     ```

3. 다음 명령을 사용하여 테스트를 실행합니다.

   ```
   gdk test-e2e run
   ```

   이 명령은 `greengrass-build` 폴더에 있는 Greengrass 핵의 최신 버전을 다운로드하고 이를 사용하여 테스트를 실행합니다. 또한 이 명령은 `ConfidenceTest` 태그가 있는 시나리오만 대상으로 하고 해당 시나리오에 대한 보고서를 생성합니다. 이 테스트 중에 생성된 AWS 리소스는 테스트가 끝나면 삭제되는 것을 확인할 수 있습니다.

예: 신뢰도 테스트 사용

Example

GDK 프로젝트에서 다운로드한 테스트 모듈은 제공된 기능 파일로 구성되어 있습니다.

다음 예제에서는 Greengrass 소프트웨어의 사물 배포 기능을 테스트하기 위해 기능 파일을 사용합니다. AWS 클라우드Greengrass를 통해 구성 요소를 배포하는 시나리오를 사용하여 이 기능의 기능을

부분적으로 테스트합니다. 이 단계는 이 사용 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다.

- 제공된 기능 파일을 사용하세요.

  현재 디렉터리의 `gg-e2e-tests/src/main/resources/greengrass/features` 폴더로 이동합니다. 다음 예와 같은 샘플을 `confidenceTest.feature` 찾을 수 있습니다.

  ```
  Feature: Confidence Test Suite

  Background:
      Given my device is registered as a Thing
      And my device is running Greengrass

  @ConfidenceTest
  Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
   is working as expected
      When I create a Greengrass deployment with components
        | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
        | aws.greengrass.Cli | LATEST                    |
      And I deploy the Greengrass deployment configuration
      Then the Greengrass deployment is COMPLETED on the device after 180 seconds
      # Update component state accordingly. Possible states: {RUNNING, FINISHED,
   BROKEN, STOPPING}
      And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-
  cli
  ```

  각 테스트 시나리오는 각 신뢰 테스트의 상단에 설명되어 있습니다. 시나리오는 각 테스트 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다. 자체 단계를 추가하거나 기존 단계를 수정하여 이러한 테스트를 확장할 수 있습니다. 각 시나리오에는 이러한 조정을 수행하는 데 도움이 되는 설명이 포함되어 있습니다.

AWS IoT Greengrass구성 요소 개발

Greengrass 코어 디바이스에서 구성 요소를 개발하고 테스트할 수 있습니다. 따라서 AWS IoT Greengrass 소프트웨어를 사용하지 않고도 소프트웨어를 만들고 반복할 수 있습니다. AWS 클라우드 구성 요소 버전을 완성하면 클라우드에 업로드하여 팀과 함께 구성 요소를 플릿 AWS IoT Greengrass 내 다른 장치에 배포할 수 있습니다. 구성 요소를 배포하는 방법에 대한 자세한 내용은 을 참조하십시오디바이스에 AWS IoT Greengrass 구성 요소 배포.

모든 구성 요소는 레시피와 아티팩트로 구성됩니다.

- 레시피

  모든 구성 요소에는 메타데이터를 정의하는 레시피 파일이 포함되어 있습니다. 레시피는 또한 구성 요소의 구성 매개 변수, 구성 요소 종속성, 수명 주기 및 플랫폼 호환성을 지정합니다. 구성 요소 수명 주기는 구성 요소를 설치, 실행 및 종료하는 명령을 정의합니다. 자세한 설명은 AWS IoT Greengrass컴포넌트 레시피 참조 섹션을 참조하세요.

  레시피는 JSON 또는 YAML 형식으로 정의할 수 있습니다.

- 아티팩트

  구성 요소에는 구성 요소 바이너리인 아티팩트가 여러 개 있을 수 있습니다. 아티팩트에는 스크립트, 컴파일된 코드, 정적 리소스 및 구성 요소가 사용하는 기타 모든 파일이 포함될 수 있습니다. 구성 요소는 구성 요소 종속성의 아티팩트를 사용할 수도 있습니다.

AWS IoT Greengrass애플리케이션에서 사용하고 디바이스에 배포할 수 있는 사전 빌드된 구성 요소를 제공합니다. 예를 들어 스트림 관리자 구성 요소를 사용하여 다양한 AWS 서비스에 데이터를 업로드하거나 지표 구성 요소를 사용하여 Amazon에 사용자 지정 CloudWatch 지표를 게시할 수 CloudWatch 있습니다. 자세한 설명은 AWS-제공된 구성 요소 섹션을 참조하세요.

AWS IoT GreengrassGreengrass 소프트웨어 카탈로그라고 하는 Greengrass 구성 요소의 색인을 큐레이션합니다. 이 카탈로그는 Greengrass 커뮤니티에서 개발한 Greengrass 구성 요소를 추적합니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다. 자세한 설명은 커뮤니티 구성 요소 섹션을 참조하세요.

AWS IoT GreengrassCore 소프트웨어는 구성 요소를 시스템 사용자 및 그룹 (예: 코어 장치에 구성한 `ggc_user` 및`ggc_group`) 으로 실행합니다. 즉, 구성 요소에는 해당 시스템 사용자의 권한이 있습니다. 홈 디렉터리가 없는 시스템 사용자를 사용하는 경우 구성 요소는 실행 명령이나 홈 디렉터리를 사용하는 코드를 사용할 수 없습니다. 즉, 예를 들어 `pip install some-library --user` 명령을 사용하여 Python 패키지를 설치할 수 없습니다. 시작하기 튜토리얼을 따라 코어 기기를 설정했다면 시스템 사용자에게 홈 디렉터리가 없는 것입니다. 구성 요소를 실행하는 사용자 및 그룹을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오구성 요소를 실행하는 사용자를 구성하십시오..

> **Note**
>
> AWS IoT Greengrass구성 요소에 시맨틱 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 버전은 구성 `1.0.0` 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

주제

구성 요소 수명 주기

구성 요소 수명 주기는 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 설치하고 실행하는 데 사용하는 단계를 정의합니다. 각 단계는 구성 요소의 작동 방식을 지정하는 스크립트 및 기타 정보를 정의합니다. 예를 들어, 구성 요소를 설치하면 AWS IoT Greengrass Core 소프트웨어가 해당 구성 요소의 `Install` 수명 주기 스크립트를 실행합니다. 코어 장치의 구성 요소에는 다음과 같은 수명 주기 상태가 있습니다.

- `NEW`— 구성 요소의 레시피와 아티팩트가 코어 장치에 로드되지만 구성 요소는 설치되지 않습니다. 구성 요소가 이 상태가 되면 해당 설치 스크립트가 실행됩니다.
- `INSTALLED`— 구성 요소가 코어 디바이스에 설치됩니다. 구성 요소는 설치 스크립트를 실행한 후 이 상태가 됩니다.
- `STARTING`— 구성 요소가 코어 디바이스에서 시작됩니다. 구성 요소는 시작 스크립트를 실행할 때 이 상태가 됩니다. 시작에 성공하면 구성 요소가 `RUNNING` 상태로 전환됩니다.
- `RUNNING`— 구성 요소가 코어 디바이스에서 실행 중입니다. 구성 요소는 실행 스크립트를 실행하거나 시작 스크립트의 백그라운드 프로세스가 활성 상태일 때 이 상태가 됩니다.

- `FINISHED`— 구성 요소가 성공적으로 실행되어 실행을 완료했습니다.
- `STOPPING`— 구성 요소가 중지되고 있습니다. 구성 요소는 종료 스크립트를 실행할 때 이 상태가 됩니다.
- `ERRORED`— 구성 요소에 오류가 발생했습니다. 구성 요소가 이 상태가 되면 해당 복구 스크립트가 실행됩니다. 그런 다음 구성 요소가 다시 시작되어 정상 사용 상태로 돌아가려고 시도합니다. 구성 요소가 성공적으로 실행되지 않은 `ERRORED` 상태로 세 번 전환되면 구성 요소는 다음과 같이 됩니다`BROKEN`.
- `BROKEN`— 구성 요소에 오류가 여러 번 발생하여 복구할 수 없습니다. 구성 요소를 다시 배포하여 문제를 해결해야 합니다.

구성 요소 유형

구성 요소 유형은 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 실행하는 방법을 지정합니다. 구성 요소에는 다음과 같은 유형이 있을 수 있습니다.

- 핵 () `aws.greengrass.nucleus`

  Greengrass 핵은 Core 소프트웨어의 최소 기능을 제공하는 구성 요소입니다. AWS IoT Greengrass 자세한 설명은 그린그래스 핵 섹션을 참조하세요.

- 플러그인 () `aws.greengrass.plugin`

  Greengrass 핵은 핵과 동일한 자바 가상 머신 (JVM) 에서 플러그인 컴포넌트를 실행합니다. 코어 기기에서 플러그인 구성 요소의 버전을 변경하면 NUCLEUS가 다시 시작됩니다. 플러그인 구성 요소를 설치하고 실행하려면 Greengrass nucleus가 시스템 서비스로 실행되도록 구성해야 합니다. 자세한 설명은 Greengrass 핵을 시스템 서비스로 구성 섹션을 참조하세요.

  에서 제공하는 AWS 여러 구성 요소는 Greengrass 핵과 직접 인터페이스할 수 있는 플러그인 구성 요소입니다. 플러그인 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

- 일반 () `aws.greengrass.generic`

  Greengrass nucleus는 구성 요소가 수명 주기를 정의하는 경우 일반 구성 요소의 수명 주기 스크립트를 실행합니다.

  이 유형은 사용자 지정 구성 요소의 기본 유형입니다.

- **`aws.greengrass.lambda`**람다 ()

Greengrass 핵은 Lambda 런처 구성 요소를 사용하여 Lambda 함수 구성 요소를 실행합니다.

Lambda 함수에서 구성 요소를 생성할 때 구성 요소는 다음과 같은 유형을 갖습니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

> **Note**
>
> 레시피에 구성 요소 유형을 지정하지 않는 것이 좋습니다. AWS IoT Greengrass구성 요소를 만들 때 자동으로 유형을 설정합니다.

AWS IoT Greengrass 구성 요소 생성

로컬 개발 컴퓨터 또는 Greengrass 코어 디바이스에서 사용자 지정 AWS IoT Greengrass 구성 요소를 개발할 수 있습니다. AWS IoT Greengrass 는 사전 정의된 구성 요소 템플릿과 커뮤니티 구성 요소에서 구성 요소를 만들고 빌드하고 게시하는 데 도움이 되는 AWS IoT Greengrass 개발 키트 명령줄 인터페이스 (GDK CLI) 를 제공합니다. 또한 내장된 셸 명령을 실행하여 구성 요소를 만들고 빌드하고 게시할 수 있습니다. 다음 옵션 중에서 선택하여 사용자 지정 Greengrass 구성 요소를 생성합니다.

- 그린그래스 개발 키트 CLI 사용

  GDK CLI를 사용하여 로컬 개발 컴퓨터에서 구성 요소를 개발합니다. GDK CLI는 구성 요소 소스 코드를 서비스에 비공개 구성 요소로 게시할 수 있는 레시피와 아티팩트로 빌드하고 패키징합니다. AWS IoT Greengrass 구성 요소를 게시할 때 구성 요소의 버전과 아티팩트 URI를 자동으로 업데이트하도록 GDK CLI를 구성할 수 있으므로 매번 레시피를 업데이트할 필요가 없습니다. GDK CLI를 사용하여 구성 요소를 개발하려면 Greengrass 소프트웨어 카탈로그의 템플릿 또는 커뮤니티 구성 요소에서 시작할 수 있습니다. 자세한 설명은 AWS IoT Greengrass개발 키트 명령줄 인터페이스 섹션을 참조하세요.
- 내장된 셸 명령을 실행합니다.

  내장 셸 명령을 실행하여 로컬 개발 컴퓨터 또는 Greengrass 코어 장치에서 구성 요소를 개발할 수 있습니다. 셸 명령을 사용하여 구성 요소 소스 코드를 아티팩트에 복사하거나 빌드할 수 있습니다. 구성 요소의 새 버전을 만들 때마다 새 구성 요소 버전으로 레시피를 만들거나 업데이트해야 합니다. 구성 요소를 AWS IoT Greengrass 서비스에 게시할 때는 레시피의 각 구성 요소 아티팩트에 대한 URI를 업데이트해야 합니다.

주제

- 구성 요소 생성 (GDK CLI)
- 구성 요소 생성 (셸 명령)

구성 요소 생성 (GDK CLI)

이 섹션의 지침에 따라 GDK CLI를 사용하여 구성 요소를 만들고 빌드하십시오.

그린그래스 컴포넌트 (GDK CLI) 를 개발하려면

1. 아직 설치하지 않았다면 개발 컴퓨터에 GDK CLI를 설치하세요. 자세한 설명은 AWS IoT Greengrass개발 키트 명령줄 인터페이스 설치 또는 업데이트 섹션을 참조하세요.
2. 구성 요소 폴더를 생성하려는 폴더로 변경합니다.

   Linux or Unix

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

3. 다운로드할 구성 요소 템플릿 또는 커뮤니티 구성 요소를 선택합니다. GDK CLI는 템플릿 또는 커뮤니티 구성 요소를 다운로드하므로 기능 예제에서 시작할 수 있습니다. 구성 요소 목록 명령을 사용하여 사용 가능한 템플릿 또는 커뮤니티 구성 요소 목록을 검색하십시오.
   - 구성 요소 템플릿을 나열하려면 다음 명령을 실행합니다. 응답의 각 줄에는 템플릿 이름과 프로그래밍 언어가 포함됩니다.

     ```
     gdk component list --template
     ```

   - 커뮤니티 구성 요소를 나열하려면 다음 명령을 실행합니다.

```
gdk component list --repository
```

4. GDK CLI가 템플릿 또는 커뮤니티 구성 요소를 다운로드하는 구성 요소 폴더를 만들고 변경합니다. 구성 요소 이름이나 이 구성 요소 폴더를 식별하는 데 도움이 되는 다른 이름으로 *HelloWorld*바꾸십시오.

   Linux or Unix

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   PowerShell

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

5. 템플릿 또는 커뮤니티 구성 요소를 현재 폴더에 다운로드합니다. 구성 요소 init 명령을 사용합니다.

   - 템플릿에서 구성 요소 폴더를 만들려면 다음 명령을 실행합니다. 템플릿 이름으로 바꾸고, *python#* 프로그래밍 언어 이름으로 바꾸십시오. *HelloWorld*

     ```
     gdk component init --template HelloWorld --language python
     ```

   - 커뮤니티 구성 요소에서 구성 요소 폴더를 만들려면 다음 명령을 실행합니다. 커뮤니티 구성 요소의 *ComponentName*이름으로 바꾸십시오.

     ```
     gdk component init --repository ComponentName
     ```

   > **Note**
   >
   > GDK CLI v1.0.0을 사용하는 경우 빈 폴더에서 이 명령을 실행해야 합니다. GDK CLI는 템플릿 또는 커뮤니티 구성 요소를 현재 폴더에 다운로드합니다.

GDK CLI v1.1.0 이상을 사용하는 경우 인수를 지정하여 GDK `--name` CLI가 템플릿 또는 커뮤니티 구성 요소를 다운로드하는 폴더를 지정할 수 있습니다. 이 인수를 사용하는 경우 존재하지 않는 폴더를 지정하십시오. GDK CLI는 폴더를 자동으로 생성합니다. 이 인수를 지정하지 않으면 GDK CLI는 비어 있어야 하는 현재 폴더를 사용합니다.

6. GDK CLI는 `gdk-config.json` 이름이 지정된 GDK CLI 구성 파일을 읽고 구성 요소를 빌드하고 게시합니다. 이 구성 파일은 구성 요소 폴더의 루트에 있습니다. 이전 단계에서 이 파일을 자동으로 생성합니다. 이 단계에서는 구성 요소에 대한 `gdk-config.json` 정보로 업데이트합니다. 다음을 따릅니다.

   a. 텍스트 편집기에서 `gdk-config.json`을 엽니다.

   b. (선택 사항) 구성 요소 이름을 변경합니다. 구성 요소 이름은 `component` 개체의 키입니다.

   c. 구성 요소 작성자를 변경합니다.

   d. (선택 사항) 구성 요소의 버전을 변경합니다. 다음 중 하나를 지정하십시오.

      - `NEXT_PATCH`— 이 옵션을 선택하면 구성 요소를 게시할 때 GDK CLI에서 버전을 설정합니다. GDK CLI는 서비스를 AWS IoT Greengrass 쿼리하여 가장 최근에 게시된 구성 요소 버전을 식별합니다. 그런 다음 버전을 해당 버전 이후의 다음 패치 버전으로 설정합니다. 이전에 구성 요소를 게시한 적이 없는 경우 GDK CLI는 버전을 사용합니다. `1.0.0`

        이 옵션을 선택하면 Greengrass CLI를 사용하여 Core 소프트웨어를 실행하는 로컬 개발 컴퓨터에 구성 요소를 로컬로 배포하고 테스트할 수 없습니다. AWS IoT Greengrass 로컬 배포를 활성화하려면 시맨틱 버전을 대신 지정해야 합니다.

      - 시맨틱 버전 (예:. **`1.0.0`** 시맨틱 버전은 메이저를 사용합니다. 마이너. 패치 넘버링 시스템. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

        구성 요소를 배포하고 테스트하려는 Greengrass 코어 장치에서 구성 요소를 개발하는 경우 이 옵션을 선택합니다. Greengrass CLI로 로컬 배포를 생성하려면 특정 버전으로 구성 요소를 빌드해야 합니다.

   e. (선택 사항) 구성 요소의 빌드 구성을 변경합니다. 빌드 구성은 GDK CLI가 구성 요소 소스를 아티팩트로 빌드하는 방법을 정의합니다. 다음 옵션 중에서 선택하십시오. `build_system`

      - `zip`— 구성 요소의 폴더를 ZIP 파일로 패키지하여 구성 요소의 유일한 아티팩트로 정의합니다. 다음 유형의 구성 요소에 대해 이 옵션을 선택합니다.
        - Python 또는 JavaScript 같은 해석된 프로그래밍 언어를 사용하는 컴포넌트
        - 코드가 아닌 파일을 패키징하는 구성 요소 (예: 기계 학습 모델 또는 기타 리소스).

GDK CLI는 구성 요소 폴더를 구성 요소 폴더와 동일한 이름의 zip 파일로 압축합니다. 예를 들어 구성 요소 폴더 이름이 인 경우 GDK CLI는 `HelloWorld` 라는 zip 파일을 생성합니다. `HelloWorld.zip`

> **Note**
>
> Windows 디바이스에서 GDK CLI 버전 1.0.0을 사용하는 경우 구성 요소 폴더 및 zip 파일 이름에는 소문자만 포함되어야 합니다.

GDK CLI는 구성 요소 폴더를 zip 파일로 압축할 때 다음 파일을 건너뛰습니다.

- `gdk-config.json` 파일
- 레시피 파일 (또는) `recipe.json` `recipe.yaml`
- 빌드 폴더 (예: `greengrass-build`

- `maven`— `mvn clean package` 명령을 실행하여 구성 요소 소스를 아티팩트에 빌드합니다. Java 구성 요소와 같이 Maven을 사용하는 구성 요소에 대해 이 옵션을 선택합니다.

  Windows 디바이스에서 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

- `gradle`— `gradle build` 명령을 실행하여 구성 요소의 소스를 아티팩트로 빌드합니다. Gradle을 사용하는 구성 요소에 대해 이 옵션을 선택합니다. 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

  `gradle`빌드 시스템은 Kotlin DSL을 빌드 파일로 지원합니다. 이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `gradlew`— `gradlew` 명령을 실행하여 구성 요소 소스를 아티팩트로 빌드합니다. Gradle 래퍼를 사용하는 구성 요소에 대해 이 옵션을 선택합니다.

  이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `custom`— 사용자 지정 명령을 실행하여 구성 요소의 소스를 레시피와 아티팩트로 빌드합니다. `custom_build_command`파라미터에 사용자 지정 명령을 지정합니다.

f. 를 지정하는 `custom` `build_system` 경우 `build` 개체에 `custom_build_command` 를 추가합니다. 에서 `custom_build_command` 단일 문자열 또는 문자열 목록을 지정합니다. 여기서 각 문자열은 명령의 한 단어입니다. 예를 들어 C++ 구성 요소에 대한 사용자 지정 빌드 명령을 실행하려면 다음을 지정할 **`["cmake", "--build", "build", "--config", "Release"]`** 수 있습니다.

g. GDK CLI v1.1.0 이상을 사용하는 경우 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 `--bucket` 수 있습니다. 이 인수를 지정하지 않으면 GDK *`bucket`*-*`region`*-*`accountId`* CLI는 이름이 인 S3 버킷에 업로드됩니다. *`###`* *`###`* 지역은 지정된 `gdk-config.json` 값이고 *`accountID#`* ID입니다. AWS 계정 버킷이 없는 경우 GDK CLI에서 버킷을 생성합니다.

   구성 요소의 게시 구성을 변경합니다. 다음을 따릅니다.

   i. 구성 요소 아티팩트를 호스팅하는 데 사용할 S3 버킷의 이름을 지정합니다.

   ii. GDK CLI가 구성 요소를 게시하는 AWS 리전 위치를 지정합니다.

이 단계를 완료하면 `gdk-config.json` 파일이 다음 예제와 비슷하게 보일 수 있습니다.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. 이름이 `recipe.yaml` or인 부품 레시피 파일을 `recipe.json` 업데이트하십시오. 다음을 따릅니다.

   a. `zip`빌드 시스템을 사용하는 템플릿 또는 커뮤니티 구성 요소를 다운로드한 경우 zip 아티팩트 이름이 구성 요소 폴더의 이름과 일치하는지 확인하십시오. GDK CLI는 구성 요소 폴더를 구성 요소 폴더와 이름이 같은 zip 파일로 압축합니다. 레시피에는 구성 요소 아티팩트 목록에 있는 zip 아티팩트 이름과 zip 아티팩트의 파일을 사용하는 라이프 사이클 스크립트에 있는 zip 아티팩트 이름이 포함됩니다. zip 파일 이름이 `Lifecycle` 구성 요소 폴더의 이름과 일치하도록 `Artifacts` 및 정의를 업데이트하십시오. 다음 부분 레시피 예제는 `Artifacts` 및 `Lifecycle` 정의의 zip 파일 이름을 강조 표시합니다.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

b. (선택 사항) 구성 요소 설명, 기본 구성, 아티팩트, 라이프사이클 스크립트 및 플랫폼 지원을 업데이트합니다. 자세한 설명은 AWS IoT Greengrass컴포넌트 레시피 참조 섹션을 참조하세요.

이 단계를 완료하면 레시피 파일이 다음 예제와 비슷해 보일 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "{COMPONENT_NAME}",
  "ComponentVersion": "{COMPONENT_VERSION}",
  "ComponentDescription": "This is a simple Hello World component written in
 Python.",
  "ComponentPublisher": "{COMPONENT_AUTHOR}",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "World"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
```

```
ComponentDescription: "This is a simple Hello World component written in
 Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

8. Greengrass 구성 요소를 개발하고 빌드합니다. 구성 요소 빌드 명령은 구성 요소 폴더의 `greengrass-build` 폴더에 레시피와 아티팩트를 생성합니다. 다음 명령을 실행합니다.

   ```
   gdk component build
   ```

구성 요소를 테스트할 준비가 되면 GDK CLI를 사용하여 서비스에 게시하십시오. AWS IoT Greengrass 그런 다음 Greengrass 코어 디바이스에 컴포넌트를 배포할 수 있습니다. 자세한 설명은 코어 디바이스에 배포할 구성 요소를 게시하세요. 섹션을 참조하세요.

구성 요소 생성 (셸 명령)

이 섹션의 지침에 따라 여러 구성 요소의 소스 코드와 아티팩트가 포함된 레시피 및 아티팩트 폴더를 만드십시오.

Greengrass 구성 요소를 개발하려면 (셸 명령)

1. 레시피와 아티팩트를 위한 하위 폴더가 있는 컴포넌트 폴더를 만드세요. Greengrass 코어 디바이스에서 다음 명령을 실행하여 이러한 폴더를 생성하고 구성 요소 폴더로 변경합니다. *~/greengrassv2* 또는 *%USERPROFILE%\ greengrassv2#* 로컬 개발에 사용할 폴더 경로로 바꾸십시오.

   Linux or Unix

   ```
   mkdir -p ~/greengrassv2/{recipes,artifacts}
   ```

```
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. 텍스트 편집기를 사용하여 구성요소의 메타데이터, 매개변수, 종속성, 수명 주기 및 플랫폼 기능을 정의하는 레시피 파일을 만드십시오. 레시피 파일 이름에 구성 요소 버전을 포함하면 어떤 레시피가 어떤 구성 요소 버전을 반영하는지 식별할 수 있습니다. 레시피에 YAML 또는 JSON 형식을 선택할 수 있습니다.

   예를 들어 Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 생성할 수 있습니다.

   JSON

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   nano recipes/com.example.HelloWorld-1.0.0.yaml
   ```

   > **Note**
   >
   > AWS IoT Greengrass 구성 요소에 시맨틱 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 버전은 구성 `1.0.0` 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

3. 구성 요소의 레시피를 정의하십시오. 자세한 설명은 AWS IoT Greengrass컴포넌트 레시피 참조 섹션을 참조하세요.

   레시피는 다음 Hello World 예제 레시피와 비슷할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
```

```
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

이 레시피는 Hello World Python 스크립트를 실행합니다. 이 스크립트는 다음 예제 스크립트와 비슷할 수 있습니다.

```
import sys

message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

4. 개발할 구성 요소 버전을 위한 폴더를 생성합니다. 각 구성 요소 버전의 아티팩트를 식별할 수 있도록 각 구성 요소 버전의 아티팩트에 별도의 폴더를 사용하는 것이 좋습니다. 다음 명령을 실행합니다.

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

PowerShell

```
mkdir artifacts/com.example.HelloWorld/1.0.0
```

> **Important**
>
> 아티팩트 폴더 경로에는 다음 형식을 사용해야 합니다. 레시피에 지정한 구성 요소 이름과 버전을 포함하십시오.
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. 이전 단계에서 생성한 폴더에 구성 요소의 아티팩트를 생성합니다. 아티팩트에는 소프트웨어, 이미지 및 구성 요소가 사용하는 기타 바이너리가 포함될 수 있습니다.

   구성 요소가 준비되면 구성 요소를 테스트하십시오.

로컬 배포를 통한 테스트 AWS IoT Greengrass 구성 요소

코어 기기에서 Greengrass 구성 요소를 개발하는 경우 로컬 배포를 생성하여 설치 및 테스트할 수 있습니다. 이 섹션의 단계에 따라 로컬 배포를 생성하세요.

로컬 개발 컴퓨터와 같은 다른 컴퓨터에서 구성 요소를 개발하는 경우 로컬 배포를 만들 수 없습니다. 대신 Greengrass 코어 디바이스에 배포하여 테스트할 수 있도록 AWS IoT Greengrass 서비스에 구성 요소를 게시하십시오. 자세한 내용은 코어 디바이스에 배포할 구성 요소를 게시하세요. 및 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

Greengrass 코어 디바이스에서 구성 요소를 테스트하려면

1. 코어 기기는 구성 요소 업데이트와 같은 이벤트를 기록합니다. 이 로그 파일을 보고 잘못된 레시피와 같은 구성 요소 관련 오류를 발견하고 문제를 해결할 수 있습니다. 이 로그 파일에는 구성 요소가 표준 출력 (stdout) 으로 인쇄하는 메시지도 표시됩니다. 새 로그 메시지를 실시간으로 관찰하려면 코어 기기에서 추가 터미널 세션을 여는 것이 좋습니다. 새 터미널 세션을 (예: SSH를 통해) 열고 다음 명령을 실행하여 로그를 확인합니다. AWS IoT Greengrass루트 폴더 */greengrass/v2* 경로로 바꾸십시오.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

구성 요소의 로그 파일도 볼 수 있습니다.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

2. 원래 터미널 세션에서 다음 명령을 실행하여 구성 요소로 코어 장치를 업데이트하십시오. AWS IoT Greengrass루트 폴더의 */greengrass/v2* 경로로 바꾸고 *~/greengrassv2#* 로컬 개발 폴더의 경로로 바꾸십시오.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.example.HelloWorld=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

> **Note**
>
> `greengrass-cli deployment create`명령을 사용하여 구성 요소의 구성 매개 변수 값을 설정할 수도 있습니다. 자세한 설명은 create 섹션을 참조하세요.

3. `greengrass-cli deployment status`명령을 사용하여 구성 요소 배포 진행 상황을 모니터링할 수 있습니다.

   Unix or Linux

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment status \
     -i deployment-id
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment status ^
     -i deployment-id
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment status `
     -i deployment-id
   ```

4. Greengrass 코어 디바이스에서 실행되는 구성 요소를 테스트하십시오. 이 버전의 구성 요소를 완료하면 AWS IoT Greengrass 서비스에 업로드할 수 있습니다. 그런 다음 구성 요소를 다른 코어 디바이스에 배포할 수 있습니다. 자세한 내용은 코어 디바이스에 배포할 구성 요소를 게시하세요.을(를) 참조하세요.

코어 디바이스에 배포할 구성 요소를 게시하세요.

구성 요소 버전을 빌드하거나 완성한 후 AWS IoT Greengrass 서비스에 게시할 수 있습니다. 그런 다음 Greengrass 코어 디바이스에 배포할 수 있습니다.

그린그래스 개발 키트 CLI (GDK CLI) 를 사용하여 구성 요소를 개발하고 구축하는 경우, GDK CLI를 사용하여 구성 요소를 에 게시할 수 있습니다. AWS 클라우드 그렇지 않으면 내장된 셸 명령과 를 사용하여 구성 요소를 게시하십시오. AWS CLI

를 AWS CloudFormation 사용하여 템플릿에서 구성 요소 및 기타 AWS 리소스를 만들 수도 있습니다. 자세한 내용은 What isAWS CloudFormation? 를 참조하십시오. 그리고 AWS::GreengrassV2::ComponentVersionAWS CloudFormation사용자 안내서에서

주제

- 구성 요소 게시 (GDK CLI)
- 구성 요소 게시 (셸 명령)

구성 요소 게시 (GDK CLI)

이 섹션의 지침에 따라 GDK CLI를 사용하여 구성 요소를 게시하십시오. GDK CLI는 빌드 아티팩트를 S3 버킷에 업로드하고, 레시피의 아티팩트 URI를 업데이트하고, 레시피에서 구성 요소를 생성합니다. GDK CLI 구성 파일에서 사용할 S3 버킷과 지역을 지정합니다.

GDK CLI v1.1.0 이상을 사용하는 경우 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 `--bucket` 수 있습니다. 이 인수를 지정하지 않으면 GDK *`bucket`*`-`*`region`*`-`*`accountId`* CLI는 이름이 인 S3 버킷에 업로드됩니다. *`###`* *`###`* 지역은 지정된 `gdk-config.json` 값이고 *`accountID#`* ID입니다. AWS 계정 버킷이 없는 경우 GDK CLI에서 버킷을 생성합니다.

> **Important**
>
> 코어 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않습니다. 이 S3 버킷을 처음 사용하는 경우, 코어 디바이스가 이 S3 버킷에서 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 설명은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용 섹션을 참조하세요.

그린그래스 컴포넌트 (GDK CLI) 를 게시하려면

1. 명령 프롬프트 또는 터미널에서 구성 요소 폴더를 엽니다.
2. 아직 빌드하지 않았다면 Greengrass 컴포넌트를 빌드하세요. 구성 요소 빌드 명령은 구성 요소 폴더의 `greengrass-build` 폴더에 레시피와 아티팩트를 생성합니다. 다음 명령을 실행합니다.

   ```
   gdk component build
   ```

3. 구성 요소를 에 AWS 클라우드 게시합니다. 구성 요소 게시 명령은 구성 요소의 아티팩트를 Amazon S3에 업로드하고 각 아티팩트의 URI로 구성 요소의 레시피를 업데이트합니다. 그런 다음 서비스에 구성 요소를 생성합니다. AWS IoT Greengrass

   > **Note**
   >
   > AWS IoT Greengrass구성 요소를 만들 때 각 아티팩트의 다이제스트를 계산합니다. 즉, 구성 요소를 생성한 후에는 S3 버킷의 아티팩트 파일을 수정할 수 없습니다. 이렇게 하면 파일 다이제스트가 일치하지 않아 이 구성 요소를 포함하는 배포가 실패합니다. 아티팩트 파일을 수정하는 경우 구성 요소의 새 버전을 만들어야 합니다.

   GDK CLI 구성 파일에서 구성 요소 버전을 지정하는 `NEXT_PATCH` 경우 GDK CLI는 서비스에 아직 없는 다음 패치 버전을 사용합니다. AWS IoT Greengrass

   다음 명령을 실행합니다.

   ```
   gdk component publish
   ```

   출력은 GDK CLI에서 생성한 구성 요소의 버전을 알려줍니다.

   구성 요소를 게시한 후 구성 요소를 코어 장치에 배포할 수 있습니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

구성 요소 게시 (셸 명령)

셸 명령과 AWS Command Line Interface (AWS CLI) 를 사용하여 구성 요소를 게시하려면 다음 절차를 따르십시오. 구성 요소를 게시할 때 다음 작업을 수행합니다.

1. 구성 요소 아티팩트를 S3 버킷에 게시합니다.

2. 각 아티팩트의 Amazon S3 URI를 구성 요소 레시피에 추가합니다.
3. 구성 요소 AWS IoT Greengrass 레시피에서 구성 요소 버전을 생성합니다.

> **Note**
>
> 업로드하는 각 구성 요소 버전은 고유해야 합니다. 구성 요소 버전을 업로드한 후에는 편집할 수 없으므로 올바른 구성 요소 버전을 업로드해야 합니다.

다음 단계에 따라 개발 컴퓨터 또는 Greengrass 코어 디바이스에서 구성 요소를 게시할 수 있습니다.

구성 요소를 게시하려면 (셸 명령)

1. 구성 요소가 AWS IoT Greengrass 서비스에 있는 버전을 사용하는 경우 구성 요소의 버전을 변경해야 합니다. 텍스트 편집기에서 레시피를 열고 버전을 증가시킨 다음 파일을 저장합니다. 구성 요소의 변경 사항을 반영하는 새 버전을 선택합니다.

   > **Note**
   >
   > AWS IoT Greengrass구성 요소에 시맨틱 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 버전은 구성 `1.0.0` 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

2. 구성 요소에 아티팩트가 있는 경우 다음을 수행하십시오.

   a. 구성 요소의 아티팩트를 의 S3 버킷에 게시하십시오. AWS 계정

      > **Tip**
      >
      > S3 버킷의 아티팩트 경로에 구성 요소 이름과 버전을 포함하는 것이 좋습니다. 이 이름 지정 체계를 사용하면 이전 버전의 구성 요소가 사용하는 아티팩트를 유지 관리할 수 있으므로 이전 구성 요소 버전을 계속 지원할 수 있습니다.

      다음 명령을 실행하여 S3 버킷에 아티팩트 파일을 게시합니다. *DOC-EXAMPLE-BUCKET# ## #### ### ####/com.example# ######. HelloWorld*/1.0.0/artifact.py를 아티팩트 파일의 경로로 입력합니다.

```
aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

> **Important**
>
> 코어 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않습니다. 이 S3 버킷을 처음 사용하는 경우, 코어 디바이스가 이 S3 버킷에서 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 설명은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용 섹션을 참조하세요.

b. 이름이 지정된 `Artifacts` 목록이 없는 경우 구성 요소 레시피에 해당 목록을 추가하십시오. `Artifacts`목록은 각 매니페스트에 나타나며, 지원되는 각 플랫폼의 구성 요소 요구 사항(또는 모든 플랫폼에 대한 구성 요소의 기본 요구 사항) 을 정의합니다.

c. 각 아티팩트를 아티팩트 목록에 추가하거나 기존 아티팩트의 URI를 업데이트하십시오. Amazon S3 URI는 버킷 이름과 버킷 내 아티팩트 객체 경로로 구성됩니다. 아티팩트의 Amazon S3 URI는 다음 예제와 비슷해야 합니다.

```
s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

이 단계를 완료하면 레시피에 다음과 같은 `Artifacts` 목록이 표시되어야 합니다.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
```

```
  ]
}
```

> **Note**
>
> ZIP 아티팩트 `"Unarchive": "ZIP"` 옵션을 추가하여 구성 요소가 배포될 때 아티팩트의 압축을 풀도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수 있습니다.

YAML

```
...
Manifests:
  - Lifecycle:
      ...
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/artifact.py
        Unarchive: NONE
```

> **Note**
>
> 이 `Unarchive: ZIP` 옵션을 사용하여 구성 요소를 배포할 때 ZIP 아티팩트의 압축을 풀도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수 있습니다. 구성 요소에서 ZIP 아티팩트를 사용하는 방법에 대한 자세한 내용은 Artifacts:DecompressedPath 레시피 변수를 참조하십시오.

레시피에 대한 자세한 내용은 AWS IoT Greengrass컴포넌트 레시피 참조 단원을 참조하십시오.

3. AWS IoT Greengrass콘솔을 사용하여 레시피 파일에서 컴포넌트를 생성할 수 있습니다.

   다음 명령을 실행하여 레시피 파일에서 컴포넌트를 생성합니다. 이 명령은 구성 요소를 만든 다음 해당 구성 요소를 사용자의 AWS 계정 개인 AWS IoT Greengrass 구성 요소로 게시합니다. *Path/to/RecipeFile# ### ###* 경로로 바꾸십시오.

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://path/to/recipeFile
   ```

arn응답에서 를 복사하여 다음 단계에서 구성 요소의 상태를 확인합니다.

> **Note**
>
> AWS IoT Greengrass구성 요소를 만들 때 각 아티팩트의 다이제스트를 계산합니다. 즉, 구성 요소를 생성한 후에는 S3 버킷의 아티팩트 파일을 수정할 수 없습니다. 이렇게 하면 파일 다이제스트가 일치하지 않아 이 구성 요소를 포함하는 배포가 실패합니다. 아티팩트 파일을 수정하는 경우 구성 요소의 새 버전을 만들어야 합니다.

4. AWS IoT Greengrass서비스의 각 구성 요소에는 상태가 있습니다. 다음 명령을 실행하여 이 절차에서 게시한 구성 요소 버전의 상태를 확인합니다. *com.example# ######. HelloWorld*쿼리할 구성 요소 버전으로 *1.0.0#* 입력합니다. 를 이전 단계의 arn ARN으로 교체합니다.

```
aws greengrassv2 describe-component --arn "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorld:versions:1.0.0"
```

이 작업은 구성 요소의 메타데이터가 포함된 응답을 반환합니다. 메타데이터에는 구성 요소 상태 및 오류 (해당하는 경우) 가 포함된 status 개체가 포함됩니다.

구성 요소 상태가 DEPLOYABLE 이면 구성 요소를 장치에 배포할 수 있습니다. 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포을(를) 참조하세요.

AWS서비스와 상호작용

Greengrass 코어 디바이스는 X.509 인증서를 사용하여 TLS 상호 인증 프로토콜을 AWS IoT Core 사용하여 연결합니다. 이러한 인증서를 사용하면 일반적으로 액세스 키 ID와 비밀 액세스 키로 구성된 AWS 자격 증명 AWS IoT 없이 기기와 상호 작용할 수 있습니다. AWS서비스 엔드포인트에서 API 작업을 호출하려면 X.509 인증서 대신 AWS 자격 증명이 필요한 서비스도 있습니다. AWS IoT Core디바이스에서 X.509 인증서를 사용하여 요청을 인증할 수 있도록 하는 자격 증명 공급자가 있습니다. AWS AWS IoT자격 증명 공급자는 X.509 인증서를 사용하여 장치를 인증하고 권한이 제한된 임시 보안 토큰의 형태로 AWS 자격 증명을 발급합니다. 디바이스는 이 토큰을 사용하여 모든 요청에 서명하고 인증할 수 있습니다. AWS 따라서 Greengrass 코어 디바이스에 AWS 자격 증명을 저장할 필요가 없습니다. 자세한 내용은 AWS IoT Core개발자 안내서의 AWS서비스에 대한 다이렉트 콜 승인을 참조하십시오.

Greengrass에서 AWS IoT 자격 증명을 가져오기 위해 코어 디바이스는 AWS IoT IAM 역할을 가리키는 역할 별칭을 사용합니다. 이 IAM 역할을 토큰 교환 역할이라고 합니다. AWS IoT GreengrassCore

소프트웨어를 설치할 때 역할 별칭과 토큰 교환 역할을 생성합니다. 코어 디바이스에서 사용하는 역할 별칭을 지정하려면 의 `iotRoleAlias` 파라미터를 구성하십시오. 그린그래스 핵

AWS IoT자격 증명 공급자는 사용자를 대신하여 토큰 교환 역할을 맡아 코어 장치에 AWS 자격 증명을 제공합니다. 이 역할에 적절한 IAM 정책을 추가하여 코어 디바이스가 S3 버킷의 구성 요소 아티팩트와 같은 AWS 리소스에 액세스하도록 허용할 수 있습니다. 토큰 교환 역할을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

Greengrass 코어 디바이스는 AWS 자격 증명을 메모리에 저장하며 자격 증명은 기본적으로 1시간 후에 만료됩니다. AWS IoT GreengrassCore 소프트웨어를 다시 시작하면 자격 증명을 다시 가져와야 합니다. UpdateRoleAlias작업을 사용하여 자격 증명의 유효 기간을 구성할 수 있습니다.

AWS IoT Greengrass공개 구성 요소인 토큰 교환 서비스 구성 요소를 제공합니다. 이 구성 요소는 서비스와 상호 작용하기 위한 사용자 지정 구성 요소에 종속성으로 정의할 수 있습니다. AWS 토큰 교환 서비스는 AWS 자격 증명을 제공하는 로컬 서버에 URI를 정의하는 환경 변수를 구성 요소에 제공합니다. `AWS_CONTAINER_CREDENTIALS_FULL_URI` AWSSDK 클라이언트를 만들면 클라이언트는 이 환경 변수를 확인하고 로컬 서버에 연결하여 AWS 자격 증명을 검색하고 이를 사용하여 API 요청에 서명합니다. 이렇게 하면 AWS SDK 및 기타 도구를 사용하여 구성 요소의 AWS 서비스를 호출할 수 있습니다. 자세한 설명은 토큰 교환 서비스 섹션을 참조하세요.

> **Important**
>
> 이러한 방식의 AWS 자격 증명 취득 지원이 2016년 7월 13일에 AWS SDK에 추가되었습니다. 구성 요소는 해당 날짜 또는 그 이후에 생성된 AWS SDK 버전을 사용해야 합니다. 자세한 내용은 Amazon Elastic 컨테이너 서비스 개발자 안내서의 지원되는 AWS SDK 사용을 참조하십시오.

사용자 지정 구성 요소의 AWS 자격 증명을 얻으려면 구성 요소 `aws.greengrass.TokenExchangeService` 레시피에서 종속성을 정의하십시오. 다음 예제 레시피는 boto3를 설치하는 구성 요소를 정의하고 토큰 교환 서비스의 AWS 자격 증명을 사용하여 Amazon S3 버킷을 나열하는 Python 스크립트를 실행합니다.

> **Note**
>
> 이 예제 구성 요소를 실행하려면 디바이스에 권한이 있어야 합니다. `s3:ListAllMyBuckets` 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list
 S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
```

```
ComponentDescription: A component that uses the token exchange service to list S3
 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py
```

이 예제 구성 요소는 Amazon S3 버킷을 `list_s3_buckets.py` 나열하는 다음 Python 스크립트를 실행합니다.

```
import boto3
import os

try:
    print("Creating boto3 S3 client...")
    s3 = boto3.client('s3')
    print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
```

```
    print("Successfully listed S3 buckets")
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

도커 컨테이너 실행

다음 위치에 저장된 이미지에서 Docker 컨테이너를 실행하도록 AWS IoT Greengrass 구성 요소를 구성할 수 있습니다.

- 아마존 Elastic 컨테이너 레지스트리 (Amazon ECR) 의 퍼블릭 및 프라이빗 이미지 리포지토리
- 퍼블릭 도커 허브 리포지토리
- 퍼블릭 도커 신뢰할 수 있는 레지스트리
- S3 버킷

사용자 지정 구성 요소에 Docker 이미지 URI를 아티팩트로 포함하여 이미지를 검색하고 코어 디바이스에서 실행합니다. Amazon ECR 및 Docker Hub 이미지의 경우 Docker 애플리케이션 관리자 구성 요소를 사용하여 프라이빗 Amazon ECR 리포지토리의 이미지를 다운로드하고 자격 증명을 관리할 수 있습니다.

주제

- 요구 사항
- Amazon ECR 또는 Docker Hub의 공개 이미지에서 도커 컨테이너 실행
- Amazon ECR의 프라이빗 이미지에서 Docker 컨테이너 실행
- Amazon S3의 이미지에서 도커 컨테이너 실행
- Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.
- Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용
- Docker 컨테이너 구성 요소 (Linux) 에서 스트림 관리자 사용

요구 사항

구성 요소에서 Docker 컨테이너를 실행하려면 다음이 필요합니다.

- 그린그래스 코어 디바이스. 계정이 없는 경우 자습서: AWS IoT Greengrass V2 시작하기 단원을 참조하십시오.

- 도커 엔진 1.9.1 이상이 그린그래스 코어 디바이스에 설치되었습니다. 버전 20.10은 Core 소프트웨어와 호환이 검증된 최신 버전입니다. AWS IoT Greengrass Docker 컨테이너를 실행하는 구성 요소를 배포하려면 먼저 코어 디바이스에 Docker를 직접 설치해야 합니다.

  > **Tip**
  >
  > 구성 요소가 설치될 때 Docker Engine을 설치하도록 코어 디바이스를 구성할 수도 있습니다. 예를 들어 다음 설치 스크립트는 Docker 이미지를 로드하기 전에 Docker 엔진을 설치합니다. 이 설치 스크립트는 Ubuntu와 같은 데비안 기반 Linux 배포판에서 작동합니다. 이 명령으로 Docker Engine을 설치하도록 구성 요소를 구성하는 경우 설치 및 명령을 실행하려면 라이프사이클 `true` 스크립트에서 `RequiresPrivilege` 로 설정해야 할 수 있습니다. `docker` 자세한 설명은 AWS IoT Greengrass컴포넌트 레시피 참조 섹션을 참조하세요.
  >
  > ```
  > apt-get install docker-ce docker-ce-cli containerd.io && docker load -i
  >  {artifacts:path}/hello-world.tar
  > ```

- Docker 컨테이너 구성 요소를 실행하는 시스템 사용자에게 루트 또는 관리자 권한이 있거나 루트 또는 관리자가 아닌 사용자로 실행하도록 Docker를 구성해야 합니다.
  - Linux 디바이스에서는 `docker` 그룹에 사용자를 추가하여 없이 `docker` `sudo` 명령을 호출할 수 있습니다.
  - Windows 장치에서는 관리자 권한 없이 `docker` 명령을 호출할 사용자를 `docker-users` 그룹에 추가할 수 있습니다.

  Linux or Unix

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 루트가 아닌 사용자를 `docker` 그룹에 추가하려면 `ggc_user` 다음 명령어를 실행합니다.

  ```
  sudo usermod -aG docker ggc_user
  ```

  자세한 내용은 루트가 아닌 사용자로 Docker 관리를 참조하십시오.

  Windows Command Prompt (CMD)

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 사용자를 `docker-users` 그룹에 추가하려면 `ggc_user` 관리자로 다음 명령을 실행합니다.

  ```
  net localgroup docker-users ggc_user /add
  ```

Windows PowerShell

그룹에 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 사용자를 `docker-users` 그룹에 추가하려면 `ggc_user` 관리자로 다음 명령을 실행합니다.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Docker 컨테이너 구성 요소에서 액세스한 파일은 Docker 컨테이너에 볼륨으로 마운트됩니다.
- 네트워크 프록시를 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성하는 경우 동일한 프록시 서버를 사용하도록 Docker를 구성해야 합니다.

환경에 적용되는 경우 이러한 요구 사항 외에도 다음 요구 사항도 충족해야 합니다.

- Docker Compose를 사용하여 Docker 컨테이너를 생성하고 시작하려면 Greengrass 코어 디바이스에 Docker Compose를 설치하고 Docker Compose 파일을 S3 버킷에 업로드하세요. Compose 파일은 구성 요소와 동일한 S3 버킷에 저장해야 합니다. AWS 계정 AWS 리전 사용자 지정 구성 요소에서 `docker-compose up` 명령을 사용하는 예제는 을 참조하십시오Amazon ECR 또는 Docker Hub의 공개 이미지에서 도커 컨테이너 실행.
- 네트워크 프록시 AWS IoT Greengrass 뒤에서 실행하는 경우 프록시 서버를 사용하도록 Docker 데몬을 구성하십시오.
- Docker 이미지가 Amazon ECR 또는 Docker Hub에 저장되어 있는 경우 Docker 구성 요소 관리자 구성 요소를 Docker 컨테이너 구성 요소의 종속 항목으로 포함하십시오. 구성 요소를 배포하기 전에 코어 디바이스에서 Docker 데몬을 시작해야 합니다.

  또한 이미지 URI를 구성 요소 아티팩트로 포함하세요. 이미지 URI는 다음 `docker:registry/image[:tag|@digest]` 예와 같은 형식이어야 합니다.

  - 프라이빗 아마존 ECR 이미지: `docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
  - 퍼블릭 아마존 ECR 이미지: `docker:public.ecr.aws/repository/image[:tag|@digest]`
  - 퍼블릭 도커 허브 이미지: `docker:name[:tag|@digest]`

  공개 리포지토리에 저장된 이미지에서 Docker 컨테이너를 실행하는 방법에 대한 자세한 내용은 을 참조하십시오. Amazon ECR 또는 Docker Hub의 공개 이미지에서 도커 컨테이너 실행
- Docker 이미지가 Amazon ECR 사설 리포지토리에 저장되어 있는 경우 토큰 교환 서비스 구성 요소를 Docker 컨테이너 구성 요소에 종속 항목으로 포함해야 합니다. 또한 Greengrass 장치 역할은 다

음 예제 IAM `ecr:GetAuthorizationToken` 정책에 `ecr:BatchGetImage` 나와 있는 것처럼,, 및 `ecr:GetDownloadUrlForLayer` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Amazon ECR 사설 리포지토리에 저장된 이미지에서 Docker 컨테이너를 실행하는 방법에 대한 자세한 내용은 을 참조하십시오. Amazon ECR의 프라이빗 이미지에서 Docker 컨테이너 실행

- Amazon ECR 프라이빗 리포지토리에 저장된 Docker 이미지를 사용하려면 프라이빗 리포지토리가 코어 AWS 리전 디바이스와 동일한 위치에 있어야 합니다.
- Docker 이미지 또는 Compose 파일이 S3 버킷에 저장되어 있는 경우, Greengrass 디바이스 역할은 다음 예시 IAM 정책에 나와 있는 것처럼 코어 디바이스가 이미지를 구성 요소 아티팩트로 다운로드하도록 허용하는 `s3:GetObject` 권한을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
```

```
}
```

Amazon S3에 저장된 이미지에서 Docker 컨테이너를 실행하는 방법에 대한 자세한 내용은 을 참조하십시오Amazon S3의 이미지에서 도커 컨테이너 실행.

- Docker 컨테이너 구성 요소에서 IPC (프로세스 간 통신), AWS 자격 증명 또는 스트림 관리자를 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음 자료를 참조하세요.
  - Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.
  - Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용
  - Docker 컨테이너 구성 요소 (Linux) 에서 스트림 관리자 사용

Amazon ECR 또는 Docker Hub의 공개 이미지에서 도커 컨테이너 실행

이 섹션에서는 Amazon ECR 및 Docker Hub에 저장된 Docker 이미지에서 Docker Compose를 사용하여 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 생성하는 방법을 설명합니다.

Docker Compose를 사용하여 Docker 컨테이너를 실행하려면

1. Docker Compose 파일을 생성하여 Amazon S3 버킷에 업로드합니다. Greengrass 장치 역할이 장치가 Compose 파일에 액세스할 수 있도록 하는 `s3:GetObject` 권한을 허용하는지 확인하세요. 다음 예제에 표시된 예제 작성 파일에는 Amazon ECR의 Amazon CloudWatch 에이전트 이미지와 Docker Hub의 MySQL 이미지가 포함되어 있습니다.

   ```
   version: "3"
   services:
     cloudwatchagent:
       image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
     mysql:
       image: "mysql:8.0"
   ```

2. 코어 디바이스에 사용자 지정 구성 요소를 생성합니다. AWS IoT Greengrass 다음 예제에 표시된 예제 레시피에는 다음과 같은 속성이 있습니다.

   - 종속성으로서의 Docker 애플리케이션 관리자 구성 요소. 이 구성 요소를 사용하면 AWS IoT Greengrass 퍼블릭 Amazon ECR 및 Docker Hub 리포지토리에서 이미지를 다운로드할 수 있습니다.
   - 퍼블릭 Amazon ECR 리포지토리의 Docker 이미지를 지정하는 구성 요소 아티팩트입니다.
   - 퍼블릭 Docker Hub 리포지토리의 Docker 이미지를 지정하는 구성 요소 아티팩트입니다.

- 실행하려는 Docker 이미지의 컨테이너가 포함된 Docker Compose 파일을 지정하는 구성 요소 아티팩트입니다.
- docker-compose up을 사용하여 지정된 이미지에서 컨테이너를 만들고 시작하는 라이프사이클 실행 스크립트입니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyDockerComposeComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses Docker Compose to run images
 from public Amazon ECR and Docker Hub.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
        }
      ]
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
```

Note

Docker 컨테이너 구성 요소에서 IPC (프로세스 간 통신), AWS 자격 증명 또는 스트림 관리자를 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음 자료를 참조하세요.

- Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.
- Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용
- Docker 컨테이너 구성 요소 (Linux) 에서 스트림 관리자 사용

3. 구성 요소를 테스트하여 예상대로 작동하는지 확인합니다.

Important

구성 요소를 배포하기 전에 Docker 데몬을 설치하고 시작해야 합니다.

구성 요소를 로컬에 배포한 후 docker container ls 명령을 실행하여 컨테이너가 실행되는지 확인할 수 있습니다.

```
docker container ls
```

4. 구성 요소가 준비되면 구성 요소를 AWS IoT Greengrass 업로드하여 다른 코어 장치에 배포하십시오. 자세한 설명은 코어 디바이스에 배포할 구성 요소를 게시하세요. 섹션을 참조하세요.

Amazon ECR의 프라이빗 이미지에서 Docker 컨테이너 실행

이 섹션에서는 Amazon ECR의 프라이빗 리포지토리에 저장된 Docker 이미지에서 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 생성하는 방법을 설명합니다.

Docker 컨테이너를 실행하려면

1. AWS IoT Greengrass코어 디바이스에서 사용자 지정 구성 요소를 생성합니다. 다음과 같은 속성을 가진 다음 예제 레시피를 사용하세요.

   - 종속성으로서의 Docker 애플리케이션 관리자 구성 요소. 이 구성 요소를 사용하면 개인 AWS IoT Greengrass 리포지토리에서 이미지를 다운로드하기 위한 자격 증명을 관리할 수 있습니다.
   - 토큰 교환 서비스 구성 요소는 종속성입니다. 이 구성 요소를 사용하면 AWS 자격 증명을 AWS IoT Greengrass 검색하여 Amazon ECR과 상호 작용할 수 있습니다.
   - 프라이빗 Amazon ECR 리포지토리의 Docker 이미지를 지정하는 구성 요소 아티팩트입니다.
   - docker run을 사용하여 이미지에서 컨테이너를 만들고 시작하는 라이프사이클 실행 스크립트입니다.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyPrivateDockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from a
    private Amazon ECR image.",
     "ComponentPublisher": "Amazon",
     "ComponentDependencies": {
       "aws.greengrass.DockerApplicationManager": {
   ```

```
      "VersionRequirement": "~2.0.0"
    },
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
 Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
```

```
        run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]"
```

> **Note**
>
> Docker 컨테이너 구성 요소에서 IPC (프로세스 간 통신), AWS 자격 증명 또는 스트림 관리자를 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음 자료를 참조하세요.
>
> - Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.
> - Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용
> - Docker 컨테이너 구성 요소 (Linux) 에서 스트림 관리자 사용

2. 구성 요소를 테스트하여 예상대로 작동하는지 확인합니다.

> **Important**
>
> 구성 요소를 배포하기 전에 Docker 데몬을 설치하고 시작해야 합니다.

구성 요소를 로컬에 배포한 후 docker container ls 명령을 실행하여 컨테이너가 실행되는지 확인할 수 있습니다.

```
docker container ls
```

3. 구성 요소를 AWS IoT Greengrass 업로드하여 다른 코어 디바이스에 배포합니다. 자세한 설명은 코어 디바이스에 배포할 구성 요소를 게시하세요. 섹션을 참조하세요.

Amazon S3의 이미지에서 도커 컨테이너 실행

이 단원에서는 Amazon S3에 저장된 Docker 이미지를 사용하여 구성 요소에서 Docker 컨테이너를 실행하는 방법을 설명합니다.

Amazon S3의 이미지에서 구성 요소의 Docker 컨테이너를 실행하려면

1. docker save 명령을 실행하여 도커 컨테이너의 백업을 생성합니다. 이 백업을 컨테이너를 실행할 구성 요소 아티팩트로 제공합니다. AWS IoT Greengrass *hello-world#* 이미지 이름으로 바꾸고 *hello-world.tar* 를 생성할 아카이브 파일 이름으로 바꾸십시오.

   ```
   docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
   ```

2. 코어 디바이스에서 사용자 지정 구성 요소를 만드세요. AWS IoT Greengrass 다음과 같은 속성을 가진 다음 예제 레시피를 사용하세요.

   - docker load를 사용하여 아카이브에서 Docker 이미지를 로드하는 라이프사이클 설치 스크립트입니다.
   - docker run을 사용하여 이미지에서 컨테이너를 만들고 시작하는 라이프사이클 실행 스크립트입니다. 이 `--rm` 옵션은 컨테이너가 종료될 때 컨테이너를 정리합니다.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": {
             "Script": "docker load -i {artifacts:path}/hello-world.tar"
           },
           "run": {
             "Script": "docker run --rm hello-world"
           }
         }
       }
     ]
   ```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
```

> **Note**
>
> Docker 컨테이너 구성 요소에서 IPC (프로세스 간 통신), AWS 자격 증명 또는 스트림 관리자를 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음 자료를 참조하세요.
>
> - Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.
> - Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용
> - Docker 컨테이너 구성 요소 (Linux) 에서 스트림 관리자 사용

3. 구성 요소를 테스트하여 예상대로 작동하는지 확인합니다.

   구성 요소를 로컬에 배포한 후 docker container ls 명령을 실행하여 컨테이너가 실행되는지 확인할 수 있습니다.

   ```
   docker container ls
   ```

4. 구성 요소가 준비되면 Docker 이미지 아카이브를 S3 버킷에 업로드하고 해당 URI를 구성 요소 레시피에 추가합니다. 그런 다음 구성 요소를 업로드하여 다른 코어 AWS IoT Greengrass 디바이스

에 배포할 수 있습니다. 자세한 설명은 코어 디바이스에 배포할 구성 요소를 게시하세요. 섹션을 참조하세요.

완료하면 구성 요소 레시피가 다음 예제와 같아야 합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from an
 image in an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": {
          "Script": "docker load -i {artifacts:path}/hello-world.tar"
        },
        "run": {
          "Script": "docker run --rm hello-world"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
```

```
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.

의 Greengrass 프로세스 간 통신 (IPC) 라이브러리를 사용하여 Greengrass 핵, 기타 Greengrass 구성 요소 등과 AWS IoT Device SDK 통신할 수 있습니다. AWS IoT Core 자세한 설명은 AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core 섹션을 참조하세요.

Docker 컨테이너 구성 요소에서 IPC를 사용하려면 다음 파라미터를 사용하여 Docker 컨테이너를 실행해야 합니다.

- IPC 소켓을 컨테이너에 마운트합니다. Greengrass 핵은 환경 변수에 IPC 소켓 파일 경로를 제공합니다. `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`
- `SVCUID`및 `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` 환경 변수를 Greengrass 핵이 구성요소에 제공하는 값으로 설정합니다. 구성 요소는 이러한 환경 변수를 사용하여 Greengrass 핵에 대한 연결을 인증합니다.

Example 예제 레시피: MQTT 메시지를 AWS IoT Core (Python) 에 게시하기

다음 레시피는 MQTT 메시지를 게시하는 예제 Docker 컨테이너 구성 요소를 정의합니다. AWS IoT Core 이 레시피의 속성은 다음과 같습니다.

- 구성 요소가 모든 주제에 대해 MQTT 메시지를 게시할 수 있도록 하는 권한 부여 정책 (`accessControl`). AWS IoT Core 자세한 내용은 AWS IoT CoreMQTT IPC 인증을 참조하십시오구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오..
- Docker 이미지를 Amazon S3의 TAR 아카이브로 지정하는 구성 요소 아티팩트입니다.

- TAR 아카이브에서 Docker 이미지를 로드하는 수명 주기 설치 스크립트.
- 이미지에서 Docker 컨테이너를 실행하는 라이프사이클 실행 스크립트. Docker 실행 명령에는 다음과 같은 인수가 포함됩니다.
  - `-v`인수는 그린그래스 IPC 소켓을 컨테이너에 마운트합니다.
  - 처음 두 `-e` 인수는 Docker 컨테이너에 필요한 환경 변수를 설정합니다.
  - 추가 `-e` 인수는 이 예제에서 사용하는 환경 변수를 설정합니다.
  - `--rm`인수는 컨테이너가 종료될 때 컨테이너를 정리합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
 message to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
            "policyDescription": "Allows access to publish to IoT Core on all
 topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
```

```
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC=
\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
          resources:
            - '*'
Manifests:
```

```
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용

토큰 교환 서비스 구성 요소를 사용하여 Greengrass 구성 요소의 AWS 서비스와 상호 작용할 수 있습니다. 이 구성 요소는 로컬 컨테이너 서버를 사용하는 코어 디바이스의 토큰 교환 역할의 AWS 자격 증명을 제공합니다. 자세한 설명은 AWS서비스와 상호작용 섹션을 참조하세요.

> **Note**
>
> 이 섹션의 예제는 Linux 코어 디바이스에서만 작동합니다.

Docker 컨테이너 구성 요소에서 토큰 교환 서비스의 AWS 자격 증명을 사용하려면 다음 매개 변수를 사용하여 Docker 컨테이너를 실행해야 합니다.

- 인수를 사용하여 호스트 네트워크에 대한 액세스를 제공합니다. `--network=host` 이 옵션을 사용하면 Docker 컨테이너를 로컬 토큰 교환 서비스에 연결하여 AWS 자격 증명을 검색할 수 있습니다. 이 인수는 리눅스용 Docker에서만 작동합니다.

  > **Warning**
  >
  > 이 옵션을 사용하면 컨테이너가 호스트의 모든 로컬 네트워크 인터페이스에 액세스할 수 있으므로 이 옵션은 호스트 네트워크에 대한 액세스 권한 없이 Docker 컨테이너를 실행하는 경

우보다 덜 안전합니다. 이 옵션을 사용하는 Docker 컨테이너 구성 요소를 개발하고 실행할 때는 이 점을 고려하세요. 자세한 내용은 Docker 설명서의 네트워크: 호스트를 참조하십시오.

- `AWS_CONTAINER_CREDENTIALS_FULL_URI`및 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 환경 변수를 Greengrass 핵이 구성요소에 제공하는 값으로 설정합니다. AWS SDK는 이러한 환경 변수를 사용하여 AWS 자격 증명을 검색합니다.

Example 예제 레시피: Docker 컨테이너 구성 요소의 S3 버킷 나열 (Python)

다음 레시피는 사용자의 S3 버킷을 나열하는 예제 Docker 컨테이너 구성 요소를 정의합니다. AWS 계정 이 레시피의 속성은 다음과 같습니다.

- 토큰 교환 서비스 구성 요소는 종속성입니다. 이러한 종속성을 통해 구성 요소는 AWS 자격 증명을 검색하여 다른 AWS 서비스와 상호 작용할 수 있습니다.
- Amazon S3의 Docker 이미지를 tar 아카이브로 지정하는 구성 요소 아티팩트입니다.
- TAR 아카이브에서 Docker 이미지를 로드하는 수명 주기 설치 스크립트.
- 이미지에서 Docker 컨테이너를 실행하는 라이프사이클 실행 스크립트. Docker 실행 명령에는 다음과 같은 인수가 포함됩니다.
  - `--network=host`인수는 컨테이너가 호스트 네트워크에 액세스할 수 있도록 하므로 컨테이너가 토큰 교환 서비스에 연결할 수 있습니다.
  - `-e`인수는 Docker 컨테이너에 필요한 환경 변수를 설정합니다.
  - `--rm`인수는 컨테이너가 종료될 때 컨테이너를 정리합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3
 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
```

```
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
 AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
```

```
            --rm list-s3-buckets
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

Docker 컨테이너 구성 요소 (Linux) 에서 스트림 관리자 사용

스트림 관리자 구성 요소를 사용하여 Greengrass 구성 요소의 데이터 스트림을 관리할 수 있습니다. 이 구성 요소를 사용하면 데이터 스트림을 처리하고 대용량 IoT 데이터를 로 전송할 수 있습니다. AWS 클라우드 AWS IoT Greengrass스트림 관리자 구성 요소와 상호 작용하는 데 사용하는 스트림 관리자 SDK를 제공합니다. 자세한 설명은 Greengrass 코어 디바이스의 데이터 스트림 관리 섹션을 참조하세요.

> **Note**
>
> 이 섹션의 예제는 Linux 코어 디바이스에서만 작동합니다.

Docker 컨테이너 구성 요소에서 스트림 관리자 SDK를 사용하려면 다음 매개 변수를 사용하여 Docker 컨테이너를 실행해야 합니다.

- 인수를 사용하여 호스트 네트워크에 대한 액세스를 제공합니다. `--network=host` 이 옵션을 사용하면 Docker 컨테이너가 로컬 TLS 연결을 통해 스트림 관리자 구성 요소와 상호 작용할 수 있습니다. 이 인수는 리눅스용 Docker에서만 작동합니다.

  > **Warning**
  >
  > 이 옵션을 사용하면 컨테이너가 호스트의 모든 로컬 네트워크 인터페이스에 액세스할 수 있으므로 이 옵션은 호스트 네트워크에 대한 액세스 권한 없이 Docker 컨테이너를 실행하는 경우보다 덜 안전합니다. 이 옵션을 사용하는 Docker 컨테이너 구성 요소를 개발하고 실행할 때는 이 점을 고려하세요. 자세한 내용은 Docker 설명서의 네트워크: 호스트를 참조하십시오.

- 인증을 요구하도록 스트림 관리자 구성 요소를 구성하는 경우 (기본 동작), `AWS_CONTAINER_CREDENTIALS_FULL_URI` 환경 변수를 Greengrass nucleus가 구성 요소에 제공하는 값으로 설정합니다. 자세한 내용은 스트림 관리자 구성을 참조하십시오.

- 기본이 아닌 포트를 사용하도록 스트림 관리자 구성 요소를 구성하는 경우 IPC (프로세스 간 통신)를 사용하여 스트림 관리자 구성 요소 구성에서 포트를 가져오십시오. IPC를 사용하려면 추가 옵션을 사용하여 Docker 컨테이너를 실행해야 합니다. 자세한 내용은 다음 자료를 참조하세요.
  - 애플리케이션 코드에서 스트림 관리자에 Connect
  - Docker 컨테이너 구성 요소에서 프로세스 간 통신을 사용하십시오.

Example 예제 레시피: Docker 컨테이너 구성 요소 (Python) 의 S3 버킷으로 파일 스트리밍

다음 레시피는 파일을 생성하여 S3 버킷으로 스트리밍하는 예제 Docker 컨테이너 구성 요소를 정의합니다. 이 레시피의 속성은 다음과 같습니다.

- 스트림 관리자 구성 요소는 종속성입니다. 이러한 종속성을 통해 구성 요소는 스트림 관리자 SDK를 사용하여 스트림 관리자 구성 요소와 상호 작용할 수 있습니다.
- Docker 이미지를 Amazon S3의 TAR 아카이브로 지정하는 구성 요소 아티팩트입니다.
- TAR 아카이브에서 Docker 이미지를 로드하는 수명 주기 설치 스크립트.
- 이미지에서 Docker 컨테이너를 실행하는 라이프사이클 실행 스크립트. Docker 실행 명령에는 다음과 같은 인수가 포함됩니다.
  - `--network=host`인수는 호스트 네트워크에 대한 컨테이너 액세스를 제공하므로 컨테이너를 스트림 관리자 구성 요소에 연결할 수 있습니다.
  - 첫 번째 `-e` 인수는 Docker 컨테이너에 필요한 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 환경 변수를 설정합니다.
  - 추가 `-e` 인수는 이 예제에서 사용하는 환경 변수를 설정합니다.
  - `-v`인수는 구성 요소의 작업 폴더를 컨테이너에 마운트합니다. 이 예제에서는 스트림 관리자를 사용하여 작업 폴더에 파일을 생성하여 Amazon S3에 해당 파일을 업로드합니다.
  - `--rm`인수는 컨테이너가 종료될 때 컨테이너를 정리합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.StreamFileToS3",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Creates a text file and uses stream manager to stream the
 file to S3.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
```

```
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "bucketName": ""
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
 -e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
 {work:path}:{work:path} --rm stream-file-to-s3"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
```

```
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e BUCKET_NAME="{configuration:/bucketName}" \
          -e WORK_PATH="{work:path}" \
          -v {work:path}:{work:path} \
          --rm stream-file-to-s3
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass컴포넌트 레시피 참조

구성 요소 레시피는 구성 요소의 세부 정보, 종속성, 아티팩트 및 수명 주기를 정의하는 파일입니다. 구성 요소 수명 주기는 예를 들어 구성 요소를 설치, 실행 및 종료하기 위해 실행할 명령을 지정합니다. AWS IoT Greengrass코어는 레시피에 정의된 수명 주기를 사용하여 구성 요소를 설치하고 실행합니다. AWS IoT Greengrass서비스는 레시피를 사용하여 구성 요소를 배포할 때 코어 장치에 배포할 종속성 및 아티팩트를 식별합니다.

레시피에서 구성 요소가 지원하는 각 플랫폼에 대한 고유한 종속성 및 수명 주기를 정의할 수 있습니다. 이 기능을 사용하여 요구 사항이 서로 다른 여러 플랫폼이 있는 장치에 구성 요소를 배포할 수 있습니다. 이 기능을 사용하여 구성 요소를 지원하지 않는 장치에 구성 요소를 설치하는 것을 AWS IoT Greengrass 방지할 수도 있습니다.

각 레시피에는 매니페스트 목록이 포함되어 있습니다. 각 매니페스트는 플랫폼 요구 사항 집합과 플랫폼이 해당 요구 사항을 충족하는 핵심 장치에 사용할 수명 주기 및 아티팩트를 지정합니다. 핵심 기기는 기기가 충족하는 플랫폼 요구 사항이 있는 첫 번째 매니페스트를 사용합니다. 플랫폼 요구 사항이 없는 매니페스트를 지정하여 코어 디바이스에 맞게 지정하십시오.

매니페스트에 없는 글로벌 라이프사이클을 지정할 수도 있습니다. 글로벌 라이프사이클에서는 라이프사이클의 하위 섹션을 식별하는 선택 키를 사용할 수 있습니다. 그런 다음 매니페스트 내에서 이러한

선택 키를 지정하여 매니페스트의 수명 주기 외에도 글로벌 수명 주기의 해당 섹션을 사용할 수 있습니다. 코어 기기는 매니페스트가 라이프사이클을 정의하지 않는 경우에만 매니페스트의 선택 키를 사용합니다. 매니페스트의 선택 항목을 사용하여 `all` 선택 키 없이 글로벌 수명 주기의 섹션을 매칭할 수 있습니다.

AWS IoT GreengrassCore 소프트웨어는 코어 디바이스와 일치하는 매니페스트를 선택한 후 다음을 수행하여 사용할 수명 주기 단계를 식별합니다.

- 선택한 매니페스트가 라이프사이클을 정의하면 코어 디바이스는 해당 라이프사이클을 사용합니다.
- 선택한 매니페스트가 라이프사이클을 정의하지 않는 경우 코어 디바이스는 글로벌 라이프사이클을 사용합니다. 코어 기기는 다음을 수행하여 글로벌 라이프사이클에서 사용할 섹션을 식별합니다.
  - 매니페스트가 선택 키를 정의하는 경우 코어 기기는 매니페스트의 선택 키가 포함된 글로벌 수명 주기 섹션을 사용합니다.
  - 매니페스트가 선택 키를 정의하지 않는 경우 코어 기기는 글로벌 라이프사이클에서 선택 키가 없는 섹션을 사용합니다. 이 동작은 선택을 정의하는 매니페스트와 동일합니다. `all`

> **Important**
>
> 구성 요소를 설치하려면 코어 기기가 매니페스트의 플랫폼 요구 사항을 하나 이상 충족해야 합니다. 코어 기기와 일치하는 매니페스트가 없으면 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 설치하지 않아 배포가 실패합니다.

JSON 또는 YAML 형식으로 레시피를 정의할 수 있습니다. 레시피 예제 섹션에는 각 형식의 레시피가 포함되어 있습니다.

주제

레시피 검증

Greengrass는 구성 요소 버전을 생성할 때 JSON 또는 YAML 구성 요소 레시피를 검증합니다. 이 레시피 검증은 JSON 또는 YAML 구성 요소 레시피에 일반적인 오류가 있는지 검사하여 잠재적인 배포 문

제를 방지합니다. 검증은 레시피에 일반적인 오류 (예: 쉼표, 중괄호, 필드 누락) 가 있는지 확인하고 레시피가 제대로 구성되어 있는지 확인합니다.

레시피 검증 오류 메시지가 표시되면 레시피에 누락된 쉼표, 중괄호 또는 필드가 있는지 확인하세요. 레시피 형식을 보고 누락된 필드가 없는지 확인하세요.

레시피 형식

컴포넌트의 레시피를 정의할 때 레시피 문서에 다음 정보를 지정합니다. YAML 및 JSON 형식의 레시피에도 동일한 구조가 적용됩니다.

`RecipeFormatVersion`

레시피의 템플릿 버전. 다음 옵션을 선택합니다.

- `2020-01-25`

`ComponentName`

이 레시피가 정의하는 구성 요소의 이름. 구성 요소 이름은 각 지역마다 고유해야 합니다. AWS 계정

> **팁**
>
> - 역방향 도메인 이름 형식을 사용하여 회사 내에서 이름이 충돌하지 않도록 하세요. 예를 들어, 회사에서 태양 에너지 프로젝트를 소유하고 `example.com` 있고 진행 중인 경우 Hello World 구성 요소의 이름을 지정할 수 있습니다. `com.example.solar.HelloWorld` 이렇게 하면 회사 내에서 구성 요소 이름이 충돌하는 것을 방지할 수 있습니다.
> - 구성 요소 이름에 `aws.greengrass` 접두사를 사용하지 마십시오. AWS IoT Greengrass제공하는 공용 구성 요소에 이 접두사를 사용합니다. 공용 구성 요소와 같은 이름을 선택하면 해당 구성 요소가 해당 구성 요소를 대체합니다. 그러면 공용 구성 요소에 종속된 구성 요소를 배포할 때 공용 구성 요소 대신 구성 요소를 AWS IoT Greengrass 제공합니다. 이 기능을 사용하면 공용 구성 요소의 동작을 재정의할 수 있지만 공용 구성 요소를 재정의하지 않으려는 경우 다른 구성 요소가 손상될 수도 있습니다.

`ComponentVersion`

구성 요소의 버전입니다. 메이저, 마이너 및 패치 값의 최대값은 999999입니다.

> **Note**
>
> AWS IoT Greengrass구성 요소에 시맨틱 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 버전은 구성 `1.0.0` 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

`ComponentDescription`

(선택 사항) 구성 요소의 설명.

`ComponentPublisher`

구성 요소의 게시자 또는 작성자.

`ComponentConfiguration`

(선택 사항) 구성 요소의 구성 또는 매개 변수를 정의하는 개체입니다. 기본 구성을 정의한 다음 구성 요소를 배포할 때 구성 요소에 제공할 구성 개체를 지정할 수 있습니다. 구성 요소 구성은 중첩된 매개 변수와 구조를 지원합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`DefaultConfiguration`

구성 요소의 기본 구성을 정의하는 개체입니다. 이 개체의 구조를 정의합니다.

> **Note**
>
> AWS IoT Greengrass구성 값에 JSON을 사용합니다. JSON은 숫자 유형을 지정하지만 정수와 부동 소수점을 구분하지는 않습니다. 따라서 구성 값이 부동 소수점으로 변환될 수 있습니다. AWS IoT Greengrass 구성 요소가 올바른 데이터 유형을 사용하도록 하려면 숫자 구성 값을 문자열로 정의하는 것이 좋습니다. 그런 다음 구성 요소가 해당 값을 정수 또는 부동 소수점으로 파싱하도록 하십시오. 이렇게 하면 구성 및 코어 장치에서 구성 값의 유형이 동일할 수 있습니다.

`ComponentDependencies`

(선택 사항) 구성 요소의 구성 요소 종속성을 각각 정의하는 개체 사전입니다. 각 개체의 키는 구성 요소 종속성의 이름을 식별합니다. AWS IoT Greengrass구성 요소가 설치될 때 구성 요소 종속성을 설치합니다. AWS IoT Greengrass종속성이 시작될 때까지 기다린 후 구성 요소를 시작합니다. 각 개체에는 다음 정보가 들어 있습니다.

`VersionRequirement`

이 종속성에 대해 호환되는 구성 요소 버전을 정의하는 npm 스타일 시맨틱 버전 제약 조건입니다. 버전 또는 버전 범위를 지정할 수 있습니다. 자세한 내용은 npm 시맨틱 버전 계산기를 참조하십시오.

`DependencyType`

(선택 사항) 이 종속성의 유형. 다음 옵션 중 하나를 선택합니다.

- `SOFT` – 종속성 상태가 변경되면 구성 요소가 다시 시작되지 않습니다.
- `HARD` – 종속성 상태가 변경되면 구성 요소가 다시 시작됩니다.

기본값은 `HARD`입니다.

`ComponentType`

(선택 사항) 구성 요소 유형.

> **Note**
>
> 레시피에 구성 요소 유형을 지정하지 않는 것이 좋습니다. AWS IoT Greengrass구성 요소를 만들 때 자동으로 유형을 설정합니다.

유형은 다음 유형 중 하나일 수 있습니다.

- `aws.greengrass.generic`— 구성 요소는 명령을 실행하거나 아티팩트를 제공합니다.
- `aws.greengrass.lambda`— 구성 요소는 Lambda 시작 관리자 구성 요소를 사용하여 Lambda 함수를 실행합니다. `ComponentSource`파라미터는 이 구성 요소가 실행하는 Lambda 함수의 ARN을 지정합니다.

  이 옵션은 Lambda 함수에서 구성 요소를 생성할 AWS IoT Greengrass 때 설정되므로 사용하지 않는 것이 좋습니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.
- `aws.greengrass.plugin`— 구성 요소는 Greengrass 핵과 동일한 자바 가상 머신 (JVM) 에서 실행됩니다. 플러그인 구성 요소를 배포하거나 다시 시작하면 Greengrass 핵이 다시 시작됩니다.

  플러그인 구성 요소는 Greengrass 핵과 동일한 로그 파일을 사용합니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

이 옵션은 Greengrass 핵과 직접 인터페이스하는 Java로 작성된 AWS -provide 구성 요소용이므로 구성 요소 레시피에는 사용하지 않는 것이 좋습니다. 어떤 공용 구성 요소가 플러그인인인지에 대한 자세한 내용은 을 참조하십시오. AWS-제공된 구성 요소

- `aws.greengrass.nucleus`— 핵 구성 요소. 자세한 설명은 그린그래스 핵 섹션을 참조하세요.

  구성 요소 레시피에는 이 옵션을 사용하지 않는 것이 좋습니다. Core 소프트웨어의 최소 기능을 제공하는 Greengrass nucleus 구성 요소를 위한 것입니다. AWS IoT Greengrass

레시피에서 구성 요소를 생성하거나 Lambda 함수에서 구성 요소를 생성할 `aws.greengrass.lambda` 때 기본값이 설정됩니다. `aws.greengrass.generic`

자세한 설명은 구성 요소 유형 섹션을 참조하세요.

`ComponentSource`

(선택 사항) 구성 요소가 실행하는 Lambda 함수의 ARN.

레시피에 구성 요소 소스를 지정하지 않는 것이 좋습니다. AWS IoT GreengrassLambda 함수에서 구성 요소를 생성할 때 이 파라미터를 자동으로 설정합니다. 자세한 설명은 AWS Lambda함수 실행 섹션을 참조하세요.

`Manifests`

구성 요소의 수명 주기, 파라미터, 플랫폼 요구 사항을 각각 정의하는 객체 목록입니다. 코어 기기가 여러 매니페스트의 플랫폼 요구 사항을 충족하는 경우 코어 기기와 일치하는 첫 번째 매니페스트를 AWS IoT Greengrass 사용합니다. 코어 기기가 올바른 매니페스트를 사용하도록 하려면 먼저 플랫폼 요구 사항이 더 엄격한 매니페스트를 정의해야 합니다. 모든 플랫폼에 적용되는 매니페스트는 목록의 마지막 매니페스트여야 합니다.

> **Important**
>
> 구성 요소를 설치하려면 코어 기기가 매니페스트의 플랫폼 요구 사항을 하나 이상 충족해야 합니다. 코어 기기와 일치하는 매니페스트가 없으면 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 설치하지 않아 배포가 실패합니다.

각 개체에는 다음 정보가 포함됩니다.

`Name`

(선택 사항) 이 매니페스트가 정의하는 플랫폼의 친숙한 이름.

이 매개 변수를 생략하면 플랫폼에서 os 이름이 AWS IoT Greengrass 생성되고 architecture

Platform

(선택 사항) 이 매니페스트가 적용되는 플랫폼을 정의하는 객체입니다. 모든 플랫폼에 적용되는 매니페스트를 정의하려면 이 매개변수를 생략하십시오.

이 객체는 코어 기기가 실행되는 플랫폼에 대한 키-값 쌍을 지정합니다. 이 구성 요소를 배포하면 AWS IoT Greengrass Core 소프트웨어는 이러한 키-값 쌍을 코어 장치의 플랫폼 속성과 비교합니다. AWS IoT GreengrassCore 소프트웨어는 항상 os 및 를 architecture 정의하며 추가 속성을 정의할 수도 있습니다. Greengrass nucleus 구성 요소를 배포할 때 코어 디바이스의 사용자 지정 플랫폼 속성을 지정할 수 있습니다. 자세한 내용은 Greengrass nucleus 구성요소의 플랫폼 오버라이드 매개변수를 참조하십시오.

각 키-값 쌍에 대해 다음 값 중 하나를 지정할 수 있습니다.

- 정확한 값 (예linux: 또는) windows 정확한 값은 문자나 숫자로 시작해야 합니다.
- *모든 값과 일치합니다. 값이 없는 경우에도 일치합니다.
- Java 스타일 정규 표현식 (예:. /windows|linux/ 정규 표현식은 슬래시 문자 () 로 시작하고 끝나야 합니다. / 예를 들어, 정규 표현식은 공백이 아닌 모든 값과 /.+/ 일치합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

os

(선택 사항) 이 매니페스트가 지원하는 플랫폼의 운영 체제 이름. 일반 플랫폼에는 다음 값이 포함됩니다.

- linux
- windows
- darwin(macOS)

architecture

(선택 사항) 이 매니페스트가 지원하는 플랫폼의 프로세서 아키텍처. 일반적인 아키텍처에는 다음 값이 포함됩니다.

- amd64
- arm
- aarch64
- x86

`architecture.detail`

(선택 사항) 이 매니페스트가 지원하는 플랫폼의 프로세서 아키텍처 세부 정보입니다. 일반적인 아키텍처 세부 정보에는 다음 값이 포함됩니다.

- `arm61`
- `arm71`
- `arm81`

`key`

(선택 사항) 이 매니페스트에 대해 정의하는 플랫폼 속성. *`##`* 플랫폼 속성의 이름으로 바꾸십시오. AWS IoT GreengrassCore 소프트웨어는 이 플랫폼 속성을 Greengrass nucleus 구성 요소 구성에서 지정한 키-값 쌍과 일치시킵니다. 자세한 내용은 Greengrass nucleus 구성요소의 플랫폼 오버라이드 매개변수를 참조하십시오.

> **Tip**
>
> 역방향 도메인 이름 형식을 사용하여 회사 내에서 이름이 충돌하지 않도록 하세요. 예를 들어, 회사에서 라디오 프로젝트를 소유하고 `example.com` 작업하는 경우 사용자 지정 플랫폼 속성에 이름을 지정할 수 있습니다. `com.example.radio.RadioModule` 이렇게 하면 회사 내에서 플랫폼 속성 이름이 충돌하는 것을 방지할 수 있습니다.

예를 들어 플랫폼 속성을 정의하여 코어 기기에서 사용할 수 있는 라디오 모듈을 기반으로 다른 매니페스트를 지정할 수 있습니다. `com.example.radio.RadioModule` 각 매니페스트에는 다양한 하드웨어 구성에 적용되는 서로 다른 아티팩트가 포함될 수 있으므로 최소한의 소프트웨어 세트만 코어 기기에 배포할 수 있습니다.

`Lifecycle`

이 매니페스트가 정의하는 플랫폼에서 구성 요소를 설치하고 실행하는 방법을 정의하는 개체 또는 문자열입니다. 모든 플랫폼에 적용되는 글로벌 라이프사이클을 정의할 수도 있습니다. 코어 기기는 사용할 매니페스트에 수명 주기가 지정되지 않은 경우에만 글로벌 수명 주기를 사용합니다.

> **Note**
>
> 이 수명 주기는 매니페스트 내에서 정의합니다. 여기서 지정하는 라이프사이클 단계는 이 매니페스트가 정의하는 플랫폼에만 적용됩니다. 모든 플랫폼에 적용되는 글로벌 라이프사이클을 정의할 수도 있습니다.

이 개체 또는 문자열에는 다음 정보가 포함됩니다.

`Setenv`

(선택 사항) 모든 라이프사이클 스크립트에 제공하는 환경 변수 사전입니다. 각 라이프 사이클 스크립트에서 이러한 환경 변수를 재정의할 수 있습니다. `Setenv`

`install`

(선택 사항) 구성 요소가 설치될 때 실행할 스크립트를 정의하는 개체 또는 문자열입니다. 또한 AWS IoT Greengrass Core 소프트웨어는 소프트웨어를 시작할 때마다 이 수명 주기 단계를 실행합니다.

`install`스크립트가 성공 코드와 함께 종료되면 구성 요소가 `INSTALLED` 상태로 전환됩니다.

이 개체 또는 문자열에는 다음 정보가 들어 있습니다.

`Script`

실행할 스크립트.

`RequiresPrivilege`

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 로 `true` 설정하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소를 실행하도록 구성한 시스템 사용자 대신 루트로 이 수명 주기 스크립트를 실행합니다. 기본값은 `false`입니다.

`Skipif`

(선택 사항) 스크립트 실행 여부를 결정하기 위한 검사입니다. 실행 파일이 경로에 있는지 또는 파일이 존재하는지 확인하도록 정의할 수 있습니다. 출력이 true인 경우 AWS IoT Greengrass Core 소프트웨어는 이 단계를 건너뛰습니다. 다음 검사 중 하나를 선택합니다.

- `onpath` *`runnable`*— 실행 가능한 항목이 시스템 경로에 있는지 확인합니다. 예를 들어, Python 3을 사용할 수 있는 경우 이 라이프사이클 단계를 **`onpath python3`** 건너뛰려면 사용하십시오.

- `exists` *`file`*— 파일이 존재하는지 확인하세요. 예를 들어, 있는 경우 이 수명 주기 단계를 `/tmp/my-configuration.db` 건너뛰는 **`exists /tmp/my-configuration.db`** 데 사용합니다.

`Timeout`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 프로세스를 종료하기 전에 스크립트를 실행할 수 있는 최대 시간 (초) 입니다.

기본값: 120초

`Setenv`

(선택 사항) 스크립트에 제공할 환경 변수 사전입니다. 이러한 환경 변수는 에서 `Lifecycle.Setenv` 제공하는 변수를 재정의합니다.

`run`

(선택 사항) 구성 요소가 시작될 때 실행할 스크립트를 정의하는 개체 또는 문자열입니다.

구성 요소는 이 수명 주기 단계가 실행될 때 `RUNNING` 상태에 들어갑니다. `run`스크립트가 성공 코드와 함께 종료되면 구성 요소가 `STOPPING` 상태로 전환됩니다. `shutdown`스크립트가 지정되면 실행되고 그렇지 않으면 구성 요소가 `FINISHED` 상태로 전환됩니다.

이 구성 요소에 종속된 구성 요소는 이 수명 주기 단계가 실행될 때 시작됩니다. 종속 구성 요소가 사용하는 서비스와 같은 백그라운드 프로세스를 실행하려면 `startup` 수명 주기 단계를 대신 사용하세요.

수명 주기가 있는 구성 요소를 배포하면 이 `run` 수명 주기 스크립트가 실행되는 즉시 코어 기기가 배포가 완료되었다고 보고할 수 있습니다. 따라서 `run` 수명 주기 스크립트가 실행 직후 실패하더라도 배포를 완료하고 성공적으로 배포할 수 있습니다. 구성 요소의 시작 스크립트 결과에 따라 배포 상태가 달라지도록 하려면 `startup` 수명 주기 단계를 대신 사용하십시오.

> **Note**
>
> `run`수명 주기 중 하나만 `startup` 정의할 수 있습니다.

이 개체 또는 문자열에는 다음 정보가 들어 있습니다.

`Script`

실행할 스크립트.

`RequiresPrivilege`

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 로 `true` 설정하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소를 실행하도록 구성한 시스템 사용자 대신 루트로 이 수명 주기 스크립트를 실행합니다. 기본값은 `false`입니다.

`Skipif`

(선택 사항) 스크립트 실행 여부를 결정하기 위한 검사입니다. 실행 파일이 경로에 있는지 또는 파일이 존재하는지 확인하도록 정의할 수 있습니다. 출력이 true인 경우 AWS IoT Greengrass Core 소프트웨어는 이 단계를 건너뛰습니다. 다음 검사 중 하나를 선택합니다.

- `onpath` *`runnable`*— 실행 가능한 항목이 시스템 경로에 있는지 확인합니다. 예를 들어, Python 3을 사용할 수 있는 경우 이 라이프사이클 단계를 **`onpath python3`** 건너뛰려면 사용하십시오.
- `exists` *`file`*— 파일이 존재하는지 확인하세요. 예를 들어, 있는 경우 이 수명 주기 단계를 `/tmp/my-configuration.db` 건너뛰는 **`exists /tmp/my-configuration.db`** 데 사용합니다.

`Timeout`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 프로세스를 종료하기 전에 스크립트를 실행할 수 있는 최대 시간 (초) 입니다.

이 수명 주기 단계는 기본적으로 제한 시간이 초과되지 않습니다. 이 제한 시간을 생략하면 `run` 스크립트는 종료될 때까지 실행됩니다.

`Setenv`

(선택 사항) 스크립트에 제공할 환경 변수 사전입니다. 이러한 환경 변수는 에서 `Lifecycle.Setenv` 제공하는 변수를 재정의합니다.

`startup`

(선택 사항) 구성 요소가 시작될 때 실행할 백그라운드 프로세스를 정의하는 개체 또는 문자열입니다.

성공적으로 종료되어야 `startup` 하는 명령을 실행하거나 구성 요소의 상태를 종속 구성 요소가 `RUNNING` 시작되기 전으로 업데이트하는 데 사용합니다. UpdateStateIPC 작업을 사용하여 구성 요소의 상태를 다음으로 `RUNNING` 설정하거나 구성 요소가 종료되지 않는 스크립트를 시작할 `ERRORED` 때 설정할 수 있습니다. 예를 들어 MySQL 프로세스를 시작하는 `startup` 단계를 정의할 수 있습니다. `/etc/init.d/mysqld start`

구성 요소는 이 수명 주기 단계가 실행될 때 STARTING 상태에 들어갑니다. startup스크립트가 성공 코드와 함께 종료되면 구성 요소가 RUNNING 상태로 전환됩니다. 그러면 종속 구성 요소가 시작될 수 있습니다.

수명 주기가 있는 구성 요소를 배포하면 이 startup 수명 주기 스크립트가 종료되거나 상태를 보고한 후 코어 디바이스에서 배포가 완료된 것으로 보고할 수 있습니다. 즉, 모든 구성 요소의 시작 스크립트가 종료되거나 상태를 보고할 IN_PROGRESS 때까지 배포 상태가 유지됩니다.

> **Note**
>
> 수명 주기 중 startup 하나만 정의할 수 있습니다. run

이 개체 또는 문자열에는 다음 정보가 들어 있습니다.

`Script`

실행할 스크립트.

`RequiresPrivilege`

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 로 true 설정하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소를 실행하도록 구성한 시스템 사용자 대신 루트로 이 수명 주기 스크립트를 실행합니다. 기본값은 false입니다.

`Skipif`

(선택 사항) 스크립트 실행 여부를 결정하기 위한 검사입니다. 실행 파일이 경로에 있는지 또는 파일이 존재하는지 확인하도록 정의할 수 있습니다. 출력이 true인 경우 AWS IoT Greengrass Core 소프트웨어는 이 단계를 건너뛰습니다. 다음 검사 중 하나를 선택합니다.

- `onpath` *`runnable`*— 실행 가능한 항목이 시스템 경로에 있는지 확인합니다. 예를 들어, Python 3을 사용할 수 있는 경우 이 라이프사이클 단계를 **`onpath python3`** 건너뛰려면 사용하십시오.
- `exists` *`file`*— 파일이 존재하는지 확인하세요. 예를 들어, 있는 경우 이 수명 주기 단계를 /tmp/my-configuration.db 건너뛰는 **`exists /tmp/my-configuration.db`** 데 사용합니다.

`Timeout`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 프로세스를 종료하기 전에 스크립트를 실행할 수 있는 최대 시간 (초) 입니다.

기본값: 120초

`Setenv`

(선택 사항) 스크립트에 제공할 환경 변수 사전입니다. 이러한 환경 변수는 에서 `Lifecycle.Setenv` 제공하는 변수를 재정의합니다.

`shutdown`

(선택 사항) 구성 요소가 종료될 때 실행할 스크립트를 정의하는 개체 또는 문자열입니다. 종료 수명 주기를 사용하여 구성 요소가 상태에 있을 때 실행하려는 코드를 실행할 수 있습니다. `STOPPING` 종료 수명 주기는 `startup` 또는 `run` 스크립트로 시작된 프로세스를 중지하는 데 사용할 수 있습니다.

에서 `startup` 백그라운드 프로세스를 시작하는 경우 구성 요소가 종료될 때 이 `shutdown` 단계를 사용하여 해당 프로세스를 중지하십시오. 예를 들어, 를 사용하여 MySQL 프로세스를 중지하는 `shutdown` 단계를 정의할 수 있습니다. `/etc/init.d/mysqld stop`

구성 요소가 `STOPPING` 상태에 들어간 후에 `shutdown` 스크립트가 실행됩니다. 스크립트가 성공적으로 완료되면 구성 요소가 `FINISHED` 상태로 전환됩니다.

이 개체 또는 문자열에는 다음 정보가 들어 있습니다.

`Script`

실행할 스크립트.

`RequiresPrivilege`

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 로 `true` 설정하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소를 실행하도록 구성한 시스템 사용자 대신 루트로 이 수명 주기 스크립트를 실행합니다. 기본값은 `false`입니다.

`Skipif`

(선택 사항) 스크립트 실행 여부를 결정하기 위한 검사입니다. 실행 파일이 경로에 있는지 또는 파일이 존재하는지 확인하도록 정의할 수 있습니다. 출력이 true인 경우 AWS IoT Greengrass Core 소프트웨어는 이 단계를 건너뛰습니다. 다음 검사 중 하나를 선택합니다.

- `onpath` *`runnable`*— 실행 가능한 항목이 시스템 경로에 있는지 확인합니다. 예를 들어, Python 3을 사용할 수 있는 경우 이 라이프사이클 단계를 **`onpath python3`** 건너뛰려면 사용하십시오.
- `exists` *`file`*— 파일이 존재하는지 확인하세요. 예를 들어, 있는 경우 이 수명 주기 단계를 `/tmp/my-configuration.db` 건너뛰는 **`exists /tmp/my-configuration.db`** 데 사용합니다.

`Timeout`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 프로세스를 종료하기 전에 스크립트를 실행할 수 있는 최대 시간 (초) 입니다.

기본값: 15초.

`Setenv`

(선택 사항) 스크립트에 제공할 환경 변수 사전입니다. 이러한 환경 변수는 에서 `Lifecycle.Setenv` 제공하는 변수를 재정의합니다.

`recover`

(선택 사항) 구성 요소에 오류가 발생할 때 실행할 스크립트를 정의하는 개체 또는 문자열입니다.

이 단계는 구성 요소가 ERRORED 상태에 들어갈 때 실행됩니다. 구성 요소가 성공적으로 복구되지 않은 상태가 ERRORED 세 번 반복되면 구성 요소가 해당 BROKEN 상태로 변경됩니다. 구성 요소를 수정하려면 BROKEN 구성 요소를 다시 배포해야 합니다.

이 개체 또는 문자열에는 다음 정보가 들어 있습니다.

`Script`

실행할 스크립트.

`RequiresPrivilege`

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 로 `true` 설정하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소를 실행하도록 구성한 시스템 사용자 대신 루트로 이 수명 주기 스크립트를 실행합니다. 기본값은 `false`입니다.

`Skipif`

(선택 사항) 스크립트 실행 여부를 결정하기 위한 검사입니다. 실행 파일이 경로에 있는지 또는 파일이 존재하는지 확인하도록 정의할 수 있습니다. 출력이 true인 경우 AWS IoT

Greengrass Core 소프트웨어는 이 단계를 건너뛰습니다. 다음 검사 중 하나를 선택합니다.

- `onpath` *`runnable`*— 실행 가능한 항목이 시스템 경로에 있는지 확인합니다. 예를 들어, Python 3을 사용할 수 있는 경우 이 라이프사이클 단계를 **`onpath python3`** 건너뛰려면 사용하십시오.
- `exists` *`file`*— 파일이 존재하는지 확인하세요. 예를 들어, 있는 경우 이 수명 주기 단계를 `/tmp/my-configuration.db` 건너뛰는 **`exists /tmp/my-configuration.db`** 데 사용합니다.

`Timeout`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 프로세스를 종료하기 전에 스크립트를 실행할 수 있는 최대 시간 (초) 입니다.

기본값: 60초

`Setenv`

(선택 사항) 스크립트에 제공할 환경 변수 사전입니다. 이러한 환경 변수는 에서 `Lifecycle.Setenv` 제공하는 변수를 재정의합니다.

`bootstrap`

(선택 사항) AWS IoT Greengrass Core 소프트웨어 또는 코어 장치를 다시 시작해야 하는 스크립트를 정의하는 개체 또는 문자열입니다. 이를 통해 운영 체제 업데이트 또는 런타임 업데이트 등을 설치한 후 다시 시작하는 구성 요소를 개발할 수 있습니다.

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어 또는 장치를 다시 시작할 필요가 없는 업데이트 또는 종속 항목을 설치하려면 설치 수명 주기를 사용하십시오.

이 수명 주기 단계는 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 배포하는 다음과 같은 경우 설치 수명 주기 단계보다 먼저 실행됩니다.

- 구성 요소가 처음으로 코어 장치에 배포됩니다.
- 구성 요소 버전이 변경됩니다.
- 구성 요소 구성 업데이트의 결과로 부트스트랩 스크립트가 변경됩니다.

AWS IoT GreengrassCore 소프트웨어가 배포에 부트스트랩 단계가 있는 모든 구성 요소의 부트스트랩 단계를 완료하면 소프트웨어가 다시 시작됩니다.

> **Important**
>
> AWS IoT GreengrassCore 소프트웨어 또는 코어 디바이스를 다시 시작하려면 AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 구성해야 합니다. AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 구성하지 않으면 소프트웨어가 다시 시작되지 않습니다. 자세한 설명은 Greengrass 핵을 시스템 서비스로 구성 섹션을 참조하세요.

이 개체 또는 문자열에는 다음 정보가 들어 있습니다.

`BootstrapOnRollback`

> **Note**
>
> 이 기능을 `BootstrapOnRollback` 활성화하면 실패한 대상 배포의 일부로 부트스트랩 수명 주기 단계를 완료했거나 실행을 시도한 구성 요소에 대해서만 실행됩니다. 이 기능은 그린그래스 핵 버전 2.12.0 이상에서 사용할 수 있습니다.

(선택 사항) 롤백 배포의 일부로 부트스트랩 수명 주기 단계를 실행할 수 있습니다. 이 옵션을 로 `true` 설정하면 롤백 배포 내에 정의된 부트스트랩 수명 주기 단계가 실행됩니다. 배포에 실패하면 롤백 배포 중에 이전 버전의 구성 요소 부트스트랩 수명 주기가 다시 실행됩니다.

기본값은 `false`입니다.

`Script`

실행할 스크립트. 이 스크립트의 종료 코드는 재시작 지침을 정의합니다. 다음 종료 코드를 사용하십시오.

- `0`— AWS IoT Greengrass Core 소프트웨어 또는 코어 디바이스를 다시 시작하지 마십시오. 모든 구성 요소가 부트스트랩된 후에도 AWS IoT Greengrass Core 소프트웨어가 다시 시작됩니다.
- `100`— AWS IoT Greengrass Core 소프트웨어 재시작을 요청합니다.
- `101`— 코어 디바이스 재시작 요청.

종료 코드 100~199는 특수 동작용입니다. 다른 종료 코드는 스크립트 오류를 나타냅니다.

`RequiresPrivilege`

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 로 `true` 설정하면 AWS IoT Greengrass Core 소프트웨어는 이 구성 요소를 실행하도록 구성한 시스템 사용자 대신 루트로 이 수명 주기 스크립트를 실행합니다. 기본값은 `false`입니다.

`Timeout`

(선택 사항) AWS IoT Greengrass Core 소프트웨어가 프로세스를 종료하기 전에 스크립트를 실행할 수 있는 최대 시간 (초) 입니다.

기본값: 120초

`Setenv`

(선택 사항) 스크립트에 제공할 환경 변수 사전입니다. 이러한 환경 변수는 에서 `Lifecycle.Setenv` 제공하는 변수를 재정의합니다.

`Selections`

(선택 사항) 이 매니페스트에 실행할 글로벌 수명 주기 섹션을 지정하는 선택 키 목록입니다. 글로벌 수명 주기에서는 모든 수준의 선택 키를 사용하여 수명 주기 단계를 정의하여 주기의 하위 섹션을 선택할 수 있습니다. 그러면 코어 디바이스는 이 매니페스트의 선택 키와 일치하는 섹션을 사용합니다. 자세한 내용은 글로벌 라이프사이클 예시를 참조하십시오.

> **Important**
>
> 코어 기기는 이 매니페스트가 라이프사이클을 정의하지 않는 경우에만 글로벌 라이프사이클의 선택 항목을 사용합니다.

글로벌 라이프사이클에서 `all` 선택 키가 없는 섹션을 실행하도록 선택 키를 지정할 수 있습니다.

`Artifacts`

(선택 사항) 이 매니페스트가 정의하는 플랫폼 구성 요소의 바이너리 아티팩트를 각각 정의하는 개체의 목록입니다. 예를 들어 코드나 이미지를 아티팩트로 정의할 수 있습니다.

구성 요소가 배포되면 AWS IoT Greengrass Core 소프트웨어가 아티팩트를 코어 장치의 폴더에 다운로드합니다. 아티팩트를 다운로드한 후 소프트웨어가 추출하는 아카이브 파일로 정의할 수도 있습니다.

레시피 변수를 사용하여 코어 디바이스에 아티팩트가 설치된 폴더의 경로를 가져올 수 있습니다.

- 일반 파일 - artifacts:path 레시피 변수를 사용하여 아티팩트가 포함된 폴더의 경로를 가져올 수 있습니다. 예를 들어, `{artifacts:path}/my_script.py` URI가 있는 아티팩트의 경로를 가져오려면 레시피에 지정하십시오. `s3://DOC-EXAMPLE-BUCKET/path/to/my_script.py`
- 추출된 아카이브 — Artifacts:decompressedPath 레시피 변수를 사용하여 추출된 아카이브 아티팩트가 들어 있는 폴더의 경로를 가져올 수 있습니다. AWS IoT GreengrassCore 소프트웨어는 각 아카이브를 아카이브와 이름이 같은 폴더에 추출합니다. 예를 들어 `s3://DOC-EXAMPLE-BUCKET/path/to/my_archive.zip` URI가 `{artifacts:decompressedPath}/my_archive/my_script.py` 있는 아카이브 `my_script.py` 아티팩트의 경로를 가져올 레시피에 지정합니다.

> **Note**
>
> 로컬 코어 기기에서 아카이브 아티팩트가 포함된 구성 요소를 개발하는 경우 해당 아티팩트에 대한 URI가 없을 수 있습니다. 아티팩트를 추출하는 Unarchive 옵션으로 구성 요소를 테스트하려면 파일 이름이 아카이브 아티팩트 파일의 이름과 일치하는 URI를 지정하십시오. 아카이브 아티팩트를 업로드할 것으로 예상되는 URI를 지정하거나 새 자리 표시자 URI를 지정할 수 있습니다. 예를 들어, 로컬 배포 중에 `my_archive.zip` 아티팩트를 추출하도록 지정할 수 있습니다. `s3://DOC-EXAMPLE-BUCKET/my_archive.zip`

각 객체에는 다음과 같은 정보가 들어 있습니다.

`URI`

S3 버킷에 있는 아티팩트의 URI. AWS IoT GreengrassCore 소프트웨어는 구성 요소가 설치될 때 이 URI에서 아티팩트를 가져옵니다. 단, 아티팩트가 디바이스에 이미 있는 경우는 예외입니다. 각 아티팩트는 각 매니페스트 내에서 고유한 파일 이름을 가져야 합니다.

`Unarchive`

(선택 사항) 압축을 풀 아카이브 유형. 다음 옵션 중 하나를 선택합니다.

- `NONE`— 파일은 압축을 풀 수 있는 아카이브가 아닙니다. AWS IoT GreengrassCore 소프트웨어는 아티팩트를 코어 디바이스의 폴더에 설치합니다. artifacts:path 레시피 변수를 사용하여 이 폴더의 경로를 가져올 수 있습니다.

- `ZIP`— 파일은 ZIP 아카이브입니다. AWS IoT GreengrassCore 소프트웨어는 아카이브를 아카이브와 이름이 같은 폴더에 추출합니다. Artifacts:decompressedPath 레시피 변수를 사용하여 이 폴더가 포함된 폴더의 경로를 가져올 수 있습니다.

기본값은 `NONE`입니다.

`Permission`

(선택 사항) 이 아티팩트 파일에 설정할 액세스 권한을 정의하는 객체입니다. 읽기 권한과 실행 권한을 설정할 수 있습니다.

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어에서는 구성 요소가 아티팩트 폴더의 아티팩트 파일을 편집하는 것을 허용하지 않으므로 쓰기 권한을 설정할 수 없습니다. 구성 요소의 아티팩트 파일을 편집하려면 해당 파일을 다른 위치에 복사하거나 새 아티팩트 파일을 게시하고 배포하십시오.

아티팩트를 압축 해제할 아카이브로 정의하면 AWS IoT Greengrass Core 소프트웨어는 아카이브에서 압축을 푼 파일에 대해 이러한 액세스 권한을 설정합니다. AWS IoT GreengrassCore 소프트웨어는 폴더의 액세스 권한을 for 및 로 `ALL` 설정합니다. `Read` `Execute` 이렇게 하면 구성 요소가 폴더에서 압축을 푼 파일을 볼 수 있습니다. 아카이브의 개별 파일에 대한 권한을 설정하려면 설치 수명 주기 스크립트에서 권한을 설정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`Read`

(선택 사항) 이 아티팩트 파일에 설정할 읽기 권한. 다른 구성 요소 (예: 이 구성 요소에 종속된 구성 요소) 가 이 아티팩트에 액세스할 수 있도록 하려면 다음을 지정합니다. `ALL` 다음 옵션 중 하나를 선택합니다.

- `NONE`— 파일을 읽을 수 없습니다.
- `OWNER`— 이 구성 요소를 실행하도록 구성한 시스템 사용자가 파일을 읽을 수 있습니다.
- `ALL`— 모든 사용자가 파일을 읽을 수 있습니다.

기본값은 `OWNER`입니다.

`Execute`

(선택 사항) 이 아티팩트 파일에 설정할 실행 권한입니다. `Execute`권한은 권한을 의미합니다. `Read` 예를 들어, 를 `ALL` 지정하면 모든 사용자가 이 아티팩트 파일을 읽고 실행할 수 있습니다. `Execute`

다음 옵션 중 하나를 선택합니다.

- `NONE`— 파일을 실행할 수 없습니다.
- `OWNER`— 구성 요소를 실행하도록 구성한 시스템 사용자가 파일을 실행할 수 있습니다.
- `ALL`— 모든 사용자가 파일을 실행할 수 있습니다.

기본값은 `NONE`입니다.

`Digest`

(읽기 전용) 아티팩트의 암호화 다이제스트 해시입니다. 구성 요소를 만들 때 AWS IoT Greengrass 는 해시 알고리즘을 사용하여 아티팩트 파일의 해시를 계산합니다. 그런 다음 구성 요소를 배포하면 Greengrass nucleus가 다운로드된 아티팩트의 해시를 계산하고 해시를 이 다이제스트와 비교하여 설치 전에 아티팩트를 확인합니다. 해시가 다이제스트와 일치하지 않으면 배포가 실패합니다.

이 매개 변수를 설정하면 AWS IoT Greengrass 구성 요소를 만들 때 설정한 값이 바뀝니다.

`Algorithm`

(읽기 전용) 아티팩트의 다이제스트 해시를 계산하는 데 AWS IoT Greengrass 사용하는 해시 알고리즘입니다.

이 매개 변수를 설정하면 구성 요소를 AWS IoT Greengrass 만들 때 설정한 값이 바뀝니다.

`Lifecycle`

구성 요소 설치 및 실행 방법을 정의하는 객체입니다. 코어 기기는 사용할 매니페스트에 수명 주기가 지정되지 않은 경우에만 글로벌 수명 주기를 사용합니다.

> **Note**
>
> 이 수명 주기는 매니페스트 외부에서 정의합니다. 해당 매니페스트와 일치하는 플랫폼에 적용되는 매니페스트 수명 주기를 정의할 수도 있습니다.

글로벌 수명 주기에서는 각 매니페스트에서 지정하는 특정 선택 키에 대해 실행되는 수명 주기를 지정할 수 있습니다. 선택 키는 각 매니페스트에 대해 실행할 글로벌 수명 주기 섹션을 식별하는 문자열입니다.

`all`선택 키는 선택 키가 없는 모든 섹션의 기본값입니다. 즉, `all` 선택 키 없이 글로벌 수명 주기의 섹션을 실행하도록 매니페스트에서 선택 키를 지정할 수 있습니다. 글로벌 수명 주기에서는 `all` 선택 키를 지정할 필요가 없습니다.

매니페스트가 라이프사이클 또는 선택 키를 정의하지 않는 경우 코어 기기는 기본적으로 선택 항목을 사용합니다. `all` 즉, 이 경우 코어 기기는 선택 키를 사용하지 않는 글로벌 수명 주기 섹션을 사용합니다.

이 개체에는 매니페스트 수명 주기와 동일한 정보가 포함되지만, 원하는 수준에서 선택 키를 지정하여 수명 주기의 하위 섹션을 선택할 수 있습니다.

> **Tip**
>
> 선택 키와 수명 주기 키 간의 충돌을 방지하려면 각 선택 키에 소문자만 사용하는 것이 좋습니다. 라이프사이클 키는 대문자로 시작합니다.

Example 최상위 선택 키를 사용한 글로벌 라이프사이클의 예

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Example 하위 수준 선택 키를 사용한 글로벌 라이프사이클 예제

```
Lifecycle:
  install:
    Script:
      key1: command1
```

```
      key2: command2
      all: command3
```

Example 여러 수준의 선택 키가 있는 글로벌 라이프사이클의 예

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script:
        key3: command3
        key4: command4
        all: command5
```

레시피 변수

레시피 변수는 레시피에 사용할 수 있도록 현재 구성 요소 및 핵의 정보를 노출합니다. 예를 들어 레시피 변수를 사용하여 라이프사이클 스크립트에서 실행하는 애플리케이션에 구성 요소 구성 매개변수를 전달할 수 있습니다.

구성 요소 레시피의 다음 섹션에서 레시피 변수를 사용할 수 있습니다.

- 라이프사이클 정의.
- 구성 요소 구성 정의 (Greengrass nucleus v2.6.0 이상을 사용하고 구성 옵션을 로 설정한 경우). interpolateComponentConfiguration`true` 구성 요소 구성 업데이트를 배포할 때 레시피 변수를 사용할 수도 있습니다.

레시피 변수는 `{recipe_variable}` 구문을 사용합니다. 중괄호는 레시피 변수를 나타냅니다.

AWS IoT Greengrass지원되는 레시피 변수는 다음과 같습니다.

component_dependency_name`:configuration:`*json_pointer*

이 레시피가 정의하는 구성 요소 또는 이 구성 요소가 종속된 구성 요소의 구성 매개 변수 값입니다.

이 변수를 사용하여 구성 요소 수명 주기에서 실행하는 스크립트에 매개 변수를 제공할 수 있습니다.

> **Note**
>
> AWS IoT Greengrass구성 요소 수명 주기 정의에서만 이 레시피 변수를 지원합니다.

이 레시피 변수에는 다음과 같은 입력이 있습니다.

- `component_dependency_name`— (선택 사항) 쿼리할 구성 요소 종속성의 이름. 이 레시피가 정의하는 구성 요소를 쿼리하려면 이 세그먼트를 생략하십시오. 직접 종속성만 지정할 수 있습니다.
- `json_pointer`— 평가할 구성 값을 가리키는 JSON 포인터. JSON 포인터는 슬래시로 시작합니다. `/` 중첩된 구성 요소 구성에서 값을 식별하려면 정방향 슬래시 (`/`) 를 사용하여 구성의 각 수준에 대해 키를 구분하십시오. 숫자를 키로 사용하여 목록의 색인을 지정할 수 있습니다. 자세한 내용은 JSON 포인터 사양을 참조하십시오.

  AWS IoT GreengrassCore는 YAML 형식의 레시피에 JSON 포인터를 사용합니다.

JSON 포인터는 다음 노드 유형을 참조할 수 있습니다.

- 밸류 노드. AWS IoT Greengrass Core는 레시피 변수를 값의 문자열 표현으로 대체합니다. Null 값은 `null` 문자열로 변환됩니다.
- 오브젝트 노드. AWS IoT Greengrass Core는 레시피 변수를 해당 객체의 직렬화된 JSON 문자열 표현으로 대체합니다.
- 노드가 없습니다. AWS IoT Greengrass Core는 레시피 변수를 대체하지 않습니다.

예를 들어, `{configuration:/Message}` 레시피 변수는 구성 요소 구성의 `Message` 키 값을 검색합니다. `{com.example.MyComponentDependency:configuration:/server/port}`레시피 변수는 구성 요소 종속성의 `server` 구성 개체에 `port` 있는 값을 검색합니다.

`component_dependency_name``:artifacts:path`

이 레시피가 정의하는 구성 요소 또는 이 구성 요소가 종속된 구성 요소에 대한 아티팩트의 루트 경로입니다.

구성 요소가 설치되면 구성 요소의 아티팩트를 이 변수가 표시하는 폴더에 AWS IoT Greengrass 복사합니다. 예를 들어 이 변수를 사용하여 구성 요소 수명 주기에서 실행할 스크립트의 위치를 식별할 수 있습니다.

이 경로의 폴더는 읽기 전용입니다. 아티팩트 파일을 수정하려면 파일을 현재 작업 디렉토리(`$PWD`또는`.`) 와 같은 다른 위치에 복사하십시오. 그런 다음 거기서 파일을 수정합니다.

구성 요소 종속성에서 아티팩트를 읽거나 실행하려면 해당 아티팩트 `Read` 또는 `Execute` 권한이 있어야 합니다. `ALL` 자세한 내용은 구성 요소 레시피에 정의한 아티팩트 권한을 참조하십시오.

이 레시피 변수에는 다음과 같은 입력이 있습니다.

- component_dependency_name— (선택 사항) 쿼리할 구성 요소 종속성의 이름. 이 레시피가 정의하는 구성 요소를 쿼리하려면 이 세그먼트를 생략하십시오. 직접 종속성만 지정할 수 있습니다.

component_dependency_name`:artifacts:decompressedPath`

이 레시피가 정의하는 구성 요소 또는 이 구성 요소가 종속된 구성 요소에 대한 압축 해제된 아카이브 아티팩트의 루트 경로입니다.

구성 요소가 설치되면 이 변수가 표시하는 폴더에 구성 요소의 아카이브 AWS IoT Greengrass 아티팩트를 압축 해제합니다. 예를 들어 이 변수를 사용하여 구성 요소 수명 주기에서 실행할 스크립트의 위치를 식별할 수 있습니다.

각 아티팩트는 압축이 풀린 경로에 있는 폴더로 압축이 풀립니다. 여기서 폴더는 아티팩트에서 확장자를 뺀 것과 같은 이름을 가집니다. 예를 들어, 이름이 지정된 ZIP 아티팩트는 폴더로 압축을 풉니다. `models.zip` `{artifacts:decompressedPath}/models`

이 경로의 폴더는 읽기 전용입니다. 아티팩트 파일을 수정하려면 파일을 현재 작업 디렉토리(`$PWD`또는`.`) 와 같은 다른 위치에 복사하십시오. 그런 다음 거기서 파일을 수정합니다.

구성 요소 종속성에서 아티팩트를 읽거나 실행하려면 해당 아티팩트 `Read` 또는 `Execute` 권한이 있어야 합니다. `ALL` 자세한 내용은 구성 요소 레시피에 정의한 아티팩트 권한을 참조하십시오.

이 레시피 변수에는 다음과 같은 입력이 있습니다.

- component_dependency_name— (선택 사항) 쿼리할 구성 요소 종속성의 이름. 이 레시피가 정의하는 구성 요소를 쿼리하려면 이 세그먼트를 생략하십시오. 직접 종속성만 지정할 수 있습니다.

component_dependency_name`:work:path`

이 기능은 Greengrass 핵 구성 요소 v2.0.4 이상에서 사용할 수 있습니다.

이 레시피가 정의하는 구성 요소 또는 이 구성 요소가 종속된 구성 요소의 작업 경로입니다. 이 레시피 변수의 값은 구성 요소 컨텍스트에서 실행할 때 `$PWD` 환경 변수 및 pwd 명령의 출력과 동일합니다.

이 레시피 변수를 사용하여 구성 요소와 종속성 간에 파일을 공유할 수 있습니다.

이 경로의 폴더는 이 레시피가 정의하는 구성 요소와 동일한 사용자 및 그룹으로 실행되는 다른 구성 요소에서 읽고 쓸 수 있습니다.

이 레시피 변수에는 다음과 같은 입력이 있습니다.

- `component_dependency_name` — (선택 사항) 쿼리할 구성 요소 종속성의 이름. 이 레시피가 정의하는 구성 요소를 쿼리하려면 이 세그먼트를 생략하십시오. 직접 종속성만 지정할 수 있습니다.

`kernel:rootPath`

AWS IoT Greengrass코어 루트 경로.

`iot:thingName`

이 기능은 Greengrass 핵 구성 요소 v2.3.0 이상에서 사용할 수 있습니다.

코어 디바이스 사물의 이름. AWS IoT

레시피 예제

다음 레시피 예제를 참조하여 구성 요소에 대한 레시피를 만들 수 있습니다.

AWS IoT GreengrassGreengrass 소프트웨어 카탈로그라고 하는 Greengrass 구성 요소의 색인을 큐레이션합니다. 이 카탈로그는 Greengrass 커뮤니티에서 개발한 Greengrass 구성 요소를 추적합니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다. 자세한 설명은 커뮤니티 구성 요소 섹션을 참조하세요.

주제

- 헬로 월드 컴포넌트 레시피
- Python 런타임 컴포넌트 예제
- 여러 필드를 지정하는 구성 요소 레시피

헬로 월드 컴포넌트 레시피

다음 레시피는 Python 스크립트를 실행하는 Hello World 구성 요소를 설명합니다. 이 구성 요소는 모든 플랫폼을 지원하며 Python 스크립트에 인수로 AWS IoT Greengrass 전달되는 `Message` 매개 변수를 받아들입니다. 다음은 시작하기 자습서에 있는 Hello World 구성 요소의 레시피입니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
```

```
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Python 런타임 컴포넌트 예제

다음 레시피는 Python을 설치하는 구성 요소를 설명합니다. 이 구성 요소는 64비트 Linux 기기를 지원합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": "apt-get update\napt-get install python3.7"
      }
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install: |
        apt-get update
        apt-get install python3.7
```

여러 필드를 지정하는 구성 요소 레시피

다음 컴포넌트 레시피는 여러 레시피 필드를 사용합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
  },
  "ComponentDependencies": {
    "BarService": {
      "VersionRequirement": "^1.1.0",
      "DependencyType": "SOFT"
    },
    "BazService": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git"
        },
        "Setenv": {
          "environment_variable1": "variable_value1",
          "environment_variable2": "variable_value2"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world_linux.py"
        }
      ]
    },
    {
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git",
          "RequiresPrivilege": "true"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
      Setenv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.zip'
        Unarchive: ZIP
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world_linux.py'
  - Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.py'
```

구성 요소 환경 변수 참조

AWS IoT GreengrassCore 소프트웨어는 구성 요소의 수명 주기 스크립트를 실행할 때 환경 변수를 설정합니다. 컴포넌트에서 이러한 환경 변수를 가져와서 사물 이름 및 Greengrass nucleus 버전을 가져

올 수 있습니다.AWS 리전 또한 소프트웨어는 구성 요소가 프로세스 간 통신 SDK를 사용하고 AWS서비스와 상호 작용하는 데 필요한 환경 변수를 설정합니다.

구성 요소의 라이프사이클 스크립트에 대한 사용자 지정 환경 변수를 설정할 수도 있습니다. 자세한 내용은 Setenv를 참조하십시오.

AWS IoT GreengrassCore 소프트웨어는 다음과 같은 환경 변수를 설정합니다.

`AWS_IOT_THING_NAME`

이 Greengrass 코어 장치를 나타내는AWS IoT 것의 이름입니다.

`AWS_REGION`

이 Greengrass 코어 장치가 작동하는AWS 리전 곳.

AWSSDK는 이 환경 변수를 사용하여 사용할 기본 지역을 식별합니다. 이 변수는 다음과 같습니다`AWS_DEFAULT_REGION`.

`AWS_DEFAULT_REGION`

이 Greengrass 코어 장치가 작동하는AWS 리전 곳.

AWS CLI는 이 환경 변수를 사용하여 사용할 기본 지역을 식별합니다. 이 변수는 다음과 같습니다`AWS_REGION`.

`GGC_VERSION`

이 Greengrass 코어 장치에서 실행되는 Greengrass 핵 구성 요소의 버전입니다.

`GG_ROOT_CA_PATH`

이 기능은 Greengrass 핵심 구성 요소 v2.5.5 이상에서 사용할 수 있습니다.

Greengrass 핵에서 사용하는 CA (루트 인증 기관) 인증서의 경로입니다.

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

구성 요소가AWS IoT Greengrass Core 소프트웨어와 통신하는 데 사용하는 IPC 소켓 경로입니다. 자세한 정보는 AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core을 참조하세요.

`SVCUID`

구성 요소가 IPC 소켓에 연결하고AWS IoT Greengrass Core 소프트웨어와 통신하는 데 사용하는 비밀 토큰입니다. 자세한 정보는 AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core을 참조하세요.

`AWS_CONTAINER_AUTHORIZATION_TOKEN`

구성 요소가 토큰 교환 서비스 구성 요소에서 자격 증명을 검색하는 데 사용하는 비밀 토큰입니다.

`AWS_CONTAINER_CREDENTIALS_FULL_URI`

구성 요소가 토큰 교환 서비스 구성 요소에서 자격 증명을 검색하도록 요청하는 URI입니다.

디바이스에 AWS IoT Greengrass 구성 요소 배포

를 AWS IoT Greengrass 사용하여 장치 또는 장치 그룹에 구성 요소를 배포할 수 있습니다. 배포를 사용하여 장치로 전송되는 구성 요소 및 구성을 정의합니다. AWS IoT GreengrassGreengrass 코어 디바이스를 나타내는 대상, AWS IoT 사물 또는 사물 그룹에 배포합니다. AWS IoT GreengrassAWS IoT Core작업을 사용하여 코어 디바이스에 배포합니다. 작업이 디바이스에 롤아웃되는 방식을 구성할 수 있습니다.

코어 디바이스 배포

각 코어 기기는 해당 기기에 대한 배포 구성 요소를 실행합니다. 동일한 대상에 대한 새 배포는 대상에 대한 이전 배포를 덮어씁니다. 배포를 만들 때 코어 장치의 기존 소프트웨어에 적용할 구성 요소 및 구성을 정의합니다.

대상 배포를 수정하면 이전 개정의 구성 요소를 새 개정의 구성 요소로 교체합니다. 예를 들어, 로그 매니저 및 시크릿 매니저 구성 요소를 사물 그룹에 `TestGroup` 배포합니다. 그런 다음 Secret Manager 구성 요소만 지정하는 또 다른 배포를 생성합니다. `TestGroup` 따라서 해당 그룹의 핵심 기기는 더 이상 로그 관리자를 실행하지 않습니다.

플랫폼 종속성 해결

코어 기기가 배포를 받으면 구성 요소가 코어 기기와 호환되는지 확인합니다. 예를 들어, 를 Firehose Windows 대상에 배포하면 배포가 실패합니다.

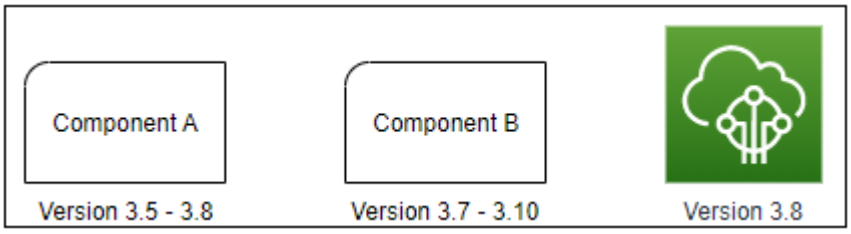
구성 요소 종속성 해결

또한 코어 디바이스는 이 사물 그룹에 다른 구성 요소를 배포하기 위한 버전 제약 조건과 각 구성 요소 종속성이 호환되는지 여부도 확인합니다. 구성 요소의 버전 제약이 겹치는 경우 Greengrass는 해당 구성 요소 중 가장 높은 버전의 구성 요소를 사용합니다. 예:

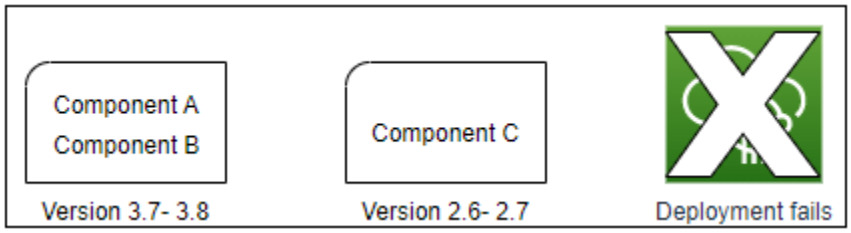
- 구성 요소 A를 에 배포합니다. `TestGroup` 구성 요소 A는 구성 요소 `com.example.PythonRuntime` 버전 3.5 - 3.10에 따라 달라집니다.

- 그런 다음 구성 요소 B를 에 TestGroup 배포합니다. 구성 요소 B는 구성 요소 com.example.PythonRuntime 버전 3.7~3.8에 따라 달라집니다.

따라서 코어 디바이스는 이 버전이 버전 제약이 겹치는 가장 적용 가능한 버전 중 가장 높은 버전이기 때문에 com.example.PythonRuntime 구성 요소 버전 3.8을 배포할 수 있다고 TestGroup 판단합니다.



그런 다음 구성 요소 C를 에 TestGroup 배포합니다. 구성 요소 C는 구성 요소 com.example.PythonRuntime 버전 2.6 - 2.7에 따라 달라집니다. 2.6 - 2.7 및 3.7 - 3.8 제약 조건을 충족하는 구성 요소 버전이 없기 때문에 이 배포가 실패합니다.



사물 그룹에서 장치 제거

사물 그룹에서 코어 디바이스를 제거하는 경우 구성 요소 배포 동작은 코어 디바이스가 실행하는 Greengrass Nucleus 버전에 따라 달라집니다.

2.5.1 and later

사물 그룹에서 코어 디바이스를 제거하는 경우 AWS IoT 정책은 권한을 부여하는지 여부에 따라 동작이 달라집니다. greengrass:ListThingGroupsForCoreDevice 핵심 장치에 대한 이 권한 및 AWS IoT 정책에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여.

- AWS IoT정책에서 이 권한을 부여하는 경우

  사물 그룹에서 핵심 장치를 AWS IoT Greengrass 제거하면 다음에 장치를 배포할 때 사물 그룹의 구성 요소가 제거됩니다. 장치의 구성 요소가 다음 배포에 포함되는 경우 해당 구성 요소는 장치에서 제거되지 않습니다.

- AWS IoT정책에서 이 권한을 부여하지 않는 경우

  사물 그룹에서 핵심 장치를 제거하는 경우 은 해당 사물 그룹의 구성 요소를 장치에서 삭제하지 AWS IoT Greengrass 않습니다.

  디바이스에서 구성 요소를 제거하려면 Greengrass CLI의 배포 생성 명령을 사용합니다. `--remove`인수로 제거할 구성 요소를 지정하고, 인수를 사용하여 사물 그룹을 지정합니다. `--groupId`

2.5.0

사물 그룹에서 코어 장치를 AWS IoT Greengrass 제거하면 다음에 장치를 배포할 때 사물 그룹의 구성 요소가 제거됩니다. 장치의 구성 요소가 다음 배포에 포함되는 경우 해당 구성 요소는 장치에서 제거되지 않습니다.

이 동작을 수행하려면 코어 기기의 AWS IoT 정책에서 `greengrass:ListThingGroupsForCoreDevice` 권한을 부여해야 합니다. 코어 디바이스에 이 권한이 없는 경우 코어 디바이스는 배포를 적용하지 못합니다. 자세한 설명은 AWS IoT Greengrass에 대한 디바이스 인증 및 권한 부여 섹션을 참조하세요.

2.0.x - 2.4.x

사물 그룹에서 핵심 장치를 제거해도 사물 그룹의 구성 요소는 장치에서 AWS IoT Greengrass 삭제되지 않습니다.

디바이스에서 구성 요소를 제거하려면 Greengrass CLI의 배포 생성 명령을 사용합니다. `--remove`인수로 제거할 구성 요소를 지정하고, 인수를 사용하여 사물 그룹을 지정합니다. `--groupId`

배포

배포는 연속적입니다. 배포를 만들 때 온라인 AWS IoT Greengrass 상태의 대상 장치에 배포를 배포하십시오. 대상 장치가 온라인 상태가 아닌 경우 다음에 연결할 때 배포를 AWS IoT Greengrass 수신합니다. 대상 사물 그룹에 핵심 장치를 추가하면 해당 사물 그룹에 대한 최신 배포를 장치에 AWS IoT Greengrass 보냅니다.

코어 기기는 기본적으로 구성 요소를 배포하기 전에 장치의 각 구성 요소에 알림을 보냅니다. Greengrass 구성 요소는 알림에 응답하여 배포를 연기할 수 있습니다. 기기의 배터리 잔량이 부족하거나 중단할 수 없는 프로세스를 실행 중인 경우에는 배포를 연기하는 것이 좋습니다. 자세한 설명은 튜

토리얼: 구성 요소 업데이트를 연기하는 Greengrass 구성 요소 개발 섹션을 참조하세요. 배포를 생성할 때 구성 요소에 알리지 않고 배포하도록 구성할 수 있습니다.

각 대상 사물 또는 사물 그룹에는 한 번에 하나의 배포가 있을 수 있습니다. 즉, 대상에 대한 배포를 만들면 해당 대상 배포의 이전 버전을 더 AWS IoT Greengrass 이상 배포하지 않습니다.

배포 옵션

배포에서는 업데이트를 받는 장치와 업데이트 배포 방법을 제어할 수 있는 몇 가지 옵션을 제공합니다. 배포를 생성할 때 다음 옵션을 구성할 수 있습니다.

- AWS IoT Greengrass구성 요소

  대상 장치에 설치하고 실행할 구성 요소를 정의합니다. AWS IoT Greengrass구성 요소는 Greengrass 코어 디바이스에서 배포하고 실행하는 소프트웨어 모듈입니다. 디바이스는 컴포넌트가 디바이스의 플랫폼을 지원하는 경우에만 컴포넌트를 수신합니다. 이렇게 하면 대상 장치가 여러 플랫폼에서 실행되는 경우에도 장치 그룹에 배포할 수 있습니다. 구성 요소가 장치의 플랫폼을 지원하지 않는 경우 구성 요소는 장치에 배포되지 않습니다.

  사용자 지정 구성 요소 및 AWS 제공된 구성 요소를 장치에 배포할 수 있습니다. 구성 요소를 배포할 때 구성 요소 종속성을 AWS IoT Greengrass 식별하여 해당 구성 요소도 배포합니다. 자세한 내용은 AWS IoT Greengrass구성 요소 개발 및 AWS-제공된 구성 요소 섹션을 참조하세요.

  각 구성 요소에 배포할 버전 및 구성 업데이트를 정의합니다. 구성 업데이트는 코어 장치에서 구성 요소의 기존 구성을 수정하는 방법 또는 구성 요소가 코어 장치에 없는 경우 구성 요소의 기본 구성을 수정하는 방법을 지정합니다. 기본값으로 재설정할 구성 값과 코어 장치에 병합할 새 구성 값을 지정할 수 있습니다. 코어 장치가 여러 대상에 대한 배포를 수신하고 각 배포에서 호환되는 구성 요소 버전을 지정하면 코어 장치는 배포를 만들 때의 타임스탬프를 기반으로 구성 업데이트를 순서대로 적용합니다. 자세한 설명은 구성 요소 구성 업데이트 섹션을 참조하세요.

  > **Important**
  >
  > 구성 요소를 배포할 때 해당 구성 요소의 모든 종속성 중에서 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 장치를 추가하거나 해당 장치를 대상으로 하는 배포를 업데이트하면 AWS 제공된 공용 구성 요소의 새 패치 버전이 핵심 장치에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 장치가 예기치 않게 다시 시작될 수 있습니다.
  > 디바이스에서 실행 중인 구성 요소가 의도하지 않게 업데이트되는 것을 방지하려면 배포를 생성할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT

GreengrassCore 소프트웨어의 업데이트 동작에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트.

- 배포 정책

  구성을 안전하게 배포할 수 있는 시기와 배포가 실패할 경우 취해야 할 조치를 정의하세요. 구성 요소가 업데이트 가능하다고 보고할 때까지 기다릴지 여부를 지정할 수 있습니다. 실패한 배포를 적용하는 경우 장치를 이전 구성으로 롤백할지 여부도 지정할 수 있습니다.

- 구성 중지

  배포를 중지할 시기와 방법을 정의합니다. 정의한 기준이 충족되면 배포가 중지되고 실패합니다. 예를 들어, 최소 개수의 장치에서 배포를 수신한 후 일정 비율의 장치가 해당 배포를 적용하지 못하면 배포를 중지하도록 구성할 수 있습니다.

- Rollout configuration(롤아웃 구성)

  대상 장치에 배포가 배포되는 속도를 정의하십시오. 최소 및 최대 속도 범위를 사용하여 기하급수적 속도 증가를 구성할 수 있습니다.

- 타임아웃 구성

  각 디바이스가 배포를 적용하는 데 필요한 최대 시간을 정의합니다. 디바이스가 지정한 기간을 초과할 경우 디바이스는 배포를 적용하지 못합니다.

Important

사용자 지정 구성 요소는 S3 버킷의 아티팩트를 정의할 수 있습니다. AWS IoT GreengrassCore 소프트웨어는 구성 요소를 배포할 때 에서 구성 요소의 아티팩트를 다운로드합니다. AWS 클라우드 코어 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않습니다. S3 버킷의 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스 역할이 해당 버킷에서 아티팩트를 다운로드할 수 있는 권한을 부여해야 합니다. 자세한 내용은 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용을(를) 참조하세요.

주제

- 배포 만들기
- 하위 배포 생성
- 배포 수정

- 배포 취소
- 배포 상태를 확인합니다 의 상태를 확인하세요 의

배포 만들기

사물 또는 사물 그룹을 대상으로 하는 배포를 생성할 수 있습니다.

배포를 생성할 때 배포할 소프트웨어 구성 요소와 배포 작업이 대상 장치에 롤아웃되는 방식을 구성합니다. 에 제공하는 JSON 파일에서 배포를 정의할 수 있습니다. AWS CLI

배포 대상은 구성 요소를 실행할 장치를 결정합니다. 코어 기기 하나에 배포하려면 사물을 지정하세요. 여러 코어 디바이스에 배포하려면 해당 디바이스를 포함하는 사물 그룹을 지정하십시오. 사물 그룹을 구성하는 방법에 대한 자세한 내용은 AWS IoT개발자 안내서의 정적 사물 그룹 및 동적 사물 그룹을 참조하십시오.

이 섹션의 단계에 따라 대상에 대한 배포를 생성하십시오. 배포가 있는 대상의 소프트웨어 구성 요소를 업데이트하는 방법에 대한 자세한 내용은 을 참조하십시오배포 수정.

> **Warning**
>
> 이 CreateDeployment작업을 통해 코어 디바이스에서 구성 요소를 제거할 수 있습니다. 구성 요소가 새 배포가 아닌 이전 배포에 있는 경우 코어 장치는 해당 구성 요소를 제거합니다. 구성 요소를 제거하지 않으려면 먼저 ListDeployments작업을 사용하여 배포 대상에 이미 기존 배포가 있는지 확인하십시오. 그런 다음 새 배포를 만들 때 GetDeployment작업을 사용하여 기존 배포에서 시작하십시오.

배포를 만들려면 (AWS CLI)

1. 라는 `deployment.json` 파일을 만든 다음 다음 JSON 객체를 파일에 복사합니다. *`TargetARN`* *#* 배포 대상으로 지정할 사물 또는 사물 그룹의 ARN으로 대체합니다. AWS IoT 사물 및 사물 그룹 ARN의 형식은 다음과 같습니다.

   - 사물: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
   - 사물 그룹: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

   ```
   {
     "targetArn": "targetArn"
   ```

```
}
```

2. 배포 대상에 수정하려는 기존 배포가 있는지 확인하십시오. 다음을 따릅니다.

   a. 다음 명령을 실행하여 배포 대상의 배포를 나열합니다. *TargetARN#* 대상 사물 또는 사물 그룹의 ARN으로 대체합니다AWS IoT.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 응답이 비어 있는 경우 대상에 기존 배포가 없는 것이므로 건너뛰어도 됩니다. Step 3 그렇지 않으면 다음 단계에서 사용할 수 있도록 응답에서 `deploymentId`을 복사합니다.

   > **Note**
   >
   > 대상의 최신 수정 버전 이외의 배포를 수정할 수도 있습니다. `--history-filter ALL`인수를 지정하여 대상의 모든 배포를 나열합니다. 그런 다음 수정하려는 배포의 ID를 복사합니다.

   b. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. *Deploymentid# ## ### ID#* 교체합니다.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   응답에는 배포의 세부 정보가 포함됩니다.

   c. 이전 명령의 응답에서 다음 키-값 쌍을 로 복사합니다. `deployment.json` 새 배포에서 이러한 값을 변경할 수 있습니다.

   - `deploymentName`— 디플로이먼트 이름.
   - `components`— 디플로이먼트의 구성 요소. 구성 요소를 제거하려면 이 개체에서 구성 요소를 제거하십시오.
   - `deploymentPolicies`— 배포 정책.
   - `iotJobConfiguration`— 배포의 작업 구성.
   - `tags`— 디플로이먼트의 태그.

3. (선택 사항) 배포 이름을 정의합니다. *######### #######* 이름으로 바꿉니다.

```
{
```

```
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
}
```

4. 각 구성 요소를 추가하여 대상 장치를 배포합니다. 이렇게 하려면 키-값 쌍을 `components` 개체에 추가합니다. 여기서 키는 구성 요소 이름이고 값은 해당 구성 요소의 세부 정보가 포함된 개체입니다. 추가하는 각 구성 요소에 대해 다음 세부 정보를 지정하십시오.

   - `version`— 배포할 구성 요소 버전.
   - `configurationUpdate`— 배포할 구성 업데이트입니다. 업데이트는 각 대상 장치에 있는 구성 요소의 기존 구성을 수정하거나, 대상 장치에 구성 요소가 없는 경우 구성 요소의 기본 구성을 수정하는 패치 작업입니다. 다음과 같은 구성 업데이트를 지정할 수 있습니다.
     - Reset `reset` update () - (선택 사항) 대상 장치에서 기본값으로 재설정할 구성 값을 정의하는 JSON 포인터 목록입니다. AWS IoT Greengrass Core 소프트웨어는 병합 업데이트를 적용하기 전에 재설정 업데이트를 적용합니다. 자세한 설명은 업데이트 재설정 섹션을 참조하세요.
     - 업데이트 병합 (`merge`) — (선택 사항) 대상 장치에 병합할 구성 값을 정의하는 JSON 문서입니다. JSON 문서를 문자열로 직렬화해야 합니다. 자세한 설명은 병합 업데이트 섹션을 참조하세요.
   - `runWith`— (선택 사항) AWS IoT Greengrass Core 소프트웨어가 코어 장치에서 이 구성 요소의 프로세스를 실행하는 데 사용하는 시스템 프로세스 옵션입니다. `runWith`개체에서 매개변수를 생략하면 AWS IoT Greengrass Core 소프트웨어는 Greengrass nucleus 구성 요소에 구성한 기본값을 사용합니다.

     다음 옵션 중 하나를 지정할 수 있습니다.

     - `posixUser`— Linux 코어 디바이스에서 이 구성 요소를 실행하는 데 사용할 POSIX 시스템 사용자 및 그룹 (선택 사항) 사용자 및 그룹 (지정된 경우) 은 각 Linux 코어 장치에 존재해야 합니다. `user:group` 형식으로 사용자와 그룹을 콜론(`:`)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass Core 소프트웨어는 사용자의 기본 그룹을 사용합니다. 자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.
     - `windowsUser`— Windows 코어 장치에서 이 구성 요소를 실행하는 데 사용할 Windows 사용자입니다. 사용자는 각 Windows 코어 장치에 존재해야 하며 사용자 이름과 암호는 LocalSystem 계정의 자격 증명 관리자 인스턴스에 저장되어 있어야 합니다. 자세한 설명은 구성 요소를 실행하는 사용자를 구성하십시오. 섹션을 참조하세요.

       이 기능은 v2.5.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

- `systemResourceLimits`— 이 구성 요소의 프로세스에 적용할 시스템 리소스 제한 시스템 리소스 제한을 일반 및 비컨테이너식 Lambda 구성 요소에 적용할 수 있습니다. 자세한 설명은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 섹션을 참조하세요.

  다음 옵션 중 하나를 지정할 수 있습니다.

  - `cpus`— 이 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 장치의 경우 이 값을 2 설정하여 이 구성 요소의 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 기기에서는 이 값을 `0.25` 설정하여 이 구성 요소의 프로세스 CPU 사용량을 25% 로 제한할 수 있습니다. 이 값을 CPU AWS IoT Greengrass 코어 수보다 큰 수로 설정하는 경우 Core 소프트웨어는 구성 요소의 CPU 사용량을 제한하지 않습니다.
  - `memory`— 이 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 RAM 크기 (KB)

  이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

Example 기본 구성 업데이트 예시

다음 예제 `components` 객체는 다음과 같은 구성 매개 변수가 필요한 구성 요소를 배포하도록 지정합니다`pythonVersion`. `com.example.PythonRuntime`

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Example 재설정 및 병합 업데이트를 포함한 구성 업데이트의 예

다음과 같은 기본 구성을 가진 산업용 대시보드 구성 요소를 예로 들어 보겠습니다. `com.example.IndustrialDashboard`

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

다음 구성 업데이트에는 다음 지침이 지정되어 있습니다.

1. HTTPS 설정을 기본값 (`true`) 으로 재설정합니다.
2. 산업용 태그 목록을 빈 목록으로 재설정합니다.
3. 두 보일러의 온도 및 압력 데이터 스트림을 식별하는 산업용 태그 목록을 병합하십시오.

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
}
```

다음 예제 components 객체는 이 산업용 대시보드 구성 요소 및 구성 업데이트를 배포하도록 지정합니다.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

5. (선택 사항) 배포를 위한 배포 정책을 정의합니다. 코어 디바이스가 안전하게 배포를 적용할 수 있는 시기 또는 코어 디바이스가 배포를 적용하지 못할 경우 취해야 할 조치를 구성할 수 있습니다. 이렇게 하려면 deploymentPolicies deployment.json 개체를 추가한 후 다음 중 하나를 수행하십시오.

   1. (선택 사항) 구성 요소 업데이트 정책을 지정합니다 (componentUpdatePolicy). 이 정책은 배포를 통해 구성 요소가 업데이트 준비가 될 때까지 업데이트를 연기할 수 있는지 여부를 정의합니다. 예를 들어 구성 요소를 다시 시작하여 업데이트를 적용하려면 먼저 리소스를 정리하거나 중요한 작업을 완료해야 할 수 있습니다. 또한 이 정책은 구성 요소가 업데이트 알림에 응답해야 하는 시간을 정의합니다.

      이 정책은 다음 매개 변수를 가진 객체입니다.

      - action— (선택 사항) 구성 요소에 알림을 보내고 업데이트 준비가 되면 보고할 때까지 기다릴지 여부. 다음 옵션 중 하나를 선택합니다.
        - NOTIFY_COMPONENTS - 배포는 해당 구성 요소를 중지하고 업데이트하기 전에 각 구성 요소에 알립니다. 구성 요소는 SubscribeToComponentUpdates IPC 작업을 사용하여 이러한 알림을 수신할 수 있습니다.

- SKIP_NOTIFY_COMPONENTS - 배포는 구성 요소에 알리거나 업데이트해도 안전할 때까지 기다리지 않습니다.

  기본값은 NOTIFY_COMPONENTS입니다.

- timeoutInSeconds각 구성 요소가 DeferComponentUpdate IPC 작업과 함께 업데이트 알림에 응답해야 하는 시간 (초) 이 시간 내에 구성 요소가 응답하지 않으면 코어 기기에서 배포가 진행됩니다.

  기본값은 60초입니다.

2. (선택 사항) 구성 검증 정책 (configurationValidationPolicy) 을 지정합니다. 이 정책은 각 구성 요소가 배포에서 구성 업데이트를 검증해야 하는 시간을 정의합니다. 구성 요소는 SubscribeToValidateConfigurationUpdates IPC 작업을 사용하여 자체 구성 업데이트에 대한 알림을 구독할 수 있습니다. 그러면 구성 요소가 SendConfigurationValidityReport IPC 작업을 사용하여 구성 업데이트가 유효한지 AWS IoT Greengrass Core 소프트웨어에 알릴 수 있습니다. 구성 업데이트가 유효하지 않으면 배포가 실패합니다.

   이 정책은 다음 매개변수가 있는 객체입니다.

   - timeoutInSeconds(선택 사항) 각 구성 요소가 구성 업데이트를 검증하는 데 걸리는 시간 (초). 이 시간 내에 구성 요소가 응답하지 않으면 코어 기기에서 배포가 진행됩니다.

     기본값은 30초입니다.

3. (선택 사항) 장애 처리 정책 (failureHandlingPolicy) 을 지정합니다. 이 정책은 배포가 실패할 경우 장치를 롤백할지 여부를 정의하는 문자열입니다. 다음 옵션 중 하나를 선택합니다.

   - ROLLBACK— 코어 디바이스에서 배포가 실패하는 경우 AWS IoT Greengrass 코어 소프트웨어는 해당 코어 디바이스를 이전 구성으로 롤백합니다.
   - DO_NOTHING— 코어 디바이스에서 배포가 실패하는 경우 AWS IoT Greengrass Core 소프트웨어는 새 구성을 유지합니다. 새 구성이 유효하지 않으면 구성 요소가 손상될 수 있습니다.

   기본값은 ROLLBACK입니다.

에서의 배포는 다음 예와 비슷할 deployment.json 수 있습니다.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
```

```
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}
```

6. (선택 사항) 배포 중지, 출시 또는 제한 시간을 정의합니다. AWS IoT GreengrassAWS IoT Core 작업을 사용하여 코어 장치에 배포를 전송하므로 이러한 옵션은 작업의 구성 옵션과 동일합니다. AWS IoT Core 자세한 내용은 AWS IoT개발자 안내서의 Job rollout 및 abort 구성을 참조하십시오.

   작업 옵션을 정의하려면 `iotJobConfiguration` 오브젝트를 추가하십시오. `deployment.json` 그런 다음 구성할 옵션을 정의합니다.

   에서의 배포는 다음 예와 비슷할 `deployment.json` 수 있습니다.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
```

```
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  },
  "iotJobConfiguration": {
    "abortConfig": {
      "criteriaList": [
        {
          "action": "CANCEL",
          "failureType": "ALL",
          "minNumberOfExecutedThings": 100,
          "thresholdPercentage": 5
        }
      ]
    },
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 5,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 10,
          "numberOfSucceededThings": 5
        }
      },
      "maximumPerMinute": 50
    },
    "timeoutConfig":  {
      "inProgressTimeoutInMinutes": 5
    }
  }
```

```
}
```

7. (선택 사항) 배포에 태그 (`tags`) 를 추가합니다. 자세한 설명은 AWS IoT Greengrass Version 2 리소스 태깅 섹션을 참조하세요.
8. 다음 명령어를 실행하여 배포를 생성합니다`deployment.json`.

   ```
   aws greengrassv2 create-deployment --cli-input-json file://deployment.json
   ```

   응답에는 이 배포를 `deploymentId` 식별하는 a가 포함됩니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 배포 상태를 확인합니다 의 상태를 확인하세요 의을(를) 참조하세요.

구성 요소 구성 업데이트

구성 요소 구성은 각 구성 요소의 매개 변수를 정의하는 JSON 개체입니다. 각 구성 요소의 레시피는 기본 구성을 정의하며, 구성 요소를 코어 장치에 배포할 때 수정되는 기본 구성을 정의합니다.

배포를 생성할 때 각 구성 요소에 적용할 구성 업데이트를 지정할 수 있습니다. 구성 업데이트는 패치 작업입니다. 즉, 업데이트는 코어 장치에 있는 구성 요소 구성을 수정합니다. 코어 기기에 구성 요소가 없는 경우 구성 업데이트는 해당 배포의 기본 구성을 수정하여 적용합니다.

구성 업데이트는 재설정 업데이트와 병합 업데이트를 정의합니다. 재설정 업데이트는 기본값으로 재설정하거나 제거할 구성 값을 정의합니다. 병합 업데이트는 구성 요소에 설정할 새 구성 값을 정의합니다. 구성 업데이트를 배포하면 AWS IoT Greengrass Core 소프트웨어가 병합 업데이트 전에 재설정 업데이트를 실행합니다.

구성 요소는 배포한 구성 업데이트의 유효성을 검사할 수 있습니다. 구성 요소는 배포에서 구성이 변경될 때 알림을 받도록 구독하며, 지원하지 않는 구성을 거부할 수 있습니다. 자세한 설명은 구성 요소 구성과 상호 작용 섹션을 참조하세요.

주제

- 업데이트 재설정
- 병합 업데이트
- 예

업데이트 재설정

재설정 업데이트는 코어 기기에서 기본값으로 재설정할 구성 값을 정의합니다. 구성 값에 기본값이 없는 경우 재설정 업데이트는 구성 요소의 구성에서 해당 값을 제거합니다. 이렇게 하면 잘못된 구성으로 인해 손상된 구성 요소를 수정하는 데 도움이 될 수 있습니다.

JSON 포인터 목록을 사용하여 재설정할 구성 값을 정의합니다. JSON 포인터는 슬래시로 시작합니다. / 중첩된 구성 요소 구성에서 값을 식별하려면 정방향 슬래시 (/) 를 사용하여 구성의 각 수준에 대해 키를 구분하십시오. 자세한 내용은 JSON 포인터 사양을 참조하십시오.

> **Note**
>
> 전체 목록만 기본값으로 재설정할 수 있습니다. 업데이트 재설정을 사용하여 목록의 개별 요소를 재설정할 수는 없습니다.

구성 요소의 전체 구성을 기본값으로 재설정하려면 빈 문자열 하나를 재설정 업데이트로 지정하십시오.

```
"reset": [""]
```

병합 업데이트

병합 업데이트는 코어의 구성 요소 구성에 삽입할 구성 값을 정의합니다. 병합 업데이트는 재설정 업데이트에서 지정한 경로의 값을 재설정한 후 AWS IoT Greengrass Core 소프트웨어가 병합하는 JSON 객체입니다. AWS CLI 또는 AWS SDK를 사용하는 경우 이 JSON 객체를 문자열로 직렬화해야 합니다.

구성 요소의 기본 구성에 없는 키-값 쌍을 병합할 수 있습니다. 같은 키를 가진 값과 유형이 다른 키-값 쌍을 병합할 수도 있습니다. 그러면 기존 값이 새 값으로 바뀝니다. 즉, 구성 개체의 구조를 변경할 수 있습니다.

null 값과 빈 문자열, 목록 및 개체를 병합할 수 있습니다.

> **Note**
>
> 목록에 요소를 삽입하거나 추가하는 용도로는 병합 업데이트를 사용할 수 없습니다. 전체 목록을 바꾸거나 각 요소에 고유한 키가 있는 개체를 정의할 수 있습니다.
> AWS IoT Greengrass 구성 값에 JSON을 사용합니다. JSON은 숫자 유형을 지정하지만 정수와 부동 소수점을 구분하지는 않습니다. 따라서 구성 값이 부동 소수점으로 변환될 수 있습니

다. AWS IoT Greengrass구성 요소가 올바른 데이터 유형을 사용하도록 하려면 숫자 구성 값을 문자열로 정의하는 것이 좋습니다. 그런 다음 구성 요소가 해당 값을 정수 또는 부동 소수점으로 파싱하도록 하십시오. 이렇게 하면 구성 및 코어 장치에서 구성 값의 유형이 동일할 수 있습니다.

병합 업데이트에 레시피 변수를 사용하십시오.

이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

Greengrass nucleus의 interpolateComponentConfiguration구성 옵션을 로 `true` 설정하면 레시피 변수 이외의 레시피 변수를 병합 업데이트에 사용할 수 있습니다. *`component_dependency_name`*`:configuration:`*`json_pointer`* 예를 들어 병합 업데이트에서 `{iot:thingName}` 레시피 변수를 사용하여 IPC (프로세스 간 통신) 권한 부여 정책과 같은 구성 요소 구성 값에 코어 장치의 AWS IoT 사물 이름을 포함할 수 있습니다.

예

다음 예제는 다음과 같은 기본 구성을 가진 대시보드 구성 요소의 구성 업데이트를 보여줍니다. 이 예제 구성 요소는 산업용 장비에 대한 정보를 표시합니다.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

산업용 대시보드 구성 요소 레시피

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
```

```
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
        "useHttps": true,
        "port": {
          "http": 80,
          "https": 443
        },
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
```

```
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
      port:
        http: 80
        https: 443
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Example 예 1: 병합 업데이트

다음 구성 업데이트를 적용하는 배포를 생성합니다. 이 경우 병합 업데이트는 지정하지만 재설정 업데이트는 지정하지 않습니다. 이 구성 업데이트는 두 보일러의 데이터가 포함된 대시보드를 HTTP 포트 8080에 표시하도록 구성요소에 지시합니다.

Console

병합할 구성

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
```

```
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

다음 명령어는 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

`dashboard-deployment.json`파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port
\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

다음 Greengrass CLI 명령은 코어 디바이스에 로컬 배포를 생성합니다.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

이 dashboard-configuration.json 파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "com.example.IndustrialDashboard": {
    "MERGE": {
      "name": "Factory 2A",
      "network": {
        "useHttps": false,
        "port": {
          "http": 8080
        }
      },
      "tags": [
        "/boiler/1/temperature",
        "/boiler/1/pressure",
        "/boiler/2/temperature",
        "/boiler/2/pressure"
      ]
    }
  }
}
```

이 업데이트 이후 대시보드 구성 요소는 다음과 같이 구성됩니다.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

Example 예 2: 업데이트 재설정 및 병합

그런 다음 재설정 업데이트와 병합 업데이트를 지정하는 다음 구성 업데이트를 적용하는 배포를 생성합니다. 이러한 업데이트는 다양한 보일러의 데이터가 포함된 기본 HTTPS 포트에 대시보드를 표시하도록 지정합니다. 이러한 업데이트는 이전 예제의 구성 업데이트로 인한 구성을 수정합니다.

Console

경로 재설정

```
[
  "/network/useHttps",
  "/tags"
]
```

병합할 구성

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

다음 명령어는 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

`dashboard-deployment2.json`파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
```

```
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/3/temperature\",\"/boiler/3/pressure\",\"/
boiler/4/temperature\",\"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

다음 Greengrass CLI 명령은 코어 디바이스에 로컬 배포를 생성합니다.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

이 `dashboard-configuration2.json` 파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
  }
}
```

이 업데이트 이후 대시보드 구성 요소는 다음과 같이 구성됩니다.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure",
  ]
}
```

하위 배포 생성

> **Note**
>
> 서브디폴로이먼트 기능은 Greengrass nucleus 버전 2.9.0 이상에서 사용할 수 있습니다. Greengrass nucleus의 이전 구성 요소 버전을 사용하는 하위 배포에는 구성을 배포할 수 없습니다.

하위 배포는 상위 배포 내에 있는 더 작은 장치 하위 집합을 대상으로 하는 배포입니다. 하위 배포를 사용하여 더 작은 장치 하위 집합에 구성을 배포할 수 있습니다. 또한 하위 배포를 만들어 부모 배포에 실패한 상위 배포를 하나 이상의 장치에 장애가 발생하는 경우 해당 상위 배포를 다시 시도할 수 있습니다. 이 기능을 사용하면 상위 배포에서 실패한 장치를 선택하고 하위 배포가 성공할 때까지 구성을 테스트할 하위 배포를 만들 수 있습니다. 하위 배포에 성공하면 해당 구성을 상위 배포에 재배포할 수 있습니다.

이 섹션의 단계에 따라 하위 배포를 만들고 상태를 확인합니다. 배포를 만드는 방법에 대한 자세한 내용은 배포 만들기를 참조하십시오.

하위 배포를 만들려면 () AWS CLI

1. 다음 명령을 실행하여 사물 그룹의 최신 배포를 검색합니다. 명령의 ARN을 쿼리할 사물 그룹의 ARN으로 대체합니다. 해당 사물 그룹의 최신 배포를 `--history-filter` **LATEST_ONLY** 보도록 설정합니다.

   ```
   aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-
   id:thinggroup/thingGroupName --history-filter LATEST_ONLY
   ```

2. `deploymentId`응답의 내용을 다음 단계에서 사용할 list-deployments 명령에 복사합니다.
3. 다음 명령을 실행하여 배포 상태를 검색합니다. 쿼리할 배포의 *deploymentId* ID로 바꿉니다.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

4. 다음 단계에서 사용할 get-deployment 명령에 대한 응답의 내용을 복사합니다. `iotJobId`
5. 다음 명령을 실행하여 지정된 작업에 대한 작업 실행 목록을 검색합니다. *JobID#* 이전 단계의 `iotJobId` JobId로 바꿉니다. *###* 필터링하려는 상태로 바꾸십시오. 다음 상태로 결과를 필터링할 수 있습니다.

   - `QUEUED`
   - `IN_PROGRESS`
   - `SUCCEEDED`
   - `FAILED`
   - `TIMED_OUT`
   - `REJECTED`
   - `REMOVED`
   - `CANCELED`

   ```
   aws iot list-job-executions-for-job --job-id jobID --status status
   ```

6. 하위 배포에 새 AWS IoT 사물 그룹을 생성하거나 기존 사물 그룹을 사용하십시오. 그런 다음 이 AWS IoT 사물 그룹에 사물을 추가합니다. 사물 그룹을 사용하여 수많은 Greengrass 코어 디바이스를 관리합니다. 소프트웨어 구성 요소를 장치에 배포할 때 개별 장치 또는 장치 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포를 사용하여 사물 그룹에 장치를 추가할 수 있습니다. 추가한 후에는 해당 사물 그룹의 소프트웨어 구성 요소를 해당 장치에 배포할 수 있습니다.

새 사물 그룹을 생성하고 새 사물 그룹에 장치를 추가하려면 다음과 같이 하십시오.

a. AWS IoT사물 그룹을 생성합니다. 새 사물 그룹의 *MyGreengrassCoreGroup*이름으로 바꾸십시오. 사물 그룹 이름에는 콜론 (:) 을 사용할 수 없습니다.

> **Note**
>
> 하위 배포용 사물 그룹을 한 것과 함께 사용하는 경우 다른 `parentTargetArn` 상위 플릿과 함께 재사용할 수 없습니다. 사물 그룹을 이미 사용하여 다른 플릿에 대한 하위 배포를 생성한 경우 API는 오류를 반환합니다.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. 프로비저닝된 Greengrass 코어를 사물 그룹에 추가합니다. 다음 파라미터를 사용하여 다음 명령을 실행합니다.

- *MyGreengrassCore*프로비저닝된 Greengrass 코어의 이름으로 바꾸십시오.
- 사물 그룹의 *MyGreengrassCoreGroup*이름으로 바꾸십시오.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

요청이 성공하면 명령이 출력되지 않습니다.

7. 라는 `deployment.json` 파일을 만든 다음 다음 JSON 객체를 파일에 복사합니다. *TargetARN#* 하위 배포의 대상으로 삼을 사물 그룹의 ARN으로 AWS IoT 대체합니다. 하위 배포 대상은 사물 그룹만 될 수 있습니다. 사물 그룹 ARN의 형식은 다음과 같습니다.

- 사물 그룹 — `arn:aws:iot:region:account-id:thinggroup/thingGroupName`

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

8. 다음 명령을 다시 실행하여 원본 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. *Deploymentid#* 의 ID로 바꾸십시오. Step 1 이 배포 구성을 사용하여 하위 배포를 구성하고 필요에 따라 변경할 수 있습니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

응답에는 배포의 세부 정보가 포함됩니다. get-deployment명령의 응답에서 다음 키-값 쌍을 로 복사합니다. `deployment.json` 하위 배포에서 이러한 값을 변경할 수 있습니다. 이 명령의 세부 정보에 대한 자세한 내용은 을 참조하십시오 GetDeployment.

- `components`— 배포의 구성 요소. 구성 요소를 제거하려면 이 개체에서 구성 요소를 제거하십시오.
- `deploymentName`— 디플로이먼트 이름.
- `deploymentPolicies`— 배포 정책.
- `iotJobConfiguration`— 배포의 작업 구성.
- `parentTargetArn`— 부모 배포의 대상.
- `tags`— 디플로이먼트의 태그.

9. 다음 명령을 실행하여 하위 배포를 `deployment.json` 생성합니다. *SubDeploymentName# ### ### #### ####*.

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-
json file://deployment.json
```

응답에는 이 하위 배포를 식별하는 a가 포함됩니다. `deploymentId` 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 배포 상태 확인을 참조하십시오.

10. 하위 배포가 성공하면 해당 구성을 사용하여 상위 배포를 수정할 수 있습니다. 이전 단계에서 사용한 `deployment.json` 것을 복사합니다. JSON 파일의 를 상위 배포의 ARN으로 바꾸고 다음 명령어를 실행하여 이 새 구성을 사용하여 상위 배포를 생성합니다. `targetArn`

> **Note**
>
> 부모 플릿의 새 배포 버전을 생성하면 해당 부모 배포의 모든 배포 수정과 하위 배포가 대체됩니다. 자세한 내용은 배포 수정을 참조하십시오.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

응답에는 이 배포를 식별하는 deploymentId a가 포함됩니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 배포 상태를 확인합니다 의 상태를 확인하세요 의을(를) 참조하세요.

배포 수정

각 대상 사물 또는 사물 그룹에는 한 번에 하나의 활성 배포가 있을 수 있습니다. 이미 배포가 있는 대상에 대한 배포를 만들면 새 배포의 소프트웨어 구성 요소가 이전 배포의 소프트웨어 구성 요소를 대체합니다. 새 배포에서 이전 배포에서 정의한 구성 요소를 정의하지 않는 경우 AWS IoT Greengrass Core 소프트웨어는 대상 코어 장치에서 해당 구성 요소를 제거합니다. 코어 장치에서 실행되는 구성 요소를 이전 배포에서 대상으로 제거하지 않도록 기존 배포를 수정할 수 있습니다.

배포를 수정하려면 이전 배포에 있는 것과 동일한 구성 요소 및 구성으로 시작하는 배포를 만들어야 합니다. CreateDeployment작업을 사용하는데, 이는 배포를 만들 때 사용하는 것과 동일한 작업입니다.

배포를 수정하려면 () AWS CLI

1. 다음 명령을 실행하여 배포 대상의 배포를 나열합니다. *TargetARN#* 대상 사물 또는 사물 그룹의 ARN으로 대체합니다AWS IoT.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 응답의 deploymentId 내용을 복사하여 다음 단계에서 사용하십시오.

> **Note**
>
> 대상의 최신 수정 버전 이외의 배포를 수정할 수도 있습니다. `--history-filter ALL`인수를 지정하여 대상의 모든 배포를 나열합니다. 그런 다음 수정하려는 배포의 ID를 복사합니다.

2. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. *Deploymentid# ## ### ID#* 교체합니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

응답에는 배포의 세부 정보가 포함됩니다.

3. `deployment.json`이라는 파일을 만들고 이전 명령의 응답을 파일에 복사합니다.
4. `deployment.json`의 JSON 객체에서 다음 키-값 페어를 제거합니다.

- `deploymentId`
- `revisionId`
- `iotJobId`
- `iotJobArn`
- `creationTimestamp`
- `isLatestForTarget`
- `deploymentStatus`

이 CreateDeployment작업에는 다음과 같은 구조의 페이로드가 필요합니다.

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. `deployment.json`에서 다음을 수행합니다.

- 배포 이름 변경 (`deploymentName`).

- 배포의 구성 요소 변경 (components).
- 배포 정책 변경 (deploymentPolicies).
- 배포의 작업 구성을 변경합니다 (iotJobConfiguration).
- 배포의 태그를 변경합니다 (tags).

이러한 배포 세부 정보를 정의하는 방법에 대한 자세한 내용은 을 참조하십시오배포 만들기.

6. 다음 명령어를 실행하여 배포를 생성합니다deployment.json.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

응답에는 이 배포를 deploymentId 식별하는 a가 포함됩니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 배포 상태를 확인합니다 의 상태를 확인하세요 의을(를) 참조하세요.

배포 취소

활성 배포를 취소하여 해당 소프트웨어 구성 요소가AWS IoT Greengrass 코어 장치에 설치되지 않도록 할 수 있습니다. 사물 그룹을 대상으로 하는 배포를 취소하면 그룹에 추가한 핵심 장치는 해당 지속적 배포를 받지 못합니다. 코어 장치가 이미 배포를 실행하는 경우 배포를 취소해도 해당 장치의 구성 요소는 변경되지 않습니다. 새 배포를 만들거나 배포를 수정하여 취소된 배포를 받은 핵심 장치에서 실행되는 구성 요소를 수정해야 합니다.

배포 취소하기 (AWS CLI)

1. 다음 명령을 실행하여 대상에 대한 최신 배포 수정 버전의 ID를 찾습니다. 새 개정을 만들면 이전 배포가 취소되므로 최신 개정은 대상에 대해 활성화할 수 있는 유일한 배포입니다. *TargetARN#* 대상 사물 또는AWS IoT 사물 그룹의 ARN으로 바꿉니다.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. deploymentId응답에서 복사하여 다음 단계에서 사용합니다.

2. 다음 명령을 실행하여 배포를 취소합니다. *deploymentId* ID를 이전 단계의 ID로 바꿉니다.

```
aws greengrassv2 cancel-deployment --deployment-id deploymentId
```

작업이 성공하면 배포 상태가 로 변경됩니다CANCELED.

배포 상태를 확인합니다 의 상태를 확인하세요 의

에서 생성한 배포의 의 의 의 상태를 확인할 수AWS IoT Greengrass 있습니다. 또한 각 코어 디바이스에AWS IoT 배포의 의 의 의 상태를 확인할 수 있습니다. 배포가 활성화되어 있는 동안AWS IoT 작업의 상태는 입니다IN_PROGRESS. 배포의 새 버전을 생성하면 이전 수정 버전의AWS IoT 작업 상태가 로 변경됩니다CANCELLED.

주제

배포 상태를 확인합니다 의 상태를 확인하세요 의

대상 또는 ID로 식별하는 배포 상태를 확인할 수 있습니다.

대상별 배포 상태 확인하기 (AWS CLI)

- 다음 명령을 실행하여 대상에 대한 최신 상태를 검색합니다. *TargetArn#* 배포의 대상이 되는 사물 또는AWS IoT 사물 그룹의 Amazon 리소스 이름 (ARN) 으로 교체합니다.

  ```
  aws greengrassv2 list-deployments --target-arn targetArn
  ```

  응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 이 배포 객체에는 배포 상태가 포함됩니다.

ID로 배포 상태를 확인하려면 (AWS CLI)

- 다음 명령을 실행하여 배포의 의 의 의 상태를 검색합니다. *##ID#* 쿼리할 배포의 ID로 바꿉니다.

  ```
  aws greengrassv2 get-deployment --deployment-id deploymentId
  ```

  응답에는 배포 상태가 포함됩니다.

디바이스 배포 상태 확인

개별 코어 디바이스에 적용되는 작업의 상태를 확인할 수 있습니다. 사물 그룹 배포의 상태를 확인할 수도 있습니다.

코어 장치의 배포 작업 상태를 확인하려면 (AWS CLI)

- 다음 명령을 실행하여 코어 디바이스의 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 *coreDeviceName*바꾸십시오.

  ```
  aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
  ```

  응답에는 코어 장치의 배포 작업 목록이 포함됩니다. 작업`deploymentId` 또는 작업으로 배포 작업을 식별할 수`targetArn` 있습니다. 각 배포 작업에는 코어 디바이스의 상태가 포함됩니다.

사물 그룹 (AWS CLI) 의 배포 상태를 확인하려면

1. 다음 명령을 실행하여 기존 배포의 ID를 검색합니다. *TargetARN#* 대상 사물 그룹의 ARN으로 바꾸십시오.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 다음 단계에서 사용할 수 있도록 해당 응답에서 복사해 둡니다.`deploymentId`

   > **Note**
   >
   > 대상에 대한 최신 배포 이외의 배포를 나열할 수도 있습니다. `--history-filter ALL`인수를 지정하여 대상에 대한 모든 배포를 나열합니다. 그런 다음 상태를 확인하려는 배포의 ID를 복사합니다.

2. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. *### ID# ## ### ID#* 교체합니다.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   해당 응답에는 배포에 대한 정보가 포함됩니다. `iotJobId`응답에서 를 복사하여 다음 단계에서 사용하십시오.

3. 다음 명령을 실행하여 배포를 위한 코어 장치의 작업 실행을 설명합니다. 바꾸기 *iotJobId*및 *coreDeviceThing###* 이전 단계의 작업 ID와 상태를 확인하려는 코어 장치로 바꾸십시오.

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

응답에는 코어 장치의 배포 작업 실행 상태와 상태에 대한 세부 정보가 포함됩니다. detailsMap에는 다음과 같은 정보가 들어 있습니다.

- detailed-deployment-status— 배포 결과 상태를 나타냅니다.
  - SUCCESSFUL— 배포에 성공했습니다.
  - FAILED_NO_STATE_CHANGE— 코어 디바이스에서 배포 적용을 준비하는 동안 배포가 실패했습니다.
  - FAILED_ROLLBACK_NOT_REQUESTED— 배포에 실패했고 배포에서 이전 작업 구성으로 롤백하도록 지정하지 않았으므로 코어 디바이스가 제대로 작동하지 않을 수 있습니다.
  - FAILED_ROLLBACK_COMPLETE— 배포에 실패했으며 코어 디바이스가 이전 작업 구성으로 성공적으로 롤백되었습니다.
  - FAILED_UNABLE_TO_ROLLBACK— 배포에 실패했고 코어 디바이스가 이전 작업 구성으로 롤백하지 못했기 때문에 코어 디바이스가 제대로 작동하지 않을 수 있습니다.

  배포에 실패한 경우deployment-failure-cause 값과 코어 장치의 로그 파일을 확인하여 문제를 식별하십시오. 코어 디바이스의 로그 파일에 액세스하는 방법에 대한 자세한 내용은 단원을 참조하십시오모니터 AWS IoT Greengrass 로그.
- deployment-failure-cause— 작업 실행이 실패한 이유에 대한 추가 세부 정보를 제공하는 오류 메시지입니다.

응답은 다음 예와 비슷합니다.

```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version
 satisfies the requirements. Check whether the version constraints conflict
 and that the component exists in your AWS ## with a version that matches the
 version constraints. If the version constraints conflict, revise deployments
```

```
 to resolve the conflict. Component com.example.HelloWorld version constraints:
 LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
 =1.0.1.",
        "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
      }
    },
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
    "queuedAt": "2022-02-15T14:45:53.098000-08:00",
    "startedAt": "2022-02-15T14:46:05.670000-08:00",
    "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
    "executionNumber": 1,
    "versionNumber": 3
  }
}
```

AWS IoT Greengrass의 로깅 및 모니터링

모니터링은 AWS IoT Greengrass와 사용자 AWS 솔루션의 신뢰성, 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

주제

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 다른 소스에서 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 로그 파일 모니터링을 참조하십시오.

- AWS CloudTrail로그 모니터링 — 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs로 전송하여 실시간으로 모니터링하고, Java로 로그 처리 애플리케이션을 작성하고, 전송 후 로그 파일이 변경되지 않았는지 확인합니다 CloudTrail. 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하십시오.
- Greengrass 시스템 상태 텔레메트리 — Greengrass 코어에서 전송된 텔레메트리 데이터를 수신하려면 구독하십시오. 자세한 설명은 the section called “시스템 상태 원격 측정 데이터 수집” 섹션을 참조하세요.
- 장치 상태 알림 Amazon에서 이벤트를 EventBridge 생성하여 배포 및 구성 요소에 관한 상태 업데이트를 받으십시오. 자세한 설명은 배포 및 구성 요소 상태 알림 받기 섹션을 참조하세요.
- 플릿 상태 서비스 — 플릿 상태 API 작업을 사용하여 코어 디바이스 및 해당 Greengrass 구성 요소의 상태를 확인합니다. AWS IoT Greengrass콘솔에서 플릿 상태 정보를 볼 수도 있습니다. 자세한 내용은 그린그래스 코어 디바이스 상태 확인을(를) 참조하세요.

모니터 AWS IoT Greengrass 로그

AWS IoT Greengrass은(는) 클라우드 서비스와 AWS IoT Greengrass 코어 소프트웨어로 구성됩니다. AWS IoT GreengrassCore 소프트웨어는 Amazon Logs 및 코어 디바이스의 로컬 파일 시스템에 CloudWatch 로그를 쓸 수 있습니다. 코어 디바이스에서 실행되는 Greengrass 구성 요소는 로그 및 로컬 파일 시스템에 CloudWatch 로그를 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.

기본적으로 AWS IoT Greengrass Core 소프트웨어는 로컬 파일 시스템에만 로그를 기록합니다. 파일 시스템 로그를 실시간으로 볼 수 있으므로 개발 및 배포한 Greengrass 구성 요소를 디버깅할 수 있습니다. 또한 로그에 로그를 기록하도록 코어 디바이스를 구성할 수 있으므로 로컬 파일 시스템에 액세스하지 않고도 코어 디바이스 문제를 해결할 수 있습니다. CloudWatch 자세한 설명은 CloudWatch 로그에 로깅을 활성화합니다. 섹션을 참조하세요.

주제

- 파일 시스템 로그에 액세스하십시오.
- 액세스 로그 CloudWatch
- 시스템 서비스 로그에 액세스
- CloudWatch 로그에 로깅을 활성화합니다.
- AWS IoT Greengrass의 로깅 구성
- AWS CloudTrail 로그

파일 시스템 로그에 액세스하십시오.

AWS IoT GreengrassCore 소프트웨어는 코어 디바이스의 */greengrass/v2*/logs 폴더에 로그를 저장합니다. 여기서 */greengrass/v2* 는 AWS IoT Greengrass 루트 폴더의 경로입니다. 로그 폴더의 구조는 다음과 같습니다.

```
/greengrass/v2
### logs
    ### greengrass.log
    ### greengrass_2021_09_14_15_0.log
    ### ComponentName.log
    ### ComponentName_2021_09_14_15_0.log
    ### main.log
```

- `greengrass.log`— AWS IoT Greengrass Core 소프트웨어 로그 파일. 이 로그 파일을 사용하면 구성 요소 및 배포에 대한 실시간 정보를 볼 수 있습니다. 이 로그 파일에는 Core 소프트웨어의 AWS IoT Greengrass 핵심인 Greengrass nucleus 및 로그 관리자 및 비밀 관리자와 같은 플러그인 구성 요소에 대한 로그가 포함되어 있습니다.
- *ComponentName*`.log`— Greengrass 구성 요소 로그 파일 구성 요소 로그 파일을 사용하여 코어 장치에서 실행되는 Greengrass 구성 요소에 대한 실시간 정보를 볼 수 있습니다. 일반 구성 요소 및 Lambda 구성 요소는 표준 출력 (stdout) 및 표준 오류 (stderr) 를 이러한 로그 파일에 기록합니다.
- `main.log`— 구성 요소 수명 주기를 처리하는 서비스의 로그 파일입니다. `main` 이 로그 파일은 항상 비어 있습니다.

플러그인, 제네릭 및 Lambda 구성 요소 간의 차이점에 대한 자세한 내용은 을 참조하십시오. 구성 요소 유형

파일 시스템 로그 사용 시 다음 사항을 고려하십시오.

- 루트 사용자 권한

  파일 시스템에서 AWS IoT Greengrass 로그를 읽을 수 있는 루트 권한이 있어야 합니다.

- 로그 파일 로테이션

  AWS IoT GreengrassCore 소프트웨어는 1시간마다 또는 파일 크기 제한을 초과할 때마다 로그 파일을 순환합니다. 순환된 로그 파일의 파일 이름에는 타임스탬프가 포함됩니다. 예를 들어 회전된 AWS IoT Greengrass Core 소프트웨어 로그 파일의 이름을 지정할 수 있습니

다. `greengrass_2021_09_14_15_0.log` 기본 파일 크기 제한은 1,024KB (1MB) 입니다. Greengrass nucleus 컴포넌트에서 파일 크기 제한을 구성할 수 있습니다.

- 로그 파일 삭제

  AWS IoT GreengrassCore 소프트웨어는 Core 소프트웨어 로그 파일 또는 Greengrass 구성 요소 로그 파일 (회전된 로그 파일 포함) 의 AWS IoT Greengrass 크기가 디스크 공간 제한을 초과하는 경우 이전 로그 파일을 정리합니다. AWS IoT GreengrassCore 소프트웨어 로그와 각 구성 요소 로그의 기본 디스크 공간 제한은 10,240KB (10MB) 입니다. Greengrass nucleus 구성 요소 또는 로그 관리자 구성 요소에서 AWS IoT Greengrass Core 소프트웨어 로그 디스크 공간 제한을 구성할 수 있습니다. 로그 관리자 구성 요소에서 각 구성 요소의 로그 디스크 공간 제한을 구성할 수 있습니다.

AWS IoT GreengrassCore 소프트웨어 로그 파일을 보려면

- 다음 명령을 실행하여 로그 파일을 실시간으로 확인합니다. AWS IoT Greengrass루트 폴더 */greengrass/v2* 경로로 바꿉니다.

  Linux or Unix

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows Command Prompt (CMD)

  ```
  type C:\greengrass\v2\logs\com.example.HelloWorld.log
  ```

  이 `type` 명령은 파일 내용을 터미널에 기록합니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰하십시오.

  PowerShell

  ```
  gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

구성 요소의 로그 파일을 보려면

- 다음 명령을 실행하여 로그 파일을 실시간으로 확인합니다. */greengrass/v2*또는 *C:\greengrass\v2* 를 AWS IoT Greengrass 루트 폴더 경로로 바꾸고 *com.example# #####. HelloWorld*구성 요소 이름과 함께.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

또한 Greengrass CLI의 `logs` 명령을 사용하여 코어 디바이스의 그린그래스 로그를 분석할 수 있습니다. `logs`명령을 사용하려면 JSON 형식 로그 파일을 출력하도록 Greengrass 핵을 구성해야 합니다. 자세한 내용은 그린그래스 커맨드 라인 인터페이스 및 로그 섹션을 참조하세요.

액세스 로그 CloudWatch

로그 관리자 구성 요소를 배포하여 CloudWatch 로그에 기록하도록 코어 장치를 구성할 수 있습니다. 자세한 설명은 CloudWatch 로그에 로깅을 활성화합니다. 섹션을 참조하세요. 그러면 Amazon CloudWatch 콘솔의 로그 페이지에서 또는 Logs API를 사용하여 CloudWatch 로그를 볼 수 있습니다.

로그 그룹 이름

```
/aws/greengrass/componentType/region/componentName
```

로그 그룹 이름은 다음 변수를 사용합니다.

- `componentType`— 구성 요소의 유형으로, 다음 중 하나일 수 있습니다.
  - `GreengrassSystemComponent`— 이 로그 그룹에는 Greengrass 핵과 동일한 JVM에서 실행되는 핵 및 플러그인 구성 요소에 대한 로그가 포함됩니다. 구성 요소는 Greengrass 핵의 일부입니다.
  - `UserComponent`— 이 로그 그룹에는 일반 구성 요소, Lambda 구성 요소 및 디바이스의 기타 애플리케이션에 대한 로그가 포함됩니다. 구성 요소는 Greengrass 핵의 일부가 아닙니다.

  자세한 설명은 구성 요소 유형 섹션을 참조하세요.
- `region`— 코어 디바이스가 사용하는 AWS 지역.
- `componentName`— 구성 요소의 이름. 시스템 로그의 경우 이 값은 입니다`System`.

로그 스트림 이름

```
/date/thing/thingName
```

로그 스트림 이름은 다음 변수를 사용합니다.

- `date`— 로그 날짜 (예:`2020/12/15`. 로그 관리자 구성 요소는 `yyyy/MM/dd` 형식을 사용합니다.
- `thingName`— 코어 디바이스의 이름.

> **Note**
>
> 사물 이름에 콜론 (`:`) 이 포함된 경우 로그 관리자는 콜론을 더하기 () 로 바꿉니다. +

로그 관리자 구성 요소를 사용하여 로그에 기록할 때는 다음 고려 사항이 적용됩니다. CloudWatch

- 로그 지연

> **Note**
>
> 순환 및 활성 로그 파일의 로그 지연을 줄이는 로그 관리자 버전 2.3.0으로 업그레이드하는 것이 좋습니다. 로그 관리자 2.3.0으로 업그레이드할 때는 Greengrass nucleus 2.9.1으로도 업그레이드하는 것이 좋습니다.

로그 관리자 구성 요소 버전 2.2.8 (이하) 은 순환된 로그 파일의 로그만 처리하고 업로드합니다. 기본적으로 AWS IoT Greengrass Core 소프트웨어는 1시간마다 또는 1,024KB가 된 이후에 로그 파일을 교체합니다. 따라서 로그 관리자 구성 요소는 AWS IoT Greengrass Core 소프트웨어 또는 Greengrass 구성 요소가 1,024KB 이상의 로그를 기록한 후에만 로그를 업로드합니다. 로그 파일 크기를 낮게 제한하여 로그 파일이 더 자주 회전하도록 구성할 수 있습니다. 이로 인해 로그 관리자 구성 요소가 로그를 로그에 CloudWatch 더 자주 업로드합니다.

로그 관리자 구성 요소 버전 2.3.0 (이상) 은 모든 로그를 처리하고 업로드합니다. 새 로그를 작성하면 로그 관리자 버전 2.3.0 (이상) 은 활성 로그 파일이 회전될 때까지 기다리지 않고 해당 활성 로그 파일을 처리하여 직접 업로드합니다. 즉, 5분 이내에 새 로그를 볼 수 있습니다.

로그 관리자 구성 요소는 새 로그를 주기적으로 업로드합니다. 기본적으로 로그 관리자 구성 요소는 5분마다 새 로그를 업로드합니다. 업로드 간격을 낮게 구성하여 로그 관리자 구성 요소가 로그를 로

그에 더 자주 업로드하도록 CloudWatch 구성할 수 있습니다. `periodicUploadIntervalSec` 이 주기적 간격을 구성하는 방법에 대한 자세한 내용은 구성을 참조하십시오.

동일한 Greengrass 파일 시스템에서 거의 실시간으로 로그를 업로드할 수 있습니다. 로그를 실시간으로 관찰해야 하는 경우 파일 시스템 로그를 사용하는 것이 좋습니다.

> **Note**
>
> 다른 파일 시스템을 사용하여 로그를 기록하는 경우 로그 관리자는 로그 관리자 구성 요소 버전 2.2.8 및 이전 버전의 동작으로 되돌아갑니다. 파일 시스템 로그에 액세스하는 방법에 대한 자세한 내용은 파일 시스템 로그 액세스를 참조하십시오.

- 클럭 스큐

  로그 관리자 구성요소는 표준 서명 버전 4 서명 프로세스를 사용하여 CloudWatch 로그에 대한 API 요청을 생성합니다. 코어 디바이스의 시스템 시간이 15분 이상 동기화되지 않으면 CloudWatch Logs는 요청을 거부합니다. 자세한 정보는 AWS 일반 참조의 서명 버전 4 서명 프로세스를 참조하십시오.

시스템 서비스 로그에 액세스

AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 구성하면 시스템 서비스 로그를 보고 소프트웨어 시작 실패와 같은 문제를 해결할 수 있습니다.

시스템 서비스 로그 (CLI) 를 보려면

1. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 시스템 서비스 로그를 확인합니다.

   Linux or Unix (systemd)

   ```
   sudo journalctl -u greengrass.service
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\greengrass.wrapper.log
   ```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.wrapper.log
```

2. Windows 장치에서 AWS IoT Greengrass Core 소프트웨어는 시스템 서비스 오류에 대한 별도의 로그 파일을 생성합니다. 다음 명령을 실행하여 시스템 서비스 오류 로그를 확인합니다.

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\greengrass.err.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.err.log
   ```

Windows 장치에서는 이벤트 뷰어 애플리케이션을 사용하여 시스템 서비스 로그를 볼 수도 있습니다.

Windows 서비스 로그를 보려면 (이벤트 뷰어)

1. 이벤트 뷰어 애플리케이션을 엽니다.
2. Windows 로그를 선택하여 확장합니다.
3. 응용 프로그램 서비스 로그를 보려면 [응용 프로그램] 을 선택합니다.
4. 소스가 있는 이벤트 로그를 찾아 엽니다 greengrass.

CloudWatch 로그에 로깅을 활성화합니다.

로그 관리자 구성 요소를 배포하여 로그에 로그를 기록하도록 코어 디바이스를 구성할 수 CloudWatch 있습니다. CloudWatch Logs for AWS IoT Greengrass Core 소프트웨어 로그를 활성화하고 특정 Greengrass 구성 요소에 대한 CloudWatch 로그를 활성화할 수 있습니다.

> **Note**
>
> Greengrass 코어 디바이스의 토큰 교환 역할은 다음 예시 IAM 정책에 나와 있는 것처럼 코어 디바이스가 CloudWatch 로그에 쓸 수 있도록 허용해야 합니다. 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치한 경우 코어 기기는 이러한 권한을 가집니다.
>
> ```
> {
> ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

코어 소프트웨어 로그를 로그에 기록하도록 AWS IoT Greengrass 코어 디바이스를 구성하려면 CloudWatch 구성 `aws.greengrass.LogManager` 요소에 `true` 대해 로 설정하는 구성 업데이트를 지정하는 uploadToCloudWatch배포를 생성하십시오. AWS IoT Greengrass 핵심 소프트웨어 로그에는 Greengrass 핵 및 플러그인 구성 요소에 대한 로그가 포함됩니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Greengrass 구성 요소의 로그를 Logs에 CloudWatch 기록하도록 코어 디바이스를 구성하려면 구성 요소 로깅 구성 목록에 구성 요소를 추가하는 구성 업데이트를 지정하는 배포를 생성하십시오. 이 목록에 구성 요소를 추가하면 로그 관리자 구성 요소가 로그를 로그에 CloudWatch 기록합니다. 구성 요소 로그에는 일반 구성 요소 및 Lambda 구성 요소에 대한 로그가 포함됩니다.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
```

```
  }
}
```

로그 관리자 구성 요소를 배포할 때 디스크 공간 제한과 코어 디바이스가 로그 파일을 Logs에 기록한 후 삭제할지 여부도 구성할 수 있습니다. CloudWatch 자세한 설명은 AWS IoT Greengrass의 로깅 구성 섹션을 참조하세요.

AWS IoT Greengrass의 로깅 구성

다음 옵션을 구성하여 Greengrass 코어 장치에 대한 로깅을 사용자 지정할 수 있습니다. 이러한 옵션을 구성하려면 Greengrass nucleus 또는 log manager 구성 요소에 대한 구성 업데이트를 지정하는 배포를 생성하십시오.

- 로그에 로그 쓰기 CloudWatch

  코어 디바이스의 문제를 원격으로 해결하려면 코어 소프트웨어와 구성 요소 로그를 로그에 기록하도록 AWS IoT Greengrass 코어 디바이스를 구성할 수 있습니다. CloudWatch 이렇게 하려면 로그 관리자 구성 요소를 배포하고 구성해야 합니다. 자세한 설명은 CloudWatch 로그에 로깅을 활성화합니다. 섹션을 참조하세요.

- 업로드된 로그 파일 삭제

  디스크 공간 사용량을 줄이려면 로그 파일을 로그에 기록한 후 로그 파일을 삭제하도록 코어 디바이스를 구성할 수 있습니다. CloudWatch 자세한 내용은 AWS IoT GreengrassCore 소프트웨어 로그 및 구성 요소 로그에 지정할 수 있는 로그 관리자 구성 요소의 `deleteLogFileAfterCloudUpload` 매개 변수를 참조하십시오.

- 로그 디스크 공간 제한

  디스크 공간 사용을 제한하려면 코어 장치에서 회전된 로그 파일을 포함하여 각 로그의 최대 디스크 공간을 구성할 수 있습니다. 예를 들어, 파일을 합친 최대 디스크 `greengrass.log` 공간과 `greengrass.log` 회전된 파일을 구성할 수 있습니다. 자세한 내용은 Greengrass nucleus 구성 요소의 `logging.totalLogsSizeKB` 매개 변수 및 AWS IoT GreengrassCore 소프트웨어 로그 및 구성 요소 로그에 대해 지정할 수 있는 로그 관리자 구성 요소의 `diskSpaceLimit` 매개 변수를 참조하십시오.

- 로그 파일 크기 제한

  각 로그 파일의 최대 파일 크기를 구성할 수 있습니다. 로그 파일이 이 파일 크기 제한을 초과하면 AWS IoT Greengrass Core 소프트웨어에서 새 로그 파일을 생성합니다. 로그 관리자 구성 요소 버전 2.28 (이하) 은 회전된 로그 파일만 CloudWatch 로그에 기록하므로 더 낮은 파일 크기 제한을 지

정하여 로그에 CloudWatch 로그를 더 자주 쓸 수 있습니다. 로그 관리자 구성 요소 버전 2.3.0 (이상)은 순환될 때까지 기다리지 않고 모든 로그를 처리하고 업로드합니다. 자세한 내용은 Greengrass nucleus 구성 요소의 로그 파일 크기 제한 매개 변수 () 를 참조하십시오. `logging.fileSizeKB`

- 최소 로그 수준

  Greengrass nucleus 구성 요소가 파일 시스템 로그에 기록하는 최소 로그 수준을 구성할 수 있습니다. 예를 들어 문제 해결에 도움이 되는 `DEBUG` 레벨 로그를 지정하거나 코어 디바이스에서 생성하는 로그의 양을 줄이기 위해 `ERROR` 레벨 로그를 지정할 수 있습니다. 자세한 내용은 Greengrass nucleus 컴포넌트의 로그 레벨 파라미터 () 를 참조하십시오. `logging.level`

  또한 로그 관리자 구성 요소가 로그에 기록하는 최소 로그 수준을 구성할 수 있습니다. CloudWatch 예를 들어, 더 높은 로그 수준을 지정하여 로깅 비용을 줄일 수 있습니다. 자세한 내용은 AWS IoT Greengrass코어 소프트웨어 로그 및 구성 요소 로그에 지정할 수 있는 로그 관리자 구성 요소의 `minimumLogLevel` 매개 변수를 참조하십시오.

- 로그에 기록할 CloudWatch 로그를 확인하는 간격

  로그 관리자 구성 요소가 로그에 로그를 기록하는 빈도를 늘리거나 줄이려면 CloudWatch 기록할 새 로그 파일이 있는지 확인하는 간격을 구성할 수 있습니다. 예를 들어, 기본 5분 간격보다 빨리 로그에서 CloudWatch 로그를 볼 수 있는 간격을 더 짧게 지정할 수 있습니다. 로그 관리자 구성 요소가 로그 파일을 더 적은 수의 요청으로 일괄 처리하므로 비용을 줄이기 위해 간격을 더 높게 지정할 수 있습니다. 자세한 내용은 로그 관리자 구성 요소의 업로드 간격 매개 변수 (`periodicUploadIntervalSec`) 를 참조하십시오.

- 로그 형식

  AWS IoT GreengrassCore 소프트웨어에서 로그를 텍스트 또는 JSON 형식으로 작성할지 선택할 수 있습니다. 로그를 읽는 경우 텍스트 형식을 선택하고, 애플리케이션을 사용하여 로그를 읽거나 분석하는 경우 JSON 형식을 선택합니다. 자세한 내용은 Greengrass nucleus 구성 요소의 로그 형식 매개 변수 () 를 참조하십시오. `logging.format`

- 로컬 파일 시스템 로그 폴더

  로그 폴더를 코어 디바이스의 다른 폴더로 변경할 수 있습니다. */greengrass/v2*/logs 자세한 내용은 Greengrass nucleus 컴포넌트의 출력 디렉토리 파라미터 () 를 참조하십시오. `logging.outputDirectory`

AWS CloudTrail 로그

AWS IoT Greengrass사용자AWS CloudTrail, 역할 또는 역할 내에서 수행한 작업에 대한 기록을 제공하는 서비스와 통합됩니다. AWS 서비스 AWS IoT Greengrass 자세한 내용은 를 AWS IoT Greengrass V2 사용하여 API 호출을 기록합니다. AWS CloudTrail을(를) 참조하세요.

를 AWS IoT Greengrass V2 사용하여 API 호출을 기록합니다. AWS CloudTrail

AWS IoT Greengrass V2 에서 AWS IoT Greengrass Version 2사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업의 기록을 제공하는 AWS 서비스와 통합됩니다. CloudTrail 모든 API 호출을 AWS IoT Greengrass 이벤트로 캡처합니다. 캡처되는 호출에는 AWS IoT Greengrass 콘솔에서의 호출과 AWS IoT Greengrass API 작업에 대한 코드 호출이 포함됩니다.

트레일을 생성하면 에 대한 이벤트를 포함하여 S3 버킷에 CloudTrail 이벤트를 지속적으로 전송할 수 AWS IoT Greengrass있습니다. 트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근 이벤트를 계속 볼 수 있습니다. 에서 수집한 CloudTrail 정보를 사용하여 요청을 받은 사람 AWS IoT Greengrass, 요청한 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

에 대한 CloudTrail 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

AWS IoT Greengrass V2 에 대한 정보 CloudTrail

CloudTrail 계정을 만들 AWS 계정 때 활성화됩니다. 에서 AWS IoT Greengrass활동이 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.

에 대한 이벤트를 포함하여 내 이벤트의 진행 중인 기록을 보려면 AWS IoT Greengrass트레일을 생성하십시오 AWS 계정. 트레일을 사용하면 CloudTrail S3 버킷에 로그 파일을 전송할 수 있습니다. 기본적으로 콘솔에서 트레일을 생성하면 트레일이 모든 AWS 리전 s에 적용됩니다. 트레일은 AWS 파티션에 있는 모든 지역의 이벤트를 기록하고 지정한 S3 버킷으로 로그 파일을 전달합니다. 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 V2 작업은 AWS IoT Greengrass V2 API Reference에 의해 CloudTrail 기록되고 문서화됩니다. 예를 들어`CreateComponentVersion`, 에 대한 호출 `CreateDeployment` 및 `CancelDeployment` 작업은 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.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부.
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증 정보를 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 CloudTrail userIdentity 요소를 참조하십시오.

AWS IoT Greengrass 의 데이터 이벤트 CloudTrail

데이터 이벤트는 리소스에서 또는 리소스에서 수행되는 리소스 작업에 대한 정보를 제공합니다 (예: 구성 요소 버전 가져오기 또는 배포 구성). 이를 데이터 영역 작업이라고도 합니다. 데이터 이벤트가 대량 활동인 경우도 있습니다. 기본적으로 데이터 이벤트를 기록하지 CloudTrail 않습니다. CloudTrail 이벤트 기록에는 데이터 이벤트가 기록되지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하십시오.

CloudTrail 콘솔 또는 CloudTrail API 작업을 사용하여 AWS IoT Greengrass 리소스 유형에 대한 데이터 이벤트를 기록할 수 있습니다. AWS CLI이 섹션의 표에는 사용 가능한 리소스 유형이 나와 AWS IoT Greengrass있습니다.

- CloudTrail 콘솔을 사용하여 데이터 이벤트를 기록하려면 트레일 또는 이벤트 데이터 저장소를 생성하여 데이터 이벤트를 기록하거나 기존 트레일 또는 이벤트 데이터 저장소를 업데이트하여 데이터 이벤트를 기록하십시오.

  1. 데이터 이벤트를 선택하여 데이터 이벤트를 기록합니다.
  2. 데이터 이벤트 유형 목록에서 데이터 이벤트를 기록하려는 리소스 유형을 선택합니다.
  3. 사용하려는 로그 선택기 템플릿을 선택합니다. 리소스 유형에 대한 모든 데이터 이벤트를 기록하거나, 모든 이벤트를 기록하거나, 모든 `readOnly` 이벤트를 기록하거나

`readOnly`, `eventName` 및 `resources.ARN` 필드를 기준으로 필터링할 사용자 지정 로그 선택기 템플릿을 만들 수 있습니다. `writeOnly`

- 를 사용하여 데이터 이벤트를 기록하려면 필드를 리소스 유형 값과 같게 설정하고 `eventCategory` 필드를 리소스 유형 값과 같게 `Data` 설정하도록 `--advanced-event-selectors` 매개 변수를 구성하십시오 (표 참조). AWS CLI`resources.type` 조건을 추가하여 `readOnly`, `eventName` 및 `resources.ARN` 필드의 값을 기준으로 필터링할 수 있습니다.
  - 데이터 이벤트를 기록하도록 트레일을 구성하려면 put-event-selectors명령을 실행합니다. 자세한 내용은 를 사용한 트레일의 데이터 이벤트 로깅을 AWS CLI 참조하십시오.
  - 데이터 이벤트를 기록하도록 이벤트 데이터 저장소를 구성하려면 create-event-data-store명령을 실행하여 데이터 이벤트를 기록할 새 이벤트 데이터 저장소를 만들거나 update-event-data-store명령을 실행하여 기존 이벤트 데이터 저장소를 업데이트하십시오. 자세한 내용은 를 사용한 이벤트 데이터 저장소의 데이터 이벤트 로깅을 참조하십시오 AWS CLI.

다음 표에는 AWS IoT Greengrass 리소스 유형이 나열되어 있습니다. 데이터 이벤트 유형 (콘솔) 열에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 수 있는 값이 표시됩니다. resources.type 값 열에는 또는 `resources.type` API를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 값이 표시됩니다. AWS CLI CloudTrail 데이터 API 로깅 대상 CloudTrail 열에는 해당 리소스 유형에 대해 로깅된 API 호출이 표시됩니다. CloudTrail

데이터 이벤트 유형(콘솔)	resources.type 값	로깅된 데이터 API CloudTrail
IoT 인증서	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
IoT 그린그래스 컴포넌트 버전	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates
IoT 그린그래스 배포	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
IoT 관련	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

> **Note**
>
> Greengrass는 액세스 거부 이벤트를 기록하지 않습니다.

`eventName`, `readOnly` 및 `resources.ARN` 필드를 필터링하여 중요한 이벤트만 로깅하도록 고급 이벤트 선택기를 구성할 수 있습니다.

필터를 추가하여 특정 데이터 API를 포함하거나 제외하세요. `eventName`

필드에 대한 자세한 내용은 AdvancedFieldSelector 섹션을 참조하세요.

다음 예는 를 사용하여 고급 선택기를 구성하는 방법을 보여줍니다. AWS CLI## 정보를 원하는 정보로 *`TrailName`*바꾸십시오.

Example — IoT 사물에 대한 로그 데이터 이벤트

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log all thing data events",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
            ]
        }
]'
```

Example — 특정 IoT 사물 API를 기반으로 필터링

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
                { "Field": "eventName", "Equals": ["PutCertificateAuthorities"] }
            ]
    }
```

```
]'
```

Example — 모든 Greengrass 데이터 이벤트 기록

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
        "Name": "Log all certificate data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::IoT::Certificate"
                ]
            }
        ]
    },
    {
        "Name": "Log all component version data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::GreengrassV2::ComponentVersion"
                ]
            }
        ]
    },
    {
        "Name": "Log all deployment version",
```

```
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::Deployment”
                ]
            }
        ]
    },
    {
        “Name”: “Log all thing data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Thing”
                ]
            }
        ]
    }
]’
```

AWS IoT Greengrass 관리 이벤트는 다음과 같습니다. CloudTrail

관리 이벤트는 AWS 계정의 리소스에서 수행되는 관리 작업에 대한 정보를 제공합니다. 이를 제어 영역 작업이라고도 합니다. 기본적으로 관리 이벤트를 CloudTrail 기록합니다.

AWS IoT Greengrass 모든 AWS IoT Greengrass 컨트롤 플레인 작업을 관리 이벤트로 기록합니다. AWS IoT Greengrass 로그되는 AWS IoT Greengrass 컨트롤 플레인 작업 목록은 AWS IoT Greengrass API 참조 CloudTrail, 버전 2를 참조하십시오.

AWS IoT Greengrass V2 로그 파일 항목 이해

트레일은 지정한 S3 버킷에 이벤트를 로그 파일로 전송할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스로부터 단일 요청을 나타냅니다. 여기에는 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보가 포함됩니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 추적이 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 `CreateDeployment` 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2021-01-06T02:38:05Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
```

```
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}
```


AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집

시스템 상태 원격 측정 데이터는 Greengrass 코어 디바이스의 중요 작업 성능을 모니터링하는 데 도움이 되는 진단 데이터입니다. 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.

다음 방법을 사용하여 Greengrass 코어 장치에서 원격 측정 데이터를 수집할 수 있습니다.

- Nucleus 텔레메트리 이미터 구성 요소 —Greengrass 코어 디바이스의 Nucleus 텔레메트리 이미터 구성 요소 () 는 기본적으로 원격 측정 데이터를 `aws.greengrass.telemetry.NucleusEmitter` 주제에 게시합니다. `$local/greengrass/telemetry` 디바이스의 클라우드 연결이 제한된 경우에도 이 주제에 게시된 데이터를 사용하여 코어 디바이스에서 로컬로 작동할 수 있습니다. 원하는 AWS IoT Core MQTT 주제에 원격 분석 데이터를 게시하도록 구성 요소를 구성할 수도 있습니다.

  텔레메트리 데이터를 게시하려면 핵 이미터 구성 요소를 코어 장치에 배포해야 합니다. 원격 분석 데이터를 로컬 주제에 게시하는 데는 비용이 들지 않습니다. 하지만 MQTT 주제를 사용하여 에 데이터를 게시하는 경우 요금이 AWS 클라우드 부과됩니다. AWS IoT Core

AWS IoT GreengrassInfluxDB 및 Grafana를 사용하여 코어 장치에서 로컬로 원격 측정 데이터를 분석하고 시각화하는 데 도움이 되는 여러 커뮤니티 구성 요소를 제공합니다. 이러한 구성 요소는 핵 방사체 구성 요소의 원격 측정 데이터를 사용합니다. 자세한 내용은 InfluxDB 게시자 구성 요소에 대한 README를 참조하십시오.

- 텔레메트리 에이전트 —Greengrass 코어 디바이스의 텔레메트리 에이전트는 고객 개입 없이 로컬 텔레메트리 데이터를 수집하여 Amazon에 게시합니다. EventBridge 코어 디바이스는 최선의 노력을 다해 원격 분석 데이터를 게시합니다. EventBridge 예를 들어, 코어 디바이스는 오프라인 상태에서 원격 측정 데이터를 제공하지 못할 수 있습니다.

  원격 분석 에이전트 기능은 모든 Greengrass 코어 디바이스에서 기본적으로 활성화됩니다. Greengrass 코어 디바이스를 설정하는 즉시 자동으로 데이터를 수신하기 시작합니다. 데이터 링크 비용을 제외하고 코어 디바이스에서 코어 디바이스로의 데이터 전송에는 요금이 AWS IoT Core 부과되지 않습니다. 이는 에이전트가 AWS 예약된 주제에 게시하기 때문입니다. 하지만 사용 사례에 따라 데이터를 받거나 처리할 때 비용이 발생할 수 있습니다.

  > **Note**
  >
  > EventBridge Amazon은 Greengrass 코어 디바이스와 같은 다양한 소스의 데이터에 애플리케이션을 연결하는 데 사용할 수 있는 이벤트 버스 서비스입니다. 자세한 내용은 Amazon이란 무엇입니까 EventBridge? 를 참조하십시오. Amazon EventBridge 사용 설명서에서 확인할 수 있습니다.

AWS IoT GreengrassCore 소프트웨어가 제대로 작동하도록 하기 위해 는 데이터를 개발 및 품질 개선 목적으로 AWS IoT Greengrass 사용합니다. 또한 이 기능은 새롭고 향상된 엣지 기능을 알려주는 데도 도움이 됩니다. AWS IoT Greengrass텔레메트리 데이터를 최대 7일 동안 보존합니다.

이 섹션에서는 원격 분석 에이전트를 구성하고 사용하는 방법에 대해 설명합니다. nucleus 텔레메트리 이미터 구성 요소 구성에 대한 자세한 내용은 을 참조하십시오. 뉴클리어스 텔레메트리 이미터

주제

- 원격 측정 지표
- 원격 분석 에이전트 설정을 구성합니다.
- 에서 텔레메트리 데이터를 구독하세요. EventBridge

원격 측정 지표

다음 표에는 원격 분석 에이전트가 게시한 메트릭이 설명되어 있습니다.

명칭	설명	
시스템		
SystemMemUsage	운영 체제를 포함하여 Greengrass 코어 디바이스의 모든 애플리케이션에서 현재 사용 중인 메모리의 양입니다.	
CpuUsage	운영 체제를 포함하여 Greengrass 코어 디바이스의 모든 애플리케이션에서 현재 사용 중인 CPU의 양입니다.	
TotalNumberOfFDs	Greengrass 코어 디바이스의 운영 체제에 저장된 파일 디스크립터 수입니다. 하나의 파일 디스크립터는 열려 있는 파일 하나를 고유하게 식별합니다.	
그린그래스 핵		
NumberOfComponentsRunning	Greengrass 코어 디바이스에서 실행 중인 구성 요소 수입니다.	
NumberOfComponentsErrored	Greengrass 코어 디바이스에서 오류 상태에 있는 구성 요소의 수입니다.	
NumberOfComponentsInstalled	Greengrass 코어 디바이스에 설치된 구성 요소 수입니다.	

명칭	설명	
NumberOfComponents Starting	Greengrass 코어 디바이스에서 시작되는 구성 요소 수입니다.	
NumberOfComponents New	Greengrass 코어 디바이스에 새로 추가된 구성 요소의 수.	
NumberOfComponents Stopping	Greengrass 코어 디바이스에서 중지되는 구성 요소의 수입니다.	
NumberOfComponents Finished	Greengrass 코어 디바이스에서 완성된 구성 요소의 수입니다.	
NumberOfComponents Broken	Greengrass 코어 디바이스에서 고장난 구성 요소의 수입니다.	
NumberOfComponents Stateless	Greengrass 코어 디바이스에서 스테이트리스 상태인 구성 요소의 수입니다.	
클라이언트 장치 인증 - 이 기능을 사용하려면 v2.4.0 이상의 클라이언트 장치 인증 구성 요소가 필요합니다.		
VerifyClientDeviceIdentity.Success	클라이언트 장치 ID가 성공했는지 확인한 횟수입니다.	
VerifyClientDeviceIdentity.Failure	클라이언트 장치 ID 확인에 실패한 횟수입니다.	
AuthorizeClientDeviceActions.Success	클라이언트 장치가 요청된 작업을 완료할 수 있는 권한을 부여받은 횟수.	

명칭	설명
AuthorizeClientDeviceActions.Failure	클라이언트 장치에 요청된 작업을 완료할 권한이 없는 횟수.
GetClientDeviceAuthToken.Success	클라이언트 장치가 성공적으로 인증된 횟수.
GetClientDeviceAuthToken.Failure	클라이언트 장치를 인증할 수 없는 횟수입니다.
SubscribeToCertificateUpdates.Success	인증서 업데이트를 성공적으로 구독한 횟수.
SubscribeToCertificateUpdates.Failure	인증서 업데이트 구독 시도에 실패한 횟수.
ServiceError	클라이언트 장치 인증에서 처리되지 않은 내부 오류 수입니다.
스트림 매니저 — 이 기능을 사용하려면 v2.7.0 이상의 Greengrass 핵 구성 요소가 필요합니다.	
BytesAppended	스트림 관리자에 추가된 데이터의 바이트 수입니다.
BytesUploadedToIoTAnalytics	스트림 관리자가 AWS IoT Analytics에서 채널로 내보내는 데이터의 바이트 수입니다.
BytesUploadedToKinesis	스트림 관리자가 Amazon Kinesis Data Streams의 스트림으로 내보내는 데이터의 바이트 수입니다.

명칭	설명	
`BytesUploadedToIoTSiteWise`	스트림 관리자가 AWS IoT SiteWise에서 에셋 속성으로 내보내는 데이터의 바이트 수입니다.	
`BytesUploadedToS3`	스트림 관리자가 Amazon S3의 객체로 내보내는 데이터의 바이트 수입니다.	

원격 분석 에이전트 설정을 구성합니다.

원격 분석 에이전트는 다음과 같은 기본 설정을 사용합니다.

- 원격 측정 에이전트는 1시간마다 원격 측정 데이터를 집계합니다.
- 원격 측정 에이전트는 24시간마다 원격 측정 메시지를 게시합니다.

원격 분석 에이전트는 QoS (서비스 품질) 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 코어로의 데이터 전송에는 요금이 부과되지 않습니다. AWS IoT Core 이는 에이전트가 AWS 예약된 주제에 게시하기 때문입니다. 하지만 사용 사례에 따라 데이터를 받거나 처리할 때 비용이 발생할 수 있습니다.

각 Greengrass 코어 장치에 대해 원격 분석 에이전트 기능을 활성화하거나 비활성화할 수 있습니다. 코어 디바이스가 데이터를 집계하고 게시하는 간격을 구성할 수도 있습니다. 텔레메트리를 구성하려면 Greengrass nucleus 구성 요소를 배포할 때 텔레메트리 구성 매개변수를 사용자 지정하십시오.

에서 텔레메트리 데이터를 구독하세요. EventBridge

Amazon에서 Greengrass 코어 디바이스의 원격 분석 에이전트에서 게시된 원격 분석 데이터를 처리하는 방법을 EventBridge 정의하는 규칙을 생성할 수 있습니다. 데이터를 EventBridge 수신하면 규칙에 정의된 대상 작업을 호출합니다. 예를 들어 알림을 보내거나, 이벤트 정보를 저장하거나, 교정 작업을 수행하거나, 기타 이벤트를 간접적으로 호출하는 이벤트 규칙을 생성할 수 있습니다.

텔레메트리 이벤트

텔레메트리 이벤트는 다음 형식을 사용합니다.


```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-11-30T20:45:53Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "ThingName": "MyGreengrassCore",
    "Schema": "2020-07-30",
    "ADP": [
      {
        "TS": 1602186483234,
        "NS": "SystemMetrics",
        "M": [
          {
            "N": "TotalNumberOfFDs",
            "Sum": 6447.0,
            "U": "Count"
          },
          {
            "N": "CpuUsage",
            "Sum": 15.458333333333332,
            "U": "Percent"
          },
          {
            "N": "SystemMemUsage",
            "Sum": 10201.0,
            "U": "Megabytes"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "GreengrassComponents",
        "M": [
          {
            "N": "NumberOfComponentsStopping",
```

```
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStarting",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsBroken",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsFinished",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsInstalled",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsRunning",
            "Sum": 7.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsNew",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsErrored",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStateless",
            "Sum": 0.0,
            "U": "Count"
          }
        ]
```

```
    },
    {
      "TS": 1602186483234,
      "NS": "aws.greengrass.ClientDeviceAuth",
      "M": [
        {
          "N": "VerifyClientDeviceIdentity.Success",
          "Sum": 3.0,
          "U": "Count"
        },
        {
          "N": "VerifyClientDeviceIdentity.Failure",
          "Sum": 1.0,
          "U": "Count"
        },
        {
          "N": "AuthorizeClientDeviceActions.Success",
          "Sum": 20.0,
          "U": "Count"
        },
        {
          "N": "AuthorizeClientDeviceActions.Failure",
          "Sum": 5.0,
          "U": "Count"
        },
        {
          "N": "GetClientDeviceAuthToken.Success",
          "Sum": 5.0,
          "U": "Count"
        },
        {
          "N": "GetClientDeviceAuthToken.Failure",
          "Sum": 2.0,
          "U": "Count"
        },
        {
          "N": "SubscribeToCertificateUpdates.Success",
          "Sum": 10.0,
          "U": "Count"
        },
        {
          "N": "SubscribeToCertificateUpdates.Failure",
          "Sum": 1.0,
          "U": "Count"
```

```
          },
          {
            "N": "ServiceError",
            "Sum": 3.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.StreamManager",
        "M": [
          {
            "N": "BytesAppended",
            "Sum": 157745524.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTAnalytics",
            "Sum": 149012.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToKinesis",
            "Sum": 12192.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTSiteWise",
            "Sum": 13321.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToS3",
            "Sum": 12213.0,
            "U": "Bytes"
          }
        ]
      }
    ]
  }
}
```

ADP 배열에는 다음과 같은 속성을 가진 집계된 데이터 포인트 목록이 포함됩니다.

TS

데이터 수집 시점의 타임스탬프.

NS

메트릭 네임스페이스.

M

지표 목록 이 지표는 다음 속성을 포함하고 있습니다.

N

지표의 이름입니다.

Sum

이 텔레메트리 이벤트의 메트릭 값 합계.

U

지표 값의 단위입니다.

각 지표에 대한 자세한 내용은 을 참조하십시오원격 측정 지표.

규칙 생성을 위한 사전 요구 사항 EventBridge

에 대한 EventBridge 규칙을 생성하기 전에 AWS IoT Greengrass 다음을 수행해야 합니다.

- 에서 이벤트, 규칙, 대상을 숙지하세요. EventBridge
- 규칙에 따라 호출되는 대상을 만들고 구성하세요. EventBridge 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 리전 위치에 있어야 합니다. 자세한 내용은 AWS 일반 참조의 서비스 엔드포인트 및 할당량을 참조하십시오.

자세한 내용은 Amazon이란 무엇입니까 EventBridge? 를 참조하십시오. 및 Amazon EventBridge 사용 EventBridge 설명서에서 Amazon 시작하기를 참조하십시오.

원격 측정 데이터를 가져오기 위한 이벤트 규칙 생성(콘솔)

다음 단계를 사용하여 Greengrass AWS Management Console 코어 디바이스에서 게시한 원격 분석 데이터를 수신하는 EventBridge 규칙을 만들 수 있습니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다. 자세한 내용은 Amazon EventBridge User Guide의 AWS 리소스에서 이벤트를 트리거하는 EventBridge 규칙 생성을 참조하십시오.

1. Amazon EventBridge 콘솔을 열고 규칙 생성을 선택합니다.
2. 이름 및 설명에 규칙의 이름과 설명을 입력합니다.
3. 패턴 정의에서 규칙 패턴을 구성합니다.
   a. [Event pattern]을 선택합니다.
   b. Pre-defined pattern by service(서비스별 사전 정의된 패턴)을 선택하십시오.
   c. 서비스 제공업체(Service provider)에 대해 AWS를 선택하세요.
   d. 서비스 이름에서 Greengrass를 선택합니다.
   e. 이벤트 유형에서 Greengrass 텔레메트리 데이터를 선택합니다.
4. 이벤트 버스 선택에서 기본 이벤트 버스 옵션을 유지합니다.
5. 대상 선택에서 대상을 구성합니다. 다음 예시에서는 Amazon SQS 대기열을 사용하지만 다른 대상 유형을 구성할 수 있습니다.
   a. Target에서 SQS 대기열을 선택합니다.
   b. Queue*에서 대상 대기열을 선택합니다.
6. 태그 - 선택 사항에서 규칙에 대한 태그를 정의하거나 필드를 비워 둡니다.
7. 생성을 선택하세요.

원격 측정 데이터(CLI) 를 가져오기 위한 이벤트 규칙 생성

다음 단계를 사용하여 Greengrass AWS CLI 코어 디바이스에서 게시한 원격 분석 데이터를 수신하는 EventBridge 규칙을 만들 수 있습니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다.

1. 규칙을 생성합니다.
   - *## ### ## #####* 사물 이름으로 바꾸십시오.

Linux or Unix

```
aws events put-rule \
  --name MyGreengrassTelemetryEventRule \
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

Windows Command Prompt (CMD)

```
aws events put-rule ^
  --name MyGreengrassTelemetryEventRule ^
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

PowerShell

```
aws events put-rule `
  --name MyGreengrassTelemetryEventRule `
  --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.

   Linux or Unix

   ```
   aws events put-targets \
     --rule MyGreengrassTelemetryEventRule \
     --targets "Id"="1","Arn"="queue-arn"
   ```

   Windows Command Prompt (CMD)

   ```
   aws events put-targets ^
     --rule MyGreengrassTelemetryEventRule ^
   ```

```
  --targets "Id"="1","Arn"="queue-arn"
```

PowerShell

```
aws events put-targets `
  --rule MyGreengrassTelemetryEventRule `
  --targets "Id"="1","Arn"="queue-arn"
```

> **Note**
>
> Amazon이 대상 대기열을 EventBridge 호출하도록 허용하려면 주제에 리소스 기반 정책을 추가해야 합니다. 자세한 내용은 Amazon 사용 EventBridge 설명서의 Amazon SQS 권한을 참조하십시오.

자세한 내용은 Amazon EventBridge 사용 설명서의 이벤트 및 이벤트 패턴을 참조하십시오. EventBridge

배포 및 구성 요소 상태 알림 받기

Amazon EventBridge 이벤트 규칙은 디바이스에서 수신한 Greengrass 배포와 디바이스에 설치된 구성 요소의 상태 변경에 대한 알림을 제공합니다. EventBridge 리소스 변경을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. AWS AWS IoT Greengrass최선을 EventBridge 다해 이러한 이벤트를 전송합니다. 즉, 모든 이벤트를 AWS IoT Greengrass 전송하려고 EventBridge 시도하지만 드문 경우이긴 하지만 이벤트가 전달되지 않을 수도 있습니다. 또한 특정 이벤트의 복사본을 여러 개 전송할 수 있습니다. 즉, 이벤트 리스너가 이벤트가 발생한 순서대로 이벤트를 수신하지 못할 수도 있습니다. AWS IoT Greengrass

> **Note**
>
> EventBridge Amazon은 Greengrass 코어 디바이스, 배포 및 구성 요소 알림과 같은 다양한 소스의 데이터와 애플리케이션을 연결하는 데 사용할 수 있는 이벤트 버스 서비스입니다. 자세한 내용은 Amazon이란 무엇입니까 EventBridge? 를 참조하십시오. Amazon EventBridge 사용 설명서에서 확인할 수 있습니다.

주제

배포 상태 변경 이벤트

AWS IoT Greengrass배포가FAILED, SUCCEEDED 및 COMPLETED 상태가 되면 이벤트를 발생시킵니다. 모든 상태 전환 또는 지정한 상태로의 전환에 대해 실행되는 EventBridge 규칙을 만들 수 있습니다. 배포가 규칙을 시작하는 상태로 전환되면 규칙에 정의된 대상 작업을 EventBridge 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

배포 상태 변경에 대한 이벤트에서 사용하는 형식은 다음과 같습니다.

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Effective Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail":{
        "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED",
        "statusDetails": {
```

```
          "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR",
 "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
          "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
        },
        "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to
 download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for
 component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access
 Denied. Ensure the IAM role associated with the core device has a policy granting
 s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
 Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
luLCYANgq08RIH+x2H+hEKc=)"
     }
}
```

배포 상태를 업데이트하는 규칙 및 이벤트를 생성할 수 있습니다. 배포가FAILED,SUCCEEDED, OR로 완료되면 이벤트가 시작됩니다. COMPLETED 코어 기기에서 배포가 실패한 경우 배포가 실패한 이유를 설명하는 자세한 응답을 받게 됩니다. 배포 오류 코드에 대한 자세한 내용은 을 참조하십시오세부 배포 오류 코드.

배포 상태

- FAILED. 배포가 실패했습니다.
- SUCCEEDED. 사물 그룹을 대상으로 하는 배포가 성공적으로 완료되었습니다.
- COMPLETED. 배포를 목표로 한 일이 성공적으로 완료되었습니다.

이벤트가 중복되거나 이벤트 순서가 잘못되었을 수 있습니다. 이벤트 순서를 정하려면 time 속성을 사용하세요.

및 에 있는 오류 코드의 전체 목록은 errorStacks 및 errorTypes 을 참조하십시오 세부 배포 오류 코드세부 구성 요소 상태 코드.

구성 요소 상태 변경 이벤트

AWS IoT Greengrass구성 요소가 다음 상태에 들어갈 때 이벤트를 발생시킵니다: ERRORED 및BROKEN. Greengrass는 배포가 완료되면 이벤트도 내보냅니다. 모든 상태 전환 또는 지정한 상태로의 전환에 대해 실행되는 EventBridge 규칙을 만들 수 있습니다. 설치된 구성 요소가 규칙을 시작하는 상태에 들어가면 규칙에 정의된 대상 동작을 EventBridge 호출합니다. 이렇게 하면 알림을 전송하고, 이벤트 정보를 캡처하고, 적절한 조치를 취하거나, 상태 변경에 대응하는 기타 이벤트를 시작할 수 있습니다.

구성 요소 상태 변경 이벤트는 다음 형식을 사용합니다.

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

설치된 구성 요소의 상태를 업데이트하는 규칙 및 이벤트를 만들 수 있습니다. 디바이스에서 구성 요소의 상태가 변경될 때 이벤트가 시작됩니다. 구성 요소에 오류가 발생하거나 손상된 이유를 설명하는 자세한 응답을 받게 됩니다. 또한 실패 이유를 나타내는 상태 코드도 받게 됩니다. 구성 요소 상태 코드에 대한 자세한 내용은 을 참조하십시오세부 구성 요소 상태 코드.

규칙 생성을 위한 사전 요구 사항 EventBridge

에 대한 EventBridge 규칙을 생성하기 전에 AWS IoT Greengrass 다음을 수행하십시오.

- 에서 이벤트, 규칙, 대상을 숙지하세요. EventBridge
- 규칙에 따라 호출되는 대상을 만들고 구성하세요. EventBridge 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 Amazon이란 무엇입니까 EventBridge? 를 참조하십시오. 및 Amazon EventBridge 사용 EventBridge 설명서에서 Amazon 시작하기를 참조하십시오.

디바이스 상태 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 User Guide의 AWS 리소스에서 이벤트를 트리거하는 EventBridge 규칙 생성을 참조하십시오.

1. Amazon EventBridge 콘솔을 엽니다.
2. 탐색 창에서 규칙을 선택합니다.
3. 규칙 생성을 선택합니다.
4. 규칙에 대해 이름과 설명을 입력합니다.

   규칙은 동일한 지역과 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 유형에서 이벤트 패턴이 있는 규칙을 선택합니다.
7. 다음을 선택합니다.
8. 이벤트 소스(Event source)에서 AWS 이벤트(events)를 선택합니다.
9. 이벤트 패턴의 경우, AWS 서비스를 선택합니다.
10. AWS 서비스 이름에서 Greengrass를 선택합니다.
11. 이벤트 유형에서 다음 중에서 선택합니다.
    - 배포 이벤트의 경우 Greengrass V2 유효 배포 상태 변경을 선택합니다.
    - 구성 요소 이벤트의 경우 Greengrass V2 설치된 구성 요소 상태 변경을 선택합니다.
12. 다음을 선택합니다.
13. 대상 유형에서 AWS서비스를 선택합니다.
14. 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. 다음을 선택합니다.

15. 다음을 선택합니다.
16. 규칙의 세부 정보를 검토하고 규칙 생성을 선택합니다.

디바이스 상태 알림 (CLI) 구성

다음 단계를 사용하여 Greengrass 상태 변경 이벤트가 있을 때 Amazon SNS 주제를 게시하는 EventBridge 규칙을 생성하십시오. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다.

1. 규칙을 생성합니다.

   - 배포 상태 변경 이벤트용.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Effective Deployment Status Change\"]}"
   ```

   - 구성 요소 상태 변경 이벤트용.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Installed Component Status Change\"]}"
   ```

   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**
>
> Amazon에서 대상 주제를 EventBridge 호출하도록 허용하려면 주제에 리소스 기반 정책을 추가해야 합니다. 자세한 내용은 Amazon 사용 설명서의 Amazon SNS EventBridge 권한을 참조하십시오.

자세한 내용은 Amazon EventBridge 사용 설명서의 이벤트 및 이벤트 패턴을 참조하십시오. EventBridge

디바이스 상태 알림 구성 (AWS CloudFormation)

AWS CloudFormation템플릿을 사용하여 Greengrass 그룹 배포의 상태 변경에 대한 알림을 보내는 EventBridge 규칙을 만들 수 있습니다. 자세한 내용은 AWS CloudFormation사용 설명서의 Amazon EventBridge 리소스 유형 참조를 참조하십시오.

다음 사항도 참조하십시오.

- 디바이스 배포 상태 확인
- 아마존이란 EventBridge 무엇입니까? Amazon EventBridge 사용 설명서에서

그린그래스 코어 디바이스 상태 확인

Greengrass 코어 디바이스는 소프트웨어 구성 요소의 상태를 에 보고합니다. AWS IoT Greengrass 각 장치의 상태 요약을 확인하고 각 장치의 각 구성 요소 상태를 확인할 수 있습니다.

코어 디바이스의 상태는 다음과 같습니다.

- `HEALTHY`— AWS IoT Greengrass Core 소프트웨어와 모든 구성 요소가 코어 장치에서 문제 없이 실행됩니다.
- `UNHEALTHY`— AWS IoT Greengrass 코어 소프트웨어 또는 구성 요소가 코어 장치에서 오류 상태에 있습니다.

> **Note**
>
> AWS IoT Greengrass개별 장치가 상태 업데이트를 에 전송하도록 합니다. AWS 클라우드 AWS IoT GreengrassCore 소프트웨어가 장치에서 실행되고 있지 않거나 장치가 에 연결되어

있지 않으면 해당 장치의 보고된 상태가 현재 상태를 반영하지 않을 수 있습니다. AWS 클라우드 상태 타임스탬프는 장치 상태가 마지막으로 업데이트된 시기를 나타냅니다.
코어 기기는 다음과 같은 시간에 상태 업데이트를 전송합니다.

- AWS IoT GreengrassCore 소프트웨어가 시작될 때
- 코어 디바이스가 다음으로부터 배포를 받는 경우 AWS 클라우드
- 코어 디바이스에 있는 구성 요소의 상태가 ERRORED 또는 상태가 되는 경우 BROKEN
- 구성할 수 있는 일정한 간격 (기본값: 24시간) 으로

AWS IoT GreengrassCore v2.7.0의 경우 로컬 배포 및 클라우드 배포가 발생할 때 코어 디바이스가 상태 업데이트를 보냅니다.

주제

코어 디바이스의 상태 확인

개별 코어 디바이스의 상태를 확인할 수 있습니다.

코어 디바이스의 상태를 확인하려면 (AWS CLI)

- 다음 명령을 실행하여 장치 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 *coreDeviceName*바꿉니다.

  ```
  aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
  ```

  응답에는 상태를 포함하여 코어 장치에 대한 정보가 포함됩니다.

핵심 장치 그룹의 상태를 확인합니다.

핵심 장치 그룹 (사물 그룹) 의 상태를 확인할 수 있습니다.

장치 그룹의 상태를 확인하려면 (AWS CLI)

- 다음 명령을 실행하여 여러 코어 디바이스의 상태를 검색합니다. 명령의 ARN을 쿼리할 사물 그룹의 ARN으로 대체합니다.

  ```
  aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
  ```

  응답에는 사물 그룹의 핵심 장치 목록이 포함됩니다. 목록의 각 항목에는 코어 디바이스의 상태가 포함됩니다.

핵심 장치 구성 요소 상태를 확인합니다.

코어 장치에 있는 소프트웨어 구성 요소의 상태 (예: 수명 주기 상태) 를 확인할 수 있습니다. 구성 요소 수명 주기 상태에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass구성 요소 개발.

코어 장치의 구성 요소 상태를 확인하려면 (AWS CLI)

- 다음 명령을 실행하여 코어 장치의 구성 요소 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 *coreDeviceName*바꿉니다.

  ```
  aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
  ```

  응답에는 코어 기기에서 실행되는 구성 요소 목록이 포함됩니다. 목록의 각 항목에는 데이터의 최신 상태, Greengrass 코어 디바이스가 특정 구성 요소가 포함된 메시지를 클라우드로 마지막으로 보낸 시점을 포함하여 구성 요소의 라이프사이클 상태가 포함됩니다. 응답에는 구성 요소를 Greengrass 코어 장치로 가져온 가장 최근의 배포 소스도 포함됩니다.

  > **Note**
  >
  > 이 명령은 Greengrass 코어 디바이스가 실행하는 구성 요소의 페이지별 목록을 검색합니다. 기본적으로 이 목록에는 다른 구성 요소의 종속 항목으로 배포된 구성 요소는 포함되지 않습니다. `topologyFilter`파라미터를 로 설정하여 응답에 종속성을 포함할 수 있습니다. `ALL`

AWS Lambda함수 실행

> **Note**
>
> AWS IoT Greengrass현재 Windows 코어 장치에서는 이 기능을 지원하지 않습니다.

AWS Lambda함수를 AWS IoT Greengrass 핵심 장치에서 실행되는 구성 요소로 가져올 수 있습니다. 다음과 같은 경우에 이 작업을 수행하는 것이 좋습니다.

- 핵심 디바이스에 배포하려는 Lambda 함수의 애플리케이션 코드가 있습니다.
- 코어 디바이스에서 AWS IoT Greengrass V2 실행하려는 AWS IoT Greengrass V1 애플리케이션이 있습니다. 자세한 설명은 2단계: 애플리케이션을 마이그레이션하기 위한 AWS IoT Greengrass V2 구성 요소 생성 및 배포 AWS IoT Greengrass V1 섹션을 참조하세요.

Lambda 함수는 다음 구성 요소에 대한 종속성을 포함합니다. 함수를 가져올 때 이러한 구성 요소를 종속성으로 정의할 필요는 없습니다. Lambda 함수 구성 요소를 배포할 때 배포에는 이러한 Lambda 구성 요소 종속성이 포함됩니다.

- Lambda 런처 구성 요소 `aws.greengrass.LambdaLauncher` () 는 프로세스 및 환경 구성을 처리합니다.
- Lambda 관리자 구성 요소 `aws.greengrass.LambdaManager` () 는 프로세스 간 통신 및 확장을 처리합니다.
- Lambda 런타임 구성 요소 `aws.greengrass.LambdaRuntimes` () 는 지원되는 각 Lambda 런타임에 대한 아티팩트를 제공합니다.

주제

- 요구 사항
- Lambda 함수 수명 주기 구성
- Lambda 함수 컨테이너화를 구성합니다.
- Lambda 함수를 구성 요소로 가져오기 (콘솔)
- Lambda 함수를 구성 요소로 가져오기 () AWS CLI

요구 사항

코어 디바이스와 Lambda 함수가 다음 요구 사항을 충족해야 Core 소프트웨어에서 AWS IoT Greengrass 함수를 실행할 수 있습니다.

- 코어 디바이스는 Lambda 함수를 실행하기 위한 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 해당 디바이스가 해당 요구 사항을 충족해야 합니다. 자세한 설명은 Lambda 함수 요구 사항 섹션을 참조하세요.
- Lambda 함수가 사용하는 프로그래밍 언어를 코어 디바이스에 설치해야 합니다.

> **Tip**
>
> 프로그래밍 언어를 설치하는 구성 요소를 생성한 다음 해당 구성 요소를 Lambda 함수 구성 요소의 종속 항목으로 지정할 수 있습니다. Greengrass는 Lambda가 지원하는 모든 버전의 Python, Node.js 및 Java 런타임을 지원합니다. Greengrass는 더 이상 사용되지 않는 Lambda 런타임 버전에 추가 제한을 적용하지 않습니다. 에서 이러한 지원 중단된 런타임을 사용하는 Lambda 함수를 실행할 수는 있지만 AWS IoT Greengrass 에서 생성할 수는 없습니다. AWS Lambda Lambda 런타임용 AWS IoT Greengrass 지원에 대한 자세한 내용은 AWS Lambda함수 실행 섹션을 참조하세요.

Lambda 함수 수명 주기 구성

Lambda 함수 수명 주기에 따라 함수가 시작되는 시점과 컨테이너를 생성 및 사용하는 방법이 결정됩니다. 수명 주기는 또한 AWS IoT Greengrass Core 소프트웨어가 함수 핸들러 외부에 있는 변수와 전처리 로직을 유지하는 방법을 결정합니다.

AWS IoT Greengrass온디맨드 (기본값) 및 수명이 긴 수명 주기를 지원합니다.

- 온디맨드 함수는 호출될 때 시작되고 실행할 작업이 남아 있지 않으면 중지됩니다. 기존 컨테이너를 재사용할 수 있는 경우가 아니면 함수를 호출할 때마다 별도의 컨테이너 (샌드박스라고도 함) 가 생성되어 호출을 처리합니다. 함수에 보내는 데이터를 모든 컨테이너가 처리할 수 있습니다.

  온디맨드 함수의 여러 호출을 동시에 실행할 수 있습니다.

  함수 핸들러 외부에서 정의한 변수와 전처리 로직은 새 컨테이너를 생성할 때 유지되지 않습니다.
- 수명이 긴 (또는 고정된) 함수는 AWS IoT Greengrass Core 소프트웨어가 단일 컨테이너에서 시작되고 실행될 때 시작됩니다. 함수로 보내는 모든 데이터를 동일한 컨테이너가 처리합니다.

AWS IoT GreengrassCore 소프트웨어가 이전 호출을 실행할 때까지 여러 호출이 대기열에 추가됩니다.

함수 핸들러 외부에서 정의한 변수와 전처리 로직은 핸들러를 호출할 때마다 보존됩니다.

초기 입력 없이 작업을 시작해야 하는 경우 수명이 긴 Lambda 함수를 사용하십시오. 예를 들어 수명이 긴 함수는 기계 학습 모델을 로드하고 처리를 시작하여 함수가 디바이스 데이터를 수신할 때 바로 사용할 수 있습니다.

> **Note**
>
> 수명이 긴 함수에는 핸들러를 호출할 때마다 타임아웃이 발생합니다. 무기한 실행되는 코드를 호출하려면 핸들러 외부에서 코드를 시작해야 합니다. 함수 초기화를 방해할 수 있는 차단 코드가 핸들러 외부에 없는지 확인하세요.
> 이러한 함수는 AWS IoT Greengrass Core 소프트웨어가 중지되지 않는 한 (예: 배포 또는 재부팅) 실행됩니다. 함수에서 포착되지 않은 예외가 발생하거나, 메모리 제한을 초과하거나, 핸들러 시간 초과와 같은 오류 상태가 되면 이러한 함수가 실행되지 않습니다.

컨테이너 재사용에 대한 자세한 내용은 Compute 블로그의 컨테이너 재사용 이해를 참조하십시오. AWS Lambda AWS

Lambda 함수 컨테이너화를 구성합니다.

기본적으로 Lambda 함수는 컨테이너 내에서 실행됩니다. AWS IoT Greengrass Greengrass 컨테이너는 함수와 호스트 간의 격리를 제공합니다. 이러한 격리는 호스트와 컨테이너 내 함수 모두의 보안을 강화합니다.

사용 사례에서 컨테이너화 없이 실행해야 하는 경우가 아니라면 Greengrass 컨테이너에서 Lambda 함수를 실행하는 것이 좋습니다. Greengrass 컨테이너에서 Lambda 함수를 실행하면 리소스에 대한 액세스를 제한하는 방법을 더 잘 제어할 수 있습니다.

다음과 같은 경우 컨테이너화 없이 Lambda 함수를 실행할 수 있습니다.

- 컨테이너 모드를 지원하지 않는 AWS IoT Greengrass 디바이스에서 실행하려고 합니다. 특수 Linux 배포판을 사용하거나 오래된 이전 커널 버전을 사용하려는 경우를 예로 들 수 있습니다.
- 자체 OverlayFS를 사용하는 다른 컨테이너 환경에서 Lambda 함수를 실행하려 하지만 Greengrass 컨테이너에서 실행 시 OverlayFS 충돌이 발생하는 경우.

- 배포 시 경로를 확인할 수 없거나 배포 후 경로가 변경될 수 있는 로컬 리소스에 액세스해야 합니다. 이 리소스의 예로는 플러그형 기기를 들 수 있습니다.
- 프로세스로 작성된 이전 애플리케이션이 있는데 Greengrass 컨테이너에서 실행할 때 문제가 발생합니다.

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 시스템 그룹에서 실행되더라도 다른 Lambda 함수의 배포된 코드에 액세스할 수 없습니다. 즉, Lambda 함수는 서로 격리된 상태로 실행됩니다. • AWS IoT GreengrassCore 소프트웨어는 Lambda 함수와 동일한 컨테이너에서 모든 하위 프로세스를 실행하므로 Lambda 함수가 중지되면 하위 프로세스도 중지됩니다.

컨테이너 없음	• 다음 기능은 컨테이너화되지 않은 Lambda 함수에서는 사용할 수 없습니다.
	  • Lambda 함수 메모리 제한.
	  • 로컬 디바이스 및 볼륨 리소스. Lambda 함수 리소스 대신 코어 디바이스의 파일 경로를 사용하여 이러한 리소스에 액세스해야 합니다.
	• 컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 액세스하는 경우 리소스 소유자를 식별하고 Lambda 함수가 아닌 리소스에 대한 액세스 권한을 설정해야 합니다.
	• 컨테이너화되지 않은 Lambda 함수는 동일한 시스템 그룹에서 실행되는 다른 Lambda 함수

컨테이너화	참고
	의 배포된 코드에 읽기 전용으로 액세스할 수 있습니다.

Lambda 함수를 배포할 때 컨테이너화를 변경하면 함수가 예상대로 작동하지 않을 수 있습니다. Lambda 함수가 새 컨테이너화 설정으로 더 이상 사용할 수 없는 로컬 리소스를 사용하는 경우 배포가 실패합니다.

- Lambda 함수를 Greengrass 컨테이너에서 실행하는 것에서 컨테이너화 없이 실행하는 것으로 변경하면 함수의 메모리 제한이 삭제됩니다. 연결된 로컬 리소스를 사용하기보다 파일 시스템에 직접 액세스해야 합니다. Lambda 함수를 배포하기 전에 연결된 리소스를 모두 제거해야 합니다.
- Lambda 함수를 컨테이너화 없이 실행하는 방식에서 컨테이너에서 실행하는 방식으로 변경하면 Lambda 함수는 파일 시스템에 대한 직접 액세스 권한을 상실합니다. 각 함수의 메모리 제한을 정의하거나 기본 16MB 메모리 제한을 수락해야 합니다. Lambda 함수를 배포할 때 각 Lambda 함수에 대해 이러한 설정을 구성할 수 있습니다.

Lambda 함수 구성 요소의 컨테이너화 설정을 변경하려면 구성 요소를 배포할 때 `containerMode` 구성 파라미터의 값을 다음 옵션 중 하나로 설정하십시오.

- `NoContainer`— 구성 요소는 격리된 런타임 환경에서 실행되지 않습니다.
- `GreengrassContainer`— 구성 요소는 AWS IoT Greengrass 컨테이너 내부의 격리된 런타임 환경에서 실행됩니다.

구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포 및 을 참조하십시오구성 요소 구성 업데이트.

Lambda 함수를 구성 요소로 가져오기 (콘솔)

AWS IoT Greengrass콘솔을 사용하여 Lambda 함수 구성 요소를 생성할 때는 AWS Lambda 기존 함수를 가져온 다음 Greengrass 디바이스에서 실행되는 구성 요소를 생성하도록 구성합니다.

시작하기 전에 Greengrass 디바이스에서 Lambda 함수를 실행하기 위한 요구 사항을 검토하십시오.

Tasks

- 1단계: 가져올 Lambda 함수 선택

1단계: 가져올 Lambda 함수 선택

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지에서 구성 요소 생성을 선택합니다.
3. 구성 요소 생성 페이지의 구성 요소 정보에서 Lambda 함수 가져오기를 선택합니다.
4. Lambda 함수에서 가져오려는 Lambda 함수를 검색하여 선택합니다.

   AWS IoT GreengrassLambda 함수의 이름을 사용하여 구성 요소를 생성합니다.
5. Lambda 함수 버전에서 가져올 버전을 선택합니다. Lambda 별칭은 다음과 같이 선택할 수 없습니다. `$LATEST`

   AWS IoT GreengrassLambda 함수 버전을 유효한 시맨틱 버전으로 사용하여 구성 요소를 생성합니다. 예를 들어 함수 버전이 `3`인 경우 구성 요소 버전은 `3.0.0`가 됩니다.

2단계: Lambda 함수 파라미터 구성

구성 요소 생성 페이지의 Lambda 함수 구성에서 Lambda 함수를 실행하는 데 사용할 다음 파라미터를 구성합니다.

1. (선택 사항) Lambda 함수가 작업 메시지를 구독하는 이벤트 소스 목록을 추가합니다. 이 함수를 로컬 게시/구독 메시지 및 MQTT 메시지에 구독하도록 이벤트 소스를 지정할 수 있습니다. AWS IoT Core Lambda 함수는 이벤트 소스로부터 메시지를 수신할 때 호출됩니다.

   > **Note**
   >
   > 이 함수를 다른 Lambda 함수 또는 구성 요소의 메시지를 구독하려면 이 Lambda 함수 구성 요소를 배포할 때 기존 구독 라우터 구성 요소를 배포하십시오. 레거시 구독 라우터 구성 요소를 배포할 때 Lambda 함수가 사용하는 구독을 지정하십시오.

이벤트 소스에서 다음을 수행하여 이벤트 소스를 추가하십시오.

a. 추가하는 각 이벤트 소스에 대해 다음 옵션을 지정하십시오.

- 주제 — 메시지를 구독하기 위한 주제입니다.
- 유형 — 이벤트 소스의 유형입니다. 다음 옵션 중 하나를 선택합니다.
  - 로컬 게시/구독 — 로컬 게시/구독 메시지를 구독합니다.

    Greengrass nucleus v2.6.0 이상 및 Lambda 관리자 v2.2.5 이상을 사용하는 경우, 이 유형을 지정할 때 주제에 MQTT 주제 와일드카드 (및) 를 사용할 수 있습니다. **+ #**
  - AWS IoT CoreAWS IoT CoreMQTT — MQTT 메시지를 구독하십시오.

    이 유형을 지정할 때 주제에 MQTT 주제 와일드카드 (**+**및**#**) 를 사용할 수 있습니다.

b. 다른 이벤트 소스를 추가하려면 [Add event source] 를 선택하고 이전 단계를 반복합니다. 이벤트 소스를 제거하려면 제거하려는 이벤트 소스 옆의 제거를 선택합니다.

2. Timeout (초) 에는 고정되지 않은 Lambda 함수가 제한 시간이 초과되기 전에 실행할 수 있는 최대 시간 (초) 을 입력합니다. 기본값은 3초입니다.
3. Pinned의 경우, Lambda 함수 구성 요소를 고정할지 여부를 선택합니다. 기본값은 True입니다.

- 고정된 (또는 수명이 긴) Lambda 함수는 시작 시 AWS IoT Greengrass 시작되어 자체 컨테이너에서 계속 실행됩니다.
- 고정되지 않은 (또는 온디맨드) Lambda 함수는 작업 항목을 수신할 때만 시작되고 지정된 최대 유휴 시간 동안 유휴 상태를 유지한 후에 종료됩니다. 함수에 여러 작업 항목이 있는 경우 AWS IoT Greengrass Core 소프트웨어는 함수의 여러 인스턴스를 생성합니다.

4. (선택 사항) 추가 파라미터에서 다음 Lambda 함수 파라미터를 설정합니다.

- 상태 제한 시간 (초) - Lambda 함수 구성 요소가 Lambda 관리자 구성 요소에 상태 업데이트를 보내는 간격 (초) 입니다. 이 파라미터는 고정된 함수에만 적용됩니다. 기본값은 60초입니다.
- 최대 대기열 크기 - Lambda 함수 구성 요소에 대한 메시지 대기열의 최대 크기입니다. AWS IoT GreengrassCore 소프트웨어는 Lambda 함수를 실행하여 각 메시지를 사용할 수 있을 때까지 메시지를 FIFO (선입선출) 대기열에 저장합니다. 기본값은 1,000개의 메시지입니다.
- 최대 인스턴스 수 - 고정되지 않은 Lambda 함수가 동시에 실행할 수 있는 최대 인스턴스 수입니다. 기본값은 100개 인스턴스입니다.

- 최대 유휴 시간 (초) - 고정되지 않은 Lambda 함수가 Core 소프트웨어가 프로세스를 중지하기 전에 유휴 상태로 유지할 수 있는 최대 시간 (초) 입니다. AWS IoT Greengrass 기본값은 60초입니다.
- 인코딩 유형 - Lambda 함수가 지원하는 페이로드 유형입니다. 다음 옵션 중 하나를 선택합니다.
  - JSON
  - 이진

  기본값은 JSON입니다.

5. (선택 사항) 실행 시 Lambda 함수에 전달할 명령줄 인수 목록을 지정합니다.

   a. 추가 파라미터, 프로세스 인수에서 인수 추가를 선택합니다.

   b. 추가하는 각 인수에 대해 함수에 전달하려는 인수를 입력합니다.

   c. 인수를 제거하려면 제거하려는 인수 옆의 제거를 선택합니다.

6. (선택 사항) Lambda 함수를 실행할 때 사용할 수 있는 환경 변수를 지정합니다. 환경 변수를 사용하면 함수 코드를 변경할 필요 없이 구성 설정을 저장하고 업데이트할 수 있습니다.

   a. 추가 매개변수, 환경 변수에서 환경 변수 추가를 선택합니다.

   b. 추가하는 각 환경 변수에 대해 다음 옵션을 지정합니다.

      - 키 — 변수 이름.
      - 값 — 이 변수의 디폴트 값입니다.

   c. 환경 변수를 제거하려면 제거하려는 환경 변수 옆의 제거를 선택합니다.

3단계: (선택 사항) Lambda 함수에 지원되는 플랫폼 지정

모든 코어 디바이스에는 운영 체제 및 아키텍처에 대한 속성이 있습니다. Lambda 함수 구성 요소를 배포하면 Core 소프트웨어는 사용자가 지정하는 플랫폼 값을 코어 디바이스의 플랫폼 속성과 비교하여 Lambda 함수가 해당 디바이스에서 지원되는지 여부를 결정합니다. AWS IoT Greengrass

> **Note**
>
> Greengrass nucleus 구성 요소를 코어 디바이스에 배포할 때 사용자 지정 플랫폼 속성을 지정할 수도 있습니다. 자세한 내용은 Greengrass nucleus 구성요소의 플랫폼 오버라이드 매개변수를 참조하십시오.

Lambda 함수 구성, 추가 파라미터, 플랫폼에서 다음을 수행하여 이 Lambda 함수가 지원하는 플랫폼을 지정합니다.

1. 각 플랫폼에 대해 다음 옵션을 지정하십시오.
   - 운영 체제 - 플랫폼의 운영 체제 이름입니다. 현재 지원되는 값은 `linux`입니다.
   - 아키텍처 - 플랫폼의 프로세서 아키텍처. 지원되는 값은 다음과 같습니다.
     - `amd64`
     - `arm`
     - `aarch64`
     - `x86`
2. 다른 플랫폼을 추가하려면 [Add platform] 을 선택하고 이전 단계를 반복합니다. 지원되는 플랫폼을 제거하려면 제거하려는 플랫폼 옆의 제거를 선택합니다.

4단계: (선택 사항) Lambda 함수의 구성 요소 종속성 지정

구성 요소 종속성은 함수가 사용하는 추가 AWS 제공 구성 요소 또는 사용자 지정 구성 요소를 식별합니다. Lambda 함수 구성 요소를 배포할 때 배포에는 함수 실행에 대한 이러한 종속성이 포함됩니다.

> **Important**
>
> AWS IoT GreengrassV1에서 실행하도록 생성한 Lambda 함수를 가져오려면 함수가 사용하는 기능 (예: 비밀, 로컬 섀도우, 스트림 관리자) 에 대한 개별 구성 요소 종속성을 정의해야 합니다. 이러한 구성 요소를 하드 종속성으로 정의하여 종속성 상태가 변경될 경우 Lambda 함수 구성 요소가 다시 시작되도록 합니다. 자세한 설명은 V1 람다 함수 가져오기 섹션을 참조하세요.

Lambda 함수 구성, 추가 파라미터, 구성 요소 종속성에서 다음 단계를 완료하여 Lambda 함수의 구성 요소 종속성을 지정하십시오.

1. [종속성 추가] 를 선택합니다.
2. 추가하는 각 구성 요소 종속성에 대해 다음 옵션을 지정합니다.
   - 구성 요소 이름 - 구성 요소 이름. 예를 들어 스트림 관리자 구성 요소를 **`aws.greengrass.StreamManager`** 포함하려면 를 입력합니다.
   - 버전 요구 사항 — 이 구성 요소 종속성의 호환 버전을 식별하는 npm 스타일 시맨틱 버전 제약 조건입니다. 단일 버전 또는 버전 범위를 지정할 수 있습니다. 예를 들어, 이 Lambda 함수가 스

트림 관리자 구성 요소의 첫 번째 메이저 버전에 있는 모든 버전에 종속되도록 **^1.0.0** 지정하려면 를 입력합니다. 시맨틱 버전 제약에 대한 자세한 내용은 npm semver 계산기를 참조하십시오.

- 유형 — 종속성 유형. 다음 옵션 중 하나를 선택합니다.
  - 하드 — 종속성 상태가 변경되면 Lambda 함수 구성 요소가 다시 시작됩니다. 이는 기본 작업입니다.
  - Soft — 종속성 상태가 변경되어도 Lambda 함수 구성 요소가 다시 시작되지 않습니다.

3. 구성 요소 종속성을 제거하려면 구성 요소 종속성 옆의 제거를 선택합니다.

5단계: (선택 사항) 컨테이너에서 Lambda 함수 실행

기본적으로 Lambda 함수는 Core 소프트웨어 내의 AWS IoT Greengrass 격리된 런타임 환경에서 실행됩니다. 격리 없이 (즉, 컨테이너 없음 모드) 프로세스로 Lambda 함수를 실행하도록 선택할 수도 있습니다.

Linux 프로세스 구성에서 격리 모드의 경우 다음 옵션 중에서 선택하여 Lambda 함수의 컨테이너화를 선택합니다.

- Greengrass 컨테이너 - Lambda 함수는 컨테이너에서 실행됩니다. 이는 기본 작업입니다.
- 컨테이너 없음 - Lambda 함수는 격리 없이 프로세스로 실행됩니다.

컨테이너에서 Lambda 함수를 실행하는 경우 다음 단계를 완료하여 Lambda 함수에 대한 프로세스 구성을 구성하십시오.

1. 컨테이너에서 사용할 수 있도록 메모리 양과 시스템 리소스 (예: 볼륨 및 디바이스) 를 구성합니다.

   컨테이너 매개변수에서 다음을 수행합니다.

   a. 메모리 크기에 컨테이너에 할당하려는 메모리 크기를 입력합니다. 메모리 크기를 MB 또는 kB 단위로 지정할 수 있습니다.

   b. 읽기 전용 sys 폴더의 경우 컨테이너가 장치 폴더의 `/sys` 정보를 읽을 수 있는지 여부를 선택합니다. 기본값은 False입니다.

2. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 볼륨을 구성합니다. 볼륨을 정의할 때 AWS IoT Greengrass Core 소프트웨어는 소스 파일을 컨테이너 내의 대상에 탑재합니다.

   a. 볼륨에서 볼륨 추가를 선택합니다.

b. 추가하는 각 볼륨에 대해 다음 옵션을 지정합니다.

- 물리적 볼륨 - 코어 디바이스의 소스 폴더 경로입니다.
- 논리적 볼륨 - 컨테이너 내 대상 폴더의 경로입니다.
- 권한 - (선택 사항) 컨테이너에서 소스 폴더에 액세스할 수 있는 권한입니다. 다음 옵션 중 하나를 선택합니다.
  - 읽기 전용 — Lambda 함수는 소스 폴더에 대한 읽기 전용 액세스 권한을 가집니다. 이는 기본 작업입니다.
  - 읽기-쓰기 — Lambda 함수는 소스 폴더에 대한 읽기/쓰기 액세스 권한을 가집니다.
- 그룹 소유자 추가 - (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 소스 폴더의 소유자로 추가할지 여부. 기본값은 False입니다.

c. 볼륨을 제거하려면 제거하려는 볼륨 옆의 제거를 선택합니다.

3. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 시스템 디바이스를 구성합니다.

a. 디바이스에서 디바이스 추가를 선택합니다.

b. 추가하는 각 장치에 대해 다음 옵션을 지정합니다.

- 마운트 경로 - 코어 디바이스의 시스템 디바이스 경로입니다.
- 권한 - (선택 사항) 컨테이너에서 시스템 장치에 액세스할 수 있는 권한입니다. 다음 옵션 중 하나를 선택합니다.
  - 읽기 전용 — Lambda 함수는 시스템 디바이스에 대한 읽기 전용 액세스 권한을 가집니다. 이는 기본 작업입니다.
  - 읽기-쓰기 — Lambda 함수는 소스 폴더에 대한 읽기/쓰기 액세스 권한을 가집니다.
- 그룹 소유자 추가 - (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 시스템 디바이스의 소유자로 추가할지 여부. 기본값은 False입니다.

6단계: Lambda 함수 구성 요소 생성

Lambda 함수 구성 요소에 대한 설정을 구성한 후 [Create] 를 선택하여 새 구성 요소 생성을 완료합니다.

코어 디바이스에서 Lambda 함수를 실행하기 위해 코어 디바이스에 새 구성 요소를 배포하면 됩니다. 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포을(를) 참조하세요.

Lambda 함수를 구성 요소로 가져오기 () AWS CLI

CreateComponentVersion작업을 사용하여 Lambda 함수에서 구성 요소를 생성합니다. 이 작업을 호출할 때 Lambda 함수를 `lambdaFunction` 가져오도록 지정하십시오.

Tasks

- 1단계: Lambda 함수 구성 정의
- 2단계: Lambda 함수 구성 요소 생성

1단계: Lambda 함수 구성 정의

1. 라는 `lambda-function-component.json` 파일을 생성한 다음 다음 JSON 객체를 파일에 복사합니다. 를 가져올 Lambda 함수의 `lambdaArn` ARN으로 대체하십시오.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
     }
   }
   ```

   > **Important**
   >
   > 가져올 함수 버전이 포함된 ARN을 지정해야 합니다. `$LATEST`같은 버전 별칭을 사용할 수 없습니다.

2. (선택 사항) 구성 요소의 이름 (`componentName`) 을 지정합니다. 이 파라미터를 생략하면 Lambda 함수의 이름으로 구성 요소가 AWS IoT Greengrass 생성됩니다.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda"
     }
   }
   ```

3. (선택 사항) 구성 요소의 version (`componentVersion`) 을 지정합니다. 이 파라미터를 생략하면 Lambda 함수 버전을 유효한 시맨틱 버전으로 사용하여 구성 요소를 AWS IoT Greengrass 생성합니다. 예를 들어 함수 버전이 `3`인 경우 구성 요소 버전은 `3.0.0`가 됩니다.

> **Note**
>
> 업로드하는 각 구성 요소 버전은 고유해야 합니다. 구성 요소 버전을 업로드한 후에는 편집할 수 없으므로 올바른 구성 요소 버전을 업로드해야 합니다.
> AWS IoT Greengrass구성 요소에 시맨틱 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 버전은 구성 `1.0.0` 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 시맨틱 버전 사양을 참조하십시오.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0"
  }
}
```

4. (선택 사항) 이 Lambda 함수가 지원하는 플랫폼을 지정합니다. 각 플랫폼에는 플랫폼을 식별하는 속성 맵이 포함되어 있습니다. 모든 핵심 기기에는 운영 체제 (os) 및 아키텍처 (architecture) 에 대한 속성이 있습니다. AWS IoT Greengrass코어 소프트웨어는 다른 플랫폼 속성을 추가할 수 있습니다. Greengrass nucleus 구성 요소를 코어 디바이스에 배포할 때 사용자 지정 플랫폼 속성을 지정할 수도 있습니다. 다음을 따릅니다.

   a. 의 Lambda 함수에 플랫폼 목록 (componentPlatforms) 을 추가합니다. `lambda-function-component.json`

      ```
      {
        "lambdaFunction": {
          "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
          "componentName": "com.example.HelloWorldLambda",
          "componentVersion": "1.0.0",
          "componentPlatforms": [

          ]
        }
      }
      ```

b. 지원되는 각 플랫폼을 목록에 추가합니다. 각 플랫폼에는 쉽게 name 식별할 수 있는 기능과 속성 맵이 있습니다. 다음 예제에서는 이 함수가 Linux를 실행하는 x86 장치를 지원하도록 지정합니다.

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

다음 예와 비슷한 문서가 들어 `lambda-function-component.json` 있을 수 있습니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}
```

5. (선택 사항) Lambda 함수의 구성 요소 종속성을 지정합니다. Lambda 함수 구성 요소를 배포할 때 배포에는 함수 실행에 대한 이러한 종속성이 포함됩니다.

> **Important**
>
> AWS IoT GreengrassV1에서 실행하도록 생성한 Lambda 함수를 가져오려면 함수가 사용하는 기능 (예: 비밀, 로컬 섀도우, 스트림 관리자) 에 대한 개별 구성 요소 종속성을 정의해야 합니다. 이러한 구성 요소를 하드 종속성으로 정의하여 종속성 상태가 변경될 경우

Lambda 함수 구성 요소가 다시 시작되도록 합니다. 자세한 설명은 V1 람다 함수 가져오기 섹션을 참조하세요.

다음을 따릅니다.

a. 의 Lambda 함수에 구성 요소 종속성 맵 (componentDependencies) 을 추가합니다. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {

    }
  }
}
```

b. 맵에 각 구성 요소 종속성을 추가합니다. 구성 요소 이름을 키로 지정하고 다음 매개 변수를 사용하여 개체를 지정합니다.

- versionRequirement— 구성 요소 종속성의 호환 버전을 식별하는 npm 스타일 시맨틱 버전 제약 조건입니다. 단일 버전 또는 버전 범위를 지정할 수 있습니다. 시맨틱 버전 제약에 대한 자세한 내용은 npm semver 계산기를 참조하십시오.
- dependencyType— (선택 사항) 종속성 유형. 다음 중에서 선택합니다.
  - SOFT— Lambda 함수 구성 요소는 종속성의 상태가 변경되어도 재시작되지 않습니다.
  - HARD— 종속성 상태가 변경되면 Lambda 함수 구성 요소가 다시 시작됩니다.

  기본값은 HARD입니다.

다음 예제는 이 Lambda 함수가 스트림 관리자 구성 요소의 첫 번째 메이저 버전에 있는 모든 버전에 종속되도록 지정합니다. 스트림 관리자가 재시작되거나 업데이트되면 Lambda 함수 구성 요소가 다시 시작됩니다.

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

다음 예제와 비슷한 문서가 `lambda-function-component.json` 포함될 수 있습니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}
```

6. (선택 사항) 함수를 실행하는 데 사용할 Lambda 함수 파라미터를 구성합니다. 환경 변수, 메시지 이벤트 소스, 타임아웃, 컨테이너 설정 등의 옵션을 구성할 수 있습니다. 다음을 따릅니다.

a. 의 Lambda 함수에 Lambda 파라미터 객체 `componentLambdaParameters` () 를 추가합니다. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {

    }
  }
}
```

b. (선택 사항) Lambda 함수가 작업 메시지를 구독하는 이벤트 소스를 지정합니다. 이 함수를 로컬 게시/구독 메시지 및 MQTT 메시지에 구독하도록 이벤트 소스를 지정할 수 있습니다. AWS IoT Core Lambda 함수는 이벤트 소스로부터 메시지를 수신할 때 호출됩니다.

> **Note**
>
> 이 함수를 다른 Lambda 함수 또는 구성 요소의 메시지를 구독하려면 이 Lambda 함수 구성 요소를 배포할 때 기존 구독 라우터 구성 요소를 배포하십시오. 레거시 구독 라우터 구성 요소를 배포할 때 Lambda 함수가 사용하는 구독을 지정하십시오.

다음을 따릅니다.

i. Lambda 함수 파라미터에 이벤트 소스 (`eventSources`) 목록을 추가합니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [

      ]
    }
  }
}
```

ii. 각 이벤트 소스를 목록에 추가합니다. 각 이벤트 소스에는 다음과 같은 매개변수가 있습니다.

- `topic`— 메시지 구독을 위한 주제입니다.
- `type`— 이벤트 소스 유형. 다음 옵션 중 하나를 선택합니다.
  - PUB_SUB – 로컬 게시/구독 메시지를 구독합니다.

    Greengrass nucleus v2.6.0 이상 및 Lambda 관리자 v2.2.5 이상을 사용하는 경우, 이 유형을 지정할 때 MQTT 주제 와일드카드 (및) 를 사용할 수 있습니다. + # `topic`
  - `IOT_CORE` - AWS IoT Core MQTT 메시지를 구독합니다.

이 유형을 지정할 때 MQTT 주제 와일드카드 (및) 를 사용할 수 있습니다. `+` `#` `topic`

다음 예제는 주제 필터와 일치하는 주제에 대해 AWS IoT Core MQTT를 구독합니다. `hello/world/+`

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

다음 예제와 비슷해 `lambda-function-component.json` 보일 수 있습니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
    }
  }
```

```
}
```

c. (선택 사항) Lambda 함수 파라미터 객체에 다음 파라미터 중 하나를 지정합니다.

- `environmentVariables`— Lambda 함수가 실행될 때 사용할 수 있는 환경 변수의 맵입니다.
- `execArgs`— Lambda 함수가 실행될 때 해당 함수에 전달할 인수 목록입니다.
- `inputPayloadEncodingType`— Lambda 함수가 지원하는 페이로드 유형. 다음 옵션 중 하나를 선택합니다.
  - `json`
  - `binary`

  기본값: `json`
- `pinned`— Lambda 함수가 고정되었는지 여부. 기본값은 `true`입니다.
  - 고정된 (또는 수명이 긴) Lambda 함수는 시작 시 AWS IoT Greengrass 시작되어 자체 컨테이너에서 계속 실행됩니다.
  - 고정되지 않은 (또는 온디맨드) Lambda 함수는 작업 항목을 수신할 때만 시작되고 지정된 최대 유휴 시간 동안 유휴 상태를 유지한 후에 종료됩니다. 함수에 여러 작업 항목이 있는 경우 AWS IoT Greengrass Core 소프트웨어는 함수의 여러 인스턴스를 생성합니다.

  함수의 최대 `maxIdleTimeInSeconds` 유휴 시간을 설정하는 데 사용합니다.
- `timeoutInSeconds`— 제한 시간이 초과되기 전에 Lambda 함수를 실행할 수 있는 최대 시간 (초). 기본값은 3초입니다.
- `statusTimeoutInSeconds`— Lambda 함수 구성 요소가 Lambda 관리자 구성 요소에 상태 업데이트를 보내는 간격 (초). 이 파라미터는 고정된 함수에만 적용됩니다. 기본값은 60초입니다.
- `maxIdleTimeInSeconds`— 고정되지 않은 Lambda 함수가 Core 소프트웨어가 프로세스를 중지하기 전에 유휴 상태로 유지할 수 있는 최대 시간 (초). AWS IoT Greengrass 기본값은 60초입니다.
- `maxInstancesCount`— 고정되지 않은 Lambda 함수가 동시에 실행할 수 있는 최대 인스턴스 수입니다. 기본값은 100개의 인스턴스입니다.
- `maxQueueSize`— Lambda 함수 구성 요소에 대한 메시지 대기열의 최대 크기. AWS IoT GreengrassCore 소프트웨어는 Lambda 함수를 실행하여 각 메시지를 사용할 수 있을 때까

지 메시지를 FIFO (first-in-first-out) 대기열에 저장합니다. 기본값은 1,000개의 메시지입니다.

다음 예와 비슷한 문서가 들어 `lambda-function-component.json` 있을 수 있습니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
```

```
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500
    }
  }
}
```

d. (선택 사항) Lambda 함수의 컨테이너 설정을 구성합니다. 기본적으로 Lambda 함수는 Core 소프트웨어 내의 AWS IoT Greengrass 격리된 런타임 환경에서 실행됩니다. 또한 격리 없이 Lambda 함수를 프로세스로 실행하도록 선택할 수 있습니다. 컨테이너에서 Lambda 함수를 실행하는 경우 컨테이너의 메모리 크기와 Lambda 함수에 사용할 수 있는 시스템 리소스를 구성합니다. 다음을 따릅니다.

i. Linux 프로세스 파라미터 객체 (`linuxProcessParams`) 를 의 Lambda 파라미터 객체에 추가합니다. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
```

```
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {

      }
    }
  }
}
```

ii. (선택 사항) Lambda 함수가 컨테이너에서 실행되는지 여부를 지정합니다. 프로세스 `isolationMode` 파라미터 객체에 파라미터를 추가하고 다음 옵션 중에서 선택하십시오.

- `GreengrassContainer`— Lambda 함수는 컨테이너에서 실행됩니다.
- `NoContainer`— Lambda 함수는 격리 없이 프로세스로 실행됩니다.

기본값은 `GreengrassContainer`입니다.

iii. (선택 사항) 컨테이너에서 Lambda 함수를 실행하는 경우 컨테이너에서 사용할 수 있도록 메모리 양과 시스템 리소스 (예: 볼륨 및 디바이스) 를 구성할 수 있습니다. 다음을 따릅니다.

A. 의 Linux 프로세스 파라미터 객체에 컨테이너 파라미터 객체 (`containerParams`) 를 추가합니다. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
```

```
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {

        }
      }
    }
  }
}
```

B. (선택 사항) `memorySizeInKB` 매개 변수를 추가하여 컨테이너의 메모리 크기를 지정합니다. 기본값은 16,384KB (16MB) 입니다.

C. (선택 사항) `mountROSysfs` 매개 변수를 추가하여 컨테이너가 장치 폴더에서 정보를 읽을 수 있는지 여부를 지정합니다. `/sys` 기본값은 `false`입니다.

D. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 볼륨을 구성합니다. 볼륨을 정의할 때 AWS IoT Greengrass Core 소프트웨어는 소스 파일을 컨테이너 내의 대상에 탑재합니다. 다음을 따릅니다.

I. 볼륨 목록 (`volumes`) 을 컨테이너 파라미터에 추가합니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
```

```
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [

          ]
        }
      }
    }
  }
}
```

II. 각 볼륨을 목록에 추가합니다. 각 볼륨에는 다음과 같은 매개변수가 있습니다.

- `sourcePath`— 코어 디바이스의 소스 폴더 경로.
- `destinationPath`— 컨테이너의 대상 폴더 경로.
- `permission`— (선택 사항) 컨테이너에서 소스 폴더에 액세스할 수 있는 권한. 다음 옵션 중 하나를 선택합니다.
  - `ro`— Lambda 함수는 소스 폴더에 대한 읽기 전용 액세스 권한을 가집니다.
  - `rw`— Lambda 함수는 소스 폴더에 대한 읽기/쓰기 액세스 권한을 가집니다.

  기본값은 `ro`입니다.
- `addGroupOwner`— (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 소스 폴더의 소유자로 추가할지 여부. 기본값은 `false`입니다.

다음 예와 비슷한 문서가 포함되어 `lambda-function-component.json` 있을 수 있습니다.

```
{
```

```
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
```

```
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

E. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 시스템 디바이스를 구성합니다. 다음을 따릅니다.

I. 시스템 디바이스 목록 (devices) 을 컨테이너 파라미터에 추가합니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
```

```
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [

          ]
        }
      }
    }
  }
}
```

II. 각 시스템 장치를 목록에 추가합니다. 각 시스템 장치에는 다음과 같은 매개변수가 있습니다.

- `path`— 코어 장치의 시스템 장치 경로.

- `permission`— (선택 사항) 컨테이너에서 시스템 장치에 액세스할 수 있는 권한. 다음 옵션 중 하나를 선택합니다.
  - `ro`— Lambda 함수는 시스템 디바이스에 대한 읽기 전용 액세스 권한을 가집니다.
  - `rw`— Lambda 함수는 시스템 디바이스에 대한 읽기/쓰기 액세스 권한을 가집니다.

  기본값은 `ro`입니다.
- `addGroupOwner`— (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 시스템 디바이스의 소유자로 추가할지 여부. 기본값은 `false`입니다.

다음 예와 비슷한 문서가 포함되어 `lambda-function-component.json` 있을 수 있습니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
```

```
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [
            {
              "path": "/dev/sda3",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

7. (선택 사항) 구성 요소에 태그 (tags) 를 추가합니다. 자세한 설명은 AWS IoT Greengrass Version 2 리소스 태깅 섹션을 참조하세요.

2단계: Lambda 함수 구성 요소 생성

1. 다음 명령을 실행하여 Lambda 함수 구성 요소를 생성합니다. `lambda-function-component.json`

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

요청이 성공하면 응답은 다음 예제와 비슷해 보입니다.

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
  "status": {
    "componentState": "REQUESTED",
    "message": "NONE",
    "errors": {}
  }
}
```

`arn`출력에서 를 복사하여 다음 단계에서 구성 요소의 상태를 확인합니다.

2. 구성 요소를 만들면 해당 상태는 입니다`REQUESTED`. 그런 다음 구성 요소를 배포할 수 있는지 AWS IoT Greengrass 확인합니다. 다음 명령을 실행하여 구성 요소 상태를 쿼리하고 구성 요소가 배포 가능한지 확인할 수 있습니다. 를 이전 단계의 `arn` ARN으로 교체합니다.

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

구성 요소가 검증되면 응답은 구성 요소 상태가 유효하다는 것을 나타냅니다. `DEPLOYABLE`

```
{
  "arn": "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
```

```
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
}
```

구성 요소가 DEPLOYABLE 완성되면 Lambda 함수를 코어 디바이스에 배포할 수 있습니다. 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포을(를) 참조하세요.

AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core

코어 기기에서 실행되는 구성 요소는 의 AWS IoT Greengrass 코어 프로세스 간 통신 (IPC) 라이브러리를 사용하여 AWS IoT Greengrass 핵 및 기타 Greengrass 구성 요소와 AWS IoT Device SDK 통신할 수 있습니다. IPC를 사용하는 사용자 지정 구성 요소를 개발하고 실행하려면 를 사용하여 AWS IoT Greengrass Core IPC AWS IoT Device SDK 서비스에 연결하고 IPC 작업을 수행해야 합니다.

IPC 인터페이스는 두 가지 유형의 작업을 지원합니다.

- 요청/응답

  구성 요소는 IPC 서비스에 요청을 보내고 요청 결과가 포함된 응답을 받습니다.

- Subscription

  구성 요소는 IPC 서비스에 구독 요청을 보내고 이에 대한 응답으로 이벤트 메시지 스트림을 예상합니다. 구성 요소는 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 구독 핸들러를 제공합니다. AWS IoT Device SDK여기에는 각 IPC 작업에 대한 올바른 응답 및 이벤트 유형이 포함된 핸들러 인터페이스가 포함되어 있습니다. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

주제

- 로컬 배포 및 구성 요소 관리
- 클라이언트 장치 인증 및 권한 부여

IPC 클라이언트 버전

Java 및 Python SDK의 이후 버전에서는 IPC 클라이언트 V2라는 향상된 버전의 IPC 클라이언트를 AWS IoT Greengrass 제공합니다. IPC 클라이언트 V2:

- IPC 작업을 사용하기 위해 작성해야 하는 코드의 양을 줄이고 IPC 클라이언트 V1에서 발생할 수 있는 일반적인 오류를 방지하는 데 도움이 됩니다.
- 구독 핸들러 콜백을 별도의 스레드에서 호출하므로 이제 구독 핸들러 콜백에서 추가 IPC 함수 호출을 비롯한 차단 코드를 실행할 수 있습니다. IPC 클라이언트 V1은 동일한 스레드를 사용하여 IPC 서버와 통신하고 구독 핸들러 콜백을 호출합니다.
- Lambda 표현식 (Java) 또는 함수 (Python) 를 사용하여 구독 작업을 호출할 수 있습니다. IPC 클라이언트 V1을 사용하려면 구독 핸들러 클래스를 정의해야 합니다.
- 각 IPC 작업의 동기 및 비동기 버전을 제공합니다. IPC 클라이언트 V1은 각 작업의 비동기 버전만 제공합니다.

이러한 개선 사항을 활용하려면 IPC 클라이언트 V2를 사용하는 것이 좋습니다. 그러나 이 설명서와 일부 온라인 콘텐츠의 많은 예제는 IPC 클라이언트 V1을 사용하는 방법만 보여줍니다. 다음 예제와 자습서를 사용하여 IPC 클라이언트 V2를 사용하는 샘플 구성 요소를 볼 수 있습니다.

- PublishToTopic예제
- SubscribeToTopic예
- 튜토리얼: 구성 요소 업데이트를 연기하는 Greengrass 구성 요소 개발
- 튜토리얼: MQTT를 통해 로컬 IoT 디바이스와 상호 작용

현재 C++ AWS IoT Device SDK v2용 버전은 IPC 클라이언트 V1만 지원합니다.

프로세스 간 통신을 위한 지원되는 SDK

AWS IoT GreengrassCore IPC 라이브러리는 다음 버전에 포함되어 있습니다. AWS IoT Device SDK

SDK	최소 버전	사용량
AWS IoT Device SDK 자바 v2용	v1.6.0	자바 v2 (IPC 클라이언트 V2) AWS IoT Device SDK 에 사용 섹션 참조
AWS IoT Device SDK 파이썬 v2용	v1.9.0	Python v2 (IPC 클라이언트 V2) AWS IoT Device SDK 에 사용 섹션 참조
AWS IoT Device SDKC++ v2의 경우	v1.17.0	C++ AWS IoT Device SDK v2에 사용하세요. 섹션 참조
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	AWS IoT Device SDK JavaScript v2에 사용 (IPC 클라이언트 V1) 섹션 참조

AWS IoT Greengrass코어 IPC 서비스에 연결

사용자 지정 구성 요소에서 프로세스 간 통신을 사용하려면 AWS IoT Greengrass Core 소프트웨어가 실행하는 IPC 서버 소켓에 대한 연결을 생성해야 합니다. 다음 작업을 완료하여 원하는 언어로 다운로드하고 사용하십시오. AWS IoT Device SDK

자바 v2 (IPC 클라이언트 V2) AWS IoT Device SDK 에 사용

자바 v2 (IPC 클라이언트 V2) AWS IoT Device SDK 에 사용하려면

1. 자바 AWS IoT Device SDK v2용 다운로드 (v1.6.0 이상).
2. 구성 요소에서 사용자 지정 코드를 실행하려면 다음 중 하나를 수행하십시오.
   - 를 포함하는 JAR 파일로 구성 요소를 빌드하고 구성 요소 레시피에서 이 JAR 파일을 실행하십시오. AWS IoT Device SDK

- AWS IoT Device SDKJAR을 구성 요소 아티팩트로 정의하고 구성 요소 레시피에서 응용 프로그램을 실행할 때 해당 아티팩트를 클래스 경로에 추가합니다.

3. 다음 코드를 사용하여 IPC 클라이언트를 생성합니다.

```
try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
}
```

Python v2 (IPC 클라이언트 V2) AWS IoT Device SDK 에 사용

Python v2 (IPC 클라이언트 V2) AWS IoT Device SDK 에 사용하려면

1. AWS IoT Device SDKPython용 다운로드 (v1.9.0 이상).
2. 구성 요소 레시피의 설치 라이프사이클에 SDK의 설치 단계를 추가합니다.
3. AWS IoT GreengrassCore IPC 서비스에 대한 연결을 생성합니다. 다음 코드를 사용하여 IPC 클라이언트를 생성합니다.

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

try:
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

C++ AWS IoT Device SDK v2에 사용하세요.

C++용 AWS IoT Device SDK v2를 빌드하려면 기기에 다음 도구가 있어야 합니다.

- C++ 11 이상
- CMake 3.1 이상

- 다음 컴파일러 중 하나:
  - GCC 4.8 이상
  - 클랑 3.9 이상
  - MSVC 2015 이상

C++ AWS IoT Device SDK v2용으로 사용하려면

1. C++ AWS IoT Device SDK v2용 버전 (v1.17.0 이상) 을 다운로드하십시오.
2. README의 설치 지침을 따라 소스에서 C++ AWS IoT Device SDK v2용 버전을 빌드하십시오.
3. C++ 빌드 도구에서 이전 단계에서 빌드한 Greengrass IPC 라이브러리를 `AWS::GreengrassIpc-cpp` 연결합니다. 다음 `CMakeLists.txt` 예제는 Greengrass IPC 라이브러리를 CMake로 빌드한 프로젝트에 연결합니다.

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
        "*.h"
        "*.cpp"
        )
add_executable(${PROJECT_NAME} ${MAIN_SRC})

set_target_properties(${PROJECT_NAME} PROPERTIES
        LINKER_LANGUAGE CXX
        CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
```

4. 구성 요소 코드에서 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성하여 IPC 클라이언트 () 를 생성합니다. `Aws::Greengrass::GreengrassCoreIpcClient` IPC 연결, 연결 해제 및 오류 이벤트를 처리하는 IPC 연결 수명 주기 핸들러를 정의해야 합니다. 다음 예제는 IPC 클라이언트와 IPC 클라이언트가 연결되고 연결이 끊기고 오류가 발생할 때 인쇄하는 IPC 클라이언트와 IPC 연결 수명 주기 핸들러를 만듭니다.

```
#include <iostream>

#include <aws/crt/Api.h>
```

```
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
 std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    // Use the IPC client to create an operation request.

    // Activate the operation request.
    auto activate = operation.Activate(request, nullptr);
    activate.wait();

    // Wait for Greengrass Core to respond to the request.
```

```
    auto responseFuture = operation.GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
        exit(-1);
    }

    // Check the result of the request.
    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    return 0;
}
```

5. 구성 요소에서 사용자 지정 코드를 실행하려면 코드를 이진 아티팩트로 빌드하고 구성 요소 레시피에서 이진 아티팩트를 실행하십시오. AWS IoT GreengrassCore 소프트웨어가 바이너리 아티팩트를 실행할 수 `OWNER` 있도록 아티팩트의 `Execute` 권한을 설정합니다.

   구성 요소 레시피의 `Manifests` 섹션은 다음 예제와 비슷할 수 있습니다.

   JSON

   ```
   {
     ...
     "Manifests": [
       {
         "Lifecycle": {
           "run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
   ```

```
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
...
Manifests:
  - Lifecycle:
      run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
        Permission:
          Execute: OWNER
```

AWS IoT Device SDK JavaScript v2에 사용 (IPC 클라이언트 V1)

NodeJS와 함께 AWS IoT Device SDK 사용할 JavaScript v2를 빌드하려면 기기에 다음 도구가 있어야 합니다.

- NodeJS 10.0 이상
  - `node -v`실행하여 노드 버전을 확인합니다.
- CMake 3.1 이상

AWS IoT Device SDK JavaScript v2용으로 사용하려면 (IPC 클라이언트 V1)

1. AWS IoT Device SDK JavaScript v2용 다운로드 (v1.12.10 이상).

2. README의 설치 지침을 따라 소스에서 AWS IoT Device SDK v2용 JavaScript 버전을 빌드하십시오.
3. AWS IoT Greengrass코어 IPC 서비스에 대한 연결을 생성합니다. 다음 단계를 완료하여 IPC 클라이언트를 생성하고 연결을 설정합니다.
4. 다음 코드를 사용하여 IPC 클라이언트를 생성합니다.

```
import * as greengrascoreipc from 'aws-iot-device-sdk-v2';

let client = greengrascoreipc.createClient();
```

5. 다음 코드를 사용하여 컴포넌트와 Greengrass 핵을 연결합니다.

```
await client.connect();
```

구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오.

사용자 지정 구성 요소가 일부 IPC 작업을 사용하도록 허용하려면 구성 요소가 특정 리소스에서 작업을 수행하도록 허용하는 권한 부여 정책을 정의해야 합니다. 각 권한 부여 정책은 정책에서 허용하는 작업 목록과 리소스 목록을 정의합니다. 예를 들어 게시/구독 메시징 IPC 서비스는 주제 리소스에 대한 게시 및 구독 작업을 정의합니다. *와일드카드를 사용하여 모든 작업 또는 모든 리소스에 대한 액세스를 허용할 수 있습니다.

구성 매개 변수를 사용하여 권한 부여 정책을 정의합니다. `accessControl` 구성 매개 변수는 구성 요소 레시피에서 설정하거나 구성 요소를 배포할 때 설정할 수 있습니다. `accessControl`객체는 IPC 서비스 식별자를 권한 부여 정책 목록에 매핑합니다. 각 IPC 서비스에 대해 여러 권한 부여 정책을 정의하여 액세스를 제어할 수 있습니다. 각 권한 부여 정책에는 모든 구성 요소 간에 고유해야 하는 정책 ID가 있습니다.

> **Tip**
>
> 고유한 정책 ID를 생성하려면 구성 요소 이름, IPC 서비스 이름 및 카운터를 조합하면 됩니다. 예를 들어, 이름이 지정된 구성 요소는 다음 ID를 가진 두 개의 게시/구독 권한 부여 정책을 정의할 `com.example.HelloWorld` 수 있습니다.
>
> - `com.example.HelloWorld:pubsub:1`
> - `com.example.HelloWorld:pubsub:2`

권한 부여 정책은 다음 형식을 사용합니다. 이 개체는 `accessControl` 구성 매개변수입니다.

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
    }
  }
}
```

YAML

```
IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
      - resource1
      - resource2
```

권한 부여 정책의 와일드카드

IPC 권한 부여 정책 `resources` 요소에 * 와일드카드를 사용하여 단일 권한 부여 정책에서 여러 리소스에 대한 액세스를 허용할 수 있습니다.

- Greengrass Nucleus의 모든 버전에서는 단일 * 문자를 리소스로 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다.
- Greengrass nucleus v2.6.0 이상에서는 리소스의 문자를 모든 * 문자 조합과 일치하도록 지정할 수 있습니다. 예를 들어 각 장치 이름이 로 시작되는 공장 내 모든 온도 조절기 장치의 상태 항목에

대한 액세스를 `factory/1/devices/Thermostat*/status` 허용하도록 지정할 수 있습니다. `Thermostat`

AWS IoT CoreMQTT IPC 서비스에 대한 권한 부여 정책을 정의할 때 MQTT 와일드카드 (+및#) 를 사용하여 여러 리소스를 매칭할 수도 있습니다. 자세한 내용은 MQTT IPC 권한 부여 정책의 MQTT 와일드카드를 참조하십시오. AWS IoT Core

권한 부여 정책의 레시피 변수

Greengrass nucleus v2.6.0 이상을 사용하고 Greengrass nucleus의 interpolateComponentConfiguration구성 옵션을 로 설정하면 권한 부여 정책에서 레시피 변수를 사용할 수 `true` 있습니다. `{iot:thingName}` MQTT 주제 또는 디바이스 섀도와 같이 코어 디바이스 이름을 포함하는 권한 부여 정책이 필요한 경우 이 레시피 변수를 사용하여 코어 디바이스 그룹에 대한 단일 권한 부여 정책을 구성할 수 있습니다. 예를 들어 섀도우 IPC 작업을 위해 구성 요소가 다음 리소스에 액세스하도록 허용할 수 있습니다.

```
$aws/things/{iot:thingName}/shadow/
```

권한 부여 정책의 특수 문자

권한 부여 정책에서 리터럴 * 또는 ? 문자를 지정하려면 이스케이프 시퀀스를 사용해야 합니다. 다음 이스케이프 시퀀스는 AWS IoT Greengrass Core 소프트웨어가 문자의 특수 의미 대신 리터럴 값을 사용하도록 지시합니다. 예를 들어, 문자는 모든 * 문자 조합과 일치하는 와일드카드입니다.

리터럴 문자	이스케이프 시퀀스	참고
*	`${*}`	
?	`${?}`	AWS IoT Greengrass현재 단일 문자와 일치하는 ? 와일드카드를 지원하지 않습니다.
$	`${$}`	이 이스케이프 시퀀스를 사용하여 포함된 `${` 리소스를 일치시키십시오. 예를 들어`${resourceName}` , 이름이 지정된 리소스와 일치시

리터럴 문자	이스케이프 시퀀스	참고
		키려면 다음을 지정해야 합니다${$}{resourceName} . 그렇지 않으면 로 시작하는 주제에 대한 액세스를 허용하는 등의 $ 리터럴을 사용하여 포함하는 $ 리소스를 일치시킬 수 있습니다. $aws

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 권한 부여 정책이 포함된 구성 요소 레시피 예시

다음 예제 구성 요소 레시피에는 권한 부여 정책을 정의하는 accessControl 객체가 포함되어 있습니다. 이 정책은 com.example.HelloWorld 구성 요소가 test/topic 주제에 게시할 수 있는 권한을 부여합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
```

```
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/HelloWorld.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
            - "test/topic"
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/HelloWorld.jar
```

Example 권한 부여 정책을 사용한 구성 요소 구성 업데이트의 예

배포의 다음 예제 구성 업데이트는 권한 부여 정책을 정의하는 `accessControl` 객체를 사용하여 구성 요소를 구성하도록 지정합니다. 이 정책은 `com.example.HelloWorld` 구성 요소가 `test/topic` 주제에 게시할 수 있는 권한을 부여합니다.

Console

병합할 구성

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

다음 명령어는 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

hello-world-deployment.json파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":
{\"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
 publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
\"resources\":[\"test/topic\"]}}}}"
      }
    }
```

```
  }
}
```

Greengrass CLI

다음 Greengrass CLI 명령은 코어 디바이스에 로컬 배포를 생성합니다.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

이 `hello-world-configuration.json` 파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  }
}
```

IPC 이벤트 스트림을 구독하세요.

IPC 작업을 사용하여 Greengrass 코어 디바이스에서 이벤트 스트림을 구독할 수 있습니다. 구독 작업을 사용하려면 구독 핸들러를 정의하고 IPC 서비스에 대한 요청을 생성하십시오. 그러면 IPC 클라이언트는 코어 기기가 구성 요소에 이벤트 메시지를 스트리밍할 때마다 구독 핸들러의 함수를 실행합니다.

구독을 종료하여 이벤트 메시지 처리를 중지할 수 있습니다. 이렇게 하려면 구독을 여는 데 사용한 구독 작업 객체에서 `closeStream()` `close()` (Java), `Close()` (Python) 또는 (C++) 를 호출해야 합니다.

AWS IoT GreengrassCore IPC 서비스는 다음과 같은 구독 작업을 지원합니다.

- SubscribeToTopic
- SubscribeToIoTCore
- SubscribeToComponentUpdates
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates

주제

- 구독 핸들러를 정의합니다.
- 서브스크립션 핸들러 예시

구독 핸들러를 정의합니다.

구독 핸들러를 정의하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 콜백 함수를 정의하십시오. IPC client V1을 사용하는 경우 클래스에서 이러한 함수를 정의해야 합니다. 이후 버전의 Java 및 Python SDK에서 사용할 수 있는 IPC 클라이언트 V2를 사용하는 경우 구독 핸들러 클래스를 만들지 않고도 이러한 함수를 정의할 수 있습니다.

Java

IPC 클라이언트 V1을 사용하는 경우 일반 인터페이스를 구현해야 합니다. `software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<`*`StreamEventType`*`>` *`StreamEventType`*구독 작업을 위한 이벤트 메시지 유형입니다. 다음 함수를 정의하여 이벤트 메시지, 오류 및 스트림 폐쇄를 처리합니다.

IPC 클라이언트 V2를 사용하는 경우 구독 핸들러 클래스 외부에서 이러한 함수를 정의하거나 람다 식을 사용할 수 있습니다.

```
void onStreamEvent(StreamEventType event)
```

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 호출하는 콜백입니다.

`boolean onStreamError(Throwable error)`

스트림 오류가 발생할 때 IPC 클라이언트가 호출하는 콜백입니다.

오류로 인해 구독 스트림을 종료하려면 true를 반환하고, 스트림을 계속 열어 두려면 false를 반환합니다.

`void onStreamClosed()`

스트림이 종료될 때 IPC 클라이언트가 호출하는 콜백입니다.

Python

IPC 클라이언트 V1을 사용하는 경우 구독 작업에 해당하는 스트림 응답 핸들러 클래스를 확장해야 합니다. AWS IoT Device SDK여기에는 각 구독 작업에 대한 구독 핸들러 클래스가 포함됩니다. *StreamEventType*구독 작업에 대한 이벤트 메시지 유형입니다. 다음 함수를 정의하여 이벤트 메시지, 오류 및 스트림 폐쇄를 처리합니다.

IPC 클라이언트 V2를 사용하는 경우 구독 핸들러 클래스 외부에서 이러한 함수를 정의하거나 람다 식을 사용할 수 있습니다.

`def on_stream_event(self, event: StreamEventType) -> None`

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 호출하는 콜백입니다.

`def on_stream_error(self, error: Exception) -> bool`

스트림 오류가 발생할 때 IPC 클라이언트가 호출하는 콜백입니다.

오류로 인해 구독 스트림을 종료하려면 true를 반환하고, 스트림을 계속 열어 두려면 false를 반환합니다.

`def on_stream_closed(self) -> None`

스트림이 종료될 때 IPC 클라이언트가 호출하는 콜백입니다.

C++

구독 작업에 해당하는 스트림 응답 핸들러 클래스에서 파생되는 클래스를 구현하십시오.
AWS IoT Device SDK여기에는 각 구독 작업에 대한 구독 핸들러 기본 클래스가 포함됩니다. *StreamEventType*구독 작업에 대한 이벤트 메시지 유형입니다. 다음 함수를 정의하여 이벤트 메시지, 오류 및 스트림 폐쇄를 처리합니다.

`void OnStreamEvent(StreamEventType *event)`

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 호출하는 콜백입니다.

`bool OnStreamError(OperationError *error)`

스트림 오류가 발생할 때 IPC 클라이언트가 호출하는 콜백입니다.

오류로 인해 구독 스트림을 종료하려면 true를 반환하고, 스트림을 계속 열어 두려면 false를 반환합니다.

`void OnStreamClosed()`

스트림이 종료될 때 IPC 클라이언트가 호출하는 콜백입니다.

JavaScript

구독 작업에 해당하는 스트림 응답 핸들러 클래스에서 파생되는 클래스를 구현하십시오. AWS IoT Device SDK여기에는 각 구독 작업에 대한 구독 핸들러 기본 클래스가 포함됩니다. *StreamEventType*구독 작업에 대한 이벤트 메시지 유형입니다. 다음 함수를 정의하여 이벤트 메시지, 오류 및 스트림 폐쇄를 처리합니다.

`on(event: 'ended', listener: StreamingOperationEndedListener)`

스트림이 종료될 때 IPC 클라이언트가 호출하는 콜백입니다.

`on(event: 'streamError', listener: StreamingRpcErrorListener)`

스트림 오류가 발생할 때 IPC 클라이언트가 호출하는 콜백입니다.

오류로 인해 구독 스트림을 종료하려면 true를 반환하고, 스트림을 계속 열어 두려면 false를 반환합니다.

`on(event: 'message', listener: (message: InboundMessageType) => void)`

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 호출하는 콜백입니다.

서브스크립션 핸들러 예시

다음 예제는 SubscribeToTopic 작업 및 구독 핸들러를 사용하여 로컬 게시/구독 메시지를 구독하는 방법을 보여줍니다.

Java (IPC client V2)

Example 예: 로컬 게시/구독 메시지 구독

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
```

```
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example 예: 로컬 게시/구독 메시지 구독

```
import sys
```

```
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)
```

```
def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example 예: 로컬 게시/구독 메시지 구독

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
```

```
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
```

```
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }
```

```
    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 예: 로컬 게시/구독 메시지 구독

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
```

```
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

IPC 모범 사례

사용자 지정 구성 요소에서 IPC를 사용하는 모범 사례는 IPC 클라이언트 V1과 IPC 클라이언트 V2에서 다릅니다. 사용하는 IPC 클라이언트 버전의 모범 사례를 따르세요.

IPC client V2

IPC 클라이언트 V2는 별도의 스레드에서 콜백 함수를 실행하므로 IPC 클라이언트 V1에 비해 IPC를 사용하고 구독 핸들러 함수를 작성할 때 따라야 할 지침이 적습니다.

- IPC 클라이언트 하나를 재사용하십시오.

  IPC 클라이언트를 생성한 후에는 열어 두고 모든 IPC 작업에 재사용하십시오. 여러 클라이언트를 생성하면 추가 리소스가 사용되고 리소스 누수가 발생할 수 있습니다.
- 예외 처리

  IPC 클라이언트 V2는 잡히지 않은 예외를 구독 핸들러 함수에 기록합니다. 코드에서 발생하는 오류를 처리하려면 핸들러 함수에서 예외를 포착해야 합니다.

IPC client V1

IPC 클라이언트 V1은 IPC 서버와 통신하고 구독 핸들러를 호출하는 단일 스레드를 사용합니다. 구독 핸들러 함수를 작성할 때는 이 동기 동작을 고려해야 합니다.

- IPC 클라이언트 1개를 재사용하십시오.

  IPC 클라이언트를 생성한 후에는 열어 두고 모든 IPC 작업에 재사용하십시오. 여러 클라이언트를 생성하면 추가 리소스가 사용되고 리소스 누수가 발생할 수 있습니다.
- 차단 코드를 비동기적으로 실행합니다.

  IPC 클라이언트 V1은 스레드가 차단된 상태에서 새 요청을 보내거나 새 이벤트 메시지를 처리할 수 없습니다. 핸들러 함수에서 실행하는 별도의 스레드에서 차단 코드를 실행해야 합니다. 차단 코드에는 `sleep` 호출, 지속적으로 실행되는 루프, 완료하는 데 시간이 걸리는 동기 I/O 요청이 포함됩니다.
- 새 IPC 요청을 비동기적으로 전송합니다.

  응답을 기다리면 요청이 핸들러 함수를 차단하기 때문에 IPC 클라이언트 V1은 구독 핸들러 함수 내에서 새 요청을 보낼 수 없습니다. IPC 요청은 핸들러 함수에서 실행하는 별도의 스레드로 보내야 합니다.

- 예외 처리

  IPC 클라이언트 V1은 구독 핸들러 함수에서 포착되지 않은 예외를 처리하지 않습니다. 핸들러 함수에서 예외가 발생하면 구독이 종료되고 해당 예외는 구성 요소 로그에 표시되지 않습니다. 구독을 열린 상태로 유지하고 코드에서 발생하는 오류를 기록하려면 핸들러 함수에서 예외를 포착해야 합니다.

로컬 메시지 게시/구독

게시/구독 (pubsub) 메시징을 사용하면 주제에 메시지를 보내고 받을 수 있습니다. 구성 요소는 주제에 메시지를 게시하여 다른 구성 요소에 메시지를 보낼 수 있습니다. 그러면 해당 주제를 구독하는 구성 요소가 수신한 메시지에 대해 조치를 취할 수 있습니다.

> **Note**
>
> 이 게시/구독 IPC 서비스를 사용하여 MQTT를 게시하거나 구독할 수 없습니다. AWS IoT Core MQTT와 AWS IoT Core 메시지를 교환하는 방법에 대한 자세한 내용은 을 참조하십시오. MQTT 메시지 게시/구독 AWS IoT Core

주제

최소 SDK 버전

다음 표에는 로컬 주제에 대한 메시지를 게시하고 구독하는 데 사용해야 하는 최소 버전이 나와 있습니다. AWS IoT Device SDK

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.2.10	

SDK	최소 버전	
AWS IoT Device SDK파이썬 v2용	v1.5.3	
AWS IoT Device SDKC++ v2의 경우	v1.17.0	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 로컬 게시/구독 메시지를 사용하려면 구성 요소가 주제에 메시지를 보내고 받을 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오. 구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오.

게시/구독 메시징의 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ipc.pubsub`

Operation	설명	리소스
`aws.greengrass#PublishToTopic`	구성 요소가 지정한 주제에 메시지를 게시할 수 있도록 합니다.	주제 문자열 (예:`test/topic` . 주제에 있는 모든 문자 조합을 * 일치시키려면 an을 사용하십시오. 이 주제 문자열은 MQTT 주제 와일드카드 (#및+) 를 지원하지 않습니다.
`aws.greengrass#SubscribeToTopic`	구성 요소가 지정한 주제에 대한 메시지를 구독할 수 있도록 합니다.	주제 문자열 (예:`test/topic` . 주제에 있는 모든 문자 조합을 * 일치시키려면 an을 사용하십시오.

Operation	설명	리소스
		Greengrass nucleus v2.6.0 이상에서는 MQTT 주제 와일드카드 (및) 가 포함된 주제를 구독할 수 있습니다. # + 이 주제 문자열은 MQTT 주제 와일드카드를 리터럴 문자로 지원합니다. 예를 들어 구성 요소의 권한 부여 정책이 액세스 권한을 부여하면 구성 요소가 구독할 수는 있지만 구독할 `test/topic/#` 수는 없습니다. `test/topic/#` `test/topic/filter`
*	구성 요소가 지정한 주제에 대한 메시지를 게시하고 구독할 수 있도록 합니다.	주제 문자열 (예:`test/topic` . 주제에 있는 모든 문자 조합을 * 일치시키려면 an을 사용하십시오. Greengrass nucleus v2.6.0 이상에서는 MQTT 주제 와일드카드 (및) 가 포함된 주제를 구독할 수 있습니다. # + 이 주제 문자열은 MQTT 주제 와일드카드를 리터럴 문자로 지원합니다. 예를 들어 구성 요소의 권한 부여 정책이 액세스 권한을 부여하면 구성 요소가 구독할 수는 있지만 구독할 `test/topic/#` 수는 없습니다. `test/topic/#` `test/topic/filter`

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 권한 부여 정책 예시

다음 예제 권한 부여 정책은 구성 요소가 모든 주제를 게시하고 구독하도록 허용합니다.

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyLocalPubSubComponent:pubsub:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToTopic",
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

PublishToTopic

주제에 메시지를 게시합니다.

요청

이 작업의 요청에는 다음과 같은 매개 변수가 있습니다.

`topic`

메시지를 게시할 주제.

`publishMessage`(Python:`publish_message`)

게시할 메시지. 이 개체`PublishMessage`, 에는 다음 정보가 들어 있습니다. `jsonMessage`및 중 하나를 지정해야 `binaryMessage` 합니다.

`jsonMessage`(Python:`json_message`)

(선택 사항) JSON 메시지입니다. 이 객체`JsonMessage`, 에는 다음 정보가 들어 있습니다.

`message`

객체로서의 JSON 메시지.

`context`

메시지 컨텍스트 (예: 메시지가 게시된 주제).

이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDKJava v2의 경우	v1.9.3
AWS IoT Device SDK파이썬 v2용	v1.11.3
AWS IoT Device SDKC++ v2의 경우	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어는 및 연산에서 동일한 `PublishToTopic` 메시지 객체를 사용합니다. `SubscribeToTopic` AWS IoT GreengrassCore 소프트웨어는 구독할 때 메시지에 이 컨텍스트 개체를 설정하고 사용자가 게시하는 메시지에서는 이 컨텍스트 개체를 무시합니다.

이 객체에는 다음 정보가 들어 `MessageContext` 있습니다.

`topic`

메시지가 게시된 주제.

`binaryMessage`(Python:`binary_message`)

(선택 사항) 바이너리 메시지. 이 객체`BinaryMessage`, 에는 다음 정보가 들어 있습니다.

`message`

블롭으로서의 바이너리 메시지.

`context`

메시지 컨텍스트 (예: 메시지가 게시된 주제)

이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.9.3	
AWS IoT Device SDK파이썬 v2용	v1.11.3	
AWS IoT Device SDKC++ v2의 경우	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어는 및 연산에서 동일한 `PublishToTopic` 메시지 객체를 사용합니다. `SubscribeToTopic` AWS IoT GreengrassCore 소프트웨어는 구독할 때 메시지에 이 컨텍스트 개체를 설정하고 사용자가 게시하는 메시지에서는 이 컨텍스트 개체를 무시합니다.

이 객체에는 다음 정보가 들어 `MessageContext` 있습니다.

`topic`

메시지가 게시된 주제.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example 예: 바이너리 메시지 게시

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
 message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
```

```
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
            GreengrassCoreIPCClientV2 ipcClient, String topic, String message)
 throws InterruptedException {
        BinaryMessage binaryMessage =
                new
 BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
 PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
                new
 PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}
```

Python (IPC client V2)

Example 예: 바이너리 메시지 게시

```
import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)


def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
```

```
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def publish_binary_message_to_topic(ipc_client, topic, message):
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
 publish_message=publish_message)


if __name__ == '__main__':
    main()
```

C++

Example 예: 바이너리 메시지 게시

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
```

```
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    String message("Hello, World!");
    int timeout = 10;

    PublishToTopicRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    BinaryMessage binaryMessage;
    binaryMessage.SetMessage(messageData);
    PublishMessage publishMessage;
    publishMessage.SetBinaryMessage(binaryMessage);
    request.SetTopic(topic);
    request.SetPublishMessage(publishMessage);

    auto operation = ipcClient.NewPublishToTopic();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
```

```
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }
    return 0;
}
```

JavaScript

Example 예: 바이너리 메시지 게시

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";

class PublishToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;
    private readonly messageString : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.messageString = "<define_your_message_string>";
        this.publishToTopic().then(r => console.log("Started workflow"));
    }

    private async publishToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const binaryMessage : BinaryMessage = {
                message: this.messageString
            }

            const publishMessage : PublishMessage = {
                binaryMessage: binaryMessage
            }

            const request : PublishToTopicRequest = {
```

```
                topic: this.topic,
                publishMessage: publishMessage
            }

            this.ipcClient.publishToTopic(request).finally(() =>
 console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))

        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();
```

SubscribeToTopic

주제에 관한 메시지를 구독하세요.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 함수가 포함된 스트림 응답 핸들러를 정의하십시오. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

이벤트 메시지 유형: `SubscriptionResponseMessage`

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`topic`

구독할 주제.

> **Note**
>
> Greengrass nucleus v2.6.0 이상에서 이 항목은 MQTT 주제 와일드카드 (및) 를 지원합니다. # +

`receiveMode`(Python:`receive_mode`)

(선택 사항) 구성 요소가 자체로부터 메시지를 수신할지 여부를 지정하는 동작입니다. 구성 요소가 자체 메시지에 대해 작동하도록 이 동작을 변경할 수 있습니다. 기본 동작은 주제에 MQTT 와일드카드가 포함되어 있는지 여부에 따라 달라집니다. 다음 옵션 중 하나를 선택합니다.

- `RECEIVE_ALL_MESSAGES`— 구독하는 구성 요소의 메시지를 포함하여 주제와 일치하는 모든 메시지를 수신합니다.

  이 모드는 MQTT 와일드카드가 포함되지 않은 주제를 구독할 때의 기본 옵션입니다.
- `RECEIVE_MESSAGES_FROM_OTHERS`— 구독하는 구성 요소의 메시지를 제외하고 주제와 일치하는 모든 메시지를 수신합니다.

  이 모드는 MQTT 와일드카드가 포함된 주제를 구독할 때의 기본 옵션입니다.

이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. 다음 표에는 수신 모드를 설정하는 데 사용해야 AWS IoT Device SDK 하는 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.9.3	
AWS IoT Device SDK파이썬 v2용	v1.11.3	

SDK	최소 버전
AWS IoT Device SDKC++ v2의 경우	v1.18.4
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

메시지 스트림. 이 개체`SubscriptionResponseMessage`, 에는 다음 정보가 들어 있습니다. 각 메시지에는 `jsonMessage` 또는 이 포함되어 `binaryMessage` 있습니다.

`jsonMessage`(Python:`json_message`)

(선택 사항) JSON 메시지입니다. 이 객체`JsonMessage`, 에는 다음 정보가 들어 있습니다.

`message`

객체로서의 JSON 메시지.

`context`

메시지 컨텍스트 (예: 메시지가 게시된 주제).

이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDKJava v2의 경우	v1.9.3
AWS IoT Device SDK파이썬 v2용	v1.11.3

SDK	최소 버전	
AWS IoT Device SDKC++ v2의 경우	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어는 및 연산에서 동일한 `PublishToTopic` 메시지 객체를 사용합니다. `SubscribeToTopic` AWS IoT GreengrassCore 소프트웨어는 구독할 때 메시지에 이 컨텍스트 개체를 설정하고 사용자가 게시하는 메시지에서는 이 컨텍스트 개체를 무시합니다.

이 객체에는 다음 정보가 들어 `MessageContext` 있습니다.

`topic`

메시지가 게시된 주제.

`binaryMessage`(Python:`binary_message`)

(선택 사항) 바이너리 메시지. 이 객체`BinaryMessage`, 에는 다음 정보가 들어 있습니다.

`message`

블롭으로서의 바이너리 메시지.

`context`

메시지 컨텍스트 (예: 메시지가 게시된 주제)

이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDKJava v2의 경우	v1.9.3
AWS IoT Device SDK파이썬 v2용	v1.11.3
AWS IoT Device SDKC++ v2의 경우	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

> **Note**
>
> AWS IoT GreengrassCore 소프트웨어는 및 연산에서 동일한 `PublishToTopic` 메시지 객체를 사용합니다. `SubscribeToTopic` AWS IoT GreengrassCore 소프트웨어는 구독할 때 메시지에 이 컨텍스트 개체를 설정하고 사용자가 게시하는 메시지에서는 이 컨텍스트 개체를 무시합니다.

이 객체에는 다음 정보가 들어 `MessageContext` 있습니다.

`topic`

메시지가 게시된 주제.

`topicName`(Python:`topic_name`)

메시지가 게시된 주제.

> **Note**
>
> 이 속성은 현재 사용되지 않습니다. Greengrass nucleus v2.6.0 이상에서는 `SubscriptionResponseMessage` a에서 `(jsonMessage|binaryMessage).context.topic` 값을 가져와 메시지가 게시된 주제를 가져올 수 있습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example 예: 로컬 게시/구독 메시지 구독

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
```

```
            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example 예: 로컬 게시/구독 메시지 구독

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
```

```
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example 예: 로컬 게시/구독 메시지 구독

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
```

```
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};
```

```
int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
```

```
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 예: 로컬 게시/구독 메시지 구독

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options
```

```
            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}
```

```
// starting point
const subscribeToTopic = new SubscribeToTopic();
```

예제

다음 예를 사용하여 구성 요소에서 게시/구독 IPC 서비스를 사용하는 방법을 알아보십시오.

게시/구독 게시자 예제 (Java, IPC 클라이언트 V1)

다음 예제 레시피를 사용하면 구성 요소를 모든 주제에 게시할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherJava:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubPublisher.jar"
      }
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar
```

다음 예제 Java 애플리케이션은 게시/구독 IPC 서비스를 사용하여 메시지를 다른 구성 요소에 게시하는 방법을 보여줍니다.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
```

```
import java.util.concurrent.TimeoutException;

public class PubSubPublisher {

    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                        .publishToTopic(publishRequest,
 Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
                    System.err.println("Timeout occurred while publishing to topic: " +
 topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
 topic: " + topic);
                    }
                    throw e;
                }
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
```

```
            System.out.println("Publisher interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

게시/구독 구독자 예제 (Java, IPC 클라이언트 V1)

다음 예제 레시피를 사용하면 구성 요소가 모든 주제를 구독할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
      }
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

다음 예제 Java 애플리케이션은 게시/구독 IPC 서비스를 사용하여 다른 구성 요소에 대한 메시지를 구독하는 방법을 보여줍니다.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                    .subscribeToTopic(subscribeRequest, Optional.of(new
 SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
 operationResponseHandler.getResponse();

            try {
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
 " + topic);
                } else {
                    System.err.println("Execution exception while subscribing to topic:
 " + topic);
                }
                throw e;
            }
```

```
            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static class SubscribeResponseHandler implements
StreamResponseHandler<SubscriptionResponseMessage> {

        @Override
        public void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
            try {
                String message = new
String(subscriptionResponseMessage.getBinaryMessage()
                        .getMessage(), StandardCharsets.UTF_8);
                System.out.println("Received new message: " + message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false; // Return true to close stream, false to keep stream open.
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to topic stream closed.");
        }
    }
```

```
}
```

게시/구독 게시자 예제 (Python, IPC 클라이언트 V1)

다음 예제 레시피를 사용하면 구성 요소를 모든 주제에 게시할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_publisher.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
```

```
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_publisher.py
```

다음 예제 Python 애플리케이션은 게시/구독 IPC 서비스를 사용하여 메시지를 다른 구성 요소에 게시하는 방법을 보여줍니다.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)


topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
 file=sys.stderr)
       except UnauthorizedError as e:
            print('Unauthorized error while publishing to topic: ' + topic,
 file=sys.stderr)
            raise e
       except Exception as e:
```

```
            print('Exception while publishing to topic: ' + topic, file=sys.stderr)
            raise e
        time.sleep(5)
except InterruptedError:
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

게시/구독 구독자 예제 (Python, IPC 클라이언트 V1)

다음 예제 레시피를 사용하면 구성 요소가 모든 주제를 구독할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
```

```
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_subscriber.py
  - Platform:
      os: windows
```

```
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_subscriber.py
```

다음 예제 Python 애플리케이션은 게시/subscribe IPC 서비스를 사용하여 다른 구성 요소에 대한 메시지를 구독하는 방법을 보여줍니다.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10


class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()
        return False  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        print('Subscribe to topic stream closed.')
```

```
try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = SubscribeToTopicRequest()
    request.topic = topic
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

게시/구독 게시자 예제 (C++)

다음 예제 레시피를 사용하면 구성 요소를 모든 주제에 게시할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.PubSubPublisherCpp/1.0.0/
greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

다음 예제 C++ 애플리케이션은 게시/구독 IPC 서비스를 사용하여 메시지를 다른 구성 요소에 게시하는 방법을 보여줍니다.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }
```

```
    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();
```

```
        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

게시/구독 구독자 (C++) 예시

다음 예제 레시피를 사용하면 구성 요소가 모든 주제를 구독할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
```

```
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
```

```
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

다음 예제 C++ 애플리케이션은 게시/구독 IPC 서비스를 사용하여 다른 구성 요소에 대한 메시지를 구독하는 방법을 보여줍니다.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                std::cout << "Received new message: " << messageString << std::endl;
            } else {
```

```
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    std::cout << "Received new message: " << messageString <<
 std::endl;
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to topic stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
```

```
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
```

```
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

MQTT 메시지 게시/구독 AWS IoT Core

AWS IoT CoreMQTT 메시징 IPC 서비스를 사용하면 MQTT 메시지를 주고 받을 수 있습니다. AWS IoT Core 구성 요소는 메시지를 AWS IoT Core 게시하고 주제에 구독하여 다른 소스의 MQTT 메시지를 처리할 수 있습니다. MQTT AWS IoT Core 구현에 대한 자세한 내용은 개발자 안내서의 MQTT를 AWS IoT Core 참조하십시오.

> **Note**
>
> 이 MQTT 메시징 IPC 서비스를 사용하면 메시지를 교환할 수 있습니다. AWS IoT Core 구성 요소 간에 메시지를 교환하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독

주제

최소 SDK 버전

다음 표에는 MQTT 메시지를 게시하고 MQTT 메시지를 구독하는 데 사용해야 하는 최소 버전이 나와 있습니다. AWS IoT Device SDK AWS IoT Core

SDK	최소 버전	
AWS IoT Device SDK자바 v2의 경우	v1.2.10	
AWS IoT Device SDK파이썬 v2용	v1.5.3	
AWS IoT Device SDKC++ v2의 경우	v1.17.0	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 AWS IoT Core MQTT 메시징을 사용하려면 구성 요소가 주제에 대한 메시지를 보내고 받을 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오..

AWS IoT CoreMQTT 메시징의 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ipc.mqttproxy`

Operation	설명	리소스
`aws.greengrass#PublishToIoTCore`	구성 요소가 지정한 MQTT 주제에 AWS IoT Core 대해 메시지를 게시할 수 있도록 합니다.	주제 문자열 (예`test/topic` : 모든 주제에 대한 * 액세스를 허용하는 또는) MQTT 주제 와일드카드 (#및+) 를 사용하여 여러 리소스를 매칭할 수 있습니다.
`aws.greengrass#SubscribeToIoTCore`	구성 요소가 지정한 주제의 메시지를 구독할 수 있도록 합니다. AWS IoT Core	주제 문자열 (예`test/topic` : 모든 주제에 대한 * 액세스를 허용하는 또는) MQTT 주제 와일드카드 (#및+) 를 사용하여

Operation	설명	리소스
		여러 리소스를 매칭할 수 있습니다.
*	구성 요소가 지정한 주제에 대한 AWS IoT Core MQTT 메시지를 게시하고 구독할 수 있도록 합니다.	모든 주제에 대한 * 액세스를 허용하는 또는 와 `test/topic` 같은 주제 문자열. MQTT 주제 와일드카드 (#및+) 를 사용하여 여러 리소스를 매칭할 수 있습니다.

MQTT 권한 부여 정책의 MQTT 와일드카드 AWS IoT Core

MQTT IPC 권한 부여 정책에서 AWS IoT Core MQTT 와일드카드를 사용할 수 있습니다. 구성 요소는 권한 부여 정책에서 허용하는 주제 필터와 일치하는 주제를 게시하고 구독할 수 있습니다. 예를 들어 구성 요소의 권한 부여 정책이 액세스 권한을 부여하는 `test/topic/#` 경우 구성 요소는 구독할 `test/topic/#` 수 있고 게시 및 구독할 수 있습니다`test/topic/filter`.

AWS IoT CoreMQTT 권한 부여 정책의 레시피 변수

v2.6.0 이상의 Greengrass 핵을 사용하는 경우 권한 부여 정책에서 레시피 변수를 사용할 수 있습니다. `{iot:thingName}` 이 기능을 사용하면 코어 장치 그룹에 대해 단일 권한 부여 정책을 구성하여 각 코어 장치가 고유한 이름을 포함하는 항목에만 액세스할 수 있습니다. 예를 들어, 구성 요소가 다음 주제 리소스에 액세스하도록 허용할 수 있습니다.

```
devices/{iot:thingName}/messages
```

자세한 내용은 레시피 변수 및 병합 업데이트에 레시피 변수를 사용하십시오. 섹션을 참조하세요.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 무제한 액세스가 포함된 권한 부여 정책 예시

다음 예제 권한 부여 정책은 구성 요소가 모든 주제를 게시하고 구독할 수 있도록 허용합니다.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - "*"
```

Example 액세스가 제한된 권한 부여 정책의 예

다음 예제 권한 부여 정책은 구성 요소가 `factory/1/events` 및 라는 두 개의 주제를 게시하고 구독할 수 있도록 허용합니다`factory/1/actions`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
```

```
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1
 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/actions
        - factory/1/events
```

Example 핵심 장치 그룹에 대한 권한 부여 정책 예시

> **Important**
>
> 이 예제에서는 v2.6.0 이상에서 사용할 수 있는 Greengrass 핵 구성 요소 기능을 사용합니다. Greengrass nucleus v2.6.0은 대부분의 레시피 변수 (예: 구성 요소 구성) 에 대한 지원을 추가합니다. `{iot:thingName}`

다음 예제 권한 부여 정책은 구성 요소가 구성 요소를 실행하는 핵심 장치의 이름이 포함된 주제를 게시하고 구독할 수 있도록 허용합니다.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/devices/{iot:thingName}/controls
```

PublishToIoTCore

주제에 대한 MQTT 메시지를 게시합니다AWS IoT Core.

MQTT 메시지를 에 게시하는 AWS IoT Core 경우 초당 100개의 트랜잭션 할당량이 설정됩니다. 이 할당량을 초과하면 Greengrass 장치에서 메시지가 처리를 위해 대기열에 추가됩니다. 또한 초당 512Kb의 데이터 할당량과 계정 전체 할당량은 초당 20,000건의 게시 (일부 경우 2,000건) 입니다. AWS 리전 MQTT 메시지 브로커 한도에 대한 자세한 내용은 메시지 브로커 및 AWS IoT Core 프로토콜 제한 [및 AWS IoT Core 할당량을] 참조하십시오.

이 할당량을 초과할 경우 Greengrass 디바이스는 메시지 게시를 로 제한합니다. AWS IoT Core 메시지는 스풀러의 메모리에 저장됩니다. 기본적으로 스풀러에 할당된 메모리는 2.5Mb입니다. 스풀러가 가득 차면 새 메시지가 거부됩니다. 스풀러의 크기를 늘릴 수 있습니다. 자세한 내용은 그린그래스 핵 설명서의 구성을(를) 참조하세요. 스풀러가 가득 차서 할당된 메모리를 늘릴 필요가 없도록 게시 요청을 초당 100개 이하로 제한하십시오.

애플리케이션에서 더 빠른 속도로 또는 더 큰 메시지를 전송해야 하는 경우 를 사용하여 Kinesis Data Streams로 메시지를 보내는 것을 고려해 보십시오. 스트림 관리자 스트림 관리자 구성 요소는 대용량 데이터를 로 전송하도록 설계되었습니다. AWS 클라우드 자세한 설명은 Greengrass 코어 디바이스의 데이터 스트림 관리 섹션을 참조하세요.

요청

이 작업의 요청에는 다음과 같은 매개 변수가 있습니다.

`topicName`(Python:`topic_name`)

메시지를 게시할 주제.

`qos`

사용할 MQTT QoS입니다. 이 열거형의 값은 `QOS` 다음과 같습니다.

- `AT_MOST_ONCE`— QoS 0. MQTT 메시지는 최대 한 번 전송됩니다.
- `AT_LEAST_ONCE`— QoS 1. MQTT 메시지가 한 번 이상 전달되었습니다.

`payload`

(선택 사항) 블럽 형태의 메시지 페이로드.

MQTT 5를 사용하는 경우 v2.10.0 이상에서 다음 기능을 사용할 수 있습니다. 그린그래스 핵 MQTT 3.1.1을 사용할 때는 이러한 기능이 무시됩니다. 다음 표에는 이러한 기능에 액세스하는 데 사용해야 하는 AWS IoT 기기 SDK의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24.0

SDK	최소 버전
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

(선택 사항) 메시지 페이로드의 형식. 를 설정하지 않으면 유형이 다음과 `payloadFormat` 같은 것으로 간주됩니다. BYTES 열거형의 값은 다음과 같습니다.

- BYTES— 페이로드의 내용은 바이너리 블롭입니다.
- UTF8— 페이로드의 내용은 UTF8 문자열입니다.

`retain`

(선택 사항) 게시할 때 MQTT 보존 옵션을 로 설정할지 여부를 나타냅니다. `true`

`userProperties`

(선택 사항) 전송할 애플리케이션별 `UserProperty` 객체 목록. `UserProperty`객체는 다음과 같이 정의됩니다.

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(선택 사항) 메시지가 만료되어 서버에서 삭제되기까지의 시간 (초). 이 값을 설정하지 않으면 메시지가 만료되지 않습니다.

`correlationData`

(선택 사항) 요청을 응답과 연결하는 데 사용할 수 있는 요청에 추가된 정보입니다.

`responseTopic`

(선택 사항) 응답 메시지에 사용해야 하는 주제입니다.

`contentType`

(선택 사항) 메시지 콘텐츠 유형의 애플리케이션별 식별자.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example 예: 메시지 게시

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                    .withTopicName(topic)
                    .withPayload(message.getBytes(StandardCharsets.UTF_8))
                    .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example 예: 메시지 게시

> **Note**
>
> 이 예제에서는 AWS IoT Device SDK Python v2용 버전 1.5.4 이상을 사용하고 있다고 가정합니다.

```
import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()
```

Java (IPC client V1)

Example 예: 메시지 게시

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            PublishToIoTCoreResponseHandler responseHandler =
                    PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
 message, qos);
            CompletableFuture<PublishToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully published to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while publishing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
```

```
            System.exit(1);
        }
    }

    public static PublishToIoTCoreResponseHandler
 publishBinaryMessageToTopic(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
 PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

 publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCClient.publishToIoTCore(publishToIoTCoreRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 예: 메시지 게시

> **Note**
>
> 이 예제에서는 AWS IoT Device SDK Python v2용 버전 1.5.4 이상을 사용하고 있다고 가정합니다.

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE
```

```
request = PublishToIoTCoreRequest()
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++

Example 예: 메시지 게시

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
```

```
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
```

```
}
```

JavaScript

Example 예: 메시지 게시

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
    }

    private async publishToIoTCore() {
        try {
            const request: PublishToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
```

```
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```

SubscribeToIoTCore

주제 또는 주제 AWS IoT Core 필터에서 MQTT 메시지를 구독하십시오. AWS IoT GreengrassCore 소프트웨어는 구성 요소의 수명 주기가 끝나면 구독을 제거합니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 함수가 포함된 스트림 응답 핸들러를 정의하십시오. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

이벤트 메시지 유형: `IoTCoreMessage`

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`topicName`(Python:`topic_name`)

구독하고 싶은 주제. MQTT 주제 와일드카드 (`#`및`+`) 를 사용하여 여러 주제를 구독할 수 있습니다.

`qos`

사용할 MQTT QoS입니다. 이 열거형의 값은 `QOS` 다음과 같습니다.

- `AT_MOST_ONCE`— QoS 0. MQTT 메시지는 최대 한 번 전송됩니다.
- `AT_LEAST_ONCE`— QoS 1. MQTT 메시지가 한 번 이상 전달되었습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

MQTT 메시지 스트림. 이 객체에는 다음 정보가 들어 `IoTCoreMessage` 있습니다.

`message`

MQTT 메시지입니다. 이 개체`MQTTMessage`, 에는 다음 정보가 들어 있습니다.

`topicName`(Python:`topic_name`)

메시지가 게시된 주제.

`payload`

(선택 사항) 블럽 형태의 메시지 페이로드입니다.

MQTT 5를 사용하는 경우 v2.10.0 이상에서 다음 기능을 사용할 수 있습니다. 그린그래스 핵 MQTT 3.1.1을 사용할 때는 이러한 기능이 무시됩니다. 다음 표에는 이러한 기능에 액세스하는 데 사용해야 하는 AWS IoT 기기 SDK의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24.0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

(선택 사항) 메시지 페이로드의 형식. 를 설정하지 않으면 유형이 다음과 `payloadFormat` 같은 것으로 간주됩니다. `BYTES` 열거형의 값은 다음과 같습니다.

- `BYTES`— 페이로드의 내용은 바이너리 블롭입니다.
- `UTF8`— 페이로드의 내용은 UTF8 문자열입니다.

`retain`

(선택 사항) 게시할 때 MQTT 보존 옵션을 로 설정할지 여부를 나타냅니다. `true`

`userProperties`

(선택 사항) 전송할 애플리케이션별 `UserProperty` 객체 목록. `UserProperty`객체는 다음과 같이 정의됩니다.

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(선택 사항) 메시지가 만료되어 서버에서 삭제되기까지의 시간 (초). 이 값을 설정하지 않으면 메시지가 만료되지 않습니다.

`correlationData`

(선택 사항) 요청을 응답과 연결하는 데 사용할 수 있는 요청에 추가된 정보입니다.

`responseTopic`

(선택 사항) 응답 메시지에 사용해야 하는 주제입니다.

`contentType`

(선택 사항) 메시지 콘텐츠 유형의 애플리케이션별 식별자입니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example 예: 메시지 구독

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
```

```
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.function.Consumer;
import java.util.function.Function;


public class SubscribeToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = ioTCoreMessage ->
                System.out.printf("Received new message on topic %s: %s%n",
                        ioTCoreMessage.getMessage().getTopicName(),
                        new String(ioTCoreMessage.getMessage().getPayload(),
 StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
                Optional.of(e -> {
                    System.err.println("Received a stream error.");
                    e.printStackTrace();
                    return false;
                });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
                System.out.println("Subscribe to IoT Core stream closed."));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                    .withTopicName(topic)
                    .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
 SubscribeToIoTCoreResponseHandler>
                    streamingResponse = ipcClientV2.subscribeToIoTCore(request,
 onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);
```

```
            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }

            // To stop subscribing, close the stream.
            streamingResponse.getHandler().closeStream();
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example 예: 메시지 구독

> **Note**
>
> 이 예제에서는 AWS IoT Device SDK Python v2용 버전 1.5.4 이상을 사용하고 있다고 가정합니다.

```
import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'

def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}:  {message}')
    except:
        traceback.print_exc()
```

```
def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():
    pass

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()
```

Java (IPC client V1)

Example 예: 메시지 구독

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
```

```
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                    new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                    SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                            streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }

            // Keep the main thread alive, or the process will exit.
```

```
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
                Optional.of(streamResponseHandler));
    }

    public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

        @Override
        public void onStreamEvent(IoTCoreMessage ioTCoreMessage) {
            try {
                String topic = ioTCoreMessage.getMessage().getTopicName();
                String message = new
String(ioTCoreMessage.getMessage().getPayload(),
                        StandardCharsets.UTF_8);
                System.out.printf("Received new message on topic %s: %s%n", topic,
message);
            } catch (Exception e) {
```

```
                System.err.println("Exception occurred while processing subscription
 response " +
                        "message.");
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false;
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to IoT Core stream closed.");
        }
    }
}
```

Python (IPC client V1)

Example 예: 메시지 구독

> **Note**
>
> 이 예제에서는 AWS IoT Device SDK Python v2용 버전 1.5.4 이상을 사용하고 있다고 가정합니다.

```
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10
```

```
ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: IoTCoreMessage) -> None:
        try:
            message = str(event.message.payload, "utf-8")
            topic_name = event.message.topic_name
            # Handle message.
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        # Handle error.
        return True  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        # Handle close.
        pass


topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)

# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()
```

C++

Example 예: 메시지 구독

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:
        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string topicName =
 message.value().GetTopicName().value().c_str();
                // Handle message.
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }
```

```
    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }
```

```
    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 예: 메시지 구독

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/
dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
```

```
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            const streamingOperation = this.ipcClient.subscribeToIoTCore(request);

            streamingOperation.on('message', (message: IoTCoreMessage) => {
                // parse the message depending on your use cases, e.g.
                if (message.message && message.message.payload) {
                    const receivedMessage = message.message.payload.toString();
                }
            });

            streamingOperation.on('streamError', (error : RpcError) => {
                // define your own error handling logic
            });

            streamingOperation.on('ended', () => {
                // define your own logic
            });

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
```

```
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToIoTCore = new SubscribeToIoTCore();
```

예제

다음 예제를 사용하여 구성 요소에서 AWS IoT Core MQTT IPC 서비스를 사용하는 방법을 알아보십시오.

예제 AWS IoT Core MQTT 퍼블리셔 (C++)

다음 예제 레시피를 사용하면 구성 요소를 모든 주제에 게시할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
```

```
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

다음 예제 C++ 애플리케이션은 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 게시하는 방법을 보여줍니다. AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
```

```
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);

        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }
```

```
        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

AWS IoT CoreMQTT 구독자 예제 (C++)

다음 예제 레시피를 사용하면 구성 요소가 모든 주제를 구독할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
          Execute: OWNER
```

다음 예제 C++ 애플리케이션은 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 구독하는 방법을 보여줍니다. AWS IoT Core

```
#include <iostream>
```

```
#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:

        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string messageTopic =
 message.value().GetTopicName().value().c_str();
                std::cout << "Received new message on topic: " << messageTopic <<
 std::endl;
                std::cout << "Message: " << messageString << std::endl;
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to IoT Core stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
```

```
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
```

```
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

구성 요소 라이프사이클과 상호 작용

구성 요소 수명 주기 IPC 서비스를 사용하여 다음을 수행할 수 있습니다.

- 코어 기기의 구성 요소 상태를 업데이트합니다.
- 구성 요소 상태 업데이트를 구독하십시오.
- 배포 중에 업데이트를 적용하기 위해 NUCLEUS가 구성 요소를 중지하는 것을 방지하십시오.
- 구성 요소 프로세스를 일시 중지하고 다시 시작합니다.

주제

- 최소 SDK 버전

- 권한 부여
- UpdateState
- SubscribeToComponentUpdates
- DeferComponentUpdate
- PauseComponent
- ResumeComponent

최소 SDK 버전

다음 표에는 구성 요소 수명 주기와 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDKJava v2의 경우	v1.2.10
AWS IoT Device SDK파이썬 v2용	v1.5.3
AWS IoT Device SDKC++ v2의 경우	v1.17.0
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0

권한 부여

사용자 지정 구성 요소의 다른 구성 요소를 일시 중지하거나 다시 시작하려면 구성 요소가 다른 구성 요소를 관리할 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오..

구성 요소 수명 주기 관리를 위한 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ipc.lifecycle`

Operation	설명	리소스
aws.greengrass#PauseComponent	구성 요소가 지정한 구성 요소를 일시 중지할 수 있도록 합니다.	구성 요소 이름 또는 모든 구성 요소에 대한 액세스를 허용하는 * 데 사용할 수 있습니다.
aws.greengrass#ResumeComponent	구성 요소가 지정한 구성 요소를 재개할 수 있도록 합니다.	구성 요소 이름 또는 모든 구성 요소에 대한 액세스를 허용하는 * 데 사용됩니다.
*	지정한 구성 요소를 구성 요소가 일시 중지했다가 다시 시작할 수 있습니다.	구성 요소 이름 또는 모든 구성 요소에 대한 액세스를 허용하는 * 데 사용할 수 있습니다.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 권한 부여 정책 예시

다음 예제 권한 부여 정책은 구성 요소가 모든 구성 요소를 일시 중지하고 재개할 수 있도록 허용합니다.

```
{
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

UpdateState

코어 디바이스의 구성 요소 상태를 업데이트합니다.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`state`

설정할 상태입니다. 이 열거형`LifecycleState`, 의 값은 다음과 같습니다.

- `RUNNING`
- `ERRORED`

응답

이 연산은 응답에 어떠한 정보도 제공하지 않습니다.

SubscribeToComponentUpdates

구독하면 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 업데이트하기 전에 알림을 받을 수 있습니다. 알림은 업데이트의 일부로 Nucleus를 다시 시작할지 여부를 지정합니다.

Nucleus는 배포의 구성 요소 업데이트 정책에서 구성 요소에 알리도록 지정한 경우에만 업데이트 알림을 보냅니다. 기본 동작은 구성 요소에 알리는 것입니다. 자세한 내용은 배포 만들기 및 CreateDeployment작업을 호출할 때 제공할 수 있는 DeploymentComponentUpdatePolicy객체를 참조하십시오.

> **Important**
>
> 로컬 배포에서는 업데이트 전에 구성 요소에 알리지 않습니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 함수가 포함된 스트림 응답 핸들러를 정의하십시오. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

이벤트 메시지 유형: `ComponentUpdatePolicyEvents`

> **Tip**
>
> 자습서를 따라 구성 요소 업데이트를 조건부로 연기하는 구성 요소를 개발하는 방법을 배울 수 있습니다. 자세한 설명은 튜토리얼: 구성 요소 업데이트를 연기하는 Greengrass 구성 요소 개발 섹션을 참조하세요.

요청

이 작업의 요청에는 매개변수가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

알림 메시지 스트림. 이 개체`ComponentUpdatePolicyEvents`, 에는 다음 정보가 들어 있습니다.

`preUpdateEvent`(Python:`pre_update_event`)

(선택 사항) 핵이 구성 요소를 업데이트하려고 한다는 것을 나타내는 이벤트입니다. DeferComponentUpdate작업에 응답하여 구성 요소를 다시 시작할 준비가 될 때까지 업데이트를 승인하거나 연기할 수 있습니다. 이 개체에는 다음 정보가 `PreComponentUpdateEvent` 들어 있습니다.

`deploymentId`(Python:`deployment_id`)

컴포넌트를 업데이트하는 AWS IoT Greengrass 디플로이먼트의 ID.

`isGgcRestarting`(Python:`is_ggc_restarting`)

업데이트를 적용하기 위해 Nucleus를 다시 시작해야 하는지 여부.

`postUpdateEvent`(Python:`post_update_event`)

(선택 사항) 핵이 구성 요소를 업데이트했음을 나타내는 이벤트입니다. 이 객체에는 다음 정보가 들어 `PostComponentUpdateEvent` 있습니다.

`deploymentId`(Python:`deployment_id`)

컴포넌트를 업데이트한 AWS IoT Greengrass 디플로이먼트의 ID.

> **Note**
>
> 이 기능을 사용하려면 v2.7.0 이상의 Greengrass 핵 구성 요소가 필요합니다.

DeferComponentUpdate

발견한 구성 요소 업데이트를 승인하거나 연기하십시오. SubscribeToComponentUpdates NUCLEUS가 구성 요소 업데이트를 진행할 준비가 되었는지 구성 요소가 다시 확인될 때까지 기다릴 시간을 지정합니다. 이 작업을 사용하여 Nucleus에 구성 요소가 업데이트할 준비가 되었음을 알릴 수도 있습니다.

구성 요소가 구성 요소 업데이트 알림에 응답하지 않는 경우 NUCLEUS는 배포의 구성 요소 업데이트 정책에 지정된 시간만큼 기다립니다. 제한 시간이 지나면 NUCLEUS는 배포를 진행합니다. 기본 구성 요소 업데이트 제한 시간은 60초입니다. 자세한 내용은 CreateDeployment작업을 호출할 때 제공할 수 있는 DeploymentComponentUpdatePolicy객체를 참조하십시오배포 만들기.

> **Tip**
>
> 자습서를 따라 구성 요소 업데이트를 조건부로 연기하는 구성 요소를 개발하는 방법을 배울 수 있습니다. 자세한 설명은 튜토리얼: 구성 요소 업데이트를 연기하는 Greengrass 구성 요소 개발 섹션을 참조하세요.

요청

이 작업의 요청에는 다음과 같은 파라미터가 포함됩니다.

`deploymentId`(Python:`deployment_id`)

연기할 AWS IoT Greengrass 배포의 ID.

`message`

(선택 사항) 업데이트를 연기할 구성 요소의 이름.

요청을 보내는 구성 요소의 이름이 기본값입니다.

`recheckAfterMs`(Python:`recheck_after_ms`)

업데이트를 연기하는 데 걸리는 시간 (밀리초). 핵은 이 시간 동안 기다린 다음 발견할 수 `PreComponentUpdateEvent` 있는 다른 시간을 보냅니다. SubscribeToComponentUpdates

업데이트를 `0` 승인하도록 지정하십시오. 이렇게 하면 구성 요소가 업데이트할 준비가 되었음을 Nucleus에 알립니다.

기본값은 0밀리초이며, 이는 업데이트 확인을 의미합니다.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

PauseComponent

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

코어 장치의 구성 요소 프로세스를 일시 중지합니다. 구성 요소를 재개하려면 ResumeComponent작업을 사용하십시오.

일반 구성 요소만 일시 중지할 수 있습니다. 다른 유형의 구성 요소를 일시 중지하려고 하면 이 작업을 수행하면 가 발생합니다. `InvalidRequestError`

> **Note**
>
> 이 작업은 Docker 컨테이너와 같은 컨테이너화된 프로세스를 일시 중지할 수 없습니다. Docker 컨테이너를 일시 중지하고 재개하려면 docker 일시 중지 및 docker 일시 중지 해제 명령을 사용할 수 있습니다.

이 작업을 수행해도 구성 요소 종속 항목이나 일시 중지한 구성 요소에 종속되는 구성 요소는 일시 중지되지 않습니다. 종속 구성 요소의 종속 항목이 일시 중지되면 종속 구성 요소에 문제가 발생할 수 있으므로 다른 구성 요소의 종속 구성 요소를 일시 중지할 때는 이 동작을 고려하세요.

배포 등을 통해 일시 중지된 구성 요소를 다시 시작하거나 종료하면 Greengrass nucleus가 구성 요소를 재개하고 종료 수명 주기를 실행합니다. 구성 요소 재시작에 대한 자세한 내용은 을 참조하십시오. RestartComponent

> **Important**
>
> 이 작업을 사용하려면 이 작업을 사용할 권한을 부여하는 권한 부여 정책을 정의해야 합니다. 자세한 설명은 권한 부여 섹션을 참조하세요.

최소 SDK 버전

다음 표에는 구성 요소를 일시 중지하고 AWS IoT Device SDK 재개하는 데 사용해야 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.4.3	
AWS IoT Device SDK파이썬 v2용	v1.6.2	
AWS IoT Device SDKC++ v2의 경우	v1.13.1	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`componentName`(Python:`component_name`)

일시 중지할 구성 요소의 이름. 이 구성 요소는 일반 구성 요소여야 합니다. 자세한 설명은 구성 요소 유형 섹션을 참조하세요.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

ResumeComponent

이 기능은 Greengrass 핵 구성 요소 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

코어 장치에서 구성 요소의 프로세스를 재개합니다. 구성 요소를 일시 중지하려면 작업을 사용합니다. PauseComponent

일시 중지된 구성 요소만 재개할 수 있습니다. 일시 중지되지 않은 구성 요소를 다시 시작하려고 하면 이 작업을 수행하면 가 발생합니다. `InvalidRequestError`

> **Important**
>
> 이 작업을 사용하려면 권한을 부여하는 권한 부여 정책을 정의해야 합니다. 자세한 설명은 권한 부여 섹션을 참조하세요.

최소 SDK 버전

다음 표에는 구성 요소를 일시 중지하고 AWS IoT Device SDK 재개하는 데 사용해야 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전
AWS IoT Device SDKJava v2의 경우	v1.4.3
AWS IoT Device SDK파이썬 v2용	v1.6.2
AWS IoT Device SDKC++ v2의 경우	v1.13.1
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`componentName`(Python:`component_name`)

재개할 구성 요소의 이름.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

구성 요소 구성과 상호 작용

구성 요소 구성 IPC 서비스를 통해 다음을 수행할 수 있습니다.

- 구성 요소 구성 매개 변수를 가져오고 설정합니다.
- 구성 요소 구성 업데이트를 구독하십시오.
- Nucleus가 업데이트를 적용하기 전에 구성 요소 구성 업데이트를 검증하십시오.

주제

최소 SDK 버전

다음 표에는 구성 요소 구성과 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.2.10	
AWS IoT Device SDK파이썬 v2용	v1.5.3	
AWS IoT Device SDKC++ v2의 경우	v1.17.0	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

GetConfiguration

코어 디바이스의 구성 요소에 대한 구성 값을 가져옵니다. 구성 값을 가져올 키 경로를 지정합니다.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

`componentName`(Python:`component_name`)

(선택 사항) 구성 요소의 이름.

요청을 보내는 구성 요소의 이름이 기본값입니다.

`keyPath`(Python:`key_path`)

구성 값의 키 경로입니다. 각 항목이 구성 개체의 단일 수준에 대한 키인 목록을 지정하십시오. 예를 들어, 다음 `port` 구성에서 값을 `["mqtt", "port"]` 가져오도록 지정합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

구성 요소의 전체 구성을 가져오려면 빈 목록을 지정하십시오.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`componentName`(Python:`component_name`)

구성 요소의 이름입니다.

`value`

요청된 구성을 객체로 표시합니다.

UpdateConfiguration

코어 디바이스에서 이 구성 요소의 구성 값을 업데이트합니다.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`keyPath`(Python:`key_path`)

(선택 사항) 업데이트할 컨테이너 노드 (객체) 의 키 경로. 구성 개체의 각 항목이 단일 수준의 키인 목록을 지정하십시오. 예를 들어, 다음 `port` 구성에서 값을 `{ "port": 443 }` 설정할 키 `["mqtt"]` 경로와 병합 값을 지정합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

키 경로는 구성의 컨테이너 노드 (개체) 를 지정해야 합니다. 구성 요소 구성에 해당 노드가 없는 경우 이 작업을 수행하면 해당 노드가 생성되고 해당 값이 의 개체로 설정됩니다`valueToMerge`.

구성 개체의 루트가 기본값입니다.

`timestamp`

현재 Unix 에포크 시간 (밀리초). 이 작업은 이 타임스탬프를 사용하여 키의 동시 업데이트를 해결합니다. 구성 요소 구성의 키의 타임스탬프가 요청의 타임스탬프보다 크면 요청이 실패합니다.

`valueToMerge`(Python:`value_to_merge`)

지정한 위치에 병합할 구성 `keyPath` 객체입니다. 자세한 설명은 구성 요소 구성 업데이트 섹션을 참조하세요.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

SubscribeToConfigurationUpdate

구독하면 구성 요소의 구성이 업데이트될 때 알림을 받을 수 있습니다. 키를 구독하면 해당 키의 하위 항목이 업데이트되면 알림을 받게 됩니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 함수가 포함된 스트림 응답 핸들러를 정의하십시오. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

이벤트 메시지 유형: `ConfigurationUpdateEvents`

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`componentName`(Python:`component_name`)

(선택 사항) 구성 요소의 이름.

요청을 보내는 구성 요소의 이름이 기본값입니다.

`keyPath`(Python:`key_path`)

구독할 구성 값의 키 경로입니다. 각 항목이 구성 개체의 단일 수준에 대한 키가 되는 목록을 지정하십시오. 예를 들어, 다음 `port` 구성에서 값을 `["mqtt", "port"]` 가져오도록 지정합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

구성 요소 구성의 모든 값에 대한 업데이트를 구독하려면 빈 목록을 지정하십시오.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

알림 메시지 스트림. 이 개체`ConfigurationUpdateEvents`, 에는 다음 정보가 들어 있습니다.

`configurationUpdateEvent`(Python:`configuration_update_event`)

구성 업데이트 이벤트. 이 개체`ConfigurationUpdateEvent`, 에는 다음 정보가 들어 있습니다.

`componentName`(Python:`component_name`)

구성 요소의 이름입니다.

`keyPath`(Python:`key_path`)

업데이트된 구성 값의 키 경로입니다.

SubscribeToValidateConfigurationUpdates

이 구성 요소의 구성이 업데이트되기 전에 알림을 받으려면 구독하십시오. 이렇게 하면 구성 요소가 자체 구성에 대한 업데이트를 검증할 수 있습니다. SendConfigurationValidityReport작업을 사용하여 컨피그레이션이 유효한지 여부를 Nucleus에 알릴 수 있습니다.

> **Important**
>
> 로컬 배포에서는 구성 요소에 업데이트를 알리지 않습니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 함수가 포함된 스트림 응답 핸들러를 정의하십시오. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

이벤트 메시지 유형: `ValidateConfigurationUpdateEvents`

요청

이 작업의 요청에는 매개변수가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

알림 메시지 스트림. 이 개체`ValidateConfigurationUpdateEvents`, 에는 다음 정보가 들어 있습니다.

`validateConfigurationUpdateEvent`(Python:`validate_configuration_update_event`)

구성 업데이트 이벤트. 이 개체`ValidateConfigurationUpdateEvent`, 에는 다음 정보가 들어 있습니다.

`deploymentId`(Python:`deployment_id`)

컴포넌트를 업데이트하는 AWS IoT Greengrass 디플로이먼트의 ID.

`configuration`

새 구성이 포함된 개체입니다.

SendConfigurationValidityReport

이 구성 요소에 대한 구성 업데이트가 유효한지 여부를 Nucleus에 알리십시오. NUCLEUS에 새 구성이 유효하지 않다고 알리면 배포가 실패합니다. 이 SubscribeToValidateConfigurationUpdates 작업을 사용하여 구독하여 구성 업데이트의 유효성을 검사하십시오.

구성 요소가 구성 업데이트 유효성 검사 알림에 응답하지 않는 경우 NUCLEUS는 배포의 구성 유효성 검사 정책에 지정된 시간만큼 기다립니다. 제한 시간이 지나면 NUCLEUS는 배포를 진행합니다. 기본 구성 요소 검증 제한 시간은 20초입니다. 자세한 내용은 배포 만들기 및 CreateDeployment작업을 호출할 때 제공할 수 있는 DeploymentConfigurationValidationPolicy객체를 참조하십시오.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

`configurationValidityReport`(Python:`configuration_validity_report`)

구성 업데이트가 유효한지 여부를 NUCLEUS에 알려주는 보고서입니다. 이 개체`ConfigurationValidityReport`, 에는 다음 정보가 들어 있습니다.

`status`

유효성 상태. 이 `ConfigurationValidityStatus` 열거형의 값은 다음과 같습니다.

- `ACCEPTED`— 구성이 유효하며 핵은 이를 이 구성 요소에 적용할 수 있습니다.
- `REJECTED`— 구성이 유효하지 않아 배포에 실패합니다.

`deploymentId`(Python:`deployment_id`)

구성 업데이트를 요청한 AWS IoT Greengrass 배포의 ID.

`message`

(선택 사항) 구성이 유효하지 않은 이유를 보고하는 메시지입니다.

응답

이 작업은 응답에 어떠한 정보도 제공하지 않습니다.

비밀 값 검색

시크릿 매니저 IPC 서비스를 사용하여 코어 디바이스의 시크릿에서 시크릿 값을 검색할 수 있습니다. Secret Manager 구성 요소를 사용하여 암호화된 암호를 핵심 장치에 배포할 수 있습니다. 그런 다음 IPC 작업을 사용하여 암호를 해독하고 해당 값을 사용자 지정 구성 요소에 사용할 수 있습니다.

주제

- 최소 SDK 버전
- 권한 부여
- GetSecretValue
- 예제

최소 SDK 버전

다음 표에는 코어 기기의 비밀번호에서 비밀 값을 검색하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.2.10	
AWS IoT Device SDKPython v2용	v1.5.3	
AWS IoT Device SDKC++ v2의 경우	v1.17.0	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 Secret Manager를 사용하려면 구성 요소가 코어 기기에 저장한 암호의 가치를 가져오도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오..

시크릿 매니저의 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.SecretManager`

Operation	설명	리소스
`aws.greengrass#GetSecretValue`	구성 요소가 코어 디바이스에서 암호화된 비밀의 가치를 가져올 수 있도록 합니다.	Secrets Manager 보안 ARN 또는 모든 비밀에 대한 액세스를 `*` 허용하기 위한 것입니다.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 권한 부여 정책 예시

다음 예제 권한 부여 정책은 구성 요소가 코어 기기의 모든 암호 값을 가져올 수 있도록 허용합니다.

> **Note**
>
> 프로덕션 환경에서는 구성 요소가 사용하는 암호만 검색하도록 권한 부여 정책의 범위를 줄이는 것이 좋습니다. 구성 요소를 배포할 때 `*` 와일드카드를 비밀 ARN 목록으로 변경할 수 있습니다.

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
```

```
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

코어 디바이스에 저장한 비밀의 값을 가져옵니다.

이 작업은 에서 보안 값을 가져오는 데 사용할 수 있는 Secrets Manager 작업과 비슷합니다AWS 클라우드. 자세한 내용을 알아보려면 AWS Secrets Manager API 참조의 GetSecretValue 섹션을 참조하십시오.

요청

이 작업의 요청에는 다음과 같은 매개 변수가 있습니다.

`secretId`(Python:`secret_id`)

얻어야 할 비밀의 이름. Amazon 리소스 이름 (ARN) 또는 암호의 친숙한 이름을 지정할 수 있습니다.

`versionId`(Python:`version_id`)

(선택 사항) 가져올 버전의 ID.

`versionId` 또는 `versionStage`를 지정할 수 있습니다.

`versionId`또는 를 지정하지 않는 `versionStage` 경우 이 작업의 기본값은 `AWSCURRENT` 레이블이 있는 버전으로 설정됩니다.

`versionStage`(Python:`version_stage`)

(선택 사항) 가져올 버전의 스테이징 레이블입니다.

`versionId` 또는 `versionStage`를 지정할 수 있습니다.

`versionId`또는 `versionStage` 를 지정하지 않으면 레이블이 있는 버전이 이 작업의 기본값이 됩니다`AWSCURRENT`.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`secretId`(Python:`secret_id`)

비밀의 ID.

`versionId`(Python:`version_id`)

이 버전의 시크릿의 ID입니다.

`versionStage`(Python:`version_stage`)

이 버전의 시크릿에 첨부된 스테이징 라벨 목록.

`secretValue`(Python:`secret_value`)

이 버전의 시크릿의 가치. 이 개체`SecretValue`, 에는 다음 정보가 들어 있습니다.

`secretString`(Python:`secret_string`)

Secrets Manager에 문자열로 제공한 보호된 비밀 정보 중 해독된 부분입니다.

`secretBinary`(Python:`secret_binary`)

(선택 사항) Secrets Manager에 바이트 배열 형식의 이진 데이터로 제공한 보호된 비밀 정보 중 복호화된 부분입니다. 이 속성에는 이진 데이터가 base64로 인코딩된 문자열로 포함되어 있습니다.

Secrets Manager 콘솔에서 시크릿을 생성한 경우에는 이 속성이 사용되지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example 예: 비밀 값 가져오기

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String secretArn = args[0];
        String versionStage = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetSecretValueResponseHandler responseHandler =
                    GetSecretValue.getSecretValue(ipcClient, secretArn,
 versionStage);
            CompletableFuture<GetSecretValueResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetSecretValueResponse response =
 futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                response.getSecretValue().postFromJson();
                String secretString = response.getSecretValue().getSecretString();
                System.out.println("Successfully retrieved secret value: " +
 secretString);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while retrieving secret: " +
 secretArn);
            } catch (ExecutionException e) {
```

```
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while retrieving secret:
 " + secretArn);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetSecretValueResponseHandler
 getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
 String versionStage) {
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 예: 비밀 값 가져오기

> **Note**
>
> 이 예제에서는 AWS IoT Device SDK Python v2용 버전 1.5.4 이상을 사용하고 있다고 가정합니다.

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
```

```
    GetSecretValueResponse,
    UnauthorizedError
)

secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Example 예: 비밀 값 가져오기

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
    private readonly secretId : string;
    private readonly versionStage : string;
    private ipcClient : greengrasscoreipc.Client

    constructor() {
        this.secretId = "<define_your_own_secretId>"
        this.versionStage = "<define_your_own_versionStage>"

        this.getSecretValue().then(r => console.log("Started workflow"));
    }

    private async getSecretValue() {
```

```
        try {
            this.ipcClient = await getIpcClient();

            const getSecretValueRequest : GetSecretValueRequest = {
                secretId: this.secretId,
                versionStage: this.versionStage,
            };

            const result = await
 this.ipcClient.getSecretValue(getSecretValueRequest);
            const secretString = result.secretValue.secretString;
            console.log("Successfully retrieved secret value: " + secretString)
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

예제

다음 예제를 사용하여 구성 요소에서 Secret Manager IPC 서비스를 사용하는 방법을 알아보십시오.

예: 프린트 시크릿 (Python, IPC 클라이언트 V1)

이 예제 구성 요소는 코어 기기에 배포한 암호 값을 인쇄합니다.

> **Important**
>
> 이 예제 구성 요소는 암호 값을 인쇄하므로 테스트 데이터를 저장하는 암호에만 사용하십시오. 중요한 정보를 저장하는 시크릿의 값을 인쇄할 때는 이 컴포넌트를 사용하지 마세요.

주제

- 레시피
- 아티팩트
- 사용량

레시피

다음 예제 레시피는 비밀 ARN 구성 매개변수를 정의하고 구성 요소가 코어 디바이스에 있는 모든 시크릿 값을 가져올 수 있도록 합니다.

> **Note**
>
> 프로덕션 환경에서는 구성 요소가 사용하는 암호만 검색하도록 권한 부여 정책의 범위를 줄이는 것이 좋습니다. 구성 요소를 배포할 때 * 와일드카드를 비밀 ARN 목록으로 변경할 수 있습니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
```

```
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
```

아티팩트

다음 예제 Python 애플리케이션은 비밀 관리자 IPC 서비스를 사용하여 핵심 기기의 비밀 값을 가져오는 방법을 보여줍니다.

```
import concurrent.futures
import json
import sys
import traceback
```

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()

    try:
        response = future_response.result(TIMEOUT)
        secret_json = json.loads(response.secret_value.secret_string)
        print('Successfully got secret: ' + secret_id)
        print('Secret value: ' + str(secret_json))
    except concurrent.futures.TimeoutError:
        print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
    except UnauthorizedError as e:
        print('Unauthorized error while getting secret: ' + secret_id,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while getting secret: ' + secret_id, file=sys.stderr)
        raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

사용량

이 예제 구성 요소를 비밀 관리자 구성 요소와 함께 사용하여 핵심 기기에 암호 값을 배포하고 인쇄할 수 있습니다.

테스트 시크릿을 만들고, 배포하고, 인쇄하려면

1. 테스트 데이터를 사용하여 Secrets Manager 시크릿을 생성합니다.

   Linux or Unix

   ```
   aws secretsmanager create-secret \
     --name MyTestGreengrassSecret \
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Windows Command Prompt (CMD)

   ```
   aws secretsmanager create-secret ^
     --name MyTestGreengrassSecret ^
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   PowerShell

   ```
   aws secretsmanager create-secret `
     --name MyTestGreengrassSecret `
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   다음 단계에서 사용할 비밀번호의 ARN을 저장합니다.

   자세한 내용은 AWS Secrets Manager사용 설명서의 암호 생성을 참조하십시오.

2. 다음 구성 병합 업데이트와 함께 비밀 관리자 구성 요소 (`aws.greengrass.SecretManager`)를 배포하십시오. 이전에 생성한 비밀번호의 ARN을 지정합니다.

   ```
   {
     "cloudSecrets": [
       {
         "arn": "arn:aws:secretsmanager:us-
   west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
       }
     ]
   ```

```
}
```

자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포 또는 Greengrass CLI 배포 명령을 참조하십시오.

3. 다음 구성 병합 업데이트를 사용하여 이 섹션의 예제 구성 요소를 생성하고 배포하십시오. 이전에 생성한 비밀번호의 ARN을 지정합니다.

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
        ]
      }
    }
  }
}
```

자세한 내용은 AWS IoT Greengrass 구성 요소 생성 단원을 참조하세요.

4. AWS IoT Greengrass코어 소프트웨어 로그를 보고 배포가 성공했는지 확인하고, `com.example.PrintSecret` 구성 요소 로그를 보면 암호 값이 인쇄되었는지 확인할 수 있습니다. 자세한 내용은 모니터 AWS IoT Greengrass 로그을(를) 참조하세요.

로컬 섀도우와 상호작용

섀도우 IPC 서비스를 사용하여 기기의 로컬 섀도우와 상호 작용할 수 있습니다. 상호 작용하도록 선택한 장치는 코어 장치일 수도 있고 연결된 클라이언트 장치일 수도 있습니다.

이러한 IPC 작업을 사용하려면 섀도우 관리자 구성 요소를 사용자 지정 구성 요소의 종속 항목으로 포함시키십시오. 그런 다음 사용자 지정 구성 요소에서 IPC 작업을 사용하여 섀도우 관리자를 통해 장치의 로컬 섀도우와 상호 작용할 수 있습니다. 사용자 지정 구성 요소가 로컬 섀도우 상태의 변화에 반응

하도록 하려면 게시/구독 IPC 서비스를 사용하여 섀도우 이벤트를 구독할 수도 있습니다. 게시/구독 서비스 사용에 대한 자세한 내용은 를 참조하십시오. 로컬 메시지 게시/구독

> **Note**
>
> 코어 디바이스가 클라이언트 디바이스 섀도우와 상호 작용할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 섀도우 관리자가 클라이언트 장치와 통신하도록 활성화를 참조하십시오.

주제

- 최소 SDK 버전
- 권한 부여
- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

최소 SDK 버전

다음 표에는 로컬 섀도우와 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는 의 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.4.0	
AWS IoT Device SDK파이썬 v2용	v1.6.0	
AWS IoT Device SDKC++ v2용	v1.17.0	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 섀도우 IPC 서비스를 사용하려면 구성 요소가 섀도우와 상호 작용할 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오..

섀도우 상호 작용에 대한 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ShadowManager`

Operation	설명	리소스
`aws.greengrass#GetThingShadow`	구성 요소가 사물의 그림자를 검색할 수 있도록 합니다.	다음 문자열 중 하나: • `$aws/things/`*thingName*`/shadow/`, 클래식 디바이스 섀도우에 대한 액세스를 허용하려면 • `$aws/things/`*thingName*`/shadow/name/`*shadowName*, 명명된 섀도우에 대한 액세스를 허용하려는 경우 • `*`모든 섀도우에 대한 액세스를 허용하기 위해서입니다.
`aws.greengrass#UpdateThingShadow`	컴포넌트가 사물의 섀도우를 업데이트할 수 있도록 합니다.	다음 문자열 중 하나: • `$aws/things/`*thingName*`/shadow/`, 클래식 디바이스 섀도우에 대한 액세스를 허용하려면 • `$aws/things/`*thingName*`/shadow/n`

Operation	설명	리소스
		ame/ *shadowName* , 명명된 섀도우에 대한 액세스를 허용하려는 경우 • *모든 섀도우에 대한 액세스를 허용하기 위해서입니다.
aws.greengrass#DeleteThingShadow	컴포넌트가 사물의 그림자를 삭제할 수 있도록 합니다.	다음 문자열 중 하나: • $aws/things/ *thingName* /shadow/, 클래식 디바이스 섀도우에 대한 액세스를 허용하려면 • $aws/things/ *thingName* /shadow/name/ *shadowName* , 명명된 섀도우에 대한 액세스 허용하기 • *, 모든 섀도우에 대한 액세스를 허용하려면
aws.greengrass#ListNamedShadowsForThing	컴포넌트가 사물에 대해 명명된 섀도우 목록을 검색할 수 있도록 합니다.	사물에 액세스하여 섀도우를 나열할 수 있는 사물 이름 문자열입니다. 모든 항목에 대한 액세스를 허용하는 * 데 사용합니다.

IPC 서비스 식별자: aws.greengrass.ipc.pubsub

Operation	설명	리소스
`aws.greengrass#SubscribeToTopic`	구성 요소가 지정한 주제에 대한 메시지를 구독할 수 있도록 합니다.	다음 주제 문자열 중 하나: • *shadowTopicPrefix* / get/accepted • *shadowTopicPrefix* / get/rejected • *shadowTopicPrefix* / delete/accepted • *shadowTopicPrefix* / delete/rejected • *shadowTopicPrefix* / update/accepted • *shadowTopicPrefix* / update/delta • *shadowTopicPrefix* / update/rejected 주제 접두사의 값은 섀도우 유형에 *shadowTopicPrefix* 따라 달라집니다. • 클래식 섀도우: $aws/things/ *thingName* /shadow • 네임드 섀도우: $aws/things/ *thingName* /shadow/name/ *shadowName* 모든 주제에 대한 액세스를 허용하는 * 데 사용합니다. Greengrass nucleus v2.6.0 이상에서는 MQTT 주제 와일드카

Operation	설명	리소스
		드 (및) 가 포함된 주제를 구독할 수 있습니다. # + 이 주제 문자열은 MQTT 주제 와일드카드를 리터럴 문자로 지원합니다. 예를 들어 구성 요소의 권한 부여 정책이 액세스 권한을 부여하면 구성 요소가 구독할 수는 있지만 구독할 `test/topic/#` 수는 없습니다. `test/topic/#` `test/topic/filter`

로컬 섀도우 권한 부여 정책의 레시피 변수

v2.6.0 이상의 Greengrass 핵을 사용하고 Greengrass 핵의 interpolateComponentConfiguration구성 옵션을 로 설정하면 권한 부여 정책에서 레시피 변수를 사용할 수 `true` 있습니다. `{iot:thingName}` 이 기능을 사용하면 각 코어 디바이스가 자체 섀도우에만 액세스할 수 있도록 코어 디바이스 그룹에 대해 단일 권한 부여 정책을 구성할 수 있습니다. 예를 들어 섀도우 IPC 작업을 위해 구성 요소가 다음 리소스에 액세스하도록 허용할 수 있습니다.

```
$aws/things/{iot:thingName}/shadow/
```

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 예: 코어 디바이스 그룹이 로컬 섀도우와 상호 작용하도록 허용

> **Important**
>
> 이 예제에서는 v2.6.0 이상에서 사용할 수 있는 Greengrass 핵 구성 요소 기능을 사용합니다. Greengrass nucleus v2.6.0은 대부분의 레시피 변수 (예: 구성 요소 구성) 에 대한 지원을 추가합니다. `{iot:thingName}` 이 기능을 활성화하려면 Greengrass interpolateComponentConfigurationnucleus의 구성 옵션을 로 설정합니다. `true` 모든 버전의

Greengrass NUCLEUS에서 작동하는 예제는 단일 코어 기기에 대한 권한 부여 정책 예제를 참조하십시오.

다음 예제 권한 부여 정책은 구성 요소가 com.example.MyShadowInteractionComponent 클래식 디바이스 섀도 및 구성 요소를 실행하는 코어 디바이스의 명명된 myNamedShadow 섀도우와 상호 작용할 수 있도록 허용합니다. 또한 이 정책을 통해 이 구성 요소는 이러한 섀도우에 대한 로컬 주제에 대한 메시지를 수신할 수 있습니다.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
```

```
        "resources": [
          "$aws/things/{iot:thingName}/shadow/get/accepted",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted
```

Example 예: 핵심 장치 그룹이 클라이언트 장치 섀도우와 상호 작용하도록 허용

> **Important**
>
> 이 기능을 사용하려면 그린그래스 뉴클리우스 v2.6.0 이상, 섀도우 매니저 v2.2.0 이상, MQTT 브리지 v2.2.0 이상이 필요합니다. 섀도우 관리자가 클라이언트 장치와 통신할 수 있도록 MQTT 브리지를 구성해야 합니다.

다음 예제 권한 부여 정책은 구성 요소가 `com.example.MyShadowInteractionComponent` 이름이 로 시작하는 클라이언트 장치의 모든 디바이스 섀도우와 상호 작용할 수 있도록 허용합니다. `MyClientDevice`

> **Note**
>
> 코어 디바이스가 클라이언트 디바이스 섀도우와 상호 작용할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 섀도우 관리자가 클라이언트 장치와 통신하도록 활성화를 참조하십시오.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
```

```
        ],
        "resources": [
          "MyClientDevice*"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*
```

Example 예: 단일 코어 기기가 로컬 섀도우와 상호 작용하도록 허용

다음 예제 권한 부여 정책은 구성 요소가 `com.example.MyShadowInteractionComponent` 클래식 장치 섀도 및 장치의 명명된 섀도우와 상호 작용할 수 `myNamedShadow` 있도록 허용합니다`MyThingName`. 또한 이 정책을 통해 이 구성 요소는 이러한 섀도우에 대한 로컬 주제에 대한 메시지를 수신할 수 있습니다.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
```

```
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
```

```
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted
```

Example 예: 코어 디바이스 그룹이 로컬 섀도우 상태 변경에 반응하도록 허용

> **Important**
>
> 이 예제에서는 v2.6.0 이상에서 사용할 수 있는 Greengrass 핵 구성 요소 기능을 사용합니다. Greengrass nucleus v2.6.0은 대부분의 레시피 변수 (예: 구성 요소 구성) 에 대한 지원을 추가합니다. `{iot:thingName}` 이 기능을 활성화하려면 Greengrass interpolateComponentConfigurationnucleus의 구성 옵션을 로 설정합니다. `true` 모든 버전의 Greengrass NUCLEUS에서 작동하는 예제는 단일 코어 기기에 대한 권한 부여 정책 예제를 참조하십시오.

다음 예제 액세스 제어 정책을 사용하면 사용자 `com.example.MyShadowReactiveComponent` 지정이 클래식 디바이스 섀도와 해당 구성 요소를 실행하는 각 코어 디바이스의 명명된 `myNamedShadow` 섀도우에 대한 `/update/delta` 주제에 대한 메시지를 받을 수 있습니다.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Example 예: 단일 코어 장치가 로컬 섀도우 상태 변경에 반응하도록 허용

다음 예제 액세스 제어 정책을 사용하면 사용자 `com.example.MyShadowReactiveComponent` 지정이 클래식 장치 섀도와 장치의 명명된 섀도우에 대한 `/update/delta` 주제에 `myNamedShadow` 대한 메시지를 받을 수 `MyThingName` 있습니다.

JSON

```
{
```

```
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/update/delta
        - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

지정된 사물에 대한 섀도우를 가져오십시오.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`thingName`(Python:`thing_name`)

사물의 이름입니다.

유형: `string`

shadowName(Python:shadow_name)

섀도우의 이름입니다. 사물의 클래식 섀도우를 지정하려면 이 매개변수를 빈 문자열 ("") 으로 설정하십시오.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 AWSManagedGreengrassV2Deployment 명명된 섀도우를 사용하여 개별 코어 장치를 대상으로 하는 배포를 관리합니다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다AWS IoT Greengrass. 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

유형: string

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

payload

블럽 형태의 응답 상태 문서.

다음 정보가 object 포함된 유형:

state

상태 정보.

이 개체에는 다음 정보가 들어 있습니다.

desired

장치에서 업데이트하도록 요청된 상태 속성 및 값.

유형: map 키-값 쌍

reported

기기에서 보고한 상태 속성 및 값.

유형: map 키-값 쌍

`delta`

원하는 상태 속성과 보고된 상태 속성 및 값 간의 차이 이 속성은 `desired` 및 `reported` 상태가 다른 경우에만 나타납니다.

유형: `map` 키-값 쌍

`metadata`

`desired`및 `reported` 섹션의 각 속성에 대한 타임스탬프를 통해 상태가 업데이트된 시기를 확인할 수 있습니다.

유형: `string`

`timestamp`

응답이 생성된 에포크 날짜 및 시간.

유형: `integer`

`clientToken`(Python:`clientToken`)

요청과 해당 응답을 일치시키는 데 사용되는 토큰

유형: `string`

`version`

로컬 섀도우 문서의 버전.

유형: `integer`

Errors

이 작업을 수행하면 다음 오류가 반환될 수 있습니다.

`InvalidArgumentsError`

로컬 섀도우 서비스가 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON이나 지원되지 않는 문자가 포함된 경우 이 문제가 발생할 수 있습니다.

`ResourceNotFoundError`

요청된 로컬 섀도우 문서를 찾을 수 없습니다.

`ServiceError`

내부 서비스 오류가 발생했거나 IPC 서비스에 대한 요청 수가 샤도우 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 매개변수에 지정된 제한을 초과했습니다.

`UnauthorizedError`

구성 요소의 권한 부여 정책에는 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example 예: 사물 그림자 가져오기

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
```

```
public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                    GetThingShadow.getThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
 futureResponse.get(TIMEOUT_SECONDS,
                        TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
 StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
 shadowName,
                        shadowPayload);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to get: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
```

```
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetThingShadowResponseHandler
 getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
 String shadowName) {
        GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
        getThingShadowRequest.setThingName(thingName);
        getThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 예: 사물 섀도우 가져오기

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest

TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName

        # retrieve the GetThingShadow response after sending the request to the IPC
 server
```

```
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 예: 사물 섀도우 가져오기

```
import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleGetThingShadowOperation(this.thingName,
                this.shadowName);
```

```
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleGetThingShadowOperation(
        thingName: string,
        shadowName: string
    ) {
        const request: GetThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        };
        const response = await this.ipcClient.getThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use caseså
        throw err
    }
}

const startScript = new GetThingShadow();
```

UpdateThingShadow

지정된 사물의 섀도우를 업데이트하십시오. 섀도우가 없는 경우 섀도우가 생성됩니다.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`thingName`(Python:`thing_name`)

사물의 이름입니다.

유형: `string`

`shadowName`(Python:`shadow_name`)

섀도우의 이름입니다. 사물의 클래식 섀도우를 지정하려면 이 매개변수를 빈 문자열 ("") 으로 설정하십시오.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명명된 섀도우를 사용하여 개별 코어 장치를 대상으로 하는 배포를 관리합니다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다AWS IoT Greengrass. 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

유형: `string`

`payload`

Blob 형태의 요청 상태 문서.

다음 정보가 `object` 포함된 유형:

`state`

업데이트할 상태 정보. 이 IPC 작업은 지정된 필드에만 영향을 줍니다.

이 객체에는 다음 정보가 들어 있습니다. 일반적으로 동일한 요청에서 `desired` 속성과 속성 중 하나를 사용하지만 둘 다 사용하지는 않습니다. `reported`

`desired`

기기에서 업데이트하도록 요청된 상태 속성 및 값

유형: `map` 키-값 쌍

`reported`

기기에서 보고한 상태 속성 및 값.

유형: `map` 키-값 쌍

clientToken(Python:client_token)

(선택 사항) 클라이언트 토큰의 요청과 해당 응답을 일치시키는 데 사용되는 토큰입니다.

유형: string

version

(선택 사항) 업데이트할 로컬 섀도우 문서의 버전. 섀도우 서비스는 지정된 버전이 최신 버전과 일치하는 경우에만 업데이트를 처리합니다.

유형: integer

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

payload

블럽 형태의 응답 상태 문서.

다음 정보가 object 포함된 유형:

state

상태 정보.

이 개체에는 다음 정보가 들어 있습니다.

desired

장치에서 업데이트하도록 요청된 상태 속성 및 값.

유형: map 키-값 쌍

reported

기기에서 보고한 상태 속성 및 값.

유형: map 키-값 쌍

delta

기기에서 보고한 상태 속성 및 값.

유형: map 키-값 쌍

`metadata`

`desired`및 `reported` 섹션의 각 속성에 대한 타임스탬프를 통해 상태가 업데이트된 시기를 확인할 수 있습니다.

유형: `string`

`timestamp`

응답이 생성된 에포크 날짜 및 시간.

유형: `integer`

`clientToken`(Python:`client_token`)

요청과 해당 응답을 일치시키는 데 사용되는 토큰입니다.

유형: `string`

`version`

업데이트가 완료된 후의 로컬 섀도우 문서 버전입니다.

유형: `integer`

Errors

이 작업을 수행하면 다음 오류가 반환될 수 있습니다.

`ConflictError`

업데이트 작업 중에 로컬 섀도우 서비스에서 버전 충돌이 발생했습니다. 이 문제는 요청 페이로드의 버전이 사용 가능한 최신 로컬 섀도 문서의 버전과 일치하지 않을 때 발생합니다.

`InvalidArgumentsError`

로컬 섀도우 서비스는 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON이나 지원되지 않는 문자가 포함된 경우 이 문제가 발생할 수 있습니다.

payloadValid에는 다음과 같은 속성이 있습니다.

- `state`노드가 존재하며, 노드는 `desired` 또는 `reported` 상태 정보를 포함하는 객체입니다.
- `desired`및 `reported` 노드는 객체이거나 null입니다. 이러한 개체 중 적어도 하나는 유효한 상태 정보를 포함해야 합니다.
- `desired`및 `reported` 개체의 깊이는 8개 노드를 초과할 수 없습니다.

- `clientToken`값의 길이는 64자를 초과할 수 없습니다.
- `version`값은 1 이상이어야 합니다.

`ServiceError`

내부 서비스 오류가 발생했거나 IPC 서비스에 대한 요청 수가 섀도우 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 매개 변수에 지정된 제한을 초과했습니다.

`UnauthorizedError`

구성 요소의 권한 부여 정책에는 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example 예: 사물 섀도우 업데이트

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
```

```
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            UpdateThingShadowResponseHandler responseHandler =
                    UpdateThingShadow.updateThingShadow(ipcClient, thingName,
 shadowName,
                            shadowPayload);
            CompletableFuture<UpdateThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully updated shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while updating shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while updating shadow: %s/
%s%n",
                            thingName, shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
```

```
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
 updateThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
 UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCClient.updateThingShadow(updateThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 예: 사물 섀도우 업데이트

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()
```

```
        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Example 예: 사물 섀도우 업데이트

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;
    private shadowDocumentStr: string;

    constructor() {
        // Define args parameters here

        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.shadowDocumentStr = "<define_your_own_payload>";

        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleUpdateThingShadowOperation(
```

```
                this.thingName,
                this.shadowName,
                this.shadowDocumentStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleUpdateThingShadowOperation(
        thingName: string,
        shadowName: string,
        payloadStr: string
    ) {
        const request: UpdateThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName,
            payload: payloadStr
        }
        // make the UpdateThingShadow request
        const response = await this.ipcClient.updateThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();
```

DeleteThingShadow

지정된 사물의 섀도우를 삭제합니다.

섀도우 관리자 v2.0.4부터 섀도우를 삭제하면 버전 번호가 증가합니다. 예를 들어 버전 `MyThingShadow` 1에서 섀도우를 삭제하면 삭제된 섀도우의 버전은 2입니다. 그런 다음 같은 이름으로 `MyThingShadow` 섀도우를 다시 만들면 해당 섀도우의 버전은 3입니다.

요청

이 작업의 요청에는 다음과 같은 매개 변수가 있습니다.

`thingName`(Python:`thing_name`)

사물의 이름입니다.

유형: `string`

`shadowName`(Python:`shadow_name`)

섀도우의 이름입니다. 사물의 클래식 섀도우를 지정하려면 이 매개변수를 빈 문자열 ("") 으로 설정하십시오.

> **Warning**
>
> 이 AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명명된 섀도우를 사용하여 개별 코어 장치를 대상으로 하는 배포를 관리합니다. 이 이름이 지정된 섀도우는 서비스에서 사용하도록 예약되어 있습니다AWS IoT Greengrass. 이름이 지정된 이 섀도우를 업데이트하거나 삭제하지 마십시오.

유형: `string`

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`payload`

빈 응답 상태 문서.

Errors

이 작업을 수행하면 다음 오류가 반환될 수 있습니다.

`InvalidArgumentsError`

로컬 섀도우 서비스가 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON이나 지원되지 않는 문자가 포함된 경우 이 문제가 발생할 수 있습니다.

`ResourceNotFoundError`

요청된 로컬 섀도우 문서를 찾을 수 없습니다.

`ServiceError`

내부 서비스 오류가 발생했거나 IPC 서비스에 대한 요청 수가 섀도우 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 매개변수에 지정된 제한을 초과했습니다.

`UnauthorizedError`

구성 요소의 권한 부여 정책에는 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example 예: 사물 섀도우 삭제

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
```

```
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            DeleteThingShadowResponseHandler responseHandler =
                    DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<DeleteThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
```

```
                    System.err.printf("Unable to find shadow to delete: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static DeleteThingShadowResponseHandler
 deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName) {
        DeleteThingShadowRequest deleteThingShadowRequest = new
 DeleteThingShadowRequest();
        deleteThingShadowRequest.setThingName(thingName);
        deleteThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 예: 사물 섀도우 삭제

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
```

```
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName

        # retrieve the DeleteThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_delete_thing_shadow()
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 예: 사물 섀도우 삭제

```
import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
```

```
            throw err
        }

        try {
            await this.handleDeleteThingShadowOperation(this.thingName,
 this.shadowName)
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

지정된 사물의 이름이 지정된 섀도우를 나열하십시오.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`thingName`(Python:`thing_name`)

사물의 이름입니다.

유형: `string`

`pageSize`(Python:`page_size`)

(선택 사항) 각 호출에서 반환되는 섀도우 이름 수.

유형: `integer`

기본값: 25

최대: 100

`nextToken`(Python:`next_token`)

(선택 사항) 다음 결과 세트를 검색하기 위한 토큰입니다. 이 값은 페이징된 결과에서 반환되며 다음 페이지를 반환하는 호출에 사용됩니다.

유형: `string`

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`results`

섀도우 이름 목록.

유형: `array`

`timestamp`

(선택 사항) 응답이 생성된 날짜 및 시간.

유형: `integer`

`nextToken`(Python:`next_token`)

(선택 사항) 시퀀스의 다음 페이지를 검색하기 위한 페이징 요청에서 사용할 토큰 값입니다. 반환할 섀도우 이름이 더 이상 없을 때는 이 토큰이 표시되지 않습니다.

유형: `string`

> **Note**
>
> 요청된 페이지 크기가 응답의 섀도우 이름 수와 정확히 일치하면 이 토큰이 존재하지만 사용하면 빈 목록이 반환됩니다.

Errors

이 작업을 수행하면 다음 오류가 반환될 수 있습니다.

`InvalidArgumentsError`

로컬 섀도우 서비스가 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON이나 지원되지 않는 문자가 포함된 경우 이 문제가 발생할 수 있습니다.

`ResourceNotFoundError`

요청된 로컬 섀도우 문서를 찾을 수 없습니다.

`ServiceError`

내부 서비스 오류가 발생했거나 IPC 서비스에 대한 요청 수가 섀도우 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 매개변수에 지정된 제한을 초과했습니다.

`UnauthorizedError`

구성 요소의 권한 부여 정책에는 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제는 사용자 지정 구성 요소 코드에서 이 작업을 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example 예: 이름이 지정된 사물의 그림자 나열

> **Note**
>
> 이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass Core IPC 서비스에 대한 연결을 생성합니다. 자세한 설명은 AWS IoT Greengrass코어 IPC 서비스에 연결 섹션을 참조하세요.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
 software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
import
 software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
```

```
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
 response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

 ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                    nextToken, 25);
                    CompletableFuture<ListNamedShadowsForThingResponse>
 futureResponse =
                            responseHandler.getResponse();
                    ListNamedShadowsForThingResponse response =
                            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                    List<String> responseNamedShadows = response.getResults();
                    namedShadows.addAll(responseNamedShadows);
                    nextToken = response.getNextToken();
                } while (nextToken != null);
                System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
                        String.join(",", namedShadows));
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while listing named shadows for
 thing: " + thingName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while listing named
 shadows for " +
                            "thing: " + thingName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.println("Unable to find thing to list named shadows:
 " + thingName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
```

```
            System.exit(1);
        }
    }

    public static ListNamedShadowsForThingResponseHandler
 listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String nextToken, int pageSize) {
        ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
                new ListNamedShadowsForThingRequest();
        listNamedShadowsForThingRequest.setThingName(thingName);
        listNamedShadowsForThingRequest.setNextToken(nextToken);
        listNamedShadowsForThingRequest.setPageSize(pageSize);
        return
 greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 예: 이름이 지정된 사물의 그림자 나열

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10

def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize

        # retrieve the ListNamedShadowsForThingRequest response after sending the
 request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()
```

```
        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 예: 사물의 이름이 지정된 그림자 나열하기

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
```

```
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
    ) {
        let request: ListNamedShadowsForThingRequest = {
            thingName: thingName,
            nextToken: nextToken,
        };
        if (pageSizeStr) {
            request.pageSize = parseInt(pageSizeStr);
        }
        // make the ListNamedShadowsForThing request
        const response = await this.ipcClient.listNamedShadowsForThing(request);
        const shadowNames = response.results;
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}
```

```
const startScript = new listNamedShadowsForThing();
```

로컬 배포 및 구성 요소 관리

> **Note**
>
> 이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

Greengrass CLI IPC 서비스를 사용하여 코어 디바이스의 로컬 배포 및 그린그래스 구성 요소를 관리할 수 있습니다.

이러한 IPC 작업을 사용하려면 Greengrass CLI 구성 요소 버전 2.6.0 이상을 사용자 지정 구성 요소의 종속 항목으로 포함해야 합니다. 그런 다음 사용자 지정 구성 요소에서 IPC 작업을 사용하여 다음을 수행할 수 있습니다.

- 로컬 배포를 생성하여 코어 디바이스의 Greengrass 구성 요소를 수정 및 구성합니다.
- 코어 디바이스에서 Greengrass 구성 요소를 다시 시작하고 중지합니다.
- 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 암호를 생성합니다.

주제

- 최소 SDK 버전
- 권한 부여
- CreateLocalDeployment
- ListLocalDeployments
- GetLocalDeploymentStatus
- ListComponents
- GetComponentDetails
- RestartComponent
- StopComponent
- CreateDebugPassword

최소 SDK 버전

다음 표에는 Greengrass CLI IPC 서비스와 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDK자바 v2의 경우	v1.2.10	
AWS IoT Device SDKPython v2용	v1.5.3	
AWS IoT Device SDKC++ v2의 경우	v1.17.0	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 Greengrass CLI IPC 서비스를 사용하려면 구성 요소가 로컬 배포 및 구성 요소를 관리할 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오. 구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오.

Greengrass CLI의 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.Cli`

Operation	설명	리소스
`aws.greengrass#CreateLocalDeployment`	구성 요소가 코어 기기에 로컬 배포를 생성할 수 있도록 합니다.	*
`aws.greengrass#ListLocalDeployments`	구성 요소가 코어 장치의 로컬 배포를 나열할 수 있도록 허용합니다.	*

Operation	설명	리소스
`aws.greengrass#GetLocalDeploymentStatus`	구성 요소가 코어 장치의 로컬 배포 상태를 가져올 수 있도록 합니다.	로컬 배포 ID 또는 모든 로컬 배포에 대한 액세스를 허용하는 * 데 사용할 수 있습니다.
`aws.greengrass#ListComponents`	구성 요소가 코어 장치의 구성 요소를 나열할 수 있도록 허용합니다.	*
`aws.greengrass#GetComponentDetails`	구성 요소가 코어 장치의 구성 요소에 대한 세부 정보를 가져올 수 있도록 합니다.	모든 구성 요소에 대한 * 액세스를 허용하는 또는 와 `com.example.HelloWorld` 같은 구성 요소 이름.
`aws.greengrass#RestartComponent`	구성 요소가 코어 장치에서 구성 요소를 다시 시작할 수 있도록 합니다.	모든 구성 요소에 대한 * 액세스를 허용하는 또는 와 `com.example.HelloWorld` 같은 구성 요소 이름.
`aws.greengrass#StopComponent`	구성 요소가 코어 장치의 구성 요소를 중지할 수 있도록 합니다.	모든 구성 요소에 대한 * 액세스를 허용하는 또는 와 `com.example.HelloWorld` 같은 구성 요소 이름.
`aws.greengrass#CreateDebugPassword`	구성 요소가 로컬 디버그 콘솔 구성 요소에 로그인하는 데 사용할 암호를 생성할 수 있도록 합니다.	*

Example 권한 부여 정책 예시

다음 예제 권한 부여 정책은 구성 요소가 로컬 배포를 만들고, 모든 로컬 배포 및 구성 요소를 보고, 이름이 지정된 구성 요소를 재시작 및 중지하도록 허용합니다. `com.example.HelloWorld`

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
```

```
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view
 deployments and components.",
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",
          "aws.greengrass#ListComponents",
          "aws.greengrass#GetComponentDetails"
        ],
        "resources": [
          "*"
        ]
      }
    },
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:2": {
        "policyDescription": "Allows access to restart and stop the Hello World
 component.",
        "operations": [
          "aws.greengrass#RestartComponent",
          "aws.greengrass#StopComponent"
        ],
        "resources": [
          "com.example.HelloWorld"
        ]
      }
    }
  }
}
```

CreateLocalDeployment

지정된 구성 요소 레시피, 아티팩트 및 런타임 인수를 사용하여 로컬 배포를 만들거나 업데이트합니다.

이 작업은 Greengrass CLI의 배포 생성 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

`recipeDirectoryPath`(Python:`recipe_directory_path`)

(선택 사항) 구성 요소 레시피 파일이 들어 있는 폴더의 절대 경로입니다.

`artifactDirectoryPath`(Python:`artifact_directory_path`)

(선택 사항) 배포에 포함할 아티팩트 파일이 들어 있는 폴더의 절대 경로입니다. 아티팩트 폴더에는 다음과 같은 폴더 구조가 포함되어야 합니다.

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd`(Python:`root_component_versions_to_add`)

(선택 사항) 코어 기기에 설치할 구성 요소 버전. 이 객체`ComponentToVersionMap`, 는 다음과 같은 키-값 쌍을 포함하는 맵입니다.

`key`

구성 요소의 이름입니다.

`value`

구성 요소의 버전입니다.

`rootComponentsToRemove`(Python:`root_components_to_remove`)

(선택 사항) 코어 기기에서 제거할 구성 요소. 각 항목이 구성 요소 이름인 목록을 지정합니다.

`componentToConfiguration`(Python:`component_to_configuration`)

(선택 사항) 배포의 각 구성 요소에 대한 구성 업데이트. 이 객체`ComponentToConfiguration`, 는 다음과 같은 키-값 쌍을 포함하는 맵입니다.

`key`

구성 요소의 이름입니다.

`value`

구성 요소의 구성 업데이트 JSON 개체입니다. JSON 객체의 형식은 다음과 같아야 합니다.

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

구성 업데이트에 대한 자세한 내용은 을 참조하십시오구성 요소 구성 업데이트.

`componentToRunWithInfo`(Python:`component_to_run_with_info`)

(선택 사항) 배포에 포함된 각 구성 요소의 런타임 구성. 이 구성에는 각 구성 요소의 프로세스를 소유한 시스템 사용자와 각 구성 요소에 적용할 시스템 제한이 포함됩니다. 이 개체는 다음과 같은 키-값 쌍을 포함하는 맵입니다. `ComponentToRunWithInfo`

`key`

구성 요소의 이름입니다.

`value`

컴포넌트의 런타임 구성. 런타임 구성 매개변수를 생략하면 AWS IoT Greengrass Core 소프트웨어는 Greengrass 핵에 구성한 기본값을 사용합니다. 이 개체`RunWithInfo`, 에는 다음 정보가 들어 있습니다.

`posixUser`(Python:`posix_user`)

(선택 사항) Linux 코어 기기에서 이 구성 요소를 실행하는 데 사용할 POSIX 시스템 사용자 및 그룹 (선택 사항) 사용자 및 그룹 (지정된 경우) 은 각 Linux 코어 장치에 존재해야 합니다. `user:group` 형식으로 사용자와 그룹을 콜론(`:`)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass Core 소프트웨어는 사용자의 기본 그룹을 사용합니다. 자세한 내용은 구성 요소를 실행하는 사용자를 구성하십시오. 단원을 참조하세요.

`windowsUser`(Python:`windows_user`)

(선택 사항) Windows 코어 장치에서 이 구성 요소를 실행하는 데 사용할 Windows 사용자입니다. 사용자는 각 Windows 코어 장치에 존재해야 하며 사용자 이름과 암호는 LocalSystem 계정의 자격 증명 관리자 인스턴스에 저장되어 있어야 합니다. 자세한 내용은 구성 요소를 실행하는 사용자를 구성하십시오. 단원을 참조하세요.

`systemResourceLimits`(Python:`system_resource_limits`)

(선택 사항) 이 구성 요소의 프로세스에 적용할 시스템 리소스 제한. 시스템 리소스 제한을 일반 및 비컨테이너식 Lambda 구성 요소에 적용할 수 있습니다. 자세한 내용은 구성 요소에 대한 시스템 리소스 제한을 구성합니다. 단원을 참조하세요.

AWS IoT Greengrass현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

이 개체`SystemResourceLimits`, 에는 다음 정보가 들어 있습니다.

`cpus`

(선택 사항) 이 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 장치의 경우 이 값을 2 설정하여 이 구성 요소의 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 기기에서는 이 값을 `0.25` 설정하여 이 구성 요소의 프로세스 CPU 사용량을 25% 로 제한할 수 있습니다. 이 값을 CPU AWS IoT Greengrass 코어 수보다 큰 수로 설정하는 경우 Core 소프트웨어는 구성 요소의 CPU 사용량을 제한하지 않습니다.

`memory`

(선택 사항) 이 구성 요소의 프로세스가 코어 장치에서 사용할 수 있는 최대 RAM 크기(KB) 입니다.

`groupName`(Python:`group_name`)

(선택 사항) 이 배포에서 대상으로 지정할 사물 그룹의 이름.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`deploymentId`(Python:`deployment_id`)

요청으로 생성된 로컬 배포의 ID.

ListLocalDeployments

최근 10개 로컬 배포의 상태를 가져옵니다.

이 작업은 Greengrass CLI의 배포 목록 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`localDeployments`(Python:`local_deployments`)

로컬 배포 목록. 이 목록의 각 개체는 다음 `LocalDeployment` 정보를 포함하는 개체입니다.

`deploymentId`(Python:`deployment_id`)

로컬 디플로이먼트의 ID.

`status`

로컬 배포 상태. 이 `DeploymentStatus` 열거형의 값은 다음과 같습니다.

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

로컬 배포 상태를 가져옵니다.

이 작업은 Greengrass CLI의 배포 상태 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

`deploymentId`(Python:`deployment_id`)

가져올 로컬 디플로이먼트의 ID.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`deployment`

로컬 배포. 이 개체`LocalDeployment`, 에는 다음 정보가 들어 있습니다.

`deploymentId`(Python:`deployment_id`)

로컬 디플로이먼트의 ID.

`status`

로컬 배포 상태. 이 `DeploymentStatus` 열거형의 값은 다음과 같습니다.

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

ListComponents

코어 디바이스에 있는 각 루트 구성 요소의 이름, 버전, 상태 및 구성을 가져옵니다. 루트 구성 요소는 배포 시 지정하는 구성 요소입니다. 이 응답에는 다른 구성 요소의 종속성으로 설치된 구성 요소는 포함되지 않습니다.

이 작업은 Greengrass CLI의 구성 요소 목록 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`components`

코어 디바이스의 루트 구성 요소 목록. 이 목록의 각 개체는 다음 정보를 포함하는 `ComponentDetails` 개체입니다.

`componentName`(Python:`component_name`)

구성 요소의 이름입니다.

`version`

구성 요소의 버전입니다.

`state`

컴포넌트의 상태. 이 상태는 다음 중 하나일 수 있습니다.

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

configuration

JSON 객체로서의 구성 요소 구성

GetComponentDetails

코어 기기에 있는 구성 요소의 버전, 상태 및 구성을 가져옵니다.

이 작업은 Greengrass CLI의 구성 요소 세부 정보 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

componentName(Python:component_name)

가져올 컴포넌트의 이름.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

componentDetails(Python:component_details)

구성 요소 세부 정보. 이 개체ComponentDetails, 에는 다음 정보가 들어 있습니다.

componentName(Python:component_name)

구성 요소의 이름입니다.

`version`

구성 요소의 버전입니다.

`state`

컴포넌트의 상태. 이 상태는 다음 중 하나일 수 있습니다.

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

JSON 객체로서의 구성 요소 구성

RestartComponent

코어 기기에서 구성 요소를 다시 시작합니다.

> **Note**
> 모든 구성 요소를 다시 시작할 수 있지만 일반 구성 요소만 다시 시작하는 것이 좋습니다.

이 작업은 Greengrass CLI의 구성 요소 재시작 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

`componentName`(Python:`component_name`)

구성 요소의 이름입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`restartStatus`(Python:`restart_status`)

재시작 요청 상태. 요청 상태는 다음 중 하나일 수 있습니다.

- `SUCCEEDED`
- `FAILED`

`message`

요청이 실패한 경우 구성 요소를 다시 시작하지 못한 이유에 대한 메시지입니다.

StopComponent

코어 디바이스에서 구성 요소의 프로세스를 중지합니다.

> **Note**
> 모든 구성 요소를 중지할 수 있지만 일반 구성 요소만 중지하는 것이 좋습니다.

이 작업은 Greengrass CLI의 구성 요소 중지 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 다음과 같은 파라미터가 있습니다.

`componentName`(Python:`component_name`)

구성 요소의 이름입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`stopStatus`(Python:`stop_status`)

중지 요청 상태. 요청 상태는 다음 중 하나일 수 있습니다.

- `SUCCEEDED`
- `FAILED`

`message`

요청이 실패한 경우 구성 요소가 중지되지 않은 이유에 대한 메시지입니다.

CreateDebugPassword

로컬 디버그 콘솔 구성 요소에 로그인하는 데 사용할 수 있는 임의의 암호를 생성합니다. 암호는 생성된 후 8시간 후에 만료됩니다.

이 작업은 Greengrass CLI의 get-debug-password 명령과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`username`

로그인하는 데 사용할 사용자 이름.

`password`

로그인할 때 사용할 비밀번호입니다.

`passwordExpiration`(Python:`password_expiration`)

비밀번호가 만료되는 시간.

`certificateSHA256Hash`(Python:`certificate_sha256_hash`)

HTTPS가 활성화되었을 때 로컬 디버그 콘솔이 사용하는 자체 서명 인증서의 SHA-256 지문입니다. 로컬 디버그 콘솔을 열 때 이 지문을 사용하여 인증서가 합법적이고 연결이 안전한지 확인하십시오.

`certificateSHA1Hash`(Python:`certificate_sha1_hash`)

HTTPS가 활성화되었을 때 로컬 디버그 콘솔이 사용하는 자체 서명 인증서의 SHA-1 지문입니다. 로컬 디버그 콘솔을 열 때 이 지문을 사용하여 인증서가 합법적이고 연결이 안전한지 확인하십시오.

클라이언트 장치 인증 및 권한 부여

> **Note**
>
> 이 기능은 v2.6.0 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다.

클라이언트 장치 인증 IPC 서비스를 사용하여 클라이언트 장치와 같은 로컬 IoT 장치가 연결할 수 있는 사용자 지정 로컬 브로커 구성 요소를 개발하십시오.

이러한 IPC 작업을 사용하려면 클라이언트 장치 인증 구성 요소 버전 2.2.0 이상을 사용자 지정 구성 요소의 종속 항목으로 포함해야 합니다. 그런 다음 사용자 지정 구성 요소에서 IPC 작업을 사용하여 다음을 수행할 수 있습니다.

- 코어 디바이스에 연결된 클라이언트 디바이스의 ID를 확인합니다.
- 클라이언트 장치를 코어 장치에 연결할 세션을 생성합니다.
- 클라이언트 장치에 작업을 수행할 권한이 있는지 확인하십시오.
- 코어 디바이스의 서버 인증서가 교체되면 알림을 받습니다.

주제

최소 SDK 버전

다음 표에는 클라이언트 장치 인증 IPC 서비스와 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는 최소 버전이 나와 있습니다.

SDK	최소 버전	
AWS IoT Device SDKJava v2의 경우	v1.9.3	
AWS IoT Device SDK파이썬 v2용	v1.11.3	
AWS IoT Device SDKC++ v2의 경우	v1.18.3	
AWS IoT Device SDKv2의 경우 JavaScript	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 클라이언트 장치 인증 IPC 서비스를 사용하려면 구성 요소가 이러한 작업을 수행할 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 을 참조하십시오. 구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오.

클라이언트 장치 인증 및 권한 부여에 대한 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.clientdevices.Auth`

Operation	설명	리소스
`aws.greengrass#VerifyClientDeviceIdentity`	구성 요소가 클라이언트 장치의 ID를 확인할 수 있도록 합니다.	*
`aws.greengrass#GetClientDeviceAuthToken`	구성 요소가 클라이언트 장치의 자격 증명을 검증하고 해당 클라이언트 장치에 대한 세션을 만들 수 있도록 합니다.	*
`aws.greengrass#AuthorizeClientDeviceAction`	구성 요소가 클라이언트 장치에 작업을 수행할 권한이 있는	*

Operation	설명	리소스
	지 여부를 확인할 수 있도록 합니다.	
`aws.greengrass#SubscribeToCertificateUpdates`	코어 장치의 서버 인증서가 교체될 때 구성 요소가 알림을 받을 수 있도록 합니다.	*
*	구성 요소가 모든 클라이언트 장치 인증 IPC 서비스 작업을 수행할 수 있도록 허용합니다.	*

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하여 구성 요소에 대한 권한 부여 정책을 구성할 수 있습니다.

Example 권한 부여 정책 예시

다음 예제 권한 부여 정책은 구성 요소가 모든 클라이언트 장치 인증 IPC 작업을 수행하도록 허용합니다.

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
 devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

VerifyClientDeviceIdentity

클라이언트 디바이스의 ID를 확인합니다. 이 작업은 클라이언트 장치가 유효한 AWS IoT 장치인지 확인합니다.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`credential`

클라이언트 디바이스의 자격 증명. 이 개체`ClientDeviceCredential`, 에는 다음 정보가 들어 있습니다.

`clientDeviceCertificate`(Python:`client_device_certificate`)

클라이언트 디바이스의 X.509 디바이스 인증서.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`isValidClientDevice`(Python:`is_valid_client_device`)

클라이언트 장치의 ID가 유효한지 여부.

GetClientDeviceAuthToken

클라이언트 장치의 자격 증명을 확인하고 클라이언트 장치에 대한 세션을 만듭니다. 이 작업은 후속 요청에서 클라이언트 장치 작업을 승인하는 데 사용할 수 있는 세션 토큰을 반환합니다.

클라이언트 장치를 성공적으로 연결하려면 클라이언트 장치 인증 구성 요소가 클라이언트 장치가 사용하는 클라이언트 ID에 대한 `mqtt:connect` 권한을 부여해야 합니다.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`credential`

클라이언트 디바이스의 자격 증명. 이 개체`CredentialDocument`, 에는 다음 정보가 들어 있습니다.

`mqttCredential`(Python:`mqtt_credential`)

클라이언트 디바이스의 MQTT 자격 증명. 클라이언트 장치가 연결하는 데 사용하는 클라이언트 ID 및 인증서를 지정합니다. 이 개체에는 다음 정보가 들어 `MQTTCredential` 있습니다.

`clientId`(Python:`client_id`)

연결하는 데 사용할 클라이언트 ID.

`certificatePem`(Python:`certificate_pem`)

연결에 사용할 X.509 디바이스 인증서.

`username`

> **Note**
> 이 속성은 현재 사용되지 않습니다.

`password`

> **Note**
> 이 속성은 현재 사용되지 않습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`clientDeviceAuthToken`(Python:`client_device_auth_token`)

클라이언트 디바이스의 세션 토큰. 후속 요청에서 이 세션 토큰을 사용하여 이 클라이언트 장치의 작업을 승인할 수 있습니다.

AuthorizeClientDeviceAction

클라이언트 장치에 리소스에 대한 작업을 수행할 권한이 있는지 확인하십시오. 클라이언트 장치 인증 정책은 클라이언트 장치가 핵심 장치에 연결되어 있는 동안 수행할 수 있는 권한을 지정합니다. 클라이언트 장치 인증 구성 요소를 구성할 때 클라이언트 장치 권한 부여 정책을 정의합니다.

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`clientDeviceAuthToken`(Python:`client_device_auth_token`)

클라이언트 디바이스의 세션 토큰.

`operation`

권한을 부여하는 작업.

`resource`

클라이언트 장치가 작업을 수행하는 리소스입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`isAuthorized`(Python:`is_authorized`)

클라이언트 기기가 리소스에서 작업을 수행할 권한이 있는지 여부.

SubscribeToCertificateUpdates

코어 디바이스가 교체될 때마다 코어 디바이스의 새 서버 인증서를 받으려면 구독하십시오. 서버 인증서가 교체되면 브로커는 새 서버 인증서를 사용하여 다시 로드해야 합니다.

클라이언트 장치 인증 구성 요소는 기본적으로 7일마다 서버 인증서를 교체합니다. 순환 간격을 2일에서 10일 사이로 구성할 수 있습니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 폐쇄를 처리하는 함수가 포함된 스트림 응답 핸들러를 정의하십시오. 자세한 설명은 IPC 이벤트 스트림을 구독하세요. 섹션을 참조하세요.

이벤트 메시지 유형: `CertificateUpdateEvent`

요청

이 작업의 요청에는 다음과 같은 매개변수가 있습니다.

`certificateOptions`(Python:`certificate_options`)

구독할 인증서 업데이트 유형. 이 개체`CertificateOptions`, 에는 다음 정보가 들어 있습니다.

`certificateType`(Python:`certificate_type`)

구독할 인증서 업데이트 유형. 다음 옵션을 선택합니다.

- `SERVER`

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

메시지 스트림. 이 개체`CertificateUpdateEvent`, 에는 다음 정보가 들어 있습니다.

`certificateUpdate`(Python:`certificate_update`)

새 인증서에 대한 정보. 이 개체`CertificateUpdate`, 에는 다음 정보가 들어 있습니다.

`certificate`

인증서입니다.

`privateKey`(Python:`private_key`)

인증서의 개인 키.

`publicKey`(Python:`public_key`)

인증서의 공개 키.

`caCertificates`(Python:`ca_certificates`)

인증서의 CA 인증서 체인에 있는 인증 기관 (CA) 인증서 목록.

로컬 IoT 기기와 상호작용

클라이언트 디바이스는 MQTT를 통해 Greengrass 코어 디바이스에 연결하고 통신하는 로컬 IoT 디바이스입니다. 클라이언트 디바이스를 코어 디바이스에 연결하여 다음 작업을 수행할 수 있습니다.

- Greengrass 컴포넌트에서 MQTT 메시지와 상호 작용합니다.
- 클라이언트 장치 간에 메시지와 데이터를 중계하고. AWS IoT Core
- Greengrass 구성 요소의 클라이언트 장치 섀도우와 상호 작용합니다.
- 클라이언트 디바이스 섀도우를 와 AWS IoT Core 동기화합니다.

클라이언트 디바이스는 클라우드 검색을 사용하여 코어 디바이스에 연결할 수 있습니다. 클라이언트 장치는 AWS IoT Greengrass 클라우드 서비스에 연결하여 연결할 수 있는 핵심 장치에 대한 정보를 검색합니다. 그런 다음 코어 디바이스에 연결하여 메시지를 처리하고 데이터를 AWS IoT Core 클라우드 서비스와 동기화할 수 있습니다.

사물과 연결 및 통신하도록 코어 장치를 구성하는 방법을 안내하는 자습서를 따를 수 AWS IoT 있습니다. 또한 이 자습서에서는 클라이언트 장치와 상호 작용하는 사용자 지정 Greengrass 구성 요소를 개발하는 방법도 살펴봅니다. 자세한 내용은 튜토리얼: MQTT를 통해 로컬 IoT 디바이스와 상호 작용을(를) 참조하세요.

주제

AWS-제공된 클라이언트 장치 구성 요소

AWS IoT Greengrass핵심 장치에 배포할 수 있는 다음과 같은 공용 구성 요소를 제공합니다. 이러한 구성 요소를 사용하면 클라이언트 장치가 코어 장치에 연결하고 통신할 수 있습니다.

> **Note**
>
> AWS제공되는 몇 가지 구성 요소는 Greengrass 핵의 특정 마이너 버전에 따라 다릅니다. 이러한 종속성 때문에 Greengrass 핵을 새 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 의존하는 핵의 특정 버전에 대한 자세한 내용은 해당 구성 요소 항목을 참조하십시오. 핵 업데이트에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트

구성 요소의 구성 요소 유형이 제네릭과 Lambda인 경우 구성 요소의 현재 버전은 제네릭 유형이고 구성 요소의 이전 버전은 Lambda 유형입니다.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
클라이언트 장치 인증	클라이언트 디바이스라고 하는 로컬 IoT 디바이스를 코어 디바이스에 연결할 수 있도록 합니다.	플러그인	Linux, Windows	예
IP 감지기	MQTT 브로커 연결 정보를 에 보고하여 AWS IoT Greengrass 클라이언트 장치가 연결 방법을 찾을 수 있도록 합니다.	플러그인	Linux, Windows	예
MQTT 브리지	클라이언트 장치, 로컬 AWS IoT Greengrass 게시/구독 및 간에 MQTT 메	플러그인	Linux, Windows	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
	시지를 중계합니다. AWS IoT Core			
MQTT 3.1.1 브로커 (모켓)	클라이언트 디바이스와 코어 디바이스 간의 메시지를 처리하는 MQTT 3.1.1 브로커를 실행합니다.	플러그인	Linux, Windows	예
맷 5 브로커 (EMQX)	클라이언트 디바이스와 코어 디바이스 간의 메시지를 처리하는 MQTT 5 브로커를 실행합니다.	일반	Linux, Windows	아니요
섀도우 매니저	코어 디바이스의 섀도우와 상호 작용할 수 있습니다. 섀도우 문서 스토리지를 관리하고 로컬 섀도우 상태를 AWS IoT Device Shadow 서비스와 동기화하는 작업도 관리합니다.	플러그인	Linux, Windows	예

클라이언트 장치를 코어 장치에 연결

클라이언트 디바이스를 코어 디바이스에 연결하도록 클라우드 검색을 구성할 수 있습니다. 클라우드 검색을 구성하면 클라이언트 장치를 AWS IoT Greengrass 클라우드 서비스에 연결하여 연결할 수 있는 핵심 장치에 대한 정보를 검색할 수 있습니다. 그러면 클라이언트 디바이스는 성공적으로 연결될 때까지 각 코어 디바이스에 연결을 시도할 수 있습니다.

클라우드 검색을 사용하려면 다음을 수행해야 합니다.

- 클라이언트 디바이스를 연결할 수 있는 코어 디바이스에 연결합니다.
- 클라이언트 디바이스가 각 코어 디바이스에 연결할 수 있는 MQTT 브로커 엔드포인트를 지정합니다.
- 클라이언트 장치를 지원할 수 있는 구성 요소를 코어 장치에 배포하십시오.

  선택적 구성 요소를 배포하여 다음 작업을 수행할 수도 있습니다.

  - 클라이언트 장치, Greengrass 구성 요소 및 AWS IoT Core 클라우드 서비스 간에 메시지를 중계합니다.
  - 코어 디바이스 MQTT 브로커 엔드포인트를 자동으로 관리하세요.
  - 로컬 클라이언트 디바이스 섀도를 관리하고 섀도우를 클라우드 서비스와 동기화합니다. AWS IoT Core

또한 코어 디바이스의 AWS IoT 정책을 검토 및 업데이트하여 코어 디바이스에 클라이언트 디바이스를 연결하는 데 필요한 권한이 있는지 확인해야 합니다. 자세한 설명은 요구 사항 섹션을 참조하세요.

클라우드 검색을 구성한 후 클라이언트 장치와 코어 장치 간의 통신을 테스트할 수 있습니다. 자세한 설명은 클라이언트 장치 통신 테스트 섹션을 참조하세요.

주제

- MQTT 브로커를 선택하세요
- MQTT 브로커를 사용하여 클라이언트 장치를AWS IoT Greengrass 코어 장치에 연결
- 클라이언트 장치 통신 테스트
- 그린그래스 디스커버리 RESTful API

요구 사항

클라이언트 디바이스를 코어 디바이스에 연결하려면 다음이 있어야 합니다.

- 코어 디바이스는 Greengrass nucleus v2.2.0 이상을 실행해야 합니다.
- 핵심 장치가 운영되는 AWS 계정 AWS 지역에서 귀하와 관련된 AWS IoT Greengrass Greengrass 서비스 역할. 자세한 설명은 Greengrass 서비스 역할 구성 섹션을 참조하세요.
- 코어 디바이스의 AWS IoT 정책은 다음 권한을 허용해야 합니다.
  - `greengrass:PutCertificateAuthorities`
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
  - `greengrass:GetConnectivityInfo`
  - `greengrass:UpdateConnectivityInfo`— (선택 사항) 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 IP Detector 구성 요소를 사용하려면 이 권한이 필요합니다.
  - `iot:GetThingShadow`, `iot:UpdateThingShadow`, 및 `iot:DeleteThingShadow` — (선택 사항) Shadow Manager 구성 요소를 사용하여 클라이언트 장치 섀도우를 동기화하려면 이러한 권한이 필요합니다AWS IoT Core. 이 기능을 사용하려면 그린그래스 뉴클리우스 v2.6.0 이상, 섀도우 매니저 v2.2.0 이상, MQTT 브리지 v2.2.0 이상이 필요합니다.

  자세한 설명은 AWS IoT사물 정책을 구성합니다. 섹션을 참조하세요.

  > **Note**
  >
  > AWS IoT GreengrassCore 소프트웨어를 설치할 때 기본 AWS IoT 정책을 사용한 경우 코어 디바이스에는 모든 작업에 대한 액세스를 허용하는 정책이 적용됩니다 ()AWS IoT. AWS IoT Greengrass `greengrass:*`
- AWS IoT클라이언트 장치로 연결할 수 있는 항목. 자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT 리소스 만들기를 참조하십시오.

- 클라이언트 장치는 클라이언트 ID를 사용하여 연결해야 합니다. 클라이언트 ID는 사물 이름입니다. 다른 클라이언트 ID는 허용되지 않습니다.
- 각 클라이언트 장치의 AWS IoT 정책에서 `greengrass:Discover` 권한을 허용해야 합니다. 자세한 설명은 클라이언트 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

주제

- Greengrass 서비스 역할 구성
- AWS IoT사물 정책을 구성합니다.

Greengrass 서비스 역할 구성

Greengrass 서비스 역할은 사용자를 대신하여 AWS 서비스의 리소스에 액세스할 수 있도록 AWS IoT Greengrass에 권한을 부여하는 AWS Identity and Access Management (IAM) 서비스 역할입니다. 이 역할을 통해 클라이언트 장치의 ID를 확인하고 핵심 장치 연결 정보를 관리할 수 있습니다. AWS IoT Greengrass

이 지역에서 이전에 Greengrass 서비스 역할을 설정하지 않은 경우, 이 지역의 Greengrass 서비스 역할을 자신의 AWS 계정 서비스 역할과 AWS IoT Greengrass 연결해야 합니다.

AWS IoT Greengrass콘솔에서 코어 장치 검색 구성 페이지를 사용하는 경우 Greengrass 서비스 역할을 자동으로 AWS IoT Greengrass 설정합니다. 그렇지 않으면 AWS IoT콘솔이나 AWS IoT Greengrass API를 사용하여 수동으로 설정할 수 있습니다.

이 섹션에서는 Greengrass 서비스 역할이 설정되었는지 확인합니다. 설정되지 않은 경우 이 AWS 계정 지역에서 연결할 새 Greengrass 서비스 역할을 생성합니다. AWS IoT Greengrass

Greengrass 서비스 역할 구성 (콘솔)

1. 이 AWS 계정 지역에서 Greengrass 서비스 역할이 귀사와 AWS IoT Greengrass 연결되어 있는지 확인하십시오. 다음을 따릅니다.

   a. AWS IoT 콘솔로 이동합니다.
   b. 탐색 창에서 설정을 선택합니다.
   c. Greengrass 서비스 역할 섹션에서 현재 서비스 역할을 찾아 Greengrass 서비스 역할이 연결되어 있는지 확인하세요.

      Greengrass 서비스 역할이 연결되어 있는 경우 IP 탐지기 구성 요소를 사용하려면 이 요구 사항을 충족해야 합니다. AWS IoT사물 정책을 구성합니다.로 이동하세요.

2. Greengrass 서비스 역할이 해당 AWS 계정 지역의 사용자와 AWS IoT Greengrass 연결되어 있지 않은 경우 Greengrass 서비스 역할을 만들어 연결하세요. 다음을 따릅니다.

   a. IAM 콘솔로 이동합니다.

   b. [Roles]를 선택합니다.

   c. Create role(역할 생성)을 선택합니다.

   d. 역할 생성 페이지에서 다음을 수행하십시오.

      i. 신뢰할 수 있는 엔티티 유형에서 선택합니다 AWS 서비스.

      ii. 사용 사례 AWS 서비스, 기타 사용 사례에서 Greengrass를 선택하고 Greengrass를 선택합니다. 이 옵션은 이 AWS IoT Greengrass 역할을 맡을 수 있는 신뢰할 수 있는 엔티티로 추가하도록 지정합니다.

      iii. 다음을 선택합니다.

      iv. 권한 정책에서 역할에 AWSGreengrassResourceAccessRolePolicy연결할 항목을 선택합니다.

      v. 다음을 선택합니다.

      vi. 역할 이름에 역할 이름 (예:) 을 입력합니다**`Greengrass_ServiceRole`**.

      vii. Create role(역할 생성)을 선택합니다.

   e. AWS IoT 콘솔로 이동합니다.

   f. 탐색 창에서 설정을 선택합니다.

   g. Greengrass 서비스 역할 섹션에서 역할 연결을 선택합니다.

   h. Greengrass 서비스 역할 업데이트 모달에서 생성한 IAM 역할을 선택한 다음 역할 연결을 선택합니다.

Greengrass 서비스 역할 구성 () AWS CLI

1. 이 AWS 계정 지역에서 Greengrass 서비스 역할이 귀사와 AWS IoT Greengrass 연결되어 있는지 확인하십시오.

   ```
   aws greengrassv2 get-service-role-for-account
   ```

   Greengrass 서비스 역할이 연결된 경우 작업은 역할에 대한 정보가 포함된 응답을 반환합니다.

   Greengrass 서비스 역할이 연결되어 있는 경우 IP 탐지기 구성 요소를 사용하려면 이 요구 사항을 충족해야 합니다. AWS IoT사물 정책을 구성합니다.로 이동하세요.

2. Greengrass 서비스 역할이 해당 AWS 계정 지역의 사용자와 AWS IoT Greengrass 연결되어 있지 않은 경우 Greengrass 서비스 역할을 만들어 연결하세요. 다음을 따릅니다.

   a. 이 역할을 수임하도록 허용하는 AWS IoT Greengrass 신뢰 정책으로 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 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": {
              "ArnLike": {
                "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
              },
              "StringEquals": {
                "aws:SourceAccount": "account-id"
              }
            }
          }
        ]
      }'
      ```

      Windows Command Prompt (CMD)

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

PowerShell

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

b. 출력의 역할 메타데이터에서 역할 ARN을 복사합니다. ARN을 사용하여 역할을 계정과 연결합니다.

c. `AWSGreengrassResourceAccessRolePolicy` 정책을 역할에 연결합니다.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

d. Greengrass 서비스 역할을 귀사와 연계하십시오AWS IoT Greengrass. AWS 계정 *role-arn#* 서비스 역할의 ARN으로 대체합니다.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

작업이 성공하면 다음 응답을 반환합니다.

```
{
  "associatedAt": "timestamp"
}
```

AWS IoT사물 정책을 구성합니다.

코어 디바이스는 X.509 디바이스 인증서를 사용하여 연결을 승인합니다. AWS AWS IoT정책을 장치 인증서에 연결하여 코어 장치에 대한 권한을 정의합니다. 자세한 내용은 데이터 영역 작업에 대한 AWS IoT 정책 및 클라이언트 AWS IoT 장치를 지원하기 위한 최소 정책 섹션을 참조하세요.

클라이언트 디바이스를 코어 디바이스에 연결하려면 코어 디바이스의 AWS IoT 정책에서 다음 권한을 허용해야 합니다.

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (선택 사항) 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 IP Detector 구성 요소를 사용하려면 이 권한이 필요합니다.
- `iot:GetThingShadow`, `iot:UpdateThingShadow`, 및 `iot:DeleteThingShadow` — (선택 사항) Shadow Manager 구성 요소를 사용하여 클라이언트 장치 섀도우를 동기화하려면 이러한 권한이 필요합니다AWS IoT Core. 이 기능을 사용하려면 그린그래스 뉴클리우스 v2.6.0 이상, 섀도우 매니저 v2.2.0 이상, MQTT 브리지 v2.2.0 이상이 필요합니다.

이 섹션에서는 코어 AWS IoT 디바이스의 정책을 검토하고 누락된 필수 권한을 추가합니다. AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 코어 장치에는 모든 AWS IoT Greengrass 작업에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다 (`greengrass:*`). 이 경우 섀도우 관리자 구성 요소를 배포하여 장치 섀도우를 동기화하려는 경우에만 AWS IoT 정책을 업데이트해야 합니다. AWS IoT Core 그렇지 않으면 이 섹션을 건너뛰어도 됩니다.

AWS IoT사물 정책 구성 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 코어 디바이스를 선택합니다.

2. 코어 디바이스 페이지에서 업데이트할 코어 디바이스를 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 코어 디바이스의 사물로 연결되는 링크를 선택합니다. 이 링크를 클릭하면 AWS IoT 콘솔에서 사물 세부 정보 페이지가 열립니다.
4. 사물 세부 정보 페이지에서 인증서를 선택합니다.
5. 인증서 탭에서 사물의 활성 인증서를 선택합니다.
6. 인증서 세부 정보 페이지에서 정책을 선택합니다.
7. 정책 탭에서 검토 및 업데이트할 AWS IoT 정책을 선택합니다. 코어 디바이스의 활성 인증서에 연결된 모든 정책에 필요한 권한을 추가할 수 있습니다.

   > **Note**
   >
   > AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 지정된 정책을 선택하는 것이 좋습니다 (있는 GreengrassV2IoTThingPolicy경우). 빠른 설치 프로그램으로 만든 코어 디바이스는 기본적으로 이 정책 이름을 사용합니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 핵심 장치에도 이러한 권한이 부여됩니다.

8. 정책 개요에서 활성 버전 편집을 선택합니다.
9. 정책에서 필요한 권한을 검토하고 누락된 필수 권한을 추가하십시오.

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`— (선택 사항) 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 IP Detector 구성 요소를 사용하려면 이 권한이 필요합니다.
   - `iot:GetThingShadow`, `iot:UpdateThingShadow`, 및 `iot:DeleteThingShadow` — (선택 사항) Shadow Manager 구성 요소를 사용하여 클라이언트 장치 섀도우를 동기화하려면 이러한 권한이 필요합니다AWS IoT Core. 이 기능을 사용하려면 그린그래스 뉴클리우스 v2.6.0 이상, 섀도우 매니저 v2.2.0 이상, MQTT 브리지 v2.2.0 이상이 필요합니다.

10. (선택 사항) 코어 디바이스가 섀도우를 동기화할 수 있도록 하려면 정책에 다음 설명을 추가하십시오. AWS IoT Core 클라이언트 장치 섀도우와 상호 작용하지만 동기화하지는 않으려는 경우 이 단계를 건너뛰십시오. AWS IoT Core *##* 및 *## ID#* 사용하는 지역 및 사용자 번호로 바꾸십시오. AWS 계정

- 이 예제 명령문을 사용하면 모든 사물의 디바이스 섀도우에 액세스할 수 있습니다. 최상의 보안 관행을 따르기 위해 코어 장치 및 핵심 장치에 연결하는 클라이언트 장치로만 액세스를 제한할 수 있습니다. 자세한 설명은 클라이언트 AWS IoT 장치를 지원하기 위한 최소 정책 섹션을 참조하세요.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

이 설명을 추가하면 정책 문서가 다음 예와 비슷해 보일 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
```

```
      "Resource": [
        "arn:aws:iot:region:account-id:thing/*"
      ]
    }
  ]
}
```

11. 새 정책 버전을 활성 버전으로 설정하려면 정책 버전 상태에서 편집한 버전을 이 정책의 활성 버전으로 설정을 선택합니다.
12. 새 버전으로 저장을 선택합니다.

AWS IoT사물 정책 구성 (AWS CLI)

1. 핵심 장치 사물의 보안 주체를 나열하십시오. AWS IoT 사물 주체는 X.509 장치 인증서 또는 기타 식별자일 수 있습니다. 다음 명령을 실행하고 코어 디바이스의 이름으로 *MyGreengrassCore*바꿉니다.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   이 작업은 코어 디바이스의 사물 주체를 나열하는 응답을 반환합니다.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. 코어 디바이스의 활성 인증서를 식별하십시오. 다음 명령을 실행하고 활성 *#### ## ### CertificateID#* 이전 단계의 각 인증서 ID로 바꿉니다. 인증서 ID는 인증서 ARN 끝에 있는 16진수 문자열입니다. `--query`인수는 인증서 상태만 출력하도록 지정합니다.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   작업은 인증서 상태를 문자열로 반환합니다. 예를 들어 인증서가 활성 상태인 경우 이 작업이 `"ACTIVE"` 출력됩니다.

3. 인증서에 첨부된 AWS IoT 정책을 나열하십시오. 다음 명령을 실행하고 인증서 ARN을 인증서의 ARN으로 교체합니다.

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

이 작업은 인증서에 연결된 AWS IoT 정책을 나열하는 응답을 반환합니다.

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. 보고 업데이트할 정책을 선택합니다.

> **Note**
>
> AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 지정된 정책을 선택하는 것이 좋습니다 (있는 GreengrassV2IoTThingPolicy경우). 빠른 설치 프로그램으로 만든 코어 디바이스는 기본적으로 이 정책 이름을 사용합니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 핵심 장치에도 이러한 권한이 부여됩니다.

5. 정책 문서를 가져오세요. 다음 명령을 실행하고 *ThingPolicyGreenGrassV2IoT#* 정책 이름으로 대체합니다.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

이 작업은 정책 문서와 정책에 대한 기타 정보가 포함된 응답을 반환합니다. 정책 문서는 문자열로 직렬화된 JSON 객체입니다.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. 온라인 변환기나 다른 도구를 사용하여 정책 문서 문자열을 JSON 객체로 변환한 다음 이 문자열을 이름이 지정된 파일에 저장합니다. `iot-policy.json`

   예를 들어 jq 도구가 설치되어 있는 경우 다음 명령을 실행하여 정책 문서를 가져와서 JSON 개체로 변환한 다음 정책 문서를 JSON 개체로 저장할 수 있습니다.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. 정책에서 필요한 권한을 검토하고 누락된 필수 권한을 추가하십시오.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 열 수 있습니다.

```
nano iot-policy.json
```

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (선택 사항) 코어 디바이스의 네트워크 연결 정보를 클라우드 서비스에 보고하는 IP Detector 구성 요소를 사용하려면 이 권한이 필요합니다. AWS IoT Greengrass
- `iot:GetThingShadow`, `iot:UpdateThingShadow`, 및 `iot:DeleteThingShadow` — (선택 사항) Shadow Manager 구성 요소를 사용하여 클라이언트 장치 섀도우를 동기화하려면 이러한 권한이 필요합니다AWS IoT Core. 이 기능을 사용하려면 그린그래스 뉴클리우스 v2.6.0 이상, 섀도우 매니저 v2.2.0 이상, MQTT 브리지 v2.2.0 이상이 필요합니다.

8. 변경 내용을 새 버전의 정책으로 저장합니다. 다음 명령을 실행하고 *ThingPolicyGreenGrassV2IoT#* 정책 이름으로 대체합니다.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

작업이 성공하면 다음 예제와 비슷한 응답이 반환됩니다.

```
{
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
```

```
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

클라이언트 장치 지원을 위한 Greengrass 구성 요소

> **Important**
>
> 코어 디바이스는 Greengrass nucleus v2.2.0 이상을 실행해야 클라이언트 디바이스를 지원할 수 있습니다.

클라이언트 장치가 코어 장치와 연결 및 통신할 수 있도록 하려면 다음 Greengrass 구성 요소를 코어 장치에 배포합니다.

- 클라이언트 장치 인증 (`aws.greengrass.clientdevices.Auth`)

  클라이언트 장치 인증 구성 요소를 배포하여 클라이언트 장치를 인증하고 클라이언트 장치 작업을 승인합니다. 이 구성 요소를 사용하면 AWS IoT 사물을 코어 장치에 연결할 수 있습니다.

  이 구성 요소를 사용하려면 몇 가지 구성이 필요합니다. 클라이언트 장치 그룹과 각 그룹이 수행할 권한이 있는 작업 (예: MQTT를 통한 연결 및 통신) 을 지정해야 합니다. 자세한 내용은 클라이언트 장치 인증 구성 요소 구성을 참조하십시오.
- MQTT 3.1.1 브로커 (모켓) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Mocquette MQTT 브로커 구성 요소를 배포하여 경량 MQTT 브로커를 실행하십시오. Moquette MQTT 브로커는 MQTT 3.1.1과 호환되며 QoS 0, QoS 1, QoS 2, 보존 메시지, 라스트 윌 메시지 및 영구 구독에 대한 로컬 지원을 포함합니다.

  이 구성 요소를 사용하도록 구성하지 않아도 됩니다. 하지만 이 구성 요소가 MQTT 브로커를 작동하는 포트를 구성할 수 있습니다. 기본적으로 포트 8883을 사용합니다.
- 맷 5 브로커 (EMQX) (`aws.greengrass.clientdevices.mqtt.EMQX`)

> **Note**
>
> EMQX MQTT 5 브로커를 사용하려면 그린그래스 뉴클레우스 v2.6.0 이상 및 클라이언트 장치 인증 v2.2.0 이상을 사용해야 합니다.

EMQX MQTT 브로커 구성 요소를 배포하여 클라이언트 장치와 코어 장치 간 통신에 MQTT 5.0 기능을 사용할 수 있습니다. EMQX MQTT 브로커는 MQTT 5.0과 호환되며 세션 및 메시지 만료 간격, 사용자 속성, 공유 구독, 주제 별칭 등에 대한 지원을 포함합니다.

이 구성 요소를 사용하도록 구성하지 않아도 됩니다. 하지만 이 구성 요소가 MQTT 브로커를 작동하는 포트를 구성할 수 있습니다. 기본적으로 포트 8883을 사용합니다.

- MQTT 브리지 (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (선택 사항) MQTT 브리지 구성 요소를 배포하여 클라이언트 장치 (로컬 MQTT), 로컬 게시/구독 및 MQTT 간에 메시지를 릴레이합니다. AWS IoT Core 클라이언트 장치를 Greengrass 구성 요소의 클라이언트 장치와 AWS IoT Core 동기화하고 클라이언트 장치와 상호 작용하도록 이 구성 요소를 구성합니다.

  이 구성 요소를 사용하려면 구성이 필요합니다. 이 구성 요소가 메시지를 릴레이하는 주제 매핑을 지정해야 합니다. 자세한 내용은 MQTT 브리지 구성 요소 구성을 참조하십시오.

- IP 감지기 (`aws.greengrass.clientdevices.IPDetector`)

  (선택 사항) 코어 디바이스의 MQTT 브로커 엔드포인트를 클라우드 서비스에 자동으로 보고하도록 IP 탐지기 구성 요소를 배포하십시오AWS IoT Greengrass. 라우터가 MQTT 브로커 포트를 코어 디바이스에 전달하는 것과 같이 복잡한 네트워크 설정이 있는 경우에는 이 구성 요소를 사용할 수 없습니다.

  이 구성 요소를 사용하도록 구성하지 않아도 됩니다.

- 섀도우 매니저 (`aws.greengrass.ShadowManager`)

> **Note**
>
> 클라이언트 디바이스 섀도를 관리하려면 Greengrass nucleus v2.6.0 이상, 섀도우 관리자 v2.2.0 이상, MQTT 브리지 v2.2.0 이상을 사용해야 합니다.

(선택 사항) 섀도우 관리자 구성 요소를 배포하여 코어 장치의 클라이언트 장치 섀도를 관리합니다. Greengrass 구성 요소는 클라이언트 장치 섀도를 가져오고 업데이트하고 삭제하여 클라이언트 장치와 상호 작용할 수 있습니다. 또한 클라이언트 장치 섀도우를 AWS IoT Core 클라우드 서비스와 동기화하도록 섀도우 관리자 구성 요소를 구성할 수 있습니다.

이 구성 요소를 클라이언트 장치 섀도와 함께 사용하려면 로컬 게시/구독을 사용하는 섀도우 관리자와 클라이언트 장치 간에 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성해야 합니다. 그렇지 않으면 이 구성 요소를 사용하기 위한 구성이 필요하지 않지만 장치 섀도우를 동기화하기 위한 구성은 필요합니다.

> **Note**
>
> MQTT 브로커 구성 요소를 하나만 배포하는 것이 좋습니다. MQTT 브리지 및 IP 탐지기 구성 요소는 한 번에 하나의 MQTT 브로커 구성 요소에서만 작동합니다. 여러 MQTT 브로커 구성 요소를 배포하는 경우 서로 다른 포트를 사용하도록 구성해야 합니다.

클라우드 디스커버리 구성 (콘솔)

AWS IoT Greengrass콘솔을 사용하여 클라이언트 장치를 연결하고, 핵심 장치 엔드포인트를 관리하고, 구성 요소를 배포하여 클라이언트 장치 지원을 활성화할 수 있습니다. 자세한 설명은 2단계: 클라이언트 장치 지원 활성화 섹션을 참조하세요.

클라우드 디스커버리 구성 () AWS CLI

AWS Command Line Interface(AWS CLI) 를 사용하여 클라이언트 장치를 연결하고, 핵심 장치 엔드포인트를 관리하고, 구성 요소를 배포하여 클라이언트 장치 지원을 활성화할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결 관리 () AWS CLI
- 코어 디바이스 엔드포인트 관리
- AWS-제공된 클라이언트 장치 구성 요소
- 배포 만들기

클라이언트 장치 연결

클라우드 검색을 사용하려면 클라이언트 장치를 코어 장치에 연결하여 핵심 장치를 검색할 수 있도록 하십시오. 그런 다음 Greengrass 검색 API를 사용하여 관련 코어 디바이스의 연결 정보와 인증서를 검색할 수 있습니다.

마찬가지로 클라이언트 디바이스를 코어 디바이스에서 분리하여 코어 디바이스를 검색하지 못하게 하십시오.

주제

- 클라이언트 장치 연결 관리 (콘솔)
- 클라이언트 장치 연결 관리 () AWS CLI
- 클라이언트 장치 연결 (API) 관리

클라이언트 장치 연결 관리 (콘솔)

AWS IoT Greengrass콘솔을 사용하여 클라이언트 장치 연결을 보고, 추가하고, 삭제할 수 있습니다.

코어 장치 (콘솔) 에 대한 클라이언트 장치 연결을 보려면

1. AWS IoT Greengrass 콘솔로 이동합니다.
2. 코어 디바이스를 선택합니다.
3. 관리할 코어 디바이스를 선택합니다.
4. 코어 디바이스의 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. 연결된 클라이언트 장치 섹션에서 코어 장치와 연결된 클라이언트 장치 (AWS IoT사물) 를 확인할 수 있습니다.

클라이언트 장치를 코어 장치 (콘솔) 에 연결하려면

1. AWS IoT Greengrass 콘솔로 이동합니다.
2. 코어 디바이스를 선택합니다.
3. 관리할 코어 디바이스를 선택합니다.
4. 코어 디바이스의 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. 연결된 클라이언트 디바이스 부분에서, 클라이언트 디바이스 연결을 선택합니다.
6. 클라이언트 디바이스를 코어 디바이스와 연결 모달에서, 연결할 각 클라이언트 디바이스에 대해 다음을 수행하세요.

a. 클라이언트 장치로 연결할 AWS IoT 사물의 이름을 입력합니다.

b. Add(추가)를 선택합니다.

7. Associate(연결)를 선택합니다.

   연결한 클라이언트 디바이스는 이제 Greengrass 검색 API를 사용하여 이 코어 디바이스를 검색할 수 있습니다.

코어 디바이스 (콘솔) 에서 클라이언트 디바이스의 연결을 끊으려면

1. AWS IoT Greengrass 콘솔로 이동합니다.
2. 코어 디바이스를 선택합니다.
3. 관리할 코어 디바이스를 선택합니다.
4. 코어 디바이스의 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. 연결된 클라이언트 장치 섹션에서 연결을 끊을 각 클라이언트 장치를 선택합니다.
6. 연결 해제를 선택합니다.
7. 확인 모드에서 연결 해제를 선택합니다.

   연결을 끊은 클라이언트 디바이스는 더 이상 Greengrass 검색 API를 사용하여 이 코어 디바이스를 검색할 수 없습니다.

클라이언트 장치 연결 관리 () AWS CLI

AWS Command Line Interface(AWS CLI) 를 사용하여 코어 장치의 클라이언트 장치 연결을 관리할 수 있습니다.

코어 장치의 클라이언트 장치 연결을 보려면 (AWS CLI)

- 다음 명령을 사용하세요with-core-device. list-client-devices-associated-.

클라이언트 장치를 코어 장치에 연결하려면 (AWS CLI)

- 다음 명령을 사용하십시오with-core-device. batch-associate-client-device-.

코어 디바이스에서 클라이언트 디바이스의 연결을 끊으려면 () AWS CLI

- 다음 명령을 사용하십시오. batch-disassociate-client-device- from-core-device.

클라이언트 장치 연결 (API) 관리

AWSAPI를 사용하여 코어 장치의 클라이언트 장치 연결을 관리할 수 있습니다.

코어 장치 (AWSAPI) 에 대한 클라이언트 장치 연결을 보려면

- 다음 작업을 사용하십시오 ListClientDevicesAssociatedWithCoreDevice.

클라이언트 장치를 코어 장치 (AWSAPI) 에 연결하려면

- 다음 작업을 사용하십시오 BatchAssociateClientDeviceWithCoreDevice.

클라이언트 장치를 코어 장치 (AWSAPI) 에서 분리하려면

- 다음 BatchDisassociateClientDeviceFromCoreDevice작업을 사용하십시오.

오프라인 상태에서 클라이언트 인증

오프라인 인증을 사용하면AWS IoT Greengrass 코어 디바이스가 클라우드에 연결되어 있지 않아도 클라이언트 디바이스가 코어 디바이스에 연결될 수 있도록 코어 디바이스를 구성할 수 있습니다. 오프라인 인증을 사용하는 경우 Greengrass 장치는 부분적으로 오프라인 환경에서도 계속 작동할 수 있습니다.

클라우드에 연결된 클라이언트 장치에 오프라인 인증을 사용하려면 다음이 필요합니다.

- 클라이언트 장치 인증구성 요소가 배포된AWS IoT Greengrass 코어 장치. 오프라인 인증에는 버전 2.3.0 이상을 사용해야 합니다.
- 클라이언트 장치를 처음 연결하는 동안 코어 장치를 위한 클라우드 연결.

클라이언트 자격 증명 저장

클라이언트 디바이스가 코어 디바이스에 처음 연결되면 코어 디바이스가AWS IoT Greengrass 서비스를 호출합니다. Greengrass는 호출되면 클라이언트 디바이스의 등록을 사물로AWS IoT 검증합니다. 또한 장치에 유효한 인증서가 있는지 확인합니다. 그런 다음 코어 디바이스는 이 정보를 로컬에 저장합니다.

다음에 디바이스가 연결되면 Greengrass 코어 디바이스는AWS IoT Greengrass 서비스를 통해 클라이언트 디바이스의 유효성을 검사하려고 시도합니다. 에AWS IoT Greengrass 연결할 수 없는 경우 코어 장치는 로컬에 저장된 장치 정보를 사용하여 클라이언트 장치의 유효성을 검사합니다.

Greengrass 코어 디바이스가 자격 증명 디바이스가 저장하는 기간을 구성할 수 있습니다. 타임아웃을 1분에서 2,147,483,647분까지 설정할 수 있습니다. 기본값은 1분으로, 오프라인 인증을 효과적으로 해제합니다. 이 제한 시간을 설정할 때는 보안 요구 사항을 고려하는 것이 좋습니다. 또한 클라우드와의 연결이 끊긴 상태에서 코어 디바이스가 실행될 것으로 예상되는 시간도 고려해야 합니다.

코어 디바이스는 자격 증명 스토리지를 세 번 업데이트합니다.

1. 디바이스가 코어 디바이스에 처음으로 연결되면
2. 코어 디바이스가 클라우드에 연결된 경우, 클라이언트 디바이스가 코어 디바이스에 다시 연결되는 경우
3. 코어 디바이스가 클라우드에 연결된 경우 하루에 한 번 전체 자격 증명 저장소를 새로 고치십시오.

Greengrass 코어 디바이스는 자격 증명 저장소를 새로 고치면 해당 ListClientDevicesAssociatedWithCoreDevice작업을 사용합니다. Greengrass는 이 작업에서 반환된 디바이스만 새로 고칩니다. 클라이언트 장치를 코어 장치에 연결하려면 을 참조하십시오클라이언트 장치 연결.

`ListClientDevicesAssociatedWithCoreDevice`작업을 사용하려면 실행과 연결된AWS Identity and Access Management (IAM) 역할에 작업 권한을 추가해야AWS IoT Greengrass 합니다.AWS 계정 자세한 정보는 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 단원을 참조하세요.

코어 디바이스 엔드포인트 관리

클라우드 검색을 사용하면 코어 디바이스의 MQTT 브로커 엔드포인트를 클라우드 서비스에 저장합니다. AWS IoT Greengrass 클라이언트 디바이스를 AWS IoT Greengrass 연결하여 관련 코어 디바이스에 대한 이러한 엔드포인트 및 기타 정보를 검색합니다.

각 코어 디바이스에 대해 엔드포인트를 자동 또는 수동으로 관리할 수 있습니다.

- IP 감지기로 엔드포인트를 자동으로 관리합니다.

  네트워크 설정이 복잡하지 않은 경우 (예: 클라이언트 장치가 코어 장치와 동일한 네트워크에 있는 경우) IP 탐지기 구성 요소를 배포하여 핵심 장치 엔드포인트를 자동으로 관리할 수 있습니다. 예를

들어 코어 디바이스가 MQTT 브로커 포트를 코어 디바이스로 전달하는 라우터 뒤에 있는 경우에는 IP 탐지기 구성 요소를 사용할 수 없습니다.

IP 탐지기 구성 요소는 사물 그룹에 있는 모든 핵심 장치의 엔드포인트를 관리하므로 사물 그룹에 배포할 때도 유용합니다. 자세한 설명은 IP 탐지기를 사용하여 엔드포인트를 자동으로 관리합니다. 섹션을 참조하세요.

- 엔드포인트를 수동으로 관리합니다.

  IP 탐지기 구성 요소를 사용할 수 없는 경우 핵심 장치 엔드포인트를 수동으로 관리해야 합니다. 콘솔 또는 API를 사용하여 이러한 엔드포인트를 업데이트할 수 있습니다. 자세한 설명은 엔드포인트를 수동으로 관리합니다. 섹션을 참조하세요.

주제

IP 탐지기를 사용하여 엔드포인트를 자동으로 관리합니다.

코어 디바이스와 동일한 네트워크에 있는 클라이언트 디바이스와 같이 간단한 네트워크 설정이 있는 경우 IP 탐지기 구성 요소를 배포하여 다음 작업을 수행할 수 있습니다.

- Greengrass 코어 장치의 로컬 네트워크 연결 정보를 모니터링합니다. 이 정보에는 코어 디바이스의 네트워크 엔드포인트와 MQTT 브로커가 작동하는 포트가 포함됩니다.
- 코어 디바이스의 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하십시오.

IP 탐지기 구성 요소는 수동으로 설정한 엔드포인트를 덮어씁니다.

> **Important**
>
> 코어 디바이스의 AWS IoT 정책에서 IP 탐지기 구성 요소를 사용할 수 있는 `greengrass:UpdateConnectivityInfo` 권한을 허용해야 합니다. 자세한 내용은 데이터 영역 작업에 대한 AWS IoT 정책 및 AWS IoT사물 정책을 구성합니다. 섹션을 참조하세요.

다음 중 하나를 수행하여 IP 탐지기 구성 요소를 배포할 수 있습니다.

- 콘솔의 검색 구성 페이지를 사용하십시오. 자세한 설명은 클라우드 디스커버리 구성 (콘솔) 섹션을 참조하세요.
- IP 탐지기를 포함하도록 배포를 생성하고 수정하십시오. 콘솔 또는 AWS API를 사용하여 AWS CLI 배포를 관리할 수 있습니다. 자세한 설명은 배포 만들기 섹션을 참조하세요.

IP 탐지기 구성 요소 (콘솔) 배포

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지에서 공용 구성 요소 탭을 선택한 다음 선택합니다 aws.greengrass.clientdevices.IPDetector.
3. aws.greengrass.clientdevices.IPDetector 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포를 만들도록 선택한 후 다음을 선택합니다.
5. 새 배포를 생성하기로 선택한 경우 배포할 대상 코어 장치 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 장치 또는 사물 그룹을 선택하고 다음을 선택합니다.
6. 구성 요소 선택 페이지에서 구성 aws.greengrass.clientdevices.IPDetector요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 구성 요소를 선택하고 aws.greengrass.clientdevices.IPDetector다음을 수행합니다.

   a. 구성 요소 구성을 선택합니다.

   b. 구성 aws.greengrass.clientdevices.IPDetector 모달의 구성 업데이트에 있는 병합할 구성에 구성 업데이트를 입력하여 IP 탐지기 구성 요소를 구성할 수 있습니다. 다음 구성 옵션 중 하나를 지정할 수 있습니다.

      - `defaultPort`— (선택 사항) 이 구성 요소가 IP 주소를 탐지할 때 보고할 MQTT 브로커 포트입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다.
      - `includeIPv4LoopbackAddrs`- (선택 사항) 이 옵션을 사용하여 IPv4 루프백 주소를 검색하고 보고할 수 있습니다. 이러한 IP 주소는 예를 들어 장치가 자체적으로 `localhost` 통신할 수 있는 IP 주소입니다. 코어 기기와 클라이언트 기기가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용하십시오.
      - `includeIPv4LinkLocalAddrs`— (선택 사항) 이 옵션을 사용하여 IPv4 링크-로컬 주소를 검색하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 동적 호스트 구성 프로토콜 (DHCP) 이 없거나 정적으로 할당된 IP 주소가 없는 경우 이 옵션을 사용하십시오.

구성 업데이트는 다음 예와 비슷할 수 있습니다.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false
}
```

c. [확인] 을 선택하여 모달을 닫고 [다음] 을 선택합니다.

8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 배포를 선택합니다.

   배포를 완료하는 데 최대 1분이 걸릴 수 있습니다.

IP 탐지기 구성 요소 배포 (AWS CLI)

IP 탐지기 구성 요소를 배포하려면 `components` 개체에 포함된 `aws.greengrass.clientdevices.IPDetector` 배포 문서를 만들고 구성 요소의 구성 업데이트를 지정하십시오. 의 지침에 따라 새 배포 만들기 배포를 만들거나 기존 배포를 수정하세요.

배포 문서를 생성할 때 다음 옵션 중 하나를 지정하여 IP 탐지기 구성 요소를 구성할 수 있습니다.

- `defaultPort`— (선택 사항) 이 구성 요소가 IP 주소를 탐지할 때 보고할 MQTT 브로커 포트입니다. MQTT 브로커가 기본 포트 8883이 아닌 다른 포트를 사용하도록 구성하는 경우 이 매개 변수를 지정해야 합니다.
- `includeIPv4LoopbackAddrs`- (선택 사항) 이 옵션을 사용하여 IPv4 루프백 주소를 검색하고 보고할 수 있습니다. 이러한 IP 주소는 예를 들어 장치가 자체적으로 `localhost` 통신할 수 있는 IP 주소입니다. 코어 기기와 클라이언트 기기가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용하십시오.
- `includeIPv4LinkLocalAddrs`— (선택 사항) 이 옵션을 사용하여 IPv4 링크-로컬 주소를 검색하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 동적 호스트 구성 프로토콜 (DHCP) 이 없거나 정적으로 할당된 IP 주소가 없는 경우 이 옵션을 사용하십시오.

다음 예제 부분 배포 문서에서는 포트 8883을 MQTT 브로커 포트로 보고하도록 지정합니다.

```
{
  ...,
```

```
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

엔드포인트를 수동으로 관리합니다.

코어 디바이스의 MQTT 브로커 엔드포인트를 수동으로 관리할 수 있습니다.

각 MQTT 브로커 엔드포인트에는 다음 정보가 있습니다.

엔드포인트 () `HostAddress`

클라이언트 디바이스가 코어 디바이스의 MQTT 브로커에 연결할 수 있는 IP 주소 또는 DNS 주소입니다.

포트 (`PortNumber`)

MQTT 브로커가 코어 디바이스에서 작동하는 포트입니다.

Mocquette MQTT 브로커 구성 요소에서 이 포트를 구성할 수 있습니다. 이 구성 요소는 기본적으로 포트 8883을 사용합니다.

메타데이터 () `Metadata`

이 엔드포인트에 연결하는 클라이언트 장치에 제공할 추가 메타데이터입니다.

주제

- 엔드포인트 관리 (콘솔)
- 엔드포인트 관리 () AWS CLI
- 엔드포인트 관리 (API)

엔드포인트 관리 (콘솔)

AWS IoT Greengrass콘솔을 사용하여 코어 디바이스의 엔드포인트를 보고, 업데이트하고, 제거할 수 있습니다.

핵심 장치의 엔드포인트를 관리하려면 (콘솔)

1. AWS IoT Greengrass 콘솔로 이동합니다.
2. 코어 디바이스를 선택합니다.
3. 관리할 코어 디바이스를 선택합니다.
4. 코어 디바이스의 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. MQTT 브로커 엔드포인트 섹션에서 코어 디바이스의 MQTT 브로커 엔드포인트를 확인할 수 있습니다. 엔드포인트 관리를 선택합니다.
6. 엔드포인트 관리 모달에서 코어 디바이스의 MQTT 브로커 엔드포인트를 추가하거나 제거합니다.
7. 업데이트를 선택합니다.

엔드포인트 관리 () AWS CLI

AWS Command Line Interface(AWS CLI) 를 사용하여 코어 장치의 엔드포인트를 관리할 수 있습니다.

> **Note**
>
> 의 AWS IoT Greengrass V2 클라이언트 장치 지원은 이전 버전과 AWS IoT Greengrass V1 호환되므로 AWS IoT Greengrass V2 또는 AWS IoT Greengrass V1 API 작업을 사용하여 핵심 장치 엔드포인트를 관리할 수 있습니다.

코어 디바이스의 엔드포인트를 가져오려면 () AWS CLI

- 다음 명령 중 하나를 사용하십시오.
  - 그린그래스v2: get-connectivity-info
  - 그린그래스: get-connectivity-info

코어 디바이스의 엔드포인트를 업데이트하려면 () AWS CLI

- 다음 명령 중 하나를 사용하십시오.
  - 그린그래스v2: update-connectivity-info
  - 그린그래스: update-connectivity-info

엔드포인트 관리 (API)

AWSAPI를 사용하여 코어 디바이스의 엔드포인트를 관리할 수 있습니다.

> **Note**
>
> 의 AWS IoT Greengrass V2 클라이언트 장치 지원은 이전 버전과 AWS IoT Greengrass V1 호환되므로 AWS IoT Greengrass V1 API 작업을 사용하여 AWS IoT Greengrass V2 핵심 장치 엔드포인트를 관리할 수 있습니다.

코어 디바이스 (API) 용 엔드포인트를 가져오려면 AWS

- 다음 작업 중 하나를 사용하십시오.
  - V2: GetConnectivityInfo
  - V1: GetConnectivityInfo

코어 디바이스 (AWSAPI) 의 엔드포인트를 업데이트하려면

- 다음 작업 중 하나를 사용하십시오.
  - V2: UpdateConnectivityInfo
  - V1: UpdateConnectivityInfo

MQTT 브로커를 선택하세요

AWS IoT Greengrass코어 디바이스에서 실행할 로컬 MQTT 브로커를 선택할 수 있는 옵션을 제공합니다. 클라이언트 디바이스는 코어 디바이스에서 실행되는 MQTT 브로커에 연결되므로 연결하려는 클라이언트 디바이스와 호환되는 MQTT 브로커를 선택하십시오.

> **Note**
>
> MQTT 브로커 구성 요소를 하나만 배포하는 것이 좋습니다. MQTT 브리지 및 IP 탐지기 구성 요소는 한 번에 하나의 MQTT 브로커 구성 요소에서만 작동합니다. 여러 MQTT 브로커 구성 요소를 배포하는 경우 서로 다른 포트를 사용하도록 구성해야 합니다.

다음 MQTT 브로커 중에서 선택할 수 있습니다.

- MQTT 3.1.1 브로커 (모켓) — `aws.greengrass.clientdevices.mqtt.Moquette`

  MQTT 3.1.1 표준을 준수하는 경량 MQTT 브로커를 사용하려면 이 옵션을 선택하십시오. AWS IoT CoreMQTT 브로커는 MQTT 3.1.1 표준과도 호환되므로 이러한 기능을 사용하여 장치 및 AWS IoT Device SDK 여러 장치에서 MQTT 3.1.1을 사용하는 애플리케이션을 만들 수 있습니다. AWS 클라우드

- MQTT 5 브로커 (EMQX) — `aws.greengrass.clientdevices.mqtt.EMQX`

  코어 디바이스와 클라이언트 디바이스 간 통신에 MQTT 5 기능을 사용하려면 이 옵션을 선택하십시오. 이 구성 요소는 Moquette MQTT 3.1.1 브로커보다 더 많은 리소스를 사용하며, Linux 코어 디바이스에서는 Docker가 필요합니다.

  MQTT 5는 MQTT 3.1.1과 이전 버전과 호환되므로 MQTT 3.1.1을 사용하는 클라이언트 디바이스를 이 브로커에 연결할 수 있습니다. Moquette MQTT 3.1.1 브로커를 실행하는 경우 이를 EMQX MQTT 5 브로커로 교체할 수 있으며, 클라이언트 디바이스는 평소와 같이 계속 연결되고 작동할 수 있습니다.

- 사용자 지정 브로커 구현

  이 옵션을 선택하면 클라이언트 장치와 통신할 사용자 지정 로컬 브로커 구성 요소를 만들 수 있습니다. MQTT 이외의 프로토콜을 사용하는 사용자 지정 로컬 브로커를 생성할 수 있습니다. AWS IoT Greengrass클라이언트 장치를 인증하고 권한을 부여하는 데 사용할 수 있는 구성 요소 SDK를 제공합니다. 자세한 정보는 AWS IoT Device SDK를 사용하여 Greengrass 핵 및 기타 구성 요소와 통신하고 AWS IoT Core 및 클라이언트 장치 인증 및 권한 부여(을)를 참조하세요.

MQTT 브로커를 사용하여 클라이언트 장치를AWS IoT Greengrass 코어 장치에 연결

AWS IoT Greengrass코어 디바이스에서 MQTT 브로커를 사용하는 경우 디바이스는 디바이스 고유의 코어 디바이스 인증 기관 (CA) 을 사용하여 클라이언트와의 상호 TLS 연결을 위한 인증서를 브로커에 발급합니다.

AWS IoT Greengrass에서 생성한 핵심 디바이스 CA를 직접 제공할 수 있습니다. 클라이언트 장치 인증구성 요소가AWS IoT Greengrass 연결될 때 코어 디바이스 CA가 등록됩니다. 자동 생성된 코어 디바이스 CA는 영구적이므로 클라이언트 디바이스 인증 구성 요소가 구성된 한 디바이스는 동일한 CA를 계속 사용합니다.

MQTT 브로커가 시작되면 인증서를 요청합니다. 클라이언트 디바이스 인증 구성 요소는 코어 디바이스 CA를 사용하여 X.509 인증서를 발급합니다. 인증서는 브로커가 시작되거나 인증서가 만료되거나 IP 주소와 같은 연결 정보가 변경될 때 교체됩니다. 자세한 정보는 로컬 MQTT 브로커의 인증서 교체을 참조하세요.

클라이언트를 MQTT 브로커에 연결하려면 다음이 필요합니다.

- 클라이언트 디바이스에는AWS IoT Greengrass 코어 디바이스 CA가 있어야 합니다. 클라우드 검색을 통해 또는 수동으로 CA를 제공하여 이 CA를 가져올 수 있습니다. 자세한 정보는 자체 인증 기관 사용을 참조하세요.
- 코어 디바이스의 FQDN (정규화된 도메인 이름) 또는 IP 주소가 코어 디바이스 CA에서 발급한 브로커 인증서에 있어야 합니다. IP 감지기구성 요소를 사용하거나 IP 주소를 수동으로 구성하여 이를 확인합니다. 자세한 정보는 코어 디바이스 엔드포인트 관리을 참조하세요.
- 클라이언트 장치 인증 구성 요소는 클라이언트 장치에 Greengrass 코어 장치에 연결할 수 있는 권한을 부여해야 합니다. 자세한 정보는 클라이언트 장치 인증을 참조하세요.

자체 인증 기관 사용

클라이언트 디바이스가 클라우드에 액세스하여 코어 디바이스를 검색할 수 없는 경우 코어 디바이스 인증 기관 (CA) 을 제공할 수 있습니다. Greengrass 코어 디바이스는 코어 디바이스 CA를 사용하여 MQTT 브로커에 대한 인증서를 발급합니다. 코어 디바이스를 구성하고 클라이언트 디바이스에 CA를 프로비저닝하면 클라이언트 디바이스가 엔드포인트에 연결하고 코어 디바이스 CA (자체 제공 CA 또는 자동 생성 CA) 를 사용하여 TLS 핸드셰이크를 확인할 수 있습니다.

핵심 장치 CA를 사용하도록클라이언트 장치 인증 구성 요소를 구성하려면 구성 요소를 배포할 때`certificateAuthority` 구성 매개 변수를 설정합니다. 구성 중에 다음 세부 정보를 제공합니다.

- 코어 디바이스 CA 인증서의 위치.
- 코어 디바이스 CA 인증서의 개인 키입니다.
- (선택 사항) 코어 디바이스 CA가 중간 CA인 경우 루트 인증서에 대한 인증서 체인입니다.

코어 디바이스 CA를 제공하는 경우, CA를 클라우드에AWS IoT Greengrass 등록합니다.

인증서를 하드웨어 보안 모듈이나 파일 시스템에 저장할 수 있습니다. 다음 예에서는 HSM/TPM을 사용하여 저장된 중간 CA의`certificateAuthority` 컨피그레이션을 보여줍니다. 인증서 체인은 디스크에만 저장할 수 있다는 점에 유의하십시오.

```
  "certificateAuthority": {
```

```
    "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
    "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

이 예에서certificateAuthority 구성 매개 변수는 파일 시스템의 중간 CA를 사용하도록 클라이언트 장치 인증 구성 요소를 구성합니다.

```
"certificateAuthority": {
    "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",
    "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

장치를AWS IoT Greengrass 코어 장치에 연결하려면 다음을 수행하십시오.

1. 조직의 루트 CA를 사용하여 GARENGARENGARESS 코어 디바이스의 CA (인증 기관) 를 생성하십시오. 중간 CA를 보안의 모범 사례로 사용하는 것이 좋습니다.
2. 루트 CA에 대한 중간 CA 인증서, 개인 키 및 인증서 체인을 Greengrass 코어 디바이스에 제공합니다. 자세한 정보는 클라이언트 장치 인증을 참조하세요. 중간 CA는 Greengrass 코어 디바이스의 코어 디바이스 CA가 되고 디바이스는 CA를AWS IoT Greengrass 등록합니다.
3. 클라이언트 디바이스를 사물로AWS IoT 등록합니다. 자세한 내용은 AWS IoT Core개발자 안내서의 사물 객체 만들기를 참조하십시오. 프라이빗 키, 퍼블릭 키, 디바이스 인증서 및 루트 CA 인증서를 클라이언트 디바이스에 추가합니다. 정보를 추가하는 방법은 장치 및 소프트웨어에 따라 다릅니다.

디바이스를 구성한 후에는 인증서와 공개 키 체인을 사용하여 Greengrass 코어 디바이스에 연결할 수 있습니다. 소프트웨어는 핵심 장치 엔드포인트를 찾는 역할을 합니다. 코어 디바이스의 엔드포인트를 수동으로 설정할 수 있습니다. 자세한 정보는 엔드포인트를 수동으로 관리합니다. 단원을 참조하세요.

클라이언트 장치 통신 테스트

클라이언트 장치는 AWS IoT Device SDK 를 사용하여 코어 장치를 검색, 연결 및 통신할 수 있습니다. 에서 Greengrass 검색 클라이언트를 사용하여 Greengrass 검색 API를 사용할 수 있습니다. Greengrass 검색 API는 클라이언트 장치가 연결할 수 있는 핵심 장치에 대한 정보를 반환합니다. AWS IoT Device SDK API 응답에는 연결할 MQTT 브로커 엔드포인트와 각 코어 디바이스의 ID를 확인하는 데 사용할 인증서가 포함됩니다. 그러면 클라이언트 디바이스가 코어 디바이스에 성공적으로 연결될 때까지 각 엔드포인트를 시도할 수 있습니다.

클라이언트 장치는 연결된 코어 장치만 검색할 수 있습니다. 클라이언트 장치와 코어 장치 간의 통신을 테스트하기 전에 클라이언트 장치를 코어 장치에 연결해야 합니다. 자세한 설명은 클라이언트 장치 연결 섹션을 참조하세요.

Greengrass 검색 API는 지정한 코어 디바이스 MQTT 브로커 엔드포인트를 반환합니다. IP 탐지기 구성 요소를 사용하여 이러한 엔드포인트를 관리하거나 각 코어 디바이스에 대해 수동으로 관리할 수 있습니다. 자세한 설명은 코어 디바이스 엔드포인트 관리 섹션을 참조하세요.

> **Note**
>
> Greengrass 검색 API를 사용하려면 클라이언트 장치에 권한이 있어야 합니다. `greengrass:Discover` 자세한 설명은 클라이언트 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

AWS IoT Device SDK는 여러 프로그래밍 언어로 제공됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 AWS IoT 디바이스 SDK 섹션을 참조하세요.

주제

- 테스트 통신 (Python)
- 통신 테스트 (C++)
- 통신 테스트 () JavaScript
- 테스트 통신 (Java)

테스트 통신 (Python)

이 섹션에서는 Python용 AWS IoT Device SDK v2의 Greengrass 검색 샘플을 사용하여 클라이언트 장치와 코어 장치 간의 통신을 테스트합니다.

> **Important**
>
> Python용 AWS IoT Device SDK v2를 사용하려면 기기에서 Python 3.6 이상을 실행해야 합니다.

통신을 테스트하려면 (Python의 경우 AWS IoT Device SDK v2)

1. AWS IoT Device SDKv2용 Python을 AWS IoT 다운로드하여 클라이언트 장치로 연결할 사물에 설치합니다.

   클라이언트 기기에서 다음을 수행하십시오.

   a. AWS IoT Device SDKv2 for Python 저장소를 복제하여 다운로드하십시오.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Python용 AWS IoT Device SDK v2를 설치합니다.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Python용 AWS IoT Device SDK v2의 샘플 폴더로 변경합니다.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. 샘플 Greengrass 검색 애플리케이션을 실행합니다. 이 애플리케이션에는 클라이언트 디바이스 사물 이름, 사용할 MQTT 주제 및 메시지, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다. 다음 예제에서는 주제에 Hello World 메시지를 보냅니다. `clients/MyClientDevice1/hello/world`

   - *MyClientDevice1#* 클라이언트 디바이스의 사물 이름으로 대체합니다.
   - *~/certs/ AmazonRoot ca1.pem# ##### ##### Amazon ## CA ### ###* 바꾸십시오.
   - *~/certs/device.pem.crt#* 클라이언트 디바이스의 디바이스 인증서 경로로 바꾸십시오.
   - *~/certs/private.pem.key#* 클라이언트 장치의 개인 키 파일 경로로 바꾸십시오.
   - *us-east-1#* 클라이언트 디바이스 및 코어 디바이스가 운영되는 지역으로 바꾸십시오. AWS

   ```
   python3 basic_discovery.py \\
     --thing_name MyClientDevice1 \\
     --topic 'clients/MyClientDevice1/hello/world' \\
     --message 'Hello World!' \\
     --ca_file ~/certs/AmazonRootCA1.pem \\
     --cert ~/certs/device.pem.crt \\
     --key ~/certs/private.pem.key \\
     --region us-east-1 \\
   ```

```
  --verbosity Warn
```

검색 샘플 애플리케이션은 메시지를 10번 전송하고 연결을 끊습니다. 또한 메시지를 게시하는 동일한 주제를 구독합니다. 애플리케이션에서 해당 주제에 대한 MQTT 메시지를 수신했다고 출력에 표시되면 클라이언트 기기는 코어 기기와 성공적으로 통신할 수 있습니다.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

애플리케이션이 대신 오류를 출력하는 경우 Greengrass 검색 문제 해결을 참조하십시오.

또한 코어 디바이스의 Greengrass 로그를 확인하여 클라이언트 디바이스가 성공적으로 연결하고 메시지를 전송하는지 확인할 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

통신 테스트 (C++)

이 섹션에서는 C++용 AWS IoT Device SDK v2의 Greengrass 검색 샘플을 사용하여 클라이언트 장치와 코어 장치 간의 통신을 테스트합니다.

C++용 AWS IoT Device SDK v2를 빌드하려면 기기에 다음 도구가 있어야 합니다.

- C++ 11 이상
- CMake 3.1 이상
- 다음 컴파일러 중 하나:
  - GCC 4.8 이상
  - 클랑 3.9 이상
  - MSVC 2015 이상

통신을 테스트하려면 (C++의 경우 AWS IoT Device SDK v2)

1. C++용 AWS IoT Device SDK v2를 다운로드하고 빌드하여 클라이언트 AWS IoT 기기로 연결할 수 있습니다.

   클라이언트 기기에서 다음과 같이 하세요.

   a. AWS IoT Device SDKv2 C++용 작업 영역용 폴더를 만들고 해당 작업 영역으로 변경합니다.

   ```
   cd
   mkdir iot-device-sdk-cpp
   cd iot-device-sdk-cpp
   ```

   b. C++용 AWS IoT Device SDK v2 저장소를 복제하여 다운로드하십시오. `--recursive`플래그는 하위 모듈을 다운로드하도록 지정합니다.

   ```
   git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
   ```

   c. C++ 빌드 출력용 AWS IoT Device SDK v2용 폴더를 만들고 이 폴더로 변경합니다.

```
mkdir aws-iot-device-sdk-cpp-v2-build
cd aws-iot-device-sdk-cpp-v2-build
```

d. C++용 AWS IoT Device SDK v2를 빌드하세요.

```
cmake -DCMAKE_INSTALL_PREFIX="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
cmake --build . --target install
```

2. AWS IoT Device SDKv2에서 C++용 Greengrass 검색 샘플 애플리케이션을 빌드하세요. 다음을 따릅니다.

a. C++용 AWS IoT Device SDK v2에서 Greengrass 디스커버리 샘플 폴더로 변경하십시오.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

b. Greengrass 디스커버리 샘플 빌드 출력을 위한 폴더를 만들고 이 폴더로 변경합니다.

```
mkdir build
cd build
```

c. Greengrass 디스커버리 샘플 애플리케이션을 빌드하세요.

```
cmake -DCMAKE_PREFIX_PATH="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ..
cmake --build . --config "Release"
```

3. 샘플 Greengrass 검색 애플리케이션을 실행합니다. 이 애플리케이션에는 클라이언트 디바이스 사물 이름, 사용할 MQTT 주제, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다. 다음 `clients/`*`MyClientDevice1`*`/hello/world` 예제는 주제를 구독하고 명령줄에 입력한 메시지를 동일한 주제에 게시합니다.

- *`MyClientDevice1#`* 클라이언트 디바이스의 사물 이름으로 바꾸십시오.
- *`~/certs/ AmazonRoot ca1.pem# ##### ##### Amazon ## CA ### ###`* 바꾸십시오.
- *`~/certs/device.pem.crt#`* 클라이언트 디바이스의 디바이스 인증서 경로로 바꾸십시오.
- *`~/certs/private.pem.key#`* 클라이언트 장치의 개인 키 파일 경로로 바꾸십시오.
- *`us-east-1#`* 클라이언트 디바이스 및 코어 디바이스가 운영되는 지역으로 바꾸십시오. AWS

```
./basic-discovery \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1
```

검색 샘플 애플리케이션은 주제를 구독하고 게시할 메시지를 입력하라는 메시지를 표시합니다.

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn
 arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to
 203.0.113.0:8883
Successfully subscribed to clients/MyClientDevice1/hello/world
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
```

애플리케이션이 대신 오류를 출력하는 경우 Greengrass 검색 문제 해결을 참조하십시오.

4. 메시지 (예**Hello World!**:) 를 입력합니다.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
Hello World!
```

애플리케이션이 주제에 대한 MQTT 메시지를 수신했다고 출력에 표시되면 클라이언트 디바이스는 코어 디바이스와 성공적으로 통신할 수 있습니다.

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

또한 코어 디바이스의 Greengrass 로그를 확인하여 클라이언트 디바이스가 성공적으로 연결하고 메시지를 전송하는지 확인할 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

통신 테스트 () JavaScript

이 섹션에서는 AWS IoT Device SDKv2 양식의 Greengrass 검색 샘플을 사용하여 클라이언트 장치와 코어 장치 간의 통신을 테스트합니다. JavaScript

> **Important**
>
> AWS IoT Device SDKv2를 사용하려면 기기에서 Node v10.0 이상을 실행해야 합니다. JavaScript

통신을 테스트하려면 (AWS IoT Device SDKv2의 경우) JavaScript

1. AWS IoT Device SDKv2 for를 다운로드하여 클라이언트 AWS IoT 장치로 연결할 사물에 설치합니다. JavaScript

   클라이언트 장치에서 다음을 수행하십시오.

   a. AWS IoT Device SDKv2를 복제하여 JavaScript 저장소용 v2를 다운로드하세요.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

   b. 용 AWS IoT Device SDK v2를 설치합니다 JavaScript.

   ```
   cd aws-iot-device-sdk-js-v2
   npm install
   ```

2. AWS IoT Device SDKv2 양식에서 Greengrass 디스커버리 샘플 폴더로 변경하십시오. JavaScript

   ```
   cd samples/node/basic_discovery
   ```

3. Greengrass 디스커버리 샘플 애플리케이션을 설치합니다.

   ```
   npm install
   ```

4. 샘플 Greengrass 검색 애플리케이션을 실행합니다. 이 애플리케이션에는 클라이언트 디바이스 사물 이름, 사용할 MQTT 주제 및 메시지, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다. 다음 예제에서는 주제에 Hello World 메시지를 보냅니다. `clients/`*`MyClientDevice1`*`/hello/world`

   - *`MyClientDevice1#`* 클라이언트 디바이스의 사물 이름으로 대체합니다.

- *~/certs/ AmazonRoot ca1.pem# ##### ##### Amazon ## CA ### ###* 바꾸십시오.
- *~/certs/device.pem.crt#* 클라이언트 디바이스의 디바이스 인증서 경로로 바꾸십시오.
- *~/certs/private.pem.key#* 클라이언트 장치의 개인 키 파일 경로로 바꾸십시오.
- *us-east-1#* 클라이언트 디바이스 및 코어 디바이스가 운영되는 지역으로 바꾸십시오. AWS

```
node dist/index.js \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --message 'Hello World!' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

검색 샘플 애플리케이션은 메시지를 10번 전송하고 연결을 끊습니다. 또한 메시지를 게시하는 동일한 주제를 구독합니다. 애플리케이션에서 해당 주제에 대한 MQTT 메시지를 수신했다고 출력에 표시되면 클라이언트 기기는 코어 기기와 성공적으로 통신할 수 있습니다.

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-
MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore","connectivity":
[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificat
["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
Trying
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
 fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
 signals thrown, you may want to install a signal trap in your application layer.
Connected to
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":1}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":2}
```

```
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":4}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

애플리케이션이 대신 오류를 출력하는 경우 Greengrass 검색 문제 해결을 참조하십시오.

또한 코어 디바이스의 Greengrass 로그를 확인하여 클라이언트 디바이스가 성공적으로 연결하고 메시지를 전송하는지 확인할 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

테스트 통신 (Java)

이 섹션에서는 AWS IoT Device SDKv2 for Java의 Greengrass 검색 샘플을 사용하여 클라이언트 장치와 코어 장치 간의 통신을 테스트합니다.

> **Important**
>
> Java용 AWS IoT Device SDK v2를 빌드하려면 기기에 다음 도구가 있어야 합니다.
>
> - Java 8 이상 버전, Java 폴더를 `JAVA_HOME` 가리키고 있어야 합니다.

- Apache Maven

통신을 테스트하려면 (Java의 경우 AWS IoT Device SDK v2)

1. AWS IoT Device SDKv2용 Java를 다운로드하고 빌드하여 클라이언트 장치로 연결할 수 있습니다. AWS IoT

   클라이언트 기기에서 다음을 수행하십시오.

   a. AWS IoT Device SDKv2 for Java 저장소를 복제하여 다운로드하십시오.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
   ```

   b. Java용 AWS IoT Device SDK v2 폴더로 변경합니다.

   c. 자바용 AWS IoT Device SDK v2를 빌드하세요.

   ```
   cd aws-iot-device-sdk-java-v2
   mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
   mvn clean install
   ```

2. 샘플 Greengrass 검색 애플리케이션을 실행합니다. 이 애플리케이션에는 클라이언트 디바이스 사물 이름, 사용할 MQTT 주제, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다. 다음 `clients/MyClientDevice1/hello/world` 예제는 주제를 구독하고 명령줄에 입력한 메시지를 동일한 주제에 게시합니다.

   - *MyClientDevice1#* 두 인스턴스를 모두 클라이언트 디바이스의 사물 이름으로 바꿉니다.
   - *$HOME/certs/ AmazonRoot ca1.pem# ##### ##### Amazon ## CA ### ###* 대체하십시오.
   - *$HOME/certs/device.pem.crt#* 클라이언트 디바이스의 디바이스 인증서 경로로 바꾸십시오.
   - *$HOME/certs/private.pem.key#* 클라이언트 장치의 개인 키 파일 경로로 바꾸십시오.
   - *us-east-1#* 클라이언트 디바이스 및 코어 디바이스가 작동하는 곳으로 바꾸십시오. AWS 리전

   ```
   DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \
     --topic 'clients/MyClientDevice1/hello/world' \
     --ca_file $HOME/certs/AmazonRootCA1.pem \
   ```

```
  --cert $HOME/certs/device.pem.crt \
  --key $HOME/certs/private.pem.key \
  --region us-east-1"

mvn exec:java -pl samples/Greengrass \
  -Dexec.mainClass=greengrass.BasicDiscovery \
  -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

검색 샘플 애플리케이션은 주제를 구독하고 게시할 메시지를 입력하라는 메시지를 표시합니다.

```
Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing
 arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Started a clean session
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
```

애플리케이션이 대신 오류를 출력하는 경우 Greengrass 검색 문제 해결을 참조하십시오.

3. 메시지 (예**Hello World!**:) 를 입력합니다.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
Hello World!
```

애플리케이션이 주제에 대한 MQTT 메시지를 수신했다고 출력에 표시되면 클라이언트 디바이스는 코어 디바이스와 성공적으로 통신할 수 있습니다.

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

또한 코어 디바이스의 Greengrass 로그를 확인하여 클라이언트 디바이스가 성공적으로 연결하고 메시지를 전송하는지 확인할 수 있습니다. 자세한 내용은 모니터 AWS IoT Greengrass 로그을(를) 참조하세요.

그린그래스 디스커버리 RESTful API

AWS IoT Greengrass클라이언트 장치가 연결할 수 있는 Greengrass 코어 장치를 식별하는 데 사용할 수 있는 `Discover` API 작업을 제공합니다. 클라이언트 디바이스는 이 데이터 플레인 작업을 사용하여 Greengrass 코어 디바이스에 연결하는 데 필요한 정보를 검색하고, 이를

BatchAssociateClientDeviceWithCoreDeviceAPI 작업에 연결합니다. 클라이언트 장치가 온라인 상태가 되면 AWS IoT Greengrass 클라우드 서비스에 연결하고 검색 API를 사용하여 다음을 찾을 수 있습니다.

- 연결된 각 그린그래스 코어 디바이스의 IP 주소 및 포트
- 코어 디바이스 CA 인증서. 클라이언트 디바이스가 Greengrass 코어 디바이스를 인증하는 데 사용할 수 있습니다.

> **Note**
>
> 또한 클라이언트 장치는 의 검색 클라이언트를 사용하여 Greengrass 코어 장치에 대한 연결 정보를 검색할 수 있습니다. AWS IoT Device SDK 디스커버리 클라이언트는 디스커버리 API를 사용합니다. 자세한 내용은 다음 자료를 참조하세요.
>
> - 클라이언트 장치 통신 테스트
> - 개발자 안내서의 그린그래스 디스커버리 RESTful API AWS IoT Greengrass Version 1

이 API 작업을 사용하려면 Greengrass 데이터 플레인 엔드포인트의 검색 API에 HTTP 요청을 보내십시오. 이 API 엔드포인트의 형식은 다음과 같습니다.

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

AWS IoT Greengrass검색 API의 지원 AWS 리전 및 엔드포인트 목록은 의 AWS IoT Greengrass V2엔드포인트 및 할당량을 참조하십시오. AWS 일반 참조 이 API 작업은 Greengrass 데이터 플레인 엔드포인트에서만 사용할 수 있습니다. 구성 요소 및 배포를 관리하는 데 사용하는 컨트롤 플레인 엔드포인트는 데이터 플레인 엔드포인트와 다릅니다.

> **Note**
>
> 및 의 검색 API는 동일합니다. AWS IoT Greengrass V1 AWS IoT Greengrass V2 AWS IoT Greengrass V1코어에 연결되는 클라이언트 장치가 있는 경우 클라이언트 장치의 코드를 변경하지 않고도 AWS IoT Greengrass V2 핵심 장치에 연결할 수 있습니다. 자세한 내용은 개발자 안내서의 그린그래스 디스커버리 RESTful API를 참조하십시오. AWS IoT Greengrass Version 1

주제

디스커버리 인증 및 권한 부여

검색 API를 사용하여 연결 정보를 검색하려면 클라이언트 장치가 X.509 클라이언트 인증서와 함께 TLS 상호 인증을 사용하여 인증해야 합니다. 자세한 내용은 개발자 안내서의 X.509 클라이언트 인증서를 참조하십시오. AWS IoT Core

클라이언트 장치에도 작업을 수행할 수 있는 권한이 있어야 합니다. `greengrass:Discover` 다음 예제 AWS IoT 정책은 이름이 지정된 AWS IoT 사물이 자체적으로 `MyClientDevice1` `Discover` 수행되도록 허용합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

> **Important**
>
> 사물 정책 변수 (`iot:Connection.Thing.*`) 는 코어 디바이스 또는 Greengrass 데이터 플레인 작업에 대한 AWS IoT 정책에서 지원되지 않습니다. 대신 이름이 비슷한 여러 장치를 매칭하는 와일드카드를 사용할 수 있습니다. 예를 `MyGreengrassDevice1` `MyGreengrassDevice2` 들어 `MyGreengrassDevice*` 일치하도록 지정하는 등의 작업을 수행할 수 있습니다.

자세한 내용은 AWS IoT Core개발자 안내서의 AWS IoT Core 정책을 참조하십시오.

요청

요청은 표준 HTTP 헤더를 포함하며 다음 예와 같이 Greengrass 검색 엔드포인트로 전송됩니다.

포트 번호는 코어 디바이스가 포트 8443을 통해 HTTPS 트래픽을 전송하도록 구성되었는지 아니면 포트 443을 통해 전송하도록 구성되었는지에 따라 달라집니다. 자세한 설명은 the section called "포트 443에서 또는 네트워크 프록시를 통해 연결" 섹션을 참조하세요.

> **Note**
>
> 이 예시에서는 권장 ATS 루트 CA 인증서와 함께 작동하는 Amazon 트러스트 서비스 (ATS) 엔드포인트를 사용합니다. 엔드포인트는 CA 인증서 유형과 일치해야 합니다

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
```

> **Note**
>
> 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.

응답

성공 시 응답 헤더에는 HTTP 200 상태 코드가 포함되고 응답 본문에는 discover 응답 문서가 포함됩니다.

> **Note**
>
> 와 AWS IoT Greengrass V1 동일한 검색 API를 AWS IoT Greengrass V2 사용하기 때문에 응답은 Greengrass 그룹과 같은 AWS IoT Greengrass V1 개념에 따라 정보를 구성합니다. 응답에는 Greengrass 그룹 목록이 포함되어 있습니다. 에서 AWS IoT Greengrass V2 각 코어 디바이스는 자체 그룹에 속하며, 그룹에는 해당 코어 디바이스와 해당 연결 정보만 포함됩니다.

검색 응답 문서 예제

다음 문서는 하나의 Greengrass 코어 장치에 연결된 클라이언트 장치에 대한 응답을 보여줍니다. 코어 디바이스에는 엔드포인트 하나와 CA 인증서 하나가 있습니다.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

다음 문서는 두 개의 코어 장치에 연결된 클라이언트 장치에 대한 응답을 보여줍니다. 코어 디바이스에는 여러 엔드포인트와 여러 그룹 CA 인증서가 있습니다.

```
{
```

```
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
              "hostAddress": "core-device-01-address-2",
              "portNumber": core-device-01-port-2,
              "metadata": "core-device-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
      "Cores": [
        {
          "thingArn":"core-device-02-thing-arn",
          "Connectivity" : [
            {
              "id": "core-device-02-connection-id",
              "hostAddress": "core-device-02-address",
              "portNumber": core-device-02-port,
              "metadata": "core-device-02-connection-1-description"
            }
          ]
        }
      ],
      "CAs": [
```

```
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

curL을 사용하여 디스커버리 API를 테스트합니다.

cURL설치한 경우 검색 API를 테스트할 수 있습니다. 다음 예제는 Greengrass 검색 API 엔드포인트에 대한 요청을 인증하기 위한 클라이언트 장치의 인증서를 지정합니다.

```
curl -i \
  --cert 1a23bc4d56.cert.pem \
  --key 1a23bc4d56.private.key \
  https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

> **Note**
>
> `-i`인수는 HTTP 응답 헤더를 출력하도록 지정합니다. 이 옵션을 사용하면 오류를 식별하는 데 도움이 될 수 있습니다.

요청이 성공하면 이 명령은 다음 예제와 비슷한 응답을 출력합니다.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "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"
      ]
    }
  ]
}
```

명령에서 오류가 출력되면 Greengrass 검색 문제 해결을 참조하십시오.

클라이언트 장치 간에 MQTT 메시지를 중계하고 AWS IoT Core

클라이언트 장치 및 간에 MQTT 메시지 및 기타 데이터를 릴레이할 수 있습니다. AWS IoT Core 클라이언트 디바이스는 코어 디바이스에서 실행되는 MQTT Broker 컴포넌트에 연결됩니다. 기본적으로 코어 디바이스는 클라이언트 디바이스와 간에 MQTT 메시지 또는 데이터를 릴레이하지 않습니다. AWS IoT Core 클라이언트 장치는 기본적으로 MQTT를 통해서만 서로 통신할 수 있습니다.

클라이언트 장치 및 AWS IoT Core 간에 MQTT 메시지를 릴레이하려면 다음을 수행하도록 MQTT 브리지 구성 요소를 구성하십시오.

- 클라이언트 장치의 메시지를 로 릴레이합니다. AWS IoT Core
- 클라이언트 AWS IoT Core 장치로 메시지를 릴레이합니다.

> **Note**
>
> 클라이언트 장치가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 AWS IoT Core 구독하는 경우에도 MQTT 브리지는 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연이 발생할 수 있습니다. AWS IoT Core 코어 디바이스의 MQTT 컨피그레이션에 대한 자세한 내용은 을 참조하십시오. MQTT 타임아웃 및 캐시 설정을 구성합니다.

주제

- MQTT 브리지 구성 요소 구성 및 배포
- 릴레이 MQTT 메시지

MQTT 브리지 구성 요소 구성 및 배포

MQTT 브리지 구성 요소는 각각 메시지 소스와 메시지 대상을 지정하는 주제 매핑 목록을 사용합니다. 클라이언트 장치 간에 메시지를 릴레이하려면 MQTT 브리지 구성 요소를 배포하고 구성 요소 구성에서 각 소스 및 대상 주제를 지정하십시오. AWS IoT Core

MQTT 브리지 구성 요소를 코어 장치 또는 코어 장치 그룹에 배포하려면 구성 요소가 포함된 배포를 만드십시오. `aws.greengrass.clientdevices.mqtt.Bridge` 배포의 MQTT 브리지 구성 요소 구성에서 주제 매핑을 지정합니다. `mqttTopicMapping`

다음 예제에서는 클라이언트 장치의 주제 필터와 일치하는 주제에 대한 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성하는 배포를 `clients/+/hello/world` 정의합니다. AWS IoT Core `merge`구성 업데이트에는 직렬화된 JSON 개체가 필요합니다. 자세한 설명은 구성 요소 구성 업데이트 섹션을 참조하세요.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/
+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
    ...
  }
}
```

릴레이 MQTT 메시지

클라이언트 장치 간에 MQTT 메시지를 릴레이하려면 MQTT Bridge 구성 요소를 구성 및 배포하고 릴레이할 주제를 지정하십시오. AWS IoT Core

Example 예: 특정 주제에 대한 메시지를 클라이언트 장치에서 다음으로 릴레이합니다. AWS IoT Core

다음 MQTT 브리지 구성 요소 구성은 클라이언트 장치의 주제 필터와 일치하는 주제에 대한 메시지 릴레이를 지정합니다. `clients/+/hello/world/event` AWS IoT Core

```
{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example 예: 클라이언트 장치에서 주제에 대한 메시지를 AWS IoT Core 릴레이합니다.

다음 MQTT 브리지 구성 요소 구성은 주제 필터와 일치하는 주제에 대한 메시지를 클라이언트 장치로 `clients/+/hello/world/event/response` AWS IoT Core 릴레이하도록 지정합니다.

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

구성 요소에서 클라이언트 장치와 상호 작용

코어 장치에 연결된 클라이언트 장치와 상호 작용하는 사용자 지정 Greengrass 구성 요소를 개발할 수 있습니다. 예를 들어, 다음을 수행하는 구성 요소를 개발할 수 있습니다.

- 클라이언트 장치의 MQTT 메시지를 기반으로 작업하고 AWS 클라우드 대상으로 데이터를 전송하십시오.
- MQTT 메시지를 클라이언트 장치에 전송하여 작업을 시작합니다.

클라이언트 디바이스는 코어 디바이스에서 실행되는 MQTT Broker 컴포넌트를 통해 코어 디바이스에 연결하고 코어 디바이스와 통신합니다. 기본적으로 클라이언트 디바이스는 MQTT를 통해서만 서로 통신할 수 있으며 Greengrass 구성요소는 이러한 MQTT 메시지를 수신하거나 클라이언트 디바이스로 메시지를 보낼 수 없습니다.

Greengrass 구성 요소는 로컬 게시/구독 인터페이스를 사용하여 코어 디바이스에서 통신합니다. Greengrass 구성 요소의 클라이언트 장치와 통신하려면 다음을 수행하도록 MQTT 브리지 구성 요소를 구성합니다.

- 클라이언트 장치의 MQTT 메시지를 로컬 게시/구독으로 릴레이합니다.
- 로컬 게시/구독의 MQTT 메시지를 클라이언트 장치로 중계합니다.

Greengrass 구성 요소의 클라이언트 장치 섀도우와 상호 작용할 수도 있습니다. 자세한 설명은 클라이언트 디바이스 섀도우와 상호 작용 및 동기화 섹션을 참조하세요.

주제

- MQTT 브리지 구성 요소 구성 및 배포
- 클라이언트 디바이스에서 MQTT 메시지 수신
- MQTT 메시지를 클라이언트 장치로 전송

MQTT 브리지 구성 요소 구성 및 배포

MQTT 브리지 구성 요소는 각각 메시지 소스와 메시지 대상을 지정하는 주제 매핑 목록을 사용합니다. 클라이언트 장치와 통신하려면 MQTT 브리지 구성 요소를 배포하고 구성 요소 구성에서 각 소스 및 대상 주제를 지정하십시오.

MQTT 브리지 구성 요소를 코어 장치 또는 코어 장치 그룹에 배포하려면 구성 요소가 포함된 배포를 만드십시오. `aws.greengrass.clientdevices.mqtt.Bridge` 배포의 MQTT 브리지 구성 요소 구성에서 주제 매핑을 지정합니다. `mqttTopicMapping`

다음 예제는 클라이언트 장치에서 로컬 게시/구독 브로커로 `clients/MyClientDevice1/hello/world` 주제를 릴레이하도록 MQTT 브리지 구성 요소를 구성하는 배포를 정의합니다. `merge`구성 업

데이트에는 직렬화된 JSON 개체가 필요합니다. 자세한 설명은 구성 요소 구성 업데이트 섹션을 참조하세요.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldPubsub": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\"HelloWorldPubsub\":{\"topic\":\"clients/
MyClientDevice1/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"Pubsub\"}}}"
      }
    }
    ...
  }
}
```

MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 주제에 대한 메시지를 릴레이할 수 있습니다. MQTT 브리지 v2.2.0 이상을 사용하는 경우 소스 브로커가 로컬 게시/구독일 때 주제 필터에 MQTT 주제 와일드카드를 사용할 수 있습니다. 자세한 내용은 MQTT 브리지 구성 요소 구성을 참조하십시오.

클라이언트 디바이스에서 MQTT 메시지 수신

MQTT 브리지 구성 요소가 클라이언트 장치로부터 메시지를 수신하도록 구성한 로컬 게시/구독 주제를 구독할 수 있습니다.

클라이언트 장치의 MQTT 메시지를 사용자 지정 구성 요소로 수신하려면

1. 클라이언트 장치가 로컬 게시/구독 주제에 게시하는 MQTT 토픽의 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성하고 배포하십시오.
2. 로컬 게시/구독 IPC 인터페이스를 사용하여 MQTT 브리지가 메시지를 중계하는 주제를 구독하십시오. 자세한 내용은 로컬 메시지 게시/구독 및 SubscribeToTopic 섹션을 참조하세요.

Connect and test 클라이언트 장치 자습서에는 클라이언트 장치의 메시지를 구독하는 구성 요소를 개발하는 섹션이 포함되어 있습니다. 자세한 설명은 4단계: 클라이언트 기기와 통신하는 구성 요소 개발 섹션을 참조하세요.

MQTT 메시지를 클라이언트 장치로 전송

MQTT 브리지 구성 요소가 클라이언트 장치에 메시지를 보내도록 구성한 로컬 게시/구독 주제를 로컬 게시/구독 항목에 게시할 수 있습니다.

MQTT 메시지를 사용자 지정 구성 요소로 클라이언트 장치에 게시하려면

1. 로컬 게시/구독 주제의 메시지를 클라이언트 장치가 구독하는 MQTT 주제로 릴레이하도록 MQTT 브리지 구성 요소를 구성하고 배포하십시오.
2. 로컬 게시/구독 IPC 인터페이스를 사용하여 MQTT 브리지가 메시지를 중계하는 주제에 게시할 수 있습니다. 자세한 정보는 로컬 메시지 게시/구독 및 PublishToTopic(을)를 참조하세요.

클라이언트 디바이스 섀도우와 상호 작용 및 동기화

Shadow Manager 구성 요소를 사용하여 클라이언트 장치 섀도우를 포함한 로컬 섀도우를 관리할 수 있습니다. 섀도우 관리자를 사용하여 다음 작업을 수행할 수 있습니다.

- Greengrass 구성 요소의 클라이언트 장치 섀도우와 상호 작용합니다.
- 클라이언트 디바이스 섀도우를 와 AWS IoT Core 동기화합니다.

> **Note**
>
> 섀도우 매니저 컴포넌트는 AWS IoT Core 기본적으로 섀도우를 동기화하지 않습니다. 동기화할 클라이언트 장치 섀도를 지정하도록 섀도우 관리자 구성 요소를 구성해야 합니다.

주제

사전 조건

클라이언트 장치 섀도와 상호 작용하고 클라이언트 장치 섀도를 동기화하려면 코어 장치가 다음 요구 사항을 충족해야 합니다. AWS IoT Core

- 코어 디바이스는 클라이언트 디바이스 지원을 위한 Greengrass 구성 요소 외에도 다음 구성 요소를 실행해야 합니다.
  - 그린그래스 뉴클리어스 v2.6.0 이상
  - 섀도우 매니저 v2.2.0 이상
  - MQTT 브리지 v2.2.0 이상
- 클라이언트 장치가 장치 섀도우 주제에 대해 통신할 수 있도록 클라이언트 장치 인증 구성 요소를 구성해야 합니다.

섀도우 관리자가 클라이언트 장치와 통신할 수 있도록 합니다.

기본적으로 섀도우 관리자 구성요소는 클라이언트 장치 섀도를 관리하지 않습니다. 이 기능을 사용하려면 클라이언트 장치와 섀도우 관리자 구성 요소 간에 MQTT 메시지를 릴레이해야 합니다. 클라이언트 디바이스는 MQTT 메시지를 사용하여 디바이스 섀도우 업데이트를 수신하고 전송합니다. 섀도우 관리자 구성 요소는 로컬 Greengrass 게시/구독 인터페이스를 구독하므로 장치 섀도우 주제에 대해 MQTT 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성할 수 있습니다.

MQTT 브리지 구성 요소는 각각 메시지 소스와 메시지 대상을 지정하는 주제 매핑 목록을 사용합니다. 섀도우 관리자 구성 요소가 클라이언트 장치 섀도를 관리할 수 있도록 하려면 MQTT 브리지 구성 요소를 배포하고 클라이언트 장치 섀도우에 대한 섀도우 주제를 지정하십시오. 로컬 MQTT와 로컬 게시/구독 간에 메시지를 양방향으로 릴레이하도록 브리지를 구성해야 합니다.

MQTT 브리지 구성 요소를 코어 장치 또는 코어 장치 그룹에 배포하려면 구성 요소가 포함된 배포를 생성하십시오. `aws.greengrass.clientdevices.mqtt.Bridge` 배포의 MQTT 브리지 구성 요소 구성에서 주제 매핑을 지정합니다. `mqttTopicMapping`

다음 예제를 사용하여 클라이언트 장치와 섀도우 관리자 구성 요소 간의 통신을 활성화하도록 MQTT 브리지 구성 요소를 구성합니다.

> **Note**
>
> AWS IoT Greengrass콘솔에서 이러한 구성 예제를 사용할 수 있습니다. AWS IoT GreengrassAPI를 사용하는 경우 `merge` 구성 업데이트에는 직렬화된 JSON 객체가 필요하므로 다음 JSON 객체를 문자열로 직렬화해야 합니다. 자세한 설명은 구성 요소 구성 업데이트 섹션을 참조하세요.

Example 예: 모든 클라이언트 디바이스 섀도우 관리

다음 MQTT 브리지 구성 예제를 사용하면 섀도우 관리자가 모든 클라이언트 장치의 모든 섀도우를 관리할 수 있습니다.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 예: 클라이언트 장치의 섀도우 관리

다음 MQTT 브리지 구성 예제를 사용하면 섀도우 관리자가 이름이 지정된 `MyClientDevice` 클라이언트 장치의 모든 섀도우를 관리할 수 있습니다.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
```

```
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 예: 모든 클라이언트 장치의 명명된 섀도우 관리

다음 MQTT 브리지 구성 예제를 사용하면 섀도우 관리자가 모든 클라이언트 장치의 이름이 지정된 `DeviceConfiguration` 섀도우를 관리할 수 있습니다.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 예: 모든 클라이언트 장치의 명명되지 않은 섀도우 관리

다음 MQTT 브리지 구성 예제를 사용하면 섀도우 관리자가 모든 클라이언트 장치에 대해 명명되지 않은 섀도우를 관리할 수 있지만 명명된 섀도는 관리할 수 없습니다.

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
```

```
      "topic": "$aws/things/+/shadow/delete/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "GetShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/get/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "UpdateShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/update",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "UpdateShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/update/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

구성 요소의 클라이언트 장치 섀도우와 상호 작용합니다.

로컬 섀도우 서비스를 사용하여 클라이언트 장치의 로컬 섀도우 문서를 읽고 수정하는 사용자 지정 구성 요소를 개발할 수 있습니다. 자세한 설명은 구성 요소의 그림자와 상호 작용 섹션을 참조하세요.

클라이언트 장치 섀도우를 다음과 동기화합니다. AWS IoT Core

로컬 클라이언트 장치 섀도우 상태를 동기화하도록 섀도우 관리자 구성 요소를 구성할 수 있습니다. AWS IoT Core 자세한 내용은 로컬 장치 섀도우를 다음과 동기화합니다. AWS IoT Core을(를) 참조하세요.

클라이언트 장치 문제 해결

이 섹션의 문제 해결 정보와 솔루션을 사용하면 Greengrass 클라이언트 장치 및 클라이언트 장치 구성 요소와 관련된 문제를 해결하는 데 도움이 됩니다.

주제

그린그래스 디스커버리 이슈

다음 정보를 사용하여 Greengrass 검색 관련 문제를 해결하십시오. 이러한 문제는 클라이언트 장치가 Greengrass 검색 API를 사용하여 연결할 수 있는 Greengrass 코어 장치를 식별할 때 발생할 수 있습니다.

주제

그린그래스 디스커버리 이슈 (HTTP API)

다음 정보를 사용하여 Greengrass 검색 관련 문제를 해결하십시오. cURL로 검색 API를 테스트하는 경우 이러한 오류가 표시될 수 있습니다.

주제

curl: (52) Empty reply from server

요청에서 비활성 AWS IoT 인증서를 지정하면 이 오류가 표시될 수 있습니다.

클라이언트 장치에 연결된 인증서가 있고 인증서가 활성 상태인지 확인하십시오. 자세한 내용은 AWS IoT Core개발자 안내서의 클라이언트 인증서에 사물 또는 정책 연결 및 클라이언트 인증서 활성화 또는 비활성화를 참조하십시오.

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

클라이언트 장치에 자체 호출 `greengrass:Discover` 권한이 없는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치의 인증서에 허용하는 정책이 있는지 확인하세요`greengrass:Discover`. `Resource`섹션의 사물 정책 변수 (`iot:Connection.Thing.*`) 는 이 권한에 사용할 수 없습니다. 자세한 설명은 디스커버리 인증 및 권한 부여 섹션을 참조하세요.

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

다음과 같은 경우에 이 오류가 표시될 수 있습니다.

- 클라이언트 장치는 Greengrass 코어 장치 또는 AWS IoT Greengrass V1 그룹과 연결되어 있지 않습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스는 클라이언트 디바이스 인증 구성 요소를 실행하지 않습니다.

클라이언트 장치가 연결하려는 코어 장치에 연결되어 있는지 확인하십시오. 그런 다음 코어 디바이스가 클라이언트 디바이스 인증 구성 요소를 실행하고 MQTT Broker 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결
- 코어 디바이스 엔드포인트 관리
- 클라우드 디스커버리 구성 (콘솔)

그린그래스 디스커버리 이슈 (파이썬의 경우 AWS IoT Device SDK v2)

Python용 v2에서 AWS IoT Device SDK Greengrass 검색과 관련된 문제를 해결하려면 다음 정보를 사용하십시오.

주제

- awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

요청에서 비활성 AWS IoT 인증서를 지정하면 이 오류가 표시될 수 있습니다.

클라이언트 장치에 연결된 인증서가 있고 인증서가 활성 상태인지 확인하십시오. 자세한 내용은 AWS IoT Core개발자 안내서의 클라이언트 인증서에 사물 또는 정책 연결 및 클라이언트 인증서 활성화 또는 비활성화를 참조하십시오.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

클라이언트 장치에 자체 호출 `greengrass:Discover` 권한이 없는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치의 인증서에 허용하는 정책이 있는지 확인하세요`greengrass:Discover`. `Resource`섹션의 사물 정책 변수 (`iot:Connection.Thing.*`) 는 이 권한에 사용할 수 없습니다. 자세한 설명은 디스커버리 인증 및 권한 부여 섹션을 참조하세요.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

다음과 같은 경우에 이 오류가 표시될 수 있습니다.

- 클라이언트 장치는 Greengrass 코어 장치 또는 AWS IoT Greengrass V1 그룹과 연결되어 있지 않습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스는 클라이언트 디바이스 인증 구성 요소를 실행하지 않습니다.

클라이언트 장치가 연결하려는 코어 장치에 연결되어 있는지 확인하십시오. 그런 다음 코어 디바이스가 클라이언트 디바이스 인증 구성 요소를 실행하고 MQTT Broker 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결

- 코어 디바이스 엔드포인트 관리
- 클라우드 디스커버리 구성 (콘솔)

그린그래스 디스커버리 이슈 (C++의 경우 AWS IoT Device SDK v2)

다음 정보를 사용하여 C++용 v2에서 Greengrass 검색 관련 문제를 AWS IoT Device SDK 해결하십시오.

주제

- aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

요청에서 비활성 AWS IoT 인증서를 지정하는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치에 연결된 인증서가 있고 인증서가 활성 상태인지 확인하십시오. 자세한 내용은 AWS IoT Core개발자 안내서의 클라이언트 인증서에 사물 또는 정책 연결 및 클라이언트 인증서 활성화 또는 비활성화를 참조하십시오.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

클라이언트 장치에 자체 호출 `greengrass:Discover` 권한이 없는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치의 인증서에 허용하는 정책이 있는지 확인하세요`greengrass:Discover`. `Resource`섹션의 사물 정책 변수 (`iot:Connection.Thing.*`) 는 이 권한에 사용할 수 없습니다. 자세한 설명은 디스커버리 인증 및 권한 부여 섹션을 참조하세요.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

다음과 같은 경우에 이 오류가 표시될 수 있습니다.

- 클라이언트 장치는 Greengrass 코어 장치 또는 AWS IoT Greengrass V1 그룹과 연결되어 있지 않습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.

- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스는 클라이언트 디바이스 인증 구성 요소를 실행하지 않습니다.

클라이언트 장치가 연결하려는 코어 장치에 연결되어 있는지 확인하십시오. 그런 다음 코어 디바이스가 클라이언트 디바이스 인증 구성 요소를 실행하고 MQTT Broker 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결
- 코어 디바이스 엔드포인트 관리
- 클라우드 디스커버리 구성 (콘솔)

그린그래스 디스커버리 이슈 (AWS IoT Device SDKv2의 경우) JavaScript

다음 정보를 사용하여 v2 양식에서 Greengrass 검색 관련 문제를 AWS IoT Device SDK해결하십시오. JavaScript

주제

- Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- Error: Discovery failed (headers: [object Object]) { response_code: 403 }
- Error: Discovery failed (headers: [object Object]) { response_code: 404 }
- Error: Discovery failed (headers: [object Object])

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

요청에서 비활성 AWS IoT 인증서를 지정하는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치에 연결된 인증서가 있고 인증서가 활성 상태인지 확인하십시오. 자세한 내용은 AWS IoT Core개발자 안내서의 클라이언트 인증서에 사물 또는 정책 연결 및 클라이언트 인증서 활성화 또는 비활성화를 참조하십시오.

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

클라이언트 장치에 자체 호출 `greengrass:Discover` 권한이 없는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치의 인증서에 허용하는 정책이 있는지 확인하세요`greengrass:Discover`. `Resource`섹션의 사물 정책 변수 (`iot:Connection.Thing.*`) 는 이 권한에 사용할 수 없습니다. 자세한 설명은 디스커버리 인증 및 권한 부여 섹션을 참조하세요.

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

다음과 같은 경우에 이 오류가 표시될 수 있습니다.

- 클라이언트 장치는 Greengrass 코어 장치 또는 AWS IoT Greengrass V1 그룹과 연결되어 있지 않습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스는 클라이언트 디바이스 인증 구성 요소를 실행하지 않습니다.

클라이언트 장치가 연결하려는 코어 장치에 연결되어 있는지 확인하십시오. 그런 다음 코어 디바이스가 클라이언트 디바이스 인증 구성 요소를 실행하고 MQTT Broker 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결
- 코어 디바이스 엔드포인트 관리
- 클라우드 디스커버리 구성 (콘솔)

Error: Discovery failed (headers: [object Object])

Greengrass 검색 샘플을 실행할 때 HTTP 응답 코드가 없는 상태에서 이 오류가 표시될 수 있습니다. 이 오류는 여러 가지 이유로 발생할 수 있습니다.

- 클라이언트 장치에 자체 호출 `greengrass:Discover` 권한이 없는 경우 이 오류가 표시될 수 있습니다.

  클라이언트 장치의 인증서에 허용하는 정책이 있는지 확인하세요`greengrass:Discover`. `Resource`섹션의 사물 정책 변수 (`iot:Connection.Thing.*`) 는 이 권한에 사용할 수 없습니다. 자세한 설명은 디스커버리 인증 및 권한 부여 섹션을 참조하세요.
- 다음과 같은 경우에 이 오류가 표시될 수 있습니다.
  - 클라이언트 장치는 Greengrass 코어 장치 또는 AWS IoT Greengrass V1 그룹과 연결되어 있지 않습니다.

- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스는 클라이언트 디바이스 인증 구성 요소를 실행하지 않습니다.

클라이언트 장치가 연결하려는 코어 장치에 연결되어 있는지 확인하십시오. 그런 다음 코어 디바이스가 클라이언트 디바이스 인증 구성 요소를 실행하고 MQTT Broker 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결
- 코어 디바이스 엔드포인트 관리
- 클라우드 디스커버리 구성 (콘솔)

그린그래스 디스커버리 이슈 (자바의 경우 AWS IoT Device SDK v2)

다음 정보를 사용하여 Java용 v2에서 Greengrass 검색 관련 문제를 AWS IoT Device SDK 해결하십시오.

주제

- software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)
- java.lang.RuntimeException: Error x-amzn-ErrorType(403)
- java.lang.RuntimeException: Error x-amzn-ErrorType(404)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

요청에서 비활성 AWS IoT 인증서를 지정하는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치에 연결된 인증서가 있고 인증서가 활성 상태인지 확인하십시오. 자세한 내용은 AWS IoT Core개발자 안내서의 클라이언트 인증서에 사물 또는 정책 연결 및 클라이언트 인증서 활성화 또는 비활성화를 참조하십시오.

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

클라이언트 장치에 자체 호출 `greengrass:Discover` 권한이 없는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치의 인증서에 허용하는 정책이 있는지 확인하세요`greengrass:Discover`. `Resource`섹션의 사물 정책 변수 (`iot:Connection.Thing.*`) 는 이 권한에 사용할 수 없습니다. 자세한 설명은 디스커버리 인증 및 권한 부여 섹션을 참조하세요.

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

다음과 같은 경우에 이 오류가 표시될 수 있습니다.

- 클라이언트 장치는 Greengrass 코어 장치 또는 AWS IoT Greengrass V1 그룹과 연결되어 있지 않습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스와 연결된 Greengrass 코어 디바이스는 클라이언트 디바이스 인증 구성 요소를 실행하지 않습니다.

클라이언트 장치가 연결하려는 코어 장치에 연결되어 있는지 확인하십시오. 그런 다음 코어 디바이스가 클라이언트 디바이스 인증 구성 요소를 실행하고 MQTT Broker 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음 자료를 참조하세요.

- 클라이언트 장치 연결
- 코어 디바이스 엔드포인트 관리
- 클라우드 디스커버리 구성 (콘솔)

MQTT 연결 문제

다음 정보를 사용하여 클라이언트 장치 MQTT 연결 문제를 해결하십시오. 이러한 문제는 클라이언트 장치가 MQTT를 통해 코어 장치에 연결하려고 할 때 발생할 수 있습니다.

주제

- io.moquette.broker.Authorizator: Client does not have read permissions on the topic
- MQTT 연결 문제 (Python)

- MQTT 연결 문제 (C++)
- MQTT 연결 문제 (Java)
- MQTT 연결 문제 () JavaScript

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

클라이언트 장치가 권한이 없는 MQTT 주제를 구독하려고 하면 Greengrass 로그에 이 오류가 표시될 수 있습니다. 오류 메시지에는 해당 주제가 포함되어 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하십시오.

- 클라이언트 장치와 일치하는 장치 그룹.
- 주제에 대한 `mqtt:subscribe` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

MQTT 연결 문제 (Python)

Python용 AWS IoT Device SDK v2를 사용할 때 다음 정보를 사용하여 클라이언트 장치 MQTT 연결 문제를 해결하십시오.

주제

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

클라이언트 장치 인증 구성 요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

클라이언트 장치 인증 구성요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

MQTT 연결 문제 (C++)

C++용 v2를 사용할 때 다음 정보를 사용하여 클라이언트 장치 MQTT 연결 문제를 해결하십시오AWS IoT Device SDK.

주제

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

클라이언트 장치 인증 구성 요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

클라이언트 장치 인증 구성요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

MQTT 연결 문제 (Java)

Java용 AWS IoT Device SDKv2를 사용할 때 다음 정보를 사용하여 클라이언트 장치 MQTT 연결 문제를 해결하십시오.

주제

- software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

클라이언트 장치 인증 구성 요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

클라이언트 장치 인증 구성요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

MQTT 연결 문제 () JavaScript

다음 정보를 사용하여 v2를 사용할 때 클라이언트 장치 MQTT 연결 문제를 해결하십시오AWS IoT Device SDK. JavaScript

주제

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

클라이언트 장치 인증 구성 요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 권한 부여 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

클라이언트 장치 인증 구성요소가 클라이언트 장치에 연결 권한을 부여하는 클라이언트 장치 인증 정책을 정의하지 않는 경우 이 오류가 표시될 수 있습니다.

클라이언트 장치 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인하세요.

- 클라이언트 장치와 일치하는 장치 그룹.
- 클라이언트 장치에 대한 `mqtt:connect` 권한을 부여하는 해당 장치 그룹의 클라이언트 장치 인증 정책.

클라이언트 장치 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 클라우드 디스커버리 구성 (콘솔)
- 클라이언트 장치 인증
- 배포 만들기

디바이스 섀도우와 상호작용

Greengrass 코어 디바이스는 컴포넌트를 사용하여 AWS IoT디바이스 섀도우와 상호 작용할 수 있습니다. 섀도우는 사물에 대한 현재 또는 원하는 상태 정보를 저장하는 JSON 문서입니다AWS IoT. 섀도는 장치가 연결되어 있는지 여부에 관계없이 다른 AWS IoT Greengrass 구성 요소가 장치의 상태를 볼 수 있도록 할 수 AWS IoT 있습니다. 각 AWS IoT 기기에는 이름 없는 고유의 클래식한 섀도우가 있습니다. 각 디바이스에 대해 이름이 지정된 섀도우를 여러 개 만들 수도 있습니다.

장치 및 서비스는 MQTT 및 예약된 MQTT 섀도우 주제를 사용하고, Device Shadow RESTAPI를 사용하는 HTTP 등을 사용하여 클라우드 섀도우를 생성, 업데이트 및 삭제할 수 있습니다. AWS CLIAWS IoT

섀도우 관리자 구성 요소를 사용하면 Greengrass 구성 요소가 로컬 섀도우 서비스 및 로컬 섀도우 게시/구독 주제를 사용하여 로컬 섀도우를 생성, 업데이트 및 삭제할 수 있습니다. 또한 섀도우 관리자는 코어 디바이스에서 이러한 로컬 섀도우 문서의 스토리지를 관리하고 섀도우 상태 정보를 클라우드 섀도우와 동기화하는 작업을 처리합니다.

섀도우 관리자 구성 요소를 사용하여 코어 장치에 연결된 클라이언트 장치의 로컬 섀도우를 관리할 수도 있습니다. 섀도우 관리자가 클라이언트 디바이스 섀도를 관리할 수 있도록 하려면 로컬 MQTT 브로커와 로컬 게시/구독 서비스 간에 메시지를 릴레이하도록 MQTT 브리지 구성 요소를 구성합니다. 자세한 설명은 클라이언트 디바이스 섀도우와 상호 작용 및 동기화 섹션을 참조하세요.

AWS IoT디바이스 섀도 개념에 대한 자세한 내용은 AWS IoT개발자 안내서의 AWS IoT Device Shadow 서비스를 참조하십시오.

주제

- 구성 요소의 그림자와 상호 작용
- 로컬 장치 섀도우를 다음과 동기화합니다. AWS IoT Core

구성 요소의 그림자와 상호 작용

Lambda 함수 구성 요소를 포함하여 로컬 섀도우 서비스를 사용하여 로컬 섀도우 문서 및 클라이언트 디바이스 섀도 문서를 읽고 수정하는 사용자 지정 구성 요소를 개발할 수 있습니다.

사용자 지정 구성 요소는 의 AWS IoT Greengrass Core IPC 라이브러리를 사용하여 로컬 섀도우 서비스와 상호 작용합니다. AWS IoT Device SDK 섀도우 관리자 구성요소는 코어 기기에서 로컬 섀도우 서비스를 활성화합니다.

섀도우 관리자 구성 요소를 Greengrass 코어 장치에 배포하려면 구성 요소가 포함된 배포를 생성하십시오`aws.greengrass.ShadowManager`.

> **Note**
>
> 기본적으로 섀도우 관리자 구성 요소를 배포하면 로컬 섀도우 작업만 활성화됩니다. 코어 디바이스 섀도 또는 클라이언트 디바이스의 섀도우에 대한 섀도우 상태 정보를 의 해당 클라우드 섀도우 문서에 동기화할 수 있도록 AWS IoT Greengrass 하려면 `synchronize` 파라미터가 포함된 섀도우 관리자 구성 요소에 대한 구성 업데이트를 만들어야 합니다. AWS IoT Core 자세한 설명은 로컬 장치 섀도우를 다음과 동기화합니다. AWS IoT Core 섹션을 참조하세요.

주제

- 섀도우 상태를 검색하고 수정합니다.
- 섀도우 상태 변경에 대응

섀도우 상태를 검색하고 수정합니다.

섀도우 IPC 작업은 로컬 섀도우 문서에서 상태 정보를 검색하고 업데이트합니다. 섀도우 관리자 구성 요소는 코어 디바이스에서 이러한 섀도우 문서의 저장을 처리합니다.

로컬 섀도우 상태를 수정하려면

1. 구성 요소가 로컬 섀도우 주제에 대한 메시지를 수신할 수 있도록 사용자 지정 구성 요소의 레시피에 권한 부여 정책을 추가합니다.

   권한 부여 정책의 예는 로컬 섀도우 IPC 권한 부여 정책 예제를 참조하십시오.
2. 섀도우 IPC 작업을 사용하여 섀도우 상태 정보를 검색하고 수정할 수 있습니다. 구성 요소 코드의 섀도우 IPC 작업 사용에 대한 자세한 내용은 을 참조하십시오. 로컬 섀도우와 상호작용

> **Note**
>
> 코어 디바이스가 클라이언트 디바이스 섀도우와 상호 작용할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 섀도우 관리자가 클라이언트 장치와 통신하도록 활성화를 참조하십시오.

섀도우 상태 변경에 대응

Greengrass 구성 요소는 로컬 게시/구독 인터페이스를 사용하여 코어 디바이스에서 통신합니다. 사용자 지정 구성 요소가 섀도우 상태 변경에 반응하도록 하려면 로컬 게시/구독 주제를 구독하면 됩니다. 이렇게 하면 구성 요소가 로컬 섀도우 주제에 대한 메시지를 수신한 다음 해당 메시지에 대해 조치를 취할 수 있습니다.

로컬 섀도우 토픽은 AWS IoT 디바이스 섀도우 MQTT 토픽과 동일한 형식을 사용합니다. 섀도우 주제에 대한 자세한 내용은 AWS IoT개발자 안내서의 Device Shadow MQTT 항목을 참조하십시오.

로컬 섀도우 상태 변경에 대응하려면

1. 사용자 지정 구성 요소의 레시피에 액세스 제어 정책을 추가하여 구성 요소가 로컬 섀도우 주제에 대한 메시지를 수신할 수 있도록 합니다.

   권한 부여 정책의 예는 로컬 섀도우 IPC 권한 부여 정책 예를 참조하십시오.
2. 구성 요소에서 사용자 지정 작업을 시작하려면 `SubscribeToTopic` IPC 작업을 사용하여 메시지를 수신하려는 섀도우 주제를 구독하십시오. 구성 요소 코드에서 로컬 게시/구독 IPC 작업을 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 로컬 메시지 게시/구독
3. Lambda 함수를 호출하려면 이벤트 소스 구성을 사용하여 섀도우 주제의 이름을 제공하고 로컬 게시/구독 주제임을 지정하십시오. Lambda 함수 구성 요소 생성에 대한 자세한 내용은 을 참조하십시오. AWS Lambda함수 실행

> **Note**
>
> 코어 디바이스가 클라이언트 디바이스 섀도우와 상호 작용할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 섀도우 관리자가 클라이언트 장치와 통신하도록 활성화를 참조하십시오.

로컬 장치 섀도우를 다음과 동기화합니다. AWS IoT Core

섀도우 관리자 구성 요소를 사용하면 로컬 장치 섀도우 상태를 AWS IoT Greengrass 동기화할 수 AWS IoT Core 있습니다. 구성 매개변수를 포함하도록 섀도우 관리자 구성 요소의 `synchronization` 구성을 수정하고 장치의 AWS IoT 사물 이름과 동기화할 섀도우를 지정해야 합니다.

섀도우 관리자를 구성하여 섀도우를 동기화하면 로컬 섀도우 문서에서 발생하든 클라우드 섀도우 문서에서 발생하든 관계없이 지정된 섀도우에 대한 모든 상태 변경이 동기화됩니다.

또한 섀도우 관리자 구성 요소가 섀도우를 실시간으로 동기화할지 또는 주기적으로 동기화하는지도 지정할 수 있습니다. 기본적으로 섀도우 관리자 구성 요소는 실시간으로 섀도우를 동기화하므로 코어 디바이스는 업데이트가 AWS IoT Core 발생할 때마다 섀도우 업데이트를 주고 받습니다. 주기적인 간격을 구성하여 대역폭 사용량과 요금을 줄일 수 있습니다.

주제

사전 조건

로컬 섀도우를 동기화하려면 다음 AWS IoT Core 섀도우 AWS IoT 정책 작업을 허용하도록 Greengrass 코어 디바이스의 정책을 구성해야 합니다. AWS IoT Core

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

자세한 내용은 다음 자료를 참조하세요.

- AWS IoT CoreAWS IoT개발자 안내서의 정책 조치
- AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책
- 코어 디바이스 정책 업데이트 AWS IoT

섀도우 관리자 구성 요소 구성

섀도우 매니저에는 로컬 섀도우 문서의 섀도우 상태 정보를 클라우드 섀도우 문서에 동기화하기 위한 섀도우 이름 매핑 목록이 필요합니다. AWS IoT Core

섀도우 상태를 동기화하려면 `aws.greengrass.ShadowManager` 구성 요소가 포함된 배포를 만들고 배포의 섀도우 관리자 구성의 `synchronize` 구성 매개 변수에 동기화할 섀도우를 지정합니다.

> **Note**
>
> 코어 디바이스가 클라이언트 디바이스 섀도우와 상호 작용할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 섀도우 관리자가 클라이언트 장치와 통신하도록 활성화를 참조하십시오.

다음 예제 구성 업데이트는 섀도우 관리자 구성 요소에 다음 섀도우를 다음과 동기화하도록 지시합니다. AWS IoT Core

- 코어 디바이스의 클래식 섀도우
- 코어 `MyCoreShadow` 디바이스의 이름
- IoT라는 이름의 클래식한 그림자 `MyDevice2`
- 이름이 지정된 `MyShadowA` 섀도우와 `MyShadowB` IoT 사물의 이름 `MyDevice1`

이 구성 업데이트는 섀도우를 AWS IoT Core 실시간으로 동기화하도록 지정합니다. 섀도우 관리자 v2.1.0 이상을 사용하는 경우 섀도우를 주기적으로 동기화하도록 섀도우 관리자 구성 요소를 구성할 수 있습니다. 이 기능을 구성하려면 동기화 전략을 로 변경하고 `periodic` 간격을 초 `delay` 단위로 지정하십시오. 자세한 내용은 섀도우 관리자 구성 요소의 전략 구성 매개 변수를 참조하십시오.

이 구성 업데이트는 섀도우를 코어 디바이스 간에 AWS IoT Core 양방향으로 동기화하도록 지정합니다. 섀도우 관리자 v2.2.0 이상을 사용하는 경우 섀도우를 한 방향으로만 동기화하도록 섀도우 관리자 구성 요소를 구성할 수 있습니다. 이 기능을 `direction` 구성하려면 `deviceToCloud` 동기화를 또는 로 변경하십시오. `cloudToDevice` 자세한 내용은 섀도우 매니저 컴포넌트의 방향 구성 파라미터를 참조하십시오.

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
```

```
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
}
```

로컬 섀도우 동기화

Greengrass 코어 디바이스가 AWS IoT 클라우드에 연결되면 섀도우 관리자는 구성 요소 구성에서 지정한 섀도우에 대해 다음 작업을 수행합니다. 동작은 지정한 섀도우 동기화 방향 구성 옵션에 따라 달라집니다. 기본적으로 섀도우 관리자는 이 `betweenDeviceAndCloud` 옵션을 사용하여 섀도우를 양방향으로 동기화합니다. Shadow Manager v2.2.0 이상을 사용하는 경우 섀도우를 한 방향 (또는) 으로만 동기화하도록 코어 디바이스를 구성할 수 `cloudToDevice` 있습니다. `deviceToCloud`

- 섀도우 동기화 방향 구성이 `betweenDeviceAndCloud` 또는 `cloudToDevice` 인 경우 섀도우 관리자는 의 클라우드 섀도우 문서에서 보고된 상태 정보를 검색합니다. AWS IoT Core 그런 다음 로컬에 저장된 섀도우 문서를 업데이트하여 장치 상태를 동기화합니다.
- 섀도 동기화 방향 구성이 `betweenDeviceAndCloud` 또는 `deviceToCloud` 인 경우 섀도우 관리자는 디바이스의 현재 상태를 클라우드 섀도 문서에 게시합니다.

섀도우 병합 충돌 동작

코어 디바이스의 인터넷 연결이 끊긴 경우와 같이 일부 경우에는 섀도우 관리자가 변경 내용을 동기화하기 전에 로컬 섀도우 서비스와 AWS IoT 클라우드의 섀도우가 변경될 수 있습니다. 따라서 로컬 섀도우 서비스와 클라우드 간에 원하는 상태와 보고되는 상태가 다릅니다. AWS IoT

섀도우 관리자가 섀도우를 동기화할 때 다음 동작에 따라 변경 내용이 병합됩니다.

- v2.2.0 이전 버전의 섀도우 관리자를 사용하거나 `betweenDeviceAndCloud` 섀도우 동기화 방향을 지정하는 경우 다음 동작이 적용됩니다.
  - 섀도우가 원하는 상태에서 병합 충돌이 발생하면 섀도우 관리자는 로컬 섀도우 문서의 충돌 부분을 클라우드의 값으로 덮어씁니다. AWS IoT
  - 섀도우가 보고된 상태에서 병합 충돌이 발생하면 섀도우 관리자는 AWS IoT 클라우드에 있는 섀도우의 충돌 부분을 로컬 섀도우 문서의 값으로 덮어씁니다.
- `deviceToCloud`섀도우 동기화 방향을 지정하면 섀도우 관리자가 AWS IoT 클라우드에 있는 섀도우의 충돌 부분을 로컬 섀도우 문서의 값으로 덮어씁니다.
- `cloudToDevice`섀도우 동기화 방향을 지정하면 섀도우 관리자가 로컬 섀도우 문서의 충돌 부분을 클라우드의 값으로 덮어씁니다. AWS IoT

Greengrass 코어 디바이스의 데이터 스트림 관리

AWS IoT Greengrass스트림 관리자를 사용하면 대용량 IoT 데이터를 보다 효율적이고 안정적으로 전송할 수 있습니다. AWS 클라우드 스트림 관리자는 데이터 스트림을 AWS IoT Greengrass Core로 내보내기 전에 Core에서 처리합니다. AWS 클라우드 Stream Manager는 머신 러닝 (ML) 추론과 같은 일반적인 엣지 시나리오와 통합됩니다. 이 시나리오에서는 AWS IoT Greengrass 코어 디바이스가 데이터를 처리하고 분석하여 데이터를 로컬 스토리지 대상으로 AWS 클라우드 내보내거나 로컬 스토리지 대상으로 내보내는 방식입니다.

Stream Manager는 사용자 지정 구성 요소 개발을 간소화하는 공통 인터페이스를 제공하므로 사용자 지정 스트림 관리 기능을 구축할 필요가 없습니다. 구성 요소는 표준화된 메커니즘을 사용하여 대용량 스트림을 처리하고 로컬 데이터 보존 정책을 관리할 수 있습니다. 각 스트림의 스토리지 유형, 크기 및 데이터 보존 정책을 정의하여 Stream Manager가 데이터를 처리하고 내보내는 방법을 제어할 수 있습니다.

Stream Manager는 연결이 간헐적이거나 제한적인 환경에서 작동합니다. 대역폭 사용, 타임아웃 동작, AWS IoT Greengrass 코어가 연결되거나 연결이 끊겼을 때 스트림 데이터를 처리하는 방식을 정의할 수 있습니다. 또한 우선 순위를 설정하여 AWS IoT Greengrass Core에서 스트림을 내보내는 순서를 제어할 수 있습니다. AWS 클라우드 이렇게 하면 다른 데이터보다 중요한 데이터를 더 빨리 처리할 수 있습니다.

저장 또는 추가 처리 및 분석을 위해 데이터를 로 자동 내보내도록 스트림 관리자를 구성할 수 있습니다. AWS 클라우드 스트림 관리자는 다음 AWS 클라우드 목적지로의 내보내기를 지원합니다.

- 채널 입력AWS IoT Analytics. AWS IoT Analytics데이터에 대한 고급 분석을 수행하여 비즈니스 결정을 내리고 머신 러닝 모델을 개선할 수 있습니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 (AWS IoT Analytics이)란 무엇입니까? 섹션을 참조하십시오.
- Amazon Kinesis Data Streams의 스트리밍. Kinesis Data Streams를 사용하여 대용량 데이터를 집계하고 이를 데이터 웨어하우스 또는 클러스터에 로드할 수 있습니다. MapReduce 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams이란 무엇입니까?를 참조하세요.
- 의 자산 속성. AWS IoT SiteWise AWS IoT SiteWise산업 장비의 데이터를 대규모로 수집, 구성 및 분석할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의(AWS IoT SiteWise이)란 무엇입니까? 섹션을 참조하십시오.
- 아마존 심플 스토리지 서비스 Amazon S3의 객체. Amazon S3를 사용하여 대량의 데이터를 저장 및 검색할 수 있습니다. 자세한 내용은 Amazon S3란 무엇입니까? 를 참조하십시오. Amazon 심플 스토리지 서비스 개발자 가이드에서 확인할 수 있습니다.

스트림 관리 워크플로우

IoT 애플리케이션은 스트림 관리자 SDK를 통해 스트림 관리자와 상호 작용합니다.

간단한 워크플로우에서 AWS IoT Greengrass 코어의 구성 요소는 시계열 온도 및 압력 지표와 같은 IoT 데이터를 소비합니다. 구성요소는 데이터를 필터링하거나 압축한 다음 Stream Manager SDK를 호출하여 스트림 관리자의 스트림에 데이터를 쓸 수 있습니다. 스트림 관리자는 스트림에 대해 정의한 정책에 따라 스트림을 로 AWS 클라우드 자동으로 내보낼 수 있습니다. 또한 구성 요소는 데이터를 로컬 데이터베이스 또는 스토리지 리포지토리로 직접 보낼 수 있습니다.

IoT 애플리케이션에는 스트림을 읽거나 스트림에 쓰는 사용자 지정 구성 요소가 여러 개 포함될 수 있습니다. 이러한 구성 요소는 스트림을 읽고 작성하여 AWS IoT Greengrass 코어 디바이스의 데이터를 필터링, 집계 및 분석할 수 있습니다. 따라서 데이터가 코어에서 로컬 목적지로 전송되기 전에 로컬 이벤트에 신속하게 대응하고 중요한 정보를 추출할 AWS 클라우드 수 있습니다.

시작하려면 스트림 관리자 구성 요소를 AWS IoT Greengrass 코어 장치에 배포하세요. 배포 시 스트림 관리자 구성 요소 매개변수를 구성하여 Greengrass 코어 장치의 모든 스트림에 적용되는 설정을 정의합니다. 이러한 매개변수를 사용하여 비즈니스 요구 사항 및 환경 제약에 따라 Stream Manager가 스트림을 저장, 처리 및 내보내는 방법을 제어할 수 있습니다.

스트림 관리자를 구성한 후 IoT 애플리케이션을 생성하고 배포할 수 있습니다. 이러한 구성 요소는 일반적으로 Stream Manager `StreamManagerClient` SDK에서 스트림을 생성하고 스트림과 상호 작용하는 데 사용하는 사용자 지정 구성 요소입니다. 스트림을 생성할 때 내보내기 대상, 우선 순위, 지속성과 같은 스트림별 정책을 정의할 수 있습니다.

요구 사항

스트림 관리자 사용을 위해 다음 요구 사항이 적용됩니다.

- 스트림 관리자에는 AWS IoT Greengrass Core 소프트웨어 외에 최소 70MB의 RAM이 필요합니다. 총 메모리 요구 사항은 워크로드에 따라 다릅니다.
- AWS IoT Greengrass구성 요소는 스트림 관리자 SDK를 사용하여 스트림 관리자와 상호 작용해야 합니다. 스트림 관리자 SDK는 다음 언어로 제공됩니다.
  - 자바용 스트림 매니저 SDK (v1.1.0 이상)
  - Node.js 용 스트림 매니저 SDK (v1.1.0 이상)
  - Python용 스트림 매니저 SDK (v1.1.0 이상)
- AWS IoT Greengrass스트림 관리자를 사용하려면 컴포넌트가 레시피에 스트림 관리자 컴포넌트 (`aws.greengrass.StreamManager`) 를 종속성으로 지정해야 합니다.

> **Note**
>
> 스트림 관리자를 사용하여 데이터를 클라우드로 내보내는 경우 스트림 관리자 구성 요소 버전 2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 수 없습니다. 스트림 관리자를 처음 배포하는 경우 스트림 관리자 구성 요소의 최신 버전을 배포하는 것이 좋습니다.

- 스트림의 AWS 클라우드 내보내기 대상을 정의하는 경우 내보내기 대상을 만들고 Greengrass 장치 역할에 액세스 권한을 부여해야 합니다. 대상에 따라 다른 요구 사항도 적용될 수 있습니다. 자세한 내용은 다음을 참조하십시오.
  - 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 Greengrass코어 디바이스의 로컬 구성 요소 간에 저장된 스트림 데이터나 전송 중인 스트림 데이터는 암호화하지 않습니다.

- 저장 시 데이터. 스트림 데이터는 스토리지 디렉터리에 로컬로 저장됩니다. 데이터 보안을 위해 파일 권한과 전체 디스크 암호화 (활성화된 경우) 를 사용합니다. AWS IoT Greengrass 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 Greengrass데이터 소스, AWS IoT Greengrass 구성 요소, Stream Manager SDK 및 스트림 관리자 간의 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.

클라이언트 인증

스트림 관리자 클라이언트는 스트림 관리자 SDK를 사용하여 스트림 관리자와 통신합니다. 클라이언트 인증이 활성화되면 Greengrass 구성 요소만 스트림 관리자의 스트림과 상호 작용할 수 있습니다. 클라이언트 인증이 비활성화되면 Greengrass 코어 장치에서 실행되는 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. 스트림 관리자 구성 요소를 코어 장치에 배포할 때 이 매개변수를 구성할 수 있습니다.

	활성화됨	Disabled(비활성)
파라미터 값	`true`(기본값 및 권장)	`false`
허용된 클라이언트	코어 디바이스의 Greengrass 구성 요소	코어 디바이스의 Greengrass 구성 요소 Greengrass 코어 디바이스에서 실행 중인 기타 프로세스

다음 사항도 참조하십시오.

- the section called “스트림 관리자 구성”
- the section called “스트림 StreamManagerClient 작업에 사용”
- the section called “지원되는 클라우드 대상의 내보내기 구성”

스트림 관리자를 사용하는 사용자 지정 구성 요소 만들기

사용자 지정 Greengrass 구성 요소의 스트림 관리자를 사용하여 IoT 장치 데이터를 저장, 처리 및 내보낼 수 있습니다. 이 섹션의 절차와 예제를 사용하여 Stream Manager와 함께 작동하는 구성 요소 레시피, 아티팩트 및 애플리케이션을 만들 수 있습니다. 구성 요소를 개발 및 테스트하는 방법에 대한 자세한 내용은 단원을 참조하세요AWS IoT Greengrass 구성 요소 생성.

주제

- 스트림 매니저를 사용하는 컴포넌트 레시피 정의
- 애플리케이션 코드에서 스트림 관리자에 Connect

스트림 매니저를 사용하는 컴포넌트 레시피 정의

사용자 지정 구성 요소에서 Stream Manager를 사용하려면 `aws.greengrass.StreamManager` 구성 요소를 종속성으로 정의해야 합니다. 스트림 매니저 SDK도 제공해야 합니다. 다음 작업을 완료하여 원하는 언어로 Stream Manager SDK를 다운로드하여 사용하세요.

자바용 스트림 매니저 SDK 사용

Java용 Stream Manager SDK는 구성 요소를 컴파일하는 데 사용할 수 있는 JAR 파일로 제공됩니다. 그런 다음 Stream Manager SDK가 포함된 애플리케이션 JAR을 생성하고, 애플리케이션 JAR을 구성 요소 아티팩트로 정의하고, 구성 요소 라이프사이클에서 애플리케이션 JAR을 실행할 수 있습니다.

자바용 스트림 매니저 SDK를 사용하려면

1. Java용 스트림 매니저 SDK JAR 파일을 다운로드합니다.
2. 다음 중 하나를 수행하여 Java 애플리케이션과 Stream Manager SDK JAR 파일에서 컴포넌트 아티팩트를 생성하세요.
   - 애플리케이션을 Stream Manager SDK JAR이 포함된 JAR 파일로 빌드하고 컴포넌트 레시피에서 이 JAR 파일을 실행합니다.
   - 스트림 매니저 SDK JAR을 컴포넌트 아티팩트로 정의합니다. 구성 요소 레시피에서 애플리케이션을 실행할 때 해당 아티팩트를 클래스 경로에 추가합니다.

   구성 요소 레시피는 다음 예시와 비슷합니다. 이 구성 요소는 StreamManagerS3.java 예제의 수정된 버전을 실행합니다. 여기에는 스트림 관리자 SDK JAR이 `StreamManagerS3.jar` 포함됩니다.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
```

```
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

구성 요소를 개발 및 테스트하는 방법에 대한 자세한 내용은 단원을 참조하세요AWS IoT Greengrass 구성 요소 생성.

파이썬용 스트림 매니저 SDK 사용

Python용 Stream Manager SDK는 구성 요소에 포함할 수 있는 소스 코드로 제공됩니다. Stream Manager SDK의 ZIP 파일을 만들고, ZIP 파일을 구성 요소 아티팩트로 정의하고, 구성 요소 라이프사이클에 SDK의 요구 사항을 설치합니다.

파이썬용 스트림 매니저 SDK를 사용하려면

1. aws-greengrass-stream-manager-sdk-python 저장소를 복제하거나 다운로드합니다.

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-python.git
```

2. Python용 Stream Manager SDK의 소스 코드가 들어 있는 `stream_manager` 폴더를 포함하는 ZIP 파일을 생성합니다. 이 ZIP 파일을 구성 요소 아티팩트로 제공하여 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 설치할 때 압축을 풀 수 있습니다. 해결 방법:

   a. 이전 단계에서 복제하거나 다운로드한 저장소가 들어 있는 폴더를 엽니다.

   ```
   cd aws-greengrass-stream-manager-sdk-python
   ```

   b. `stream_manager`폴더를 이름이 지정된 ZIP 파일로 `stream_manager_sdk.zip` 압축합니다.

   Linux or Unix

   ```
   zip -rv stream_manager_sdk.zip stream_manager
   ```

   Windows Command Prompt (CMD)

   ```
   tar -acvf stream_manager_sdk.zip stream_manager
   ```

   PowerShell

   ```
   Compress-Archive stream_manager stream_manager_sdk.zip
   ```

   c. `stream_manager_sdk.zip`파일에 `stream_manager` 폴더와 해당 내용이 들어 있는지 확인합니다. 다음 명령을 실행하여 ZIP 파일의 내용을 나열합니다.

   Linux or Unix

   ```
   unzip -l stream_manager_sdk.zip
   ```

   Windows Command Prompt (CMD)

   ```
   tar -tf stream_manager_sdk.zip
   ```

다음과 같이 출력됩니다.

```
Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 20:45   stream_manager/
      913  02-24-2021 20:45   stream_manager/__init__.py
     9719  02-24-2021 20:45   stream_manager/utilinternal.py
     1412  02-24-2021 20:45   stream_manager/exceptions.py
     1004  02-24-2021 20:45   stream_manager/util.py
        0  02-24-2021 20:45   stream_manager/data/
   254463  02-24-2021 20:45   stream_manager/data/__init__.py
    26515  02-24-2021 20:45   stream_manager/streammanagerclient.py
---------                     -------
   294026                     8 files
```

3. 스트림 매니저 SDK 아티팩트를 컴포넌트의 아티팩트 폴더에 복사합니다. 구성 요소는 스트림 관리자 SDK ZIP 파일 외에도 SDK `requirements.txt` 파일을 사용하여 스트림 관리자 SDK의 종속성을 설치합니다. *~/greengrass-components#* 로컬 개발에 사용하는 폴더 경로로 바꾸십시오.

   Linux or Unix

   ```
   cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
   com.example.StreamManagerS3Python/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 requirements.txt
   ```

   PowerShell

   ```
   cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0\
   ```

4. 컴포넌트 레시피를 생성하세요. 레시피에서 다음을 수행합니다.

a. `stream_manager_sdk.zip`및 를 `requirements.txt` 아티팩트로 정의합니다.

b. Python 애플리케이션을 아티팩트로 정의하십시오.

c. 설치 라이프사이클에서 에서 Stream Manager SDK 요구 사항을 설치하십시오. `requirements.txt`

d. 실행 라이프사이클에서 Stream Manager SDK를 PYTHONPATH 추가하고 Python 애플리케이션을 실행합니다.

구성 요소 레시피는 다음 예시와 비슷합니다. 이 구성 요소는 stream_manager_s3.py 예제를 실행합니다.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Python",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
```

```
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

구성 요소를 개발 및 테스트하는 방법에 대한 자세한 내용은 단원을 참조하세요AWS IoT Greengrass 구성 요소 생성.

다음과 같은 경우 스트림 매니저 SDK를 사용하세요 JavaScript

에 대한 Stream Manager JavaScript SDK는 구성 요소에 포함할 수 있는 소스 코드로 제공됩니다. Stream Manager SDK의 ZIP 파일을 생성하고 ZIP 파일을 구성 요소 아티팩트로 정의한 다음 구성 요소 라이프사이클에 SDK를 설치합니다.

스트림 매니저 SDK를 사용하려면 JavaScript

1. aws-greengrass-stream-manager-sdk-js 저장소를 복제하거나 다운로드합니다.

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
   ```

2. 에 대한 Stream Manager SDK의 소스 코드가 들어 있는 `aws-greengrass-stream-manager-sdk` 폴더를 포함하는 ZIP 파일을 만드세요. JavaScript 이 ZIP 파일을 구성 요소 아티팩트로 제공하여 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 설치할 때 압축을 풀 수 있습니다. 해결 방법:

   a. 이전 단계에서 복제하거나 다운로드한 저장소가 들어 있는 폴더를 엽니다.

      ```
      cd aws-greengrass-stream-manager-sdk-js
      ```

   b. `aws-greengrass-stream-manager-sdk`폴더를 이름이 지정된 ZIP 파일로 `stream-manager-sdk.zip` 압축합니다.

      Linux or Unix

      ```
      zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
      ```

      Windows Command Prompt (CMD)

      ```
      tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
      ```

      PowerShell

      ```
      Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
      ```

   c. `stream-manager-sdk.zip`파일에 `aws-greengrass-stream-manager-sdk` 폴더와 해당 내용이 들어 있는지 확인합니다. 다음 명령을 실행하여 ZIP 파일의 내용을 나열합니다.

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

다음과 같이 출력됩니다.

```
Archive:  stream-manager-sdk.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/
      369  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/package.json
     1017  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/util.js
     8374  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/utilInternal.js
     1937  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/exceptions.js
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/
   353343  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/index.js
    22599  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/client.js
      216  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/index.js
---------                     -------
   387855                     9 files
```

3. 스트림 매니저 SDK 아티팩트를 컴포넌트의 아티팩트 폴더에 복사합니다. *~/greengrass-components#* 로컬 개발에 사용하는 폴더 경로로 바꾸십시오.

   Linux or Unix

   ```
   cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
   com.example.StreamManagerS3JS/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
   ```

PowerShell

```
cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
\com.example.StreamManagerS3JS\1.0.0\
```

4. 컴포넌트 레시피를 생성하세요. 레시피에서 다음을 수행합니다.

   a. `stream-manager-sdk.zip`아티팩트로 정의합니다.

   b. JavaScript애플리케이션을 아티팩트로 정의하십시오.

   c. 설치 라이프사이클에서 `stream-manager-sdk.zip` 아티팩트에서 Stream Manager SDK를 설치합니다. 이 `npm install` 명령은 스트림 매니저 SDK와 종속성을 포함하는 `node_modules` 폴더를 만듭니다.

   d. 실행 라이프사이클에서 `node_modules` 폴더를 추가하고 `NODE_PATH` JavaScript 애플리케이션을 실행합니다.

   구성 요소 레시피는 다음 예시와 비슷합니다. 이 구성 요소는 StreamManagerS3 예제를 실행합니다.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3JS",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
```

```
        "run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

구성 요소를 개발 및 테스트하는 방법에 대한 자세한 내용은 단원을 참조하세요AWS IoT Greengrass 구성 요소 생성.

애플리케이션 코드에서 스트림 관리자에 Connect

애플리케이션의 스트림 매니저에 연결하려면 스트림 매니저 `StreamManagerClient` SDK에서 인스턴스를 만드세요. 이 클라이언트는 기본 포트 8088 또는 지정한 포트에서 Stream Manager 구성 요소에 연결됩니다. 인스턴스를 만든 `StreamManagerClient` 후 사용하는 방법에 대한 자세한 내용은 단원을 참조하세요스트림 StreamManagerClient 작업에 사용.

Example 예: 기본 포트를 사용하여 스트림 관리자에 Connect

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;

public class MyStreamManagerComponent {

    void connectToStreamManagerWithDefaultPort() {
        StreamManagerClient client = StreamManagerClientFactory.standard().build();

        // Use the client.
    }
}
```

Python

```
from stream_manager import (
    StreamManagerClient
)

def connect_to_stream_manager_with_default_port():
    client = StreamManagerClient()

    # Use the client.
```

JavaScript

```
const {
    StreamManagerClient
} = require('aws-greengrass-stream-manager-sdk');

function connectToStreamManagerWithDefaultPort() {
    const client = new StreamManagerClient();
```

```
    // Use the client.
}
```

Example 예: 기본 포트가 아닌 포트를 사용하여 스트림 관리자에 Connect

기본 포트가 아닌 다른 포트로 Stream Manager를 구성하는 경우 프로세스 간 통신을 사용하여 구성 요소 구성에서 포트를 검색해야 합니다.

> **Note**
>
> `port`구성 매개변수에는 스트림 관리자를 `STREAM_MANAGER_SERVER_PORT` 배포할 때 지정하는 값이 포함됩니다.

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
 IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
 GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
            greengrassCoreIPCClient.getConfiguration(request,
 Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

 .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
 StreamManagerClientFactory.standard().withClientConfig(config).build();

    // Use the client.
```

```
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
    StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

스트림 StreamManagerClient 작업에 사용

Greengrass 코어 디바이스에서 실행되는 사용자 정의 Greengrass 구성 요소는 Stream Manager SDK의 객체를 사용하여 `StreamManagerClient` 스트림 관리자에서 스트림을 만든 다음 스트림과 상호 작용할 수 있습니다. 구성 요소가 스트림을 생성할 때 스트림의 AWS 클라우드 대상, 우선 순위 지정, 기타 내보내기 및 데이터 보존 정책을 정의합니다. 스트림 관리자로 데이터를 보내기 위해 구성 요소는 스트림에 데이터를 추가합니다. 스트림에 대해 내보내기 대상이 정의된 경우 스트림 관리자는 스트림을 자동으로 내보냅니다.

> **Note**
>
> 일반적으로 스트림 관리자의 클라이언트는 사용자 정의 Greengrass 구성 요소입니다. 비즈니스 사례에 필요한 경우 Greengrass 코어 (예: Docker 컨테이너) 에서 실행되는 구성 요소가 아닌 프로세스가 스트림 관리자와 상호 작용하도록 허용할 수도 있습니다. 자세한 설명은 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.

Lambda 함수에서 스트림 관리자를 사용하는 경우 Lambda 함수는 함수 핸들러 외부에서 인스턴스화해야 합니다. `StreamManagerClient` 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 Greengrass 컴포넌트는 create 메서드를 호출하고 객체를 전달합니다. `MessageStreamDefinition` 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 설명은 the section called “지원되는 클라우드 대상의 내보내기 구성” 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.
- T ime-to-live (TTL). 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.

- MessageStreamDefinition자바 SDK에서
- MessageStreamDefinitionNode.js SDK에서
- MessageStreamDefinitionPython SDK에서

> **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 생성된 후 Greengrass 구성요소는 스트림에 메시지를 추가하여 내보낼 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 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

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: 메시지_스트림 생성하기 | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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)
                                    .withS3(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

자바 SDK createMessageStream레퍼런스: | 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)
            .withS3(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: | createMessageStreamMessageStreamDefinition

내보내기 대상 구성에 대한 자세한 내용은 the section called “지원되는 클라우드 대상의 내보내기 구성” 단원을 참조하세요.

메시지 추가

내보내기를 위해 스트림 관리자로 데이터를 보내려면 Greengrass 구성요소가 데이터를 대상 스트림에 추가합니다. 내보내기 대상은 이 메소드에 전달할 데이터 유형을 결정합니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

AWS IoT Analytics 또는 Kinesis Data Streams 내보내기 대상

다음 코드 조각은 `StreamName`이라는 스트림에 메시지를 추가합니다. Kinesis Data Streams 대상의 경우 Greengrass 구성 요소는 데이터 덩어리를 추가합니다. AWS IoT Analytics

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

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 =
 StreamManagerClientFactory.standard().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. 대상의 AWS IoT SiteWise 경우 Greengrass 컴포넌트는 직렬화된 객체를 추가합니다. `PutAssetPropertyValueEntry` 자세한 설명은 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 내용을 알아보려면 AWS IoT SiteWise API 참조의 BatchPutAssetPropertyValue 섹션을 참조하십시오.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
 not earlier
    # than 10 minutes in the past. 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",
 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: 메시지 추가 | 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 and also needs
 timestamps not earlier
    // than 10 minutes in the past. 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.
}
```

자바 SDK 레퍼런스: 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: 메시지 추가하기 | PutAssetPropertyValueEntry

Amazon S3 대상

다음 코드 조각은 `StreamName`이라는 스트림에 내보내기 작업을 추가합니다. Amazon S3 대상의 경우 Greengrass 구성 요소는 소스 입력 파일 및 대상 Amazon S3 `S3ExportTaskDefinition` 객체에 대한 정보가 포함된 직렬화된 객체를 추가합니다. 지정된 객체가 없는 경우 Stream Manager가 자동으로 생성합니다. 자세한 설명은 the section called “Amazon S3로 내보내기” 섹션을 참조하세요.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

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",
 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: 메시지 추가 | S3 ExportTaskDefinition

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.
}
```

자바 SDK 참조: AppendMessage | S3 ExportTaskDefinition

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 | S3 ExportTaskDefinition

메시지 읽기

스트림의 메시지 읽기

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

다음 코드 조각은 `StreamName`이라는 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: 메시지 읽기 | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().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.
}
```

자바 SDK 레퍼런스: 읽기메시지 | 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: 읽기메시지 | ReadMessagesOptions

스트림 나열

스트림 매니저에서 스트림 목록을 가져옵니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

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 =
 StreamManagerClientFactory.standard().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 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

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 =
 StreamManagerClientFactory.standard().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.
}
```

자바 SDK 레퍼런스: 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.

Greengrass 구성 요소는 다음과 같은 고급 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 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

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: | updateMessageStreamMessageStreamDefinition

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.
}
```

[자바 SDK 레퍼런스: 업데이트_메시지_스트림 | 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 updateMessageStream레퍼런스: | 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 스트림 매니저 SDK 버전: 파이썬: 1.1.0 | 자바: 1.1.0 | Node.js: 1.1.0

예제

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 =
 StreamManagerClientFactory.standard().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.
});
```

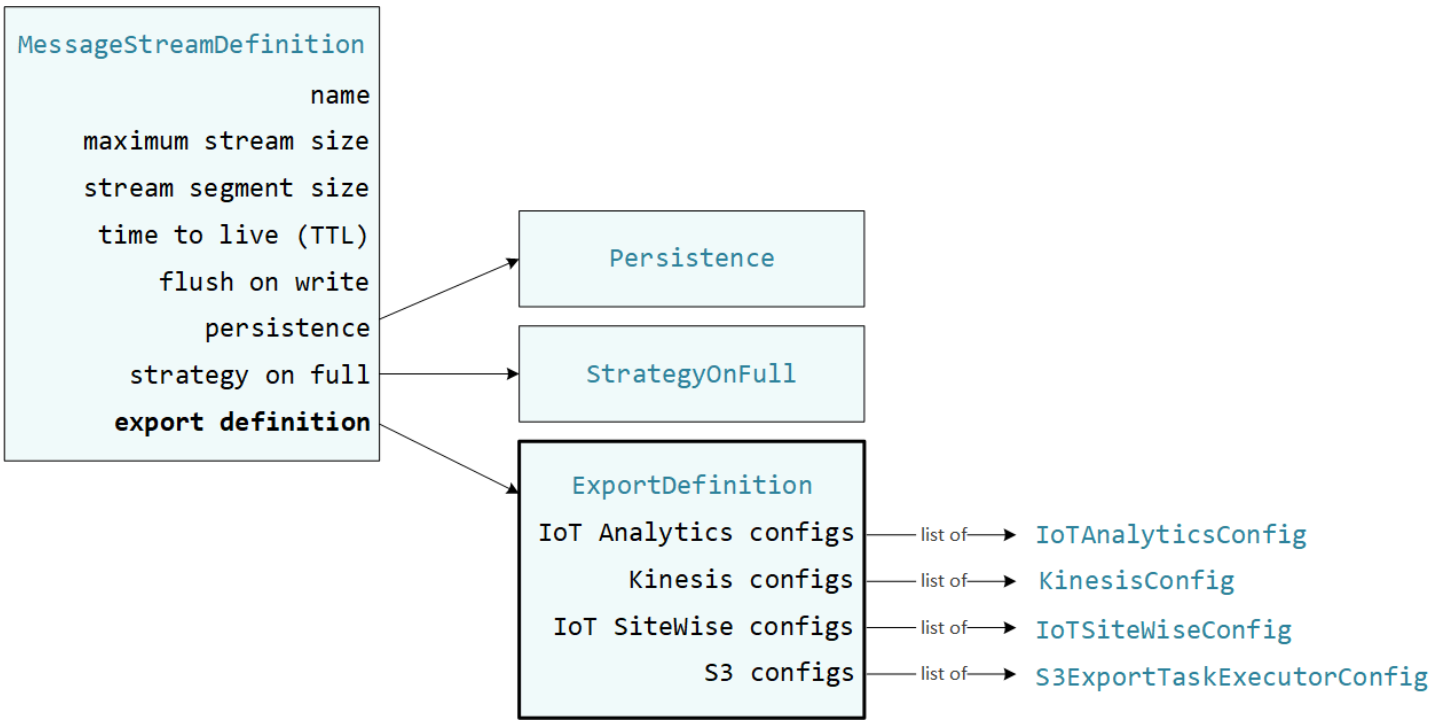
Node.js SDK 레퍼런스: deleteMessageStream

다음 사항도 참조하십시오.

- Greengrass 코어 디바이스의 데이터 스트림 관리
- AWS IoT Greengrass 스트림 관리자 구성
- 지원되는 AWS 클라우드 대상의 구성 내보내기
- `StreamManagerClient`스트림 매니저 SDK 레퍼런스에서:
  - Python
  - Java
  - Node.js

지원되는 AWS 클라우드 대상의 구성 내보내기

사용자 정의 Greengrass 구성 요소는 `StreamManagerClient` 스트림 관리자 SDK에서 스트림 관리자와 상호 작용하는 데 사용됩니다. 구성 요소가 스트림을 만들거나 스트림을 업데이트하면 내보내기 정의를 포함하여 스트림 속성을 나타내는 `MessageStreamDefinition` 개체를 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.

Stream Manager SDK에서 Greengrass 구성요소는 를 사용하여 `IoTAnalyticsConfig` 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python AnalyticsConfig SDK에서의 IoT
- 자바 AnalyticsConfig SDK에서의 IoT
- Node.js AnalyticsConfig SDK의 IoT

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 입력 대상 채널은 Greengrass 코어 AWS 계정 기기와 AWS IoT Analytics AWS 리전 동일해야 합니다.
- 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스은 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 사용해서). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

AWS IoT Analytics로 내보내기

로 내보내는 스트림을 생성하기 위해 AWS IoT Analytics Greengrass 구성요소는 하나 이상의 `IoTAnalyticsConfig` 객체를 포함하는 내보내기 정의가 있는 스트림을 생성합니다. 이 객체는 대상 채널, 배치 크기, 배치 간격, 우선 순위와 같은 내보내기 설정을 정의합니다.

Greengrass 구성 요소는 장치로부터 데이터를 수신할 때 데이터 덩어리가 포함된 메시지를 대상 스트림에 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 집계하여 데이터 웨어하우스나 클러스터에 로드하는 데 사용됩니다. MapReduce 자세한 내용은 Amazon Kinesis 개발자 안내서의 Amazon Kinesis Data Streams이란 무엇입니까?를 참조하세요.

Stream Manager SDK에서 Greengrass 구성요소는 를 사용하여 `KinesisConfig` 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- KinesisConfigPython SDK에서
- KinesisConfig자바 SDK에서
- KinesisConfigNode.js SDK에서

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- Kinesis Data Streams의 대상 스트림은 Greengrass AWS 계정 코어 AWS 리전 디바이스와 동일해야 합니다.
- 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스은 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 사용해서). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Kinesis Data Streams로 내보내기

Kinesis Data Streams로 내보내는 스트림을 생성하기 위해 Greengrass 구성 요소는 하나 이상의 객체를 포함하는 내보내기 정의가 있는 스트림을 생성합니다. `KinesisConfig` 이 객체는 대상 데이터 스트림, 배치 크기, 배치 간격, 우선 순위와 같은 내보내기 설정을 정의합니다.

Greengrass 구성 요소는 장치로부터 데이터를 수신할 때 데이터 덩어리가 포함된 메시지를 대상 스트림에 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.

Stream Manager SDK에서 Greengrass 구성요소는 를 사용하여 `IoTSiteWiseConfig` 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python SiteWiseConfig SDK에서의 IoT
- 자바 SiteWiseConfig SDK에서의 IoT
- Node.js SiteWiseConfig SDK의 IoT

> **Note**
>
> AWS또한 OPC-UA 소스에서 데이터를 스트리밍하는 데 사용할 수 있는 사전 구축된 솔루션을 제공하는 AWS IoT SiteWise 구성 요소도 제공합니다. 자세한 설명은 IoT SiteWise OPC-UA 컬렉터 섹션을 참조하세요.

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 의 대상 자산 속성은 Greengrass 코어 AWS 계정 AWS 리전 디바이스와 AWS IoT SiteWise 동일해야 합니다.

  > **Note**
  >
  > AWS IoT SiteWise지원하는 서버 목록은 일반 AWS 리전 참조의 AWS IoT SiteWise엔드포인트 및 할당량을 참조하십시오. AWS

- 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스은 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 사용 설명서의 BatchPutAssetPropertyValue 권한 부여를 AWS IoT SiteWise 참조하십시오.

AWS IoT SiteWise로 내보내기

로 내보내는 스트림을 생성하기 위해 AWS IoT SiteWise Greengrass 구성요소는 하나 이상의 `IoTSiteWiseConfig` 객체를 포함하는 내보내기 정의가 있는 스트림을 생성합니다. 이 객체는 배치 크기, 배치 간격 및 우선 순위와 같은 내보내기 설정을 정의합니다.

Greengrass 구성 요소가 장치로부터 자산 속성 데이터를 수신하면 데이터가 포함된 메시지를 대상 스트림에 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 내용을 알아보려면 AWS IoT SiteWise API 참조의 BatchPutAssetPropertyValue 섹션을 참조하십시오.

그런 다음 스트림 관리자는 스트림의 내보내기 구성에 정의된 배치 설정 및 우선 순위에 따라 데이터를 내보냅니다.

스트림 관리자 설정과 Greengrass 구성 요소 로직을 조정하여 내보내기 전략을 설계할 수 있습니다. 예:

- 거의 실시간으로 내보내려면 배치 크기 및 간격을 낮게 설정하고 스트림이 수신되면 스트림에 데이터를 추가하십시오.
- 일괄 처리를 최적화하고, 대역폭 제약을 완화하거나, 비용을 최소화하기 위해 Greengrass 구성 요소는 스트림에 데이터를 추가하기 전에 단일 자산 자산에 대해 수신된 timestamp-quality-value (TQV) 데이터 포인트를 폴링할 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.

Stream Manager SDK에서 Greengrass 구성요소는 를 사용하여 `S3ExportTaskExecutorConfig` 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하십시오.

- Python ExportTaskExecutorConfig SDK의 S3
- 자바 ExportTaskExecutorConfigSDK의 S3
- Node.js ExportTaskExecutorConfigSDK의 S3

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 대상 Amazon S3 버킷은 Greengrass 코어 AWS 계정 디바이스와 동일해야 합니다.
- Greengrass 컨테이너 모드에서 실행되는 Lambda 함수가 입력 파일을 입력 파일 디렉터리에 쓰는 경우, 디렉터리를 쓰기 권한이 있는 컨테이너에 볼륨으로 마운트해야 합니다. 이렇게 하면 파일이 루트 파일 시스템에 기록되고 컨테이너 외부에서 실행되는 스트림 관리자 구성 요소에서 볼 수 있습니다.
- Docker 컨테이너 구성 요소가 입력 파일 디렉터리에 입력 파일을 쓰는 경우 쓰기 권한이 있는 컨테이너의 볼륨으로 디렉터리를 마운트해야 합니다. 이렇게 하면 파일이 루트 파일 시스템에 기록되고 컨테이너 외부에서 실행되는 스트림 관리자 구성 요소에서 볼 수 있습니다.

- 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스은 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 사용해서). 자세한 내용은 IAM 사용 설명서의 IAM 정책 추가 및 제거를 참조하세요.

Amazon S3로 내보내기

Amazon S3로 내보내는 스트림을 생성하기 위해 Greengrass 구성 요소는 `S3ExportTaskExecutorConfig` 객체를 사용하여 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 Greengrass 구성요소는 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.

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}` 자리 표시자에 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 설명은 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 Greengrass 구성요소를 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.
2. Greengrass 구성 요소는 디렉토리를 스캔하고 새 파일이 생성되면 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. Greengrass 구성요소는 상태 스트림에서 메시지를 읽어 내보내기 상태를 모니터링합니다. 내보내기 작업이 완료된 후 Greengrass 구성 요소는 해당 입력 파일을 삭제할 수 있습니다. 자세한 설명은 the section called “내보내기 작업 모니터링” 섹션을 참조하세요.

내보내기 작업 모니터링

IoT 애플리케이션이 Amazon S3 내보내기 상태를 모니터링하는 데 사용하는 코드를 작성할 수 있습니다. Greengrass 구성 요소는 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. 이 설정은 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 워크플로는 Greengrass 구성요소가 상태 스트림을 사용하여 내보내기 상태를 모니터링하는 방법을 보여줍니다.

1. 이전 워크플로에서 설명한 대로 Greengrass 구성요소는 내보내기 작업에 대한 상태 메시지를 상태 스트림에 기록하도록 구성된 스트림에 내보내기 작업을 추가합니다. 추가 작업은 작업 ID를 나타내는 시퀀스 번호를 반환합니다.
2. Greengrass 구성 요소는 상태 스트림에서 메시지를 순차적으로 읽은 다음 스트림 이름과 작업 ID 또는 메시지 컨텍스트의 내보내기 작업 속성을 기반으로 메시지를 필터링합니다. 예를 들어 Greengrass 구성 요소는 메시지 컨텍스트의 `S3ExportTaskDefinition` 개체로 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 time-to-live (TTL) 기간이 만료되어 작업이 중지되었습니다.

> **Note**
>
> 작업 상태가 `InProgress` 또는 `Warning` 상태를 가질 수도 있습니다. 이벤트가 작업 실행에 영향을 주지 않는 오류를 반환하면 스트림 관리자에서 경고를 발행합니다. 예를 들어 부분 업로드를 정리하지 못하면 경고가 반환됩니다.

3. 내보내기 작업이 완료된 후 Greengrass 구성 요소는 해당 입력 파일을 삭제할 수 있습니다.

다음 예제는 Greengrass 구성 요소가 상태 메시지를 읽고 처리하는 방법을 보여줍니다.

Python

```
import time
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from 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: 메시지 읽기 | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
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 =
 StreamManagerClientFactory.standard().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.
}
```

자바 SDK 레퍼런스: 읽기메시지 | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require(*'aws-greengrass-stream-manager-sdk'*);

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: 읽기메시지 | StatusMessage

AWS IoT Greengrass 스트림 관리자 구성

Greengrass 코어 디바이스에서 스트림 관리자는 IoT 디바이스 데이터를 저장, 처리 및 내보낼 수 있습니다. 스트림 관리자는 런타임 설정을 구성하는 데 사용하는 매개변수를 제공합니다. 이 설정은 Greengrass 코어 디바이스의 모든 스트림에 적용됩니다. 구성 요소를 배포할 때 AWS IoT Greengrass 콘솔 또는 API를 사용하여 스트림 관리자 설정을 구성할 수 있습니다. 변경 사항은 배포가 완료된 후에 적용됩니다.

스트림 관리자 파라미터

Stream Manager는 구성 요소를 핵심 장치에 배포할 때 구성할 수 있는 다음과 같은 매개 변수를 제공합니다. 모든 파라미터는 선택 사항입니다.

스토리지 디렉터리

파라미터 이름: `STREAM_MANAGER_STORE_ROOT_DIR`

스트림을 저장하는 데 사용되는 로컬 폴더의 절대 경로입니다. 이 값은 슬래시로 시작해야 합니다 (예: `/data`).

기존 폴더를 지정해야 하며, 스트림 관리자 구성 요소를 실행하는 시스템 사용자에게 이 폴더에 대한 읽기 및 쓰기 권한이 있어야 합니다. 예를 들어 다음 명령을 실행하여 폴더를 만들고 구성할 수 있으며`/var/greengrass/streams`, 이 폴더를 스트림 관리자 루트 폴더로 지정할 수 있습니다. 이 명령을 사용하면 기본 시스템 사용자 () 가 이 폴더를 읽고 쓸 수 있습니다. `ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

스트림 데이터 보안에 대한 자세한 내용은 the section called “로컬 데이터 보안” 섹션을 참조하십시오.

기본값: *`/greengrass/v2`*`/work/aws.greengrass.StreamManager`

[Server port]

파라미터 이름: `STREAM_MANAGER_SERVER_PORT`

스트림 관리자와 통신하는 데 사용되는 로컬 포트 번호입니다. 기본값은 `8088`입니다.

사용 가능한 임의 포트를 `0` 사용하도록 지정할 수 있습니다.

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 Stream Manager SDK에 의해 제어됩니다. 이 매개변수는 스트림 작업을 위해 Stream Manager SDK를 호출할 수 있는 클라이언트를 결정합니다. 자세한 설명은 the section called “클라이언트 인증” 섹션을 참조하세요.

유효한 값은 `true` 또는 `false`입니다. 기본값은 `true`입니다(권장).

- `true`. Greengrass 구성 요소만 클라이언트로 허용합니다. 구성 요소는 내부 AWS IoT Greengrass 코어 프로토콜을 사용하여 Stream Manager SDK로 인증합니다.
- `false`. AWS IoT GreengrassCore에서 실행되는 모든 프로세스를 클라이언트가 되도록 허용합니다. 비즈니스 케이스에서 요구하는 `false` 경우가 아니면 값을 로 설정하지 마십시오. 예를 들어 코어 장치의 구성 요소가 아닌 프로세스가 스트림 관리자와 직접 통신해야 하는 `false` 경우에만 사용하십시오.

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.

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 제약이 있는 기기에서는 내보내기 스트림 수를 제한하세요.

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.

로깅 수준

파라미터 이름: `LOG_LEVEL`

구성 요소의 로깅 수준. 여기에 레벨 순서대로 나열된 다음 로그 수준 중에서 선택하십시오.

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

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 Greengrass 구성 요소는 Amazon S3로 내보내는 스트림을 생성할 때 이 임계값을 설정합니다. 기본 임계값 크기는 5MB입니다.

다음 사항도 참조하십시오.

- Greengrass 코어 디바이스의 데이터 스트림 관리
- 스트림 StreamManagerClient 작업에 사용
- 지원되는 AWS 클라우드 대상의 구성 내보내기

기계 학습 추론 수행

를 사용하면 클라우드 학습 모델을 사용하여 로컬에서 생성된 데이터에 대해 엣지 디바이스에서 기계 학습 (ML) 추론을 수행할 수 있습니다. AWS IoT Greengrass 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.

AWS IoT Greengrass추론을 수행하는 데 필요한 단계를 더 효율적으로 만듭니다. 어디서나 추론 모델을 학습시키고 로컬에 기계 학습 구성 요소로 배포할 수 있습니다. 예를 들어 Amazon에서 딥 러닝 모델을 구축하고 SageMaker 학습시키거나 Amazon Lookout for Vision에서 컴퓨터 비전 모델을 구축하고 학습시킬 수 있습니다. 그런 다음 이러한 모델을 Amazon S3 버킷에 저장하여 이러한 모델을 구성 요소의 아티팩트로 사용하여 코어 디바이스에서 추론을 수행할 수 있습니다.

주제

- AWS IoT Greengrass ML 추론 작동 방식
- AWS IoT Greengrass버전 2에서는 무엇이 다릅니까?
- 요구 사항
- 지원되는 모델 소스
- 지원되는 기계 학습 런타임
- AWS-머신 러닝 컴포넌트 제공
- 그린그래스 코어 디바이스에서 Amazon SageMaker Edge Manager 사용
- Amazon Lookout for Vision
- 머신 러닝 구성 요소 사용자 지정
- 머신 러닝 추론 문제 해결

AWS IoT Greengrass ML 추론 작동 방식

AWS디바이스에서 기계 학습 추론을 수행하기 위한 1단계 배포를 생성하는 데 사용할 수 있는 기계 학습 구성 요소를 제공합니다. 또한 이러한 구성 요소를 템플릿으로 사용하여 특정 요구 사항에 맞는 사용자 지정 구성 요소를 만들 수 있습니다.

AWS다음과 같은 범주의 기계 학습 구성 요소를 제공합니다.

- 모델 구성 요소 —기계 학습 모델을 Greengrass 아티팩트로 포함합니다.

- 런타임 구성 요소 - Greengrass 코어 장치에 기계 학습 프레임워크 및 해당 종속성을 설치하는 스크립트가 들어 있습니다.
- 추론 구성 요소 - 추론 코드를 포함하며 기계 학습 프레임워크를 설치하고 사전 학습된 기계 학습 모델을 다운로드하기 위한 구성 요소 종속성을 포함합니다.

기계 학습 추론을 수행하기 위해 생성하는 각 배포는 추론 응용 프로그램을 실행하고, 기계 학습 프레임워크를 설치하고, 기계 학습 모델을 다운로드하는 하나 이상의 구성 요소로 구성됩니다. AWS제공된 구성 요소를 사용하여 샘플 추론을 수행하려면 핵심 장치에 추론 구성 요소를 배포합니다. 그러면 해당 모델 및 런타임 구성 요소가 종속 항목으로 자동 포함됩니다. 배포를 사용자 지정하려면 샘플 모델 구성 요소를 사용자 지정 모델 구성 요소로 연결하거나 교체하거나 AWS 제공된 구성 요소의 구성 요소 레시피를 템플릿으로 사용하여 사용자 지정 추론, 모델 및 런타임 구성 요소를 만들 수 있습니다.

사용자 지정 구성 요소를 사용하여 기계 학습 추론을 수행하려면:

1. 모델 구성 요소 만들기. 이 구성 요소에는 추론을 수행하는 데 사용할 기계 학습 모델이 포함되어 있습니다. AWS사전 학습된 DLR 및 TensorFlow Lite 모델 샘플을 제공합니다. 사용자 지정 모델을 사용하려면 자체 모델 구성 요소를 만드십시오.
2. 런타임 컴포넌트 생성. 이 구성 요소에는 모델의 기계 학습 런타임을 설치하는 데 필요한 스크립트가 포함되어 있습니다. AWSDLR (딥 러닝 런타임) 및 TensorFlow Lite용 샘플 런타임 구성 요소를 제공합니다. 사용자 지정 모델 및 추론 코드와 함께 다른 런타임을 사용하려면 자체 런타임 구성 요소를 만드십시오.
3. 추론 컴포넌트 만들기. 이 구성 요소에는 추론 코드가 포함되며 모델 및 런타임 구성 요소가 종속성으로 포함됩니다. AWSDLR 및 Lite를 사용한 이미지 분류 및 객체 감지를 위한 샘플 추론 구성 요소를 제공합니다. TensorFlow 다른 유형의 추론을 수행하거나 사용자 지정 모델 및 런타임을 사용하려면 자체 추론 구성 요소를 만드십시오.
4. 추론 구성 요소를 배포하십시오. 이 구성 요소를 배포하면 모델 및 런타임 구성 요소 AWS IoT Greengrass 종속성도 자동으로 배포됩니다.

AWS제공된 구성 요소를 시작하려면 을 참조하십시오. the section called “샘플 이미지 분류 추론 수행”

사용자 지정 기계 학습 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오머신 러닝 구성 요소 사용자 지정.

AWS IoT Greengrass버전 2에서는 무엇이 다릅니까?

AWS IoT Greengrass모델, 런타임, 추론 코드와 같은 기계 학습용 기능 단위를 구성 요소로 통합하여 한 단계 프로세스를 통해 기계 학습 런타임을 설치하고, 학습된 모델을 다운로드하고, 장치에서 추론을 수행할 수 있습니다.

AWS제공된 기계 학습 구성 요소를 사용하면 샘플 추론 코드와 사전 학습된 모델을 사용하여 유연하게 기계 학습 추론을 시작할 수 있습니다. 사용자 지정 모델 구성 요소를 플러그인하여 제공되는 추론 및 런타임 구성 요소와 함께 자체 사용자 지정 학습 모델을 사용할 수 있습니다. AWS 완전히 사용자 지정된 기계 학습 솔루션의 경우 공용 구성 요소를 템플릿으로 사용하여 사용자 지정 구성 요소를 만들고 원하는 런타임, 모델 또는 추론 유형을 사용할 수 있습니다.

요구 사항

기계 학습 구성 요소를 만들고 사용하려면 다음이 있어야 합니다.

- 그린그래스 코어 디바이스. 계정이 없는 경우 자습서: AWS IoT Greengrass V2 시작하기 단원을 참조하십시오.
- AWS제공된 샘플 머신 러닝 구성 요소를 사용하기 위한 최소 500MB의 로컬 스토리지 공간.

지원되는 모델 소스

AWS IoT GreengrassAmazon S3에 저장된 사용자 지정 학습 기계 학습 모델 사용을 지원합니다. Amazon SageMaker Edge 패키징 작업을 사용하여 SageMaker NEO로 컴파일된 모델의 모델 구성 요소를 직접 생성할 수도 있습니다. SageMaker Edge Manager와 함께 AWS IoT Greengrass 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 그린그래스 코어 디바이스에서 Amazon SageMaker Edge Manager 사용 또한 Amazon Lookout for Vision 모델 패키징 작업을 사용하여 Lookout for Vision 모델용 모델 구성 요소를 생성할 수 있습니다. Lookout for AWS IoT Greengrass Vision과 함께 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. Amazon Lookout for Vision

모델이 포함된 S3 버킷은 다음 요구 사항을 충족해야 합니다.

- SSE-C를 사용하여 암호화해서는 안 됩니다. 서버 측 암호화를 사용하는 버킷의 경우, AWS IoT Greengrass 기계 학습 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.

- 이름에는 마침표 () 가 포함되지 않아야 합니다. . 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 이름 지정 규칙에서 SSL과 함께 가상 호스팅 스타일 버킷을 사용하는 것과 관련된 규칙을 참조하십시오.
- 모델 소스를 저장하는 S3 버킷은 기계 학습 구성 AWS 계정 요소와 AWS 리전 동일해야 합니다.
- AWS IoT Greengrass모델 소스에 대한 `read` 권한이 있어야 합니다. S3 AWS IoT Greengrass 버킷에 액세스할 수 있으려면 Greengrass 디바이스 역할이 작업을 `s3:GetObject` 허용해야 합니다. 디바이스 역할에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

지원되는 기계 학습 런타임

AWS IoT Greengrass원하는 기계 학습 런타임을 사용하여 사용자 지정 학습된 모델을 사용하여 기계 학습 추론을 수행하는 사용자 지정 구성 요소를 만들 수 있습니다. 사용자 지정 기계 학습 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오. 머신 러닝 구성 요소 사용자 지정

기계 학습을 더 효율적으로 시작할 수 있도록 다음 기계 학습 런타임을 사용하는 샘플 추론, 모델 및 런타임 구성 요소를 AWS IoT Greengrass 제공합니다.

- 딥 러닝 런타임 (DLR) v1.6.0 및 v1.3.0
- TensorFlow 라이트 v2.5.0

AWS-머신 러닝 컴포넌트 제공

다음 표에는 기계 학습에 사용되는 AWS -제공 구성 요소가 나열되어 있습니다.

> **Note**
>
> AWS제공되는 몇 가지 구성 요소는 Greengrass 핵의 특정 마이너 버전에 따라 다릅니다. 이러한 종속성 때문에 Greengrass 핵을 새 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 의존하는 핵의 특정 버전에 대한 자세한 내용은 해당 구성 요소 항목을 참조하십시오. 핵 업데이트에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass코어 소프트웨어 (OTA) 업데이트

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
Lookout for Vision Edge Agent를 찾아보세요	Greengrass 코어 디바이스에 Amazon Lookout for Vision 런타임을 배포하므로 컴퓨터 비전을 사용하여 산업용 제품의 결함을 찾을 수 있습니다.	일반	Linux	아니요
SageMaker 엣지 매니저	Greengrass 코어 디바이스에 Amazon SageMaker Edge Manager 에이전트를 배포합니다.	일반	Linux, Windows	아니요
DLR 이미지 분류	DLR 이미지 분류 모델 스토어와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 기기에서 이미지 분류 추론을 수행하는 추론	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
	구성 요소입니다.			
DLR 객체 감지	DLR 개체 탐지 모델 저장소와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 개체 감지 모델을 다운로드하고, 지원되는 장치에서 개체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 이미지 분류 모델 스토어	샘플 ResNet-50개의 이미지 분류 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 객체 감지 모델 스토어	샘플 YoloV3 객체 감지 모델을 Greengrass 아티팩트로 포함하는 모델 컴포넌트입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
DLR 런타임	Greengrass 코어 디바이스에 DLR 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트가 포함된 런타임 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow 라이트 이미지 분류	TensorFlow Lite 이미지 분류 모델 저장소와 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 기기에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow 라이트 오브젝트 감지	TensorFlow Lite 개체 감지 모델 저장소 및 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 개체 감지 모델을 다운로드하고, 지원되는 장치에서 개체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow 라이트 이미지 분류 모델 스토어	샘플 MobileNet v1 모델을 Greengrass 아티팩트로 포함하는 모델 컴포넌트입니다.	일반	Linux, Windows	아니요
TensorFlow Lite 객체 감지 모델 스토어	샘플 싱글 샷 감지 (SSD) MobileNet 모델을 Greengrass 아티팩트로 포함하는 모델 컴포넌트입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow 라이트 런타임	Greengrass 코어 기기에 TensorFlow Lite 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트가 포함된 런타임 구성 요소입니다.	일반	Linux, Windows	아니요

그린그래스 코어 디바이스에서 Amazon SageMaker Edge Manager 사용

> **Important**
>
> SageMaker 엣지 매니저는 2024년 4월 26일에 단종됩니다. 에지 장치에 모델을 계속 배포하는 방법에 대한 자세한 내용은 SageMaker Edge Manager 수명 종료를 참조하십시오.

Amazon SageMaker Edge Manager는 엣지 디바이스에서 실행되는 소프트웨어 에이전트입니다. SageMaker Edge Manager는 엣지 디바이스에 대한 모델 관리를 제공하므로 Amazon SageMaker NEO로 컴파일된 모델을 Greengrass 코어 디바이스에서 직접 패키징하고 사용할 수 있습니다. SageMaker Edge Manager를 사용하면 코어 디바이스에서 모델 입력 및 출력 데이터를 샘플링하고 모니터링 및 분석을 AWS 클라우드 위해 해당 데이터를 로 전송할 수도 있습니다. SageMaker Edge Manager는 SageMaker Neo를 사용하여 대상 하드웨어에 맞게 모델을 최적화하므로 DLR 런타임을 장치에 직접 설치할 필요가 없습니다. Greengrass 디바이스에서 SageMaker Edge Manager는 로컬 AWS IoT 인증서를 로드하거나 AWS IoT 자격 증명 공급자 엔드포인트를 직접 호출하지 않습니다. 대신 SageMaker Edge Manager는 토큰 교환 서비스를 사용하여 TES 엔드포인트에서 임시 자격 증명을 가져옵니다.

이 섹션에서는 Greengrass 코어 디바이스에서 SageMaker Edge Manager가 작동하는 방식을 설명합니다.

Greengrass 디바이스에서 SageMaker 엣지 매니저가 작동하는 방식

SageMaker Edge Manager 에이전트를 핵심 장치에 배포하려면 `aws.greengrass.SageMakerEdgeManager` 구성 요소가 포함된 배포를 생성하십시오. AWS IoT Greengrass장치의 Edge Manager 에이전트 설치 및 수명 주기를 관리합니다. 새 버전의 에이전트 바이너리를 사용할 수 있게 되면 업데이트된 버전의 `aws.greengrass.SageMakerEdgeManager` 구성 요소를 배포하여 장치에 설치된 에이전트 버전을 업그레이드하십시오.

와 함께 SageMaker Edge Manager를 사용하는 AWS IoT Greengrass 경우 워크플로에는 다음과 같은 상위 단계가 포함됩니다.

1. Neo로 SageMaker 모델을 컴파일하세요.
2. SageMaker Edge 패키징 작업을 사용하여 SageMaker NEO 컴파일된 모델을 패키징하십시오. 모델에 대한 에지 패키징 작업을 실행할 때 패키징된 모델을 Greengrass 코어 장치에 배포할 수 있는 아티팩트로 사용하여 모델 구성 요소를 생성하도록 선택할 수 있습니다.
3. 사용자 지정 추론 구성 요소를 만드세요. 이 추론 구성 요소를 사용하여 Edge Manager 에이전트와 상호 작용하여 코어 장치에서 추론을 수행합니다. 이러한 작업에는 모델 로드, 추론 실행을 위한 예측 요청 호출, 구성 요소 종료 시 모델 언로드가 포함됩니다.
4. SageMaker Edge Manager 구성 요소, 패키지된 모델 구성 요소 및 추론 구성 요소를 배포하여 장치의 SageMaker 추론 엔진 (Edge Manager 에이전트) 에서 모델을 실행하십시오.

Edge Manager와 함께 SageMaker 작동하는 에지 패키징 작업 및 추론 구성 요소를 생성하는 방법에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 모델 패키지 및 Edge Manager 에이전트 배포를 참조하십시오. AWS IoT Greengrass

이 튜토리얼: SageMaker 엣지 매니저 시작하기 자습서에서는 샘플 추론 및 모델 구성 요소를 생성하는 데 사용할 수 있는 AWS -제공 예제 코드를 사용하여 기존 Greengrass 코어 장치에서 SageMaker Edge Manager 에이전트를 설정하고 사용하는 방법을 보여줍니다.

Greengrass 코어 장치에서 SageMaker Edge Manager를 사용하는 경우 데이터 캡처 기능을 사용하여 샘플 데이터를 에 업로드할 수도 있습니다. AWS 클라우드 캡처 데이터는 향후 분석을 위해 추론 입력, 추론 결과 및 추가 추론 데이터를 S3 버킷 또는 로컬 디렉터리에 업로드하는 데 사용하는 SageMaker 기능입니다. SageMaker Edge Manager에서 캡처 데이터를 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 모델 관리를 참조하십시오.

요구 사항

Greengrass 코어 디바이스에서 SageMaker Edge Manager 에이전트를 사용하려면 다음 요구 사항을 충족해야 합니다.

- 아마존 리눅스 2, 데비안 기반 리눅스 플랫폼 (x86_64 또는 Armv8) 또는 윈도우 (x86_64) 에서 실행되는 그린그래스 코어 디바이스. 계정이 없는 경우 자습서: AWS IoT Greengrass V2 시작하기 단원을 참조하십시오.
- Python 3.6 이상 (사용 중인 Python `pip` 버전용 포함) 이 코어 기기에 설치되어 있어야 합니다.
- Greengrass 장치 역할은 다음과 같이 구성되었습니다.
  - 다음 IAM 정책 예제와 같이 역할을 `credentials.iot.amazonaws.com` 허용하고 `sagemaker.amazonaws.com` 역할을 맡을 수 있는 신뢰 관계입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - AmazonSageMakerEdgeDeviceFleetPolicyIAM 관리형 정책.
  - `s3:PutObject`작업은 다음 IAM 정책 예제에 나와 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Greengrass 코어 AWS 계정 디바이스와 AWS 리전 동일한 디바이스에서 생성된 Amazon S3 버킷 SageMaker Edge Manager에는 에지 디바이스 플릿을 생성하고 추론 실행의 샘플 데이터를 디바이스에 저장하기 위한 S3 버킷이 필요합니다. S3 버킷 생성에 대한 자세한 내용은 Amazon S3 시작하기를 참조하십시오.
- Greengrass 코어 디바이스와 동일한 AWS IoT 역할 별칭을 사용하는 SageMaker 에지 디바이스 플릿입니다. 자세한 설명은 에지 디바이스 플릿 생성 섹션을 참조하세요.
- Greengrass 코어 디바이스가 Edge 디바이스 플릿에 에지 SageMaker 디바이스로 등록되었습니다. 에지 디바이스 이름은 코어 디바이스의 AWS IoT 사물 이름과 일치해야 합니다. 자세한 설명은 그린그래스 코어 디바이스 등록 섹션을 참조하세요.

SageMaker Edge Manager로 시작하세요.

자습서를 완료하여 SageMaker Edge Manager 사용을 시작할 수 있습니다. 이 자습서에서는 기존 코어 장치에서 AWS 제공된 샘플 구성 요소와 함께 SageMaker Edge Manager를 사용하여 시작하는 방법을 보여줍니다. 이러한 샘플 구성 요소는 SageMaker Edge Manager 구성 요소를 종속 항목으로 사용하여 Edge Manager 에이전트를 배포하고 Neo를 사용하여 컴파일된 사전 학습된 모델을 사용하여 추론을 수행합니다. SageMaker 자세한 내용은 튜토리얼: SageMaker 엣지 매니저 시작하기을(를) 참조하세요.

Amazon Lookout for Vision

> **Note**
>
> AWS IoT Greengrass현재 Windows 코어 장치에서는 이 기능을 지원하지 않습니다.

아마존 Lookout for Vision 산업용 제품의 시각적 결함을 찾는 데 사용할 수AWS 서비스 있는 도구입니다. 컴퓨터 비전을 사용하여 산업용 제품의 부품 누락, 차량 또는 구조물의 손상, 생산 라인의 불규칙성, 인쇄 회로 기판의 커패시터 누락, 실리콘 웨이퍼 또는 품질이 중요한 기타 물리적 품목의 결함을 식별합니다. 자세한 내용은 Amazon Lookout for Vision 아마존 Lookout for Vision 개발자 가이드에서 확인할 수 있습니다.

Lookout for Vision 추론을 사용하여 Greengrass 코어 장치에서 시각적 결함을 찾아내는 Greengrass 애플리케이션을 만들 수 있습니다. Lookout for Vision 워크플로를 Greengrass 코어 장치에 배포한 후에는 에서 Lookout for Vision 서비스에 연결하지 않고도 컴퓨터 비전을 수행할 수AWS 클라우드 있습니다. Lookout for Vision Vision을 사용하는 Greengrass 애플리케이션을 만들려면 다음과 같은 Greengrass 구성 요소를 설정하고 배포해야 합니다.

- 비전 모델 구성 요소 찾기 — Lookout for Vision 머신 러닝 모델을 Greengrass 아티팩트로 포함합니다. Lookout for Vision 콘솔 및 API를 사용하여 사전 학습된 기계 학습 모델을 패키징하는 모델 구성 요소를 생성할 수 있습니다. 이러한 구성 요소는 사용자의 비공개 Greengrass 구성 요소입니다AWS 계정. 자세한 내용은 Amazon Lookout for Vision 개발자 안내서의 비전을 위한 룩아웃 모델 생성 및 비전 룩아웃 모델 패키징을 참조하십시오.
- Lookout for Vision Edge Agent 구성 요소 - 컴퓨터 비전을 사용하여 사용자가 제공하는 머신 러닝 모델을 사용하여 이상 현상을 탐지하는 로컬 Lookout for Vision 런타임 서버를 제공합니다. 이 구성 요소는AWS -제공 구성 요소입니다. 자세한 내용은 Vision Edge 에이전트 룩아웃 구성 요소를 참조하십시오.
- Lookout for Vision 클라이언트 애플리케이션 구성 요소 — Lookout for Vision Edge Agent 구성 요소와 상호 작용하여 이상 징후가 있는지 이미지를 처리합니다. 이미지와 비디오 스트림을 로컬 Lookout for Vision Edge Agent로 전송하고 기계 학습 모델이 감지한 이상 현상을 보고하는 사용자 지정 클라이언트 애플리케이션 구성 요소를 개발할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 개발자 안내서의 클라이언트 애플리케이션 구성 요소 작성 및 Lookout for Vision Edge Agent API 참조를 참조하십시오.

이러한 구성 요소를 생성, 구성 및 사용하는 방법에 대한 자세한 내용은 Amazon Lookout for Vision 개발자 안내서의 엣지 디바이스에서 Lookout for Vision 모델 사용을 참조하십시오.

머신 러닝 구성 요소 사용자 지정

에서는 샘플 기계 학습 구성 요소를 구성하여 추론AWS IoT Greengrass, 모델 및 런타임 구성 요소를 구성 요소로 사용하여 장치에서 기계 학습 추론을 수행하는 방법을 사용자 지정할 수 있습니다. AWS IoT Greengrass또한 샘플 구성 요소를 템플릿으로 사용하고 필요에 따라 사용자 지정 구성 요소를 만

들 수 있는 유연성을 제공합니다. 이 모듈식 접근 방식을 조합하여 다음과 같은 방식으로 기계 학습 추론 구성 요소를 사용자 지정할 수 있습니다.

샘플 추론 구성 요소 사용

- 추론 구성 요소를 배포할 때 구성을 수정하십시오.
- 샘플 모델 스토어 구성 요소를 사용자 지정 모델 구성 요소로 대체하여 샘플 추론 구성 요소와 함께 사용자 지정 모델을 사용하십시오. 사용자 지정 모델은 샘플 모델과 동일한 런타임을 사용하여 학습해야 합니다.

사용자 지정 추론 구성 요소 사용

- 공개 모델 구성 요소 및 런타임 구성 요소를 사용자 지정 추론 구성 요소의 종속 항목으로 추가하여 샘플 모델 및 런타임에 사용자 지정 추론 코드를 사용할 수 있습니다.
- 사용자 지정 모델 구성 요소 또는 런타임 구성 요소를 만들고 사용자 지정 추론 구성 요소의 종속 항목으로 추가합니다. 사용자 지정 추론 코드를 사용하거나 샘플 구성 요소를 제공하지 AWS IoT Greengrass 않는 런타임을 사용하려면 사용자 지정 구성 요소를 사용해야 합니다.

주제

공개 추론 구성 요소의 구성 수정

AWS IoT Greengrass콘솔의 구성 요소 페이지에는 해당 구성 요소의 기본 구성이 표시됩니다. 예를 들어, TensorFlow Lite 이미지 분류 구성 요소의 기본 구성은 다음과 같습니다.

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
```

```
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
}
```

공개 추론 구성 요소를 배포할 때 기본 구성을 수정하여 배포를 사용자 지정할 수 있습니다. 각 공용 추론 구성 요소에 사용할 수 있는 구성 매개 변수에 대한 자세한 내용은 의 구성 요소 항목을 참조하십시오. AWS-머신 러닝 컴포넌트 제공

이 섹션에서는 AWS IoT Greengrass 콘솔에서 수정된 구성 요소를 배포하는 방법을 설명합니다. 를 사용하여 구성 요소를 배포하는 방법에 대한 자세한 내용은 을 AWS CLI 참조하십시오배포 만들기.

수정된 공개 추론 구성 요소를 배포하려면 (콘솔)

1. AWS IoT Greengrass 콘솔에 로그인합니다.
2. 탐색 메뉴에서 구성 요소를 선택합니다.
3. 구성 요소 페이지의 공용 구성 요소 탭에서 배포하려는 구성 요소를 선택합니다.
4. 구성 요소 페이지에서 배포를 선택합니다.
5. 배포에 추가에서 다음 중 하나를 선택합니다.
   a. 이 구성 요소를 대상 디바이스의 기존 배포에 병합하려면 기존 배포에 추가를 선택한 다음 수정하려는 배포를 선택합니다.
   b. 대상 디바이스에서 새 배포를 생성하려면 새 배포 생성을 선택합니다. 디바이스에 기존 배포가 있는 경우 이 단계를 선택하면 기존 배포가 대체됩니다.
6. 대상 지정 페이지에서 다음 작업을 수행합니다.
   a. 배포 정보 아래에서 친숙한 배포 이름을 입력하거나 수정합니다.
   b. 배포 대상 아래에서 배포 대상을 선택하고 다음을 선택합니다. 기존 배포 수정 시 배포 대상을 변경할 수 없습니다.
7. 구성 요소 선택 페이지의 공용 구성 요소에서 구성이 수정된 추론 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.

8. 구성 요소 구성 페이지에서 다음을 수행하십시오.

   a. 추론 구성 요소를 선택하고 구성 요소 구성을 선택합니다.

   b. 구성 업데이트에서 업데이트하려는 구성 값을 입력합니다. 예를 들어 병합할 구성 상자에 다음 구성 업데이트를 입력하여 추론 간격을 15초로 변경하고 구성 요소가 `custom.jpg` 폴더에서 이름이 지정된 이미지를 찾도록 지시합니다. `/custom-ml-inference/images/`

   ```
   {
     "InferenceInterval": "15",
     "ImageName": "custom.jpg",
     "ImageDirectory": "/custom-ml-inference/images/"
   }
   ```

   구성 요소의 전체 구성을 기본값으로 재설정하려면 경로 재설정 상자에 빈 문자열 "" 하나를 지정하십시오.

   c. 확인을 선택하고 다음을 선택합니다.

9. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.

10. 검토 페이지에서 [Deploy] 를 선택합니다.

샘플 추론 구성 요소가 포함된 사용자 지정 모델을 사용하십시오.

샘플 런타임 구성 요소를 AWS IoT Greengrass 제공하는 런타임용 자체 기계 학습 모델과 함께 샘플 추론 구성 요소를 사용하려면 해당 모델을 아티팩트로 사용하는 구성 요소로 공개 모델 구성 요소를 재정의해야 합니다. 상위 수준에서 다음 단계를 완료하여 샘플 추론 구성 요소와 함께 사용자 지정 모델을 사용하십시오.

1. S3 버킷의 사용자 지정 모델을 아티팩트로 사용하는 모델 구성 요소를 생성합니다. 교체하려는 모델과 동일한 런타임을 사용하여 사용자 지정 모델을 학습해야 합니다.
2. 사용자 지정 모델을 사용하도록 추론 구성 요소의 구성 파라미터를 수정하십시오. `ModelResourceKey` 추론 구성 요소의 구성 업데이트에 대한 자세한 내용은 을 참조하십시오. 공개 추론 구성 요소의 구성 수정

추론 구성 요소를 배포할 때 구성 요소 종속성의 최신 버전을 AWS IoT Greengrass 찾습니다. 구성 요소의 최신 사용자 지정 버전이 동일한 및 에 있는 경우 종속 공개 모델 구성 요소를 재정의합니다. AWS 계정 AWS 리전

사용자 지정 모델 구성 요소 생성 (콘솔)

1. 모델을 S3 버킷에 업로드합니다. 모델을 S3 버킷으로 업로드하는 방법에 대한 자세한 내용은 Amazon 심플 스토리지 서비스 사용 설명서의 Amazon S3 버킷 사용을 참조하십시오.

   > **Note**
   >
   > 아티팩트는 구성 AWS 계정 요소와 AWS 리전 동일한 S3 버킷에 저장해야 합니다. 이러한 아티팩트에 액세스할 수 AWS IoT Greengrass 있으려면 Greengrass 장치 역할이 작업을 `s3:GetObject` 허용해야 합니다. 장치 역할에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

2. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
3. 퍼블릭 모델 스토어 컴포넌트의 컴포넌트 레시피를 검색합니다.

   a. 구성 요소 페이지의 공개 구성 요소 탭에서 새 버전을 만들려는 공개 모델 구성 요소를 찾아 선택합니다. 예: `variant.DLR.ImageClassification.ModelStore`.

   b. 구성 요소 페이지에서 레시피 보기를 선택하고 표시된 JSON 레시피를 복사합니다.
4. 구성 요소 페이지의 내 구성 요소 탭에서 구성 요소 생성을 선택합니다.
5. 구성 요소 생성 페이지의 구성 요소 정보에서 구성 요소 소스로 레시피를 JSON으로 입력을 선택합니다.
6. 레시피 상자에 이전에 복사한 구성 요소 레시피를 붙여넣습니다.
7. 레시피에서 다음 값을 업데이트하십시오.

   - `ComponentVersion`: 컴포넌트의 마이너 버전을 증가시킵니다.

     공개 모델 구성 요소를 재정의하는 사용자 지정 구성 요소를 만들 때는 기존 구성 요소 버전의 부 버전만 업데이트해야 합니다. 예를 들어, 공개 구성 요소 버전이 인 `2.1.0` 경우 버전을 사용하여 사용자 지정 구성 요소를 만들 수 있습니다. `2.1.1`
   - `Manifests.Artifacts.Uri`: 각 URI 값을 사용하려는 모델의 Amazon S3 URI로 업데이트합니다.

   > **Note**
   >
   > 구성 요소 이름을 변경하지 마십시오.

8. 구성 요소 생성을 선택합니다.

사용자 지정 모델 구성 요소 만들기 (AWS CLI)

1. 모델을 S3 버킷에 업로드합니다. 모델을 S3 버킷으로 업로드하는 방법에 대한 자세한 내용은 Amazon 심플 스토리지 서비스 사용 설명서의 Amazon S3 버킷 사용을 참조하십시오.

   > **Note**
   >
   > 아티팩트는 구성 AWS 계정 요소와 AWS 리전 동일한 S3 버킷에 저장해야 합니다. 이러한 아티팩트에 액세스할 수 AWS IoT Greengrass 있으려면 Greengrass 장치 역할이 작업을 `s3:GetObject` 허용해야 합니다. 장치 역할에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

2. 다음 명령을 실행하여 공용 구성 요소의 구성 요소 레시피를 검색합니다. 이 명령은 명령에서 제공하는 출력 파일에 구성 요소 레시피를 기록합니다. 검색된 base64로 인코딩된 문자열을 필요에 따라 JSON 또는 YAML로 변환합니다.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn <arn> \
       --recipe-output-format <recipe-format> \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn <arn> ^
       --recipe-output-format <recipe-format> ^
       --query recipe ^
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

   PowerShell

   ```
   aws greengrassv2 get-component `
       --arn <arn> `
       --recipe-output-format <recipe-format> `
       --query recipe `
   ```

```
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

3. 레시피 파일 이름을 로 업데이트합니다. 여기서 구성 요소 버전은 새 구성 요소의 대상 버전입니다. *<component-name>*-*<component-version>* 예: `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`.
4. 레시피에서 다음 값을 업데이트하십시오.

   - `ComponentVersion`: 컴포넌트의 마이너 버전을 증가시킵니다.

     공개 모델 구성 요소를 재정의하는 사용자 지정 구성 요소를 만들 때는 기존 구성 요소 버전의 부 버전만 업데이트해야 합니다. 예를 들어, 공개 구성 요소 버전이 인 `2.1.0` 경우 버전을 사용하여 사용자 지정 구성 요소를 만들 수 있습니다. `2.1.1`

   - `Manifests.Artifacts.Uri`: 각 URI 값을 사용하려는 모델의 Amazon S3 URI로 업데이트합니다.

   > **Note**
   >
   > 구성 요소 이름을 변경하지 마십시오.

5. 다음 명령을 실행하여 검색하고 수정한 레시피를 사용하여 새 구성 요소를 생성합니다.

   ```
   aws greengrassv2 create-component-version \
       --inline-recipe fileb://path/to/component/recipe
   ```

   > **Note**
   >
   > 이 단계는 AWS IoT Greengrass 서비스의 구성 요소를 생성합니다. AWS 클라우드 Greengrass CLI를 사용하여 구성 요소를 클라우드에 업로드하기 전에 로컬에서 개발, 테스트 및 배포할 수 있습니다. 자세한 설명은 AWS IoT Greengrass구성 요소 개발 섹션을 참조하세요.

구성 요소 생성에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass구성 요소 개발

사용자 지정 기계 학습 구성 요소 만들기

사용자 지정 추론 코드 또는 샘플 구성 요소를 제공하지 AWS IoT Greengrass 않는 런타임을 사용하려면 사용자 지정 구성 요소를 만들어야 합니다. AWS제공된 샘플 기계 학습 모델 및 런타임과 함께 사용자 지정 추론 코드를 사용하거나 자체 모델 및 런타임으로 완전히 사용자 지정된 기계 학습 추론 솔루션을 개발할 수 있습니다. 모델에서 샘플 런타임 구성 요소를 AWS IoT Greengrass 제공하는 런타임을 사용하는 경우 해당 런타임 구성 요소를 사용할 수 있으며, 추론 코드와 사용하려는 모델에 대한 사용자 지정 구성 요소만 만들면 됩니다.

주제

- 퍼블릭 컴포넌트의 레시피를 검색하세요.
- 샘플 구성 요소 아티팩트를 검색하십시오.
- 구성 요소 아티팩트를 S3 버킷에 업로드합니다.
- 사용자 지정 구성 요소 만들기

퍼블릭 컴포넌트의 레시피를 검색하세요.

기존 공개 기계 학습 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 구성 요소를 만들 수 있습니다. 최신 버전의 공용 구성 요소에 대한 구성 요소 레시피를 보려면 콘솔을 사용하거나 다음과 AWS CLI 같이 사용하십시오.

- 콘솔 사용

  1. 구성 요소 페이지의 공용 구성 요소 탭에서 공용 구성 요소를 찾아 선택합니다.
  2. 구성 요소 페이지에서 레시피 보기를 선택합니다.

- AWS CLI 사용

  다음 명령을 실행하여 공개 변형 구성 요소의 구성 요소 레시피를 검색합니다. 이 명령은 명령에서 제공하는 JSON 또는 YAML 레시피 파일에 구성 요소 레시피를 기록합니다.

  Linux, macOS, or Unix

  ```
  aws greengrassv2 get-component \
      --arn <arn> \
      --recipe-output-format <recipe-format> \
      --query recipe \
      --output text | base64 --decode > <recipe-file>
  ```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn <arn> ^
    --recipe-output-format <recipe-format> ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

명령의 값을 다음과 같이 바꿉니다.

- *<arn>*. 퍼블릭 구성 요소의 Amazon 리소스 이름 (ARN).
- *<recipe-format>*. 레시피 파일을 생성하려는 형식. 지원되는 값은 JSON 및 YAML입니다.
- *<recipe-file>*. 해당 형식의 레시피 이름*<component-name>*-*<component-version>*.

샘플 구성 요소 아티팩트를 검색하십시오.

공용 기계 학습 구성 요소에서 사용하는 아티팩트를 템플릿으로 사용하여 추론 코드나 런타임 설치 스크립트와 같은 사용자 지정 구성 요소 아티팩트를 만들 수 있습니다.

공개 기계 학습 구성 요소에 포함된 샘플 아티팩트를 보려면 공용 추론 구성 요소를 배포한 다음 장치의 폴더에서 아티팩트를 확인하십시오. */greengrass/v2*/packages/artifacts-unarchived/*component-name*/*component-version*/

구성 요소 아티팩트를 S3 버킷에 업로드합니다.

사용자 지정 구성 요소를 생성하려면 먼저 구성 요소 아티팩트를 S3 버킷에 업로드하고 구성 요소 레시피에 S3 URI를 사용해야 합니다. 예를 들어, 추론 구성 요소에서 사용자 지정 추론 코드를 사용하려

면 코드를 S3 버킷에 업로드하십시오. 그런 다음 추론 코드의 Amazon S3 URI를 구성 요소의 아티팩트로 사용할 수 있습니다.

S3 버킷으로 콘텐츠를 업로드하는 방법에 대한 자세한 내용은 Amazon 심플 스토리지 서비스 사용 설명서의 Amazon S3 버킷 사용을 참조하십시오.

> **Note**
>
> 아티팩트는 구성 AWS 계정 요소와 AWS 리전 동일한 S3 버킷에 저장해야 합니다. 이러한 아티팩트에 액세스할 수 AWS IoT Greengrass 있으려면 Greengrass 장치 역할이 작업을 `s3:GetObject` 허용해야 합니다. 장치 역할에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

사용자 지정 구성 요소 만들기

검색한 아티팩트와 레시피를 사용하여 사용자 지정 기계 학습 구성 요소를 만들 수 있습니다. 예시는 사용자 지정 추론 구성 요소 만들기 단원을 참조하세요.

구성 요소를 만들고 Greengrass 장치에 배포하는 방법에 대한 자세한 내용은 및 을 참조하십시오. AWS IoT Greengrass구성 요소 개발 디바이스에 AWS IoT Greengrass 구성 요소 배포

사용자 지정 추론 구성 요소 만들기

이 섹션에서는 DLR 이미지 분류 구성 요소를 템플릿으로 사용하여 사용자 지정 추론 구성 요소를 만드는 방법을 보여줍니다.

주제

- Amazon S3 버킷에 추론 코드를 업로드합니다.
- 추론 구성 요소의 레시피를 만드세요.
- 추론 구성 요소 만들기

Amazon S3 버킷에 추론 코드를 업로드합니다.

추론 코드를 생성한 다음 S3 버킷에 업로드합니다. S3 버킷으로 콘텐츠를 업로드하는 방법에 대한 자세한 내용은 Amazon 심플 스토리지 서비스 사용 설명서의 Amazon S3 버킷 사용을 참조하십시오.

> **Note**
>
> 아티팩트는 구성 AWS 계정 요소와 AWS 리전 동일한 S3 버킷에 저장해야 합니다. 이러한 아티팩트에 액세스할 수 AWS IoT Greengrass 있으려면 Greengrass 장치 역할이 작업을 `s3:GetObject` 허용해야 합니다. 장치 역할에 대한 자세한 내용은 을 참조하십시오. 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

추론 구성 요소의 레시피를 만드세요.

1. 다음 명령을 실행하여 DLR 이미지 분류 구성 요소의 구성 요소 레시피를 검색합니다. 이 명령은 명령에서 제공하는 JSON 또는 YAML 레시피 파일에 구성 요소 레시피를 기록합니다.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    \
       --recipe-output-format JSON | YAML \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    ^
       --recipe-output-format JSON | YAML ^
       --query recipe ^
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

   PowerShell

   ```
   aws greengrassv2 get-component `
   ```

```
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 `
    --recipe-output-format JSON | YAML `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

<recipe-file>형식의 레시피 이름으로 바꾸십시오. *<component-name>-<component-version>*

2. 레시피의 ComponentDependencies 객체에서 사용하려는 모델 및 런타임 구성 요소에 따라 다음 중 하나 이상을 수행하십시오.

   - DLR로 컴파일된 모델을 사용하려면 DLR 구성 요소 종속성을 유지하세요. 다음 예제와 같이 사용자 지정 런타임 구성 요소에 대한 종속 항목으로 대체할 수도 있습니다.

     런타임 구성 요소

     JSON

     ```
     {
         "<runtime-component>": {
             "VersionRequirement": "<version>",
             "DependencyType": "HARD"
         }
     }
     ```

     YAML

     ```
     <runtime-component>:
         VersionRequirement: "<version>"
         DependencyType: HARD
     ```

   - DLR 이미지 분류 모델 스토어 종속성을 유지하여 AWS 제공하는 사전 학습된 ResNet -50 모델을 사용하거나 사용자 지정 모델 구성 요소를 사용하도록 수정하십시오. 공개 모델 구성 요소에 대한 종속성을 포함하는 경우 구성 요소의 이후 사용자 지정 버전이 동일한 AWS 계정 버전에 존재하면 추론 구성 요소는 해당 사용자 지정 구성 요소를 사용합니다. AWS 리전 다음 예제와 같이 모델 구성 요소 종속성을 지정합니다.

공개 모델 구성 요소

JSON

```
{
    "variant.DLR.ImageClassification.ModelStore": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
variant.DLR.ImageClassification.ModelStore:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

커스텀 모델 컴포넌트

JSON

```
{
    "<custom-model-component>": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. `ComponentConfiguration`개체에 이 구성 요소의 기본 구성을 추가합니다. 나중에 구성 요소를 배포할 때 이 구성을 수정할 수 있습니다. 다음 발췌문은 DLR 이미지 분류 구성 요소의 구성 요소 구성을 보여줍니다.

   예를 들어 사용자 지정 모델 구성 요소를 사용자 지정 추론 구성 요소의 종속 항목으로 사용하는 경우 사용 중인 모델의 이름을 `ModelResourceKey` 제공하도록 수정하십시오.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
    aws.greengrass.ipc.mqttproxy:
        'aws.greengrass.ImageClassification:mqttproxy:1':
            policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
            operations:
                - 'aws.greengrass#PublishToIoTCore'
            resources:
                - ml/dlr/image-classification
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
```

```
    armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
    x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
    aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. Manifests개체에 구성 요소를 다른 플랫폼에 배포할 때 사용되는 이 구성 요소의 구성 및 아티팩트에 대한 정보와 구성 요소를 성공적으로 실행하는 데 필요한 기타 정보를 제공하십시오. 다음 발췌문은 DLR 이미지 분류 구성 요소의 Linux 플랫폼용 Manifests 개체 구성을 보여줍니다.

   JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
          "DLR_IC_MODEL_DIR":
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "run": {
          "RequiresPrivilege": true,
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

YAML

```
Manifests:
  - Platform:
      os: linux
      architecture: arm
    Name: 32-bit armv7l - Linux (raspberry pi)
    Artifacts:
      - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip
        Unarchive: ZIP
    Lifecycle:
      Setenv:
        DLR_IC_MODEL_DIR:
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}"
        DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/
image_classification/sample_images/"
      run:
        RequiresPrivilege: true
        script: |-
          . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/
activate
          python3 {artifacts:decompressedPath}/image_classification/inference.py
```

구성 요소 레시피 만들기에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass컴포넌트 레시피 참조

추론 구성 요소 만들기

AWS IoT Greengrass콘솔이나 AWS CLI 를 사용하여 방금 정의한 레시피를 사용하여 구성 요소를 생성합니다. 구성 요소를 만든 후 이를 배포하여 디바이스에서 추론을 수행할 수 있습니다. 추론 구성 요소를 배포하는 방법에 대한 예는 을 참조하십시오. 자습서: Lite를 사용하여 TensorFlow 샘플 이미지 분류 추론 수행

사용자 지정 추론 구성 요소 생성 (콘솔)

1. AWS IoT Greengrass 콘솔에 로그인합니다.
2. 탐색 메뉴에서 구성 요소를 선택합니다.
3. 구성 요소 페이지의 내 구성 요소 탭에서 구성 요소 생성을 선택합니다.

4. 구성 요소 생성 페이지의 구성 요소 정보에서 레시피를 JSON으로 입력 또는 레시피를 구성 요소 소스로 YAML로 입력을 선택합니다.
5. 레시피 상자에 생성한 사용자 지정 레시피를 입력합니다.
6. 구성 요소 생성을 클릭합니다.

사용자 지정 추론 구성 요소 만들기 () AWS CLI

다음 명령을 실행하여 생성한 레시피를 사용하여 새 사용자 지정 구성 요소를 생성합니다.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/recipe/file
```

> **Note**
>
> 이 단계는 AWS IoT Greengrass 서비스의 구성 요소를 생성합니다AWS 클라우드. Greengrass CLI를 사용하여 구성 요소를 클라우드에 업로드하기 전에 로컬에서 개발, 테스트 및 배포할 수 있습니다. 자세한 내용은 AWS IoT Greengrass구성 요소 개발을(를) 참조하세요.

머신 러닝 추론 문제 해결

이 섹션의 문제 해결 정보와 솔루션을 사용하면 기계 학습 구성 요소 관련 문제를 해결하는 데 도움이 됩니다. 공개 기계 학습 추론 구성 요소의 경우 다음 구성 요소 로그의 오류 메시지를 참조하십시오.

Linux or Unix

- */greengrass/v2*/logs/aws.greengrass.DLRImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.DLRObjectDetection.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteObjectDetection.log

Windows

- *C:\greengrass\v2*\logs\aws.greengrass.DLRImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.DLRObjectDetection.log

- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteObjectDetection.log

구성 요소가 올바르게 설치된 경우 구성 요소 로그에는 추론에 사용하는 라이브러리의 위치가 포함됩니다.

문제

라이브러리를 가져오지 못했습니다.

Raspberry Pi 디바이스에 배포하는 동안 설치 프로그램 스크립트가 필수 라이브러리를 다운로드하지 못하면 다음 오류가 발생합니다.

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
 3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

구성 요소를 `sudo apt-get update` 다시 실행하고 배포하십시오.

Cannot open shared object file

Raspberry Pi 기기에 배포하는 `opencv-python` 동안 설치 프로그램 스크립트가 필수 종속성을 다운로드하지 못하면 다음과 비슷한 오류가 발생할 수 있습니다.

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

다음 명령을 실행하여 에 대한 종속성을 수동으로 설치합니다. opencv-python

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

ML 런타임 라이브러리 또는 해당 종속성이 제대로 설치되지 않은 경우 ML 런타임 구성 요소 로그 (`variant.DLR.log`또는`variant.TensorFlowLite.log`) 에서 이 오류가 표시될 수 있습니다. 이 오류는 다음과 같은 경우에 발생할 수 있습니다.

- 기본적으로 활성화되는 `UseInstaller` 옵션을 사용하는 경우 이 오류는 ML 런타임 구성 요소가 런타임 또는 해당 종속 항목을 설치하지 못했음을 나타냅니다. 다음을 따릅니다.
  1. `UseInstaller`옵션을 비활성화하도록 ML 런타임 구성 요소를 구성합니다.
  2. ML 런타임과 해당 종속 항목을 설치하고 ML 구성 요소를 실행하는 시스템 사용자가 사용할 수 있도록 합니다. 자세한 내용은 다음 자료를 참조하세요.
     - DLR 런타임 옵션 UseInstaller
     - TensorFlow라이트 런타임 옵션 UseInstaller
- 이 `UseInstaller` 옵션을 사용하지 않는 경우 이 오류는 ML 구성 요소를 실행하는 시스템 사용자를 위해 ML 런타임 또는 해당 종속성이 설치되지 않았음을 나타냅니다. 다음을 따릅니다.
  1. ML 구성 요소를 실행하는 시스템 사용자용으로 라이브러리가 설치되어 있는지 확인하세요. *`ggc_user# ### ###`* 이름으로 바꾸고 *`tflite_runtime#`* 확인할 라이브러리 이름으로 바꾸십시오.

     Linux or Unix

     ```
     sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
     ```

     Windows

     ```
     runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
     ```
  2. 라이브러리가 설치되지 않은 경우 해당 사용자를 위해 설치하세요. *`ggc_user# ### ###`* 이름으로 바꾸고 *`tflite_runtime#`* 라이브러리 이름으로 바꾸십시오.

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
```

Windows

```
runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
```

각 ML 런타임의 종속성에 대한 자세한 내용은 다음을 참조하십시오.

- DLR 런타임 옵션 UseInstaller
- TensorFlow라이트 런타임 옵션 UseInstaller

3. 문제가 지속되면 다른 사용자를 위해 라이브러리를 설치하여 이 장치가 라이브러리를 설치할 수 있는지 확인하세요. 사용자는 예를 들어 사용자, 루트 사용자 또는 관리자 사용자일 수 있습니다. 모든 사용자를 위해 라이브러리를 성공적으로 설치할 수 없는 경우 단말기가 라이브러리를 지원하지 않을 수 있습니다. 라이브러리 설명서를 참조하여 요구 사항을 검토하고 설치 문제를 해결하십시오.

CUDA 지원 장치가 검색되지 않습니다.

GPU 가속을 사용할 때 다음 오류가 표시될 수 있습니다. 다음 명령을 실행하여 Greengrass 사용자의 GPU 액세스를 활성화합니다.

```
sudo usermod -a -G video ggc_user
```

해당 파일이나 디렉터리가 없습니다.

다음 오류는 런타임 구성 요소가 가상 환경을 올바르게 설정하지 못했음을 나타냅니다.

- *MLRootPath*/greengrass_ml_dlr_conda/bin/conda: No such file or directory
- *MLRootPath*/greengrass_ml_dlr_venv/bin/activate: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_conda/bin/conda: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_venv/bin/activate: No such file or directory

로그를 확인하여 모든 런타임 종속성이 올바르게 설치되었는지 확인하십시오. 설치 스크립트로 설치한 라이브러리에 대한 자세한 내용은 다음 항목을 참조하십시오.

- DLR 런타임
- TensorFlow 라이트 런타임

기본적으로 *RootPathML#* 로 설정됩니다. */greengrass/v2*/work/*component-name*/greengrass_ml 이 위치를 변경하려면 배포에 DLR 런타임 또는 TensorFlow 라이트 런타임 런타임 구성 요소를 직접 포함하고 구성 병합 업데이트에서 MLRootPath 매개변수의 수정된 값을 지정하십시오. 구성 요소 구성에 대한 자세한 내용은 을 참조하십시오구성 요소 구성 업데이트.

> **Note**
>
> DLR 구성 요소 v1.3.x의 경우 추론 구성 요소의 구성에서 MLRootPath 매개 변수를 설정하며 기본값은 입니다. $HOME/greengrass_ml

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi에서 기계 학습 추론을 실행할 때 다음 오류가 표시될 수 있습니다.

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

이 오류는 Raspberry Pi OS Bullseye에 OpenCV에 필요한 NumPy 버전보다 이전 버전이 포함되어 있기 때문에 발생합니다. 이 문제를 해결하려면 다음 명령을 실행하여 최신 버전으로 업그레이드하십시오. NumPy

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

Raspberry Pi OS Bullseye를 실행하는 라즈베리 파이에서 기계 학습 추론을 실행할 때 다음 오류가 표시될 수 있습니다.

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
 ensure that the camera has been enabled.
```

이 오류는 Raspberry Pi OS Bullseye에 ML 구성 요소와 호환되지 않는 새 카메라 스택이 포함되어 있기 때문에 발생합니다. 이 문제를 해결하려면 레거시 카메라 스택을 활성화하세요.

레거시 카메라 스택을 활성화하려면

1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

   ```
   sudo raspi-config
   ```

2. 인터페이스 옵션을 선택합니다.
3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
4. Raspberry Pi를 재부팅합니다.

메모리 오류

다음 오류는 일반적으로 장치에 메모리가 충분하지 않아 구성 요소 프로세스가 중단될 때 발생합니다.

- `stderr. Killed.`
- `exitCode=137`

공개 기계 학습 추론 구성 요소를 배포하려면 최소 500MB의 메모리를 사용하는 것이 좋습니다.

디스크 공간 오류

이 `no space left on device` 오류는 일반적으로 디바이스의 저장 공간이 충분하지 않을 때 발생합니다. 구성 요소를 다시 배포하기 전에 디바이스에 사용 가능한 디스크 공간이 충분한지 확인하십시오. 공용 기계 학습 추론 구성 요소를 배포하려면 최소 500MB의 디스크 여유 공간을 확보하는 것이 좋습니다.

제한 시간 오류

공용 기계 학습 구성 요소는 200MB보다 큰 대용량 기계 학습 모델 파일을 다운로드합니다. 배포 중에 다운로드 시간이 초과되면 인터넷 연결 속도를 확인하고 배포를 다시 시도하세요.

다음을 사용하여 Greengrass 코어 장치를 관리하십시오. AWS Systems Manager

> **Note**
>
> AWS IoT Greengrass현재 Windows 코어 장치에서는 이 기능을 지원하지 않습니다.

Systems Manager는 Amazon EC2 인스턴스AWS, 온프레미스 서버 및 VM (가상 머신), 엣지 디바이스를 비롯한 인프라를 보고 제어하는 데 사용할 수 있는 AWS 서비스입니다. Systems Manager를 사용하면 운영 데이터를 보고, 운영 작업을 자동화하고, 보안 및 규정 준수를 유지할 수 있습니다. Systems Manager에 시스템을 등록하면 이를 관리 노드라고 합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의(AWS Systems Manager이)란 무엇입니까? 섹션을 참조하십시오.

AWS Systems Manager에이전트 (Systems Manager Agent) 는 장치에 설치하여 Systems Manager가 장치를 업데이트, 관리 및 구성할 수 있도록 하는 소프트웨어입니다. Greengrass 코어 디바이스에 Systems Manager 에이전트를 설치하려면 Systems Manager Agent 구성 요소를 배포하십시오. Systems Manager 에이전트를 처음 배포하면 코어 디바이스가 Systems Manager 관리 노드로 등록됩니다. Systems Manager 에이전트는 장치에서 실행되어 의 Systems Manager 서비스와 통신할 수 있도록 AWS 클라우드 합니다. Systems Manager Agent 구성 요소를 설치하고 구성하는 방법에 대한 자세한 내용은 을 참조하십시오AWS Systems Manager 에이전트 설치.

Systems Manager 도구 및 기능을 기능이라고 합니다. 그린그래스 코어 디바이스는 모든 Systems Manager 기능을 지원합니다. 이러한 기능 및 Systems Manager를 사용하여 핵심 장치를 관리하는 방법에 대한 자세한 내용은 사용 AWS Systems Manager설명서의 Systems Manager 기능을 참조하십시오.

AWS Systems Manager는 Systems Manager 관리형 노드를 위한 표준 인스턴스 계층과 고급 인스턴스 등급을 제공합니다. Systems Manager를 처음 사용하는 경우 표준 인스턴스 등급부터 시작합니다. 표준 인스턴스 등급에서는 관리형 노드를 1개당 최대 1,000개까지 등록할 수 있습니다. AWS 리전 AWS 계정 단일 계정 및 지역에 1,000개가 넘는 관리형 노드를 등록해야 하거나 세션 관리자 기능을 사용해야 하는 경우에는 고급 인스턴스 등급을 사용하십시오. 자세한 내용은 사용 설명서의 인스턴스 계층 구성을 참조하십시오. AWS Systems Manager

주제

- AWS Systems Manager 에이전트 설치

- AWS Systems Manager 에이전트 제거

AWS Systems Manager 에이전트 설치

AWS Systems Manager에이전트 (Systems Manager Agent) 는 Systems Manager가 Greengrass 코어 디바이스, Amazon EC2 인스턴스 및 기타 리소스를 업데이트, 관리 및 구성할 수 있도록 설치되는 Amazon 소프트웨어입니다. 에이전트는 에서 Systems Manager 서비스의 요청을 처리하고 실행합니다AWS 클라우드. 그런 다음 에이전트는 상태 및 런타임 정보를 Systems Manager 서비스에 다시 보냅니다. 자세한 내용은 AWS Systems Manager사용 설명서의 Systems Manager 에이전트 정보를 참조하십시오.

AWS는 Systems Manager 에이전트를 Greengrass 구성 요소로 제공하며, 이를 Greengrass 코어 디바이스에 배포하여 Systems Manager를 통해 관리할 수 있습니다. Systems Manager Agent 구성 요소는 Systems Manager Agent 소프트웨어를 설치하고 코어 디바이스를 Systems Manager의 관리 노드로 등록합니다. 이 페이지의 단계에 따라 사전 요구 사항을 완료하고 Systems Manager Agent 구성 요소를 코어 장치 또는 핵심 장치 그룹에 배포하십시오.

주제

- 1단계: 일반 Systems Manager 설정 단계 완료
- 2단계: Systems Manager를 위한 IAM 서비스 역할 생성
- 3단계: 토큰 교환 역할에 권한 추가
- 4단계: Systems Manager 에이전트 구성 요소 배포
- 5단계: Systems Manager를 사용하여 코어 디바이스 등록 확인

1단계: 일반 Systems Manager 설정 단계 완료

아직 완료하지 않은 경우 의 일반 설정 단계를 완료하십시오. AWS Systems Manager 자세한 내용은 AWS Systems Manager사용 설명서의 전체 일반 Systems Manager 설치 단계를 참조하십시오.

2단계: Systems Manager를 위한 IAM 서비스 역할 생성

Systems Manager 에이전트는 AWS Identity and Access Management (IAM) 서비스 역할을 사용하여 AWS Systems Manager 통신합니다. Systems Manager는 이 역할을 맡아 각 코어 디바이스에서 Systems Manager 기능을 활성화합니다. 또한 Systems Manager Agent 구성 요소는 구성 요소를 배포할 때 이 역할을 사용하여 코어 장치를 Systems Manager 관리 노드로 등록합니다. 아직 생성하지 않았다면 Systems Manager Agent 구성 요소가 사용할 Systems Manager 서비스 역할을 생성하십시오.

자세한 내용은 사용 설명서의 에지 디바이스용 IAM 서비스 역할 생성을 참조하십시오. AWS Systems Manager

3단계: 토큰 교환 역할에 권한 추가

Greengrass 코어 디바이스는 토큰 교환 역할이라고 하는 IAM 서비스 역할을 사용하여 서비스와 상호 작용합니다. AWS 각 코어 디바이스에는 AWS IoT Greengrass코어 소프트웨어를 설치할 때 생성하는 토큰 교환 역할이 있습니다. Systems Manager 에이전트와 같은 많은 Greengrass 구성 요소에는 이 역할에 대한 추가 권한이 필요합니다. Systems Manager 에이전트 구성 요소에는 다음과 같은 권한이 필요하며, 여기에는 에서 생성한 역할을 사용할 수 있는 권한이 포함됩니다2단계: Systems Manager를 위한 IAM 서비스 역할 생성.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

아직 추가하지 않았다면 코어 디바이스의 토큰 교환 역할에 이러한 권한을 추가하여 Systems Manager Agent가 작동할 수 있도록 하십시오. 토큰 교환 역할에 새 정책을 추가하여 이 권한을 부여할 수 있습니다.

토큰 교환 역할에 권한을 추가하려면 (콘솔)

1. IAM 콘솔 탐색 메뉴에서 [Roles] 를 선택합니다.

2. AWS IoT GreengrassCore 소프트웨어를 설치할 때 토큰 교환 역할로 설정한 IAM 역할을 선택합니다. AWS IoT GreengrassCore 소프트웨어를 설치할 때 토큰 교환 역할의 이름을 지정하지 않은 경우 이름이 지정된 `GreengrassV2TokenExchangeRole` 역할이 생성됩니다.
3. 권한에서 권한 추가를 선택한 다음 정책 연결을 선택합니다.
4. 정책 생성을 선택합니다. 새 브라우저 탭에 정책 생성 페이지가 열립니다.
5. [Create policy(정책 생성)] 페이지에서 다음을 수행합니다.

   a. JSON을 선택하여 JSON 편집기를 엽니다.

   b. 다음 정책을 JSON 편집기에 붙여넣습니다. *SSM#* 에서 생성한 서비스 역할의 ServiceRole 이름으로 바꾸십시오. 2단계: Systems Manager를 위한 IAM 서비스 역할 생성

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

   c. 다음: 태그를 선택합니다.

   d. 다음: 검토를 선택합니다.

   e. 정책 이름을 입력합니다(예: **`GreengrassSSMAgentComponentPolicy`**).

   f. 정책 생성을 선택합니다.

   g. 토큰 교환 역할이 열려 있는 이전 브라우저 탭으로 전환하십시오.

6. 권한 추가 페이지에서 새로 고침 버튼을 선택한 다음 이전 단계에서 생성한 Greengrass Systems Manager 에이전트 정책을 선택합니다.
7. 정책 연결을 선택합니다.

   이 토큰 교환 역할을 사용하는 핵심 장치는 이제 Systems Manager 서비스와 상호 작용할 수 있는 권한을 갖게 되었습니다.

토큰 교환 역할에 권한을 추가하려면 (AWS CLI)

Systems Manager를 사용할 수 있는 권한을 부여하는 정책을 추가하려면

1. 라는 `ssm-agent-component-policy.json` 파일을 만들고 다음 JSON을 파일에 복사합니다. *SSM#* 에서 생성한 서비스 역할의 ServiceRole 이름으로 바꾸십시오. 2단계: Systems Manager를 위한 IAM 서비스 역할 생성

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

2. 다음 명령을 실행하여 의 정책 문서에서 `ssm-agent-component-policy.json` 정책을 생성합니다.

Linux or Unix

```
aws iam create-policy \
  --policy-name GreengrassSSMAgentComponentPolicy \
  --policy-document file://ssm-agent-component-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name GreengrassSSMAgentComponentPolicy ^
  --policy-document file://ssm-agent-component-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name GreengrassSSMAgentComponentPolicy `
  --policy-document file://ssm-agent-component-policy.json
```

출력의 정책 메타데이터에서 Amazon 리소스 이름 (ARN) 정책을 복사합니다. 다음 단계에서 이 ARN을 사용하여 이 정책을 핵심 장치 역할에 연결합니다.

3. 다음 명령을 실행하여 정책을 토큰 교환 역할에 연결합니다.

   - *GreenGrassV2#* Core 소프트웨어를 설치할 때 지정한 토큰 교환 역할의 이름으로 TokenExchangeRole 바꾸십시오. AWS IoT Greengrass AWS IoT GreengrassCore 소프트웨어를 설치할 때 토큰 교환 역할의 이름을 지정하지 않은 경우 이름이 지정된 역할이 생성되었습니다. `GreengrassV2TokenExchangeRole`
   - 정책 ARN을 이전 단계의 ARN으로 교체합니다.

   Linux or Unix

   ```
   aws iam attach-role-policy \
     --role-name GreengrassV2TokenExchangeRole \
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
```

명령 출력이 없으면 성공한 것입니다. 이 토큰 교환 역할을 사용하는 핵심 장치는 이제 Systems Manager 서비스와 상호 작용할 수 있는 권한을 갖게 되었습니다.

4단계: Systems Manager 에이전트 구성 요소 배포

Systems Manager 에이전트 구성 요소를 배포하고 구성하려면 다음 단계를 완료하십시오. 구성 요소를 단일 코어 장치 또는 핵심 장치 그룹에 배포할 수 있습니다.

Systems Manager 에이전트 구성 요소를 배포하려면 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지에서 공용 구성 요소 탭을 선택한 다음 선택합니다 aws.greengrass.SystemsManagerAgent.
3. aws.greengrass.SystemsManagerAgent 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포를 만들도록 선택한 후 다음을 선택합니다.
5. 새 배포를 생성하기로 선택한 경우 배포할 대상 코어 장치 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 장치 또는 사물 그룹을 선택하고 다음을 선택합니다.
6. 구성 요소 선택 페이지에서 구성 aws.greengrass.SystemsManagerAgent요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 구성 요소를 선택하고 aws.greengrass.SystemsManagerAgent다음을 수행합니다.

a. 구성 요소 구성을 선택합니다.

b. 구성 aws.greengrass.SystemsManagerAgent 모달의 구성 업데이트에 있는 병합할 구성에 다음 구성 업데이트를 입력합니다. *SSM#* 에서 생성한 서비스 역할의 ServiceRole 이름으로 바꾸십시오. 2단계: Systems Manager를 위한 IAM 서비스 역할 생성

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false
}
```

> **Note**
>
> 코어 디바이스에서 하이브리드 활성화로 등록된 Systems Manager Agent를 이미 실행하고 있는 경우 `SSMOverrideExistingRegistration` 로 `true` 변경하십시오. 이 매개 변수는 Systems Manager Agent가 하이브리드 활성화가 적용된 장치에서 이미 실행 중인 경우 Systems Manager Agent 구성 요소가 코어 장치를 등록할지 여부를 지정합니다.
>
> Systems Manager Agent 구성 요소가 코어 장치용으로 생성하는 Systems Manager 관리 노드에 추가할 태그 (`SSMResourceTags`) 를 지정할 수도 있습니다. 자세한 내용은 Systems Manager 에이전트 구성 요소 구성을 참조하십시오.

c. [확인] 을 선택하여 모달을 닫고 [다음] 을 선택합니다.

8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.

9. 검토 페이지에서 배포를 선택합니다.

   배포를 완료하는 데 최대 1분이 걸릴 수 있습니다.

Systems Manager 에이전트 구성 요소를 배포하려면 (AWS CLI)

Systems Manager Agent 구성 요소를 배포하려면 `components` 개체에 포함된 `aws.greengrass.SystemsManagerAgent` 배포 문서를 만들고 구성 요소에 대한 구성 업데이트를 지정합니다. 의 지침에 따라 새 배포 만들기 배포를 만들거나 기존 배포를 수정하십시오.

다음 예제 부분 배포 문서는 이름이 지정된 `SSMServiceRole` 서비스 역할을 사용하도록 지정합니다. *SSM#* 에서 2단계: Systems Manager를 위한 IAM 서비스 역할 생성 생성한 서비스 역할 이름으로 ServiceRole 바꾸십시오.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

> **Note**
>
> 코어 디바이스에서 하이브리드 활성화로 등록된 Systems Manager Agent를 이미 실행하고 있는 경우 `SSMOverrideExistingRegistration` 로 `true` 변경하십시오. 이 매개 변수는 Systems Manager Agent가 하이브리드 활성화가 적용된 장치에서 이미 실행 중인 경우 Systems Manager Agent 구성 요소가 코어 장치를 등록할지 여부를 지정합니다.
> Systems Manager Agent 구성 요소가 코어 장치용으로 생성하는 Systems Manager 관리 노드에 추가할 태그 (`SSMResourceTags`) 를 지정할 수도 있습니다. 자세한 내용은 Systems Manager 에이전트 구성 요소 구성을 참조하십시오.

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS IoT Greengrass서비스를 사용하여 배포 상태를 확인하고, AWS IoT Greengrass 핵심 소프트웨어 로그와 Systems Manager Agent 구성 요소 로그를 확인하여 Systems Manager Agent가 성공적으로 실행되는지 확인할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 배포 상태를 확인합니다 의 상태를 확인하세요 의
- 모니터 AWS IoT Greengrass 로그
- AWS Systems Manager사용 설명서에서 Systems Manager 에이전트 로그 보기

배포가 실패하거나 Systems Manager Agent가 실행되지 않는 경우 각 코어 장치의 배포 문제를 해결할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 문제 해결 AWS IoT Greengrass V2

- AWS Systems Manager사용 설명서의 Systems Manager 에이전트 문제 해결

5단계: Systems Manager를 사용하여 코어 디바이스 등록 확인

Systems Manager Agent 구성 요소가 실행되면 코어 디바이스가 Systems Manager에 관리 노드로 등록됩니다. AWS IoT Greengrass콘솔, Systems Manager 콘솔 및 Systems Manager API를 사용하여 코어 디바이스가 관리 노드로 등록되었는지 확인할 수 있습니다. AWS콘솔과 API의 일부에서는 관리형 노드를 인스턴스라고도 합니다.

코어 디바이스 등록을 확인하려면 (AWS IoT Greengrass콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 코어 디바이스를 선택합니다.
2. 확인할 코어 기기를 선택합니다.
3. 코어 디바이스의 세부 정보 페이지에서 AWS Systems Manager인스턴스 속성을 찾으십시오. 이 속성이 존재하고 Systems Manager 콘솔에 대한 링크를 표시하는 경우 코어 디바이스는 관리 노드로 등록됩니다.

   AWS Systems Manager핑 상태 속성을 찾아 코어 디바이스의 Systems Manager 에이전트 상태를 확인할 수도 있습니다. 상태가 온라인이면 Systems Manager를 사용하여 핵심 장치를 관리할 수 있습니다.

코어 디바이스 등록을 확인하려면 (Systems Manager 콘솔)

1. Systems Manager 콘솔 탐색 메뉴에서 플릿 관리자를 선택합니다.
2. 관리형 노드에서 다음을 수행하십시오.
   a. 소스 유형이 인 경우 필터를 추가합니다 AWS::IoT::Thing.
   b. 소스 ID가 확인할 코어 디바이스의 이름인 필터를 추가합니다.
3. 관리형 노드 테이블에서 코어 디바이스를 찾을 수 있습니다. 코어 기기가 테이블에 있는 경우 관리 노드로 등록됩니다.

   Systems Manager Agent의 핑 상태 속성을 사용하여 코어 디바이스의 Systems Manager 에이전트 상태를 확인할 수도 있습니다. 상태가 온라인이면 Systems Manager를 사용하여 핵심 장치를 관리할 수 있습니다.

코어 디바이스 등록을 확인하려면 (AWS CLI)

- DescribeInstanceInformation작업을 사용하여 지정한 필터와 일치하는 관리형 노드 목록을 가져올 수 있습니다. 다음 명령을 실행하여 코어 디바이스가 관리 노드로 등록되었는지 확인합니다. 확인할 코어 디바이스의 *MyGreengrassCore*이름으로 바꾸십시오.

  ```
  aws ssm describe-instance-information --filter
   Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
  ```

  응답에는 필터와 일치하는 관리형 노드 목록이 포함됩니다. 목록에 관리 노드가 포함된 경우 코어 장치는 관리 노드로 등록됩니다. 응답에서 코어 기기의 관리 노드에 대한 기타 정보도 찾을 수 있습니다. `PingStatus`속성이 `Online` 인 경우 Systems Manager를 사용하여 핵심 장치를 관리할 수 있습니다.

코어 디바이스가 Systems Manager에서 관리 노드로 등록되었는지 확인한 후 Systems Manager 콘솔 및 API를 사용하여 해당 코어 디바이스를 관리할 수 있습니다. Greengrass 코어 장치를 관리하는 데 사용할 수 있는 Systems Manager 기능에 대한 자세한 내용은 사용 AWS Systems Manager설명서의 Systems Manager 기능을 참조하십시오.

AWS Systems Manager 에이전트 제거

Greengrass 코어 디바이스를 더 이상 에서 관리하지 않으려면 Systems Manager에서 코어 디바이스를 등록 취소하고 디바이스에서AWS Systems Manager 에이전트 (Systems Manager Agent) 를 제거하면 됩니다.AWS Systems Manager

언제든지 코어 디바이스를 다시 등록할 수 있습니다. 이렇게 하려면 Systems Manager 에이전트 구성 요소를 다시 배포하여 코어 디바이스를 설치할 때 Systems Manager Manager에 등록합니다. Systems Manager Manager는 등록 취소된 코어 디바이스에 대한 명령 기록을 30일 동안 저장합니다.

주제

- 1단계: Systems Manager 코어 장치 등록 취소
- 2단계: Systems Manager 에이전트 구성 요소 제거
- 3단계: Systems Manager 에이전트 소프트웨어를 제거할 수 있습니다.

1단계: Systems Manager 코어 장치 등록 취소

Systems Manager 콘솔이나 API를 사용하여 코어 디바이스를 등록 취소할 수 있습니다. 자세한 내용은 AWS Systems Manager사용 설명서의 관리 노드 등록 취소를 참조하십시오.

2단계: Systems Manager 에이전트 구성 요소 제거

코어 디바이스를 등록 취소한 후 디바이스에서 Systems Manager Agent 구성 요소를 제거합니다. Greengrass 코어 장치에서 구성 요소를 제거하려면 구성 요소를 설치한 배포를 수정하고 배포에서 구성 요소를 제거하십시오. AWS IoT Greengrass핵심 장치 배포에서 해당 구성 요소를 지정하지 않은 경우 Core 소프트웨어는 구성 요소를 제거합니다. 자세한 정보는 디바이스에 AWS IoT Greengrass 구성 요소 배포을 참조하세요.

Systems Manager 에이전트 구성 요소를 제거하려면 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 핵심 장치를 선택합니다.
2. Systems Manager 에이전트 구성 요소를 제거하려는 코어 디바이스를 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 배포를 선택합니다.
4. Systems Manager 에이전트 구성 요소를 코어 장치에 배포하는 배포를 선택합니다.
5. 배포 세부 정보 페이지에서 개정을 선택합니다.
6. 배포 수정 모달에서 배포 수정을 선택합니다.
7. 1단계: 대상 지정에서 다음을 선택합니다.
8. 2단계: 구성 요소 선택에서 구성 aws.greengrass.SystemsManagerAgent요소 선택을 취소한 후 다음을 선택합니다.
9. 3단계: 구성 요소 구성에서 다음을 선택합니다.
10. 4단계: 고급 설정 구성에서 다음을 선택합니다.
11. 5단계: 검토에서 배포를 선택합니다.

Systems Manager 에이전트 구성 요소 (CLI) 를 제거하려면

Systems Manager Agent 구성 요소를 제거하려면 해당 구성 요소를 배포하는 배포를 수정하고 배포에서 제거하십시오. 자세한 정보는 배포 수정을 참조하세요.

배포를 완료되는 데 몇 분 정도 걸릴 수 있습니다. AWS IoT Greengrass서비스를 사용하여 배포의 상태를 확인할 수 있습니다. 자세한 정보는 배포 상태를 확인합니다 의 상태를 확인하세요 의을 참조하세요.

3단계: Systems Manager 에이전트 소프트웨어를 제거할 수 있습니다.

Systems Manager Agent 구성 요소를 제거한 후에도 Systems Manager Agent 소프트웨어는 코어 장치에서 계속 실행됩니다. Systems Manager Agent 소프트웨어를 제거하려면 코어 장치에서 명령을 실행할 수 있습니다. 자세한 내용은 AWS Systems Manager사용 설명서의 Linux 인스턴스에서 Systems Manager 에이전트 제거를 참조하십시오.

AWS IoT Greengrass의 보안

AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. 여러분은 AWS 고객으로서 보안에 민감한 기관의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와(과) 귀하의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 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 AWS서비스 를 참조하십시오.
- 클라우드의 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 Greengrass을(를) 구성하는 방법을 보여줍니다. 또한 AWS IoT Greengrass 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스 사용 방법을 알아봅니다.

주제

- AWS IoT Greengrass 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)
- AWS IoT Greengrass의 보안 모범 사례

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
- 저장된 데이터 암호화
- Greengrass 코어 디바이스를 위한 키 관리

전송 중 데이터 암호화

AWS IoT Greengrass에서는 두 가지 통신 모드로 데이터를 전송할 수 있습니다.

- the section called “인터넷을 통해 전송 중인 데이터”. Greengrass 코어와 간 통신AWS IoT Greengrass인터넷을 통해 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. 로 전송되는 모든 데이터AWS 클라우드는 MQTT 또는 HTTPS 프로토콜을 사용하여 TLS 연결을 통해 전송되므로 기본적으로 안전합니다.AWS IoT Greengrass를 사용합니다.AWS IoT전송 보안 모델. 자세한 내용은 단원을 참조하십시오.전송 보안의AWS IoT Core개발자 안내서.

코어 디바이스의 데이터

데이터가 디바이스를 떠나지 않기 때문에 AWS IoT Greengrass는 Greengrass 코어 디바이스에서 로컬로 교환되는 데이터를 암호화하지 않습니다. 여기에는 사용자 정의 구성 요소 간의 통신이 포함됩니다.AWS IoT디바이스 SDK 및 공용 구성 요소 (예: 스트림 관리자)

저장된 데이터 암호화

AWS IoT Greengrass가 데이터를 저장합니다.

- the section called “는 저장된 데이터AWS 클라우드”. 이 데이터는 암호화됩니다.
- the section called “Greengrass 코어에 저장된 데이터”. 이 데이터는 암호화되지 않습니다(암호의 로컬 사본 제외).

는 저장된 데이터AWS 클라우드

AWS IoT Greengrass는 다음 위치에 저장된 고객 데이터를 암호화합니다.AWS 클라우드. 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 단원을 참조하십시오.보안 관리자구성 요소.

Greengrass 코어 디바이스를 위한 키 관리

Greengrass 코어 디바이스에 암호화 (퍼블릭 및 프라이빗) 키를 안전하게 저장하는 것은 고객의 책임입니다.AWS IoT Greengrass다음 시나리오에서 공개 키와 개인 키를 사용합니다.

- IoT 클라이언트 키는 IoT 인증서와 함께 사용되어 Greengrass 코어가 AWS IoT Core에 연결될 때 TLS(전송 계층 보안) 핸드셰이크를 인증합니다. 자세한 정보는 the section called “디바이스 인증 및 권한 부여”을 참조하십시오.

> **Note**
>
> 키와 인증서를 코어 프라이빗 키와 코어 디바이스 인증서라고도 합니다.

Greengrass 코어 디바이스는 파일 시스템 권한 또는하드웨어 보안 모듈. 파일 시스템 기반 프라이빗 키를 사용하는 경우, 코어 디바이스의 보안 스토리지에 대한 책임은 사용자에게 있습니다.

하드웨어 보안 통합

> **Note**
>
> 이 기능은 v2.5.3 이상의 Greengrass 핵 구성 요소에서 사용할 수 있습니다. AWS IoT Greengrass현재 Windows 코어 디바이스에서는 이 기능을 지원하지 않습니다.

PKCS #11 인터페이스를 통해 하드웨어 보안 모듈 (HSM) 을 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수 있습니다. 이 기능을 사용하면 장치의 개인 키와 인증서를 안전하게 저장하여 소프트웨어에 노출되거나 복제되지 않도록 할 수 있습니다. 개인 키와 인증서를 HSM 또는 TPM (신뢰할 수 있는 플랫폼 모듈) 과 같은 하드웨어 모듈에 저장할 수 있습니다.

AWS IoT GreengrassCore 소프트웨어는 개인 키와 X.509 인증서를 사용하여 및 서비스에 대한 연결을 인증합니다. AWS IoT AWS IoT Greengrass 보안 관리자 구성 요소는 이 개인 키를 사용하여 Greengrass 코어 장치에 배포한 암호를 안전하게 암호화하고 해독합니다. HSM을 사용하도록 코어 디바이스를 구성하면 이러한 구성 요소는 HSM에 저장한 개인 키와 인증서를 사용합니다.

또한 Mocquette MQTT 브로커 구성 요소는 로컬 MQTT 서버 인증서용 개인 키를 저장합니다. 이 구성 요소는 구성 요소 작업 폴더에 있는 장치 파일 시스템의 개인 키를 저장합니다. 현재는 이 개인 키 또는 인증서를 HSM에 저장하는 것을 AWS IoT Greengrass 지원하지 않습니다.

> **Tip**
>
> AWS파트너 장치 카탈로그에서 이 기능을 지원하는 장치를 검색하십시오.

주제

요구 사항

Greengrass 코어 디바이스에서 HSM을 사용하려면 다음 요구 사항을 충족해야 합니다.

- 그린그래스 뉴클리어스 v2.5.3 이상이 코어 디바이스에 설치되었습니다. AWS IoT Greengrass코어 디바이스에 Core 소프트웨어를 설치할 때 호환되는 버전을 선택할 수 있습니다.
- 코어 기기에 설치된 PKCS #11 공급자 구성 요소. AWS IoT Greengrass코어 장치에 Core 소프트웨어를 설치할 때 이 구성 요소를 다운로드하고 설치할 수 있습니다.
- PKCS #1 v1.5 서명 체계와 RSA-2048 키 크기 (이상) 또는 ECC 키가 있는 RSA 키를 지원하는 하드웨어 보안 모듈입니다.

  > **Note**
  >
  > ECC 키가 있는 하드웨어 보안 모듈을 사용하려면 Greengrass nucleus v2.5.6 이상을 사용해야 합니다.
  > 하드웨어 보안 모듈과 시크릿 관리자를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

- AWS IoT GreengrassCore 소프트웨어가 런타임 시 (libdl 사용) 로드하여 PKCS #11 함수를 호출할 수 있는 PKCS #11 공급자 라이브러리입니다. PKCS #11 공급자 라이브러리는 다음과 같은 PKCS #11 API 작업을 구현해야 합니다.
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`

  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- 하드웨어 모듈은 PKCS#11 사양에 정의된 것처럼 슬롯 레이블로 확인할 수 있어야 합니다.
- HSM에서 개체 ID를 지원하는 경우 개인 키와 인증서를 HSM의 동일한 슬롯에 저장하고 동일한 개체 레이블과 개체 ID를 사용해야 합니다.
- 인증서와 개인 키는 개체 레이블로 확인할 수 있어야 합니다.
- 개인 키에는 다음과 같은 권한이 있어야 합니다.
  - `sign`
  - `decrypt`
- (선택 사항) Secret Manager 구성 요소를 사용하려면 버전 2.1.0 이상을 사용해야 하며, 개인 키에는 다음과 같은 권한이 있어야 합니다.
  - `unwrap`
  - `wrap`

하드웨어 보안 베스트 프랙티스

Greengrass 코어 디바이스에서 하드웨어 보안을 구성할 때는 다음 모범 사례를 고려하십시오.

- 내부 하드웨어 난수 생성기를 사용하여 HSM에서 직접 프라이빗 키를 생성합니다. 이 방법은 개인 키가 HSM 내에 남아 있기 때문에 다른 곳에서 생성한 개인 키를 가져오는 것보다 더 안전합니다.
- 개인 키를 변경할 수 없도록 구성하고 내보내기를 금지하십시오.

- HSM 하드웨어 벤더가 권장하는 프로비저닝 도구를 사용하여 하드웨어 보호 개인 키를 사용하여 CSR (인증서 서명 요청) 을 생성한 다음 콘솔 또는 API를 사용하여 클라이언트 인증서를 생성하십시오. AWS IoT

> **Note**
>
> HSM에서 개인 키를 생성할 때는 키를 교체하는 보안 모범 사례가 적용되지 않습니다.

하드웨어 보안이 적용된 AWS IoT Greengrass Core 소프트웨어 설치

AWS IoT GreengrassCore 소프트웨어를 설치할 때 HSM에서 생성한 개인 키를 사용하도록 소프트웨어를 구성할 수 있습니다. 이 접근 방식은 보안 모범 사례에 따라 HSM에서 개인 키를 생성하여 개인 키가 HSM 내에 유지되도록 합니다.

하드웨어 보안이 적용된 AWS IoT Greengrass Core 소프트웨어를 설치하려면 다음과 같이 하십시오.

1. HSM에서 개인 키를 생성합니다.
2. 개인 키에서 인증서 서명 요청 (CSR) 을 생성합니다.
3. CSR에서 인증서를 생성합니다. 다른 루트 인증 기관 (CA) 에서 서명한 AWS IoT 인증서를 만들거나 다른 루트 인증 기관 (CA) 에서 서명한 인증서를 만들 수 있습니다. 다른 루트 CA를 사용하는 방법에 대한 자세한 내용은 AWS IoT Core개발자 안내서의 자체 클라이언트 인증서 만들기를 참조하십시오.
4. AWS IoT인증서를 다운로드하여 HSM으로 가져오십시오.
5. PKCS #11 제공자 구성 요소와 HSM의 개인 키 및 인증서를 사용하도록 지정하는 구성 파일에서 AWS IoT Greengrass Core 소프트웨어를 설치합니다.

다음 설치 옵션 중 하나를 선택하여 하드웨어 보안이 적용된 AWS IoT Greengrass Core 소프트웨어를 설치할 수 있습니다.

- 수동 설치

  필요한 AWS 리소스를 수동으로 생성하고 하드웨어 보안을 구성하려면 이 옵션을 선택합니다. 자세한 설명은 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.
- 사용자 지정 프로비저닝을 통한 설치

이 옵션을 선택하면 필요한 AWS 리소스를 자동으로 생성하고 하드웨어 보안을 구성하는 사용자 정의 Java 응용 프로그램을 개발할 수 있습니다. 자세한 설명은 사용자 지정 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

현재는 자동 리소스 프로비저닝 또는 AWS IoT 플릿 프로비저닝으로 설치하는 경우 하드웨어 보안이 적용된 AWS IoT Greengrass Core 소프트웨어 설치를 AWS IoT Greengrass 지원하지 않습니다.

기존 코어 기기에 하드웨어 보안을 구성합니다.

코어 디바이스의 개인 키와 인증서를 HSM으로 가져와 하드웨어 보안을 구성할 수 있습니다.

> **고려 사항**
>
> - 코어 디바이스의 파일 시스템에 대한 루트 액세스 권한이 있어야 합니다.
> - 이 절차에서는 AWS IoT Greengrass 코어 소프트웨어를 종료하여 하드웨어 보안을 구성하는 동안 코어 디바이스가 오프라인 상태이고 사용할 수 없도록 합니다.

기존 코어 장치에 하드웨어 보안을 구성하려면 다음과 같이 하십시오.

1. HSM을 초기화합니다.
2. PKCS #11 제공자 구성 요소를 코어 디바이스에 배포합니다.
3. AWS IoT Greengrass코어 소프트웨어를 중지하십시오.
4. 코어 디바이스의 개인 키와 인증서를 HSM으로 가져옵니다.
5. HSM의 개인 키와 인증서를 사용하도록 AWS IoT Greengrass Core 소프트웨어의 구성 파일을 업데이트하십시오.
6. AWS IoT Greengrass코어 소프트웨어를 시작합니다.

1단계: 하드웨어 보안 모듈 초기화

다음 단계를 완료하여 코어 기기에서 HSM을 초기화하십시오.

하드웨어 보안 모듈을 초기화하려면

- HSM에서 PKCS #11 토큰을 초기화하고 토큰용 슬롯 ID와 사용자 PIN을 저장합니다. 토큰 초기화 방법을 알아보려면 HSM 설명서를 확인하세요. 나중에 PKCS #11 제공자 구성 요소를 배포하고 구성할 때 슬롯 ID와 사용자 PIN을 사용합니다.

2단계: PKCS #11 공급자 구성 요소 배포

PKCS #11 공급자 구성 요소를 배포하고 구성하려면 다음 단계를 완료하십시오. 구성 요소를 하나 이상의 코어 장치에 배포할 수 있습니다.

PKCS #11 공급자 구성 요소를 배포하려면 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지에서 공용 구성 요소 탭을 선택한 다음 선택합니다 aws.greengrass.crypto.Pkcs11Provider.
3. aws.greengrass.crypto.Pkcs11Provider 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포를 만들도록 선택한 후 다음을 선택합니다.
5. 새 배포를 생성하기로 선택한 경우 배포할 대상 코어 장치 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 장치 또는 사물 그룹을 선택하고 다음을 선택합니다.
6. 구성 요소 선택 페이지의 공용 구성 요소에서 을 선택하고 다음을 선택합니다 aws.greengrass.crypto.Pkcs11Provider.
7. 구성 요소 구성 페이지에서 구성 요소를 선택하고 aws.greengrass.crypto.Pkcs11Provider다음을 수행합니다.

   a. 구성 요소 구성을 선택합니다.

   b. 구성 aws.greengrass.crypto.Pkcs11Provider 모달의 구성 업데이트에 있는 병합할 구성에 다음 구성 업데이트를 입력합니다. 다음 구성 매개변수를 대상 코어 장치의 값으로 업데이트하십시오. 이전에 PKCS #11 토큰을 초기화한 슬롯 ID와 사용자 PIN을 지정하십시오. 나중에 개인 키와 인증서를 HSM의 이 슬롯으로 가져옵니다.

      `name`

      PKCS #11 구성의 이름.

`library`

AWS IoT Greengrass코어 소프트웨어가 libdl로 로드할 수 있는 PKCS #11 구현 라이브러리의 절대 파일 경로입니다.

`slot`

개인 키와 디바이스 인증서가 포함된 슬롯의 ID입니다. 이 값은 슬롯 인덱스 또는 슬롯 레이블과 다릅니다.

`userPin`

슬롯에 액세스하는 데 사용할 사용자 PIN.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

c. 확인을 선택하여 모달을 닫고 다음을 선택합니다.

8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 배포를 선택합니다.

배포를 완료하는 데 최대 1분이 걸릴 수 있습니다.

PKCS #11 공급자 구성 요소를 배포하려면 () AWS CLI

PKCS #11 공급자 구성 요소를 배포하려면 `components` 개체에 포함하는 `aws.greengrass.crypto.Pkcs11Provider` 배포 문서를 만들고 구성 요소에 대한 구성 업데이트를 지정하십시오. 의 지침에 따라 새 배포 만들기 배포를 만들거나 기존 배포를 수정하십시오.

다음 예제 부분 배포 문서는 PKCS #11 제공자 구성 요소를 배포하고 구성하도록 지정합니다. 다음 구성 매개변수를 대상 코어 장치 값으로 업데이트하십시오. 나중에 개인 키와 인증서를 HSM으로 가져올 때 사용할 수 있도록 슬롯 ID와 사용자 PIN을 저장하십시오.

`name`

PKCS #11 구성의 이름입니다.

`library`

AWS IoT Greengrass코어 소프트웨어가 libdl로 로드할 수 있는 PKCS #11 구현 라이브러리의 절대 파일 경로입니다.

`slot`

개인 키와 디바이스 인증서가 포함된 슬롯의 ID입니다. 이 값은 슬롯 인덱스 또는 슬롯 레이블과 다릅니다.

`userPin`

슬롯에 액세스하는 데 사용할 사용자 PIN.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/
libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
}
```

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS IoT Greengrass서비스를 사용하여 배포 상태를 확인할 수 있습니다. AWS IoT Greengrass코어 소프트웨어 로그를 확인하여 PKCS #11 제공자 구성 요소가 성공적으로 배포되었는지 확인할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 배포 상태를 확인합니다 의 상태를 확인하세요 의
- 모니터 AWS IoT Greengrass 로그

배포가 실패할 경우 각 코어 장치의 배포 문제를 해결할 수 있습니다. 자세한 설명은 문제 해결 AWS IoT Greengrass V2 섹션을 참조하세요.

3단계: 코어 장치의 구성 업데이트

AWS IoT GreengrassCore 소프트웨어는 장치 작동 방식을 지정하는 구성 파일을 사용합니다. 이 구성 파일에는 장치가 연결하는 데 사용하는 개인 키와 인증서를 찾을 수 있는 위치가 포함되어 AWS 클라우드 있습니다. 코어 디바이스의 개인 키와 인증서를 HSM으로 가져오고 HSM을 사용하도록 구성 파일을 업데이트하려면 다음 단계를 완료하십시오.

하드웨어 보안을 사용하도록 코어 디바이스의 컨피그레이션을 업데이트하려면

1. AWS IoT GreengrassCore 소프트웨어를 중지하십시오. systemd를 사용하여 AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 구성한 경우 다음 명령을 실행하여 소프트웨어를 중지할 수 있습니다.

   ```
   sudo systemctl stop greengrass.service
   ```

2. 코어 디바이스의 개인 키와 인증서 파일을 찾으십시오.

   - 자동 프로비저닝 또는 플릿 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치한 경우 개인 키는 에 있고 인증서는 에 있습니다. */greengrass/v2*/privKey.key */greengrass/v2*/thingCert.crt
   - 수동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치한 경우 */greengrass/v2*/private.pem.key 기본적으로 개인 키가 존재하고 인증서가 기본적으로 존재합니다. */greengrass/v2*/device.pem.crt

   system.privateKeyPath및 system.certificateFilePath 속성을 확인하여 이러한 파일의 위치를 찾을 수도 있습니다. */greengrass/v2*/config/effectiveConfig.yaml
3. 개인 키와 인증서를 HSM으로 가져옵니다. 개인 키와 인증서를 가져오는 방법을 알아보려면 HSM 설명서를 참조하십시오. 이전에 PKCS #11 토큰을 초기화한 슬롯 ID 및 사용자 PIN을 사용하여 개인 키와 인증서를 가져오십시오. 개인 키와 인증서에 동일한 개체 레이블과 개체 ID를 사용해야 합니다. 각 파일을 가져올 때 지정한 개체 레이블을 저장합니다. 나중에 HSM의 개인 키와 인증서를 사용하도록 AWS IoT Greengrass Core 소프트웨어 구성을 업데이트할 때 이 레이블을 사용합니다.
4. HSM의 개인 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 구성을 업데이트하십시오. 구성을 업데이트하려면 AWS IoT Greengrass 코어 구성 파일을 수정하고 업데이트된 구성 파일로 AWS IoT Greengrass Core 소프트웨어를 실행하여 새 구성을 적용합니다.

다음을 따릅니다.

a. AWS IoT GreengrassCore 구성 파일의 백업을 생성합니다. 하드웨어 보안을 구성할 때 문제가 발생하는 경우 이 백업을 사용하여 코어 장치를 복원할 수 있습니다.

```
sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
```

b. 텍스트 편집기에서 AWS IoT Greengrass 코어 구성 파일을 엽니다. 예를 들어 다음 명령을 실행하여 GNU nano를 사용하여 파일을 편집할 수 있습니다. Greengrass 루트 폴더의 */greengrass/v2* 경로로 바꾸십시오.

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

c. 의 값을 HSM의 프라이빗 키에 대한 PKCS #11 `system.privateKeyPath` URI로 대체하십시오. *iotdevicekey#* 이전에 개인 키와 인증서를 가져온 개체 레이블로 바꾸십시오.

```
pkcs11:object=iotdevicekey;type=private
```

d. 의 값을 HSM에 있는 인증서의 PKCS #11 `system.certificateFilePath` URI로 바꾸십시오. *iotdevicekey#* 이전에 개인 키와 인증서를 가져온 개체 레이블로 바꾸십시오.

```
pkcs11:object=iotdevicekey;type=cert
```

이 단계를 완료하면 AWS IoT Greengrass 코어 구성 파일의 `system` 속성이 다음 예와 비슷해질 것입니다.

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. 업데이트된 `effectiveConfig.yaml` 파일에 구성을 적용합니다. `Greengrass.jar--init-config`파라미터와 함께 실행하여 구성을 적용합니다`effectiveConfig.yaml`. Greengrass 루트 폴더의 */greengrass/v2* 경로로 바꾸십시오.

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
```

```
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. AWS IoT GreengrassCore 소프트웨어를 시작합니다. systemd를 사용하여 AWS IoT GreengrassCore 소프트웨어를 시스템 서비스로 구성한 경우 다음 명령을 실행하여 소프트웨어를 시작할 수 있습니다.

   ```
   sudo systemctl start greengrass.service
   ```

   자세한 설명은 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.

7. AWS IoT GreengrassCore 소프트웨어 로그를 확인하여 소프트웨어가 시작되고 에 AWS 클라우드 연결되는지 확인합니다. AWS IoT GreengrassCore 소프트웨어는 개인 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 연결합니다.

   ```
   sudo tail -f /greengrass/v2/logs/greengrass.log
   ```

   다음 정보 수준 로그 메시지는 AWS IoT Greengrass Core 소프트웨어가 AWS IoT 및 AWS IoT Greengrass 서비스에 성공적으로 연결되었음을 나타냅니다.

   ```
   2021-12-06T22:47:53.702Z [INFO] (Thread-3)
    com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
    Core. {clientId=MyGreengrassCore5, sessionPresent=false}
   ```

8. (선택 사항) AWS IoT Greengrass Core 소프트웨어가 HSM의 개인 키 및 인증서와 함께 작동하는지 확인한 후 장치의 파일 시스템에서 개인 키와 인증서 파일을 삭제하십시오. 다음 명령을 실행하고 파일 경로를 개인 키 및 인증서 파일의 경로로 바꿉니다.

   ```
   sudo rm /greengrass/v2/privKey.key
   sudo rm /greengrass/v2/thingCert.crt
   ```

PKCS #11 지원이 없는 하드웨어를 사용하십시오.

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. 하지만 하드웨어에 해당하는 PKCS #11 라이브러리 구현이 없거나 사용자 지정 PKCS #11 공급자를 작성하려는 경우에는 Amazon Web Services 엔터프라이즈 지원 담당자에게 통합 관련 질문을 문의하십시오.

다음 사항도 참조하십시오.

- PKCS #11 암호화 토큰 인터페이스 사용 가이드 버전 2.4.0
- 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.

- 그린그래스 코어 디바이스

  Greengrass 코어 디바이스는 인증서와 AWS IoT 정책을 사용하여 및 에 AWS IoT Core 연결합니다. AWS IoT Greengrass 또한 인증서 및 정책을 통해 구성 요소 및 구성을 코어 장치에 AWS IoT Greengrass 배포할 수 있습니다.
- 클라이언트 디바이스

  MQTT 클라이언트 장치는 인증서와 정책을 사용하여 AWS IoT Greengrass 서비스에 연결합니다. AWS IoT Core 이를 통해 클라이언트 디바이스는 AWS IoT Greengrass 클라우드 검색을 사용하여 Greengrass 코어 디바이스를 찾고 연결할 수 있습니다. 클라이언트 장치는 동일한 인증서를 사용하여 AWS IoT Core 클라우드 서비스 및 코어 장치에 연결합니다. 또한 클라이언트 디바이스는 코어 디바이스와의 상호 인증을 위해 검색 정보를 사용합니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

X.509 인증서

코어 디바이스와 클라이언트 디바이스 간, 디바이스 및/또는 디바이스 간 통신을 AWS IoT Greengrass 인증해야 합니다. AWS IoT Core 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.

- 인터넷에 연결되거나 인터넷을 통해 연결되는 AWS IoT Core Greengrass 코어 디바이스의 AWS IoT 클라이언트 구성 요소입니다. AWS IoT Greengrass
- 인터넷을 AWS IoT Greengrass 통해 연결하여 코어 디바이스를 검색하는 클라이언트 디바이스.
- 로컬 네트워크를 통해 그룹의 Greengrass 디바이스에 연결하는 Greengrass 코어의 MQTT 브로커 구성 요소입니다.

AWS IoT Greengrass코어 디바이스는 Greengrass 루트 폴더에 인증서를 저장합니다.

CA(인증 기관) 인증서

Greengrass 코어 디바이스와 클라이언트 디바이스는 및 서비스를 통한 인증에 사용되는 루트 CA 인증서를 다운로드합니다AWS IoT Core. AWS IoT Greengrass 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.

또한 클라이언트 디바이스는 Greengrass 코어 디바이스 CA 인증서를 다운로드합니다. 이들은 이 인증서를 사용하여 상호 인증 중에 코어 디바이스의 MQTT 서버 인증서를 검증합니다.

로컬 MQTT 브로커의 인증서 교체

클라이언트 장치 지원을 활성화하면 Greengrass 코어 장치는 클라이언트 장치가 상호 인증에 사용하는 로컬 MQTT 서버 인증서를 생성합니다. 이 인증서는 코어 디바이스가 클라우드에 저장하는 코어 디바이스 CA 인증서에 의해 서명됩니다AWS IoT Greengrass. 클라이언트 디바이스는 코어 디바이스를 발견하면 코어 디바이스 CA 인증서를 검색합니다. 코어 디바이스에 연결할 때 코어 디바이스 CA 인증서를 사용하여 코어 디바이스의 MQTT 서버 인증서를 확인합니다. 코어 디바이스 CA 인증서는 5년 후에 만료됩니다.

MQTT 서버 인증서는 기본적으로 7일마다 만료되며 이 기간을 2일에서 10일 사이로 구성할 수 있습니다. 이 제한된 기간은 보안 모범 사례를 기반으로 합니다. 이 로테이션은 공격자가 MQTT 서버 인증서와 개인 키를 도용하여 Greengrass 코어 디바이스를 가장하려는 위협을 완화하는 데 도움이 됩니다.

Greengrass 코어 디바이스는 MQTT 서버 인증서가 만료되기 24시간 전에 교체합니다. Greengrass 코어 디바이스는 새 인증서를 생성하고 로컬 MQTT 브로커를 다시 시작합니다. 이 경우 Greengrass 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 끊어집니다. 클라이언트 디바이스는 잠시 후 Greengrass 코어 디바이스에 다시 연결할 수 있습니다.

데이터 영역 작업에 대한 AWS IoT 정책

AWS IoT정책을 사용하여 AWS IoT Core 및 AWS IoT Greengrass 데이터 플레인에 대한 액세스를 승인하십시오. AWS IoT Core데이터 플레인은 장치, 사용자 및 애플리케이션에 대한 작업을 제공합니다. 이러한 작업에는 주제에 연결하고 주제를 구독하는 AWS IoT Core 기능이 포함됩니다. AWS IoT Greengrass데이터 플레인은 Greengrass 디바이스에 대한 작업을 제공합니다. 자세한 설명은 AWS IoT Greengrass V2 정책 작업 섹션을 참조하세요. 이러한 작업에는 구성 요소 종속성을 해결하고 공용 구성 요소 아티팩트를 다운로드하는 기능이 포함됩니다.

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.

> **Important**
>
> 사물 정책 변수 (`iot:Connection.Thing.*`) 는 코어 디바이스 또는 Greengrass 데이터 플레인 작업에 대한 AWS IoT 정책에서 지원되지 않습니다. 대신 이름이 비슷한 여러 장치를 매칭하는 와일드카드를 사용할 수 있습니다. 예를 `MyGreengrassDevice1` `MyGreengrassDevice2` 들어 `MyGreengrassDevice*` 일치하도록 지정하는 등의 작업을 수행할 수 있습니다.

> **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 V2 정책 작업

AWS IoT Greengrass V2Greengrass 코어 디바이스 및 클라이언트 디바이스가 정책에서 AWS IoT 사용할 수 있는 다음과 같은 정책 조치를 정의합니다. 정책 작업에 사용할 리소스를 지정하려면 해당 리소스의 Amazon 리소스 이름 (ARN) 을 사용합니다.

핵심 디바이스 작업

`greengrass:GetComponentVersionArtifact`

퍼블릭 구성 요소 아티팩트 또는 Lambda 구성 요소 아티팩트를 다운로드하기 위해 미리 서명된 URL을 가져올 권한을 부여합니다.

이 권한은 코어 디바이스가 퍼블릭 구성 요소 또는 아티팩트가 있는 Lambda를 지정하는 배포를 수신할 때 평가됩니다. 코어 디바이스에 아티팩트가 이미 있는 경우 아티팩트를 다시 다운로드하지 않습니다.

리소스 유형: `componentVersion`

리소스 ARN 형식: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

배포를 위한 구성 요소, 버전 및 플랫폼 요구 사항을 충족하는 구성 요소 목록을 식별할 권한을 부여합니다. 요구 사항이 충돌하거나 요구 사항을 충족하는 구성 요소가 없는 경우 이 작업은 오류를 반환하고 디바이스에서 배포가 실패합니다.

이 권한은 핵심 기기가 구성 요소를 지정하는 배포를 받을 때 평가됩니다.

리소스 유형: 없음

리소스 ARN 형식: *

`greengrass:GetDeploymentConfiguration`

대용량 배포 문서를 다운로드하기 위해 미리 서명된 URL을 가져올 권한을 부여합니다.

이 권한은 코어 디바이스가 7KB (배포가 사물을 대상으로 하는 경우) 또는 31KB (배포가 사물 그룹을 대상으로 하는 경우) 보다 큰 배포 문서를 지정하는 배포를 받을 때 평가됩니다. 배포 문서에는 구성 요소 구성, 배포 정책 및 배포 메타데이터가 포함됩니다. 자세한 설명은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요.

이 기능은 Greengrass 핵 구성 요소 v2.3.0 이상에서 사용할 수 있습니다.

리소스 유형: 없음

리소스 ARN 형식: *

`greengrass:ListThingGroupsForCoreDevice`

핵심 장치의 사물 그룹 계층 구조를 가져올 권한을 부여합니다.

이 권한은 코어 디바이스가 배포를 수신할 때 AWS IoT Greengrass 확인됩니다. 코어 디바이스는 이 작업을 사용하여 마지막 배포 이후 사물 그룹에서 제거되었는지 여부를 식별합니다. 코어 장치가 사물 그룹에서 제거되고 해당 사물 그룹이 코어 장치에 대한 배포 대상인 경우 코어 장치는 해당 배포로 설치된 구성 요소를 제거합니다.

이 기능은 v2.5.0 이상의 Greengrass 핵 구성 요소에서 사용됩니다.

리소스 유형: (코어 디바이스) `thing`

리소스 ARN 형식: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

코어 디바이스에 연결된 클라이언트 디바이스의 ID를 확인할 수 있는 권한을 부여합니다.

이 권한은 코어 장치가 클라이언트 장치 인증 구성 요소를 실행하고 클라이언트 장치로부터 MQTT 연결을 수신할 때 평가됩니다. 클라이언트 기기는 기기 인증서를 제공합니다. AWS IoT 그러면 코어 디바이스가 디바이스 인증서를 AWS IoT Greengrass 클라우드 서비스에 전송하여 클라이언트 디바이스의 ID를 확인합니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

리소스 유형: 없음

리소스 ARN 형식: *

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

클라이언트 장치가 AWS IoT 인증서와 연결되어 있는지 여부를 확인할 수 있는 권한을 부여합니다.

이 권한은 코어 장치가 클라이언트 장치 인증 구성 요소를 실행하고 클라이언트 장치가 MQTT를 통해 연결하도록 승인할 때 평가됩니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

> **Note**
>
> 코어 디바이스에서 이 작업을 사용하려면 Greengrass 서비스 역할을 사용자와 연결하고 권한을 허용해야 합니다. AWS 계정 `iot:DescribeCertificate`

리소스 유형: `thing` (클라이언트 장치)

리소스 ARN 형식: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

클라이언트 디바이스가 코어 디바이스를 확인하기 위해 다운로드할 수 있는 CA (인증 기관) 인증서를 업로드할 권한을 부여합니다.

이 권한은 코어 장치가 클라이언트 장치 인증 구성 요소를 설치하고 실행할 때 평가됩니다. 이 구성 요소는 로컬 인증 기관을 만들고 이 작업을 사용하여 CA 인증서를 업로드합니다. 클라이언트 장치는 Discover 작업을 사용하여 연결할 수 있는 핵심 장치를 찾을 때 이러한 CA 인증서를 다운로드합니다. 클라이언트 디바이스가 코어 디바이스의 MQTT 브로커에 연결되면 이러한 CA 인증서를 사용하여 코어 디바이스의 ID를 확인합니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

리소스 유형: 없음

ARN 형식: *

`greengrass:GetConnectivityInfo`

코어 디바이스의 연결 정보를 가져올 권한을 부여합니다. 이 정보는 클라이언트 장치를 코어 장치에 연결할 수 있는 방법을 설명합니다.

이 권한은 코어 장치가 클라이언트 장치 인증 구성 요소를 설치하고 실행할 때 평가됩니다. 이 구성 요소는 연결 정보를 사용하여 작업과 함께 AWS IoT Greengrass 클라우드 서비스에 업로드할 유효

한 CA 인증서를 생성합니다. PutCertificateAuthories 클라이언트 디바이스는 이러한 CA 인증서를 사용하여 코어 디바이스의 ID를 확인합니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

AWS IoT Greengrass컨트롤 플레인에서 이 작업을 사용하여 코어 디바이스의 연결 정보를 볼 수도 있습니다. 자세한 내용을 알아보려면 AWS IoT Greengrass V1 API 참조의 GetConnectivityInfo 섹션을 참조하십시오.

리소스 유형: `thing` (코어 디바이스)

리소스 ARN 형식: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:UpdateConnectivityInfo`

코어 디바이스의 연결 정보를 업데이트할 권한을 부여합니다. 이 정보는 클라이언트 장치를 코어 장치에 연결할 수 있는 방법을 설명합니다.

이 권한은 코어 장치가 IP 탐지기 구성 요소를 실행할 때 평가됩니다. 이 구성 요소는 클라이언트 장치가 로컬 네트워크의 핵심 장치에 연결하는 데 필요한 정보를 식별합니다. 그러면 이 구성 요소는 이 작업을 사용하여 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 업로드하므로 클라이언트 장치가 검색 작업을 통해 이 정보를 검색할 수 있습니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

AWS IoT Greengrass컨트롤 플레인에서 이 작업을 사용하여 코어 디바이스의 연결 정보를 수동으로 업데이트할 수도 있습니다. 자세한 내용을 알아보려면 AWS IoT Greengrass V1 API 참조의 UpdateConnectivityInfo 섹션을 참조하십시오.

리소스 유형: `thing` (코어 디바이스)

리소스 ARN 형식: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

클라이언트 디바이스 작업

`greengrass:Discover`

클라이언트 장치가 연결할 수 있는 핵심 장치의 연결 정보를 검색할 수 있는 권한을 부여합니다. 이 정보는 클라이언트 장치를 코어 장치에 연결할 수 있는 방법을 설명합니다. 클라이언트 장치는 BatchAssociateClientDeviceWithCoreDevice작업을 사용하여 연결된 핵심 장치만 검색할 수 있습니다. 자세한 설명은 로컬 IoT 기기와 상호작용 섹션을 참조하세요.

리소스 유형: `thing` (클라이언트 장치)

리소스 ARN 형식: `arn:aws:iot:region:account-id:thing/client-device-thing-name`

코어 디바이스 정책 업데이트 AWS IoT

AWS IoT Greengrass및 AWS IoT 콘솔 또는 AWS IoT API를 사용하여 핵심 장치의 정책을 보고 업데이트할 수 있습니다. AWS IoT

> **Note**
>
> AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 코어 기기에는 모든 AWS IoT Greengrass 작업에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다 (). `greengrass:*` 다음 단계에 따라 코어 장치가 사용하는 작업에만 액세스를 제한할 수 있습니다.

핵심 장치 AWS IoT 정책 검토 및 업데이트 (콘솔)

1. AWS IoT Greengrass콘솔 탐색 메뉴에서 Core devices를 선택합니다.
2. 코어 디바이스 페이지에서 업데이트할 코어 디바이스를 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 코어 디바이스의 사물로 연결되는 링크를 선택합니다. 이 링크를 클릭하면 AWS IoT 콘솔에서 사물 세부 정보 페이지가 열립니다.
4. 사물 세부 정보 페이지에서 인증서를 선택합니다.
5. 인증서 탭에서 사물의 활성 인증서를 선택합니다.
6. 인증서 세부 정보 페이지에서 정책을 선택합니다.
7. 정책 탭에서 검토 및 업데이트할 AWS IoT 정책을 선택합니다. 코어 디바이스의 활성 인증서에 연결된 모든 정책에 필요한 권한을 추가할 수 있습니다.

> **Note**
>
> AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 지정된 정책을 선택하는 것이 좋습니다 (있는 GreengrassV2IoTThingPolicy경우). 빠른 설치 프로그램으로 만든 코어 디바이스는 기본적으로 이 정책 이름을 사용합니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 핵심 장치에도 이러한 권한이 부여됩니다.

8. 정책 개요에서 활성 버전 편집을 선택합니다.
9. 정책을 검토하고 필요에 따라 권한을 추가, 제거 또는 편집합니다.
10. 새 정책 버전을 활성 버전으로 설정하려면 정책 버전 상태에서 편집한 버전을 이 정책의 활성 버전으로 설정을 선택합니다.
11. 새 버전으로 저장을 선택합니다.

핵심 기기 AWS IoT 정책 검토 및 업데이트 (AWS CLI)

1. 핵심 장치 사물에 대한 원칙을 나열하십시오. AWS IoT 사물 주체는 X.509 장치 인증서 또는 기타 식별자일 수 있습니다. 다음 명령을 실행하고 코어 디바이스의 이름으로 *MyGreengrassCore*바꿉니다.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   이 작업은 코어 디바이스의 사물 주체를 나열하는 응답을 반환합니다.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. 코어 디바이스의 활성 인증서를 식별하십시오. 다음 명령을 실행하고 활성 *#### ## ### CertificateID#* 이전 단계의 각 인증서 ID로 바꿉니다. 인증서 ID는 인증서 ARN 끝에 있는 16진수 문자열입니다. `--query`인수는 인증서 상태만 출력하도록 지정합니다.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   작업은 인증서 상태를 문자열로 반환합니다. 예를 들어 인증서가 활성 상태인 경우 이 작업이 `"ACTIVE"` 출력됩니다.

3. 인증서에 첨부된 AWS IoT 정책을 나열하십시오. 다음 명령을 실행하고 인증서 ARN을 인증서의 ARN으로 교체합니다.

   ```
   aws iot list-principal-policies --principal arn:aws:iot:us-
   west-2:123456789012:cert/certificateId
   ```

이 작업은 인증서에 연결된 AWS IoT 정책을 나열하는 응답을 반환합니다.

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. 보고 업데이트할 정책을 선택합니다.

> **Note**
>
> AWS IoT Greengrass코어 소프트웨어 설치 프로그램을 사용하여 리소스를 프로비저닝한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 지정된 정책을 선택하는 것이 좋습니다 (있는 GreengrassV2IoTThingPolicy경우). 빠른 설치 프로그램으로 만든 코어 디바이스는 기본적으로 이 정책 이름을 사용합니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 핵심 장치에도 이러한 권한이 부여됩니다.

5. 정책 문서를 가져오세요. 다음 명령을 실행하고 *ThingPolicyGreenGrassV2IoT#* 정책 이름으로 대체합니다.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

이 작업은 정책 문서와 정책에 대한 기타 정보가 포함된 응답을 반환합니다. 정책 문서는 문자열로 직렬화된 JSON 객체입니다.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
```

```
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. 온라인 변환기나 다른 도구를 사용하여 정책 문서 문자열을 JSON 객체로 변환한 다음 이 문자열을 이름이 지정된 파일에 저장합니다. `iot-policy.json`

   예를 들어 jq 도구가 설치되어 있는 경우 다음 명령을 실행하여 정책 문서를 가져와서 JSON 개체로 변환한 다음 정책 문서를 JSON 개체로 저장할 수 있습니다.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. 정책 문서를 검토하고 필요에 따라 권한을 추가, 제거 또는 편집하십시오.

   예를 들어, Linux 기반 시스템에서는 다음 명령을 실행하여 GNU nano를 사용하여 파일을 열 수 있습니다.

   ```
   nano iot-policy.json
   ```

   완료하면 정책 문서가 코어 디바이스의 최소 AWS IoT 정책과 비슷해 보일 수 있습니다.

8. 변경 내용을 새 버전의 정책으로 저장합니다. 다음 명령을 실행하고 *ThingPolicyGreenGrassV2IoT#* 정책 이름으로 대체합니다.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

작업이 성공하면 다음 예제와 비슷한 응답이 반환됩니다.

```
{
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책

Important

이후 버전의 Greengrass nucleus 구성 요소에는 최소 정책에 대한 추가 권한이 필요합니다. AWS IoT 추가 권한을 부여하려면 코어 디바이스의 AWS IoT 정책을 업데이트해야 할 수 있습니다.

- Greengrass nucleus v2.5.0 이상을 실행하는 코어 디바이스는 사물 그룹에서 코어 디바이스를 제거할 때 `greengrass:ListThingGroupsForCoreDevice` 권한을 사용하여 구성 요소를 제거합니다.

- Greengrass nucleus v2.3.0 이상을 실행하는 코어 디바이스는 `greengrass:GetDeploymentConfiguration` 권한을 사용하여 대규모 배포 구성 문서를 지원합니다.

다음 예제 정책에는 코어 디바이스용 기본 Greengrass 기능을 지원하는 데 필요한 최소 작업 세트가 포함됩니다.

- `Connect`정책에는 코어 디바이스 사물 이름 뒤에 * 와일드카드가 포함됩니다 (예:). *`core-device-thing-name`** 코어 디바이스는 동일한 디바이스 인증서를 사용하여 여러 개의 동시 구독을 수행하지만 연결의 클라이언트 ID가 코어 디바이스 사물 이름과 정확히 일치하지 않을 수 있습니다. AWS IoT Core 처음 50개 구독 이후에는 코어 기기가 클라이언트 *`core-device-thing-name`*#*`number`* ID로 사용하며, 이 ID는 구독이 50개 추가될 때마다 *`number`* 증가합니다. 예를 들어, 라는 이름의 코어 디바이스에서 150개의 동시 구독을 `MyCoreDevice` 생성하는 경우 다음 클라이언트 ID를 사용합니다.
  - 구독 1~50: `MyCoreDevice`
  - 구독 51건부터 100건까지: `MyCoreDevice#2`
  - 서브스크립션 101~150: `MyCoreDevice#3`

  와일드카드를 사용하면 접미사가 있는 이러한 클라이언트 ID를 사용하는 코어 장치를 연결할 수 있습니다.
- 이 정책은 코어 디바이스가 섀도우 상태에 사용되는 주제를 포함하여 메시지를 게시하고 구독하고 메시지를 수신할 수 있는 MQTT 정책과 주제 필터를 나열합니다. Greengrass 구성 요소와 클라이언트 장치 간의 AWS IoT Core 메시지 교환을 지원하려면 허용하려는 주제 및 주제 필터를 지정합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 게시/구독 정책 예제를 참조하십시오.
- 이 정책은 원격 분석 데이터에 대해 다음 주제에 게시할 권한을 부여합니다.

  ```
  $aws/things/core-device-thing-name/greengrass/health/json
  ```

  원격 분석을 사용하지 않도록 설정한 핵심 장치에 대해서는 이 권한을 제거할 수 있습니다. 자세한 설명은 AWS IoT Greengrass핵심 장치에서 시스템 상태 원격 측정 데이터 수집 섹션을 참조하세요.
- 정책은 역할 별칭을 통해 AWS IoT IAM 역할을 수임할 권한을 부여합니다. 코어 디바이스는 토큰 교환 역할이라고 하는 이 역할을 사용하여 요청을 AWS 인증하는 데 사용할 수 있는 AWS 자격 증명을 획득합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

AWS IoT GreengrassCore 소프트웨어를 설치할 때 이 권한만 포함하는 두 번째 AWS IoT 정책을 생성하여 연결합니다. 코어 장치의 기본 정책에 이 권한을 포함하면 다른 AWS IoT AWS IoT 정책을 분리하고 삭제할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:region:account-id:client/core-device-thing-name*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/greengrass/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/greengrassv2/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/jobs/*",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-thing-name/jobs/*",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-thing-name/shadow/*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:region:account-id:rolealias/token-exchange-role-
alias-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
}
```

클라이언트 AWS IoT 장치를 지원하기 위한 최소 정책

다음 예제 정책에는 코어 장치에서 클라이언트 장치와의 상호 작용을 지원하는 데 필요한 최소 작업 집합이 포함되어 있습니다. 클라이언트 장치를 지원하려면 코어 장치에 기본 작동을 위한 최소 AWS IoT 정책 외에도 이 AWS IoT 정책의 사용 권한이 있어야 합니다.

- 정책을 통해 코어 기기는 자체 연결 정보를 업데이트할 수 있습니다. 이 권한 (`greengrass:UpdateConnectivityInfo`) 은 IP 탐지기 구성 요소를 코어 장치에 배포하는 경우에만 필요합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
```

```
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:region:account-id:thing/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-device-thing-name"
            ]
        }
    ]
}
```

클라이언트 디바이스에 대한 최소 AWS IoT 정책

다음 예제 정책에는 클라이언트 장치가 MQTT를 통해 연결 및 통신하는 핵심 장치를 검색하는 데 필요한 최소 작업 집합이 포함되어 있습니다. 클라이언트 장치 AWS IoT 정책에는 장치가 관련 Greengrass 코어 장치에 대한 연결 정보를 검색할 수 있도록 허용하는 `greengrass:Discover` 작업이 포함되어야 합니다. `Resource`섹션에서 Greengrass 코어 디바이스의 ARN이 아닌 클라이언트 디바이스의 Amazon 리소스 이름 (ARN) 을 지정하십시오.

- 이 정책은 모든 MQTT 주제에 대한 통신을 허용합니다. 베스트 보안 프랙티스를 따르려면 `iot:Publishiot:Subscribe`, 및 `iot:Receive` 권한을 클라이언트 장치가 사용 사례에 요구하는 최소 항목으로 제한하십시오.
- 정책을 통해 AWS IoT 사물은 모든 사물에 대한 핵심 장치를 검색할 수 있습니다. 최상의 보안 관행을 따르려면 클라이언트 장치의 사물 또는 AWS IoT AWS IoT 사물 집합과 일치하는 와일드카드로 `greengrass:Discover` 권한을 제한하십시오.

  > **Important**
  >
  > 사물 정책 변수 (`iot:Connection.Thing.*`) 는 코어 디바이스 또는 Greengrass 데이터 플레인 작업에 대한 AWS IoT 정책에서 지원되지 않습니다. 대신 이름이 비슷한 여러 장치를 매칭하는 와일드카드를 사용할 수 있습니다. 예를 `MyGreengrassDevice1` `MyGreengrassDevice2` 들어 `MyGreengrassDevice*` 일치하도록 지정하는 등의 작업을 수행할 수 있습니다.

- Greengrass 코어 디바이스가 클라이언트 디바이스의 섀도우 동기화 `iot:DeleteThingShadow` 작업을 처리하기 때문에 클라이언트 디바이스의 AWS IoT 정책에는 일반적으로 권한 또는 작업이 필요하지 않습니다. `iot:GetThingShadow` `iot:UpdateThingShadow` 코어 디바이스가 클라이

언트 디바이스 섀도우를 처리할 수 있도록 하려면 코어 디바이스의 AWS IoT 정책에서 이러한 작업을 허용하고 Resource 섹션에 클라이언트 디바이스의 ARN이 포함되어 있는지 확인하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "greengrass:Discover"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/*"
            ]
        }
    ]
}
```

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 이

해하십시오. 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 에 대한 액세스를 관리하려면AWS IoT Greengrass 먼저 에 사용할 수 있는 IAM 기능을 이해해야AWS IoT Greengrass 합니다.

IAM 기능	Greengrass에서 지원합니까?
리소스 수준 권한이 있는 자격 증명 기반 정책	예
리소스 기반 정책	아니요
ACL(액세스 제어 목록)	아니요
태그 기반 승인	예
임시 자격 증명	예
서비스 연결 역할	아니요
서비스 역할	예

다른AWS 서비스에서 IAM을 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서 의 IAM으로 작업하는AWS 서비스 를 참조하십시오.

AWS IoT Greengrass에 대한 자격 증명 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 설명서 의 IAM JSON 정책 요소 참조 를 참조하십시오.

작업

관리자는 AWS JSON 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 태스크를 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 포함시킵니다.

AWS IoT Greengrass의 정책 작업은 작업 앞에 `greengrass:` 접두사를 사용합니다. 예를 들어 사용자가`ListCoreDevices` API 작업을 사용하여 핵심 장치를 나열하도록AWS 계정 허용하려면 해당 정책에`greengrass:ListCoreDevices` 작업을 포함합니다. 정책 문에는 `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 대괄호 (`[]`) 로 구분합니다.

```
"Action": [
  "greengrass:action1",
  "greengrass:action2",
  "greengrass:action3"
]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `List`라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 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 정보는 the section called “가능한 최소 권한 부여”을 참조하세요.

전체AWS IoT Greengrass 작업 목록은 IAM 사용 설명서에서 정의한AWS IoT Greengrass 작업을 참조하십시오.

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 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `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 명령문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

다음 표에는 정책 설명의 `Resource` 요소에 사용할 수 있는 AWS IoT Greengrass 리소스 ARN이 나와 있습니다. AWS IoT Greengrass작업에 지원되는 리소스 수준 권한 매핑은 IAM 사용AWS IoT Greengrass 설명서에서 정의한 작업을 참조하십시오.

일부 AWS IoT Greengrass 작업(예: 일부 목록 작업)은 특정 리소스에 대해 수행할 수 없습니다. 이러한 경우 와일드카드만 사용해야 합니다.

```
"Resource": "*"
```

명령문에 여러 리소스 ARN을 지정하려면 다음과 같이 대괄호 (`[]`) 사이에 리소스 ARN을 나열하고 쉼표로 구분합니다.

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

ARN 형식에 대한 자세한 내용은 의 Amazon 리소스 이름 (ARN) 및AWS 서비스 네임스페이스를 참조하십시오 Amazon Web Services 일반 참조.

조건 키

관리자는 AWS JSON 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.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 선택 사항입니다. 같음이나 미만 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` 태스크를 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우 AWS는 논리적 `OR` 태스크를 사용하여 조건을 평가합니다. 문의 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 키를 참조하세요.

예

AWS IoT Greengrass 자격 증명 기반 정책의 예를 보려면 the section called “자격 증명 기반 정책 예제” 단원을 참조하세요.

AWS IoT Greengrass를 위한 리소스 기반 정책

AWS IoT Greengrass는 리소스 기반 정책을 지원하지 않습니다.

ACL(액세스 제어 목록)

AWS IoT Greengrass는 ACL을 지원하지 않습니다.

AWS IoT Greengrass 태그 기반 권한 부여

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 제공하십시오. 자세한 정보는 리소스 태깅을 참조하세요.

에 대한 IAM 역할AWS IoT Greengrass

IAM 역할은 특정 권한을 가지고 있는 AWS 계정 계정 내 엔터티입니다.

AWS IoT Greengrass에서 임시 자격 증명 사용

임시 자격 증명은 페더레이션을 통해 로그인하거나, IAM 역할을 수임하는 데 사용됩니다. AssumeRole또는 등의AWS STS API 작업을 호출하여 임시 보안 자격 증명을 얻습니다 GetFederationToken.

Greengrass 코어에서는 장치 역할에 대한 임시 자격 증명을 Greengrass 구성 요소에 사용할 수 있습니다. 구성 요소가 SDK를 사용하는 경우AWS SDK가 자동으로 이 작업을 수행하므로 자격 증명을 얻기 위한 로직을 추가할 필요가 없습니다.AWS

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 태스크를 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 Greengrass코어 디바이스는 서비스 역할을 사용하여 Greengrass 구성 요소 및 Lambda 함수가 사용자를 대신하여 일부AWS 리소스에 액세스할 수 있도록 합니다. 자세한 정보는 the section called “핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스”을 참조하세요.

AWS IoT Greengrass는 서비스 역할을 사용하여 사용자를 대신하여 일부 AWS 리소스에 액세스합니다. 자세한 정보는 Greengrass 서비스 역할 단원을 참조하세요.

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 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 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.

정책 예

다음 예제의 고객 정의 정책은 일반적인 시나리오에 대한 권한을 부여합니다.

예

- 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": "*"
        }
    ]
}
```

핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

AWS IoT Greengrass코어 디바이스는AWS IoT Core호출을 승인하는 자격 증명 공급자AWS서비스. 이AWS IoT Core자격 증명 공급자를 사용하면 장치가 X.509 인증서를 인증할 고유 장치 ID로 사용할

수 있습니다.AWS요청. 이렇게 하면 을 저장할 필요가 없습니다.AWS에 액세스 키 ID 및 보안 액세스 키.AWS IoT Greengrass코어 디바이스. 자세한 내용은 단원을 참조하십시오.에 직접 통화 권한 부여 AWS서비스의AWS IoT Core개발자 안내서.

실행할 때AWS IoT Greengrass핵심 소프트웨어, 프로비저닝하도록 선택할 수 있습니다.AWS 핵심 디바이스에 필요한 리소스입니다. 여기에는 다음이 포함됩니다.AWS Identity and Access Management(IAM) 핵심 디바이스가 다음을 통해 가정하는 역할AWS IoT Core자격 증명 공급자. 사용`--provision true`핵심 디바이스가 임시로 가져올 수 있도록 하는 역할 및 정책을 구성하는 인수AWS자격 증명. 이 인수는 또한AWS IoT이 IAM 역할을 가리키는 역할 별칭입니다. IAM 역할의 이름을 지정하고AWS IoT사용할 역할 별칭입니다. 지정하는 경우`--provision true`이러한 다른 이름 매개 변수가 없으면 Greengrass 코어 디바이스는 다음과 같은 기본 리소스를 생성하고 사용합니다.

- IAM 역할:`GreengrassV2TokenExchangeRole`

  이 역할의 정책은 입니다.`GreengrassV2TokenExchangeRoleAccess`그리고 허용하는 신뢰 관계`credentials.iot.amazonaws.com`역할을 위임합니다. 이 정책에는 핵심 장치에 대한 최소 권한이 포함됩니다.

  > **Important**
  >
  > 이 정책에는 S3 버킷의 파일에 대한 액세스가 포함되지 않습니다. 역할에 권한을 추가해야 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색할 수 있습니다. 자세한 정보는 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용을 참조하십시오.

- AWS IoT역할 별칭:`GreengrassV2TokenExchangeRoleAlias`

  이 역할 별칭은 IAM 역할을 나타냅니다.

자세한 정보는 3단계:AWS IoT Greengrass 핵심 소프트웨어 설치을 참조하십시오.

기존 코어 디바이스에 대한 역할 별칭을 설정할 수도 있습니다. 이렇게 하려면 을 구성합니다.`iotRoleAlias`의 구성 매개 변수Greengrass 핵 성분.

임시 취득 가능AWS수행할 이 IAM 역할의 자격 증명AWS사용자 지정 구성 요소에서 작업을 수행합니다. 자세한 정보는 AWS서비스와 상호작용을 참조하십시오.

주제

- 코어 디바이스에 대한 서비스 역할 권한
- 구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용

코어 디바이스에 대한 서비스 역할 권한

이 역할을 통해 다음 서비스가 역할을 위임할 수 있습니다.

- `credentials.iot.amazonaws.com`

이AWS IoT Greengrass핵심 소프트웨어는 이 역할을 생성하며, 다음 권한 정책을 사용하여 핵심 장치가 연결되고 로그를 보낼 수 있도록 허용합니다.AWS. 정책의 이름은 기본적으로 로 끝나는 IAM 역할의 이름으로 지정됩니다.`Access`. 예를 들어 기본 IAM 역할 이름을 사용하는 경우 이 정책의 이름은`GreengrassV2TokenExchangeRoleAccess`.

Greengrass nucleus v2.5.0 and later

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
```

```
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

구성 요소 아티팩트에 대한 S3 버킷에 대한 액세스 허용

기본 코어 디바이스 역할은 코어 디바이스가 S3 버킷에 액세스하는 것을 허용하지 않습니다. S3 버킷에 아티팩트가 있는 구성 요소를 배포하려면`s3:GetObject`핵심 장치가 구성 요소 아티팩트를 다운로드할 수 있도록 허용하는 권한입니다. 핵심 장치 역할에 새 정책을 추가하여 이 권한을 부여할 수 있습니다.

Amazon S3 구성 요소 아티팩트에 대한 액세스를 허용하는 정책을 추가하려면

1. 이라는 파일 생성`component-artifact-policy.json`그리고 다음 JSON을 파일로 복사합니다. 이 정책은 S3 버킷의 모든 파일에 대한 액세스를 허용합니다. Replace*## ## ##*S3 버킷의 이름을 사용하여 코어 디바이스에 액세스할 수 있습니다.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
       }
     ]
   }
   ```

2. 다음 명령을 실행하여 의 정책 문서에서 정책을 생성합니다.`component-artifact-policy.json`.

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name MyGreengrassV2ComponentArtifactPolicy \
     --policy-document file://component-artifact-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name MyGreengrassV2ComponentArtifactPolicy ^
     --policy-document file://component-artifact-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name MyGreengrassV2ComponentArtifactPolicy `
     --policy-document file://component-artifact-policy.json
   ```

출력의 정책 메타데이터에서 Amazon 리소스 이름 (ARN) 정책을 복사합니다. 다음 단계에서 이 ARN을 사용하여 이 정책을 핵심 디바이스 역할에 연결합니다.

3. 다음 명령을 실행하여 정책을 핵심 디바이스 역할에 연결합니다. Replace*#####V2#######*을 실행할 때 지정한 역할의 이름을 사용합니다.AWS IoT Greengrass코어 소프트웨어. 그리고 정책 ARN을 이전 단계의 ARN으로 바꿉니다.

   Linux or Unix

   ```
   aws iam attach-role-policy \
     --role-name GreengrassV2TokenExchangeRole \
     --policy-arn
    arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam attach-role-policy ^
     --role-name GreengrassV2TokenExchangeRole ^
     --policy-arn
    arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
   ```

   PowerShell

   ```
   aws iam attach-role-policy `
     --role-name GreengrassV2TokenExchangeRole `
     --policy-arn
    arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
   ```

   명령에 출력이 없으면 성공하고 핵심 디바이스가 이 S3 버킷에 업로드하는 아티팩트에 액세스할 수 있습니다.

리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책

설치할 때AWS IoT Greengrass핵심 소프트웨어, 필요한 프로비저닝 가능AWS리소스 (예:AWS IoT디바이스에 대한 사물과 IAM 역할 로컬 개발 도구를 디바이스에 배포할 수도 있습니다. 설치 프로그램에 필요한 사항AWS사용자 인증 프로그램에서 이러한 작업을 수행할 수 있도록AWS 계정. 자세한 정보는 AWS IoT Greengrass 코어 소프트웨어 설치을 참조하십시오.

다음 예제 정책에는 설치 관리자가 이러한 리소스를 프로비저닝하는 데 필요한 최소 작업 세트가 포함됩니다. 이러한 권한은 다음을 지정하는 경우 필요합니다.`--provision`인수에 대한 인수입니다. Replace*account-id*당신과 함께AWS 계정ID 및 바꾸기*#####V2########*를 사용하여 지정한 토큰 교환 역할의 이름을 사용합니다.`--tes-role-name` 설치 관리자 인수.

> **Note**
>
> 이`DeployDevTools`정책 설명은 를 지정한 경우에만 필요함`--deploy-dev-tools`인수에 대한 인수입니다.

Greengrass nucleus v2.5.0 and later

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-id:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
```

```
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Earlier than v2.5.0

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
```

```
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

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. 이 역할을 통해 클라이언트 장치의 ID를 확인하고 핵심 장치 연결 정보를 관리할 수 있습니다. AWS IoT Greengrass

> **Note**
>
> AWS IoT Greengrass V1또한 이 역할을 사용하여 필수 작업을 수행합니다. 자세한 내용은 AWS IoT Greengrass V1개발자 안내서의 Greengrass 서비스 역할을 참조하십시오.

AWS IoT Greengrass가 리소스에 액세스하도록 허용하려면 Greengrass 서비스 역할을 AWS 계정 계정과 연결하고 AWS IoT Greengrass를 신뢰할 수 있는 엔터티로 지정해야 합니다. 역할에는 사용하는 AWS IoT Greengrass 기능에 대해 동등한 권한을 정의하는 AWSGreengrassResourceAccessRolePolicy관리형 정책 또는 사용자 지정 정책이 포함되어야 합니다. AWSAWS리소스에 액세스하는 데 AWS IoT Greengrass 사용하는 권한 세트를 정의하는 이 정책을 유지 관리합니다. 자세한 설명은 AWS 관리형 정책: AWSGreengrassResourceAccessRolePolicy 섹션을 참조하세요.

동일한 Greengrass 서비스 역할을 AWS 리전 여러 곳에서 재사용할 수 있지만 사용하는 AWS 리전 모든 곳의 계정과 연결해야 합니다. AWS IoT Greengrass 서비스 역할이 현재 AWS 리전 구성되지 않은 경우 코어 디바이스는 클라이언트 디바이스를 확인하지 못하고 연결 정보를 업데이트하지 못합니다.

다음 섹션에서는 OR를 사용하여 Greengrass 서비스 역할을 생성하고 관리하는 방법을 설명합니다. AWS Management Console AWS CLI

주제

- Greengrass 서비스 역할 관리 (콘솔)
- 그린그래스 서비스 역할 (CLI) 관리
- 다음 사항도 참조하십시오.

> **Note**
>
> 서비스 수준 액세스를 승인하는 서비스 역할 외에도 Greengrass 코어 디바이스에 토큰 교환 역할을 할당합니다. 토큰 교환 역할은 코어 디바이스의 Greengrass 구성 요소 및 Lambda 함수가 서비스에 액세스할 수 있는 방법을 제어하는 별도의 IAM 역할입니다. AWS 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

Greengrass 서비스 역할 관리 (콘솔)

AWS IoT 콘솔에서는 Greengrass 서비스 역할을 손쉽게 관리할 수 있습니다. 예를 들어 코어 장치에 대한 클라이언트 장치 검색을 구성하면 AWS 계정 콘솔은 사용자가 현재 Greengrass 서비스 역할에 연결되어 있는지 확인합니다. AWS 리전 연결이 되어 있지 않으면 콘솔이 사용자를 대신하여 서비스 역할을 생성 및 구성할 수 있습니다. 자세한 설명은 the section called “Greengrass 서비스 역할 생성” 섹션을 참조하세요.

다음과 같은 역할 관리 작업에서 콘솔을 사용할 수 있습니다.

주제

- Greengrass 서비스 역할 검색(콘솔)
- Greengrass 서비스 역할 생성(콘솔)
- Greengrass 서비스 역할 변경(콘솔)
- Greengrass 서비스 역할 분리(콘솔)

> **Note**
>
> 콘솔에 로그인한 사용자는 해당 서비스 역할을 보고, 생성하고, 변경할 수 있는 권한을 가져야 합니다.

Greengrass 서비스 역할 검색(콘솔)

다음 단계를 사용하여 현재 AWS 리전 사용 중인 AWS IoT Greengrass 서비스 역할을 찾으십시오.

1. AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 설정을 선택합니다.

3. Greengrass service role(Greengrass 서비스 역할) 섹션으로 스크롤하여 서비스 역할과 그 정책을 확인합니다.

   서비스 역할이 보이지 않는 경우 콘솔에서 자동으로 생성하거나 구성할 수 있습니다. 자세한 설명은 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
명칭	`Greengrass_ServiceRole`
신뢰할 수 있는 엔터티.	`AWS service: greengrass`
정책	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> AWS IoT Greengrass V1기기 설정 스크립트로 이 역할을 생성하는 경우 역할 이름은 입니다`GreengrassServiceRole_`*`random-string`*.

핵심 장치에 대한 클라이언트 장치 검색을 구성하면 콘솔은 Greengrass 서비스 역할이 현재 사용자 AWS 계정 역할과 연결되어 있는지 확인합니다. AWS 리전 연결되지 않은 경우 AWS IoT Greengrass 가 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 부여하는지 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 생성한 경우 콘솔로 돌아가서 해당 역할을 사용자의 AWS IoT AWS 계정 콘솔에 연결하십시오. 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 콘솔로 이동합니다.
2. 탐색 창에서 설정을 선택합니다.
3. 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.
4. 연결할 Greengrass 서비스 역할을 선택합니다.
5. 역할 연결을 선택합니다.

Greengrass 서비스 역할 분리(콘솔)

다음 절차를 사용하여 현재 계정에서 Greengrass 서비스 역할을 AWS 분리하십시오. AWS 리전 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 콘솔로 이동합니다.
2. 탐색 창에서 설정을 선택합니다.
3. Greengrass 서비스 역할에서 분리를 선택합니다.
4. 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.

그린그래스 서비스 역할 (CLI) 관리

다음 절차에서는 AWS Command Line Interface 가 사용자 계정을 사용하도록 설치 및 구성되어 있다고 가정합니다. AWS 계정 자세한 내용은 AWS Command Line Interface사용 설명서의 설치, 업데이트, 제거 AWS CLI 및 구성을 참조하십시오. AWS CLI

다음과 같은 역할 관리 작업에서 AWS CLI을 사용할 수 있습니다.

주제

- Greengrass 서비스 역할 가져오기(CLI)
- Greengrass 서비스 역할 생성(CLI)
- Greengrass 서비스 역할 제거(CLI)

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 greengrassv2 get-service-role-for-account --region region
```

Greengrass 서비스 역할이 이미 계정에 연결되어 있는 경우 요청은 다음 역할 메타데이터를 반환합니다.

```
{
  "associatedAt": "timestamp",
  "roleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

요청에서 역할 메타데이터를 반환하지 않는 경우 서비스 역할을 만들고 (존재하지 않는 경우) 서비스 역할을 만들어 계정과 연결해야 합니다. AWS 리전

Greengrass 서비스 역할 생성(CLI)

다음 단계에 따라 역할을 생성하고 AWS 계정과 연결합니다.

IAM을 사용하여 서비스 역할을 생성하려면

1. 이 역할을 수임하도록 허용하는 AWS IoT Greengrass 신뢰 정책으로 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 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": {
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
           },
   ```

```
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

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

PowerShell

```
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 greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region
```

요청이 성공하면 다음과 같은 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 greengrassv2 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 참조하십시오.

- IAM 사용 설명서의 역할을 만들어 AWS 사용자에게 권한 위임.
- IAM 사용 설명서의 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-계정
  - disassociate-service-role-from-계정
  - get-service-role-for-계정
- AWS CLI 명령 참조에서 사용 가능한 IAM 명령
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

AWS IoT Greengrass의 AWS 관리형 정책

AWS 관리형 정책은 AWS에 의해 생성되고 관리되는 독립 실행형 정책입니다. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있기 때문에 특정 사용 사례에 대해 최소 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에서 정의한 권한은 변경할 수 없습니다. AWS에서 AWS 관리형 정책에 정의된 권한을 업데이트할 경우 정책이 연결되어 있는 모든 보안 주체 엔터티(사용자, 그룹 및 역할)에도 업데이트가 적용됩니다. 새로운 AWS 서비스를 시작하거나 새로운 API 작업을 기존 서비스에 이용하는 경우 AWS가 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

주제

- AWS 관리형 정책: AWSGreengrassFullAccess
- AWS 관리형 정책: AWSGreengrassReadOnlyAccess
- AWS 관리형 정책: AWSGreengrassResourceAccessRolePolicy
- AWS 관리형 정책으로 AWS IoT Greengrass 업데이트

AWS 관리형 정책: AWSGreengrassFullAccess

`AWSGreengrassFullAccess` 정책을 IAM 자격 증명에 연결할 수 있습니다.

이 정책은 보안 주체에게 모든 액세스 권한을 부여하는 관리 권한을 부여합니다.AWS IoT Greengrass 행동.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `greengrass`— 교장이 모든 항목에 대한 전체 액세스 권한 부여AWS IoT Greengrass행동.

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

AWS 관리형 정책: AWSGreengrassReadOnlyAccess

AWSGreengrassReadOnlyAccess 정책을 IAM 자격 증명에 연결할 수 있습니다.

이 정책은 보안 주체가 정보를 볼 수 있지만 수정할 수는 없도록 허용하는 읽기 전용 권한을 부여합니다.AWS IoT Greengrass. 예를 들어 이러한 권한이 있는 보안 주체는 Greengrass 코어 장치에 배포된 구성 요소 목록을 볼 수 있지만 해당 장치에서 실행되는 구성 요소를 변경하기 위한 배포를 만들 수는 없습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- greengrass— 보안 주체가 항목 목록이나 항목에 대한 세부 정보를 반환하는 작업을 수행할 수 있습니다. 여기에는 다음으로 시작하는 API 작업이 포함됩니다.List또는Get.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:List*",
                "greengrass:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: AWSGreengrassResourceAccessRolePolicy

첨부할 수 있습니다AWSGreengrassResourceAccessRolePolicyIAM 엔티티에 대한 정책AWS IoT Greengrass또한 이 정책을 허용하는 서비스 역할에 연결합니다.AWS IoT Greengrass귀하를 대신하여 작업을 수행할 수 있습니다. 자세한 정보는 Greengrass 서비스 역할을 참조하세요.

이 정책은 다음을 허용하는 관리 권한을 부여합니다.AWS IoT GreengrassLambda 함수 검색, 관리와 같은 필수 작업 수행AWS IoT디바이스 섀도우 및 Greengrass 클라이언트 디바이스 확인

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `greengrass`— 그린그래스 리소스를 관리합니다.
- `iot(*Shadow)` — 관리AWS IoT이름에 다음과 같은 특수 식별자가 있는 섀도우 이러한 권한은 다음과 같은 경우에 필요합니다.AWS IoT Greengrass핵심 장치와 통신할 수 있습니다.
  - `*-gci`—AWS IoT Greengrass이 섀도우를 사용하여 코어 디바이스 연결 정보를 저장하므로 클라이언트 디바이스가 코어 디바이스를 검색하고 코어 디바이스에 연결할 수 있습니다.
  - `*-gcm`—AWS IoT Greengrass V1이 섀도우를 사용하여 Greengrass 그룹의 인증 기관 (CA) 인증서가 교체되었음을 코어 디바이스에 알립니다.
  - `*-gda`—AWS IoT Greengrass V1이 섀도우를 사용하여 코어 장치에 배포를 알립니다.
  - `GG_*`— 미사용.
- `iot(DescribeThing과DescribeCertificate)` — 에 대한 정보 검색AWS IoT사물 및 인증서. 이러한 권한은 다음과 같은 경우에 필요합니다.AWS IoT Greengrass코어 장치에 연결되는 클라이언트 장치를 확인할 수 있습니다. 자세한 정보는 로컬 IoT 기기와 상호작용을 참조하세요.
- `lambda`— 에 대한 정보 검색AWS Lambda기능. 이 권한은 다음과 같은 경우에 필요합니다.AWS IoT Greengrass V1람다 함수를 그린그래스 코어에 배포할 수 있습니다. 자세한 내용은 을 참조하십시오.에서 Lambda 함수를 실행합니다.AWS IoT Greengrass핵심에서AWS IoT Greengrass V1개발자 가이드.
- `secretsmanager`— 의 값을 검색합니다.AWS Secrets Manager이름이 로 시작하는 비밀`greengrass-`. 이 권한은 다음과 같은 경우에 필요합니다.AWS IoT Greengrass V1시크릿 매니저 시크릿을 그린그래스 코어에 배포할 수 있습니다. 자세한 내용은 을 참조하십시오.비밀을 다음 위치에 배포하세요AWS IoT Greengrass핵심에서AWS IoT Greengrass V1개발자 가이드.
- `s3`— 이름이 포함된 S3 버킷에서 파일 객체를 검색합니다.`greengrass`또는`sagemaker`. 이러한 권한은 다음과 같은 경우에 필요합니다.AWS IoT Greengrass V1S3 버킷에 저장한 기계 학습 리소스를 배포할 수 있습니다. 자세한 내용은 을 참조하십시오.머신 러닝 리소스에서AWS IoT Greengrass V1개발자 가이드.
- `sagemaker`— 아마존에 대한 정보 검색SageMaker기계 학습 추론 모델. 이 권한은 다음과 같은 경우에 필요합니다.AWS IoT Greengrass V1ML 모델을 그린그래스 코어에 배포할 수 있습니다. 자세한 내용은 을 참조하십시오.머신 러닝 추론 수행에서AWS IoT Greengrass V1개발자 가이드.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGreengrassAccessToShadows",
            "Action": [
                "iot:DeleteThingShadow",
```

```
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
```

```
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
    ]
}
```

AWS 관리형 정책으로 AWS IoT Greengrass 업데이트

업데이트에 대한 세부 정보를 볼 수 있습니다.AWS에 대한 관리형 정책AWS IoT Greengrass이 서비스가 이러한 변경 사항을 추적하기 시작한 시점부터 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 RSS 피드를 구독하십시오.AWS IoT Greengrass V2문서 히스토리 페이지.

변경 사항	설명	날짜
AWS IoT Greengrass에서 변경 사항 추적 시작	AWS IoT Greengrass가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다.	2021년 7월 2일

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.

의 가치`aws:SourceArn`은 (는) 와 연결된 Greengrass 고객 리소스여야 합니다.`sts:AssumeRole`요청.

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::account-id:*`.

를 사용하는 정책의 예를 들어`aws:SourceArn`과`aws:SourceAccount`전역 조건 컨텍스트 키,Greengrass 서비스 역할 생성.

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
- iam을 수행하도록 인증되지 않음:PassRole
- 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 다음과 같은 경우에 발생합니다.`mateojackson`IAM 사용자가 코어 디바이스에 대한 세부 정보를 보려고 하지만, 그렇지 않은 경우`greengrass:GetCoreDevice`권한.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

이 경우 Mateo는 `arn:aws:greengrass:us-west-2:123456789012:coreDevices/MyGreengrassCore` 태스크를 사용하여 `greengrass:GetCoreDevice` 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

작업 시 발생할 수 있는 일반적인 IAM 문제는 다음과 같습니다.AWS IoT Greengrass.

iam을 수행하도록 인증되지 않음:PassRole

`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 사람이AWS리소스입니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 자세한 내용을 알아보려면 다음 섹션을 참조하세요.다른 사람의 한 명의 IAM 사용자에 액세스 권한 제공AWS 계정소유과액세스 권한 제공AWS 계정제3자가 소유하고 있는 경우에서IAM 사용 설명서.

AWS IoT Greengrass는 리소스 기반 정책 또는 ACL(액세스 제어 목록)을 기반으로 하는 교차 계정 액세스를 지원하지 않습니다.

프록시 또는 방화벽을 통한 장치 트래픽 허용

Greengrass 코어 디바이스와 Greengrass 구성 요소는 서비스 및 기타 웹 사이트에 대한 아웃바운드 요청을 수행합니다. AWS 보안 조치로 아웃바운드 트래픽을 작은 범위의 엔드포인트 및 포트로 제한할 수 있습니다. 엔드포인트 및 포트에 대한 다음 정보를 사용하여 프록시, 방화벽 또는 Amazon VPC 보안 그룹을 통한 디바이스 트래픽을 제한할 수 있습니다. 프록시를 사용하도록 코어 디바이스를 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 포트 443에서 또는 네트워크 프록시를 통해 연결

주제

- 기본 작업을 위한 엔드포인트
- 자동 프로비저닝을 통한 설치용 엔드포인트
- 제공된 구성 요소의 엔드포인트 AWS

기본 작업을 위한 엔드포인트

Greengrass 코어 디바이스는 기본 작동을 위해 다음 엔드포인트와 포트를 사용합니다.

엔드포인트를 검색합니다. AWS IoT

AWS IoT엔드포인트를 가져와서 나중에 사용할 수 있도록 저장하세요AWS 계정. 장치는 이러한 엔드포인트를 사용하여 연결합니다. AWS IoT 다음을 따릅니다.

1. 사용자의 AWS IoT 데이터 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 사용자의 AWS IoT 자격 증명 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

엔드포인트	포트	필수	설명
greengrass-ats.iot. *region*.amazonaws.com	8443 또는 443	예	배포 설치 및 클라이언트 장치 사용과 같은 데이터 플레인 작업에 사용됩니다.
device-data-prefix -ats.iot. *region*.amazonaws.com	MQTT: 8883 또는 443 HTTPS: 8443 또는 443	예	MQTT 통신 및 섀도우 동기화와 같은 장치 관리를 위한 데이터 플레인 작업에 사용됩니다. AWS IoT Core
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	예	AWS자격 증명을 획득하는 데 사용됩니다. 이 자격 증명은 코어 디바이스가

엔드포인트	포트	필수	설명
			Amazon S3에서 구성 요소 아티팩트를 다운로드하고 다른 작업을 수행하는 데 사용합니다. 자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

`*.s3.amazonaws.com`
 	
`*.s3.region.amazonaws.com`
	443	예	배포에 사용됩니다. 엔드포인트 접두사는 내부적으로 제어되며 언제든지 변경될 수 있으므로 이 형식에는 * 문자가 포함됩니다.

엔드포인트	포트	필수	설명
data.iot. *region*.amazonaws.com	443	아니요	코어 디바이스가 v2.4.0 이전의 Greengrass nucleus 버전을 실행하고 네트워크 프록시를 사용하도록 구성된 경우 필요합니다. 코어 디바이스는 프록시 뒤에서 MQTT 통신을 위해 이 엔드포인트를 사용합니다. AWS IoT Core 자세한 설명은 네트워크 프록시를 구성하십시오. 섹션을 참조하세요.

자동 프로비저닝을 통한 설치용 엔드포인트

Greengrass 코어 디바이스는 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치할 때 다음 엔드포인트와 포트를 사용합니다.

엔드포인트	포트	필수	설명
`iot.region.amazonaws.com`	443	예	AWS IoT리소스를 생성하고 기존 리소스에 대한 정보를 검색하는 데 사용됩니다. AWS IoT
`iam.amazonaws.com`	443	예	IAM 리소스를 생성하고 기존 IAM 리소스에 대한 정보를 검색하는 데 사용됩니다.
`sts.region.amazonaws.com`	443	예	ID를 가져오는 데 사용됩니다. AWS 계정
`greengrass. region.amazonaws.com`	443	아니요	`--deploy-dev-tools` 인수를 사용하여 Greengras

엔드포인트	포트	필수	설명
			s CLI 구성 요소를 코어 디바이스에 배포하는 경우 필수입니다.

제공된 구성 요소의 엔드포인트 AWS

Greengrass 코어 디바이스는 실행하는 소프트웨어 구성 요소에 따라 추가 엔드포인트를 사용합니다. AWS제공된 각 구성 요소에 필요한 엔드포인트는 이 개발자 안내서의 각 구성 요소 페이지에 있는 요구 사항 섹션에서 찾을 수 있습니다. 자세한 내용은 AWS-제공된 구성 요소을(를) 참조하세요.

AWS IoT Greengrass의 규정 준수 확인

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하고 관심 있는 규정 준수 프로그램을 선택하십시오. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하십시오.

AWS Artifact를 사용하여 서드 파티 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하십시오.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다.AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서 – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS환경을 배포하기 위한 단계를 제공합니다.
- Amazon Web Services에서 HIPAA 보안 및 규정 준수 기술 백서 설계 - 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 방법을 설명합니다.

> **Note**
>
> 모든 AWS 서비스에 HIPAA 자격이 있는 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하십시오.

- AWS 규정 준수 리소스 - 고객 조직이 속한 산업 및 위치에 적용될 수 있는 워크북 및 가이드 컬렉션입니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에서는 AWS 서비스를 보호하기 위한 모범 사례를 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 가이드의 규칙을 사용하여 리소스 평가 – AWS Config 서비스는 내부 사례, 산업 지침 및 규제에 대한 리소스 구성의 준수 상태를 평가합니다.
- AWS Security Hub – 이 AWS 서비스는 AWS내의 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 제어를 사용하여 AWS리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하십시오.
- AWS Audit Manager - 이 AWS 서비스는 AWS 사용을 지속적으로 감사하여 리스크를 관리하고 규정 및 업계 표준을 준수하는 방법을 간소화할 수 있도록 지원합니다.

AWS IoT Greengrass의 복원성

이AWS글로벌 인프라는 Amazon Web Services 리전 및 가용 영역을 중심으로 구축됩니다. EACAWS 리전에서는 에서는 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.

자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 뿐만 아니라 AWS IoT Greengrass도 데이터 복원력과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

- 로컬 파일 시스템 및 에 로그를 기록하도록 Greengrass 코어 디바이스를 구성할 수 있습니다. CloudWatch 로그. 코어 디바이스와 연결이 끊어지면 파일 시스템에 계속 메시지를 기록할 수 있습니다. 다시 연결하면 로그 메시지를 CloudWatch 로그. 자세한 정보는 모니터 AWS IoT Greengrass 로그을 참조하십시오.
- 배포 중에 코어 디바이스의 전원이 손실되는 경우, 배포 후 배포가 재개됩니다.AWS IoT Greengrass 핵심 소프트웨어가 다시 시작됩니다.
- 코어 디바이스와 인터넷 연결이 끊어지면 Greengrass 클라이언트 디바이스는 로컬 네트워크를 통해 계속 통신할 수 있습니다.

- 읽은 Greengrass 구성 요소를 작성할 수 있습니다.스트림 관리자데이터를 스트리밍하고 로컬 스토리지 대상으로 전송합니다.

AWS IoT Greengrass의 인프라 보안

관리형 서비스인 AWS IoT Greengrass는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 AWS IoT Greengrass에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.2 이상을 지원해야 합니다. TLS 1.3 이상을 권장합니다. 클라이언트는 DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)와 같은 PFS(전달 완전 보안)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

요청은 액세스 키 ID 및 IAM 보안 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 Greengrass환경에서 장치는 X.509 인증서와 암호화 키를 사용하여 에 연결하고 인증합니다. AWS 클라우드 자세한 정보는 the section called “디바이스 인증 및 권한 부여” 단원을 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. 자세한 내용은 다음 항목을 참조하십시오.

- 이Device Defender 컴포넌트

- AWS IoT Device Defender의AWS IoT Core개발자 안내서.

AWS IoT Greengrass 환경에서는 다음 고려 사항을 알고 있어야 합니다.

- 물리적 디바이스, 디바이스의 파일 시스템 및 로컬 네트워크를 보호하는 것은 사용자의 책임입니다.
- AWS IoT Greengrass는 Greengrass 컨테이너에서 실행되는지 여부에 관계없이 사용자 정의 Greengrass 구성 요소에 대해 네트워크 격리를 강제하지 않습니다. 따라서 Greengrass 구성 요소가 시스템 또는 네트워크를 통해 외부에서 실행 중인 다른 프로세스와 통신할 수 있습니다.

코드 무결성AWS IoT Greengrass V2

AWS IoT Greengrass에서 소프트웨어 구성 요소를 배포합니다.AWS 클라우드를 실행하는 장치에 AWS IoT Greengrass코어 소프트웨어. 이 소프트웨어 구성 요소는 다음과 같습니다AWS-제공된 구성 요소과커스텀 구성 요소업로드하는 경우AWS 계정. 모든 구성 요소는 레시피로 구성됩니다. 레시피는 컴포넌트의 메타데이터와 컴파일된 코드 및 정적 리소스와 같은 구성 요소 바이너리인 임의의 수의 아티팩트를 정의합니다. 구성 요소 아티팩트는 Amazon S3 저장됩니다.

Greengrass 구성 요소를 개발하고 배포할 때 구성 요소 아티팩트와 함께 작동하는 기본 단계를 수행합니다.AWS 계정디바이스에서 다음을 수행합니다.

1. 아티팩트를 생성하여 S3 버킷에 업로드합니다.
2. 의 레시피 및 아티팩트에서 구성 요소 만들기AWS IoT Greengrass서비스, 다음을 계산하는암호화 해시각 유물 중.
3. Greengrass 핵심 디바이스에 컴포넌트를 배포하여 각 아티팩트의 무결성을 다운로드하고 확인합니다.

AWSGreengrass 코어 디바이스에 컴포넌트를 배포하는 경우를 포함하여 S3 버킷에 아티팩트를 업로드한 후 아티팩트의 무결성을 유지할 책임이 있습니다. S3 버킷에 아티팩트를 업로드하기 전에 소프트웨어 아티팩트를 보호할 책임이 있습니다. 또한 귀하의 리소스에 대한 액세스 보안을 유지할 책임도 있습니다.AWS 계정구성 요소 아티팩트를 업로드하는 S3 버킷을 포함합니다.

> **Note**
>
> Amazon S3 S3는 S3 객체 잠금이라는 기능을 제공하여 S3 버킷의 구성 요소 아티팩트에 대한 변경 사항을 방지하는 데 사용할 수 있습니다.AWS 계정. S3 객체 잠금을 사용하면 구성 요

소 아티팩트가 삭제되거나 덮어쓰지 않도록 할 수 있습니다. 자세한 내용은 단원을 참조하십시오.S3 객체 잠금 사용의아마존 심플 스토리지 서비스 사용 설명서.

일시AWS는 공용 구성 요소를 게시하고 사용자 지정 구성 요소를 업로드하면AWS IoT Greengrass각 구성 요소 아티팩트에 대한 암호화 다이제스트를 계산합니다.AWS IoT Greengrass각 아티팩트의 다이제스트 및 해당 다이제스트를 계산하는 데 사용되는 해시 알고리즘을 포함하도록 구성 요소 레시피를 업데이트합니다. 이 다이제스트는 아티팩트의 무결성을 보장합니다.AWS 클라우드또는 다운로드 중에 파일 다이제스트가 다음 다이제스트와 일치하지 않습니다.AWS IoT Greengrass은 (는) 구성요소 레시피에 저장합니다. 자세한 내용은 단원을 참조하십시오.컴포넌트 레시피 참조의 아티팩트.

구성 요소를 핵심 장치에 배포하는 경우AWS IoT Greengrass핵심 소프트웨어는 컴포넌트 레시피와 레시피가 정의하는 각 컴포넌트 아티팩트를 다운로드합니다. 이AWS IoT Greengrass코어 소프트웨어는 다운로드한 각 아티팩트 파일의 다이제스트를 계산하여 레시피에서 해당 아티팩트의 다이제스트와 비교합니다. 다이제스트가 일치하지 않는 경우 배포가 실패하고AWS IoT Greengrass코어 소프트웨어는 장치의 파일 시스템에서 다운로드한 아티팩트를 삭제합니다. 코어 장치 간 연결 방법에 대한 자세한 내용은AWS IoT Greengrass보안이 보장됩니다.전송 중 데이터 암호화.

핵심 장치의 파일 시스템에서 구성 요소 아티팩트 파일을 보호할 책임은 사용자에게 있습니다. 이AWS IoT Greengrass핵심 소프트웨어는 아티팩트를packagesGreengrass 루트 폴더에 있는 폴더입니다. 이AWS IoT Device Defender핵심 장치를 분석, 감사 및 모니터링합니다. 자세한 내용은 AWS IoT Greengrass의 구성 및 취약성 분석 단원을 참조하세요.

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 서비스의 구성 요소, 배포 및 핵심 장치를 관리할 수 있습니다. AWS IoT Greengrass 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.

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 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 및 와 같은 CreateDeployment작업이 포함됩니다. ListEffectiveDeployments 컨트롤 플레인에는 데이터 플레인 작업인 ResolveComponentCandidates및 Discover와 같은 작업이 포함되지 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:ListEffectiveDeployments"
            ],
            "Resource": "*"
        }
    ]
}
```

VPC에서 AWS IoT Greengrass 코어 디바이스 운영

공용 인터넷 액세스 없이 VPC에서 Greengrass 코어 디바이스를 운영하고 배포를 수행할 수 있습니다. 최소한 해당 DNS 별칭을 사용하여 다음 VPC 엔드포인트를 설정해야 합니다. VPC 엔드포인트를 생성하고 사용하는 방법에 대한 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트 생성을 참조하십시오.

> **Note**
>
> AWS IoT data및 AWS IoT 자격 증명에 대해 DNS 레코드를 자동으로 생성하는 VPC 기능이 비활성화되었습니다. 이러한 엔드포인트를 연결하려면 프라이빗 DNS 레코드를 수동으로 생성해야 합니다. 자세한 내용은 인터페이스 엔드포인트용 프라이빗 DNS를 참조하십시오. VPC 제한에 대한 자세한 내용은 AWS IoT Core VPC 엔드포인트 제한을 참조하십시오.

사전 조건

- 수동 프로비저닝 단계를 사용하여 AWS IoT Greengrass Core 소프트웨어를 설치해야 합니다. 자세한 설명은 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 섹션을 참조하세요.

제한 사항

- VPC에서 Greengrass 코어 디바이스를 운영하는 것은 중국 지역 및 지역에서 지원되지 않습니다. AWS GovCloud (US) Regions
- AWS IoT자격 증명 공급자 VPC 엔드포인트의 AWS IoT data 제한 사항에 대한 자세한 내용은 제한을 참조하십시오.

VPC에서 작동하도록 Greengrass 코어 디바이스를 설정합니다.

1. AWS IoT엔드포인트를 가져와서 나중에 사용할 수 있도록 저장하세요AWS 계정. 장치는 이러한 엔드포인트를 사용하여 연결합니다. AWS IoT 다음을 따릅니다.

   a. 사용자의 AWS IoT 데이터 엔드포인트를 가져오세요. AWS 계정

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

   요청이 성공하면 응답은 다음 예와 비슷합니다.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   b. 사용자의 AWS IoT 자격 증명 엔드포인트를 가져오세요. AWS 계정

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

요청이 성공하면 응답은 다음 예와 비슷합니다.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

2. AWS IoT자격 증명 엔드포인트를 위한 AWS IoT data Amazon VPC 인터페이스를 생성합니다.

   a. VPC 엔드포인트 콘솔로 이동한 다음 왼쪽 메뉴의 Virtual Private Cloud(VPC)에서 엔드포인트를 선택한 다음 엔드포인트 생성을 선택합니다.

   b. 엔드포인트 생성 페이지에서 다음 정보를 지정합니다.

   - 서비스 범주에서 AWS 서비스를 선택합니다.
   - 서비스 이름에 키워드 `iot`를 입력하여 검색합니다. 표시된 `iot` 서비스 목록에서 엔드포인트를 선택합니다.

     AWS IoT Core 데이터 플레인용 VPC 엔드포인트를 생성하는 경우 해당 리전에 대한 AWS IoT Core 데이터 플레인 API 엔드포인트를 선택합니다. 엔드포인트 형식은 `com.amazonaws.region.iot.data`이(가) 될 것입니다.

     AWS IoT Core 자격 증명 공급자용 VPC 엔드포인트를 생성하는 경우 해당 리전의 AWS IoT Core 자격 증명 공급자 엔드포인트를 선택합니다. 엔드포인트 형식은 `com.amazonaws.region.iot.credentials`이(가) 될 것입니다.

     > **Note**
     >
     > 중국 리전의 AWS IoT Core 데이터 플레인용 서비스 이름 형식은 `cn.com.amazonaws.region.iot.data`입니다. 중국 리전에서는 AWS IoT Core 자격 증명 공급자용 VPC 엔드포인트 생성 기능이 지원되지 않습니다.

   - VPC 및 서브넷에서 엔드포인트를 생성하려는 VPC 및 엔드포인트 네트워크를 생성하려는 가용 영역(AZ)을 선택합니다.
   - DNS 이름 활성화에서 이 엔드포인트에 대해 활성화를 선택하지 않도록 합니다. AWS IoT Core 데이터 플레인이나 AWS IoT Core 자격 증명 공급자 모두 아직 프라이빗 DNS 이름을 지원하지 않습니다.

- 보안 그룹에서 엔드포인트 네트워크 인터페이스와 연결하려는 보안 그룹을 선택합니다.
- 선택적으로 태그를 추가하거나 제거할 수 있습니다. 태그는 엔드포인트와 연결하는 데 사용하는 이름-값 페어입니다.

c. VPC 엔드포인트를 생성하려면 엔드포인트 생성을 선택합니다.

3. AWS PrivateLink 엔드포인트를 생성하면 엔드포인트의 세부 정보 탭에 DNS 이름 목록이 표시됩니다. 이러한 DNS 이름 중 하나를 사용하여 프라이빗 호스팅 영역을 구성할 수 있습니다.
4. Amazon S3 엔드포인트를 생성합니다. 자세한 내용은 Amazon S3용 VPC 엔드포인트 생성을 참조하십시오.
5. AWS제공된 Greengrass 구성 요소를 사용하는 경우 추가 엔드포인트 및 구성이 필요할 수 있습니다. 엔드포인트 요구 사항을 보려면 AWS 제공된 구성 요소 목록에서 구성 요소를 선택하고 요구 사항 섹션을 확인하십시오. 예를 들어, 로그 관리자 구성 요소 요구 사항에서는 이 구성 요소가 엔드포인트에 대한 아웃바운드 요청을 수행할 수 있어야 한다고 권장합니다. `logs.region.amazonaws.com`

   자체 구성 요소를 사용하는 경우 종속성을 검토하고 추가 테스트를 수행하여 추가 엔드포인트가 필요한지 확인해야 할 수 있습니다.
6. Greengrass 핵 구성에서는 로 `greengrassDataPlaneEndpoint` 설정해야 합니다. **`iotdata`** 자세한 내용은 Greengrass 핵 구성을 참조하십시오.
7. 해당 `us-east-1` 지역에 있는 경우 Greengrass 핵 **`REGIONAL`** 구성에서 구성 매개변수를 `s3EndpointType` 로 설정하십시오. 이 기능은 그린그래스 핵 버전 2.11.3 이상에서 사용할 수 있습니다.

Example 예: 구성 요소 구성

```
{
"aws.greengrass.Nucleus": {
   "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
     }
   }
}
```

다음 표에는 해당 사용자 지정 프라이빗 DNS 별칭에 대한 정보가 나와 있습니다.

Service	VPC 엔드포인트 서비스 이름	VPC 엔드포인트 유형	사용자 지정 프라이빗 DNS 별칭	참고
AWS IoT data	com.amazonaws. *region*.iot.d	인터페이스	*prefix*-ats.iot. *region*.a s.com	프라이빗 DNS 레코드는 계정의 AWS IoT data 엔드포인트와 일치해야 합니다. aws iot describe-endpoint --endpoint-type iot:Data-ATS
AWS IoT 자격 증명	com.amazonaws. *region*.iot.c entials	인터페이스	*prefix*.c als.iot. s.com	프라이빗 DNS 레코드는 계정 AWS IoT 자격 증명 엔드포인트와 일치해야 합니다aws iot describe-

Service	VPC 엔드포인트 서비스 이름	VPC 엔드포인트 유형	사용자 지정 프라이빗 DNS 별칭	참고
				`endpoint --endpoint-type iot:CredentialProvider .`
Amazon S3	`com.amazonaws.` *region* `.s3`	인터페이스		DNS 레코드는 자동으로 생성됩니다.

AWS IoT Greengrass의 보안 모범 사례

이 주제에는 AWS IoT Greengrass에 대한 보안 모범 사례가 포함되어 있습니다.

가능한 최소 권한 부여

구성 요소를 권한이 없는 사용자로 실행하여 구성 요소에 대한 최소 권한 원칙을 따르세요. 꼭 필요한 경우가 아니면 구성 요소를 루트로 실행해서는 안 됩니다.

IAM 역할에서 최소 권한 세트를 사용하십시오. 사용을 제한하십시오.*에 대한 와일드카드`Action`과`Resource`IAM 정책의 속성 대신, 가능한 경우 한정된 작업과 리소스를 선언하십시오. 최소 권한 및 기타 정책 모범 사례에 대한 자세한 내용은 the section called “정책 모범 사례” 단원을 참조하십시오.

최소 권한 모범 사례는 다음에도 적용됩니다.AWS IoT그린그래스 코어에 적용하는 정책

Greengrass 구성 요소의 자격 증명을 하드코딩하지 마십시오.

사용자 정의 Greengrass 구성 요소의 자격 증명을 하드코딩하지 마십시오. 자격 증명에 대한 보호를 강화하려면 다음을 수행하십시오.

- 다음과 상호 작용하려면AWS서비스, 에서 특정 작업 및 리소스에 대한 권한을 정의하십시오.그린그래스 코어 디바이스 서비스 역할.
- 를 사용하세요시크릿 매니저 컴포넌트자격 증명을 저장하기 위함입니다. 또는 함수가 다음을 사용하는 경우AWSSDK는 기본 자격 증명 공급자 체인의 자격 증명을 사용하십시오.

민감한 정보를 기록하지 않음

자격 증명 및 기타 개인 식별 정보(PII)의 로깅을 방지해야 합니다. 코어 디바이스의 로컬 로그에 액세스하려면 루트 권한과 액세스 권한이 필요하지만 다음 보호 조치를 구현하는 것이 좋습니다. CloudWatch 로그에는 IAM 권한이 필요합니다.

- MQTT 주제 경로에는 민감한 정보를 사용하지 마십시오.
- AWS IoT Core 레지스트리의 디바이스(사물) 이름, 유형 및 속성에 민감한 정보를 사용하지 마십시오.
- 사용자 정의 Greengrass 구성 요소 또는 Lambda 함수에 민감한 정보를 기록하지 마십시오.
- Greengrass 리소스의 이름과 ID에 민감한 정보를 사용하지 마십시오.
  - 코어 디바이스
  - 구성 요소
  - 배포
  - Loggers

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 다음을 참조하십시오.기기 시계를 동기화된 상태로 유지베스트 프랙티스AWS IoT Core 개발자 가이드.

암호 제품군 권장 사항

Greengrass 디폴트는 디바이스에서 사용 가능한 최신 TLS 암호 제품군을 선택합니다. 디바이스에서 레거시 암호 제품군 사용을 비활성화하는 것을 고려해 보십시오. CBC 암호 그룹을 예로 들 수 있습니다.

자세한 내용은 다음을 참조하십시오.Java 암호화 구성.

다음 사항도 참조하세요.

- 의 보안 모범 사례AWS IoT Core에서AWS IoT개발자 가이드
- 산업용 IoT 솔루션을 위한 10가지 보안 골든 룰에 대해사물 인터넷 켜짐AWS공식 블로그

AWS IoT Device TesterAWS IoT GreengrassV2에 사용하기

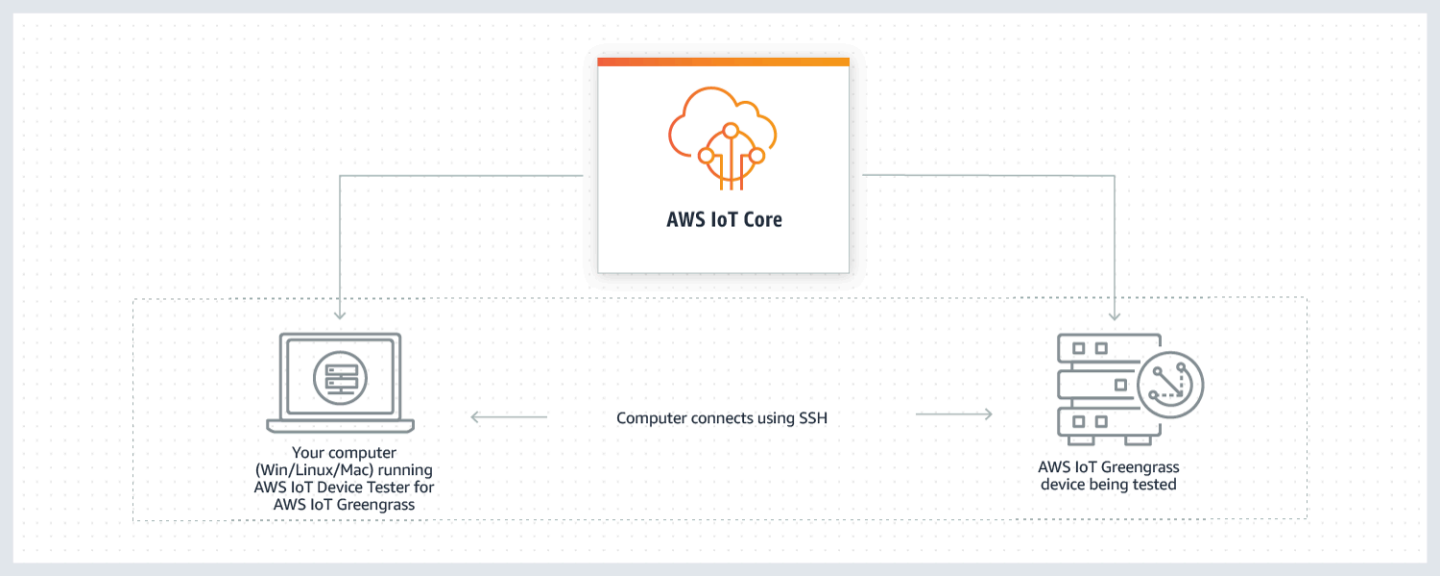
AWS IoT Device Tester(IDT)는 IoT 디바이스를 검증할 수 있는 다운로드 가능한 테스트 프레임워크입니다. IDT AWS IoT Greengrass for를 사용하여 AWS IoT Greengrass 검증 제품군을 실행하고 기기에 맞는 사용자 지정 테스트 스위트를 만들고 실행할 수 있습니다.

AWS IoT Greengrass용 IDT는 테스트 대상 디바이스에 연결된 호스트 컴퓨터(Windows, macOS 또는 Linux)에서 실행됩니다. 이 IDT는 테스트를 실행하고 결과를 집계합니다. 또한 테스트 프로세스를 관리하기 위한 명령줄 인터페이스를 제공합니다.

AWS IoT Greengrass 검증 제품군

AWS IoT Device TesterAWS IoT GreengrassV2용으로 사용하여 AWS IoT Greengrass Core 소프트웨어가 하드웨어에서 실행되고 Core 소프트웨어와 통신할 수 있는지 확인하십시오. AWS 클라우드 또한를 사용하여 end-to-end 테스트를 수행합니다AWS IoT Core. 예를 들어, 장치가 구성 요소를 배포하고 업그레이드할 수 있는지 확인합니다.

AWS Partner 디바이스 카탈로그에 하드웨어를 추가하려면 AWS IoT Greengrass 검증 제품군을 실행하여 AWS IoT에 제출할 수 있는 테스트 보고서를 생성하십시오. 자세한 내용은 AWS 디바이스 검증 프로그램을 참조하십시오.



IDT for AWS IoT Greengrass V2는 테스트 도구 모음과 테스트 그룹의 개념을 사용하여 테스트를 구성합니다.

- 테스트 제품군은 디바이스가 AWS IoT Greengrass의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 구성 요소 배포와 같은 특정 기능과 관련된 개별 테스트 세트입니다.

자세한 설명은 IDT를 사용하여 AWS IoT Greengrass 검증 제품군 실행 섹션을 참조하세요.

사용자 지정 테스트 도구 모음

IDT v4.0.1부터 IDT for AWS IoT Greengrass V2는 표준화된 구성 설정 및 결과 형식을 장치 및 장치 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.

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 Device TesterAWS IoT Greengrass V2용 지원 버전

이 항목에는 V2용 IDT의 지원되는 버전이 나와 있습니다. AWS IoT Greengrass 대상 V2 버전을 지원하는 최신 버전의 AWS IoT Greengrass V2용 IDT를 사용하는 것이 가장 좋습니다. AWS IoT Greengrass 의 새 릴리스를 사용하려면 V2용 AWS IoT Greengrass IDT의 새 버전을 다운로드해야 할 AWS IoT Greengrass 수 있습니다. AWS IoT Greengrass V2용 IDT가 사용 중인 버전과 호환되지 않는 경우 테스트 실행을 시작할 때 알림을 받게 됩니다. AWS IoT Greengrass

소프트웨어를 다운로드하면 AWS IoT Device Tester 사용권 계약에 동의하는 것으로 간주됩니다.

> **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.

V2용 AWS IoT Greengrass 최신 IDT 버전

이 버전의 AWS IoT Greengrass V2용 IDT를 여기에 나열된 AWS IoT Greengrass 버전과 함께 사용할 수 있습니다.

IDT v4.9.2에 대한 AWS IoT Greengrass

지원되는 AWS IoT Greengrass 버전:

- 그린그래스 뉴클리어스 v2.12.0, v2.11.0, v2.10.0 및 v2.9.5

IDT 소프트웨어 다운로드:

- 리눅스용 테스트 스위트 GGV2Q_2.5.2가 포함된 IDT v4.9.2
- macOS용 테스트 스위트 GGV2Q_2.5.2가 포함된 IDT v4.9.2
- IDT v4.9.2 (윈도우용 테스트 스위트 GGV2Q_2.5.2 포함)

릴리스 정보:

- Java 8이 지원 중단되어 Lambda 테스트 스위트가 실패하는 문제를 수정합니다.

추가 참고 사항

- 디바이스에서 HSM을 사용하고 nucleus 2.10.x를 사용하는 경우 Greengrass Nucleus 버전 2.11.0 이상으로 마이그레이션하십시오.

테스트 제품군 버전:

`GGV2Q_2.5.2`

- 2024.03.18이 출시되었습니다.

V2용 미지원 버전 AWS IoT Device TesterAWS IoT Greengrass

이 항목에는 지원되지 않는 V2용 IDT 버전이 나와 있습니다. AWS IoT Greengrass 지원되지 않는 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다. 자세한 설명은 the section called “에 AWS IoT Device Tester 대한 지원 정책 AWS IoT Greengrass” 섹션을 참조하세요.

IDT v4.9.1에 대한 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.12.0, 2.11.0, 2.10.0 및 2.9.5를 실행하는 장치를 검증하고 검증할 수 있습니다.
- 사소한 버그가 수정됨.

테스트 제품군 버전:

`GGV2Q_2.5.1`

- 2023.10.05가 출시되었습니다.

IDT v4.7.0에 대한 AWS IoT Greengrass

지원되는 AWS IoT Greengrass 버전:

- 그린그래스 뉴클리어스 v2.11.0, v2.10.0 및 v2.9.5

릴리스 정보:

- 코어 소프트웨어 버전 2.11.0, 2.10.0 및 2.9.5를 실행하는 장치를 검증하고 검증할 수 있습니다. AWS IoT Greengrass
- AWS Systems Manager Parameter Store에 IDT 사용자 데이터 값을 저장하고 자리 표시자 구문을 사용하여 구성으로 가져올 수 있는 지원을 추가합니다.
- 사소한 버그가 수정됨.

테스트 제품군 버전:

`GGV2Q_2.5.0`

- 2022.12.13이 릴리스되었습니다.

IDT v4.5.11에 대한 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 및 2.6.0을 실행하는 장치를 검증하고 검증할 수 있습니다.
- 코어 기기에서 PreInstalled Greengrass를 테스트하기 위한 지원을 추가합니다.
- 사소한 버그가 수정됨.

테스트 제품군 버전:

`GGV2Q_2.4.1`

- 2022.10.13이 출시되었습니다.

IDT v4.5.8에 대한 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.7.0, 2.6.0 및 2.5.6을 실행하는 장치를 검증하고 검증할 수 있습니다.
- 코어 디바이스에서 PreInstalled Greengrass를 사용하여 테스트할 수 있습니다.
- 사소한 버그가 수정됨.

테스트 제품군 버전:

`GGV2Q_2.4.0`

- 2022.08.12가 출시되었습니다.

IDT v4.5.3에 대한 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.7.0, 2.6.0, 2.5.6, 2.5.5, 2.5.4 및 2.5.3을 실행하는 장치를 검증하고 검증할 수 있습니다.
- ECR 기반 docker 이미지를 사용하도록 DockerApplicationManager 테스트를 업데이트합니다.
- 사소한 버그가 수정됨.

테스트 제품군 버전:

`GGV2Q_2.3.1`

- 2022.04.15가 출시되었습니다.

IDT v4.5.1에 대한 AWS IoT Greengrass

릴리스 정보:

- Core 소프트웨어 v2.5.3을 실행하는 AWS IoT Greengrass 장치를 검증하고 검증할 수 있습니다.
- 하드웨어 보안 모듈 (HSM) 을 사용하여 코어 소프트웨어에서 사용하는 개인 키와 인증서를 저장하는 Linux 기반 장치를 검증하고 검증하기 위한 지원을 추가합니다. AWS IoT Greengrass
- 사용자 지정 테스트 제품군을 구성하기 위한 새로운 IDT 테스트 오케스트레이터를 구현합니다. 자세한 설명은 IDT 테스트 오케스트레이터 구성 섹션을 참조하세요.
- 추가 사소한 버그 수정.

테스트 제품군 버전:

`GGV2Q_2.3.0`

- 2022.01.11이 출시되었습니다.

IDT v4.4.1에 대한 AWS IoT Greengrass

릴리스 정보:

- Core 소프트웨어 v2.5.2를 실행하는 AWS IoT Greengrass 장치를 검증하고 검증할 수 있습니다.
- 사용자 정의 IAM 역할을 테스트 대상 기기가 리소스와 상호 작용하는 것으로 간주하는 토큰 교환 역할로 사용할 수 있도록 지원을 추가합니다. AWS

  파일에서 IAM 역할을 지정할 수 있습니다. `userdata.json` 사용자 지정 역할을 지정하는 경우 IDT는 테스트 실행 중에 기본 토큰 교환 역할을 생성하는 대신 해당 역할을 사용합니다.

- 추가 사소한 버그 수정.

테스트 제품군 버전:

`GGV2Q_2.2.1`

- 2021.12.12가 출시되었습니다.

IDT v4.4.0에 대한 AWS IoT Greengrass

릴리스 정보:

- Core 소프트웨어 v2.5.0을 실행하는 AWS IoT Greengrass 장치를 검증하고 검증할 수 있습니다.
- Windows에서 Core 소프트웨어를 실행하는 장치를 검증하고 검증하기 위한 지원을 추가합니다. AWS IoT Greengrass
- SSH (Secure Shell) 장치 연결을 위한 공개 키 유효성 검사 사용을 지원합니다.
- 보안 모범 사례로 IDT 권한 IAM 정책을 개선합니다.
- 추가 사소한 버그 수정.

테스트 제품군 버전:

`GGV2Q_2.1.0`

- 2021.11.19가 출시되었습니다.

IDT v4.2.0에 대한 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass Core 소프트웨어 v2.2.0 이상 버전을 실행하는 장치에서 다음 기능의 검증을 위한 지원이 포함됩니다.
  - Docker—디바이스가 Amazon Elastic Container 레지스트리 (Amazon ECR) 에서 Docker 컨테이너 이미지를 다운로드할 수 있는지 검증합니다.
  - 기계 학습 - 디바이스가 딥 러닝 런타임 또는 Lite ML 프레임워크를 사용하여 기계 학습 (ML) 추론을 수행할 수 있는지 검증합니다. TensorFlow
  - 스트림 관리자 - 디바이스에서 스트림 관리자를 다운로드, 설치 및 실행할 수 있는지 확인합니다. AWS IoT Greengrass
- AWS IoT Greengrass 코어 소프트웨어 v2.4.0, v2.3.0, v2.2.0 및 v2.1.0을 실행하는 장치를 검증하고 검증할 수 있습니다.
- *# ### #### ### ### #### # ### < > ### ######. test-case-id <device-tester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>*
- 추가 사소한 버그 수정.

테스트 제품군 버전:

`GGV2Q_2.0.1`

- 2021.08.31이 출시되었습니다.

IDT v4.1.0에 대한 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 v2.3.0, v2.2.0, v2.1.0 및 v2.0.5를 실행하는 장치를 검증하고 검증할 수 있습니다.
- 및 속성을 지정해야 하는 요구 사항을 `userdata.json` 제거하여 구성을 개선합니다. `GreengrassNucleusVersion` `GreengrassCLIVersion`
- 코어 소프트웨어 v2.1.0 이상 버전에 대한 Lambda 및 MQTT 기능 AWS IoT Greengrass 검증에 대한 지원이 포함됩니다. 이제 AWS IoT Greengrass V2용 IDT를 사용하여 코어 디바이스에서 Lambda 함수를 실행할 수 있고 해당 디바이스가 MQTT 주제를 게시하고 구독할 수 있는지 검증할 수 있습니다. AWS IoT Core
- 로깅 기능을 개선합니다.
- 추가 사소한 버그 수정.

테스트 제품군 버전:

`GGV2Q_1.1.1`

- 2021.06.18이 출시되었습니다.

IDT v4.0.2에 대한 AWS IoT Greengrass

릴리스 정보:

- Core 소프트웨어 v2.1.0을 실행하는 AWS IoT Greengrass 장치를 검증하고 검증할 수 있습니다.
- 코어 소프트웨어 v2.1.0 이상 버전에 대한 Lambda 및 MQTT 기능 AWS IoT Greengrass 검증에 대한 지원을 추가합니다. 이제 AWS IoT Greengrass V2용 IDT를 사용하여 코어 디바이스에서 Lambda 함수를 실행할 수 있고 해당 디바이스가 MQTT 주제를 게시하고 구독할 수 있는지 검증할 수 있습니다. AWS IoT Core
- 로깅 기능을 개선합니다.
- 추가 사소한 버그 수정.

테스트 제품군 버전:

`GGV2Q_1.1.1`

- 2021.05.05를 출시했습니다.

IDT v4.0.1에 대한 AWS IoT Greengrass

릴리스 정보:

- 버전 2 소프트웨어를 실행하는 AWS IoT Greengrass 장치를 검증하고 검증할 수 있습니다.
- for를 사용하여 AWS IoT Device Tester 사용자 지정 테스트 스위트를 개발하고 실행할 수 있습니다. AWS IoT Greengrass자세한 설명은 IDT를 사용하여 자체 테스트 도구 모음을 개발하고 실행하십시오. 섹션을 참조하세요.
- macOS 및 Windows용 코드 서명 IDT 애플리케이션을 제공합니다. macOS에서는 IDT에 대한 보안 예외를 허용해야 할 수 있습니다. 자세한 설명은 macOS의 보안 예외 섹션을 참조하세요.

테스트 제품군 버전:

`GGV2Q_1.0.0`

- 2020.12.22가 출시되었습니다.
- `value`배열에서 해당 테스트를 로 설정하지 않는 한 테스트 스위트는 검증에 필요한 테스트만 실행합니다. `features` `yes`

V2용 AWS IoT Greengrass IDT 다운로드

이 항목에서는 AWS IoT Device Tester AWS IoT Greengrass V2용 다운로드 옵션에 대해 설명합니다. 다음 소프트웨어 다운로드 링크 중 하나를 사용하거나 지침에 따라 프로그래밍 방식으로 IDT를 다운로드할 수 있습니다.

주제

- 수동으로 IDT 다운로드
- 프로그래밍 방식으로 IDT 다운로드

소프트웨어를 다운로드하면 AWS IoT Device Tester 사용권 계약에 동의하는 것으로 간주됩니다.

> **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.

수동으로 IDT 다운로드

이 항목에는 AWS IoT Greengrass V2용 IDT의 지원되는 버전이 나와 있습니다. 대상 V2 버전을 지원하는 최신 버전의 AWS IoT Greengrass V2용 IDT를 사용하는 것이 가장 좋습니다. AWS IoT Greengrass 의 새 릴리스를 사용하려면 V2용 AWS IoT Greengrass IDT의 새 버전을 다운로드해야 할 AWS IoT Greengrass 수 있습니다. AWS IoT Greengrass V2용 IDT가 사용 중인 버전과 호환되지 않는 경우 테스트 실행을 시작할 때 알림을 받게 됩니다. AWS IoT Greengrass

IDT v4.9.2에 대한 AWS IoT Greengrass

지원되는 AWS IoT Greengrass 버전:

- 그린그래스 뉴클리어스 v2.12.0, v2.11.0, v2.10.0 및 v2.9.5

IDT 소프트웨어 다운로드:

- 리눅스용 테스트 스위트 GGV2Q_2.5.2가 포함된 IDT v4.9.2
- macOS용 테스트 스위트 GGV2Q_2.5.2가 포함된 IDT v4.9.2
- IDT v4.9.2 (윈도우용 테스트 스위트 GGV2Q_2.5.2 포함)

릴리스 정보:

- Java 8이 지원 중단되어 Lambda 테스트 스위트가 실패하는 문제를 수정합니다.

추가 참고 사항

- 디바이스에서 HSM을 사용하고 nucleus 2.10.x를 사용하는 경우 Greengrass Nucleus 버전 2.11.0 이상으로 마이그레이션하십시오.

테스트 제품군 버전:

`GGV2Q_2.5.2`

- 2024.03.18이 출시되었습니다.

프로그래밍 방식으로 IDT 다운로드

IDT는 프로그래밍 방식으로 IDT를 다운로드할 수 있는 URL을 검색하는 데 사용할 수 있는 API 작업을 제공합니다. 또한 이 API 작업을 사용하여 IDT가 최신 버전인지 확인할 수 있습니다. 이 API 작업에는 다음과 같은 엔드포인트가 있습니다.

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

이 API 작업을 직접 호출하려면 **`iot-device-tester:LatestIdt`** 작업을 수행할 수 있는 권한이 있어야 합니다. AWS 서명을 포함하고 서비스 `iot-device-tester` 이름으로 사용하십시오.

API 요청

HostOs — 호스트 컴퓨터의 운영 체제. 다음 옵션 중 하나를 선택합니다.

- `mac`
- `linux`
- `windows`

TestSuiteType — 테스트 스위트의 유형. 다음 옵션을 선택합니다.

`GGV2`— V2용 AWS IoT Greengrass IDT

ProductVersion

(선택 사항) 그린그래스 핵의 버전. 이 서비스는 해당 버전의 Greengrass 핵에 대해 호환되는 최신 버전의 IDT를 반환합니다. 이 옵션을 지정하지 않으면 서비스가 최신 버전의 IDT를 반환합니다.

API 응답

API 응답은 다음 형식을 갖습니다. `DownloadURL`에는 zip 파일이 포함됩니다.

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

예

다음 예제를 참조하여 프로그래밍 방식으로 IDT를 다운로드할 수 있습니다. 이들 예제는 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY` 환경 변수에 저장한 보안 인증 정보를 사용합니다. 보안 모범 사례를 따르려면 코드에 보안 인증 정보를 저장하지 마세요.

Example 예제: cURL 버전 7.75.0 이상을 사용하여 다운로드(Mac 및 Linux)

cURL 버전 7.75.0 이상을 사용하는 경우 `aws-sigv4` 플래그를 사용하여 API 요청에 서명할 수 있습니다. 이 예제에서는 jq를 사용하여 응답에서 다운로드 URL을 파싱합니다.

> **Warning**
>
> `aws-sigv4`플래그를 사용하려면 curl GET 요청의 쿼리 파라미터가 또는 순서여야 합니다. HostOs/ProductVersion/TestSuiteType HostOs/TestSuiteType 이 순서를 준수하지 않으면 API Gateway에서 표준 문자열에 대해 일치하지 않는 서명을 가져오는 오류가 발생합니다. 선택적 매개 ProductVersion 변수가 포함된 경우 AWS IoT Device Tester V2용 AWS IoT Greengrass 지원 버전에 설명된 대로 지원되는 제품 버전을 사용해야 합니다.

- *us-west-2#* 귀하의 것으로 교체하십시오. AWS 리전리전 코드의 목록은 리전 엔드포인트를 참조하세요.
- *linux*를 호스트 시스템의 운영 체제로 바꿉니다.
- *2.5.3#* 현재 사용 중인 핵 버전으로 바꾸십시오. AWS IoT Greengrass

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example 예제: 이전 버전의 cURL을 사용하여 다운로드(Mac 및 Linux)

AWS 서명하고 계산한 서명과 함께 다음 curl 명령을 사용할 수 있습니다. AWS 서명을 서명 및 계산하는 방법에 대한 자세한 내용은 AWS API 요청 서명을 참조하십시오.

- *linux*를 호스트 시스템의 운영 체제로 바꿉니다.
- *Timestamp*를 날짜 및 시간(예: **20220210T004606Z**)으로 바꿉니다.
- *Date*를 날짜(예: **20220210**)로 바꿉니다.
- 내 것으로 *AWSRegion*바꾸십시오 AWS 리전. 리전 코드의 목록은 리전 엔드포인트를 참조하세요.
- 생성한 *AWSSignature AWS* 서명으로 바꾸십시오.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example 예제: Python 스크립트를 사용하여 다운로드

이 예제에서는 Python 요청 라이브러리를 사용합니다. 이 예제는 Python 예제에서 AWS 일반 참조의 AWS API 요청에 서명하도록 수정되었습니다.

- *us-west-2*를 해당 리전으로 바꿉니다. 리전 코드의 목록은 리전 엔드포인트를 참조하세요.
- *linux*를 호스트 시스템의 운영 체제로 바꿉니다.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'

# Key derivation functions. See:
```

```
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
```

```
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
```

```
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

IDT를 사용하여 AWS IoT Greengrass 검증 제품군 실행

AWS IoT Device Tester AWS IoT Greengrass V2용으로 사용하여 Core 소프트웨어가 하드웨어에서 실행되고 AWS IoT Greengrass Core 소프트웨어와 통신할 수 있는지 확인할 수 있습니다. AWS 클라우드또한 를 사용하여 end-to-end 테스트를 수행합니다 AWS IoT Core. 예를 들어, 장치가 구성 요소를 배포하고 업그레이드할 수 있는지 확인합니다.

IDT for AWS IoT Greengrass V2는 장치를 테스트하는 AWS IoT 것 외에도 검증 프로세스를 용이하게 AWS 계정 하기 위한 리소스 (예: 사물, 그룹 등) 를 생성합니다.

이러한 리소스를 생성하기 위해 IDT for AWS IoT Greengrass V2는 `config.json` 파일에 구성된 AWS 자격 증명을 사용하여 사용자를 대신하여 API를 호출합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

IDT for AWS IoT Greengrass V2를 사용하여 AWS IoT Greengrass 검증 제품군을 실행하면 다음 단계가 수행됩니다.

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. 보드에서 검증에 필요한 테스트를 통과했는지를 나타내는 테스트 보고서를 생성합니다.

테스트 제품군 버전

IDT for AWS IoT Greengrass V2는 테스트를 테스트 도구 모음과 테스트 그룹으로 구성합니다.

- 테스트 제품군은 디바이스가 AWS IoT Greengrass의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 구성 요소 배포와 같은 특정 기능과 관련된 개별 테스트 세트입니다.

테스트 스위트는 예를 들어 *major*.*minor*.*patch* 형식을 사용하여 버전이 관리됩니다. `GGV2Q_1.0.0` IDT를 다운로드하면 패키지에 최신 Greengrass 인증 제품군 버전이 포함됩니다.

> **Important**
>
> 지원되지 않는 테스트 제품군 버전의 테스트는 디바이스 검증에 유효하지 않습니다. IDT는 지원되지 않는 버전에 대한 검증 보고서를 인쇄하지 않습니다. 자세한 설명은 the section called "에 AWS IoT Device Tester 대한 지원 정책 AWS IoT Greengrass" 섹션을 참조하세요. 를 `list-supported-products` 실행하여 현재 IDT 버전에서 지원하는 테스트 AWS IoT Greengrass 도구 모음의 버전과 테스트 도구 모음을 나열할 수 있습니다.

테스트 그룹 설명

코어 검증을 위한 필수 테스트 그룹

이러한 테스트 그룹은 AWS IoT Greengrass V2 기기를 기기 카탈로그에 사용할 자격을 부여하는 데 필요합니다. AWS Partner

핵심 종속성

장치가 AWS IoT Greengrass Core 소프트웨어의 모든 소프트웨어 및 하드웨어 요구 사항을 충족하는지 확인합니다. 이 테스트 그룹에는 다음과 같은 테스트 사례가 포함됩니다.

자바 버전

테스트 중인 장치에 필수 Java 버전이 설치되어 있는지 확인합니다. AWS IoT Greengrass Java 8 이상이 필요합니다.

PreTest 유효성 검사

디바이스가 테스트 실행을 위한 소프트웨어 요구 사항을 충족하는지 확인합니다.

- Linux 기반 장치의 경우 이 테스트에서는 장치가 다음 Linux 명령을 실행할 수 있는지 확인합니다.

  `chmod`, `cp`, `echo`, `grep`, `kill`, `ln`, `mkinfo`, `ps`, `rm`, `sh`, `uname`

- Windows 기반 장치의 경우 이 테스트에서는 장치에 다음과 같은 Microsoft 소프트웨어가 설치되어 있는지 확인합니다.

  파워셸 v5.1 이상, .NET v4.6.1 이상, 비주얼 C++ 2017 이상, 유틸리티 PsExec

버전 검사기

AWS IoT Greengrass 제공된 버전이 사용 중인 AWS IoT 디바이스 테스터 버전과 호환되는지 확인합니다.

구성 요소

장치가 구성 요소를 배포하고 업그레이드할 수 있는지 확인합니다. 이 테스트 그룹에는 다음과 같은 테스트가 포함됩니다.

클라우드 구성 요소

클라우드 구성 요소의 장치 기능을 검증합니다.

로컬 구성 요소

로컬 구성 요소의 장치 기능을 검증합니다.

Lambda

이 테스트는 Windows 기반 장치에는 적용할 수 없습니다.

디바이스가 Java 런타임을 사용하는 Lambda 함수 구성 요소를 배포할 수 있는지, Lambda 함수가 MQTT 주제를 작업 메시지의 이벤트 소스로 AWS IoT Core 사용할 수 있는지 검증합니다.

MQTT

디바이스가 MQTT 주제를 구독하고 게시할 수 있는지 검증합니다. AWS IoT Core

선택적 테스트 그룹

> **Note**
>
> 이러한 테스트 그룹은 선택 사항이며 자격을 갖춘 Linux 기반 Greengrass 코어 장치에만 사용됩니다. 선택적 테스트에 적합하도록 선택하면 기기 카탈로그에 추가 기능이 포함된 기기가 나열됩니다. AWS Partner

Docker 종속성

기기가 AWS-제공된 Docker 애플리케이션 관리자 () 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다. `aws.greengrass.DockerApplicationManager`

Docker 애플리케이션 관리자 자격

디바이스가 Amazon ECR에서 Docker 컨테이너 이미지를 다운로드할 수 있는지 검증합니다.

Machine Learning 종속성

기기가 AWS제공된 기계 학습 (ML) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

Machine Learning 추론 테스트

장치가 딥러닝 런타임 및 TensorFlow Lite ML 프레임워크를 사용하여 ML 추론을 수행할 수 있는지 검증합니다.

스트림 관리자 종속성

장치가 AWS IoT Greengrass 스트림 관리자를 다운로드, 설치 및 실행할 수 있는지 확인합니다.

HSI(하드웨어 보안 통합)

> **Note**
>
> 이 테스트는 IDT v4.5.1 이상에서 Linux 기반 장치에만 사용할 수 있습니다. AWS IoT Greengrass 현재 Windows 장치에 대한 하드웨어 보안 통합을 지원하지 않습니다.

장치가 하드웨어 보안 모듈 (HSM) 에 저장된 개인 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 연결을 인증할 수 있는지 확인합니다. 또한 이 테스트에서는 AWS-제공된 PKCS #11 provider 구성 요소가 공급업체에서 제공한 PKCS #11 라이브러리를 사용하여 HSM과 인터페이스할 수 있는지 확인합니다. 자세한 내용은 하드웨어 보안 통합을(를) 참조하세요.

AWS IoT Greengrass 검증 제품군 실행을 위한 사전 요구 사항

이 섹션에서는 AWS IoT Device Tester (IDT) 를 사용하기 위한 사전 요구 사항을 설명합니다. AWS IoT Greengrass

for의 최신 버전을 다운로드하십시오. AWS IoT Device TesterAWS IoT Greengrass

최신 버전의 IDT를 다운로드하고 파일 시스템에서 읽기/쓰기 권한이 있는 위치 (*< device-tester-extract-location >*) 에 소프트웨어를 추출하십시오.

> **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.

소프트웨어 다운로드 AWS IoT Greengrass

IDT for AWS IoT Greengrass V2는 장치가 특정 버전과의 호환성을 테스트합니다. AWS IoT Greengrass 다음 명령을 실행하여 AWS IoT Greengrass Core 소프트웨어를 라는 `aws.greengrass.nucleus.zip` 파일에 다운로드합니다. *### ## ##* IDT 버전에서 지원되는 nucleus 구성 요소 버전으로 교체하십시오.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip -
OutFile aws.greengrass.nucleus.zip
```

다운로드한 `aws.greengrass.nucleus.zip` 파일을 폴더에 넣습니다. *<device-tester-extract-location>*`/products/`

> **Note**
>
> 동일한 운영 체제 및 아키텍처에 대해 이 디렉터리에 여러 파일을 배치하지 마십시오.

AWS 계정 생성 및 구성

AWS IoT Device TesterAWS IoT GreengrassV2용으로 사용하려면 먼저 다음 단계를 수행해야 합니다.

1. AWS 계정을 설정합니다. 이미 AWS 계정이 있으면 2단계로 건너뜁니다.
2. IDT에 대한 권한을 구성합니다.

이러한 계정 권한을 통해 IDT는 사용자를 대신하여 AWS 서비스에 액세스하고 AWS IoT 사물 및 AWS IoT Greengrass 구성 요소와 같은 AWS 리소스를 생성할 수 있습니다.

이러한 리소스를 생성하기 위해 IDT for AWS IoT Greengrass V2는 `config.json` 파일에 구성된 AWS 자격 증명을 사용하여 사용자를 대신하여 API를 호출합니다. 이러한 리소스는 테스트 중 다양한 시점에서 프로비저닝됩니다.

> **Note**
>
> 대부분의 테스트가 AWS프리 티어에 해당되지만 등록할 때 신용 카드를 제공해야 합니다. AWS 계정 자세한 내용은 계정에 프리 티어가 적용되는데 결제 방법이 필요한 이유는 무엇입니까?를 참조하십시오.

1단계: 설정 AWS 계정

이 단계에는 AWS 계정을 생성하고 구성합니다. 이미 AWS 계정이 있으면 the section called “2단계: IDT에 대한 권한 구성” 단계로 건너뜁니다.

AWS 계정가 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입하려면

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

   가입 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정에 가입하면 AWS 계정 루트 사용자 항목이 생성됩니다. 루트 사용자에게 계정의 모든 AWS 서비스 및 리소스에 대한 액세스 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

다음 옵션 중 하나를 선택하여 관리 사용자를 생성합니다.

관리자를 관리하는 방법 한 가지 선택	목적	By	다른 방법
IAM Identity Center에서 (권장)	단기 보안 인증 정보를 사용하여 AWS에 액세스합니다. 이는 보안 모범 사례와 일치합니다. 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.	AWS IAM Identity Center 사용 설명서의 시작하기 지침을 따르세요.	AWS Command Line Interface 사용 설명서의 AWS IAM Identity Center 사용할 AWS CLI 구성을 통해 프로그래밍 방식의 액세스를 구성합니다.
IAM에서 (권장되지 않음)	장기 보안 인증 정보를 사용하여 AWS에 액세스합니다.	IAM 사용 설명서의 첫 IAM 관리 사용자 및 사용자 그룹 만들기에 나온 지침을 따릅니다.	IAM 사용 설명서에 나온 IAM 사용자의 액세스 키 관리에서 프로그래밍 방식 액세스를 구성합니다.

2단계: IDT에 대한 권한 구성

이 단계에서는 IDT for AWS IoT Greengrass V2에서 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. AWS Management Consoleor AWS Command Line Interface(AWS CLI) 를 사용하여 IAM 정책을 생성하고 IDT용 테스트 사용자를 만든 다음 정책을 사용자에게 연결할 수 있습니다. IDT용 테스트 사용자를 이미 생성한 경우 으로 건너뛰십시오. IDT 테스트를 실행하도록 디바이스 구성

IDT에 대한 권한을 구성하려면(콘솔)

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. 사용하지 PreInstalled 않는 경우 JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꾸십시오. 를 사용하는 PreInstalled 경우 다음 단계로 진행하십시오.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
```

```
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
```

```
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
```

```
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

c. 를 사용하는 PreInstalled 경우 JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꾸십시오. 반드시 다음 사항을 확인하세요.

- `iotResources`명령문의 *`ThingName`* 및 *`ThingGroup#`* 테스트 대상 장치 (DUT) 에 Greengrass를 설치하는 동안 생성된 사물 이름 및 사물 그룹으로 바꾸어 권한을 추가합니다.
- *`#### #### PassRole # RoleAlias# DUT# roleAliasResources passRoleForResources Greengrass# #### ## ### ### #####.`*

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
```

```
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
```

```
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
```

```
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
```

```
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> 사용자 지정 IAM 역할을 테스트 대상 디바이스의 토큰 교환 역할로 사용하려면 정책의 명령문과 `roleAliasResources` 설명문을 업데이트하여 사용자 지정 IAM 역할 리소스를 허용해야 합니다. `passRoleForResources`

d. 정책 검토를 선택합니다.

e. 이름에서 **`IDTGreengrassIAMPermissions`**을 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.

f. 정책 생성을 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. Set permissions(권한 설정) 페이지에서 Attach existing policies to user directly(사용자에게 직접 기존 정책 연결)를 선택합니다.

      ii. 이전 단계에서 만든 IDTGreengrassIAMPermissions 정책을 검색합니다. 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 AWS 자격 증명 파일을 구성합니다.

4. 다음 단계: 물리적 디바이스를 구성합니다.

IDT에 대한 권한을 구성하려면(AWS CLI)

1. 컴퓨터에 AWS CLI를 설치하고 구성합니다(아직 설치되어 있지 않은 경우). AWS Command Line Interface 사용 설명서의 AWS CLI 설치 단계를 따르세요.

> **Note**
>
> AWS CLI는 명령줄 쉘에서 AWS 서비스와 상호 작용하는 데 사용할 수 있는 오픈 소스 도구입니다.

2. IDT 및 AWS IoT Greengrass 역할을 관리할 수 있는 권한을 부여하는 고객 관리형 정책을 만듭니다.

a. 를 사용하지 PreInstalled 않는 경우 텍스트 편집기를 열고 다음 정책 내용을 JSON 파일에 저장하십시오. 를 사용하는 PreInstalled 경우 다음 단계로 진행하십시오.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
```

```
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
```

```
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
```

```
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
```

```
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

b. 를 사용하는 PreInstalled 경우 텍스트 편집기를 열고 다음 정책 내용을 JSON 파일에 저장합니다. 반드시 다음 사항을 확인하세요.

- 테스트 대상 장치 (DUT) 에 Greengrass를 설치하는 동안 생성된 `iotResources` 명령문에서 *`ThingName# ThingGroup#`* 교체하여 권한을 추가합니다.
- *`#### #### PassRole # RoleAlias# DUT# roleAliasResources passRoleForResources Greengrass# #### ## ### ### #####.`*

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
```

```
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
```

```
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
```

```
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> 사용자 지정 IAM 역할을 테스트 대상 디바이스의 토큰 교환 역할로 사용하려면 정책의 명령문과 `roleAliasResources` 설명문을 업데이트하여 사용자 지정 IAM 역할 리소스를 허용해야 합니다. `passRoleForResources`

c. 다음 명령어를 실행하여 이라는 고객 관리형 정책을 생성합니다. `IDTGreengrassIAMPermissions` 이전 단계에서 만든 JSON 파일의 전체 *`policy.json`* 경로로 바꾸십시오.

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document file://policy.json
```

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. 명령의 *`<account-id>`*를 AWS 계정의 ID로 바꿉니다.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
 arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

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 디바이스를 구성합니다.

AWS IoT Device Tester 권한

다음 정책은 AWS IoT Device Tester 권한에 대해 설명합니다.

AWS IoT Device Tester버전 확인 및 자동 업데이트 기능에 이러한 권한이 필요합니다.

- `iot-device-tester:SupportedVersion`

  지원되는 제품, 테스트 제품군 및 IDT 버전의 목록을 가져올 수 있는 권한을 AWS IoT Device Tester에 부여합니다.

- `iot-device-tester:LatestIdt`

  다운로드할 수 있는 최신 IDT 버전을 가져올 수 있는 권한을 AWS IoT Device Tester에 부여합니다.

- `iot-device-tester:CheckVersion`

  IDT, 테스트 제품군 및 제품의 버전 호환성을 확인할 수 있는 권한을 AWS IoT Device Tester에 부여합니다.

- `iot-device-tester:DownloadTestSuite`

  테스트 도구 모음 업데이트를 다운로드할 수 있는 AWS IoT Device Tester 권한을 부여합니다.

AWS IoT Device Tester또한 선택적 지표 보고에 다음 권한을 사용합니다.

- `iot-device-tester:SendMetrics`

  AWS IoT Device Tester내부 사용에 대한 지표를 AWS 수집할 수 있는 권한을 부여합니다. 이 권한을 생략하면 이러한 지표는 수집되지 않습니다.

IDT 테스트를 실행하도록 디바이스 구성

IDT에서 장치 검증 테스트를 실행할 수 있도록 하려면 장치에 액세스하도록 호스트 컴퓨터를 구성하고 장치에 대한 사용자 권한을 구성해야 합니다.

호스트 컴퓨터에 Java를 설치합니다.

IDT v4.2.0부터 선택적 검증 테스트를 AWS IoT Greengrass 실행하려면 Java를 실행해야 합니다.

Java 버전 8 이상을 사용할 수 있습니다. Amazon Corretto 또는 OpenJDK 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.

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 설명은 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 사용

합니다. 이 명령은 테스트 대상 디바이스에서 `~/.ssh/authorized_keys` 파일에 퍼블릭 키를 추가합니다. 예:

$ ssh-copy-id *<remote-ssh-user>*@*<remote-device-ip>*

테스트 대상 디바이스에 로그인하는 데 사용되는 사용자 이름은 어디에 *remote-ssh-user*있고, *remote-device-ip*은 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 설명은 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.

Windows 장치의 사용자 자격 증명을 구성합니다.

Windows 기반 장치를 검증하려면 테스트 대상 장치의 LocalSystem 계정에서 다음 사용자에 대한 사용자 자격 증명을 구성해야 합니다.

- 기본 Greengrass 사용자 () `ggc_user`
- 테스트 대상 기기에 연결하는 데 사용하는 사용자. `device.json`파일에서 이 사용자를 구성합니다.

테스트 대상 장치의 LocalSystem 계정에서 각 사용자를 만든 다음 해당 LocalSystem 계정의 Credential Manager 인스턴스에 해당 사용자의 사용자 이름과 암호를 저장해야 합니다.

Windows 장치에서 사용자를 구성하려면

1. Windows 명령 프롬프트 (`cmd.exe`) 를 관리자 권한으로 엽니다.
2. Windows 디바이스의 LocalSystem 계정에 사용자를 생성합니다. 만들려는 각 사용자에 대해 다음 명령을 실행합니다. *`## Greengrass #### ## ### ### # #####.`* `ggc_user` *`##### ## ####`* 대체하십시오.

   ```
   net user /add user-name password
   ```

3. Microsoft에서 PsExec유틸리티를 다운로드하여 장치에 설치합니다.
4. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다.

   자격 증명 관리자에서 구성하려는 각 사용자에 대해 다음 명령을 실행합니다. *`## Greengrass #### ## ### ### # #####.`* `ggc_user` *`##### ###`* 설정한 사용자 비밀번호로 대체합니다.

   ```
   psexec -s cmd /c cmdkey /generic:user-name /user:user-name /pass:password
   ```

가 PsExec License Agreement열리면 라이선스에 Accept동의하도록 선택하고 명령을 실행합니다.

> **Note**
>
> Windows 장치에서는 LocalSystem 계정이 Greengrass nucleus를 실행하므로 PsExec 유틸리티를 사용하여 계정에 사용자 정보를 저장해야 합니다. LocalSystem 자격 증명 관리자 애플리케이션을 사용하면 계정 대신 현재 로그온한 사용자의 Windows 계정에 이 정보가 저장됩니다. LocalSystem

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 자동화하려면 AWS IoT Greengrass V2용 IDT에서 암호를 입력하라는 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
   ```

사용자 지정 토큰 교환 역할을 구성하세요.

사용자 지정 IAM 역할을 테스트 대상 기기가 리소스와 상호 작용하는 것으로 가정하는 토큰 교환 역할로 AWS 사용할 수 있습니다. IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 역할 생성 단원을 참조하세요.

IDT가 사용자 지정 IAM 역할을 사용하도록 허용하려면 다음 요구 사항을 충족해야 합니다. 이 역할에는 필요한 최소 정책 조치만 추가하는 것이 좋습니다.

- 매개변수를 로 설정하려면 userdata.json 구성 파일을 업데이트해야 합니다. `GreengrassV2TokenExchangeRole` `true`
- 사용자 지정 IAM 역할은 다음과 같은 최소 신뢰 정책으로 구성되어야 합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "credentials.iot.amazonaws.com",
               "lambda.amazonaws.com",
               "sagemaker.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

- 사용자 지정 IAM 역할은 다음과 같은 최소 권한 정책으로 구성해야 합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
```

```
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- 사용자 지정 IAM 역할의 이름은 테스트 사용자의 IAM 권한에 지정하는 IAM 역할 리소스와 일치해야 합니다. 기본적으로 테스트 사용자 정책은 역할 이름에 `idt-` 접두사가 있는 IAM 역할에 대한 액세스를 허용합니다. IAM 역할 이름에 이 접두사를 사용하지 않는 경우 다음 예와 같이 테스트 사용자 정책의 명령문 및 `passRoleForResources` 명령문에 `arn:aws:iam::*:role/`*`custom-iam-role-name`* 리소스를 추가하십시오. `roleAliasResources`

  Example **`passRoleForResources`** 명령문

```
{
   "Sid":"passRoleForResources",
   "Effect":"Allow",
   "Action":"iam:PassRole",
   "Resource":"arn:aws:iam::*:role/custom-iam-role-name",
   "Condition":{
      "StringEquals":{
         "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
         ]
      }
   }
```

```
}
```

Example **`roleAliasResources`** 명령문

```
{
   "Sid":"roleAliasResources",
   "Effect":"Allow",
   "Action":[
      "iot:CreateRoleAlias",
      "iot:DescribeRoleAlias",
      "iot:DeleteRoleAlias",
      "iot:TagResource",
      "iam:GetRole"
   ],
   "Resource":[
      "arn:aws:iot:*:*:rolealias/idt-*",
      "arn:aws:iam::*:role/custom-iam-role-name"
   ]
}
```

선택적 기능을 테스트하도록 디바이스 구성

이 섹션에서는 선택적 Docker 및 기계 학습 (ML) 기능에 대한 IDT 테스트를 실행하기 위한 기기 요구 사항을 설명합니다. 이러한 기능을 테스트하려는 경우에만 기기가 이러한 요구 사항을 충족하는지 확인해야 합니다. 그렇지 않은 경우 the section called “IDT 설정 구성”를 계속 진행합니다.

주제

- Docker 자격 요구 사항
- ML 자격 요구 사항
- HSM 자격 요구 사항

Docker 자격 요구 사항

AWS IoT GreengrassV2용 IDT는 사용자 지정 Docker 컨테이너 구성 요소를 사용하여 사용자 지정 Docker 컨테이너 구성 요소를 사용하여 실행할 수 있는 Docker 컨테이너 이미지를 장치에서 다운로드할 수 있는지 확인하는 Docker 검증 테스트를 제공합니다. AWS 사용자 지정 Docker 구성 요소를 만드는 방법에 대한 자세한 내용은 을 참조하십시오. 도커 컨테이너 실행

Docker 검증 테스트를 실행하려면 테스트 대상 장치가 다음 요구 사항을 충족하여 Docker 애플리케이션 관리자 구성 요소를 배포해야 합니다.

- 도커 엔진 1.9.1 이상이 그린그래스 코어 디바이스에 설치되었습니다. 버전 20.10은 Core 소프트웨어와 함께 작동하는 것으로 검증된 최신 버전입니다. AWS IoT Greengrass Docker 컨테이너를 실행하는 구성 요소를 배포하려면 먼저 코어 디바이스에 Docker를 직접 설치해야 합니다.
- Docker 데몬은 이 구성 요소를 배포하기 전에 코어 디바이스에서 시작되어 실행되었습니다.
- Docker 컨테이너 구성 요소를 실행하는 시스템 사용자에게 루트 또는 관리자 권한이 있거나 루트 또는 관리자가 아닌 사용자로 실행하도록 Docker를 구성해야 합니다.
  - Linux 디바이스에서는 `docker` 그룹에 사용자를 추가하여 없이 명령을 `docker` 호출할 수 있습니다. `sudo`
  - Windows 장치에서는 관리자 권한 없이 `docker` 명령을 호출할 사용자를 `docker-users` 그룹에 추가할 수 있습니다.

  Linux or Unix

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 루트가 아닌 사용자를 `docker` 그룹에 추가하려면 `ggc_user` 다음 명령어를 실행합니다.

  ```
  sudo usermod -aG docker ggc_user
  ```

  자세한 내용은 루트가 아닌 사용자로 Docker 관리를 참조하십시오.

  Windows Command Prompt (CMD)

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 사용자를 `docker-users` 그룹에 추가하려면 `ggc_user` 관리자로 다음 명령을 실행합니다.

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자 또는 사용자를 `docker-users` 그룹에 추가하려면 `ggc_user` 관리자로 다음 명령을 실행합니다.

  ```
  Add-LocalGroupMember -Group docker-users -Member ggc_user
  ```

ML 자격 요구 사항

IDT for AWS IoT Greengrass V2는 장치가 AWS 제공된 기계 학습 구성 요소를 사용하여 딥 러닝 런타임 또는 TensorFlow Lite ML 프레임워크를 사용하여 로컬에서 ML 추론을 수행할 수 있는지 검증하는 ML 자격 테스트를 제공합니다. Greengrass 디바이스에서 ML 추론을 실행하는 방법에 대한 자세한 내용은 을 참조하십시오. 기계 학습 추론 수행

ML 자격 테스트를 실행하려면 테스트 대상 장치가 다음 요구 사항을 충족하여 기계 학습 구성 요소를 배포해야 합니다.

- Amazon Linux 2 또는 우분투 18.04를 실행하는 Greengrass 코어 디바이스의 경우 GNU C 라이브러리 (glibc) 버전 2.27 이상이 디바이스에 설치되어 있습니다.
- 라즈베리 파이와 같은 ARMv7L 디바이스에서는 OpenCV-Python에 대한 종속성이 디바이스에 설치되어 있습니다. 다음 명령을 실행하여 종속 항목을 설치합니다.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- 라즈베리파이 OS Bullseye를 실행하는 라즈베리파이 디바이스는 다음 요구사항을 충족해야 합니다.
  - NumPy 1.22.4 이상이 디바이스에 설치되었습니다. Raspberry Pi OS Bullseye에는 이전 버전이 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 업그레이드할 수 있습니다. NumPy NumPy

    ```
    pip3 install --upgrade numpy
    ```

  - 기기에서 레거시 카메라 스택이 활성화되었습니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되고 호환되지 않는 새로운 카메라 스택이 포함되어 있으므로 레거시 카메라 스택을 활성화해야 합니다.

    레거시 카메라 스택을 활성화하려면

    1. 다음 명령을 실행하여 Raspberry Pi 구성 도구를 엽니다.

       ```
       sudo raspi-config
       ```

    2. 인터페이스 옵션을 선택합니다.
    3. 레거시 카메라를 선택하여 레거시 카메라 스택을 활성화합니다.
    4. Raspberry Pi를 재부팅합니다.

HSM 자격 요구 사항

AWS IoT Greengrass장치의 PKCS 하드웨어 보안 모듈 (HSM) 과 통합할 수 있는 PKCS #11 공급자 구성 요소를 제공합니다. HSM 설정은 장치 및 선택한 HSM 모듈에 따라 다릅니다. IDT 구성 설정에 설명된 대로 예상 HSM 구성이 제공되는 한, IDT는 이 선택적 기능 검증 테스트를 실행하는 데 필요한 정보를 보유하게 됩니다.

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.

AWSconfig.json에서 자격 증명을 구성합니다.

<device_tester_extract_location>/configs/config.json 파일에서 IAM 사용자 보안 인증을 구성해야 합니다. 에서 만든 V2용 IDT 사용자의 자격 증명을 사용하십시오. AWS IoT Greengrass the section called “AWS 계정 생성 및 구성” 두 가지 방법 중 하나로 자격 증명을 지정할 수 있습니다.

- 보안 인증 파일에서
- 환경 변수로

보안 인증 파일을 사용하여 AWS 보안 인증 정보 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>
```

credentials파일의 AWS 자격 증명을 사용하도록 AWS IoT Greengrass V2용 IDT를 구성하려면 다음과 같이 config.json 파일을 편집하십시오.

```
{
  "awsRegion": "region",
  "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 GreengrassV2용 IDT는 AWS_ACCESS_KEY_ID 및 AWS_SECRET_ACCESS_KEY 환경 변수를 사용하여 자격 증명을 저장할 수 있습니다. AWS

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": "region",
  "auth": {
    "method": "environment"
  }
```

```
}
```

device.json 구성

IDT for AWS IoT Greengrass V2에는 AWS 자격 증명 외에도 테스트가 실행되는 기기에 대한 정보가 필요합니다. 예제 정보로는 IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처 등이 있습니다.

<device_tester_extract_location>/configs/device.json에 있는 device.json 템플릿을 사용하여 이 정보를 제공해야 합니다.

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
        "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
```

```
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "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.

다음 설명과 같이 값이 포함된 모든 속성이 필요합니다.

`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 적격 보드를 추적하는 데 사용됩니다.

> **Note**
>
> 기기 카탈로그에 기기를 리스팅하려면 여기에 지정하는 SKU가 리스팅 프로세스에서 사용하는 SKU와 일치해야 합니다. AWS Partner

`features`

디바이스의 지원되는 기능이 포함된 배열입니다. 모든 기능이 필요합니다.

`arch`

테스트 실행에서 검증한 지원되는 운영 체제 아키텍처. 유효한 값은 다음과 같습니다.

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`ml`

기기가 AWS 제공된 기계 학습 (ML) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

또한 이 기능을 활성화하면 기기가 딥러닝 런타임 및 TensorFlow Lite ML 프레임워크를 사용하여 ML 추론을 수행할 수 있는지도 검증됩니다.

유효한 값은 `dlr` 및 또는 의 모든 조합입니다. `tensorflowlite` `no`

`docker`

기기가 AWS -제공된 Docker 애플리케이션 관리자 () 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다. `aws.greengrass.DockerApplicationManager`

이 기능을 활성화하면 디바이스가 Amazon ECR에서 Docker 컨테이너 이미지를 다운로드할 수 있는지도 검증됩니다.

유효한 값은 또는 의 모든 조합입니다. `yes` `no`

`streamManagement`

장치가 AWS IoT Greengrass스트림 관리자를 다운로드, 설치 및 실행할 수 있는지 확인합니다.

유효한 값은 `yes` 또는 `no` 의 모든 조합입니다.

`hsi`

장치가 하드웨어 보안 모듈 (HSM) 에 저장된 개인 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 연결을 인증할 수 있는지 확인합니다. 또한 이 테스트에서는

AWS -제공된 PKCS #11 provider 구성 요소가 공급업체에서 제공한 PKCS #11 라이브러리를 사용하여 HSM과 인터페이스할 수 있는지 확인합니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요.

유효한 값은 hsm 또는 no입니다.

> **Note**
>
> IDT v4.2.0 이상 버전은, 및 기능 테스트를 지원합니다. ml docker streamManagement 이러한 기능을 테스트하지 않으려면 해당 값을 로 설정하십시오. no

> **Note**
>
> IDT v4.5.1 이상 버전에서만 테스트를 수행할 수 있습니다. hsi

`devices.id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`devices.operatingSystem`

기기 운영 체제. 지원되는 값은 Linux 및 Windows입니다.

`connectivity.protocol`

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 현재 지원되는 유일한 값은 물리적 ssh 장치용입니다.

`connectivity.ip`

테스트 대상 디바이스의 IP 입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항으로, SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 22입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.publicKeyPath`

선택 사항으로, 테스트 대상 디바이스에 대한 연결을 인증하는 데 사용되는 퍼블릭 키의 전체 경로입니다.

`publicKeyPath`를 지정하면 IDT가 테스트 대상 디바이스에 SSH 연결을 설정할 때 디바이스의 퍼블릭 키를 검증합니다. 이 값을 지정하지 않으면 IDT는 SSH 연결을 생성하지만 디바이스의 퍼블릭 키를 확인하지는 않습니다.

퍼블릭 키의 경로를 지정하고 안전한 방법을 사용하여 이 퍼블릭 키를 가져오는 것이 좋습니다. 표준 명령줄 기반 SSH 클라이언트의 경우 퍼블릭 키는 `known_hosts` 파일에 제공됩니다. 별도의 퍼블릭 키 파일을 지정하는 경우 이 파일은 `known_hosts` 파일과 동일한 형식, 즉, *`ip-address key-type public-key`*을 사용해야 합니다. 동일한 IP 주소를 가진 항목이 여러 개 있는 경우 IDT에서 사용하는 키 유형 항목이 파일의 다른 항목 앞에 있어야 합니다.

`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.

userdata.json을 구성합니다.

AWS IoT GreengrassV2용 IDT에는 테스트 아티팩트 및 소프트웨어 위치에 대한 추가 정보도 필요합니다. AWS IoT Greengrass

<device_tester_extract_location>/configs/userdata.json에 있는 userdata.json 템플릿을 사용하여 이 정보를 제공해야 합니다.

```
{
    "TempResourcesDirOnDevice": "/path/to/temp/folder",
    "InstallationDirRootOnDevice": "/path/to/installation/folder",
    "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
    "PreInstalled": "yes/no",
    "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
 "hsm": {
        "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
        "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
        "slotId": "slot-id",
        "slotLabel": "slot-label",
        "slotUserPin": "slot-pin",
        "keyLabel": "key-label",
        "preloadedCertificateArn": "certificate-arn"
        "rootCA": "path/to/root-ca"
    }
}
```

다음 설명과 같이 값이 포함된 모든 속성이 필요합니다.

`TempResourcesDirOnDevice`

테스트 아티팩트를 저장할 테스트 대상 기기의 임시 폴더의 전체 경로입니다. 이 디렉터리에 쓰는 데 sudo 권한이 필요하지 않은지 확인하세요.

> **Note**
>
> IDT는 테스트 실행이 끝나면 이 폴더의 콘텐츠를 삭제합니다.

`InstallationDirRootOnDevice`

설치할 기기에 있는 폴더의 전체 경로입니다. AWS IoT Greengrass PreInstalled Greengrass의 경우 이 경로는 그린그래스 설치 디렉토리의 경로입니다.

이 폴더에 필요한 파일 권한을 설정해야 합니다. 설치 경로의 각 폴더에 대해 다음 명령을 실행합니다.

```
sudo chmod 755 folder-name
```

`GreengrassNucleusZip`

호스트 컴퓨터에 있는 그린그래스 핵 ZIP (`greengrass-nucleus-latest.zip`) 파일의 전체 경로입니다. PreInstalledGreengrass를 사용한 테스트에는 이 필드가 필요하지 않습니다.

> **Note**
>
> IDT용 Greengrass 뉴클레우스의 지원되는 버전에 대한 자세한 내용은 을 참조하십시오. AWS IoT Greengrass V2용 AWS IoT Greengrass 최신 IDT 버전 최신 Greengrass 소프트웨어를 다운로드하려면 소프트웨어 다운로드를 AWS IoT Greengrass 참조하십시오.

`PreInstalled`

이 기능은 Linux 디바이스의 IDT v4.5.8 이상 버전에서만 사용할 수 있습니다.

(선택 사항) 값이 *yes#* 경우 IDT는 에서 `InstallationDirRootOnDevice` 언급한 경로를 Greengrass가 설치된 디렉토리로 간주합니다.

장치에 Greengrass를 설치하는 방법에 대한 자세한 내용은 을 참조하십시오. 자동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 수동 프로비저닝으로 설치하는 경우 사물을 수동으로 생성할 때 “새 AWS IoT 사물 그룹 또는 기존 사물 그룹에 사물 추가” 단계를 포함시키십시오. AWS IoT IDT에서는 설치 설정 중에 사물 및 사물 그룹이 생성된다고 가정합니다. 이러한 값이 파일에 반영되었는지 확인하십시오. `effectiveConfig.yaml` IDT는 `effectiveConfig.yaml` 아래에 `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml` 있는 파일을 확인합니다.

HSM으로 테스트를 실행하려면 `aws.greengrass.crypto.Pkcs11Provider` 필드가 에서 업데이트되었는지 확인하세요. `effectiveConfig.yaml`

`GreengrassV2TokenExchangeRole`

(선택사항) 테스트 대상 기기가 리소스와 상호작용하는 것으로 가정하는 토큰 교환 역할로 사용하려는 사용자 지정 IAM 역할. AWS

> **Note**
>
> IDT는 테스트 실행 중에 기본 토큰 교환 역할을 생성하는 대신 이 사용자 지정 IAM 역할을 사용합니다. 사용자 지정 역할을 사용하는 경우 사용자가 IAM 역할 및 정책을 생성하고 삭제할 수 있도록 허용하는 `iamResourcesUpdate` 명령문을 제외하도록 테스트 사용자의 IAM 권한을 업데이트할 수 있습니다.

사용자 지정 IAM 역할을 토큰 교환 역할로 생성하는 방법에 대한 자세한 내용은 을 참조하십시오. 사용자 지정 토큰 교환 역할을 구성하세요.

`hsm`

이 기능은 IDT v4.5.1 이상에서 사용할 수 있습니다.

(선택 사항) AWS IoT Greengrass 하드웨어 보안 모듈 (HSM) 을 사용한 테스트를 위한 구성 정보. 그렇지 않으면 `hsm` 속성을 생략해야 합니다. 자세한 설명은 하드웨어 보안 통합 섹션을 참조하세요.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

> **Warning**
>
> 하드웨어 보안 모듈을 IDT와 다른 시스템 간에 공유하는 경우 HSM 구성은 민감한 데이터로 간주될 수 있습니다. 이 경우 AWS Parameter Store SecureString 파라미터에 구성 값을 저장하고 테스트 실행 중에 이를 가져오도록 IDT를 구성하면 이러한 구성 값을 일반 텍스트로 보호하는 것을 피할 수 있습니다. 자세한 내용은 ??? 단원을 참조하세요.

`hsm.greengrassPkcsPluginJar`

IDT 호스트 시스템에 다운로드한 PKCS #11 공급자 구성 요소의 전체 경로입니다. AWS IoT Greengrass이 구성 요소를 JAR 파일로 제공하여 설치 중에 프로비저닝 플러그인으로 지정할 수 있도록 다운로드할 수 있습니다. 구성 요소 JAR 파일의 최신 버전을 다음 URL로 다

운로드할 수 있습니다. https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

`hsm.pkcs11ProviderLibrary`

HSM과 상호 작용하기 위해 하드웨어 보안 모듈 (HSM) 공급업체에서 제공하는 PKCS #11 라이브러리의 전체 경로입니다.

`hsm.slotId`

키와 인증서를 로드할 HSM 슬롯을 식별하는 데 사용되는 슬롯 ID입니다.

`hsm.slotLabel`

키와 인증서를 로드하는 HSM 슬롯을 식별하는 데 사용되는 슬롯 레이블입니다.

`hsm.slotUserPin`

IDT가 HSM에 AWS IoT Greengrass Core 소프트웨어를 인증하는 데 사용하는 사용자 PIN입니다.

> **Note**
>
> 보안 모범 사례로서 프로덕션 디바이스에서는 동일한 사용자 PIN을 사용하지 마십시오.

`hsm.keyLabel`

하드웨어 모듈에서 키를 식별하는 데 사용되는 레이블입니다. 키와 인증서 모두 동일한 키 라벨을 사용해야 합니다.

`hsm.preloadedCertificateArn`

클라우드에 업로드된 디바이스 인증서의 Amazon 리소스 이름 (ARN). AWS IoT

이전에 HSM의 키를 사용하여 이 인증서를 생성하고 HSM으로 가져와 클라우드에 업로드한 적이 있어야 합니다. AWS IoT 인증서 생성 및 가져오기에 대한 자세한 내용은 HSM 설명서를 참조하십시오.

config.json에 제공한 것과 동일한 계정 및 지역에 인증서를 업로드해야 합니다. . 인증서를 업로드하는 AWS IoT 방법에 대한 자세한 내용은 AWS IoT개발자 안내서의 클라이언트 인증서 수동 등록을 참조하십시오.

`hsm.rootCAPath`

(선택 사항) IDT 호스트 시스템에서 인증서를 서명한 루트 인증 기관 (CA) 에 대한 전체 경로. 이는 Amazon 루트 CA가 생성한 HSM의 인증서에 서명하지 않은 경우 필요합니다.

파라미터 스토어에서 AWS 구성을 가져옵니다.

AWS IoT디바이스 테스터 (IDT) 에는 AWSSystems Manager 파라미터 저장소에서 구성 값을 가져오는 옵션 기능이 포함되어 있습니다. AWS 파라미터 스토어를 사용하면 구성을 안전하고 암호화하여 저장할 수 있습니다. 구성된 경우 IDT는 매개변수를 파일 내에 일반 텍스트로 저장하는 대신 AWS Parameter Store에서 매개변수를 가져올 수 있습니다. `userdata.json` 이는 암호, 핀 및 기타 비밀과 같이 암호화하여 저장해야 하는 모든 민감한 데이터에 유용합니다.

1. 이 기능을 사용하려면 IDT 사용자를 만들 때 사용한 권한을 업데이트하여 IDT에서 사용하도록 구성된 매개변수에 대한 GetParameter 작업을 허용해야 합니다. 다음은 IDT 사용자에게 추가할 수 있는 권한 설명의 예입니다. 자세한 내용은 AWS Systems Manager사용자 가이드를 참조하십시오.

   ```
   {
          "Sid":"parameterStoreResources",
          "Effect": "Allow",
           "Action": [
               "ssm:GetParameter"
           ],
           "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
   }
   ```

   위 권한은 와일드카드 문자를 사용하여 이름이 로 IDT 시작하는 모든 매개 변수를 가져올 수 있도록 구성되어 있습니다. * IDT가 사용 중인 매개변수의 이름을 기반으로 구성된 매개변수를 가져올 수 있도록 필요에 맞게 사용자 지정해야 합니다.

2. 구성 값은 파라미터 저장소에 저장해야 합니다. AWS 이 작업은 AWS 콘솔 또는 AWS CLI에서 수행할 수 있습니다. AWS 파라미터 저장소에서는 암호화된 저장소나 암호화되지 않은 저장소를 선택할 수 있습니다. 비밀, 암호, 핀과 같은 민감한 값을 저장하려면 파라미터 유형인 암호화된 옵션을 사용해야 합니다. SecureString AWSCLI를 사용하여 파라미터를 업로드하려면 다음 명령을 사용할 수 있습니다.

   ```
   aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
    SecureString
   ```

다음 명령을 사용하여 파라미터가 저장되었는지 확인할 수 있습니다. (선택 사항) `--with-decryption` 플래그를 사용하여 SecureString 복호화된 파라미터를 가져올 수 있습니다.

```
aws ssm get-parameter --name IDT-example-name
```

CLI를 사용하면 AWS 현재 CLI 사용자의 AWS 영역에 파라미터가 업로드되고 IDT는 에 구성된 리전에서 파라미터를 가져옵니다. `config.json` AWSCLI에서 지역을 확인하려면 다음을 사용하세요.

```
aws configure get region
```

3. AWS클라우드에 구성 값이 있으면 IDT 구성 내의 모든 값을 업데이트하여 클라우드에서 가져올 수 있습니다. AWS 이렇게 `{{AWS.Parameter.parameter_name}}` 하려면 양식의 IDT 구성에 있는 자리 표시자를 사용하여 Parameter Store에서 해당 이름으로 파라미터를 가져오면 됩니다. AWS

   예를 들어 2단계의 `IDT-example-name` 파라미터를 HSM 구성의 HSM 키 레이블로 사용한다고 가정해 보겠습니다. 이렇게 하려면 다음과 같이 업데이트하면 됩니다. `userdata.json`

   ```
   "hsm": {
           "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
           [...]
       }
   ```

   IDT는 2단계에서 설정한 이 매개변수 값을 `IDT-example-value` 런타임에 가져옵니다. 이 구성은 설정과 `"keyLabel": "IDT-example-value"` 비슷하지만 대신 해당 값이 암호화된 상태로 클라우드에 AWS 저장됩니다.

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` 명령어를 사용하여 테스트 모음을 실행합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite  \\
    --suite-id suite-id  \\
```

```
    --group-id group-id  \\
    --pool-id your-device-pool \\
    --test-id test-id  \\
    --update-idt y|n  \\
    --userdata userdata.json
```

모든 옵션은 선택 사항입니다. 예를 들어, device.json 파일에 정의된 동일한 장치 집합인 장치 풀이 하나뿐인 pool-id 경우 생략할 수 있습니다. 또는 tests 폴더에서 최신 테스트 제품군 버전을 실행하려면 suite-id를 생략할 수 있습니다.

> **Note**
>
> 상위 테스트 제품군 버전이 온라인으로 제공되는 경우 IDT가 메시지를 표시합니다. 자세한 설명은 the section called “테스트 제품군 버전” 섹션을 참조하세요.

검증 제품군을 실행하기 위한 예제 명령

다음 명령줄 예제는 기기 풀에 대한 검증 테스트를 실행하는 방법을 보여줍니다. run-suite 및 기타 IDT 명령에 대한 자세한 내용은 the section called “IDT 명령” 단원을 참조하십시오.

다음 명령을 사용하여 지정된 테스트 스위트의 모든 테스트 그룹을 실행합니다. list-suites명령어는 tests 폴더에 있는 테스트 스위트를 나열합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --pool-id <pool-id> \
    --userdata userdata.json
```

다음 명령어를 사용하여 테스트 스위트에서 특정 테스트 그룹을 실행합니다. list-groups명령어는 테스트 스위트의 테스트 그룹을 나열합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --group-id <group-id> \
    --pool-id <pool-id> \
    --userdata userdata.json
```

다음 명령어를 사용하여 테스트 그룹에서 특정 테스트 사례를 실행합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
```

```
    --group-id <group-id> \
    --test-id <test-id> \
    --userdata userdata.json
```

다음 명령어를 사용하여 테스트 그룹에서 여러 테스트 사례를 실행할 수 있습니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id1>,<test-id2>
    --userdata userdata.json
```

다음 명령어를 사용하여 테스트 그룹의 모든 테스트 사례를 나열합니다.

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

테스트 그룹 종속성을 올바른 순서로 실행하는 전체 검증 테스트 스위트를 실행하는 것이 좋습니다. 특정 테스트 그룹을 실행하기로 선택한 경우 관련 테스트 그룹을 실행하기 전에 먼저 종속성 검사기 테스트 그룹을 실행하여 모든 Greengrass 종속성이 설치되었는지 확인하는 것이 좋습니다. 예:

- 코어 자격 테스트 그룹을 실행하기 전에 `coredependencies`를 실행합니다.

V2용 IDT 명령 AWS IoT Greengrass

IDT 명령은 *<device-tester-extract-location>*/bin 디렉터리에 있습니다. 테스트 스위트를 실행하려면 다음 형식으로 명령을 제공합니다.

`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 사례를 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)입니다. 지정하지 않으면 IDT는 에서 구성된 설정에 따라 테스트 스위트에서 적절한 테스트 그룹을 모두 실행합니다. `device.json` IDT는 구성된 설정을 기반으로 기기에서 지원하지 않는 테스트 그룹을 실행하지 않습니다. 목록에 테스트 그룹이 지정되어 있더라도 마찬가지입니다. `group-id`
- `test-id`. 실행할 테스트 사례(쉼표로 구분된 목록)입니다. 지정된 경우, `group-id`은(는) 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 디바이스 풀. `device.json` 파일에 여러 디바이스 풀이 정의되어 있는 경우 하나의 풀을 지정해야 합니다.
- `stop-on-first-failure`. 첫 번째 실패 시 실행을 중지하도록 IDT를 구성합니다. 지정된 테스트 그룹을 디버깅하려는 `group-id` 경우 이 옵션을 함께 사용하십시오. 전체 테스트 제품군을 실행하여 검증 보고서를 생성할 때는 이 옵션을 사용하지 마시기 바랍니다.
- `update-idt`. IDT 업데이트 프롬프트에 대한 응답을 설정합니다. IDT에서 새 버전이 있다고 감지하면 `Y` 응답으로 테스트 실행이 중지됩니다. `N`응답은 테스트 실행을 계속합니다.
- `userdata`. 테스트 아티팩트 경로에 대한 정보가 들어 있는 `userdata.json` 파일의 전체 경로입니다. 이 옵션은 `run-suite` 명령에 필요합니다. `userdata.json`*`### devicetester_extract_location /devicetester_ggv2_ [win|mac|linux] / configs/ ##### ### ###.`*

`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.

오류 문제를 해결하려면 을 참조하십시오에 대한 IDT 문제 해결AWS IoT GreengrassV2.

결과 보기

실행하는 동안 IDT는 콘솔, 로그 파일 및 테스트 보고서에 오류를 작성합니다. IDT는 자격 테스트 제품군을 완료한 후 두 개의 테스트 보고서를 생성합니다. 이 보고서는 에 있습니다*<device-tester-extract-location>*/results/*<execution-id>*/. 두 보고서 모두 자격 테스트 도구 모음 실행 결과를 캡처합니다.

awsiotdevicetester_report.xml는 디바이스 카탈로그에 디바이스를AWS 등록하기 위해 제출하는 자격 테스트 보고서입니다.AWS Partner 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
- 로컬 리소스 액세스, 섀도우 및 MQTT와 같은 장치 기능을 기반으로 테스트된 라이브러리별 테스트 결과 분석입니다.

GGV2Q_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

AWS IoT Device Tester결과 해석

awsiotdevicetester_report.xml 또는 awsiotdevicetester_report.xml의 보고서 섹션에는 실행된 테스트 및 결과가 나열됩니다.

첫 번째 XML<testsuites> 태그에는 테스트 실행 요약이 들어 있습니다. 예시:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 태그에 사용되는 속성

`name`

테스트 제품군의 이름입니다.

`time`

검증 제품군을 실행하는 데 걸린 시간 (초) 입니다.

`tests`

실행된 테스트 수입니다.

`failures`

실행되었지만 통과하지 못한 테스트의 수입니다.

`errors`

IDT에서 실행할 수 없는 테스트 수입니다.

`disabled`

이 속성은 무시하십시오. 사용되지 않습니다.

`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.

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></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. 디바이스 카탈로그에 디바이스를 등록하려면 이 보고서를 자격 증빙 자료로 사용할 수 있습니다.AWS Partner

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. 예시:

```
<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 안에는 테스트 그룹에 대해 실행된 각 테스트에 대한 태그가 있습니다.<testcase> 예시:

```
<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 케이스를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예시:

```
<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 에서 테스트를 실행하여 로그를 생성합니다*<devicetester-extract-location>*/results/*<execution-id>*/logs. 두 개의 로그 세트가 생성됩니다.

`test_manager.log`

의 Test Manager 구성 요소에서 생성된 로그AWS IoT Device Tester (예: 구성, 테스트 시퀀싱 및 보고서 생성과 관련된 로그)

<test-case-id>`.log (for example, lambdaDeploymentTest.log)`

테스트 대상 기기의 로그를 포함한 테스트 그룹 내 테스트 사례의 로그입니다. IDT v4.2.0부터 IDT는*<devicetester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>/* 디렉터리 내의 별도 *<test-case-id >* 폴더에 각 테스트 사례에 대한 테스트 로그를 그룹화합니다.

IDT를 사용하여 자체 테스트 도구 모음을 개발하고 실행하십시오.

IDT v4.0.1부터 IDT for AWS IoT Greengrass V2는 표준화된 구성 설정 및 결과 형식을 장치 및 장치 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 제품군을 개발하고 실행합니다.

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 제품군의 특정 설정 구성에 대한 정보를 포함해야 합니다.

사용자 지정 테스트 제품군을 실행하려면

- 테스트하려는 디바이스를 설정합니다.
- 사용하려는 테스트 제품군에 필요한 설정 구성을 구현합니다.
- IDT를 사용하여 사용자 지정 테스트 제품군을 실행합니다.
- IDT에서 실행한 테스트의 테스트 결과 및 실행 로그를 확인합니다.

의 최신 버전을 다운로드하십시오. AWS IoT Device TesterAWS IoT Greengrass

최신 버전의 IDT를 다운로드하고 파일 시스템에서 읽기/쓰기 권한이 있는 위치 (*< device-tester-extract-location >*) 에 소프트웨어를 추출하십시오.

> **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.

테스트 제품군 생성 워크플로

테스트 제품군은 세 가지 유형의 파일로 구성됩니다.

- 테스트 도구 모음 실행 방법에 대한 정보를 IDT에 제공하는 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 생성합니다.

1. 테스트 제품군의 구성 파일을 생성합니다.
2. 테스트 제품군의 테스트 로직이 포함된 테스트 사례 실행 파일을 생성합니다.
3. 테스트 제품군을 실행하는 데 테스트 실행기에 필요한 구성 정보를 확인하고 문서화합니다.
4. IDT가 예상대로 테스트 제품군을 실행하고 테스트 결과를 생성할 수 있는지 확인합니다.

샘플 사용자 지정 테스트 제품군을 빠르게 빌드하고 실행하려면 튜토리얼: 샘플 IDT 테스트 도구 모음 구축 및 실행의 지침을 따릅니다.

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 제품군의 소스 코드가 포함되어 있습니다. 이 튜토리얼을 완료하여 샘플 테스트 스위트를 빌드하고 실행하여 IDT for 를 사용하여 사용자 지정 테스트 스위트를 실행하는 AWS IoT Greengrass 방법을 이해할 수 있습니다.

이 자습서에서는 다음 단계를 완료합니다.

1. 샘플 테스트 제품군 빌드
2. IDT를 사용하여 샘플 테스트 제품군을 실행합니다.

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - AWS IoT Device Tester의 최신 버전
  - Python 3.7 이상

    컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

    ```
    python3 --version
    ```

    Windows에서 이 명령을 사용하여 오류가 반환되면 `python --version`을 대신 사용하세요. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을 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 구성합니다. *<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 객체에 다음 정보를 제공합니다.

id

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

connectivity.ip

디바이스의 IP 주소입니다.

connectivity.port

선택 사항으로, 디바이스에 대한 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 이름입니다.

connectivity.auth.credentials.privKeyPath

디바이스에 로그인하는 데 사용되는 프라이빗 키의 전체 경로입니다.

이 값은 connectivity.auth.method가 pki로 설정된 경우에만 적용됩니다.

devices.connectivity.auth.credentials.password

디바이스에 로그인하기 위해 사용되는 암호입니다.

이 값은 connectivity.auth.method가 password로 설정된 경우에만 적용됩니다.

> **Note**
>
> method가 pki로 설정된 경우에만 privKeyPath를 지정합니다.
> method가 password로 설정된 경우에만 password를 지정합니다.

샘플 테스트 제품군 빌드

<device-tester-extract-location>/samples/python 폴더에는 제공된 빌드 스크립트를 사용하여 테스트 제품군에 결합할 수 있는 샘플 구성 파일, 소스 코드 및 IDT 클라이언트 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 제품군을 빌드하려면 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 *<device-tester-extract-location>*/tests 폴더 내 IDTSampleSuitePython_1.0.0 폴더에 샘플 테스트 제품군이 생성됩니다. IDTSampleSuitePython_1.0.0폴더의 파일을 검토하여 샘플 테스트 도구 모음의 구조를 이해하고 테스트 사례 실행 파일 및 테스트 구성 JSON 파일의 다양한 예를 확인하세요.

> **Note**
>
> 샘플 테스트 제품군에는 python 소스 코드가 포함되어 있습니다. 테스트 제품군 코드에 민감한 정보를 포함하지 마십시오.

다음 단계: IDT를 사용하여 생성한 샘플 테스트 제품군을 실행합니다.

IDT를 사용하여 샘플 테스트 제품군을 실행합니다.

샘플 테스트 제품군을 실행하려면 호스트 컴퓨터에서 다음 명령을 실행합니다.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT는 샘플 테스트 제품군을 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 자습서 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 사례가 성공적으로 실행되지 않습니다.

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 이 자습서의 모든 사전 조건을 충족하는지 확인하세요.

테스트 대상 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 제품군은 다음을 결합합니다.

- 테스트 로직이 포함된 테스트 실행 파일
- 테스트 제품군을 설명하는 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 제품군 디렉터리 생성
2. 구성 파일 생성
3. 테스트 사례 실행 파일 생성
4. 테스트 제품군 실행

사전 조건

이 튜토리얼을 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - AWS IoT Device Tester의 최신 버전
  - Python 3.7 이상

    컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

    ```
    python3 --version
    ```

    Windows에서 이 명령을 사용하여 오류가 반환되면 `python --version`을 대신 사용하세요. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을 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 제품군 디렉터리 생성

IDT는 각 테스트 제품군 내의 테스트 그룹에 테스트 사례를 논리적으로 분리합니다. 각 테스트 사례는 테스트 그룹 내에 있어야 합니다. 이 자습서에서는 `MyTestSuite_1.0.0`이라는 폴더를 생성하고 이 폴더 내에 다음 디렉터리 트리를 생성합니다.

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

구성 파일 생성

테스트 제품군에는 다음과 같은 필수 구성 파일이 포함되어야 합니다.

필수 구성 파일

`suite.json`

테스트 제품군 정보가 포함되어 있습니다. suite.json 구성 섹션을 참조하세요.

`group.json`

테스트 그룹에 대한 정보가 포함되어 있습니다. 테스트 제품군의 각 테스트 그룹에 대한 `group.json` 파일을 생성해야 합니다. group.json 구성 섹션을 참조하세요.

`test.json`

테스트 사례에 대한 정보가 들어 있습니다. 테스트 제품군의 각 테스트 사례에 대한 `test.json` 파일을 생성해야 합니다. test.json 구성 섹션을 참조하세요.

1. `MyTestSuite_1.0.0/suite` 폴더에서 다음 폴더 구조로 `suite.json` 파일을 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 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 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 자습서에서는 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 제품군에는 여러 테스트 사례 실행 파일이 포함될 수 있습니다. 이 자습서에서는 테스트 사례 실행 파일을 하나만 생성합니다.

1. 테스트 제품군 파일을 생성합니다.

   MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 폴더 안에 다음 내용으로 myTestCase.py라는 파일을 생성합니다.

   ```
   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 대상 디바이스에서 SSH 명령을 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 구성합니다. *<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 객체에 다음 정보를 제공합니다.

`id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`connectivity.ip`

디바이스의 IP 주소입니다.

`connectivity.port`

선택 사항으로, 디바이스에 대한 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 이름입니다.

`connectivity.auth.credentials.privKeyPath`

디바이스에 로그인하는 데 사용되는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`devices.connectivity.auth.credentials.password`

디바이스에 로그인하기 위해 사용되는 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.

테스트 제품군 실행

테스트 제품군을 생성한 후에는 예상대로 작동하는지 확인해야 합니다. 이를 위해 기존 디바이스 풀로 테스트 제품군을 실행하려면 다음 단계를 완료합니다.

1. MyTestSuite_1.0.0 폴더를 *<device-tester-extract-location>*/tests에 복사합니다.
2. 다음 명령을 실행합니다.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT는 테스트 제품군을 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 자습서 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 사례가 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 자습서의 모든 사전 조건을 충족합니다.

테스트 대상 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 제품군을 작성할 때 포함하는 구성 파일을 생성하는 형식을 설명합니다.

필수 구성 파일

`suite.json`

테스트 제품군 정보가 포함되어 있습니다. suite.json 구성 섹션을 참조하세요.

`group.json`

테스트 그룹에 대한 정보가 포함되어 있습니다. 테스트 제품군의 각 테스트 그룹에 대한 `group.json` 파일을 생성해야 합니다. group.json 구성 섹션을 참조하세요.

`test.json`

테스트 사례에 대한 정보가 들어 있습니다. 테스트 제품군의 각 테스트 사례에 대한 `test.json` 파일을 생성해야 합니다. test.json 구성 섹션을 참조하세요.

선택적 구성 파일

`test_orchestrator.yaml` 또는 `state_machine.json`

IDT가 테스트 제품군을 실행할 때 테스트가 실행되는 방법을 정의합니다. SSe test_orchestrator.yaml 구성.

> **Note**
>
> IDT v4.5.1부터 `test_orchestrator.yaml` 파일을 사용하여 테스트 워크플로를 정의합니다. 이전 버전의 IDT에서는 `state_machine.json` 파일을 사용했습니다. 상태 시스템에 대한 자세한 내용은 IDT 상태 머신 구성 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.json 구성 섹션을 참조하세요.

구성 파일은 다음과 같이 *<custom-test-suite-folder>*에 저장됩니다.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

suite.json 구성

`suite.json` 파일은 환경 변수를 설정하고 테스트 제품군을 실행하는 데 사용자 데이터가 필요한지 여부를 결정합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>*`/suite/suite.json` 파일을 구성합니다.

```
{
```

```
    "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 제품군의 고유한 사용자 정의 ID입니다. `id`의 값은 `suite.json` 파일이 있는 테스트 제품군 폴더의 이름과 일치해야 합니다. 제품군 이름과 제품군 버전도 다음 요구 사항을 충족해야 합니다.

- *<suite-name>*은 밑줄을 포함할 수 없습니다.
- *<suite-version>*은 다음과 같이 *x.x.x*로 표시됩니다. 여기서 x는 숫자입니다.

ID는 IDT에서 생성한 테스트 보고서에 표시됩니다.

`title`

이 테스트 제품군에서 테스트 중인 제품 또는 기능의 사용자 정의 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 제품군의 용도에 대한 짧은 설명입니다.

`userDataRequired`

테스트 실행기가 `userdata.json` 파일에 사용자 지정 정보를 포함해야 하는지 여부를 정의합니다. 이 값을 `true`로 설정하는 경우, 테스트 제품군 폴더에도 `userdata_schema.json` 파일을 포함해야 합니다.

`environmentVariables`

선택 사항으로, 이 테스트 제품군에 설정할 환경 변수의 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

group.json 구성

`group.json` 파일은 테스트 그룹이 필수인지 옵션인지 여부를 정의합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/*<test-group>*/group.json 파일을 구성합니다.

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`id`

테스트 그룹의 고유한 사용자 정의 ID입니다. 의 값은 `group.json` 파일이 있는 테스트 그룹 폴더의 이름과 `id` 일치해야 하며 밑줄 () 을 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 사항으로, IDT에서 필수 테스트 실행을 완료한 후 이 테스트 그룹을 선택적 그룹으로 표시하도록 `true`로 설정합니다. 기본값은 `false`입니다.

test.json 구성

`test.json` 파일은 테스트 사례 실행 파일 및 테스트 사례에서 사용되는 환경 변수를 결정합니다. 테스트 사례 실행 파일 생성에 대한 자세한 내용은 IDT 테스트 케이스 실행 파일 생성 섹션을 참조하세요.

다음 템플릿을 사용하여 *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json 파일을 구성합니다.

```
{
    "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 사례의 고유한 사용자 정의 ID입니다. 의 값은 `test.json` 파일이 있는 테스트 사례 폴더의 이름과 `id` 일치해야 하며 밑줄 () _ 을 포함할 수 없습니다. ID는 IDT에서 생성한 테스트 보고서에 사용됩니다.

`title`

테스트 사례를 나타내는 서술형 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

`details`

테스트 사례의 용도에 대한 짧은 설명입니다.

`requireDUT`

선택 사항으로, 이 테스트를 실행하는 데 디바이스가 필요한 경우, `true`로 설정하고, 그렇지 않으면 `false`로 설정합니다. 기본값은 `true`입니다. 테스트 실행기는 `device.json` 파일에서 테스트를 실행하는 데 사용할 디바이스를 구성합니다.

`requiredResources`

선택 사항으로, 이 테스트를 실행하는 데 필요한 리소스 디바이스에 대한 정보를 제공하는 배열입니다.

`requiredResources.name`

이 테스트를 실행할 때 리소스 디바이스에 부여하는 고유한 이름입니다.

`requiredResources.features`

사용자 정의 리소스 디바이스 기능의 배열입니다.

`requiredResources.features.name`

기능의 이름입니다. 이 디바이스를 사용하려는 디바이스 기능입니다. 이 이름은 테스트 실행기가 `resource.json` 파일에 제공한 기능 이름과 일치합니다.

`requiredResources.features.version`

선택 사항으로, 기능의 버전입니다. 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 사항으로, 이 기능이 지원할 수 있는 동시 테스트 수입니다. 기본 값은 1입니다. IDT에서 개별 기능에 대해 서로 다른 디바이스를 사용하도록 하려면 이 값을 1로 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 사례 실행 파일입니다. 지원되는 값은 `linux`, `mac` 및 `win`입니다.

`execution.os.cmd`

지정된 운영 체제에서 실행하려는 테스트 사례 실행 파일의 경로입니다. 이 위치는 시스템 경로에 있어야 합니다.

`execution.os.args`

선택 사항으로, 테스트 사례 실행 파일을 실행하기 위해 제공할 인수입니다.

`environmentVariables`

선택 사항으로, 이 테스트 사례에 설정된 환경 변수의 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.

test_orchestrator.yaml 구성

테스트 오케스트레이터는 테스트 제품군 실행 흐름을 제어하는 구조입니다. 테스트 제품군의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 최종 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 제품군에 사용자 정의 테스트 오케스트레이터가 포함되어 있지 않은 경우 IDT가 테스트 오케스트레이터를 생성합니다.

기본 테스트 오케스트레이터는 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 오케스트레이터의 작동 방식에 대한 자세한 내용은 IDT 테스트 오케스트레이터 구성 섹션을 참조하세요.

userdata_schema.json 구성

`userdata_schema.json` 파일은 테스트 실행기가 사용자 데이터를 제공하는 스키마를 결정합니다. 테스트 제품군에 `device.json` 파일에 없는 정보가 필요한 경우, 사용자 데이터가 필요합니다. 예를 들어 테스트에는 사용자가 제공해야 하는 Wi-Fi 네트워크 보안 인증 정보, 특정 개방 포트 또는 인증서가 필요할 수 있습니다. 이 정보는 `userdata`라는 입력 파라미터로 IDT에 제공될 수 있습니다. 해당 값은 사용자가 *`<device-tester-extract-location>`*`/config` 폴더에 생성하는 `userdata.json` 파일입니다. `userdata.json` 파일 형식은 테스트 제품군에 포함된 `userdata_schema.json` 파일을 기반으로 합니다.

테스트 실행기가 `userdata.json` 파일을 제공해야 함을 나타내려면

1. suite.json 파일에서 userDataRequired를 true로 설정합니다.
2. *<custom-test-suite-folder>*에서 userdata_schema.json 파일을 생성합니다.
3. userdata_schema.json 파일을 편집하여 유효한 IETF Draft v4 JSON 스키마를 생성합니다.

IDT는 테스트 제품군을 실행하면 자동으로 스키마를 읽고 이를 사용하여 테스트 실행기가 제공한 userdata.json 파일을 검증합니다. 유효한 경우, userdata.json 파일의 내용은 IDT 컨텍스트와 테스트 오케스트레이터 컨텍스트 모두에서 사용할 수 있습니다.

IDT 테스트 오케스트레이터 구성

IDT v4.5.1부터 IDT에는 새로운 기능이 포함되어 있습니다.테스트 오케스트레이터구성 요소. 테스트 오케스트레이터는 테스트 세트 실행 흐름을 제어하고 IDT가 모든 테스트 실행을 마친 후 테스트 보고서를 생성하는 IDT 구성 요소입니다. 테스트 오케스트레이터는 테스트 선택 및 사용자 정의 규칙에 따라 테스트가 실행되는 순서를 결정합니다.

테스트 세트에 사용자 정의 테스트 오케스트레이터가 포함되어 있지 않은 경우 IDT는 테스트 오케스트레이터를 생성합니다.

기본 테스트 오케스트레이터는 다음 기능을 수행합니다.

- 테스트 러너에게 전체 테스트 세트 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않으면 에서는 테스트 스위트의 모든 테스트 그룹을 임의의 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례에 대한 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

테스트 오케스트레이터가 IDT 테스트 오케스트레이터를 대체합니다. IDT 테스트 오케스트레이터 대신 테스트 오케스트레이터를 사용하여 테스트 스위트를 개발하는 것이 좋습니다. 테스트 오케스트레이터는 다음과 같은 향상된 기능을 제공합니다.

- IDT 상태 머신에서 사용하는 명령형 형식과 비교하여 선언적 형식을 사용합니다. 이를 통해지정하다어떤 테스트를 실행하고 싶은지언제실행하려는 경우.
- 특정 그룹 처리, 보고서 생성, 오류 처리 및 결과 추적 관리따라서 필요하지 않습니다.를 눌러 이러한 작업을 수동으로 관리합니다.
- 기본적으로 주석을 지원하는 YAML 형식을 사용합니다.
- 필수80%동일한 워크플로를 정의하는 테스트 오케스트레이터보다 디스크 공간이 적습니다.

- 사전 테스트 유효성 검사를 추가하여 워크플로 정의에 잘못된 테스트 ID 또는 순환 종속성이 포함되어 있지 않은지 확인합니다.

테스트 오케스트레이터 형식

다음 템플릿을 사용하여 고유한 템플릿을 구성할 수 있습니다.*<custom-test-suite-folder>*/suite/test_orchestrator.yaml파일:

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Aliases`

선택 사항. 컨텍스트 표현식에 매핑되는 사용자 정의 문자열입니다. 별칭을 사용하면 다음과 같은 이름을 생성할 수 있습니다.테스트 오케스트레이터 구성에서 컨텍스트 표현식을 식별합니다. 이 기능은 여러 위치에서 사용하는 복잡한 컨텍스트 표현식이나 표현식을 생성하는 경우에 특히 유용합니다.

컨텍스트 표현식을 사용하여 다른 IDT 구성의 데이터에 액세스할 수 있는 컨텍스트 쿼리를 저장할 수 있습니다. 자세한 정보는 컨텍스트에서 데이터 액세스을 참조하십시오.

Example 예

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

선택 사항. 조건 목록 및 각 조건이 충족될 때 실행되는 해당 테스트 케이스입니다. 각 조건에는 여러 테스트 사례가 있을 수 있지만, 주어진 테스트 케이스를 하나의 조건에만 할당할 수 있습니다.

기본적으로 IDT는 이 목록의 조건에 할당되지 않은 테스트 케이스를 실행합니다. 이 섹션을 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.

의 각 항목`ConditionalTests`에는 다음 파라미터가 포함되어 있습니다.

`Condition`

로 평가되는 컨텍스트 표현식입니다.부울USD 상당. 평가된 값이 true이면 IDT는 에 지정된 테스트 케이스를 실행합니다.`Tests`파라미터.

`Tests`

테스트 설명자 목록입니다.

각 테스트 설명자는 테스트 그룹 ID와 하나 이상의 테스트 사례 ID를 사용하여 특정 테스트 그룹에서 실행할 개별 테스트를 식별합니다. 테스트 설명자는 다음 형식을 사용합니다.

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example 예

다음 예제에서는 다음과 같이 정의할 수 있는 제네릭 컨텍스트 표현식을 사용합니다.`Aliases`.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
```

```
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

정의된 조건에 따라 IDT는 다음과 같이 테스트 그룹을 선택합니다.

- 다음의 경우,`Condition1`true, IDT는 테스트 그룹 A, B 및 C에서 테스트를 실행합니다.
- 다음의 경우,`Condition2`true, IDT는 테스트 그룹 C와 D에서 테스트를 실행합니다.

`Order`

선택 사항. 테스트 실행 순서입니다. 테스트 그룹 수준에서 테스트 순서를 지정합니다. 이 섹션을 지정하지 않으면 IDT는 모든 적용 가능한 테스트 그룹을 임의의 순서로 실행합니다. 의 가치`Order`은 그룹 설명자 목록 목록입니다. 리스팅하지 않은 모든 테스트 그룹`Order`는 나열된 다른 테스트 그룹과 병렬로 실행할 수 있습니다.

각 그룹 설명자 목록에는 그룹 설명자 중 하나가 더 포함되어 있으며 각 설명자에 지정된 그룹을 실행하는 순서를 식별합니다. 다음 형식을 사용하여 개별 그룹 설명자를 정의할 수 있습니다.

- *`group-id`*—기존 테스트 그룹의 그룹 ID입니다.
- [*`group-id`*, *`group-id`*]서로에 대해 임의의 순서로 실행할 수 있는 테스트 그룹 목록입니다.
- `"*"`와일드카드 이는 현재 그룹 설명자 목록에 아직 지정되지 않은 모든 테스트 그룹 목록과 동일합니다.

에 대한 가치`Order`다음 요구 사항도 충족해야 합니다.

- 그룹 설명자에서 지정한 테스트 그룹 ID는 테스트 스위트에 있어야 합니다.
- 각 그룹 설명자 목록은 하나 이상의 테스트 그룹을 포함해야 합니다.
- 각 그룹 설명자 목록에는 고유한 그룹 ID가 포함되어야 합니다. 개별 그룹 설명자 내에서는 테스트 그룹 ID를 반복할 수 없습니다.
- 그룹 설명자 목록에는 와일드카드 그룹 설명자가 하나 이상 포함될 수 있습니다. 와일드카드 그룹 설명자는 목록의 첫 번째 항목이나 마지막 항목이어야 합니다.

Example 예제

테스트 그룹 A, B, C, D 및 E를 포함하는 테스트 스위트의 경우 다음 예제 목록은 IDT가 먼저 테스트 그룹 A를 실행한 다음 테스트 그룹 B를 실행한 다음 테스트 그룹 C, D 및 E를 임의의 순서로 실행하도록 지정하는 다양한 방법을 보여줍니다.

-
```
Order:
    - - A
      - B
```

```
      - [C, D, E]
```

- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

`Features`

선택 사항. IDT에 추가하려는 제품 기능 목록`awsiotdevicetester_report.xml`파일. 이 섹션을 지정하지 않으면 IDT는 보고서에 제품 기능을 추가하지 않습니다.

제품 기능은 장치가 충족할 수 있는 특정 기준에 대한 사용자 정의 정보입니다. 예를 들어, MQTT 제품 기능은 장치가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. In`awsiotdevicetester_report.xml`제품 기능은 다음과 같이 설정됩니다.`supported`,`not-supported`또는 지정된 테스트가 통과되었는지 여부에 따라 사용자 정의 값입니다.

의 각 항목`Features`은 다음 파라미터로 구성됩니다.

`Name`

기능 이름입니다.

`Value`

선택 사항. 대신 보고서에 사용할 사용자 정의 값입니다.`supported`. 이 값을 지정하지 않으면 기반 IDT는 피쳐 값을 로 설정합니다.`supported`또는`not-supported`테스트 결과를 기반으로 합니다. 동일한 피처를 서로 다른 조건으로 테스트하는 경우, 에서 해당 피처의 각 인스턴스에 대해 사용자 정의 값을 사용할 수 있습니다.`Features`list 및 IDT는 지원되는 조건에 대한 기능 값을 연결합니다. 자세한 내용은 단원을 참조하십시오.

`Condition`

로 평가되는 컨텍스트 표현식입니다.부울USD 상당. 평가된 값이 true인 경우 IDT는 테스트 세트 실행을 마친 후 테스트 보고서에 기능을 추가합니다. 평가된 값이 false이면 테스트가 보고서에 포함되지 않습니다.

`Tests`

선택 사항. 테스트 설명자 목록입니다. 기능을 지원하려면 이 목록에 지정된 모든 테스트를 통과해야 합니다.

이 목록의 각 테스트 설명자는 테스트 그룹 ID와 하나 이상의 테스트 사례 ID를 사용하여 특정 테스트 그룹에서 실행할 개별 테스트를 식별합니다. 테스트 설명자는 다음 형식을 사용합니다.

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

다음 항목을 지정해야 합니다.`Tests`또는`OneOfTests`의 각 기능에 대해`Features`나열

`OneOfTests`

선택 사항. 테스트 설명자 목록입니다. 기능이 지원되려면 이 목록에 지정된 테스트 중 하나 이상이 통과해야 합니다.

이 목록의 각 테스트 설명자는 테스트 그룹 ID와 하나 이상의 테스트 사례 ID를 사용하여 특정 테스트 그룹에서 실행할 개별 테스트를 식별합니다. 테스트 설명자는 다음 형식을 사용합니다.

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

다음 항목을 지정해야 합니다.`Tests`또는`OneOfTests`의 각 기능에 대해`Features`나열

`IsRequired`

테스트 보고서에 기능이 필요한지 여부를 정의하는 부울 값입니다. 기본값은 `false`입니다.

Example

오케스트레이터 컨텍스트

테스트 오케스트레이터 컨텍스트는 실행 중에 테스트 오케스트레이터가 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 테스트 오케스트레이터 컨텍스트는 테스트 오케스트레이터에서

만 액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어, 테스트 러너에 의해 구성된 정보를`userdata.json`파일을 사용하여 특정 테스트를 실행해야 하는지 여부를 확인합니다.

테스트 오케스트레이터 컨텍스트는 다음 형식을 사용합니다.

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

테스트 실행을 위해 선택한 장치 풀에 대한 정보입니다. 선택한 디바이스 풀의 경우 이 정보는 에 정의된 해당 최상위 디바이스 풀 배열 요소에서 검색됩니다.`device.json`파일.

`userData`

의 정보`userdata.json`파일.

`config`

의 정보`config.json`파일.

JsonPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의에서 JSONPath 쿼리의 구문은 다음과 같습니다.`{{query}}`. 테스트 오케스트레이터 컨텍스트에서 데이터에 액세스할 때 각 값이 문자열, 숫자 또는부울.

컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 섹션을 참조하십시오.IDT 컨텍스트 사용.

IDT 상태 머신 구성

> **Important**
>
> IDT v4.5.1부터 이 상태 머신은 더 이상 사용되지 않습니다. 새로운 테스트 오케스트레이터를 사용하는 것이 좋습니다. 자세한 정보는 IDT 테스트 오케스트레이터 구성을 참조하십시오.

상태 머신은 테스트 세트 실행 흐름을 제어하는 구조입니다. 테스트 스위트의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 종료 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 세트에 사용자 정의 상태 머신이 포함되어 있지 않으면 IDT가 자동으로 상태 머신을 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 러너에게 전체 테스트 세트 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않으면 에서는 테스트 스위트의 모든 테스트 그룹을 임의의 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례에 대한 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 테스트 제품군의 상태 머신은 다음 기준을 충족해야 합니다.

- 각 상태는 테스트 그룹이나 제품 보고서 파일을 실행하는 등 IDT가 수행할 작업에 해당합니다.
- 상태로 전환하면 상태와 연관된 작업이 실행됩니다.
- 각 상태는 다음 상태에 대한 전환 규칙을 정의합니다.
- 종료 상태는 다음 중 하나여야 합니다.`Succeed`또는`Fail`.

상태 머신 형식

다음 템플릿을 사용하여 고유한 템플릿을 구성할 수 있습니다.*<custom-test-suite-folder>*/`suite/state_machine.json`파일:

```
{
  "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 머신에 대한 설명입니다.

`StartAt`

IDT에서 테스트 제품군의 실행을 시작하는 상태의 이름. 의 가치`StartAt`에 나열된 상태 중 하나로 설정해야 합니다.`States`객체입니다.

`States`

사용자 정의 상태 이름을 유효한 IDT 상태로 매핑하는 객체입니다. 각 주.*## ##*객체에 매핑된 유효한 상태의 정의가 포함되어 있습니다.*## ##*.

이`States`객체는 다음을 포함해야 합니다.`Succeed`과`Fail`상태입니다. 유효한 상태에 대한 자세한 내용은 단원을 참조하십시오.유효한 상태 및 상태 정의.

유효한 상태 및 상태 정의

이 섹션에서는 IDT 상태 시스템에서 사용할 수 있는 모든 유효한 상태의 상태 정의에 대해 설명합니다. 다음 상태 중 일부는 테스트 케이스 수준의 구성을 지원합니다. 그러나 절대적으로 필요한 경우가 아니면 테스트 케이스 수준 대신 테스트 그룹 수준에서 상태 전환 규칙을 구성하는 것이 좋습니다.

주 정의

- RunTask

- Choice
- Parallel
- 추가 제품기능
- 보고서
- 로그 메시지
- 그룹 선택
- Fail
- Succeed

RunTask

이RunTaskstate 는 테스트 스위트에 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 전환될 상태의 이름.

`TestGroup`

선택 사항. 실행할 테스트 그룹의 ID입니다. 이 값을 지정하지 않으면 IDT는 테스트 러너가 선택하는 테스트 그룹을 실행합니다.

`TestCases`

선택 사항. 에 지정된 그룹의 테스트 케이스 ID로 구성된 배열`TestGroup`. 의 가치에 따라`TestGroup`과`TestCases`, IDT는 다음과 같이 테스트 실행 동작을 결정합니다.

- 두 경우 모두`TestGroup`과`TestCases`를 지정하면 IDT는 테스트 그룹에서 지정된 테스트 케이스를 실행합니다.

- 일시TestCases는 지정되지만TestGroup가 지정되지 않은 경우 IDT는 지정된 테스트 케이스를 실행합니다.
- 일시TestGroup이 지정되지만TestCases를 지정하지 않으면 IDT는 지정된 테스트 그룹 내에서 모든 테스트 사례를 실행합니다.
- 둘 다TestGroup또는TestCases를 지정하면 IDT는 테스트 러너가 IDT CLI에서 선택하는 테스트 그룹에서 모든 테스트 케이스를 실행합니다. 테스트 러너에 대해 그룹 선택을 활성화하려면 두 가지를 모두 포함해야 합니다.RunTask과Choice귀하의 상태state_machine.json파일. 이 기능의 작동 방식에 대한 예는 단원을 참조하십시오.상태 머신 예제: 사용자가 선택한 테스트 그룹 실행.

  테스트 러너에 IDT CLI 명령을 활성화하는 방법에 대한 자세한 내용은 단원을 참조하십시오.the section called “IDT CLI 명령을 활성화합니다”.

ResultVar

테스트 실행 결과와 함께 설정할 컨텍스트 변수의 이름입니다. 에 대한 값을 지정하지 않은 경우 이 값을 지정하지 마십시오.TestGroup. IDT는 사용자가 정의한 변수의 값을 설정합니다.ResultVar에true또는false다음을 기준으로 합니다.

- 변수 이름이 형식인 경우*text*_*text*_passed를 설정하면 첫 번째 테스트 그룹의 모든 테스트가 통과되었는지 또는 건너뛰었는지 여부에 따라 값이 설정됩니다.
- 다른 모든 경우에는 값이 모든 테스트 그룹의 모든 테스트가 통과되었는지 또는 건너뛰었는지 여부로 설정됩니다.

일반적으로 를 사용합니다.RunTaskstate 는 개별 테스트 케이스 ID를 지정하지 않고 테스트 그룹 ID를 지정하여 IDT가 지정된 테스트 그룹의 모든 테스트 케이스를 실행하도록 합니다. 이 상태에서 실행되는 모든 테스트 케이스는 무작위로 병렬로 실행됩니다. 그러나 모든 테스트 케이스에서 장치를 실행해야 하고 단일 장치만 사용할 수 있는 경우 테스트 케이스가 순차적으로 실행됩니다.

오류 처리

지정된 테스트 그룹 또는 테스트 케이스 ID가 유효하지 않은 경우 이 상태는RunTaskError실행 오류. 상태에 실행 오류가 발생하면hasExecutionError상태 머신 컨텍스트의 변수true.

Choice

이Choicestate를 사용하면 사용자 정의 조건에 따라 변환할 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`

에 정의된 표현식이 없는 경우 전환될 기본 상태`Choices`다음과 같이 평가할 수 있습니다.`true`.

`FallthroughOnError`

선택 사항. 표현식을 평가할 때 상태가 오류가 발생하는 경우의 동작을 지정합니다. 로 설정`true`평가에서 오류가 발생하는 경우 표현식을 건너뛰려는 경우 일치하는 표현식이 없으면 상태 머신이`Default`시/도. 만약`FallthroughOnError`값이 지정되지 않은 경우 기본값은`false`.

`Choices`

현재 상태에서 작업을 실행한 후 전환할 상태를 결정하는 표현식 및 상태의 배열입니다.

`Choices.Expression`

부울 값으로 평가되는 표현식 문자열입니다. 표현식이 다음과 같이 평가되는 경우`true`를 누른 다음 상태 머신이 에 정의된 상태로 전환됩니다.`Choices.Next`. 표현식 문자열은 상태 머신 컨텍스트에서 값을 검색한 다음 해당 값에 대한 연산을 수행하여 부울 값에 도달합니다. 상태 머신 컨텍스트 액세스에 대한 자세한 내용은 단원을 참조하십시오.상태 머신 컨텍스트.

`Choices.Next`

에 정의된 표현식인 경우 전환될 상태의 이름`Choices.Expression`는 로 평가됩니다.`true`.

오류 처리

이`Choice`state는 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.

는 사용할 수 없습니다.Catch이 상태의 오류를 처리하는 블록입니다. 오류가 발생했을 때 상태 머신의 실행을 중지하려면 다음을 설정해야 합니다.FallthroughOnErrorfalse. 하지만 설정하는 것이 좋습니다.FallthroughOnErrortrue를 사용하고 사용 사례에 따라 다음 중 하나를 수행합니다.

- 액세스 중인 변수가 어떤 경우에는 존재하지 않을 것으로 예상되는 경우Default및 추가Choices블록을 사용하여 다음 상태를 지정합니다.
- 액세스하고 있는 변수가 항상 존재해야 하는 경우Default시/도Fail.

Parallel

이Parallelstate를 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 전환될 상태의 이름.

`Branches`

실행할 상태 머신 정의의 배열입니다. 각 상태 시스템 정의에는 자체 시스템 정의가 포함되어야StartAt,Succeed, 및Fail상태입니다. 이 배열의 상태 머신 정의는 자체 정의 외부의 상태를 참조할 수 없습니다.

> **Note**
>
> 각 브랜치 상태 머신은 동일한 상태 머신 컨텍스트를 공유하므로 한 브랜치에서 변수를 설정한 다음 다른 분기에서 해당 변수를 읽으면 예기치 않은 동작이 발생할 수 있습니다.

이Parallelstate는 모든 브랜치 상태 머신을 실행한 후에만 다음 상태로 이동합니다. 디바이스가 필요한 각 상태는 디바이스를 사용할 수 있을 때까지 실행될 때까지 기다립니다. 여러 장치를 사용할 수 있는 경우 이 상태는 여러 그룹의 테스트 케이스를 parallel 실행합니다. 사용 가능한 장치가 충분하지 않으면 테스트 케이스가 순차적으로 실행됩니다. 테스트 케이스는 병렬로 실행될 때 임의의 순서로 실행되기 때문에 다른 장치를 사용하여 동일한 테스트 그룹에서 테스트를 실행할 수 있습니다.

오류 처리

브랜치 상태 머신과 상위 상태 머신이 모두Fail실행 오류를 처리하기 위한 state입니다.

분기 상태 머신은 상위 상태 머신에 실행 오류를 전송하지 않으므로Catch블록 - 브랜치 상태 머신에서 실행 오류를 처리합니다. 대신 를 사용합니다.hasExecutionErrors공유 상태 머신 컨텍스트의 값입니다. 이 기능의 작동 방식에 대한 예는 단원을 참조하십시오.상태 머신 예제: 두 개의 테스트 그룹을 병렬로 실행.

추가 제품기능

이AddProductFeaturesstate를 사용하면 제품 기능을awsiotdevicetester_report.xmlIDT에 의해 생성된 파일입니다.

제품 기능은 장치가 충족할 수 있는 특정 기준에 대한 사용자 정의 정보입니다. 예를 들어 는MQTT제품 기능은 장치가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. 보고서에서 제품 기능은 다음과 같이 설정됩니다.supported,not-supported또는 지정된 테스트가 통과되었는지 여부에 따라 사용자 지정 값

> **Note**
>
> 이AddProductFeaturesstate 는 자체적으로 보고서를 생성하지 않습니다. 이 상태는 다음 단계로 전환해야 합니다.Reportstate보고서를 생성합니다.

```
{
    "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 전환될 상태의 이름.

`Features`

에 표시할 다양한 제품 기능`awsiotdevicetester_report.xml`파일.

`Feature`

기능의 이름입니다.

`FeatureValue`

선택 사항. 보고서에 사용할 사용자 지정 값 대신`supported`. 이 값을 지정하지 않으면 테스트 결과에 따라 피쳐 값이 로 설정됩니다.`supported`또는`not-supported`.

에 대한 사용자 지정 값을 사용하는 경우`FeatureValue`를 사용하여 서로 다른 조건을 사용하여 동일한 피쳐를 테스트할 수 있으며 IDT는 지원되는 조건에 대한 피쳐 값을 연결합니다. 예를 들어, 다음 발췌문은`MyFeature`두 개의 개별 피쳐 값이 있는 피쳐:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
```

```
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

두 테스트 그룹 모두 통과하면 기능 값이 로 설정됩니다.`first-feature-supported, second-feature-supported`.

`Groups`

선택 사항. 테스트 그룹 ID의 어레이입니다. 기능이 지원되려면 지정된 각 테스트 그룹 내의 모든 테스트를 통과해야 합니다.

`OneOfGroups`

선택 사항. 테스트 그룹 ID의 어레이입니다. 이 기능을 지원하려면 지정된 테스트 그룹 중 하나 이상의 테스트를 통과해야 합니다.

`TestCases`

선택 사항. 테스트 케이스 ID의 어레이입니다. 이 값을 지정하면 다음 사항이 적용됩니다.

- 이 기능을 지원하려면 지정된 모든 테스트 케이스가 통과해야 합니다.
- `Groups`테스트 그룹 ID는 하나만 포함해야 합니다.
- `OneOfGroups`는 지정되지 않아야 합니다.

`IsRequired`

선택 사항. 로 설정`false`보고서에서 이 기능을 선택적 기능으로 표시합니다. 기본값은 `true`입니다.

`ExecutionMethods`

선택 사항. 다음과 일치하는 실행 메서드의 배열`protocol`에 지정된 값`device.json`파일. 이 값을 지정하면 테스트 러너는 다음을 지정해야 합니다.`protocol`보고서에 피쳐를 포함하기 위해 이 배열의 값 중 하나와 일치하는 값입니다. 이 값을 지정하지 않으면 피처가 항상 보고서에 포함됩니다.

이`AddProductFeaturesstate`, 값을 설정해야 합니다.`ResultVar`의`RunTask`다음 값 중 하나를 입력합니다.

- 개별 테스트 사례 ID를 지정한 경우ResultVar에*group-id_test-id*_passed.
- 개별 테스트 사례 ID를 지정하지 않은 경우ResultVar에*group-id*_passed.

이AddProductFeatures상태는 다음과 같은 방식으로 테스트 결과를 확인합니다.

- 테스트 케이스 ID를 지정하지 않은 경우 각 테스트 그룹의 결과는*group-id*_passed상태 머신 컨텍스트의 변수입니다.
- 테스트 사례 ID를 지정한 경우 각 테스트의 결과는*group-id_test-id*_passed상태 머신 컨텍스트의 변수입니다.

오류 처리

이 상태에서 제공된 그룹 ID가 유효한 그룹 ID가 아닌 경우 이 상태는AddProductFeaturesError실행 오류. 상태에 실행 오류가 발생하면hasExecutionErrors상태 머신 컨텍스트의 변수true.

보고서

이Reportstate가 생성됩니다.*suite-name*_Report.xml과awsiotdevicetester_report.xml파일을 생성합니다. 이 상태는 보고서를 콘솔로 스트리밍합니다.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환될 상태의 이름.

항상 다음으로 전환해야 합니다.Report테스트 실행 흐름의 끝을 향하여 테스트 러너가 테스트 결과를 볼 수 있도록 합니다. 일반적으로 이 상태 이후의 다음 상태는Succeed.

오류 처리

이 상태에서 보고서 생성에 문제가 발생하면ReportError실행 오류.

로그 메시지

이LogMessagestate가 생성됩니다.test_manager.log콘솔에 로그 메시지를 파일링하고 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 전환될 상태의 이름.

`Level`

로그 메시지를 작성할 오류 수준입니다. 유효하지 않은 레벨을 지정하면 이 상태에서 오류 메시지가 생성되고 무시됩니다.

`Message`

기록할 메시지입니다.

그룹 선택

이SelectGroupstate 는 상태 머신 컨텍스트를 업데이트하여 선택된 그룹을 나타냅니다. 이 상태로 설정된 값은 이후의 모든 경우에 사용됩니다.Choice상태입니다.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Next`

현재 상태에서 작업을 실행한 후 전환될 상태의 이름.

`TestGroups`

선택된 것으로 표시될 테스트 그룹 배열입니다. 이 배열의 각 테스트 그룹 ID에 대해*group-id*_selected변수가 로 설정되어 있습니다.`true`는 컨텍스트에서 입니다. IDT가 지정된 그룹이 있는지 여부를 검증하지 않으므로 유효한 테스트 그룹 ID를 제공해야 합니다.

Fail

이`Fail`state 는 상태 머신이 올바르게 실행되지 않았음을 나타냅니다. 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
    "Type": "Fail"
}
```

Succeed

이`Succeed`state는 상태 머신이 올바르게 실행되었음을 나타냅니다. 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
    "Type": "Succeed"
}
```

상태 머신 컨텍스트

상태 머신 컨텍스트는 실행 중에 상태 머신에서 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 상태 머신 컨텍스트는 상태 머신에서만 액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어, 테스트 러너에 의해 구성된 정보를`userdata.json`파일을 사용하여 특정 테스트를 실행해야 하는지 여부를 확인합니다.

상태 시스템 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 장치 풀에 대한 정보입니다. 선택한 디바이스 풀의 경우 이 정보는 에 정의된 해당 최상위 디바이스 풀 배열 요소에서 검색됩니다.`device.json`파일.

`userData`

의 정보`userdata.json`파일.

`config`

정보 핀`config.json`파일.

`suiteFailed`

이 값은 로 설정됩니다.`false`상태 머신이 시작될 때. 에서 테스트 그룹이 실패하는 경우`RunTask`state이면 이 값이 로 설정됩니다.`true`상태 시스템 실행의 나머지 기간 동안.

`specificTestGroups`

테스트 러너가 전체 테스트 세트 대신 실행할 특정 테스트 그룹을 선택하면 이 키가 만들어지고 특정 테스트 그룹 ID 목록이 포함됩니다.

`specificTestCases`

테스트 러너가 전체 테스트 세트 대신 실행할 특정 테스트 케이스를 선택하면 이 키가 만들어지고 특정 테스트 사례 ID 목록이 포함됩니다.

`hasExecutionErrors`

상태 머신이 시작되면 종료되지 않습니다. 실행 오류가 발생하는 상태가 있으면 이 변수가 만들어지고 로 설정됩니다.`true`상태 시스템 실행의 나머지 기간 동안.

JsonPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의에서 JSONPath 쿼리의 구문은 다음과 같습니다.{{$.*query*}}. 일부 상태 내에서 JSONPath 쿼리를 자리 표시자 문자열로 사용할 수 있습니다. IDT는 자리 표시자 문자열을 컨텍스트에서 평가된 JSONPath 쿼리 값으로 바꿉니다. 다음 값에 자리 표시자를 사용할 수 있습니다.

- 이TestCases가치RunTask상태입니다.
- 이Expression값Choice시/도.

상태 시스템 컨텍스트에서 데이터에 액세스할 때 다음 조건을 충족해야 합니다.

- JSON 경로는 로 시작해야 합니다.$.
- 각 값은 문자열, 숫자 또는 부울로 평가되어야 합니다.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 단원을 참조하십시오.IDT 컨텍스트 사용.

실행 오류

실행 오류는 상태 머신이 상태를 실행할 때 발생하는 상태 머신 정의의 오류입니다. IDT는 각 오류에 대한 정보를 기록합니다.test_manager.log콘솔에 로그 메시지를 파일링하고 스트리밍합니다.

다음 메서드를 사용하여 실행 오류를 처리할 수 있습니다.

- 추가Catch블록상태 정의에서.
- 의 값을 확인하십시오.hasExecutionErrors값상태 시스템 컨텍스트에서.

캐치

를 사용하려면Catch를 사용하여 상태 정의에 다음을 추가합니다.

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

`Catch.ErrorEquals`

포착할 오류 유형의 배열입니다. 실행 오류가 지정된 값 중 하나와 일치하면 상태 머신이 에 지정된 상태로 전환됩니다.`Catch.Next`. 발생하는 오류 유형에 대한 자세한 내용은 각 상태 정의를 참조하십시오.

`Catch.Next`

현재 상태에 지정된 값 중 하나와 일치하는 실행 오류가 발생하는 경우 다음으로 전환할 다음 상태`Catch.ErrorEquals`.

Catch 블록은 하나가 일치할 때까지 순차적으로 처리됩니다. 오류 없음이 Catch 블록에 나열된 것과 일치하지 않으면 상태 머신이 계속 실행됩니다. 실행 오류는 잘못된 상태 정의의 결과이므로 상태에 실행 오류가 발생하면 실패 상태로 전환하는 것이 좋습니다.

하스실행 오류

일부 상태에서 실행 오류가 발생하면 오류를 발생시키는 것 외에도`hasExecutionError`값:`true`상태 머신 컨텍스트에서 이 값을 사용하여 오류 발생 시점을 감지한 다음`Choice`상태 머신을`Fail`시/도.

이 방법에는 다음과 같은 특성이 있습니다.

- 상태 머신이 지정된 값으로 시작하지 않습니다.`hasExecutionError`를 설정하면 특정 상태가 설정될 때까지 이 값을 사용할 수 없습니다. 즉, 를 명시적으로 설정해야 합니다.`FallthroughOnError`에`false`의 경우`Choice`는 실행 오류가 발생하지 않을 경우 상태 시스템이 중지되지 않도록 이 값에 액세스합니다.
- 일단 설정되면`true`,`hasExecutionError`이 (가) false로 설정되거나 컨텍스트에서 제거되지 않습니다. 즉, 이 값은 처음으로 로 설정된 경우에만 유용함을 의미합니다.`true`모든 후속 주에서는 의미 있는 가치를 제공하지 않습니다.
- 이`hasExecutionError`값은 의 모든 브랜치 상태 머신과 공유됩니다.`Parallel`state로, 액세스 순서에 따라 예기치 않은 결과가 발생할 수 있습니다.

이러한 특성 때문에 Catch 블록을 대신 사용할 수 있는 경우 이 메서드를 사용하지 않는 것이 좋습니다.

상태 머신 예제

이 단원에서는 상태 시스템 구성의 예를 들어보겠습니다.

예제

- 상태 머신 예제: 단일 테스트 그룹 실행
- 상태 머신 예제: 사용자가 선택한 테스트 그룹 실행
- 상태 머신 예제: 제품 기능을 사용하여 단일 테스트 그룹 실행
- 상태 머신 예제: 두 개의 테스트 그룹을 병렬로 실행

상태 머신 예제: 단일 테스트 그룹 실행

상태 머신:

- id를 사용하여 테스트 그룹을 실행합니다.`GroupA`, 이 제품군의 제품군의 존재해야 합니다.`group.json`파일.
- 실행 오류 및 전환 여부를 확인합니다.`Fail`발견된 경우
- 보고서 생성 및 전환`Succeed`오류가 없는 경우`Fail`그렇지 않습니다.

```
{
    "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

상태 머신:

- 테스트 러너가 특정 테스트 그룹을 선택했는지 확인합니다. 테스트 러너는 테스트 그룹을 선택하지 않고 테스트 케이스를 선택할 수 없기 때문에 상태 머신은 특정 테스트 케이스를 확인하지 않습니다.
- 테스트 그룹을 선택한 경우:
  - 선택한 테스트 그룹 내에서 테스트 케이스를 실행합니다. 이렇게 하기 위해 상태 머신에서 테스트 그룹 또는 테스트 케이스를 명시적으로 지정하지 않습니다.`RunTask`시/도.
  - 모든 테스트를 실행하고 종료한 후 보고서를 생성합니다.
- 테스트 그룹이 선택되지 않은 경우
  - 테스트 그룹에서 테스트 실행`GroupA`.
  - 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 기능을 사용하여 단일 테스트 그룹 실행

상태 머신:

- 테스트 그룹을 실행합니다.`GroupA`.
- 실행 오류 및 전환 여부를 확인합니다.`Fail`발견된 경우
- 를 추가합니다.`FeatureThatDependsOnGroupA`의 기능 을`awsiotdevicetester_report.xml`파일:
  - 다음의 경우,`GroupA`가공 패스를 수행하면 피쳐가 로 설정됩니다.`supported`.
  - 이 기능은 보고서에서 선택 사항으로 표시되지 않습니다.
- 보고서 생성 및 전환`Succeed`오류가 없는 경우`Fail`그렇지 않으면

```
{
    "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 개의 테스트 그룹을 병렬로 실행

상태 머신:

- 를 실행합니다.GroupA과GroupB테스트 그룹을 parallel. 이ResultVar에 의해 컨텍스트에 저장된 변수RunTask브랜치 상태 머신의 상태를 다음 위치에서 사용할 수 있습니다.AddProductFeatures시/도.
- 실행 오류 및 전환 여부를 확인합니다.Fail발견된 경우 이 상태 머신은 a를 사용하지 않습니다.Catch이 메서드가 분기 상태 시스템에서 실행 오류를 감지하지 못하기 때문에 block.
- 에 피처를 추가합니다.awsiotdevicetester_report.xml전달된 그룹을 기반으로 하는 파일
  - 다음의 경우,GroupA가공 패스를 수행하면 피쳐가 로 설정됩니다.supported.

  - 이 기능은 보고서에서 선택 사항으로 표시되지 않습니다.
- 보고서 생성 및 전환Succeed오류가 없는 경우Fail그렇지 않으면

장치 풀에 두 개의 장치가 구성된 경우 둘 다GroupA과GroupB동시에 실행될 수 있습니다. 하지만 중 하나일 경우GroupA또는GroupB여러 테스트가 있으면 두 장치 모두 해당 테스트에 할당 될 수 있습니다. 하나의 장치만 구성된 경우 테스트 그룹이 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 사례 실행 파일을 생성하여 테스트 도구 모음 폴더에 배치할 수 있습니다.

- `test.json`파일의 인수 또는 환경 변수를 사용하여 실행할 테스트를 결정하는 테스트 도구 모음의 경우 전체 테스트 도구 모음에 대해 실행 가능한 단일 테스트 사례 또는 테스트 도구 모음의 각 테스트 그룹에 대한 테스트 실행 파일을 만들 수 있습니다.

- 지정된 명령을 기반으로 특정 테스트를 실행하려는 테스트 도구 모음의 경우 테스트 도구 모음의 각 테스트 사례에 대해 실행 가능한 테스트 사례를 하나 생성합니다.

테스트 작성자는 사용 사례에 적합한 접근 방식을 결정하고 그에 따라 테스트 케이스 실행 파일을 구성할 수 있습니다. 각`test.json` 파일에 올바른 테스트 케이스 실행 파일 경로를 제공하고 지정된 실행 파일이 올바르게 실행되는지 확인하십시오.

모든 기기에서 테스트 케이스를 실행할 준비가 되면 IDT는 다음 파일을 읽습니다.

- 선택한 테스트`test.json` 사례에 따라 시작할 프로세스와 설정할 환경 변수가 결정됩니다.
- 테스트 스위트의`suite.json` 경우 설정할 환경 변수가 결정됩니다.

IDT는`test.json` 파일에 지정된 명령과 인수를 기반으로 필수 테스트 실행 프로세스를 시작하고 필요한 환경 변수를 프로세스에 전달합니다.

IDT 클라이언트 SDK 사용

IDT Client SDK를 사용하면 IDT 및 테스트 대상 기기와 상호 작용하는 데 사용할 수 있는 API 명령으로 테스트 실행 파일에 테스트 로직을 작성하는 방법을 간소화할 수 있습니다. IDT는 현재 다음과 같은 SDK를 제공합니다.

- IDT 클라이언트 SDK for
- IDT 클라이언트 SDK for Go
- IDT 클라이언트 SDK for Java

이러한 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

- 장치 상호 작용
- IDT 인터랙션
- 호스트 상호작용

장치 상호 작용

다음 명령을 사용하면 추가 장치 상호 작용 및 연결 관리 기능을 구현하지 않고도 테스트 중인 장치와 통신할 수 있습니다.

`ExecuteOnDevice`

테스트 스위트가 SSH 또는 Docker 셸 연결을 지원하는 장치에서 셸 명령을 실행할 수 있습니다.

`CopyToDevice`

테스트 스위트가 IDT를 실행하는 호스트 시스템의 로컬 파일을 SSH 또는 Docker 셸 연결을 지원하는 기기의 지정된 위치로 복사할 수 있습니다.

`ReadFromDevice`

테스트 스위트가 UART 연결을 지원하는 장치의 직렬 포트에서 읽을 수 있도록 합니다.

> **Note**
>
> IDT는 컨텍스트의 장치 액세스 정보를 사용하여 만든 장치에 대한 직접 연결을 관리하지 않으므로 테스트 사례 실행 파일에서 이러한 장치 상호 작용 API 명령을 사용하는 것이 좋습니다. 그러나 이러한 명령이 테스트 사례 요구 사항을 충족하지 않는 경우 IDT 컨텍스트에서 장치 액세스 정보를 검색하고 이를 사용하여 테스트 도구 모음에서 장치에 직접 연결할 수 있습니다. 직접 연결하려면 테스트 대상 장치`device.connectivity` 및 리소스 장치에 대한`resource.devices.connectivity` 필드 및 필드에서 각각 정보를 검색하십시오. IDT 컨텍스트 사용에 대한 자세한 내용은 단원을 참조하십시오IDT 컨텍스트 사용.

IDT 인터랙션

다음 명령을 사용하면 테스트 스위트가 IDT와 통신할 수 있습니다.

`PollForNotifications`

테스트 도구 모음에서 IDT의 알림을 확인할 수 있습니다.

`GetContextValue` 및 `GetContextString`

테스트 스위트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 정보는 IDT 컨텍스트 사용을 참조하세요.

`SendResult`

테스트 도구 모음에서 테스트 사례 결과를 IDT에 보고할 수 있습니다. 이 명령은 테스트 스위트의 각 테스트 케이스 끝에서 호출해야 합니다.

호스트 상호작용

다음 명령을 사용하면 테스트 스위트가 호스트 컴퓨터와 통신할 수 있습니다.

`PollForNotifications`

테스트 도구 모음에서 IDT의 알림을 확인할 수 있습니다.

`GetContextValue` 및 `GetContextString`

테스트 스위트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 정보는 IDT 컨텍스트 사용을 참조하세요.

`ExecuteOnHost`

테스트 스위트가 로컬 시스템에서 명령을 실행할 수 있도록 하고 IDT가 테스트 케이스 실행 가능 수명 주기를 관리할 수 있도록 합니다.

IDT CLI 명령을 활성화합니다

IDT CLI`run-suite` 명령은 테스트 실행기가 테스트 실행을 사용자 지정할 수 있는 몇 가지 옵션을 제공합니다. 테스트 실행자가 이러한 옵션을 사용하여 사용자 지정 테스트 도구 모음을 실행할 수 있도록 IDT CLI에 대한 지원을 구현해야 합니다. 지원을 구현하지 않아도 테스트 러너는 여전히 테스트를 실행할 수 있지만 일부 CLI 옵션은 제대로 작동하지 않습니다. 이상적인 고객 경험을 제공하려면 IDT CLI의`run-suite` 명령에 대한 다음 인수에 대한 지원을 구현하는 것이 좋습니다.

`timeout-multiplier`

테스트를 실행하는 동안 모든 타임아웃에 적용할 1.0보다 큰 값을 지정합니다.

테스트 러너는 이 인수를 사용하여 실행하려는 테스트 케이스의 제한 시간을 늘릴 수 있습니다. 테스트 실행기가`run-suite` 명령에 이 인수를 지정하면 IDT는 이를 사용하여 IDT_TEST_TIMEOUT 환경 변수의 값을 계산하고 IDT 컨텍스트에서`config.timeoutMultiplier` 필드를 설정합니다. 이 주장을 뒷받침하려면 다음을 수행하여야 합니다.

- `test.json`파일의 타임아웃 값을 직접 사용하는 대신 IDT_TEST_TIMEOUT 환경 변수를 읽어 올바르게 계산된 타임아웃 값을 구하십시오.

- IDT 컨텍스트에서`config.timeoutMultiplier` 값을 검색하고 장기 실행 타임아웃에 적용합니다.

  타임아웃 이벤트로 인한 조기 종료에 대한 자세한 내용은 을 참조하십시오종료 동작 지정.

`stop-on-first-failure`

  IDT에서 오류가 발생할 경우 모든 테스트 실행을 중단하도록 지정합니다.

  테스트 실행기가`run-suite` 명령에 이 인수를 지정하면 IDT는 오류가 발생하는 즉시 테스트 실행을 중지합니다. 그러나 테스트 케이스가 parallel 실행되면 예상치 못한 결과가 발생할 수 있습니다. 지원을 구현하려면 IDT에서 이 이벤트가 발생할 경우 테스트 로직에서 실행 중인 모든 테스트 케이스를 중지하고 임시 리소스를 정리한 다음 테스트 결과를 IDT에 보고하도록 지시해야 합니다. 실패 시 조기 종료에 대한 자세한 내용은 단원을 참조하십시오종료 동작 지정.

`group-id` 및 `test-id`

  IDT에서 선택한 테스트 그룹 또는 테스트 사례만 실행하도록 지정합니다.

  테스트 러너는`run-suite` 명령과 함께 이러한 인수를 사용하여 다음과 같은 테스트 실행 동작을 지정할 수 있습니다.

  - 지정된 테스트 그룹 내에서 모든 테스트를 실행합니다.
  - 지정된 테스트 그룹 내에서 선택한 테스트를 실행합니다.

  이러한 인수를 뒷받침하려면 테스트 도구 모음의 테스트 오케스트레이터가 테스트 오케스트레이터에 특정`RunTask` 및`Choice` 상태 집합을 포함해야 합니다. 사용자 지정 상태 머신을 사용하지 않는 경우 기본 IDT 테스트 오케스트레이터에 필요한 상태가 포함되므로 추가 조치를 취할 필요가 없습니다. 그러나 사용자 지정 테스트 오케스트레이터를 사용하는 경우상태 머신 예제: 사용자가 선택한 테스트 그룹 실행 샘플로 사용하여 테스트 오케스트레이터에 필요한 상태를 추가하십시오.

IDT CLI 명령에 대한 자세한 내용은 단원을 참조하십시오사용자 지정 테스트 도구 모음 디버그 및 실행.

이벤트 로그 작성

테스트가 실행되는 동안 콘솔에`stdout` 데이터를 보내고`stderr` 콘솔에 이벤트 로그와 오류 메시지를 기록합니다. 콘솔 메시지의 형식에 대한 자세한 내용은 단원을 참조하십시오콘솔 메시지 형식.

IDT가 테스트 도구 모음 실행을 마치면*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs` 폴더에 있는`test_manager.log` 파일에서도 이 정보를 확인할 수 있습니다.

테스트 대상 장치의 로그를 포함하여 테스트 실행의 로그를*<device-tester-extract-location>*/results/*execution-id*/logs 폴더에 있는*<group-id>*_*<test-id>* 파일에 기록하도록 각 테스트 케이스를 구성할 수 있습니다. 이렇게 하려면testData.logFilePath 쿼리를 사용하여 IDT 컨텍스트에서 로그 파일의 경로를 검색하고 해당 경로에 파일을 만든 다음 원하는 내용을 기록합니다. IDT는 실행 중인 테스트 케이스를 기반으로 경로를 자동으로 업데이트합니다. 테스트 사례에 대한 로그 파일을 만들지 않도록 선택하면 해당 테스트 사례에 대한 파일이 생성되지 않습니다.

텍스트 실행 파일을 설정하여*<device-tester-extract-location>*/logs 폴더에 필요에 따라 추가 로그 파일을 생성할 수도 있습니다. 파일을 덮어쓰지 않도록 로그 파일 이름에 고유한 접두사를 지정하는 것이 좋습니다.

IDT에 결과 보고

IDT는 테스트 결과를awsiotdevicetester_report.xml 및*suite-name*_report.xml 파일에 기록합니다. 이 보고서 파일은 단원을 참조하십시오*<device-tester-extract-location>*/results/*<execution-id>*/. 두 보고서 모두 테스트 스위트 실행 결과를 캡처합니다. IDT가 이러한 보고서에 사용하는 스키마에 대한 자세한 내용은 을 참조하십시오.IDT 테스트 결과 및 로그 검토

*suite-name*_report.xml파일 내용을 채우려면 테스트 실행이 완료되기 전에SendResult 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 컨텍스트의testData.testArtifactsPath 필드에 저장됩니다. 이 폴더에서 IDT는 찾은 첫 번째 알파벳순으로 정렬된 XML 파일을 테스트 결과로 사용합니다.

테스트 로직이 JUnit XML 결과를 생성하는 경우 결과를 구문 분석한 다음 API를 사용하여 IDT에 제출하는 대신 테스트 결과를 아티팩트 폴더의 XML 파일에 기록하여 결과를 IDT에 직접 제공할 수 있습니다.

이 방법을 사용하는 경우 테스트 로직이 테스트 결과를 정확하게 요약하고 결과 파일의 형식을 파일과 동일한 형식으로 지정해야 합니다.*suite-name*_report.xml IDT는 다음 예외를 제외하고 사용자가 제공한 데이터에 대해 어떠한 검증도 수행하지 않습니다.

- IDT는testsuites 태그의 모든 속성을 무시합니다. 대신 보고된 다른 테스트 그룹 결과에서 태그 속성을 계산합니다.

- 안에 하나 이상의testsuite 태그가 있어야 합니다testsuites.

IDT는 모든 테스트 사례에 동일한 아티팩트 폴더를 사용하고 테스트 실행 간에 결과 파일을 삭제하지 않으므로 이 메서드를 사용하면 IDT가 잘못된 파일을 읽을 경우 잘못된 보고가 발생할 수도 있습니다. 모든 테스트 사례에서 생성된 XML 결과 파일에 동일한 이름을 사용하여 각 테스트 사례의 결과를 덮어쓰고 IDT에서 올바른 결과를 사용할 수 있는지 확인하는 것이 좋습니다. 테스트 도구 모음의 보고에는 여러 가지 방법을 사용할 수 있지만, 즉 일부 테스트 사례에는 XML 결과 파일을 사용하고 다른 경우에는 API를 통해 결과를 제출할 수 있지만 이 방법은 권장되지 않습니다.

종료 동작 지정

테스트 케이스에서 실패나 오류 결과가 보고되더라도 항상 종료 코드 0으로 종료되도록 텍스트 실행 파일을 구성하세요. 0이 아닌 종료 코드는 테스트 케이스가 실행되지 않았거나 테스트 케이스 실행 파일이 결과를 IDT에 전달할 수 없는 경우에만 사용하십시오. IDT가 0이 아닌 종료 코드를 수신하면 테스트 케이스에 오류가 발생하여 실행되지 않았음을 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 구성하십시오.

제한 시간

테스트 케이스가test.json 파일에 지정된 타임아웃 값보다 오래 실행될 때 발생합니다. 테스트 실행기가timeout-multiplier 인수를 사용하여 타임아웃 멀티플라이어를 지정한 경우 IDT는 멀티플라이어를 사용하여 타임아웃 값을 계산합니다.

이 이벤트를 감지하려면 IDT_TEST_TIMEOUT 환경 변수를 사용하십시오. 테스트 러너가 테스트를 시작하면 IDT_TEST_TIMEOUT 환경 변수의 값을 계산된 제한 시간 값 (초 단위) 으로 설정하고 변수를 테스트 케이스 실행 파일로 전달합니다. 변수 값을 읽어 적절한 타이머를 설정할 수 있습니다.

인터럽트

테스트 러너가 IDT를 중단할 때 발생합니다. 예를 들어, 키를 누릅니다Ctrl+C.

터미널은 신호를 모든 하위 프로세스에 전달하므로 테스트 케이스에서 신호 처리기를 구성하여 인터럽트 신호를 감지할 수 있습니다.

또는 주기적으로 API를 폴링하여PollForNotifications API 응답의CancellationRequested 불리언 값을 확인할 수 있습니다. IDT는 인터럽트 신호를 수신하면CancellationRequested 부울 값을 로 설정합니다true.

첫 번째 장애 시 중지

현재 테스트 케이스와 병렬로 실행 중인 테스트 케이스가 실패하고 테스트 실행자가`stop-on-first-failure` 인수를 사용하여 오류가 발생할 경우 IDT가 중지되도록 지정할 때 발생합니다.

이 이벤트를 감지하려면 정기적으로 API를 폴링하여`PollForNotifications` API 응답의`CancellationRequested` 불리언 값을 확인할 수 있습니다. IDT에서 장애가 발생하고 첫 번째 장애 시 중지하도록 구성된 경우`CancellationRequested` 부울 값을 로 설정합니다`true`.

이러한 이벤트가 발생하면 IDT는 현재 실행 중인 테스트 케이스의 실행이 완료될 때까지 5분간 기다립니다. 실행 중인 모든 테스트 사례가 5분 이내에 종료되지 않으면 IDT는 각 프로세스를 강제로 중지합니다. 프로세스가 종료되기 전에 테스트 결과를 받지 못한 경우 IDT는 테스트 케이스를 제한 시간이 초과된 것으로 표시합니다. 가장 좋은 방법은 테스트 케이스에서 이벤트 중 하나가 발생할 때 다음 작업을 수행하도록 하는 것입니다.

1. 일반 테스트 로직 실행을 중지하세요.
2. 테스트 대상 기기의 테스트 아티팩트와 같은 임시 리소스를 모두 정리합니다.
3. 테스트 실패 또는 오류와 같은 테스트 결과를 IDT에 보고합니다.
4. 출구.

IDT 컨텍스트 사용

IDT가 테스트 스위트를 실행하면 테스트 스위트는 각 테스트 실행 방법을 결정하는 데 사용할 수 있는 데이터 집합에 액세스할 수 있습니다. 이 데이터를 IDT 컨텍스트라고 합니다. 예를 들어, 테스트 러너가 제공하는 사용자 데이터 구성`userdata.json`IDT 컨텍스트에서 테스트 스위트에 파일을 사용할 수 있습니다.

IDT 컨텍스트는 읽기 전용 JSON 문서로 간주될 수 있습니다. 테스트 스위트는 객체, 배열, 숫자 등과 같은 표준 JSON 데이터 유형을 사용하여 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`

다음 정보config.json파일. 이config필드에는 다음과 같은 추가 필드도 포함됩니다.

`config.timeoutMultiplier`

테스트 스위트에서 사용하는 모든 제한 시간 값에 대한 승수입니다. 이 값은 IDT CLI의 테스트 러너에 의해 지정됩니다. 기본값은 1입니다.

`device`

테스트 실행을 위해 선택한 장치에 대한 정보입니다. 이 정보는devices배열 요소device.json파일선택한 장치에 대해.

`devicePool`

테스트 실행을 위해 선택한 장치 풀에 대한 정보입니다. 이 정보는 에 정의된 최상위 디바이스 풀 배열 요소와 동일합니다.device.json선택한 장치 풀의 파일입니다.

`resource`

리소스 디바이스에 대한 정보`resource.json`파일.

`resource.devices`

이 정보는`devices`배열이 정의됨`resource.json`파일. 각`device`element는 다음 추가 필드를 포함합니다.

`resource.device.name`

리소스 디바이스의 이름입니다. 이 값은`requiredResource.name`의 가치`test.json`파일.

`testData.awsCredentials`

이AWS테스트에서 연결하기 위해 사용하는 자격 증명AWS클라우드. 이 정보는 다음을 통해 얻을 수 있습니다.`config.json`파일.

`testData.logFilePath`

테스트 사례에서 로그 메시지를 기록하는 로그 파일의 경로입니다. 이 파일이 없는 경우 테스트 스위트에서 이 파일을 생성합니다.

`userData`

테스트 러너가 제공하는 정보`userdata.json`파일.

컨텍스트에서 데이터 액세스

JSON 파일 및 텍스트 실행 파일에서 JSONPath 표기법을 사용하여 컨텍스트를 쿼리 할 수 있습니다.`GetContextValue`과`GetContextString`API. IDT 컨텍스트에 액세스하는 JSONPath 문자열의 구문은 다음과 같이 다릅니다.

- `suite.json`과`test.json`, 를 사용합니다.`{{`*`query`*`}}`. 즉, 루트 요소를 사용하지 마십시오.`$.`표현을 시작합니다.
- `test_orchestrator.yaml`, 를 사용합니다.`{{`*`query`*`}}`.

  더 이상 사용되지 않는 상태 머신을 사용하는 경우`state_machine.json`, 를 사용합니다.`{{$.`*`query`*`}}`.
- API 명령에서는 다음을 사용합니다.*`query`*또는`{{$.`*`query`*`}}`명령에 따라 다릅니다. 자세한 내용은 SDK의 인라인 설명서 단원을 참조하십시오.

다음 표에서는 일반적인 JSONPath 표현식의 연산자에 대해 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.
`.### ##`	Accesses the child element with name `### # ##` 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 access the `awsRegion` value in the `config` object is `$.##.AWS##` .
`[##:#]`	Filters elements from an array, retrieving items beginning from the `##` index and going up to the `##` index, both inclusive.
`[###1, ###2,..., ###N]`	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 사용합니다.

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

이 구문에서:

- `jsonpath`는 표준 JSON 구문을 사용하는 JSONPath입니다.
- `value`은 표준 JSON 구문을 사용하는 모든 사용자 정의 값입니다.
- `operator`은 (는) 다음 연산자 중 하나입니다.
  - <(작음)

- <=(작거나 같음)
- ==(같음)

  표현식의 JSONPath 또는 값이 배열, 부울 또는 객체 값이면 사용할 수 있는 유일한 지원 이진 연산자입니다.
- >=(크거나 같음)
- >(큼)
- =~(정규 표현식 일치). 필터 표현식에서 이 연산자를 사용하려면 표현식 왼쪽의 JSONPath 또는 값이 문자열로 평가되고 오른쪽은 다음 패턴 값이어야 합니다.RE2 구문.

{{형식으로 JsonPath 쿼리를 사용할 수 있습니다.*##*}}} 의 자리 표시자 문자열 로`args`과`environmentVariables`필드`test.json`파일 및 내부`environmentVariables`필드`suite.json`파일을 생성합니다. IDT는 컨텍스트 조회를 수행하고 쿼리의 평가된 값으로 필드를 채웁니다. 예:`suite.jsonfile`, 자리 표시자 문자열을 사용하여 각 테스트 케이스에 따라 변경되는 환경 변수 값을 지정할 수 있으며 IDT는 환경 변수를 각 테스트 사례에 대한 올바른 값으로 채 웁니다. 그러나 에서 자리 표시자 문자열을 사용하는 경우`test.json`과`suite.jsonfiles`의 경우 다음 고려 사항이 쿼리에 적용됩니다.

- 각 발생은 다음과 같습니다.`devicePool`쿼리의 키는 모두 소문자로 표시됩니다. 즉, 사용`devicepool`대신.
- 배열의 경우 문자열 배열만 사용할 수 있습니다. 또한 어레이는 비표준이 아닙니다.`item1, item2,...,itemN`형식 배열에 요소가 하나만 포함되어 있으면 다음과 같이 직렬화됩니다.`item`문자열 필드와 구별 할 수 없게 만듭니다.
- 자리 표시자를 사용하여 컨텍스트에서 객체를 검색할 수 없습니다.

이러한 고려 사항으로 인해 가능하면 API를 사용하여 의 자리 표시자 문자열 대신 테스트 논리의 컨텍스트에 액세스하는 것이 좋습니다.`test.json`과`suite.json`파일을 생성합니다. 그러나 경우에 따라 JSONPath 자리 표시자를 사용하여 단일 문자열을 검색하여 환경 변수로 설정하는 것이 더 편리할 수 있습니다.

테스트 러너를 위한 설정 구성

사용자 지정 테스트 제품군을 실행하려면 테스트 실행기가 실행하려는 테스트 제품군을 기반으로 설정을 구성해야 합니다. 설정은 *<device-tester-extract-location>*`/configs/` 폴더에 있는 구성 파일 템플릿을 기반으로 지정됩니다. 필요한 경우 테스트 실행기는 IDT가 AWS 클라우드에 연결하는 데 사용할 AWS 보안 인증 정보도 설정해야 합니다.

테스트 작성자는 테스트 제품군을 디버깅하도록 이러한 파일을 구성해야 합니다. 테스트 실행기가 테스트 제품군을 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 실행기는 *<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 사항으로, 디바이스의 지원되는 기능이 포함된 배열입니다. 디바이스 기능은 테스트 제품군에서 구성한 사용자 정의 값입니다. `device.json` 파일에 포함할 기능 이름 및 값에 대한 정보를 테스트 실행기에게 제공해야 합니다. 예를 들어, 다른 디바이스의 MQTT 서버 역할을 하는 디바이스를 테스트하려는 경우 이름이 `MQTT_QOS`로 지정된 기능에 대해 지원되는 특정 수준을 검

증하도록 테스트 로직을 구성할 수 있습니다. 테스트 실행기는 이 기능 이름을 제공하고 기능 값을 디바이스에서 지원하는 QOS 수준으로 설정합니다. 쿼리를 사용하여 IDT 컨텍스트에서 또는 `devicePool.features` 쿼리를 사용하여 테스트 오케스트레이터 컨텍스트에서 제공된 정보를 검색할 수 있습니다. `pool.features`

`features.name`

특성의 이름입니다.

`features.value`

지원되는 기능 값입니다.

`features.configs`

필요한 경우 기능에 대한 구성 설정입니다.

`features.config.name`

구성 설정의 이름입니다.

`features.config.value`

지원되는 설정값입니다.

`devices`

풀에서 테스트할 디바이스의 배열입니다. 최소 1개의 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 물리적 디바이스의 경우 `ssh` 및 `uart`, Docker 컨테이너의 경우 `docker`뿐입니다.

`connectivity.ip`

테스트 대상 디바이스의 IP 입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항으로, SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 22입니다.

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 사항으로, 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 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.containerUser`

선택 사항으로, 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 22입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

> **Note**
>
> 테스트 실행기가 테스트를 위해 잘못된 기기 연결을 구성했는지 확인하려면 테스트 오케스트레이터 `pool.Devices[0].Connectivity.Protocol` 컨텍스트에서 검색한 다음 해당 상태를 예상한 값과 비교할 수 있습니다. `Choice` 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 제품군에 필요하지만 `device.json` 파일에 지정되지 않은 모든 추가 정보가 들어 있습니다. 이 파일의 형식은 테스트 제품군에 정의된 `userdata_scheme.json` 파일에 따라 달라집니다. 테스트 작성자인 경우, 이 정보를 작성한 테스트 제품군을 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 실행기는 resource.json 파일을 사용하여 필수 기능을 갖춘 리소스 디바이스 풀을 제공합니다. 이 정보를 작성한 테스트 제품군을 실행할 사용자에게 제공해야 합니다.

테스트 실행기는 *<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-version>",
                "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 사항으로, 디바이스의 지원되는 기능이 포함된 배열입니다. 이 필드에 필요한 정보는 테스트 제품군의 test.json 파일에 정의되어 있으며, 실행할 테스트와 이 테스트를 실행하는 방법을 결정합니다. 테스트 제품군에 기능이 필요하지 않은 경우에는 이 필드가 필요하지 않습니다.

`features.name`

기능의 이름입니다.

`features.version`

기능 버전입니다.

`features.jobSlots`

디바이스를 동시에 사용할 수 있는 테스트 수를 나타내는 설정입니다. 기본 값은 1입니다.

`devices`

풀에서 테스트할 디바이스의 배열입니다. 최소 1개의 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 물리적 디바이스의 경우 ssh 및 uart, Docker 컨테이너의 경우 docker뿐입니다.

connectivity.ip

테스트 대상 디바이스의 IP 입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

connectivity.port

선택 사항으로, SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 22입니다.

이 속성은 connectivity.protocol이 ssh로 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.serialPort`

선택 사항으로, 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 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.containerUser`

선택 사항으로, 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 22입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

(선택 사항) config.json 구성

`config.json` 파일 형식의 IDT용 구성 정보가 들어 있습니다. 일반적으로 테스트 실행기는 IDT에 대한 AWS 사용자 보안 인증 정보 및 선택적으로 AWS 리전을 제공하는 경우를 제외하고는 이 파일을 수정할 필요가 없습니다. 필요한 권한이 있는 AWS 보안 인증 정보가 제공되면 AWS IoT Device Tester는 사용량 지표를 수집하여 AWS에 제출합니다. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. 자세한 설명은 IDT 사용량 지표 섹션을 참조하세요.

테스트 실행기는 다음 방법 중 하나로 AWS 보안 인증 정보를 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 인증 정보를 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 인증 정보를 구성하기 위해 테스트 실행기는 *<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**
>
> *# ### ## ### < ># #### #####. device-tester-extract-location*

`log.location`

< device-tester-extract-location ># 있는 로그 폴더의 경로입니다.

`configFiles.root`

구성 파일이 포함된 폴더 경로입니다.

`configFiles.device`

`device.json` 파일 경로입니다.

`testPath`

테스트 제품군이 들어 있는 폴더의 경로입니다.

`reportPath`

IDT에서 테스트 제품군을 실행한 후 테스트 결과를 포함할 폴더의 경로입니다.

`awsRegion`

선택 사항으로, 테스트 제품군에서 사용할 AWS 리전입니다. 설정하지 않으면 테스트 제품군은 각 테스트 제품군에 지정된 기본 리전을 사용합니다.

`auth.method`

IDT가 AWS 보안 인증 정보를 검색하는 데 사용하는 메서드입니다. 지원되는 값은 보안 인증 파일에서 보안 인증 정보를 검색하는 `file`과 환경 변수를 사용하여 보안 인증 정보를 검색하는 `environment`입니다.

`auth.credentials.profile`

보안 인증 파일에서 사용할 보안 인증 프로필입니다. 이 속성은 `auth.method`이 `file`로 설정된 경우에만 적용됩니다.

사용자 지정 테스트 도구 모음 디버그 및 실행

필요한 구성이 설정되면 IDT에서 테스트 제품군을 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 제품군은 디바이스와 상호 작용하고, 컨텍스트를 제공하고, 결과를 받기 위해 IDT를 사용하기 때문에 IDT 상호 작용 없이 IDE에서 테스트 제품군을 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 제품군을 시작하거나 테스트 오케스트레이터를 실행하지 않습니다. 대신 IDE와 상호 작용하여 IDE에서 실행 중인 테스트 제품군의 요청에 응답하고 콘솔에 로그를 인쇄합니다. IDT는 타임아웃이 발생하지 않으며 수동으로 중단될 때까지 종료를 기다립니다. 디버그 모드에서도 IDT는 테스트 오케스트레이터를 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 제품군을 디버깅하려면 다음 단계를 완료합니다.

1. 테스트 제품군을 실행하는 데 필요한 설정 구성 파일을 생성합니다. 예를 들어, 테스트 제품군에 `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 제품군의 요청을 기다린 다음 요청에 응답합니다. 또한 IDT는 IDT 클라이언트 SDK의 사례 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 제품군을 실행합니다.

   필요한 횟수만큼 테스트 제품군을 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 섹션에서는 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 사례를 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)입니다. 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.
- `test-id`. 실행할 테스트 사례(쉼표로 구분된 목록)입니다. 지정된 경우, `group-id`은(는) 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 디바이스 풀입니다. `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 제품군을 실행하는 데 필요한 사용자 데이터 정보가 포함된 파일을 설정합니다. 이는 테스트 제품군 `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 제품군을 디버그 모드에서 실행합니다. 자세한 내용은 IDT를 디버그 모드에서 실행합니다.을(를) 참조하세요.

IDT 테스트 결과 및 로그 검토

이 섹션에서는 IDT가 콘솔 로그 및 테스트 보고서를 생성하는 형식에 대해 설명합니다.

콘솔 메시지 형식

AWS IoT Device Tester에서는 테스트 스위트를 시작할 때 콘솔에 메시지를 인쇄하기 위한 표준 형식을 사용합니다. 다음 발췌문은 IDT에서 생성된 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 구성됩니다.

`time`

기록된 이벤트에 대한 전체 ISO 8601 타임스탬프입니다.

`level`

기록된 이벤트에 대한 메시지 수준입니다. 일반적으로 로깅된 메시지 수준은 다음 중 하나입니다.`info`,`warn`또는`error`. IDT 발행`fatal`또는`panic`메시지가 예상되는 이벤트가 발생하여 일찍 종료되는 경우 메시지를 표시합니다.

`msg`

기록된 메시지입니다.

`executionId`

현재 IDT 프로세스에 대한 고유 ID 문자열입니다. 이 ID는 개별 IDT 실행을 구별하는 데 사용됩니다.

테스트 스위트에서 생성된 콘솔 메시지는 테스트 중인 장치와 테스트 세트, 테스트 그룹 및 IDT가 실행되는 테스트 케이스에 대한 추가 정보를 제공합니다. 다음 발췌문은 테스트 스위트에서 생성된 콘솔 메시지의 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 세트 특정 부분에는 다음 필드가 포함되어 있습니다.

`suiteId`

현재 실행 중인 테스트 제품군의 이름입니다.

`groupId`

현재 실행 중인 테스트 그룹의 ID입니다.

`testCaseId`

실행 중인 테스트 사례의 ID입니다.

`deviceId`

현재 테스트 케이스에서 사용 중인 테스트 대상 장치의 ID입니다.

IDT가 테스트 실행을 완료할 때 콘솔에 테스트 요약을 인쇄하려면 다음을 포함해야 합니다.Reportstate테스트 오케스트레이터에서 테스트 요약에는 테스트 세트, 실행된 각 그룹에 대한 테스트 결과, 생성된 로그 및 보고서 파일의 위치에 대한 정보가 포함되어 있습니다. 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 Device Tester보고서 스키마

awsiotdevicetester_report.xml는 서명된 보고서이며 다음 정보가 포함됩니다.

- IDT 버전
- 테스트 제품군 버전입니다.
- 보고서에 서명하는 데 사용되는 보고서 서명 및 키입니다.
- 에 지정된 디바이스 SKU 및 디바이스 풀 이름device.json파일.
- 테스트된 제품 버전 및 장치 기능
- 테스트 결과의 집계 요약 이 정보는 다음 정보에 포함된 정보와 동일합니다.*suite-name*_report.xml파일.

```
<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. 로 표시됨 기능`required`테스트 스위트에서 디바이스의 유효성을 검사하는 데 필요합니다. 다음 코드 조각은 `awsiotdevicetester_report.xml` 파일에 이 정보가 나타나는 방식을 보여 줍니다.

```
<feature name="ssh" value="supported" type="required"></feature>
```

로 표시됨 기능`optional`은 (는) 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 세트 보고서 스키마

*suite-name*_Result.xml 보고서는 JUnit XML 형식입니다. Jenkins, Bamboo 등과 같은 지속적 통합 및 배포 플랫폼에 이 보고서를 통합할 수 있습니다. 테스트의 집계 요약에는 테스트의 집계 결과가 포함됩니다.

```
<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>
```

두 가지 모두에 있는 보고서 섹션awsiotdevicetester_report.xml또는*suite-name*_report.xml의 목록에는 실행된 테스트와 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 제품군의 실행에 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 있는AWS 자격 증명을 제공하면 사용 지표를AWS IoT Device Tester 수집하여 에AWS 제출합니다. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. IDT는 다음과 같은 정보를 수집합니다.

- AWS 계정IDT를 실행하는 데 사용된 ID
- 테스트를 실행하는 데 사용되는 IDTAWS CLI 명령
- 실행되는 테스트 스위트

- *<device-tester-extract-location >* 폴더의 테스트 스위트
- 디바이스 풀에 구성된 디바이스 수
- 테스트 케이스 이름 및 실행 시간
- 테스트 결과 정보 (예: 테스트 통과, 실패, 오류 발생 또는 건너뛰었는지 여부)
- 제품 기능 테스트
- IDT 탈퇴 행동 (예: 예상치 못한 퇴장 또는 조기 퇴장)

IDT가 보내는 모든 정보는*<device-tester-extract-location>*/results/*<execution-id>*/ 폴더의metrics.log 파일에도 기록됩니다. 로그 파일을 통해 테스트 실행 중에 수집된 정보를 볼 수 있습니다. 이 파일은 사용량 지표를 수집하도록 선택한 경우에만 생성됩니다.

지표 수집을 비활성화하기 위해 추가 조치를 취할 필요는 없습니다. 자격 증명을 저장하지 마십시오.AWS 자격AWS 증명을 저장한 경우 자격 증명에 액세스하도록config.json 파일을 구성하지 마십시오.

AWS 자격 증명 구성

아직 가 (과) 상호 작용할 수 있는AWS 계정 기회가 있다면 받아야 합니다. 이미 계정이AWS 계정 있는 경우 IDT가 사용자를 대신하여 사용 지표를 전송할 수 있도록 계정에 필요한 권한을 구성하기만 하면 됩니다.AWS

1단계: AWS 계정 생성

이 단계에서는 를 생성하고 구성합니다AWS 계정. 이미 가지고AWS 계정 있다면 건너뛰세요the section called “2단계: IDT에 대한 권한 구성”.

AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입

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

   등록 절차 중 전화를 받고 전화 키패드를 사용하여 확인 코드를 입력하는 과정이 있습니다.

   AWS 계정 루트 사용자에 가입하면 AWS 계정 루트 사용자가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스하는 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 태스크를 수행하는 것입니다.

관리자 사용자를 생성하려면 다음 옵션 중 하나를 선택합니다.

관리자를 관리하는 한 가지 방법 선택	To	By	다른 방법
IAM Identity Center에서 (권장)	단기 보안 인증 정보를 사용하여 AWS에 액세스합니다. 이는 보안 모범 사례에 부합됩니다. 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.	AWS IAM Identity Center 사용 설명서의 시작하기에 나온 지침을 따릅니다.	AWS Command Line Interface 사용 설명서에 나온 AWS IAM Identity Center을 사용하도록 AWS CLI 구성 단계를 수행하여 프로그래밍 방식 액세스를 구성합니다.
IAM에서 (권장되지 않음)	장기 보안 인증 정보를 사용하여 AWS에 액세스합니다.	IAM 사용 설명서의 첫 번째 IAM 관리자 및 사용자 그룹 만들기에 나온 지침을 따릅니다.	IAM 사용 설명서에 나온 IAM 사용자의 액세스 키 관리 단계를 수행하여 프로그래밍 방식 액세스를 구성합니다.

2단계: IDT에 대한 권한 구성

이 단계에서는 IDT가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. AWS Management Console또는AWS 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 정책(Policies)을 선택한 후 정책 생성(Create policy)을 선택합니다.

   b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

   c. Review policy(정책 검토)를 선택합니다.

   d. 이름(Name)에 **IDTUsageMetricsIAMPermissions**을 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.

   e. [정책 생성(Create policy)]을 선택합니다.

3. IAM 사용자를 생성하고 사용자에게 권한을 연결합니다.

   a. IAM 사용자를 생성합니다. IAM 사용 설명서의 IAM 사용자 생성 (콘솔) 에 나와 있는 1~5단계를 따르십시오. 이미 IAM 사용자를 생성한 경우에는 다음 단계로 넘어갑니다.

   b. IAM 사용자에 권한 연결:

      i. Set permissions(권한 설정) 페이지에서 Attach existing policies to user directly(사용자에게 직접 기존 정책 연결)를 선택합니다.

      ii. 이전 단계에서 생성한 IDTUsageMetrics IAM 권한 정책을 검색합니다. 확인란을 선택합니다.

   c. Next: Tags(다음: 태그)를 선택합니다.

   d. Next: Review(다음: 검토)를 선택하여 선택 사항의 요약을 봅니다.

   e. 사용자 생성(Create user)을 선택합니다.

   f. 사용자의 액세스 키(액세스 키 ID와 보안 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 사용하여AWS 자격 증명 파일을 구성할 수 있습니다.

IDT에 대한 권한을 구성하려면(AWS CLI)

AWS CLI를 사용하여 AWS IoT Greengrass용 IDT에 대한 권한을 구성하려면 다음 단계를 수행하십시오.

1. 컴퓨터에 AWS CLI를 설치하고 구성합니다(아직 설치되어 있지 않은 경우). AWS Command Line Interface사용 설명서의 설치에AWS CLI 나와 있는 단계를 따르십시오.

   > **Note**
   >
   > AWS CLI은 (과) 상호 작용할AWS 수 있는 오픈 소스 도구입니다.

2. IDT 및AWS IoT Greengrass 역할 관리 권한을 부여하는 다음과 같은 고객 관리형 정책을 생성하십시오.

   Linux 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.

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

3. 에 (과) 상호 작용할 수 있는 권한을 과 (과) 상호 작용할 수 있는 IAM 사용자를 생성합니다AWS IoT Greengrass.

   a. IAM 사용자를 생성합니다.

   ```
   aws iam create-user --user-name user-name
   ```

   b. 생성한IDTUsageMetricsIAMPermissions 정책을 IAM 사용자에게 연결합니다. ### ### # IAM 사용자 이름으로 바꾸고 <account-id>명령에서 사용자의 ID로 바꾸십시오AWS 계정.

   ```
   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 사용하여AWS 자격 증명 파일을 구성할 수 있습니다.

IDT에AWS 자격 증명 제공

IDT가AWS 사용자 자격 증명에 액세스하고 지표를 제출하도록AWS 허용하려면 다음을 수행하십시오.

1. IAM 사용자의AWS 자격 증명을 환경 변수나 자격 증명 파일로 저장합니다.

   a. 환경 변수를 사용하려면 다음 명령을 실행합니다.

      Linux or Unix

      ```
      export AWS_ACCESS_KEY_ID=access-key
      export AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      Windows Command Prompt (CMD)

      ```
      set AWS_ACCESS_KEY_ID=access-key
      set AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      PowerShell

      ```
      $env:AWS_ACCESS_KEY_ID="access-key"
      $env:AWS_SECRET_ACCESS_KEY="secret-access-key"
      ```

   b. 자격 증명 파일을 사용하려면`~/.aws/credentials` 파일에 다음 정보를 추가합니다.

      ```
      [profile-name]
      aws_access_key_id=access-key
      aws_secret_access_key=secret-access-key
      ```

2. `config.json`파일의`auth` 섹션을 구성합니다. 자세한 정보는 (선택 사항) config.json 구성 단원을 참조하세요.

에 대한 IDT 문제 해결AWS IoT GreengrassV2

IDT를 위한AWS IoT GreengrassV2는 오류 유형에 따라 다양한 위치에 오류를 기록합니다. IDT는 콘솔, 로그 파일 및 테스트 보고서에 오류를 기록합니다.

오류를 찾을 수 있는 위치

테스트가 실행되는 동안 콘솔에 높은 수준의 오류가 표시되고, 모든 테스트가 완료되면 실패한 테스트의 요약이 표시됩니다.`awsiotdevicetester_report.xml`테스트 실패의 원인이 된 모든 오류의 요약이 들어 있습니다. IDT는 테스트 실행 중에 콘솔에 표시되는 테스트 실행용 UUID가 있는 디렉터리에 각 테스트 실행의 로그 파일을 저장합니다.

IDT 테스트 로그 디렉터리는 다음과 같습니다.*<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`. 이 디렉토리에는 표에 표시된 다음 파일이 들어 있습니다. 이는 디버깅에 유용합니다.

파일	설명
`test_manager.log`	테스트가 실행되는 동안 콘솔에 기록된 로그입니다. 이 파일 끝에 있는 결과 요약에는 실패한 테스트 목록이 포함되어 있습니다. 이 파일의 경고 및 오류 로그에서 실패에 대한 일부 정보를 확인할 수 있습니다.
test-group-id `/`*test-case-id* `/`*test-name* `.log`	테스트 그룹의 특정 테스트에 대한 세부 로그 Greengrass 구성 요소를 배포하는 테스트의 경우 테스트 케이스 로그 파일이 호출됩니다.`greengrass-test-run.log` .
test-group-id `/`*test-case-id* `/greengrass.log`	에 대한 상세 로그AWS IoT Greengrass코어 소프트웨어. IDT는 설치 테스트를 실행할 때 테스트 대상 기기에서 이 파일을 복사합니다.AWS IoT Greengrass디바이스의 코어 소프트웨어. 이 로그 파일의 메시지에 대한 자세한 내용은 을 참조하십시오.문제 해결 AWS IoT Greengrass V2.
test-group-id `/`*test-case-id*`/`*component-name* `.log`	테스트 실행 중에 배포된 Greengrass 구성 요소에 대한 세부 로그입니다. IDT는 특정 구성 요소

파일	설명
	를 배포하는 테스트를 실행할 때 테스트 대상 장치의 구성 요소 로그 파일을 복사합니다. 각 구성 요소 로그 파일의 이름은 배포된 구성 요소의 이름과 일치합니다. 이 로그 파일의 메시지에 대한 자세한 내용은 을 참조하십시오.문제 해결 AWS IoT Greengrass V2.

다음에 대한 IDT 문제 해결AWS IoT GreengrassV2 오류

IDT를 실행하기 전에AWS IoT Greengrass올바른 구성 파일을 준비하세요. 구문 분석 및 구성 오류가 발생하는 경우 첫 번째 단계는 환경에 적합한 구성 템플릿을 찾아 사용하는 것입니다.

그래도 문제가 발생할 경우 다음 디버깅 프로세스를 참조하십시오.

주제

- 별칭 해상도 오류
- 충돌 오류
- 테스트를 시작할 수 없음 오류
- Docker 자격 이미지에 오류가 있습니다.
- 자격 증명을 읽지 못했습니다.
- 다음과 같은 오류 안내 PreInstalled 그린그래스
- 잘못된 서명 예외
- 기계 학습 자격 오류
- 오픈 테스트 프레임워크 (OTF) 배포 실패
- 구문 분석 오류
- 권한 거부 오류
- 검증 보고서 생성 오류
- 필수 파라미터 누락 오류
- macOS의 보안 예외
- SSH 연결 오류
- 스트림 매니저 자격 오류

- 제한 시간 오류
- 버전 확인 오류

별칭 해상도 오류

사용자 지정 테스트 스위트를 실행하면 콘솔과 테스트 스위트에서 다음 오류가 표시될 수 있습니다.`test_manager.log`.

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

이 오류는 IDT 테스트 오케스트레이터에서 구성된 별칭이 제대로 확인되지 않거나 확인된 값이 구성 파일에 없는 경우 발생할 수 있습니다. 이 오류를 해결하려면 다음을 확인하십시오.`device.json`과`userdata.json` 테스트 스위트에 필요한 올바른 정보를 포함하세요. 에 필요한 구성에 대한 자세한 내용은AWS IoT Greengrass검증은 다음을 참조하십시오.AWS IoT Greengrass 검증 도구 모음을 실행하기 위한 IDT 설정 구성..

충돌 오류

를 실행할 때 다음 오류가 표시될 수 있습니다.AWS IoT Greengrass둘 이상의 장치에서 검증 제품군을 동시에 실행할 수 있습니다.

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-
id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409,
 RequestID: “id” }, Message_: “Component [com.example.IDTHelloWorld : 1.0.0] for
 account [account-id] already exists with state: [DEPLOYABLE]” }
```

에 대한 동시 테스트 실행은 아직 지원되지 않습니다.AWS IoT Greengrass검증 제품군. 각 장치에 대해 검증 제품군을 순차적으로 실행합니다.

테스트를 시작할 수 없음 오류

테스트를 시작하려고 할 때 발생한 실패를 가리키는 오류가 발생할 수 있습니다. 몇 가지 원인이 있을 수 있으므로 다음을 수행하십시오.

- 실행 명령의 풀 이름이 실제로 존재하는지 확인하십시오. IDT는 사용자 계정에서 직접 풀 이름을 참조합니다.`device.json`파일.
- 풀에 있는 디바이스에 올바른 구성 파라미터가 있는지 확인합니다.

Docker 자격 이미지에 오류가 있습니다.

Docker 애플리케이션 관리자 자격 테스트에서는 다음을 사용합니다.amazon/amazon-ec2-metadata-mockAmazon ECR의 컨테이너 이미지를 사용하여 테스트 대상 디바이스를 검증합니다.

테스트 대상 디바이스의 Docker 컨테이너에 이미지가 이미 있는 경우 다음 오류가 발생할 수 있습니다.

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

이전에 이 이미지를 다운로드하고 실행한 경우amazon/amazon-ec2-metadata-mock디바이스의 컨테이너, 검증 테스트를 실행하기 전에 테스트 대상 디바이스에서 이 이미지를 제거해야 합니다.

자격 증명을 읽지 못했습니다.

Windows 장치를 테스트할 때 다음과 같은 문제가 발생할 수 있습니다.Failed to read credential에 오류가 발생했습니다.greengrass.log테스트 대상 장치에 연결하는 데 사용하는 사용자가 해당 장치의 자격 증명 관리자에 설정되어 있지 않은 경우 파일을 저장합니다.

이 오류를 해결하려면 테스트 대상 장치의 자격 증명 관리자에서 IDT 사용자의 사용자 및 암호를 구성하십시오.

자세한 정보는 Windows 장치의 사용자 자격 증명을 구성합니다.을 참조하세요.

다음과 같은 오류 안내 PreInstalled 그린그래스

IDT를 실행하는 동안 PreInstalled 그린그래스, 다음과 같은 오류가 발생하는 경우Guice또는ErrorInCustomProvider, 파일이 있는지 확인userdata.json가 있습니다InstalledDirRootOnDevice그린그래스 설치 폴더로 설정되었습니다. IDT가 파일을 확인합니다.effectiveConfig.yaml아래에<InstallationDirRootOnDevice>/config/effectiveConfig.yaml.

자세한 정보는 Windows 장치의 사용자 자격 증명을 구성합니다.을 참조하세요.

잘못된 서명 예외

Lambda 자격 테스트를 실행할 때 다음과 같은 문제가 발생할 수 있습니다.invalidsignatureexceptionIDT 호스트 시스템에서 네트워크 액세스 문제가 발생하는 경우 오류가 발생합니다. 라우터를 재설정하고 테스트를 다시 실행하세요.

기계 학습 자격 오류

기계 학습 (ML) 자격 테스트를 실행할 때 장치가 요구 사항을 충족하지 않으면 자격 인증이 실패할 수 있습니다.요구 사항배포하려면AWS-ML 구성 요소 제공. ML 자격 오류를 해결하려면 다음을 수행하십시오.

- 구성 요소 로그에서 테스트 실행 중에 배포된 구성 요소의 오류 세부 정보를 찾아보세요. 구성 요소 로그는 다음 위치에 있습니다.*<device-tester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>*디렉터리.
- 추가-Dgg.persist=installed.software인수를 에test.json실패한 테스트 케이스에 대한 파일입니다. The test.json파일은 다음 위치에 있습니다.*<device-tester-extract-location>*/tests/GGV2Q_*version* directory.

오픈 테스트 프레임워크 (OTF) 배포 실패

OTF 테스트에서 배포가 완료되지 않는 경우 상위 폴더에 설정된 권한이 원인일 수 있습니다.TempResourcesDirOnDevice과InstallationDirRootOnDevice. 이 폴더의 권한을 올바르게 설정하려면 다음 명령을 실행합니다. 바꾸기*folder-name*상위 폴더의 이름으로

```
sudo chmod 755 folder-name
```

구문 분석 오류

JSON 구성에 오타가 있으면 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 정보만으로도 오류를 수정하는 데 충분하지만 여전히 오류를 찾을 수 없는 경우 IDE, Atom 또는 Sublime과 같은 텍스트 편집기 또는 JSONLint와 같은 온라인 도구를 통해 수동으로 유효성 검사를 수행할 수 있습니다.

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.

검증 보고서 생성 오류

IDT는 네 가지 최신 버전을 지원합니다.*major*.*minor*의 버전AWS IoT Greengrass제출할 수 있는 검증 보고서를 생성하기 위한 V2 검증 세트 (GGV2Q)AWS Partner Network디바이스를 다음 목록에 포함시키려면AWS Partner기기 카탈로그. 이전 버전의 검증 제품군에서는 검증 보고서가 생성되지 않습니다.

지원 정책에 대해 궁금한 점이 있으면 문의하십시오.AWS Support.

필수 파라미터 누락 오류

IDT가 새 기능을 추가하면 구성 파일이 변경될 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 스키마의 유효성을 검사하여 지원되는 최신 버전을 사용하고 있는지 확인합니다.

macOS의 보안 예외

macOS 호스트 컴퓨터에서 IDT를 실행하면 IDT 실행이 차단됩니다. IDT를 실행하려면 IDT 런타임 기능의 일부인 실행 파일에 보안 예외를 부여하십시오. 호스트 컴퓨터에 경고 메시지가 표시되면 해당하는 각 실행 파일에 대해 다음을 수행하십시오.

IDT 실행 파일에 보안 예외 허용하기

1. macOS 컴퓨터의 Apple 메뉴에서 열기시스템 환경설정.
2. 고르세요보안 및 개인 정보그 다음엔일반탭에서 잠금 아이콘을 선택하여 보안 설정을 변경합니다.
3. 차단된 경우`devicetester_mac_x86-64`메시지를 찾아보세요`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`그리고 선택하세요아무튼 허용.
4. 관련된 모든 실행 파일을 검토할 때까지 IDT 테스트를 재개하십시오.

SSH 연결 오류

IDT가 테스트 대상 기기에 연결할 수 없는 경우 연결 실패가 기록됩니다.`/results/<execution-id>/logs/<test-case-id>.log`. 테스트 대상 장치에 연결하는 것은 IDT가 수행하는 첫 번째 작업 중 하나이기 때문에 SSH 메시지는 이 로그 파일의 맨 위에 표시됩니다.

대부분의 Windows 구성에서는 PuTTY 터미널 애플리케이션을 사용하여 Linux 호스트에 연결합니다. 이 애플리케이션을 사용하려면 표준 PEM 개인 키 파일을 PPK라는 전용 Windows 형식으로 변환해야 합니다. SSH를 다음과 같이 구성하는 경우`device.json`파일에는 PEM 파일을 사용하십시오. PPK 파일을 사용하는 경우 IDT는 PPK 파일을 사용하여 SSH 연결을 생성할 수 없습니다.AWS IoT Greengrass기기 및 테스트를 실행할 수 없습니다.

IDT v4.4.0부터 테스트 중인 기기에서 SFTP를 활성화하지 않은 경우 로그 파일에 다음 오류가 표시될 수 있습니다.

```
SSH connection failed with EOF
```

이 오류를 해결하려면 디바이스에서 SFTP를 활성화하세요.

스트림 매니저 자격 오류

스트림 관리자 자격 테스트를 실행하면 다음과 같은 오류가 표시될 수 있습니다.`com.aws.StreamManagerExport.log`파일.

```
Failed to upload data to S3
```

이 오류는 스트림 관리자가 다음을 사용할 때 발생할 수 있습니다.AWS의 자격 증명`~/root/.aws/credentials`IDT가 테스트 대상 기기로 내보내는 환경 자격 증명을 사용하는 대신 기기에 파일을 저

장하세요. 이 문제를 방지하려면 다음을 삭제하십시오.credentials디바이스에서 파일을 저장하고 자격 테스트를 다시 실행하세요.

제한 시간 오류

각 테스트 제한 시간의 기본값에 적용되는 제한 시간 배수를 지정하여 각 테스트의 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. 예시:

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

자세한 내용을 보려면 `run-suite --help`를 실행하십시오.

일부 타임아웃 오류는 구성 문제로 인해 IDT 테스트 케이스를 완료할 수 없을 때 발생합니다. 타임아웃 멀티플라이어를 늘려서는 이러한 오류를 해결할 수 없습니다. 테스트 실행의 로그를 사용하여 기본 구성 문제를 해결하십시오.

- MQTT 또는 Lambda 구성 요소 로그에 다음이 포함된 경우`Access denied`오류, Greengrass 설치 폴더에 올바른 파일 권한이 없을 수 있습니다. 설치 경로에 정의한 설치 경로의 각 폴더에 대해 다음 명령을 실행합니다.`userdata.json`파일.

  ```
  sudo chmod 755 folder-name
  ```

- 그린그래스 로그에 Greengrass CLI 배포가 완료되지 않은 것으로 나타나면 다음을 수행하십시오.
  - 다음을 확인하십시오.`bash`테스트 대상 장치에 설치되었습니다.
  - 만약`userdata.json`파일에 다음이 포함되어 있습니다.`GreengrassCliVersion`구성 파라미터, 제거하세요. 이 매개변수는 IDT v4.1.0 이상 버전에서 더 이상 사용되지 않습니다. 자세한 정보는 userdata.json을 구성합니다.을 참조하세요.
- Lambda 배포 테스트가 “Lambda 게시 검증 중: 시간 초과”라는 오류 메시지와 함께 실패하고 테스트 로그 파일에 오류가 발생한 경우 (`idt-gg2-lambda-function-idt-<resource-id>.log`) 라고 적혀 있습니다.`Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.`, 다음과 같이 하세요.
  - 어떤 폴더가 사용되었는지 확인하십시오.`InstallationDirRootOnDevice`에서`userdata.json`파일.

- 장치에 올바른 사용자 권한이 설정되어 있는지 확인하세요. 자세한 내용은 을 참조하십시오.디바이스의 사용자 권한 구성.

버전 확인 오류

IDT는 다음과 같은 경우 다음 오류를 발생시킵니다.AWSIDT 사용자의 사용자 자격 증명에는 필요한 IAM 권한이 없습니다.

```
Failed to check version compatibility
```

The AWS필요한 IAM 권한이 없는 사용자

에 AWS IoT Device Tester 대한 지원 정책 AWS IoT Greengrass

AWS IoT Device Tester for AWS IoT Greengrass 는 기기 카탈로그에 포함할 AWS IoT Greengrass 기기를 검증하고 검증하는 데 사용되는 테스트 자동화 도구입니다.AWS Partner 장치를 테스트하거나 AWS IoT Device Tester 검증하려면 AWS IoT Greengrass 및 의 최신 버전을 사용하는 것이 좋습니다.

지원되는 각 버전마다 최소 한 가지 버전의 AWS IoT Device Tester 를 사용할 수 있습니다. AWS IoT Greengrass지원되는 버전은 Greengrass 핵 버전을 참조하십시오. AWS IoT Greengrass지원되는 버전은 AWS IoT Device Tester을 참조하십시오. AWS IoT Device TesterAWS IoT Greengrass V2용 지원 버전

AWS IoT Greengrass 및 의 지원되는 모든 버전을 사용하여 장치를 테스트하거나 AWS IoT Device Tester 검증할 수도 있습니다. 지원되지 않는 버전을 계속 사용할 수 있지만 해당 버전에는 AWS IoT Device Tester버그 수정이나 업데이트가 제공되지 않습니다. 지원 정책에 대한 질문이 있는 경우 문의하세요 AWS Support.

그린그래스 기반 IoT 솔루션

유로텍의 에브리웨어는 GreenEdge 현재 프리뷰 릴리즈 중이며 변경될 수 있습니다. AWS IoT Greengrass 이 솔루션은 AWS에서 지원되지 않습니다. 이 장치에 문제가 있는 경우 Eurotech에 문의해야 합니다.

AWS IoT GreengrassGreengrass 설치 경험을 최적화하기 위한 파트너의 솔루션을 제공합니다. 다음은 Eurotech와 파트너십을 AWS 맺고 제공하는 솔루션입니다. 이 솔루션에는 AWS IoT Greengrass 코어 에지 런타임 및 추가 기능이 사전 설치되어 있습니다.

유로텍

AWSEurotech와 제휴하여 AWS IoT Greengrass Core 소프트웨어가 사전 설치된 장치를 찾는 고객에게 IoT 솔루션을 제공합니다. Eurotech의 GreenEdge Everyware는 에서 사전 구성 및 사전 검증한 IoT 에지 소프트웨어입니다. AWS 이 솔루션은 Greengrass와 Eurotech Everyware 소프트웨어 프레임워크(ESF) 의 기능을 통합하여 고객에게 모드버스, OPC-UA 클라이언트/서버, S7, 트윈캣, J1939, DNP3 마스터/아웃스테이션 등과 같은 프로토콜 어댑터를 통해 광범위한 사우스바운드 연결을 제공합니다. 이 솔루션을 사용하면 모든 노스바운드 AWS 서비스 (예:,, Amazon S3 AWS 클라우드 및 Amazon Kinesis Video Streams) 로 AWS IoT Core 데이터를 보내고 연결할 수도 있습니다. AWS IoT SiteWise AWS IoT Analytics 이 솔루션은 Eurotech의 디바이스 관리 솔루션인 Everyware Cloud와 결합하여 디바이스 온보딩 및 대량 배포를 간소화하는 새로운 제로 터치 프로비저닝 서비스를 도입합니다.

유로텍에 대한 자세한 내용은 유로텍을 참조하십시오.

문제 해결 AWS IoT Greengrass V2

이 섹션의 문제 해결 정보와 솔루션을 사용하면 관련 문제를 해결하는 데 도움이 됩니다. AWS IoT Greengrass Version 2

주제

AWS IoT Greengrass Core 소프트웨어 및 구성 요소 로그 보기

AWS IoT Greengrass Core 소프트웨어는 코어 장치에 대한 실시간 정보를 보는 데 사용할 수 있는 로그를 로컬 파일 시스템에 기록합니다. 또한 로그를 로그에 기록하도록 코어 디바이스를 구성하여 CloudWatch 코어 디바이스의 문제를 원격으로 해결할 수 있습니다. 이러한 로그는 구성 요소, 배포 및 핵심 장치와 관련된 문제를 식별하는 데 도움이 될 수 있습니다. 자세한 설명은 모니터 AWS IoT Greengrass 로그 섹션을 참조하세요.

AWS IoT Greengrass 핵심 소프트웨어 문제

AWS IoT Greengrass 코어 소프트웨어 문제를 해결합니다.

주제

- AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 시작할 수 없습니다.
- Nucleus를 시스템 서비스로 설정할 수 없습니다.
- 에 연결할 수 없습니다. AWS IoT Core
- 메모리 부족 오류
- 그린그래스 CLI를 설치할 수 없습니다.
- User root is not allowed to execute
- com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with
- Failed to map segment from shared object: operation not permitted
- Windows 서비스를 설정하지 못했습니다.
- com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
- com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime
- software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid
- software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy
- Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request
- Operation aws.greengrass#<operation> is not supported by Greengrass
- java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed
- java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

코어 디바이스를 설정할 수 없습니다.

AWS IoT Greengrass 코어 소프트웨어 설치 프로그램이 실패하고 코어 장치를 설정할 수 없는 경우 소프트웨어를 제거하고 다시 시도해야 할 수 있습니다. 자세한 설명은 AWS IoT GreengrassCore 소프트웨어 제거 섹션을 참조하세요.

AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 시작할 수 없습니다.

AWS IoT Greengrass Core 소프트웨어가 시작되지 않는 경우 시스템 서비스 로그를 확인하여 문제를 식별하십시오. 한 가지 일반적인 문제는 PATH 환경 변수 (Linux) 또는 PATH 시스템 변수 (Windows) 에서 Java를 사용할 수 없는 경우입니다.

Nucleus를 시스템 서비스로 설정할 수 없습니다.

AWS IoT Greengrass Core 소프트웨어 설치 프로그램이 시스템 AWS IoT Greengrass 서비스로 설정되지 않을 경우 이 오류가 표시될 수 있습니다. Linux 장치에서 이 오류는 일반적으로 코어 장치에 systemd init 시스템이 없는 경우 발생합니다. 설치 프로그램은 시스템 서비스 설정에 실패하더라도 AWS IoT Greengrass 코어 소프트웨어를 성공적으로 설정할 수 있습니다.

다음 중 하나를 수행합니다.

- AWS IoT Greengrass Core 소프트웨어를 시스템 서비스로 구성하고 실행합니다. 의 모든 기능을 사용하려면 소프트웨어를 시스템 서비스로 구성해야 AWS IoT Greengrass 합니다. systemd를 설치하거나 다른 init 시스템을 사용할 수 있습니다. 자세한 설명은 Greengrass 핵을 시스템 서비스로 구성 섹션을 참조하세요.
- 시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어를 실행합니다. 설치 프로그램이 Greengrass 루트 폴더에 설정한 로더 스크립트를 사용하여 소프트웨어를 실행할 수 있습니다. 자세한 설명은 시스템 서비스 없이 AWS IoT Greengrass Core 소프트웨어 실행 섹션을 참조하세요.

에 연결할 수 없습니다. AWS IoT Core

예를 들어 AWS IoT Greengrass Core 소프트웨어를 AWS IoT Core 연결하여 배포 작업을 검색할 수 없는 경우 이 오류가 표시될 수 있습니다. 다음을 따릅니다.

- 코어 장치가 인터넷에 연결할 수 있는지 확인하고 AWS IoT Core. 장치가 연결되는 AWS IoT Core 엔드포인트에 대한 자세한 내용은 을 참조하십시오AWS IoT Greengrass Core 소프트웨어 구성.

- 코어 디바이스의 AWS IoT 사물이`iot:Connect`, `iot:Publish`iot:Receive, 및 `iot:Subscribe` 권한을 허용하는 인증서를 사용하는지 확인하세요.
- 코어 디바이스에서 네트워크 프록시를 사용하는 경우 코어 디바이스에 디바이스 역할이 있고 코어 디바이스의 역할이`iot:Connect`, `iot:Publish`iot:Receive, 및 `iot:Subscribe` 권한을 허용하는지 확인하세요.

메모리 부족 오류

이 오류는 일반적으로 장치의 메모리가 부족하여 Java 힙에 객체를 할당할 수 없는 경우에 발생합니다. 메모리가 제한된 기기에서는 메모리 할당을 제어하기 위해 최대 힙 크기를 지정해야 할 수 있습니다. 자세한 설명은 JVM 옵션으로 메모리 할당을 제어하세요. 섹션을 참조하세요.

그린그래스 CLI를 설치할 수 없습니다.

Core용 AWS IoT Greengrass 설치 명령에서 `--deploy-dev-tools` 인수를 사용할 때 다음 콘솔 메시지가 표시될 수 있습니다.

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
 deployment for Greengrass first party dev tools, please manually create a deployment
 if you wish to
```

이는 코어 디바이스가 기존 배포가 있는 사물 그룹의 멤버이기 때문에 Greengrass CLI 구성 요소가 설치되지 않은 경우에 발생합니다. 이 메시지가 표시되면 Greengrass CLI 구성 요소 (`aws.greengrass.Cli`) 를 디바이스에 수동으로 배포하여 Greengrass CLI를 설치할 수 있습니다. 자세한 설명은 그린그래스 CLI 설치 섹션을 참조하세요.

User root is not allowed to execute

일반적으로 AWS IoT Greengrass `root` Core 소프트웨어를 실행하는 사용자에게 사용자 및 그룹과 `sudo` 함께 실행할 권한이 없는 경우 이 오류가 표시될 수 있습니다. 기본 `ggc_user` 시스템 사용자의 경우 이 오류는 다음과 같습니다.

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

`/etc/sudoers`파일이 사용자에게 다른 `sudo` 그룹으로 실행할 수 있는 권한을 부여하는지 확인하세요. 에서 사용자의 권한은 다음 예와 `/etc/sudoers` 같아야 합니다.

```
root    ALL=(ALL:ALL) ALL
```

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

코어 기기가 구성 요소를 실행하려고 하는데 Greengrass nucleus가 구성 요소를 실행하는 데 사용할 기본 시스템 사용자를 지정하지 않는 경우 이 오류가 표시될 수 있습니다.

이 문제를 해결하려면 구성 요소를 실행하는 기본 시스템 사용자를 지정하도록 Greengrass nucleus를 구성하십시오. 자세한 내용은 구성 요소를 실행하는 사용자를 구성하십시오. 및 기본 구성 요소 사용자 구성 섹션을 참조하세요.

Failed to map segment from shared object: operation not permitted

권한이 있는 `/tmp` `noexec` 폴더가 마운트되어 AWS IoT Greengrass Core 소프트웨어가 시작되지 않는 경우 이 오류가 표시될 수 있습니다. CRT (AWS 공용 런타임) 라이브러리는 기본적으로 이 `/tmp` 폴더를 사용합니다.

다음 중 하나를 수행합니다.

- 다음 명령을 실행하여 `exec` 권한이 있는 `/tmp` 폴더를 다시 마운트하고 다시 시도하십시오.

```
sudo mount -o remount,exec /tmp
```

- Greengrass nucleus v2.5.0 이상을 실행하는 경우 JVM 옵션을 설정하여 CRT 라이브러리가 사용하는 폴더를 변경할 수 있습니다. AWS 배포 시 또는 Core 소프트웨어를 설치할 때 Greengrass nucleus 구성 요소 구성에서 `jvmOptions` 매개변수를 지정할 수 있습니다. AWS IoT Greengrass */path/to/use# CRT ###### ###* 수 있는 폴더 경로로 바꾸십시오. AWS

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

Windows 서비스를 설정하지 못했습니다.

Microsoft Windows 2016 장치에 AWS IoT Greengrass Core 소프트웨어를 설치하는 경우 이 오류가 표시될 수 있습니다. AWS IoT Greengrass Core 소프트웨어는 Windows 2016에서 지원되지 않습니다. 지원되는 운영 체제 목록은 을 참조하십시오지원하는 플랫폼.

Windows 2016을 사용해야 하는 경우 다음 작업을 수행할 수 있습니다.

1. 다운로드한 AWS IoT Greengrass Core 설치 아카이브의 압축을 풉니다.
2. `Greengrass`디렉토리에서 `bin/greengrass.xml.template` 파일을 엽니다.
3. `<autoRefresh>`태그 바로 앞에 있는 파일 끝에 `</service>` 태그를 추가합니다.

```
  </log>
  <autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

루트 CA (인증 기관) 파일 없이 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 이 오류가 표시될 수 있습니다.

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
 service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
 com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

설치 프로그램에 제공하는 구성 파일의 `rootCaPath` 매개 변수와 함께 유효한 루트 CA 파일을 지정했는지 확인하십시오. 자세한 설명은 AWS IoT Greengrass 코어 소프트웨어 설치 섹션을 참조하세요.

com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime

코어 디바이스를 연결하여 배포 작업 알림을 AWS IoT Core 구독할 수 없는 경우 이 경고 메시지가 표시될 수 있습니다. 다음을 따릅니다.

- 코어 디바이스가 인터넷에 연결되어 있고 구성한 AWS IoT 데이터 엔드포인트에 도달할 수 있는지 확인합니다. 코어 디바이스가 사용하는 엔드포인트에 대한 자세한 내용은 을 참조하십시오프록시 또는 방화벽을 통한 장치 트래픽 허용.
- Greengrass 로그에서 다른 근본 원인을 확인할 수 있는 다른 오류가 있는지 확인하십시오.

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

자동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치하고 설치 프로그램이 유효하지 않은 AWS 세션 토큰을 사용하는 경우 이 오류가 표시될 수 있습니다. 다음을 따릅니다.

- 임시 보안 자격 증명을 사용하는 경우 세션 토큰이 올바른지, 전체 세션 토큰을 복사하여 붙여넣고 있는지 확인하십시오.
- 장기 보안 자격 증명을 사용하는 경우 이전에 임시 자격 증명을 사용한 시점의 세션 토큰이 장치에 없는지 확인하십시오. 다음을 따릅니다.

  1. 다음 명령을 실행하여 세션 토큰 환경 변수를 설정 해제합니다.

     Linux or Unix

     ```
     unset AWS_SESSION_TOKEN
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_SESSION_TOKEN=
     ```

     PowerShell

     ```
     Remove-Item Env:\AWS_SESSION_TOKEN
     ```

  2. AWS 자격 증명 파일 에 세션 토큰 이 포함되어 있는지 확인합니다. `~/.aws/credentials` `aws_session_token` 그렇다면 파일에서 해당 줄을 제거하세요.

     ```
     aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
     +FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
     IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
     ```

AWS 자격 증명을 제공하지 않고도 AWS IoT Greengrass Core 소프트웨어를 설치할 수 있습니다. 자세한 내용은 수동 리소스 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치 또는 AWS IoT 플릿 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어 설치를 참조하세요.

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

자동 프로비저닝으로 AWS IoT Greengrass Core 소프트웨어를 설치하고 설치 프로그램이 필요한 권한이 없는 AWS 자격 증명을 사용하는 경우 이 오류가 표시될 수 있습니다. 필요한 권한에 대한 자세한 내용은 을 참조하십시오. 리소스 프로비저닝을 위한 설치 관리자를 위한 최소 IAM 정책

자격 증명의 IAM 자격 증명에 대한 권한을 확인하고 누락된 필수 권한을 IAM ID에 부여하십시오.

Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request

섀도우 관리자 구성 요소를 사용하여 디바이스 섀도우를 동기화할 때 이 오류가 표시될 수 있습니다. AWS IoT Core HTTP 403 상태 코드는 코어 디바이스의 AWS IoT 정책이 호출 `GetThingShadow` 권한을 부여하지 않기 때문에 이 오류가 발생했음을 나타냅니다.

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
 cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
 com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
 name=MyGreengrassCore, shadow name=MyShadow}
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
 software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
 null (Service: IotDataPlane, Status Code: 403, Request ID:
 f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

로컬 섀도우를 동기화하려면 코어 디바이스의 AWS IoT 정책이 다음 권한을 부여해야 합니다. AWS IoT Core

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

코어 디바이스의 AWS IoT 정책을 확인하고 누락된 필수 권한을 추가하십시오. 자세한 내용은 다음 자료를 참조하세요:

- AWS IoT Core AWS IoT 개발자 안내서의 정책 조치

- 코어 디바이스 정책 업데이트 AWS IoT

Operation aws.greengrass#<operation> is not supported by Greengrass

사용자 지정 Greengrass 구성 요소에서 프로세스 간 통신 (IPC) 작업을 사용하고 필수 AWS 제공 구성 요소가 코어 장치에 설치되지 않은 경우 이 오류가 표시될 수 있습니다.

이 문제를 해결하려면 구성 요소 레시피에 필수 구성 요소를 종속 항목으로 추가하여 구성 요소를 배포할 때 AWS IoT Greengrass Core 소프트웨어에서 필수 구성 요소를 설치하도록 하십시오.

- 비밀 값 검색 — `aws.greengrass.SecretManager`
- 로컬 섀도우와 상호작용하세요 — `aws.greengrass.ShadowManager`
- 로컬 배포 및 구성 요소 관리 — `aws.greengrass.Cli` v2.6.0 이상
- 클라이언트 장치 인증 및 권한 부여 — v2.2.0 이상 `aws.greengrass.clientdevices.Auth`

java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)

존재하지 않거나 올바른 권한이 있는 루트 폴더를 사용하도록 스트림 관리자를 구성하면 스트림 관리자 로그 파일 (`aws.greengrass.StreamManager.log`) 에 이 오류가 표시될 수 있습니다. 이 폴더를 구성하는 방법에 대한 자세한 내용은 스트림 관리자 구성을 참조하십시오.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

이 오류는 하드웨어 보안 모듈 (HSM) 을 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성할 때 지정한 개인 키 또는 인증서를 PKCS #11 provider 구성 요소가 찾거나 로드하지 못할 때 발생합니다. 다음을 따릅니다.

- AWS IoT Greengrass Core 소프트웨어에서 사용하도록 구성한 슬롯, 사용자 PIN 및 개체 레이블을 사용하여 개인 키와 인증서가 HSM에 저장되어 있는지 확인합니다.
- 개인 키와 인증서가 HSM에서 동일한 개체 레이블을 사용하는지 확인하십시오.
- HSM이 개체 ID를 지원하는 경우 개인 키와 인증서가 HSM에서 동일한 개체 ID를 사용하는지 확인하세요.

HSM의 보안 토큰에 대한 세부 정보를 쿼리하는 방법을 알아보려면 HSM 설명서를 참조하십시오. 보안 토큰의 슬롯, 개체 레이블 또는 개체 ID를 변경해야 하는 경우 HSM 설명서에서 변경 방법을 알아보세요.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

이 오류는 Secret Manager 구성 요소를 사용하여 시크릿을 배포할 때 발생할 수 있습니다 AWS Secrets Manager . 코어 디바이스의 토큰 교환 IAM 역할이 시크릿 가져오기 권한을 부여하지 않으면 배포가 실패하고 Greengrass 로그에 이 오류가 포함됩니다.

코어 디바이스가 시크릿을 다운로드하도록 승인하려면

1. 코어 디바이스의 토큰 교환 역할에 `secretsmanager:GetSecretValue` 권한을 추가합니다. 다음 예제 정책 설명은 암호의 값을 가져올 수 있는 권한을 부여합니다.

   ```
   {
       "Effect": "Allow",
       "Action": [
           "secretsmanager:GetSecretValue"
       ],
       "Resource": [
           "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
   abcdef"
       ]
   }
   ```

   자세한 설명은 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스 섹션을 참조하세요.

2. 배포를 코어 디바이스에 다시 적용합니다. 다음 중 하나를 수행합니다.
   - 변경 없이 배포를 수정하십시오. 코어 디바이스는 수정된 배포를 수신하면 암호를 다시 다운로드하려고 합니다. 자세한 설명은 배포 수정 섹션을 참조하세요.
   - AWS IoT Greengrass Core 소프트웨어를 다시 시작하여 배포를 다시 시도합니다. 자세한 정보는 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.

   Secret Manager가 시크릿을 성공적으로 다운로드하면 배포가 성공합니다.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep Access to KMS is not allowed

이 오류는 Secret Manager 구성 요소를 사용하여 AWS Key Management Service 키로 암호화된 AWS Secrets Manager 암호를 배포할 때 발생할 수 있습니다. 코어 디바이스의 토큰 교환 IAM 역할이 암호 해독 권한을 부여하지 않으면 배포가 실패하고 Greengrass 로그에 이 오류가 포함됩니다.

문제를 해결하려면 코어 디바이스의 토큰 `kms:Decrypt` 교환 역할에 권한을 추가하세요. 자세한 내용은 다음 자료를 참조하세요:

- 사용 설명서의 AWS Secrets Manager 비밀 암호화 및 암호 해독
- 핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

하드웨어 보안이 적용된 AWS IoT Greengrass Core 소프트웨어를 설치하려고 하고 하드웨어 보안 통합을 지원하지 않는 이전 Greengrass nucleus 버전을 사용할 때 이 오류가 표시될 수 있습니다. 하드웨어 보안 통합을 사용하려면 Greengrass nucleus v2.5.3 이상을 사용해야 합니다.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

Core를 시스템 서비스로 실행할 때 TPM2 라이브러리를 사용할 때 이 오류가 표시될 수 있습니다. AWS IoT Greengrass

이 오류는 AWS IoT Greengrass 코어 시스템 서비스 파일에 PKCS #11 저장소의 위치를 제공하는 환경 변수를 추가해야 함을 나타냅니다.

자세한 내용은 구성 요소 설명서의 요구 사항 섹션을 참조하십시오. PKCS #11 제공업체

AWS IoT Greengrass 클라우드 문제

다음 정보를 사용하여 AWS IoT Greengrass 콘솔 및 API 관련 문제를 해결하세요. 각 항목은 작업을 수행할 때 표시될 수 있는 오류 메시지에 해당합니다.

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

AWS IoT Greengrass 콘솔에서 또는 CreateComponentVersion작업을 통해 구성 요소 버전을 생성할 때 이 오류가 표시될 수 있습니다.

이 오류는 레시피가 유효한 JSON 또는 YAML이 아님을 나타냅니다. 레시피의 구문을 확인하고 구문 문제를 수정한 다음 다시 시도하세요. 온라인 JSON 또는 YAML 구문 검사기를 사용하여 레시피의 구문 문제를 식별할 수 있습니다.

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

AWS IoT Greengrass 콘솔에서 또는 작업을 통해 구성 요소 버전을 만들 때 이 오류가 표시될 수 있습니다. CreateComponentVersion 이 오류는 구성 요소 레시피의 S3 아티팩트가 유효하지 않음을 나타냅니다.

다음을 따릅니다.

- 구성 요소를 생성한 AWS 리전 위치와 동일한 위치에 S3 버킷이 있는지 확인하십시오. AWS IoT Greengrass 구성 요소 아티팩트에 대한 리전 간 요청은 지원하지 않습니다.
- 아티팩트 URI가 유효한 S3 객체 URL인지 확인하고 해당 S3 객체 URL에 아티팩트가 존재하는지 확인하십시오.
- S3 객체 URL에서 해당 아티팩트에 액세스할 AWS 계정 권한이 있는지 확인하십시오.

INACTIVE deployment status

필수 종속 AWS IoT 정책 없이 ListDeploymentsAPI를 호출하면 `INACTIVE` 배포 상태를 얻을 수 있습니다. 정확한 배포 상태를 확인하려면 필요한 권한이 있어야 합니다. 에 정의된 작업을 살펴보고 필요한 권한을 따라가면 종속 작업을 찾을 수 `ListDeployments` 있습니다. AWS IoT Greengrass V2 필수 종속 AWS IoT 권한이 없어도 배포 상태는 계속 표시되지만 배포 상태가 정확하지 않을 수 있습니다. `INACTIVE`

핵심 장치 배포 문제

Greengrass 코어 디바이스의 배포 문제를 해결합니다. 각 항목은 코어 기기에 표시될 수 있는 로그 메시지에 해당합니다.

주제

- Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact
- Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.
- Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>
- software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
- com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component
- Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service
- Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration
- Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy
- Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration
- Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadExceptio Failed to download artifact

코어 기기가 배포를 적용할 때 AWS IoT Greengrass 코어 소프트웨어가 구성 요소 아티팩트를 다운로드하지 못하면 이 오류가 표시될 수 있습니다. 이 오류로 인해 배포가 실패합니다.

이 오류가 발생하면 로그에는 특정 문제를 식별하는 데 사용할 수 있는 스택 추적도 포함됩니다. 다음 각 항목은 `Failed to download artifact` 오류 메시지의 스택 추적에서 볼 수 있는 메시지에 해당합니다.

주제

- software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)
- software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

다음과 같은 경우 PackageDownloadException 오류에 이 스택 추적이 포함될 수 있습니다.

- 구성 요소 레시피에 지정한 S3 객체 URL에서는 구성 요소 아티팩트를 사용할 수 없습니다. 아티팩트를 S3 버킷에 업로드했고 아티팩트 URI가 버킷에 있는 아티팩트의 S3 객체 URL과 일치하는지 확인하십시오.
- 코어 디바이스의 토큰 교환 역할로는 AWS IoT Greengrass Core 소프트웨어가 구성 요소 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 다운로드할 수 없습니다. 토큰 교환 역할이 아티팩트를 `s3:GetObject` 사용할 수 있는 S3 객체 URL을 허용하는지 확인하십시오.

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

코어 디바이스에 호출 `s3:GetBucketLocation` 권한이 없는 경우 이 스택 추적이 PackageDownloadException 오류에 포함될 수 있습니다. 오류 메시지에는 다음 메시지도 포함됩니다.

```
reason: Failed to determine S3 bucket location
```

코어 디바이스의 토큰 교환 역할이 아티팩트를 사용할 수 있는 S3 버킷을 `s3:GetBucketLocation` 허용하는지 확인하십시오.

Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchI Integrity check for downloaded artifact failed. Probably due to file corruption.

코어 디바이스가 배포를 적용할 때 AWS IoT Greengrass 코어 소프트웨어가 구성 요소 아티팩트를 다운로드하지 못하면 이 오류가 표시될 수 있습니다. 다운로드한 아티팩트 파일의 체크섬이 구성 요소를 만들 때 AWS IoT Greengrass 계산된 체크섬과 일치하지 않아 배포가 실패합니다.

다음을 따릅니다.

- 아티팩트 파일을 호스팅하는 S3 버킷에서 아티팩트 파일이 변경되었는지 확인하십시오. 구성 요소를 생성한 이후 파일이 변경된 경우, 코어 디바이스에서 예상하는 이전 버전으로 복원하십시오. 파일을 이전 버전으로 복원할 수 없거나 새 버전의 파일을 사용하려면 아티팩트 파일을 사용하여 구성 요소의 새 버전을 만드십시오.
- 코어 디바이스의 인터넷 연결을 확인하세요. 아티팩트 파일을 다운로드하는 동안 파일이 손상되면 이 오류가 발생할 수 있습니다. 새 배포를 만들고 다시 시도하세요.

Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVers Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

코어 디바이스에서 해당 코어 디바이스의 배포 요구 사항을 충족하는 구성 요소 버전을 찾을 수 없는 경우 이 오류가 표시될 수 있습니다. 코어 기기는 AWS IoT Greengrass 서비스와 로컬 기기의 구성 요소를 확인합니다. 오류 메시지에는 각 배포의 대상과 구성 요소에 대한 해당 배포의 버전 요구 사항이 포함됩니다. 배포 대상은 사물, 사물 그룹 또는 `LOCAL_DEPLOYMENT` 코어 장치의 로컬 배포를 나타내는 사물 그룹일 수 있습니다.

이 문제는 다음과 같은 경우에 발생할 수 있습니다.

- 코어 디바이스는 구성 요소 버전 요구 사항이 충돌하는 여러 배포의 대상입니다. 예를 들어, 코어 기기는 `com.example.HelloWorld` 구성 요소가 포함된 여러 배포의 대상일 수 있습니다. 한 배포에는 버전 1.0.0이 필요하고 다른 배포에는 버전 1.0.1이 필요합니다. 두 요구 사항을 모두 충족하는 구성 요소를 보유하는 것은 불가능하므로 배포가 실패합니다.

- 구성 요소 버전이 AWS IoT Greengrass 서비스나 로컬 장치에 존재하지 않습니다. 예를 들어 구성 요소가 삭제되었을 수 있습니다.
- 버전 요구 사항을 충족하는 구성 요소 버전이 있지만 핵심 장치의 플랫폼과 호환되는 버전은 없습니다.
- 코어 디바이스의 AWS IoT 정책은 `greengrass:ResolveComponentCandidates` 권한을 부여하지 않습니다. 오류 `Status Code: 403` 로그에서 이 문제를 찾아보십시오. 이 문제를 해결하려면 코어 디바이스의 AWS IoT 정책에 `greengrass:ResolveComponentCandidates` 권한을 추가하십시오. 자세한 설명은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 섹션을 참조하세요.

이 문제를 해결하려면 호환되는 구성 요소 버전을 포함하도록 배포를 수정하거나 호환되지 않는 버전을 제거하십시오. 클라우드 배포를 수정하는 방법에 대한 자세한 내용은 을 참조하십시오. 배포 수정 로컬 배포를 수정하는 방법에 대한 자세한 내용은 AWS IoT Greengrass CLI 배포 create 명령을 참조하십시오.

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundE The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

구성 요소를 코어 디바이스에 배포할 때 구성 요소에 코어 디바이스의 플랫폼과 호환되는 플랫폼이 나열되지 않는 경우 이 오류가 표시될 수 있습니다. 다음 중 하나를 수행합니다.

- 구성 요소가 사용자 지정 Greengrass 구성 요소인 경우 핵심 장치와 호환되도록 구성 요소를 업데이트할 수 있습니다. 코어 디바이스의 플랫폼과 일치하는 새 매니페스트를 추가하거나 코어 디바이스의 플랫폼과 일치하도록 기존 매니페스트를 업데이트하십시오. 자세한 설명은 AWS IoT Greengrass 컴포넌트 레시피 참조 섹션을 참조하세요.
- 에서 구성 요소를 제공하는 AWS 경우 다른 버전의 구성 요소가 코어 기기와 호환되는지 확인하세요. 호환되는 버전이 없는 경우 AWS IoT Greengrass 태그를 AWS re:Post사용하여 당사에 문의하거나 문의해 주십시오 AWS Support.

com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component

Greengrass 핵에 종속된 구성 요소를 배포하고 코어 디바이스에서 사용 가능한 최신 마이너 버전보다 이전 Greengrass nucleus 버전을 실행하는 경우 이 오류가 표시될 수 있습니다. 이 오류는 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 호환되는 최신 버전으로 자동 업데이트하려고 하기 때문에 발생합니다. 그러나 AWS IoT Greengrass 코어 소프트웨어는 Greengrass 핵이 새 마이너 버전으로 업데이트되는 것을 방지합니다. AWS 제공되는 여러 구성 요소가 Greengrass 핵의 특정 마이너 버전에 의존하기 때문입니다. 자세한 설명은 Greengrass 핵 업데이트 동작 섹션을 참조하세요.

배포를 수정하여 사용하려는 Greengrass nucleus 버전을 지정해야 합니다. 다음 중 하나를 수행합니다.

- 배포를 수정하여 코어 디바이스에서 현재 실행 중인 Greengrass nucleus 버전을 지정하십시오.
- Greengrass 핵의 최신 마이너 버전을 지정하도록 배포를 수정하십시오. 이 옵션을 선택하는 경우 Greengrass AWS Nucleus의 특정 마이너 버전에 따라 제공되는 모든 구성 요소의 버전도 업데이트해야 합니다. 자세한 설명은 AWS-제공된 구성 요소 섹션을 참조하세요.

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

Greengrass 디바이스를 한 사물 그룹에서 다른 사물 그룹으로 이동한 다음 Greengrass를 재시작해야 하는 배포가 있는 원래 그룹으로 다시 이동할 때 이 오류가 표시될 수 있습니다.

이 문제를 해결하려면 디바이스의 시작 디렉터리를 다시 생성하십시오. 또한 Greengrass 핵을 버전 2.9.6 이상으로 업그레이드하는 것이 좋습니다.

다음은 시작 디렉터리를 다시 생성하는 Linux 스크립트입니다. 라는 `fix_directory.sh` 파일에 스크립트를 저장합니다.

```
#!/bin/bash

set -e
```

```
GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
  mkdir -p $GG_ROOT/alts/directory_fix
  echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
  ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
  echo "Creating directory: $TARGET"
  mkdir -p "$TARGET"
fi

DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

스크립트를 실행하려면 다음 명령을 실행합니다.

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDow Greengrass Cloud Service returned an error when getting full deployment configuration

코어 장치가 7KB (사물을 대상으로 하는 배포의 경우) 또는 31KB (사물 그룹을 대상으로 하는 배포의 경우) 보다 큰 배포 문서인 대규모 배포 문서를 수신하면 이 오류가 발생할 수 있습니다. 대규모 배포 문서를 검색하려면 코어 디바이스의 AWS IoT 정책에서 권한을 허용해야 합니다. `greengrass:GetDeploymentConfiguration` 이 오류는 코어 장치에 이 권한이 없는 경우 발생할

수 있습니다. 이 오류가 발생하면 배포가 무기한 재시도되며 상태는 In progress () `IN_PROGRESS` 입니다.

이 문제를 해결하려면 코어 디바이스의 AWS IoT 정책에 `greengrass:GetDeploymentConfiguration` 권한을 추가하십시오. 자세한 설명은 코어 디바이스 정책 업데이트 AWS IoT 섹션을 참조하세요.

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

코어 디바이스가 배포를 수신하고 코어 디바이스의 AWS IoT 정책이 `greengrass:ListThingGroupsForCoreDevice` 권한을 허용하지 않는 경우 이 경고가 표시될 수 있습니다. 배포를 생성하면 코어 장치는 이 권한을 사용하여 사물 그룹을 식별하고 코어 장치를 제거한 사물 그룹의 구성 요소를 제거합니다. 코어 디바이스에서 Greengrass nucleus v2.5.0을 실행하는 경우 배포가 실패합니다. 코어 디바이스에서 Greengrass nucleus v2.5.1 이상을 실행하는 경우 배포는 진행되지만 구성 요소를 제거하지는 않습니다. 사물 그룹 제거 동작에 대한 자세한 내용은 을 참조하십시오. 디바이스에 AWS IoT Greengrass 구성 요소 배포

코어 장치를 제거하는 사물 그룹의 구성 요소를 제거하도록 코어 장치 동작을 업데이트하려면 핵심 장치의 AWS IoT 정책에 `greengrass:ListThingGroupsForCoreDevice` 권한을 추가하십시오. 자세한 설명은 코어 디바이스 정책 업데이트 AWS IoT 섹션을 참조하세요.

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

코어 디바이스가 `DEBUG` 로그 수준에서 오류를 기록하기 때문에 이 정보 메시지가 오류 없이 여러 번 인쇄되는 것을 볼 수 있습니다. 이 문제는 코어 디바이스가 대용량 배포 문서를 수신할 때 발생할 수 있습니다. 이 문제가 발생하면 배포가 무기한 재시도되며 상태는 In progress () `IN_PROGRESS` 입니다. 이 문제를 해결하는 방법에 대한 자세한 내용은 이 문제 해결 항목을 참조하십시오.

Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataE
null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

데이터플레인 API에 `iot:Connect` 권한이 없는 경우 이 오류가 표시될 수 있습니다. 올바른 정책이 없는 경우 경고를 받게 됩니다. `GreengrassV2DataException: 403` 권한 정책을 만들려면 다음 지침을 따르십시오AWS IoT 정책 생성.

핵심 장치 구성 요소 문제

코어 디바이스의 Greengrass 구성 요소 문제를 해결합니다.

주제

- Warn: '<command>' is not recognized as an internal or external command
- Python 스크립트는 메시지를 기록하지 않습니다
- 기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않습니다.
- awsiot.greengrasscoreipc.model.UnauthorizedError
- com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)
- com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers
- Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"
- copyFrom: <configurationPath> is already a container, not a leaf
- com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'
- java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.
- aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Warn: '<command>' is not recognized as an internal or external command

AWS IoT Greengrass Core 소프트웨어가 구성 요소의 수명 주기 스크립트에서 명령을 실행하지 못하면 Greengrass 구성 요소의 로그에서 이 오류가 표시될 수 있습니다. 구성 요소의 상태는 이 오류의 `BROKEN` 결과로 나타납니다. 이 오류는 구성 요소 (예:) 를 실행하는 시스템 사용자가 PATH의 폴더에서 명령의 실행 파일을 찾을 수 없는 경우 발생할 수 있습니다. `ggc_user`

Windows 장치에서는 실행 파일이 들어 있는 폴더가 구성 요소를 실행하는 시스템 사용자의 폴더에 있는지 확인하십시오. `PATH` 에서 누락된 경우 다음 중 하나를 수행하십시오. `PATH`

- 모든 사용자가 사용할 수 있는 `PATH` 시스템 변수에 실행 파일 폴더를 추가합니다. 그런 다음 구성 요소를 다시 시작합니다.

  Greengrass nucleus 2.5.0을 실행하는 경우 `PATH` 시스템 변수를 업데이트한 후 AWS IoT Greengrass Core 소프트웨어를 다시 시작하여 업데이트된 구성요소를 실행해야 합니다. `PATH` 소프트웨어를 다시 시작한 `PATH` 후 AWS IoT Greengrass Core 소프트웨어에서 업데이트된 버전을 사용하지 않는 경우 장치를 다시 시작하고 다시 시도하십시오. 자세한 설명은 AWS IoT GreengrassCore 소프트웨어 실행 섹션을 참조하세요.
- 구성 요소를 실행하는 시스템 `PATH` 사용자의 사용자 변수에 실행 파일 폴더를 추가합니다.

Python 스크립트는 메시지를 기록하지 않습니다

Greengrass 코어 디바이스는 구성 요소 관련 문제를 식별하는 데 사용할 수 있는 로그를 수집합니다. Python `stdout` 스크립트와 `stderr` 메시지가 구성 요소 로그에 표시되지 않는 경우 Python에서 이러한 표준 출력 스트림에 대해 버퍼를 플러시하거나 버퍼링을 비활성화해야 할 수 있습니다. 다음을 수행합니다.

- -u 인수를 사용하여 Python을 실행하여 및 에서의 `stdout` 버퍼링을 비활성화합니다. `stderr`

  Linux or Unix

  ```
  python3 -u hello_world.py
  ```

  Windows

  ```
  py -3 -u hello_world.py
  ```

- 컴포넌트의 레시피에서 Setenv를 사용하여 PYTHONUNBUFFERED 환경 변수를 비어 있지 않은 문자열로 설정합니다. 이 환경 변수는 `stdout` 및 `stderr` 에서의 버퍼링을 비활성화합니다.

- 또는 스트림의 버퍼를 플러시합니다. `stdout` `stderr` 다음 중 하나를 수행합니다.
  - 인쇄할 때 메시지를 플러시합니다.

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

  - 인쇄한 후 메시지를 플러시합니다. 스트림을 플러시하기 전에 여러 메시지를 보낼 수 있습니다.

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

Python 스크립트가 로그 메시지를 출력하는지 확인하는 방법에 대한 자세한 내용은 을 참조하십시오[모니터 AWS IoT Greengrass 로그](#).

기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않습니다.

구성 요소 `DefaultConfiguration` 레시피에서 를 변경해도 배포 중에 새 기본 구성이 구성 요소의 기존 구성을 대체하지 않습니다. 새 기본 구성을 적용하려면 구성 요소 구성을 기본 설정으로 다시 설정해야 합니다. 구성 요소를 배포할 때 빈 문자열 하나를 [재설정 업데이트로](#) 지정하십시오.

Console

경로 재설정

```
[""]
```

AWS CLI

다음 명령어는 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-
deployment.json
```

`reset-configuration-deployment.json`파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {,
        "reset": [""]
      }
    }
  }
}
```

Greengrass CLI

다음 Greengrass CLI 명령은 코어 디바이스에 로컬 배포를 생성합니다.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

이 `reset-configuration-deployment.json` 파일에는 다음과 같은 JSON 문서가 들어 있습니다.

```
{
  "com.example.HelloWorld": {
    "RESET": [""]
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

구성 요소에 리소스에서 IPC 작업을 수행할 권한이 없는 경우 Greengrass 구성 요소의 로그에서 이 오류가 표시될 수 있습니다. 구성 요소에 IPC 작업을 호출할 권한을 부여하려면 구성 요소의 구성에서 IPC 권한 부여 정책을 정의하십시오. 자세한 설명은 구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오. 섹션을 참조하세요.

> **Tip**
>
> 구성 요소 `DefaultConfiguration` 레시피에서 를 변경하는 경우 구성 요소 구성을 새 기본 구성으로 재설정해야 합니다. 구성 요소를 배포할 때 빈 문자열 하나를 재설정 업데이트로 지정하십시오. 자세한 설명은 기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않습니다. 섹션을 참조하세요.

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

코어 디바이스의 모든 구성 요소를 포함한 여러 IPC 권한 부여 정책이 동일한 정책 ID를 사용하는 경우 이 오류가 발생할 수 있습니다.

구성 요소의 IPC 권한 부여 정책을 확인하고 중복된 항목을 수정한 다음 다시 시도하십시오. 고유한 정책 ID를 만들려면 구성 요소 이름, IPC 서비스 이름 및 카운터를 조합하는 것이 좋습니다. 자세한 설명은 구성 요소가 IPC 작업을 수행할 수 있도록 승인하십시오. 섹션을 참조하세요.

> **Tip**
>
> 구성 요소 `DefaultConfiguration` 레시피에서 를 변경하는 경우 구성 요소 구성을 새 기본 구성으로 재설정해야 합니다. 구성 요소를 배포할 때 빈 문자열 하나를 재설정 업데이트로 지정하십시오. 자세한 설명은 기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않습니다. 섹션을 참조하세요.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

코어 기기가 토큰 교환 서비스로부터 AWS 자격 증명을 가져올 수 없는 경우 이 오류가 표시될 수 있습니다. HTTP 400 상태 코드는 코어 디바이스의 토큰 교환 IAM 역할이 존재하지 않거나 AWS IoT 자격 증명 제공자가 맡을 수 있는 신뢰 관계가 없기 때문에 이 오류가 발생했음을 나타냅니다.

다음을 따릅니다.

1. 코어 디바이스가 사용하는 토큰 교환 역할을 식별하십시오. 오류 메시지에는 토큰 교환 역할을 가리키는 코어 디바이스의 AWS IoT 역할 별칭이 포함됩니다. 개발 컴퓨터에서 다음 명령을 실행하

고 오류 메시지에 *MyGreengrassCoreTokenExchangeRoleAlias*있는 AWS IoT 역할 별칭의 이름으로 바꿉니다.

```
aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
```

응답에는 토큰 교환 IAM 역할의 Amazon 리소스 이름 (ARN) 이 포함됩니다.

```
{
  "roleAliasDescription": {
    "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
    "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
MyGreengrassCoreTokenExchangeRoleAlias",
    "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
    "owner": "123456789012",
    "credentialDurationSeconds": 3600,
    "creationDate": "2021-02-05T16:46:18.042000-08:00",
    "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
  }
}
```

2. 역할이 존재하는지 확인하십시오. 다음 명령을 실행하고 *MyGreengrassV2#TokenExchangeRole* 토큰 교환 역할의 이름으로 대체합니다.

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

명령에서 NoSuchEntity 오류가 반환되는 경우 역할은 존재하지 않으므로 역할을 만들어야 합니다. 이 역할을 만들고 구성하는 방법에 대한 자세한 내용은 을 참조하십시오핵심 디바이스가 상호 작용할 수 있도록 권한 부여AWS서비스.

3. 역할에 AWS IoT 자격 증명 제공자가 맡을 수 있는 신뢰 관계가 있는지 확인하십시오. 이전 단계의 응답에는 역할의 신뢰 관계를 정의하는 a가 포함되어 있습니다. AssumeRolePolicyDocument 역할은 이를 수임할 수 credentials.iot.amazonaws.com 있는 신뢰 관계를 정의해야 합니다. 이 문서는 다음 예와 비슷해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

역할의 신뢰 관계에서 해당 역할을 `credentials.iot.amazonaws.com` 맡을 수 없는 경우 이 신뢰 관계를 역할에 추가해야 합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 역할 수정을 참조하십시오.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

코어 기기가 토큰 교환 서비스로부터 AWS 자격 증명을 가져올 수 없는 경우 이 오류가 표시될 수 있습니다. HTTP 403 상태 코드는 코어 기기의 AWS IoT 정책이 코어 기기의 AWS IoT 역할 별칭에 대한 `iot:AssumeRoleWithCertificate` 권한을 부여하지 않기 때문에 이 오류가 발생했음을 나타냅니다.

코어 디바이스의 AWS IoT 정책을 검토하고 코어 디바이스의 AWS IoT 역할 `iot:AssumeRoleWithCertificate` 별칭에 대한 권한을 추가하십시오. 오류 메시지에는 핵심 장치의 현재 AWS IoT 역할 별칭이 포함됩니다. 이 권한 및 핵심 장치 AWS IoT 정책을 업데이트하는 방법에 대한 자세한 내용은 AWS IoT Greengrass V2코어 디바이스에 대한 최소 AWS IoT 정책 및 코어 디바이스 정책 업데이트 AWS IoT 을 참조하십시오.

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

구성 요소가 AWS 자격 증명을 요청하려고 하는데 토큰 교환 서비스에 연결할 수 없는 경우 이 오류가 표시될 수 있습니다.

다음을 따릅니다.

- 구성 요소가 토큰 교환 서비스 구성 요소에 대한 종속성을 선언했는지 확인하십시오. `aws.greengrass.TokenExchangeService` 그렇지 않으면 종속성을 추가하고 구성 요소를 재배포하세요.
- 구성 요소가 docker에서 실행되는 경우 에 따라 올바른 네트워크 설정과 환경 변수를 적용해야 합니다. Docker 컨테이너 구성 요소 (Linux) 에서 AWS 자격 증명 사용

- 구성 요소가 NodeJS로 작성된 경우 dns를 설정하십시오. setDefaultResult로 주문하십시오. **`ipv4first`**
- 로 `::1` 시작하고 다음을 포함하는 항목이 있는지 검사하십시오 `/etc/hostslocalhost`. 항목을 삭제하여 구성 요소가 잘못된 주소의 토큰 교환 서비스에 연결되었는지 확인하세요.

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

구성 요소가 토큰 교환 서비스를 실행하지 않고 구성 요소가 AWS 자격 증명을 요청하려고 할 때 이 오류가 표시될 수 있습니다.

다음을 따릅니다.

- 구성 요소가 토큰 교환 서비스 구성 요소에 대한 종속성을 선언했는지 확인하십시오. `aws.greengrass.TokenExchangeService` 그렇지 않으면 종속성을 추가하고 구성 요소를 재배포하세요.
- 구성 요소가 수명 주기 동안 AWS 자격 증명을 사용하는지 확인하세요. `install` AWS IoT Greengrass `install`수명 주기 동안 토큰 교환 서비스의 가용성을 보장하지 않습니다. 구성 요소를 업데이트하여 AWS 자격 증명을 사용하는 코드를 `startup` 또는 `run` 수명 주기로 이동한 다음 구성 요소를 재배포하십시오.

copyFrom: <configurationPath> is already a container, not a leaf

구성 값을 컨테이너 유형 (목록 또는 개체) 에서 컨테이너가 아닌 유형 (문자열, 숫자 또는 부울) 으로 변경할 때 이 오류가 표시될 수 있습니다. 다음을 따릅니다.

1. 구성 요소의 레시피를 확인하여 기본 구성이 해당 구성 값을 목록으로 설정하는지 또는 개체로 설정하는지 확인하십시오. 그렇다면 해당 구성 값을 제거하거나 변경하세요.
2. 배포를 생성하여 해당 구성 값을 기본값으로 재설정합니다. 자세한 내용은 배포 만들기 및 구성 요소 구성 업데이트 섹션을 참조하세요.

그런 다음 해당 구성 값을 문자열, 숫자 또는 부울로 설정할 수 있습니다.

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'

Docker 애플리케이션 관리자 구성 요소가 Amazon Elastic Container Registry (Amazon ECR) 의 프라이빗 리포지토리에서 Docker 이미지를 다운로드하려고 할 때 Greengrass 핵 로그에 이 오류가 표시될 수 있습니다. 이 오류는 `wincred` Docker 자격 증명 도우미 () 를 사용하는 경우 발생합니다. `docker-credential-wincred` 따라서 Amazon ECR은 로그인 자격 증명을 저장할 수 없습니다.

다음 조치 중 하나를 취하십시오.

- `wincred`Docker 자격 증명 도우미를 사용하지 않는 경우 코어 디바이스에서 `docker-credential-wincred` 프로그램을 제거하세요.
- `wincred`Docker 자격 증명 도우미를 사용하는 경우 다음을 수행하십시오.
  1. 코어 디바이스에서 `docker-credential-wincred` 프로그램 이름을 변경합니다. Windows Docker 자격 증명 도우미의 새 `wincred` 이름으로 바꾸십시오. 예를 들어 이름을 로 바꿀 수 있습니다. `docker-credential-wincredreal`
  2. Windows Docker 자격 증명 도우미의 새 이름을 사용하도록 Docker 구성 파일 (`.docker/config.json`) 의 `credsStore` 옵션을 업데이트하십시오. 예를 들어, 프로그램 이름을 로 변경한 경우 옵션을 로 `docker-credential-wincredreal` 업데이트하십시오. `credsStore` `wincredreal`

     ```
     {
       "credsStore": "wincredreal"
     }
     ```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

구성 요소의 프로세스 (예:) 를 실행하는 시스템 사용자의 암호가 만료된 경우 Windows 코어 장치에서 이 오류가 표시될 수 있습니다. `ggc_user` 따라서 AWS IoT Greengrass Core 소프트웨어는 해당 시스템 사용자로 구성 요소 프로세스를 실행할 수 없습니다.

Greengrass 시스템 사용자 비밀번호를 업데이트하려면

1. 관리자로 다음 명령을 실행하여 사용자 비밀번호를 설정합니다. *ggc_user#* 시스템 사용자로 바꾸고, *##### ### #####* 바꿉니다.

   ```
   net user ggc_user password
   ```

2. PsExec 유틸리티를 사용하여 계정의 Credential Manager 인스턴스에 사용자의 새 암호를 저장합니다. LocalSystem *###* 설정한 사용자 암호로 바꾸십시오.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

> **Tip**
>
> Windows 구성에 따라 사용자 암호가 향후 날짜에 만료되도록 설정될 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 업데이트하십시오. 사용자 비밀번호가 만료되지 않도록 설정할 수도 있습니다.
>
> - 사용자 및 암호가 언제 만료되는지 확인하려면 다음 명령을 실행합니다.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 사용자 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - 이 명령이 더 이상 사용되지 않는 Windows 10 이상을 사용하는 경우 `wmic` 다음 명령을 실행하세요. PowerShell
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

스트림 관리자 v2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 때 구성 요소가 시작되지 않으면 스트림 관리자 구성 요소의 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}
```

스트림 관리자 v2.0.7을 배포한 후 이후 버전으로 업그레이드하려면 스트림 관리자 v2.0.12로 직접 업그레이드해야 합니다. 스트림 관리자 구성 요소에 대한 자세한 내용은 을 참조하십시오. 스트림 관리자

코어 디바이스 Lambda 함수 구성 요소 문제

코어 디바이스의 Lambda 함수 구성 요소 문제를 해결합니다.

주제

- The following cgroup subsystems are not mounted: devices, memory
- ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

The following cgroup subsystems are not mounted: devices, memory

다음과 같은 경우에 컨테이너화된 Lambda 함수를 실행할 때 이 오류가 표시될 수 있습니다.

- 코어 디바이스에는 메모리 또는 디바이스 cgroup에 대해 cgroup v1이 활성화되어 있지 않습니다.
- 코어 기기에는 cgroups v2가 활성화되어 있습니다. 그린그래스 람다 함수에는 cgroups v1이 필요하며 cgroups v1과 v2는 상호 배타적입니다.

cgroups v1을 활성화하려면 다음 Linux 커널 파라미터를 사용하여 디바이스를 부팅하십시오.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

> **Tip**
>
> Raspberry Pi에서는 `/boot/cmdline.txt` 파일을 편집하여 기기의 커널 매개변수를 설정합니다.

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

기존 구독 라우터 구성 요소에서 구독을 지정하지 않고 Core SDK를 사용하는 AWS IoT Greengrass V1 Lambda 함수를 V2 코어 디바이스에서 실행할 때 이 오류가 표시될 수 있습니다. 이 문제를 해결하려면 필요한 구독을 지정하도록 레거시 구독 라우터를 배포 및 구성하십시오. 자세한 설명은 V1 람다 함수 가져오기 섹션을 참조하세요.

구성 요소 버전이 중단되었습니다.

코어 장치의 구성 요소 버전이 중단되면 PHD (Personal Health Dashboard) 에 알림이 표시될 수 있습니다. 구성 요소 버전은 단종된 지 60분 이내에 PHD에게 이 알림을 보냅니다.

수정이 필요한 배포를 확인하려면 다음을 사용하여 다음을 수행하십시오. AWS Command Line Interface

1. 다음 명령을 실행하여 핵심 장치 목록을 가져옵니다.

   ```
   aws greengrassv2 list-core-devices
   ```

2. 다음 명령을 실행하여 1단계에서 각 코어 디바이스의 구성 요소 상태를 검색합니다. 쿼리할 각 코어 디바이스의 이름으로 *coreDeviceName* 바꿉니다.

   ```
   aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
   ```

3. 이전 단계에서 설치한 중단된 구성 요소 버전이 설치된 코어 디바이스를 수집하십시오.
4. 다음 명령을 실행하여 3단계에서 각 코어 장치에 대한 모든 배포 작업의 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 *coreDeviceName* 바꿉니다.

   ```
   aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
   ```

응답에는 코어 장치의 배포 작업 목록이 포함됩니다. 배포를 수정하여 다른 구성 요소 버전을 선택할 수 있습니다. 배포를 수정하는 방법에 대한 자세한 내용은 배포 수정을 참조하십시오.

Greengrass 명령줄 인터페이스 문제

Greengrass CLI에서 문제를 해결합니다.

주제

- java.lang.RuntimeException: Unable to create ipc client

java.lang.RuntimeException: Unable to create ipc client

Greengrass CLI 명령을 실행하고 AWS IoT Greengrass Core 소프트웨어가 설치된 위치와 다른 루트 폴더를 지정할 때 이 오류가 표시될 수 있습니다.

다음 중 하나를 수행하여 루트 경로를 설정하고 AWS IoT Greengrass Core 소프트웨어 설치 */greengrass/v2* 경로로 바꾸십시오.

- `GGC_ROOT_PATH` 환경 변수를 */greengrass/v2*로 설정합니다.
- 다음 예와 같이 명령에 `--ggcRootPath` */greengrass/v2* 인수를 추가합니다.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface 이슈

에 대한 AWS CLI AWS IoT Greengrass V2 문제를 해결하십시오.

주제

- Error: Invalid choice: 'greengrassv2'

Error: Invalid choice: 'greengrassv2'

AWS CLI (예:`aws greengrassv2 list-core-devices`) 를 AWS IoT Greengrass V2 사용하여 명령을 실행할 때 이 오류가 표시될 수 있습니다.

이 오류는 지원되지 않는 버전이 있음을 나타냅니다 AWS IoT Greengrass V2. AWS CLI AWS IoT Greengrass V2 와 함께 사용하려면 다음 버전 중 하나 이상이 있어야 합니다. AWS CLI

- AWS CLI V1 최소 버전: v1.18.197
- 최소 V2 버전: v2.1.11 AWS CLI

> **Tip**
>
> 다음 명령을 실행하여 사용 중인 버전을 확인할 수 있습니다. AWS CLI
>
> ```
> aws --version
> ```

이 문제를 AWS CLI 해결하려면 를 지원하는 최신 버전으로 AWS IoT Greengrass V2 업데이트하십시오. 자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI의 설치, 업데이트, 제거를 참조하세요.

세부 배포 오류 코드

이 섹션의 오류 코드 및 해결 방법을 사용하면 Greengrass nucleus 버전 2.8.0 이상을 사용할 때 구성 요소 배포 관련 문제를 해결하는 데 도움이 됩니다.

Greengrass Nucleus는 배포 오류를 가장 구체적이지 않은 코드부터 사용 가능한 가장 구체적인 코드까지의 계층 구조로 보고합니다. 이 계층 구조를 사용하면 배포 오류의 원인을 정확히 찾아낼 수 있습니다. 예를 들어 다음과 같은 오류 계층이 발생할 수 있습니다.

- 배포_실패
  - 아티팩트_다운로드_오류
    - IO_오류
      - 디스크 공간_크리티컬

오류 코드는 유형별로 구성됩니다. 각 유형은 발생할 수 있는 오류 클래스를 나타냅니다.AWS IoT Greengrass콘솔, API 및 에서 이러한 오류 유형을 보고합니다.AWS CLI. 오류 계층에서 보고된 오류에 따라 오류 유형이 두 개 이상 있을 수 있습니다. 위 예제에서 반환되는 오류 유형은 다음과 같습니다.`DEVICE_ERROR`.

유형은 다음과 같습니다.

- 권한_오류— 권한이 필요한 작업에 대한 액세스가 거부되었습니다.
- 요청_오류— 배포 문서의 문제로 인해 오류가 발생했습니다.
- 구성 요소_레시피_오류— 구성 요소 레시피의 문제로 인해 오류가 발생했습니다.
- AWS_구성 요소_오류— 시작 또는 제거 시 오류가 발생했습니다.AWS제공된 구성 요소.
- 사용자_구성 요소_오류— 사용자 구성 요소를 시작하거나 제거할 때 오류가 발생했습니다.
- 구성 요소_오류— 구성 요소를 시작하거나 제거할 때 오류가 발생했지만 Greengrass 핵은 구성 요소가 구성요소인지 확인할 수 없습니다.AWS제공된 구성 요소 또는 사용자 구성 요소.
- 디바이스_오류— 로컬 I/O에서 오류가 발생했거나 다른 디바이스 오류가 발생했습니다.
- 종속성_오류— 배포에서 Amazon S3에서 아티팩트를 다운로드하거나 ECR 레지스트리에서 이미지를 가져오지 못했습니다.
- HTTP_ERROR— HTTP 요청에서 오류가 발생했습니다.
- 네트워크 오류— 디바이스 네트워크에서 오류가 발생했습니다.
- 핵_오류— Greengrass 핵은 구성 요소를 찾을 수 없거나 활성 핵 버전을 찾을 수 없었습니다.
- 서버_오류— 서버가 요청에 대한 응답으로 500 오류를 반환했습니다.
- 클라우드_서비스_오류— 에서 오류가 발생했습니다.AWS IoT Greengrass클라우드 서비스.
- 알 수 없는_오류— 구성 요소에서 확인되지 않은 예외가 발생했습니다.

이 섹션의 많은 오류는 다음과 같은 추가 정보를 보고합니다.AWS IoT Greengrass코어 로그. 이러한 로그는 코어 디바이스의 로컬 파일 시스템에 저장됩니다. 에 대한 로그가 있습니다.AWS IoT Greengrass코어 코어 소프트웨어 및 각 개별 구성 요소에 대한 로그 액세스에 대한 자세한 내용은 을 참조하십시오.파일 시스템 로그에 액세스하십시오..

권한 오류

액세스 거부됨

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS권한이 올바르게 설정되지 않았으므로 서비스 작업에서 403 오류가 반환됩니다. 자세한 내용은 보다 구체적인 오류 코드를 확인하세요.

GET_배포_구성_액세스_거부됨

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS IoT정책에서 호출 권한을 허용하지 않습니다.`GetDeploymentConfiguration`오퍼레이션. 추가`greengrass::GetDeploymentConfiguration`핵심 장치 정책에 대한 권한.

GET_구성 요소_버전_아티팩트_액세스가 거부되었습니다

코어 장치의 경우 이 오류가 발생할 수 있습니다.AWS IoT정책은 허용하지 않습니다greengrass:GetComponentVersionArtifact허가. 핵심 장치의 정책에 권한을 추가합니다.

해결_구성 요소_후보_액세스_거부됨

코어 장치의 경우 이 오류가 발생할 수 있습니다.AWS IoT정책은 허용하지 않습니다greengrass:ResolveComponentCandidates허가. 핵심 장치의 정책에 권한을 추가합니다.

GET_ECR_자격 증명_오류

배포에서 ECR의 프라이빗 레지스트리로 인증할 수 없을 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인한 다음 배포를 다시 시도하십시오.

도커용_승인되지 않은 사용자

Greengrass 사용자가 Docker를 사용할 권한이 없을 때 이 오류가 발생할 수 있습니다. Greengrass를 루트로 실행 중이거나 사용자가 루트에 추가되었는지 확인하십시오.docker그룹. 그런 다음 배포를 다시 시도해 보십시오.

S3_액세스가 거부되었습니다

Amazon S3 작업에서 403 오류가 반환될 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

S3_HEAD_객체_액세스가 거부되었습니다.

기기의 토큰 교환 역할이 이를 허용하지 않는 경우에도 이 오류가 발생할 수 있습니다.AWS IoT Greengrass구성 요소 레시피에 지정하거나 구성 요소 아티팩트를 사용할 수 없는 S3 객체 URL에서 구성 요소 아티팩트를 다운로드하기 위한 핵심 소프트웨어입니다. 토큰 교환 역할이 허용되는지 확인s3:GetObject아티팩트를 사용할 수 있고 아티팩트가 있는 S3 객체 URL용.

S3_GET_버킷_위치_액세스_거부됨

기기의 토큰 교환 역할이 이를 허용하지 않을 때 이 오류가 발생할 수 있습니다.s3:GetBucketLocation아티팩트를 사용할 수 있는 Amazon S3 버킷에 대한 권한. 디바이스에서 권한을 허용하는지 확인한 다음 배포를 다시 시도하세요.

S3_GET_객체_액세스가 거부되었습니다.

기기의 토큰 교환 역할이 이를 허용하지 않는 경우에도 이 오류가 발생할 수 있습니다.AWS IoT Greengrass구성 요소 레시피에 지정하거나 구성 요소 아티팩트를 사용할 수 없는 S3 객체 URL에서 구성 요소 아티팩트를 다운로드하기 위한 핵심 소프트웨어입니다. 토큰 교환 역할이 허용되는지 확인s3:GetObject아티팩트를 사용할 수 있고 아티팩트가 있는 S3 객체 URL용.

요청 오류

핵_누락_필요_기능

배포의 nucleus 버전이 대용량 구성을 다운로드하거나 Linux 리소스 제한을 설정하는 등 요청된 작업을 수행할 수 없을 때 이 오류가 발생할 수 있습니다. 작업을 지원하는 nucleus 버전으로 배포를 다시 시도하십시오.

다중_핵_해결_오류

배포에서 여러 Nucleus 구성 요소를 배포하려고 할 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

구성 요소_순환_종속성_오류

배포의 두 구성 요소가 서로 의존하는 경우 이 오류가 발생할 수 있습니다. 배포의 구성 요소가 서로 의존하지 않도록 구성 요소 설정을 수정하십시오.

무단 핵_마이너_버전_업데이트

배포의 구성 요소에 nucleus 마이너 버전 업데이트가 필요한데 해당 버전이 배포에 지정되지 않은 경우 이 오류가 발생할 수 있습니다. 이렇게 하면 다른 버전에 의존하는 구성 요소에 대한 우발적인 마이너 버전 업데이트를 줄일 수 있습니다. 배포에 새로운 마이너 뉴클리어스 버전을 포함시키십시오.

누락된 도커_애플리케이션_관리자

Docker 애플리케이션 관리자를 배포하지 않고 Docker 구성 요소를 배포할 때 이 오류가 발생할 수 있습니다. 배포에 Docker 애플리케이션 관리자가 포함되어 있는지 확인하십시오.

누락된_토큰_교환_서비스

배포에서 토큰 교환 서비스를 배포하지 않고 프라이빗 ECR 레지스트리에서 Docker 이미지 아티팩트를 다운로드하려고 할 때 이 오류가 발생할 수 있습니다. 배포에 토큰 교환 서비스가 포함되어 있는지 확인하십시오.

구성 요소 버전 요구 사항이 충족되지 않음

버전 제약조건 충돌이 있거나 구성 요소 버전이 없는 경우 이 오류가 발생할 수 있습니다. 자세한 정보는 Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>을 참조하세요.

스로틀링_오류

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS서비스 운영이 요금 할당량을 초과했습니다. 배포를 다시 시도하십시오.

충돌한_요청

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS배포에서 한 번에 두 개 이상의 작업을 수행하려고 하기 때문에 서비스 작업에서 409 오류가 반환됩니다. 배포를 다시 시도하십시오.

리소스를 찾을 수 없음

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS리소스를 찾을 수 없기 때문에 서비스 작업에서 404 오류가 반환됩니다. 로그에서 누락된 리소스를 확인하십시오.

_구성_포함____유효하지 않음_실행

다음과 같은 경우에 이 오류가 발생할 수 있습니다.`posixUser`,`posixGroup`, 또는`windowsUser`구성 요소를 실행하기 위해 지정한 정보가 유효하지 않습니다. 사용자가 유효한지 확인한 다음 배포를 다시 시도합니다.

지원되지 않는 지역

배포에 지정된 지역이 에서 지원되지 않을 때 이 오류가 발생할 수 있습니다.AWS IoT Greengrass. 지역을 확인하고 배포를 다시 시도하세요.

IOT_CRED_엔드포인트_올바르지 않음_

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS IoT구성에 지정된 자격 증명 엔드포인트가 유효하지 않습니다. 엔드포인트를 확인하고 요청을 다시 시도하세요.

IOT_데이터_엔드포인트_올바르지 않음_

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS IoT구성에 지정된 데이터 엔드포인트가 유효하지 않습니다. 엔드포인트를 확인하고 요청을 다시 시도하세요.

S3_HEAD_OBJECT_리소스를 _찾을 수 없음_

구성 요소 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 사용할 수 없을 때 이 오류가 발생할 수 있습니다. 아티팩트를 S3 버킷에 업로드했는지, 아티팩트 URI가 버킷에 있는 아티팩트의 S3 객체 URL과 일치하는지 확인합니다.

S3_GET_버킷_위치_리소스_찾을 수 없음_

Amazon S3 버킷을 찾을 수 없을 때 이 오류가 발생할 수 있습니다. 버킷이 존재하는지 확인하고 배포를 다시 시도합니다.

S3_GET_OBJECT_리소스를 _찾을 수 없음_

구성 요소 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 사용할 수 없을 때 이 오류가 발생할 수 있습니다. 아티팩트를 S3 버킷에 업로드했는지, 아티팩트 URI가 버킷에 있는 아티팩트의 S3 객체 URL과 일치하는지 확인합니다.

IO_매핑_오류

배포 문서 또는 레시피를 파싱할 때 I/O 오류가 발생할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

구성 요소 레시피 오류

레시피_구문분석_오류

레시피 구조에 오류가 있어 배포 레시피를 파싱할 수 없을 때 이 오류가 발생할 수 있습니다. 레시피의 형식이 올바른지 확인하고 배포를 다시 시도하세요.

레시피_메타데이터_구문분석_오류

클라우드에서 다운로드한 배포 레시피 메타데이터를 파싱하지 못할 때 이 오류가 발생할 수 있습니다. AWS Support에 문의하십시오.

아티팩트_URI_올바르지 않음_

레시피의 아티팩트 URI가 올바르게 포맷되지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피의 URI를 업데이트한 다음 배포를 다시 시도하세요.

S3_ARTIFACT_URI_NOT_VALID

레시피에 있는 아티팩트의 Amazon S3 URI가 유효하지 않을 때 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피의 URI를 업데이트한 다음 배포를 다시 시도하세요.

도커_아티팩트_URI_NOT_VALID

레시피에 있는 아티팩트의 Docker URI가 유효하지 않을 때 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피의 URI를 업데이트한 다음 배포를 다시 시도하세요.

빈_아티팩트_URI

아티팩트의 URI가 레시피에 지정되지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 URI가 누락된 아티팩트를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

빈_아티팩트_스킴

아티팩트에 대한 URI 스키마가 정의되지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피의 URI를 업데이트한 다음 배포를 다시 시도하세요.

지원되지 않는_아티팩트_스킴

실행 중인 nucleus 버전에서 URI 스킴을 지원하지 않을 때 이 오류가 발생할 수 있습니다. URI가 유효하지 않거나 nucleus 버전을 업데이트해야 합니다. URI가 유효하지 않은 경우 로그에서 유효하지 않은 URI를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

레시피_누락_매니페스트

매니페스트 섹션이 레시피에 포함되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피에 매니페스트를 추가하고 배포를 다시 시도하세요.

레시피_누락_아티팩트_해시_알고리즘

해시 알고리즘이 없는 레시피 내에 로컬이 아닌 아티팩트를 지정하면 이 오류가 발생할 수 있습니다. 알고리즘을 아티팩트에 추가한 다음 요청을 다시 시도합니다.

아티팩트_체크섬_불일치

다운로드한 아티팩트의 다이제스트가 레시피에 지정된 다이제스트와 다
를 때 이 오류가 발생할 수 있습니다. 레시피에 올바른 다이제스트가 포함
되어 있는지 확인한 다음 배포를 다시 시도하십시오. 자세한 내용은 Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption. 섹션을 참조하세요.

구성 요소_종속성_유효하지 않음_

배포 레시피에 지정된 종속성 유형이 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 레시피를 확인한 다음 요청을 다시 시도하세요.

구성_보간_오류

레시피 변수를 보간할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

IO_매핑_오류

배포 문서 또는 레시피를 파싱할 때 I/O 오류가 발생할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

AWS구성 요소 오류, 사용자 구성 요소 오류, 구성 요소 오류

구성요소에 문제가 있을 때 다음과 같은 오류 코드가 반환됩니다. 보고되는 실제 오류 유형은 오류를 발생시킨 특정 구성 요소에 따라 달라집니다. 그린그래스 핵이 해당 구성 요소를 다음과 같은 구성요소로 식별할 경우AWS IoT Greengrass, 반환됩니다`AWS_COMPONENT_ERROR`. 구성 요소가 사용자 구성 요소로 식별되면 Greengrass 핵이 반환됩니다.`USER_COMPONENT_ERROR`. 그린그래스 핵이 알아차리지 못하면 돌아온다`COMPONENT_ERROR`.

구성 요소_업데이트_오류

배포 중에 구성 요소가 업데이트되지 않을 때 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하십시오.

컴포넌트_파손

배포 중에 구성 요소가 손상되면 이 오류가 발생할 수 있습니다. 구성 요소 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도합니다.

제거_구성요소_오류

배포 중에 Nucleus가 구성 요소를 제거할 수 없을 때 이 오류가 발생할 수 있습니다. 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도합니다.

구성 요소_부트스트랩_타임아웃

구성 요소의 부트스트랩 작업이 구성된 제한 시간보다 오래 걸렸을 때 이 오류가 발생할 수 있습니다. 부트스트랩 작업의 제한 시간을 늘리거나 실행 시간을 줄인 다음 배포를 다시 시도하십시오.

구성 요소_부트스트랩_오류

구성 요소의 부트스트랩 작업에 오류가 있을 때 이 오류가 발생할 수 있습니다. 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도하십시오.

구성 요소_구성_잘못_유효하지 않음

Nucleus가 구성 요소에 대해 배포된 구성을 검증하지 못할 때 이 오류가 발생할 수 있습니다. 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도하십시오.

디바이스 오류

IO_쓰기_오류

파일에 쓸 때 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

IO_읽기_오류

파일을 읽을 때 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

디스크 공간_크리티컬

디스크 공간이 부족하여 배포 요청을 완료할 수 없을 때 이 오류가 발생할 수 있습니다. 사용 가능한 공간이 20Mb 이상이거나 더 큰 아티팩트를 담을 수 있을 만큼 충분해야 합니다. 디스크 공간을 확보한 다음 배포를 다시 시도하십시오.

IO_파일_속성_오류

파일 시스템에서 기존 파일 크기를 검색할 수 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

설정_권한_오류

다운로드한 아티팩트 또는 아티팩트 디렉터리에 권한을 설정할 수 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

IO_UNZIP_오류

아티팩트의 압축을 풀 수 없을 때 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

현지_레시피_찾을 수 없음_

레시피 파일의 로컬 사본을 찾을 수 없을 때 이 오류가 발생할 수 있습니다. 배포를 다시 시도해 보십시오.

로컬_레시피_손상됨

레시피의 로컬 사본이 다운로드 이후 변경된 경우 이 오류가 발생할 수 있습니다. 레시피의 기존 사본을 삭제하고 배포를 다시 시도하십시오.

로컬_레시피_메타데이터를 _찾을 수 없음_

레시피 메타데이터 파일의 로컬 복사본을 찾을 수 없을 때 이 오류가 발생할 수 있습니다. 배포를 다시 시도해 보십시오.

런치_디렉터리_손상됨

디렉토리가 Greengrass 핵을 시작하는 데 사용된 경우 이 오류가 발생할 수 있습니다 (`/greengrass/v2/alts/current`) 핵이 마지막으로 시작된 이후로 수정되었습니다. Nucleus를 다시 시작한 다음 배포를 다시 시도합니다.

해싱_알고리즘_사용 불가

기기의 Java 배포가 필수 해싱 알고리즘을 지원하지 않거나 구성 요소 레시피에 지정된 해시 알고리즘이 유효하지 않은 경우 이 오류가 발생할 수 있습니다.

디바이스_구성이 아티팩트_다운로드에 유효하지 않음_

디바이스 구성에 오류가 있어 배포로 인해 Amazon S3 또는 Greengrass 클라우드에서 아티팩트를 다운로드할 수 없는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 구성 오류를 확인한 다음 배포를 다시 시도합니다.

종속성 오류

도커_오류

Docker 이미지를 가져올 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

도커_서비스_사용할 수 없음

Greengrass가 Docker 레지스트리에 로그인하지 못할 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인한 다음 배포를 다시 시도하십시오.

도커_로그인_오류

Docker에 로그인할 때 예상치 못한 오류가 발생할 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인한 다음 배포를 다시 시도하십시오.

도커_풀_오류

레지스트리에서 Docker 이미지를 가져올 때 예상치 못한 오류가 발생할 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인한 다음 배포를 다시 시도하십시오.

도커_이미지_잘못_유효하지 않음

요청된 Docker 이미지가 존재하지 않을 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인하고 배포를 다시 시도하십시오.

도커_이미지_쿼리_오류

Docker에서 사용 가능한 이미지를 쿼리할 때 예상치 못한 오류가 발생할 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인하고 배포를 다시 시도합니다.

S3_ERROR

Amazon S3 아티팩트를 다운로드할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

S3_리소스를 찾을 수 없음_

Amazon S3 작업에서 404 오류가 반환될 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

S3_잘못된 요청

Amazon S3 작업에서 400 오류가 반환될 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인하고 요청을 다시 시도하십시오.

HTTP 오류

HTTP_요청_오류

HTTP 요청을 할 때 오류가 발생했을 때 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인합니다.

다운로드_배포_문서_오류

배포 문서를 다운로드할 때 HTTP 오류가 발생하면 이 오류가 발생할 수 있습니다. 로그에서 특정 HTTP 오류를 확인합니다.

GET_그린그래스_아티팩트_크기_오류

공용 구성 요소 아티팩트의 크기를 가져올 때 HTTP 오류가 발생하면 이 오류가 발생할 수 있습니다. 로그에서 특정 HTTP 오류를 확인합니다.

다운로드_그린그래스_아티팩트_오류

공용 구성 요소 아티팩트를 다운로드할 때 HTTP 오류가 발생하면 이 오류가 발생할 수 있습니다. 로그에서 특정 HTTP 오류를 확인합니다.

네트워크 오류

네트워크 오류

배포 중에 연결 문제가 발생할 때 이 오류가 발생할 수 있습니다. 디바이스의 인터넷 연결을 확인하고 배포를 다시 시도합니다.

뉴클리어스 에러

잘못된 요청

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS클라우드 오퍼레이션은 400 오류를 반환합니다. 로그를 확인하여 오류를 일으킨 API를 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

핵_버전_찾을 수 없음_

코어 장치가 활성 핵의 버전을 찾을 수 없을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

핵_재시작_실패

Nucleus 재시작이 필요한 배포 중에 Nucleus가 다시 시작되지 않을 때 이 오류가 발생할 수 있습니다. 로더 로그에서 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

설치된 구성 요소를 찾을 수 없음

Nucleus가 설치된 구성 요소를 찾을 수 없을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

배포_문서_유효하지 않음_

디바이스가 유효하지 않은 배포 문서를 수신할 때 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하십시오.

빈_배포_요청

기기가 빈 배포 요청을 받을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

배포_문서_구문분석_오류

배포 요청 형식이 예상 형식과 일치하지 않을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

구성 요소_메타데이터__유효하지 않은_배포

배포 요청에 유효하지 않은 구성 요소 메타데이터가 포함된 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

런치_디렉터리_손상됨

Greengrass 디바이스를 한 사물 그룹에서 다른 사물 그룹으로 이동한 다음 Greengrass를 다시 시작해야 하는 배포 시 원래 그룹으로 다시 이동할 때 이 오류가 발생할 수 있습니다. 오류를 해결하려면 기기에서 Greengrass의 시작 디렉터리를 다시 생성하십시오.

자세한 정보는 Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service을 참조하세요.

서버 오류

서버_오류

다음과 같은 경우에 이 오류가 발생할 수 있습니다.AWS서비스가 지금 요청을 처리할 수 없기 때문에 서비스 작업에서 500 오류가 반환됩니다. 나중에 배포를 다시 시도하십시오.

S3_서버_오류

Amazon S3 작업에서 500 오류가 반환될 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

클라우드 서비스 오류

해결_구성요소_후보_잘못된_응답

Greengrass 클라우드 서비스가 호환되지 않는 응답을 보내면 이 오류가 발생할 수 있습니다.`ResolveComponentCandidates`오퍼레이션. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

배포_문서_크기_초과됨

요청된 배포 문서가 최대 크기 할당량을 초과할 때 이 오류가 발생할 수 있습니다. 배포 문서의 크기를 줄이고 배포를 다시 시도합니다.

그린그래스_유물_크기_찾을 수 없음_

Greengrass가 공용 구성 요소 아티팩트의 크기를 가져올 수 없을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

배포_문서_유효하지 않음_

디바이스가 유효하지 않은 배포 문서를 수신할 때 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하십시오.

빈_배포_요청

기기가 빈 배포 요청을 받을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

배포_문서_구문분석_오류

배포 요청 형식이 예상 형식과 일치하지 않을 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

구성 요소_메타데이터__유효하지 않은_배포

배포 요청에 유효하지 않은 구성 요소 메타데이터가 포함된 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

일반 오류

이러한 일반 오류에는 관련 오류 유형이 없습니다.

배포_중단됨

Nucleus 종료 또는 기타 외부 이벤트로 인해 배포를 완료할 수 없을 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

아티팩트_다운로드_오류

아티팩트를 다운로드하는 데 문제가 있을 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

사용 가능한_구성 요소_버전 없음

구성 요소 버전이 클라우드 또는 로컬에 존재하지 않거나 종속성 해결 충돌이 있는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

구성 요소_패키지_로딩_오류

다운로드한 아티팩트를 처리하는 중 오류가 발생할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

클라우드_API_오류

를 호출하는 중 오류가 발생했을 때 이 오류가 발생할 수 있습니다.AWS서비스 API. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

IO_오류

배포 중에 I/O 오류가 발생할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드나 로그를 확인하십시오.

구성 요소_업데이트_오류

배포 중에 구성 요소가 업데이트되지 않을 때 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하십시오.

알 수 없는 오류

배포_실패

확인되지 않은 예외가 발생하여 배포에 실패할 때 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

배포_유형_유효하지 않음_

배포 유형이 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 최신 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하십시오.AWS Support.

세부 구성 요소 상태 코드

이 섹션의 상태 코드 및 솔루션을 사용하면 Greengrass nucleus 버전 2.8.0 이상을 사용할 때 구성 요소 관련 문제를 해결하는 데 도움이 됩니다.

이 항목의 많은 상태는 AWS IoT Greengrass 코어 로그에 추가 정보를 보고합니다. 이러한 로그는 코어 장치의 로컬 파일 시스템에 저장됩니다. 각 개별 구성 요소에 대한 로그가 있습니다. 로그 액세스에 대한 자세한 내용은 을 참조하십시오파일 시스템 로그에 액세스하십시오..

설치 오류

설치 스크립트를 실행하는 동안 오류가 발생할 때 이 메시지가 표시될 수 있습니다. 오류 코드는 구성 요소 로그에 보고됩니다. 설치 스크립트에서 오류를 확인하고 구성 요소를 다시 배포합니다.

설치 구성이 유효하지 않음

레시피의 `Install` 섹션이 유효하지 않아 구성 요소 설치를 완료하지 못한 경우 이 오류가 발생할 수 있습니다. 레시피의 설치 섹션에서 오류를 확인하고 배포를 다시 시도하세요.

설치_IO_오류

구성 요소 설치 중에 I/O 오류가 발생했을 때 이 오류가 발생할 수 있습니다. 오류 세부 내용을 보려면 을 호출할 수 있습니다.

설치_누락_기본_실행

구성 요소를 설치할 때 사용할 사용자 또는 그룹을 결정할 AWS IoT Greengrass 수 없는 경우 이 오류가 발생할 수 있습니다. 설치 레시피의 `runWith` 섹션에 유효한 사용자 또는 그룹이 포함되어 있는지 확인하십시오.

설치 시간 초과

구성된 제한 시간 내에 설치 스크립트가 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피 `Install` 섹션에 지정된 `Timeout` 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 설치 스크립트를 수정하십시오.

시작_오류

시작 스크립트를 실행하는 동안 오류가 발생할 때 이 메시지가 표시될 수 있습니다. 오류 코드는 구성 요소 로그에 보고됩니다. 설치 스크립트에서 오류를 확인하고 구성 요소를 다시 배포합니다.

시작_구성_유효하지 않음_

레시피의 `Startup` 섹션이 유효하지 않아 구성 요소 설치를 완료하지 못한 경우 이 오류가 발생할 수 있습니다. 레시피의 시작 섹션에서 오류를 확인하고 배포를 다시 시도하세요.

시작_IO_오류

구성 요소 시작 중에 I/O 오류가 발생했을 때 이 오류가 발생할 수 있습니다. 오류 세부 내용을 보려면 을 호출할 수 있습니다.

시작_누락_기본_실행

구성 요소를 실행할 때 사용할 사용자 또는 그룹을 결정할 AWS IoT Greengrass 수 없는 경우 이 오류가 발생할 수 있습니다. 스타트업 레시피의 `runWith` 섹션에 유효한 사용자 또는 그룹이 포함되어 있는지 확인하세요.

시작_타임아웃

구성된 제한 시간 내에 시작 스크립트가 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피 `Startup` 섹션에 지정된 `Timeout` 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 시작 스크립트를 수정하십시오.

실행 오류

구성 요소 스크립트를 실행하는 동안 오류가 발생할 때 이 메시지가 표시될 수 있습니다. 오류 코드는 구성 요소 로그에 보고됩니다. 실행 스크립트에서 오류를 확인하고 구성 요소를 다시 배포합니다.

실행_누락_기본_실행

구성 요소를 실행할 때 사용할 사용자 또는 그룹을 결정할 AWS IoT Greengrass 수 없는 경우 이 오류가 발생할 수 있습니다. 실행 레시피의 `runWith` 섹션에 유효한 사용자 또는 그룹이 포함되어 있는지 확인하세요.

실행_구성_잘못_유효하지 않음

레시피의 `Run` 섹션이 유효하지 않아 구성 요소를 실행할 수 없는 경우 이 오류가 발생할 수 있습니다. 레시피의 실행 섹션에서 오류를 확인하고 배포를 다시 시도하세요.

실행_IO_오류

구성 요소가 실행되는 동안 I/O 오류가 발생했을 때 이 오류가 발생할 수 있습니다. 오류 세부 내용을 보려면 을 호출할 수 있습니다.

런_타임아웃

구성된 제한 시간 내에 실행 스크립트가 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피 `Run` 섹션에 지정된 `Timeout` 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 실행 스크립트를 수정하십시오.

종료_오류

구성 요소 스크립트를 종료하는 동안 오류가 발생할 때 이 오류가 발생할 수 있습니다. 오류 코드는 구성 요소 로그에 보고됩니다. 종료 스크립트에서 오류를 확인하고 구성 요소를 다시 배포합니다.

종료_타임아웃

구성된 제한 시간 내에 종료 스크립트가 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피 `Shutdown` 섹션에 지정된 `Timeout` 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 실행 스크립트를 수정하십시오.

AWS IoT Greengrass Version 2 리소스 태깅

태그를 사용하여 AWS IoT Greengrass의 리소스를 구성하고 관리할 수 있습니다. 태그를 사용하여 리소스에 메타데이터를 할당하고, IAM 정책에서 태그를 사용하여 리소스에 대한 조건부 액세스를 정의할 수 있습니다.

> **Note**
>
> 현재 AWS IoT 결제 그룹 또는 비용 할당 보고서에서는 Greengrass 리소스 태그가 지원되지 않습니다.

AWS IoT Greengrass V2에서 태그 사용

태그를 사용하여 목적, 소유자, 환경 또는 사용 사례에 맞는 기타 기준에 따라 AWS IoT Greengrass 리소스를 분류할 수 있습니다. 동일한 유형의 리소스가 많은 경우 태그를 사용하여 특정 리소스를 보다 쉽게 식별할 수 있습니다.

각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 예를 들어, 핵심 디바이스에 대해 각 디바이스를 소유한 고객별로 이를 추적하는 데 도움이 되는 태그 세트를 정의할 수 있습니다. 각 리소스 유형에 대한 요건을 충족하는 태그 키 세트를 생성하는 것이 좋습니다. 일정한 태그 키 집합을 사용하면 리소스를 더 쉽게 관리할 수 있습니다.

태그가 붙은AWS Management Console

의 태그 편집기는 모든 서비스의 리소스에 대해 모든AWS 서비스의 리소스에 대한 태그를 생성하고 관리할 수 있는 중앙 통합 방법을AWS Management Console 제공합니다. 자세한 내용은 AWS Resource Groups사용자 안내서의 Tag Editor를 참조하십시오.

AWS IoT Greengrass V2API를 사용한 태그

AWS IoT Greengrass V2API를 사용하여 태그를 사용할 수도 있습니다. 태그를 생성하기 전에 태그 지정에 대한 제한 사항을 유의하십시오. 자세한 내용은 의 태그 이름 지정 및 사용 규칙을 참조하십시오 AWS 일반 참조.

- 리소스를 만들 때 태그를 추가하려면 리소스의 `tags` 속성에서 태그를 정의합니다.
- 기존 리소스에 태그를 추가하거나 태그 값을 업데이트하려면 TagResource작업을 사용합니다.
- 리소스에서 태그를 제거하려면 UntagResource작업을 사용합니다.

- 리소스와 연결된 태그를 검색하려면 ListTagsForResource작업을 사용하거나 리소스를 설명하고 해당tags 속성을 검사하십시오.

다음 표에는AWS IoT Greengrass V2 API와 해당Create andDescribe 또는Get 작업을 사용하여 태그를 지정할 수 있는 리소스가 나와 있습니다.

태그 지정 가능한 AWS IoT Greengrass V2 리소스

리소스	작업 생성	설명 또는 작업 받기
코어 디바이스	없음. 장치에서AWS IoT Greengrass Core 소프트웨어를 실행하여 코어 장치를 생성합니다.	GetCoreDevice
구성 요소	CreateComponentVersion	DescribeComponent, GetComponent
배포	CreateDeployment	GetDeployment

다음 작업을 사용하여 태그 지정이 가능한 리소스의 태그를 보고 관리합니다.

- TagResource— 리소스에 태그를 추가하거나 기존 태그의 값을 업데이트합니다.
- ListTagsForResource— 리소스의 태그를 나열합니다.
- UntagResource— 리소스에서 태그를 제거합니다.

언제든지 리소스에 태그를 추가하거나 리소스에서 태그를 제거할 수 있습니다. 태그 키의 값을 변경하려면 동일한 키와 새로운 값을 정의하는 리소스에 태그를 추가합니다. 새 값은 이전 값을 대체합니다. 값을 빈 문자열로 설정할 수 있지만,값을 Null로 설정할 수는 없습니다.

리소스를 삭제하면 해당 리소스와 연결된 태그도 삭제됩니다.

IAM 정책에 태그 사용

IAM 정책에서 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.

> **Note**
>
> 태그를 사용하여 리소스에 대한 사용자의 액세스를 허용하거나 거부하는 경우, 동일한 리소스에 대한 사용자의 태그 추가 또는 제거 권한을 거부해야 합니다. 그렇지 않으면 사용자가 제한을 피해 태그를 수정하여 리소스에 대한 액세스 권한을 얻을 수 있습니다.

정책 설명의Condition 요소 (Condition블록이라고도 함) 에서 다음과 같은 조건 컨텍스트 키와 값을 사용할 수 있습니다.

`greengrassv2:ResourceTag/`*`tag-key`*`: `*`tag-value`*

특정 태그가 있는 리소스에 대한 작업을 허용하거나 거부합니다.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

태그 지정 가능한 리소스를 만들거나 수정할 때 특정 태그를 사용하거나 사용하지 않도록 요구합니다.

`aws:TagKeys: [`*`tag-key`*`, ...]`

태그 지정 가능한 리소스를 만들거나 수정할 때 특정 태그 키 세트를 사용하거나 사용하지 않도록 요구합니다.

> **Note**
>
> IAM 정책의 조건 컨텍스트 키와 값은 태깅 가능한 리소스가 필수 파라미터로 있는 작업에만 적용됩니다. 예를 들어, 에 대해 태그 기반 조건부 액세스를 설정할 수 ListCoreDevices있습니다.

자세한 내용은 IAM 사용 설명서의 리소스 태그 및 IAM JSON 정책 참조를 사용한 리소스 액세스 제어를 참조하십시오.AWS

AWS CloudFormation을 사용하여 AWS IoT Greengrass 리소스 생성

AWS IoT Greengrass는 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 서비스인 AWS CloudFormation과 통합됩니다. 모든 것을 설명하는 템플릿을 만듭니다.AWS원하는 리소스 (예: 구성 요소 버전 및 배포) 및AWS CloudFormation해당 리소스를 자동으로 프로비저닝하고 구성합니다.

AWS CloudFormation을 사용할 때 템플릿을 재사용하여 AWS IoT Greengrass 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번 설명한 후 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝할 수 있습니다.

AWS IoT Greengrass 및 AWS CloudFormation 템플릿

AWS IoT Greengrass 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 AWS CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하면 AWS CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer이란 무엇입니까?를 참조하세요.

AWS IoT Greengrass에서 구성 요소 버전 및 배포를 만들 수 있습니다.AWS CloudFormation. 구성 요소 버전 및 배포를 위한 JSON 및 YAML 템플릿의 예를 비롯한 자세한 내용은 를 참조하세요.AWS IoT Greengrass리소스 유형 참조에서AWS CloudFormation사용 설명서.

ComponentVersion 템플릿 예

다음은 간단한 구성 요소 버전의 YAML 템플릿입니다. JSON 레시피에는 가독성을 위해 줄 바꿈이 추가되었습니다.

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
  TestSimpleComponentVersion:
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
```

```
      InlineRecipe: !Sub
        - "{\n
            \"RecipeFormatVersion\": \"2020-01-25\",\n
            \"ComponentName\": \"component1\",\n
            \"ComponentVersion\": \"${ComponentVersion}\",\n
            \"ComponentType\": \"aws.greengrass.generic\",\n
            \"ComponentDescription\": \"This\",\n
            \"ComponentPublisher\": \"You\",\n
            \"Manifests\": [\n
            {\n
              \"Platform\": {\n
              \"os\": \"darwin\"\n
            },\n
            \"Lifecycle\": {},\n
            \"Artifacts\": []\n
           },\n
           {\n
             \"Lifecycle\": {},\n
             \"Artifacts\": []\n
           }\n
          ],\n
          \"Lifecycle\": {\n
          \"install\": {\n
            \"script\": \"yuminstallpython\"\n
          }\n
         }\n
        }"
        - { ComponentVersion: !Ref ComponentVersion }
```

배포 템플릿 예

다음은 배포를 위한 간단한 템플릿을 정의하는 YAML 파일입니다.

```
Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
  TestDeployment:
    Type: AWS::GreengrassV2::Deployment
    Properties:
      Components:
```

```
        component1:
          ComponentVersion: !Ref ComponentVersion
      TargetArn: !Ref TargetArn
      DeploymentName: CloudFormationIntegrationTest
      DeploymentPolicies:
        FailureHandlingPolicy: DO_NOTHING
        ComponentUpdatePolicy:
          TimeoutInSeconds: 5000
          Action: SKIP_NOTIFY_COMPONENTS
        ConfigurationValidationPolicy:
          TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

AWS CloudFormation에 대해 자세히 알아보기

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation API 참조
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

오픈소스 AWS IoT Greengrass 코어 소프트웨어

AWS IoT Greengrass코어 소프트웨어의 AWS IoT Greengrass Version 2 엣지 런타임 (nucleus) 및 기타 구성 요소는 오픈 소스입니다. 즉, 코드를 검토하여 애플리케이션과의 상호 작용 문제를 해결할 수 있습니다. 또한 특정 소프트웨어 및 하드웨어 요구 사항에 맞게 AWS IoT Greengrass Core 소프트웨어를 사용자 지정하고 확장할 수 있습니다.

AWS IoT Greengrass코어 소프트웨어의 오픈 소스 리포지토리에 대한 자세한 내용은 의 aws-greengrass 조직을 참조하십시오. GitHub 오픈 소스 소프트웨어 사용에는 해당 리포지토리의 오픈 소스 라이선스가 적용됩니다. GitHub

오픈 소스 라이선스가 적용되지 않는 AWS IoT Greengrass 코어 소프트웨어 및 구성 요소의 사용은 AWS Greengrass Core 소프트웨어 라이선스의 적용을 받습니다.

AWS IoT Greengrass V2 개발자 안내서의 문서 기록

다음 표에는 이번 릴리스의 설명서가 설명되어 AWS IoT Greengrass Version 2있습니다.

- API 버전: 2020-11-30

변경 사항	설명	날짜
AWS IoT Device Tester v4.9.2와 GGV2Q v2.5.2가 출시되었습니다.	V2용 IDT 버전 4.9.2를 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 V2 AWS IoT Greengrass 퀄리티 스위트 (GGV2Q) v2.5.2가 포함되어 있으며 그린그래스 뉴클리어스 버전 2.12.0, 2.11.0, 2.10.0, 2.9.5를 지원합니다.	2024년 3월 18일
룩아웃 포 비전 엣지 에이전트 v1.2.0 출시	Lookout for Vision 엣지 에이전트 v1.2.0을 사용할 수 있습니다.	2024년 3월 11일
AWS IoT Greengrass 코어 v2.12.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.2를 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2024년 2월 15일
섀도우 매니저 v2.3.6 출시	섀도우 매니저 v2.3.6을 사용할 수 있습니다. 이 릴리스에서는 장치가 오프라인 상태일 때 AWS 클라우드 업데이트를 통해 삭제된 섀도우 속성이 다시 연결되더라도 로컬 섀도우에 계속 남아 있는 문제를 수정합니다.	2024년 2월 14일
람다 런처 v2.0.13 출시	Lambda 런처 구성 요소 버전 2.0.13을 사용할 수 있습니다.	2024년 2월 14일

	이번 릴리스에는 일반적인 버그 수정 및 개선 사항이 포함되어 있습니다.	
디스크 스풀러 v1.0.3 출시	디스크 스풀러 구성 요소 v1.0.3을 사용할 수 있습니다. 이 릴리스는 데이터베이스 연결을 재사용하여 성능을 개선합니다.	2024년 2월 14일
룩아웃 포 비전 엣지 에이전트 v1.1.9 출시	Lookout for Vision 엣지 에이전트 v1.1.9를 사용할 수 있습니다.	2024년 1월 17일
그린그래스 개발 키트 CLI v1.6.2	그린그래스 개발 키트 CLI의 버전 1.6.2를 사용할 수 있습니다. 이 버전은 상대 경로로 인해 Windows gradlew.bat 가 작동하지 않는 문제를 해결합니다. 이 버전에는 추가 개선 사항도 포함되어 있습니다.	2024년 1월 16일
새 CloudTrail 데이터 이벤트	이제 AWS CloudTrail 데이터 이벤트를 기록하여 구성 요소 가져오기 또는 배포 구성과 같은 리소스 작업에 대한 정보를 얻을 수 있습니다. 이러한 이벤트를 사용하여 Greengrass 장치 작동에 대한 통찰력을 얻으십시오.	2023년 12월 20일
룩아웃 포 비전 엣지 에이전트 v1.1.8 출시	Lookout for Vision 엣지 에이전트 v1.1.8을 사용할 수 있습니다.	2023년 12월 12일

스트림 매니저 v2.1.12가 출시되었습니다.	이제 스트림 매니저 v2.1.12를 사용할 수 있습니다. 이번 릴리스에서는 Greengrass가 AWS 서비스 호출을 위한 자격 증명 세트를 선택하는 데 사용하는 순서를 변경합니다.	2023년 12월 8일
MQTT 브리지 v2.3.1이 출시되었습니다.	MQTT 브리지 v2.3.1을 사용할 수 있습니다. 이 릴리스는 로컬 MQTT 클라이언트가 연결 해제 루프에 빠지는 드문 문제를 수정합니다.	2023년 12월 8일
디스크 스풀러 v1.0.2 출시	디스크 스풀러 구성 요소 v1.0.2를 사용할 수 있습니다. 이 릴리스에서는 특정 경우에 MQTT 메시지 형식 필드가 유지되지 않는 문제가 수정되었습니다.	2023년 12월 8일
클라이언트 장치 인증 구성 요소 v2.4.5 출시	클라이언트 장치 인증 구성 요소 v2.4.5를 사용할 수 있습니다. 이번 릴리스는 선택 규칙의 사물 이름 끝에 와일드카드 지원을 추가하고 특정 경우에 인증서가 새 연결 정보로 업데이트되지 않는 문제를 수정했습니다.	2023년 12월 8일
AWS IoT Greengrass Core v2.12.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.1을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 12월 8일

그린그래스 개발 키트 CLI v1.6.1	그린그래스 개발 키트 CLI의 버전 1.6.1을 사용할 수 있습니다. 이 버전에는 버그 수정 및 개선 사항이 포함되어 있습니다.	2023년 12월 6일
레시피 검증	구성 요소 버전을 생성할 때 구성 요소 레시피를 검증하는 레시피 검증 기능을 추가했습니다.	2023년 11월 16일
퍼블리셔가 지원하는 구성 요소	AWS IoT Greengrass 이제 퍼블리셔 지원 구성 요소를 제공합니다. 이러한 구성 요소는 타사 공급업체에서 개발, 제공 및 서비스합니다.	2023년 11월 16일
그린그래스 테스팅 프레임워크 v1.2.0 출시	그린그래스 테스팅 프레임워크 v1.2.0을 사용할 수 있습니다.	2023년 11월 15일

그린그래스 개발 키트 CLI v1.6.0	그린그래스 개발 키트 CLI의 버전 1.6.0을 사용할 수 있습니다. 이 버전은 `component build` 및 `component publish` 명령 중에 Greengrass 레시피 스키마에 대한 레시피 검증 검사를 추가합니다. 이 업데이트를 통해 개발자는 구성 요소 생성 프로세스 초기에 구성 요소 레시피 내에서 실행 가능한 문제를 식별할 수 있습니다. 또한 이 버전에는 명령으로 가져올 수 있는 신뢰도 테스트 도구 모음이 템플릿에 추가되었습니다. `test-e2e init` 이 신뢰도 테스트 스위트에는 기본 구성 요소 테스트 요구 사항에 맞게 사용 및 확장할 수 있는 8개의 일반 테스트가 포함되어 있습니다.	2023년 11월 15일
AWS IoT Device Tester v4.9.1은 그린그래스 뉴클리어스 버전 2.12.0을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.9.1은 이제 그린그래스 핵 버전 2.12.0을 지원합니다.	2023년 11월 7일
AWS IoT Greengrass 코어 v2.12.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 11월 7일
VPC에서 그린그래스 코어 디바이스 운영	VPC에서 Greengrass 코어 디바이스를 운영할 수 있습니다. 이 기능을 사용하면 공용 인터넷 액세스 없이 VPC에서 배포를 수행할 수 있습니다.	2023년 11월 3일

그린그래스 CLI v2.12.0 출시	그린그래스 CLI 구성 요소 v2.12.0을 사용할 수 있습니다.	2023년 10월 30일
스트림 매니저 v2.1.10이 출시되었습니다.	이제 스트림 매니저 v2.1.10을 사용할 수 있습니다. 이 릴리스는 HTTPS 프록시 구성이 Greengrass CA 인증서 체인을 신뢰하지 않는 문제를 수정합니다.	2023년 10월 26일
람다 런처 v2.0.12 출시	Lambda 런처 구성 요소 버전 2.0.12를 사용할 수 있습니다. 이 릴리스는 이전 프로세스가 제대로 중지되지 않은 경우 Lambda Launcher에서 오류가 발생할 수 있는 문제를 수정합니다.	2023년 10월 26일
그린그래스 개발 키트 CLI v1.5.0	그린그래스 개발 키트 CLI의 버전 1.5.0을 사용할 수 있습니다. 이 버전은 `excludes` 빌드 옵션이 인식되는 경우 해당 패턴을 업데이트합니다. `build_system zip` 이제 이 버전에서는 와일드카드 문자를 기반으로 경로 이름과 일치하는 글로브 패턴을 인식합니다. 이를 통해 제외할 디렉터리를 사용자 지정할 수 있습니다.	2023년 10월 26일
룩아웃 포 비전 엣지 에이전트 v1.1.7 출시	Lookout for Vision 엣지 에이전트 v1.1.7을 사용할 수 있습니다.	2023년 10월 24일

섀도우 매니저 v2.3.4 출시	섀도우 매니저 v2.3.4를 사용할 수 있습니다. 이 릴리스에는 null 및 빈 섀도우 상태 문서에 대한 지원이 추가되었습니다.	2023년 10월 18일
로그 관리자 v2.3.6이 출시되었습니다.	로그 관리자 구성 요소 v2.3.6을 사용할 수 있습니다.	2023년 10월 18일
로컬 디버그 콘솔 v2.4.0 출시	로컬 디버그 콘솔 구성 요소 v2.4.0을 사용할 수 있습니다.	2023년 10월 18일
람다 매니저 v2.3.1 출시	Lambda 관리자 구성 요소 v2.3.1을 사용할 수 있습니다.	2023년 10월 18일
그린그래스 CLI v2.11.3 출시	그린그래스 CLI 구성 요소 v2.11.3을 사용할 수 있습니다.	2023년 10월 18일
AWS IoT Greengrass 코어 v2.11.3 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.11.3을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 10월 18일
보안 터널링 v1.0.17이 출시되었습니다.	보안 터널링 v1.0.17을 사용할 수 있습니다.	2023년 10월 4일
그린그래스 개발 키트 CLI v1.4.0	그린그래스 개발 키트 CLI의 버전 1.4.0을 사용할 수 있습니다. 이 버전에는 기존 GDK 구성 파일 내의 필드를 수정하기 위한 대화형 프롬프트를 시작하는 새 `config` 명령이 추가되었습니다. 또한 이 버전은 계속 진행하기 전에 `gdk component build` 및 `gdk component publish` 명령을 수정하여 레시피 크기가 Greengrass 요구 사항 (<=16000바이트) 내에 있는지 확인합니다.	2023년 10월 2일

모켓 MQTT 3.1.1 브로커 v2.3.5가 출시되었습니다.	모켓 MQTT 3.1.1 브로커 컴포넌트 v2.3.5를 사용할 수 있습니다. 이 버전은 Moquette를 버전 0.17로 업데이트합니다.	2023년 9월 28일
MQTT 브리지 v2.3.0 출시	MQTT 브리지 v2.3.0을 사용할 수 있습니다. 이 릴리스에는 로컬 MQTT 소스와 로컬 MQTT 소스 간의 AWS IoT Core 브리징을 위한 MQTT 5 지원이 추가되었습니다.	2023년 9월 28일
룩아웃 포 비전 엣지 에이전트 v1.1.6 출시	Lookout for Vision 엣지 에이전트 v1.1.6을 사용할 수 있습니다.	2023년 9월 27일
람다 매니저 v2.3.0 출시	Lambda 관리자 구성 요소 v2.3.0을 사용할 수 있습니다.	2023년 9월 15일
람다 런처 v2.0.11 출시	Lambda 런처 구성 요소 버전 2.0.11을 사용할 수 있습니다. 이 버전은 람다 매니저 2.3.0을 지원합니다.	2023년 9월 15일
모켓 MQTT 3.1.1 브로커 v2.3.4 출시	모켓 MQTT 3.1.1 브로커 컴포넌트 v2.3.4를 사용할 수 있습니다.	2023년 9월 1일
Greengrass 테스트 프레임워크	GTF는 자동화를 지원하는 end-to-end 빌딩 블록 모음입니다. 이를 통해 AWS IoT Greengrass Version 2 내부 고객은 서비스 팀이 소프트웨어 변경 검증, 자동 승인 및 품질 보증 목적으로 사용하는 것과 동일한 테스트 프레임워크를 사용할 수 있습니다.	2023년 8월 11일

AWS IoT Greengrass 코어 v2.11.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.11.2를 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 8월 9일
그린그래스 개발 키트 CLI v1.3.0	그린그래스 개발 키트 CLI의 버전 1.3.0을 사용할 수 있습니다. 이 버전에는 오픈 테스트 프레임워크를 사용한 구성 요소 end-to-end 테스트를 지원하는 새 `test-e2e` 명령이 추가되었습니다.	2023년 7월 21일
AWS IoT Greengrass 코어 v2.11.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.11.1을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 7월 21일
디스크 스풀러 v1.0.0 출시	디스크 스풀러 구성 요소 v1.0.0을 사용할 수 있습니다.	2023년 6월 28일
AWS IoT Greengrass 코어 v2.11.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.11.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 6월 28일
AWS IoT Greengrass 코어 v2.10.3 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.3을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 6월 21일
AWS IoT Greengrass 코어 v2.10.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.2를 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 6월 5일

AWS IoT Greengrass 코어 v2.10.1 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.10.1을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 5월 11일
AWS IoT Greengrass 코어 v2.10.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 5월 9일
SageMaker 엣지 매니저는 단종되었습니다.	아마존 SageMaker 엣지 매니저 구성 요소는 2024년 4월 26일에 단종됩니다.	2023년 4월 28일
AWS IoT Greengrass 코어 v2.9.6 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.6을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 4월 20일
로그 관리자 v2.3.2 출시	로그 관리자 구성 요소 v2.3.2를 사용할 수 있습니다.	2023년 4월 19일
스트림 매니저 v2.1.4 출시	이제 스트림 매니저 v2.1.4를 사용할 수 있습니다. 이번 릴리스에서는 단일 배치 내에서 타임스탬프가 동일한 동일한 프로퍼티 에셋에 대한 항목이 `ConflictingOperationException` SiteWise API에서 반환되어 스트림 관리자가 계속 재시도하는 문제를 수정했습니다. 또한 이번 릴리스에서는 기본 연결 제한 시간을 3초에서 1분으로 업데이트합니다.	2023년 4월 13일

그린그래스 개발 키트 CLI v1.2.3	그린그래스 개발 키트 CLI의 버전 1.2.3을 사용할 수 있습니다. 이 버전에는 버그 수정이 포함되어 있습니다.	2023년 4월 13일
클라이언트 장치 인증 구성 요소 v2.4.0 출시	클라이언트 장치 인증 구성 요소 v2.4.0을 사용할 수 있습니다. 이번 릴리스에는 Greengrass Client Device 대시보드에 표시할 수 있는 운영 메트릭을 내보내는 클라이언트 장치 인증에 대한 지원이 추가되었습니다.	2023년 4월 10일
그린그래스 개발 키트 CLI v1.2.2	그린그래스 개발 키트 CLI의 버전 1.2.2를 사용할 수 있습니다. 이 버전에는 개선 사항 및 버그 수정이 포함되어 있습니다.	2023년 4월 7일
AWS IoT Greengrass 코어 v2.9.5 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.5를 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 3월 30일
스트림 매니저 v2.1.3 출시	이제 스트림 매니저 v2.1.3을 사용할 수 있습니다. 이 릴리스는 Windows OS에서 SYSTEM 사용자로 실행할 때 발생하는 시작 문제를 수정합니다.	2023년 3월 7일
모드버스-RTU 프로토콜 어댑터 v2.1.5 출시	모드버스-RTU 프로토콜 어댑터 컴포넌트 v2.1.5를 사용할 수 있습니다. 이번 릴리스는 작업 관련 문제를 수정했습니다. `ReadDiscreteInput`	2023년 3월 7일

클라이언트 장치 인증 구성 요소 v2.3.2 출시	클라이언트 장치 인증 구성 요소 v2.3.2를 사용할 수 있습니다. 이 릴리스에는 구성 요소가 오프라인 상태에서 다시 시작할 때 인증서 주체를 올바르게 생성하도록 호스트 이름 정보 캐싱에 대한 지원이 추가되었습니다.	2023년 3월 7일
AWS IoT Device Tester v4.7.0은 그린그래스 핵 버전 2.9.4를 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.7.0은 이제 그린그래스 핵 버전 2.9.4를 지원합니다.	2023년 3월 2일
그린그래스 커맨드 라인 인터페이스 v1.2.0 출시	Greengrass 명령줄 인터페이스 v1.2.0을 사용할 수 있습니다.	2023년 2월 28일
AWS IoT Greengrass 코어 v2.9.4 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.4를 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 2월 24일
섀도우 매니저 v2.3.1 출시	섀도우 매니저 v2.3.1을 사용할 수 있습니다. 이 릴리스에서는 클라우드 섀도우 업데이트가 동기화되지 않을 수 있는 조건을 수정합니다. 또한 이번 릴리스에서는 명명된 섀도우 동기화 구성의 변경 사항이 하나의 명명된 섀도우에만 적용되는 문제를 수정했습니다.	2023년 2월 21일
AWS IoT Device Tester v4.7.0은 그린그래스 핵 버전 2.9.3을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.7.0은 이제 그린그래스 핵 버전 2.9.3을 지원합니다.	2023년 2월 9일

IAM 모범 사례 업데이트	IAM 모범 사례에 따라 가이드가 업데이트되었습니다. 자세한 내용은 IAM의 보안 모범 사례를 참조하십시오.	2023년 2월 3일
AWS IoT Greengrass 코어 v2.9.3 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.3을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2023년 2월 1일
로그 관리자 v2.3.1이 출시되었습니다.	로그 관리자 v2.3.1을 사용할 수 있습니다.	2023년 1월 27일
AWS IoT Device Tester v4.7.0은 그린그래스 핵 버전 2.9.2를 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.7.0은 이제 그린그래스 핵 버전 2.9.2를 지원합니다.	2023년 1월 3일
섀도우 매니저 v2.3.0 출시	섀도우 매니저 v2.3.0을 사용할 수 있습니다. 이 릴리스에서는 장치가 Greengrass 장치 개인 키를 하드웨어 보안 모듈에 저장할 때 섀도우가 동기화되지 않을 수 있는 문제를 수정합니다.	2022년 12월 29일
AWS IoT 플릿 프로비저닝 플러그인 v1.2.0 출시	AWS IoT 플릿 프로비저닝 플러그인 v1.2.0을 사용할 수 있습니다. 이번 릴리스에는 구성 가능한 개인 키 경로가 포함된 인증서 서명 요청을 통한 장치 프로비저닝에 대한 지원이 추가되었습니다.	2022년 12월 22일
AWS IoT Greengrass 코어 v2.9.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.2를 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2022년 12월 22일

AWS IoT Device Tester v4.7.0과 함께 GGV2Q v2.5.0이 출시되었습니다.	V2용 IDT 버전 4.7.0을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 쿼리티 스위트 (GGV2Q) v2.5.0이 포함되어 있으며 그린그래스 뉴클리어스 버전 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 및 2.6.0을 지원합니다.	2022년 12월 13일
섀도우 매니저 v2.2.4 출시	로컬 섀도우 문서를 업데이트할 때 섀도우 크기 검증이 클라우드와 일치하지 않던 문제를 수정합니다. 또한 배포가 구성 노드에서 a를 수행하는 경우 섀도우 관리자가 구성 업데이트 수신을 중지하는 RESET 문제도 수정됩니다.	2022년 12월 8일
룩아웃 포 비전 엣지 에이전트 1.1.1 출시	Lookout Edge Agent 구성 요소 v1.1.1을 사용할 수 있습니다.	2022년 12월 5일
로그 매니저 v2.3.0 출시	로그 관리자 구성 요소 v2.3.0을 사용할 수 있습니다.	2022년 11월 18일
AWS IoT Device Tester v4.5.11은 그린그래스 뉴클리어스 버전 2.9.1을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.5.11은 이제 그린그래스 핵 버전 2.9.1을 지원합니다.	2022년 11월 18일
AWS IoT Greengrass 코어 v2.9.1 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.1을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2022년 11월 18일

AWS IoT Device Tester v4.5.11은 그린그래스 뉴클리어스 버전 2.9.0을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.5.11은 이제 그린그래스 핵 버전 2.9.0을 지원합니다.	2022년 11월 17일
스트림 매니저 v2.1.2 출시	이제 스트림 매니저 v2.1.2를 사용할 수 있습니다. 이 릴리스는 영어가 아닌 언어를 사용하는 Windows OS의 문제를 수정합니다.	2022년 11월 15일
AWS IoT Greengrass 코어 v2.9.0 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.9.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2022년 11월 15일
AWS IoT Device Tester v4.5.11은 그린그래스 뉴클리어스 버전 2.8.1을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.5.11은 이제 그린그래스 뉴클리어스 버전 2.8.1을 지원합니다.	2022년 10월 19일
AWS IoT Device Tester v4.5.11과 함께 GGV2Q v2.4.1이 출시되었습니다.	V2용 IDT 버전 4.5.11을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.4.1이 포함되어 있으며 그린그래스 뉴클리어스 버전 2.8.0, 2.7.0 및 2.6.0을 지원합니다.	2022년 10월 13일
AWS IoT Greengrass 코어 v2.8.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.8.1을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2022년 10월 13일

AWS IoT Greengrass 코어 v2.8.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.8.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS	2022년 10월 7일
AWS CloudFormation 배포에 대한 지원이 추가되었습니다.	AWS CloudFormation 이제 AWS IoT Greengrass 배포를 리소스로 지원합니다.	2022년 10월 6일
SageMaker 엣지 매니저 v1.3.0 출시	아마존 SageMaker 엣지 매니저 구성 요소 v1.3.0을 사용할 수 있습니다. 이 릴리스에는 TensorRT 모델 캐시의 디스크 크기를 설정하는 이 구성 요소에 대한 지원이 추가되었으며, 예측 동시성을 개선하여 GPU와 같은 기기 가속기 엔진을 더 잘 사용할 수 있도록 합니다.	2022년 9월 1일
IPC (프로세스 간 통신) 클라이언트 V2 사용	IPC 클라이언트 V2에 대한 정보가 추가되었습니다. 이 정보는 IPC 작업을 사용하기 위해 작성해야 하는 코드의 양을 줄이고 IPC 클라이언트 V1에서 발생할 수 있는 일반적인 오류를 방지하는 데 도움이 됩니다.	2022년 8월 12일
AWS IoT Device Tester GGV2Q v2.4.0과 함께 v4.5.8이 출시되었습니다.	V2용 IDT 버전 4.5.8을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.4.0이 포함되어 있으며 그린그래스 뉴클리어스 버전 2.7.0, 2.6.0 및 2.5.6을 지원합니다.	2022년 8월 12일

SageMaker 엣지 매니저 v1.2.0 출시	아마존 SageMaker 엣지 매니저 구성 요소 v1.2.0을 사용할 수 있습니다. 이번 릴리스에는 Amazon S3에 업로드한 SageMaker NEO 컴파일 모델을 자동으로 검색하는 이 구성 요소에 대한 지원이 추가되어 배포를 생성할 필요 없이 새 모델을 배포할 수 있습니다. AWS IoT Greengrass	2022년 8월 3일
AWS IoT Device Tester v4.5.3은 그린그래스 핵 버전 2.7.0을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.5.3은 이제 그린그래스 핵 버전 2.7.0을 지원합니다.	2022년 8월 1일
스트림 매니저 v2.1.0 출시	이제 스트림 매니저 v2.1.0을 사용할 수 있습니다. 이번 릴리스에는 EventBridge Amazon에 텔레메트리 지표를 전송할 수 있는 지원이 포함되어 있습니다.	2022년 7월 28일
AWS IoT Greengrass 코어 v2.7.0 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.7.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS여기에는 EventBridge Amazon에 텔레메트리 지표를 전송할 수 있는 지원이 포함됩니다.	2022년 7월 28일

IoT SiteWise 퍼블리셔 v2.2.0 출시	IoT SiteWise 퍼블리셔 구성 요소 v2.2.0을 사용할 수 있습니다. 이번 릴리스에서는 데이터를 AWS IoT SiteWise 서비스에 전송하기 전에 데이터를 압축하도록 구성 요소를 업데이트하여 대역폭 사용량을 최대 75% 까지 줄입니다.	2022년 7월 19일
자습서: 클라이언트 장치 섀도우와 상호 작용하는 구성 요소 개발	자습서에 새 모듈이 추가되었습니다. MQTT를 통해 로컬 IoT 장치와 상호 작용하여 클라이언트 장치 섀도우와 상호 작용하는 구성 요소를 개발하는 방법을 배울 수 있습니다.	2022년 7월 18일
로컬 MQTT 브로커를 선택하세요.	클라이언트 디바이스를 코어 디바이스에 연결하는 로컬 MQTT 브로커를 선택하는 방법에 대한 정보가 추가되었습니다.	2022년 7월 18일
AWS IoT Device Tester v4.5.3은 그린그래스 핵 버전 2.6.0을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.5.3은 이제 그린그래스 핵 버전 2.6.0을 지원합니다.	2022년 6월 29일

AWS IoT Greengrass 코어 v2.6.0 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.6.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS여기에는 클라이언트 디바이스 섀도 및 클라이언트 디바이스용 로컬 MQTT 5 브로커에 대한 지원이 포함됩니다. 또한 로컬 게시/구독 주제의 와일드카드, 구성 요소 구성의 레시피 변수, IPC 인증 정책의 와일드카드에 대한 지원도 포함됩니다. 이러한 기능을 사용하면 여러 핵심 장치에 배포하는 구성 요소를 보다 쉽게 개발하고 구성할 수 있습니다. 이 릴리스에는 코어 장치의 로컬 배포 및 구성 요소를 관리하는 IPC 작업을 사용하기 위한 구성 요소에 대한 지원도 포함됩니다.	2022년 6월 27일
클라이언트 장치 구성 요소 업데이트	클라이언트 장치 인증 v2.1.0, MQTT 브로커 (모켓) v2.1.0, MQTT브리지 v2.1.1 및 IP 탐지기 v2.1.2를 사용할 수 있습니다. 이 릴리스는 인증서 로테이션을 개선하고 MQTT Broker 성능을 개선하며 이러한 구성 요소가 컨피그레이션 재설정 업데이트를 처리하는 방식과 관련된 문제를 수정합니다.	2022년 6월 14일
AWS IoT Device Tester v4.5.3은 그린그래스 핵 버전 2.5.6을 지원합니다.	AWS IoT Greengrass V2용 IDT 버전 4.5.3은 이제 그린그래스 핵 버전 2.5.6을 지원합니다.	2022년 6월 1일

AWS IoT Greengrass 코어 v2.5.6 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.5.6을 제공하고 제공된 구성 요소를 업데이트합니다. AWS여기에는 ECC 키가 있는 하드웨어 보안 모듈에 대한 지원이 포함됩니다. 또한 기타 버그 수정 및 개선 사항도 포함됩니다.	2022년 5월 31일
AWS IoT 플릿 프로비저닝 플러그인 v1.1.0 출시	AWS IoT 플릿 프로비저닝 플러그인 v1.1.0을 사용할 수 있습니다. 이번 릴리스에는 Windows 디바이스에서 플러그인을 구성할 때 추가 파일 경로 형식에 대한 지원이 추가되었습니다.	2022년 5월 12일
새로운 Lambda 런타임 출시	파이썬 3.9, 자바 11, NodeJS 14와 같은 새로운 Lambda 런타임에 대한 지원이 추가되었습니다.	2022년 5월 10일
구성 요소 업데이트를 연기하는 Greengrass 구성 요소 개발	배포에서 구성 요소 업데이트를 연기하는 Greengrass 구성 요소를 개발하는 방법을 배우기 위해 따라할 수 있는 자습서가 추가되었습니다. 예를 들어 기기의 배터리 잔량이 부족하거나 중단할 수 없는 프로세스를 실행하는 동안에는 업데이트를 연기하고 싶을 수 있습니다.	2022년 5월 4일

CloudWatch 메트릭 v3.1.0 및 v3.1.0이 출시되었습니다. AWS IoT Device Defender	CloudWatch 메트릭 구성 요소 v3.1.0 및 구성 요소 v3.1.0을 사용할 수 있습니다. AWS IoT Device Defender 이 릴리스는 HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443을 통한 연결 또는 네트워크 프록시를 통한 연결 및 HTTPS 프록시를 신뢰하도록 코어 장치 활성화를 참조하십시오.	2022년 4월 27일
에서 마이그레이션하십시오. AWS IoT Greengrass Version 1	에서 AWS IoT Greengrass V1 마이그레이션할 때 따를 수 있는 가이드가 추가되었습니다 AWS IoT Greengrass V2.	2022년 4월 26일
AWS IoT Device Tester 지원되는 버전에 GGV2Q v2.3.1이 업데이트된 v4.5.3과 GGV2Q v2.3.0이 포함된 IDT v4.5.1이 추가되었습니다.	V2 검증 제품군 (GGV2Q) v2.3.1이 포함된 AWS IoT Greengrass AWS IoT Greengrass V2용 IDT 버전 4.5.3이 그린그래스 뉴클리어스 버전 2.5.5, 2.5.4 및 2.5.3에 대한 지원을 포함하도록 업데이트되었습니다. 이 업데이트에는 V2 검증 제품군 (GGV2Q) v2.3.0이 포함된 IDT 4.5.1도 지원 버전으로 포함되어 있습니다. AWS IoT Greengrass AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.3.0이 포함된 IDT 4.5.1은 그린그래스 뉴클리어스 버전 2.5.3을 지원합니다.	2022년 4월 25일

모드버스-RTU 프로토콜 어댑터 v2.1.0 출시	모드버스-RTU 프로토콜 어댑터 컴포넌트 v2.1.0을 사용할 수 있습니다. 이번 릴리스에는 Modbus RTU 장치와의 직렬 통신을 구성하기 위해 지정할 수 있는 새 매개변수가 추가되었습니다.	2022년 4월 20일
CloudWatch 메트릭스 v2.1.0, Firehose v2.1.0, 아마존 SNS v2.1.0 출시	CloudWatch 메트릭 구성 요소 v2.1.0, Firehose 구성 요소 v2.1.0 및 Amazon SNS 구성 요소 v2.1.0을 사용할 수 있습니다. 이번 릴리스에는 HTTPS 네트워크 프록시 구성에 대한 지원이 추가되었습니다. 자세한 내용은 포트 443을 통한 연결 또는 네트워크 프록시를 통한 연결 및 HTTPS 프록시를 신뢰하도록 코어 장치 활성화를 참조하십시오.	2022년 4월 19일
AWS IoT Device Tester GGV2Q v2.3.1이 포함된 v4.5.3이 출시되었습니다.	V2용 IDT 버전 4.5.3을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.3.1이 포함되어 있으며 그린그래스 핵 버전 2.5.5를 지원합니다.	2022년 4월 15일

AWS IoT Greengrass 코어 v2.5.5 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.5를 제공하고 제공된 구성 요소를 업데이트합니다 AWS. 영어 이외의 표시 언어를 사용하는 Windows 장치에 대한 지원이 추가되었습니다. 또한 특정 시나리오에서 코어 디바이스가 프로비저닝한 후 AWS IoT Greengrass 클라우드 서비스에 상태를 보고하지 않는 문제도 수정됩니다.	2022년 4월 6일
AWS IoT Greengrass Core v2.5.4 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.5.4를 제공하고 제공된 구성 요소를 업데이트합니다. AWS여기에는 버그 수정 및 개선 사항이 포함됩니다.	2022년 3월 23일
AWS IoT Device Tester 프로그래밍 방식으로 다운로드	프로그래밍 방식으로 IDT를 다운로드하는 방법에 대한 AWS IoT Greengrass V2 정보가 추가되었습니다.	2022년 3월 15일
그린그래스 개발 키트 CLI v1.1.0	그린그래스 개발 키트 CLI의 버전 1.1.0을 사용할 수 있습니다. 이 버전은 및 명령에 새 인수를 추가합니다. `component init` `component publish` 또한 이 버전은 구성 요소가 빌드되지 않은 경우 구성 요소를 빌드하도록 `component publish` 명령을 업데이트합니다.	2022년 2월 24일

섀도우 매니저 v2.1.0 출시	섀도우 매니저 컴포넌트 v2.1.0을 사용할 수 있습니다. 이번 릴리스에는 컴포넌트가 섀도우를 동기화하는 간격을 구성하는 옵션이 추가되었습니다. AWS IoT Core예를 들어 간격을 더 길게 지정하여 대역폭 사용량과 요금을 줄일 수 있습니다.	2022년 2월 3일
코어 소프트웨어 v2.5.3용 도커 파일 및 도커 이미지 AWS IoT Greengrass	이제 코어 소프트웨어 v2.5.3용 도커파일 및 도커 이미지를 사용할 수 있습니다. AWS IoT Greengrass	2022년 1월 12일
AWS IoT Device Tester GGV2Q v2.3.0이 출시된 v4.5.1	V2용 IDT 버전 4.5.1을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.3.0이 포함되며, 하드웨어 보안 모듈 (HSM) 을 사용하여 코어 소프트웨어에서 사용하는 개인 키와 인증서를 저장하는 Linux 기반 장치의 검증 및 검증을 지원합니다. AWS IoT Greengrass	2022년 1월 11일
AWS IoT Greengrass 코어 v2.5.3 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.5.3을 제공하고 제공된 구성 요소를 업데이트합니다. AWS 여기에는 하드웨어 보안 모듈 (HSM) 에 안전하게 저장하는 개인 키와 인증서를 사용하도록 AWS IoT Greengrass Core 소프트웨어를 구성할 수 있는 지원이 포함됩니다.	2022년 1월 6일

코어 소프트웨어 v2.5.2용 도커 파일 및 도커 이미지 AWS IoT Greengrass	이제 코어 소프트웨어 v2.5.2용 도커파일 및 도커 이미지를 사용할 수 있습니다. AWS IoT Greengrass	2021년 12월 20일
AWS IoT Device Tester GGV2Q v2.2.1과 함께 v4.4.1이 출시되었습니다.	V2용 IDT 버전 4.4.1을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.2.1이 포함되어 있으며 장치 검증을 위한 Greengrass nucleus 버전 2.5.2를 지원합니다.	2021년 12월 12일
Amazon Lookout for Vision을 사용하여 기계 학습 추론 수행	Greengrass 코어 디바이스에서 Lookout for Vision을 사용하여 기계 학습 추론을 수행하는 방법에 대한 정보가 추가되었습니다. Lookout for Vision은 컴퓨터 비전을 사용하여 산업용 제품의 시각적 결함을 찾아냅니다.	2021년 12월 8일
AWS IoT Device Tester v4.4.1과 GGV2Q v2.2.0이 출시되었습니다.	V2용 IDT 버전 4.4.1을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.2.0이 포함되어 있으며 장치 검증을 위한 Greengrass nucleus 버전 2.5.2를 지원합니다.	2021년 12월 6일

AWS IoT Greengrass 코어 v2.5.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.2를 제공하고 제공된 구성 요소를 업데이트합니다. AWS Greengrass 핵 업데이트 이후 발생하는 Windows 서비스 관련 문제를 수정합니다. 또한 Windows 디바이스의 AWS IoT Device Defender 구성 요소에 대한 지원도 포함됩니다.	2021년 12월 3일
Kinesis Video Streams 구성 요소를 위한 새로운 에지 커넥터	Kinesis Video Streams 구성 요소용 에지 커넥터 버전 1.0.0을 사용할 수 있습니다. 이 AWS제공자는 로컬 카메라에서 비디오 피드를 읽고 Kinesis Video Streams에 스트림을 게시합니다. 이 구성 요소는 와 AWS IoT TwinMaker 통합되어 Grafana 대시보드에서 비디오 스트림 및 기타 데이터를 보고 관리할 수 있습니다.	2021년 11월 30일
다음을 사용하여 Greengrass 코어 장치를 관리하십시오. AWS Systems Manager	Greengrass 코어 디바이스를 관리하는 방법에 대한 정보가 추가되었습니다. AWS Systems Manager Systems Manager는 운영 데이터를 보고, 운영 작업을 자동화하고, 보안 및 규정 준수를 유지할 수 있는 AWS 서비스입니다.	2021년 11월 29일

그린그래스 개발 키트 CLI	사용자 지정 Greengrass 구성 요소를 개발하는 데 도움이 되도록 로컬 AWS IoT Greengrass 개발 컴퓨터에 다운로드할 수 있는 도구인 개발 키트 명령줄 인터페이스 (GDK CLI) 에 대한 정보가 추가되었습니다. GDK CLI를 사용하여 사용자 지정 구성 요소를 만들고 빌드하고 게시할 수 있습니다.	2021년 11월 29일
커뮤니티에서 제공하는 Greengrass 구성 요소	Greengrass 커뮤니티에서 개발한 Greengrass 구성 요소의 색인인 Greengrass 소프트웨어 카탈로그에 대한 정보가 추가되었습니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다.	2021년 11월 29일
AWS IoT Greengrass 코어 v2.5.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.1을 제공하고 제공된 구성 요소를 업데이트합니다. AWS 여기에는 Windows 장치에서의 32비트 자바에 대한 지원이 포함됩니다. 또한 새 사물 그룹 제거 동작과 Windows 디바이스의 시스템 환경 변수 로드 관련 문제도 수정됩니다.	2021년 11월 23일

AWS IoT Device Tester v4.4.0과 함께 GGV2Q v2.1.0이 출시되었습니다.	V2용 IDT 버전 4.4.0을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.1.0이 포함되어 있으며 Greengrass nucleus 버전 2.5.0을 실행하는 Windows 기반 그린그래스 장치의 검증을 지원합니다.	2021년 11월 19일
AWS IoT Greengrass 코어 v2.5.0 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.5.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS여기에는 Windows 장치에서 AWS IoT Greengrass Core 소프트웨어를 실행하기 위한 지원이 포함됩니다. 또한 사물 그룹 제거 동작을 변경하고 HTTPS 프록시에 대한 지원을 추가합니다.	2021년 11월 12일
SageMaker 엣지 매니저 v1.1.0 출시	아마존 SageMaker 엣지 매니저 구성 요소 v1.1.0을 사용할 수 있습니다. 이번 릴리스에는 Amazon Linux 2를 실행하는 Greengrass 코어 디바이스에 대한 지원이 추가되었으며, 디바이스의 캡처 데이터 폴더 위치를 지정하는 새 구성 파라미터가 추가되었습니다.	2021년 11월 3일

교차 서비스 혼동된 대리자 예방 업데이트	AWS IoT Greengrass V2 IAM 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용할 수 있도록 지원하여 혼동되는 부정 문제를 방지합니다.	2021년 11월 1일
클라이언트 디바이스 구성 요소 업데이트	클라이언트 장치 인증 v2.0.3, IP 감지기 v2.1.0, MQTT 브리지 v2.1.0 및 MQTT 브로커 (모켓) v2.0.2를 사용할 수 있습니다. 이 릴리스에는 기본이 아닌 MQTT 브로커 포트에 대한 전체 지원이 추가되었으며 기타 버그 수정 및 개선 사항이 포함되어 있습니다.	2021년 10월 28일
섀도우 매니저 v2.0.4 출시	섀도우 매니저 컴포넌트 v2.0.4를 사용할 수 있습니다. 이번 릴리스에서는 섀도우 관리자가 이전에 삭제된 섀도우의 새로 생성된 버전을 삭제하던 문제를 수정했습니다. 이번 릴리스부터 `DeleteThingShadow` IPC 작업을 통해 섀도우 버전이 증가합니다.	2021년 10월 20일
로그 관리자 v2.2.0이 출시되었습니다.	로그 관리자 구성 요소 v2.2.0을 사용할 수 있습니다. 이제 로그 관리자가 구성 맵을 사용하여 구성 요소 로그 구성을 제공할 수 있습니다.	2021년 10월 20일

람다 매니저 v2.1.4 출시	Lambda 관리자 구성 요소 v2.1.4를 사용할 수 있습니다. 이 릴리스에서는 NodeJS 런타임을 사용하는 Lambda 함수가 메시지를 하나만 처리하던 문제를 수정했습니다.	2021년 10월 20일
Docker 컨테이너 구성 요소에서 프로세스 간 통신, AWS 자격 증명 및 스트림 관리자를 사용하십시오.	사용자 지정 Docker 컨테이너 구성 요소에 IPC (프로세스 간 통신), AWS 자격 증명 및 스트림 관리자를 사용하는 방법에 대한 정보가 추가되었습니다.	2021년 10월 19일
새로운 핵 원격 측정 이미터 구성 요소	핵 원격 측정 이미터 구성 요소 버전 1.0.0을 사용할 수 있습니다. AWS제공된 이 구성 요소는 시스템 상태 원격 측정 데이터를 수집하여 로컬 주제 및 MQTT 주제에 지속적으로 게시합니다. AWS IoT Core	2021년 9월 30일
프록시 또는 방화벽을 통한 장치 트래픽 허용	Greengrass 코어 디바이스가 사용하는 엔드포인트 및 포트에 대한 정보가 추가되어 보안 조치로 트래픽을 제한할 수 있습니다.	2021년 9월 16일
AWS IoT Device Tester GGV2Q v2.0.1이 포함된 v4.2.0이 출시되었습니다.	AWS IoT Greengrass V2용 IDT 버전 4.2.0은 V2 검증 제품군 (GGV2Q) AWS IoT Greengrass v2.0.1로 업데이트되었습니다. 이 릴리스는 장치 검증을 위한 Greengrass nucleus 버전 2.4.0을 지원합니다.	2021년 8월 31일

머신 러닝 설치 프로그램 구성 요소가 업데이트되었습니다.	DLR 설치 프로그램 구성 요소 v1.6.5 및 TensorFlow Lite 설치 프로그램 구성 요소 v2.5.4를 사용할 수 있습니다. 이러한 구성 요소 버전에는 기본 설치 스크립트를 사용하지 않도록 설정하는 데 사용할 수 있는 새 `UseInstaller` 구성 매개 변수가 포함되어 있습니다.	2021년 8월 30일
다음과 같은 Linux 지원이 내장되어 있습니다. AWS IoT Greengrass	의 BitBake AWS IoT Greengrass V2 레시피는 의 `meta-aws` 프로젝트에서 확인할 수 GitHub 있습니다. 이 레시피를 사용하여 Yocto Project를 사용하여 사용자 지정 Linux 기반 운영 체제를 구축할 수 있습니다.	2021년 8월 20일
코드 무결성	Greengrass 코어 디바이스가 에서 다운로드하는 소프트웨어의 무결성을 AWS IoT Greengrass V2 확인하는 방법에 대한 정보가 추가되었습니다. AWS 클라우드	2021년 8월 19일
VPC 엔드포인트(AWS PrivateLink)	AWS IoT Greengrass 이제 컨트롤 플레인의 인터페이스 VPC 엔드포인트 (AWS PrivateLink) 를 AWS IoT Greengrass 지원합니다. VPC와 AWS IoT Greengrass 컨트롤 플레인 사이에 프라이빗 연결을 설정할 수 있습니다.	2021년 8월 16일

스트림 매니저 v2.0.12가 출시되었습니다.	이제 스트림 매니저 v2.0.12를 사용할 수 있습니다. 이 릴리스에서는 스트림 관리자 구성 요소 버전 2.0.7에서 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 수 없었던 문제가 수정되었습니다.	2021년 8월 10일
코어 소프트웨어 v2.4.0용 도커 파일 및 도커 이미지 AWS IoT Greengrass	이제 코어 소프트웨어 v2.4.0용 도커파일 및 도커 이미지를 사용할 수 있습니다. AWS IoT Greengrass	2021년 8월 9일
AWS IoT Greengrass 코어 v2.4.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.4.0을 제공하고 제공된 구성 요소를 업데이트합니다 AWS. 여기에는 구성 요소 시스템 리소스 제한, 구성 요소를 일시 중지 및 재개하기 위한 IPC 작업, 프로비저닝 플러그인에 대한 지원이 포함됩니다.	2021년 8월 3일
새 AWS IoT SiteWise 구성 요소	IoT AWS SiteWise OPC-UA 컬렉터, IoT SiteWise 퍼블리셔, IoT 프로세서에 대해 AWS IoT SiteWise제공되는 다음과 같은 구성 요소가 추가되었습니다. SiteWise	2021년 7월 29일

AWS IoT Device Tester v4.2.0과 함께 GGV2Q v2.0.0이 출시되었습니다.	V2용 IDT 버전 4.2.0을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.0.0이 포함되며 Docker 구성 요소, 기계 학습 및 스트림 관리자에 대한 선택적 검증 테스트에 대한 지원이 포함됩니다.	2021년 7월 14일
AWS IoT Greengrass C++ v2용 코어 IPC 라이브러리는 C++ v2에서 사용할 수 있습니다. AWS IoT Device SDK	C++ AWS IoT Device SDK v2용 버전 1.13.0은 AWS IoT Greengrass 코어 IPC를 지원하므로 Core 소프트웨어와 상호 작용하는 C++로 구성 요소를 개발할 수 있습니다. AWS IoT Greengrass	2021년 7월 14일
SageMaker 엣지 매니저 구성 요소 v1.0.2 출시	Amazon SageMaker Edge Manager 구성 요소 v1.0.2를 사용할 수 있습니다. 이번 릴리스는 구성 요소 수명 주기의 설치 스크립트를 업데이트합니다. 이제 이 구성 요소를 배포하기 전에 코어 기기에 Python 3.6 이상 버전 (사용 중인 Python 버전 포함`pip`) 이 설치되어 있어야 합니다.	2021년 7월 12일
AWS IoT Greengrass V2에 AWS IoT Device Tester 대한 지원 업데이트	AWS IoT Greengrass V2용 IDT 버전 4.1.0은 이제 장치 검증을 위한 그린그래스 핵 버전 2.3.0을 사용할 수 있습니다.	2021년 7월 8일

코어 소프트웨어 v2.3.0용 도커 파일 및 도커 이미지 AWS IoT Greengrass	이제 코어 소프트웨어 v2.3.0용 도커파일 및 도커 이미지를 사용할 수 있습니다. AWS IoT Greengrass	2021년 7월 7일
AWS 관리형 정책	의 AWS 관리형 정책에 대한 정보가 추가되었습니다 AWS IoT Greengrass.	2021년 7월 2일
새로운 권장 JVM 옵션	AWS IoT Greengrass Core 소프트웨어의 메모리 할당을 제어하기 위한 권장 JVM 옵션에 대한 정보가 추가되었습니다.	2021년 6월 30일
AWS IoT Greengrass Core v2.3.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.3.0을 제공하고 제공된 구성 요소를 업데이트합니다 AWS. 배포 시 대규모 구성 요소 구성 문서에 대한 지원이 포함됩니다.	2021년 6월 29일
코어 소프트웨어 v2.2.0용 도커 파일 및 도커 이미지 AWS IoT Greengrass	이제 코어 소프트웨어 v2.2.0용 도커파일 및 도커 이미지를 사용할 수 있습니다. AWS IoT Greengrass	2021년 6월 28일
AWS IoT Device Tester v4.1.0과 GGV2Q v1.1.1이 출시되었습니다.	V2용 IDT 버전 4.1.0을 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v1.1.1이 포함되어 있으며 장치 검증을 위해 그린그래스 핵 v2.2.0, v2.1.0 및 v2.0.5를 사용할 수 있습니다.	2021년 6월 18일

AWS IoT Greengrass 코어 v2.2.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.2.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS 여기에는 클라이언트 장치에 대한 지원을 추가하고 로컬 섀도우 서비스를 추가하기 위해 배포할 수 있는 구성 요소가 포함됩니다.	2021년 6월 18일
람다 런처 v2.0.6 출시	Lambda 런처 구성 요소 버전 2.0.6을 사용할 수 있습니다. 이 버전에는 성능 개선 및 버그 수정이 포함되어 있습니다.	2021년 6월 13일
새 SageMaker Edge Manager 구성 요소가 출시되었습니다.	Amazon SageMaker Edge Manager 구성 요소 버전 1.0.0은 다음에서 사용할 수 있습니다. AWS IoT Greengrass이 구성 요소는 Greengrass 코어 디바이스에 SageMaker Edge Manager 에이전트 바이너리를 설치합니다.	2021년 6월 10일
구성 요소 유형	에 구성 요소 유형에 대한 정보가 추가되었습니다 AWS IoT Greengrass. 구성 요소 유형은 AWS IoT Greengrass Core 소프트웨어가 구성 요소를 실행하는 방법을 지정합니다.	2021년 6월 3일

AWS IoT Device Tester v4.0.2와 함께 GGV2Q v1.1.0이 출시되었습니다.	V2용 IDT 버전 4.0.2를 사용할 수 있습니다. AWS IoT Greengrass 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v1.1.0이 포함되어 있으며 장치 검증을 위해 그린그래스 핵 v2.1.0과 그린그래스 CLI v2.1.0을 함께 사용할 수 있습니다. 여기에는 MQTT 및 Lambda에 대한 새로운 필수 테스트 그룹과 기타 사소한 버그 수정 및 개선 사항도 포함됩니다.	2021년 5월 5일
코어 소프트웨어 v2.1.0용 도커 파일 및 도커 이미지 AWS IoT Greengrass	이제 코어 소프트웨어 v2.1.0용 도커파일 및 도커 이미지를 사용할 수 있습니다. AWS IoT Greengrass Docker 이미지를 사용하면 Amazon Linux 2를 기본 운영 체제로 사용하는 Docker 컨테이너에서 AWS IoT Greengrass Core 소프트웨어를 실행할 수 있습니다.	2021년 4월 27일
AWS IoT Greengrass 코어 v2.1.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.1.0을 제공하고 제공된 구성 요소를 업데이트합니다. AWS 여기에는 프라이빗 Amazon ECR 리포지토리에서 Docker 이미지를 다운로드하는 데 사용할 수 있는 새로운 구성 요소와 Lite를 사용하여 기계 학습 추론을 수행하는 새 샘플 구성 요소가 포함되어 있습니다. TensorFlow	2021년 4월 26일

Secrets Manager를 사용하는 예제 컴포넌트	코어 디바이스에 배포한 AWS Secrets Manager 비밀의 값을 인쇄하는 예제 구성 요소를 추가했습니다.	2021년 4월 8일
Greengrass 코어 디바이스에 대한 최소 AWS IoT 정책	코어 디바이스에서 기본 Greengrass 기능을 지원하는 데 필요한 최소 권한 세트에 대한 정보가 추가되었습니다.	2021년 4월 2일
IPC 이벤트 스트림 구독	프로세스 간 통신 (IPC) 작업을 사용하여 Greengrass 코어 디바이스에서 이벤트 스트림을 구독하는 방법에 대한 정보가 추가되었습니다.	2021년 4월 1일
에 AWS IoT Device Tester 대한 지원 업데이트 AWS IoT Greengrass	AWS IoT Greengrass V2용 IDT 버전 4.0.1은 이제 장치 검증을 위해 그린그래스 핵 버전 2.0.5와 그린그래스 CLI 버전 2.0.5를 함께 사용할 수 있습니다.	2021년 3월 17일
스트림 관리자를 사용하는 사용자 지정 구성 요소 생성	데이터 스트림을 관리하는 애플리케이션을 개발하기 위해 구성 요소 레시피와 아티팩트를 구성하는 방법에 대한 정보가 추가되었습니다.	2021년 3월 9일
AWS IoT Greengrass Core v2.0.5 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.0.5를 제공하고 제공된 구성 요소를 업데이트합니다. AWS 네트워크 프록시 지원 문제 및 AWS 중국 지역의 Greengrass 데이터 플레인 엔드포인트 관련 문제를 수정합니다.	2021년 3월 9일

구성 요소 환경 변수 참조	AWS IoT Greengrass Core 소프트웨어가 구성 요소에 설정하는 환경 변수에 대한 정보가 추가되었습니다. 이러한 환경 변수를 사용하여 사물 이름 및 Greengrass AWS 리전 nucleus 버전을 가져올 수 있습니다.	2021년 2월 23일
수동 설치	필요한 AWS 리소스를 수동으로 생성하거나 방화벽 또는 네트워크 프록시 뒤에 설치하는 방법에 대한 정보가 추가되었습니다. 수동 설치를 사용하면 필수 AWS IoT 및 IAM 리소스를 생성하므로 설치 관리자에게 에서 리소스를 AWS 계정생성할 권한을 부여할 필요가 없습니다. 포트 443이나 네트워크 프록시를 통해 연결하도록 디바이스를 구성할 수도 있습니다.	2021년 2월 17일
AWS IoT Greengrass Python AWS IoT Device SDK v2용 코어 IPC 라이브러리 업데이트	AWS IoT Device SDK Python v2용 버전 1.5.4는 AWS IoT Greengrass 코어 IPC 서비스에 연결하는 데 필요한 단계를 단순화합니다.	2021년 2월 11일
에 AWS IoT Device Tester 대한 지원 업데이트 AWS IoT Greengrass	AWS IoT Greengrass V2용 IDT 버전 4.0.1은 이제 장치 검증을 위해 그린그래스 핵 버전 2.0.4와 그린그래스 CLI 버전 2.0.4를 함께 사용할 수 있습니다.	2021년 2월 5일

Lambda 함수를 가져오기 위한 새로운 자습서	Lambda 함수를 Greengrass 코어 디바이스에서 실행되는 구성 요소로 가져오기 위한 새로운 콘솔 기반 자습서가 추가되었습니다.	2021년 2월 5일
AWS IoT Greengrass Core v2.0.4 소프트웨어 업데이트	이 릴리스에서는 Greengrass 핵 구성 요소 버전 2.0.4를 제공합니다. 여기에는 포트 443을 통한 HTTPS 통신을 구성하고 버그를 수정하기 위한 새 `greengrassDataPlanePort` 매개변수가 포함되어 있습니다. 이제 최소 IAM 정책에 따라 AWS IoT Greengrass Core 소프트웨어 설치 프로그램을 실행할 `sts:GetCallerIdentity` 때 `iam:GetPolicy` 및 가 필요합니다. `--provision true`	2021년 2월 4일
새로운 보안 터널링 구성 요소가 출시되었습니다.	에서 보안 터널링 구성 요소 버전 1.0.0을 사용할 수 있습니다. AWS IoT Greengrass AWS제공된 이 구성 요소는 AWS IoT 보안 터널링을 사용하여 제한된 방화벽 뒤에 있는 Greengrass 코어 장치와 안전한 양방향 통신을 설정합니다.	2021년 1월 21일

AWS IoT Device TesterAWS IoT Greengrass v4.0.1용 출시	V2용 AWS IoT Greengrass IDT 버전 4.0.1을 사용할 수 있습니다. 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 22일
의 초기 릴리스 AWS IoT Greengrass Version 2	AWS IoT Greengrass V2 의 새로운 메이저 버전 AWS IoT Greengrass릴리스입니다. 이 버전에는 모듈식 소프트웨어 구성 요소 및 지속적인 배포와 같은 여러 기능이 추가되었습니다. 이러한 기능을 통해 에지 애플리케이션을 더 쉽게 개발하고 관리할 수 있습니다.	2020년 12월 15일

AWS 용어집

최신 AWS 용어는 참조의 AWS 용어집을 참조하십시오.AWS 용어집

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