



개발자 안내서, 버전 2

AWS IoT Greengrass



AWS IoT Greengrass: 개발자 안내서, 버전 2

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

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

AWS IoT Greengrass란 무엇인가요?	1
새로운 기능	1
처음 사용하는 사용자의 경우	2
기존 사용자의 경우	2
AWS IoT Greengrass 작동 방식	2
주요 개념	3
의 기능 AWS IoT Greengrass	5
Greengrass 기능 호환성	7
AWS IoT Greengrass nucleus 런타임 선택	15
Greengrass nucleus	15
Greengrass nucleus lite	15
버전 2의 새 기능	19
AWS IoT Greengrass 코어 v2.16.1 소프트웨어 업데이트	21
퍼블릭 구성 요소 업데이트	21
AWS IoT Greengrass 코어 v2.16.0 소프트웨어 업데이트	22
퍼블릭 구성 요소 업데이트	23
AWS IoT Greengrass 코어 v2.15.1 소프트웨어 업데이트	24
퍼블릭 구성 요소 업데이트	25
AWS IoT Greengrass 코어 v2.15.0 소프트웨어 업데이트	25
퍼블릭 구성 요소 업데이트	26
AWS IoT Greengrass 코어 v2.14.3 소프트웨어 업데이트	28
퍼블릭 구성 요소 업데이트	28
AWS IoT Greengrass 코어 v2.14.2 소프트웨어 업데이트	29
퍼블릭 구성 요소 업데이트	30
AWS IoT Greengrass 코어 v2.14.1 소프트웨어 업데이트	31
퍼블릭 구성 요소 업데이트	31
AWS IoT Greengrass 코어 v2.14.0 소프트웨어 업데이트	32
퍼블릭 구성 요소 업데이트	34
AWS IoT Greengrass 코어 v2.13.0 소프트웨어 업데이트	36
퍼블릭 구성 요소 업데이트	36
AWS IoT Greengrass 코어 v2.12.6 소프트웨어 업데이트	38
퍼블릭 구성 요소 업데이트	38
AWS IoT Greengrass 코어 v2.12.5 소프트웨어 업데이트	39
퍼블릭 구성 요소 업데이트	39

AWS IoT Greengrass 코어 v2.12.4 소프트웨어 업데이트	40
퍼블릭 구성 요소 업데이트	41
AWS IoT Greengrass 코어 v2.12.3 소프트웨어 업데이트	41
퍼블릭 구성 요소 업데이트	42
AWS IoT Greengrass 코어 v2.12.2 소프트웨어 업데이트	43
퍼블릭 구성 요소 업데이트	44
AWS IoT Greengrass 코어 v2.12.1 소프트웨어 업데이트	45
퍼블릭 구성 요소 업데이트	45
AWS IoT Greengrass 코어 v2.12.0 소프트웨어 업데이트	47
퍼블릭 구성 요소 업데이트	47
AWS IoT Greengrass 코어 v2.11.3 소프트웨어 업데이트	48
퍼블릭 구성 요소 업데이트	48
AWS IoT Greengrass 코어 v2.11.2 소프트웨어 업데이트	50
퍼블릭 구성 요소 업데이트	50
AWS IoT Greengrass 코어 v2.11.1 소프트웨어 업데이트	51
퍼블릭 구성 요소 업데이트	51
AWS IoT Greengrass 코어 v2.11.0 소프트웨어 업데이트	52
퍼블릭 구성 요소 업데이트	52
AWS IoT Greengrass 코어 v2.10.3 소프트웨어 업데이트	54
퍼블릭 구성 요소 업데이트	54
AWS IoT Greengrass 코어 v2.10.2 소프트웨어 업데이트	55
퍼블릭 구성 요소 업데이트	55
AWS IoT Greengrass 코어 v2.10.1 소프트웨어 업데이트	57
퍼블릭 구성 요소 업데이트	57
AWS IoT Greengrass 코어 v2.10.0 소프트웨어 업데이트	58
퍼블릭 구성 요소 업데이트	59
AWS IoT Greengrass 코어 v2.9.6 소프트웨어 업데이트	60
퍼블릭 구성 요소 업데이트	60
AWS IoT Greengrass 코어 v2.9.5 소프트웨어 업데이트	61
퍼블릭 구성 요소 업데이트	61
AWS IoT Greengrass 코어 v2.9.4 소프트웨어 업데이트	62
퍼블릭 구성 요소 업데이트	63
AWS IoT Greengrass 코어 v2.9.3 소프트웨어 업데이트	64
퍼블릭 구성 요소 업데이트	64
AWS IoT Greengrass 코어 v2.9.2 소프트웨어 업데이트	65
퍼블릭 구성 요소 업데이트	65

AWS IoT Greengrass 코어 v2.9.1 소프트웨어 업데이트	66
퍼블릭 구성 요소 업데이트	66
AWS IoT Greengrass 코어 v2.9.0 소프트웨어 업데이트	67
퍼블릭 구성 요소 업데이트	68
AWS IoT Greengrass 코어 v2.8.1 소프트웨어 업데이트	70
퍼블릭 구성 요소 업데이트	70
AWS IoT Greengrass 코어 v2.8.0 소프트웨어 업데이트	71
퍼블릭 구성 요소 업데이트	72
AWS IoT Greengrass 코어 v2.7.0 소프트웨어 업데이트	73
퍼블릭 구성 요소 업데이트	74
AWS IoT Greengrass 코어 v2.6.0 소프트웨어 업데이트	75
퍼블릭 구성 요소 업데이트	76
AWS IoT Greengrass 코어 v2.5.6 소프트웨어 업데이트	80
퍼블릭 구성 요소 업데이트	81
AWS IoT Greengrass 코어 v2.5.5 소프트웨어 업데이트	82
퍼블릭 구성 요소 업데이트	82
AWS IoT Greengrass 코어 v2.5.4 소프트웨어 업데이트	83
퍼블릭 구성 요소 업데이트	83
AWS IoT Greengrass 코어 v2.5.3 소프트웨어 업데이트	84
퍼블릭 구성 요소 업데이트	85
AWS IoT Greengrass 코어 v2.5.2 소프트웨어 업데이트	86
퍼블릭 구성 요소 업데이트	86
AWS IoT Greengrass 코어 v2.5.1 소프트웨어 업데이트	87
퍼블릭 구성 요소 업데이트	88
AWS IoT Greengrass 코어 v2.5.0 소프트웨어 업데이트	89
플랫폼 지원 업데이트	90
퍼블릭 구성 요소 업데이트	90
AWS IoT Greengrass 코어 v2.4.0 소프트웨어 업데이트	94
퍼블릭 구성 요소 업데이트	94
AWS IoT Greengrass 코어 v2.3.0 소프트웨어 업데이트	96
퍼블릭 구성 요소 업데이트	97
AWS IoT Greengrass 코어 v2.2.0 소프트웨어 업데이트	98
퍼블릭 구성 요소 업데이트	99
AWS IoT Greengrass 코어 v2.1.0 소프트웨어 업데이트	101
플랫폼 지원 업데이트	102
퍼블릭 구성 요소 업데이트	103

AWS IoT Greengrass 코어 v2.0.5 소프트웨어 업데이트	109
퍼블릭 구성 요소 업데이트	109
AWS IoT Greengrass 코어 v2.0.4 소프트웨어 업데이트	110
퍼블릭 구성 요소 업데이트	110
버전1에서 마이그레이션	113
마이그레이션 개요	113
V1과 V2 간의 차이점	115
런타임 선택	124
이벤트 소스 호환성 매트릭스	124
런타임 선택 결정 흐름	125
다음 단계	126
테스트 디바이스 설정	126
테스트 디바이스 설정(Greengrass nucleus)	127
테스트 디바이스 설정(Greengrass nucleus lite)	132
V1 코어 디바이스를 V2로 업그레이드	201
1단계: AWS IoT Greengrass 코어 소프트웨어 v2.x 설치	201
2단계: 코어 디바이스에 Greengrass V2 구성 요소 배포	205
시작하기	207
사전 조건	208
1단계: AWS 계정 설정	209
에 가입 AWS 계정	209
관리자 액세스 권한이 있는 사용자 생성	210
2단계: 환경 설정	211
3단계: AWS IoT Greengrass 코어 소프트웨어 설치	217
AWS IoT Greengrass 코어 소프트웨어 설치(콘솔)	218
AWS IoT Greengrass 코어 소프트웨어 설치(CLI)	224
Greengrass 소프트웨어 실행(Linux)	229
디바이스에서 Greengrass CLI 설치 확인	230
4단계: 디바이스에서 구성 요소 개발 및 테스트	231
5단계: AWS IoT Greengrass 서비스에서 구성 요소 생성	243
6단계: 구성 요소 배포	254
다음 단계	259
Greengrass 코어 디바이스 설정	260
지원하는 플랫폼	260
장치 요구 사항	260
Lambda 함수 요구 사항	261

설정 AWS 계정	262
AWS IoT Greengrass 코어 소프트웨어 설치	264
자동 프로비저닝을 사용한 설치	266
수동 프로비저닝을 사용하여 설치	281
플릿 프로비저닝을 사용하여 설치	318
사용자 지정 프로비저닝으로 설치	362
설치 프로그램 인수	379
AWS IoT Greengrass 코어 소프트웨어 실행	383
AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 실행되는지 확인	384
AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 실행	386
시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어 실행	386
Docker AWS IoT Greengrass 에서 실행	387
지원되는 플랫폼 및 요구 사항	387
소프트웨어 다운로드	388
AWS 리소스를 프로비저닝하는 방법 선택	388
Dockerfile에서 AWS IoT Greengrass 이미지 빌드	389
자동 프로비저닝을 사용하여 Docker AWS IoT Greengrass 에서 실행	395
수동 프로비저닝을 사용하여 Docker AWS IoT Greengrass 에서 실행	403
Docker 컨테이너 AWS IoT Greengrass 의 문제 해결	421
AWS IoT Greengrass 코어 소프트웨어 구성	424
Greengrass nucleus 구성 요소 배포	424
Greengrass nucleus를 시스템 서비스로 구성	424
JVM 옵션으로 메모리 할당 제어	428
구성 요소를 실행하는 사용자 구성	430
시스템 리소스 제한 구성	434
포트 443에서 또는 네트워크 프록시를 통해 연결	437
프라이빗 CA에서 서명한 디바이스 인증서 사용	445
MQTT 제한 시간 및 캐시 설정 구성	445
IPv6 네트워크에서 Greengrass Nucleus 구성	445
AWS IoT Greengrass 코어 소프트웨어(OTA) 업데이트	446
요구 사항	447
코어 디바이스에 대한 고려 사항	447
Greengrass nucleus 업데이트 동작	447
OTA 업데이트를 수행합니다.	449
AWS IoT Greengrass 코어 소프트웨어 제거	449
자습서	453

자습서 비디오	453
동영상	453
구성 요소 업데이트가 지연되는 구성 요소 개발	454
사전 조건	455
1단계: Greengrass 개발 키트 CLI 설치	456
2단계: 업데이트가 지연되는 구성 요소 개발	457
3단계: AWS IoT Greengrass 서비스에 구성 요소 게시	466
4단계: 디바이스에서 구성 요소 배포 및 테스트	469
MQTT를 통해 로컬 IoT 디바이스와 상호 작용	474
사전 조건	474
1단계: 코어 디바이스 AWS IoT 정책 검토 및 업데이트	475
2단계: 클라이언트 디바이스 지원 활성화	477
3단계: 클라이언트 디바이스 연결	482
4단계: 클라이언트 디바이스와 통신하는 구성 요소를 개발합니다.	485
5단계: 클라이언트 디바이스 새도와 상호 작용하는 구성 요소를 개발합니다.	492
TPM으로 Greengrass Nucleus 보호	518
사전 조건	518
1단계: TPM2 도구 및 종속성 설치	519
2단계: PKCS#11 저장소 초기화 및 슬롯 생성	519
3단계: 토큰 및 키 생성	521
4단계: 인증서 서명 요청(CSR) 생성	523
5단계: 사물 인증서 생성	524
6단계: 사물 인증서를 TPM에 가져오기	524
7단계: PKCS#11 객체 URL 캡처	524
8단계: TPM2 지원을 사용하여 Greengrass 구성 및 설치	526
9단계: 설치 확인	527
문제 해결	528
다음 단계	528
TPM으로 Greengrass Nucleus Lite 보호	529
사전 조건	529
1단계: NitroTPM 인스턴스 설정	530
2단계: TPM 도구 설치 및 구성	530
3단계: OpenSSL TPM2 공급자 구성	530
4단계: 영구 TPM 키 생성	531
5단계: 인증서 서명 요청(CSR) 생성	532
6단계: CSR에서 인증서 생성	532

7단계: TPM 지원으로 Greengrass Nucleus Lite 설정	533
문제 해결	533
다음 단계	534
SageMaker AI Edge Manager 시작하기	535
사전 조건	536
SageMaker AI Edge Manager에서 설정	538
샘플 구성 요소 생성	539
샘플 이미지 분류 추론 실행	540
샘플 이미지 분류 추론 수행	544
사전 조건	544
1단계: 기본 알림 주제 구독	545
2단계: TensorFlow Lite 이미지 분류 구성 요소 배포	546
3단계: 추론 결과 보기	547
다음 단계	549
카메라의 이미지에 대한 샘플 이미지 분류 추론 수행	550
사전 조건	551
1단계: 디바이스에서 카메라 모듈 구성	552
2단계: 기본 알림 주제에 대한 구독 확인	554
3단계: TensorFlow Lite 이미지 분류 구성 요소 구성 수정 및 배포	554
4단계: 추론 결과 보기	556
다음 단계	557
Components	558
AWS 제공 구성 요소	558
Greengrass nucleus	570
Greengrass nucleus lite	610
클라이언트 디바이스 인증	616
CloudWatch 지표	689
AWS IoT Device Defender	713
디스크 스펠러	730
Docker 애플리케이션 관리자	734
Kinesis Video Streams용 엣지 커넥터	744
Greengrass CLI	752
IP 감지기	765
Firehose	776
Lambda 런처	793
Lambda 관리자	797

Lambda 런타임	806
레거시 구독 라우터	808
로컬 디버그 콘솔	820
로그 관리자	836
기계 학습 구성 요소	882
Modbus-RTU 프로토콜 어댑터	994
MQTT 브리지	1025
MQTT 3.1.1 브로커(Moquette)	1048
MQTT 5 브로커(EMQX)	1055
nucleus 원격 측정 이미터	1072
PKCS#11 공급자	1085
보안 암호 관리자	1093
보안 터널링	1105
새도 관리자	1116
Amazon SNS	1144
스트림 관리자	1161
시스템 로그 전달자	1174
Systems Manager Agent	1180
토큰 교환 서비스	1186
IoT SiteWise OPC UA 수집기	1190
IoT SiteWise OPC UA 데이터 소스 시뮬레이터	1200
IoT SiteWise 게시자	1203
IoT SiteWise 프로세서	1217
게시자 지원 구성 요소	1231
AIShield.Edge	1232
AI EdgeLabs Sensor	1232
Greengrass S3 Ingestor	1233
커뮤니티 구성 요소	1233
Greengrass 개발 도구	1237
Greengrass 개발 키트 CLI	1238
Greengrass 명령줄 인터페이스	1267
Greengrass Testing Framework 사용	1284
구성 요소 개발	1299
구성 요소 수명 주기	1301
구성 요소 유형	1302
구성 요소 생성	1303

로컬 배포를 사용한 구성 요소 테스트	1315
배포할 구성 요소 게시	1317
AWS 서비스와 상호 작용	1323
Docker 컨테이너 실행	1327
레시피 참조	1349
환경 변수	1379
디바이스에 구성 요소 배포	1380
코어 디바이스 배포	1380
플랫폼 종속성 확인	1381
구성 요소 종속성 확인	1381
사물 그룹에서 디바이스 제거	1388
배포	1389
배포 옵션	1389
배포 만들기	1391
하위 배포 생성	1410
배포 수정	1414
배포 취소	1415
배포 상태 확인	1416
로깅 및 모니터링	1420
모니터링 도구	1420
Greengrass 로그 모니터링	1421
파일 시스템 로그 액세스	1422
CloudWatch Logs 액세스	1424
시스템 서비스 로그 액세스	1426
CloudWatch Logs에 로깅 활성화	1427
AWS IoT Greengrass의 로깅 구성	1429
AWS CloudTrail 로그	1431
CloudTrail로 API 직접 호출 로깅	1431
AWS IoT Greengrass V2 CloudTrail의 정보	1431
AWS IoT Greengrass CloudTrail의 데이터 이벤트	1432
AWS IoT Greengrass CloudTrail의 관리 이벤트	1437
AWS IoT Greengrass V2 로그 파일 항목 이해	1437
시스템 상태 원격 측정 데이터 수집	1438
원격 측정 지표	1440
원격 측정 에이전트 설정	1444
EventBridge에서 원격 측정 데이터 구독	1444

배포 및 구성 요소 상태 알림 받기	1452
배포 상태 변경 이벤트	1453
구성 요소 상태 변경 이벤트	1455
EventBridge 규칙 생성을 위한 사전 조건	1457
디바이스 상태 알림 구성(콘솔)	1458
디바이스 상태 알림 구성(CLI)	1459
디바이스 상태 알림 구성(CloudFormation)	1460
다음 사항도 참조하세요.	1460
코어 디바이스 상태 확인	1460
코어 디바이스 상태 확인	1461
코어 디바이스 그룹 상태 확인	1462
코어 디바이스 구성 요소 상태 확인	1462
Lambda 함수 실행	1463
요구 사항	1464
Lambda 함수 수명 주기 구성	1464
Lambda 함수 컨테이너화 구성	1465
Lambda 함수를 구성 요소로 가져오기(콘솔)	1467
1단계: 가져올 Lambda 함수 선택	1468
2단계: Lambda 함수 파라미터 구성	1468
3단계: (선택 사항) Lambda 함수에 지원되는 플랫폼 지정	1470
4단계: (선택 사항) Lambda 함수의 구성 요소 종속성 지정	1471
5단계: (선택 사항) 컨테이너에서 Lambda 함수 실행	1472
6단계: Lambda 함수 구성 요소 생성	1474
Lambda 함수를 구성 요소로 가져오기(CLI)	1474
1단계: Lambda 함수 구성 정의	1474
2단계: Lambda 함수 구성 요소 생성	1494
Greengrass nucleus, 기타 구성 요소 및와 통신 AWS IoT Core	1496
IPC 클라이언트 버전	1497
지원되는 SDK	1497
AWS IoT Greengrass 코어 IPC 서비스에 연결	1498
구성 요소에 IPC 작업을 수행할 수 있는 권한 부여	1504
권한 부여 정책의 와일드카드	1505
권한 부여 정책의 레시피 변수	1506
권한 부여 정책의 특수 문자	1506
권한 부여 정책 예제	1507
IPC 이벤트 스트림 구독	1510

구독 핸들러 정의	1511
구독 핸들러 예제	1513
IPC 모범 사례	1525
로컬 메시지 게시/구독	1527
최소 SDK 버전	1527
권한 부여	1528
PublishToTopic	1530
SubscribeToTopic	1540
예제	1556
AWS IoT Core MQTT 메시지 게시/구독	1590
최소 SDK 버전	1591
권한 부여	1592
PublishToIoTCore	1596
SubscribeToIoTCore	1608
예제	1625
구성 요소 수명 주기와 연동	1645
최소 SDK 버전	1645
권한 부여	1646
UpdateState	1647
SubscribeToComponentUpdates	1649
DeferComponentUpdate	1651
PauseComponent	1652
ResumeComponent	1654
구성 요소 구성과 연동	1655
최소 SDK 버전	1655
GetConfiguration	1656
UpdateConfiguration	1659
SubscribeToConfigurationUpdate	1662
SubscribeToValidateConfigurationUpdates	1667
SendConfigurationValidityReport	1668
보안 암호 값 검색	1669
최소 SDK 버전	1669
권한 부여	1670
GetSecretValue	1671
예제	1677
로컬 새도와 연동	1683

최소 SDK 버전	1684
권한 부여	1685
GetThingShadow	1696
UpdateThingShadow	1703
DeleteThingShadow	1712
ListNamedShadowsForThing	1717
로컬 배포 및 구성 요소 관리	1725
최소 SDK 버전	1726
권한 부여	1726
CreateLocalDeployment	1728
ListLocalDeployments	1731
GetLocalDeploymentStatus	1732
ListComponents	1733
GetComponentDetails	1734
RestartComponent	1735
StopComponent	1738
CreateDebugPassword	1739
클라이언트 디바이스 인증 및 권한 부여	1740
최소 SDK 버전	1741
권한 부여	1741
VerifyClientDeviceIdentity	1743
GetClientDeviceAuthToken	1743
AuthorizeClientDeviceAction	1745
SubscribeToCertificateUpdates	1745
로컬 IoT 디바이스와 상호 작용	1748
클라이언트 디바이스 구성 요소	1748
코어 디바이스에 클라이언트 디바이스 연결	1751
요구 사항	1752
클라이언트 디바이스 지원용 Greengrass 구성 요소	1764
클라우드 검색 구성(콘솔)	1766
클라우드 검색 구성(AWS CLI)	1766
클라이언트 디바이스 연결	1767
오프라인 상태에서 클라이언트 인증	1769
코어 디바이스 엔드포인트 관리	1770
MQTT 브로커 선택	1777
MQTT 브로커에 연결	1778

통신 테스트	1781
Greengrass Discovery RESTful API	1792
클라이언트 디바이스와 간에 MQTT 메시지 릴레이 AWS IoT Core	1798
MQTT 브리지 구성 요소 구성 및 배포	1799
MQTT 메시지 전달	1800
구성 요소의 클라이언트 디바이스와 상호 작용	1801
MQTT 브리지 구성 요소 구성 및 배포	1802
클라이언트 디바이스에서 MQTT 메시지 수신	1803
클라이언트 디바이스로 MQTT 메시지 전송	1803
클라이언트 디바이스 새도와 상호 작용 및 동기화	1803
사전 조건	1804
새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정	1804
구성 요소의 클라이언트 디바이스 새도와 상호 작용	1808
클라이언트 디바이스 새도우를와 동기화 AWS IoT Core	1808
로컬 메시징에 IPv6 사용	1808
IPv6를 사용하도록 IP 감지기 구성	1808
문제 해결	1812
Greengrass 검색 문제	1812
MQTT 연결 문제	1820
디바이스 새도와 연동	1827
구성 요소에서 새도와 연동	1827
새도 상태 검색 및 수정	1828
새도 상태 변경에 대응	1829
로컬 디바이스 새도우를와 동기화 AWS IoT Core	1829
사전 조건	1830
새도 관리자 구성 요소 구성	1830
로컬 새도 동기화	1832
새도 병합 충돌 동작	1832
데이터 스트림 관리	1834
스트림 관리 워크플로우	1835
요구 사항	1835
데이터 보안	1836
로컬 데이터 보안	1836
클라이언트 인증	1837
다음 사항도 참조하세요.	1837
스트림 관리자 구성	1837

스트림 관리자 파라미터	1838
다음 사항도 참조하세요.	1840
스트림 관리자를 사용하는 사용자 지정 구성 요소 생성	1840
스트림 관리자를 사용하는 구성 요소 레시피 정의	1841
애플리케이션 코드에서 스트림 관리자에 연결	1853
StreamManagerClient를 사용하여 스트림 작업	1856
메시지 스트림 생성	1857
메시지 추가	1860
메시지 읽기	1867
스트림 나열	1869
메시지 스트림 설명	1871
메시지 스트림 업데이트	1873
메시지 스트림 삭제	1877
다음 사항도 참조하세요.	1878
지원되는 클라우드 대상에 대한 구성 내보내기	1879
기계 학습 추론 수행	1894
AWS IoT Greengrass ML 추론 작동 방식	1894
AWS IoT Greengrass 버전 2의 차이점은 무엇인가요?	1895
요구 사항	1896
지원되는 모델 소스	1896
지원되는 런타임	1897
기계 학습 구성 요소	1897
SageMaker AI Edge Manager 사용	1902
작동 방식	1903
요구 사항	1904
SageMaker AI Edge Manager 시작하기	1905
기계 학습 구성 요소 사용자 지정	1906
퍼블릭 추론 구성 요소의 구성 수정	1906
샘플 추론 구성 요소와 함께 사용자 지정 모델 사용	1908
사용자 지정 기계 학습 구성 요소 생성	1912
사용자 지정 추론 구성 요소 생성	1914
문제 해결	1921
라이브러리를 가져오지 못함	1922
Cannot open shared object file	1923
Error: ModuleNotFoundError: No module named '<library>'	1923
CUDA 지원 디바이스가 감지되지 않음	1924

해당 파일 또는 디렉터리 없음	1925
RuntimeError: module compiled against API version 0xf but this version of NumPy is	
<version>	1925
picamera.exc.PiCameraError: Camera is not enabled	1926
메모리 오류	1926
디스크 공간 오류	1927
제한 시간 오류	1927
AWS Systems Manager를 사용하여 코어 디바이스 관리	1928
Systems Manager 에이전트 설치	1929
1단계: 일반 Systems Manager 설정 단계 완료	1929
2단계: Systems Manager에 대한 IAM 서비스 역할 생성	1929
3단계: 토큰 교환 역할에 권한 추가	1930
4단계: Systems Manager 에이전트 구성 요소 배포	1934
5단계: Systems Manager를 사용하여 코어 디바이스 등록 확인	1937
Systems Manager Agent 제거	1938
1단계: Systems Manager에서 코어 디바이스 등록 취소	1939
2단계: Systems Manager Agent 구성 요소 제거	1939
3단계: Systems Manager Agent 소프트웨어 제거	1940
보안	1941
데이터 보호	1942
데이터 암호화	1943
하드웨어 보안 통합	1945
장치 인증 및 권한 부여	1956
X.509 인증서	1957
AWS IoT 정책	1958
코어 디바이스의 AWS IoT 정책 업데이트	1963
최소 AWS IoT 정책	1967
클라이언트 디바이스를 지원하는 최소 AWS IoT 정책	1970
클라이언트 디바이스에 대한 최소 AWS IoT 정책	1972
ID 및 액세스 관리	1974
대상	1974
ID를 통한 인증	1975
정책을 사용하여 액세스 관리	1976
다음 사항도 참조하세요.	1977
AWS IoT Greengrass 에서 IAM을 사용하는 방법	1978
ID 기반 정책 예시	1982

코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여	1984
설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책	1989
Greengrass 서비스 역할	1993
AWS 관리형 정책	2002
교차 서비스 혼동된 대리인 방지	2008
자격 증명 및 액세스 문제 해결	2008
프록시 또는 방화벽을 통해 디바이스 트래픽 허용	2010
기본 작업을 위한 엔드포인트	2010
자동 프로비저닝으로 설치하기 위한 엔드포인트	2016
AWS제공 구성 요소에 대한 엔드포인트	2017
규정 준수 확인	2017
FIPS 엔드포인트	2017
배포를 통해 FIPS 엔드포인트 활성화	2018
수동 리소스 프로비저닝을 사용하여 FIPS 엔드포인트로 nucleus 설치	2020
플릿 프로비저닝을 사용하여 FIPS 엔드포인트 설치	2052
자동 리소스 프로비저닝을 사용하여 FIPS 엔드포인트 설치	2069
FIPS 규정 준수 자사 구성 요소	2085
복원성	2086
인프라 보안	2087
구성 및 취약성 분석	2087
코드 무결성	2088
VPC 엔드포인트(AWS PrivateLink)	2089
AWS IoT Greengrass VPC 엔드포인트에 대한 고려 사항	2090
AWS IoT Greengrass 컨트롤 플레인 작업을 위한 인터페이스 VPC 엔드포인트 생성	2090
에 대한 VPC 엔드포인트 정책 생성 AWS IoT Greengrass	2091
VPC에서 AWS IoT Greengrass 코어 디바이스 작동	2091
보안 모범 사례	2096
가능한 최소 권한 부여	2096
Greengrass 구성 요소에서 자격 증명을 하드 코딩하지 마세요.	2096
민감한 정보를 기록하지 않음	2097
장치의 시계를 동기화 상태로 유지	2097
Cipher Suite 권장 사항	2097
다음 사항도 참조하세요.	2098
AWS IoT Greengrass V2용 AWS IoT Device Tester 사용	2099
AWS IoT Greengrass 검증 제품군	2099
사용자 지정 테스트 제품군	2100

지원되는 버전	2100
AWS IoT Greengrass V2의 최신 IDT 버전	2100
용 이전 IDT 버전 AWS IoT Greengrass	2101
지원되지 않는 AWS IoT Device Tester for AWS IoT Greengrass V2 버전	2102
AWS IoT Greengrass V2용 IDT 다운로드	2107
수동으로 IDT 다운로드	2107
프로그래밍 방식으로 IDT 다운로드	2108
IDT를 사용하여 AWS IoT Greengrass 검증 제품군 실행	2114
테스트 제품군 버전	2114
테스트 그룹 설명	2115
사전 조건	2118
IDT 테스트를 실행하도록 장치 구성	2139
IDT 설정 구성	2148
AWS IoT Greengrass 검증 제품군 실행	2165
결과 및 로그 이해	2168
IDT를 사용하여 자체 테스트 제품군 개발 및 실행	2172
AWS IoT Greengrass 용 IDT의 최신 버전을 다운로드합니다.	2118
테스트 도구 모음 생성 워크플로	2173
튜토리얼: 샘플 IDT 테스트 도구 모음 구축 및 실행	2174
튜토리얼: 간단한 IDT 테스트 도구 모음 개발	2179
IDT 테스트 도구 모음 구성 파일 만들기	2188
IDT 테스트 오케스트레이터 구성	2196
IDT 상태 머신 구성	2203
IDT 테스트 케이스 실행 파일 생성	2226
IDT 컨텍스트 사용	2232
테스트 러너를 위한 설정 구성	2237
사용자 지정 테스트 도구 모음 디버그 및 실행	2248
IDT 테스트 결과 및 로그 검토	2251
IDT 사용량 지표	2257
IDT for AWS IoT Greengrass V2 문제 해결	2263
오류를 찾을 수 있는 곳	2263
IDT for AWS IoT Greengrass V2 오류 해결	2264
AWS IoT Device Tester용 AWS IoT Greengrass에 대한 지원 정책	2271
Greengrass 기반 IoT 솔루션	2272
Eurotech	2272
문제 해결	2273

AWS IoT Greengrass 코어 소프트웨어 및 구성 요소 로그 보기	2273
AWS IoT Greengrass 핵심 소프트웨어 문제	2273
ListDeployments API에서 ThrottlingException 발생	2275
코어 디바이스를 설정할 수 없음	2275
AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 시작할 수 없음	2275
nucleus를 시스템 서비스로 설정할 수 없음	2275
에 연결할 수 없음 AWS IoT Core	2276
메모리 부족 오류	2276
Greengrass CLI를 설치할 수 없음	2276
User root is not allowed to execute	2277
com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/ group to run with	2277
Failed to map segment from shared object: operation not permitted	2277
Windows 서비스를 설정하지 못함	2278
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager	2278
com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime	2279
software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid	2279
software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy	2280
Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request	2280
Operation aws.greengrass#<operation> is not supported by Greengrass	2281
java.io.FileNotFoundException: <stream-manager-store-root-dir>/ stream_manager_metadata_store (Permission denied)	2281
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist	2282
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn> .	2282
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed	2283
java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi	2283
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED	2284
Greengrass core device stuck on nucleus v2.12.3	2284

Greengrass nucleus v2.14.0 systemd template issue	2286
AWS IoT Greengrass 클라우드 문제	2287
An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null	2287
Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}	2287
INACTIVE deployment status	2288
코어 디바이스 배포 문제	2288
Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact	2289
Error:	
com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.	2290
Error:	
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>	2291
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility	2292
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	2292
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	2293
Info:	
com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration	2294
Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy	2294
Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration	2295

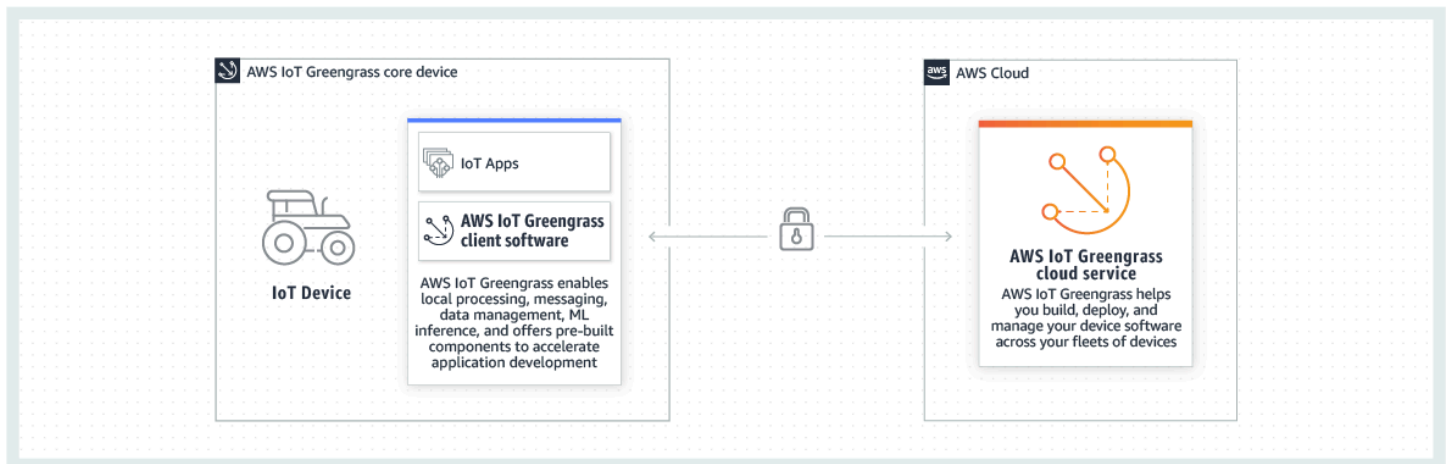
Caused by:	
software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException:	
null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)	2295
코어 디바이스 구성 요소 문제	2295
Warn: '<command>' is not recognized as an internal or external command	2296
Python 스크립트는 메시지를 기록하지 않습니다.	2296
기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않음	2297
awsiot.greengrasscoreipc.model.UnauthorizedError	2299
com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"	2299
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)	2300
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)	2301
com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers	2302
Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"	2302
copyFrom: <configurationPath> is already a container, not a leaf	2303
com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'	2303
java.io.IOException: Cannot run program "cmd": [LogonUser] The password for this account has expired.	2304
aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant	2305
코어 디바이스 Lambda 함수 구성 요소 문제	2305
The following cgroup subsystems are not mounted: devices, memory	2305
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>'	2306
구성 요소 버전 중단	2306
Greengrass CLI 문제	2307
java.lang.RuntimeException: Unable to create ipc client	2307
AWS CLI 문제	2307
Error: Invalid choice: 'greengrassv2'	2308
세부 배포 오류 코드	2308
권한 오류	2310

요청 오류	2311
구성 요소 레시피 오류	2313
AWS 구성 요소 오류, 사용자 구성 요소 오류, 구성 요소 오류	2315
디바이스 오류	2316
종속성 오류	2317
HTTP 오류	2318
네트워크 오류	2319
Nucleus 오류	2319
서버 오류	2320
클라우드 서비스 오류	2321
일반 오류	2322
알 수 없는 오류	2323
세부 구성 요소 상태 코드	2323
리소스에 태그 지정	2326
에서 태그 사용 AWS IoT Greengrass V2	2326
로 태그 지정 AWS Management Console	2326
AWS IoT Greengrass V2 API로 태그 지정	2326
IAM 정책에 태그 사용	2327
AWS CloudFormation 리소스	2329
AWS IoT Greengrass 및 CloudFormation 템플릿	2329
ComponentVersion 템플릿 예제	2329
배포 템플릿 예제	2330
에 대해 자세히 알아보기 CloudFormation	2331
오픈 소스 소프트웨어	2332
문서 이력	2333
AWS 용어집	2389
.....	mmcccxc

AWS IoT Greengrass란 무엇인가요?

AWS IoT Greengrass 는 디바이스에서 IoT 애플리케이션을 빌드, 배포 및 관리하는 데 도움이 되는 오픈 소스 사물 인터넷(IoT) 엣지 런타임 및 클라우드 서비스입니다. AWS IoT Greengrass 를 사용하여 디바이스가 생성한 데이터에 대해 로컬로 작동하고, 기계 학습 모델을 기반으로 예측을 실행하고, 디바이스 데이터를 필터링 및 집계할 수 있는 소프트웨어를 구축할 수 있습니다. AWS IoT Greengrass 사용하면 디바이스가 해당 데이터가 생성된 위치와 더 가까운 곳에서 데이터를 수집 및 분석하고, 로컬 이벤트에 자율적으로 대응하고, 로컬 네트워크의 다른 디바이스와 안전하게 통신할 수 있습니다. Greengrass 디바이스는와 안전하게 통신 AWS IoT Core 하고 IoT 데이터를 로 내보낼 수도 있습니다 AWS 클라우드. AWS IoT Greengrass 를 사용하여 엣지 디바이스를 AWS 서비스 또는 타사 서비스에 연결할 수 있는 구성 요소라고 하는 사전 구축된 소프트웨어 모듈을 사용하여 엣지 애플리케이션을 구축할 수 있습니다. AWS IoT Greengrass 를 사용하여 Lambda 함수, Docker 컨테이너, 기본 운영 체제 프로세스 또는 원하는 사용자 지정 런타임을 사용하여 소프트웨어를 패키징하고 실행할 수도 있습니다.

다음 예제에서는 AWS IoT Greengrass 디바이스와 상호 작용하는 방식을 보여줍니다 AWS 클라우드.



새로운 기능

AWS IoT Greengrass V2 에는 새로운 기능과 개선 사항이 도입되었습니다. 다음은 버전 2에서 제공되는 새로운 기능에 대한 자세한 내용을 포함하고 있습니다.

- [의 새로운 기능 AWS IoT Greengrass Version 2](#)

를 처음 사용하는 경우 AWS IoT Greengrass

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

- [AWS IoT Greengrass 작동 방식](#)

그런 다음 [시작하기 자습서](#)에 따라의 기본 기능을 사용해 보세요 AWS IoT Greengrass. 이 자습서에서는 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하고, Hello World 구성 요소를 개발하고, 배포를 위해 해당 구성 요소를 패키징합니다.

의 기존 사용자 AWS IoT Greengrass V1

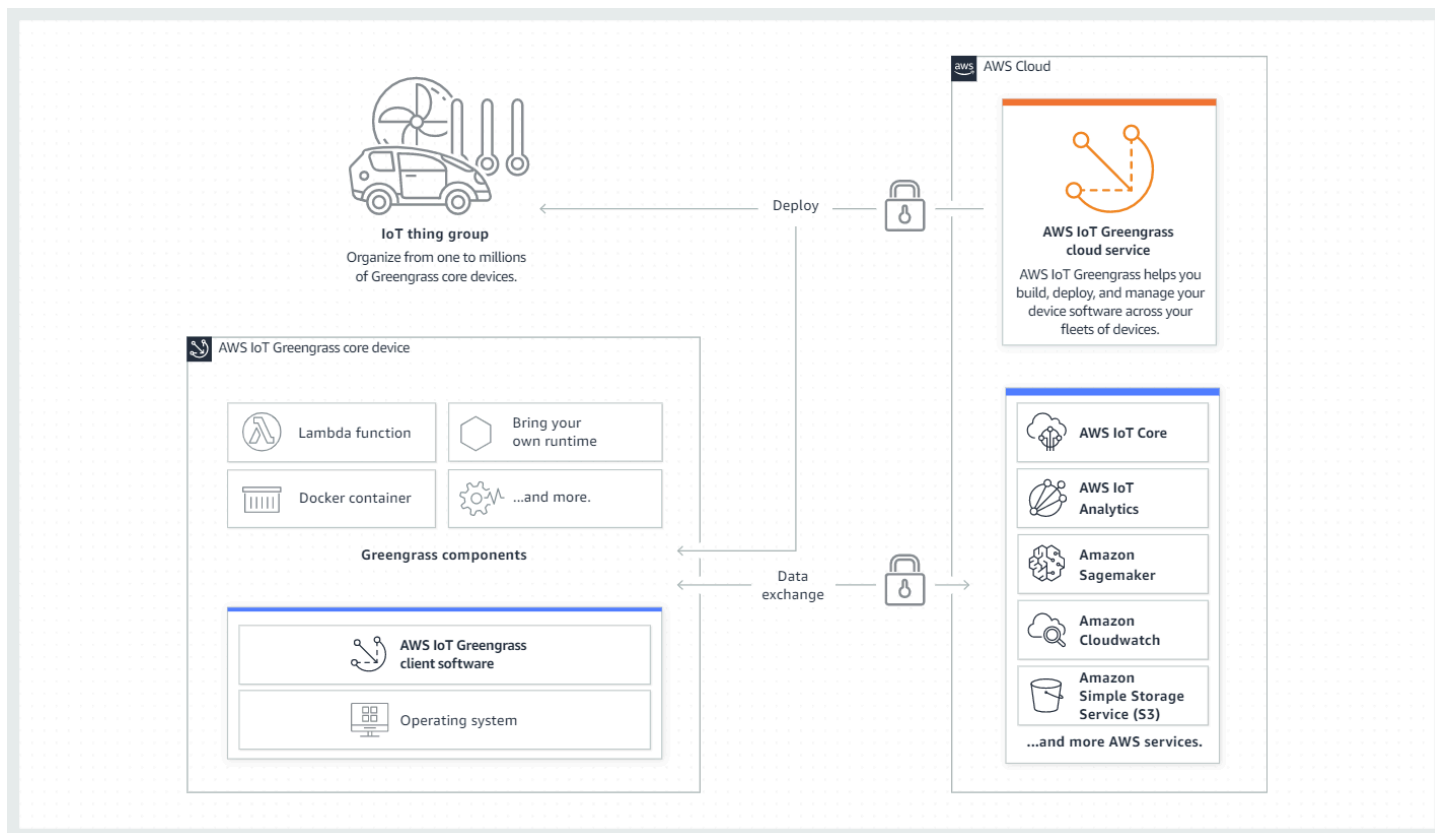
지원 종료 공지: 2026 AWS 년 10월 7일에는에 대한 지원을 종료합니다 AWS IoT Greengrass Version 1. 2026년 10월 7일 이후에는 AWS IoT Greengrass V1 콘솔 또는 AWS IoT Greengrass V1 리소스에 더 이상 액세스할 수 없습니다. 버전 1에서 버전 2로 이전하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 버전 1에서 마이그레이션](#) 섹션을 참조하세요.

AWS IoT Greengrass 작동 방식

AWS IoT Greengrass 코어 소프트웨어라고도 하는 AWS IoT Greengrass 클라이언트 소프트웨어는 ARM 또는 x86 아키텍처를 사용하는 디바이스의 경우 Ubuntu 또는 Raspberry Pi OS와 같은 Windows 및 Linux 기반 배포판에서 실행됩니다. 를 사용하면 디바이스가 생성하는 데이터에 대해 로컬로 작동하도록 프로그래밍하고, 기계 학습 모델을 기반으로 예측을 실행하고, 디바이스 데이터를 필터링 및 집계 AWS IoT Greengrass할 수 있습니다.는 선택한 AWS Lambda 함수, Docker 컨테이너, 기본 OS 프로세스 또는 사용자 지정 런타임의 로컬 실행을 AWS IoT Greengrass 활성화합니다.

AWS IoT Greengrass 는 엣지 디바이스 기능을 쉽게 확장할 수 있는 구성 요소라는 사전 구축된 소프트웨어 모듈을 제공합니다. AWS IoT Greengrass 구성 요소를 사용하면 엣지에서 AWS 서비스 및 타사 애플리케이션에 연결할 수 있습니다. IoT 애플리케이션을 개발한 후를 AWS IoT Greengrass 사용하면 필드의 디바이스 플릿에서 해당 애플리케이션을 원격으로 배포, 구성 및 관리할 수 있습니다.

다음 예제는 AWS IoT Greengrass 디바이스와의 AWS IoT Greengrass 클라우드 서비스 및 기타 AWS 서비스와 상호 작용하는 방법을 보여줍니다 AWS 클라우드.



의 주요 개념 AWS IoT Greengrass

다음은 AWS IoT Greengrass 이해 및 사용에 있어 필수 개념입니다.

AWS IoT 사물

AWS IoT 사물은 특정 디바이스 또는 논리적 개체의 표현입니다. 사물에 대한 정보는 AWS IoT 레지스트리에 저장됩니다.

Greengrass 코어 디바이스

AWS IoT Greengrass 코어 소프트웨어를 실행하는 디바이스입니다. Greengrass 코어 디바이스는 AWS IoT 사물입니다. AWS IoT 사물 그룹에 여러 코어 디바이스를 추가하여 Greengrass 코어 디바이스 그룹을 생성하고 관리할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 디바이스 설정](#) 단원을 참조하십시오.

Greengrass 클라이언트 디바이스

MQTT를 통해 Greengrass 코어 디바이스에 연결하고 통신하는 디바이스입니다. Greengrass 클라이언트 디바이스는 AWS IoT 사물입니다. 코어 디바이스는 연결된 클라이언트 디바이스에서 데이터를 처리, 필터링 및 집계할 수 있습니다. 클라이언트 디바이스, AWS IoT Core 클라우드 서비스

및 Greengrass 구성 요소 간에 MQTT 메시지를 릴레이하도록 코어 디바이스를 구성할 수 있습니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

클라이언트 디바이스는 [FreeRTOS](#)를 실행하거나 [AWS IoT Device SDK](#) 또는 [Greengrass Discovery API](#)를 사용하여 연결할 수 있는 코어 디바이스에 관한 정보를 가져올 수 있습니다.

Greengrass 구성 요소

Greengrass 코어 디바이스에 배포되고 실행되는 소프트웨어 모듈입니다. 를 사용하여 개발 및 배포되는 모든 소프트웨어는 구성 요소로 모델링 AWS IoT Greengrass 됩니다. 애플리케이션에서 사용할 수 있는 기능을 제공하는 사전 구축된 퍼블릭 구성 요소를 AWS IoT Greengrass 제공합니다. 로컬 디바이스 또는 클라우드에서 사용자 지정 구성 요소를 직접 개발할 수도 있습니다. 사용자 지정 구성 요소를 개발한 후 AWS IoT Greengrass 클라우드 서비스를 사용하여 단일 또는 여러 코어 디바이스에 배포할 수 있습니다. 사용자 지정 구성 요소를 생성하고 이 구성 요소를 코어 디바이스에 배포할 수 있습니다. 이 경우 코어 디바이스는 다음 리소스를 다운로드하여 구성 요소를 실행합니다.

- 레시피: 구성 요소 세부 정보, 구성 및 파라미터를 정의하여 소프트웨어 모듈을 설명하는 JSON 또는 YAML 파일입니다.
- 아티팩트: 디바이스에서 실행할 소프트웨어를 정의하는 소스 코드, 바이너리 또는 스크립트입니다. 아티팩트를 처음부터 생성하거나 Lambda 함수, Docker 컨테이너 또는 사용자 지정 런타임을 사용하여 구성 요소를 생성할 수 있습니다.
- 종속성: 종속 구성 요소의 자동 업데이트 또는 재시작을 적용할 수 있는 구성 요소 간의 관계입니다. 예를 들어 암호화 구성 요소에 종속되는 보안 메시지 처리 구성 요소가 있을 수 있습니다. 이렇게 하면 암호화 구성 요소에 대한 모든 업데이트 시 메시지 처리 구성 요소가 자동으로 업데이트되고 재시작될 수 있습니다.

자세한 내용은 [AWS 제공 구성 요소](#) 및 [AWS IoT Greengrass 구성 요소 개발](#) 섹션을 참조하세요.

배포

구성 요소를 보내고 원하는 구성 요소 구성을 대상 디바이스에 적용하는 프로세스로, 단일 Greengrass 코어 디바이스 또는 Greengrass 코어 디바이스 그룹이 될 수 있습니다. 배포는 업데이트된 구성 요소 구성을 대상에 자동으로 적용하고 종속성으로 정의된 다른 구성 요소를 포함합니다. 기존 배포를 복제하여 동일한 구성 요소를 사용하지만 다른 대상에 배포되는 새 배포를 생성할 수도 있습니다. 배포는 연속적입니다. 따라서 구성 요소 또는 배포의 구성 요소 구성에 대한 모든 업데이트가 모든 대상으로 자동 전송됩니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어

버전 2.14부터는 디바이스 런타임의 두 가지 대체 구현인 nucleus라고 하는 실행 파일을 AWS IoT Greengrass 제공합니다. 첫 번째이자 이전에는 유일했던 nucleus는 Java로 구현되어 있습니다. 이 선택지는 아키텍처와 운영 체제 전반에서 가장 높은 이식성을 제공합니다. 그러나 Java 가상 머신에 대한 의존성이 있어 메모리 사용량이 큰 단점이 있습니다.

두 번째이자 새로 추가된 nucleus는 C로 구현되어 있습니다. 이 선택지는 메모리 사용량을 크게 줄여줍니다. 하지만 대상 아키텍처와 운영 체제마다 별도로 배포하거나 소스에서 컴파일해야 합니다. 둘을 구분해야 하는 경우 첫 번째 구현을 Greengrass nucleus라고 하고 후자를 Greengrass nucleus lite라고 합니다.

- 선택적 구성 요소: 이러한 구성 가능한 구성 요소는에서 제공하며 엣지 디바이스에서 추가 기능을 AWS IoT Greengrass 활성화합니다. 요구 사항에 따라 데이터 스트리밍, 로컬 기계 학습 추론 또는 로컬 명령줄 인터페이스와 같이 디바이스에 배포하려는 선택적 구성 요소를 선택할 수 있습니다. 자세한 내용은 [AWS 제공 구성 요소](#) 단원을 참조하십시오.

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

의 기능 AWS IoT Greengrass

AWS IoT Greengrass Version 2 는 다음 요소로 구성됩니다.

- 소프트웨어 배포
 - AWS IoT Greengrass 코어 소프트웨어의 최소 설치인 [Greengrass nucleus 구성 요소입니다](#). 이 구성 요소는 Greengrass 구성 요소의 배포, 오케스트레이션 및 수명 주기를 관리합니다.
 - 서비스, 프로토콜 및 소프트웨어와 통합되는 추가 선택적 [AWS제공 구성 요소](#).
 - 사용자 지정 Greengrass 구성 요소를 생성, 테스트, 빌드, 게시 및 배포하는 데 사용할 수 있는 [Greengrass 개발 도구](#).
 - AWS IoT Device SDK- 사용자 지정 Greengrass 구성 요소를 위한 [프로세스 간 통신\(IPC\) 라이브러리](#)와 클라이언트 디바이스를 위한 [Greengrass 검색 라이브러리](#)가 포함되어 있습니다.
 - 코어 디바이스의 [데이터 스트림 관리](#)에 사용할 수 있는 Stream Manager SDK.
- 클라우드 서비스
 - AWS IoT Greengrass V2 API
 - AWS IoT Greengrass V2 콘솔

AWS IoT Greengrass 코어 소프트웨어

엣지 디바이스에서 실행되는 AWS IoT Greengrass 코어 소프트웨어를 사용하여 다음을 수행할 수 있습니다.

- AWS 클라우드로의 자동 내보내기를 사용하여 로컬 디바이스에서 데이터 스트림을 처리합니다. 자세한 내용은 [Greengrass 코어 디바이스에서 데이터 스트림 관리](#) 단원을 참조하십시오.
- AWS IoT 및 구성 요소 간의 MQTT 메시징을 지원합니다. 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 단원을 참조하십시오.
- MQTT를 통해 연결하고 통신하는 로컬 디바이스와 상호 작용합니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.
- 구성 요소 간의 로컬 게시 및 구독 메시징을 지원합니다. 자세한 내용은 [로컬 메시지 게시/구독](#) 단원을 참조하십시오.
- 구성 요소 및 Lambda 함수를 배포하고 간접적으로 호출합니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.
- 스크립트 설치 및 실행 지원과 같은 구성 요소 수명 주기를 관리합니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 단원을 참조하십시오.
- AWS IoT Greengrass 코어 소프트웨어 및 사용자 지정 구성 요소의 안전한 over-the-air(OTA) 소프트웨어 업데이트를 수행합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트 및 디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.
- 로컬 보안 호의 안전하고 암호화된 저장과 구성 요소에 의해 제어되는 액세스를 제공합니다. 자세한 내용은 [보안 암호 관리자](#) 단원을 참조하십시오.
- 디바이스 인증 및 권한 부여를 통해 디바이스와 AWS 클라우드 간의 연결을 보호합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.

지속적 소프트웨어 배포를 생성하는 AWS IoT Greengrass APIs 통해 Greengrass 코어 디바이스를 구성하고 관리합니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

일부 기능은 특정 플랫폼에서만 지원됩니다. 자세한 내용은 [Greengrass 기능 호환성](#) 단원을 참조하십시오.

지원되는 플랫폼, 요구 사항 및 다운로드에 대한 자세한 내용은 [AWS IoT Greengrass 코어 디바이스 설정](#) 섹션을 참조하세요.

이 소프트웨어를 다운로드하면 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 동의하는 것입니다.

Greengrass 기능 호환성

AWS IoT Greengrass 는 다양한 운영 체제를 실행하는 디바이스를 지원합니다. 일부 기능은 특정 운영 체제에서만 지원됩니다. 다음 표를 사용하여 지원되는 각 운영 체제에서 사용할 수 있는 기능을 알아보세요. 지원되는 운영 체제, 요구 사항 및 Greengrass 코어 디바이스 설정 방법에 대한 자세한 내용은 [AWS IoT Greengrass 코어 디바이스 설정](#) 섹션을 참조하세요.

메시징

기능	Linux	Windows	Greengrass Nucleus Lite(Linux)
AWS IoT 및 구성 요소 간에 MQTT 메시지 교환	 예	 예	 예 (MQTT5 확장 제외)
구성 요소 간에 로컬 게시/구독 메시지 교환	 예	 예	 예
MQTT를 통해 로컬 IoT 디바이스와 연동	 예	 예	 아니요
Modbus-RTU 구성 요소를 사용하여 로컬 Modbus-RTU 디바이스와 연동	 예	 아니요	 아니요

보안

기능	Linux	Windows	Greengrass lite(Linux)
디바이스 인증 및 권한 부여를 사용하여 보안 연결	 예	 예	 예
에서 암호화된 보안 암호 배포 및 액세스 AWS Secrets Manager	 예	 예	 아니요
HSM(하드웨어 보안 모듈)을 사용하여 디바이스의 프라이빗 키와 인증서를 안전하게 저장	 예	 아니요	<u>예</u>
를 사용하여 코어 디바이스 감사 AWS IoT Device Defender	 예	 예	 아니요
자격 AWS 증명을 사용하여 AWS 서비스와 상호 작용	 예	 예	 예

설치

기능	Linux	Windows	Greengrass lite(Linux)
자동 프로비저닝 AWS IoT Greengrass 을 사용하여 설치	 예	 예	 아니요
수동 프로비저닝 AWS IoT Greengrass 을 사용하여 설치	 예	 예	 예
AWS IoT 플릿 프로비저닝 AWS IoT Greengrass 을 사용하여 설치	 예	 예	 예
사용자 지정 프로비저닝 AWS IoT Greengrass 을 사용하여 설치	<u>예</u>	<u>예</u>	 예
사전 구축된 Docker 이미지를 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행	 예	 아니요	 아니요

 Note

AWS IoT Greengrass 는 시스템 지원 Docker 컨테이너에 설치하고 실행할 수 있습니다.

원격 유지 관리 및 업데이트

기능	Linux	Windows	Greengrass lite(Linux)
안전한 무선 업데이트 (OTA) 소프트웨어 업데이트 수행	 예	 예	 예
를 사용하여 코어 디바이스 관리 AWS Systems Manager	 예	 아니요	 아니요
AWS IoT 보안 터널링을 사용하여 코어 디바이스에 연결	 예	 아니요	 예

기계 학습

기능	Linux	Windows	Greengrass lite(Linux)
Amazon SageMaker AI Edge Manager를 사용하여 기계 학습 추론 수행	 예	 예	 아니요
Amazon Lookout for Vision을 사용하여 기계 학습 추론 수행	 예	 아니요	 아니요
DLR을 사용하여 기계 학습 추론 수행	 예	 예	 아니요

기능	Linux	Windows	Greengrass lite(Linux)
TensorFlow를 사용하여 기계 학습 추론 수행	 예	 예	 아니요

구성 요소 기능

기능	Linux	Windows	Greengrass lite(Linux)
Lambda 함수 배포 및 호출	 예	 아니요	 아니요
구성 요소에서 Docker 컨테이너 실행	 예	 예	 예
스트림 관리자를 사용하여 대용량 데이터 스트림 처리 및 내보내기	 예	 예	 예
수명 주기 스크립트를 사용하여 구성 요소 수명 주기 관리	 예	 예	 예
디바이스 새도와 연동	 예	 예	 아니요

기능	Linux	Windows	Greengrass lite(Linux)
Amazon CloudWatch Logs에 로그 업로드	 예	 예	 예
Amazon CloudWatch 지표 구성 요소를 사용하여 Amazon CloudWatch 지표에 데이터 업로드	 예	 예	 예
Amazon SNS 구성 요소를 사용하여 Amazon Simple Notification Service에 메시지 게시	 예	 아니요	 아니요
스트림 관리자를 사용하여 Amazon Kinesis Data Streams 전송 스트림에 데이터 게시	 예	 예	 예
Firehose 구성 요소를 사용하여 Amazon Data Firehose 전송 스트림에 데이터 게시	 예	 아니요	 아니요
실시간 시스템 원격 측정 지표 수집 및 관련 작업 수행	 예	 예	 아니요

기능	Linux	Windows	Greengrass lite(Linux)
구성 요소 프로세스에 대한 시스템 리소스 제한 구성	 예	 아니요	 아니요
구성 요소 프로세스 일시 중지 및 재개	 예	 아니요	 아니요
AWS IoT SiteWise 구성 요소를 AWS IoT SiteWise 사용하여와 통합	 예	 예	 아니요
Kinesis Video Streams 구성 요소용 엣지 커넥터를 사용하여 Amazon Kinesis Video Streams에 비디오 스트림 게시	 예	 아니요	 아니요

구성 요소 개발

기능	Linux	Windows	Greengrass lite(Linux)
코어 디바이스에서 로컬로 구성 요소 개발	 예	 예	 예

기능	Linux	Windows	Greengrass lite(Linux)
AWS IoT Greengrass CLI를 사용하여 코어 디바이스와 상호 작용	 예	 예	 아니요
로컬 디버그 콘솔을 사용하여 코어 디바이스와 연동	 예	 예	 아니요
사용자 지정 구성 요소에서 Python AWS IoT Device SDK 용 사용	 예	 예	 예
사용자 지정 구성 요소에서 C++ AWS IoT Device SDK 용 사용	 예	 예	 예
사용자 지정 구성 요소에서 AWS IoT Device SDK for Java 사용	 예	 예	 예

디바이스 인증

기능	Linux	Windows	Greengrass lite(Linux)
AWS IoT Device Tester 용 AWS IoT Greengrass V2 를 사용하여 IoT 디바이스 검증	 예	 예	 아니요

AWS IoT Greengrass nucleus 런타임 선택

버전 2.14.0부터는 nucleus라고 하는 실행 파일인 디바이스 런타임의 두 가지 대체 구현을 AWS IoT Greengrass 제공합니다. 구현 차이점에도 불구하고 두 런타임 모두 AWS IoT Greengrass 서비스 및 APIs와 호환되며 Greengrass SDK를 사용하여에서 제공하는 구성 요소를 배포 AWS 하거나 사용자 지정 구성 요소를 개발할 수 있습니다. 필요에 따라 동일한 플릿 내에서 두 유형의 nucleus를 사용하여 디바이스를 혼합할 수도 있습니다.

그러나 원하는 이동성이나 특정 메모리 절감 효과를 얻으려면, Greengrass 디바이스에 배포하는 nucleus가 AWS IoT 솔루션의 개발을 가속화하는 데 사용하려는 구성 요소와 호환되는지 반드시 확인해야 합니다. 구성 요소 호환성에 대한 자세한 내용은 [구성 요소](#)를 참조하세요.

궁극적으로 특정 사용 사례, 디바이스 제약 조건, 기능 요구 사항 및 운영 체제에 따라 두 가지 Greengrass 런타임 옵션 중 어느 것을 선택할지가 결정됩니다.

Greengrass nucleus

AWS IoT Greengrass nucleus는 컴퓨팅 리소스가 더 많은 게이트웨이, 서버 및 엣지 디바이스를 비롯한 다양한 디바이스 AWS IoT Greengrass 에서 실행할 수 있는 완전한 기능을 갖춘 런타임입니다. 다음과 같은 경우 Greengrass nucleus를 선택하는 것이 좋습니다.

- 컴퓨팅 리소스: 디바이스에 128MB 이상의 RAM 및 비교적 강력한 프로세서(예: 1GHz 이상의 클럭)와 같은 충분한 컴퓨팅 리소스가 있습니다.
- 전체 OS 지원 필요: Greengrass nucleus는 가장 광범위한 운영 체제(대부분의 Linux 배포판 및 Windows 포함)를 지원합니다.
- 구성 요소 호환성: Greengrass nucleus는 서비스 팀 및 파트너가 게시한 기존 구성 요소와 가장 완벽하게 호환됩니다 AWS IoT .

Greengrass nucleus lite

AWS IoT Greengrass nucleus lite는 리소스가 제한된 디바이스 AWS IoT Greengrass 에서 실행할 수 있는 경량 오픈 소스 런타임입니다. 이는 스마트 홈 허브, 스마트 에너지 측정기, 스마트 차량, 엣지 AI, 로봇틱스 등 대량의 애플리케이션에 사용되는 저가 싱글보드 컴퓨터에 유용할 수 있습니다. 디바이스가 다음과 같은 경우 Greengrass nucleus lite를 선택하는 것이 좋습니다.

- 리소스 제약: 디바이스가 RAM(512MB 이하), 저장 공간(플래시), 저성능 프로세서(1GHz 미만) 등 제한된 리소스를 가진 경우.

- 종속성 제한: 디바이스 공급업체 소프트웨어 플랫폼은 Greengrass nucleus에 필요한 Java 또는 특정 JVM을 지원하지 않습니다.
- 운영 체제: 디바이스가 systemd를 지원하는 Linux 배포판(예: Ubuntu, Yocto)을 실행하는 경우.

Greengrass nucleus lite의 현재 제한 사항

AWS IoT Greengrass v2.14.0에 포함된 Greengrass nucleus lite(v.2.0.0) 런타임은 Greengrass nucleus(v2.14.0)에서 사용할 수 있는 기능의 하위 집합을 제공합니다.

AWS IoT Greengrass IPC(프로세스 간 통신) 메커니즘을 사용하면 구성 요소가 Greengrass nucleus와 통신할 수 있습니다. 경량 버전인 nucleus lite는 다음과 같은 일부 기능만 지원합니다.

기능	가용성
SubscribeToTopic	Available
PublishToTopic	Available
PublishToIoTCore	Available
SubscribeToIoTCore	Available
UpdateState	현재 사용할 수 없음
SubscribeToComponentUpdates	현재 사용할 수 없음
DeferComponentUpdate	현재 사용할 수 없음
GetConfiguration	Available
UpdateConfiguration	Available
SubscribeToConfigurationUpdate	Available
SubscribeToValidateConfigurationUpdates	현재 사용할 수 없습니다.
SendConfigurationValidityReport	현재 사용할 수 없습니다.
GetSecretValue	현재 사용할 수 없습니다.

기능	가용성
PutComponentMetric	현재 사용할 수 없음
GetComponentDetails	현재 사용할 수 없음
RestartComponent	현재 사용할 수 없음
StopComponent	현재 사용할 수 없음
CreateLocalDeployment	Available
CancelLocalDeployment	현재 사용할 수 없음
GetLocalDeploymentStatus	현재 사용할 수 없음
ListLocalDeployments	현재 사용할 수 없음
ListComponents	현재 사용할 수 없음
ValidateAuthorizationToken	Available
CreateDebugPassword	현재 사용할 수 없음
PauseComponent	현재 사용할 수 없음
ResumeComponent	현재 사용할 수 없음
GetThingShadow	현재 사용할 수 없음
UpdateThingShadow	현재 사용할 수 없음
DeleteThingShadow	현재 사용할 수 없음
ListNamedShadowsForThing	현재 사용할 수 없음
SubscribeToCertificateUpdates	현재 사용할 수 없음
VerifyClientDeviceIdentity	현재 사용할 수 없음
GetClientDeviceAuthToken	현재 사용할 수 없음

기능	가용성
AuthorizeClientDeviceAction	현재 사용할 수 없음

의 새로운 기능 AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 는 다음 기능을 도입 AWS IoT Greengrass 하는의 메이저 버전입니다.

- AWS IoT Greengrass nucleus lite - AWS IoT Greengrass nucleus 버전 2.14.0 이상은 AWS IoT Greengrass nucleus lite를 제공합니다. Greengrass nucleus lite는 리소스가 제한된 디바이스에서 Greengrass를 실행할 수 있게 해주는 경량 오픈 소스 런타임입니다. 이는 스마트 홈 허브, 스마트 에너지 측정기, 스마트 차량, 엣지 AI, 로봇틱스 등 대량의 애플리케이션에 사용되는 저가 싱글보드 컴퓨터에 유용할 수 있습니다. 자세한 내용은 [Greengrass nucleus lite](#) 단원을 참조하십시오.
- 게시자 지원 구성 요소 - AWS IoT Greengrass 이제는 게시자 지원 구성 요소를 제공합니다. 이러한 구성 요소는 타사 공급업체에서 개발, 제공 및 서비스합니다. 자세한 내용은 [게시자 지원 구성 요소](#) 단원을 참조하십시오.
- VPC에서 Greengrass 디바이스 작동 - 이제 VPC에서 Greengrass 코어 디바이스를 작동할 수 있습니다. 이를 사용하면 퍼블릭 인터넷에 액세스하지 않고 VPC에서 배포를 수행할 수 있습니다. 자세한 내용은 [VPC에서 AWS IoT Greengrass 코어 디바이스 작동](#) 단원을 참조하십시오.
- GTF(Greengrass Testing Framework) - AWS IoT Greengrass Version 2 에 대한 GTF를 이제 사용할 수 있습니다. GTF는 중단 간 자동화를 지원하는 빌딩 블록의 컬렉션입니다. 이를 통해 AWS IoT Greengrass Version 2 내부 고객은 서비스 팀이 소프트웨어 변경 검증, 자동 수락 및 품질 보증 목적으로 사용하는 것과 동일한 테스트 프레임워크를 사용할 수 있습니다. 자세한 내용은 [Greengrass Testing Framework on Github](#)를 참조하세요.
- PSA 인증 - AWS IoT Greengrass nucleus 버전 2.7.0 이상이 이제 플랫폼 보안 아키텍처(PSA) 인증을 받았습니다. 자세한 내용은 [AWS IoT Greengrass PSA 인증](#)을 참조하세요.

AWS IoT Greengrass 릴리스 정보에서는 AWS IoT Greengrass 새로운 기능, 업데이트 및 개선 사항, 일반 fixes. AWS IoT Greengrass has와 같은 릴리스에 대한 세부 정보를 제공합니다.

- 에 대한 새로운 기능 릴리스 AWS IoT Greengrass
- AWS IoT Greengrass 코어 소프트웨어 업데이트

이 섹션에는 최신 릴리스 AWS IoT Greengrass V2 정보와 주요 기능 변경 사항 및 중요한 버그 수정 사항이 모두 포함되어 있습니다. 추가적으로 사소한 수정 사항에 대한 자세한 내용은 GitHub의 [aws-greengrass](#) 조직을 참조하세요.

릴리스 노트

- [릴리스: 2025년 12월 23일 AWS IoT Greengrass 코어 v2.16.1 소프트웨어 업데이트](#)

- [릴리스: 2025년 11월 6일 AWS IoT Greengrass 코어 v2.16.0 소프트웨어 업데이트](#)
- [2025년 11월 6일 릴리스: AWS IoT Greengrass Core v2.15.1 소프트웨어 업데이트](#)
- [릴리스: 2025년 7월 3일 AWS IoT Greengrass 코어 v2.15.0 소프트웨어 업데이트](#)
- [2025년 4월 11일 릴리스: AWS IoT Greengrass Core v2.14.3 소프트웨어 업데이트](#)
- [2025년 3월 27일 릴리스: AWS IoT Greengrass Core v2.14.2 소프트웨어 업데이트](#)
- [2025년 2월 7일 릴리스: AWS IoT Greengrass Core v2.14.1 소프트웨어 업데이트](#)
- [릴리스: 2024년 12월 16일 AWS IoT Greengrass 코어 v2.14.0 소프트웨어 업데이트](#)
- [2024년 8월 26일 릴리스: AWS IoT Greengrass Core v2.13.0 소프트웨어 업데이트](#)
- [릴리스: 2024년 5월 24일 AWS IoT Greengrass 코어 v2.12.6 소프트웨어 업데이트](#)
- [2024년 4월 25일 릴리스: AWS IoT Greengrass Core v2.12.5 소프트웨어 업데이트](#)
- [2024년 4월 2일 릴리스: AWS IoT Greengrass Core v2.12.4 소프트웨어 업데이트](#)
- [2024년 3월 27일 릴리스: AWS IoT Greengrass Core v2.12.3 소프트웨어 업데이트](#)
- [2024년 2월 15일 릴리스: AWS IoT Greengrass Core v2.12.2 소프트웨어 업데이트](#)
- [2023년 12월 8일 릴리스: AWS IoT Greengrass Core v2.12.1 소프트웨어 업데이트](#)
- [릴리스: 2023년 11월 7일 AWS IoT Greengrass 코어 v2.12.0 소프트웨어 업데이트](#)
- [2023년 10월 18일 릴리스: AWS IoT Greengrass Core v2.11.3 소프트웨어 업데이트](#)
- [2023년 8월 9일 릴리스: AWS IoT Greengrass Core v2.11.2 소프트웨어 업데이트](#)
- [릴리스: 2023년 7월 21일 AWS IoT Greengrass 코어 v2.11.1 소프트웨어 업데이트](#)
- [2023년 6월 28일 릴리스: AWS IoT Greengrass Core v2.11.0 소프트웨어 업데이트](#)
- [2023년 6월 21일 릴리스: AWS IoT Greengrass Core v2.10.3 소프트웨어 업데이트](#)
- [릴리스: 2023년 6월 5일 AWS IoT Greengrass 코어 v2.10.2 소프트웨어 업데이트](#)
- [릴리스: 2023년 5월 11일 AWS IoT Greengrass 코어 v2.10.1 소프트웨어 업데이트](#)
- [릴리스: 2023년 5월 9일 AWS IoT Greengrass 코어 v2.10.0 소프트웨어 업데이트](#)
- [릴리스: 2023년 4월 20일 AWS IoT Greengrass 코어 v2.9.6 소프트웨어 업데이트](#)
- [릴리스: 2023년 3월 30일 AWS IoT Greengrass 코어 v2.9.5 소프트웨어 업데이트](#)
- [릴리스: 2023년 2월 24일 AWS IoT Greengrass 코어 v2.9.4 소프트웨어 업데이트](#)
- [릴리스: 2023년 2월 1일 AWS IoT Greengrass 코어 v2.9.3 소프트웨어 업데이트](#)
- [릴리스: 2022년 12월 22일 AWS IoT Greengrass 코어 v2.9.2 소프트웨어 업데이트](#)
- [릴리스: 2022년 11월 18일 AWS IoT Greengrass 코어 v2.9.1 소프트웨어 업데이트](#)

- [릴리스: 2022년 11월 15일 AWS IoT Greengrass 코어 v2.9.0 소프트웨어 업데이트](#)
- [2022년 10월 13일 릴리스: AWS IoT Greengrass Core v2.8.1 소프트웨어 업데이트](#)
- [2022년 10월 7일 릴리스: AWS IoT Greengrass Core v2.8.0 소프트웨어 업데이트](#)
- [릴리스: 2022년 7월 28일 AWS IoT Greengrass 코어 v2.7.0 소프트웨어 업데이트](#)
- [2022년 6월 27일 릴리스: AWS IoT Greengrass Core v2.6.0 소프트웨어 업데이트](#)
- [릴리스: 2022년 5월 31일 AWS IoT Greengrass 코어 v2.5.6 소프트웨어 업데이트](#)
- [릴리스: 2022년 4월 6일 AWS IoT Greengrass 코어 v2.5.5 소프트웨어 업데이트](#)
- [릴리스: 2022년 3월 23일 AWS IoT Greengrass 코어 v2.5.4 소프트웨어 업데이트](#)
- [릴리스: 2022년 1월 6일 AWS IoT Greengrass 코어 v2.5.3 소프트웨어 업데이트](#)
- [2021년 12월 3일 릴리스: AWS IoT Greengrass Core v2.5.2 소프트웨어 업데이트](#)
- [2021년 11월 23일 릴리스: AWS IoT Greengrass Core v2.5.1 소프트웨어 업데이트](#)
- [릴리스: 2021년 11월 12일 AWS IoT Greengrass 코어 v2.5.0 소프트웨어 업데이트](#)
- [릴리스: 2021년 8월 3일 AWS IoT Greengrass 코어 v2.4.0 소프트웨어 업데이트](#)
- [2021년 6월 29일 릴리스: AWS IoT Greengrass Core v2.3.0 소프트웨어 업데이트](#)
- [2021년 6월 18일 릴리스: AWS IoT Greengrass Core v2.2.0 소프트웨어 업데이트](#)
- [2021년 4월 26일 릴리스: AWS IoT Greengrass Core v2.1.0 소프트웨어 업데이트](#)
- [릴리스: 2021년 3월 9일 AWS IoT Greengrass 코어 v2.0.5 소프트웨어 업데이트](#)
- [릴리스: 2021년 2월 4일 AWS IoT Greengrass 코어 v2.0.4 소프트웨어 업데이트](#)

릴리스: 2025년 12월 23일 AWS IoT Greengrass 코어 v2.16.1 소프트웨어 업데이트

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.16.1과 Greengrass CLI 구성 요소의 버전 2.16.1을 제공합니다.

릴리스 날짜: 2025년 12월 23일

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 버전 2.16.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 토큰 교환 서비스 실패 후 자격 증명 재시도 간격에 대한 구성을 추가합니다.
Greengrass CLI	Greengrass CLI 버전 2.16.1을 사용할 수 있습니다. Greengrass Nucleus v2.16.1 릴리스용으로 업데이트된 버전입니다.

릴리스: 2025년 11월 6일 AWS IoT Greengrass 코어 v2.16.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.16.0이 제공되고 제공 AWS구성 요소가 업데이트됩니다.

릴리스 날짜: 2025년 11월 6일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 버전 2.16.0을 사용할 수 있습니다. 새로운 기능 - Nucleus는 이제 cgroups V2를 지원합니다. 버그 수정 및 개선 사항 - 배포가 취소될 때 Nucleus가 오래된 배포를 정리하지 않는 문제를 해결합니다.
Greengrass nucleus lite	Greengrass nucleus lite 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 - IoT Core MQTT 권한 부여를 위한 TPM 2.0 사용 지원. - 샘플 Apt 패키지는 이제 Ubuntu 22.04, Ubuntu 24.04, Debian 12, Debian 13 등 더 많은 운영 체제를 지원합니다. - 이제 RestartComponent IPC가 지원됩니다.

구성 요소	세부 정보
	버그 수정 및 개선 사항 - 로컬 배포에는 더 이상 인터넷 액세스가 필요하지 않습니다. - Greengrass Nucleus 런타임 동작과 일치하도록 GetConfiguration이 업데이트되었습니다. (변경 위반) - 일반적인 버그 수정 및 개선 사항.
새도 관리자	새도 관리자 버전 2.3.12을 사용할 수 있습니다. 버그 수정 및 개선 사항 1,024개 이상의 디바이스 새도우가 구성된 경우 배포가 차단되는 문제를 해결합니다.
로그 관리자	로그 관리자 버전 2.3.11을 사용할 수 있습니다. 버그 수정 및 개선 사항 업로드된 로그 파일의 오래된 정보로 인해 Log Manager 런타임 구성이 무기한 증가하는 문제를 해결합니다.
시스템 로그 전달자	시스템 로그 전달자의 버전 2.1.0을 사용할 수 있습니다. 버그 수정 및 개선 사항 - Greengrass nucleus를 올바르게 지원하도록 구성 요소 레시피를 업데이트합니다. - 업로드할 로그가 없는 경우 로깅 출력이 개선되었습니다. - 일반적인 버그 수정 및 개선 사항.

2025년 11월 6일 릴리스: AWS IoT Greengrass Core v2.15.1 소프트웨어 업데이트

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.15.1과 Greengrass CLI 구성 요소의 버전 2.15.1을 제공합니다.

릴리스 날짜: 2025년 11월 6일

릴리스 세부 정보

• [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus 버전 2.15.1	버그 수정 및 개선 사항 부트스트랩 배포 중 30초 종료 제한 시간이 지난 후 Nucleus가 구성 요소 프로세스를 정리하지 못하는 문제를 해결합니다.
Greengrass CLI 버전 2.15.1	Greengrass Nucleus v2.15.1 릴리스용으로 업데이트된 버전입니다.

릴리스: 2025년 7월 3일 AWS IoT Greengrass 코어 v2.15.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.15.0이 제공되고 제공 AWS구성 요소가 업데이트됩니다.

릴리스 날짜: 2025년 7월 3일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 버전 2.15.0을 사용할 수 있습니다. 새로운 기능 • CPU 및 OS 세부 정보와 같은 호스트 시스템 정보를 포함하는 원격 측정 기능을 추가합니다. 자세한 내용은 원격 측정 지표를 참조하세요. • deploymentConfigurationTimeSource 구성을 추가합니다. 자세한 내용은 Greengrass nucleus 구성을 참조하세요.

구성 요소	세부 정보
	버그 수정 및 개선 사항 • 이제 Nucleus는 기본적으로 로컬 버전 및 아티팩트의 사용을 우선시하여, 배포의 일관성을 향상시키고 외부 종속성을 줄입니다. 사용자는 배포 문서에서 다른 기본 설정을 지정하여 이 동작을 재정의할 수 있습니다. • 또한 제공된 Windows 사용자가 Windows에서 일반적으로 허용되는 문자 집합과 일치하지 않을 경우 새로운 경고 로그가 추가됩니다.
Greengrass CLI	Greengrass CLI 버전 2.15.0을 사용할 수 있습니다. 버그 수정 및 개선 사항 • 인증 구성이 변경될 때 구성 요소가 다시 설치되지 않고 계속 다시 시작되도록 수정 사항을 추가합니다. • 일반적인 버그 수정 및 개선 사항.
Greengrass nucleus lite	Greengrass nucleus lite의 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 • 컨테이너 이미지 아티팩트 URI에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 • 일반적인 버그 수정 및 개선 사항.
로그 관리자	로그 관리자의 버전 2.3.10을 사용할 수 있습니다. 새로운 기능 • 로그 업로드 이벤트 세부 정보를 로컬 트랜잭션 로그(config.tlog)에 저장하는 빈도를 제어할 수 있는 새로운 구성 키(updateToTlogIntervalSec)가 추가됩니다. 버그 수정 및 개선 사항 • 소켓 연결 오류 발생 시 CloudWatch 클라이언트를 새로 고치도록 로그 관리자 기능이 개선됩니다.

구성 요소	세부 정보
보안 암호 관리자	보안 암호 관리자의 버전 2.2.6을 사용할 수 있습니다.
	버그 수정 및 개선 사항 TPM(Trusted Platform Module)이 느리거나 응답하지 않아 보안 암호 관리자가 비밀을 가져오지 못하는 문제를 수정합니다.

2025년 4월 11일 릴리스: AWS IoT Greengrass Core v2.14.3 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.14.3이 제공되고 제공 AWS 구성 요소가 업데이트됩니다.

릴리스 날짜: 2025년 4월 11일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 버전 2.14.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 • 토큰 교환 서비스가 포트 구성 변경 시 재시작되도록 허용합니다. • 트리거되지 않은 고정되지 않은 Lambda의 경우, 플릿 상태 서비스가 구성 요소 상태 변경 메시지를 보내지 않는 문제를 수정합니다. • 새 구성 요소 버전 배포 시 구성 요소가 정상적으로 종료되지 않는 문제를 수정합니다. • 내장 플러그인의 주입 후크가 두 번 실행되어 불필요한 수명 주기 이벤트와 중복 로그가 발생하는 문제를 수정합니다. • Windows 디바이스에서 구성 요소 수명 주기 로그를 개선합니다.
Greengrass CLI	Greengrass CLI 버전 2.14.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 • Greengrass nucleus 버전 2.14.3 릴리스용으로 업데이트된 버전입니다.
보안 암호 관리자	보안 암호 관리자의 버전 2.2.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 • 로컬 암호 저장소에 쓰는 빈도를 줄입니다. 이제 보안 암호 관리자는 보안 암호가 업데이트될 때만 로컬 저장소에 기록합니다.
EMQX MQTT 브로커	EMQX MQTT 브로커의 버전 2.0.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 • 경로에 공백이 포함된 경우 Windows에서 EMQX가 시작되지 않는 문제를 수정합니다.

2025년 3월 27일 릴리스: AWS IoT Greengrass Core v2.14.2 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.14.2가 제공되고 제공 AWS 구성 요소가 업데이트됩니다.

릴리스 날짜: 2025년 3월 27일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 버전 2.14.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 • HTTP 클라이언트가 상호 인증으로 구성되지 않는 문제를 수정합니다.
Greengrass CLI	Greengrass CLI 버전 2.14.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 • Greengrass nucleus 버전 2.14.2 릴리스용으로 업데이트된 버전입니다.

2025년 2월 7일 릴리스: AWS IoT Greengrass Core v2.14.1 소프트웨어 업데이트

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.14.1과 제공 AWS 구성 요소에 대한 업데이트를 제공합니다.

릴리스 날짜: 2025년 2월 7일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 버전 2.14.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 • Greengrass를 새로 설치할 때 구성 요소가 제대로 중지되지 않는 문제를 수정합니다.

구성 요소	세부 정보
Greengrass CLI	Greengrass CLI 버전 2.14.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 Greengrass nucleus 버전 2.14.1 릴리스용으로 업데이트된 버전입니다.
보안 터널링	보안 터널링 구성 요소의 버전 1.1.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 Greengrass nucleus lite를 지원하기 위한 구성을 추가합니다.

릴리스: 2024년 12월 16일 AWS IoT Greengrass 코어 v2.14.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.14.0과 새로운 AWS IoT Greengrass nucleus lite 업데이트를 제공합니다. AWS IoT Greengrass nucleus lite는 버전 2에서 AWS IoT Greengrass 사용할 수 있는 새로운 런타임입니다. 이는 메모리 사용량을 줄인 경량 대안으로, 리소스가 제한된 디바이스에 적합합니다. 이는 nucleus 기능의 일부만 구현하며, 향후 릴리스에서 기능 호환성을 점차 확대할 예정입니다. 소스 코드는 현재 [Github](#)에서 확인할 수 있습니다. nucleus lite 런타임을 사용하면 다음을 수행할 수 있습니다.

- Greengrass 코어 디바이스에 구성 요소를 배포할 수 있습니다. 동일한 레시피 형식을 사용하지만 일부 고급 기능은 아직 지원되지 않을 수 있습니다.
- Greengrass 구성 요소로 배포된 애플리케이션은 디바이스 SDKs를 사용하여 AWS IoT Core MQTT 액세스, 로컬 pub/sub 및 Greengrass 구성 액세스와 같은 지원되는 Greengrass IPC APIs에 액세스할 수 있습니다. [지원되는 IPC API](#) 목록은 호환성 차트를 참조하세요.
- nucleus lite 지원을 위해 일부 AWS 관리형 구성 요소가 업데이트되었습니다. 기존 호환 구성 요소 목록은 [AWS제공 구성 요소](#)를 참조하세요.

새로운 기능:

- 메모리 및 디스크 공간 사용량이 더 적습니다(RAM 5MB 미만, 저장소 5MB 미만).
- 구성 요소가 호스트 시스템의 서비스 관리자(systemd, 현재 지원되는 Linux 플랫폼 기준)와 통합됩니다.

주의 사항:

- AWS IoT Greengrass nucleus lite 레시피는 대/소문자를 구분합니다. 정확한 (키) 대소문자 구분이 <https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html> 레시피 참조에 있는 것처럼 사용되었는지 확인합니다.
- nucleus lite 런타임은 사물 그룹 배포를 지원하며, (단일) 코어 디바이스 배포 대상 유형은 아직 지원하지 않습니다. 단일 Greengrass 디바이스에 배포하려면 해당 디바이스만 포함된 사물 그룹을 사용하세요.
- nucleus lite 런타임은 제한된 메모리 리소스를 사용합니다. classic 런타임에서 사용량에 따라 확장되는 기능이 lite에서는 리소스 부족으로 실패할 수 있습니다. 여기에는 한 번에 최대 50개의 MQTT 구독만 허용되는 현행 제한과 레시피 파일 크기 및 배포에 대한 상한이 포함됩니다. 이러한 제한의 일부는 lite 런타임을 직접 컴파일할 경우 컴파일 시점에 구성할 수 있습니다.
- nucleus lite 런타임은 Java를 포함하지 않습니다. Java가 필요한 구성 요소를 사용하려면 시스템에 Java가 설치되어 있어야 하거나, 구성 요소를 통해 Java를 설치해야 합니다.
- nucleus lite 런타임을 소스에서 컴파일하여 시스템에 맞게 자체 빌드를 사용하는 것을 권장합니다. Yocto 시스템의 경우, 시스템 이미지에 nucleus lite 런타임을 통합할 수 있는 계층이 제공됩니다.
- 현재 nucleus lite는 systemd를 사용하는 Linux 시스템 또는 systemd를 사용하는 컨테이너 이미지를 가정합니다.
- 레시피 스크립트로 Docker 컨테이너를 관리할 수 있지만, Greengrass 관리형 컨테이너 아티팩트는 아직 제공되지 않습니다.
- nucleus lite 런타임은 PKCS11 모듈에 저장된 키를 아직 지원하지 않습니다. 사용 사례에 보안 요소에 저장된 키가 필요한 경우, 현재 classic 런타임이 이 사용 사례를 지원할 수 있습니다. 디바이스 자격 증명의 유출을 방지하려면 프로덕션 디바이스가 전체 디스크 암호화를 사용하고 있는지 확인합니다.

nucleus lite 도입과 함께, nucleus v2.14.0도 출시됩니다. 이 업데이트에는 기존 Greengrass nucleus에 대한 주요 개선 사항이 포함되어 있습니다.

주요 기능 및 개선 사항

- 새로운 듀얼 스택 엔드포인트 지원을 통해 IPv6 네트워크 통신이 가능합니다.
- nucleus 재시작 실패 및 디렉터리 손상에 대한 복원력이 강화되었습니다.
- IPC PubSub 구독 종료 시 발생하던 메모리 누수 문제를 수정합니다.

릴리스 날짜: 2024년 12월 16일

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus lite	Greengrass nucleus lite의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 - 메모리 및 디스크 공간 사용량이 더 적습니다(RAM 5MB 미만, 저장소 5MB 미만). - 구성 요소가 호스트 시스템의 서비스 관리자(systemd, 현재 지원되는 Linux 플랫폼 기준)와 통합됩니다.
Greengrass nucleus	Greengrass nucleus 버전 2.14.0을 사용할 수 있습니다. 새로운 기능 - 새로운 듀얼 스택 엔드포인트 지원을 통해 IPv6 네트워크 통신이 가능합니다. - nucleus 재시작 실패 및 디렉터리 손상에 대한 복원력이 강화되었습니다. 버그 수정 및 개선 사항 - IPC PubSub 구독 종료 시 발생하던 메모리 누수 문제를 수정합니다.

구성 요소	세부 정보
	• skipif 조건이 true일 때 시작 시간 초과로 인해 구성 요소가 ERRORED 상태로 진입하는 실행 수명 주기 문제를 수정합니다. • TLS 정책이 TLS13_1_3_2022_10으로 설정된 AWS IoT Core 경우 코어 디바이스가에 연결되지 않는 문제를 해결합니다.
Greengrass CLI	Greengrass CLI 버전 2.14.0을 사용할 수 있습니다. 버그 수정 및 개선 사항 • cli 명령에서 배포 대상 파라미터를 검증합니다.
스트림 관리자	스트림 관리자의 버전 2.14.0을 사용할 수 있습니다. 새로운 기능 • 시작 시간 초과용 새 구성 키를 추가합니다. 기본값은 120초입니다. • Greengrass nucleus lite용 레시피 지원을 추가합니다.
MQTT 5 브로커 (EMQX)	MQTT 5 브로커(EMQX)의 버전 2.0.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 • 클라이언트 디바이스 인증 구성 요소가 준비되기 전에 EMQX가 먼저 시작되는 문제를 수정합니다.
Lambda 런타임 구성 요소	Lambda 런타임 구성 요소의 버전 2.0.9를 사용할 수 있습니다. 버그 수정 및 개선 사항 • Python 3.12를 사용하여 구문 경고 수정
Lambda 관리자 구성 요소	Lambda 관리자 구성 요소의 버전 2.3.5를 사용할 수 있습니다. 버그 수정 및 개선 사항 • 사용 가능한 경우 nio 대신 epoll을 사용하여 성능을 개선합니다.
보안 암호 관리자 구성 요소	Secret Manager 구성 요소의 버전 2.2.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 • 보안 암호 관리자가 부분 arn으로 구성된 보안 암호를 다운로드하지 않는 문제를 수정합니다.

구성 요소	세부 정보
보안 터널링 구성 요소	보안 터널링 구성 요소의 버전 1.1.0을 사용할 수 있습니다. 새로운 기능 Greengrass nucleus lite에 대한 레시피 지원 추가
CloudWatch 지표 구성 요소	CloudWatch 지표 구성 요소의 버전 1.1.0을 사용할 수 있습니다. 새로운 기능 Greengrass nucleus lite에 대한 레시피 지원 추가

2024년 8월 26일 릴리스: AWS IoT Greengrass Core v2.13.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.13.0이 제공됩니다.

릴리스 날짜: 2024년 8월 26일

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 [섹션을 참조하세요](#) [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.13.0을 사용할 수 있습니다. 새로운 기능 - Nucleus에서 FIPS 엔드포인트가 지원됩니다. 자세한 내용은 FIPS 엔드포인트 단원을 참조하십시오. 버그 수정 및 개선 사항 - 배포 개선 사항 취소 - 이제 새 구성이 병합되는 동안과 서비스가 시작될 때까지 기다리는 동안 배포를 취소할 수 있습니다.
스트림 관리자	스트림 관리자 구성 요소의 버전 2.1.13을 사용할 수 있습니다. 버그 수정 및 개선 사항 에서 FIPS 엔드포인트 지원 AWS IoT SiteWise
보안 암호 관리자	보안 암호 관리자 구성 요소의 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 - 새 구성 요소 구성 키를 통해 구성된 보안 암호의 정기 교체 지원에 대한 지원을 추가합니다. - 요청당 보안 암호를 새로 고치기 위해 GetSecretValue IPC 요청에서 새로운 요청 파라미터에 대한 지원을 추가합니다.
IP 감지기	IP 감지기 구성 요소의 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 IPv6 지원을 추가합니다. 이제 로컬 메시징에 IPv6를 사용할 수 있습니다.
클라이언트 디바이스 인증	클라이언트 디바이스 인증의 버전 2.5.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 - 일반 버그 및 수정 사항. - FIPS 엔드포인트가 지원됩니다.
로컬 디버그 콘솔	로컬 디버그 콘솔의 버전 2.4.3을 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 사항 STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH가 바이트/초 대신 Mbps로 잘못 표시되는 문제를 해결합니다.

릴리스: 2024년 5월 24일 AWS IoT Greengrass 코어 v2.12.6 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.6이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2024년 5월 24일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.12.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 Jetson Nano를 포함한 특정 ARMv8 프로세서에서 시작 시 충돌을 일으키는 문제를 해결합니다.
Greengrass CLI	Greengrass CLI 버전 2.12.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 Greengrass nucleus 버전 2.12.6 릴리스용으로 업데이트된 버전입니다.
보안 암호 관리자	보안 암호 관리자의 버전 2.1.8을 사용할 수 있습니다. 버그 수정 및 개선 사항 보안 암호 관리자가 부분 ARN을 수락하지 않는 문제를 해결합니다.

2024년 4월 25일 릴리스: AWS IoT Greengrass Core v2.12.5 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.5가 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2024년 4월 25일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.12.5를 사용할 수 있습니다. 버그 수정 및 개선 사항 - 하드 종속성으로 이전에 손상된 구성 요소를 롤백하는 동안 배포 롤백이 간헐적으로 중단되는 문제를 해결합니다. - 플릿 프로비저닝 후 nucleus가 상태 업데이트를 게시하지 않는 문제를 해결합니다. 404 오류 발생 이후 GetDeploymentConfiguration API에 대한 재시도가 추가됩니다.

2024년 4월 2일 릴리스: AWS IoT Greengrass Core v2.12.4 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.4가 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2024년 4월 2일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.12.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 일부 Linux 디바이스 시작 도중 nucleus가 교착 상태가 되는 문제를 해결합니다.

2024년 3월 27일 릴리스: AWS IoT Greengrass Core v2.12.3 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.3이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2024년 3월 27일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.12.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 - nucleus가 다시 시작된 후 구성 요소 복구 중에 nucleus가 올바른 구성 요소 상태를 보고하지 않는 문제를 해결합니다. - 일반적인 버그 수정 및 개선 사항.
새도 관리자	새도 관리자 구성 요소의 버전 2.3.7을 사용할 수 있습니다. 버그 수정 및 개선 사항 새도 관리자가 새도 관리자 동기화 중에 주기적으로 NullPointerException 오류를 기록하는 문제를 해결합니다.
플릿 프로비저닝	AWS IoT 플릿 프로비저닝 플러그인 버전 1.2.1을 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 사항 Greengrass nucleus 시작 중에 플릿 프로비저닝 플러그인이 오프라인 상태가 되는 문제를 해결합니다. 플릿 프로비저닝 플러그인이 이제 MQTT 연결 직접 호출을 무기한 재시도합니다.
IP 감지기	디스크 스펠러 구성 요소의 버전 2.1.9를 사용할 수 있습니다. 버그 수정 및 개선 사항 디버그 로그 수준에서만 로그를 전송하도록 IP 획득 단계가 조정됩니다.
Moquette MQTT 3.1.1 브로커 구성 요소	Moquette MQTT 3.1.1 브로커 구성 요소의 버전 2.3.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.3.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
로컬 디버그 콘솔	로컬 디버그 콘솔 구성 요소의 버전 2.4.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.2가 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2024년 2월 15일

릴리스 세부 정보

• [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.12.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 - 이전 로그가 제대로 정리되지 않은 문제를 해결합니다. - 일반적인 버그 수정 및 개선 사항.
새도 관리자	새도 관리자 구성 요소의 버전 2.3.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 디바이스가 오프라인 상태인 동안 AWS 클라우드 업데이트를 통해 삭제되는 새도 속성이 재연결 후에도 로컬 새도에 계속 존재하는 문제를 해결합니다.
Lambda 런처	Lambda 런처 구성 요소의 버전 2.0.13을 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
디스크 스펠러	디스크 스펠러 구성 요소 의 버전 1.0.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 데이터베이스 연결을 재사용하여 성능을 개선합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.1이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2023년 12월 8일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass

코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.12.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 nucleus가 배포 주제에 대한 MQTT 구독을 복제하여 추가 로깅 및 MQTT 게시가 발생할 수 있는 문제를 해결합니다.
클라이언트 디바이스 인증	클라이언트 디바이스 인증 구성 요소의 버전 2.4.5를 사용할 수 있습니다. 새로운 기능 selectionRule 파라미터를 사용하여 사물 이름을 선택할 수 있도록 와 일드카드 접두사에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 경우에 따라 인증서가 새 연결 정보로 업데이트되지 않는 문제를 해결합니다.
디스크 스펠러	디스크 스펠러 구성 요소의 버전 1.0.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 특정 사례에서 MQTT 메시지 형식이 지속되지 않은 문제를 해결합니다.
MQTT 브리지	디스크 스펠러 구성 요소의 버전 2.3.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 로컬 MQTT 클라이언트에서 연결 해제 루프가 발생하는 문제를 해결합니다.
스트림 관리자	스트림 관리자 구성 요소의 버전 2.1.12를 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 사항
	AWS 서비스 요청에 Greengrass 자격 증명이 선호되도록 자격 증명이 사용되는 순서를 업데이트합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.12.0이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2023년 11월 7일

릴리스 하이라이트

- 롤백 시 부트스트랩 - AWS IoT Greengrass 이제 라는 Greengrass nucleus 구성 파라미터를 제공합니다BootstrapOnRollback. 이 기능을 사용하면 롤백 배포의 일부로 부트스트랩 수명 주기 단계를 실행할 수 있습니다.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

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

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.11.3이 제공됩니다.

릴리스 날짜: 2023년 10월 18일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.11.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 - 종속성이 실패할 때 구성 요소가 부적절하게 시작될 수 있는 nucleus 문제를 해결합니다. 새로운 기능 - 구성 가능한 S3 엔드포인트 유형이 추가됩니다.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.3.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 특정 오류에 대한 로그 수준이 조정됩니다.
로컬 디버그 콘솔	Lambda 관리자 구성 요소의 버전 2.4.0을 사용할 수 있습니다. 새로운 기능 스트림 관리자 디버깅 콘솔이 추가됩니다.
로그 관리자	로그 관리자 구성 요소의 버전 2.3.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 특정 오류에 대한 로그 수준이 조정됩니다.
새도 관리자	새도 관리자 구성 요소의 버전 2.3.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 null 및 빈 새도 상태 문서에 대한 지원을 추가합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.11.2가 제공됩니다.

릴리스 날짜: 2023년 8월 9일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.11.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 • 많은 수(50개 초과)의 구독이 사용 중일 때 오프라인으로 표시될 수 있는 nucleus MQTT 5 클라이언트의 문제를 해결합니다. • Docker 다이얼 TCP 실패에 대한 재시도가 추가됩니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.11.1이 제공됩니다.

릴리스 날짜: 2023년 7월 21일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.11.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 • 부트스트랩 작업이 실패하고 배포 메타데이터 파일이 손상된 경우 nucleus가 시작되지 않는 문제를 해결합니다. • 온디맨드 Lambda 구성 요소가 배포 상태 업데이트에 보고되지 않는 문제를 해결합니다.

구성 요소	세부 정보
	중복 권한 부여 정책 ID에 대한 지원을 추가합니다.
Lambda 관리자	Lambda 관리자의 버전 2.2.11을 사용할 수 있습니다. 버그 수정 및 개선 사항 Lambda 구성이 변경될 때 LegacySubscriptionRouter 구성이 업데이트되지 않는 문제를 해결합니다.

2023년 6월 28일 릴리스: AWS IoT Greengrass Core v2.11.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.11.0이 제공됩니다.

릴리스 날짜: 2023년 6월 28일

릴리스 하이라이트

- 영구 디스크 스펠러 - AWS IoT Greengrass 이제는 Greengrass 코어 디바이스에서 스펠링된 메시지에 대한 영구 스펠러 구현을 제공합니다 AWS IoT Core. 이 구성 요소에서는 이러한 아웃바운드 메시지를 디스크에 저장합니다. 자세한 내용은 [디스크 스펠러](#) 단원을 참조하십시오.
- 로컬 배포 개선 - 이제 로컬 배포를 취소하고, 배포 실패 처리 정책을 설정하고, 구체적인 배포 상태를 가져올 수 있습니다.
- 로깅 속도 개선 - 로그 관리자 구성 요소의 로그 업로드 속도가 개선되었습니다.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.11.0을 사용할 수 있습니다. 새로운 기능 - 로컬 배포를 취소할 수 있습니다. - 로컬 배포 실패 처리 정책을 구성할 수 있습니다. - 디스크 스펠러 플러그인에 대한 지원을 추가합니다.
Greengrass CLI	Greengrass CLI 버전 2.11.0을 사용할 수 있습니다. 새로운 기능 - 로컬 배포를 취소할 수 있습니다. - 로컬 배포 실패 처리 정책을 구성할 수 있습니다. - 자세한 배포 상태 보고가 개선됩니다.
디스크 스펠러	디스크 스펠러 구성 요소의 버전 1.0.0을 사용할 수 있습니다. 디스크 스펠러 구성 요소는 Greengrass 코어 디바이스에서 로 전송된 메시지의 영구 스토리지를 제공합니다 AWS IoT Core.
로그 관리자	로그 관리자 구성 요소의 버전 2.3.5를 사용할 수 있습니다.

구성 요소	세부 정보
	개선 사항
	로그 업로드 속도가 개선됩니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.10.3이 제공됩니다.

릴리스 날짜: 2023년 6월 21일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.10.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 PKCS#11 공급자를 사용할 때 Greengrass가 배포 알림을 구독하지 않는 문제를 해결합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.10.2가 제공됩니다.

릴리스 날짜: 2023년 6월 5일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.10.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 - 구성 요소 수명 주기의 대소문자를 구분하지 않는 구문 분석이 허용됩니다. - 환경 PATH 변수가 올바르게 다시 생성되지 않은 문제를 해결합니다. - 특수 문자가 있는 사용자 이름의 스트림 관리자를 포함한 구성 요소에 대한 프록시 URI 인코딩이 수정됩니다.
클라이언트 디바이스 인증	클라이언트 디바이스 인증 구성 요소의 버전 2.4.2를 사용할 수 있습니다. 새로운 기능 새로운 <code>startupTimeoutSeconds</code> 구성 옵션이 추가됩니다.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.2.9를 사용할 수 있습니다. 버그 수정 및 개선 사항 왜곡된 클럭으로 인해 포트 번호가 손상되는 문제를 해결합니다.
로그 관리자	로그 관리자 구성 요소의 버전 2.3.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 <code>periodicUploadIntervalSec</code> 파라미터를 분수 값으로 설정할 수 있도록 지원을 추가합니다. 최소값은 1마이크로초입니다. - 로그 관리자가 CloudWatch <code>putLogEvents</code> 제한을 준수하지 않는 문제를 해결합니다.
MQTT 3.1 브로커(Moquette)	MQTT 3.1 브로커(Moquette) 구성 요소의 버전 2.3.3을 사용할 수 있습니다. 새로운 기능 새로운 <code>startupTimeoutSeconds</code> 구성 옵션이 추가됩니다.
MQTT 브리지	MQTT 브리지 구성 요소의 버전 2.2.6을 사용할 수 있습니다.

구성 요소	세부 정보
	새로운 기능 새로운 <code>startupTimeoutSeconds</code> 구성 옵션이 추가됩니다.
스트림 관리자	스트림 관리자 구성 요소의 버전 2.1.7을 사용할 수 있습니다. 버그 수정 및 개선 사항 스트림 관리자가 프록시 구성을 올바르게 읽지 못하는 문제를 해결합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.10.1이 제공됩니다.

릴리스 날짜: 2023년 5월 11일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass

코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.10.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 - Jetson Nano를 포함한 특정 ARMv8 프로세서에서 시작 시 충돌을 일으키는 문제를 해결합니다. - Greengrass가 구성 요소의 표준에서 더 이상 종료되지 않고 동작을 2.10.0 이전 동작으로 되돌립니다.
스트림 관리자	새로운 스트림 관리자의 버전 2.1.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 Jetson Nano를 포함한 특정 ARMv8 프로세서에서 시작 시 충돌을 일으키는 문제를 해결합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.10.0이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2023년 5월 9일

릴리스 하이라이트

- MQTT5 지원 - AWS IoT Greengrass 이제는 MQTT5 AWS IoT Core 를 사용하여에서 메시지 전송 및 수신을 지원합니다. 자세한 내용은 [AWS IoT Core MQTT 메시지 게시](#)를 참조하세요.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.10.0을 사용할 수 있습니다. 새로운 기능 - 빈 정규식에 대한 interpolateComponentConfiguration 지원이 추가됩니다. 이제 Greengrass가 루트 구성 객체에서 보간합니다. - MQTT5에 대한 지원을 추가합니다. - 스캔 없이 플러그인 구성 요소를 빠르게 로드하는 메커니즘이 추가됩니다. - 사용하지 않는 Docker 이미지를 삭제하여 Greengrass가 디스크 공간을 절약할 수 있습니다. 버그 수정 및 개선 사항 - 롤백이 배포에서 특정 구성 값을 그대로 두는 문제를 해결합니다. - Greengrass nucleus가 사용자 지정 비AWS 자격 증명 및 데이터 엔드포인트의 AWS 도메인 시퀀스를 검증하는 문제를 해결합니다. - 다중 그룹 종속성 확인을 업데이트하여 활성 버전으로 잠그는 대신 AWS 클라우드 협상을 통해 모든 그룹 종속성을 다시 해결합니다. 이 업데이트

구성 요소	세부 정보
	를 통해 배포 오류 코드 <code>INSTALLED_COMPONENT_NOT_FOUND</code> 도 제거됩니다. • Greengrass nucleus가 업데이트되어 로컬에 이미 있는 Docker 이미지의 다운로드를 건너뛵니다. • Greengrass nucleus가 업데이트되어 제한 시간이 만료되기 전에 구성 요소 설치 단계를 다시 시작합니다. • 사소한 추가 수정 및 개선 사항.
새도 관리자	새로운 새도 관리자의 버전 2.3.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 로컬 새도 데이터베이스가 손상되었을 때 새도 관리자가 BROKEN 상태가 되는 문제를 해결합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.6이 제공됩니다.

릴리스 날짜: 2023년 4월 20일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디

바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.9.6을 사용할 수 있습니다. 버그 수정 및 개선 사항 LAUNCH_DIRECTORY_CORRUPTED 오류로 인해 Greengrass 배포가 실패하고 후속 디바이스 재부팅에서 Greengrass가 시작되지 않는 문제를 해결합니다. 이 오류는 Greengrass 디바이스를 Greengrass를 다시 시작해야 하는 배포가 있는 여러 사물 그룹 간에 이동할 때 발생할 수 있습니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.5가 제공됩니다.

릴리스 날짜: 2023년 3월 30일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.9.5를 사용할 수 있습니다. 새로운 기능 - Greengrass nucleus 소프트웨어 서명 확인에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - 로컬 레시피 메타데이터 리전이 Greengrass nucleus 시작 리전과 일치하지 않을 때 배포가 실패하는 문제를 해결합니다. 이러한 일이 발생할 때 Greengrass nucleus는 이제 클라우드와 재협상합니다. - MQTT 메시지 스펠러가 메시지를 채우고 메시지가 제거되지 않는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.4가 제공됩니다.

릴리스 날짜: 2023년 2월 24일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.9.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 • QOS 0 메시지를 삭제하기 전에 null 메시지를 확인합니다. • 1,024자 제한을 초과하는 경우 작업 상태 세부 정보 값이 잘립니다. • Windows용 부트스트랩 스크립트가 업데이트되어 해당 경로에 공백이 포함된 경우 Greengrass 루트 경로를 올바르게 읽습니다. • 구독 응답이 전송되지 않은 경우 클라이언트 메시지가 삭제 AWS IoT Core 되도록 구독을 업데이트합니다. • 기본 구성 파일이 손상되거나 누락된 경우 nucleus가 백업 파일에서 구성을 로드하게 됩니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.3이 제공됩니다.

릴리스 날짜: 2023년 2월 1일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 에서 제공하는 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.9.3을 사용할 수 있습니다. 버그 수정 및 개선 사항 • MQTT 클라이언트 ID가 중복되지 않게 됩니다. • 더 강력한 파일 읽기 및 쓰기 기능이 추가되어 손상 방지와 복구가 가능합니다.

구성 요소	세부 정보
	- 특정 네트워크 관련 오류 시 Docker 이미지 가져오기를 재시도합니다. - MQTT 연결에 대해 noProxyAddresses 옵션이 추가됩니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.2가 제공됩니다.

릴리스 날짜: 2022년 12월 22일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 의 버전 2.9.2를 사용할 수 있습니다.

구성 요소	세부 정보
	버그 수정 및 개선 사항 • interpolateComponentConfiguration 구성이 진행 중인 배포에 적용되지 않는 문제를 해결합니다. • OSHI를 사용하여 모든 하위 프로세스가 나열됩니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.1이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2022년 11월 18일

릴리스 하이라이트

- 로그 관리자 - 이제 로그 관리자가 새 파일이 교체될 때까지 기다리는 대신 활성 로그 파일을 처리하고 직접 업로드합니다. 이 개선 사항으로 로그 지연이 크게 감소합니다. 자세한 내용은 [로그 관리](#) [자](#)를 참조하세요.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.9.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 배포가 플러그인 구성 요소를 제거하는 경우 Greengrass가 다시 시작되는 수정 사항이 추가됩니다.
로그 관리자	새로운 로그 관리자의 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 기다리는 대신 활성 로그 파일을 처리하고 직접 업로드하여 로그 지연이 감소합니다. 버그 수정 및 개선 사항 - 고유한 이름으로 파일을 교체할 때 로그 교체 지원이 개선됩니다. - 사소한 추가 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.9.0이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2022년 11월 15일

릴리스 하이라이트

- 오프라인 인증 - AWS IoT Greengrass 이제에서 오프라인 인증을 지원합니다. AWS IoT Greengrass 코어 디바이스가 클라우드에 연결되지 않은 경우에도 클라이언트 디바이스가 코어 디바이스에 연결할 수 있도록 코어 디바이스를 구성할 수 있습니다. 자세한 내용은 [오프라인 인증](#)을 참조하세요.
- 하위 배포 - 이제 하위 배포를 생성할 수 있습니다. 하위 배포를 사용하여 실패한 배포를 확인할 수 있습니다. 각 하위 배포가 더 작은 디바이스 하위 집합에서 실패한 배포의 서로 다른 구성을 테스트할 수 있습니다. 자세한 내용은 [하위 배포 생성](#)을 참조하세요.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 의 버전 2.9.0을 사용할 수 있습니다.

구성 요소	세부 정보
	새로운 기능 • 더 작은 디바이스 하위 집합으로 배포를 재시도하는 하위 배포 생성 기능이 추가됩니다. 이 기능을 사용하면 보다 효율적으로 실패한 배포를 테스트하고 해결할 수 있습니다. 버그 수정 및 개선 사항 • useradd, groupadd 및 usermod가 없는 시스템에 대한 지원이 개선됩니다. • 사소한 추가 수정 및 개선 사항.
클라이언트 디바이스 인증	클라이언트 디바이스 인증 구성 요소의 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 • 클라이언트 디바이스의 오프라인 인증에 대한 지원을 추가합니다. 이 기능을 사용하면 코어 디바이스가 인터넷에 연결되지 않을 때 클라이언트 디바이스가 코어 디바이스에 계속 연결될 수 있습니다. • 고객 제공 CA(인증 기관)에 대한 지원을 추가합니다. 코어 디바이스는 고객이 제공한 CA를 루트 인증서로 사용하여 MQTT 브로커 인증서를 생성합니다.
MQTT 5 브로커 (EMQX)	MQTT 5 브로커(EMQX) 구성 요소의 버전 1.2.0을 사용할 수 있습니다. 새로운 기능 인증서 체인에 대한 지원을 추가합니다.
Moquette MQTT 브로커	새로운 Moquette MQTT 브로커 구성 요소의 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 인증서 체인에 대한 지원을 추가합니다.
보안 암호 관리자	보안 암호 관리자의 새로운 버전 2.1.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 보안 암호 관리자가 배포되고 Greengrass nucleus가 다시 시작될 때 캐시된 보안 암호가 제거되는 문제를 해결합니다.

구성 요소	세부 정보
스트림 관리자	새로운 스트림 관리자의 버전 2.1.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 영어 이외의 언어를 사용하는 Windows OS 문제를 해결합니다.

2022년 10월 13일 릴리스: AWS IoT Greengrass Core v2.8.1 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.8.1이 제공됩니다.

릴리스 날짜: 2022년 10월 13일

Note

Greengrass nucleus 버전 2.8.0을 사용하는 경우 Greengrass nucleus 버전 2.8.1로 업그레이드하는 것이 좋습니다.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass

코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.8.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 • Greengrass API 오류에서 배포 오류 코드가 올바르게 생성되지 않은 문제를 해결합니다. • 배포 중에 구성 요소가 ERRORED 상태에 도달할 때 플릿 상태 업데이트가 부정확한 정보를 전송하는 문제를 해결합니다. • Greengrass에 기존 구독이 50개보다 많을 때 배포를 완료할 수 없는 문제를 해결합니다.

2022년 10월 7일 릴리스: AWS IoT Greengrass Core v2.8.0 소프트웨어 업데이트

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.8.0과 MQTT 5 브로커(EMQX) 구성 요소의 버전 1.1.0이 제공됩니다.

릴리스 날짜: 2022년 10월 7일

릴리스 하이라이트

- 배포 오류 코드 - Greengrass nucleus에서 이제 구성 요소 배포를 완료할 수 없는 경우 자세한 오류 코드가 포함된 [배포 상태 응답](#)을 보고합니다. 자세한 내용은 [세부 배포 오류 코드](#) 단원을 참조하십시오.
- 구성 요소 오류 상태 - Greengrass nucleus에서 이제 구성 요소가 BROKEN 또는 ERRORED 상태가 될 때 자세한 오류 상태가 포함된 [구성 요소 상태](#) 응답을 보고합니다. 자세한 내용은 [세부 구성 요소 상태 코드](#) 단원을 참조하십시오.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.8.0을 사용할 수 있습니다. 새로운 기능 - 코어 디바이스에 구성 요소를 배포하는 데 문제가 있는 경우 자세한 오류 코드가 포함된 배포 상태 응답을 보고하도록 Greengrass nucleus가 업데이트됩니다. 자세한 내용은 세부 배포 오류 코드 단원을 참조하십시오. - 구성 요소가 BROKEN 또는 ERRORED 상태가 될 때 자세한 오류 코드가 포함된 구성 요소 상태 응답을 보고하도록 Greengrass nucleus가 업데이트됩니다. 자세한 내용은 세부 구성 요소 상태 코드 단원을 참조하십시오. - 상태 메시지 필드가 확장되어 디바이스의 클라우드 가용성 정보가 개선됩니다. - 플릿 상태 서비스 견고성이 개선됩니다. 버그 수정 및 개선 사항 - 구성이 변경될 때 손상된 구성 요소를 다시 설치할 수 있습니다. - 부트스트랩 배포 중 nucleus 재시작으로 인해 배포가 실패하는 문제를 해결합니다.

구성 요소	세부 정보
	- Windows에서 루트 경로에 공백이 있을 때 설치가 실패하는 문제를 해결합니다. - 배포 중에 구성 요소가 종료되고 새 버전의 종료 스크립트를 사용하는 문제를 해결합니다. - 다양한 종료 개선 사항. - 사소한 추가 수정 및 개선 사항.
MQTT 5 브로커 (EMQX)	MQTT 5 브로커(EMQX) 구성 요소의 버전 1.1.0을 사용할 수 있습니다. 새로운 기능 - 브로커 옵션 및 플러그인을 포함한 EMQX 구성에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - EMQX가 버전 4.4.9로 업데이트됩니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.7.0, 스트림 관리자 구성 요소의 버전 2.1.0, Lambda 관리자 구성 요소의 버전 2.2.5가 제공됩니다.

릴리스 날짜: 2022년 7월 28일

릴리스 하이라이트

- 스트림 관리자 원격 측정 지표 - 스트림 관리자는 이제 Amazon EventBridge에 원격 측정 지표를 자동으로 보내므로 코어 디바이스가 업로드하는 데이터의 볼륨을 모니터링 및 분석하는 클라우드 애플리케이션을 생성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집](#) 단원을 참조하십시오.
- 사용자 지정 인증 기관(CA) - CA가 등록되지 않은 사용자 지정 인증서 CA로 서명된 클라이언트 인증서가 이제 지원 AWS IoT됩니다. 자세한 내용은 [프라이빗 CA에서 서명한 디바이스 인증서 사용](#) 단원을 참조하십시오.

릴리스 세부 정보

• [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.7.0을 사용할 수 있습니다. 새로운 기능 - 코어 디바이스가 로컬 배포를 적용할 때 AWS IoT Greengrass 클라우드로 상태 업데이트를 보내도록 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 iotdata로 설정할 수 있습니다. 자세한 내용은 프라이빗 CA에서 서명한 디바이스 인증서 사용 단원을 참조하십시오. 버그 수정 및 개선 사항 - nucleus가 중지되거나 다시 시작될 때 Greengrass nucleus가 특정 시나리오에서 배포를 롤백하는 문제를 해결합니다. 이제 nucleus가 다시 시작된 후 nucleus가 배포를 재개합니다.

구성 요소	세부 정보
	• 소프트웨어를 시스템 서비스로 설정하도록 지정할 때 <code>--start</code> 인수를 준수하도록 Greengrass 설치 관리자가 업데이트됩니다. • nucleus가 구성 요소를 업데이트한 이벤트에서 배포 ID를 설정하도록 SubscribeToComponentUpdates의 동작이 업데이트됩니다. • 사소한 추가 수정 및 개선 사항.
스트림 관리자	스트림 관리자 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • 원격 측정 지표를 Amazon EventBridge에 자동으로 전송하도록 이 구성 요소가 업데이트됩니다. 자세한 내용은 AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집 단원을 참조하십시오. 이 기능은 Greengrass nucleus 구성 요소의 v2.7.0 이상에서 사용할 수 있습니다. • Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.2.5를 사용할 수 있습니다. 새로운 기능 • 로컬 게시/구독 메시지를 구독하는 이벤트 소스에서 MQTT 주제 와일드 카드에 대한 지원을 추가합니다. 이 기능을 사용하려면 Greengrass nucleus 구성 요소의 v2.6.0 이상이 필요합니다. • Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.

2022년 6월 27일 릴리스: AWS IoT Greengrass Core v2.6.0 소프트웨어 업데이트

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.6.0, 새로운 AWS제공 구성 요소 및 AWS제공 구성 요소에 대한 업데이트를 제공합니다.

릴리스 날짜: 2022년 6월 27일

릴리스 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](#)을(를) 참조하세요.
- 클라이언트 디바이스 새도 지원 - 이제 사용자 지정 구성 요소의 클라이언트 디바이스 새도와 상호 작용하고 클라이언트 디바이스 새도를 AWS IoT Core와 동기화할 수 있습니다. 자세한 내용은 [클라이언트 디바이스 새도와 상호 작용 및 동기화](#) 단원을 참조하십시오.
- 클라이언트 디바이스에 대한 로컬 MQTT 5 지원 - 이제 EMQX MQTT 5 브로커를 배포하여 클라이언트 디바이스와 코어 디바이스 간의 통신에서 MQTT 5 기능을 사용할 수 있습니다. 자세한 내용은 [MQTT 5 브로커\(EMQX\)](#) 및 [코어 디바이스에 클라이언트 디바이스 연결](#)을(를) 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 업데이트에서 레시피 변수 사용](#)을(를) 참조하세요.
- IPC 권한 부여 정책의 와일드카드 - 이제 * 와일드카드를 사용하여 IPC(프로세스 간 통신) 권한 부여 정책의 문자 조합과 일치시킬 수 있습니다. 이 와일드카드를 사용하면 단일 권한 부여 정책에서 여러 리소스에 대한 액세스를 허용할 수 있습니다. 자세한 내용은 [권한 부여 정책의 와일드카드](#) 단원을 참조하십시오.
- 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 관리](#)를 참조하세요.
- 클라이언트 디바이스를 인증 및 권한을 부여하는 IPC 작업 - 이제 이러한 작업을 사용하여 사용자 지정 로컬 브로커 구성 요소를 생성할 수 있습니다. 자세한 내용은 [IPC: 클라이언트 디바이스 인증 및 권한 부여](#)를 참조하세요.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.6.0을 사용할 수 있습니다. 새로운 기능 - 로컬 게시/구독 주제를 구독할 때 MQTT 와일드카드에 대한 지원을 추가합니다. 자세한 내용은 로컬 메시지 게시/구독 및 SubscribeToTopic을(를) 참조하세요. - 구성 요소 구성에서 <i>component_dependency_name</i> :configuration: <i>json_pointer</i> 레시피 변수 이외의 레시피 변수에 대한 지원을 추가합니다. 레시피에서 구성 요소의 DefaultConfiguration 을 정의하거나 배포에서 구성 요소를 구성할 때 이러한 레시피 변수를 사용할 수 있습니다. 이 기능을 활성화하려면 interpolateComponentConfiguration 구성 옵션을 true로 설정합니다. 자세한 내용은 레시피 변수 및 병합 업데이트에서 레시피 변수 사용을(를) 참조하세요. - IPC(프로세스 간 통신) 권한 부여 정책의 * 와일드카드에 대한 전체 지원이 추가됩니다. 이제 모든 문자 조합과 일치하도록 리소스 문자열에 * 문자를 지정할 수 있습니다. 자세한 내용은 권한 부여 정책의 와일드카드 단원을 참조하십시오. - 사용자 지정 구성 요소에서 Greengrass CLI가 사용하는 IPC 작업을 직접 호출할 수 있도록 지원을 추가합니다. 이러한 IPC 작업을 사용하여 로컬 배포를 관리하고, 구성 요소 세부 정보를 보고, 로컬 디버그 콘솔에 로

구성 요소	세부 정보
	그인하는 데 사용할 수 있는 암호를 생성할 수 있습니다. 자세한 내용은 IPC: 로컬 배포 및 구성 요소 관리를 참조하세요. 버그 수정 및 개선 사항 - 특정 시나리오에서 하드 종속성이 다시 시작되거나 상태가 변경될 때 종속 구성 요소가 반응하지 않는 문제를 해결합니다. - 배포 실패 시 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 보고하는 오류 메시지를 개선합니다. - nucleus가 다시 시작되는 특정 시나리오에서 Greengrass nucleus가 사물 배포를 두 번 적용하는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.
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 사용하는 코어 디바이스와 통신할 수 있습니다.

구성 요소	세부 정보
새도 관리자	새도 관리자 구성 요소의 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 - 로컬 게시/구독 인터페이스를 통해 로컬 새도 서비스에 대한 지원을 추가합니다. 이제 새도 MQTT 주제에 대해 로컬 게시/구독 메시지 브로커와 통신하여 코어 디바이스에서 새도를 가져오고 업데이트하고 삭제할 수 있습니다. 이 기능을 사용하면 MQTT 브리지를 사용하여 클라이언트 디바이스를 로컬 새도 서비스에 연결하고 클라이언트 디바이스와 로컬 게시/구독 인터페이스 간에 새도 주제에 대한 메시지를 릴레이할 수 있습니다. 이 기능을 사용하려면 Greengrass nucleus 구성 요소의 v2.6.0 이상이 필요합니다. 클라이언트 디바이스를 로컬 새도 서비스에 연결하려면 MQTT 브리지 구성 요소 v2.2.0 이상을 사용해야 합니다. - 로컬 새도 서비스와 AWS 클라우드간에 새도를 동기화하는 방향을 사용자 지정하도록 구성할 수 있는 <code>direction</code> 옵션을 추가합니다. AWS 클라우드에 대한 대역폭 및 연결을 줄이도록 이 옵션을 구성할 수 있습니다.
클라이언트 디바이스 인증	클라이언트 디바이스 인증 구성 요소의 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 - 사용자 지정 구성 요소에 대한 지원이 추가되어 IPC(프로세스 간 통신) 작업을 직접적으로 호출하여 클라이언트 디바이스를 인증하고 권한을 부여합니다. 예를 들어 사용자 지정 MQTT 브로커 구성 요소에서 이러한 작업을 사용할 수 있습니다. 자세한 내용은 IPC: 클라이언트 디바이스 인증 및 권한 부여를 참조하세요. <code>maxActiveAuthTokens</code>, <code>cloudQueueSize</code> 및 <code>threadPoolSize</code> 옵션이 추가되어 이를 구성하면 이 구성 요소의 성능을 조정할 수 있습니다.

구성 요소	세부 정보
MQTT 브리지	MQTT 브리지 구성 요소의 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 Greengrass nucleus 구성 요소의 v2.6.0 이상이 필요합니다. - 메시지를 릴레이할 때 대상 주제에 접두사를 추가하도록 MQTT 브리지를 구성하기 위해 지정할 수 있는 targetTopicPrefix 옵션이 추가됩니다.
Greengrass CLI	Greengrass CLI 버전 2.6.0을 사용할 수 있습니다. 새로운 기능 - 사용자 지정 구성 요소에서 Greengrass CLI가 사용하는 IPC(프로세스 간 통신) 작업을 직접 호출할 수 있도록 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 참조하세요. 버그 수정 및 개선 사항 - 사소한 추가 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.5.6과 로그 관리자 구성 요소의 버전 2.2.4가 제공됩니다.

릴리스 날짜: 2022년 5월 31일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.5.6을 사용할 수 있습니다. 새로운 기능 - ECC 키를 사용하는 하드웨어 보안 모듈에 대한 지원을 추가합니다. HSM(하드웨어 보안 모듈)을 사용하여 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 내용은 하드웨어 보안 통합 단원을 참조하십시오. 버그 수정 및 개선 사항 - 특정 시나리오에서 설치 스크립트가 손상된 구성 요소를 배포할 때 배포가 완료되지 않는 문제를 해결합니다. - 시작 중 성능이 개선됩니다. - 사소한 추가 수정 및 개선 사항.
로그 관리자	로그 관리자 구성 요소의 버전 2.2.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 잘못된 구성을 처리할 때의 안정성이 개선됩니다.

구성 요소	세부 정보
	사소한 추가 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.5.5가 제공됩니다.

릴리스 날짜: 2022년 4월 6일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus 의 버전 2.5.5를 사용할 수 있습니다.

구성 요소	세부 정보
	새로운 기능 - 구성 요소에 대해 GG_ROOT_CA_PATH 환경 변수가 추가되어 사용자 지정 구성 요소의 루트 CA(인증 기관) 인증서에 액세스할 수 있습니다. 버그 수정 및 개선 사항 - 영어 이외의 표시 언어를 사용하는 Windows 디바이스에 대한 지원을 추가합니다. - Greengrass nucleus가 부울 설치 관리자 인수를 구문 분석하는 방법이 업데이트되어 부울 값 없이 부울 인수를 지정하여 true 값을 지정할 수 있습니다. 예를 들어 이제 --provision true 대신 --provision 을 지정하여 자동 리소스 프로비저닝을 사용하여 설치할 수 있습니다. - 또한 특정 시나리오에서 프로비저닝 후 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 상태를 보고하지 않는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.5.4와 Lambda 런처 구성 요소의 버전 2.0.10이 제공됩니다.

릴리스 날짜: 2022년 3월 23일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.5.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
Lambda 런처	Lambda 런처 구성 요소의 버전 2.0.10을 사용할 수 있습니다. 버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.5.3과 새로운 PKCS#11 공급자 구성 요소가 제공됩니다.

릴리스 날짜: 2022년 1월 6일

릴리스 하이라이트

- 하드웨어 보안 통합 - 이제 하드웨어 보안 모듈(HSM)에 안전하게 저장하는 프라이빗 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.5.3을 사용할 수 있습니다. 새로운 기능 하드웨어 보안 통합에 대한 지원을 추가합니다. HSM(하드웨어 보안 모듈)을 사용하여 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 내용은 하드웨어 보안 통합 단원을 참조하십시오.

구성 요소	세부 정보
	버그 수정 및 개선 사항 nucleus가 AWS IoT Core와의 MQTT 연결을 설정하는 동안 런타임 예외 문제를 해결합니다.
PKCS#11 공급자	PKCS#11 공급자 구성 요소 의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 하드웨어 보안 통합에 대한 지원을 추가합니다. HSM(하드웨어 보안 모듈)을 사용하여 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 내용은 하드웨어 보안 통합 단원을 참조하십시오.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.5.2가 제공됩니다.

릴리스 날짜: 2021년 12월 3일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass

코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.5.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 Greengrass nucleus가 업데이트된 후 Windows 서비스가 중지하거나 디바이스를 재부팅한 후 다시 시작되지 않는 문제를 해결합니다.
AWS IoT Device Defender	AWS IoT Device Defender 구성 요소의 버전 3.0.1을 사용할 수 있습니다. AWS IoT Device Defender 구성 요소의 이 버전에는 버전 2.x와 다른 구성 파라미터가 필요합니다. 버전 2.x에 기본값이 아닌 구성을 사용하고 v2.x에서 v3.x로 업그레이드하려면 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 AWS IoT Device Defender 구성 요소 구성을 참조하세요. 새로운 기능 - Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. - 구성 요소 유형을 Lambda 구성 요소에서 일반 구성 요소로 변경합니다. 이 구성 요소에서는 이제 구독 생성이 레거시 구독 라우터 구성 요소에 종속되지 않습니다. - 구성 요소 종속성이 설치되는 설치 스크립트를 선택적으로 비활성화할 수 있는 새 UseInstaller 구성 파라미터를 추가합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.5.1이 제공됩니다.

릴리스 날짜: 2021년 11월 23일

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.5.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 - Windows에서 Java 런타임 환경(JRE)의 32비트 버전에 대한 지원을 추가합니다. - AWS IoT 정책이 <code>greengrass:ListThingGroupsForCoreDevice</code> 권한을 부여하지 않는 코어 디바이스에 대한 사물 그룹 제거 동작이 변경됩니다. 이 버전을 사용하면 배포가 계속되고, 경고가 기록되며, 사물 그룹에서 코어 디바이스를 제거할 때 구성 요소가 제거되지 않습니다. 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포 단원을 참조하십시오. - Greengrass 구성 요소 프로세스에 Greengrass nucleus가 제공되도록 시스템 환경 변수 관련 문제를 해결합니다. 이제 구성 요소를 다시 시작하여 최신 시스템 환경 변수를 사용할 수 있습니다.

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

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.0, 새로운 AWS제공 구성 요소 및 AWS제공 구성 요소에 대한 업데이트를 제공합니다.

릴리스 날짜: 2021년 11월 12일

릴리스 하이라이트

- Windows 디바이스 지원 - 이제 Windows 운영 체제를 실행하는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 실행할 수 있습니다. 자세한 내용은 [Greengrass 기능 호환성](#) 단원을 참조하십시오.
- 새로운 사물 그룹 제거 동작 - 이제 사물 그룹에서 코어 디바이스를 제거하여 해당 디바이스에 대한 다음 배포에서 해당 사물 그룹의 구성 요소를 제거할 수 있습니다.

Important

이 변경으로 인해 코어 디바이스의 AWS IoT 정책에 `greengrass:ListThingGroupsForCoreDevice` 권한이 있어야 합니다. [AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 기본 AWS IoT 정책은 이 권한을 `greengrass:*` 포함하는 것을 허용합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.

- 하드웨어 보안 지원 - 이제 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있도록 하드웨어 보안 모듈(HSM)을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.
- HTTPS 프록시 지원 - 이제 HTTPS 프록시를 통해 연결하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 단원을 참조하십시오.

릴리스 세부 정보

- [플랫폼 지원 업데이트](#)
- [퍼블릭 구성 요소 업데이트](#)

플랫폼 지원 업데이트

플랫폼	세부 정보
Windows	AWS IoT Greengrass 이제는 다음 버전의 Windows에서 AWS IoT Greengrass 코어 소프트웨어 실행을 지원합니다. - Windows 10 - Windows Server 2019 자세한 내용은 Greengrass 기능 호환성 단원을 참조하십시오.

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.5.0을 사용할 수 있습니다. 새로운 기능 Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다.

구성 요소	세부 정보
	• 사물 그룹 제거 동작이 변경됩니다. 이 버전을 사용하면 사물 그룹에서 코어 디바이스를 제거하여 다음 배포에서 사물 그룹의 구성 요소를 제거할 수 있습니다. 이 변경으로 인해 코어 디바이스의 AWS IoT 정책에 <code>greengrass:ListThingGroupsForCoreDevice</code> 권한이 있어야 합니다. AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝한 경우 기본 AWS IoT 정책은 이 권한을 <code>greengrass:*</code> 포함하는 것을 허용합니다. 자세한 내용은 에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass 단원을 참조하십시오. • HTTPS 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오. • 새 <code>windowsUser</code> 구성 파라미터가 추가됩니다. 이 파라미터를 사용하여 Windows 코어 디바이스에서 구성 요소 실행에 사용할 기본 사용자를 지정할 수 있습니다. 자세한 내용은 구성 요소를 실행하는 사용자 구성 단원을 참조하십시오. • 느린 네트워크에서 성능을 개선하기 위해 HTTP 요청 제한 시간을 사용자 지정하는 데 사용할 수 있는 <code>httpClient</code> 구성 옵션이 새로 추가됩니다. 자세한 내용은 httpClient 구성 파라미터를 참조하세요. 버그 수정 및 개선 사항 • 부트스트랩 수명 주기 옵션이 수정되어 구성 요소에서 코어 디바이스를 다시 시작합니다. • 레시피 변수의 하이픈에 대한 지원을 추가합니다. • 온디맨드 Lambda 함수 구성 요소에 대한 IPC 권한 부여가 수정됩니다. • 로그 메시지가 개선되고 중요하지 않은 로그가 INFO에서 DEBUG 수준으로 변경되어 유용성이 증가합니다. • 자동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 Greengrass nucleus가 생성하는 기본 토큰 교환 역할에서 <code>iot:DescribeCertificate</code> 권한을 제거합니다. 이 권한은 Greengrass nucleus에서 사용하지 않습니다. • <code>iam:CreatePolicy</code>를 동일한 정책에 사용할 수 있는 경우 자동 프로비저닝 스크립트에 <code>iam:GetPolicy</code> 권한이 필요하지 않도록 문제를 해결합니다.

구성 요소	세부 정보
	• 사소한 추가 수정 및 개선 사항.
Greengrass CLI	Greengrass CLI 버전 2.5.0을 사용할 수 있습니다. 새로운 기능 • Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. • Windows 디바이스에서 시스템 그룹의 Greengrass CLI 사용 승인을 지정할 수 있도록 새 AuthorizedWindowsGroups 구성 파라미터가 추가됩니다. • 로컬 배포에 대해 windowsUser 파라미터가 추가됩니다. 이 파라미터를 사용하여 Windows 코어 디바이스에서 구성 요소 실행에 사용할 사용자를 지정할 수 있습니다.

구성 요소	세부 정보
CloudWatch 지표	CloudWatch 지표 구성 요소의 버전 3.0.0을 사용할 수 있습니다. CloudWatch 지표 구성 요소의 이 버전에는 버전 2.x와 다른 구성 파라미터가 필요합니다. 버전 2.x에 기본값이 아닌 구성을 사용하고 v2.x에서 v3.x로 업그레이드하려면 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 CloudWatch 지표 구성 요소 구성 섹션을 참조하세요. 새로운 기능 • Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. • 구성 요소 유형을 Lambda 구성 요소에서 일반 구성 요소로 변경합니다. 이 구성 요소에서는 이제 구독 생성이 레거시 구독 라우터 구성 요소에 종속되지 않습니다. • 구성 요소가 메시지를 수신하기 위해 구독하는 주제를 지정하는 새 <code>InputTopic</code> 구성 파라미터를 추가합니다. • 구성 요소에서 상태 응답이 게시되는 주제를 지정하는 새 <code>OutputTopic</code> 구성 파라미터를 추가합니다. • AWS IoT Core MQTT 주제를 게시하고 구독할지 여부를 지정하는 새 <code>PubSubToIoTCore</code> 구성 파라미터를 추가합니다. • 구성 요소 종속성이 설치되는 설치 스크립트를 선택적으로 비활성화할 수 있는 새 <code>UseInstaller</code> 구성 파라미터를 추가합니다. 버그 수정 및 개선 사항 입력 데이터의 중복 타임스탬프에 대한 지원을 추가합니다.
Lambda 관리자	Lambda 관리자 구성 요소의 버전 2.2.0을 사용할 수 있습니다. 버그 수정 및 개선 사항 • 재시작 후 Lambda 함수가 로그를 쓸 수 없는 문제를 해결합니다. • 주제에 와일드카드가 있을 때 레거시 구독 라우터가 중복 메시지를 보내는 문제를 해결합니다. • 고정되지 않은 Lambda 함수가 AWS IoT Device SDK에서 Greengrass IPC(프로세스 간 통신) 라이브러리를 사용할 수 없는 문제를 해결합니다.

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

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.4.0, 새로운 AWS제공 구성 요소 및 AWS제공 구성 요소에 대한 업데이트를 제공합니다.

릴리스 날짜: 2021년 8월 3일

릴리스 하이라이트

- 시스템 리소스 제한 - 이제 Greengrass nucleus 구성 요소에서 시스템 리소스 제한이 지원됩니다. 각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 [구성 요소에 대한 시스템 리소스 제한 구성](#) 단원을 참조하십시오.
- 구성 요소 일시 중지/재개 - 이제 Greengrass nucleus에서 구성 요소 일시 중지 및 재개가 지원됩니다. IPC(프로세스 간 통신) 라이브러리를 사용하여 다른 구성 요소의 프로세스를 일시 중지 및 재개하는 사용자 지정 구성 요소를 개발할 수 있습니다. 자세한 내용은 [PauseComponent](#) 및 [ResumeComponent](#) 섹션을 참조하세요.
- AWS IoT 플릿 프로비저닝을 사용하여 설치 - 새 AWS IoT 플릿 프로비저닝 플러그인을 사용하여 연결하는 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치 AWS IoT 하여 필요한 AWS 리소스를 프로비저닝합니다. 디바이스는 클레임 인증서를 사용하여 프로비저닝합니다. 제조 중에 디바이스에 클레임 인증서를 내장할 수 있으므로 각 디바이스가 온라인 상태가 되는 즉시 프로비저닝할 수 있습니다. 자세한 내용은 [AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.
- 사용자 지정 프로비저닝을 사용하여 설치 - 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 필요한 AWS 리소스를 프로비저닝할 사용자 지정 프로비저닝 플러그인을 개발합니다. 설치 중에 실행되는 Java 애플리케이션을 생성하여 사용자 지정 사용 사례에 맞게 Greengrass 코어 디바이스를 설정할 수 있습니다. 자세한 내용은 [사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.4.0을 사용할 수 있습니다. 새로운 기능 • 시스템 리소스 제한에 대한 지원을 추가합니다. 각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 구성 요소에 대한 시스템 리소스 제한 구성 단원을 참조하십시오. • 구성 요소를 일시 중지하고 재개하도록 IPC 작업이 추가됩니다. 자세한 내용은 PauseComponent 및 ResumeComponent(를) 참조하세요. • 플러그인 프로비저닝에 대한 지원을 추가합니다. 설치 중에 실행할 JAR 파일을 지정하여 Greengrass 코어 디바이스에 필요한 AWS 리소스를 프로비저닝할 수 있습니다. Greengrass nucleus에는 사용자 지정 프로비저닝 플러그인 개발을 구현할 수 있는 인터페이스가 포함되어 있습니다. 자세한 내용은 사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치 단원을 참조하십시오. • AWS IoT Greengrass 코어 소프트웨어 설치 관리자에 선택적 thing-name-policy 인수를 추가합니다. 자동 리소스 프로비저닝으로 코어 소프트웨어를 설치할 때 옵션을 사용하여 기존 또는 사용자 지정 AWS IoT 정책을 지정할 수 있습니다. AWS IoT Greengrass

구성 요소	세부 정보
	버그 수정 및 개선 사항 - 시작 시 로깅 구성이 업데이트됩니다. 이렇게 하면 시작 시 로깅 구성이 적용되지 않는 문제를 해결합니다. - 설치 중에 Greengrass 루트 폴더의 구성 요소 스토어를 가리키도록 nucleus 로더 심링크가 업데이트됩니다. 이 업데이트를 통해 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 다운로드하는 JAR 파일 및 기타 nucleus 아티팩트를 삭제할 수 있습니다. - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.
Greengrass CLI	Greengrass CLI 버전 2.4.0을 사용할 수 있습니다. 새로운 기능 시스템 리소스 제한에 대한 지원을 추가합니다. 로컬 배포를 생성할 때 각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 구성 요소에 대한 시스템 리소스 제한 구성 및 배포 생성 명령을 참조하세요.
AWS IoT 클레임 별 플릿 프로비저닝	이제 클레임 플러그인을 통한 AWS IoT 플릿 프로비저닝을 사용할 수 있습니다. 자세한 내용은 AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치 단원을 참조하십시오. 새로운 기능 AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하는 지원을 추가합니다. 설치하는 동안 디바이스는 AWS IoT에 연결하여 필요한 AWS 리소스를 프로비저닝하고 일반 작업에 사용할 디바이스 인증서를 다운로드합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.3.0이 제공됩니다.

릴리스 날짜: 2021년 6월 29일

릴리스 하이라이트

- 대규모 구성 지원 - Greengrass nucleus 구성 요소에서 이제 배포 문서가 최대 10MB까지 지원됩니다. 이제 Greengrass 구성 요소에 더 큰 구성 업데이트를 배포할 수 있습니다.

Note

이 기능을 사용하려면 코어 디바이스의 AWS IoT 정책에서 `greengrass:GetDeploymentConfiguration` 권한을 허용해야 합니다. [AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 코어 디바이스의 AWS IoT 정책은 이 권한이 `greengrass:*` 포함된 것을 허용합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS 제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS 제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 [섹션을 참조하세요 AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.3.0을 사용할 수 있습니다. 새로운 기능 - 최대 10MB의 배포 구성 문서, 7KB(대상 사물 배포의 경우) 또는 31KB(대상 사물 그룹 배포의 경우)에 대한 지원을 추가합니다. 이 기능을 사용하려면 코어 디바이스의 AWS IoT 정책에서 <code>greengrass:GetDeploymentConfiguration</code> 권한을 허용해야 합니다. AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝한 경우 코어 디바이스의 AWS IoT 정책은 이 권한이 <code>greengrass:*</code> 포함된 것을 허용합니다. 자세한 내용은 에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass 단원을 참조하십시오. <code>iot:thingName</code> 레시피 변수가 추가됩니다. 이 레시피 변수를 사용하여 레시피에서 코어 디바이스의 AWS IoT 사물 이름을 가져올 수 있습니다. 자세한 내용은 레시피 변수 단원을 참조하십시오. 버그 수정 및 개선 사항 - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.

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

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.2.0, 새로운 AWS제공 구성 요소 및 AWS제공 구성 요소에 대한 업데이트를 제공합니다.

릴리스 날짜: 2021년 6월 18일

릴리스 하이라이트

- 클라이언트 디바이스 지원 AWS- 새로 제공되는 클라이언트 디바이스 구성 요소를 사용하면 클라우드 검색을 사용하여 클라이언트 디바이스를 코어 디바이스에 연결할 수 있습니다. Greengrass 구성 요소에서 클라이언트 디바이스와 동기화 AWS IoT Core 하고 클라이언트 디바이스와 상호 작용할 수 있습니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

- 로컬 새도 서비스 - 새로운 새도 관리자 구성 요소를 사용하면 코어 디바이스에서 로컬 새도 서비스를 사용할 수 있습니다. 이 새도 서비스를 사용하여 AWS IoT Device SDK의 Greengrass IPC(프로세스 간 통신) 라이브러리를 사용하여 오프라인 상태에서 로컬 새도와 상호 작용할 수 있습니다. 새도 관리자 구성 요소를 사용하여 로컬 새도 상태를 동기화할 수도 있습니다 AWS IoT Core. 자세한 내용은 [디바이스 새도와 연동](#) 단원을 참조하십시오.

릴리스 세부 정보

- [퍼블릭 구성 요소 업데이트](#)

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.2.0을 사용할 수 있습니다. 새로운 기능 - 로컬 새도 관리를 위한 IPC 작업이 추가됩니다. 버그 수정 및 개선 사항 - JAR 파일의 크기가 감소합니다. - 메모리 사용량이 감소합니다.

구성 요소	세부 정보
	• 경우에 따라 로그 구성이 업데이트되지 않는 문제가 해결됩니다. • 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.
새도 관리자	새로운 새도 관리자 구성 요소의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 클래식 새도 및 명명된 새도에 대한 지원을 추가합니다. • IPC를 사용하는 로컬 새도 관리에 대한 지원을 추가합니다. • 와의 새도우 동기화에 대한 지원을 추가합니다 AWS IoT Core.
클라이언트 디바이스 인증	새로운 클라이언트 디바이스 인증 구성 요소의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • MQTT를 통해 코어 디바이스에 연결하는 로컬 IoT 디바이스인 Greengrass 클라이언트 디바이스에 대한 지원을 추가합니다. • 클라이언트 디바이스 및 해당 MQTT 작업의 인증 및 권한 부여에 대한 지원을 추가합니다.
Moquette MQTT 브로커	새로운 Moquette MQTT 브로커 구성 요소의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 클라이언트 디바이스와의 통신을 처리하는 로컬 Moquette MQTT 브로커에 대한 지원을 추가합니다.
MQTT 브리지	새로운 MQTT 브리지 구성 요소의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • 로컬 MQTT 브로커, 로컬 Greengrass 게시/구독 브로커 및 AWS IoT Core MQTT 브로커 간에 메시지를 릴레이하는 지원을 추가합니다.

구성 요소	세부 정보
IP 감지기	새로운 IP 감지기 구성 요소의 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 클라이언트 디바이스가 연결할 수 있도록 코어 디바이스의 로컬 MQTT 브로커 엔드포인트를 AWS IoT Greengrass 클라우드 서비스에 보고하는 지원을 추가합니다.
로그 관리자	로그 관리자 구성 요소의 버전 2.1.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 경우에 따라 시스템 로그 구성이 업데이트되지 않는 문제를 해결합니다.
DLR 객체 감지	DLR 객체 감지의 버전 2.1.2를 사용할 수 있습니다. 버그 수정 및 개선 사항 샘플 DLR 객체 감지 추론 결과에서 잘못된 경계 상자가 발생하는 이미지 크기 조정 문제를 해결합니다.
TensorFlow Lite 객체 감지	TensorFlow Lite 객체 감지 구성 요소의 버전 2.1.1을 사용할 수 있습니다. 버그 수정 및 개선 사항 샘플 TensorFlow Lite 객체 감지 추론 결과에서 잘못된 경계 상자가 발생하는 이미지 크기 조정 문제를 해결합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.1.0이 제공되고 AWS제공 구성 요소가 업데이트됩니다.

릴리스 날짜: 2021년 4월 26일

릴리스 하이라이트

- Docker Hub 및 Amazon Elastic Container Registry(Amazon ECR) 통합 - 새로운 Docker 애플리케이션 관리자 구성 요소를 사용하면 Amazon ECR에서 퍼블릭 또는 프라이빗 이미지를 다운로드할 수

있습니다. 이 구성 요소를 사용하여 Docker Hub 및 AWS Marketplace에서 퍼블릭 이미지를 다운로드할 수도 있습니다. 자세한 내용은 [Docker 컨테이너 실행](#) 단원을 참조하십시오.

- AWS IoT Greengrass 코어 소프트웨어용 Dockerfile 및 Docker 이미지 - Greengrass Docker 이미지를 사용하여 Amazon Linux 2를 기본 운영 체제로 사용하는 Docker 컨테이너 AWS IoT Greengrass에서 실행할 수 있습니다. AWS IoT Greengrass Dockerfile을 사용하여 자체 Greengrass 이미지를 빌드할 수도 있습니다. 자세한 내용은 [Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.
- 추가 기계 학습 프레임워크 및 플랫폼 지원 - 사전 훈련된 모델을 사용하여 TensorFlow Lite 2.5.0 및 DLR 1.6.0을 사용하는 샘플 이미지 분류 및 객체 감지를 수행하는 샘플 기계 학습 추론 구성 요소를 배포할 수 있습니다. 또한 Armv8(AArch64) 디바이스에 대한 샘플 기계 학습 지원이 확장됩니다. 자세한 내용은 [기계 학습 추론 수행](#) 단원을 참조하십시오.

릴리스 세부 정보

- [플랫폼 지원 업데이트](#)
- [퍼블릭 구성 요소 업데이트](#)

플랫폼 지원 업데이트

플랫폼	세부 정보
Docker	AWS IoT Greengrass 이제에 대한 Dockerfile 및 Docker 이미지를 사용할 수 있습니다. Dockerfile AWS IoT Greengrass 는 Amazon Linux 2(x86_64) 기본 이미지에 AWS IoT Greengrass 코어 소프트웨어 및 종속성이 설치된 컨테이너 이미지를 빌드하기 위한 Dockerfile을 제공합니다. 다른 플랫폼 아키텍처에서 실행되도록 Dockerfile의 기본 이미지를 수정할 수 AWS IoT Greengrass 있습니다. 도커 이미지 AWS IoT Greengrass 는 Amazon Linux 2(x86_64) 기본 이미지에 AWS IoT Greengrass 코어 소프트웨어 및 종속성이 설치된 사전 빌드된 Docker 이미지를 제공합니다.

플랫폼	세부 정보
	자세한 내용은 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행 단원을 참조하십시오.

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 [섹션을 참조하세요](#) [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 - Amazon ECR의 프라이빗 리포지토리에서 Docker 이미지 다운로드가 지원됩니다. - 다음 파라미터가 추가되어 코어 디바이스에서 MQTT 구성을 사용자 지정합니다. <code>maxInFlightPublishes</code> - 동시에 이동할 수 있는 확인되지 않은 MQTT QoS 1 메시지의 최대 수입니다. <code>maxPublishRetry</code> - 게시에 실패한 메시지의 최대 재시도 횟수입니다.

구성 요소	세부 정보
	• <code>fleetstatusservice</code> 구성 파라미터가 추가되어 코어 디바이스가 디바이스 상태를 AWS 클라우드에 게시하는 간격을 구성합니다. • 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요. 버그 수정 및 개선 사항 • <code>nucleus</code>가 다시 시작될 때 새도 배포가 중복되는 문제를 해결합니다. • 서비스 로드 예외가 발생했을 때 <code>nucleus</code>가 충돌하는 문제를 해결합니다. • 순환 종속성이 포함된 배포 실패에 대한 구성 요소 종속성 확인이 개선됩니다. • 해당 구성 요소가 이전에 코어 디바이스에서 제거된 경우 플러그인 구성 요소가 재배포되지 않는 문제를 해결합니다. • HOME 환경 변수가 Lambda 구성 요소 또는 루트로 실행되는 구성 요소의 <code>/greengrass/v2 /work</code> 디렉터리로 설정되는 문제가 해결됩니다. 이제 HOME 변수가 구성 요소를 실행하는 사용자의 홈 디렉터리로 올바르게 설정됩니다. • 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.
Docker 애플리케이션 관리자	Docker 애플리케이션 관리자 구성 요소의 새로운 버전 2.0.0을 사용할 수 있습니다. 새로운 기능 • Amazon ECR의 프라이빗 리포지토리에서 이미지를 다운로드하는 자격 증명을 관리합니다. • Amazon ECR, Docker Hub 및에서 퍼블릭 이미지를 다운로드합니다 AWS Marketplace.
Lambda 런처	Lambda 런처 구성 요소의 버전 2.0.4를 사용할 수 있습니다. 버그 수정 및 개선 사항 • 구성 요소가 <code>AddGroupOwner</code> 를 Lambda 함수 컨테이너로 올바르게 전달하지 않는 문제를 해결합니다.

구성 요소	세부 정보
레거시 구독 라우터	레거시 구독 라우터 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 버그 수정 및 개선 사항 source 및 target에 대한 ARN 대신 구성 요소 이름을 지정할 수 있도록 하는 지원이 추가됩니다. 구독의 구성 요소 이름을 지정하는 경우 Lambda 함수 버전이 변경될 때마다 구독을 재구성할 필요가 없습니다.
로컬 디버그 콘솔	로컬 디버그 콘솔 구성 요소의 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 사항 - 구성 편집기에서 플래시바 메시지를 해제할 수 있습니다.
로그 관리자	로그 관리자 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 버그 수정 및 개선 사항 - 표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에 대해 작동하는 logFileDirectoryPath 및 logFileRegex에 기본값을 사용합니다. - CloudWatch Logs에 로그를 업로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. - 로그 스트림 이름에서 콜론 문자(:)가 올바르게 처리됩니다. CloudWatch Logs 로그 스트림 이름에 콜론을 포함할 수 없습니다. - 로그 스트림에서 사물 그룹 이름을 제거하여 로그 스트림 이름을 간소화합니다. - 정상적인 동작 중에 인쇄되는 오류 로그 메시지가 제거됩니다.

구성 요소	세부 정보
DLR 이미지 분류	DLR 이미지 분류 구성 요소의 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • 딥 러닝 런타임 v1.6.0을 사용합니다. • Armv8(AArch64) 플랫폼에서 샘플 이미지 분류 지원이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다. • 샘플 추론을 위해 카메라 통합이 활성화됩니다. 새로운 UseCamera 구성 파라미터를 사용하여 샘플 추론 코드가 Greengrass 코어 디바이스의 카메라에 액세스하고 캡처된 이미지에서 로컬로 추론을 실행할 수 있습니다. • AWS 클라우드에 추론 결과 게시에 대한 지원이 추가됩니다. 새로운 PublishResultsOnTopic 구성 파라미터를 사용하여 결과를 게시하려는 주제를 지정합니다. • 새로운 ImageDirectory 구성 파라미터가 추가되어 추론을 수행할 이미지의 사용자 지정 디렉터리를 지정할 수 있습니다. 버그 수정 및 개선 사항 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과가 작성됩니다. • AWS IoT Greengrass 코어 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 로깅합니다. • AWS IoT Device SDK 를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다.

구성 요소	세부 정보
DLR 객체 감지	DLR 객체 감지 구성 요소의 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • 딥 러닝 런타임 v1.6.0을 사용합니다. • Armv8(AArch64) 플랫폼에서 샘플 객체 감지에 대한 지원이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다. • 샘플 추론을 위해 카메라 통합이 활성화됩니다. 새로운 UseCamera 구성 파라미터를 사용하여 샘플 추론 코드가 Greengrass 코어 디바이스의 카메라에 액세스하고 캡처된 이미지에서 로컬로 추론을 실행할 수 있습니다. • AWS 클라우드에 추론 결과 게시에 대한 지원이 추가됩니다. 새로운 PublishResultsOnTopic 구성 파라미터를 사용하여 결과를 게시하려는 주제를 지정합니다. • 새로운 ImageDirectory 구성 파라미터가 추가되어 추론을 수행할 이미지의 사용자 지정 디렉터리를 지정할 수 있습니다. 버그 수정 및 개선 사항 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과가 작성됩니다. • AWS IoT Greengrass 코어 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 로깅합니다. • AWS IoT Device SDK 를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다.
DLR 이미지 분류 모델 저장소	DLR 이미지 분류 모델 저장소 구성 요소의 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • Armv8(AArch64) 플랫폼에 대한 샘플 ResNet-50 이미지 분류 모델이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다.

구성 요소	세부 정보
DLR 객체 감지 모델 저장소	DLR 객체 감지 모델 저장소 구성 요소의 버전 2.1.1을 사용할 수 있습니다. 새로운 기능 • Armv8(AArch64) 플랫폼에 대한 샘플 YOLOv3 객체 감지 모델이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다.
DLR 설치 관리자	DLR 구성 요소의 버전 1.6.1을 사용할 수 있습니다. 새로운 기능 • 딥 러닝 런타임 v1.6.0 및 해당 종속성을 설치합니다. • Armv8(AArch64) 플랫폼의 DLR 설치에 대한 지원을 추가합니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다. 버그 수정 및 개선 사항 • 가상 환경에 AWS IoT Device SDK 를 설치하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다. • 사소한 추가 버그 수정 및 개선 사항입니다.
TensorFlow Lite 이미지 분류	새로운 TensorFlow Lite 이미지 분류 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • TensorFlow Lite를 사용하는 샘플 이미지 분류 추론에 대한 지원이 추가됩니다.
TensorFlow Lite 객체 감지	새로운 TensorFlow Lite 객체 감지 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 • TensorFlow Lite를 사용하는 샘플 객체 감지 추론에 대한 지원이 추가됩니다.

구성 요소	세부 정보
TensorFlow Lite 이미지 분류 모델 저장소	새로운 TensorFlow Lite 이미지 분류 모델 저장소 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 TensorFlow Lite를 사용하는 샘플 이미지 분류 추론을 위해 사전 훈련된 MobileNet v1 정량화 모델이 제공됩니다.
TensorFlow Lite 객체 감지 모델 저장소	새로운 TensorFlow Lite 객체 감지 모델 저장소 구성 요소의 버전 2.1.0을 사용할 수 있습니다. 새로운 기능 TensorFlow Lite를 사용하는 샘플 객체 감지 추론을 위해 COCO 데이터 세트에 대해 사전 훈련된 SSD(Single Shot Detection) MobileNet 모델이 제공됩니다.
TensorFlow Lite	새로운 TensorFlow Lite 구성 요소의 버전 2.5.0을 사용할 수 있습니다. 새로운 기능 Armv7, Armv8(AArch64) 및 x86_64 플랫폼의 가상 환경에 TensorFlow Lite v1.6.0 및 해당 종속성이 설치됩니다.

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

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.0.5와 업데이트 제공 구성 요소를 AWS제공합니다. AWS 중국 리전에서 네트워크 프록시 지원 관련 문제와 Greengrass 데이터 영역 엔드포인트 관련 문제를 해결합니다.

릴리스 날짜: 2021년 3월 9일

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.0.5를 사용할 수 있습니다. 버그 수정 및 개선 사항 • AWS제공된 구성 요소를 다운로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. • AWS 중국 리전에서 올바른 Greengrass 데이터 영역 엔드포인트를 사용합니다.

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

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.0.4가 제공됩니다. 포트 443을 통해 HTTPS 통신을 구성하는 새로운 greengrassDataPlanePort 파라미터와 버그 수정이 포함되어 있습니다. 이제 최소 IAM 정책에는 AWS IoT Greengrass 코어 소프트웨어 설치 관리자가에서 실행될 sts:GetCallerIdentity 때 iam:GetPolicy 및가 필요합니다--provision true.

릴리스 날짜: 2021년 2월 4일

퍼블릭 구성 요소 업데이트

다음 표에는 새로운 기능과 업데이트된 기능이 포함된 AWS제공 구성 요소가 나열되어 있습니다.

⚠ Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

구성 요소	세부 정보
Greengrass nucleus	Greengrass nucleus의 버전 2.0.4를 사용할 수 있습니다. 새로운 기능 • 포트 443을 통한 HTTPS 트래픽이 활성화됩니다. nucleus 구성 요소의 버전 2.0.4에 대한 새 greengrassDataPlanePort 구성 파라미터를 사용하여 기본 포트 8443 대신 포트 443을 통해 이동하도록 HTTPS 통신을 구성할 수 있습니다. 자세한 내용은 포트 443을 통해 HTTPS 구성 단원을 참조하십시오. • 작업 경로 레시피 변수가 추가됩니다. 이 레시피 변수를 사용하여 구성 요소의 작업 폴더 경로를 가져올 수 있고, 이 폴더를 사용하여 구성 요소와 해당 종속성 간에 파일을 공유할 수 있습니다. 자세한 내용은 작업 경로 레시피 변수를 참조하십시오. 버그 수정 및 개선 사항 • 역할 정책이 이미 있는 경우 토큰 교환 AWS Identity and Access Management (IAM) 역할 정책의 생성을 방지합니다. 이 변경으로 인해 <code>--provision true</code> 로 실행할 때 설치 관리자에 <code>iam:GetPolicy</code> 및 <code>sts:GetCallerIdentity</code> 가 필요합니다. 자세한 내용은 설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책 단원을 참조하십시오. • 아직 성공적으로 등록되지 않은 배포의 취소가 올바르게 처리됩니다.

구성 요소	세부 정보
	• 배포를 롤백할 때 최신 타임스탬프가 있는 이전 항목을 제거하도록 구성이 업데이트됩니다. • 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.

AWS IoT Greengrass 버전 1에서 마이그레이션

AWS IoT Greengrass Version 2 는 AWS IoT Greengrass 코어 소프트웨어, APIs 및 콘솔의 메이저 버전 릴리스입니다. AWS IoT Greengrass V2 는 모듈식 애플리케이션 AWS IoT Greengrass V1, 대규모 디바이스에 대한 배포, 추가 플랫폼 지원과 같은 몇 가지 개선 사항을 도입합니다.

Note

지원 종료 알림: 2026년 10월 7일에 AWS 에 대한 지원이 종료됩니다 AWS IoT Greengrass Version 1. 2026년 10월 7일 이후에는 AWS IoT Greengrass V1 콘솔 또는 AWS IoT Greengrass V1 리소스에 더 이상 액세스할 수 없습니다.

이 가이드의 지침에 따라에서 로 마이그레이션 AWS IoT Greengrass V1 합니다 AWS IoT Greengrass V2.

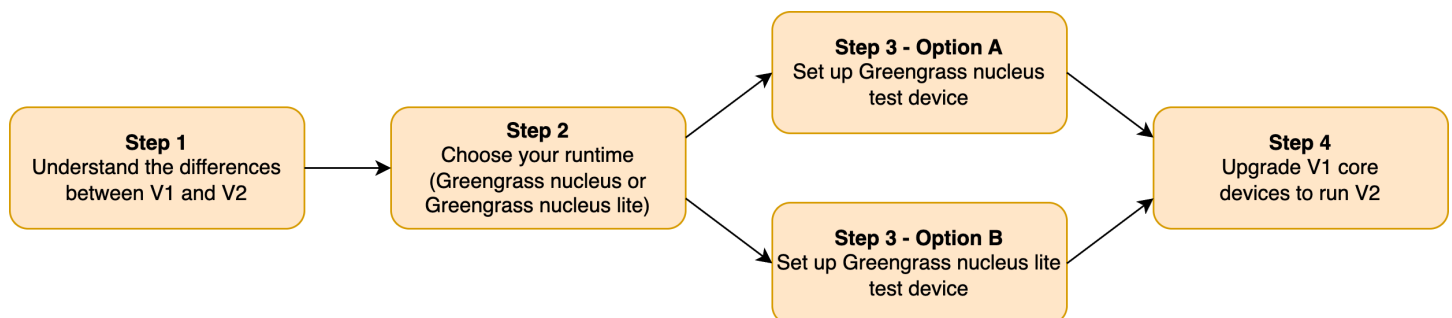
마이그레이션 개요

상위 수준에서 다음 절차를 사용하여 코어 디바이스에서 로 업그레이드 AWS IoT Greengrass V1 할 수 있습니다 AWS IoT Greengrass V2.

마이그레이션하기 전에 두 가지 런타임 옵션 중 하나를 선택합니다.

- Greengrass nucleus(마이그레이션 작업 감소, 전체 기능 지원)
- Greengrass nucleus lite(더 높은 마이그레이션 작업, 리소스 제한 디바이스용으로 설계됨).

따르는 정확한 절차는 디바이스 리소스, 필수 기능 및 특정 환경 요구 사항에 따라 달라집니다.



1. [V1과 V2 사이의 차이점 이해](#)

AWS IoT Greengrass V2 는 디바이스 플릿 및 배포 가능한 소프트웨어에 대한 새로운 기본 개념을 도입하고, V2는 V1의 여러 개념을 간소화합니다.

AWS IoT Greengrass V2 클라우드 서비스 및 AWS IoT Greengrass 코어 소프트웨어 v2.x는 AWS IoT Greengrass V1 클라우드 서비스 및 AWS IoT Greengrass 코어 소프트웨어 v1.x와 역호환되지 않습니다. 따라서 AWS IoT Greengrass V1 over-the-air(OTA) 업데이트는 코어 디바이스를 V1에서 V2로 업그레이드할 수 없습니다.

2. [런타임 선택\(Greengrass nucleus 또는 Greengrass nucleus lite\)](#)

디바이스 리소스 및 기능 요구 사항에 따라 Greengrass nucleus 또는 Greengrass nucleus lite를 결정합니다.

- Greengrass nucleus 경로: 마이그레이션 작업 감소. Lambda 함수는 코드 변경을 최소화하면서 Lambda 구성 요소로 가져올 수 있습니다. V1 기능(로컬 새도우 서비스, 클라이언트 디바이스, 커넥터)을 지원합니다.
- Greengrass nucleus lite 경로: 마이그레이션 작업 증가. Lambda 함수는 일반 구성 요소로 변환해야 하며, AWS IoT Greengrass 코어 SDK 대신 AWS IoT Device SDK V2/AWS IoT Greengrass 구성 요소 SDK를 사용하려면 코드를 변경해야 합니다. 로컬 새도우 서비스, 클라이언트 디바이스 또는 커넥터를 지원하지 않습니다.

3. [V2에서 V1 애플리케이션을 테스트할 새 디바이스 설정](#)

프로덕션 중 디바이스에 대한 위험을 최소화하려면 V2에서 V1 애플리케이션을 테스트할 새 디바이스를 생성합니다. 런타임 선택에 따라 설정 가이드를 선택합니다.

- 옵션 A - Greengrass nucleus 런타임: [V2에서 V1 애플리케이션을 테스트하도록 새 디바이스를 설정합니다](#). 코드 변경을 최소화하면서 Lambda 함수를 Lambda 구성 요소로 가져옵니다.
- 옵션 B - Greengrass nucleus lite 런타임: [V2\(Greengrass nucleus lite\)에서 V1 애플리케이션을 테스트하도록 새 디바이스를 설정합니다](#). 를 사용하여 Lambda 함수를 일반 구성 요소로 변환합니다 AWS IoT Device SDK.

4. [V2를 실행하도록 V1 코어 디바이스 업그레이드](#)

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

AWS IoT Greengrass V1과 AWS IoT Greengrass V2의 차이점

AWS IoT Greengrass V2는 디바이스, 플릿 및 배포 가능한 소프트웨어에 대한 새로운 기본 개념을 도입합니다. 이 섹션에서는 V2와 다른 V1 개념에 대해 설명합니다.

Greengrass 개념 및 용어

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
애플리케이션 코드	AWS IoT Greengrass V1에서 Lambda 함수는 코어 디바이스에서 실행되는 소프트웨어를 정의합니다. 각 Greengrass 그룹에서 함수에서 사용하는 구독 및 로컬 리소스를 정의합니다. AWS IoT Greengrass 코어 소프트웨어가 컨테이너화된 Lambda 런타임 환경에서 실행되는 Lambda 함수의 경우 메모리 제한과 같은 컨테이너 파라미터를 정의합니다.	AWS IoT Greengrass V2에서 구성 요소는 코어 디바이스에서 실행되는 소프트웨어입니다. • 각 구성 요소에는 구성 요소 수명 주기의 각 단계에서 실행할 구성 요소의 메타데이터, 파라미터, 종속성 및 스크립트를 정의하는 레시피가 있습니다. • 레시피는 스크립트, 컴파일된 코드 및 정적 리소스와 같은 이진 파일인 구성 요소의 아티팩트도 정의합니다. • 코어 디바이스에 구성 요소를 배포하면 코어 디바이스는 구성 요소 레시피와 아티팩트를 다운로드하고 구성 요소를 실행합니다. AWS IoT Greengrass V2의 Lambda 런타임 환경에서 실행되는 구성 요소로 V1 Lambda 함수를 가져올 수 있습니다. Lambda 함수를 가져올 때 함수에 대한 구독, 로컬 리소스 및 컨테이너 파라미터를 지정

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		합니다. 자세한 내용은 2단계: AWS IoT Greengrass V1 애플리케이션 마이그레이션을 위한 AWS IoT Greengrass V2 구성 요소 생성 및 배포 섹션을 참조하세요. 사용자 지정 구성 요소와 생성 방법에 대한 자세한 내용은 AWS IoT Greengrass 구성 요소 개발 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 그룹 및 배포	AWS IoT Greengrass V1에서 그룹은 코어 디바이스, 해당 코어 디바이스의 설정 및 소프트웨어, 코어 디바이스에 연결할 수 있는 AWS IoT 사물 목록을 정의합니다. 배포를 생성하여 그룹의 구성을 코어 디바이스로 보냅니다.	AWS IoT Greengrass V2에서는 배포를 사용하여 코어 디바이스에서 실행되는 소프트웨어 구성 요소 및 구성을 정의합니다. • 각 배포는 단일 코어 디바이스(AWS IoT 사물) 또는 여러 개의 코어 디바이스를 포함할 수 있는 AWS IoT 사물 그룹을 대상으로 합니다. • 사물 그룹에 대한 배포는 연속적이므로 사물 그룹에 코어 디바이스를 추가하면 해당 그룹에 대한 소프트웨어 구성을 수신합니다. 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포 섹션을 참조하세요. AWS IoT Greengrass V2에서 Greengrass CLI를 사용하여 로컬 배포를 생성하고 개발하는 디바이스에서 사용자 지정 소프트웨어 구성 요소를 테스트할 수도 있습니다. 자세한 내용은 AWS IoT Greengrass 구성 요소 생성 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 코어 소프트웨어	AWS IoT Greengrass V1에서 AWS IoT Greengrass 코어 소프트웨어는 소프트웨어와 모든 기능이 포함된 단일 패키지입니다. AWS IoT Greengrass 코어 소프트웨어를 설치하는 엣지 디바이스를 Greengrass 코어라고 합니다.	AWS IoT Greengrass V2에서 AWS IoT Greengrass 코어 소프트웨어는 모듈식이므로 설치할 항목을 선택하고 메모리 공간을 관리할 수 있습니다. • Greengrass nucleus 구성 요소는 AWS IoT Greengrass 코어 소프트웨어의 최소 필수 설치입니다. nucleus를 설치하는 엣지 디바이스를 Greengrass 코어 디바이스라고 합니다. • nucleus는 코어 디바이스에서 다른 구성 요소의 배포, 오케스트레이션 및 수명 주기 관리를 처리합니다. • 스트림 관리자, 보안 암호 관리자 및 로그 관리자와 같은 기능은 필요한 경우에만 배포하는 구성 요소입니다. 자세한 내용은 AWS 제공 구성 요소 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Connectors	AWS IoT Greengrass V1에서 커넥터는 AWS IoT Greengrass V1 코어 디바이스에 배포하는 사전 빌드된 모듈로 로컬 인프라, 디바이스 프로토콜, AWS, 기타 클라우드 서비스와 상호 작용합니다.	AWS IoT Greengrass V2에서 AWS는 V1의 커넥터에서 제공하는 기능을 구현하는 Greengrass 구성 요소를 제공합니다. 다음 AWS IoT Greengrass V2 구성 요소는 Greengrass V1 커넥터 기능을 제공합니다. • CloudWatch 지표 구성 요소 • AWS IoT Device Defender 구성 요소 • Firehose 구성 요소 • Modbus-RTU 프로토콜 어댑터 구성 요소 • Amazon SNS 구성 요소 자세한 내용은 AWS 제공 구성 요소 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
연결된 디바이스(Greengrass 디바이스)	AWS IoT Greengrass V1에서 연결된 디바이스는 Greengrass 그룹에 추가하여 해당 그룹의 코어 디바이스에 연결하고 MQTT를 통해 통신하는 AWS IoT 사물입니다. 연결된 디바이스를 추가 또는 제거할 때마다 해당 그룹을 배포해야 합니다. 구독을 사용하여 코어 디바이스의 연결된 디바이스, AWS IoT Core 및 애플리케이션 간에 메시지를 릴레이합니다.	AWS IoT Greengrass V2에서 연결된 디바이스를 Greengrass 클라이언트 디바이스라고 합니다. - 클라이언트 디바이스를 코어 디바이스에 연결하고 MQTT를 통해 통신합니다. - 클라이언트 디바이스에 연결 권한을 부여하려면 클라이언트 디바이스 그룹에 적용할 수 있는 권한 부여 정책을 정의하고, 따라서 클라이언트 디바이스를 추가하거나 제거하기 위해 배포를 생성할 필요가 없습니다. - 클라이언트 디바이스, AWS IoT Core 및 Greengrass 구성 요소 간에 메시지를 릴레이하려면 선택적 MQTT 브리지 구성 요소를 구성할 수 있습니다. AWS IoT Greengrass V1 및 AWS IoT Greengrass V2 모두에서 디바이스는 FreeRTOS를 실행하거나 AWS IoT Device SDK 또는 Greengrass Discovery API를 사용하여 연결할 수 있는 코어 디바이스에 관한 정보를 가져올 수 있습니다. Greengrass Discovery API는 이전 버전과 호환되므로 V1 코어 디바이스

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		에 연결하는 클라이언트 디바이스가 있는 경우 코드를 변경하지 않고 V2 코어 디바이스에 연결할 수 있습니다. 클라이언트 디바이스에 대한 자세한 내용은 로컬 IoT 디바이스와 상호 작용 섹션을 참조하세요.
로컬 리소스	AWS IoT Greengrass V1에서 컨테이너에서 실행되는 Lambda 함수는 코어 디바이스 파일 시스템의 볼륨 및 디바이스에 액세스하도록 구성할 수 있습니다. 이러한 파일 시스템 리소스를 로컬 리소스라고 합니다.	AWS IoT Greengrass V2에서 Lambda 함수, Docker 컨테이너 또는 기본 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 비 Lambda 구성 요소는 코어 디바이스의 로컬 리소스에서 직접 사용할 수 있으므로 구성 요소가 사용하는 로컬 리소스를 지정할 필요가 없습니다.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
로컬 새도 서비스	AWS IoT Greengrass V1에서 로컬 새도 서비스는 기본적으로 활성화되고, 이름이 지정되지 않은 클래식 새도만 지원합니다. Lambda 함수에서 AWS IoT Greengrass 코어 SDK를 사용하여 디바이스의 새도와 상호 작용합니다.	AWS IoT Greengrass V2에서는 새도 관리자 구성 요소를 배포하여 로컬 새도 서비스를 활성화합니다. • Lambda 함수 및 사용자 지정 구성 요소의 AWS IoT Device SDK V2를 사용하여 디바이스의 새도와 상호 작용할 수 있습니다. • 로컬 새도 서비스는 명명된 새도를 지원합니다. • 로컬 새도 서비스를 사용하면 새도를 삭제하고 삭제된 새도를 AWS IoT Core와 동기화할 수 있습니다. 자세한 내용은 디바이스 새도와 연동 섹션을 참조하세요.

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
구독	AWS IoT Greengrass V1에서 Lambda 함수, 커넥터, 연결된 디바이스, AWS IoT Core MQTT 브로커 및 로컬 새도 서비스 간의 통신 채널을 지정하도록 Greengrass 그룹에 대한 구독을 정의합니다. 구독으로 Lambda 함수가 함수 페이로드로 사용할 이벤트 메시지를 수신하는 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 새도 서비스와 상호 작용합니다. - 다른 구성 요소 또는 AWS IoT Core MQTT 브로커의 메시지에 반응하는 구성 요소를 개발하려면 로컬 게시/구독 메시징 및 AWS IoT Core MQTT 메시징에 IPC(프로세스 간 통신) 인터페이스를 사용할 수 있습니다. - 로컬 새도 서비스와 상호 작용하는 구성 요소를 개발하려면 로컬 새도 서비스용 IPC 인터페이스를 사용할 수 있습니다. - 구성 요소 구성에서 구성 요소에 사용할 권한이 있는 주제와 로컬 새도를 지정하도록 권한 부여 정책을 정의합니다. - 클라이언트 디바이스, 로컬 게시/구독 브로커 및 AWS IoT Core MQTT 브로커 간의 통신 채널을 구성하려면 MQTT 브리지 구성 요소를 구성 및 배포합니다. MQTT

개념	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		브리지 구성 요소를 사용하면 구성 요소의 클라이언트 디바이스와 상호 작용하고 클라이언트 디바이스와 AWS IoT Core 간에 메시지를 릴레이할 수 있습니다.
기타 AWS 서비스 액세스	AWS IoT Greengrass V1에서 그룹 역할이라는 AWS Identity and Access Management(IAM) 역할을 Greengrass 그룹에 연결합니다. 그룹 역할은 그룹의 코어 디바이스에 있는 Lambda 함수와 AWS IoT Greengrass 기능이 AWS 서비스 액세스에 사용하는 권한을 정의합니다.	AWS IoT Greengrass V2에서 Greengrass 코어 디바이스에 AWS IoT 역할 별칭을 연결합니다. 역할 별칭은 토큰 교환 역할이라는 IAM 역할을 가리킵니다. 토큰 교환 역할은 코어 디바이스의 Greengrass 구성 요소가 AWS 서비스에 액세스하는데 사용하는 권한을 정의합니다. 자세한 내용은 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여 섹션을 참조하세요.

런타임 선택(Greengrass nucleus 또는 Greengrass nucleus lite)

Greengrass nucleus와 Greengrass nucleus lite 중에서 선택하는 것은 디바이스 리소스와 Lambda 함수가 사용하는 기능에 따라 달라집니다. 다음 표의 이벤트 소스 호환성 매트릭스를 검토한 다음 결정 흐름도를 사용하여 마이그레이션에 적합한 런타임을 결정합니다. Greengrass nucleus 및 Greengrass nucleus lite 기능에 대한 자세한 비교는 [런타임 선택을 참조하세요](#).

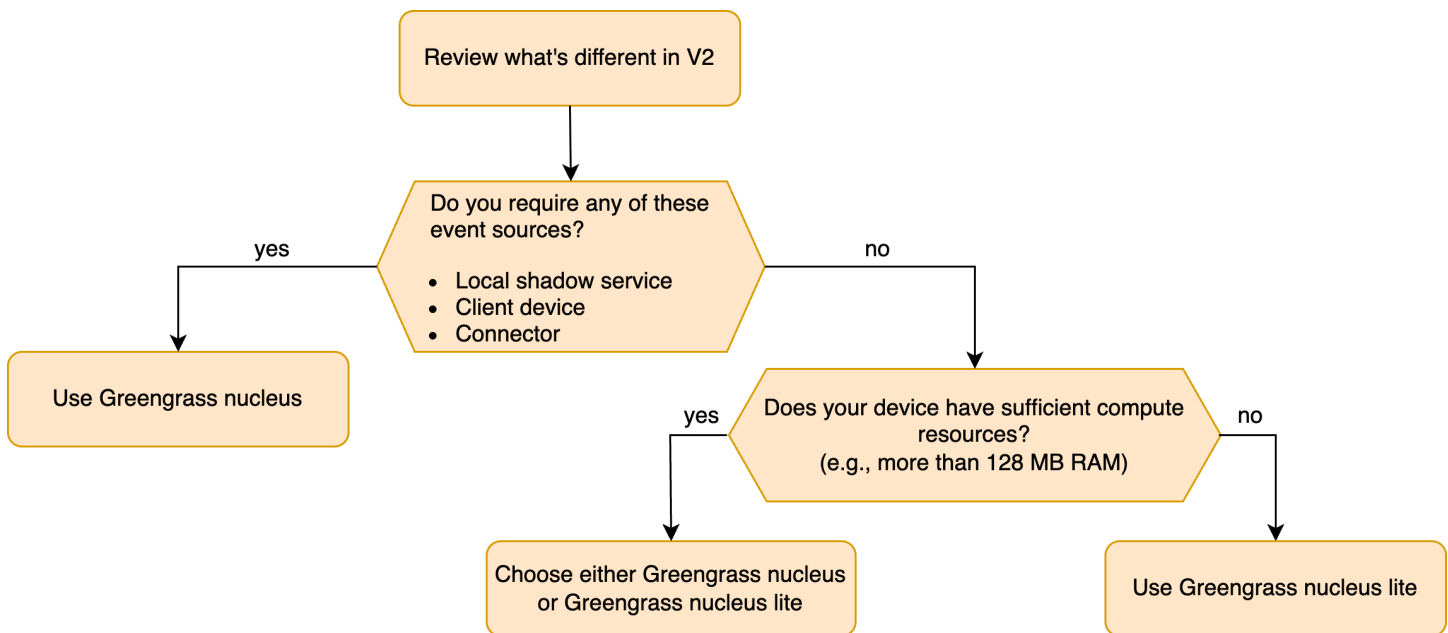
이벤트 소스 호환성 매트릭스

에서 AWS IoT Greengrass V1 Lambda 함수는 다른 Lambda 함수, 로컬 새도우 서비스 AWS IoT Core, 클라이언트 디바이스 및 커넥터의 5가지 유형의 이벤트 소스와 통신할 수 있습니다. 다음 표에는 각 V2 런타임에서 지원되는 이벤트 소스가 나와 있습니다.

참고: 이벤트 소스 이름은 AWS IoT Greengrass V1 용어를 사용합니다. V2로 마이그레이션할 때 Lambda 함수는 Lambda 구성 요소(Greengrass nucleus에서만 지원됨) 또는 일반 구성 요소(Greengrass nucleus 및 Greengrass nucleus lite 모두에서 지원됨)로 변환됩니다.

이벤트 소스	Greengrass nucleus	Greengrass nucleus lite
그룹의 기타 Lambda 함수	✓ (Lambda 구성 요소 및 일반 구성 요소)	✓ (일반 구성 요소만 해당)
AWS IoT Core 서비스	✓	✓
로컬 새도 서비스	✓	✗
클라이언트 장치	✓	✗
커넥터	✓	✗

런타임 선택 결정 흐름



참고

1. Greengrass nucleus lite 요구 사항 및 호환성 세부 정보는 [Greengrass nucleus lite](#)를 참조하세요. Greengrass nucleus lite는 최소 5MB RAM이 필요하며 리소스가 제한된 디바이스용으로 설계되었습니다.
2. 결정 흐름은 일반적인 사용 사례를 기반으로 지침을 제공하지만 엄격한 요구 사항은 아닙니다. 리소스가 제한된 디바이스와 리소스가 충분한 디바이스가 모두 있는 고객은 일부 디바이스가 두 런타임을 모두 지원할 수 있더라도 운영 간소화를 위해 모든 디바이스에서 단일 런타임을 사용하도록 선택할 수 있습니다.

다음 단계

런타임을 선택한 후 테스트 디바이스 설정을 진행합니다.

- Greengrass nucleus 런타임의 경우: [V2에서 V1 애플리케이션을 테스트하도록 새 디바이스 설정](#)
- Greengrass nucleus lite 런타임의 경우: [Greengrass nucleus lite로 새 디바이스 설정](#)

V2에서 V1 애플리케이션을 테스트할 새 디바이스 설정

프로덕션 디바이스에 대한 위험을 최소화하려면 프로덕션 디바이스를 업그레이드하기 전에 V2에서 V1 애플리케이션을 테스트할 새 디바이스를 생성합니다.

이전 단계의 런타임 선택에 따라 다음 설정 가이드 중 하나를 선택합니다.

- 옵션 A - Greengrass nucleus 런타임: Greengrass nucleus를 선택한 [V2에서 V1 애플리케이션을 테스트할 새 디바이스 설정](#) 경우를 따릅니다. 이 옵션을 사용하면 코드 변경을 최소화하면서 Lambda 함수를 Lambda 구성 요소로 가져올 수 있으며 로컬 새도우 서비스, 클라이언트 디바이스 및 커넥터와 같은 V1 기능을 지원합니다.
- 옵션 B - Greengrass nucleus lite 런타임: Greengrass nucleus lite를 선택한 [V2에서 V1 애플리케이션을 테스트하도록 새 디바이스 설정\(Greengrass nucleus lite\)](#) 경우를 따릅니다. 이 옵션을 사용하려면 AWS IoT Device SDK V2 또는 구성 요소 SDK를 사용하여 Lambda 함수를 일반 AWS IoT Greengrass 구성 요소로 변환해야 하지만 리소스가 제한된 디바이스에 최적화되어 있습니다.

V2에서 V1 애플리케이션을 테스트할 새 디바이스 설정

AWS IoT Greengrass V1 애플리케이션에 대한 구성 AWS요소 및 함수를 배포하고 AWS Lambda 테스트하도록 새 AWS IoT Greengrass V2 코어 디바이스를 설정합니다. 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 디바이스 AWS IoT Greengrass V2 에 설치

새 디바이스에 AWS IoT Greengrass 코어 소프트웨어 v2.x를 설치합니다. [시작하기 자습서](#)에 따라 디바이스를 설정하고 구성 요소를 개발하고 배포하는 방법을 알아볼 수 있습니다. 이 자습서에서는 [자동 프로비저닝](#)을 사용하여 디바이스를 빠르게 설정합니다. AWS IoT Greengrass 코어 소프트웨어 v2.x를 설치할 때 디바이스에서 직접 구성 요소를 개발, 테스트 및 디버깅할 수 있도록 [Greengrass CLI](#)를 배포할 `--deploy-dev-tools` 인수를 지정합니다. 프록시 뒤에 AWS IoT Greengrass 코어 소프트웨어를 설치하거나 하드웨어 보안 모듈(HSM)을 사용하는 방법을 포함하여 다른 설치 옵션에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어 설치](#).

(선택 사항) Amazon CloudWatch Logs에 로깅 활성화

V2 코어 디바이스에서 Amazon CloudWatch Logs에 로그를 업로드할 수 있도록 하려면 AWS제공 [로그 관리자 구성 요소](#)를 배포하면 됩니다. CloudWatch Logs를 사용하여 구성 요소 로그를 볼 수 있으므로 코어 디바이스의 파일 시스템에 액세스하지 않고도 디버깅하고 문제를 해결할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

2단계: AWS IoT Greengrass V1 애플리케이션 마이그레이션을 위한 AWS IoT Greengrass V2 구성 요소 생성 및 배포

대부분의 AWS IoT Greengrass V1 애플리케이션은에서 실행할 수 있습니다 AWS IoT Greengrass V2. Lambda 함수를에서 실행되는 구성 요소로 가져올 수 있으며 AWS IoT Greengrass 커넥터 AWS IoT Greengrass V2와 동일한 기능을 제공하는 [AWS제공 구성 요소](#)를 사용할 수 있습니다.

또한 사용자 지정 구성 요소를 개발하여 Greengrass 코어 디바이스에서 실행할 기능이나 런타임을 빌드할 수도 있습니다. 로컬에서 구성 요소를 개발하고 테스트하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 생성](#) 섹션을 참조하세요.

주제

- [V1 Lambda 함수 가져오기](#)
- [V1 커넥터 사용](#)

- [Docker 컨테이너 실행](#)
- [V1 Greengrass 디바이스 연결](#)
- [로컬 새도 서비스 활성화](#)
- [와 통합 AWS IoT SiteWise](#)

V1 Lambda 함수 가져오기

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

- V1 Lambda 함수를 Greengrass 구성 요소로 직접 가져옵니다.
- AWS IoT Device SDK v2에서 Greengrass 라이브러리를 사용하도록 Lambda 함수를 업데이트한 다음 Lambda 함수를 Greengrass 구성 요소로 가져옵니다.
- Lambda가 아닌 코드와 AWS IoT Device SDK v2를 사용하여 Lambda 함수와 동일한 기능을 구현하는 사용자 지정 구성 요소를 생성합니다.

Lambda 함수가 스트림 관리자 또는 로컬 보안 암호와 같은 기능을 사용하는 경우 이러한 기능을 패키징하는 AWS제공 구성 요소에 대한 종속성을 정의해야 합니다. Lambda 함수 구성 요소를 배포하는 경우 배포는 종속성으로 정의한 각 기능의 구성 요소도 포함합니다. 배포에서 코어 디바이스에 배포할 보안 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. V2

- 다른 AWS 서비스에 액세스

Lambda 함수가 자격 AWS 증명을 사용하여 다른 AWS 서비스에 요청하는 경우 코어 디바이스의 토큰 교환 역할은 코어 디바이스가 Lambda 함수가 사용하는 AWS 작업을 수행하도록 허용해야 합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

- 스트림 관리자

Lambda 함수가 스트림 관리자를 사용하는 경우 함수를 가져올 때

`aws.greengrass.StreamManager`를 구성 요소 종속성으로 지정합니다. 스트림 관리자 구성 요소를 배포하는 경우 대상 코어 디바이스에 설정할 스트림 관리자 파라미터를 지정합니다. 코어 디바이스의 토큰 교환 역할은 코어 디바이스가 스트림 관리자와 함께 사용하는 AWS 클라우드 대상에 액세스할 수 있도록 허용해야 합니다. 자세한 내용은 [스트림 관리자](#) 단원을 참조하십시오.

• 로컬 보안 암호

Lambda 함수가 로컬 보안 암호를 사용하는 경우 함수를 가져올 때

`aws.greengrass.SecretManager`를 구성 요소 종속성으로 지정합니다. 보안 암호 관리자 구성 요소를 배포하는 경우 대상 코어 디바이스에 배포할 보안 암호 리소스를 지정합니다. 코어 디바이스의 토큰 교환 역할은 코어 디바이스가 배포할 보안 암호 리소스를 검색하도록 허용해야 합니다. 자세한 내용은 [보안 암호 관리자](#) 단원을 참조하십시오.

Lambda 함수 구성 요소를 배포할 때 AWS IoT Device SDK V2에서 [GetSecretValue](#) IPC 작업을 사용할 수 있는 권한을 부여하는 [IPC 권한 부여 정책](#)을 갖도록 구성합니다.

• 로컬 새도

Lambda 함수가 로컬 새도와 상호 작용하는 경우 AWS IoT Device SDK V2를 사용하도록 Lambda 함수 코드를 업데이트해야 합니다. 함수를 가져오는 경우 `aws.greengrass.ShadowManager`를 구성 요소 종속성으로 지정해야 합니다. 자세한 내용은 [디바이스 새도와 연동](#) 단원을 참조하십시오.

Lambda 함수 구성 요소를 배포할 때 AWS IoT Device SDK V2에서 [새도우 IPC 작업을 사용할 수 있는 권한을 부여하는 IPC 권한 부여 정책](#)을 갖도록 구성합니다.

• 구독

- Lambda 함수가 클라우드 소스의 메시지를 구독하는 경우 함수를 가져올 때 해당 구독을 이벤트 소스로 지정합니다.
- Lambda 함수가 다른 Lambda 함수의 메시지를 구독하거나 Lambda 함수가 AWS IoT Core 또는 다른 Lambda 함수에 메시지를 게시하는 경우 Lambda 함수를 배포할 때 [레거시 구독 라우터 구성 요소](#)를 구성하고 배포합니다. 레거시 구독 라우터 구성 요소를 배포하는 경우 Lambda 함수가 사용하는 구독을 지정합니다.

Note

레거시 구독 라우터 구성 요소는 Lambda 함수가 AWS IoT Greengrass 코어 SDK의 `publish()` 함수를 사용하는 경우에만 필요합니다. Lambda 함수 코드를 업데이트하여 AWS IoT Device SDK V2의 프로세스 간 통신(IPC) 인터페이스를 사용하는 경우 레거시 구독 라우터 구성 요소를 배포할 필요가 없습니다. 자세한 내용은 다음 [프로세스 간 통신](#) 서비스를 참조하세요.

- [로컬 메시지 게시/구독](#)
- [AWS IoT Core MQTT 메시지 게시/구독](#)

- Lambda 함수가 로컬 연결 디바이스의 메시지를 구독하는 경우 함수를 가져올 때 해당 구독을 이벤트 소스로 지정합니다. 또한 [MQTT 브리지 구성 요소](#)를 구성하고 배포하여 연결 디바이스의 메시지를 이벤트 소스로 지정한 로컬 게시/구독 주제로 전달해야 합니다.
- Lambda 함수가 로컬에 연결된 디바이스에 메시지를 게시하는 경우 AWS IoT Device SDK V2를 사용하여 [로컬 게시/구독 메시지를 게시](#)하도록 Lambda 함수 코드를 업데이트해야 합니다. 또한 [로컬 게시/구독 메시지 브로커에서 연결 디바이스로 메시지를 전달하기 위해 MQTT 브리지 구성 요소](#)를 구성하고 배포해야 합니다.
- 로컬 볼륨 및 디바이스

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

자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.

V1 커넥터 사용

일부 AWS IoT Greengrass 커넥터와 동일한 기능을 제공하는 AWS 구성 요소를 배포할 수 있습니다. 배포를 생성하는 경우 커넥터의 파라미터를 구성할 수 있습니다.

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

- [CloudWatch 지표 구성 요소](#)
- [AWS IoT Device Defender component](#)
- [Firehose 구성 요소](#)
- [Modbus-RTU 프로토콜 어댑터 구성 요소](#)
- [Amazon SNS 구성 요소](#)

Docker 컨테이너 실행

AWS IoT Greengrass V2 는 V1 Docker 애플리케이션 배포 커넥터를 직접 대체하는 구성 요소를 제공하지 않습니다. 그러나 Docker 애플리케이션 관리자 구성 요소를 사용하여 Docker 이미지를 다운로드한 다음 다운로드한 이미지에서 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 생성할 수 있습니다. 자세한 내용은 [Docker 컨테이너 실행](#) 및 [Docker 애플리케이션 관리자](#) 섹션을 참조하세요.

V1 Greengrass 디바이스 연결

의 연결된 디바이스 AWS IoT Greengrass V1 를의 클라이언트 디바이스라고 합니다 AWS IoT Greengrass V2. 클라이언트 디바이스에 대한 AWS IoT Greengrass V2 지원은 이전 버전과 호환 AWS

IoT Greengrass V1이므로 애플리케이션 코드를 변경하지 않고도 V1 클라이언트 디바이스를 V2 코어 디바이스에 연결할 수 있습니다. 클라이언트 디바이스가 V2 코어 디바이스에 연결할 수 있도록 하려면 클라이언트 디바이스 지원을 지원하는 Greengrass 구성 요소를 배포하고 클라이언트 디바이스를 코어 디바이스에 연결합니다. 클라이언트 디바이스, AWS IoT Core 클라우드 서비스, Greengrass 구성 요소 (Lambda 함수 포함) 간 메시지를 전달하려면 [MQTT 브리지 구성 요소](#)를 배포하고 구성합니다. [IP 감지기 구성 요소](#)를 배포하여 연결 정보를 자동으로 감지하거나 엔드포인트를 수동으로 관리할 수 있습니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

로컬 새도 서비스 활성화

에서 AWS IoT Greengrass V2로컬 새도 서비스는 AWS제공 새도 관리자 구성 요소에 의해 구현됩니다. 이는 명명된 새도에 대한 지원 AWS IoT Greengrass V2 도 포함됩니다. 구성 요소가 로컬 새도와 상호 작용하고 새도 상태에 동기화하도록 하려면 새도 관리자 구성 요소를 AWS IoT Core구성 및 배포하고 구성 요소 코드에서 새도 IPC 작업을 사용합니다. 자세한 내용은 [디바이스 새도와 연동](#) 단원을 참조하십시오.

와 통합 AWS IoT SiteWise

V1 코어 디바이스를 AWS IoT SiteWise 게이트웨이로 사용하는 경우 [지침에 따라](#) 새 V2 코어 디바이스를 AWS IoT SiteWise 게이트웨이로 설정합니다. AWS IoT SiteWise 는 AWS IoT SiteWise 구성 요소를 배포하는 설치 스크립트를 제공합니다.

3단계: AWS IoT Greengrass V2 애플리케이션 테스트

새로운 V2 코어 디바이스에 V2 구성 요소를 생성하고 배포한 후에는 애플리케이션이 기대치를 충족하는지 확인합니다. 디바이스의 로그를 확인하여 구성 요소의 표준 출력(stdout) 및 표준 오류(stderr) 메시지를 볼 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

[Greengrass CLI](#)를 코어 디바이스에 배포한 경우 이를 사용하여 구성 요소 및 해당 구성을 디버깅할 수 있습니다. 자세한 내용은 [Greengrass CLI 명령](#) 단원을 참조하십시오.

애플리케이션이 V2 코어 디바이스에서 작동하는지 확인한 후에는 애플리케이션의 Greengrass 구성 요소를 다른 코어 디바이스에 배포할 수 있습니다. 네이티브 프로세스 또는 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 개발한 경우 먼저 [해당 구성 요소를 서비스에 게시](#)하여 다른 코어 디바이스에 배포해야 합니다. AWS IoT Greengrass

V2에서 V1 애플리케이션을 테스트하도록 새 디바이스 설정(Greengrass nucleus lite)

Greengrass nucleus lite로 새 디바이스를 설정하여 AWS IoT Greengrass V1 애플리케이션을 V2로 마이그레이션하기 위해 생성한 일반 구성 요소를 테스트합니다. Greengrass nucleus lite는 리소스가 제한된 디바이스에 최적화된 경량 런타임입니다. 이 디바이스를 사용하여 네이티브 프로세스를 실행하는 사용자 지정 Greengrass 구성 요소를 개발하고 테스트할 수 있습니다. Greengrass nucleus lite 디바이스에서 애플리케이션을 테스트한 후 Greengrass nucleus lite를 실행하는 다른 디바이스에 V2 구성 요소를 배포하거나 기존 V1 코어 디바이스를 V2로 업그레이드할 수 있습니다.

1단계: 새 디바이스에 Greengrass nucleus lite 설치

새 디바이스에 Greengrass nucleus lite를 설치합니다. [Greengrass nucleus lite 설치 가이드](#)에 따라 디바이스를 설정합니다.

Note

Greengrass nucleus lite는 현재 로컬 새도우 서비스, 클라이언트 디바이스 또는 커넥터를 지원하지 않습니다. 이 가이드를 진행하기 전에 V1 애플리케이션이 이러한 기능에 의존하지 않는지 확인하세요.

2단계: AWS IoT Greengrass V1 Lambda 함수를 마이그레이션하기 위한 일반 구성 요소 생성 및 배포

Greengrass nucleus lite에서 AWS IoT Greengrass V1 Lambda 함수의 기능을 복제하려면 함수를 일반 구성 요소로 변환해야 합니다. 여기에는 AWS IoT Greengrass 코어 SDK 대신 AWS IoT Device SDK V2 또는 AWS IoT Greengrass 구성 요소 SDK를 사용하도록 Lambda 함수 코드를 다시 작성하는 작업이 포함됩니다.

다음 표에는 이 설명서의 V2 구성 요소 예제에 사용된 SDKs가 나열되어 있습니다.

SDK	최소 버전
AWS IoT Device SDK Python v2용	v1.11.3
AWS IoT Device SDK Java v2용	v1.9.3

SDK	최소 버전
AWS IoT Device SDK JavaScript v2용	v1.12.0
AWS IoT Greengrass 구성 요소 SDK(C/C++)(현재 미리 보기 중)	v0.4.0

아래 예제는 두 가지 주요 시나리오에 대해 여러 프로그래밍 언어로 구성 요소 코드, 레시피 및 빌드 지침과 함께 AWS IoT Greengrass V1 코어 SDK 및 이에 상응하는 일반 구성 요소를 사용하는 Lambda 함수를 보여줍니다.

- 로컬 통신 - 로컬 pub/sub를 사용하여 동일한 디바이스의 다른 구성 요소와 통신하는 구성 요소
- 클라우드 통신 - AWS IoT Core 또는 기타 AWS 서비스와 통신하는 구성 요소

시나리오 1: 로컬 통신(게시자 → 프로세서 → 구독자)

이 시나리오에서는 로컬 pub/sub 통신을 사용하는 V1 Lambda 함수를 V2 일반 구성 요소로 변환하는 방법을 보여줍니다.

애플리케이션 아키텍처

이 예제에는 세 가지 구성 요소가 포함되어 있습니다.

- 게시자 Lambda가 온도 데이터 게시
- 프로세서 Lambda는 데이터를 수신하고 처리합니다.
- 프로세서 Lambda가 처리된 결과를 구독자 Lambda에 게시

아래 코드 예제는 로컬 통신을 위한 메시지 구독 및 게시를 모두 보여주는 프로세서 Lambda에 중점을 둡니다.

V1 그룹 구독

에서 AWS IoT Greengrass V1다음 그룹 구독은 Lambda 함수 간의 통신을 활성화합니다.

구독 1: 게시자 → 프로세서

- 소스: Lambda(게시자)
- 대상: Lambda(프로세서)

- 주제: 센서/온도

구독 2: 프로세서 → 구독자

- 소스: Lambda(프로세서)
- 대상: Lambda(구독자)
- 주제: lambda/alerts

프로세서 Lambda 함수(V1)

Python

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')

def lambda_handler(event, context):
    """
    Receives temperature from publisher Lambda,
    processes it, and forwards to subscriber Lambda
    """
    # Receive from publisher Lambda.
    sensor_id = event['sensor_id']
    temperature = event['temperature']

    print(f"Received from sensor {sensor_id}: {temperature}°F")

    # Process: Check if temperature is high.
    if temperature > 80:
        alert_data = {
            'sensor_id': sensor_id,
            'temperature': temperature,
            'alert': 'HIGH_TEMPERATURE'
        }

        # Publish to another Lambda using greengrasssdk.
        iot_client.publish(
            topic='lambda/alerts',
            payload=json.dumps(alert_data)
        )
```

```
print(f"Alert sent to subscriber Lambda")

return {'statusCode': 200}
```

Java

```
import com.amazonaws.greengrass.javasdk.IotDataClient;
import com.amazonaws.greengrass.javasdk.model.PublishRequest;
import com.amazonaws.services.lambda.runtime.Context;
import com.google.gson.Gson;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;

public class TemperatureProcessorLambda {
    private static final Gson gson = new Gson();
    private final IotDataClient iotDataClient;

    public TemperatureProcessorLambda() {
        this.iotDataClient = new IotDataClient();
    }

    public String handleRequest(Map<String, Object> event, Context context) {
        /*
         * Receives temperature from publisher Lambda,
         * processes it, and forwards to subscriber Lambda
         */

        // Receive from publisher Lambda.
        String sensorId = (String) event.get("sensor_id");
        Number temp = (Number) event.get("temperature");
        int temperature = temp.intValue();

        System.out.println("Received from sensor " + sensorId + ": " + temperature +
            "°F");

        // Process: Check if temperature is high.
        if (temperature > 80) {
            Map<String, Object> alertData = new HashMap<>();
            alertData.put("sensor_id", sensorId);
            alertData.put("temperature", temperature);
```

```

        alertData.put("alert", "HIGH_TEMPERATURE");

        // Publish to another Lambda using greengrasssdk.
        String payload = gson.toJson(alertData);
        PublishRequest publishRequest = new PublishRequest()
            .withTopic("lambda/alerts")

        .withPayload(ByteBuffer.wrap(payload.getBytes(StandardCharsets.UTF_8)));

        iotDataClient.publish(publishRequest);

        System.out.println("Alert sent to subscriber Lambda");
    }

    return "Success";
}
}

```

JavaScript

```

const greengrasssdk = require('aws-greengrass-core-sdk');

const iotClient = new greengrasssdk.IotData();

/**
 * Greengrass v1 Lambda function
 * Receives temperature from publisher Lambda,
 * processes it, and forwards to subscriber Lambda
 */
exports.handler = function(event, context) {
    // Receive from publisher Lambda.
    const sensorId = event.sensor_id;
    const temperature = event.temperature;

    console.log(`Received from sensor ${sensorId}: ${temperature}°F`);

    // Process: Check if temperature is high.
    if (temperature > 80) {
        const alertData = {
            sensor_id: sensorId,
            temperature: temperature,
            alert: 'HIGH_TEMPERATURE'
        };
    };
}

```

```

// Publish to another Lambda using greengrasssdk.
const params = {
  topic: 'lambda/alerts',
  payload: JSON.stringify(alertData)
};

iotClient.publish(params, (err) => {
  if (err) {
    console.error('Error publishing alert:', err);
    context.fail(err);
  } else {
    console.log('Alert sent to subscriber Lambda');
    context.succeed('Success');
  }
});
} else {
  context.succeed('Success');
}
};

```

C

```

#include <aws/greengrass/greengrasssdk.h>
#include <stdio.h>
#include <string.h>
#include <jansson.h> // For JSON parsing.

static aws_greengrass_iot_data_client *iot_client = NULL;

void on_message_received(const char *topic, const uint8_t *payload, size_t
payload_len, void *user_data) {
  // Parse the incoming message.
  char *payload_str = strdup((char *)payload, payload_len);
  json_error_t error;
  json_t *event = json_loads(payload_str, 0, &error);
  free(payload_str);

  if (!event) {
    fprintf(stderr, "Error parsing JSON: %s\n", error.text);
    return;
  }
}

```

```
// Receive from publisher Lambda.
json_t *sensor_id_obj = json_object_get(event, "sensor_id");
json_t *temperature_obj = json_object_get(event, "temperature");

const char *sensor_id = json_string_value(sensor_id_obj);
int temperature = json_integer_value(temperature_obj);

printf("Received from sensor %s: %d°F\n", sensor_id, temperature);

// Process: Check if temperature is high.
if (temperature > 80) {
    // Create alert data.
    json_t *alert_data = json_object();
    json_object_set_new(alert_data, "sensor_id", json_string(sensor_id));
    json_object_set_new(alert_data, "temperature", json_integer(temperature));
    json_object_set_new(alert_data, "alert", json_string("HIGH_TEMPERATURE"));

    // Convert to JSON string.
    char *alert_payload = json_dumps(alert_data, JSON_COMPACT);

    // Publish to another Lambda using greengrasssdk.
    aws_greengrass_publish_params params = {
        .topic = "lambda/alerts",
        .payload = (uint8_t *)alert_payload,
        .payload_len = strlen(alert_payload)
    };

    aws_greengrass_iot_data_publish(iot_client, &params);

    printf("Alert sent to subscriber Lambda\n");

    free(alert_payload);
    json_decref(alert_data);
}

json_decref(event);
}

int main(int argc, char *argv[]) {
    // Initialize Greengrass SDK.
    iot_client = aws_greengrass_iot_data_client_new();

    // Subscribe to temperature sensor topic.
    aws_greengrass_subscribe_params subscribe_params = {
```

```

        .topic = "sensors/temperature",
        .callback = on_message_received,
        .user_data = NULL
    };

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

    printf("Temperature Processor Lambda started\n");
    printf("Subscribed to sensors/temperature\n");
    printf("Waiting for sensor data...\n");

    // Keep the Lambda running.
    while (1) {
        sleep(1);
    }

    return 0;
}

```

C++

```

#include <aws/greengrass/greengrasssdk.h>
#include <iostream>
#include <string>
#include <memory>
#include <jansson.h> // For JSON parsing.
#include <unistd.h>

class TemperatureProcessor {
private:
    std::unique_ptr<aws_greengrass_iot_data_client,
        decltype(&aws_greengrass_iot_data_client_destroy)> iot_client;

    static void message_callback_wrapper(const char *topic,
        const uint8_t *payload,
        size_t payload_len,
        void *user_data) {
        auto* processor = static_cast<TemperatureProcessor*>(user_data);
        processor->on_message_received(topic, payload, payload_len);
    }

public:
    TemperatureProcessor()

```

```

        : iot_client(aws_greengrass_iot_data_client_new(),
                    aws_greengrass_iot_data_client_destroy) {
    if (!iot_client) {
        throw std::runtime_error("Failed to create Greengrass IoT client");
    }
}

void on_message_received(const char *topic,
                        const uint8_t *payload,
                        size_t payload_len) {
    // Parse the incoming message.
    std::string payload_str(reinterpret_cast<const char*>(payload),
payload_len);

    json_error_t error;
    json_t *event = json_loads(payload_str.c_str(), 0, &error);

    if (!event) {
        std::cerr << "Error parsing JSON: " << error.text << std::endl;
        return;
    }

    json_t *sensor_id_obj = json_object_get(event, "sensor_id");
    json_t *temperature_obj = json_object_get(event, "temperature");

    const char *sensor_id = json_string_value(sensor_id_obj);
    int temperature = json_integer_value(temperature_obj);

    std::cout << "Received from sensor " << sensor_id
              << ": " << temperature << "°F" << std::endl;

    if (temperature > 80) {
        send_alert(sensor_id, temperature);
    }

    json_decref(event);
}

void send_alert(const char *sensor_id, int temperature) {
    // Create alert data.
    json_t *alert_data = json_object();
    json_object_set_new(alert_data, "sensor_id", json_string(sensor_id));
    json_object_set_new(alert_data, "temperature", json_integer(temperature));
    json_object_set_new(alert_data, "alert", json_string("HIGH_TEMPERATURE"));
}

```

```

    // Convert to JSON string.
    char *alert_payload = json_dumps(alert_data, JSON_COMPACT);

    // Publish to another Lambda using greengrasssdk.
    aws_greengrass_publish_params params = {
        .topic = "lambda/alerts",
        .payload = reinterpret_cast<uint8_t*>(alert_payload),
        .payload_len = strlen(alert_payload)
    };

    aws_greengrass_iot_data_publish(iot_client.get(), &params);

    std::cout << "Alert sent to subscriber Lambda" << std::endl;

    free(alert_payload);
    json_decref(alert_data);
}

void subscribe_to_topic(const std::string& topic) {
    aws_greengrass_subscribe_params subscribe_params = {
        .topic = topic.c_str(),
        .callback = message_callback_wrapper,
        .user_data = this
    };

    aws_greengrass_iot_data_subscribe(iot_client.get(), &subscribe_params);

    std::cout << "Temperature Processor Lambda started" << std::endl;
    std::cout << "Subscribed to " << topic << std::endl;
    std::cout << "Waiting for sensor data..." << std::endl;
}

void run() {
    // Keep the Lambda running.
    while (true) {
        sleep(1);
    }
}

};

int main(int argc, char *argv[]) {
    try {
        TemperatureProcessor processor;

```



```

        processor.subscribe_to_topic("sensors/temperature");
        processor.run();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```

일반 구성 요소(V2)

에서 동일한 기능을 구현하려면 다음을 사용하여 일반 구성 요소를 AWS IoT Greengrass V2 생성합니다.

1. 구성 요소 코드

Python

사전 조건: 이 구성 요소 코드를 사용하기 전에 Greengrass 디바이스에 Python AWS IoT Device SDK 용을 설치하고 확인합니다.

```

# Install the SDK
pip3 install awsiotsdk

# Verify installation
python3 -c "import awsiot.greengrasscoreipc.clientv2; print('SDK installed successfully')"
```

설치 중에 종속성 충돌이 발생하는 경우 특정 버전의를 설치해 보십시오 AWS IoT Device SDK.

확인 명령이 "SDK installed successfully"를 인쇄하면 구성 요소 코드를 사용할 준비가 된 것입니다.

```

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    JsonMessage
)
import time

ipc_client = GreengrassCoreIPCClientV2()
```

```
def on_sensor_data(event):
    """
    Receives temperature from sensor publisher component,
    processes it, and forwards to alert component
    """
    try:
        # Receive from publisher component.
        data = event.json_message.message
        sensor_id = data['sensor_id']
        temperature = data['temperature']

        print(f"Received from sensor {sensor_id}: {temperature}°F")

        # Process: Check if temperature is high.
        if temperature > 80:
            alert_data = {
                'sensor_id': sensor_id,
                'temperature': temperature,
                'alert': 'HIGH_TEMPERATURE'
            }

            # Publish to another component (AlertHandler).
            ipc_client.publish_to_topic(
                topic='component/alerts',
                publish_message=PublishMessage(
                    json_message=JsonMessage(message=alert_data)
                )
            )

            print(f"Alert sent to AlertHandler component")

    except Exception as e:
        print(f"Error processing sensor data: {e}")

def main():
    print("Temperature Processor component starting...")

    # Subscribe to sensor data from publisher component.
    ipc_client.subscribe_to_topic(
        topic='sensors/temperature',
        on_stream_event=on_sensor_data
    )

    print("Subscribed to sensors/temperature")
```

```
print("Waiting for sensor data...")

# Keep running.
while True:
    time.sleep(1)

if __name__ == '__main__':
    main()
```

Java

```
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.JsonMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

public class TemperatureProcessor {
    private static GreengrassCoreIPCClientV2 ipcClient;

    public static void main(String[] args) {
        System.out.println("Temperature Processor component starting...");

        try (GreengrassCoreIPCClientV2 client =
GreengrassCoreIPCClientV2.builder().build()) {
            ipcClient = client;

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest()
                .withTopic("sensors/temperature");

            ipcClient.subscribeToTopic(
                subscribeRequest,
                TemperatureProcessor::onSensorData,
                Optional.empty(),
                Optional.empty()
            );

            System.out.println("Subscribed to sensors/temperature");
            System.out.println("Waiting for sensor data...");
```

```

        while (true) {
            Thread.sleep(1000);
        }
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
        e.printStackTrace();
    }
}

public static void onSensorData(SubscriptionResponseMessage message) {
    try {
        Map<String, Object> data = message.getJsonMessage().getMessage();
        String sensorId = (String) data.get("sensor_id");
        Number temp = (Number) data.get("temperature");
        int temperature = temp.intValue();

        System.out.println("Received from sensor " + sensorId + ": " +
            temperature + "F");

        if (temperature > 80) {
            Map<String, Object> alertData = new HashMap<>();
            alertData.put("sensor_id", sensorId);
            alertData.put("temperature", temperature);
            alertData.put("alert", "HIGH_TEMPERATURE");

            JsonMessage jsonMessage = new JsonMessage().withMessage(alertData);
            PublishMessage publishMessage = new
PublishMessage().withJsonMessage(jsonMessage);
            PublishToTopicRequest publishRequest = new PublishToTopicRequest()
                .withTopic("component/alerts")
                .withPublishMessage(publishMessage);

            ipcClient.publishToTopic(publishRequest);
            System.out.println("Alert sent to AlertHandler component");
        }
    } catch (Exception e) {
        System.err.println("Error processing sensor data: " + e.getMessage());
    }
}
}

```

JavaScript

```
const greengrasscoreipc = require('aws-iot-device-sdk-v2').greengrasscoreipc;

class TemperatureProcessor {
  constructor() {
    this.ipcClient = null;
  }

  async start() {
    console.log('Temperature Processor component starting...');

    try {
      this.ipcClient = greengrasscoreipc.createClient();
      await this.ipcClient.connect();

      const subscribeRequest = {
        topic: 'sensors/temperature'
      };

      const streamingOperation =
this.ipcClient.subscribeToTopic(subscribeRequest);

      streamingOperation.on('message', (message) => {
        this.onSensorData(message);
      });

      streamingOperation.on('streamError', (error) => {
        console.error('Stream error:', error);
      });

      streamingOperation.on('ended', () => {
        console.log('Subscription stream ended');
      });

      await streamingOperation.activate();

      console.log('Subscribed to sensors/temperature');
      console.log('Waiting for sensor data...');

    } catch (error) {
      console.error('Error starting component:', error);
      process.exit(1);
    }
  }
}
```

```
}

async onSensorData(message) {
  try {
    const data = message.jsonMessage.message;

    const sensorId = data.sensor_id;
    const temperature = data.temperature;

    console.log(`Received from sensor ${sensorId}: ${temperature}°F`);

    if (temperature > 80) {
      const alertData = {
        sensor_id: sensorId,
        temperature: temperature,
        alert: 'HIGH_TEMPERATURE'
      };

      const publishRequest = {
        topic: 'component/alerts',
        publishMessage: {
          jsonMessage: {
            message: alertData
          }
        }
      };

      await this.ipcClient.publishToTopic(publishRequest);
      console.log('Alert sent to AlertHandler component');
    }

  } catch (error) {
    console.error('Error processing sensor data:', error);
  }
}

// Start the component.
const processor = new TemperatureProcessor();
processor.start();
```

C

```

#include <gg/buffer.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/map.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

#define SUBSCRIBE_TOPIC "sensors/temperature"
#define PUBLISH_TOPIC "component/alerts"

typedef struct {
    char sensor_id[64];
    int64_t temperature;
} AlertData;

static pthread_mutex_t alert_mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t alert_cond = PTHREAD_COND_INITIALIZER;
static AlertData pending_alert;
static bool has_pending_alert = false;

static void *alert_publisher_thread(void *arg) {
    (void) arg;

    while (true) {
        pthread_mutex_lock(&alert_mutex);
        while (!has_pending_alert) {
            pthread_cond_wait(&alert_cond, &alert_mutex);
        }

        AlertData alert = pending_alert;
        has_pending_alert = false;
        pthread_mutex_unlock(&alert_mutex);

        GgBuffer sensor_id_buf = { .data = (uint8_t *)alert.sensor_id, .len =
strlen(alert.sensor_id) };
        GgMap payload = GG_MAP(

```

```

        gg_kv(GG_STR("sensor_id"), gg_obj_buf(sensor_id_buf)),
        gg_kv(GG_STR("temperature"), gg_obj_i64(alert.temperature)),
        gg_kv(GG_STR("alert"), gg_obj_buf(GG_STR("HIGH_TEMPERATURE")))
    );

    GgError ret = ggipc_publish_to_topic_json(GG_STR(PUBLISH_TOPIC), payload);

    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to publish alert\n");
    } else {
        printf("Alert sent to AlertHandler component\n");
    }
}

return NULL;
}

static void on_sensor_data(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) topic;
    (void) handle;

    if (gg_obj_type(payload) != GG_TYPE_MAP) {
        fprintf(stderr, "Expected JSON message\n");
        return;
    }

    GgMap map = gg_obj_into_map(payload);

    GgObject *sensor_id_obj;
    if (!gg_map_get(map, GG_STR("sensor_id"), &sensor_id_obj)) {
        fprintf(stderr, "Missing sensor_id field\n");
        return;
    }
    GgBuffer sensor_id = gg_obj_into_buf(*sensor_id_obj);

    GgObject *temperature_obj;
    if (!gg_map_get(map, GG_STR("temperature"), &temperature_obj)) {
        fprintf(stderr, "Missing temperature field\n");
        return;
    }
    int64_t temperature = gg_obj_into_i64(*temperature_obj);

```



```

printf("Received from sensor %.*s: %lld°F\n",
      (int)sensor_id.len, sensor_id.data, (long long)temperature);

if (temperature > 80) {
    pthread_mutex_lock(&alert_mutex);
    snprintf(pending_alert.sensor_id, sizeof(pending_alert.sensor_id),
            "%.*s", (int)sensor_id.len, sensor_id.data);
    pending_alert.temperature = temperature;
    has_pending_alert = true;
    pthread_cond_signal(&alert_cond);
    pthread_mutex_unlock(&alert_mutex);
}
}

int main(void) {
    setvbuf(stdout, NULL, _IONBF, 0);
    printf("Temperature Processor component starting...\n");

    gg_sdk_init();

    GgError ret = ggipc_connect();
    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to connect to Greengrass nucleus\n");
        exit(1);
    }
    printf("Connected to Greengrass IPC\n");

    // Start alert publisher thread.
    pthread_t alert_thread;
    if (pthread_create(&alert_thread, NULL, alert_publisher_thread, NULL) != 0) {
        fprintf(stderr, "Failed to create alert publisher thread\n");
        exit(1);
    }

    GgIpcSubscriptionHandle handle;
    ret = ggipc_subscribe_to_topic(
        GG_STR(SUBSCRIBE_TOPIC),
        &on_sensor_data,
        NULL,
        &handle
    );

    if (ret != GG_ERR_OK) {

```

```

        fprintf(stderr, "Failed to subscribe to topic\n");
        exit(1);
    }

    printf("Subscribed to %s\n", SUBSCRIBE_TOPIC);
    printf("Waiting for sensor data...\n");

    // Keep running.
    while (true) {
        sleep(1);
    }

    return 0;
}

```

C++

```

#include <gg/ipc/client.hpp>
#include <gg/buffer.hpp>
#include <gg/object.hpp>
#include <gg/types.hpp>

#include <chrono>
#include <condition_variable>
#include <iostream>
#include <mutex>
#include <string>
#include <string_view>
#include <thread>

struct AlertData {
    std::string sensor_id;
    int64_t temperature;
};

static std::mutex alert_mutex;
static std::condition_variable alert_cv;
static AlertData pending_alert;
static bool has_pending_alert = false;

void alert_publisher_thread() {
    auto& client = gg::ipc::Client::get();

```

```

while (true) {
    std::unique_lock<std::mutex> lock(alert_mutex);
    alert_cv.wait(lock, [] { return has_pending_alert; });

    AlertData alert = pending_alert;
    has_pending_alert = false;
    lock.unlock();

    // Create alert payload as JSON string.
    std::string json_payload = "{\"sensor_id\":\"" + alert.sensor_id +
                                "\", \"temperature\":\"" +
std::to_string(alert.temperature) +
                                "\", \"alert\":\"HIGH_TEMPERATURE\"}";

    // Convert to Buffer and publish.
    gg::Buffer buffer(json_payload);
    auto error = client.publish_to_topic("component/alerts", buffer);

    if (error) {
        std::cerr << "Failed to publish alert\n";
    } else {
        std::cout << "Alert sent to AlertHandler component\n";
    }
}

}

class SensorCallback : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription& handle
    ) override {
        (void) topic;
        (void) handle;

        // Payload is a Buffer containing JSON string.
        if (payload.index() != GG_TYPE_BUF) {
            std::cerr << "Expected Buffer message\n";
            return;
        }

        // Extract buffer using gg::get.
        auto buffer = gg::get<std::span<uint8_t>>(payload);

```

```

        std::string json_str(reinterpret_cast<const char*>(buffer.data()),
buffer.size());

        // Simple JSON parsing for demo.
        std::string sensor_id = "sensor1";
        int64_t temperature = 0;

        // Extract temperature (simple string search).
        size_t temp_pos = json_str.find("\"temperature\":");
        if (temp_pos != std::string::npos) {
            temp_pos += 14; // Skip "temperature".
            size_t end_pos = json_str.find_first_of(",}", temp_pos);
            if (end_pos != std::string::npos) {
                temperature = std::stoll(json_str.substr(temp_pos, end_pos -
temp_pos));
            }
        }

        std::cout << "Received from sensor " << sensor_id << ": "
            << temperature << "°F\n";

        if (temperature > 80) {
            std::lock_guard<std::mutex> lock(alert_mutex);
            pending_alert = {sensor_id, temperature};
            has_pending_alert = true;
            alert_cv.notify_one();
        }
    }
};

int main() {
    // Disable stdout buffering for real-time logging.
    std::cout.setf(std::ios::unitbuf);

    std::cout << "Temperature Processor component starting..." << std::endl;

    auto& client = gg::ipc::Client::get();
    std::cout << "Got client instance" << std::endl;

    auto error = client.connect();
    std::cout << "Connect returned, error code: " << error.value() << std::endl;

    if (error) {

```

```

        std::cerr << "Failed to connect to Greengrass nucleus: " << error.message()
<< std::endl;
        return 1;
    }

    std::cout << "Connected to Greengrass IPC" << std::endl;

    // Start alert publisher thread.
    std::thread alert_thread(alert_publisher_thread);
    alert_thread.detach();

    // Handler must be static lifetime if subscription handle is not held.
    static SensorCallback handler;
    error = client.subscribe_to_topic("sensors/temperature", handler);

    if (error) {
        std::cerr << "Failed to subscribe to topic: " << error.message() <<
std::endl;
        return 1;
    }

    std::cout << "Subscribed to sensors/temperature" << std::endl;
    std::cout << "Waiting for sensor data..." << std::endl;

    // Keep running.
    while (true) {
        using namespace std::chrono_literals;
        std::this_thread::sleep_for(1s);
    }

    return 0;
}

```

2. 구성 요소 빌드 및 패키징

일부 언어는 배포 전에 빌드 또는 패키징이 필요합니다.

Python

Python은 컴파일이 필요하지 않습니다. 구성 요소는 .py 파일을 직접 사용할 수 있습니다.

Java

모든 종속성이 번들로 제공되는 실행 파일 JAR을 빌드하려면:

1. 프로젝트 디렉터리에 pom.xml 파일을 생성합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- Basic project information: organization, component name, and version --
>
  <groupId>com.example</groupId>
  <artifactId>temperature-processor</artifactId>
  <version>1.0.0</version>

  <properties>
    <!-- Java 11 LTS (Long Term Support) is recommended for Greengrass v2
components -->
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- AWS IoT Device SDK for Java - provides IPC client for Greengrass v2
local communication -->
    <dependency>
      <groupId>software.amazon.awssdk.iotdevicesdk</groupId>
      <artifactId>aws-iot-device-sdk</artifactId>
      <version>1.25.1</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- Maven Shade Plugin - creates a standalone JAR with all
dependencies included for Greengrass deployment -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.4</version>
        <executions>
          <execution>
            <phase>package</phase>
```

```

        <goals>
          <goal>shade</goal>
        </goals>
        <configuration>
          <transformers>
            <!-- Set the main class for the executable JAR
-->
            <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
              <mainClass>TemperatureProcessor</mainClass>
            </transformer>
          </transformers>
          <filters>
            <!-- Exclude signature files to avoid security
exceptions -->
            <filter>
              <artifact>*:*</artifact>
              <excludes>
                <exclude>META-INF/*.SF</exclude>
                <exclude>META-INF/*.DSA</exclude>
                <exclude>META-INF/*.RSA</exclude>
              </excludes>
            </filter>
          </filters>
        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>
</project>

```

2. JAR을 빌드합니다.

```
mvn clean package
```

이렇게 하면 모든 종속성이 포함된 target/temperature-processor-1.0.0.jar이 생성됩니다.

3. 배포를 위해 JAR을 S3 버킷에 업로드합니다.

JavaScript

Node.js 구성 요소를 모든 종속성으로 패키징하려면:

1. package.json 파일 생성:

```
{
  "name": "temperature-processor",
  "version": "1.0.0",
  "description": "Temperature processor component for Greengrass v2",
  "main": "TemperatureProcessor.js",
  "dependencies": {
    "aws-iot-device-sdk-v2": "^1.21.0"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}
```

2. 개발 시스템에 종속성을 설치합니다.

```
npm install
```

이렇게 하면 AWS IoT Device SDK v2가 포함된 node_modules 폴더가 생성됩니다.

3. 배포용 패키지:

```
zip -r TemperatureProcessor.zip TemperatureProcessor.js node_modules/
package.json
```

4. 배포를 위해 zip 파일을 S3 버킷에 업로드합니다.

Note

Greengrass 디바이스에는 Node.js 런타임(버전 14 이상)이 설치되어 있어야 합니다. 구성 요소 아티팩트 `npm install`에는 번들링된 node_modules 폴더의 모든 종속성이 포함되므로 Greengrass 코어 디바이스에서 실행할 필요가 없습니다.

C

사전 조건:

SDK와 구성 요소를 빌드하려면 다음과 같은 빌드 종속성이 필요합니다.

- GCC 또는 Clang
- CMake(최소 버전 3.22)
- 또는 Ninja 만들기

빌드 종속성 설치:

Ubuntu/Debian의 경우:

```
sudo apt install build-essential cmake
```

Amazon Linux에서:

```
sudo yum install gcc cmake make
```

구성 요소에 대한 CMakeLists.txt 파일을 생성합니다.

```
cmake_minimum_required(VERSION 3.10)
project(TemperatureProcessor C)

set(CMAKE_C_STANDARD 11)

# Add AWS Greengrass Component SDK
add_subdirectory(aws-greengrass-component-sdk)

# Build your component executable
add_executable(temperature_processor temperature_processor.c)
target_link_libraries(temperature_processor gg-sdk)
```

빌드 단계:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
```

```
make -C build -j$(nproc)

# The binary 'temperature_processor' is in ./build/
# Upload this binary to S3 for deployment
```

C++

사전 조건:

SDK와 구성 요소를 빌드하려면 다음과 같은 빌드 종속성이 필요합니다.

- C++20을 지원하는 GCC 또는 Clang
- CMake(최소 버전 3.22)
- 또는 Ninja 만들기

빌드 종속성 설치:

Ubuntu/Debian의 경우:

```
sudo apt install build-essential cmake
```

Amazon Linux에서:

```
sudo yum install gcc-c++ cmake make
```

구성 요소에 대한 CMakeLists.txt 파일을 생성합니다.

```
cmake_minimum_required(VERSION 3.10)
project(TemperatureProcessor CXX)

set(CMAKE_CXX_STANDARD 20)

# Add SDK as subdirectory
add_subdirectory(aws-greengrass-component-sdk)

# Add C++ SDK subdirectory
add_subdirectory(aws-greengrass-component-sdk/cpp)

add_executable(temperature_processor temperature_processor.cpp)
target_link_libraries(temperature_processor gg-sdk++)
```

빌드 단계:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'temperature_processor' will be in ./build/
# Upload this binary to S3 for deployment
```

3. 구성 요소 레시피

구성 요소에서 사용하는 실제 주제로 "리소스" 배열을 업데이트합니다.

Python

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
```

```

        "aws.greengrass#PublishToTopic"
    ],
    "resources": [
        "component/alerts"
    ]
    }
}
},
"Manifests": [
    {
        "Platform": {
            "os": "linux",
            "runtime": "*"
        },
        "Lifecycle": {
            "run": "python3 -u {artifacts:path}/temperature_processor.py"
        },
        "Artifacts": [
            {
                "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/temperature_processor.py"
            }
        ]
    }
]
}

```

Java

```

{
    "RecipeFormatVersion": "2020-01-25",
    "ComponentName": "com.example.TemperatureProcessor",
    "ComponentVersion": "1.0.0",
    "ComponentType": "aws.greengrass.generic",
    "ComponentDescription": "Receives sensor data and forwards alerts to AlertHandler",
    "ComponentPublisher": "[Your Company]",
    "ComponentConfiguration": {
        "DefaultConfiguration": {
            "accessControl": {
                "aws.greengrass.ipc.pubsub": {

```

```

    "com.example.TemperatureProcessor:pubsub:1": {
      "policyDescription": "Allows access to subscribe to sensor topics",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "sensors/temperature"
      ]
    },
    "com.example.TemperatureProcessor:pubsub:2": {
      "policyDescription": "Allows access to publish to alert topics",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "component/alerts"
      ]
    }
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "java -jar {artifacts:path}/TemperatureProcessor.jar"
    },
    "Artifacts": [
      {
        "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/
TemperatureProcessor.jar"
      }
    ]
  }
]
}

```

JavaScript

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "component/alerts"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {

```

```

    "run": "cd {artifacts:decompressedPath}/TemperatureProcessor && node
TemperatureProcessor.js"
  },
  "Artifacts": [
    {
      "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/
TemperatureProcessor.zip",
      "Unarchive": "ZIP"
    }
  ]
}
]
}
}

```

C/C++

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [

```

```

        "component/alerts"
      ]
    }
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "{artifacts:path}/temperature_processor"
    },
    "Artifacts": [
      {
        "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/temperature_processor"
      }
    ]
  }
]
}

```

시나리오 2: 클라우드 통신

이 시나리오는와 통신하는 V1 Lambda 함수를 V2 일반 구성 요소로 변환하는 AWS IoT Core 방법을 보여줍니다.

애플리케이션 아키텍처

이 예제에서는 클라우드 연결 아키텍처를 사용합니다.

- AWS IoT Core 는 디바이스에 명령을 전송합니다.
- 컨트롤러 Lambda가 명령을 수신하고 처리합니다.
- 컨트롤러 Lambda가 원격 측정 데이터를 로 다시 전송 AWS IoT Core

이 예제에서는 컨트롤러 Lambda에 중점을 둡니다. 컨트롤러 Lambda는에서 메시지를 수신하고에 메시지를 게시하는 것을 모두 보여줍니다 AWS IoT Core.

V1 그룹 구독

에서 AWS IoT Greengrass V1다음 그룹 구독은 Lambda 함수와 간의 통신을 활성화합니다 AWS IoT Core.

구독 1: IoT Cloud → Lambda

- 소스: IoT Cloud
- 대상: Lambda(DeviceController)
- 주제: 명령/device1

구독 2: Lambda → IoT Cloud

- 소스: Lambda(DeviceController)
- 대상: IoT Cloud
- 주제: 원격 측정/디바이스1

컨트롤러 Lambda 함수(V1)

Python

```
import greengrasssdk
import json
import time

iot_client = greengrasssdk.client('iot-data')

def lambda_handler(event, context):
    """
    Receives commands from IoT Core,
    processes them, and sends telemetry back to cloud
    """
    # Receive command from IoT Core.
    command = event.get('command')
    device_id = event.get('device_id', 'device1')

    print(f"Received command from cloud: {command}")

    # Process command.
    if command == 'get_status':
```

```

        status = get_device_status()

        # Send telemetry back to IoT Core.
        telemetry_data = {
            'device_id': device_id,
            'status': status,
            'timestamp': time.time()
        }

        iot_client.publish(
            topic=f'telemetry/{device_id}',
            payload=json.dumps(telemetry_data)
        )

        print(f"Telemetry sent to cloud: {telemetry_data}")

    return {'statusCode': 200}

def get_device_status():
    """Get current device status"""
    # Simulate getting device status.
    return 'online'

```

Java

```

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.greengrass.javasdk.IotDataClient;
import com.amazonaws.greengrass.javasdk.model.PublishRequest;
import com.google.gson.Gson;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;

public class DeviceControllerLambda {
    private static final Gson gson = new Gson();
    private final IotDataClient iotDataClient;

    public DeviceControllerLambda() {
        this.iotDataClient = new IotDataClient();
    }

    public String handleRequest(Map<String, Object> event, Context context) {

```

```
/*
 * Receives commands from IoT Core,
 * processes them, and sends telemetry back to cloud
 */

// Receive command from IoT Core.
String command = (String) event.get("command");
String deviceId = event.containsKey("device_id") ?
    (String) event.get("device_id") : "device1";

System.out.println("Received command from cloud: " + command);

// Process command.
if ("get_status".equals(command)) {
    String status = getDeviceStatus();

    // Send telemetry back to IoT Core.
    Map<String, Object> telemetryData = new HashMap<>();
    telemetryData.put("device_id", deviceId);
    telemetryData.put("status", status);
    telemetryData.put("timestamp", System.currentTimeMillis() / 1000.0);

    String payload = gson.toJson(telemetryData);
    PublishRequest publishRequest = new PublishRequest()
        .withTopic("telemetry/" + deviceId)

.withPayload(ByteBuffer.wrap(payload.getBytes(StandardCharsets.UTF_8)));

    iotDataClient.publish(publishRequest);

    System.out.println("Telemetry sent to cloud: " + telemetryData);
}

return "Success";
}

private String getDeviceStatus() {
    // Simulate getting device status.
    return "online";
}
}
```

JavaScript

```
const greengrasssdk = require('aws-greengrass-core-sdk');

const iotClient = new greengrasssdk.IotData();

/**
 * Receives commands from IoT Core and sends telemetry back.
 */
exports.handler = function(event, context) {
  console.log('Received command from IoT Core:', JSON.stringify(event));

  const command = event.command;
  const deviceId = event.device_id || 'device1';

  console.log(`Processing command: ${command}`);

  if (command === 'get_status') {
    const status = 'online';

    const telemetryData = {
      device_id: deviceId,
      status: status,
      timestamp: Date.now() / 1000
    };

    // Publish telemetry to IoT Core using greengrasssdk.
    const params = {
      topic: `telemetry/${deviceId}`,
      payload: JSON.stringify(telemetryData)
    };

    iotClient.publish(params, (err) => {
      if (err) {
        console.error('Error publishing telemetry:', err);
        context.fail(err);
      } else {
        console.log('Telemetry sent to IoT Core:',
          JSON.stringify(telemetryData));
        context.succeed('Success');
      }
    });
  } else {
    context.succeed('Success');
  }
}
```

```

    }
};

```

C

```

#include <aws/greengrass/greengrasssdk.h>
#include <stdio.h>
#include <string.h>
#include <jansson.h>
#include <time.h>

static aws_greengrass_iot_data_client *iot_client = NULL;

const char* get_device_status(void) {
    // Simulate getting device status.
    return "online";
}

void on_cloud_command(const char *topic, const uint8_t *payload, size_t payload_len,
void *user_data) {
    // Parse incoming command from IoT Core.
    char *payload_str = strdup((char *)payload, payload_len);
    json_error_t error;
    json_t *event = json_loads(payload_str, 0, &error);
    free(payload_str);

    if (!event) {
        fprintf(stderr, "Error parsing JSON: %s\n", error.text);
        return;
    }

    // Extract command and device_id.
    json_t *command_obj = json_object_get(event, "command");
    json_t *device_id_obj = json_object_get(event, "device_id");

    const char *command = json_string_value(command_obj);
    const char *device_id = device_id_obj ? json_string_value(device_id_obj) :
"device1";

    printf("Received command from cloud: %s\n", command);

    // Process command.
    if (command && strcmp(command, "get_status") == 0) {

```

```
    const char *status = get_device_status();

    // Send telemetry back to IoT Core.
    json_t *telemetry_data = json_object();
    json_object_set_new(telemetry_data, "device_id", json_string(device_id));
    json_object_set_new(telemetry_data, "status", json_string(status));
    json_object_set_new(telemetry_data, "timestamp", json_real(time(NULL)));

    char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

    // Publish telemetry to IoT Core.
    char telemetry_topic[256];
    snprintf(telemetry_topic, sizeof(telemetry_topic), "telemetry/%s",
device_id);

    aws_greengrass_publish_params params = {
        .topic = telemetry_topic,
        .payload = (uint8_t *)telemetry_payload,
        .payload_len = strlen(telemetry_payload)
    };

    aws_greengrass_iot_data_publish(iot_client, &params);

    printf("Telemetry sent to cloud: %s\n", telemetry_payload);

    free(telemetry_payload);
    json_decref(telemetry_data);
}

json_decref(event);
}

int main(int argc, char *argv[]) {
    // Initialize Greengrass SDK.
    iot_client = aws_greengrass_iot_data_client_new();

    // Subscribe to commands from IoT Core.
    aws_greengrass_subscribe_params subscribe_params = {
        .topic = "commands/device1",
        .callback = on_cloud_command,
        .user_data = NULL
    };

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);
```

```

printf("Device Controller Lambda started\n");
printf("Subscribed to commands/device1\n");
printf("Waiting for commands from IoT Core...\n");

// Keep the Lambda running.
while (1) {
    sleep(1);
}

return 0;
}

```

C++

```

#include <aws/greengrass/greengrasssdk.h>
#include <iostream>
#include <string>
#include <memory>
#include <jansson.h>
#include <ctime>
#include <unistd.h>

class DeviceController {
private:
    std::unique_ptr<aws_greengrass_iot_data_client,
    decltype(&aws_greengrass_iot_data_client_destroy)> iot_client;

    static void message_callback_wrapper(const char *topic, const uint8_t *payload,
                                         size_t payload_len, void *user_data) {
        auto* controller = static_cast<DeviceController*>(user_data);
        controller->on_cloud_command(topic, payload, payload_len);
    }

    std::string get_device_status() {
        // Simulate getting device status.
        return "online";
    }

public:
    DeviceController()
        : iot_client(aws_greengrass_iot_data_client_new(),
                    aws_greengrass_iot_data_client_destroy) {

```

```

    if (!iot_client) {
        throw std::runtime_error("Failed to create Greengrass IoT client");
    }
}

void on_cloud_command(const char *topic, const uint8_t *payload, size_t
payload_len) {
    // Parse incoming command from IoT Core.
    std::string payload_str(reinterpret_cast<const char*>(payload),
payload_len);

    json_error_t error;
    json_t *event = json_loads(payload_str.c_str(), 0, &error);

    if (!event) {
        std::cerr << "Error parsing JSON: " << error.text << std::endl;
        return;
    }

    // Extract command and device_id.
    json_t *command_obj = json_object_get(event, "command");
    json_t *device_id_obj = json_object_get(event, "device_id");

    const char *command = json_string_value(command_obj);
    const char *device_id = device_id_obj ? json_string_value(device_id_obj) :
"device1";

    std::cout << "Received command from cloud: " << command << std::endl;

    // Process command.
    if (command && std::string(command) == "get_status") {
        std::string status = get_device_status();

        // Send telemetry back to IoT Core.
        json_t *telemetry_data = json_object();
        json_object_set_new(telemetry_data, "device_id",
json_string(device_id));
        json_object_set_new(telemetry_data, "status",
json_string(status.c_str()));
        json_object_set_new(telemetry_data, "timestamp",
json_real(std::time(nullptr)));

        char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

```



```

        // Publish telemetry to IoT Core.
        std::string telemetry_topic = "telemetry/" + std::string(device_id);

        aws_greengrass_publish_params params = {
            .topic = telemetry_topic.c_str(),
            .payload = reinterpret_cast<uint8_t*>(telemetry_payload),
            .payload_len = strlen(telemetry_payload)
        };

        aws_greengrass_iot_data_publish(iot_client.get(), &params);

        std::cout << "Telemetry sent to cloud: " << telemetry_payload <<
std::endl;

        free(telemetry_payload);
        json_decref(telemetry_data);
    }

    json_decref(event);
}

void subscribe_to_topic(const std::string& topic) {
    aws_greengrass_subscribe_params subscribe_params = {
        .topic = topic.c_str(),
        .callback = message_callback_wrapper,
        .user_data = this
    };

    aws_greengrass_iot_data_subscribe(iot_client.get(), &subscribe_params);

    std::cout << "Device Controller Lambda started" << std::endl;
    std::cout << "Subscribed to " << topic << std::endl;
    std::cout << "Waiting for commands from IoT Core..." << std::endl;
}

void run() {
    // Keep the Lambda running.
    while (true) {
        sleep(1);
    }
}

};

int main(int argc, char *argv[]) {

```

```

    try {
        DeviceController controller;
        controller.subscribe_to_topic("commands/device1");
        controller.run();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```

일반 구성 요소(V2)

에서 동일한 기능을 구현하려면 다음을 사용하여 일반 구성 요소를 AWS IoT Greengrass V2 생성합니다.

1. 구성 요소 코드

Python

사전 조건: 이 구성 요소 코드를 사용하기 전에 Greengrass 디바이스에 Python AWS IoT Device SDK 용을 설치하고 확인합니다.

```

# Install the SDK
pip3 install awsiotsdk

# Verify installation
python3 -c "import awsiot.greengrasscoreipc.clientv2; print('SDK installed successfully')"

```

설치 중에 종속성 충돌이 발생하는 경우 특정 버전의를 설치해 보십시오 AWS IoT Device SDK.

확인 명령이 "SDK installed successfully"를 인쇄하면 구성 요소 코드를 사용할 준비가 된 것입니다.

```

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import QOS
import json
import time

ipc_client = GreengrassCoreIPCClientV2()

```

```
def on_command(event):
    """
    Receives commands from IoT Core,
    processes them, and sends telemetry back to cloud
    """
    try:
        # Receive command from IoT Core.
        data = json.loads(event.message.payload.decode('utf-8'))
        command = data.get('command')
        device_id = data.get('device_id', 'device1')

        print(f"Received command from cloud: {command}")

        # Process command.
        if command == 'get_status':
            status = get_device_status()

            # Send telemetry back to IoT Core.
            telemetry_data = {
                'device_id': device_id,
                'status': status,
                'timestamp': time.time()
            }

            ipc_client.publish_to_iot_core(
                topic_name=f'telemetry/{device_id}',
                qos=QOS.AT_LEAST_ONCE,
                payload=json.dumps(telemetry_data).encode('utf-8')
            )

            print(f"Telemetry sent to cloud: {telemetry_data}")

    except Exception as e:
        print(f"Error processing command: {e}")

def get_device_status():
    """Get current device status"""
    # Simulate getting device status.
    return 'online'

def main():
    print("Device Controller component starting...")

    # Subscribe to commands from IoT Core.
```

```

ipc_client.subscribe_to_iot_core(
    topic_name='commands/device1',
    qos=QOS.AT_LEAST_ONCE,
    on_stream_event=on_command
)

print("Subscribed to commands/device1 from IoT Core")
print("Waiting for commands from cloud...")

# Keep running.
while True:
    time.sleep(1)

if __name__ == '__main__':
    main()

```

Java

```

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class DeviceController {
    private static GreengrassCoreIPCClientV2 ipcClient;

    public static void main(String[] args) {
        System.out.println("Device Controller component starting...");

        try (GreengrassCoreIPCClientV2 client =
GreengrassCoreIPCClientV2.builder().build()) {
            ipcClient = client;

            SubscribeToIoTCoreRequest subscribeRequest = new
SubscribeToIoTCoreRequest()
                .withTopicName("commands/device1")
                .withQos(QOS.AT_LEAST_ONCE);

```

```
        ipcClient.subscribeToIoTCore(
            subscribeRequest,
            DeviceController::onCommand,
            Optional.empty(),
            Optional.empty()
        );

        System.out.println("Subscribed to commands/device1 from IoT Core");
        System.out.println("Waiting for commands from cloud...");

        while (true) {
            Thread.sleep(1000);
        }
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
        e.printStackTrace();
    }
}

public static void onCommand(IoTCoreMessage message) {
    try {
        String payload = new String(message.getMessage().getPayload(),
StandardCharsets.UTF_8);

        // Simple JSON parsing.
        String command = extractJsonValue(payload, "command");
        String deviceId = extractJsonValue(payload, "device_id");
        if (deviceId == null || deviceId.isEmpty()) {
            deviceId = "device1";
        }

        System.out.println("Received command from cloud: " + command);

        if ("get_status".equals(command)) {
            String status = getDeviceStatus();

            // Build JSON manually.
            String telemetryJson = String.format(
                "{\"device_id\":\"%s\",\"status\":\"%s\",\"timestamp\":\"%.3f\"}",
                deviceId, status, System.currentTimeMillis() / 1000.0
            );
            byte[] telemetryBytes =
telemetryJson.getBytes(StandardCharsets.UTF_8);
```

```

        PublishToIoTCoreRequest publishRequest = new
PublishToIoTCoreRequest()
            .withTopicName("telemetry/" + deviceId)
            .withQos(QOS.AT_LEAST_ONCE)
            .withPayload(telemetryBytes);

        ipcClient.publishToIoTCore(publishRequest);

        System.out.println("Telemetry sent to cloud: " + telemetryJson);
    }
} catch (Exception e) {
    System.err.println("Error processing command: " + e.getMessage());
}
}

private static String extractJsonValue(String json, String key) {
    Pattern pattern = Pattern.compile "\"" + Pattern.quote(key) + "\"\\s*:\\s*
\\\"([^\"]+)\\\"");
    Matcher matcher = pattern.matcher(json);
    return matcher.find() ? matcher.group(1) : null;
}

private static String getDeviceStatus() {
    return "online";
}
}

```

JavaScript

```

const greengrasscoreipc = require('aws-iot-device-sdk-v2').greengrasscoreipc;

class DeviceController {
    constructor() {
        this.ipcClient = null;
    }

    async start() {
        console.log('Device Controller component starting...');

        try {
            this.ipcClient = greengrasscoreipc.createClient();
            await this.ipcClient.connect();
        }
    }
}

```

```
const subscribeRequest = {
  topicName: 'commands/device1',
  qos: 1
};

const streamingOperation =
this.ipcClient.subscribeToIoTCore(subscribeRequest);

streamingOperation.on('message', (message) => {
  this.onCommand(message);
});

streamingOperation.on('streamError', (error) => {
  console.error('Stream error:', error);
});

streamingOperation.on('ended', () => {
  console.log('Subscription stream ended');
});

await streamingOperation.activate();

console.log('Subscribed to commands/device1 from IoT Core');
console.log('Waiting for commands from cloud...');

} catch (error) {
  console.error('Error starting component:', error);
  process.exit(1);
}
}

async onCommand(message) {
  try {
    const payload = message.message.payload.toString('utf-8');
    const data = JSON.parse(payload);

    const command = data.command;
    const deviceId = data.device_id || 'device1';

    console.log(`Received command from cloud: ${command}`);

    if (command === 'get_status') {
      const status = this.getDeviceStatus();
```

```

        const telemetryData = {
            device_id: deviceId,
            status: status,
            timestamp: Date.now() / 1000
        };

        const telemetryJson = JSON.stringify(telemetryData);

        const publishRequest = {
            topicName: `telemetry/${deviceId}`,
            qos: 1,
            payload: Buffer.from(telemetryJson, 'utf-8')
        };

        await this.ipcClient.publishToIoTCore(publishRequest);
        console.log(`Telemetry sent to cloud: ${telemetryJson}`);
    }

    } catch (error) {
        console.error('Error processing command:', error);
    }
}

getDeviceStatus() {
    return 'online';
}
}

// Start the component.
const controller = new DeviceController();
controller.start();

```

C

```

#include <gg/buffer.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/map.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdbool.h>
#include <stdio.h>

```



```
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <pthread.h>

#define COMMAND_TOPIC "commands/device1"
#define TELEMETRY_TOPIC "telemetry/device1"

typedef struct {
    char device_id[64];
    char command[64];
} CommandData;

static pthread_mutex_t command_mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t command_cond = PTHREAD_COND_INITIALIZER;
static CommandData pending_command;
static bool has_pending_command = false;

const char* get_device_status(void) {
    // Simulate getting device status.
    return "online";
}

static void *telemetry_publisher_thread(void *arg) {
    (void) arg;

    while (true) {
        pthread_mutex_lock(&command_mutex);
        while (!has_pending_command) {
            pthread_cond_wait(&command_cond, &command_mutex);
        }

        CommandData cmd = pending_command;
        has_pending_command = false;
        pthread_mutex_unlock(&command_mutex);

        // Process command.
        if (strcmp(cmd.command, "get_status") == 0) {
            const char *status = get_device_status();

            // Create telemetry JSON string.
            char telemetry_json[512];
            snprintf(telemetry_json, sizeof(telemetry_json),
                "{\"device_id\": \"%s\", \"status\": \"%s\", \"timestamp\": %ld}\",",
```

```

        cmd.device_id, status, time(NULL));

    GgBuffer telemetry_buf = {
        .data = (uint8_t *)telemetry_json,
        .len = strlen(telemetry_json)
    };

    // Publish telemetry to IoT Core.
    GgError ret = ggipc_publish_to_iot_core(GG_STR(TELEMETRY_TOPIC),
    telemetry_buf, 0);

    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to publish telemetry to IoT Core\n");
    } else {
        printf("Telemetry sent to cloud: device_id=%s, status=%s\n",
    cmd.device_id, status);
    }
}

return NULL;
}

static void on_cloud_command(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) topic;
    (void) handle;

    printf("Received command from IoT Core: %.*s\n", (int)payload.len,
    payload.data);

    // Parse JSON payload (comes as raw buffer from IoT Core).
    // For simplicity, we'll do basic string parsing.

    // Extract command and device_id from JSON string.
    char payload_str[512];
    snprintf(payload_str, sizeof(payload_str), "%.*s", (int)payload.len,
    payload.data);

    // Simple JSON parsing (looking for "command":"get_status").
    char *command_start = strstr(payload_str, "\"command\"");
    char *device_id_start = strstr(payload_str, "\"device_id\"");

```

```

if (command_start) {
    pthread_mutex_lock(&command_mutex);

    char *cmd_value = strstr(command_start, ":");
    if (cmd_value) {
        cmd_value = strchr(cmd_value, '"');
        if (cmd_value) {
            cmd_value++;
            char *cmd_end = strchr(cmd_value, '"');
            if (cmd_end) {
                size_t cmd_len = cmd_end - cmd_value;
                if (cmd_len < sizeof(pending_command.command)) {
                    strncpy(pending_command.command, cmd_value, cmd_len);
                    pending_command.command[cmd_len] = '\0';
                }
            }
        }
    }

    // Extract device_id or use default.
    if (device_id_start) {
        char *dev_value = strstr(device_id_start, ":");
        if (dev_value) {
            dev_value = strchr(dev_value, '"');
            if (dev_value) {
                dev_value++;
                char *dev_end = strchr(dev_value, '"');
                if (dev_end) {
                    size_t dev_len = dev_end - dev_value;
                    if (dev_len < sizeof(pending_command.device_id)) {
                        strncpy(pending_command.device_id, dev_value, dev_len);
                        pending_command.device_id[dev_len] = '\0';
                    }
                }
            }
        }
    } else {
        strncpy(pending_command.device_id, "device1");
    }

    printf("Received command from cloud: %s\n", pending_command.command);

    has_pending_command = true;

```

```
        pthread_cond_signal(&command_cond);
        pthread_mutex_unlock(&command_mutex);
    }
}

int main(void) {
    setvbuf(stdout, NULL, _IONBF, 0);
    printf("Device Controller component starting...\n");

    gg_sdk_init();

    GgError ret = ggipc_connect();
    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to connect to Greengrass nucleus\n");
        exit(1);
    }
    printf("Connected to Greengrass IPC\n");

    // Start telemetry publisher thread.
    pthread_t telemetry_thread;
    if (pthread_create(&telemetry_thread, NULL, telemetry_publisher_thread, NULL) !=
0) {
        fprintf(stderr, "Failed to create telemetry publisher thread\n");
        exit(1);
    }

    // Subscribe to commands from IoT Core.
    GgIpcSubscriptionHandle handle;
    ret = ggipc_subscribe_to_iot_core(
        GG_STR(COMMAND_TOPIC),
        0,
        &on_cloud_command,
        NULL,
        &handle
    );

    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to subscribe to IoT Core topic\n");
        exit(1);
    }

    printf("Subscribed to %s\n", COMMAND_TOPIC);
    printf("Waiting for commands from IoT Core...\n");
```

```

    while (true) {
        sleep(1);
    }

    return 0;
}

```

C++

```

#include <gg/ipc/client.hpp>
#include <gg/buffer.hpp>
#include <gg/object.hpp>
#include <gg/types.hpp>

#include <chrono>
#include <condition_variable>
#include <ctime>
#include <iostream>
#include <mutex>
#include <string>
#include <string_view>
#include <thread>

constexpr std::string_view COMMAND_TOPIC = "commands/device1";
constexpr std::string_view TELEMETRY_TOPIC = "telemetry/device1";

struct CommandData {
    std::string device_id;
    std::string command;
};

static std::mutex command_mutex;
static std::condition_variable command_cv;
static CommandData pending_command;
static bool has_pending_command = false;

std::string get_device_status() {
    // Simulate getting device status.
    return "online";
}

void telemetry_publisher_thread() {
    auto& client = gg::ipc::Client::get();

```

```

while (true) {
    std::unique_lock<std::mutex> lock(command_mutex);
    command_cv.wait(lock, [] { return has_pending_command; });

    CommandData cmd = pending_command;
    has_pending_command = false;
    lock.unlock();

    // Process command.
    if (cmd.command == "get_status") {
        std::string status = get_device_status();

        // Get current timestamp.
        auto now = std::time(nullptr);

        // Create telemetry JSON payload.
        std::string telemetry_payload = "{\"device_id\":\"" + cmd.device_id +
                                         "\",\"status\":\"" + status +
                                         "\",\"timestamp\":\"" + std::to_string(now)
+ "}\"";

        // Publish telemetry to IoT Core.
        gg::Buffer telemetry_buffer(telemetry_payload);
        auto error = client.publish_to_iot_core(TELEMETRY_TOPIC,
telemetry_buffer);

        if (error) {
            std::cerr << "Failed to publish telemetry to IoT Core: "
                      << error.message() << std::endl;
        } else {
            std::cout << "Telemetry sent to cloud: device_id=" << cmd.device_id
                      << ", status=" << status << std::endl;
        }
    }
}

class CloudCommandCallback : public gg::ipc::IoTCoreTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription& handle
    ) override {

```

```

    (void) topic;
    (void) handle;

    // Payload is a Buffer containing JSON string from IoT Core.
    if (payload.index() != GG_TYPE_BUF) {
        std::cerr << "Expected Buffer message\n";
        return;
    }

    // Extract buffer.
    auto buffer = gg::get<std::span<uint8_t>>(payload);
    std::string json_str(reinterpret_cast<const char*>(buffer.data()),
buffer.size());

    std::cout << "Received command from IoT Core: " << json_str << std::endl;

    // Simple JSON parsing for demo.
    std::string command;
    std::string device_id = "device1"; // Default

    // Extract command.
    size_t cmd_pos = json_str.find("\"command\":");
    if (cmd_pos != std::string::npos) {
        size_t start = json_str.find("\"", cmd_pos + 10) + 1;
        size_t end = json_str.find("\"", start);
        if (end != std::string::npos) {
            command = json_str.substr(start, end - start);
        }
    }

    // Extract device_id if present.
    size_t dev_pos = json_str.find("\"device_id\":");
    if (dev_pos != std::string::npos) {
        size_t start = json_str.find("\"", dev_pos + 12) + 1;
        size_t end = json_str.find("\"", start);
        if (end != std::string::npos) {
            device_id = json_str.substr(start, end - start);
        }
    }

    if (!command.empty()) {
        std::lock_guard<std::mutex> lock(command_mutex);
        pending_command = {device_id, command};
        has_pending_command = true;
    }

```

```
        command_cv.notify_one();

        std::cout << "Received command from cloud: " << command << std::endl;
    }
}

};

int main() {
    // Disable stdout buffering for real-time logging in systemd/Greengrass.
    std::cout.setf(std::ios::unitbuf);

    std::cout << "Device Controller component starting..." << std::endl;

    auto& client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to connect to Greengrass nucleus: "
                    << error.message() << std::endl;
        return 1;
    }

    std::cout << "Connected to Greengrass IPC" << std::endl;

    // Start telemetry publisher thread.
    std::thread telemetry_thread(telemetry_publisher_thread);
    telemetry_thread.detach();

    // Subscribe to commands from IoT Core.
    static CloudCommandCallback handler;
    error = client.subscribe_to_iot_core(COMMAND_TOPIC, handler);

    if (error) {
        std::cerr << "Failed to subscribe to IoT Core topic: "
                    << error.message() << std::endl;
        return 1;
    }

    std::cout << "Subscribed to " << COMMAND_TOPIC << std::endl;
    std::cout << "Waiting for commands from IoT Core..." << std::endl;

    // Keep running.
    while (true) {
        using namespace std::chrono_literals;
```



```

        std::this_thread::sleep_for(1s);
    }

    return 0;
}

```

2. 구성 요소 빌드 및 패키징

일부 언어는 배포 전에 빌드 또는 패키징이 필요합니다.

Python

Python은 컴파일이 필요하지 않습니다. 구성 요소는 .py 파일을 직접 사용할 수 있습니다.

Java

모든 종속성이 번들로 제공되는 실행 파일 JAR을 빌드하려면:

1. 프로젝트 디렉터리에 pom.xml 파일을 생성합니다.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Basic project information: organization, component name, and version --
>
    <groupId>com.example</groupId>
    <artifactId>device-controller</artifactId>
    <version>1.0.0</version>

    <properties>
        <!-- Java 11 LTS (Long Term Support) is recommended for Greengrass v2
components -->
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- AWS IoT Device SDK for Java - provides IPC client for Greengrass v2
cloud communication -->

```

```

    <dependency>
      <groupId>software.amazon.awssdk.iotdevicesdk</groupId>
      <artifactId>aws-iot-device-sdk</artifactId>
      <version>1.25.1</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- Maven Shade Plugin - creates a standalone JAR with all
dependencies included for Greengrass deployment -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.4</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <!-- Set the main class for the executable JAR
-->
                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                  <mainClass>DeviceController</mainClass>
                </transformer>
              </transformers>
              <filters>
                <!-- Exclude signature files to avoid security
exceptions -->
                <filter>
                  <artifact>*:*</artifact>
                  <excludes>
                    <exclude>META-INF/*.SF</exclude>
                    <exclude>META-INF/*.DSA</exclude>
                    <exclude>META-INF/*.RSA</exclude>
                  </excludes>
                </filter>
              </filters>
            </configuration>
          </execution>

```

```

        </executions>
      </plugin>
    </plugins>
  </build>
</project>

```

2. JAR을 빌드합니다.

```
mvn clean package
```

이렇게 하면 모든 종속성이 포함된 `target/device-controller-1.0.0.jar`이 생성됩니다.

3. 배포를 위해 JAR을 S3 버킷에 업로드합니다.

JavaScript

Node.js 구성 요소를 모든 종속성으로 패키징하려면:

1. `package.json` 파일 생성:

```

{
  "name": "device-controller",
  "version": "1.0.0",
  "description": "Device controller component for Greengrass v2",
  "main": "DeviceController.js",
  "dependencies": {
    "aws-iot-device-sdk-v2": "^1.21.0"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}

```

2. 개발 시스템에 종속성을 설치합니다.

```
npm install
```

이렇게 하면 AWS IoT Device SDK v2가 포함된 `node_modules` 폴더가 생성됩니다.

3. 배포용 패키지:

```
zip -r DeviceController.zip DeviceController.js node_modules/ package.json
```

4. zip 파일을 S3 버킷에 업로드하여 배포합니다.

Note

Greengrass 디바이스에는 Node.js 런타임(버전 14 이상)이 설치되어 있어야 합니다. 구성 요소 아티팩트 `npm install`에는 번들링된 `node_modules` 폴더의 모든 종속성이 포함되므로 Greengrass 코어 디바이스에서 실행할 필요가 없습니다.

C

사전 조건:

SDK와 구성 요소를 빌드하려면 다음과 같은 빌드 종속성이 필요합니다.

- GCC 또는 Clang
- CMake(최소 버전 3.22)
- 또는 Ninja 만들기

빌드 종속성 설치:

Ubuntu/Debian의 경우:

```
sudo apt install build-essential cmake
```

Amazon Linux에서:

```
sudo yum install gcc cmake make
```

구성 요소에 대한 CMakeLists.txt 파일을 생성합니다.

```
cmake_minimum_required(VERSION 3.10)
project(DeviceController C)

set(CMAKE_C_STANDARD 11)
```

```
# Add AWS Greengrass Component SDK
add_subdirectory(aws-greengrass-component-sdk)

# Build your component executable
add_executable(device_controller device_controller.c)
target_link_libraries(device_controller gg-sdk)
```

빌드 단계:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'device_controller' is in ./build/
# Upload this binary to S3 for deployment
```

C++

사전 조건:

SDK와 구성 요소를 빌드하려면 다음과 같은 빌드 종속성이 필요합니다.

- C++20을 지원하는 GCC 또는 Clang
- CMake(최소 버전 3.22)
- 또는 Ninja 만들기

빌드 종속성 설치:

Ubuntu/Debian의 경우:

```
sudo apt install build-essential cmake
```

Amazon Linux에서:

```
sudo yum install gcc-c++ cmake make
```

구성 요소에 대한 CMakeLists.txt 파일을 생성합니다.

```

cmake_minimum_required(VERSION 3.10)
project(DeviceController CXX)

set(CMAKE_CXX_STANDARD 20)

# Add SDK as subdirectory
add_subdirectory(aws-greengrass-component-sdk)

# Add C++ SDK subdirectory
add_subdirectory(aws-greengrass-component-sdk/cpp)

add_executable(device_controller device_controller.cpp)
target_link_libraries(device_controller gg-sdk++)

```

빌드 단계:

```

# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'device_controller' will be in ./build/
# Upload this binary to S3 for deployment

```

3. 구성 요소 레시피

구성 요소에서 사용하는 실제 주제로 "리소스" 배열을 업데이트합니다.

Python

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {

```

```

"accessControl": {
  "aws.greengrass.ipc.mqttproxy": {
    "com.example.DeviceController:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to IoT Core topics",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "commands/device1"
      ]
    },
    "com.example.DeviceController:mqttproxy:2": {
      "policyDescription": "Allows access to publish to IoT Core topics",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "telemetry/device1"
      ]
    }
  }
},
"ComponentDependencies": {
  "aws.greengrass.TokenExchangeService": {
    "VersionRequirement": ">=2.0.0",
    "DependencyType": "HARD"
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "python3 -u {artifacts:path}/device_controller.py"
    },
    "Artifacts": [
      {
        "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/device_controller.py"
      }
    ]
  }
]

```

```

    ]
  }
]
}

```

Java

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/device1"
            ]
          }
        }
      }
    }
  },
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {

```



```

    "VersionRequirement": ">=2.0.0",
    "DependencyType": "HARD"
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/DeviceController.jar"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/DeviceController.jar"
        }
      ]
    }
  ]
}

```

JavaScript

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to command topics from IoT Core",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ]
          }
        }
      }
    }
  }
}

```

```

        "resources": [
            "commands/device1"
        ],
    },
    "com.example.DeviceController:mqttproxy:2": {
        "policyDescription": "Allows access to publish telemetry to IoT Core",
        "operations": [
            "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
            "telemetry/*"
        ]
    }
}
},
"Manifests": [
    {
        "Platform": {
            "os": "linux",
            "runtime": "*"
        },
        "Lifecycle": {
            "run": "cd {artifacts:decompressedPath}/DeviceController && node
DeviceController.js"
        },
        "Artifacts": [
            {
                "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
DeviceController.zip",
                "Unarchive": "ZIP"
            }
        ]
    }
]
}

```

C/C++

```

{
    "RecipeFormatVersion": "2020-01-25",
    "ComponentName": "com.example.DeviceController",

```

```

"ComponentVersion": "1.0.0",
"ComponentType": "aws.greengrass.generic",
"ComponentDescription": "Receives commands from IoT Core and sends telemetry back
to cloud",
"ComponentPublisher": "[Your Company]",
"ComponentConfiguration": {
  "DefaultConfiguration": {
    "accessControl": {
      "aws.greengrass.ipc.mqttproxy": {
        "com.example.DeviceController:mqttproxy:1": {
          "policyDescription": "Allows access to subscribe to IoT Core topics",
          "operations": ["aws.greengrass#SubscribeToIoTCore"],
          "resources": ["commands/device1"]
        },
        "com.example.DeviceController:mqttproxy:2": {
          "policyDescription": "Allows access to publish to IoT Core topics",
          "operations": ["aws.greengrass#PublishToIoTCore"],
          "resources": ["telemetry/device1"]
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": ""
      },
      "Lifecycle": {
        "run": "{artifacts:path}/device_controller"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller"
        }
      ]
    }
  ]
}

```

Greengrass V1 코어 디바이스를 Greengrass V2로 업그레이드

애플리케이션 및 구성 요소가 AWS IoT Greengrass V2 코어 디바이스에서 작동하는지 확인한 후 프로덕션 디바이스와 같이 현재 v1.x를 실행하는 디바이스에 AWS IoT Greengrass 코어 소프트웨어 v2.x를 설치할 수 있습니다. 그런 다음 Greengrass V2 구성 요소를 배포하여 디바이스에서 Greengrass 애플리케이션을 실행합니다.

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

Tip

디바이스 플릿에 대한 업그레이드 프로세스를 자동화하는 스크립트를 생성하는 것이 좋습니다. [AWS Systems Manager](#)를 사용하여 플릿을 관리하려면 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 코어 소프트웨어 v2.x 설치

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

- [더 적은 단계로 업그레이드](#)

더 적은 단계로 업그레이드하려면 v2.x 소프트웨어를 설치하기 전에 v1.x 소프트웨어를 제거하면 됩니다.

- [가동 중지 시간을 최소화하면서 업그레이드](#)

가동 중지 시간을 최소화하면서 업그레이드하려면 AWS IoT Greengrass 코어 소프트웨어의 두 버전을 동시에 설치할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어 v2.x를 설치하고 V2 애플리케이션이 올바르게 작동하는지 확인한 후 AWS IoT Greengrass 코어 소프트웨어 v1.x를 제거합니다. 이 옵션을 선택하기 전에 AWS IoT Greengrass 코어 소프트웨어의 두 버전을 동시에 실행하는 데 필요한 추가 RAM을 고려하세요.

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

순차적으로 업그레이드하려면 디바이스에 v2.x를 설치하기 전에 AWS IoT Greengrass 코어 소프트웨어 v1.x를 제거합니다.

AWS IoT Greengrass 코어 소프트웨어 v1.x를 제거하려면

1. AWS IoT Greengrass 코어 소프트웨어 v1.x가 서비스로 실행 중인 경우 서비스를 중지, 비활성화 및 제거해야 합니다.

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

```
sudo systemctl stop greengrass
```

- b. 서비스가 중지될 때까지 기다립니다. list 명령을 사용하여 디바이스의 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 Greengrass 코어 소프트웨어 v1.x가 서비스로 실행되지 않는 경우 다음 명령을 사용하여 데몬을 중지합니다. **greengrass-root**를 Greengrass 루트 폴더의 이름으로 바꿉니다. 기본 위치는 /greengrass입니다.

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

3. (선택 사항) Greengrass 루트 폴더와 해당되는 경우 [사용자 지정 쓰기 폴더](#)를 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
```

Greengrass nucleus의 경우:의 [설치 지침을 AWS IoT Greengrass V2](#) 사용하여 Greengrass nucleus를 디바이스에 설치할 수 있습니다.

Greengrass nucleus lite의 경우: [Greengrass nucleus lite의 설치 지침](#)을 사용하여 Greengrass nucleus lite를 설치할 수 있습니다.

Tip

V1에서 V2로 마이그레이션할 때 코어 디바이스의 자격 증명을 재사용하려면 지침에 따라 [수동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치합니다](#). 먼저 디바이스에서 V1 코어 소프트웨어를 제거한 다음 V1 코어 디바이스의 AWS IoT 사물과 인증서를 재사용하고 인증서 AWS IoT 정책을 업데이트하여 v2.x 소프트웨어에 필요한 권한을 부여합니다.

이미 v1.x를 실행 중인 디바이스에 AWS IoT Greengrass 코어 소프트웨어 v2.x 설치

AWS IoT Greengrass 이미 코어 소프트웨어 v1.x를 실행 중인 디바이스에 AWS IoT Greengrass 코어 v2.x 소프트웨어를 설치하는 경우 다음 사항에 유의하세요.

- V2 코어 디바이스의 AWS IoT 사물 이름은 고유해야 합니다. V1 코어 디바이스와 동일한 사물 이름을 사용하지 마세요.
- AWS IoT Greengrass 코어 소프트웨어 v2.x에 사용하는 포트는 v1.x에 사용하는 포트와 달라야 합니다.
 - 8088 이외의 포트를 사용하도록 V1 스트림 관리자를 구성합니다. 자세한 내용은 [스트림 관리자 구성](#)을 참조하세요.
 - 8883 이외의 포트를 사용하도록 V1 MQTT 브로커를 구성합니다. 자세한 내용은 [Configure the MQTT port for local messaging](#)을 참조하세요.
- AWS IoT Greengrass V2 는 Greengrass 시스템 서비스의 이름을 변경하는 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 서비스의 이름을 바꿉니다.
- 시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어 v2.x를 설치한 다음 이외의 이름으로 [소프트웨어를 시스템 서비스로 수동으로 구성합니다](#)greengrass.

v1.x용 Greengrass 서비스의 이름을 바꾸려면

1. AWS IoT Greengrass 코어 소프트웨어 v1.x 서비스를 중지합니다.

```
sudo systemctl stop greengrass
```

2. 서비스가 중지될 때까지 기다립니다. 서비스를 중지하는 데 최대 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. 서비스의 이름을 바꿉니다.

```
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 [AWS IoT Greengrass V2설치 지침](#)을 사용하여 디바이스에 소프트웨어를 설치할 수 있습니다.

Tip

V1에서 V2로 마이그레이션할 때 코어 디바이스의 자격 증명을 재사용하려면 지침에 따라 [수동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치합니다](#). 먼저 디바이스에서 V1

코어 소프트웨어를 제거한 다음 V1 코어 디바이스의 AWS IoT 사물과 인증서를 재사용하고 인증서 AWS IoT 정책을 업데이트하여 v2.x 소프트웨어에 필요한 권한을 부여합니다.

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

디바이스에 AWS IoT Greengrass 코어 소프트웨어 v2.x를 설치한 후 선택한 런타임에 따라 구성 요소를 배포합니다.

Greengrass nucleus의 경우:

다음 리소스가 포함된 배포를 생성합니다. 유사한 디바이스 플릿에 구성 요소를 배포하려면 해당 디바이스가 포함된 사물 그룹에 대한 배포를 생성합니다.

- V1 Lambda 함수에서 생성한 Lambda 함수 구성 요소입니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.
- V1 구독을 사용하는 경우 [레거시 구독 라우터 구성 요소](#)입니다.
- 스트림 관리자를 사용하는 경우 [스트림 관리자 구성 요소](#)입니다. 자세한 내용은 [Greengrass 코어 디바이스에서 데이터 스트림 관리](#) 단원을 참조하십시오.
- 로컬 보안 암호를 사용하는 경우 [보안 암호 관리자 구성 요소](#)입니다.
- V1 커넥터를 사용하는 경우 [AWS제공 커넥터 구성 요소](#)입니다.
- Docker 컨테이너를 사용하는 경우 [Docker 애플리케이션 관리자 구성 요소](#)입니다. 자세한 내용은 [Docker 컨테이너 실행](#) 단원을 참조하십시오.
- 연결된 디바이스를 사용하는 경우 [클라이언트 디바이스 지원을 위한 구성 요소](#)입니다. 또한 클라이언트 디바이스 지원을 활성화하고 클라이언트 디바이스를 코어 디바이스와 연결해야 합니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.
- 디바이스 새도를 사용하는 경우 [새도 관리자 구성 요소](#)입니다. 자세한 내용은 [디바이스 새도와 연동](#) 단원을 참조하십시오.
- Greengrass 코어 디바이스의 로그를 Amazon CloudWatch Logs로 업로드하는 경우 [로그 관리자 구성 요소](#)입니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.
- 와 통합하는 경우 지침에 AWS IoT SiteWise 따라 V2 코어 디바이스를 AWS IoT SiteWise 게이트웨이로 설정합니다.는 AWS IoT SiteWise 구성 요소를 배포하는 설치 스크립트를 AWS IoT SiteWise 제공합니다. <https://docs.aws.amazon.com/iot-sitewise/latest/userguide/configure-gateway-ggv2.html>
- 사용자 지정 기능을 구현하기 위해 개발한 사용자 지정 구성 요소입니다.

배포 생성 및 수정에 대한 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.

Greengrass nucleus lite의 경우:

[마이그레이션 가이드의 2단계](#)에서 생성한 일반 구성 요소를 Greengrass nucleus lite 디바이스에 배포합니다.

1. 생성한 구성 요소 레시피를 사용하여에서 구성 요소 생성
2. 일반 구성 요소가 포함된 Greengrass nucleus lite 디바이스를 대상으로 배포 생성
3. 구성 요소가 올바르게 실행되고 있는지 확인

자습서: AWS IoT Greengrass V2 시작하기

이 시작하기 자습서를 완료하여 AWS IoT Greengrass V2의 기본 기능을 학습할 수 있습니다. 이 자습서에서는 다음을 수행합니다.

1. Raspberry Pi와 같은 Linux 디바이스 또는 Windows 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하고 구성합니다. 이 디바이스는 Greengrass 코어 디바이스입니다.
2. Greengrass 코어 디바이스에서 Hello World 구성 요소를 개발합니다. 구성 요소는 Greengrass 코어 디바이스에서 실행되는 소프트웨어 모듈입니다.
3. AWS 클라우드의 AWS IoT Greengrass V2로 구성 요소를 업로드합니다.
4. AWS 클라우드에서 Greengrass 코어 디바이스로 해당 구성 요소를 배포합니다.

Note

이 자습서에서는 개발 환경을 설정하고 AWS IoT Greengrass의 기능을 탐색하는 방법을 설명합니다. 프로덕션 디바이스를 설정 및 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [AWS IoT Greengrass 코어 디바이스 설정](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)

이 자습서의 예상 소요 시간은 20~30분입니다.

주제

- [사전 조건](#)
- [1단계: AWS 계정 설정](#)
- [2단계: 환경 설정](#)
- [3단계: AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [4단계: 디바이스에서 구성 요소 개발 및 테스트](#)
- [5단계: AWS IoT Greengrass 서비스에서 구성 요소 생성](#)
- [6단계: 구성 요소 배포](#)
- [다음 단계](#)

사전 조건

이 시작하기 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 없으면 [1단계: AWS 계정 설정](#) 단원을 참조하세요.
- AWS IoT Greengrass V2를 지원하는 [AWS 리전](#) 사용. 지원되는 리전 목록은 AWS 일반 참조에서 [AWS IoT Greengrass V2 엔드포인트 및 할당량](#)을 참조하세요.
- 관리자 권한이 있는 AWS Identity and Access Management(IAM) 사용자.
- [Raspberry Pi OS](#)(기존의 Raspbian)를 사용하는 Raspberry Pi 또는 Windows 10 디바이스와 같이 Greengrass 코어 디바이스로 설정할 디바이스. 이 디바이스에 대한 관리자 권한을 보유하거나 sudo를 통한 경로 등으로 관리자 권한을 획득할 수 있어야 합니다. 이 디바이스는 인터넷에 연결되어 있어야 합니다.

AWS IoT Greengrass 코어 소프트웨어 설치 및 실행 요구 사항을 충족하는 다른 디바이스를 사용하도록 선택할 수도 있습니다.

개발 컴퓨터가 이러한 요구 사항을 충족하는 경우 이 자습서에서 Greengrass 코어 디바이스로 설정할 수 있습니다.

- 디바이스의 모든 사용자에게 대해 [Python](#) 3.5 이상 버전이 설치되고 PATH 환경 변수에 추가되어야 합니다. Windows에서는 모든 사용자에게 대해 Windows용 Python 런처가 설치되어 있어야 합니다.

Important

Windows에서 Python은 기본적으로 모든 사용자를 대상으로 설치되지 않습니다. Python을 설치할 때는 AWS IoT Greengrass 코어 소프트웨어에서 Python 스크립트가 실행되도록 설치 구성을 사용자 지정해야 합니다. 예를 들어 그래픽 Python 설치 관리자를 사용하는 경우 다음을 수행합니다.

1. 모든 사용자에게 대해 런처 설치(권장)를 선택합니다.
2. 를 선택합니다..Customize installation
3. 를 선택합니다..Next
4. 을 선택합니다..Install for all users
5. 을 선택합니다..Add Python to environment variables
6. 설치를 선택합니다.

자세한 내용은 Python 3 설명서의 [Using Python on Windows](#)를 참조하세요.

- 자격 증명을 사용하여 개발 컴퓨터와 디바이스에 AWS Command Line Interface(AWS CLI)가 설치되어 있어야 합니다. 동일한 AWS 리전을 사용하여 개발 컴퓨터와 디바이스에서 AWS CLI를 구성해야 합니다. AWS CLI와 함께 AWS IoT Greengrass V2를 사용하려면 다음 버전 또는 그 이상이 있어야 합니다.
- 최소 AWS CLI V1 버전: v1.18.197
- 최소 AWS CLI V2 버전: v2.1.11

Tip

다음 명령을 실행하여 보유하고 있는 AWS CLI의 버전을 확인할 수 있습니다.

```
aws --version
```

자세한 내용은 AWS Command Line Interface 사용 설명서에서 [AWS CLI 설치, 업데이트, 제거 및 AWS CLI 구성](#) 섹션을 참조하세요.

Note

32비트 운영 체제를 사용하는 Raspberry Pi와 같은 32비트 ARM 디바이스를 사용하는 경우 AWS CLI V1을 설치합니다. AWS CLI 32비트 ARM 디바이스에서는 V2를 사용할 수 없습니다. 자세한 내용은 [AWS CLI 버전 1 설치, 업데이트 및 제거](#) 단원을 참조하세요.

1단계: AWS 계정 설정

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. <https://portal.aws.amazon.com/billing/signup>을 엽니다.
2. 온라인 지시 사항을 따릅니다.

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

에 가입하면 AWS 계정 루트 사용자의 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 수행하는 것입니다.

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

관리자 액세스 권한이 있는 사용자 생성

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

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자 [AWS Management Console](#)로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 [루트 사용자 로 로그인](#)을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

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

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [AWS IAM Identity Center 설정](#)을 참조하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 자격 증명 소스 IAM Identity Center 디렉터리로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 [기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요](#).

관리 액세스 권한이 있는 사용자로 로그인

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

IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 [AWS 액세스 포털에 로그인](#)을 참조하세요.

추가 사용자에게 액세스 권한 할당

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

지침은 AWS IAM Identity Center 사용 설명서의 [Create a permission set](#)를 참조하세요.

2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [Add groups](#)를 참조하세요.

2단계: 환경 설정

Note

이러한 단계는 nucleus lite에는 적용되지 않습니다.

이 섹션의 단계에 따라 AWS IoT Greengrass 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

Linux 디바이스 설정(Raspberry Pi)

이 단계에서는 Raspberry Pi OS와 함께 Raspberry Pi를 사용한다고 가정합니다. 다른 디바이스 또는 운영 체제를 사용하는 경우 해당 디바이스의 관련 설명서를 참조하세요.

용 Raspberry Pi를 설정하려면 AWS IoT Greengrass V2

1. Raspberry Pi에 원격으로 연결하려면 Pi에서 SSH를 활성화해야 합니다. 자세한 내용은 Raspberry Pi 설명서에서 [SSH\(Secure shell\)](#)를 참조하세요.
2. SSH를 사용하여 연결할 Raspberry Pi의 IP 주소를 찾습니다. 이를 위해 Raspberry Pi에서 다음 명령을 실행할 수 있습니다.

```
hostname -I
```

3. SSH를 사용하여 Raspberry Pi에 연결합니다.

개발 컴퓨터에서 다음 명령을 실행합니다. *username*을 로그인할 사용자의 이름으로 바꾸고 *pi-ip-address*를 이전 단계에서 찾은 IP 주소로 바꿉니다.

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

Important

개발 컴퓨터에서 이전 버전의 Windows를 사용하는 경우 ssh 명령이 없거나 ssh가 있어도 Raspberry Pi에 연결하지 못할 수 있습니다. Raspberry Pi에 연결하려면 무료 오픈 소스 SSH 클라이언트인 [PuTTY](#)를 설치하고 구성하면 됩니다. [PuTTY 설명서](#)를 참조하여 Raspberry Pi에 연결합니다.

4. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. Raspberry Pi에서 다음 명령을 사용하여 Java 11을 설치합니다.

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

설치가 완료되면 다음 명령을 실행하여 Raspberry Pi에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 출력은 다음 예제와 비슷할 것입니다.

```
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: Raspberry Pi에서 커널 파라미터 설정

디바이스가 Raspberry Pi인 경우 다음 단계를 완료하여 Linux 커널 파라미터를 보고 업데이트할 수 있습니다.

1. `/boot/cmdline.txt` 파일을 엽니다. 이 파일은 Raspberry Pi가 부팅될 때 적용할 Linux 커널 파라미터를 지정합니다.

예를 들어 Linux 기반 시스템에서 GNU nano를 사용하여 파일을 열도록 다음 명령을 실행할 수 있습니다.

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

2. `/boot/cmdline.txt` 파일에 다음 커널 파라미터가 포함되어 있는지 확인합니다. `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.txt` 파일에 이러한 파라미터가 포함되어 있지 않거나 값이 다른 파라미터가 포함되어 있는 경우 이러한 파라미터와 값을 포함하도록 파일을 업데이트합니다.

3. `/boot/cmdline.txt` 파일을 업데이트한 경우 Raspberry Pi를 재부팅하여 변경 사항을 적용합니다.

```
sudo reboot
```

Linux 디바이스 설정(기타)

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

Windows 디바이스 설정

용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
 - a. Windows 키를 눌러 시작 메뉴를 엽니다.
 - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
 - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
 - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
 - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
 - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.

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

- g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.

4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- wmic 명령이 더는 사용되지 않는 Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

Note

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

3단계: AWS IoT Greengrass 코어 소프트웨어 설치

Tip

[AWS IoT Greengrass를 빠르게 설정하고 실험하려면 IoT Greengrass AI 에이전트 컨텍스트 팩을](#) 사용해 보는 것이 좋습니다. AWS IoT 에이전트 컨텍스트 팩을 사용하면 AI 에이전트가 Greengrass Nucleus 및 Nucleus Lite를 설정하고, 구성 요소를 배포하고, 일반적인 문제를 해결할 수 있습니다.

이 섹션의 단계에 따라 Raspberry Pi를 로컬 개발에 사용할 수 있는 AWS IoT Greengrass 코어 디바이스로 설정합니다. 이 단원에서는 다음을 수행하는 설치 관리자를 다운로드하고 실행하여 디바이스에 대한 AWS IoT Greengrass 코어 소프트웨어를 구성합니다.

- Greengrass nucleus 구성 요소를 설치합니다. nucleus는 필수 구성 요소이며 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 실행하기 위한 최소 요구 사항입니다. 자세한 내용은 [Greengrass nucleus 구성 요소](#)를 참조하세요.
- 디바이스를 AWS IoT 사물로 등록하고 디바이스를 연결할 수 있는 디지털 인증서를 다운로드합니다. AWS. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.
- AWS IoT 사물 그룹, 즉 사물 그룹 또는 플릿에 디바이스의 AWS IoT 사물을 추가합니다. 사물 그룹을 사용하여 Greengrass 코어 디바이스의 플릿을 관리할 수 있습니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹에 배포할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서에서 [AWS IoT로 디바이스 관리](#)를 참조하세요.
- Greengrass 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 하는 IAM 역할을 생성합니다. 기본적으로 이 역할을 사용하면 디바이스가 Amazon CloudWatch Logs와 상호 작용 AWS IoT 하고 로그를 전송할 수 있습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

- 코어 디바이스에서 개발하는 사용자 지정 구성 요소를 테스트하는 데 사용할 수 있는 AWS IoT Greengrass 명령줄 인터페이스(greengrass-cli)를 설치합니다. 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어 설치(콘솔)

1. [AWS IoT Greengrass 콘솔](#)에 로그인합니다.
2. Greengrass 시작하기에서 코어 디바이스 설정을 선택합니다.
3. 1단계: Greengrass 코어 디바이스 등록에서 코어 디바이스 이름에 Greengrass 코어 디바이스의 AWS IoT 사물 이름을 입력합니다. 사물이 존재하지 않는 경우 설치 관리자가 이를 생성합니다.
4. 2단계: 사물 그룹에 추가하여 지속적 배포 적용에서 사물 그룹에 대해 코어 디바이스를 추가할 AWS IoT 사물 그룹을 선택합니다.
 - 새 그룹 이름 입력을 선택하는 다음 사물 그룹 이름에 생성할 새 그룹의 이름을 입력합니다. 설치 프로그램에서 새 그룹을 생성합니다.
 - 기존 그룹 선택을 선택하는 경우 사물 그룹 이름에서 사용하려는 기존 그룹을 선택합니다.
 - 그룹 없음을 선택하면 설치 프로그램에서 코어 디바이스를 사물 그룹에 추가하지 않습니다.
5. 3단계: Greengrass 코어 소프트웨어 설치에서 다음 단계를 완료합니다.

Nucleus classic

1. 코어 디바이스의 소프트웨어 런타임으로 Nucleus classic을 선택합니다.
2. 코어 디바이스의 운영 체제인 Linux 또는 Windows를 선택합니다.
3. 설치 관리자가 코어 디바이스에 대한 AWS IoT 및 IAM 리소스를 프로비저닝할 수 있도록 디바이스에 AWS 자격 증명을 제공합니다. 보안을 강화하려면 프로비저닝에 필요한 최소 권한만 허용하는 IAM 역할에 대한 임시 자격 증명을 받는 것이 좋습니다. 자세한 내용은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

Note

설치 관리자에서는 자격 증명을 저장하거나 저장하지 않습니다.

디바이스에서 다음 중 하나를 수행하여 자격 증명을 검색하고 AWS IoT Greengrass 코어 소프트웨어 설치 관리자에서 사용할 수 있도록 합니다.

- (권장) AWS IAM Identity Center의 임시 자격 증명 사용

- a. IAM Identity Center에서 액세스 키 ID, 비밀 액세스 키, 세션 토큰을 제공합니다. 자세한 내용은 IAM Identity Center 사용 설명서의 [임시 자격 증명 가져오기 및 새로 고침](#)에서 수동 자격 증명 새로 고침을 참조하세요.
- b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- IAM 역할에서 임시 보안 인증을 사용합니다.
 - a. 수입하는 IAM 역할에서 액세스 키 ID, 비밀 액세스 키, 세션 토큰을 제공합니다. 이러한 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서에서 [임시 보안 인증 요청](#)을 참조하세요.
 - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- IAM 사용자의 장기 자격 증명 사용:
 - a. IAM 사용자의 액세스 키 ID 및 비밀 액세스 키를 제공합니다. 나중에 삭제하는 프로비저닝을 위한 IAM 사용자를 생성할 수 있습니다. 사용자에게 제공할 IAM 정책은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요. 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서에서 [IAM 사용자의 액세스 키 관리](#)를 참조하세요.
 - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/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 사용자를 삭제합니다.
 - d. (선택 사항) 기존 IAM 사용자의 액세스 키 ID 및 비밀 액세스 키를 사용한 경우 해당 키가 더 이상 유효하지 않도록 사용자의 키를 업데이트합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [액세스 키 업데이트](#)를 참조하세요.
4. 설치 프로그램 실행에서 다음 단계를 완료합니다.
- a. 설치 관리자 다운로드에서 복사를 선택하고 코어 디바이스에서 복사된 명령을 실행합니다. 이 명령은 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드하고 디바이스에 압축을 풉니다.
 - b. 설치 관리자 실행에서 복사를 선택하고 코어 디바이스에서 복사된 명령을 실행합니다. 이 명령은 이전에 지정한 AWS IoT 사물 그룹 및 사물 그룹 이름을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 실행하고 코어 디바이스에 대한 AWS 리소스를 설정합니다.

이 명령은 다음 작업도 수행합니다.

- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

 Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

- 코어 디바이스에서 사용자 지정 Greengrass 구성 요소를 개발할 수 있는 명령줄 도구인 [AWS IoT Greengrass CLI 구성 요소](#)를 배포합니다.
- ggc_user 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 ggc_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.

이 명령을 실행하면 설치 관리자가 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 디바이스가 있지만 이 디바이스에 [systemd](#)가 없는 경우 설치 관리자는 소프트웨어를 시스템 서비스로 설정하지 않으며, nucleus를 시스템 서비스로 설정하기 위한 성공 메시지가 표시되지 않습니다.

Nucleus lite

1. 코어 디바이스의 소프트웨어 런타임으로 Nucleus lite를 선택합니다.
2. 디바이스를 Greengrass 코어 디바이스로 프로비저닝하기 위한 디바이스 설정 방법을 선택합니다.

옵션 1: 패키지 다운로드로 디바이스 설정(약 1MB)

1. AWS IoT 사물과 Greengrass용 역할을 생성합니다.
2. 디바이스가 AWS IoT에 연결하는 데 필요한 AWS IoT 리소스가 포함된 ZIP 파일을 다운로드합니다.
 - AWS IoT의 인증 기관을 사용해 생성된 인증서와 프라이빗 키.
 - 디바이스용 Greengrass 설치를 시작하는 스키마 파일.
3. Raspberry Pi에 최신 Greengrass Nucleus lite 런타임을 설치하는 패키지를 다운로드합니다.
4. 디바이스를 AWS IoT Greengrass Core 디바이스가 되도록 프로비저닝하고 AWS IoT에 연결합니다.
 - a. USB 썸 드라이브, SCP/FTP, SD 카드를 사용하여 Greengrass 패키지와 연결 키트를 디바이스로 전송합니다.

- b. 디바이스의 /GreengrassInstaller 디렉터리에서 greengrass-package.zip 파일을 압축 해제합니다.
 - c. 디바이스의 /directory에서 연결 키트 압축 파일을 압축 해제합니다.
 - d. 디바이스에서 제공된 명령을 실행하여 AWS IoT Greengrass를 설치합니다.
5. 그런 다음 코어 디바이스 보기를 선택합니다.

옵션 2: 사전 구성된 전체 디스크 샘플 이미지 다운로드로 디바이스 설정(약 100MB)

1. AWS IoT 사물과 Greengrass용 역할을 생성합니다.
2. 디바이스가 AWS IoT에 연결하는 데 필요한 AWS IoT 리소스가 포함된 ZIP 파일을 다운로드합니다.
 - AWS IoT의 인증 기관을 사용해 생성된 인증서와 프라이빗 키.
 - 디바이스용 Greengrass 설치를 시작하는 스키마 파일.
3. Greengrass와 운영 체제가 포함된 사전 구성된 전체 디스크 샘플 이미지를 다운로드합니다.
 - a. 연결 키트를 전송하고 이미지를 디바이스에 플래시하려면, 이미지와 함께 다운로드된 readme 파일의 지침을 따릅니다.
 - b. Greengrass 설치를 시작하려면 디바이스를 켜고 플래시된 이미지로 부팅합니다.
4. 그런 다음 코어 디바이스 보기를 선택합니다.

옵션 3: 자체 사용자 지정 빌드로 디바이스 설정

1. AWS IoT 사물과 Greengrass용 역할을 생성합니다.
2. 디바이스가 AWS IoT에 연결하는 데 필요한 AWS IoT 리소스가 포함된 ZIP 파일을 다운로드합니다.
 - AWS IoT의 인증 기관을 사용해 생성된 인증서와 프라이빗 키.
 - 디바이스용 Greengrass 설치를 시작하는 스키마 파일.
3. Yocto를 사용하여 소스 코드로 자체 이미지를 사용자 지정 및 빌드한 후, 연결 키트를 사용해 nucleus lite를 설치합니다. 자세한 내용은 GitHub 지침을 참조합니다.
 - 그런 다음 코어 디바이스 보기를 선택합니다.

AWS IoT Greengrass 코어 소프트웨어 설치(CLI)

Note

이러한 단계는 nucleus lite에는 적용되지 않습니다.

AWS IoT Greengrass 코어 소프트웨어를 설치 및 구성

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 코어 소프트웨어를 `greengrass-nucleus-latest.zip`라는 파일에 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 계약](#)에 동의하는 것입니다.

3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 해제합니다. *GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

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 코어 소프트웨어 설치 관리자를 시작합니다. 이 명령은 다음 작업을 수행합니다.
 - 코어 디바이스가 작동하는 데 필요한 AWS 리소스를 생성합니다.
 - AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

- 코어 디바이스에서 사용자 지정 Greengrass 구성 요소를 개발할 수 있는 명령줄 도구인 [AWS IoT Greengrass CLI 구성 요소](#)를 배포합니다.
- `ggc_user` 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 `ggc_group` 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.

다음과 같이 명령의 인수 값을 바꿉니다.

- `/greengrass/v2` 또는 `C:\greengrass\v2`: AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.
- `GreengrassInstaller`. AWS IoT Greengrass 코어 소프트웨어 설치 관리자의 압축을 푼 폴더의 경로입니다.
- `region`. 리소스를 찾거나 생성할 AWS 리전입니다.
- `MyGreengrassCore`. Greengrass 코어 디바이스에 대한 AWS IoT 사물의 이름입니다. 사물이 존재하지 않는 경우 설치 관리자가 이를 생성합니다. 설치 관리자는 인증서를 다운로드하여 AWS IoT 사물로 인증합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 섹션을 참조하세요.

Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

- `MyGreengrassCoreGroup`. Greengrass 코어 디바이스에 대한 AWS IoT 사물 그룹의 이름입니다. 사물 그룹이 존재하지 않는 경우 설치 관리자가 이를 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 활성 배포가 있는 경우 코어 디바이스는 배포에서 지정한 소프트웨어를 다운로드하고 실행합니다.

Note

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

- `GreengrassV2IoTThingPolicy`. Greengrass 코어 디바이스가 AWS IoT 및 AWS IoT Greengrass와 통신할 수 있도록 허용하는 AWS IoT 정책의 이름입니다. AWS IoT 정책이 없는 경우 설치 관리자에서 이 이름으로 허용적인 AWS IoT 정책을 생성합니다. 사용 사례에 대해 이 정책의 권한을 제한할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 섹션을 참조하세요.

- g. ***GreengrassV2TokenExchangeRole***. Greengrass 코어 디바이스가 임시 AWS 자격 증명을 가져올 수 있도록 허용하는 IAM 역할의 이름입니다. 역할이 없는 경우 설치 관리자는 이를 생성하고 ***GreengrassV2TokenExchangeRoleAccess***라는 정책을 생성 및 연결합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.
- h. ***GreengrassCoreTokenExchangeRoleAlias***. Greengrass 코어 디바이스가 나중에 임시 자격 증명을 가져올 수 있도록 허용하는 IAM 역할의 별칭입니다. 역할 별칭이 없는 경우 설치 관리자에서 이를 생성하고 지정한 IAM 역할로 가리킵니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 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 디바이스에서 AWS IoT Greengrass를 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용되는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 nucleus 구성 요소의 jvmOptions 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 섹션을 참조하세요.

이 명령을 실행하면 설치 관리자가 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 디바이스가 있지만 이 디바이스에 [systemd](#)가 없는 경우 설치 관리자는 소프트웨어를 시스템 서비스로 설정하지 않으며, nucleus를 시스템 서비스로 설정하기 위한 성공 메시지가 표시되지 않습니다.

(선택 사항) Greengrass 소프트웨어 실행(Linux)

Note

이러한 단계는 nucleus lite에는 적용되지 않습니다.

소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않은 경우에는 사용자가 소프트웨어를 실행해야 합니다. 설치 관리자가 소프트웨어를 시스템 서비스로 설정했는지 확인하려면 설치 관리자 출력에서 다음 줄을 찾으세요.

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

이 메시지가 표시되지 않으면 다음을 수행하여 소프트웨어를 실행합니다.

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 Greengrass 코어 소프트웨어를 계속 실행하려면 현재 명령 셸을 열어 두어야 합니다. SSH를 사용하여 코어 디바이스에 연결하는 경우 개발 컴퓨터에서 다음 명령을 실행하여 코어 디바이스에서 추가 명령을 실행하는 데 사용할 수 있는 두 번째 SSH 세션을 엽니다. `username`을 로그인할 사용자의 이름으로 바꾸고 `pi-ip-address`를 디바이스의 IP 주소로 바꿉니다.

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

Greengrass 시스템 서비스와 상호 작용하는 방법에 대한 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 섹션을 참조하세요.

디바이스에서 Greengrass CLI 설치 확인

Note

이러한 단계는 nucleus lite에는 적용되지 않습니다.

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

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

coreDeviceExecutionStatus는 코어 디바이스에 대한 배포 상태를 나타냅니다. 상태가 SUCCEEDED인 경우 다음 명령을 실행하여 Greengrass CLI가 설치되고 실행되는지 확인합니다. */greengrass/v2*를 루트 폴더의 경로로 바꿉니다.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

이 명령은 Greengrass CLI에 대한 도움말 정보를 출력합니다. greengrass-cli를 찾을 수 없는 경우 배포에서 Greengrass CLI를 설치하지 못했을 수 있습니다. 자세한 내용은 [문제 해결 AWS IoT Greengrass V2](#) 단원을 참조하십시오.

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

- *region*을 사용하는 AWS 리전 로 바꿉니다. 디바이스 AWS CLI 에서를 구성하는 데 사용한 AWS 리전 것과 동일한를 사용해야 합니다.
- *account-id*를 AWS 계정 ID로 바꿉니다.
- *MyGreengrassCore*를 코어 디바이스의 이름으로 바꿉니다.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.16.1"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.16.1\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{"aws.greengrass.Cli":{"componentVersion":"2.16.1"}}'
```

 Tip

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

AWS IoT Greengrass 코어 소프트웨어 및 로컬 개발 도구는 디바이스에서 실행됩니다. 다음으로 디바이스에서 Hello World AWS IoT Greengrass 구성 요소를 개발할 수 있습니다.

4단계: 디바이스에서 구성 요소 개발 및 테스트

구성 요소는 AWS IoT Greengrass 코어 디바이스에서 실행되는 소프트웨어 모듈입니다. 구성 요소를 사용하면 복잡한 애플리케이션을 여기저기 Greengrass 코어 디바이스에서 재사용할 수 있는 불연속 빌딩 블록으로 생성하고 관리할 수 있습니다. 모든 구성 요소는 레시피와 아티팩트로 구성되어 있습니다.

- 레시피

모든 구성 요소에는 메타데이터가 정의되는 레시피 파일이 있습니다. 레시피에서는 구성 요소의 구성 파라미터, 구성 요소 종속성, 수명 주기 및 플랫폼 호환성도 지정됩니다. 구성 요소 수명 주기에서는 구성 요소 설치, 실행 및 종료 명령이 정의됩니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 단원을 참조하십시오.

레시피를 [JSON](#) 또는 [YAML](#) 형식으로 정의할 수 있습니다.

- 아티팩트

구성 요소의 구성 요소 바이너리인 아티팩트는 수량 제한이 없습니다. 아티팩트에는 구성 요소에서 사용되는 스크립트, 컴파일된 코드, 정적 리소스 및 기타 파일이 포함될 수 있습니다. 구성 요소에서는 구성 요소 종속성의 아티팩트도 사용될 수도 있습니다.

를 사용하면 Greengrass CLI를 사용하여 AWS 클라우드와의 상호 작용 없이 Greengrass 코어 디바이스에서 로컬로 구성 요소를 개발하고 테스트 AWS IoT Greengrass할 수 있습니다. 로컬 구성 요소를 완료하면 구성 요소 레시피와 아티팩트를 사용하여 AWS 클라우드의 AWS IoT Greengrass 서비스에서 해당 구성 요소를 생성한 다음 모든 Greengrass 코어 디바이스에 배포할 수 있습니다. 구성 요소에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#)를 참조하세요.

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

디바이스에서 Hello World 구성 요소를 개발하는 방법

1. 레시피 및 아티팩트의 하위 폴더로 구성 요소의 폴더를 생성합니다. Greengrass 코어 디바이스에서 다음 명령을 실행하여 이러한 폴더를 생성하고 구성 요소 폴더로 변경합니다. 로컬 개발에 사용할 폴더 경로로 `~/greengrassv2` 또는 `%USERPROFILE%\greengrassv2`를 바꿉니다.

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 구성 요소의 메타데이터, 파라미터, 종속성, 수명 주기 및 플랫폼 기능이 정의되는 레시피 파일을 생성합니다. 레시피에 반영되는 구성 요소 버전을 식별할 수 있도록 레시피 파일 이름에 구성 요소 버전을 포함합니다. 레시피에 YAML 또는 JSON 형식을 선택할 수 있습니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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 는 구성 요소에 의미 체계 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 1.0.0 버전은 구성 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

3. 파일에 다음 레시피를 붙여 넣습니다.

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 world인 파라미터(Message)가 정의합니다. Manifests 섹션에서는 플랫폼에 대한 수명 주기 지침과 아티팩트 세트인 매니페스트가 정의합니다. 예를 들어 여러 매니페스트를 정의하여 다양한 플랫폼에 상이한 설치 지침을 지정할 수 있습니다. 매니페스트의 Lifecycle 섹션에서는 Message 파라미터 값을 인수로 사용하여 Hello World 스크립트가 실행되도록 Greengrass 코어 디바이스에 지시합니다.

4. 다음 명령을 실행하여 구성 요소 아티팩트의 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. 레시피에서 지정하는 구성 요소 이름과 버전을 포함합니다.

```
artifacts/componentName/componentVersion/
```

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

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

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

다음 Python 스크립트를 복사하여 파일에 붙여 넣습니다.

```
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 AWS IoT Greengrass 코어에 구성 요소를 배포합니다. */greengrass/v2* 또는 *C:\greengrass\v2*를 AWS IoT Greengrass V2 루트 폴더로 바꾸고, *~/greengrassv2* 또는 *%USERPROFILE%\greengrassv2*를 구성 요소 개발 폴더로 바꿉니다.

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 명령에서는 recipes의 레시피가 사용되고 artifacts의 Python 스크립트가 사용되는 구성 요소가 추가됩니다. --merge 옵션에서는 지정하는 구성 요소와 버전이 추가 또는 업데이트됩니다.

7. AWS IoT Greengrass 코어 소프트웨어는 구성 요소 프로세스의 stdout을 logs 폴더의 로그 파일에 저장합니다. 다음 명령을 실행하여 Hello World 구성 요소가 실행되고 메시지가 인쇄되는지 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

다음 예제와 비슷한 메시지가 표시되어야 합니다.

```
Hello, world!
```

Note

파일이 없으면 로컬 배포가 아직 완료되지 않았을 수 있습니다. 15초 이내에 파일이 존재하지 않으면 배포에 실패했을 수 있습니다. 예를 들어 레시피가 유효하지 않은 경우에 실패가 발생할 수 있습니다. 다음 명령을 실행하여 AWS IoT Greengrass 코어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

- 로컬 구성 요소를 수정하여 코드를 반복하고 테스트합니다. 텍스트 편집기 hello_world.py에서 4행에 다음 코드를 추가하여 AWS IoT Greengrass 코어가 로깅하는 메시지를 편집합니다.

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

이제 hello_world.py 스크립트에 다음 내용이 있어야 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)
```

- 다음 명령을 실행하여 변경 사항으로 구성 요소를 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 명령에서는 최신 Hello World 아티팩트로 com.example.HelloWorld 구성 요소가 업데이트 됩니다.

10. 다음 명령을 실행하여 구성 요소를 다시 시작합니다. 구성 요소를 다시 시작하면 최신 변경 사항이 코어 디바이스에서 사용됩니다.

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 요소에서 새 메시지가 인쇄되는지 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

다음 예제와 비슷한 메시지가 표시되어야 합니다.

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

12. 구성 요소의 구성 파라미터를 업데이트하여 상이한 구성을 테스트할 수 있습니다. 구성 요소를 배포할 때 코어 디바이스에서 구성 요소의 구성이 수정되는 방법이 정의되는 구성 업데이트를 지정할 수 있습니다. 기본값으로 재설정할 구성 값과 코어 디바이스에 병합할 새 구성 값을 지정할 수 있습니다. 자세한 내용은 [구성 요소 구성 업데이트](#) 단원을 참조하십시오.

해결 방법:

- a. 텍스트 편집기를 사용하여 구성 업데이트가 포함되는 hello-world-config-update.json이라는 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

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

- b. 다음 JSON 객체를 복사하여 파일에 붙여 넣습니다. 이 JSON 객체에서는 값 friend가 Message 파라미터에 병합되어 값이 업데이트되는 구성 업데이트가 정의됩니다. 이 구성 업데이트에서는 재설정할 값이 지정되지 않습니다. 병합 업데이트를 통해 기존 값이 대체되므로 Message 파라미터를 재설정할 필요가 없습니다.

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

- c. 다음 명령을 실행하여 구성 업데이트를 Hello World 구성 요소에 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 요소에서 새 메시지가 출력되는지 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

다음 예제와 비슷한 메시지가 표시되어야 합니다.

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

13. 구성 요소는 테스트를 마치면 코어 디바이스에서 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 AWS IoT Greengrass에 업로드한 후 코어 디바이스에 다시 배포하려면 이 단계가 필요합니다. 그렇지 않으면 로컬 배포에서 다른 구성 요소 버전이 지정되므로 버전 호환성 오류로 배포가 실패합니다.

다음 명령을 실행하고 디바이스의 com.example.HelloWorld 구성 요소 목록에 구성 요소가 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 완료되었으며 이제 AWS IoT Greengrass 클라우드 서비스에 업로드할 수 있습니다. 그런 다음에 구성 요소를 다른 코어 디바이스에 배포할 수 있습니다.

5단계: AWS IoT Greengrass 서비스에서 구성 요소 생성

코어 디바이스에서 구성 요소 개발을 완료하면 이를 AWS 클라우드의 AWS IoT Greengrass 서비스에 업로드할 수 있습니다. [AWS IoT Greengrass 콘솔](#)에서 구성 요소를 직접 생성할 수도 있습니다. AWS IoT Greengrass 는 개별 디바이스 또는 디바이스 플릿에 배포할 수 있도록 구성 요소를 호스팅하는 구성 요소 관리 서비스를 제공합니다. AWS IoT Greengrass 서비스에 구성 요소를 업로드하려면 다음 단계를 완료합니다.

- S3 버킷에 구성 요소 아티팩트를 업로드합니다.
- 각 아티팩트의 Amazon Simple Storage Service(Amazon S3) URI를 구성 요소 레시피에 추가합니다.
- 구성 요소 레시피 AWS IoT Greengrass 에서에 구성 요소를 생성합니다.

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

에서 구성 요소 생성 AWS IoT Greengrass (콘솔)

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

기존 S3 버킷을 사용하거나 새 버킷을 생성할 수 있습니다.

- [Amazon S3 콘솔](#)의 버킷에서 버킷 생성을 선택합니다.
- 버킷 이름에 고유한 버킷 이름을 입력합니다. 예를 들어 **greengrass-component-artifacts-*region*-123456789012**를 사용할 수 있습니다. **123456789012**을 AWS 계정 ID로 바꾸고 **##**을 자습서에서 AWS 리전 사용하는 로 바꿉니다.
- AWS 리전에서 이 자습서에 사용할 AWS 리전을 선택합니다.
- 버킷 생성을 선택합니다.
- 버킷에서 생성한 버킷을 선택하고 `hello_world.py` 스크립트를 버킷의 `artifacts/com.example.HelloWorld/1.0.0` 폴더에 업로드합니다. S3 버킷에 객체를 업로드하는 데 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [객체 업로드](#)를 참조하세요.
- S3 버킷에 있는 `hello_world.py` 객체의 S3 URI를 복사합니다. 이 URI는 다음 예제와 유사할 것입니다. `amzn-s3-demo-bucket`을 S3 버킷의 이름으로 바꿉니다.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

- 코어 디바이스가 S3 버킷의 구성 요소 아티팩트에 액세스하도록 허용합니다.

각 코어 디바이스에는 AWS 클라우드와 상호 작용 AWS IoT 하고 로그를 전송할 수 있는 [코어 디바이스 IAM 역할](#)이 있습니다. 이 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않으므로 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색할 수 있도록 허용하는 정책을 생성하고 연결해야 합니다.

디바이스의 역할이 이미 S3 버킷에 대한 액세스를 허용하는 경우 이 단계를 건너뛸 수 있습니다. 그렇지 않으면 다음과 같이 액세스를 허용하는 IAM 정책을 생성하여 역할에 연결합니다.

- [IAM 콘솔](#) 탐색 메뉴에서 정책을 선택한 다음 정책 생성을 선택합니다.
- JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다. `amzn-s3-demo-bucket`을 코어 디바이스가 다운로드할 구성 요소 아티팩트를 포함하는 S3 버킷의 이름으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ]
    }
  ]
}
```

```
    ],
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
  }
]
}
```

- c. 다음을 선택합니다.
 - d. 정책 세부 정보 섹션에서 이름에 **MyGreengrassV2ComponentArtifactPolicy**를 입력합니다.
 - e. 정책 생성을 선택합니다.
 - f. [IAM 콘솔](#) 탐색 메뉴에서 역할을 선택한 다음 코어 디바이스에 대한 역할 이름을 선택합니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 때이 역할 이름을 지정했습니다. 이름을 지정하지 않은 경우 기본값은 GreengrassV2TokenExchangeRole입니다.
 - g. 권한에서 권한 추가, 정책 연결을 차례로 선택합니다.
 - h. 권한 추가 페이지에서 생성한 MyGreengrassV2ComponentArtifactPolicy 정책 옆의 확인란을 선택한 다음 권한 추가를 선택합니다.
3. 구성 요소 레시피를 사용하여 [AWS IoT Greengrass 콘솔](#)에서 구성 요소를 생성합니다.
 - a. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택한 다음 구성 요소 생성을 선택합니다.
 - b. 구성 요소 정보에서 레시피를 JSON으로 입력을 선택합니다. 자리 표시자 레시피는 다음 예제와 유사할 것입니다.

```
{
  "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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}

```

- c. 각 Artifacts 섹션의 자리 표시자 URI를 hello_world.py 객체의 S3 URI로 바꿉니다.
- d. 구성 요소 생성을 선택합니다.
- e. com.example.HelloWorld 구성 요소 페이지에서 구성 요소의 상태가 배포 가능한지 확인합니다.

에서 구성 요소 생성 AWS IoT Greengrass (AWS CLI)

Hello World 구성 요소를 업로드하려면

1. 에서 S3 버킷을 사용하여 AWS IoT Greengrass 구성 요소 아티팩트 AWS 계정을 호스팅합니다. 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 ID 및 로 버킷을 생성 AWS 리전 하여 고유한 버킷 이름을 형성합니다. **123456789012**을 AWS 계정 ID로 바꾸고 **##**을 이 자습서에서 AWS 리전 사용하는 로 바꿉니다.

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

요청에 성공하는 경우 명령은 다음 정보를 출력합니다.

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

2. 코어 디바이스가 S3 버킷의 구성 요소 아티팩트에 액세스하도록 허용합니다.

각 코어 디바이스에는와 상호 작용 AWS IoT 하고 로그를 전송할 수 있는 [코어 디바이스 IAM 역할](#)이 있습니다 AWS 클라우드. 이 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않으므로 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색할 수 있도록 허용하는 정책을 생성하고 연결해야 합니다.

코어 디바이스의 역할이 이미 S3 버킷에 대한 액세스를 허용하는 경우 이 단계를 건너뛸 수 있습니다. 그렇지 않으면 다음과 같이 액세스를 허용하는 IAM 정책을 생성하여 역할에 연결합니다.

- a. `component-artifact-policy.json`이라는 파일을 만들고 다음 JSON을 파일로 복사합니다. 이 정책은 S3 버킷의 모든 파일에 대한 액세스를 허용합니다. `amzn-s3-demo-bucket`을 S3 버킷의 이름으로 바꿉니다.

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

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

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 정책 Amazon 리소스 이름(ARN)을 복사합니다. 이 ARN을 사용하여 다음 단계에서 이 정책을 코어 디바이스 역할에 연결합니다.

- c. 다음 명령을 실행하여 정책을 코어 디바이스 역할에 연결합니다.
*GreengrassV2TokenExchangeRole*을 코어 디바이스에 대한 역할의 이름으로 바꿉니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 때 이 역할 이름을 지정했습니다. 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
```

명령의 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 스크립트를 AWS IoT Greengrass 코어에 스크립트가 있는 버킷의 동일한 경로에 업로드합니다. `amzn-s3-demo-bucket`을 S3 버킷의 이름으로 바꿉니다.

Linux or Unix

```
aws s3 cp \
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py \
  s3://amzn-s3-demo-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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

요청에 성공하는 경우 명령은 `upload:`로 시작하는 줄을 출력합니다.

4. 아티팩트의 Amazon S3 URI를 구성 요소 레시피에 추가합니다.

Amazon S3 URI는 버킷 이름과 버킷의 아티팩트 객체 경로로 구성되어 있습니다. 스크립트 아티팩트의 Amazon S3 URI는 이전 단계에서 아티팩트를 업로드한 URI입니다. 이 URI는 다음 예제와 유사할 것입니다. `amzn-s3-demo-bucket`을 S3 버킷의 이름으로 바꿉니다.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

레시피에 아티팩트를 추가하려면 Amazon S3 URI를 사용하여 구조가 포함된 Artifacts의 목록을 추가합니다.

JSON

```
"Artifacts": [  
  {  
    "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/  
hello_world.py"  
  }  
]
```

텍스트 편집기에서 레시피 파일을 엽니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

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

레시피에 아티팩트를 추가합니다. 레시피 파일은 다음 예제와 유사할 것입니다.

```
{  
  "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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
      }
    ]
  }
]
}

```

YAML

```

Artifacts:
  - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
    hello_world.py

```

텍스트 편집기에서 레시피 파일을 엽니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

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

레시피에 아티팩트를 추가합니다. 레시피 파일은 다음 예제와 유사할 것입니다.

```
---
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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
      hello_world.py
```

- 레시피 AWS IoT Greengrass 에서에 구성 요소 리소스를 생성합니다. 다음 명령을 실행하여 이진 파일로 제공하는 레시피에서 구성 요소를 생성합니다.

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
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

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

출력에서 arn을 복사하여 다음 단계에서 구성 요소의 상태를 확인합니다.

 Note

구성 요소 페이지의 [AWS IoT Greengrass 콘솔](#)에서 Hello World 구성 요소를 확인할 수도 있습니다.

- 구성 요소가 생성되고 배포할 준비가 되었는지 확인합니다. 구성 요소를 생성할 때 구성 요소의 상태는 REQUESTED입니다. 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. arn을 이전 단계의 ARN으로 바꿉니다.

```
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 DEPLOYABLE임을 나타냅니다.

```
{
```



```

"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"
  }
]
}

```

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

6단계: 구성 요소 배포

AWS IoT Greengrass를 통해 개별 디바이스 또는 디바이스 그룹에 구성 요소를 배포할 수 있습니다. 구성 요소를 배포하면 AWS IoT Greengrass에서는 대상 디바이스마다 해당 구성 요소의 소프트웨어가 설치 및 실행됩니다. 구성 요소마다 배포할 구성 요소와 배포할 구성 업데이트를 지정합니다. 배포 대상인 디바이스로 배포가 돌아옴되는 방식을 제어할 수도 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.

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

구성 요소를 배포하는 방법(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 내 구성 요소 탭에서 com.example.HelloWorld를 선택합니다.
3. com.example.HelloWorld 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 새 배포 생성과 다음을 차례로 선택합니다.

5. 대상 지정 페이지에서 다음 작업을 수행합니다.
 - a. 이름(Name) 상자에 **Deployment for MyGreengrassCore**를 입력합니다.
 - b. 배포 대상에는 코어 디바이스와 코어 디바이스의 AWS IoT 사물 이름을 선택합니다. 이 자습서에서는 기본값이 *MyGreengrassCore*입니다.
 - c. 다음을 선택합니다.
6. 구성 요소 선택 페이지의 내 구성 요소에서 com.example.HelloWorld 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 com.example.HelloWorld를 선택하고 다음을 수행합니다.
 - a. 구성 요소 구성을 선택합니다.
 - b. 구성 업데이트 아래에 있는 병합할 구성에 다음 구성을 입력합니다.

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

이 구성 업데이트에서는 이 배포의 디바이스에 대해 Hello World Message 파라미터가 universe로 설정됩니다.

- c. 확인을 선택합니다.
 - d. 다음을 선택합니다.
8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 배포를 선택합니다.
10. 배포가 완료되었는지 확인합니다. 배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. Hello World 로그를 점검하여 변경을 확인합니다. 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 예제와 비슷한 메시지가 표시되어야 합니다.

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

Note

로그 메시지가 변경되지 않으면 배포에 실패했거나 코어 디바이스에 도달하지 못한 것입니다. 코어 디바이스가 인터넷에 연결되어 있지 않거나 S3 버킷에서 아티팩트를 검색할 권한이 없으면 이 문제가 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

자세한 내용은 [문제 해결 AWS IoT Greengrass V2](#) 섹션을 참조하세요.

구성 요소 배포(AWS CLI)

Hello World 구성 요소를 배포하려면

1. 개발 컴퓨터에서 `hello-world-deployment.json`이라는 파일을 생성하고 다음 JSON을 파일로 복사합니다. 이 파일에서는 배포되는 구성 요소와 구성이 정의됩니다.

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

이 구성 파일에서는 이전 절차에서 개발하고 게시한 Hello World 구성 요소의 1.0.0 버전 배포가 지정됩니다. `configurationUpdate`에서는 JSON에서 인코딩된 문자열로 구성 요소 구성 병합이 지정됩니다. 이 구성 업데이트에서는 이 배포의 디바이스에 대해 Hello World Message 파라미터가 `universe`로 설정됩니다.

2. 다음 명령을 실행하여 Greengrass 코어 디바이스에 구성 요소를 배포합니다. 개별 디바이스인 사물 또는 디바이스 그룹인 사물 그룹에 배포할 수 있습니다. 코어의 AWS IoT 사물 이름으로 *MyGreengrassCore*를 바꿉니다.

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
```

다음 예제 출력과 비슷한 출력이 명령에서 반환됩니다.

```
{
  "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 완료되었는지 확인합니다. 배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. Hello World 로그를 점검하여 변경을 확인합니다. 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 예제와 비슷한 메시지가 표시되어야 합니다.

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

 Note

로그 메시지가 변경되지 않으면 배포에 실패했거나 코어 디바이스에 도달하지 못한 것입니다. 코어 디바이스가 인터넷에 연결되어 있지 않거나 S3 버킷에서 아티팩트를 검색할 권한이 없으면 이 문제가 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

자세한 내용은 [문제 해결 AWS IoT Greengrass V2](#) 섹션을 참조하세요.

다음 단계

이 자습서를 완료했습니다. AWS IoT Greengrass 코어 소프트웨어와 Hello World 구성 요소가 디바이스에서 실행됩니다. 또한 Hello World 구성 요소는 AWS IoT Greengrass 클라우드 서비스에서 다른 디바이스에 배포할 수 있습니다. 이 자습서에서 살펴보는 주제에 대한 자세한 내용은 다음을 참조하세요.

- [AWS IoT Greengrass 구성 요소 생성](#)
- [코어 디바이스에 배포할 구성 요소를 게시하세요.](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)

AWS IoT Greengrass 코어 디바이스 설정

이 섹션의 작업을 완료하여 AWS IoT Greengrass 코어 소프트웨어를 설치, 구성 및 실행합니다.

Note

이 섹션에서는 AWS IoT Greengrass 코어 소프트웨어의 고급 설치 및 구성에 대해 설명합니다. 이 단계는 nucleus lite에는 적용되지 않습니다. 처음 사용하는 경우 먼저 [시작하기 자습서](#)를 완료하여 코어 디바이스를 설정하고 기능을 살펴보는 AWS IoT Greengrass V2것이 좋습니다 AWS IoT Greengrass.

주제

- [지원하는 플랫폼](#)
- [장치 요구 사항](#)
- [Lambda 함수 요구 사항](#)
- [설정 AWS 계정](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [AWS IoT Greengrass 코어 소프트웨어 실행](#)
- [Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행](#)
- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#)
- [AWS IoT Greengrass 코어 소프트웨어 제거](#)

지원하는 플랫폼

- [Greengrass nucleus 지원 플랫폼](#)
- [Greengrass nucleus lite 지원 플랫폼](#)

장치 요구 사항

- [Greengrass nucleus 디바이스 요구 사항](#)

- [Greengrass nucleus lite 디바이스 요구 사항](#)

Lambda 함수 요구 사항

Important

Greengrass Lambda 함수는 현재 Greengrass nucleus lite에서 지원되지 않습니다.

Lambda 함수를 실행하려면 디바이스가 다음과 같은 요구 사항을 충족해야 합니다.

- Linux 기반 운영 체제.
- 디바이스에 `mkfifo` 셸 명령이 있어야 합니다.
- 디바이스에서 Lambda 함수에 필요한 프로그래밍 언어 라이브러리를 실행해야 합니다. 디바이스에 필수 라이브러리를 설치하고 `PATH` 환경 변수에 추가해야 합니다. Greengrass는 Python, Node.js, Java 런타임의 모든 Lambda 지원 버전을 지원합니다. Greengrass는 더 이상 사용되지 않는 Lambda 런타임 버전에 추가 제한 사항을 적용하지 않습니다. Lambda 런타임 AWS IoT Greengrass 지원에 대한 자세한 내용은 섹션을 참조하세요 [AWS Lambda 함수 실행](#).
- 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 다음과 같은 요구 사항을 충족해야 합니다.
 - Linux 커널 버전 4.4 이상.
 - 커널은 [cgroups](#) v1을 지원해야 하며 사용자가 다음 cgroups를 활성화하고 탑재해야 합니다.
 - 컨테이너화된 Lambda 함수의 메모리 제한을 설정 AWS IoT Greengrass 하기 위한의 메모리 cgroup입니다.
 - 시스템 디바이스 또는 볼륨에 액세스하기 위한 컨테이너화된 Lambda 함수의 `devices` cgroup.

AWS IoT Greengrass 코어 소프트웨어는 cgroups v2를 지원하지 않습니다.

이 요구 사항을 충족하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

Tip

Raspberry Pi에서 `/boot/cmdline.txt` 파일을 편집하여 디바이스의 커널 파라미터를 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 사용 중인 Linux 배포의 설명서를 확인합니다. AWS IoT Device Tester 용 AWS IoT Greengrass 를 사용하여 디바이스가 이러한 요구 사항을 충족하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass V2용 AWS IoT Device Tester 사용](#) 단원을 참조하십시오.

설정 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. <https://portal.aws.amazon.com/billing/signup>을 엽니다

설정 AWS 계정

2. 온라인 지시 사항을 따릅니다.

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

에 가입하면 AWS 계정 AWS 계정 루트 사용자인 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 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 사례 를 참조하세요.	AWS IAM Identity Center 사용 설명서의 시작하기 지침을 따릅니다.	AWS Command Line Interface 사용 설명서에서 사용하도록 AWS CLI를 구성 AWS IAM Identity Center 하여 프로그래밍 방식 액세스를 구성합니다.
IAM에서 (권장되지 않음)	장기 보안 인증 정보를 사용하여 AWS에 액세스합니다.	IAM 사용 설명서의 비상 액세스를 위한 IAM 사용자 생성 에 나와 있는 지침을 따르세요.	IAM 사용 설명서에 나온 IAM 사용자의 액세스 키 관리 단계를 수행하여 프로그래밍 방식의 액세스를 구성합니다.

AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass는 AWS를 엣지 디바이스로 확장하면서 생성한 데이터를 작업할 수 있고, 관리, 분석 및 장기 저장 용도로 AWS 클라우드를 사용합니다. AWS IoT Greengrass 및 AWS 클라우드와 통합할 엣지 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하고 설치하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인하세요.

AWS IoT Greengrass 코어 소프트웨어에는 디바이스를 Greengrass 코어 디바이스로 설정하는 설치 관리자가 포함되어 있습니다. 설치 관리자를 실행할 때 사용할 루트 폴더 및 AWS 리전과 같은 옵션을 구성할 수 있습니다. 설치 관리자에서 필요한 AWS IoT 및 IAM 리소스를 생성하도록 선택할 수 있습니다. 로컬 개발 도구를 배포하여 사용자 지정 구성 요소 개발에 사용하는 디바이스를 구성하도록 선택할 수도 있습니다.

AWS IoT Greengrass 코어 소프트웨어에서 AWS 클라우드에 연결하고 작동하려면 다음 AWS IoT 및 IAM 리소스가 필요합니다.

- AWS IoT 사물. 디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 AWS에 인증할 수 있습니다. 이 인증서를 사용하면 디바이스가 AWS IoT 및 AWS IoT Greengrass와 통신할 수 있습니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 섹션을 참조하세요.
- (선택 사항) AWS IoT 사물 그룹. 사물 그룹을 사용하여 Greengrass 코어 디바이스 플릿을 관리합니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹에 배포할 수 있습니다. 사물 그룹에 디바이스를 추가하여 사물 그룹의 소프트웨어 구성 요소를 디바이스에 배포할 수 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.
- IAM 역할. Greengrass 코어 디바이스에서는 AWS IoT Core 자격 증명 공급자를 통해 IAM 역할을 사용하여 AWS 서비스를 직접 호출할 권한을 부여합니다. 이 역할을 사용하면 디바이스가 AWS IoT와 상호 작용하고, Amazon CloudWatch Logs로 로그를 전송하고, Amazon Simple Storage Service(Amazon S3)에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.
- AWS IoT 역할 별칭. Greengrass 코어 디바이스는 역할 별칭을 사용하여 사용할 IAM 역할을 식별합니다. 역할 별칭을 사용하면 IAM 역할을 변경하면서 디바이스 구성을 동일하게 유지할 수 있습니다.

자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스 직접 호출에 대한 권한 부여](#)를 참조하세요.

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

- 빠른 설치

이 옵션을 선택하면 가능한 짧은 단계를 거쳐 Greengrass 코어 디바이스를 설정할 수 있습니다. 설치 관리자로 필요한 AWS IoT 및 IAM 리소스가 생성됩니다. 이 옵션을 사용하려면 설치 관리자에 AWS 자격 증명을 제공하여 AWS 계정에서 리소스를 생성해야 합니다.

이 옵션을 사용하여 방화벽 또는 네트워크 프록시 뒤에 설치할 수 없습니다. 디바이스가 방화벽 또는 네트워크 프록시 뒤에 있는 경우 [수동 설치](#)를 고려하세요.

자세한 내용은 [자동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 섹션을 참조하세요.

- 수동 설치

필요한 AWS 리소스를 수동으로 생성하거나 방화벽 또는 네트워크 프록시 뒤에 설치하려면 이 옵션을 선택합니다. 수동 설치를 사용하면 필요한 AWS IoT 및 IAM 리소스를 생성하므로 설치 관리자에 AWS 계정에서 리소스를 생성할 권한을 부여할 필요가 없습니다. 포트 443 또는 네트워크 프록시를 통해 연결하도록 디바이스를 구성할 수도 있습니다. HSM(하드웨어 보안 모듈), TPM(Trusted Platform Module) 또는 다른 암호화 요소에 저장하는 프라이빗 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수도 있습니다.

자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 섹션을 참조하세요.

- AWS IoT 플릿 프로비저닝을 사용한 설치

이 옵션을 선택하여 AWS IoT 플릿 프로비저닝 템플릿에서 필요한 AWS 리소스를 생성합니다. 플릿에서 유사한 디바이스를 생성하거나 차량 또는 스마트 홈 디바이스와 같이 고객이 나중에 활성화하는 디바이스를 제조하는 경우 이 옵션을 선택할 수 있습니다. 디바이스는 클레임 인증서를 사용하여 디바이스가 정상 작동을 위해 AWS 클라우드에 연결하는 데 사용하는 X.509 클라이언트 인증서를 포함한 AWS 리소스를 인증 및 프로비저닝합니다. 제조 중에 클레임 인증서를 디바이스의 하드웨어에 임베드하거나 플래시할 수 있으며, 동일한 클레임 인증서와 키를 사용하여 여러 디바이스를 프로비저닝할 수 있습니다. 포트 443 또는 네트워크 프록시를 통해 연결하도록 디바이스를 구성할 수도 있습니다.

자세한 내용은 [AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 섹션을 참조하세요.

- 사용자 지정 프로비저닝을 사용한 설치

이 옵션을 선택하면 필요한 AWS 리소스를 프로비저닝하는 사용자 지정 Java 애플리케이션을 개발합니다. [자체 X.509 클라이언트 인증서를 생성](#)하거나 프로비저닝 프로세스를 효과적으로 제어하려는 경우 이 옵션을 선택할 수 있습니다. AWS IoT Greengrass는 사용자 지정 프로비저닝 애플리케이션과 AWS IoT Greengrass 코어 소프트웨어 설치 관리자 간에 정보를 교환하기 위해 구현할 수 있는 인터페이스를 제공합니다.

자세한 내용은 [사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 섹션을 참조하세요.

또한 AWS IoT Greengrass는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 컨테이너화된 환경을 제공합니다. Dockerfile을 사용하여 [Docker 컨테이너에서 AWS IoT Greengrass를 실행](#)할 수 있습니다.

주제

- [자동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [설치 프로그램 인수](#)

자동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass 코어 소프트웨어에는 디바이스를 Greengrass 코어 디바이스로 설정하는 설치 프로그램이 포함되어 있습니다. 디바이스를 빠르게 설정하기 위해 설치 관리자는 코어 디바이스가 작동하는 데 필요한 AWS IoT 사물, AWS IoT 사물 그룹, IAM 역할 및 AWS IoT 역할 별칭을 프로비저닝할 수 있습니다. 설치 관리자는 또한 로컬 개발 도구를 코어 디바이스에 배포할 수 있으므로 디바이스를 사용하여 사용자 지정 소프트웨어 구성 요소를 개발하고 테스트할 수 있습니다. 설치 관리자는 이러한 리소스를 프로비저닝하고 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 디바이스에 배포하여 개발 디바이스로 사용할 수도 있

습니다. 이렇게 하면 설치 관리자 실행 시 디바이스에 더 적은 권한을 제공할 수 있습니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인합니다.

주제

- [디바이스 환경 설정](#)
- [디바이스에 AWS 자격 증명 제공](#)
- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)

디바이스 환경 설정

이 섹션의 단계에 따라 AWS IoT Greengrass 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

Windows 디바이스 설정

Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
 - a. Windows 키를 눌러 시작 메뉴를 엽니다.
 - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
 - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
 - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
 - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
 - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.

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


- g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- wmic 명령이 더는 사용되지 않는 Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

Note

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

디바이스에 AWS 자격 증명 제공

설치 관리자가 필요한 AWS 리소스를 프로비저닝할 수 있도록 디바이스에 AWS 자격 증명을 제공합니다. 필요한 권한에 대한 자세한 정보는 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

디바이스에 AWS 자격 증명을 제공하려면

- 설치 관리자가 코어 디바이스에 대한 AWS IoT 및 IAM 리소스를 프로비저닝할 수 있도록 디바이스에 AWS 자격 증명을 제공합니다. 보안을 강화하려면 프로비저닝에 필요한 최소 권한만 허용하는 IAM 역할에 대한 임시 자격 증명을 받는 것이 좋습니다. 자세한 내용은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

Note

설치 관리자에서는 자격 증명을 저장하거나 저장하지 않습니다.

디바이스에서 다음 중 하나를 수행하여 자격 증명을 검색하고 AWS IoT Greengrass 코어 소프트웨어 설치 관리자가 사용할 수 있도록 합니다.

- (권장)의 임시 자격 증명 사용 AWS IAM Identity Center
 - a. IAM Identity Center에서 액세스 키 ID, 비밀 액세스 키, 세션 토큰을 제공합니다. 자세한 내용은 IAM Identity Center 사용 설명서의 [임시 자격 증명 가져오기 및 새로 고침](#)에서 수동 자격 증명 새로 고침을 참조하세요.
 - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- IAM 역할에서 임시 보안 인증을 사용합니다.
 - a. 수입하는 IAM 역할에서 액세스 키 ID, 비밀 액세스 키, 세션 토큰을 제공합니다. 이러한 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서에서 [임시 보안 인증 요청](#)을 참조하세요.
 - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- IAM 사용자의 장기 자격 증명 사용:
 - a. IAM 사용자의 액세스 키 ID 및 비밀 액세스 키를 제공합니다. 나중에 삭제하는 프로비저닝을 위한 IAM 사용자를 생성할 수 있습니다. 사용자에게 제공할 IAM 정책은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요. 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서에서 [IAM 사용자의 액세스 키 관리](#)를 참조하세요.
 - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

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 사용자를 삭제합니다.
- d. (선택 사항) 기존 IAM 사용자의 액세스 키 ID 및 비밀 액세스 키를 사용한 경우 해당 키가 더 이상 유효하지 않도록 사용자의 키를 업데이트합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [액세스 키 업데이트](#)를 참조하세요.

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

다음 위치에서 AWS IoT Greengrass 코어 소프트웨어의 특정 버전을 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다. `greengrass-nucleus-latest.zip`.

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 계약](#)에 동의하는 것입니다.

2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

Note

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.

- a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6_10*을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6_10*을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.

- i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

- ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. -verify 및 -certs 옵션과 함께 Jarsigner -verbose 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다.

*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후이 폴더를 제거할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자 실행:

- 코어 디바이스가 작동하는 데 필요한 AWS 리소스를 생성합니다.
- ggc_user 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 ggc_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.

- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

⚠ Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

로컬 개발 도구를 사용하여 개발 디바이스를 설정하려면 `--deploy-dev-tools true` 인수를 지정합니다. 설치 완료 후 로컬 개발 도구를 배포하는 데 최대 1분이 걸릴 수 있습니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

ℹ Note

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 nucleus 구성 요소의 `jvmOptions` 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. AWS IoT Greengrass 코어 설치 관리자를 실행합니다. 다음과 같이 명령의 인수 값을 바꿉니다.

ℹ Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 `C:\greengrass\v2` 또는 `D:\greengrass\v2`와 같은 루트 폴더를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

- a. `/greengrass/v2` 또는 `C:\greengrass\v2`: AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.
- b. `GreengrassInstaller`. AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 압축 해제한 폴더의 경로입니다.
- c. `region`. 리소스를 찾거나 생성할 AWS 리전입니다.

- d. **MyGreengrassCore**. Greengrass 코어 디바이스에 대한 AWS IoT 사물의 이름입니다. 사물이 존재하지 않는 경우 설치 관리자가 이를 생성합니다. 설치 관리자는 인증서를 다운로드하여 AWS IoT 사물로 인증합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.

 Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

- e. **MyGreengrassCoreGroup**. Greengrass 코어 디바이스의 AWS IoT 사물 그룹 이름입니다. 사물 그룹이 존재하지 않는 경우 설치 관리자가 이를 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 활성 배포가 있는 경우 코어 디바이스는 배포에서 지정한 소프트웨어를 다운로드하고 실행합니다.

 Note

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

- f. **GreengrassV2IoTThingPolicy**. Greengrass 코어 디바이스가 AWS IoT 및와 통신할 수 있도록 허용하는 AWS IoT 정책의 이름입니다 AWS IoT Greengrass. 정책이 없는 경우 AWS IoT 설치 관리자는이 이름으로 허용 AWS IoT 정책을 생성합니다. 사용 사례에 대해 이 정책의 권한을 제한할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.
- g. **GreengrassV2TokenExchangeRole**. Greengrass 코어 디바이스가 임시 AWS 자격 증명을 가져오도록 허용하는 IAM 역할의 이름입니다. 역할이 없는 경우 설치 관리자는 이를 생성하고 **GreengrassV2TokenExchangeRoleAccess**라는 정책을 생성 및 연결합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.
- h. **GreengrassCoreTokenExchangeRoleAlias**. Greengrass 코어 디바이스가 나중에 임시 자격 증명을 가져올 수 있도록 허용하는 IAM 역할의 별칭입니다. 역할 별칭이 없는 경우 설치 관리자에서 이를 생성하고 지정한 IAM 역할로 가리킵니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

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 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

설치 관리자가 성공하면 다음 메시지가 인쇄됩니다.

- --provision을 지정하면 리소스가 성공적으로 구성된 경우 설치 관리자에서 Successfully configured Nucleus with provisioned resource details가 인쇄됩니다.
 - --deploy-dev-tools를 지정하면 배포가 성공적으로 생성된 경우 설치 관리자에서 Configured Nucleus to deploy aws.greengrass.Cli component가 인쇄됩니다.
 - --setup-system-service true를 지정하면 소프트웨어가 설정되고 서비스로 실행되면 설치 관리자에서 Successfully set up Nucleus as a system service가 인쇄됩니다.
 - --setup-system-service true를 지정하지 않으면 소프트웨어가 실행되면 설치 관리자에서 Launched Nucleus successfully가 인쇄됩니다.
2. [Greengrass nucleus](#) v2.0.4 이상을 설치한 경우 이 단계를 건너뛰니다. 최신 버전의 소프트웨어를 다운로드한 경우 v2.0.4 이상을 설치한 것입니다.

다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 루트 폴더에 필요한 파일 권한을 설정합니다. `/greengrass/v2`를 설치 명령에서 지정한 루트 폴더로 바꾸고 `/greengrass`를 루트 폴더의 상위 폴더로 바꿉니다.

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

i Note

기본적으로 설치 관리자가 생성하는 IAM 역할은 S3 버킷의 구성 요소 아티팩트에 대한 액세스를 허용하지 않습니다. Amazon S3에서 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하

려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오. 구성 요소 아티팩트에 대한 S3 버킷이 아직 없는 경우 버킷을 생성한 후 권한을 추가할 수 있습니다.

Note

AWS IoT Greengrass 코어 소프트웨어가 클라우드에 연결되면 디바이스가 코어 디바이스로 인식됩니다.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass 참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass 코어 소프트웨어에는 디바이스를 Greengrass 코어 디바이스로 설정하는 설치 프로그램이 포함되어 있습니다. 디바이스를 수동으로 설정하려면 디바이스에서 사용할 필수 AWS IoT 및 IAM 리소스를 생성할 수 있습니다. 이러한 리소스를 수동으로 생성하는 경우 설치 프로그램에 AWS 자격 증명을 제공할 필요가 없습니다.

AWS IoT Greengrass 코어 소프트웨어를 수동으로 설치할 때 네트워크 프록시를 사용하거나 포트 443에서 AWS에 연결하도록 디바이스를 구성할 수도 있습니다. 예를 들어 디바이스가 방화벽 또는 네트워크 프록시 뒤에서 실행되는 경우 이러한 구성 옵션을 지정해야 할 수 있습니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 단원을 참조하십시오.

[PKCS#11 인터페이스](#)를 통해 하드웨어 보안 모듈(HSM)을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수도 있습니다. 이 기능을 사용하면 프라이빗 키와 인증서 파일을 소프트웨어에 노출되거나 복제되지 않도록 안전하게 저장할 수 있습니다. 프라이빗 키와 인증서를 HSM, TPM(Trusted Platform Module) 또는 다른 암호화 요소와 같은 하드웨어 모듈에 저장할 수 있습니다. 이 기능은 Linux

디바이스에서만 사용할 수 있습니다. 하드웨어 보안 및 이를 사용하는 데 필요한 요구 사항에 대한 자세한 내용은 [하드웨어 보안 통합](#) 섹션을 참조하세요.

⚠ Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인합니다.

주제

- [AWS IoT 엔드포인트 검색](#)
- [AWS IoT 사물 생성](#)
- [사물 인증서 생성](#)
- [사물 인증서 구성](#)
- [토큰 교환 역할 생성](#)
- [디바이스에 인증서 다운로드](#)
- [디바이스 환경 설정](#)
- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)

AWS IoT 엔드포인트 검색

의 AWS IoT 엔드포인트를 가져 AWS 계정과 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. 해결 방법:

1. 의 AWS IoT 데이터 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 의 AWS IoT 자격 증명 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

AWS IoT 사물 생성

AWS IoT 사물에는 연결하는 디바이스 및 논리적 엔터티를 나타냅니다 AWS IoT. Greengrass 코어 디바이스는 AWS IoT 사물입니다. 디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다 AWS.

이 섹션에서는 디바이스를 나타내는 AWS IoT 사물을 생성합니다.

AWS IoT 사물을 생성하려면

1. 디바이스에 대한 AWS IoT 사물을 생성합니다. 개발 컴퓨터에서 다음 명령을 실행합니다.
 - *MyGreengrassCore*를 사용할 사물 이름으로 바꿉니다. 이 이름은 Greengrass 코어 디바이스의 이름이기도 합니다.

Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

```
aws iot create-thing --thing-name MyGreengrassCore
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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 또는 기존 사물 그룹에 사물을 추가합니다. 사물 그룹을 사용하여 Greengrass 코어 디바이스 플릿을 관리합니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포가 있는 사물 그룹에 디바이스를 추가하여 사물 그룹의 소프트웨어 구성 요소를 디바이스에 배포할 수 있습니다. 해결 방법:

- a. (선택 사항) AWS IoT 사물 그룹을 생성합니다.

- *MyGreengrassCoreGroup*을 생성할 사물 그룹의 이름으로 바꿉니다.

 Note

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. AWS IoT 사물을 사물 그룹에 추가합니다.

- *MyGreengrassCore*를 AWS IoT 사물의 이름으로 바꿉니다.
- *MyGreengrassCoreGroup*을 사물 그룹의 이름으로 바꿉니다.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 명령에 출력이 없습니다.

사물 인증서 생성

디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다 AWS. 이 인증서를 사용하면 디바이스가 AWS IoT 및와 통신할 수 있습니다 AWS IoT Greengrass.

이 섹션에서는 디바이스가 AWS에 연결하는 데 사용할 수 있는 인증서를 생성하고 다운로드합니다.

하드웨어 보안 모듈(HSM)을 사용하여 프라이빗 키와 인증서를 안전하게 저장하도록 AWS IoT Greengrass 코어 소프트웨어를 구성하려면 단계에 따라 HSM의 프라이빗 키에서 인증서를 생성합니다. 그렇지 않으면 단계에 따라 AWS IoT 서비스에서 인증서와 프라이빗 키를 생성합니다. 하드웨어 보안 기능은 Linux 디바이스에서만 사용할 수 있습니다. 하드웨어 보안 및 이를 사용하는 데 필요한 요구 사항에 대한 자세한 내용은 [하드웨어 보안 통합](#) 섹션을 참조하세요.

AWS IoT 서비스에서 인증서 및 프라이빗 키 생성

사물 인증서를 생성하려면

1. AWS IoT 사물에 대한 인증서를 다운로드할 폴더를 생성합니다.

```
mkdir greengrass-v2-certs
```

2. AWS IoT 사물에 대한 인증서를 생성하고 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
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBIDELMakGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xZDASBgNVBAwTC01BTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXN0Q21sYWMyHhAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvbi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBIDELMakGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xZDASBgNVBAwTC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXN0Q21sYWMy
```



```

HzAdBgkqhkiG9w0BCQEWEG5vb251QGFtYXpva5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLyGVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BCQEFAAQCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEUuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTtwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUSTZeccyNCx2EXAMPLEEv9mQ0UXP6plfgxwKRX2fEXAMPLEda\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLE1nI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}

```

이후 인증서 구성에 사용할 인증서의 Amazon 리소스 이름(ARN)을 저장합니다.

HSM의 프라이빗 키에서 인증서 생성

Note

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

사물 인증서를 생성하려면

1. 코어 디바이스에서 HSM의 PKCS#11 토큰을 초기화하고, 프라이빗 키를 생성합니다. 프라이빗 키는 RSA-2048 키 크기(또는 그 이상)의 RSA 키나 ECC 키여야 합니다.

Note

ECC 키와 함께 하드웨어 보안 모듈을 사용하려면 [Greengrass nucleus](#) v2.5.6 이상을 사용해야 합니다.

하드웨어 보안 모듈 및 [보안 암호 관리자](#)를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

HSM의 설명서를 확인하여 토큰을 초기화하고 프라이빗 키를 생성하는 방법을 알아봅니다. HSM이 객체 ID를 지원하는 경우 프라이빗 키를 생성할 때 객체 ID를 지정합니다. 토큰을 초기화하고 프라이빗 키를 생성할 때 지정한 슬롯 ID, 사용자 PIN, 객체 레이블, 객체 ID(HSM에서 이를 사용하는 경우)를 저장합니다. 나중에 사물 인증서를 HSM으로 가져오고 AWS IoT Greengrass 코어 소프트웨어를 구성할 때 이러한 값을 사용합니다.

2. 프라이빗 키에서 인증서 서명 요청(CSR)을 생성합니다. 이 CSR을 AWS IoT 사용하여 HSM에서 생성한 프라이빗 키에 대한 사물 인증서를 생성합니다. 프라이빗 키에서 CSR을 생성하는 방법에 대한 자세한 내용은 HSM 설명서를 참조하세요. CSR은 `iotdevicekey.csr`과 같은 파일입니다.
3. 디바이스에서 개발 컴퓨터로 CSR을 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 `scp` 명령을 사용하여 CSR을 전송할 수 있습니다. **`device-ip-address`**를 디바이스의 IP 주소로 바꾸고, **`~/iotdevicekey.csr`**을 디바이스의 CSR 파일 경로로 바꿉니다.

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 개발 컴퓨터에서 AWS IoT 사물에 대한 인증서를 다운로드할 폴더를 생성합니다.

```
mkdir greengrass-v2-certs
```

5. CSR 파일을 사용하여 AWS IoT 사물에 대한 인증서를 생성하고 개발 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
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
```

```

"certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
"certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
"certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBIDELMAKGA1UEBhMCMVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBASTC0lBTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXRhbnQ2LsYWMxHZAAdBgkqhkiG9w0BCQEWEG5vb251QGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAKGA1UEBh
MCMVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBASTC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2LsYWMx
HZAAdBgkqhkiG9w0BCQEWEG5vb251QGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEibb30hjZnzcvcQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJlJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}

```

이후 인증서 구성에 사용할 인증서의 ARN을 저장합니다.

사물 인증서 구성

사물 인증서를 이전에 생성한 AWS IoT 사물에 연결하고 인증서에 정책을 추가하여 AWS IoT 코어 디바이스에 대한 AWS IoT 권한을 정의합니다.

사물의 인증서를 구성하려면

1. 인증서를 AWS IoT 사물에 연결합니다.

- *MyGreengrassCore*를 AWS IoT 사물의 이름으로 바꿉니다.
- 인증서 Amazon 리소스 이름(ARN)을 이전 단계에서 생성한 인증서의 ARN으로 바꿉니다.

```

aws iot attach-thing-principal --thing-name MyGreengrassCore
--principal arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4

```

요청에 성공하는 경우 명령에 출력이 없습니다.

2. Greengrass 코어 디바이스에 대한 AWS IoT 권한을 정의하는 AWS IoT 정책을 생성하고 연결합니다. 다음 정책은 모든 MQTT 주제 및 Greengrass 작업에 대한 액세스를 허용하므로 디바이스에서 사용자 지정 애플리케이션 및 새로운 Greengrass 작업이 필요한 향후 변경 사항과 호환됩니다. 사용 사례를 기반으로 이 정책을 제한할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.

이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 AWS IoT 정책을 연결할 수 있습니다.

해결 방법:

- a. Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-v2-iot-policy.json
```

다음 JSON을 파일로 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.

- *GreengrassV2IoTThingPolicy*를 생성할 정책의 이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-document file://greengrass-v2-iot-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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 사물의 인증서에 AWS IoT 정책을 연결합니다.

- *GreengrassV2IoTThingPolicy*를 연결할 정책의 이름으로 바꿉니다.
- 대상 ARN을 AWS IoT 사물 인증서의 ARN으로 바꿉니다.

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청에 성공하는 경우 명령에 출력이 없습니다.

토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라는 IAM 서비스 역할을 사용하여 AWS 서비스에 대한 호출을 승인합니다. 디바이스는 자격 AWS IoT 증명 공급자를 사용하여이 역할에 대한 임시 AWS 자격 증명을 가져오므로 디바이스가 상호 작용하고 AWS IoT, Amazon CloudWatch Logs로 로그를 전송하고, Amazon S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

AWS IoT Anrole 별칭을 사용하여 Greengrass 코어 디바이스에 대한 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 디바이스의 토큰 교환 역할을 변경할 수 있지만 디바이스 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스 직접 호출에 대한 권한 부여](#)를 참조하세요.

이 섹션에서는 토큰 교환 IAM 역할과 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다. 이미 Greengrass 코어 디바이스를 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 생성하려면

1. 디바이스가 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 해결 방법:
 - a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-trust-policy.json
```

다음 JSON을 파일로 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.

- *GreengrassV2TokenExchangeRole*을 생성할 IAM 역할의 이름으로 바꿉니다.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일로 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 S3에서 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오.

구성 요소 아티팩트에 대한 S3 버킷이 아직 없는 경우 버킷을 생성한 후 권한을 추가할 수 있습니다.

- d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreengrassV2TokenExchangeRoleAccess*를 생성할 IAM 정책의 이름으로 바꿉니다.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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.

- *GreengrassV2TokenExchangeRole*을 IAM 역할의 이름으로 바꿉니다.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 변경합니다.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

요청에 성공하는 경우 명령에 출력이 없습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

- *GreengrassCoreTokenExchangeRoleAlias*를 생성할 역할 별칭의 이름으로 바꿉니다.
- 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 변경합니다.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

역할 별칭을 생성하려면 토큰 교환 IAM 역할을 AWS IoT에 전달할 권한이 있어야 합니다. 역할 별칭을 생성하려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 권한이 있는지 확인합니다. 자세한 내용은 사용 AWS Identity and Access Management 설명서의 [AWS 서비스에 역할을 전달할 수 있는 사용자 권한 부여](#)를 참조하세요.

3. Greengrass 코어 디바이스가 역할 별칭을 사용하여 토큰 교환 역할을 수입하도록 허용하는 AWS IoT 정책을 생성하고 연결합니다. 이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 역할 별칭 AWS IoT 정책을 연결할 수 있습니다. 해결 방법:
 - a. (선택 사항) 역할 별칭에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-v2-iot-role-alias-policy.json
```

다음 JSON을 파일로 복사합니다.

- 리소스 ARN을 역할 별칭의 ARN으로 바꿉니다.

```
{
  "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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*를 생성할 AWS IoT 정책의 이름으로 바꿉니다.

```

aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json

```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```

{
  "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 사물의 인증서에 AWS IoT 정책을 연결합니다.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*를 역할 별칭 AWS IoT 정책의 이름으로 바꿉니다.
- 대상 ARN을 AWS IoT 사물 인증서의 ARN으로 바꿉니다.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청에 성공하는 경우 명령에 출력이 없습니다.

디바이스에 인증서 다운로드

앞서 디바이스의 인증서를 개발 컴퓨터에 다운로드했습니다. 이 섹션에서는 인증서를 코어 디바이스에 복사하여 AWS IoT에 연결하는 데 사용하는 인증서로 디바이스를 설정합니다. Amazon 루트 CA(인증 기관) 인증서도 다운로드합니다. HSM을 사용하는 경우 이 섹션의 HSM으로 인증서 파일도 가져옵니다.

- 이전에 AWS IoT 서비스에서 사물 인증서와 프라이빗 키를 생성한 경우 단계에 따라 프라이빗 키와 인증서 파일이 있는 인증서를 다운로드합니다.
- 먼저 HSM(하드웨어 보안 모듈)의 프라이빗 키에서 사물 인증서를 생성한 경우 단계에 따라 프라이빗 키와 HSM의 인증서가 포함된 인증서를 다운로드합니다.

프라이빗 키 및 인증서 파일이 포함된 인증서 다운로드

디바이스에 인증서를 다운로드하려면

1. 개발 컴퓨터에서 디바이스로 AWS IoT 사물 인증서를 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 scp 명령을 사용하여 인증서를 전송할 수 있습니다. *device-ip-address*를 디바이스의 IP 주소로 바꿉니다.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중에이 폴더에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: C:\greengrass\v2 또는 D:\greengrass\v2)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

Linux or Unix

- `/greengrass/v2`를 사용할 폴더로 바꿉니다.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 권한을 설정합니다.

- `/greengrass`를 루트 폴더의 상위로 바꿉니다.

```
sudo chmod 755 /greengrass
```

4. AWS IoT 사물 인증서를 Greengrass 루트 폴더에 복사합니다.

Linux or Unix

- `/greengrass/v2`를 Greengrass 루트 폴더로 바꿉니다.

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

PowerShell

- C:\greengrass\v2를 사용할 폴더로 바꿉니다.

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. Amazon 루트 인증 기관(CA) 인증서를 다운로드합니다. 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
```

HSM에서 프라이빗 키 및 인증서 파일이 포함된 인증서 다운로드

Note

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

디바이스에 인증서를 다운로드하려면

1. 개발 컴퓨터에서 디바이스로 AWS IoT 사물 인증서를 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 `scp` 명령을 사용하여 인증서를 전송할 수 있습니다. `device-ip-address`를 디바이스의 IP 주소로 바꿉니다.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중 에이 폴더에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: C:\greengrass\v2 또는 D:\greengrass\v2)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

Linux or Unix

- `/greengrass/v2`를 사용할 폴더로 바꿉니다.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 권한을 설정합니다.

- `/greengrass`를 루트 폴더의 상위로 바꿉니다.

```
sudo chmod 755 /greengrass
```

4. 사물 인증서 파일인 `~/greengrass-v2-certs/device.pem.crt`를 HSM으로 가져옵니다. 인증서를 가져오는 방법을 알아보려면 HSM 설명서를 확인하세요. 이전에 HSM에서 프라이빗 키를 생성한 것과 동일한 토큰, 슬롯 ID, 사용자 PIN, 객체 레이블 및 객체 ID(HSM에서 사용하는 경우)를 사용하여 인증서를 가져옵니다.

Note

앞서 객체 ID 없이 프라이빗 키를 생성했고, 인증서에 객체 ID가 있는 경우 프라이빗 키의 객체 ID를 인증서와 동일한 값으로 설정합니다. 프라이빗 키 객체의 객체 ID를 설정하는 방법을 알아보려면 HSM의 설명서를 확인하세요.

5. (선택 사항) HSM에만 존재하도록 사물 인증서 파일을 삭제합니다.

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. Amazon 루트 인증 기관(CA) 인증서를 다운로드합니다. 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 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

Windows 디바이스 설정

Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
 - a. Windows 키를 눌러 시작 메뉴를 엽니다.
 - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
 - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
 - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
 - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
 - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.


```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- wmic 명령이 더는 사용되지 않는 Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

Note

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

다음 위치에서 특정 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다
greengrass-nucleus-latest.zip.

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 계약](#)에 동의하는 것입니다.

2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

Note

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.

- a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6_10*을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6_10*을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.
 - i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

- ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. -verify 및 -certs 옵션과 함께 Jarsigner -verbose 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.
- 3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다.
*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후이 폴더를 제거할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자를 실행합니다.

- 이전에 생성한 AWS 리소스와 인증서를 사용하도록 지정하는 부분 구성 파일에서를 설치합니다. AWS IoT Greengrass 코어 소프트웨어는 디바이스에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 관리자는 제공된 부분 구성 파일에서 전체 구성 파일을 생성합니다.
- 코어 디바이스에서 소프트웨어 구성 요소를 실행하는 데 ggc_user 시스템 사용자가 사용되도록 지정합니다. Linux 디바이스에서 이 명령은 ggc_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

⚠ Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

i Note

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 nucleus 구성 요소의 jvmOptions 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

- 이전에 AWS IoT 서비스에서 사물 인증서와 프라이빗 키를 생성한 경우 단계에 따라 프라이빗 키와 인증서 파일이 있는 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.
- 이전에 하드웨어 보안 모듈(HSM)의 프라이빗 키에서 사물 인증서를 생성한 경우 단계에 따라 프라이빗 키와 인증서가 있는 AWS IoT Greengrass 코어 소프트웨어를 HSM에 설치합니다.

프라이빗 키 및 인증서 파일을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.

- **GreengrassInstaller**를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 텍스트 편집기를 사용하여 config.yaml 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```


다음 YAML 내용을 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터와 Greengrass nucleus 파라미터를 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: "2.16.1"
    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"
```

뒤이어 다음과 같이 하세요.

- `/greengrass/v2`의 각 인스턴스를 Greengrass 루트 폴더로 바꿉니다.
- `MyGreengrassCore`를 AWS IoT 사물의 이름으로 바꿉니다.
- `2.16.1`을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.
- `us-west-2`를 리소스를 생성한 AWS 리전 로 바꿉니다.
- `GreengrassCoreTokenExchangeRoleAlias`를 토큰 교환 역할 별칭의 이름으로 바꿉니다.
- `를` AWS IoT 데이터 엔드포인트 `iotDataEndpoint`로 바꿉니다.
- `를` AWS IoT 자격 증명 엔드포인트 `iotCredEndpoint`로 바꿉니다.

Note

이 구성 파일에서는 다음 예제와 같이 사용할 포트 및 네트워크 프록시와 같은 다른 nucleus 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성](#)을 참조하세요.

```

---
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: "2.16.1"
    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 관리자를 실행하고, `--init-config`를 지정하여 구성 파일을 제공합니다.

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.
- `GreengrassInstaller`의 각 인스턴스를 설치 관리자를 압축 해제한 폴더로 바꿉니다.

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 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

Note

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

HSM에서 프라이빗 키와 인증서를 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

Note

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.

- **GreengrassInstaller**를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. AWS IoT Greengrass 코어 소프트웨어가 HSM에서 프라이빗 키와 인증서를 사용하도록 하려면 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 [PKCS#11 공급자 구성 요소를](#) 설치합니다. PKCS#11 공급자 구성 요소는 설치 도중 구성할 수 있는 플러그인입니다. 다음 위치에서 PKCS#11 공급자 구성 요소의 최신 버전을 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>

PKCS#11 공급자 플러그인을 `aws.greengrass.crypto.Pkcs11Provider.jar`이라는 파일에 다운로드합니다. *GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

```
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 계약](#)에 동의하는 것입니다.

3. 텍스트 편집기를 사용하여 `config.yaml` 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 내용을 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터, Greengrass nucleus 파라미터 및 PKCS#11 공급자 파라미터를 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: "2.16.1"
    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"
```

뒤이어 다음과 같이 하세요.

- PKCS#11 URI의 각 *iotdevicekey* 인스턴스를 프라이빗 키를 생성하고 인증서를 가져온 객체 레이블로 바꿉니다.
- */greengrass/v2*의 각 인스턴스를 Greengrass 루트 폴더로 바꿉니다.
- *MyGreengrassCore*를 AWS IoT 사물의 이름으로 바꿉니다.
- *2.16.1*을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.
- *us-west-2*를 리소스를 생성한 AWS 리전 로 바꿉니다.
- *GreengrassCoreTokenExchangeRoleAlias*를 토큰 교환 역할 별칭의 이름으로 바꿉니다.
- 를 AWS IoT 데이터 엔드포인트 *iotDataEndpoint*로 바꿉니다.
- 를 AWS IoT 자격 증명 엔드포인트 *iotCredEndpoint*로 바꿉니다.
- *aws.greengrass.crypto.Pkcs11Provider* 구성 요소의 구성 파라미터를 코어 디바이스의 HSM 구성 값으로 바꿉니다.

Note

이 구성 파일에서는 다음 예제와 같이 사용할 포트 및 네트워크 프록시와 같은 다른 nucleus 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성](#)을 참조하세요.

```
---
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: "2.16.1"
```

```

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: "softsm_pkcs11"
      library: "/usr/local/Cellar/softsm/2.6.1/lib/softsm/libsoftsm2.so"
      slot: 1
      userPin: "1234"

```

4. 설치 관리자를 실행하고, `--init-config`를 지정하여 구성 파일을 제공합니다.

- `/greengrass/v2`를 Greengrass 루트 폴더로 바꿉니다.
- `GreengrassInstaller`의 각 인스턴스를 설치 관리자를 압축 해제한 폴더로 바꿉니다.

```

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 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

Note

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

Note

AWS IoT Greengrass 코어 소프트웨어가 클라우드에 연결되면 디바이스가 코어 디바이스로 인식됩니다.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass 참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

이 기능은 [Greengrass nucleus 구성 요소](#) v2.4.0 이상에서 사용할 수 있습니다.

AWS IoT 플릿 프로비저닝을 사용하면 처음 연결할 때 X.509 디바이스 인증서와 프라이빗 키를 생성하고 디바이스 AWS IoT에 안전하게 전송 AWS IoT 하도록을 구성할 수 있습니다. Amazon Root 인증 기관(CA)에서 서명한 클라이언트 인증서를 AWS IoT 제공합니다. 플릿 프로비저닝으로 프로비저닝하는 Greengrass 코어 디바이스에 대한 사물 그룹, 사물 유형 및 권한을 지정 AWS IoT 하도록을 구성할 수도 있습니다. 프로비저닝 템플릿을 정의하여가 AWS IoT 각 디바이스를 프로비저닝하는 방법을 정의합니다. 프로비저닝 템플릿은 프로비저닝 시 디바이스에 대해 생성할 사물, 정책 및 인증서 리소스를 지정합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [프로비저닝 템플릿](#)을 참조하세요.

AWS IoT Greengrass는 AWS IoT 플릿 프로비저닝으로 생성된 AWS 리소스를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 사용할 수 있는 AWS IoT 플릿 프로비저닝 플러그인을 제공합니다. 플릿 프로비저닝 플러그인은 클레임에 의한 프로비저닝을 사용합니다. 디바이스는 프로비저닝 클레임 인증서와 프라이빗 키를 사용하여 일반 작업에 사용할 수 있는 고유한 X.509 디바이스 인증서와 프라이빗 키를 획득합니다. 제조 중에 클레임 인증서와 프라이빗 키를 각 디바이스에 내장할 수 있으므로 고객은 나중에 온라인 상태가 되면 각 디바이스를 활성화할 수 있습니다. 여러 디바이스에 동일한 클레임 인증서와 프라이빗 키를 사용할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [클레임에 의한 프로비저닝](#)을 참조하세요.

Note

플릿 프로비저닝 플러그인은 현재 HSM(하드웨어 보안 모듈)에 프라이빗 키 및 인증서 파일을 저장하는 것을 지원하지 않습니다. HSM을 사용하려면 [수동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치합니다](#).

AWS IoT 플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 Greengrass 코어 디바이스를 프로비저닝 AWS 계정 하는 데 AWS IoT 사용하는 리소스에 설정해야 합니다. 이러한 리소스에는 프로비저닝 템플릿, 클레임 인증서 및 [토큰 교환 IAM 역할](#)이 포함됩니다. 이러한 리소스를 생성한 후 재사용하여 플릿에 여러 코어 디바이스를 프로비저닝할 수 있습니다. 자세한 내용은 [Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝 설정](#) 단원을 참조하십시오.

Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인합니다.

주제

- [사전 조건](#)
- [AWS IoT 엔드포인트 검색](#)
- [디바이스에 인증서 다운로드](#)
- [디바이스 환경 설정](#)
- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT 플릿 프로비저닝 플러그인 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝 설정](#)
- [AWS IoT 플릿 프로비저닝 플러그인 구성](#)
- [AWS IoT 플릿 프로비저닝 플러그인 변경 로그](#)

사전 조건

AWS IoT 플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 먼저 [Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝을 설정해야](#) 합니다. 이 단계를 한 번 완료한 후 플릿 프로비저닝을 사용하여 원하는 수의 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다.

AWS IoT 엔드포인트 검색

의 AWS IoT 엔드포인트를 가져 AWS 계정와 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. 해결 방법:

1. 의 AWS IoT 데이터 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 의 AWS IoT 자격 증명 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
}
```

디바이스에 인증서 다운로드

디바이스는 클레임 인증서와 프라이빗 키를 사용하여 AWS 리소스를 프로비저닝하고 X.509 디바이스 인증서를 획득하기 위한 요청을 인증합니다. 제조 중에 클레임 인증서와 프라이빗 키를 디바이스에 내장하거나 설치 중에 인증서와 키를 디바이스에 복사할 수 있습니다. 이 섹션에서는 클레임 인증서와 프라이빗 키를 디바이스에 복사합니다. 또한 Amazon 루트 CA(인증 기관) 인증서를 디바이스에 다운로드할 수도 있습니다.

Important

프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스 등 어디에서든 항상 보안 유지되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 클레임 인증서의 무단 사용과 같은 오용 징후를 모니터링하는 것이 좋습니다. 오용이 감지되면 프로비저닝 클레임 인증서를 디바이스 프로비저닝에 사용할 수 없도록 비활성화합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT모니터링](#)을 참조하세요.

디바이스 수와에 자신을 등록하는 디바이스를 더 잘 관리할 수 있도록 AWS 계정플릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. 사전 프로비저닝 후크

는 디바이스가 등록 중에 제공하는 템플릿 파라미터를 검증하는 AWS Lambda 함수입니다. 예를 들어 디바이스 ID를 데이터베이스와 비교하여 디바이스에 프로비저닝 권한이 있는지 확인하는 사전 프로비저닝 후크를 생성할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사전 프로비저닝 후크](#)를 참조하세요.

디바이스에 클레임 인증서 다운로드

1. 클레임 인증서와 프라이빗 키를 디바이스에 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 scp 명령을 사용하여 클레임 인증서와 프라이빗 키를 전송할 수 있습니다. 다음 예제 명령은 개발 컴퓨터의 claim-certs 폴더에서 이러한 파일을 디바이스로 전송합니다. *device-ip-address*를 디바이스의 IP 주소로 바꿉니다.

```
scp -r claim-certs/ device-ip-address:~
```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중 에이 폴더에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: C:\greengrass\v2 또는 D:\greengrass\v2)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

Linux or Unix

- */greengrass/v2*를 사용할 폴더로 바꿉니다.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *C:\greengrass\v2*를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 권한을 설정합니다.

- `/greengrass`를 루트 폴더의 상위로 바꿉니다.

```
sudo chmod 755 /greengrass
```

4. 클레임 인증서를 Greengrass 루트 폴더로 옮깁니다.

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.

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 다운로드합니다. 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 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 `--component-default-user` 설치 관리자가 사용자 및 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 `root`)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 `sudo` 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

Windows 디바이스 설정

Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
 - a. Windows 키를 눌러 시작 메뉴를 엽니다.
 - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
 - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
 - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
 - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
 - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.


```
C:\Program Files\Amazon Corretto\jdk11.0.13_8\bin
```
 - g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.


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

Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- wmic 명령이 더는 사용되지 않는 Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

Note

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential

Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

다음 위치에서 특정 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다greengrass-nucleus-latest.zip.

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 계약](#)에 동의하는 것입니다.

2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

Note

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.

a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. `jdk17.0.6_10`을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. `jdk17.0.6_10`을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.

i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. `-verify` 및 `-certs` 옵션과 함께 `Jarsigner -verbose` 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.
3. AWS IoT Greengrass 코어 소프트웨어의 압축을 디바이스의 폴더에 풋습니다.
*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./ GreengrassInstaller/lib/Greengrass.jar --version
```

Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후이 폴더를 제거할 수 있습니다.

AWS IoT 플릿 프로비저닝 플러그인 다운로드

다음 위치에서 최신 버전의 AWS IoT 플릿 프로비저닝 플러그인을 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar>

Note

다음 위치에서 특정 버전의 AWS IoT 플릿 프로비저닝 플러그인을 다운로드할 수 있습니다. **version**을 다운로드할 버전으로 바꿉니다. 플릿 프로비저닝 플러그인의 각 버전에 대한 자세한 내용은 [AWS IoT 플릿 프로비저닝 플러그인 변경 로그](#) 섹션을 참조하세요.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

플릿 프로비저닝 플러그인은 오픈 소스입니다. 소스 코드를 보려면 GitHub의 [AWS IoT 플릿 프로비저닝 플러그인](#)을 참조하세요.

AWS IoT 플릿 프로비저닝 플러그인을 다운로드하려면

- 디바이스에서 AWS IoT 플릿 프로비저닝 플러그인을 라는 파일에 다운로드합니다. `aws.greengrass.FleetProvisioningByClaim.jar`. **GreengrassInstaller**를 사용하여 폴더로 바꿉니다.

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 계약](#)에 동의하는 것입니다.

AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자를 실행합니다.

- 플릿 프로비저닝 플러그인을 사용하여 AWS 리소스를 프로비저닝하도록 지정하는 부분 구성 파일에서 설치를 실행합니다. AWS IoT Greengrass 코어 소프트웨어는 디바이스에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 관리자는 제공된 부분 구성 파일에서 전체 구성 파일과 플릿 프로비저닝 플러그인이 생성하는 AWS 리소스를 생성합니다.
- ggc_user 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 ggc_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

Note

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해

nucleus 구성 요소의 `jvmOptions` 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.
 - *GreengrassInstaller*를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 텍스트 편집기를 사용하여 `config.yaml` 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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 [AWS IoT 플릿 프로비저닝 플러그인 구성](#) 섹션을 참조하세요.

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  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: "2.16.1"
  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"

```

뒤이어 다음과 같이 하세요.

- *2.16.1*을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.
- */greengrass/v2*의 각 인스턴스 또는 *C:\\greengrass\\v2*를 Greengrass 루트 폴더로 바꿉니다.

Note

Windows 디바이스에서는 경로 구분자를 이중 백슬래시(\\)로 지정해야 합니다(예: C:\\greengrass\\v2).

- *us-west-2*를 프로비저닝 템플릿 및 기타 리소스를 생성한 AWS 리전으로 바꿉니다.
- 를 AWS IoT 데이터 엔드포인트 *iotDataEndpoint*로 바꿉니다.
- 를 AWS IoT 자격 증명 엔드포인트 *iotCredentialEndpoint*로 바꿉니다.
- *GreengrassCoreTokenExchangeRoleAlias*를 토큰 교환 역할 별칭의 이름으로 바꿉니다.
- *GreengrassFleetProvisioningTemplate*을 플릿 프로비저닝 템플릿의 이름으로 바꿉니다.
- *claimCertificatePath*를 디바이스의 클레임 인증서 경로로 바꿉니다.
- *claimCertificatePrivateKeyPath*를 디바이스의 클레임 인증서 프라이빗 키 경로로 바꿉니다.
- 템플릿 파라미터(*templateParameters*)를 디바이스 프로비저닝에 사용하는 값으로 바꿉니다. 이 예제는 *ThingName* 및 *ThingGroupName* 파라미터를 정의하는 [예제 템플릿](#)을 참조합니다.

Note

이 구성 파일에서는 다음 예제와 같이 사용할 포트 및 네트워크 프록시 등 다른 구성 옵션을 사용자 지정할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성](#)을 참조하세요.

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    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: "2.16.1"
    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 예제와 같이 system 아래에서 rootCaPath를 추가로 지정해야 합니다.

Linux or Unix

```

---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...

```

Windows

```

---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...

```

- 설치 관리자를 실행합니다. --trusted-plugin을 지정하여 플릿 프로비저닝 플러그인을 제공하고, --init-config를 지정하여 구성 파일을 제공합니다.
 - /greengrass/v2를 Greengrass 루트 폴더로 바꿉니다.
 - GreengrassInstaller의 각 인스턴스를 설치 관리자를 압축 해제한 폴더로 바꿉니다.

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 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다.

니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

Note

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass 참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝 설정

[플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치](#)하려면 먼저에서 다음 리소스를 설정해야 합니다 AWS 계정. 이러한 리소스를 통해 디바이스에는 등록 AWS IoT 하고 Greengrass 코어 디바이스로 작동할 수 있습니다. 이 섹션의 단계를 한 번 수행하여 AWS 계정에서 이러한 리소스를 생성 및 구성합니다.

- 코어 디바이스가 AWS 서비스에 대한 직접 호출의 승인에 사용하는 토큰 교환 IAM 역할입니다.
- 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭입니다.
- (선택 사항) 코어 디바이스가 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 호출을 승인하는 데 사용하는 AWS IoT 정책입니다. 이 AWS IoT 정책은 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭에 대한 `iot:AssumeRoleWithCertificate` 권한을 허용해야 합니다.

플릿의 모든 코어 디바이스에 단일 AWS IoT 정책을 사용하거나 각 코어 디바이스에 대한 AWS IoT 정책을 생성하도록 플릿 프로비저닝 템플릿을 구성할 수 있습니다.

- AWS IoT 플릿 프로비저닝 템플릿입니다. 이 템플릿은 다음을 지정해야 합니다.
 - AWS IoT 사물 리소스입니다. 기존 사물 그룹 목록을 지정하여 온라인 상태일 때 각 디바이스에 구성 요소를 배포할 수 있습니다.
 - AWS IoT 정책 리소스입니다. 이 리소스는 다음 속성 중 하나를 정의할 수 있습니다.
 - 기존 AWS IoT 정책의 이름입니다. 이 옵션을 선택하면이 템플릿에서 생성한 코어 디바이스가 동일한 AWS IoT 정책을 사용하며 해당 권한을 플릿으로 관리할 수 있습니다.
 - AWS IoT 정책 문서. 이 옵션을 선택하면이 템플릿에서 생성하는 각 코어 디바이스는 고유한 AWS IoT 정책을 사용하며 각 개별 코어 디바이스에 대한 권한을 관리할 수 있습니다.
 - AWS IoT 인증서 리소스입니다. 이 인증서 리소스는 `AWS::IoT::Certificate::Id` 파라미터를 사용하여 코어 디바이스에 인증서를 연결해야 합니다. 자세한 내용은 AWS IoT 개발자 안내서의 [JIT\(Just-in-Time\) 프로비저닝](#)을 참조하세요.
- 플릿 AWS IoT 프로비저닝 템플릿에 대한 프로비저닝 클레임 인증서 및 프라이빗 키입니다. 제조 중에 디바이스에 이 인증서와 프라이빗 키를 내장할 수 있으므로 디바이스가 온라인 상태가 되면 등록하고 프로비저닝할 수 있습니다.

Important

프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스 등 어디에서든 항상 보안 유지되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 클레임 인증서의 무단 사용과 같은 오용 징후를 모니터링하는 것이 좋습니다. 오용이 감지되

면 프로비저닝 클레임 인증서를 디바이스 프로비저닝에 사용할 수 없도록 비활성화합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT모니터링](#)를 참조하세요.

디바이스 수와에 자신을 등록하는 디바이스를 더 잘 관리할 수 있도록 AWS 계정폴릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. 사전 프로비저닝 후크는 디바이스가 등록 중에 제공하는 템플릿 파라미터를 검증하는 AWS Lambda 함수입니다. 예를 들어 디바이스 ID를 데이터베이스와 비교하여 디바이스에 프로비저닝 권한이 있는지 확인하는 사전 프로비저닝 후크를 생성할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사전 프로비저닝 후크](#)를 참조하세요.

- 디바이스가 폴릿 프로비저닝 템플릿을 등록하고 사용할 수 있도록 프로비저닝 클레임 인증서에 연결하는 AWS IoT 정책입니다.

주제

- [토큰 교환 역할 생성](#)
- [AWS IoT 정책 생성](#)
- [폴릿 프로비저닝 템플릿 생성](#)
- [프로비저닝 클레임 인증서 및 프라이빗 키 생성](#)

토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라는 IAM 서비스 역할을 사용하여 AWS 서비스에 대한 호출을 승인합니다. 디바이스는 자격 AWS IoT 증명 공급자를 사용하여이 역할에 대한 임시 AWS 자격 증명을 가져오므로 디바이스가 상호 작용하고 AWS IoT, Amazon CloudWatch Logs로 로그를 전송하고, Amazon S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

AWS IoT Anrole 별칭을 사용하여 Greengrass 코어 디바이스에 대한 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 디바이스의 토큰 교환 역할을 변경할 수 있지만 디바이스 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스 직접 호출에 대한 권한 부여](#)를 참조하세요.

이 섹션에서는 토큰 교환 IAM 역할과 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다. 이미 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 토큰 교환 역할 및 역할 별칭을 사용할 수 있습니다.

토큰 교환 IAM 역할을 생성하려면

1. 디바이스가 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 해결 방법:

- a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-trust-policy.json
```

다음 JSON을 파일로 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.

- *GreengrassV2TokenExchangeRole*을 생성할 IAM 역할의 이름으로 바꿉니다.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file:///device-role-trust-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일로 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 S3에서 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오.

구성 요소 아티팩트에 대한 S3 버킷이 아직 없는 경우 버킷을 생성한 후 권한을 추가할 수 있습니다.

d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreengrassV2TokenExchangeRoleAccess*를 생성할 IAM 정책의 이름으로 바꿉니다.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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.

- *GreengrassV2TokenExchangeRole*을 IAM 역할의 이름으로 바꿉니다.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 변경합니다.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

요청에 성공하는 경우 명령에 출력이 없습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

- *GreengrassCoreTokenExchangeRoleAlias*를 생성할 역할 별칭의 이름으로 바꿉니다.
- 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 변경합니다.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

역할 별칭을 생성하려면 토큰 교환 IAM 역할을 AWS IoT에 전달할 권한이 있어야 합니다. 역할 별칭을 생성하려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 권한이 있는지 확인합니다. 자세한 내용은 사용 AWS Identity and Access Management 설명서의 [AWS 서비스에 역할을 전달할 수 있는 사용자 권한 부여](#)를 참조하세요.

AWS IoT 정책 생성

디바이스를 AWS IoT 사물로 등록한 후 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다. 이 인증서에는 디바이스가 인증서와 함께 사용할 수 있는 권한을 정의하는 AWS IoT 정책이 하나 이상 포함되어 있습니다. 이러한 정책은 디바이스가 AWS IoT 및와 통신할 수 있도록 허용합니다. AWS IoT Greengrass.

AWS IoT 플릿 프로비저닝을 통해 디바이스는 AWS IoT에 연결하여 디바이스 인증서를 생성하고 다운로드합니다. 다음 섹션에서 생성하는 플릿 프로비저닝 템플릿에서가 모든 디바이스의 인증서에 동일한 AWS IoT 정책을 AWS IoT 연결할지 아니면 각 디바이스에 대해 새 정책을 생성할지 지정할 수 있습니다.

이 섹션에서는가 모든 디바이스의 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 정책을 참조할 수 있습니다.

AWS IoT 정책을 생성하려면

- Greengrass 코어 디바이스 플릿에 대한 AWS IoT 권한을 정의하는 AWS IoT 정책을 생성합니다. 다음 정책은 모든 MQTT 주제 및 Greengrass 작업에 대한 액세스를 허용하므로 디바이스에서 사용자 지정 애플리케이션 및 새로운 Greengrass 작업이 필요한 향후 변경 사항과 호환됩니다. 또한 이 정책은 디바이스가 이전 섹션에서 생성한 토큰 교환 역할을 사용하도록 허용하는 `iot:AssumeRoleWithCertificate` 권한을 허용합니다. 사용 사례를 기반으로 이 정책을 제한할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.

해결 방법:

- Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-v2-iot-policy.json
```

다음 JSON을 파일로 복사합니다.

- `iot:AssumeRoleWithCertificate` 리소스를 이전 섹션에서 생성한 AWS IoT 역할 별칭의 ARN으로 바꿉니다.

JSON

```
{
  "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-east-1:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
  }
]
}

```

b. AWS IoT 정책 문서에서 정책을 생성합니다.

- *GreengrassV2IoTThingPolicy*를 생성할 정책의 이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

JSON

```

{
  "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 템플릿 생성

AWS IoT 플릿 프로비저닝 템플릿은 AWS IoT 사물, 정책 및 인증서를 프로비저닝하는 방법을 정의합니다. 플릿 프로비저닝 플러그인으로 Greengrass 코어 디바이스를 프로비저닝하려면 다음을 지정하는 템플릿을 생성해야 합니다.

- AWS IoT 사물 리소스입니다. 기존 사물 그룹 목록을 지정하여 온라인 상태일 때 각 디바이스에 구성 요소를 배포할 수 있습니다.
- AWS IoT 정책 리소스입니다. 이 리소스는 다음 속성 중 하나를 정의할 수 있습니다.
 - 기존 AWS IoT 정책의 이름입니다. 이 옵션을 선택하면이 템플릿에서 생성한 코어 디바이스가 동일한 AWS IoT 정책을 사용하며 해당 권한을 플릿으로 관리할 수 있습니다.
 - AWS IoT 정책 문서. 이 옵션을 선택하면이 템플릿에서 생성하는 각 코어 디바이스는 고유한 AWS IoT 정책을 사용하며 각 개별 코어 디바이스에 대한 권한을 관리할 수 있습니다.
- AWS IoT 인증서 리소스입니다. 이 인증서 리소스는 `AWS::IoT::Certificate::Id` 파라미터를 사용하여 코어 디바이스에 인증서를 연결해야 합니다. 자세한 내용은 AWS IoT 개발자 안내서의 [JIT\(Just-in-Time\) 프로비저닝](#)을 참조하세요.

템플릿에서 지정하여 기존 AWS IoT 사물 그룹 목록에 사물을 추가할 수 있습니다. 코어 디바이스가 AWS IoT Greengrass 에 처음 연결되면 코어 디바이스가 멤버인 각 사물 그룹에 대해 Greengrass 배포를 수신합니다. 사물 그룹을 사용하여 온라인 상태가 되는 즉시 각 디바이스에 최신 소프트웨어를 배포할 수 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

이 AWS IoT 서비스에는 디바이스를 프로비저닝할 AWS 계정 때에서 AWS IoT 리소스를 생성하고 업데이트할 수 있는 권한이 필요합니다. AWS IoT 서비스에 액세스 권한을 부여하려면 템플릿을 생성할 때 IAM 역할을 생성하고 제공합니다. 디바이스를 프로비저닝할 때에서 사용할 AWS IoT 수 있는 모든 권한에 대한 액세스를 허용하는 관리형 정책인 [AWSIoTThingsRegistration](#)을 AWS IoT 제공합니다. 이 관리형 정책을 사용하거나 사용 사례에 대한 관리형 정책의 권한을 범위로 좁히는 사용자 지정 정책을 생성할 수 있습니다.

이 섹션에서는 디바이스에 대한 리소스를 AWS IoT 프로비저닝하도록 허용하는 IAM 역할을 생성하고 해당 IAM 역할을 사용하는 플릿 프로비저닝 템플릿을 생성합니다.

플릿 프로비저닝 템플릿 생성

1. 에서 리소스를 프로비저닝하기 위해가 말 AWS IoT 을 수 있는 IAM 역할을 생성합니다 AWS 계정. 해결 방법:

- a. 가 역할을 수입 AWS IoT 하도록 허용하는 신뢰 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano aws-iot-trust-policy.json
```

다음 JSON을 파일로 복사합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
    },
  ],
}
```

```

        "Action": "sts:AssumeRole"
    }
]
}

```

b. 신뢰 정책 문서를 사용하여 IAM 역할을 생성합니다.

- *GreengrassFleetProvisioningRole*을 생성할 IAM 역할의 이름으로 바꿉니다.

```
aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-policy-document file://aws-iot-trust-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": "*"
    }
  ]
}

```

- c. 디바이스를 프로비저닝할 때에서 사용할 AWS IoT 수 있는 모든 권한에 대한 액세스를 허용하는 [AWSIoTThingsRegistration](#) 정책을 검토합니다. 이 관리형 정책을 사용하거나 사용 사례에 대해 범위가 축소된 권한을 정의하는 사용자 지정 정책을 생성할 수 있습니다. 사용자 지정 정책을 생성하려는 경우 지금 생성합니다.
- d. 플릿 프로비저닝 역할에 IAM 정책을 연결합니다.

- **GreengrassFleetProvisioningRole**을 IAM 역할의 이름으로 바꿉니다.
- 이전 단계에서 사용자 지정 정책을 생성한 경우 정책 ARN을 사용할 IAM 정책의 ARN으로 바꿉니다.

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

요청에 성공하는 경우 명령에 출력이 없습니다.

2. (선택 사항) 디바이스가 등록 중에 제공하는 템플릿 파라미터를 검증하는 AWS Lambda 함수인 사전 프로비저닝 후크를 생성합니다. 사전 프로비저닝 후크를 사용하여 AWS 계정에 온보딩된 디바이스와 그 수를 더 효과적으로 제어할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사전 프로비저닝 후크](#)를 참조하세요.
3. 플릿 프로비저닝 템플릿을 생성합니다. 해결 방법:
 - a. 프로비저닝 템플릿 문서를 포함하는 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-fleet-provisioning-template.json
```

프로비저닝 템플릿 문서를 작성합니다. 다음 속성으로 AWS IoT 사물을 생성하도록 지정하는 다음 예제 프로비저닝 템플릿부터 시작할 수 있습니다.

- 사물의 이름은 ThingName 템플릿 파라미터에서 지정한 값입니다.
- 사물은 ThingGroupName 템플릿 파라미터에서 지정한 사물 그룹의 멤버입니다. 사물 그룹은에 있어야 합니다 AWS 계정.
- 사물의 인증서에는 라는 AWS IoT 정책이 GreengrassV2IoTThingPolicy 연결되어 있습니다.

자세한 내용은 AWS IoT Core 개발자 안내서의 [프로비저닝 템플릿](#)을 참조하세요.

```
{
  "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

MyThing, *MyPolicy* 및 *MyCertificate*는 플릿 프로비저닝 템플릿의 각 리소스 사양을 식별하는 임의의 이름입니다. 템플릿에서 생성하는 리소스에는 이러한 이름을 사용하지 AWS IoT 않습니다. 이러한 이름을 그대로 사용하거나 템플릿의 각 리소스를 식별하는 데 도움이 되는 값으로 바꿀 수 있습니다.

- b. 프로비저닝 템플릿 문서에서 플릿 프로비저닝 템플릿을 생성합니다.
- *GreengrassFleetProvisioningTemplate*을 생성할 템플릿의 이름으로 바꿉니다.
 - 템플릿 설명을 템플릿에 대한 설명으로 바꿉니다.
 - 프로비저닝 역할 ARN을 이전에 생성한 역할의 ARN으로 바꿉니다.

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 --pre-provisioning-hook 인수로 사전 프로비저닝 후크의 Lambda 함수의 ARN을 지정합니다.

```
--pre-provisioning-hook targetArn=arn:aws:lambda:us-
west-2:123456789012:function:GreengrassPreProvisioningHook
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
  "templateName": "GreengrassFleetProvisioningTemplate",
  "defaultVersionId": 1
}
```

프로비저닝 클레임 인증서 및 프라이빗 키 생성

클레임 인증서는 디바이스가 AWS IoT 사물로 등록하고 일반 작업에 사용할 고유한 X.509 디바이스 인증서를 검색할 수 있도록 허용하는 X.509 인증서입니다. 클레임 인증서를 생성한 후 디바이스가 이를 사용하여 고유한 디바이스 인증서를 생성하고 플릿 프로비저닝 템플릿으로 프로비저닝할 수 있도록 허용하는 AWS IoT 정책을 연결합니다. 클레임 인증서가 있는 디바이스는 AWS IoT 정책에서 허용하는 프로비저닝 템플릿만 사용하여 프로비저닝할 수 있습니다.

이 섹션에서는 클레임 인증서를 생성하고 디바이스가 이전 섹션에서 생성한 플릿 프로비저닝 템플릿을 사용하도록 구성합니다.

⚠ Important

프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스 등 어디에서든 항상 보안 유지되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 클레임 인증서의 무단 사용과 같은 오용 징후를 모니터링하는 것이 좋습니다. 오용이 감지되면 프로비저닝 클레임 인증서를 디바이스 프로비저닝에 사용할 수 없도록 비활성화합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT모니터링](#)를 참조하세요.

디바이스 수와에 자신을 등록하는 디바이스를 더 잘 관리할 수 있도록 AWS 계정폴릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. 사전 프로비저닝 후크는 디바이스가 등록 중에 제공하는 템플릿 파라미터를 검증하는 AWS Lambda 함수입니다. 예를 들어 디바이스 ID를 데이터베이스와 비교하여 디바이스에 프로비저닝 권한이 있는지 확인하는 사전 프로비저닝 후크를 생성할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사전 프로비저닝 후크](#)를 참조하세요.

프로비저닝 클레임 인증서 및 프라이빗 키 생성

1. 클레임 인증서와 프라이빗 키를 다운로드할 폴더를 생성합니다.

```
mkdir claim-certs
```

2. 프로비저닝에 사용할 인증서와 프라이빗 키를 생성하고 저장합니다.는 Amazon Root 인증 기관 (CA)에서 서명한 클라이언트 인증서를 AWS IoT 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
```

요청에 성공하는 경우 응답에 인증서에 대한 정보가 포함됩니다. 나중에 사용하도록 인증서의 ARN을 저장합니다.

3. 디바이스가 인증서를 사용하여 고유한 디바이스 인증서를 생성하고 플릿 프로비저닝 템플릿으로 프로비저닝할 수 있도록 허용하는 AWS IoT 정책을 생성하고 연결합니다. 다음 정책은 디바이스 프로비저닝 MQTT API에 대한 액세스를 허용합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [디바이스 프로비저닝 MQTT API](#)를 참조하세요.

해결 방법:

- a. Greengrass 코어 디바이스에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-provisioning-claim-iot-policy.json
```

다음 JSON을 파일로 복사합니다.

- **##**의 각 인스턴스를 플릿 프로비저닝을 설정한 AWS 리전 로 바꿉니다.
- **account-id**를 각각 AWS 계정 ID로 바꿉니다.
- **GreengrassFleetProvisioningTemplate**를 각각 이전 섹션에서 생성한 플릿 프로비저닝 템플릿의 이름으로 바꿉니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

{
  "Effect": "Allow",
  "Action": "iot:Connect",
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "iot:Publish",
    "iot:Receive"
  ],
  "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/create/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-templates/GreengrassFleetProvisioningTemplate/provision/*"
  ]
},
{
  "Effect": "Allow",
  "Action": "iot:Subscribe",
  "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/create/*",
    "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-templates/GreengrassFleetProvisioningTemplate/provision/*"
  ]
}
]
}

```

b. AWS IoT 정책 문서에서 정책을 생성합니다.

- *GreengrassProvisioningClaimPolicy*를 생성할 정책의 이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-document file://greengrass-provisioning-claim-iot-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

JSON

```
{
  "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:us-east-1:123456789012:topic/$aws/certificates/create/*\",
          \"arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
          \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/create/*\",
          \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }\",
  "policyVersionId": "1"
}
```


4. AWS IoT 정책을 프로비저닝 클레임 인증서에 연결합니다.

- *GreengrassProvisioningClaimPolicy*를 연결할 정책의 이름으로 바꿉니다.
- 대상 ARN을 프로비저닝 클레임 인증서의 ARN으로 바꿉니다.

```
aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청에 성공하는 경우 명령에 출력이 없습니다.

이제 디바이스에 등록하고 Greengrass 코어 디바이스로 자신을 프로비저닝하는 데 사용할 수 있는 프로비저닝 클레임 인증서 AWS IoT 와 프라이빗 키가 생겼습니다. 제조 중에 디바이스에 클레임 인증서와 프라이빗 키를 임베드하거나 AWS IoT Greengrass 코어 소프트웨어를 설치하기 전에 인증서와 키를 디바이스에 복사할 수 있습니다. 자세한 내용은 [AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

AWS IoT 플릿 프로비저닝 플러그인 구성

AWS IoT 플릿 프로비저닝 플러그인은 [플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치할](#) 때 사용자 지정할 수 있는 다음과 같은 구성 파라미터를 제공합니다.

rootPath

AWS IoT Greengrass 코어 소프트웨어의 루트로 사용할 폴더의 경로입니다.

awsRegion

플릿 프로비저닝 플러그인 AWS 리전 이 리소스를 프로비저닝 AWS 하는 데 사용하는 입니다.

iotDataEndpoint

의 AWS IoT 데이터 엔드포인트입니다 AWS 계정.

iotCredentialEndpoint

의 AWS IoT 자격 증명 엔드포인트입니다 AWS 계정.

iotRoleAlias

토큰 교환 IAM AWS IoT 역할을 가리키는 역할 별칭입니다. 자격 AWS IoT 증명 공급자는 Greengrass 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록이 역할을 맡습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

provisioningTemplate

AWS 리소스를 프로비저닝하는 데 사용할 AWS IoT 플릿 프로비저닝 템플릿입니다. 이 템플릿은 다음을 지정해야 합니다.

- AWS IoT 사물 리소스입니다. 기존 사물 그룹 목록을 지정하여 온라인 상태일 때 각 디바이스에 구성 요소를 배포할 수 있습니다.
- AWS IoT 정책 리소스입니다. 이 리소스는 다음 속성 중 하나를 정의할 수 있습니다.
 - 기존 AWS IoT 정책의 이름입니다. 이 옵션을 선택하면이 템플릿에서 생성한 코어 디바이스가 동일한 AWS IoT 정책을 사용하며 해당 권한을 플릿으로 관리할 수 있습니다.
 - AWS IoT 정책 문서. 이 옵션을 선택하면이 템플릿에서 생성하는 각 코어 디바이스는 고유한 AWS IoT 정책을 사용하며 각 개별 코어 디바이스에 대한 권한을 관리할 수 있습니다.
- AWS IoT 인증서 리소스입니다. 이 인증서 리소스는 `AWS::IoT::Certificate::Id` 파라미터를 사용하여 코어 디바이스에 인증서를 연결해야 합니다. 자세한 내용은 AWS IoT 개발자 안내서의 [JIT\(Just-in-Time\) 프로비저닝](#)을 참조하세요.

자세한 내용은 AWS IoT Core 개발자 안내서의 [프로비저닝 템플릿](#)을 참조하세요.

claimCertificatePath

provisioningTemplate에서 지정한 프로비저닝 템플릿에 대한 프로비저닝 클레임 인증서의 경로입니다. 자세한 정보는 AWS IoT Core API 참조의 [CreateProvisioningClaim](#)을 참조하세요.

claimCertificatePrivateKeyPath

provisioningTemplate에서 지정한 프로비저닝 템플릿에 대한 프로비저닝 클레임 인증서 프라이빗 키의 경로입니다. 자세한 정보는 AWS IoT Core API 참조의 [CreateProvisioningClaim](#)을 참조하세요.

Important

프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스 등 어디에서든 항상 보안 유지되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 클레임 인증서의 무단 사용과 같은 오용 징후를 모니터링하는 것이 좋습니다. 오용이

감지되면 프로비저닝 클레임 인증서를 디바이스 프로비저닝에 사용할 수 없도록 비활성화합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT모니터링](#)를 참조하세요. 디바이스 수와에 자신을 등록하는 디바이스를 더 잘 관리할 수 있도록 AWS 계정플릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. 사전 프로비저닝 후크는 디바이스가 등록 중에 제공하는 템플릿 파라미터를 검증하는 AWS Lambda 함수입니다. 예를 들어 디바이스 ID를 데이터베이스와 비교하여 디바이스에 프로비저닝 권한이 있는지 확인하는 사전 프로비저닝 후크를 생성할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사전 프로비저닝 후크](#)를 참조하세요.

rootCaPath

Amazon 루트 CA(인증 기관) 인증서의 경로입니다.

templateParameters

(선택 사항) 플릿 프로비저닝 템플릿에 제공할 파라미터의 맵입니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [프로비저닝 템플릿 파라미터 섹션](#)을 참조하세요.

deviceId

(선택 사항) 플릿 프로비저닝 플러그인이 AWS IoT에 대한 MQTT 연결을 생성할 때 클라이언트 ID로 사용할 디바이스 식별자입니다.

기본값: 무작위 UUID.

mqttPort

(선택 사항) MQTT 연결에 사용할 포트입니다.

기본값: 8883

proxyUrl

(선택 사항) scheme://userinfo@host:port 형식으로 표시되는 프록시 서버의 URL입니다. HTTPS 프록시를 사용하려면 플릿 프로비저닝 플러그인 버전 1.1.0 이상을 사용해야 합니다.

- scheme - 체계로, http 또는 https여야 합니다.

Important

Greengrass 코어 디바이스에서 HTTPS 프록시가 사용되려면 [Greengrass nucleus v2.5.0](#) 이상을 실행해야 합니다.

HTTPS 프록시를 구성하는 경우 코어 디바이스의 Amazon 루트 CA 인증서에 프록시 서버 CA 인증서를 추가해야 합니다. 자세한 내용은 [HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정](#) 단원을 참조하십시오.

- `userinfo` - (선택 사항) 사용자 이름 및 암호 정보입니다. `url`에서 이 정보를 지정하면 Greengrass 코어 디바이스에서는 `username` 및 `password` 필드를 무시합니다.
- `host` - 프록시 서버의 호스트 이름 또는 IP 주소입니다.
- `port` - (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 디바이스에서는 다음 기본값이 사용됩니다.
 - `http` - 80
 - `https` - 443

`proxyUserName`

(선택 사항) 프록시 서버를 인증하는 사용자 이름입니다.

`proxyPassword`

(선택 사항) 프록시 서버를 인증하는 사용자 이름입니다.

`csrPath`

(선택 사항) CSR(인증서 서명 요청)에서 디바이스 인증서를 생성하는 데 사용할 CSR 파일의 경로입니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [클레임에 의한 프로비저닝](#)을 참조하세요.

`csrPrivateKeyPath`

(선택 사항, `csrPath`가 선언된 경우 필수 사항) CSR을 생성하는 데 사용되는 프라이빗 키의 경로입니다. 프라이빗 키를 사용하여 CSR을 생성한 상태여야 합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [클레임에 의한 프로비저닝](#)을 참조하세요.

`certificatePath`

(선택 사항) 다운로드한 디바이스 인증서를 저장할 경로입니다.

`privateKeyPath`

(선택 사항) 다운로드한 디바이스 프라이빗 키를 저장할 경로입니다.

AWS IoT 플릿 프로비저닝 플러그인 변경 로그

다음 표에서는 클레임 플러그인(`aws.greengrass.FleetProvisioningByClaim`)을 기준으로 AWS IoT 플릿 프로비저닝의 각 버전에 대한 변경 사항을 설명합니다.

버전	변경 사항
1.2.2	버그 수정 및 개선 사항 사용자 지정 코어 디바이스 인증서(certificatePath) 및 프라이빗 키(privateKeyPath) 경로에 대한 지원이 추가되었습니다.
1.2.1	버그 수정 및 개선 사항 Greengrass nucleus 시작 중에 플릿 프로비저닝 플러그인이 오프라인 상태가 되는 문제를 해결합니다. 플릿 프로비저닝 플러그인이 이제 MQTT 연결 직접 호출을 무기한 재시도합니다.
1.2.0	버그 수정 및 개선 사항 - 구성 가능한 프라이빗 키 경로를 사용하는 인증서 서명 요청을 통한 디바이스 프로비저닝이 지원됩니다. - 사소한 수정 및 개선 사항
1.1.0	버그 수정 및 개선 사항 - Windows 디바이스에서 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 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정을(를) 참조하세요.
1.0.0	초기 버전입니다.

사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

이 기능은 [Greengrass nucleus 구성 요소](#) v2.4.0 이상에서 사용할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어 설치 관리자는 필요한 AWS 리소스를 프로비저닝하는 사용자 지정 플러그인에서 구현할 수 있는 Java 인터페이스를 제공합니다. 사용자 지정 X.509 클라이언트 인증서가 사용되거나 다른 설치 프로세스에서 지원되지 않는 복잡한 프로비저닝 단계가 실행되는 프로비저닝 플러그인을 개발할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [자체 클라이언트 인증서 생성](#)을 참조하세요.

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 사용자 지정 프로비저닝 플러그인을 실행하려면 설치 프로그램에 제공하는 JAR 파일을 생성합니다. 설치 관리자가 플러그인을 실행하고 플러그인은 Greengrass 코어 디바이스의 AWS 리소스를 정의하는 프로비저닝 구성을 반환합니다. 설치 관리자는 이 정보를 사용하여 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 구성합니다. 자세한 내용은 [사용자 지정 프로비저닝 플러그인 개발](#) 단원을 참조하십시오.

Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인합니다.

주제

- [사전 조건](#)
- [디바이스 환경 설정](#)
- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [사용자 지정 프로비저닝 플러그인 개발](#)

사전 조건

사용자 지정 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 다음이 있어야 합니다.

- DeviceIdentityInterface가 구현되는 사용자 지정 프로비저닝 플러그인용 JAR 파일. 사용자 지정 프로비저닝 플러그인에서는 각 시스템 및 nucleus 구성 파라미터에 대한 값이 반환되어야 합니다. 그렇지 않으면 설치 중 구성 파일에서 해당 값을 제공해야 합니다. 자세한 내용은 [사용자 지정 프로비저닝 플러그인 개발](#) 단원을 참조하십시오.

디바이스 환경 설정

이 섹션의 단계에 따라 AWS IoT Greengrass 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

Windows 디바이스 설정

Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
 - a. Windows 키를 눌러 시작 메뉴를 엽니다.
 - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
 - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
 - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
 - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
 - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.


```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- wmic 명령이 더는 사용되지 않는 Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

Note

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

다음 위치에서 특정 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다
greengrass-nucleus-latest.zip.

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 계약](#)에 동의하는 것입니다.

2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

Note

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.

- a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6_10*을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6_10*을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.
 - i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

- ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. -verify 및 -certs 옵션과 함께 Jarsigner -verbose 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.
- 3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다.
*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./ GreengrassInstaller/lib/Greengrass.jar --version
```

Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는 이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는 이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후 이 폴더를 제거할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자를 실행합니다.

- 사용자 지정 프로비저닝 플러그인을 사용하여 AWS 리소스를 프로비저닝하도록 지정하는 부분 구성 파일에서를 설치합니다. AWS IoT Greengrass 코어 소프트웨어는 디바이스에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 관리자는 사용자가 제공하는 부분 구성 파일과 사용자 지정 프로비저닝 플러그인이 생성하는 AWS 리소스에서 전체 구성 파일을 생성합니다.
- ggc_user 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 ggc_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.

- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

Note

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 nucleus 구성 요소의 jvmOptions 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어를 설치하려면(Linux)

1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.
 - **GreengrassInstaller**를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 텍스트 편집기를 사용하여 config.yaml 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 내용을 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: "2.16.1"
    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: ""
```

뒤이어 다음과 같이 하세요.

- **2.16.1**을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.
- **/greengrass/v2**의 각 인스턴스를 Greengrass 루트 폴더로 바꿉니다.
- (선택 사항) 시스템 및 nucleus 구성 값을 지정합니다. 이러한 값은 프로비저닝 플러그인에서 제공되지 않으면 설정해야 합니다.
- (선택 사항) 프로비저닝 플러그인에 제공되는 구성 파라미터를 지정합니다.

Note

이 구성 파일에서는 사용되는 포트 및 네트워크 프록시와 같은 기타 구성 옵션을 다음 예제와 같이 사용자 지정할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성](#)을 참조하세요.

```
---
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: "2.16.1"
    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 플러그인이 제공되는 `--trusted-plugin`을 지정하고, 구성 파일이 제공되는 `--init-config`를 지정합니다.

Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: `C:\greengrass\v2` 또는 `D:\greengrass\v2`)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.
- 설치 프로그램을 압축 해제한 폴더로 *GreengrassInstaller*의 각 인스턴스를 바꿉니다.

- 플러그인의 JAR 파일 경로로 사용자 지정 프로비저닝 플러그인 JAR 파일의 경로를 바꿉니다.

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 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다.

니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

Note

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass 참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

사용자 지정 프로비저닝 플러그인 개발

사용자 지정 프로비저닝 플러그인을 개발하려면

`com.aws.greengrass.provisioning.DeviceIdentityInterface` 인터페이스가 구현되는 Java 클래스를 생성합니다. 프로젝트에 Greengrass nucleus JAR 파일을 포함하여 이 인터페이스와 해당 클래스에 액세스할 수 있습니다. 이 인터페이스에서는 플러그인 구성이 입력되고 프로비저닝 구성이 출력되는 방법이 정의됩니다. 프로비저닝 구성에서는 시스템 및 [Greengrass nucleus 구성 요소](#)에 대한 구성 정의됩니다. AWS IoT Greengrass 코어 소프트웨어 설치 관리자는 이 프로비저닝 구성을 사용하여 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 구성합니다.

사용자 지정 프로비저닝 플러그인을 개발한 후 설치 중에 플러그인을 실행하기 위해 AWS IoT Greengrass 코어 소프트웨어 설치 프로그램에 제공할 수 있는 JAR 파일로 빌드합니다. 설치 프로그램에서는 설치 프로그램에서 사용되는 동일한 JVM에서 사용자 지정 프로비저닝 플러그인이 실행되므로 플러그인 코드만 있는 JAR를 생성할 수 있습니다.

Note

[AWS IoT 플릿 프로비저닝 플러그인](#)에서는 설치 중 플릿 프로비저닝이 사용되도록 `DeviceIdentityInterface`가 구현됩니다. 플릿 프로비저닝 플러그인은 오픈 소스이므로 소스 코드를 탐색하여 프로비저닝 플러그인 인터페이스를 사용하는 방법의 예제를 참조할 수 있습니다. 자세한 내용은 GitHub의 [AWS IoT 플릿 프로비저닝 플러그인](#)을 참조하세요.

주제

- [요구 사항](#)
- [DeviceIdentityInterface 인터페이스 구현](#)

요구 사항

사용자 지정 프로비저닝 플러그인을 개발하려면 다음과 같은 요구 사항이 충족되는 Java 클래스를 생성해야 합니다.

- `com.aws.greengrass` 패키지 또는 패키지 내 `com.aws.greengrass` 패키지를 사용합니다.
- 인수가 없는 생성자가 있습니다.
- `DeviceIdentityInterface` 인터페이스를 구현합니다. 자세한 내용은 [DeviceIdentityInterface 인터페이스 구현](#) 단원을 참조하십시오.

DeviceIdentityInterface 인터페이스 구현

사용자 지정 플러그인에서 `com.aws.greengrass.provisioning.DeviceIdentityInterface` 인터페이스를 사용하려면 프로젝트에 Greengrass nucleus를 종속성으로 추가합니다.

사용자 지정 프로비저닝 플러그인 프로젝트에서 `DeviceIdentityInterface`를 사용하려면

- Greengrass nucleus JAR 파일을 라이브러리로 추가하거나 Greengrass nucleus를 Maven 종속성으로 추가할 수 있습니다. 다음 중 하나를 수행하세요.
- Greengrass nucleus JAR 파일을 라이브러리로 추가하려면 Greengrass nucleus JAR이 포함된 AWS IoT Greengrass 코어 소프트웨어를 다운로드합니다. 다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.
- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

ZIP 파일의 `lib` 폴더에서 Greengrass nucleus JAR 파일(`Greengrass.jar`)을 찾을 수 있습니다. 프로젝트에 이 JAR 파일을 추가합니다.

- Maven 프로젝트에서 Greengrass nucleus를 사용하려면 `com.aws.greengrass` 그룹의 `nucleus` 아티팩트에서 종속성을 추가합니다. Greengrass nucleus는 Maven Central Repository에서 사용될 수 없으므로 `greengrass-common` 리포지토리로 추가해야 합니다.

```
<project ...>
  ...
  <repositories>
    <repository>
      <id>greengrass-common</id>
      <name>greengrass common</name>
      <url>https://d2jrmugq4soidf.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>
```

DeviceIdentityInterface 인터페이스

`com.aws.greengrass.provisioning.DeviceIdentityInterface` 인터페이스의 모양은 다음과 같습니다.

Note

GitHub에서 [Greengrass nucleus 소스 코드](#)의 [com.aws.greengrass.provisioning package](#)에서도 이러한 클래스를 탐색할 수 있습니다.

```
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 {}
```

SystemConfiguration 및의 각 구성 값은 AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 NucleusConfiguration 필요하지만 null을 반환할 수 있습니다. 사용자 지정 프로비저닝 플러그인이 구성 값에 null 대해를 반환하는 경우 AWS IoT Greengrass 코어 소프트웨어 설치 프로그램에 제공할 config.yaml 파일을 생성할 때 시스템 또는 nucleus 구성에 해당 값을 제공해야 합니다. config.yaml에서도 정의한 옵션에 대해 null이 아닌 값이 사용자 지정 프로비저닝 플러그인이 반환되면 설치 프로그램에서는 플러그인에서 반환된 값으로 config.yaml의 값이 바뀝니다.

설치 프로그램 인수

AWS IoT Greengrass 코어 소프트웨어에는 소프트웨어가 설정되고 Greengrass 코어 디바이스에서 실행될 필수 AWS 리소스가 프로비저닝되는 설치 프로그램이 포함되어 있습니다. 설치 프로그램에는 지정하여 설치를 구성할 수 있는 다음 인수가 포함되어 있습니다.

-h, --help

(선택 사항) 설치 프로그램의 도움말 정보를 표시합니다.

--version

(선택 사항) AWS IoT Greengrass 코어 소프트웨어의 버전을 표시합니다.

-Droot

(선택 사항) AWS IoT Greengrass 코어 소프트웨어의 루트로 사용할 폴더의 경로입니다.

Note

이 인수를 통해 JVM 속성이 설정되므로 설치 프로그램을 실행할 때 -jar 앞에 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 Greengrass 코어 소프트웨어에서 필수 AWS 리소스를 검색하거나 생성하는 데 사용되는 AWS 리전입니다.

-p, --provision

(선택 사항) 이 디바이스를 AWS IoT 사물로 등록하고 코어 디바이스에 필요한 AWS 리소스를 프로비저닝할 수 있습니다. true를 지정하면 AWS IoT Greengrass 코어 소프트웨어에서는 AWS IoT 사물, AWS IoT 사물 그룹(선택 사항), IAM 역할 및 AWS IoT 역할 별칭이 프로비저닝됩니다.

기본값: false

-tn, --thing-name

(선택 사항) 이 코어 디바이스로 등록하는 AWS IoT 사물의 이름입니다. 이름 있는 정책이 AWS 계정에 없으면 AWS IoT Greengrass 코어 소프트웨어에서 생성됩니다.

 Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

이 인수를 적용하려면 --provision true를 지정해야 합니다.

기본값: GreengrassV2IotThing_ 및 무작위 UUID.

-tgn, --thing-group-name

(선택 사항) 이 코어 디바이스의 AWS IoT 사물을 추가하는 AWS IoT 사물 그룹의 이름입니다. 배포의 대상이 이 사물 그룹이라면 이 코어 디바이스에서는 AWS IoT Greengrass에 연결될 때 해당 배포가 수신됩니다. 이 이름의 사물 그룹이 AWS 계정에 없으면 AWS IoT Greengrass 코어 소프트웨어에서 생성됩니다.

 Note

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

이 인수를 적용하려면 --provision true를 지정해야 합니다.

-tpn, --thing-policy-name

이 기능은 [Greengrass nucleus 구성 요소](#) v2.4.0 이상에서 사용할 수 있습니다.

(선택 사항) 이 코어 디바이스의 AWS IoT 사물 인증서에 연결할 AWS IoT 정책의 이름입니다. 이 이름의 AWS IoT 정책이 AWS 계정에 없으면 AWS IoT Greengrass 코어 소프트웨어에서 생성됩니다.

AWS IoT Greengrass 코어 소프트웨어에서는 기본적으로 허용적 AWS IoT 정책이 생성됩니다. 이 정책의 범위를 좁히거나 사용 사례에 대한 권한을 제한하는 사용자 지정 정책을 생성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 섹션을 참조하세요.

이 인수를 적용하려면 `--provision true`를 지정해야 합니다.

기본값: GreengrassV2IoTThingPolicy

`-trn, --tes-role-name`

(선택 사항) 코어 디바이스에서 AWS 서비스와 상호 작용할 수 있도록 AWS 자격 증명을 획득하는데 사용할 IAM 역할의 이름입니다. 이 이름의 역할이 AWS 계정에 없으면 AWS IoT Greengrass 코어 소프트웨어에서 GreengrassV2TokenExchangeRoleAccess 정책을 통해 생성됩니다. 이 역할에는 구성 요소 아티팩트를 호스팅하는 S3 버킷에 액세스할 권한이 없습니다. 따라서 구성 요소를 생성할 때 아티팩트의 S3 버킷과 객체에 권한을 추가해야 합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

이 인수를 적용하려면 `--provision true`를 지정해야 합니다.

기본값: GreengrassV2TokenExchangeRole

`-tra, --tes-role-alias-name`

(선택 사항) 이 코어 디바이스에 대한 AWS 자격 증명을 제공하는 IAM 역할을 지목하는 AWS IoT 역할 별칭의 이름입니다. 이 이름의 역할 별칭이 AWS 계정에 없으면 AWS IoT Greengrass 코어 소프트웨어에서 생성되며, 지정하는 IAM 역할이 지목됩니다.

이 인수를 적용하려면 `--provision true`를 지정해야 합니다.

기본값: GreengrassV2TokenExchangeRoleAlias

`-ss, --setup-system-service`

(선택 사항) AWS IoT Greengrass 코어 소프트웨어를 이 디바이스가 부팅될 때 실행되는 시스템 서비스로 설정할 수 있습니다. 시스템 서비스 이름은 greengrass입니다. 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 섹션을 참조하세요.

Linux 운영 체제에서 이 인수를 사용하려면 디바이스에서 systemd init 시스템을 사용할 수 있어야 합니다.

⚠ Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

기본값: false

`-u, --component-default-user`

AWS IoT Greengrass 코어 소프트웨어에서 구성 요소 실행에 사용되는 사용자의 이름 또는 ID입니다. 예를 들어, **ggc_user**를 지정할 수 있습니다. Windows 운영 체제에서 설치 프로그램을 실행할 때 이 값이 필요합니다.

Linux 운영 체제에서는 선택적으로 그룹을 지정할 수도 있습니다. 콜론으로 구분되는 사용자 및 그룹을 지정합니다. 예를 들어 **ggc_user:ggc_group**입니다.

Linux 운영 체제에는 다음 추가 고려 사항이 적용됩니다.

- 루트로 실행하는 경우 기본 구성 요소 사용자는 구성 파일에 정의되어 있는 사용자입니다. 구성 파일에서 사용자가 정의되지 않으면 이 기본값은 `ggc_user:ggc_group`으로 설정됩니다. `ggc_user` 또는 `ggc_group`이 없으면 소프트웨어에서 생성합니다.
- 루트 사용자가 아닌 사용자로 실행하면 AWS IoT Greengrass 코어 소프트웨어에서는 해당 사용자가 구성 요소 실행에 사용됩니다.
- 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어에서는 시스템 사용자의 기본 그룹이 사용됩니다.

자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 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 디버깅할 수 있습니다.

⚠ Important

이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제공합니다. 필요한 코어 디바이스에만 이 구성 요소를 배포하여 최소 권한 원칙을 따릅니다.

이 인수를 적용하려면 `--provision true`를 지정해야 합니다.

기본값: `false`

`-init, --init-config`

(선택 사항) AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 사용할 구성 파일의 경로입니다. 예를 들어 이 옵션을 사용하여 특정 nucleus 구성으로 새 코어 디바이스를 설정할 수 있습니다.

Important

지정하는 구성 파일은 코어 디바이스의 기존 구성 파일과 병합됩니다. 여기에는 코어 디바이스의 구성 요소 및 구성 요소 구성이 포함됩니다. 구성 파일에는 변경하려는 구성만 나열되는 것이 좋습니다.

`-tp, --trusted-plugin`

(선택 사항) 신뢰할 수 있는 플러그인으로 로드할 JAR 파일의 경로입니다. 이 옵션을 사용하여 예컨대 [플릿 프로비저닝](#) 또는 [사용자 지정 프로비저닝](#)으로 설치하거나 프라이빗 키 및 인증서로 [하드웨어 보안 모듈](#)에 설치할 프로비저닝 플러그인 JAR 파일을 제공합니다.

`-s, --start`

(선택 사항) 설치하고 선택적으로 리소스를 프로비저닝한 후 AWS IoT Greengrass 코어 소프트웨어를 시작할 수 있습니다.

기본값: `true`

AWS IoT Greengrass 코어 소프트웨어 실행

[AWS IoT Greengrass 코어 소프트웨어를 설치한](#) 후 실행하여 디바이스를 연결합니다 AWS IoT Greengrass.

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 [systemd](#)를 사용하여 시스템 서비스로 설치할지 여부를 지정할 수 있습니다. 이 옵션을 선택하면 설치 관리자가 소프트웨어를 실행하고 디바이스 부팅 시 이를 실행하도록 구성합니다.

⚠ Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

주제

- [AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 실행되는지 확인](#)
- [AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 실행](#)
- [시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어 실행](#)

AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 실행되는지 확인

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 `--setup-system-service true` 인수를 지정하여 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치할 수 있습니다. Linux 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정하려면 [systemd](#) init 시스템이 필요합니다. 이 옵션을 사용하면 설치 관리자가 소프트웨어를 실행하고 디바이스 부팅 시 이를 실행하도록 구성합니다. AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 성공적으로 설치하면 설치 관리자가 다음 메시지를 출력합니다.

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

이전에 AWS IoT Greengrass 코어 소프트웨어를 설치했지만 설치 관리자 출력이 없는 경우 소프트웨어가 시스템 서비스로 설치되었는지 확인할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 설치되어 있는지 확인하려면

- 다음 명령을 실행하여 Greengrass 시스템 서비스의 상태를 확인합니다.

Linux or Unix (systemd)

```
sudo systemctl status greengrass.service
```

AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 설치되고 활성 상태인 경우 응답은 다음 예제와 유사합니다.

```
# 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...

```

systemctl 또는 greengrass.service를 찾을 수 없는 경우 AWS IoT Greengrass 코어 소프트웨어는 시스템 서비스로 설치되지 않습니다. 소프트웨어를 실행하려면 [시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어 실행](#) 섹션을 참조하세요.

Windows Command Prompt (CMD)

```
sc query greengrass
```

AWS IoT Greengrass 코어 소프트웨어가 Windows 서비스로 설치되고 활성 상태인 경우 응답은 다음 예제와 유사합니다.

```

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 Greengrass 코어 소프트웨어가 Windows 서비스로 설치되고 활성 상태인 경우 응답은 다음 예제와 유사합니다.

Status	Name	DisplayName
Running	greengrass	greengrass

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 실행

AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 설치된 경우 시스템 서비스 관리자를 사용하여 소프트웨어를 시작, 중지 및 관리할 수 있습니다. 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 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 코어 소프트웨어 실행

Linux 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어가 시스템 서비스로 설치되지 않은 경우 소프트웨어의 로더 스크립트를 실행하여 소프트웨어를 실행할 수 있습니다.

시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어를 실행하려면

- 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어를 시작합니다. 터미널에서이 명령을 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어를 계속 실행하려면 터미널 세션을 열어 두어야 합니다.
- `/greengrass/v2` 또는 `C:\greengrass\v2`를 사용하는 Greengrass 루트 폴더로 바꿉니다.

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

소프트웨어가 성공적으로 시작되면 다음 메시지를 인쇄합니다.

Launched Nucleus successfully.

Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행

AWS IoT Greengrass 는 Docker 컨테이너에서 실행되도록 구성할 수 있습니다. Docker는 Linux 컨테이너를 기반으로 하는 애플리케이션을 빌드, 실행, 테스트, 배포할 수 있는 도구를 제공하는 플랫폼입니다. AWS IoT Greengrass Docker 이미지를 실행할 때 자격 증명을 Docker 컨테이너에 제공할지 여부를 선택하고 AWS IoT Greengrass 코어 소프트웨어 설치 관리자가 Greengrass 코어 디바이스가 작동하는 데 필요한 리소스를 자동으로 프로비저닝 AWS 하도록 허용할 수 있습니다. 자격 증명을 제공하지 AWS 않으려면 AWS 리소스를 수동으로 프로비저닝하고 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행할 수 있습니다.

주제

- [지원되는 플랫폼 및 요구 사항](#)
- [AWS IoT Greengrass Docker 소프트웨어 다운로드](#)
- [AWS 리소스를 프로비저닝하는 방법 선택](#)
- [Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지 빌드](#)
- [자동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행](#)
- [수동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행](#)
- [Docker 컨테이너 AWS IoT Greengrass 의 문제 해결](#)

지원되는 플랫폼 및 요구 사항

Docker 컨테이너에 AWS IoT Greengrass 코어 소프트웨어를 설치하고 실행하려면 호스트 컴퓨터가 다음과 같은 최소 요구 사항을 충족해야 합니다.

- 인터넷에 연결된 Linux 기반 운영 체제.
- [Docker Engine](#) 버전 18.09 이상.
- (선택 사항) [Docker Compose](#) 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 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.

프로세스 모드에서 구성 요소 실행

AWS IoT Greengrass 는 AWS IoT Greengrass Docker 컨테이너 내의 격리된 런타임 환경에서 Lambda 함수 또는 AWS제공 구성 요소의 실행을 지원하지 않습니다. 이러한 구성 요소는 격리 없이 프로세스 모드에서 실행해야 합니다.

Lambda 함수 구성 요소를 구성하는 경우 격리 모드를 컨테이너 없음으로 설정합니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.

다음 AWS제공 구성 요소를 배포할 때 containerMode 파라미터를 로 설정하도록 각 구성 요소의 구성을 업데이트합니다NoContainer. 구성 업데이트에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

- [CloudWatch 지표](#)
- [Device Defender](#)
- [Firehose](#)
- [Modbus-RTU 프로토콜 어댑터](#)
- [Amazon SNS](#)

AWS IoT Greengrass Docker 소프트웨어 다운로드

AWS IoT Greengrass 는 Amazon Linux 2(x86_64) 기본 이미지에 AWS IoT Greengrass 코어 소프트웨어 및 종속성이 설치된 컨테이너 이미지를 빌드하기 위한 Dockerfile을 제공합니다. 다른 플랫폼 아키텍처에서 실행되도록 Dockerfile의 기본 이미지를 수정할 수 AWS IoT Greengrass 있습니다.

[GitHub](#)에서 Dockerfile 패키지를 다운로드합니다.

Dockerfile은 Greengrass의 이전 버전을 사용합니다. 원하는 Greengrass 버전을 사용하도록 파일을 업데이트해야 합니다. Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지를 빌드하는 방법에 대한 자세한 내용은 섹션을 참조하세요[Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지 빌드](#).

AWS 리소스를 프로비저닝하는 방법 선택

Docker 컨테이너에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 Greengrass 코어 디바이스가 작동하는 데 필요한 AWS 리소스를 자동으로 프로비저닝할지 아니면 수동으로 프로비저닝하는 리소스를 사용할지 선택할 수 있습니다.

- 자동 리소스 프로비저닝 - AWS IoT Greengrass 컨테이너 이미지를 처음 실행할 때 설치 관리자가 AWS IoT 사물, AWS IoT 사물 그룹, IAM 역할 및 AWS IoT 역할 별칭을 프로비저닝합니다. 설치 관

리자는 또한 로컬 개발 도구를 코어 디바이스에 배포할 수 있으므로 디바이스를 사용하여 사용자 지정 소프트웨어 구성 요소를 개발하고 테스트할 수 있습니다. 이러한 리소스를 자동으로 프로비저닝하려면 Docker 이미지에 환경 변수로 AWS 자격 증명을 제공해야 합니다.

자동 프로비저닝을 사용하려면 Docker 환경 변수(`PROVISION=true`)를 설정하고 자격 증명 파일을 탑재하여 컨테이너에 AWS 자격 증명을 제공해야 합니다.

- 수동 리소스 프로비저닝 - 컨테이너에 AWS 자격 증명을 제공하지 않으려면 AWS IoT Greengrass 컨테이너 이미지를 실행하기 전에 리소스를 수동으로 프로비저닝 AWS 할 수 있습니다. 이러한 리소스에 대한 정보를 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 프로그램에 제공하려면 구성 파일을 생성해야 합니다.

수동 프로비저닝을 사용하려면 Docker 환경 변수(`PROVISION=false`)를 설정해야 합니다. 수동 프로비저닝이 기본 옵션입니다.

자세한 내용은 [Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지 빌드](#) 단원을 참조하십시오.

Dockerfile에서 AWS IoT Greengrass 컨테이너 이미지 빌드

AWS 는 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행하기 위해 다운로드하고 사용할 수 있는 Dockerfile을 제공합니다. Dockerfiles에는 AWS IoT Greengrass 컨테이너 이미지를 빌드하기 위한 소스 코드가 포함되어 있습니다.

AWS IoT Greengrass 컨테이너 이미지를 빌드하기 전에 설치할 AWS IoT Greengrass 코어 소프트웨어 버전을 선택하도록 Dockerfile을 구성해야 합니다. 환경 변수를 구성하여 설치 중 리소스를 프로비저닝하는 방법을 선택하고 다른 설치 옵션을 사용자 지정할 수도 있습니다. 이 섹션에서는 AWS IoT Greengrass Dockerfile에서 Docker 이미지를 구성하고 빌드하는 방법을 설명합니다.

Dockerfile 패키지 다운로드

GitHub에서 AWS IoT Greengrass Dockerfile 패키지를 다운로드할 수 있습니다.

[AWS Greengrass Docker 리포지토리](#)

패키지를 다운로드한 후 컴퓨터의 `download-directory/aws-greengrass-docker-nucleus-version` 폴더에 내용을 추출합니다. Dockerfile은 Greengrass의 이전 버전을 사용합니다. 원하는 Greengrass 버전을 사용하도록 파일을 업데이트해야 합니다.

AWS IoT Greengrass 코어 소프트웨어 버전 지정

Dockerfile에서 다음 빌드 인수를 사용하여 AWS IoT Greengrass Docker 이미지에 사용할 AWS IoT Greengrass 코어 소프트웨어의 버전을 지정합니다. 기본적으로 Dockerfile은 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 사용합니다.

GREENGRASS_RELEASE_VERSION

AWS IoT Greengrass 코어 소프트웨어의 버전입니다. 기본적으로 Dockerfile에서는 Greengrass nucleus의 사용 가능한 최신 버전이 다운로드됩니다. 다운로드하려는 nucleus의 버전으로 값을 설정합니다.

환경 변수 설정

환경 변수를 사용하면 AWS IoT Greengrass 코어 소프트웨어가 Docker 컨테이너에 설치되는 방식을 사용자 지정할 수 있습니다. 다양한 방법으로 AWS IoT Greengrass 도커 이미지의 환경 변수를 설정할 수 있습니다.

- 동일한 환경 변수를 사용하여 여러 이미지를 생성하려면 Dockerfile에서 곧바로 환경 변수를 설정합니다.
- `docker run`을 사용하여 컨테이너를 시작하는 경우 환경 변수를 명령의 인수로 전달하거나 환경 변수 파일에서 환경 변수를 설정한 다음에 파일을 인수로 전달합니다. Docker에서 환경 변수를 설정하는 방법에 대한 자세한 내용은 Docker 설명서의 [환경 변수](#)를 참조하세요.
- `docker-compose up`을 사용하여 컨테이너를 시작하는 경우 환경 변수 파일에서 환경 변수를 설정한 다음에 파일을 인수로 전달합니다. Compose에서 환경 변수를 설정하는 방법에 대한 자세한 내용은 [Docker 설명서](#)를 참조하세요.

AWS IoT Greengrass Docker 이미지에 대해 다음 환경 변수를 구성할 수 있습니다.

Note

Dockerfile에서 `TINI_KILL_PROCESS_GROUP` 변수는 수정하지 마세요. 이 변수를 사용하면 PIDs 그룹의 모든 `PIDSIGTERM`로 전달할 수 있으므로 Docker 컨테이너가 중지될 때 AWS IoT Greengrass 코어 소프트웨어가 올바르게 종료될 수 있습니다.

GGC_ROOT_PATH

(선택 사항) AWS IoT Greengrass 코어 소프트웨어의 루트로 사용할 컨테이너 내 폴더의 경로입니다.

기본값: /greengrass/v2

PROVISION

(선택 사항) AWS IoT Greengrass 코어가 AWS 리소스를 프로비저닝할지 여부를 결정합니다.

- 를 지정하면 true AWS IoT Greengrass 코어 소프트웨어는 컨테이너 이미지를 AWS IoT 사물로 등록하고 Greengrass 코어 디바이스에 필요한 AWS 리소스를 프로비저닝합니다. AWS IoT Greengrass 코어 소프트웨어는 AWS IoT 사물, AWS IoT 사물 그룹(선택 사항), IAM 역할 및 AWS IoT 역할 별칭을 프로비저닝합니다. 자세한 내용은 [자동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행](#) 단원을 참조하십시오.
- 를 지정하는 경우 수동으로 생성한 AWS 리소스와 인증서를 사용하도록 지정하는 구성 파일을 생성하여 AWS IoT Greengrass 코어 설치 프로그램에 제공해야 false합니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행](#) 단원을 참조하십시오.

기본값: false

AWS_REGION

(선택 사항) AWS 리전 AWS IoT Greengrass 코어 소프트웨어가 필요한 AWS 리소스를 검색하거나 생성하는 데 사용하는 입니다.

기본값: us-east-1.

THING_NAME

(선택 사항)이 코어 디바이스로 등록하는 AWS IoT 사물의 이름입니다. 이 이름의 사물이에 없으면 AWS 계정 AWS IoT Greengrass 코어 소프트웨어가 생성합니다.

이 인수를 적용하려면 PROVISION=true를 지정해야 합니다.

기본값: GreengrassV2IotThing_ 및 무작위 UUID.

THING_GROUP_NAME

(선택 사항)이 코어 디바이스 AWS IoT 의를 추가하는 AWS IoT 사물 그룹의 이름 배포가이 사물 그룹을 대상으로 하는 경우 해당 그룹의이 및 기타 코어 디바이스는 연결될 때 해당 배포를 수신합니다

다 AWS IoT Greengrass. 이 이름의 사물 그룹이 없는 경우 AWS 계정 AWS IoT Greengrass 코어 소프트웨어가 이를 생성합니다.

이 인수를 적용하려면 `PROVISION=true`를 지정해야 합니다.

TES_ROLE_NAME

(선택 사항) Greengrass 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 자격 증명을 획득하는 데 사용할 IAM 역할의 이름입니다. 이 이름의 역할이 없는 경우 AWS 계정 AWS IoT Greengrass 코어 소프트웨어가 `GreengrassV2TokenExchangeRoleAccess` 정책을 사용하여 생성합니다. 이 역할에는 구성 요소 아티팩트를 호스팅하는 S3 버킷에 액세스할 권한이 없습니다. 따라서 구성 요소를 생성할 때 아티팩트의 S3 버킷과 객체에 권한을 추가해야 합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

기본값: `GreengrassV2TokenExchangeRole`

TES_ROLE_ALIAS_NAME

(선택 사항) Greengrass 코어 디바이스에 대한 AWS 자격 증명을 제공하는 IAM 역할을 가리키는 역할 AWS IoT 별칭의 이름입니다. 이 이름의 역할 별칭이 없는 경우 AWS 계정 AWS IoT Greengrass 코어 소프트웨어가 이를 생성하고 지정한 IAM 역할을 가리킵니다.

기본값: `GreengrassV2TokenExchangeRoleAlias`

COMPONENT_DEFAULT_USER

(선택 사항) AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하는 데 사용하는 시스템 사용자 및 그룹의 이름 또는 ID입니다. 콜론으로 구분되는 사용자 및 그룹을 지정합니다. 그룹은 선택 사항입니다. 예를 들어 `ggc_user:ggc_group` 또는 `ggc_user`를 지정할 수 있습니다.

- 루트로 실행하면 구성 파일에서 정의되는 사용자 및 그룹으로 이 기본값이 설정됩니다. 구성 파일에서 사용자 및 그룹이 정의되지 않으면 이 기본값은 `ggc_user:ggc_group`으로 설정됩니다. `ggc_user` 또는 `ggc_group`이 없으면 소프트웨어에서 생성합니다.
- 루트가 아닌 사용자로 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어는 해당 사용자를 사용하여 구성 요소를 실행합니다.
- 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 시스템 사용자의 기본 그룹을 사용합니다.

자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 단원을 참조하십시오.

DEPLOY_DEV_TOOLS

컨테이너 이미지에서 [Greengrass CLI 구성 요소](#)를 다운로드하고 배포할지 여부를 정의합니다. Greengrass CLI를 사용하여 로컬에서 구성 요소를 개발하고 디버깅할 수 있습니다.

⚠ Important

이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제공합니다. 필요한 코어 디바이스에만 이 구성 요소를 배포하여 최소 권한 원칙을 따릅니다.

기본값: false

INIT_CONFIG

(선택 사항) AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 사용할 구성 파일의 경로입니다. 예를 들어 이 옵션을 사용하여 특정 nucleus 구성으로 새 Greengrass 코어 디바이스를 설정하거나 수동으로 프로비저닝한 리소스를 지정할 수 있습니다. 이 인수에서 지정하는 경로에 구성 파일을 탑재해야 합니다.

TRUSTED_PLUGIN

이 기능은 [Greengrass nucleus 구성 요소](#) v2.4.0 이상에서 사용할 수 있습니다.

(선택 사항) 신뢰할 수 있는 플러그인으로 로드할 JAR 파일의 경로입니다. [플릿 프로비저닝](#) 또는 [사용자 지정](#) 프로비저닝과 함께 설치할 정도로 프로비저닝 플러그인 JAR 파일을 제공하려면 이 옵션을 사용합니다.

THING_POLICY_NAME

이 기능은 [Greengrass nucleus 구성 요소](#) v2.4.0 이상에서 사용할 수 있습니다.

(선택 사항)이 코어 디바이스의 AWS IoT 사물 인증서에 연결할 AWS IoT 정책의 이름입니다. AWS IoT Greengrass 코어 소프트웨어에이 이름 AWS 계정 의 AWS IoT 정책이 없는 경우 정책이 생성됩니다.

이 인수를 적용하려면 PROVISION=true를 지정해야 합니다.

i Note

AWS IoT Greengrass 코어 소프트웨어는 기본적으로 허용 AWS IoT 정책을 생성합니다. 이 정책의 범위를 좁히거나 사용 사례에 대한 권한을 제한하는 사용자 지정 정책을 생성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.

설치할 종속성 지정

AWS IoT Greengrass Dockerfile의 RUN 명령은 AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 실행하기 위해 컨테이너 환경을 준비합니다. AWS IoT Greengrass 코어 소프트웨어 설치 프로그램이 Docker 컨테이너에서 실행되기 전에 설치되는 종속성을 사용자 지정할 수 있습니다.

AWS IoT Greengrass 이미지 빌드

AWS IoT Greengrass Dockerfile을 사용하여 AWS IoT Greengrass 컨테이너 이미지를 빌드합니다. Docker CLI 또는 Docker Compose CLI를 사용하여 이미지를 빌드하고 컨테이너를 시작할 수 있습니다. Docker CLI를 사용하여 이미지를 빌드한 다음에 Docker Compose를 사용하여 해당 이미지로부터 컨테이너를 시작할 수도 있습니다.

Docker

1. 호스트 머신에서 다음 명령을 실행하여 구성된 Dockerfile이 있는 디렉터리로 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 머신에서 다음 명령을 실행하여 Dockerfile과 Compose 파일이 있는 디렉터리로 전환합니다.

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. 다음 명령을 실행하여 Compose 파일을 사용하여 AWS IoT Greengrass 컨테이너 이미지를 빌드합니다.

```
docker-compose -f docker-compose.yml build
```

AWS IoT Greengrass 컨테이너 이미지를 성공적으로 생성했습니다. Docker 이미지에는 AWS IoT Greengrass 코어 소프트웨어가 설치되어 있습니다. 이제 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행할 수 있습니다.

자동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행

이 자습서에서는 자동으로 프로비저닝된 AWS 리소스 및 로컬 개발 도구를 사용하여 Docker 컨테이너에 AWS IoT Greengrass 코어 소프트웨어를 설치하고 실행하는 방법을 보여줍니다. 이 개발 환경을 사용하여 Docker 컨테이너의 AWS IoT Greengrass 기능을 탐색할 수 있습니다. 소프트웨어가 이러한 리소스를 프로비저닝하고 로컬 개발 도구를 배포하려면 AWS 자격 증명이 필요합니다.

컨테이너에 AWS 자격 증명을 제공할 수 없는 경우 코어 디바이스가 AWS 작동하는 데 필요한 리소스를 프로비저닝할 수 있습니다. 개발 도구를 코어 디바이스에 배포하여 개발 디바이스로 사용할 수도 있습니다. 이렇게 하면 컨테이너 실행 시 디바이스에 더 적은 권한을 제공할 수 있습니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행](#) 단원을 참조하십시오.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 없으면 [설정 AWS 계정](#) 섹션을 참조하세요.
- Greengrass 코어 디바이스에 대한 AWS IoT 및 AWS IAM 리소스를 프로비저닝할 권한이 있는 IAM 사용자입니다. AWS IoT Greengrass 코어 소프트웨어 설치 관리자는 자격 AWS 증명을 사용하여 이러한 리소스를 자동으로 프로비저닝합니다. 리소스를 자동으로 프로비저닝하기 위한 최소 IAM 정책에 대한 자세한 내용은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.
- AWS IoT Greengrass Docker 이미지. [AWS IoT Greengrass Dockerfile에서 이미지를 빌드할 수 있습니다.](#)
- Docker 컨테이너를 실행하는 호스트 컴퓨터는 다음과 같은 요구 사항을 충족해야 합니다.
 - 인터넷에 연결된 Linux 기반 운영 체제.
 - [Docker Engine](#) 버전 18.09 이상.
 - (선택 사항) [Docker Compose](#) 버전 1.22 이상. Docker Compose는 Docker Compose CLI를 사용하여 Docker 이미지를 실행하려는 경우에만 필요합니다.

자격 AWS 증명 구성

이 단계에서는 호스트 컴퓨터에 AWS 보안 자격 증명이 포함된 자격 증명 파일을 생성합니다. AWS IoT Greengrass Docker 이미지를 실행할 때 자격 증명 파일이 포함된 폴더를 Docker 컨테이너의 /

root/.aws/에 탑재해야 합니다. AWS IoT Greengrass 설치 관리자는 이러한 자격 증명을 사용하여 에서 리소스를 프로비저닝합니다 AWS 계정. 설치 관리자가 리소스를 자동으로 프로비저닝하는 데 필요한 최소 IAM 정책에 대한 자세한 내용은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

1. 다음 중 하나를 검색합니다.

- IAM 사용자에게 대한 장기 자격 증명. 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 사용자의 액세스 키 관리](#)를 참조하세요.
- (권장) IAM 역할에 대한 임시 자격 증명. 임시 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [AWS CLI에서 임시 보안 인증 사용](#)을 참조하세요.

2. 자격 증명 파일을 배치할 폴더를 생성합니다.

```
mkdir ./greengrass-v2-credentials
```

3. 텍스트 편집기를 사용하여 ./greengrass-v2-credentials 폴더에 credentials라는 이름의 구성 파일을 생성합니다.

예를 들어 다음 명령을 실행하면 GNU nano를 사용하여 credentials 파일을 생성할 수 있습니다.

```
nano ./greengrass-v2-credentials/credentials
```

4. 다음 형식으로 자격 AWS 증명을 credentials 파일에 추가합니다.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
= AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

임시 자격 증명에만 aws_session_token을 포함합니다.

Important

AWS IoT Greengrass 컨테이너를 시작한 후 호스트 컴퓨터에서 자격 증명 파일을 제거합니다. 자격 증명 파일을 제거하지 않으면 AWS 자격 증명이 컨테이너 내에 탑재된 상태로 유지됩니다.

다. 자세한 내용은 [컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

환경 파일 생성

이 자습서에서는 환경 파일을 사용하여 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 관리자로 전달될 환경 변수를 설정합니다. `docker run` 명령의 [-e 또는 --env 인수](#)를 사용하여 Docker 컨테이너에서 환경 변수를 설정하거나 `docker-compose.yml` 파일의 [environment 블록](#)에서 변수를 설정할 수 있습니다.

1. 텍스트 편집기를 사용하여 `.env`라는 이름의 환경 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 현재 디렉터리에 `.env`를 생성할 수 있습니다.

```
nano .env
```

2. 다음 콘텐츠를 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_ROOT` 환경 변수를 사용하여 이 값을 지정할 수 있습니다.
- `region`. 리소스를 생성한 AWS 리전입니다.
- `MyGreengrassCore`. AWS IoT 사물의 이름입니다. 사물이 존재하지 않는 경우 설치 관리자가 이를 생성합니다. 설치 관리자는 인증서를 다운로드하여 AWS IoT 사물로 인증합니다.
- `MyGreengrassCoreGroup`. AWS IoT 사물 그룹의 이름입니다. 사물 그룹이 존재하지 않는 경우 설치 관리자가 이를 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 활성 배포가 있는 경우 코어 디바이스는 배포에서 지정한 소프트웨어를 다운로드하고 실행합니다.

- **GreengrassV2TokenExchangeRole**. Greengrass 코어 디바이스에서 임시 AWS 자격 증명을 가져올 수 있도록 허용하는 IAM 토큰 교환 역할의 이름으로 바꿉니다. 역할이 없는 경우 설치 관리자는 이를 생성하고 **GreengrassV2TokenExchangeRoleAccess**라는 정책을 생성 및 연결합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.
- **GreengrassCoreTokenExchangeRoleAlias**. 토큰 교환 역할 별칭입니다. 역할 별칭이 없는 경우 설치 관리자가 이를 생성하고 지정한 IAM 토큰 교환 역할을 가리킵니다. 자세한 내용은 다음 섹션을 참조하세요.

Note

DEPLOY_DEV_TOOLS 환경 변수를 true로 설정하여 Docker 컨테이너 내에서 사용자 지정 구성 요소를 개발할 수 있도록 하는 [Greengrass CLI 구성 요소](#)를 배포할 수 있습니다. 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제공합니다. 필요한 코어 디바이스에만 이 구성 요소를 배포하여 최소 권한 원칙을 따르세요.

컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행

이 자습서에서는 Docker 컨테이너에 빌드한 Docker 이미지를 시작하는 방법을 설명합니다. Docker CLI 또는 Docker Compose CLI를 사용하여 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 이미지를 실행할 수 있습니다.

Docker

1. Docker 컨테이너를 시작하려면 다음 명령을 실행합니다.

```
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 실행](#)을 위해 다음 인수를 사용합니다.

- **--rm**. 컨테이너가 종료될 때 컨테이너를 정리합니다.

- [--init](#). 컨테이너에서 init 프로세스를 사용합니다.

Note

--init 인수는 Docker 컨테이너를 중지할 때 AWS IoT Greengrass 코어 소프트웨어를 종료하는 데 필요합니다.

- [-it](#). (선택 사항) 전경의 Docker 컨테이너를 대화형 프로세스로 실행합니다. 이를 대신 Docker 컨테이너를 분리 모드로 실행하기 위한 [-d](#) 인수로 바꿀 수 있습니다. 자세한 내용은 Docker 설명서의 [Detached vs foreground](#)를 참조하세요.
- [--name](#). aws-iot-greengrass라는 이름의 컨테이너를 실행합니다.
- [-v](#). 볼륨을 Docker 컨테이너에 탑재하여 구성 파일과 인증서 파일을 컨테이너 내에서 AWS IoT Greengrass 실행할 수 있도록 합니다.
- [--env-file](#). (선택 사항) 환경 파일을 지정하여 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 관리자로 전달될 환경 변수를 설정합니다. 이 인수는 환경 변수를 설정하기 위해 [환경 파일](#)을 생성한 경우에만 필요합니다. 환경 파일을 생성하지 않은 경우 --env 인수를 사용하여 Docker 실행 명령에서 환경 변수를 직접 설정할 수 있습니다.
- [-p](#). (선택 사항) 8883 컨테이너 포트를 호스트 시스템에 게시합니다. AWS IoT Greengrass는 MQTT 트래픽에 포트 8883을 사용하므로 이 인수는 MQTT를 통해 연결하고 통신하려는 경우에 필요합니다. 다른 포트를 열려면 추가 -p 인수를 사용합니다.

Note

Docker 컨테이너를 강화된 보안으로 실행하려면 --cap-drop 및 --cap-add 인수를 사용하여 컨테이너에 대한 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 [런타임 권한 및 Linux 기능](#)을 참조하세요.

2. 호스트 디바이스의 `./greengrass-v2-credentials`에서 자격 증명을 제거합니다.

```
rm -rf ./greengrass-v2-credentials
```

Important

이러한 자격 증명은 설정 중에만 코어 디바이스에 필요한 광범위한 권한을 제공하므로 이를 제거합니다. 자격 증명을 제거하지 않으면 컨테이너에서 실행되는 Greengrass 구

성 요소와 기타 프로세스가 해당 자격 증명에 액세스할 수 있습니다. Greengrass 구성 요소에 AWS 자격 증명을 제공해야 하는 경우 토큰 교환 서비스를 사용합니다. 자세한 내용은 [AWS 서비스와 상호 작용](#) 단원을 참조하십시오.

Docker Compose

1. 텍스트 편집기를 사용하여 `docker-compose.yml`이라는 이름의 Docker Compose 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 현재 디렉터리에 `docker-compose.yml`를 생성할 수 있습니다.

```
nano docker-compose.yml
```

Note

[GitHub](#)에서 AWS제공 Compose 파일의 최신 버전을 다운로드하여 사용할 수도 있습니다.

2. Compose 파일에 다음 콘텐츠를 추가합니다. 파일은 다음 예제와 비슷할 것입니다. *docker-image*를 Docker 이미지의 이름으로 바꿉니다.

```
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 파라미터는 선택 사항입니다.

- `ports` - 8883 컨테이너 포트를 호스트 시스템에 게시합니다. 이 파라미터는 MQTT 트래픽에 포트 8883을 AWS IoT Greengrass 사용하기 때문에 MQTT를 통해 연결하고 통신하려는 경우에 필요합니다.
- `env_file`- 환경 파일을 지정하여 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 관리자로 전달될 환경 변수를 설정합니다. 이 파라미터는 환경 변수를 설정하기 위해 [환경 파일](#)을 생성한 경우에만 필요합니다. 환경 파일을 생성하지 않은 경우 [환경](#) 파라미터를 사용하여 Compose 파일에서 변수를 직접 설정할 수 있습니다.

Note

Docker 컨테이너를 강화된 보안으로 실행하려면 Compose 파일의 `cap_drop` 및 `cap_add`를 사용하여 컨테이너에 대한 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 [런타임 권한 및 Linux 기능](#)을 참조하세요.

3. Docker 컨테이너를 시작하려면 다음 명령을 실행합니다.

```
docker-compose -f docker-compose.yml up
```

4. 호스트 디바이스의 `./greengrass-v2-credentials`에서 자격 증명을 제거합니다.

```
rm -rf ./greengrass-v2-credentials
```

Important

이러한 자격 증명은 설정 중에만 코어 디바이스에 필요한 광범위한 권한을 제공하므로 이를 제거합니다. 자격 증명을 제거하지 않으면 컨테이너에서 실행되는 Greengrass 구성 요소와 기타 프로세스가 해당 자격 증명에 액세스할 수 있습니다. Greengrass 구성 요소에 AWS 자격 증명을 제공해야 하는 경우 토큰 교환 서비스를 사용합니다. 자세한 내용은 [AWS 서비스와 상호 작용](#) 단원을 참조하십시오.

다음 단계

AWS IoT Greengrass 코어 소프트웨어가 이제 Docker 컨테이너에서 실행 중입니다. 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 코어 소프트웨어를 탐색할 수 있습니다.

```
docker exec -it container-id /bin/bash
```

간단한 구성 요소 생성에 대한 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#)의 [4단계: 디바이스에서 구성 요소 개발 및 테스트](#) 섹션을 참조하세요.

Note

`docker exec`를 사용하여 Docker 컨테이너 내에서 명령을 실행하면 이 명령이 Docker 로그에 기록되지 않습니다. Docker 로그에 명령을 기록하려면 Docker 컨테이너에 대화형 셸을 연결합니다. 자세한 내용은 [대화형 셸을 Docker 컨테이너에 연결](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 로그 파일은 `greengrass.log` 하며에 있습니다 `/greengrass/v2/logs`. 구성 요소 로그 파일도 동일한 디렉터리에 있습니다. Greengrass 로그를 호스트의 임시 디렉터리에 복사하려면 다음 명령을 실행합니다.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

컨테이너가 종료되거나 제거된 후 로그를 유지하려면 호스트의 임시 로그 디렉터리에 전체 Greengrass 디렉터리를 탑재하는 대신 `/greengrass/v2/logs` 디렉터리만 바인딩 탑재하는 것이 좋습니다. 자세한 내용은 [Docker 컨테이너 외부에서 Greengrass 로그 유지](#) 단원을 참조하십시오.

실행 중인 AWS IoT Greengrass Docker 컨테이너를 중지하려면 `docker stop` 또는 `docker-compose -f docker-compose.yml stop`을 실행합니다. 이 작업은 SIGTERM을 Greengrass 프로세스로 전송하고 컨테이너에서 시작된 모든 관련 프로세스를 종료합니다. Docker 컨테이너는 `docker-init` 실행 파일을 사용하여 프로세스 PID 1로 초기화되며, 이는 남아 있는 좀비 프로세스를 제거하는데 도움이 됩니다. 자세한 내용은 Docker 설명서의 [Specify an init process](#)를 참조하세요.

Docker 컨테이너에서 AWS IoT Greengrass를 실행하는 데 발생하는 문제 해결에 대한 자세한 내용은 [Docker 컨테이너 AWS IoT Greengrass의 문제 해결](#) 섹션을 참조하세요.

수동 리소스 프로비저닝을 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행

이 자습서에서는 수동으로 프로비저닝된 AWS 리소스를 사용하여 Docker 컨테이너에 AWS IoT Greengrass 코어 소프트웨어를 설치하고 실행하는 방법을 보여줍니다.

주제

- [사전 조건](#)
- [AWS IoT 엔드포인트 검색](#)
- [AWS IoT 사물 생성](#)
- [사물 인증서 생성](#)
- [토큰 교환 역할 생성](#)
- [디바이스에 인증서 다운로드](#)
- [구성 파일 생성](#)
- [환경 파일 생성](#)
- [컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행](#)
- [다음 단계](#)

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 없으면 [설정 AWS 계정](#) 섹션을 참조하세요.
- AWS IoT Greengrass Docker 이미지. [AWS IoT Greengrass Dockerfile에서 이미지를 빌드할 수 있습니다.](#)
- Docker 컨테이너를 실행하는 호스트 컴퓨터는 다음과 같은 요구 사항을 충족해야 합니다.
 - 인터넷에 연결된 Linux 기반 운영 체제.
 - [Docker Engine](#) 버전 18.09 이상.
 - (선택 사항) [Docker Compose](#) 버전 1.22 이상. Docker Compose는 Docker Compose CLI를 사용하여 Docker 이미지를 실행하려는 경우에만 필요합니다.

AWS IoT 엔드포인트 검색

의 AWS IoT 엔드포인트를 가져 AWS 계정와 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. 해결 방법:

1. 의 AWS IoT 데이터 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 의 AWS IoT 자격 증명 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
}
```

AWS IoT 사물 생성

AWS IoT 사물에는 연결하는 디바이스 및 논리적 엔터티를 나타냅니다 AWS IoT. Greengrass 코어 디바이스는 AWS IoT 사물입니다. 디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다 AWS.

이 섹션에서는 디바이스를 나타내는 AWS IoT 사물을 생성합니다.

AWS IoT 사물을 생성하려면

1. 디바이스에 대한 AWS IoT 사물을 생성합니다. 개발 컴퓨터에서 다음 명령을 실행합니다.
 - *MyGreengrassCore*를 사용할 사물 이름으로 바꿉니다. 이 이름은 Greengrass 코어 디바이스의 이름이기도 합니다.

Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

```
aws iot create-thing --thing-name MyGreengrassCore
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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 또는 기존 사물 그룹에 사물을 추가합니다. 사물 그룹을 사용하여 Greengrass 코어 디바이스 플릿을 관리합니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포가 있는 사물 그룹에 디바이스를 추가하여 사물 그룹의 소프트웨어 구성 요소를 디바이스에 배포할 수 있습니다. 해결 방법:

- a. (선택 사항) AWS IoT 사물 그룹을 생성합니다.

- *MyGreengrassCoreGroup*을 생성할 사물 그룹의 이름으로 바꿉니다.

Note

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
}
```



```
{
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. AWS IoT 사물을 사물 그룹에 추가합니다.

- *MyGreengrassCore*를 AWS IoT 사물의 이름으로 바꿉니다.
- *MyGreengrassCoreGroup*을 사물 그룹의 이름으로 바꿉니다.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 명령에 출력이 없습니다.

사물 인증서 생성

디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다 AWS. 이 인증서를 사용하면 디바이스가 AWS IoT 및와 통신할 수 있습니다 AWS IoT Greengrass.

이 섹션에서는 디바이스가 AWS에 연결하는 데 사용할 수 있는 인증서를 생성하고 다운로드합니다.

사물 인증서를 생성하려면

1. AWS IoT 사물에 대한 인증서를 다운로드할 폴더를 생성합니다.

```
mkdir greengrass-v2-certs
```

2. AWS IoT 사물에 대한 인증서를 생성하고 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
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId": "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----"
```

```

MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBiDELMakGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBA5TC01BTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXRhbnQ21sYWMxHzAdBgkqhkiG9w0BCQEWEG5vb251QGfYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMakGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBA5TC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ21sYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb251QGfYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySwTc2XADZ4nB+BLyGVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcvcQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEATCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQ0CAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTwo
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEeahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEEv9mQ0UXP6p1fgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLE1nI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}

```

이후 인증서 구성에 사용할 인증서의 Amazon 리소스 이름(ARN)을 저장합니다.

그런 다음 사물 인증서를 구성합니다. 자세한 내용은 [사물 인증서 구성](#) 단원을 참조하십시오.

토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라는 IAM 서비스 역할을 사용하여 AWS 서비스에 대한 호출을 승인합니다. 디바이스는 자격 AWS IoT 증명 공급자를 사용하여이 역할에 대한 임시 AWS 자격

증명을 가져오므로 디바이스가 상호 작용하고 AWS IoT, Amazon CloudWatch Logs로 로그를 전송하고, Amazon S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

AWS IoT Anrole 별칭을 사용하여 Greengrass 코어 디바이스에 대한 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 디바이스의 토큰 교환 역할을 변경할 수 있지만 디바이스 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스 직접 호출에 대한 권한 부여](#)를 참조하세요.

이 단원에서는 토큰 교환 IAM 역할과 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다. 이미 Greengrass 코어 디바이스를 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 생성하려면

1. 디바이스가 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 해결 방법:

a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-trust-policy.json
```

다음 JSON을 파일로 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.

- *GreengrassV2TokenExchangeRole*을 생성할 IAM 역할의 이름으로 바꿉니다.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일로 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 S3에서 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오.

구성 요소 아티팩트에 대한 S3 버킷이 아직 없는 경우 버킷을 생성한 후 권한을 추가할 수 있습니다.

d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreengrassV2TokenExchangeRoleAccess*를 생성할 IAM 정책의 이름으로 바꿉니다.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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.

- *GreengrassV2TokenExchangeRole*을 IAM 역할의 이름으로 바꿉니다.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 변경합니다.

```

aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess

```

요청에 성공하는 경우 명령에 출력이 없습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

- *GreengrassCoreTokenExchangeRoleAlias*를 생성할 역할 별칭의 이름으로 바꿉니다.
- 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 변경합니다.

```

aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole

```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```

{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}

```

Note

역할 별칭을 생성하려면 토큰 교환 IAM 역할을 AWS IoT에 전달할 권한이 있어야 합니다. 역할 별칭을 생성하려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 권한이 있는지

확인합니다. 자세한 내용은 사용 AWS Identity and Access Management 설명서의 [AWS 서비스에 역할을 전달할 수 있는 사용자 권한 부여](#)를 참조하세요.

3. Greengrass 코어 디바이스가 역할 별칭을 사용하여 토큰 교환 역할을 수입하도록 허용하는 AWS IoT 정책을 생성하고 연결합니다. 이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 역할 별칭 AWS IoT 정책을 연결할 수 있습니다. 해결 방법:

- a. (선택 사항) 역할 별칭에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-v2-iot-role-alias-policy.json
```

다음 JSON을 파일로 복사합니다.

- 리소스 ARN을 역할 별칭의 ARN으로 바꿉니다.

```
{
  "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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*를 생성할 AWS IoT 정책의 이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy  
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
  "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 사물의 인증서에 AWS IoT 정책을 연결합니다.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*를 역할 별칭 AWS IoT 정책의 이름으로 바꿉니다.
- 대상 ARN을 AWS IoT 사물 인증서의 ARN으로 바꿉니다.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청에 성공하는 경우 명령에 출력이 없습니다.

디바이스에 인증서 다운로드

앞서 디바이스의 인증서를 개발 컴퓨터에 다운로드했습니다. 이 섹션에서는 Amazon 루트 인증 기관(CA) 인증서를 다운로드합니다. 그런 다음 개발 컴퓨터와 다른 컴퓨터에서 Docker에서 AWS IoT Greengrass 코어 소프트웨어를 실행하려는 경우 인증서를 해당 호스트 컴퓨터에 복사합니다. AWS IoT Greengrass 코어 소프트웨어는 이러한 인증서를 사용하여 AWS IoT 클라우드 서비스에 연결합니다.

디바이스에 인증서를 다운로드하려면

1. 개발 컴퓨터에서 Amazon 루트 인증 기관(CA) 인증서를 다운로드합니다. AWS IoT 인증서는 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 코어 소프트웨어를 실행하려는 경우 인증서를 호스트 컴퓨터에 복사합니다. 개발 컴퓨터와 호스트 컴퓨터에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 scp 명령을 사용하여 인증서를 전송할 수 있습니다. *device-ip-address*를 호스트 컴퓨터의 IP 주소로 바꿉니다.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

구성 파일 생성

1. 호스트 컴퓨터에서 구성 파일을 배치할 폴더를 생성합니다.

```
mkdir ./greengrass-v2-config
```

2. 텍스트 편집기를 사용하여 config.yaml 폴더에 ./greengrass-v2-config라는 이름의 구성 파일을 생성합니다.

예를 들어 다음 명령을 실행하면 GNU nano를 사용하여 config.yaml을 생성할 수 있습니다.

```
nano ./greengrass-v2-config/config.yaml
```

3. 다음 YAML 콘텐츠를 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터와 Greengrass nucleus 파라미터를 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 바꿉니다.

- `/tmp/certs`. 컨테이너를 시작할 때 다운로드한 인증서를 탑재하는 Docker 컨테이너의 디렉터리입니다.
- `/greengrass/v2`. 설치에 사용하려는 Greengrass 루트 폴더입니다. GGC_ROOT 환경 변수를 사용하여 이 값을 지정할 수 있습니다.
- `MyGreengrassCore`. AWS IoT 사물의 이름입니다.
- `nucleus-version`. 설치할 AWS IoT Greengrass 코어 소프트웨어의 버전입니다. 이 값은 다운로드한 Docker 이미지 또는 Dockerfile의 버전과 일치해야 합니다. latest 태그와 함께 Greengrass Docker 이미지를 다운로드한 경우 `docker inspect image-id`를 사용하여 이미지 버전을 확인합니다.
- `region`. AWS IoT 리소스를 생성한 AWS 리전입니다. 또한 [환경 파일](#)에서 AWS_REGION 환경 변수에 대해 동일한 값을 지정해야 합니다.
- `GreengrassCoreTokenExchangeRoleAlias`. 토큰 교환 역할 별칭입니다.
- `device-data-prefix`. AWS IoT 데이터 엔드포인트의 접두사입니다.

- *device-credentials-prefix*. AWS IoT 자격 증명 엔드포인트의 접두사입니다.

환경 파일 생성

이 자습서에서는 환경 파일을 사용하여 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 관리자로 전달될 환경 변수를 설정합니다. `docker run` 명령의 [-e 또는 --env 인수](#)를 사용하여 Docker 컨테이너에서 환경 변수를 설정하거나 `docker-compose.yml` 파일의 [environment 블록](#)에서 변수를 설정할 수 있습니다.

1. 텍스트 편집기를 사용하여 `.env`라는 이름의 환경 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 현재 디렉터리에 `.env`를 생성할 수 있습니다.

```
nano .env
```

2. 다음 콘텐츠를 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 Greengrass 코어 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.
- *region*. AWS IoT 리소스를 생성한 AWS 리전입니다. [구성 파일](#)에서 `awsRegion` 구성 파라미터에 대해 동일한 값을 지정해야 합니다.
- */tmp/config/*. Docker 컨테이너를 시작할 때 구성 파일을 탑재하는 폴더입니다.

Note

`DEPLOY_DEV_TOOLS` 환경 변수를 `true`로 설정하여 Docker 컨테이너 내에서 사용자 지정 구성 요소를 개발할 수 있도록 하는 [Greengrass CLI 구성 요소](#)를 배포할 수 있습니다. 이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제

공합니다. 필요한 코어 디바이스에만 이 구성 요소를 배포하여 최소 권한 원칙을 따르세요.

컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행

이 자습서에서는 Docker 컨테이너에 빌드한 Docker 이미지를 시작하는 방법을 설명합니다. Docker CLI 또는 Docker Compose CLI를 사용하여 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 이미지를 실행할 수 있습니다.

Docker

- 이 자습서에서는 Docker 컨테이너에 빌드한 Docker 이미지를 시작하는 방법을 설명합니다.

```
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 실행](#)을 위해 다음 인수를 사용합니다.

- [--rm](#). 컨테이너가 종료될 때 컨테이너를 정리합니다.
- [--init](#). 컨테이너에서 init 프로세스를 사용합니다.

Note

--init 인수는 Docker 컨테이너를 중지할 때 AWS IoT Greengrass 코어 소프트웨어를 종료하는 데 필요합니다.

- [-it](#). (선택 사항) 전경의 Docker 컨테이너를 대화형 프로세스로 실행합니다. 이를 대신 Docker 컨테이너를 분리 모드로 실행하기 위한 `-d` 인수로 바꿀 수 있습니다. 자세한 내용은 Docker 설명서의 [Detached vs foreground](#)를 참조하세요.
- [--name](#). `aws-iot-greengrass`라는 이름의 컨테이너를 실행합니다.
- [-v](#). 볼륨을 Docker 컨테이너에 탑재하여 구성 파일과 인증서 파일을 컨테이너 내에서 AWS IoT Greengrass 실행할 수 있도록 합니다.

- [--env-file](#). (선택 사항) 환경 파일을 지정하여 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 관리자로 전달될 환경 변수를 설정합니다. 이 인수는 환경 변수를 설정하기 위해 [환경 파일](#)을 생성한 경우에만 필요합니다. 환경 파일을 생성하지 않은 경우 --env 인수를 사용하여 Docker 실행 명령에서 환경 변수를 직접 설정할 수 있습니다.
- [-p](#). (선택 사항) 8883 컨테이너 포트를 호스트 시스템에 게시합니다. AWS IoT Greengrass는 MQTT 트래픽에 포트 8883을 사용하므로 이 인수는 MQTT를 통해 연결하고 통신하려는 경우에 필요합니다. 다른 포트를 열려면 추가 -p 인수를 사용합니다.

Note

Docker 컨테이너를 강화된 보안으로 실행하려면 --cap-drop 및 --cap-add 인수를 사용하여 컨테이너에 대한 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 [런타임 권한 및 Linux 기능](#)을 참조하세요.

Docker Compose

1. 텍스트 편집기를 사용하여 docker-compose.yml이라는 이름의 Docker Compose 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 현재 디렉터리에 docker-compose.yml를 생성할 수 있습니다.

```
nano docker-compose.yml
```

Note

[GitHub](#)에서 AWS제공 Compose 파일의 최신 버전을 다운로드하여 사용할 수도 있습니다.

2. Compose 파일에 다음 콘텐츠를 추가합니다. 파일은 다음 예제와 비슷할 것입니다. *your-container-name:version*을 Docker 이미지의 이름으로 바꿉니다.

```
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 Compose 파일의 다음 파라미터는 선택 사항입니다.

- **ports** - 8883 컨테이너 포트를 호스트 시스템에 게시합니다. 이 파라미터는 MQTT 트래픽에 포트 8883을 AWS IoT Greengrass 사용하기 때문에 MQTT를 통해 연결하고 통신하려는 경우에 필요합니다.
- **env_file** - 환경 파일을 지정하여 Docker 컨테이너 내의 AWS IoT Greengrass 코어 소프트웨어 설치 관리자로 전달될 환경 변수를 설정합니다. 이 파라미터는 환경 변수를 설정하기 위해 [환경 파일](#)을 생성한 경우에만 필요합니다. 환경 파일을 생성하지 않은 경우 [환경](#) 파라미터를 사용하여 Compose 파일에서 변수를 직접 설정할 수 있습니다.

Note

Docker 컨테이너를 강화된 보안으로 실행하려면 Compose 파일의 `cap_drop` 및 `cap_add`를 사용하여 컨테이너에 대한 Linux 기능을 선택적으로 활성화할 수 있습니다. 자세한 내용은 Docker 설명서의 [런타임 권한 및 Linux 기능](#)을 참조하세요.

3. 컨테이너를 시작하려면 다음 명령을 실행합니다.

```
docker-compose -f docker-compose.yml up
```

다음 단계

AWS IoT Greengrass 코어 소프트웨어가 이제 Docker 컨테이너에서 실행 중입니다. 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 코어 소프트웨어를 탐색할 수 있습니다.

```
docker exec -it container-id /bin/bash
```

간단한 구성 요소 생성에 대한 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#)의 [4단계: 디바이스에서 구성 요소 개발 및 테스트](#) 섹션을 참조하세요.

Note

`docker exec`를 사용하여 Docker 컨테이너 내에서 명령을 실행하면 이 명령이 Docker 로그에 기록되지 않습니다. Docker 로그에 명령을 기록하려면 Docker 컨테이너에 대화형 셸을 연결합니다. 자세한 내용은 [대화형 셸을 Docker 컨테이너에 연결](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 로그 파일은 `greengrass.log` 하며에 있습니다 `/greengrass/v2/logs`. 구성 요소 로그 파일도 동일한 디렉터리에 있습니다. Greengrass 로그를 호스트의 임시 디렉터리에 복사하려면 다음 명령을 실행합니다.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

컨테이너가 종료되거나 제거된 후 로그를 유지하려면 호스트의 임시 로그 디렉터리에 전체 Greengrass 디렉터리를 탑재하는 대신 `/greengrass/v2/logs` 디렉터리만 바인딩 탑재하는 것이 좋습니다. 자세한 내용은 [Docker 컨테이너 외부에서 Greengrass 로그 유지](#) 단원을 참조하십시오.

실행 중인 AWS IoT Greengrass Docker 컨테이너를 중지하려면 `docker stop` 또는 `docker-compose -f docker-compose.yml stop`. 이 작업은 SIGTERM을 Greengrass 프로세스로 전송하고 컨테이너에서 시작된 모든 관련 프로세스를 종료합니다. Docker 컨테이너는 `docker-init` 실행 파일을 사용하여 프로세스 PID 1로 초기화되며, 이는 남아 있는 좀비 프로세스를 제거하는데 도움이 됩니다. 자세한 내용은 Docker 설명서의 [Specify an init process](#)를 참조하세요.

Docker 컨테이너에서 AWS IoT Greengrass 를 실행하는 데 발생하는 문제 해결에 대한 자세한 내용은 [Docker 컨테이너 AWS IoT Greengrass 의 문제 해결](#) 섹션을 참조하세요.

Docker 컨테이너 AWS IoT Greengrass 의 문제 해결

다음 정보를 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행 시 발생하는 문제를 해결하고 Docker 컨테이너 AWS IoT Greengrass 에서의 문제를 디버깅할 수 있습니다.

주제

- [Docker 컨테이너 실행 문제 해결](#)
- [Docker 컨테이너 AWS IoT Greengrass 에서 디버깅](#)

Docker 컨테이너 실행 문제 해결

다음 정보를 사용하여 Docker 컨테이너 AWS IoT Greengrass 에서 실행과 관련된 문제를 해결할 수 있습니다.

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.](#)
- [오류: GetAuthorizationToken 작업 직접 호출 도중 오류 발생\(AccessDeniedException\): 사용자: arn:aws:iam::account-id:user/<user-name>에게 다음을 수행할 권한 없음: ecr:GetAuthorizationToken on resource: *](#)
- [오류: 풀 속도 제한에 도달함](#)

오류: TTY가 아닌 장치에서 대화형 로그인을 수행할 수 없습니다.

`aws ecr get-login-password` 명령을 실행할 때 이 오류가 발생할 수 있습니다. 최신 AWS CLI 버전 2 또는 버전 1을 설치했는지 확인합니다. AWS CLI 버전 2를 사용하는 것이 좋습니다. 자세한 내용은 AWS Command Line Interface 사용자 가이드에서 [AWS CLI설치](#)를 참조하세요.

오류: 알 수 없는 옵션: no-include-email

`aws ecr get-login` 명령을 실행할 때 이 오류가 발생할 수 있습니다. 최신 AWS CLI 버전이 설치되어 있는지 확인합니다(예: 실행: `pip install awscli --upgrade --user`). 자세한 내용은 AWS Command Line Interface 사용 설명서 [의 Microsoft Windows AWS Command Line Interface 에 설치](#)를 참조하세요.

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.

오류: GetAuthorizationToken 작업 직접 호출 도중 오류 발생(AccessDeniedException):

사용자: arn:aws:iam::**account-id**:user/<user-name>에게 다음을 수행할 권한 없음:

ecr:GetAuthorizationToken on resource: *

Amazon ECR 리포지토리에 액세스할 충분한 권한이 없는데 `aws ecr get-login-password` 명령을 실행할 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 Amazon ECR 사용 설명서의 [Amazon ECR 리포지토리 정책 예제](#) 및 [하나의 Amazon ECR 리포지토리에 액세스](#)를 참조하세요.

오류: 풀 속도 제한에 도달함

Docker Hub는 익명 및 Free Docker Hub 사용자가 수행할 수 있는 풀 요청 수를 제한합니다. 익명 또는 무료 사용자 풀 요청의 속도 제한을 초과하면 다음 오류 중 하나가 발생합니다.

ERROR: toomanyrequests: Too Many Requests.

You have reached your pull rate limit.

이러한 오류를 해결하려면 몇 시간 기다린 다음 다른 풀 요청을 시도할 수 있습니다. 많은 수의 풀 요청을 일관적으로 제출할 계획인 경우 [Docker Hub 웹 사이트](#)에서 속도 제한, 그리고 Docker 계정 인증 및 업그레이드 옵션 관련 정보를 참조하세요.

Docker 컨테이너 AWS IoT Greengrass 에서 디버깅

도커 컨테이너로 문제를 디버그하려면 Greengrass 런타임 로그를 유지하거나 대화형 셸을 도커 컨테이너에 연결할 수 있습니다.

Docker 컨테이너 외부에서 Greengrass 로그 유지

AWS IoT Greengrass 컨테이너를 중지한 후 다음 `docker cp` 명령을 사용하여 Greengrass 로그를 Docker 컨테이너에서 임시 로그 디렉터리로 복사할 수 있습니다.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

컨테이너가 종료되거나 제거된 후에도 로그를 유지하려면 `/greengrass/v2/logs` 디렉터리를 바인드 마운트한 후 AWS IoT Greengrass Docker 컨테이너를 실행해야 합니다.

`/greengrass/v2/logs` 디렉터리를 바인드 마운트하려면 새 AWS IoT Greengrass Docker 컨테이너를 실행할 때 다음 중 하나를 수행합니다.

- `docker run` 명령에 `-v /tmp/logs:/greengrass/v2/logs:ro`를 포함합니다.

`docker-compose up` 명령을 실행하기 전에 다음 줄을 포함하도록 Compose 파일의 `volumes` 블록을 수정합니다.

```
volumes:
  - /tmp/logs:/greengrass/v2/logs:ro
```

그런 다음 호스트 `/tmp/logs`의에서 로그를 확인하여 AWS IoT Greengrass 가 Docker 컨테이너 내에서 실행되는 동안 Greengrass 로그를 볼 수 있습니다.

Greengrass Docker 컨테이너 실행에 대한 자세한 내용은 [수동 프로비저닝을 사용하여 Docker AWS IoT Greengrass 에서 실행](#) 및 [자동 프로비저닝을 사용하여 Docker AWS IoT Greengrass 에서 실행](#) 섹션을 참조하세요.

대화형 셸을 Docker 컨테이너에 연결

`docker exec`를 사용하여 Docker 컨테이너 내에서 명령을 실행하면 이 명령이 Docker 로그에 캡처되지 않습니다. Docker 로그에 명령을 로깅하면 Greengrass Docker 컨테이너의 상태를 조사하는 데 도움이 될 수 있습니다. 다음 중 하나를 수행하세요.

- 별도의 터미널에서 다음 명령을 실행하여 터미널의 표준 입력, 출력 및 오류를 실행 중인 컨테이너에 연결합니다. 이를 통해 현재 터미널에서 Docker 컨테이너를 보고 제어할 수 있습니다.

```
docker attach container-id
```

- 별도의 터미널에서 다음 명령을 실행합니다. 이렇게 하면 컨테이너가 연결되지 않은 경우에도 대화형 모드에서 명령을 실행할 수 있습니다.

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

일반적인 AWS IoT Greengrass 문제 해결은 [섹션을 참조하세요](#) [문제 해결](#).

AWS IoT Greengrass 코어 소프트웨어 구성

AWS IoT Greengrass 코어 소프트웨어는 소프트웨어를 구성하는 데 사용할 수 있는 옵션을 제공합니다. 배포를 생성하여 각 AWS IoT Greengrass 코어 디바이스에서 코어 소프트웨어를 구성할 수 있습니다.

주제

- [Greengrass nucleus 구성 요소 배포](#)
- [Greengrass nucleus를 시스템 서비스로 구성](#)
- [JVM 옵션으로 메모리 할당 제어](#)
- [구성 요소를 실행하는 사용자 구성](#)
- [구성 요소에 대한 시스템 리소스 제한 구성](#)
- [포트 443에서 또는 네트워크 프록시를 통해 연결](#)
- [프라이빗 CA에서 서명한 디바이스 인증서 사용](#)
- [MQTT 제한 시간 및 캐시 설정 구성](#)
- [IPv6 네트워크에서 Greengrass Nucleus 구성](#)

Greengrass nucleus 구성 요소 배포

AWS IoT Greengrass 는 Greengrass AWS IoT Greengrass 코어 디바이스에 배포할 수 있는 구성 요소로 코어 소프트웨어를 제공합니다. 배포를 생성하여 여러 Greengrass 코어 디바이스에 동일한 구성을 적용할 수 있습니다. 자세한 내용은 [Greengrass nucleus](#) 및 [AWS IoT Greengrass 코어 소프트웨어 \(OTA\) 업데이트](#) 섹션을 참조하세요.

Greengrass nucleus를 시스템 서비스로 구성

다음은 수행하려면 AWS IoT Greengrass 코어 소프트웨어를 디바이스의 init 시스템에서 시스템 서비스로 구성해야 합니다.

- 디바이스가 부팅되면 AWS IoT Greengrass 코어 소프트웨어를 시작합니다. 대규모 디바이스 플릿을 관리하는 경우에 좋습니다.
- 플러그인 구성 요소를 설치하고 실행합니다. AWS 여러 제공 구성 요소는 플러그인 구성 요소로, Greengrass nucleus와 직접 인터페이스할 수 있습니다. 구성 요소 유형에 대한 자세한 정보는 [구성 요소 유형](#)을(를) 참조하세요.

- 코어 디바이스의 AWS IoT Greengrass 코어 소프트웨어에 over-the-air(OTA) 업데이트를 적용합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#) 단원을 참조하십시오.
- 배포가 구성 요소를 새 버전으로 업데이트하거나 특정 구성 파라미터를 업데이트할 때 구성 요소가 AWS IoT Greengrass 코어 소프트웨어 또는 코어 디바이스를 다시 시작하도록 활성화합니다. 자세한 내용은 [부트스트랩 수명 주기 단계](#)를 참조하세요.

⚠ Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

주제

- [nucleus를 시스템 서비스로 구성\(Linux\)](#)
- [nucleus를 시스템 서비스로 구성\(Windows\)](#)

nucleus를 시스템 서비스로 구성(Linux)

Linux 디바이스에서는 다양한 init 시스템이 지원됩니다(예: initd, systemd, SystemV). AWS IoT Greengrass 코어 소프트웨어를 설치할 때 `--setup-system-service true` 인수를 사용하여 nucleus를 시스템 서비스로 시작하고 디바이스가 부팅될 때 시작하도록 구성합니다. 설치 관리자는 AWS IoT Greengrass 코어 소프트웨어를 systemd를 사용하는 시스템 서비스로 구성합니다.

시스템 서비스로 실행되도록 nucleus를 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 디바이스 시작을 구성하고 AWS IoT Greengrass 코어 소프트웨어를 시작하거나 중지할 수 있습니다.

- 서비스 상태를 확인하려면(systemd)

```
sudo systemctl status greengrass.service
```

- 디바이스가 부팅될 때 시작되도록 nucleus를 활성화하는 방법입니다.

```
sudo systemctl enable greengrass.service
```

- 디바이스가 부팅될 때 nucleus 시작을 중지하는 방법입니다.

```
sudo systemctl disable greengrass.service
```

- AWS IoT Greengrass 코어 소프트웨어를 시작하는 방법.

```
sudo systemctl start greengrass.service
```

- AWS IoT Greengrass 코어 소프트웨어를 중지합니다.

```
sudo systemctl stop greengrass.service
```

nucleus를 시스템 서비스로 구성(Windows)

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 `--setup-system-service true` 인수를 사용하여 nucleus를 Windows 서비스로 시작하고 디바이스가 부팅될 때 시작되도록 구성합니다.

서비스를 구성한 후 다음 명령을 실행하여 부팅 시 디바이스 시작을 구성하고 AWS IoT Greengrass 코어 소프트웨어를 시작하거나 중지할 수 있습니다. 이러한 명령을 실행하려면 관리자로 명령 프롬프트 또는 PowerShell을 실행해야 합니다.

Windows Command Prompt (CMD)

- 서비스 상태를 확인하려면

```
sc query "greengrass"
```

- 디바이스가 부팅될 때 시작되도록 nucleus를 활성화하는 방법입니다.

```
sc config "greengrass" start=auto
```

- 디바이스가 부팅될 때 nucleus 시작을 중지하는 방법입니다.

```
sc config "greengrass" start=disabled
```

- AWS IoT Greengrass 코어 소프트웨어를 시작하는 방법.

```
sc start "greengrass"
```

- AWS IoT Greengrass 코어 소프트웨어를 중지합니다.

```
sc stop "greengrass"
```

Note

Windows 디바이스에서 AWS IoT Greengrass 코어 소프트웨어는 Greengrass 구성 요소 프로세스를 종료하는 동안 종료 신호를 무시합니다. 이 명령을 실행할 때 AWS IoT Greengrass 코어 소프트웨어가 종료 신호를 무시하는 경우 몇 초 정도 기다렸다가 다시 시도하세요.

PowerShell

- 서비스 상태를 확인하려면

```
Get-Service -Name "greengrass"
```

- 디바이스가 부팅될 때 시작되도록 nucleus를 활성화하는 방법입니다.

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- 디바이스가 부팅될 때 nucleus 시작을 중지하는 방법입니다.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- AWS IoT Greengrass 코어 소프트웨어를 시작하는 방법.

```
Start-Service -Name "greengrass"
```

- AWS IoT Greengrass 코어 소프트웨어를 중지합니다.

```
Stop-Service -Name "greengrass"
```

Note

Windows 디바이스에서 AWS IoT Greengrass 코어 소프트웨어는 Greengrass 구성 요소 프로세스를 종료하는 동안 종료 신호를 무시합니다. 이 명령을 실행할 때 AWS IoT Greengrass 코어 소프트웨어가 종료 신호를 무시하는 경우 몇 초 정도 기다렸다가 다시 시도하세요.

JVM 옵션으로 메모리 할당 제어

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 Java 가상 머신(JVM) 옵션을 사용하여 AWS IoT Greengrass 코어 소프트웨어가 사용하는 메모리 양을 제어하는 최대 힙 크기, 가비지 수집 모드 및 컴파일러 옵션을 제어할 수 있습니다. JVM의 힙 크기에 따라 [폐영역 회수](#) 발생 전 또는 애플리케이션의 메모리 부족 전에 애플리케이션에서 사용될 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 새 배포를 생성하거나 nucleus 구성 요소가 포함된 기존 배포를 수정하고 [nucleus 구성 요소 구성](#)의 `jvmOptions` 구성 파라미터에서 JVM 옵션을 지정합니다.

요구 사항에 따라 메모리 할당을 줄이거나 최소 메모리 할당으로 AWS IoT Greengrass 코어 소프트웨어를 실행할 수 있습니다.

줄어든 메모리 할당

메모리 할당이 감소된 AWS IoT Greengrass 코어 소프트웨어를 실행하려면 다음 예제 구성 병합 업데이트를 사용하여 nucleus 구성에서 JVM 옵션을 설정하는 것이 좋습니다.

```
{
  "jvmOptions": "-XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

최소 메모리 할당

최소 메모리 할당으로 AWS IoT Greengrass 코어 소프트웨어를 실행하려면 다음 예제 구성 병합 업데이트를 사용하여 nucleus 구성에서 JVM 옵션을 설정하는 것이 좋습니다.

```
{
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
}
```

Important

최소 메모리 할당으로 AWS IoT Greengrass 코어 소프트웨어를 실행하면 더 적은 메모리를 사용할 때 JVM이 더 많은 처리를 수행하기 때문에 사양이 낮은 시스템에 상당한 성능 영향을 미칠 수 있습니다. 메모리와 필요한 성능의 균형이 유지되도록 옵션을 조정하는 것이 좋습니다.

이러한 예제 구성 병합 업데이트에서는 다음 JVM 옵션이 사용됩니다.

-XX:+UseSerialGC

JVM 힙 스페이스에 직렬 폐영역 회수가 사용되도록 지정합니다. 직렬 폐영역 회수기는 느린 편이지만, 다른 JVM 폐영역 회수 구현보다 메모리가 적게 사용됩니다.

-XX:TieredStopAtLevel=1

Java JIT(Just-In-Time) 컴파일러가 한 번 사용되도록 JVM에 지시합니다. JIT 방식으로 컴파일된 코드에서는 디바이스 메모리의 스페이스가 사용되므로 JIT 컴파일러를 두 번 이상 사용하면 단일 컴파일보다 많은 메모리가 소비됩니다.

-XmxNNm

최대 JVM 힙 크기를 설정합니다.

Important

최대 힙 크기를 너무 낮게 설정하면 성능이 느려지거나 메모리 부족 오류가 발생할 수 있습니다. **-XmxNNm** 옵션으로 최대 크기를 설정하기 전에 현재 힙 사용량을 측정하는 것이 좋습니다. JVM 옵션으로 **-XX:NativeMemoryTracking=detail** JVM을 구성합니다. 그런 다음에 [jcmd 유틸리티](#) 내의 `VM.native_memory` 명령 요청을 사용하여 현재 힙 사용량을 측정합니다.

힙 측정이 옵션이 아니라면 -Xmx64m을 시작 값으로 사용하여 힙 크기를 64MB로 제한합니다. 그런 다음에는 최대 힙 크기를 점진적으로 줄일 수 있습니다. 최소 메모리 할당의 경우 -Xmx32m을 시작 값으로 사용하여 힙 크기를 32MB로 제한합니다.

실제 요구 사항에 따라 -Xmx 값을 늘리거나 줄일 수 있지만, 최대 힙 크기를 16MB 미만으로 설정하지 않는 것이 좋습니다. 필요한 JVM 힙 크기의 양은 코어 디바이스에 배포된 플러그인 구성 요소에 따라 시간 경과에 따라 달라질 수 있습니다. 최대 힙 크기가 환경에 비해 너무 낮으면 메모리 부족으로 인해 AWS IoT Greengrass 코어 소프트웨어에 예기치 않은 오류가 발생할 수 있습니다. 부족한 메모리 때문에 성능이 느려지거나 오류가 발생하면 알려진 좋은 설정으로 되돌립니다. 예를 들어, 일반적인 커밋 힙 크기가 41428KB라면 -Xmx40m을 사용하여 힙 사용량을 약간 제한합니다.

-Xint

JIT(Just-In-Time) 컴파일러가 사용되지 않도록 JVM에 지시합니다. 그 대신에 해석 전용 모드에서 JVM이 실행됩니다. 이 모드는 JIT 방식으로 컴파일된 코드가 실행되는 것보다 느립니다(저사양 시스템의 배포는 20배 느릴 수 있음). 그러나 컴파일된 코드에서는 메모리의 스페이스가 사용되지 않습니다.

구성 병합 파일 업데이트 생성에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

구성 요소를 실행하는 사용자 구성

AWS IoT Greengrass 코어 소프트웨어는 시스템 사용자 및 소프트웨어를 실행하는 그룹과 다른 그룹으로 구성 요소 프로세스를 실행할 수 있습니다. 이렇게 하면 AWS IoT Greengrass 코어 디바이스에서 실행되는 구성 요소에 권한을 부여하지 않고도 코어 소프트웨어를 루트 또는 관리자 사용자로 실행할 수 있으므로 보안이 향상됩니다.

다음 표에는 AWS IoT Greengrass 코어 소프트웨어가 사용자가 지정한 사용자로 실행할 수 있는 구성 요소의 유형이 나와 있습니다. 자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

구성 요소 유형	구성 요소 사용자 구성
nucleus	 아니요

구성 요소 유형	구성 요소 사용자 구성
플러그인	 아니요
일반	 예
Lambda(컨테이너화되지 않았음)	 예
Lambda(컨테이너화되었음)	 예

배포 구성에서 구성 요소 사용자를 지정하려면 먼저 구성 요소 사용자를 생성해야 합니다. Windows 기반 디바이스에서는 LocalSystem 계정의 자격 증명 관리자 인스턴스에서 사용자의 사용자 이름과 암호도 저장해야 합니다. 자세한 내용은 [Windows 디바이스의 구성 요소 사용자 설정](#) 단원을 참조하십시오.

Linux 기반 디바이스에서 구성 요소 사용자를 구성할 때 선택적으로 그룹을 지정할 수도 있습니다.

user:group 형식으로 사용자와 그룹을 콜론(:)으로 구분하여 지정합니다. 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 기본적으로 사용자의 기본 그룹으로 설정됩니다. 이름 또는 ID를 사용하여 사용자와 그룹을 식별할 수 있습니다.

Linux 기반 디바이스에서는 알 수 없는 사용자라고도 하는 존재하지 않는 시스템 사용자로 구성 요소를 실행하여 보안을 강화할 수도 있습니다. Linux 프로세스에서는 동일한 사용자가 실행하는 다른 프로세스에 신호를 보낼 수 있습니다. 알 수 없는 사용자는 다른 프로세스를 실행하지 않으므로 알 수 없는 사용자로 구성 요소를 실행하여 구성 요소에서 코어 디바이스의 다른 구성 요소에 신호를 보내지 못하도록 할 수 있습니다. 알 수 없는 사용자로 구성 요소를 실행하려면 코어 디바이스에 존재하지 않는 사용자 ID를 지정합니다. 알 수 없는 그룹으로 실행되도록 존재하지 않는 그룹 ID를 지정할 수도 있습니다.

구성 요소와 코어 디바이스마다 사용자를 구성할 수 있습니다.

- 구성 요소에 대해 구성

각 구성 요소를 해당 구성 요소에 지정된 사용자를 통해 실행되도록 구성할 수 있습니다. 배포를 생성할 때 해당 구성 요소에 대한 `runWith` 구성에서 구성 요소마다 사용자를 지정할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어는 구성된 구성 요소를 지정된 사용자로 실행합니다. 그렇지 않으면 코어 디바이스에 대해 구성하는 기본 사용자로 구성 요소가 실행되도록 기본적으로 설정됩니다. 배포 구성에서 구성 요소 사용자를 지정하는 방법에 대한 자세한 내용은 [배포 만들기](#)의 [runWith](#) 구성 파라미터를 참조하세요.

- 코어 디바이스의 기본 사용자 구성

AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하는 데 사용하는 기본 사용자를 구성할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행할 때 해당 구성 요소에 사용자를 지정했는지 확인하고 이를 사용하여 구성 요소를 실행합니다. 구성 요소가 사용자를 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 코어 디바이스에 대해 구성된 기본 사용자로 구성 요소를 실행합니다. 자세한 내용은 [기본 구성 요소 사용자 구성](#) 단원을 참조하십시오.

Note

Windows 기반 디바이스에서는 구성 요소를 실행할 기본 사용자를 하나 이상 지정해야 합니다.

Linux 기반 디바이스에서는 구성 요소를 실행할 사용자를 구성하지 않으면 다음 고려 사항이 적용됩니다.

- AWS IoT Greengrass 코어 소프트웨어를 루트로 실행하면 소프트웨어가 구성 요소를 실행하지 않습니다. 루트로 실행하는 경우 구성 요소를 실행할 기본 사용자를 지정해야 합니다.
- 루트가 아닌 사용자로 AWS IoT Greengrass 코어 소프트웨어를 실행하면 소프트웨어가 해당 사용자로 구성 요소를 실행합니다.

주제

- [Windows 디바이스의 구성 요소 사용자 설정](#)
- [기본 구성 요소 사용자 구성](#)

Windows 디바이스의 구성 요소 사용자 설정

Windows 기반 디바이스의 구성 요소 사용자 설정 방법

1. 디바이스의 LocalSystem 계정에서 구성 요소 사용자를 생성합니다.

```
net user /add component-user password
```

2. [Microsoft의 PsExec 유틸리티](#)를 사용하여 구성 요소 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다.

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

Note

Windows 기반 디바이스의 LocalSystem 계정에서는 Greengrass nucleus가 실행되며, PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

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 코어 소프트웨어를 설치할 때 기본 사용자를 설정할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

aws.greengrass.Nucleus 구성 요소에 대해 다음 구성 업데이트가 지정되는 [배포를 생성](#)합니다.

Linux

```
{
  "runWithDefault": {
```

```

    "posixUser": "ggc_user:ggc_group"
  }
}

```

Windows

```

{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}

```

Note

지정한 사용자가 존재해야 하며, 이 사용자의 사용자 이름과 암호는 Windows 디바이스에 있는 LocalSystem 계정의 Credential Manager 인스턴스에 저장되어야 합니다. 자세한 내용은 [Windows 디바이스의 구성 요소 사용자 설정](#) 단원을 참조하십시오.

다음 예제에서는 ggc_user는 기본 사용자, ggc_group은 기본 그룹으로 구성되는 Linux 기반 디바이스에 대한 배포가 정의됩니다. merge 구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```

{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}

```

구성 요소에 대한 시스템 리소스 제한 구성

Note

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다.

시스템 리소스 제한이 지원되는 구성 요소의 유형이 다음 표에 나와 있습니다. 자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

구성 요소 유형	시스템 리소스 제한 구성
nucleus	 아니요
플러그인	 아니요
일반	 예
Lambda(컨테이너화되지 않았음)	 예
Lambda(컨테이너화되었음)	 아니요

⚠ Important

[Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행할](#) 때는 시스템 리소스 제한이 지원되지 않습니다.

구성 요소와 코어 디바이스마다 시스템 리소스 제한을 구성할 수 있습니다.

- 구성 요소에 대해 구성

해당 구성 요소에 지정된 시스템 리소스 제한으로 각 구성 요소를 구성할 수 있습니다. 배포를 생성할 때 배포의 각 구성 요소에 대한 시스템 리소스 제한을 지정할 수 있습니다. 구성 요소에서 시스템 리소스 제한이 지원되면 AWS IoT Greengrass 코어 소프트웨어에서는 구성 요소의 프로세스에 제한이 적용됩니다. 구성 요소에 대한 시스템 리소스 제한을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 코어 디바이스에 대해 구성된 기본값을 사용합니다. 자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

- 코어 디바이스의 기본값 구성

AWS IoT Greengrass 코어 소프트웨어가 이러한 제한을 지원하는 구성 요소에 적용하는 기본 시스템 리소스 제한을 구성할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하면 해당 구성 요소에 대해 지정한 시스템 리소스 제한이 적용됩니다. 해당 구성 요소가 시스템 리소스 제한을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 코어 디바이스에 대해 구성된 기본 시스템 리소스 제한을 적용합니다. 기본 시스템 리소스 제한을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 기본적으로 시스템 리소스 제한을 적용하지 않습니다. 자세한 내용은 [기본 시스템 리소스 제한 구성](#) 단원을 참조하십시오.

기본 시스템 리소스 제한 구성

[Greengrass nucleus 구성 요소](#)를 배포하여 코어 디바이스에 대한 기본 시스템 리소스 제한을 구성할 수 있습니다. 기본 시스템 리소스 제한을 구성하려면 `aws.greengrass.Nucleus` 구성 요소에 다음 구성 업데이트가 지정되는 [배포를 생성](#)합니다.

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

```
}
```

다음 예제에서는 CPU 시간 제한이 CPU 코어 4개 디바이스의 50% 사용량에 해당하는 2로 구성되는 배포를 정의합니다. 이 예제에서는 메모리 사용량을 100MB로 구성합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "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. 다른 제한적 환경에서는 모든 트래픽이 인터넷 연결 전에 프록시를 경유해야 할 수도 있습니다.

Note

Greengrass [nucleus 구성 요소 v2.0.3 이하를 실행하는 Greengrass](#) 코어 디바이스는 포트 8443을 사용하여 AWS IoT Greengrass 데이터 영역 엔드포인트에 연결합니다. 이러한 디바이스에서는 포트 8443에서 이 엔드포인트에 연결할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

이러한 시나리오에서 통신을 활성화하기 위해서는 다음 구성 옵션을 AWS IoT Greengrass 제공합니다.

- 포트 443을 통한 MQTT 통신입니다. 네트워크에서 포트 443에 대한 연결이 허용되면 MQTT 트래픽에 기본 포트 8883 대신에 포트 443이 사용되도록 Greengrass 코어를 구성할 수 있습니다. 이는 포트 443으로 직접 연결되거나 네트워크 프록시 서버를 통해 연결되는 것일 수 있습니다. 인증서 기반

클라이언트 인증이 사용되는 기본 구성과 달리 포트 443의 MQTT에서는 인증에 [디바이스 서비스 역할](#)이 사용됩니다.

자세한 내용은 [포트 443을 통해 MQTT 구성](#) 단원을 참조하십시오.

- 포트 443을 통한 HTTPS 통신입니다. AWS IoT Greengrass 코어 소프트웨어는 기본적으로 포트 8443을 통해 HTTPS 트래픽을 전송하지만 포트 443을 사용하도록 구성할 수 있습니다. AWS IoT Greengrass 는 [Application Layer Protocol Network](#)(ALPN) TLS 확장을 사용하여 연결을 활성화합니다. 기본 구성과 마찬가지로 포트 443의 HTTPS에서는 인증서 기반 클라이언트 인증이 사용됩니다.

Important

ALPN을 사용하고 포트 443을 통한 HTTPS 통신을 활성화하려면 코어 디바이스에서 Java 8 업데이트 252 이상이 실행되어야 합니다. Java 버전 9 이상의 모든 업데이트에서는 ALPN도 지원됩니다.

자세한 내용은 [포트 443을 통해 HTTPS 구성](#) 단원을 참조하십시오.

- 네트워크 프록시를 통한 연결. Greengrass 코어 코어가 연결되는 중개자로 작동하도록 네트워크 프록시 서버를 구성할 수 있습니다. AWS IoT Greengrass 에서는 HTTP 및 HTTPS 프록시에 대한 기본 인증이 지원됩니다.

Greengrass 코어 디바이스에서 HTTPS 프록시가 사용되려면 [Greengrass nucleus](#) v2.5.0 이상을 실행해야 합니다.

AWS IoT Greengrass 코어 소프트웨어는 ALL_PROXY, HTTP_PROXYHTTPS_PROXY, 및 NO_PROXY 환경 변수를 통해 프록시 구성을 구성 요소에 전달합니다. 프록시를 통해 연결되려면 구성 요소에서 이러한 설정이 사용되어야 합니다. 일반적으로 이러한 환경 변수가 연결을 만드는 기본값으로 사용되는 공통 라이브러리(예: boto3, cURL 및 python requests 패키지)가 구성 요소에서 사용됩니다. 구성 요소에서 이러한 환경 변수도 지정되는 경우 AWS IoT Greengrass 에서 재정의되지 않습니다.

자세한 내용은 [네트워크 프록시 구성](#) 단원을 참조하십시오.

포트 443을 통해 MQTT 구성

기존 코어 디바이스의 포트 443을 통하거나 새 코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 MQTT를 구성할 수 있습니다.

주제

- [기존 코어 디바이스의 포트 443을 통해 MQTT 구성](#)
- [설치 중 포트 443을 통해 MQTT 구성](#)

기존 코어 디바이스의 포트 443을 통해 MQTT 구성

배포를 사용하여 단일 코어 디바이스 또는 코어 디바이스 그룹의 포트 443을 통해 MQTT를 구성할 수 있습니다. 이 배포에서는 [nucleus 구성 요소](#) 구성을 업데이트합니다. mqtt 구성을 업데이트하면 nucleus가 다시 시작됩니다.

포트 443을 통해 MQTT를 구성하려면 aws.greengrass.Nucleus 구성 요소에 다음 구성 업데이트가 지정되는 [배포를 생성](#)합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

다음 예제에서는 포트 443을 통해 MQTT가 구성되는 배포를 정의합니다. merge 구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

설치 중 포트 443을 통해 MQTT 구성

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 포트 443을 통해 MQTT를 구성할 수 있습니다. --init-config 설치 프로그램 인수를 사용하여 포트 443을 통해 MQTT를 구성합니다. [수동 프로비저닝](#), [플릿 프로비저닝](#) 또는 [사용자 지정 프로비저닝](#)으로 설치할 때 이 인수를 지정할 수 있습니다.

포트 443을 통해 HTTPS 구성

이 기능에는 [Greengrass nucleus](#) v2.0.4 이상이 필요합니다.

기존 코어 디바이스의 포트 443을 통하거나 새 코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 HTTPS를 구성할 수 있습니다.

주제

- [기존 코어 디바이스의 포트 443을 통해 HTTPS 구성](#)
- [설치 중 포트 443을 통해 HTTPS 구성](#)

기존 코어 디바이스의 포트 443을 통해 HTTPS 구성

배포를 사용하여 단일 코어 디바이스 또는 코어 디바이스 그룹의 포트 443을 통해 HTTPS를 구성할 수 있습니다. 이 배포에서는 [nucleus 구성 요소](#) 구성을 업데이트합니다.

포트 443을 통해 HTTPS를 구성하려면 `aws.greengrass.Nucleus` 구성 요소에 다음 구성 업데이트가 지정되는 [배포를 생성](#)합니다.

```
{
  "greengrassDataPlanePort": 443
}
```

다음 예제에서는 포트 443을 통해 HTTPS가 구성되는 배포를 정의합니다. `merge` 구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

설치 중 포트 443을 통해 HTTPS 구성

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 포트 443을 통해 HTTPS를 구성할 수 있습니다. `--init-config` 설치 프로그램 인수를 사용하여 포트 443을 통해 HTTPS를 구성합

니다. [수동 프로비저닝](#), [플릿 프로비저닝](#) 또는 [사용자 지정 프로비저닝](#)으로 설치할 때 이 인수를 지정할 수 있습니다.

네트워크 프록시 구성

이 섹션의 절차에 따라 HTTP 또는 HTTPS 네트워크 프록시를 통해 인터넷에 연결되도록 Greengrass 코어 디바이스를 구성합니다. 코어 디바이스에서 사용되는 엔드포인트 및 포트에 대한 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 섹션을 참조하세요.

Important

코어 디바이스에서 v2.4.0 이전 버전의 [Greengrass nucleus](#)가 실행되는 경우 네트워크 프록시를 사용하려면 다음 권한이 디바이스의 역할에서 허용되어야 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 디바이스는 MQTT를 사용하여에서 배포를 수신하고 설치 AWS 클라우드하므로 역할에 대한 이러한 권한을 정의하지 않으면 디바이스가 작동하지 않습니다. 일반적으로 디바이스에서는 MQTT 연결 인증에 X.509 인증서가 사용되지만, 디바이스에서 프록시가 사용될 때는 이 방식으로 인증될 수 없습니다.

디바이스 역할을 구성하는 방법에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

주제

- [기존 코어 디바이스에서 네트워크 프록시 구성](#)
- [설치 중 네트워크 프록시 구성](#)
- [HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정](#)
- [networkProxy 객체](#)

기존 코어 디바이스에서 네트워크 프록시 구성

배포를 사용하여 단일 코어 디바이스 또는 코어 디바이스 그룹에서 네트워크 프록시를 구성할 수 있습니다. 이 배포에서는 [nucleus 구성 요소](#) 구성을 업데이트합니다. networkProxy 구성을 업데이트하면 nucleus가 다시 시작됩니다.

네트워크 프록시를 구성하려면 다음 구성 업데이트가 병합되는 aws.greengrass.Nucleus 구성 요소에 대한 [배포를 생성](#)합니다. 이 구성 업데이트에는 [networkProxy 객체](#)가 있습니다.

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

다음 예제에서는 네트워크 프록시가 구성되는 배포를 정의합니다. merge 구성 업데이트에는 직렬화된 JSON 객체가 필요합니다.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "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 중 네트워크 프록시 구성

코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 네트워크 프록시를 구성할 수 있습니다. --init-config 설치 프로그램 인수를 사용하여 네트워크 프록시를 구성합니다. [수동 프로비저닝](#), [플릿 프로비저닝](#) 또는 [사용자 지정 프로비저닝](#)으로 설치할 때 이 인수를 지정할 수 있습니다.

HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정

HTTPS 프록시가 사용되도록 코어 디바이스를 구성할 때는 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스에 프록시 서버 인증서 체인을 추가해야 합니다. 그렇지 않으면 코어 디바이스에서 프록시를 통해 트래픽을 라우팅하려고 할 때 오류가 발생할 수 있습니다. 코어 디바이스의 Amazon 루트 CA 인증서 파일에 프록시 서버 CA 인증서를 추가합니다.

HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스를 설정하는 방법

1. 코어 디바이스에서 Amazon 루트 CA 인증서 파일을 찾습니다.

- [자동 프로비저닝](#)과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치한 경우 Amazon 루트 CA 인증서 파일은 `/greengrass/v2/rootCA.pem`에 있습니다.
- [수동](#) 또는 [플릿 프로비저닝](#)으로 AWS IoT Greengrass 코어 소프트웨어를 설치한 경우 Amazon 루트 CA 인증서 파일이 `/greengrass/v2/AmazonRootCA1.pem`에 있을 수 있습니다.

Amazon 루트 CA 인증서가 이러한 위치에 없는 경우 `/greengrass/v2/config/effectiveConfig.yaml`에서 `system.rootCaPath` 속성을 확인하여 해당 위치를 찾습니다.

2. Amazon 루트 CA 인증서 파일에 프록시 서버 CA 인증서의 내용을 추가합니다.

다음 예제에서는 Amazon 루트 CA 인증서 파일에 추가된 프록시 서버 CA 인증서를 보여줍니다.

```
-----BEGIN CERTIFICATE-----
MIIEFTCCA v2gAwIQWgIVAMHSAzWG/5YVRYtRQ0xXUTEpHuEmApzGCSqGSIb3DQEK
\ nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBjb216
... content of proxy CA certificate ...
+vHIR1t0e5JAm5\ noTIZGoFbK82A0/ n07f/ t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPULGk1gCE84a0X\ n7Rp/ lND/ PuMZ/ s8Yj1kY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/ m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPm1jZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QDExBBKw
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+ymRQ+hDaXJioblXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

networkProxy 객체

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

scheme://userinfo@host:port 형식으로 표시되는 프록시 서버의 URL입니다.

- scheme - 체계로, http 또는 https여야 합니다.

Important

Greengrass 코어 디바이스에서 HTTPS 프록시가 사용되려면 [Greengrass nucleus v2.5.0](#) 이상을 실행해야 합니다.

HTTPS 프록시를 구성하는 경우 코어 디바이스의 Amazon 루트 CA 인증서에 프록시 서버 CA 인증서를 추가해야 합니다. 자세한 내용은 [HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정](#) 단원을 참조하십시오.

- userinfo - (선택 사항) 사용자 이름 및 암호 정보입니다. url에서 이 정보를 지정하면 Greengrass 코어 디바이스에서는 username 및 password 필드를 무시합니다.
- host - 프록시 서버의 호스트 이름 또는 IP 주소입니다.
- port - (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 디바이스에서는 다음 기본값이 사용됩니다.
 - http - 80
 - https - 443

username

(선택 사항) 프록시 서버를 인증하는 사용자 이름입니다.

password

(선택 사항) 프록시 서버를 인증하는 암호입니다.

프라이빗 CA에서 서명한 디바이스 인증서 사용

사용자 지정 프라이빗 인증 기관(CA)을 사용하는 경우 Greengrass nucleus의 **greengrassDataPlaneEndpoint**를 **iotdata**로 설정해야 합니다. **--init-config** [설치 프로그램 인수](#)를 사용하여 배포 또는 설치 중에 이 옵션을 설정할 수 있습니다.

디바이스가 연결되는 Greengrass 데이터 플레인 엔드포인트를 사용자 지정할 수 있습니다. 이 구성 옵션을 **iotdata**로 설정하여 **iotDataEndpoint**를 통해 지정할 수 있는 IoT 데이터 엔드포인트와 동일한 엔드포인트로 Greengrass 데이터 플레인 엔드포인트를 설정할 수 있습니다.

MQTT 제한 시간 및 캐시 설정 구성

환경에서 AWS IoT Greengrass 구성 요소는 MQTT를 사용하여와 통신할 수 있습니다 AWS IoT Core. AWS IoT Greengrass 코어 소프트웨어는 구성 요소에 대한 MQTT 메시지를 관리합니다. 코어 디바이스에서 AWS 클라우드연결이 끊어지면 소프트웨어에서는 나중에 연결이 복원될 때 다시 시도할 MQTT 메시지가 캐싱됩니다. 메시지 제한 시간 및 캐시 크기와 같은 설정을 구성할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성 요소](#)의 **mqtt** 및 **mqtt.spooler** 구성 파라미터를 참조하세요.

AWS IoT Core 는 MQTT 메시지 브로커에 서비스 할당량을 부과합니다. 이러한 할당량은 코어 디바이스와 AWS IoT Core간에 보내는 메시지에 적용될 수 있습니다. 자세한 내용은 AWS 일반 참조의 [AWS IoT Core 메시지 브로커 서비스 할당량](#)을 참조하세요.

IPv6 네트워크에서 Greengrass Nucleus 구성

Greengrass Nucleus는 Greengrass API를 AWS IoT Core 통해와 대화합니다. [APIs](#) Greengrass API는 듀얼 스택 환경에서 IPv6를 지원합니다.

IPv6용 듀얼 스택 엔드포인트를 활성화하려면:

- 시스템 속성 **aws.useDualstackEndpoint=true** 및 **java.net.preferIPv6Addresses=true**를 **jvmOptions**에 추가합니다.
- **s3EndpointType**를 **DUALSTACK**로 설정합니다.

이 옵션은 [배포](#) 시 지정하거나, **--init-config** [설치 인자](#)를 사용하여 수동으로 프로비저닝할 수 있습니다. 자세한 내용은 [Amazon S3 듀얼 스택 엔드포인트 사용](#)을 참조하세요.

Example배포용 코드:

```
{
```



```

    "jvmOptions": "-Daws.useDualstackEndpoint=true",
    "s3EndpointType": "DUALSTACK"
  }

```

Example수동 프로비저닝을 통한 config.yaml:

```

---
system:
  ...
services:
  aws.greengrass.Nucleus:
    ...
    configuration:
      ...
      jvmOptions: "-Daws.useDualstackEndpoint=true -Djava.net.preferIPv6Addresses=true"
      s3EndpointType: "DUALSTACK"

```

AWS IoT Greengrass 코어 소프트웨어(OTA) 업데이트

AWS IoT Greengrass 코어 소프트웨어는 [Greengrass nucleus 구성 요소](#)와 소프트웨어의 무선 업데이트(OTA)를 수행하기 위해 디바이스에 배포할 수 있는 기타 구성 요소(선택 사항)로 구성되어 있습니다. 이 기능은 AWS IoT Greengrass 코어 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 디바이스에 대한 고려 사항](#)
- [Greengrass nucleus 업데이트 동작](#)
- [OTA 업데이트를 수행합니다.](#)

요구 사항

AWS IoT Greengrass 코어 소프트웨어의 OTA 업데이트를 배포하려면 다음과 같은 요구 사항이 적용됩니다.

- 배포를 수신하려면 Greengrass 코어 디바이스가 AWS 클라우드에 연결되어 있어야 합니다.
- Greengrass 코어 디바이스는 AWS IoT Core 및 AWS IoT Greengrass를 사용하여 인증을 위한 인증서 및 키로 올바르게 구성되고 프로비저닝되어야 합니다.
- AWS IoT Greengrass 코어 소프트웨어는 시스템 서비스로 설정하고 실행해야 합니다. JAR 파일 (Greengrass.jar)에서 nucleus를 실행하는 경우에는 OTA 업데이트가 작동하지 않습니다. 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 섹션을 참조하세요.

코어 디바이스에 대한 고려 사항

OTA 업데이트를 수행하기 전에 업데이트하는 코어 디바이스와 연결된 클라이언트 디바이스에 미치는 영향에 유의하세요.

- Greengrass nucleus가 종료됩니다.
- 코어 디바이스에서 실행되는 모든 구성 요소도 종료됩니다. 해당 구성 요소가 로컬 리소스에 쓰는 경우에 제대로 종료하지 않으면 해당 리소스가 잘못된 상태로 남을 수 있습니다. 구성 요소는 [프로세스 간 통신](#)을 사용하여 사용하는 리소스를 정리할 때까지 업데이트를 연기하도록 nucleus 구성 요소에 지시할 수 있습니다.
- nucleus 구성 요소가 종료되는 동안 코어 디바이스는 AWS 클라우드 및 로컬 디바이스와의 연결이 끊어집니다. 코어 디바이스는 종료된 상태에서는 클라이언트 디바이스의 메시지를 라우팅하지 않습니다.
- 구성 요소로 실행되는 수명이 긴 Lambda 함수는 동적 상태 정보를 잃고 보류 중인 모든 작업을 삭제합니다.

Greengrass nucleus 업데이트 동작

구성 요소를 배포할 때 AWS IoT Greengrass는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 설치합니다. 이로 인해 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS에서 제공한 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

[Greengrass nucleus 구성 요소](#)의 버전이 변경되면 디바이스의 nucleus 및 기타 모든 구성 요소를 포함하는 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항을 적용합니다. nucleus 구성 요소가 업데이트될 때 [코어 디바이스에 미치는 영향](#)으로 인해 새 nucleus 패치 버전이 디바이스에 배포되는 시기를 제어하고 싶을 수 있습니다. 이렇게 하려면 배포에 Greengrass nucleus 구성 요소를 직접 포함해야 합니다. 구성 요소를 직접 포함한다는 것은 배포 구성에 해당 구성 요소의 특정 버전을 포함하고 해당 구성 요소를 디바이스에 배포할 때 구성 요소 종속성에 의존하지 않음을 의미합니다. 구성 요소 레시피에서 종속성을 정의하는 방법에 대한 자세한 내용은 [레시피 형식](#) 섹션을 참조하세요.

다음 표를 검토하여 작업 및 배포 구성에 따른 Greengrass nucleus 구성 요소의 업데이트 동작을 이해합니다.

Action	배포 구성	Nucleus 업데이트 동작
배포를 수정하지 않고 기존 배포가 대상으로 하는 사물 그룹에 새 디바이스를 추가합니다.	배포는 Greengrass nucleus를 직접 포함하지 않습니다. 배포는 하나 이상의 AWS 제공 구성 요소를 직접 포함하거나 AWS 제공 구성 요소 또는 Greengrass nucleus에 따라 달라지는 사용자 지정 구성 요소를 포함합니다.	새 디바이스에서는 모든 구성 요소 종속성 요구 사항을 충족하는 최신 패치 버전의 nucleus를 설치합니다. 기존 디바이스에서는 nucleus의 설치된 버전을 업데이트하지 않습니다.
배포를 수정하지 않고 기존 배포가 대상으로 하는 사물 그룹에 새 디바이스를 추가합니다.	배포는 Greengrass nucleus의 특정 버전을 직접 포함합니다.	새 디바이스에서 지정된 nucleus 버전을 설치합니다. 기존 디바이스에서는 nucleus의 설치된 버전을 업데이트하지 않습니다.
새 배포를 생성하거나 기존 배포를 수정합니다.	배포는 Greengrass nucleus를 직접 포함하지 않습니다. 배포는 하나 이상의 AWS 제공 구성 요소를 직접 포함하거나 AWS 제공 구성 요소 또는 Greengrass nucleus에 따라 달라지는 사용자 지정 구성 요소를 포함합니다.	모든 대상 디바이스에서 대상 사물 그룹에 추가하는 새 디바이스를 포함하여 모든 구성 요소 종속성 요구 사항을 충족하는 nucleus의 최신 패치 버전을 설치합니다.

Action	배포 구성	Nucleus 업데이트 동작
	라지는 사용자 지정 구성 요소를 포함합니다.	
새 배포를 생성하거나 기존 배포를 수정합니다.	배포는 Greengrass nucleus의 특정 버전을 직접 포함합니다.	모든 대상 디바이스에 대상 사물 그룹에 추가하는 새 디바이스를 포함하여 지정된 nucleus 버전을 설치합니다.

OTA 업데이트를 수행합니다.

OTA 업데이트를 수행하려면 [nucleus 구성 요소](#)와 설치할 버전이 포함된 [배포를 생성](#)합니다.

AWS IoT Greengrass 코어 소프트웨어 제거

AWS IoT Greengrass 코어 소프트웨어를 제거하여 Greengrass 코어 디바이스로 사용하지 않으려는 디바이스에서 이를 제거할 수 있습니다. 이러한 단계를 사용하여 실패한 설치를 정리할 수도 있습니다.

AWS IoT Greengrass 코어 소프트웨어를 제거하려면

1. 소프트웨어를 시스템 서비스로 실행하는 경우 해당 서비스를 중지, 비활성화, 제거해야 합니다. 운영 체제에 맞도록 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

이러한 명령을 실행하려면 관리자 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

이러한 명령을 실행하려면 관리자 권한으로 PowerShell을 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 이상인 경우:

```
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 폴더의 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 Greengrass 서비스에서 코어 디바이스를 삭제합니다. 이 단계에서는 AWS 클라우드에서 코어 디바이스의 상태 정보를 제거합니다. 동일한 이름의 코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 다시 설치하려는 경우 이 단계를 완료해야 합니다.

- AWS IoT Greengrass 콘솔에서 코어 디바이스를 삭제하려면 다음을 수행합니다.
 - a. [AWS IoT Greengrass 콘솔](#)로 이동합니다.
 - b. 코어 디바이스를 선택합니다.
 - c. 삭제할 코어 디바이스를 선택합니다.
 - d. 삭제를 선택합니다.

- e. 확인 모달에서 삭제를 선택합니다.
- AWS Command Line Interface로 코어 디바이스를 삭제하려면 [DeleteCoreDevice](#) 작업을 사용합니다. 다음 명령을 실행하고 *MyGreengrassCore*를 코어 디바이스의 이름으로 바꿉니다.

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 자습서

AWS IoT Greengrass 는 엣지 디바이스에서 AWS Lambda 함수, 기계 학습 모델 및 기타 코드를 실행할 수 있는 서비스입니다. 이를 통해 데이터를 로컬에서 처리하여 지연 시간과 대역폭 비용을 줄이면서 클라우드와의 안전한 통신을 유지할 수 있습니다.

다음 자습서를 완료하여 AWS IoT Greengrass V2 및 해당 기능에 대해 알아볼 수 있습니다.

주제

- [자습서 비디오](#)
- [자습서: 구성 요소 업데이트가 지연되는 Greengrass 구성 요소 개발](#)
- [자습서: MQTT를 통해 로컬 IoT 디바이스와 상호 작용](#)
- [자습서: 신뢰할 수 있는 플랫폼 모듈\(TPM\)을 사용하여 Greengrass Nucleus 보호](#)
- [자습서: 신뢰할 수 있는 플랫폼 모듈\(TPM\)을 사용하여 AWS IoT Greengrass Nucleus Lite 보호](#)
- [자습서: SageMaker AI Edge Manager 시작하기](#)
- [자습서: TensorFlow Lite를 사용하여 샘플 이미지 분류 추론 수행](#)
- [자습서: TensorFlow Lite를 사용하여 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행](#)

자습서 비디오

동영상

다음은 AWS IoT Greengrass V2 관련 비디오 자습서 목록입니다.

- [AWS IoT Greengrass nucleus lite 자습서: 시작하기](#)

2025년 4월 10일 - 이 자습서에서는 Greengrass nucleus lite가 무엇인지, AWS IoT 디바이스에 언제 선택해야 하는지를 배웁니다. nucleus lite 설치 방법과 구성 요소 배포 방법을 학습합니다.

- [AWS IoT Greengrass V2 자습서: 시작하기](#)

2021년 12월 16일 - AWS IoT Greengrass V2 시작하기. 이 자습서에서는 Greengrass를 설치하고 첫 번째 배포를 완료하는 방법을 배웁니다.

- [로컬 디버그 콘솔을 사용한 AWS IoT Greengrass V2 구성 요소 개발](#)

2023년 8월 21일 - Greengrass V2 로컬 디버그 콘솔을 사용하여 구성 요소를 빠르게 개발하고 디버깅하는 방법을 학습합니다.

- [MQTT 클라이언트 디바이스를 AWS IoT Greengrass V2에 연결](#)

2023년 8월 2일 - 이 자습서에서는 MQTT를 통해 코어 디바이스에 연결되는 로컬 AWS IoT 디바이스와 상호 작용하는 방법을 배웁니다.

- [AWS IoT Greengrass V2 자습서: MQTT 애플리케이션 사용](#)

2021년 12월 16일 - AWS IoT Core를 통해 MQTT를 게시하고 구독하는 AWS IoT Greengrass V2 네이티브 구성 요소를 생성, 테스트, 배포하는 방법을 학습합니다.

자습서: 구성 요소 업데이트가 지연되는 Greengrass 구성 요소 개발

이 자습서를 완료하여 무선 배포 업데이트가 지연되는 구성 요소를 개발할 수 있습니다. 디바이스에 업데이트를 배포할 때 다음과 같은 조건에 따라 업데이트를 지연시키는 것이 좋습니다.

- 디바이스의 배터리 잔량이 부족합니다.
- 중단될 수 없는 프로세스 또는 작업이 디바이스에서 실행되고 있습니다.
- 디바이스가 제한적이거나 비용이 많이 드는 인터넷에 연결되어 있습니다.

Note

구성 요소는 AWS IoT Greengrass 코어 디바이스에서 실행되는 소프트웨어 모듈입니다. 구성 요소를 사용하면 복잡한 애플리케이션을 여기저기 Greengrass 코어 디바이스에서 재사용할 수 있는 불연속 빌딩 블록으로 생성하고 관리할 수 있습니다.

이 자습서에서는 다음을 수행합니다.

1. 개발 컴퓨터에 Greengrass 개발 키트 CLI(GDK CLI)를 설치합니다. GDK CLI에서는 사용자 지정 Greengrass 구성 요소를 개발하는 데 도움이 되는 특성이 제공됩니다.
2. 코어 디바이스의 배터리 잔량이 임계값 미만일 때 구성 요소 업데이트가 지연되는 Hello World 구성 요소를 개발합니다. 이 구성 요소에서는 [SubscribeToComponentUpdates](#) IPC 작업이 사용되는 업데이트 알림이 구독됩니다. 알림이 수신되면 배터리 잔량이 사용자 지정 가능한 임계값보다 낮은지 확인됩니다. 배터리 잔량이 임계값 미만인 경우 [DeferComponentUpdate](#) IPC 작업이 사용되어 30초 동안 업데이트가 지연됩니다. GDK CLI를 사용하여 개발 컴퓨터에서 이 구성 요소를 개발합니다.

Note

이 구성 요소에서는 코어 디바이스에서 생성하는 파일의 배터리 잔량이 옳히며 실제 배터리가 모방되므로 배터리 없이 코어 디바이스에서 이 자습서를 완료할 수 있습니다.

3. 해당 구성 요소를 AWS IoT Greengrass 서비스에 게시합니다.
4. 해당 구성 요소에서 Greengrass 코어 디바이스 AWS 클라우드 로 배포하여 테스트합니다. 그런 다음에 코어 디바이스에서 가상 배터리 잔량을 수정하고 추가 배포를 생성하여 배터리 잔량이 부족할 때 코어 디바이스에서 업데이트가 지연되는 방식을 참조합니다.

이 자습서의 예상 소요 시간은 20~30분입니다.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 없으면 [설정 AWS 계정](#) 단원을 참조하세요.
- 관리자 권한이 있는 AWS Identity and Access Management(IAM) 사용자.
- 인터넷에 연결된 Greengrass 코어 디바이스. 코어 디바이스 설정 방법에 대한 자세한 내용은 [AWS IoT Greengrass 코어 디바이스 설정](#) 단원을 참조하세요.
- 디바이스의 모든 사용자에게 대해 [Python](#) 3.6 이상이 설치되고 PATH 환경 변수에 추가되어야 합니다. Windows에서는 모든 사용자에게 대해 Windows용 Python 런처가 설치되어 있어야 합니다.

Important

Windows에서 Python은 기본적으로 모든 사용자를 대상으로 설치되지 않습니다. Python을 설치할 때는 AWS IoT Greengrass 코어 소프트웨어에서 Python 스크립트가 실행되도록 설치 구성을 사용자 지정해야 합니다. 예를 들어 그래픽 Python 설치 관리자를 사용하는 경우 다음을 수행합니다.

1. 모든 사용자에게 대해 런처 설치(권장)를 선택합니다.
2. 를 선택합니다..Customize installation
3. 를 선택합니다..Next
4. 을 선택합니다..Install for all users
5. 을 선택합니다..Add Python to environment variables

6. 설치를 선택합니다.

자세한 내용은 Python 3 설명서의 [Windows에서 Python 사용](#)을 참조하세요.

- 인터넷에 연결된 Windows, macOS 또는 Unix 계열 개발 컴퓨터.
- 개발 컴퓨터에 설치된 [Python](#) 3.6 이상.
- 개발 컴퓨터에 설치된 [Git](#).
- 자격 증명을 사용하여 개발 컴퓨터에 AWS Command Line Interface(AWS CLI)가 설치되어 있어야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 [AWS CLI 설치, 업데이트, 제거](#) 및 [AWS CLI 구성](#) 단원을 참조하세요.

Note

Raspberry Pi 또는 다른 32비트 ARM 디바이스를 사용하는 경우 AWS CLI V1을 설치합니다. AWS CLI 32비트 ARM 디바이스에서는 V2를 사용할 수 없습니다. 자세한 내용은 [AWS CLI 버전 1 설치, 업데이트 및 제거](#) 단원을 참조하세요.

1단계: Greengrass 개발 키트 CLI 설치

[Greengrass 개발 키트 CLI\(GDK CLI\)](#)는 사용자 지정 Greengrass 구성 요소를 개발하는 데 도움이 되는 기능을 제공합니다. GDK CLI를 사용하여 사용자 지정 구성 요소를 생성, 빌드 및 게시할 수 있습니다.

개발 컴퓨터에 GDK CLI를 설치하지 않은 경우 다음 단계를 완료하여 설치합니다.

GDK CLI의 최신 버전 설치

1. 개발 컴퓨터에서 다음 명령을 실행하여 [GitHub 리포지토리](#)에서 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
```

gdk 명령을 찾을 수 없는 경우 해당 폴더를 PATH에 추가합니다.

- Linux 디바이스에서 PATH에 `/home/MyUser/.local/bin`을 추가하고 *MyUser*를 사용자 이름으로 바꿉니다.
- Windows 디바이스에서 PATH에 *PythonPath*\\Scripts를 추가하고 *PythonPath*를 디바이스의 Python 폴더 경로로 바꿉니다.

2단계: 업데이트가 지연되는 구성 요소 개발

이 섹션에서는 코어 디바이스의 배터리 잔량이 구성 요소를 배포할 때 구성하는 임계값 미만일 때 구성 요소 업데이트가 지연되는 Python의 Hello World 구성 요소를 개발합니다. 이 구성 요소에서는 Python용 AWS IoT Device SDK v2의 [프로세스 간 통신\(IPC\) 인터페이스](#)를 사용합니다. 코어 디바이스에 배포가 수신될 때 [SubscribeToComponentUpdates](#) IPC 작업을 사용하여 알림을 받습니다. 그런 다음에 [DeferComponentUpdate](#) IPC 작업을 사용하여 디바이스의 배터리 잔량에 따라 업데이트를 연기하거나 승인합니다.

업데이트가 지연되는 Hello World 구성 요소를 개발하려면

1. 개발 컴퓨터에서 구성 요소 소스 코드의 폴더를 생성합니다.

```
mkdir com.example.BatteryAwareHelloWorld
cd com.example.BatteryAwareHelloWorld
```

2. 텍스트 편집기를 사용하여 `gdk-config.json`이라는 파일을 생성합니다. GDK CLI에서는 `gdk-config.json`라는 [GDK CLI 구성 파일](#)의 구성 요소 읽기, 빌드 및 게시가 진행됩니다. 이 구성 파일은 구성 요소 폴더의 루트에 있습니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano gdk-config.json
```

다음 JSON을 파일로 복사합니다.

- 자신의 이름으로 *Amazon*을 바꿉니다.
- *us-west-2*를 코어 디바이스가 작동하는 AWS 리전 로 바꿉니다. GDK CLI에서는 이 AWS 리전에 구성 요소가 게시됩니다.
- 사용할 S3 버킷 접두사로 *greengrass-component-artifacts*를 바꿉니다. GDK CLI를 사용하여 구성 요소를 게시하면 GDK CLI는 다음 형식을 사용하여 이름이이 값,

AWS 리전 및 AWS 계정 ID로 구성된 S3 버킷에 구성 요소의 아티팩트를 업로드합니다 ***bucketPrefix-region-accountId***.

예를 들어 **greengrass-component-artifacts** 및를 지정 **us-west-2**하고 AWS 계정 ID가 인 경우 **123456789012** GDK CLI는 라는 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 파일에서 다음 구성이 지정됩니다.

- GDK CLI가 Greengrass 구성 요소를 클라우드 서비스에 게시할 AWS IoT Greengrass 때 사용할 버전입니다.는 AWS IoT Greengrass 클라우드 서비스에서 사용할 수 있는 최신 버전 이후의 다음 패치 버전을 선택하도록 NEXT_PATCH 지정합니다. 구성 요소에 아직 AWS IoT Greengrass 클라우드 서비스에 버전이 없는 경우 GDK CLI는를 사용합니다1.0.0.
 - 구성 요소의 빌드 시스템입니다. zip 빌드 시스템을 사용하면 GDK CLI에서는 구성 요소의 단일 아티팩트가 되는 ZIP 파일로 구성 요소의 소스가 패키징됩니다.
 - GDK CLI AWS 리전 가 Greengrass 구성 요소를 게시하는 입니다.
 - GDK CLI에서 구성 요소의 아티팩트가 업로드되는 S3 버킷의 접두사입니다.
3. 텍스트 편집기를 사용하여 main.py라는 파일에서 구성 요소 소스 코드를 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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 애플리케이션에서는 다음이 수행됩니다.

- 나중에 코어 디바이스에서 생성할 가상 배터리 잔량 파일에서 코어 디바이스의 배터리 잔량을 읽습니다. 이 가상 배터리 잔량 파일에서는 실제 배터리가 모방되므로 배터리가 없는 코어 디바이스에서 이 자습서를 완료할 수 있습니다.

- 배터리 임계값 및 가상 배터리 잔량 파일의 경로에 대한 명령줄 인수를 읽습니다. 구성 요소 레지스트리에서는 구성 파라미터를 기반으로 이러한 명령줄 인수가 설정되므로 구성 요소를 배포할 때 이러한 값을 사용자 지정할 수 있습니다.
- Python용 v2의 IPC 클라이언트 V2를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 통신합니다. [AWS IoT Device SDK](#) 원래 IPC 클라이언트에 비해 IPC 클라이언트 V2에서는 사용자 지정 구성 요소에서 IPC를 사용하려면 작성해야 할 코드 양이 감소합니다.
- [SubscribeToComponentUpdates](#) IPC 작업이 사용되는 업데이트 알림을 구독합니다. AWS IoT Greengrass 코어 소프트웨어는 각 배포 후에도 알림을 보냅니다. 구성 요소에서는 알림을 수신될 때마다 다음 함수가 직접적으로 호출됩니다. 알림이 예정된 배포용이라면 구성 요소에서는 배터리 잔량이 임계값보다 낮은지 확인됩니다. 배터리 잔량이 임계값 미만인 경우 구성 요소에서는 [DeferComponentUpdate](#) IPC 작업이 사용되어 30초 동안 업데이트가 지연됩니다. 그렇지 않으면 배터리 잔량이 임계값 미만인 경우 구성 요소에서 업데이트가 승인되므로 업데이트가 진행될 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 Greengrass 코어 소프트웨어는 로컬 배포에 대한 업데이트 알림을 보내지 않으므로 AWS IoT Greengrass 클라우드 서비스를 사용하여이 구성 요소를 배포하여 테스트합니다.

4. 텍스트 편집기를 사용하여 `recipe.json` 또는 `recipe.yaml`이라는 파일에서 구성 요소 레시피를 생성합니다. 구성 요소 레시피에서는 구성 요소의 메타데이터, 기본 구성 파라미터 및 플랫폼별 수명 주기 스크립트가 정의됩니다.

JSON

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano recipe.json
```

다음 JSON을 파일로 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 시스템에서 다음 명령을 실행하면 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 awsiot-sdk --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 awsiot-sdk --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 레시피에서는 다음이 지정됩니다.

- 배터리 임계값, Linux 코어 디바이스의 가상 배터리 파일 경로 및 Windows 코어 디바이스의 가상 배터리 파일 경로에 대한 기본 구성 파라미터입니다.
- Python용 AWS IoT Device SDK v2의 최신 버전이 설치되는 `install` 수명 주기입니다.
- `main.py`에서 Python 애플리케이션 실행되는 `run` 수명 주기입니다.
- GDK CLI에서 구성 요소 레시피가 빌드될 때 정보가 바뀌는 자리 표시자(예: `COMPONENT_NAME` 및 `COMPONENT_VERSION`)입니다.

구성 요소 레시피에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 섹션을 참조하세요.

3단계: AWS IoT Greengrass 서비스에 구성 요소 게시

이 단원에서는 Hello World 구성 요소를 AWS IoT Greengrass 클라우드 서비스에 게시합니다. AWS IoT Greengrass 클라우드 서비스에서 구성 요소를 사용할 수 있게 되면 코어 디바이스에 배포할 수 있습니다. GDK CLI를 사용하여 개발 컴퓨터에서 AWS IoT Greengrass 클라우드 서비스로 구성 요소를 게시합니다. GDK CLI는 구성 요소의 레시피와 아티팩트를 업로드합니다.

Hello World 구성 요소를 AWS IoT Greengrass 서비스에 게시하려면

1. 다음 명령을 실행하여 GDK CLI를 통해 구성 요소를 빌드합니다. [구성 요소 빌드 명령](#)은 GDK CLI 구성 파일을 기반으로 레시피와 아티팩트를 생성합니다. 이 프로세스에서 GDK CLI는 구성 요소의 소스 코드를 포함하는 ZIP 파일을 생성합니다.

```
gdk component build
```

다음 예제와 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 명령](#)은 구성 요소의 ZIP 파일 아티팩트를 S3 버킷에 업로드합니다. 그런 다음 구성 요소 레시피에서 ZIP 파일의 S3 URI를 업데이트하고 레시피를 AWS IoT Greengrass 서비스에 업로드합니다. 이 프로세스에서 GDK CLI는 AWS IoT Greengrass 클라우드 서비스에서 이미 사용할 수 있는 Hello World 구성 요소의 버전을 확인하므로 해당 버전 이후의 다음 패치 버전을 선택할 수 있습니다. 구성 요소가 아직 없는 경우 GDK CLI는 버전 1.0.0을 사용합니다.

```
gdk component publish
```

다음 예제와 비슷한 메시지가 표시되어야 합니다. 출력에서는 GDK CLI가 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 다운로드할 수 있도록 합니다.
4. (선택 사항) AWS IoT Greengrass 콘솔에서 구성 요소를 보고 성공적으로 업로드되었는지 확인합니다. 해결 방법:
 - a. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
 - b. 구성 요소 페이지에서 내 구성 요소 탭을 선택한 다음에 com.example.BatteryAwareHelloWorld를 선택합니다.

이 페이지에서 구성 요소의 레시피와 구성 요소에 대한 기타 정보를 볼 수 있습니다.

5. 코어 디바이스가 S3 버킷의 구성 요소 아티팩트에 액세스하도록 허용합니다.

각 코어 디바이스에는 AWS 클라우드와 상호 작용 AWS IoT 하고 로그를 전송할 수 있는 [코어 디바이스 IAM 역할](#)이 있습니다. 이 디바이스 역할은 기본적으로 S3 버킷에 대한 액세스를 허용하지 않으므로 코어 디바이스가 S3 버킷에서 구성 요소 아티팩트를 검색할 수 있도록 허용하는 정책을 생성하고 연결해야 합니다.

디바이스의 역할이 이미 S3 버킷에 대한 액세스를 허용하는 경우 이 단계를 건너뛸 수 있습니다. 그렇지 않으면 다음과 같이 액세스를 허용하는 IAM 정책을 생성하여 역할에 연결합니다.

- a. [IAM 콘솔](#) 탐색 메뉴에서 정책을 선택한 다음 정책 생성을 선택합니다.
- b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다. *greengrass-component-artifacts-us-west-2-123456789012*를 GDK CLI가 구성 요소의 아티팩트를 업로드한 S3 버킷의 이름으로 바꿉니다.

예를 들어 GDK CLI 구성 파일에서 **greengrass-component-artifacts** 및 **us-west-2**를 지정했고 AWS 계정 ID가 **123456789012**인 경우 GDK CLI는 greengrass-component-artifacts-us-west-2-123456789012라는 S3 버킷을 사용합니다.

JSON

```
{
  "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 섹션에서 이름에 **MyGreengrassV2ComponentArtifactPolicy**를 입력합니다.
- e. 정책 생성을 선택합니다.

- f. [IAM 콘솔](#) 탐색 메뉴에서 역할을 선택한 다음 코어 디바이스에 대한 역할 이름을 선택합니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 때이 역할 이름을 지정했습니다. 이름을 지정하지 않은 경우 기본값은 GreengrassV2TokenExchangeRole입니다.
- g. 권한에서 권한 추가, 정책 연결을 차례로 선택합니다.
- h. 권한 추가 페이지에서 생성한 MyGreengrassV2ComponentArtifactPolicy 정책 옆의 확인란을 선택한 다음 권한 추가를 선택합니다.

4단계: 디바이스에서 구성 요소 배포 및 테스트

이 섹션에서는 코어 디바이스에 구성 요소를 배포하여 해당 기능을 테스트합니다. 실제 배터리가 모방되는 가상 배터리 잔량 파일을 코어 디바이스에서 생성합니다. 그런 다음에 코어 디바이스에서 추가 배포를 생성하고 구성 요소 로그 파일을 관찰하여 구성 요소 지연 및 승인 업데이트를 참조합니다.

업데이트가 지연되는 Hello World 구성 요소를 배포하고 테스트하려면

1. 텍스트 편집기를 사용하여 가상 배터리 잔량 파일을 생성합니다. 이 파일에서는 실제 배터리가 모방됩니다.
 - Linux 코어 디바이스에서 /home/ggc_user/virtual_battery.json이라는 파일을 생성합니다. sudo 권한으로 텍스트 편집기를 실행합니다.
 - Windows 코어 디바이스에서 C:\Users\ggc_user\virtual_battery.json이라는 파일을 생성합니다. 관리자로 텍스트 편집기를 실행합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
sudo nano /home/ggc_user/virtual_battery.json
```

다음 JSON을 파일로 복사합니다.

```
{
  "battery_level": 50
}
```

2. Hello World 구성 요소를 코어 디바이스에 배포합니다. 해결 방법:
 - a. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.

- b. 구성 요소 페이지에서 내 구성 요소 탭을 선택한 다음에 `com.example.BatteryAwareHelloWorld`를 선택합니다.
- c. `com.example.BatteryAwareHelloWorld` 페이지에서 배포를 선택합니다.
- d. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포 생성을 선택하고 다음을 선택합니다.
- e. 새 배포 생성을 선택한 경우 배포의 대상 코어 디바이스 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 디바이스 또는 사물 그룹을 선택하고 다음을 선택합니다.
- f. 구성 요소 선택 페이지에서 `com.example.BatteryAwareHelloWorld` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
- g. 구성 요소 구성 페이지에서 `com.example.BatteryAwareHelloWorld`를 선택하고 다음을 수행합니다.
 - i. 구성 요소 구성을 선택합니다.
 - ii. 구성 `com.example.BatteryAwareHelloWorld` 모달에서 구성 업데이트 아래의 병합할 구성에 다음 구성 업데이트를 입력합니다.

```
{
  "BatteryThreshold": 70
}
```

- iii. 확인을 선택하여 모달을 닫고 다음을 선택합니다.
- h. 고급 설정 확인 페이지에서 배포 정책 섹션의 구성 요소 업데이트 정책에서 구성 요소에 알림이 선택되었는지 확인합니다. 구성 요소에 알림은 새 배포를 생성할 때 기본적으로 선택됩니다.
- i. 검토 페이지에서 배포를 선택합니다.

배포를 완료하는 데 최대 1분이 걸릴 수 있습니다.

3. AWS IoT Greengrass 코어 소프트웨어는 구성 요소 프로세스의 stdout을 `logs` 폴더의 로그 파일에 저장합니다. 다음 명령을 실행하여 Hello World 구성 요소가 실행되고 상태 메시지가 인쇄되는지 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 예제와 비슷한 메시지가 표시되어야 합니다.

```
Hello, World! Battery level (50) is below threshold (70), so the component will defer updates.
```

Note

파일이 없으면 배포가 아직 완료되지 않았을 수 있습니다. 30초 이내에 파일이 존재하지 않으면 배포에 실패했을 수 있습니다. 예를 들어, S3 버킷에서 구성 요소의 아티팩트를 다운로드할 권한이 코어 디바이스에 없으면 이 문제가 발생할 수 있습니다. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

4. 코어 디바이스에 대한 새 배포를 생성하여 구성 요소에서 업데이트가 지연되는지 확인합니다. 해결 방법:
 - a. [AWS IoT Greengrass 콘솔](#) 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 다음에 배포를 선택합니다.
5. 다음 명령을 실행하여 구성 요소의 로그를 다시 보고 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 예제와 비슷한 메시지가 표시되어야 합니다. 구성 요소에서는 업데이트가 30초 동안 지연되므로 이 메시지가 구성 요소에서 반복적으로 인쇄됩니다.

```
Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
```

6. 배포가 진행될 수 있도록 텍스트 편집기를 사용하여 가상 배터리 잔량 파일을 편집하고 배터리 잔량을 임계값보다 높은 값으로 변경합니다.
 - Linux 코어 디바이스에서 /home/ggc_user/virtual_battery.json이라는 파일을 편집합니다. sudo 권한으로 텍스트 편집기를 실행합니다.
 - Windows 코어 디바이스에서 C:\Users\ggc_user\virtual_battery.json이라는 파일을 편집합니다. 관리자로 텍스트 편집기를 실행합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
sudo nano /home/ggc_user/virtual_battery.json
```

배터리 잔량을 80으로 변경합니다.

```
{  
  "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 예제와 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 자습서를 완료했습니다. Hello World 구성 요소에서는 코어 디바이스의 배터리 잔량에 따라 업데이트가 지연되거나 승인됩니다. 이 자습서에서 살펴보는 주제에 대한 자세한 내용은 다음을 참조하세요.

- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)

- [AWS IoT Device SDK 를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core](#)
- [AWS IoT Greengrass 개발 키트 명령줄 인터페이스](#)

자습서: MQTT를 통해 로컬 IoT 디바이스와 상호 작용

이 자습서를 완료하면 코어 디바이스를 구성하여 MQTT를 통해 코어 디바이스에 연결되는 클라이언트 디바이스라는 IoT 디바이스와 상호 작용할 수 있습니다. 이 자습서에서는 클라우드 검색을 사용하여 코어 디바이스에 클라이언트 디바이스로 연결하도록 AWS IoT 사물을 구성합니다. 클라우드 검색을 구성하면 클라이언트 디바이스가 AWS IoT Greengrass 클라우드 서비스에 요청을 보내 코어 디바이스를 검색할 수 있습니다. AWS IoT Greengrass의 응답에는 검색하도록 클라이언트 디바이스를 구성하는 코어 디바이스에 대한 연결 정보와 인증서가 포함되어 있습니다. 그러면 클라이언트 디바이스에서는 이 정보를 사용하여 MQTT를 통해 통신할 수 있는 사용 가능한 코어 디바이스에 연결할 수 있습니다.

이 자습서에서는 다음을 수행합니다.

1. 필요한 경우 코어 디바이스의 권한을 검토하고 업데이트합니다.
2. 클라우드 검색을 사용하여 코어 디바이스를 검색할 수 있도록 클라이언트 디바이스를 코어 디바이스에 연결합니다.
3. 클라이언트 디바이스 지원을 활성화할 Greengrass 구성 요소를 코어 디바이스에 배포합니다.
4. 클라이언트 디바이스를 코어 디바이스에 연결하고 AWS IoT Core 클라우드 서비스와 통신을 테스트합니다.
5. 클라이언트 디바이스와 통신하는 사용자 지정 Greengrass 구성 요소를 개발합니다.
6. 클라이언트 디바이스의 [AWS IoT 디바이스 새도](#)와 상호 작용하는 사용자 지정 구성 요소를 개발합니다.

이 자습서에서는 단일 코어 디바이스와 단일 클라이언트 디바이스를 사용합니다. 자습서를 따라 여러 클라이언트 디바이스를 연결하고 테스트할 수도 있습니다.

이 자습서의 예상 소요 시간은 30~60분입니다.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 없으면 [설정 AWS 계정](#) 섹션을 참조하세요.

- 관리자 권한이 있는 AWS Identity and Access Management (IAM) 사용자.
- Greengrass 코어 디바이스. 코어 디바이스 설정 방법에 대한 자세한 내용은 [AWS IoT Greengrass 코어 디바이스 설정](#) 섹션을 참조하세요.
- 코어 디바이스에서 Greengrass nucleus v2.6.0 이상이 실행되어야 합니다. 이 버전에는 로컬 게시/구독 통신의 와일드카드에 대한 지원과 클라이언트 디바이스 새도에 대한 지원이 포함되어 있습니다.

Note

클라이언트 디바이스 지원에는 Greengrass nucleus v2.2.0 이상이 필요합니다. 그러나 이 자습서에서는 더 새로운 특성(예: 로컬 게시/구독의 MQTT 와일드카드에 대한 지원 및 클라이언트 디바이스 새도에 대한 지원)을 살펴봅니다. 이러한 특성에는 Greengrass nucleus 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 CLI가 실행되어야 합니다. 자세한 내용은 [Greengrass CLI 설치](#) 단원을 참조하십시오.
- 이 자습서에서 클라이언트 디바이스로 연결할 AWS IoT 사물입니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 리소스 생성](#)을 참조하세요.
- 클라이언트 디바이스의 AWS IoT 정책은 `greengrass:Discover` 권한을 허용해야 합니다. 자세한 내용은 [클라이언트 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.
- 클라이언트 디바이스는 코어 디바이스와 동일한 네트워크에 있어야 합니다.
- 클라이언트 디바이스에서 [Python 3](#)가 실행되어야 합니다.
- 클라이언트 디바이스에서 [Git](#)이 실행되어야 합니다.

1단계: 코어 디바이스 AWS IoT 정책 검토 및 업데이트

클라이언트 디바이스를 지원하려면 코어 디바이스의 AWS IoT 정책에서 다음 권한을 허용해야 합니다.

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`

- `greengrass:UpdateConnectivityInfo` – (선택 사항)이 권한은 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 [IP 감지기 구성 요소](#)를 사용하는 데 필요합니다.

코어 디바이스에 대한 이러한 권한 및 AWS IoT 정책에 대한 자세한 내용은 [AWS IoT 데이터 영역 작업에 대한 정책](#) 및 섹션을 참조하세요 [클라이언트 디바이스를 지원하는 최소 AWS IoT 정책](#).

이 섹션에서는 코어 디바이스에 대한 AWS IoT 정책을 검토하고 누락된 필수 권한을 추가합니다. [AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 코어 디바이스에는 모든 AWS IoT Greengrass 작업()에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다 `greengrass:*`. 이 경우 디바이스 새도우를 동기화하도록 새도 관리자 구성 요소를 구성하려는 경우에만 AWS IoT 정책을 업데이트해야 합니다 AWS IoT Core. 그렇지 않으면 이 섹션을 건너뛰어도 됩니다.

코어 디바이스의 AWS IoT 정책을 검토하고 업데이트하려면

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 코어 디바이스를 선택합니다.
2. 업데이트할 코어 디바이스를 코어 디바이스 페이지에서 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 코어 디바이스의 사물에 대한 링크를 선택합니다. 이 링크를 통해 사물 세부 정보 페이지가 AWS IoT 콘솔에서 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 검토하고 업데이트할 AWS IoT 정책을 선택합니다. 코어 디바이스의 활성 인증서에 연결된 원하는 정책에 필수 권한을 추가할 수 있습니다.

Note

[AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 `GreengrassV2IoTThingPolicy`인 정책을 선택하는 것이 좋습니다(있는 경우). 빠른 설치 프로그램으로 생성한 코어 디바이스에서는 기본적으로 이 정책 이름이 사용됩니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 코어 디바이스에도 이러한 권한을 부여하는 것입니다.

8. 정책 개요에서 활성 버전 편집을 선택합니다.
9. 필수 권한에 대한 정책을 검토하고 누락된 필수 권한이 있으면 추가합니다.

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 디바이스 지원 활성화

클라우드 검색을 사용하여 코어 디바이스에 연결하는 클라이언트 디바이스를 위해 디바이스를 연결해야 합니다. 클라이언트 디바이스를 코어 디바이스에 연결하면 해당 클라이언트 디바이스에서는 코어 디바이스의 IP 주소와 인증서를 검색하여 연결에 사용할 수 있게 됩니다.

클라이언트 디바이스가 코어 디바이스에 안전하게 연결하고 Greengrass 구성 요소와 통신할 AWS IoT Core 수 있도록 하려면 코어 디바이스에 다음 Greengrass 구성 요소를 배포합니다.

- [클라이언트 디바이스 인증](#) (`aws.greengrass.clientdevices.Auth`)

클라이언트 디바이스를 인증하고 클라이언트 디바이스 작업을 승인할 클라이언트 디바이스 인증 구성 요소를 배포합니다. 이 구성 요소를 사용하면 AWS IoT 사물을 코어 디바이스에 연결할 수 있습니다.

이 구성 요소를 사용하려면 몇 가지 구성이 필요합니다. 클라이언트 디바이스의 그룹과 각 그룹에서 수행하도록 승인되는 작업(예: MQTT를 통한 연결 및 통신)을 지정해야 합니다. 자세한 내용은 [클라이언트 디바이스 인증 구성 요소 구성](#)을 참조하세요.

- [MQTT 3.1.1 브로커\(Moquette\)](#) (`aws.greengrass.clientdevices.mqtt.Moquette`)

경량 MQTT 브로커를 실행할 Moquette MQTT 브로커 구성 요소를 배포합니다. Moquette MQTT 브로커에서는 MQTT 3.1.1을 준수하며 QoS 0, QoS 1, QoS 2, 보존된 메시지, Last Will 메시지 및 영구 구독에 대한 로컬 지원이 포함되어 있습니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다. 그러나 이 구성 요소에서 MQTT 브로커를 운영하는 포트는 구성해도 됩니다. 기본적으로 포트 8883이 사용됩니다.

- [MQTT 브리지](#) (`aws.greengrass.clientdevices.mqtt.Bridge`)

(선택 사항) MQTT 브리지 구성 요소를 배포하여 클라이언트 디바이스(로컬 MQTT), 로컬 게시/구독 및 AWS IoT Core MQTT 간에 메시지를 릴레이합니다. 클라이언트 디바이스와 동기화 AWS IoT Core 하고 Greengrass 구성 요소의 클라이언트 디바이스와 상호 작용하도록 이 구성 요소를 구성합니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다. 이 구성 요소에서 메시지를 릴레이하는 주제 매핑을 지정해야 합니다. 자세한 내용은 [MQTT 브리지 구성 요소 구성](#)을 참조하세요.

- [IP 감지기](#) (`aws.greengrass.clientdevices.IPDetector`)

(선택 사항) 코어 디바이스의 MQTT 브로커 엔드포인트를 AWS IoT Greengrass 클라우드 서비스에 자동으로 보고하도록 IP 감지기 구성 요소를 배포합니다. 네트워크 설정이 복잡한 경우(예: 라우터에서 MQTT 브로커 포트를 코어 디바이스에 전달하는 경우) 이 구성 요소를 사용할 수 없습니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다.

이 섹션에서는 AWS IoT Greengrass 콘솔을 사용하여 클라이언트 디바이스를 연결하고 클라이언트 디바이스 구성 요소를 코어 디바이스에 배포합니다.

클라이언트 디바이스 지원을 활성화하려면

1. [AWS IoT Greengrass 콘솔](#)로 이동합니다.
2. 왼쪽 탐색 메뉴에서 코어 디바이스를 선택합니다.
3. 클라이언트 디바이스 지원을 활성화하려는 코어 디바이스를 코어 디바이스 페이지에서 선택합니다.
4. 코어 디바이스 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. 클라이언트 디바이스 탭에서 클라우드 검색 구성을 선택합니다.

코어 디바이스 검색 구성 페이지가 열립니다. 이 페이지에서 클라이언트 디바이스를 코어 디바이스에 연결하고 클라이언트 디바이스 구성 요소를 배포할 수 있습니다. 이 페이지의 1단계: 대상 코어 디바이스 선택에서는 코어 디바이스가 자동으로 선택됩니다.

Note

이 페이지를 사용하여 사물 그룹에 대한 코어 디바이스 검색을 구성할 수도 있습니다. 이 옵션을 선택하면 사물 그룹의 모든 코어 디바이스에 클라이언트 디바이스 구성 요소를 배포할 수 있습니다. 그러나 이 옵션을 선택하는 경우 나중에 배포를 생성하면 각 코어 디바이스에 수동으로 클라이언트 디바이스를 연결해야 합니다. 이 자습서에서는 단일 코어 디바이스를 구성합니다.

6. 2단계: 클라이언트 디바이스 연결에서 클라이언트 디바이스의 AWS IoT 사물을 코어 디바이스에 연결합니다. 그러면 클라이언트 디바이스에서는 클라우드 검색을 사용하여 코어 디바이스의 연결 정보와 인증서를 검색할 수 있습니다. 다음을 수행합니다.

- a. 클라이언트 디바이스 연결을 선택합니다.
 - b. 클라이언트 디바이스를 코어 디바이스와 연결 modal에 연결할 AWS IoT 사물의 이름을 입력합니다.
 - c. 추가를 선택합니다.
 - d. 연결을 선택합니다.
7. 3단계: Greengrass 구성 요소 구성 및 연결에서 구성 요소를 배포하여 클라이언트 디바이스 지원을 활성화합니다. 대상 코어 디바이스에 이전 배포가 있으면 이 페이지에서 해당 배포가 수정됩니다. 그렇지 않으면 이 페이지에서 코어 디바이스에 대한 새 배포가 생성됩니다. 다음을 수행하여 클라이언트 디바이스 구성 요소를 구성하고 배포합니다.
- a. 이 자습서를 완료하려면 코어 디바이스에서 [Greengrass nucleus](#) v2.6.0 이상이 실행되어야 합니다. 코어 디바이스에서 이전 버전이 실행되는 경우 다음을 수행합니다.
 - i. aws.greengrass.Nucleus 구성 요소를 배포할 확인란을 선택합니다.
 - ii. aws.greengrass.Nucleus 구성 요소에는 구성 편집을 선택합니다.
 - iii. 구성 요소 버전에는 버전 2.6.0 이상을 선택합니다.
 - iv. 확인을 선택합니다.

 Note

이전 마이너 버전에서 Greengrass nucleus를 업그레이드하고 코어 디바이스가 nucleus에 의존하는 [AWS제공 구성](#) 요소를 실행하는 경우 AWS제공 구성 요소도 최신 버전으로 업데이트해야 합니다. 이 자습서의 뒷부분에서 배포를 검토할 때 이러한 구성 요소의 버전을 구성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#) 단원을 참조하십시오.

- b. aws.greengrass.clientdevices.Auth 구성 요소에는 구성 편집을 선택합니다.
- c. 클라이언트 디바이스 인증 구성 요소에 대한 구성 편집 modal에서 코어 디바이스의 MQTT 브로커에 대한 게시 및 구독을 클라이언트 디바이스에 허용하는 권한 부여 정책을 구성합니다. 다음을 수행합니다.
 - i. 구성에서 클라이언트 디바이스 권한 부여 정책이 있는 다음 구성을 병합할 구성 코드 블록에 입력합니다. 각 디바이스 그룹 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

- 이 정책에서는 모든 MQTT 주제에 대한 연결과 통신을 이름이 MyClientDevice로 시작하는 클라이언트 디바이스에 허용합니다. 클라이언트 디바이스로 연결할 AWS IoT 사물의 이름으로 *MyClientDevice**를 바꿉니다. 클라이언트 디바이스의 이름과 일치하는 * 와일드카드가 포함된 이름을 지정할 수도 있습니다. * 와일드카드는 이름 끝에 있어야 합니다.

연결할 두 번째 클라이언트 디바이스가 있으면 *MyOtherClientDevice**를 해당 클라이언트 디바이스의 이름 또는 해당 클라이언트 디바이스의 이름과 일치하는 와일드카드 패턴으로 바꿉니다. 그렇지 않으면 MyOtherClientDevice*와 이름이 일치하는 클라이언트 디바이스에 연결과 통신을 허용하는 선택 규칙의 이 섹션을 제거하거나 유지할 수 있습니다.

- 이 정책에서는 모든 MQTT 주제에 대한 연결과 통신도 이름이 MyOtherClientDevice로 시작하는 클라이언트 디바이스에 허용하는 OR 연산자를 사용합니다. 이 절을 선택 규칙에서 제거하거나 연결할 클라이언트 디바이스와 일치하도록 수정할 수 있습니다.
- 이 정책에서는 모든 MQTT 주제에 대한 게시 및 구독을 클라이언트 디바이스에 허용합니다. 보안 모범 사례를 따르려면 클라이언트 디바이스에서 통신에 사용하는 주제의 최소 세트로 mqtt:publish 및 mqtt:subscribe 작업을 제한합니다.

```
{
  "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 디바이스 인증 구성 요소 구성](#)을 참조하세요.

- ii. 확인을 선택합니다.
- d. `aws.greengrass.clientdevices.mqtt.Bridge` 구성 요소에는 구성 편집을 선택합니다.
- e. MQTT 브리지 구성 요소에 대한 구성 편집 모달에서 MQTT 메시지를 클라이언트 디바이스에서 AWS IoT Core로 릴레이하는 주제 매핑을 구성합니다. 해결 방법:
 - i. 구성에서 다음 구성을 병합할 구성 코드 블록에 입력합니다. 이 구성을 통해 `clients/+hello/world` 주제 필터에 대한 MQTT 메시지를 클라이언트 디바이스에서 AWS IoT Core 클라우드 서비스로 릴레이하는 방법이 지정됩니다. 예를 들어 이 주제 필터는 `clients/MyClientDevice1/hello/world` 주제와 일치합니다.

```

{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {

```

```

        "topic": "clients/+/hello/world",
        "source": "LocalMqtt",
        "target": "IotCore"
    }
}
}

```

자세한 내용은 [MQTT 브리지 구성 요소 구성](#)을 참조하세요.

ii. 확인을 선택합니다.

8. 검토 및 배포를 선택하여 이 페이지에서 자동으로 생성되는 배포를 검토합니다.
9. 이전에 이 리전의 [Greengrass 서비스 역할](#)을 설정한 적이 없으면 서비스 역할을 자동으로 설정하는 모달이 콘솔에서 열립니다. 클라이언트 디바이스 인증 구성 요소에서는 이 서비스 역할을 사용하여 클라이언트 디바이스의 ID를 확인하고, IP 감지기 구성 요소에서는 이 서비스 역할을 사용하여 코어 디바이스 연결 정보를 관리합니다. 권한 부여를 선택합니다.
10. 검토 페이지에서 배포를 선택하여 코어 디바이스로 배포를 시작합니다.
11. 배포에 성공했는지 확인하려면 배포 상태를 확인하고 코어 디바이스에 대한 로그를 확인합니다. 코어 디바이스에서 배포 상태를 확인하려면 배포 개요에서 대상을 선택하면 됩니다. 자세한 내용은 다음을 참조하세요.

- [배포 상태 확인](#)
- [AWS IoT Greengrass 로그 모니터링](#)

3단계: 클라이언트 디바이스 연결

클라이언트 디바이스를 사용하여 코어 디바이스를 검색, 연결 및 통신 AWS IoT Device SDK 할 수 있습니다. 클라이언트 디바이스는 AWS IoT 사물이어야 합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사물 객체 생성](#)을 참조하세요.

이 섹션에서는 [Python용 AWS IoT Device SDK v2](#)를 설치하고 AWS IoT Device SDK의 Greengrass 검색 샘플 애플리케이션을 실행합니다.

Note

AWS IoT Device SDK 는 다른 프로그래밍 언어로도 사용할 수 있습니다. 이 자습서에서는 Python용 AWS IoT Device SDK v2를 사용하지만, 사용 사례에 따라 다른 SDK를 탐색해도 됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 디바이스 SDK](#) 섹션을 참조하세요.

클라이언트 디바이스를 코어 디바이스에 연결하려면

1. 클라이언트 디바이스로 연결할 AWS IoT 사물에 [AWS IoT Device SDK Python용 v2](#)를 다운로드 하여 설치합니다.

클라이언트 디바이스에서 다음을 수행합니다.

- a. AWS IoT Device SDK v2 for Python 리포지토리를 복제하여 다운로드합니다.

```
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. AWS IoT Device SDK v2 for Python의 샘플 폴더로 변경합니다.

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

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 Hello World라는 메시지를 `clients/MyClientDevice1/hello/world` 주제로 보냅니다.

Note

이 주제는 메시지를 AWS IoT Core 이전에 릴레이하도록 MQTT 브리지를 구성한 주제와 일치합니다.

- 클라이언트 디바이스의 사물 이름으로 `MyClientDevice1`을 바꿉니다.
- `~/certs/AmazonRootCA1.pem`을 클라이언트 디바이스의 Amazon 루트 CA 인증서 경로로 바꿉니다.
- `~/certs/device.pem.crt`를 클라이언트 디바이스의 디바이스 인증서 경로로 바꿉니다.
- `~/certs/private.pem.key`를 클라이언트 디바이스의 프라이빗 키 파일 경로로 바꿉니다.
- `us-east-1`을 클라이언트 디바이스와 코어 디바이스가 작동하는 AWS 리전으로 바꿉니다.

```
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 10회 전송하고 연결을 해제합니다. 또한 메시지를 게시하는 주제와 동일한 주제를 구독합니다. 출력에서 애플리케이션이 해당 주제에 대한 MQTT 메시지를 수신했음을 나타내는 경우 클라이언트 디바이스는 코어 디바이스와 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}'
```

그 대신에 애플리케이션에서 오류가 출력되면 [Greengrass 검색 문제 해결](#)을 참조하세요.

코어 디바이스에서 Greengrass 로그를 보고 클라이언트 디바이스가 메시지를 성공적으로 연결하고 전송하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

4. MQTT 브리지가 클라이언트 디바이스의 메시지를 로 릴레이하는지 확인합니다 AWS IoT Core. AWS IoT Core 콘솔에서 MQTT 테스트 클라이언트를 사용하여 MQTT 주제 필터를 구독할 수 있습니다. 해결 방법:
 - a. [AWS IoT 콘솔](#)로 이동합니다.
 - b. 왼쪽 탐색 창의 테스트에서 MQTT 테스트 클라이언트를 선택합니다.
 - c. 주제 구독 탭에서 주제 필터에 대해 코어 디바이스의 클라이언트 디바이스 메시지를 구독할 `clients/+ /hello/world`을 입력합니다.
 - d. 구독을 선택합니다.
 - e. 다시 클라이언트 디바이스에서 게시/구독 애플리케이션을 실행합니다.

클라이언트 디바이스에서 보내는 이 주제 필터와 일치하는 주제에 대한 메시지가 MQTT 테스트 클라이언트에 표시됩니다.

4단계: 클라이언트 디바이스와 통신하는 구성 요소를 개발합니다.

클라이언트 디바이스와 통신하는 Greengrass 구성 요소를 개발할 수 있습니다. 구성 요소에서는 [프로세스 간 통신\(IPC\)](#)과 [로컬 게시/구독 인터페이스](#)를 사용하여 코어 디바이스에서 통신합니다. 클라이언트 디바이스와 상호 작용하려면 클라이언트 디바이스와 로컬 게시/구독 인터페이스 간 메시지를 릴레이할 MQTT 브리지 구성 요소를 구성합니다.

이 섹션에서는 클라이언트 디바이스에서 로컬 게시/구독 인터페이스로 메시지를 릴레이하는 MQTT 브리지 구성 요소를 업데이트합니다. 그런 다음에 이러한 메시지를 구독하고 수신 시 메시지를 인쇄하는 구성 요소를 개발합니다.

클라이언트 디바이스와 통신하는 구성 요소를 개발하려면

1. 배포를 코어 디바이스로 조정하고 클라이언트 디바이스에서 로컬 게시/구독으로 메시지를 릴레이할 MQTT 브리지를 구성합니다. 다음을 수행합니다.

- a. [AWS IoT Greengrass 콘솔](#)로 이동합니다.
- b. 왼쪽 탐색 메뉴에서 코어 디바이스를 선택합니다.
- c. 코어 디바이스 페이지에서 이 자습서에 사용하는 코어 디바이스를 선택합니다.
- d. 코어 디바이스 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
- e. 클라이언트 디바이스 탭에서 클라우드 검색 구성을 선택합니다.

코어 디바이스 검색 구성 페이지가 열립니다. 이 페이지에서 코어 디바이스에 배포되는 클라이언트 디바이스 구성 요소를 변경하거나 구성할 수 있습니다.

- f. 3단계에서 `aws.greengrass.clientdevices.mqtt.Bridge` 구성 요소에는 구성 편집을 선택합니다.
- g. MQTT 브리지 구성 요소에 대한 구성 편집 모달에서 MQTT 메시지를 클라이언트 디바이스에서 로컬 게시/구독 인터페이스로 릴레이하는 주제 매핑을 구성합니다. 다음을 수행합니다.
- i. 구성에서 다음 구성을 병합할 구성 코드 블록에 입력합니다. 이 구성을 통해 `clients/+ /hello/world` 주제 필터와 일치하는 주제에 대한 MQTT 메시지를 클라이언트 디바이스에서 AWS IoT Core 클라우드 서비스와 로컬 Greengrass 게시/구독 브로커로 릴레이하는 방법이 지정됩니다.

```
{
  "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 구성](#)을 참조하세요.

- ii. 확인을 선택합니다.

- h. 검토 및 배포를 선택하여 이 페이지에서 자동으로 생성되는 배포를 검토합니다.
- i. 검토 페이지에서 배포를 선택하여 코어 디바이스로 배포를 시작합니다.
- j. 배포에 성공했는지 확인하려면 배포 상태를 확인하고 코어 디바이스에 대한 로그를 확인합니다. 코어 디바이스에서 배포 상태를 확인하려면 배포 개요에서 대상을 선택하면 됩니다. 자세한 내용은 다음을 참조하세요.

- [배포 상태 확인](#)
- [AWS IoT Greengrass 로그 모니터링](#)

2. 클라이언트 디바이스의 Hello World 메시지를 구독하는 Greengrass 구성 요소를 개발하고 배포합니다. 다음을 수행합니다.

- a. 코어 디바이스에서 레시피 및 아티팩트 폴더를 생성합니다.

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 경로에 다음 형식을 사용해야 합니다. 레시피에서 지정하는 구성 요소 이름과 버전을 포함합니다.

```
artifacts/componentName/componentVersion/
```

- b. 텍스트 편집기를 사용하여 다음 내용으로 레시피를 생성합니다. 이 레시피는 Python용 AWS IoT Device SDK v2를 설치하고 주제를 구독하고 메시지를 인쇄하는 스크립트를 실행하도록 지정합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 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 awsiodtsdk",
```

```

        "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 내용이 있는 `hello_world_subscriber.py`라는 Python 스크립트 아티팩트를 생성합니다. 이 애플리케이션에서는 게시/구독 IPC 서비스를 사용하여 `clients/+ /hello/world` 주제를 구독하고 수신되는 메시지를 인쇄합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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 V2를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 통신합니다. [AWS IoT Device SDK](#) 원래 IPC 클라이언트에 비해 IPC 클라이언트 V2에서는 사용자 지정 구성 요소에서 IPC를 사용하려면 작성해야 할 코드 양이 감소합니다.

- 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"
```

- 구성 요소 로그를 보며 구성 요소가 설치되고 주제가 구독되는지 확인합니다.

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 열어놓고 코어 디바이스에서 메시지가 수신되는지 확인할 수 있습니다.

- 클라이언트 디바이스에서 다시 샘플 Greengrass 검색 애플리케이션을 실행하여 코어 디바이스로 메시지를 보냅니다.

```
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
```

- 다시 구성 요소 로그를 보며 구성 요소에서 클라이언트 디바이스의 메시지가 수신되고 인쇄되는지 확인합니다.

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 디바이스 새도와 상호 작용하는 구성 요소를 개발합니다.

클라이언트 디바이스의 [AWS IoT 디바이스 새도](#)와 상호 작용하는 Greengrass 구성 요소를 개발할 수 있습니다. 새도우는 클라이언트 디바이스와 같은 AWS IoT 사물에 대한 현재 또는 원하는 상태 정보를 저장하는 JSON 문서입니다. 사용자 지정 구성 요소에서는 클라이언트 디바이스가 AWS IoT에 연결되지 않았더라도 클라이언트 디바이스의 새도에 액세스하여 해당 상태를 관리할 수 있습니다. 각 AWS IoT 사물에는 이름이 지정되지 않은 새도우가 있으며, 각 사물에 대해 이름이 지정된 새도우를 여러 개 생성할 수도 있습니다.

이 섹션에서는 코어 디바이스에서 새도를 관리할 [새도 관리자 구성 요소](#)를 배포합니다. MQTT 브리지 구성 요소를 업데이트하여 클라이언트 디바이스와 새도 관리자 구성 요소 간 새도 메시지를 릴레이할 수도 있습니다. 그런 다음에 클라이언트 디바이스의 새도가 업데이트되는 구성 요소를 개발하고, 구성 요소의 새도 업데이트에 응답하는 클라이언트 디바이스에 샘플 애플리케이션을 실행할 수 있습니다. 이 구성 요소는 클라이언트 디바이스로 연결되는 스마트 조명의 색상 상태를 코어 디바이스에서 관리하는 스마트 조명 관리 애플리케이션을 나타냅니다.

클라이언트 디바이스 새도와 상호 작용하는 구성 요소를 개발하려면

- 배포를 새도 관리자 구성 요소를 배포할 코어 디바이스로 조정하고 새도 관리자에서 통신하는 클라이언트 디바이스와 로컬 게시/구독 간 새도 메시지를 릴레이할 MQTT 브리지 구성 요소를 구성합니다. 다음을 수행합니다.
 - [AWS IoT Greengrass 콘솔](#)로 이동합니다.
 - 왼쪽 탐색 메뉴에서 코어 디바이스를 선택합니다.
 - 코어 디바이스 페이지에서 이 자습서에 사용하는 코어 디바이스를 선택합니다.

- d. 코어 디바이스 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
- e. 클라이언트 디바이스 탭에서 클라우드 검색 구성을 선택합니다.

코어 디바이스 검색 구성 페이지가 열립니다. 이 페이지에서 코어 디바이스에 배포되는 클라이언트 디바이스 구성 요소를 변경하거나 구성할 수 있습니다.

- f. 3단계에서 `aws.greengrass.clientdevices.mqtt.Bridge` 구성 요소에는 구성 편집을 선택합니다.
- g. MQTT 브리지 구성 요소에 대한 구성 편집 모달에서 클라이언트 디바이스와 로컬 게시/구독 인터페이스 간 [디바이스 새도 주제](#)에 대한 MQTT 메시지를 릴레이하는 주제 매핑을 구성합니다. 배포에서 호환 MQTT 브리지 버전도 지정되는지 확인합니다. 클라이언트 디바이스 새도 지원에는 MQTT 브리지 v2.2.0 이상이 필요합니다. 다음을 수행합니다.
 - i. 구성 요소 버전에는 버전 2.2.0 이상을 선택합니다.
 - ii. 구성에서 다음 구성을 병합할 구성 코드 블록에 입력합니다. 이 구성을 통해 새도 주제에 대한 MQTT 메시지를 릴레이하는 방법이 지정됩니다.

```
{
  "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 구성](#)을 참조하세요.

iii. 확인을 선택합니다.

- h. 3단계에서 배포할 aws.greengrass.ShadowManager 구성 요소를 선택합니다.
- i. 검토 및 배포를 선택하여 이 페이지에서 자동으로 생성되는 배포를 검토합니다.
- j. 검토 페이지에서 배포를 선택하여 코어 디바이스로 배포를 시작합니다.
- k. 배포에 성공했는지 확인하려면 배포 상태를 확인하고 코어 디바이스에 대한 로그를 확인합니다. 코어 디바이스에서 배포 상태를 확인하려면 배포 개요에서 대상을 선택하면 됩니다. 자세한 내용은 다음을 참조하세요.

- [배포 상태 확인](#)

- [AWS IoT Greengrass 로그 모니터링](#)

2. 스마트 조명 클라이언트 디바이스를 관리하는 Greengrass 구성 요소를 개발하고 배포합니다. 다음을 수행합니다.
 - a. 코어 디바이스에서 구성 요소의 아티팩트 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 경로에 다음 형식을 사용해야 합니다. 레시피에서 지정하는 구성 요소 이름과 버전을 포함합니다.

```
artifacts/componentName/componentVersion/
```

- b. 텍스트 편집기를 사용하여 다음 내용으로 레시피를 생성합니다. 이 레시피는 Python용 AWS IoT Device SDK v2를 설치하고 스마트 조명 클라이언트 디바이스의 새도우와 상호 작용하는 스크립트를 실행하여 색상을 관리하도록 지정합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 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 내용이 있는 `smart_light_manager.py`라는 Python 스크립트 아티팩트를 생성합니다. 이 애플리케이션에서는 클라이언트 디바이스 새도를 가져오고

업데이트할 새도 IPC 서비스와 보고된 새도 업데이트를 받을 로컬 게시/구독 IPC 서비스를 사용합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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
                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 애플리케이션에서는 다음을 수행합니다.

- 구성 요소의 구성을 읽으며 관리할 스마트 조명 클라이언트 디바이스의 목록을 가져옵니다.
 - [SubscribeToConfigurationUpdate](#) IPC 작업을 사용하여 구성 업데이트 알림을 구독합니다. AWS IoT Greengrass 코어 소프트웨어는 구성 요소의 구성이 변경될 때마다 알림을 보냅니다. 구성 요소에서는 구성 업데이트 알림을 받으면 관리하는 스마트 조명 클라이언트 디바이스의 목록을 업데이트합니다.
 - 초기 색상 상태를 가져올 각 스마트 조명 클라이언트 디바이스의 색도를 가져옵니다.
 - 15초마다 각 스마트 조명 클라이언트 디바이스의 색상을 무작위 색상으로 설정합니다. 구성 요소에서는 클라이언트 디바이스의 사물 색도를 업데이트하여 색상을 변경합니다. 이 작업에서는 MQTT를 통해 클라이언트 디바이스로 색도 델타 이벤트를 보냅니다.
 - [SubscribeToTopic](#) IPC 작업을 사용하여 로컬 게시/구독 인터페이스에서 색도 업데이트 승인 메시지를 구독합니다. 이 구성 요소에서는 이러한 메시지를 받아서 각 스마트 조명 클라이언트 디바이스의 색상을 추적합니다. 스마트 조명 클라이언트 디바이스에서는 색도 업데이트를 받으면 업데이트를 받았다는 것이 확인되는 MQTT 메시지를 보냅니다. MQTT 브리지에서는 이 메시지를 로컬 게시/구독 인터페이스로 릴레이합니다.
- d. Greengrass CLI를 사용하여 구성 요소를 배포합니다. 이 구성 요소를 배포할 때 색도를 관리하는 클라이언트 디바이스 목록(`smartLightDeviceNames`)을 지정합니다. 클라이언트 디바이스의 사물 이름으로 *MyClientDevice1*을 바꿉니다.

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 보며 구성 요소가 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 요소에서는 스마트 조명 클라이언트 디바이스의 색상을 변경하라는 요청을 보냅니다. 새도 관리자에서는 요청을 받고 새도의 `desired` 상태를 설정합니다. 그러나 스마트 조명 클라이언트 디바이스가 아직 실행되지 않으므로 새도의 `reported` 상태가 변경되지 않습니다. 구성 요소의 로그에는 다음 메시지가 포함되어 있습니다.

```
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 열어놓고 언제 구성 요소에서 메시지가 인쇄되는지 참조할 수 있습니다.

4. Greengrass 검색을 사용하고 디바이스 새도 업데이트를 구독하는 샘플 애플리케이션을 다운로드 하고 실행합니다. 클라이언트 디바이스에서 다음을 수행합니다.
 - a. 텍스트 편집기를 사용하여 다음 내용이 있는 `basic_discovery_shadow.py`라는 Python 스크립트를 생성합니다. 이 애플리케이션에서는 Greengrass 검색과 새도를 사용하여 클라이언트 디바이스와 코어 디바이스 사이에 속성을 동기화 상태로 유지합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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

# ----- ARGUMENT PARSING
# -----
import argparse

def parse_sample_input():
    parser = argparse.ArgumentParser(
        description="AWS IoT Greengrass Discovery Shadow",
        formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    )

    # Connection / TLS
    parser.add_argument("--cert", required=True, dest="input_cert",
                        help="Path to the certificate file to use during mTLS connection establishment")
    parser.add_argument("--key", required=True, dest="input_key",
                        help="Path to the private key file to use during mTLS connection establishment")
    parser.add_argument("--ca_file", dest="input_ca", help="Path to optional CA bundle (PEM)")

    # Shadow
    parser.add_argument("--thing_name", required=True, dest="input_thing_name",
                        help="The name assigned to your IoT Thing.")
    parser.add_argument("--shadow_property", dest="input_shadow_property",
                        default="color",
                        help="The name of the shadow property you want to change (optional, default='color')")
    parser.add_argument("--region", dest="input_region", help="The region to connect through.", required=True)
    parser.add_argument("--print_discover_resp_only",
                        dest="input_print_discover_resp_only", default=False)

    return parser.parse_args()

args = parse_sample_input()

```

```

# ----- ARGUMENT PARSING END
# -----

# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = args.input_thing_name
shadow_property = args.input_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_interrupted(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=args.input_cert,
                        pri_key_filepath=args.input_key,

```

```
ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
        on_connection_interrupted=on_connection_interrupted,
        on_connection_resumed=on_connection_resumed,
        client_id=args.input_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(args.input_cert,args.input_key)
    if args.input_ca:
        tls_options.override_default_trust_store_from_path(None,
args.input_ca)
    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, args.input_region)
    resp_future = discovery_client.discover(args.input_thing_name)
    discover_response = resp_future.result()

    print(discover_response)
    if args.print_discover_resp_only:
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.

```

```
# Note that it 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 애플리케이션에서는 다음을 수행합니다.

- Greengrass 검색을 사용하여 검색하고 코어 디바이스에 연결합니다.
- 코어 디바이스에 새도 문서를 요청하여 속성의 초기 상태를 가져옵니다.

- 속성의 `desired` 값이 자체 `reported` 값과 다를 때 코어 디바이스에서 보내는 새도 델타 이벤트를 구독합니다. 애플리케이션에서는 새도 델타 이벤트를 받으면 속성의 값을 변경하고 해당 `reported` 값으로 새 값을 설정할 코어 디바이스로 업데이트를 보냅니다.

이 애플리케이션에서는 Greengrass 검색과 AWS IoT Device SDK v2의 새도 샘플을 결합합니다.

- b. 샘플 애플리케이션을 실행합니다. 이 애플리케이션에는 클라이언트 디바이스 사물 이름, 사용할 새도 속성, 연결을 인증하고 보호하는 인증서를 지정하는 인수가 필요합니다.
 - 클라이언트 디바이스의 사물 이름으로 `MyClientDevice1`을 바꿉니다.
 - `~/certs/AmazonRootCA1.pem`을 클라이언트 디바이스의 Amazon 루트 CA 인증서 경로로 바꿉니다.
 - `~/certs/device.pem.crt`를 클라이언트 디바이스의 디바이스 인증서 경로로 바꿉니다.
 - `~/certs/private.pem.key`를 클라이언트 디바이스의 프라이빗 키 파일 경로로 바꿉니다.
 - `us-east-1`을 클라이언트 디바이스 및 코어 디바이스가 작동하는 AWS 리전으로 바꿉니다.

```
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 애플리케이션에서는 새도 주제를 구독하며 코어 디바이스의 새도 델타 이벤트를 받을 때까지 기다립니다. 애플리케이션에서 새도 델타 이벤트를 받고 응답하는 것으로 출력에 나타나면 클라이언트 디바이스에서는 코어 디바이스의 새도와 상호 작용할 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-----
\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.

```

그 대신에 애플리케이션에서 오류가 출력되면 [Greengrass 검색 문제 해결](#)을 참조하세요.

코어 디바이스에서 Greengrass 로그를 보고 클라이언트 디바이스가 메시지를 성공적으로 연결하고 전송하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

5. 다시 구성 요소 로그를 보며 구성 요소에서 스마트 조명 클라이언트 디바이스의 색도 업데이트 확인을 받는지 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 요소에서는 스마트 조명 클라이언트 디바이스의 색상이 변경되었다는 것이 확인되는 메시지를 로그에 남깁니다.

```

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

코어 디바이스와 클라이언트 디바이스 사이에 클라이언트 디바이스의 색도가 동기화됩니다. 그러나 코어 디바이스는 클라이언트 디바이스의 색도우를와 동기화하지 않습니다 AWS IoT Core. 예를 들어 색도우를와 동기화 AWS IoT Core 하여 플릿에 있는 모든 디바이스의 상태를 보거나 수정할 수 있습니다. 색도를 동기화하도록 색도 관리자 구성 요소를 구성하는 방법에 대한 자세한 내용은 섹션을 AWS IoT Core참조하세요 [로컬 디바이스 색도우를와 동기화 AWS IoT Core](#).

이 자습서를 완료했습니다. 클라이언트 디바이스는 코어 디바이스에 연결하고, MQTT 메시지를 AWS IoT Core 및 Greengrass 구성 요소로 보내고, 코어 디바이스에서 색도우 업데이트를 수신합니다. 이 자습서에서 다른 주제에 대한 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라이언트 디바이스 통신 테스트](#)
- [Greengrass Discovery RESTful API](#)
- [클라이언트 디바이스와 간에 MQTT 메시지 릴레이 AWS IoT Core](#)
- [구성 요소의 클라이언트 디바이스와 상호 작용](#)

- [디바이스 새도와 연동](#)
- [클라이언트 디바이스 새도와 상호 작용 및 동기화](#)

자습서: 신뢰할 수 있는 플랫폼 모듈(TPM)을 사용하여 Greengrass Nucleus 보호

Note

이 자습서의 메커니즘은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#)에서만 지원됩니다.

이 자습서에는 TPM2 칩을 하드웨어 보안 모듈(HSM)로 사용하여 프라이빗 키와 CSR을 생성하는 방법에 대한 지침이 포함되어 있습니다. 이것은 [사물 인증서 생성](#)에 사용됩니다.

이 자습서에서는 PKCS#11 인터페이스를 사용하여 신뢰할 수 있는 플랫폼 모듈(TPM)로 AWS IoT Greengrass 코어 소프트웨어(Greengrass Nucleus)를 구성하여 디바이스 보안을 강화하는 방법을 보여줍니다. 이 TPM 통합은 디바이스 자격 증명 및에 연결하는 데 사용되는 프라이빗 키와 인증서를 변조 방지 하드웨어에 AWS IoT Core 안전하게 저장하여 위장 또는 기타 악의적인 활동을 위한 추출을 방지합니다.

이 통합을 완료하면 Greengrass 코어 디바이스는 ID 및 AWS IoT 서비스와의 통신에 TPM 보호 프라이빗 키를 사용합니다.

Greengrass 디바이스 보안에 대한 자세한 내용은 [의 보안 AWS IoT Greengrass](#) 섹션을 참조하세요.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- TPM 2.0 하드웨어 또는 펌웨어 TPM 2.0이 있는 Linux 호환 디바이스.
- 이 자습서의 지침은 Ubuntu 24.04 LTS를 기준으로 정의됩니다.
 - [Linux TPM2 소프트웨어 스택](#)을 지원하는 모든 Linux 배포판에서 이 메커니즘 지원이 가능합니다.
- 가 AWS CLI 설치되고 다음에 대한 권한으로 구성된 개발자 시스템:
 - AWS IoT 리소스 생성 및 관리
 - IAM 역할 및 정책 생성 및 관리
- 디바이스에 설치된 Java 런타임 환경(JRE) 버전 8 이상.

- 디바이스에 설치된 다음 소프트웨어 패키지:
 - curl
 - jq
- 디바이스에서 루트 권한 또는 sudo 권한.

1단계: TPM2 도구 및 종속성 설치

이 단계에서는 필요한 TPM2 소프트웨어 도구와 라이브러리를 설치합니다.

1. 패키지 관리자를 업데이트하고 다음 명령을 실행하여 TPM2 도구와 종속성을 설치합니다.

```
sudo apt-get update && sudo apt-get install tpm2-tools \
tpm2-abrmd \
tpm2-tss-engine-tools \
gnutls-bin \
libtpm2-pkcs11-1 \
libtpm2-pkcs11-tools \
libtpm2-pkcs11-1-dev \
python3-tpm2-pkcs11-tools \
libengine-pkcs11-openssl \
libtss2-tcti-tabrmd0
```

2. OpenSSL 엔진 3을 사용하는 OpenSSL TPM2 공급자 패키지를 Ubuntu 24.04에 설치합니다.

```
sudo apt-get install tpm2-openssl
```

2단계: PKCS#11 저장소 초기화 및 슬롯 생성

1. 데이터를 저장할 디렉터리를 생성합니다.

```
sudo mkdir -p /etc/tpm2_pkcs11
```

2. 저장 위치를 환경 변수로 설정합니다. 스토어 계층 구조에 대한 자세한 내용은 [초기화](#)를 참조하세요.

```
export TPM2_PKCS11_STORE=/etc/tpm2_store
```

3. 기본 객체를 사용하여 TPM2 토큰을 초기화합니다.

```
sudo tpm2_ptool init
```

사용 가능한 옵션은 다음과 같습니다.

`hierarchy-auth HIERARCHY_AUTH`

계층 구조에 기본 객체를 추가할 때 사용하는 인증 비밀번호.

`primary-auth PRIMARY_AUTH`

기본 프라이머리 키 객체에 대한 인증 값.

기본값은 빈 인증 값입니다.

`primary-handle [PRIMARY_HANDLE]`

기본 프라이머리 키 객체를 사용합니다.

기본값: 0x81000001

`transient-parents`

지정된 템플릿의 임시 기본 객체를 사용합니다.

값: tpm2-tools-default, tpm2-tools-ecc-default, tss2-engine-key

`path PATH`

스토어 디렉터리의 위치입니다. 지정한 경우 디렉터리가 존재해야 합니다. 지정하지 않으면 환경 변수 `TPM2_PKCS11_STORE`를 확인하여 검색합니다. 환경 변수도 설정되어 있지 않으면 `/etc/tpm2_pkcs11`을 확인합니다. 해당 디렉터리를 찾을 수 없거나 생성할 수 없으면 기본적으로 현재 작업 디렉터리로 설정됩니다.

3단계: 토큰 및 키 생성

1. PKCS#11 토큰을 생성합니다.

```
sudo tpm2_ptool addtoken -pid=1 -userpin=USERPIN -sopin=SOPIN -label=greengrass
```

사용 가능한 옵션은 다음과 같습니다.

`--pid PID`

이 토큰과 연결할 기본 객체 ID입니다.

`--sopin SOPIN`

관리자 핀입니다. 이 핀은 객체 복구에 사용됩니다.

`--userpin USERPIN`

사용자 핀입니다. 이 핀은 객체 사용을 위한 인증에 사용됩니다.

`--label LABEL`

사용 중인 프로파일을 식별하는 고유 레이블은 고유해야 합니다.

`--hierarchy-auth HIERARCHY_AUTH`

2. ECC 키 객체를 생성합니다.

```
sudo tpm2_ptool addkey -algorithm=ecc256 -label=greengrass -userpin=***** -key-label=greenkey
```

`--label LABEL`

키를 가져올 토큰의 레이블을 지정합니다.

`--key-label KEY_LABEL`

가져온 키의 레이블을 지정합니다. 기본값은 정수 값입니다.

--id ID

키 ID입니다. 기본값은 8바이트의 무작위 16진수입니다.

--attr-always-authenticate

CKA_ALWAYS_AUTHENTICATE 속성을 CK_TRUE로 설정합니다.

--hierarchy-auth HIERARCHY_AUTH

임시 객체에 필요한 계층 인증을 지정합니다.

--sopin SOPIN

관리자 핀입니다.

--userpin USERPIN

사용자 핀입니다.

--algorithm

{rsa1024,rsa2048,rsa3072,rsa4096,aes128,aes256,ecc224,ecc256,ecc384,ecc521,hmac:sha1,hmac:sha256}

키 유형입니다.

3. 토큰에서 TPM2-TSS 개체를 내보내 인증 데이터를 캡처합니다.

```
yaml_ecc0=$(sudo tpm2_ptool export --label="greengrass" --key-label="greenkey" --
userpin="*****")
```

출력 예시:

```
> echo $yaml_ecc0
object-auth: 706c1cad8a5238871b30149705255926
primary-object:
  auth: ''
  hierarchy: owner
  is_transient: false
```

Note

이 명령을 실행한 디렉토리에는 TSS2 프라이빗 키인 `greenkey.pem`이라는 파일이 함께 생성됩니다. 이 키를 사용하여 `tpm2` OpenSSL 공급자로 CSR을 생성할 수 있습니다. TSS2 PRIVATE KEY 파일은 TPM에 의해 보호되며 다른 시스템에서 사용할 수 없습니다. TSS2 키와 OpenSSL 관련 자세한 내용은 [프라이빗 또는 퍼블릭 키 저장](#)을 참조하세요.

4. TSS 프라이빗 키의 인증 데이터를 캡처합니다.

```
auth_ecc0=$(echo "$yaml_ecc0" | grep "object-auth" | cut -d' ' -f2-)
```

4단계: 인증서 서명 요청(CSR) 생성

이 단계에서는 TPM2로 보호된 프라이빗 키를 사용하여 CSR을 생성합니다.

1. TPM2 공급자를 사용하여 CSR을 생성합니다.

```
sudo openssl req -new -provider tpm2 -provider base -key greenkey.pem -passin "pass:$auth_ecc0" -out "$H"$HOSTNAME".csr"
```

CSR 생성을 진행하면 다음 정보를 입력하라는 메시지가 표시됩니다.

- 국가명(2자리 코드)
- 주 또는 지방 이름
- 시 이름
- 조직 이름
- 조직 단위 이름
- 일반 이름
- 이메일 주소

또는 OpenSSL 구성 파일을 사용하여 무인으로 CSR을 생성할 수도 있습니다. 자세한 내용은 [OpenSSL 설명서](#)를 참조하세요.

2. 동일한 시스템에서 CSR을 생성하지 않는 경우 생성된 CSR을 AWS 자격 증명이 구성된 시스템에 복사합니다.

5단계: 사물 인증서 생성

AWS IoT 사물 인증서를 생성합니다. 사물 인증서 요청을 승인하는 방법에 대한 자세한 내용은 [사물 인증서 생성](#) 섹션을 참조하세요.

6단계: 사물 인증서를 TPM에 가져오기

1. 사물 인증서를 디바이스에 복사합니다.
2. 사물 인증서를 Greengrass 토큰에 추가합니다.

```
sudo tpm2_ptool addcert --label=greengrass --key-label=greenkey device.pem.crt
```

사용 가능한 옵션은 다음과 같습니다.

`--help`

도움말 메시지를 표시하고 종료합니다.

`--label LABEL`

제거할 프로파일 레이블을 지정합니다.

`--key-label KEY_LABEL`

관련된 프라이빗 키 레이블을 지정합니다.

`--key-id KEY_ID`

관련된 프라이빗 키 ID를 16진수로 지정합니다.

`cert`

추가할 X.509 PEM 인증서를 지정합니다.

7단계: PKCS#11 객체 URL 캡처

gnutls-bin 패키지에서 제공하는 p11tool을 사용하여 PKCS#11 토큰 URL 및 객체 URL을 가져옵니다.

1. Greengrass 토큰의 토큰 URL을 캡처합니다.

```
TOKEN=$(sudo p11tool --list-token-urls | grep "token=greengrass")
```

2. Greengrass 토큰의 객체 URL을 가져옵니다. 3단계에서 사용한 것과 동일한 핀을 사용합니다.

```
sudo p11tool --login --list-all "${TOKEN}"
```

출력 예시:

```
Token 'greengrass' with URL
'pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
requires user PIN
Enter PIN:
WARNING: Needed CKA_VALUE but didn't find encrypted blob
Object 0:
    URL:
    pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=private
    Type: Private key (EC/ECDSA-SECP256R1)
    Label: greenkey
    Flags: CKA_PRIVATE; CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
    ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 1:
    URL:
    pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=public
    Type: Public key (EC/ECDSA-SECP256R1)
    Label: greenkey
    ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 2:
    URL:
    pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=cert
    Type: X.509 Certificate (EC/ECDSA-SECP256R1)
    Expires: Fri Dec 31 18:59:59 2049
    Label: greenkey
    ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31
```

3. 프라이빗 키 및 인증서의 객체 URL을 캡처합니다.

8단계: TPM2 지원을 사용하여 Greengrass 구성 및 설치

1. 사물 인증서를 구성합니다. 자세한 내용은 [사물 인증서 구성](#)을 참조하세요.
2. 의 HSM에 프라이빗 키와 인증서를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치하는 지침을 완료합니다. [AWS IoT Greengrass 코어 소프트웨어 설치](#). 그런 후, 다음 단계를 따라 PKCS#11 인터페이스를 통해 TPM2를 활용하도록 설치 구성을 진행합니다.
3. Greengrass 설치 프로그램 위치에 PKCS#11 공급자 구성 요소가 다운로드되어 저장되어 있는지 확인합니다.
4. 텍스트 편집기를 사용하여 config.yaml 구성 파일을 생성하고 설치 관리자에 제공합니다. 예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

5. 다음 YAML 내용을 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터, Greengrass nucleus 파라미터 및 PKCS#11 공급자 파라미터를 지정합니다.

```
---
system:
  certificateFilePath: "pkcs11:model=SW%20%20%20TPM
%00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
%34%35;object=greenkey;type=cert"
  privateKeyPath: "pkcs11:model=SW%20%20%20TPM
%00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
%34%35;object=greenkey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "myThing"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.14.0"
    configuration:
      awsRegion: "us-east-1"
      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: "tpm2_pkcs11"
library: "/usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so"
slot: 1
userPin: "123456"
```

6. 설치 환경에 맞는 파라미터로 파일을 편집합니다.

- a. 7단계에서 캡처한 PKCS#11 URL을 사용하여 certificateFilePath 및 privateKeyPath를 Update certificateFilePath 및 privateKeyPath로 업데이트합니다.
- b. AWS IoT 엔드포인트를 기반으로 iotDataEndpoint 및 iotCredEndpoint를 업데이트합니다.
- c. `aws.greengrass.crypto.Pkcs11Provider` 구성에서 플랫폼을 기반으로 라이브러리를 업데이트합니다.

Note

위 예시는 X86_64 아키텍처 기준입니다. ARM64 디바이스의 경우 파일 경로가 유사하나 다를 수 있습니다.

7. [AWS IoT Greengrass 코어 소프트웨어 설치](#)에서 Greengrass 설치 단계를 완료합니다.

9단계: 설치 확인

이 단계에서는 TPM2 통합이 적용된 Greengrass가 올바르게 실행되고 있는지를 확인합니다.

1. Greengrass 서비스 상태를 확인합니다.

```
sudo systemctl status greengrass.service
```

2. 오류가 없는지 확인하기 위해 Greengrass 로그를 확인합니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

3. [AWS IoT 콘솔](#)에서 디바이스 연결 상태를 확인합니다.

- a. [AWS IoT Greengrass 콘솔](#)에 로그인합니다.
- b. 관리에서 Greengrass 디바이스를 확장하고 코어 디바이스를 선택합니다.
- c. 디바이스가 연결되어 있는지 확인합니다. 디바이스가 연결되어 있으면 상태가 HEALTHY로 표시 됩니다. 자세한 내용은 [Greengrass 코어 디바이스 상태 확인](#) 단원을 참조하십시오.

문제 해결

TPM2가 활성화된 Greengrass 디바이스 설정 또는 작동 중에 문제가 발생하면 다음 문제 해결 단계를 시도하세요.

- 주요 Greengrass 로그 파일을 확인합니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

- PKCS#11 공급자 구성을 확인합니다.

```
sudo cat /greengrass/v2/config/effectiveConfig.yaml
```

- TPM2 서비스가 실행 중인지 확인합니다.

```
sudo systemctl status tpm2-abrmd.service
```

- TPM2 키에 액세스할 수 있는지 확인합니다.

```
sudo pkcs11-tool -module /usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so -l -p  
123456 -list-objects
```

- 운영 체제에서 Clevis 또는 systemd-cryptenroll과 같은 TPM2 저장소 루트 키를 사용하는 전체 디스크 암호화가 설정되어 있는 경우, 이 도구들이 사용하는 영구 핸들과 동일한 핸들을 사용하지 않았는지 확인합니다. 동일한 핸들을 사용할 경우, 디스크 암호화 메커니즘에 영향을 줄 수 있습니다. 모든 생성 및 사용 중인 영구 핸들을 확인하려면 다음 명령을 실행합니다.

```
sudo tpm2_getcap handles-persistent
```

다음 단계

TPM2가 통합된 Greengrass 코어 디바이스 구성을 성공적으로 완료했으므로, 이제 다음 작업을 수행할 수 있습니다.

- 보안이 강화된 Greengrass 디바이스에 구성 요소를 배포합니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.
- TPM2 통합이 적용된 Greengrass 디바이스를 추가로 설정합니다.

Greengrass 디바이스 보안에 대한 자세한 내용은 [의 보안 AWS IoT Greengrass](#) 섹션을 참조하세요.

자습서: 신뢰할 수 있는 플랫폼 모듈(TPM)을 사용하여 AWS IoT Greengrass Nucleus Lite 보호

이 자습서에서는 AWS IoT Greengrass nucleus lite에 대한 신뢰할 수 있는 플랫폼 모듈(TPM) 지원을 활성화하고 구성하는 방법을 설명합니다. TPM은 보안 키 스토리지를 위한 하드웨어 기반 신뢰 루트를 제공합니다. 이 보안 기능은 암호화 작업과 민감한 자격 증명을 보호하여 디바이스 보안 및 무결성을 향상시킵니다.

이 통합을 완료하면 AWS IoT Greengrass 코어 디바이스는 ID 및 AWS IoT 서비스와의 통신에 TPM 보호 프라이빗 키를 사용합니다.

AWS IoT Greengrass 디바이스의 보안에 대한 자세한 내용은 섹션을 참조하세요 [의 보안 AWS IoT Greengrass](#).

Important

이 메커니즘은 수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass nucleus lite를 설치하는 경우에만 지원됩니다.

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- TPM 2.0 하드웨어 또는 [NitroTPM](#)이 있는 Linux 호환 디바이스
- Greengrass Nucleus Lite가 설치된 개발자 시스템입니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 설치\(콘솔\)](#)를 참조하세요.
- 이 자습서의 지침은 Ubuntu 24.04 LTS를 기준으로 정의됩니다.
- [Linux TPM2 소프트웨어 스택](#)을 지원하는 모든 Linux 배포판에서 이 메커니즘 지원이 가능합니다.
- [AWS CLI](#)가 설치되고 다음 권한으로 구성된 개발용 머신:
 - AWS IoT 리소스 생성 및 관리
 - IAM 역할 및 정책 생성 및 관리
- 디바이스에서 루트 권한 또는 sudo 권한.

이 자습서에는 TPM2 칩을 하드웨어 보안 모듈(HSM)로 사용하여 프라이빗 키와 AWS IoT 사물 인증서를 생성하는 데 사용되는 CSR을 생성하는 방법에 대한 지침이 포함되어 있습니다.

1단계: NitroTPM 인스턴스 설정

1. NitroTPM 인스턴스를 설정합니다. 자세한 내용은 [NitroTPM](#)을 참조하세요.
2. 이전 단계에서 생성한 사용자 지정 AMI를 사용하여 최종 인스턴스를 시작합니다.

Important

SSH에 연결할 때는 대신 ubuntu 사용자를 사용합니다. root.

3. 다음 명령을 실행하여 TPM 디바이스가 있고 작동하는지 확인합니다.

```
ls -la /dev/tpm*
```

/dev/tpm0 및 /dev/tpmrm0 디바이스가 표시되어야 합니다.

2단계: TPM 도구 설치 및 구성

1. 다음 명령을 실행하여 필요한 패키지를 설치합니다.

```
sudo apt update
sudo apt install tpm2-openssl tpm2-tools tpm2-abrmd libtss2-tcti-tabrmd0
```

2. 다음 명령을 실행하여 TPM 디바이스의 권한을 확인합니다.

```
ls -l /dev/tpm0      # Should be owned by tss:root with permissions 0660
ls -l /dev/tpmrm0   # Should be owned by tss:tss with permissions 0660
```

3단계: OpenSSL TPM2 공급자 구성

1. OpenSSL 구성 파일을 편집합니다.

```
sudo vi /etc/ssl/openssl.cnf
```

2. 다음 구성을 추가합니다.

```
[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
tpm2 = tpm2_sect

[default_sect]
activate = 1

[tpm2_sect]
identity = tpm2
module = /usr/local/lib64/tpm2.so
activate = 1
```

3. 필요에 따라 모듈 경로를 조정합니다. 다음을 사용하여 올바른 경로를 찾을 수 있습니다.

```
find /usr -name "tpm2.so"
```

4단계: 영구 TPM 키 생성

1. 다음 명령을 실행하여 기본 키를 생성합니다.

```
sudo tpm2_createprimary -C o -c primary.ctx
```

2. 다음 명령을 실행하여 ECC 키 객체를 생성합니다.

```
sudo tpm2_create -C primary.ctx -g sha256 -G ecc256 -r device.priv -u device.pub
```

3. 다음 명령을 실행하여 키를 로드합니다.

```
sudo tpm2_load -C primary.ctx -r device.priv -u device.pub -c device.ctx
```

4. 다음 명령을 실행하여 키를 영구화합니다.

```
sudo tpm2_evictcontrol -C o -c device.ctx 0x81000002
```

이렇게 하면 핸들(예:)이 있는 영구 키가 생성됩니다 0x81000002.

5단계: 인증서 서명 요청(CSR) 생성

이 단계에서는 TPM2-protected 프라이빗 키를 사용하여 인증서 서명 요청(CSR)을 생성합니다.

1. TPM 키를 사용하여 CSR을 생성합니다.

```
openssl req -new -provider tpm2 -key "handle:0x81000002" \
  -out device.csr \
  -subj "/CN=TPMThing"
```

2. 0x81000002를 선택한 핸들 값으로 바꾸고 TPMThing를 원하는 사물 이름으로 바꿉니다.

6단계: CSR에서 인증서 생성

1. 개발 컴퓨터에서 AWS IoT 사물에 대한 인증서를 다운로드한 폴더를 생성합니다.

```
mkdir greengrass-v2-certs
```

2. CSR 파일을 사용하여 AWS IoT 사물에 대한 인증서를 생성하고 개발 컴퓨터에 다운로드합니다.

```
aws iot create-certificate-from-csr \
  --set-as-active \
  --certificate-signing-request file://path_to_device.csr \
  --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

요청에 성공하는 경우 응답은 다음 예제와 비슷하게 보입니다.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
  "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBIDELMAKGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xZDASBgNVBASTC0lBTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXRhbnQ2LsYWMxH3AdBgkqhkiG9w0BCQEWEG5vb251QGFtYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAKGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xZDASBgNVBASTC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2LsYWMx
H3AdBgkqhkiG9w0BCQEWEG5vb251QGFtYXpvi5jb20wgZ8wDQYJKoZIhvcNAQE
```

```
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLyGVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

7단계: TPM 지원으로 Greengrass Nucleus Lite 설정

Greengrass Nucleus Lite에서 TPM 지원을 활성화하려면 다음과 같이 변경합니다.

1. TPM 액세스를 위해 tss 그룹에 ggcore 사용자를 추가하여 사용자 권한을 구성합니다.

```
sudo usermod -a -G tss ggcore
```

2. 다음 단계를 완료하여 자격 증명 디렉터리를 업데이트합니다.

- 자격 증명 디렉터리에서 프라이빗 키 파일을 제거합니다.
- 영구 TPM 키를 사용하므로 프라이빗 키 파일을 복사할 필요가 없습니다.
- 새를 해당 자격 증명 디렉터리 `device.pem.crt`에 복사합니다.

3. 다음 TPM별 구성 `config.yaml`으로 편집합니다.

```
system:
  privateKeyPath: "handle:0x81000002" # Use your chosen handle
  certificateFilePath: "" # Replace with the path of device.pem.crt
  ...
```

4. 다음 명령을 실행하여 Greengrass Nucleus Lite를 재부팅합니다.

```
systemctl restart greengrass-lite.target
```

문제 해결

TPM2-enabled AWS IoT Greengrass 디바이스를 설정하거나 작동하는 동안 문제가 발생하면 다음 문제 해결 단계를 시도하세요.

TPM 디바이스를 찾을 수 없음(NitroTPM 사용 시)

/dev/tpm0이 없는 경우 다음 단계를 수행합니다.

1. NitroTPM 지원 인스턴스 유형을 사용하고 있는지 확인합니다.
2. 를 사용하여 AMI가 생성되었는지 확인합니다--tpm-support v2.0.
3. 인스턴스가 사용자 지정 AMI에서 시작되었는지 확인합니다.

권한 거부 오류

TPM 액세스 오류가 발생하면 다음을 수행합니다.

1. 사용자가 tss 그룹에 있는지 확인합니다groups \$USER.
2. 다음 명령을 사용하여 TPM 디바이스의 권한을 확인합니다.

```
ls -l /dev/tpm*
```

3. tss 그룹에를 추가한 후 로그아웃했다가 다시 로그인했는지 확인합니다.

OpenSSL 공급자 문제

TPM 공급자를 찾을 수 없는 경우 다음을 수행합니다.

1. 에서 tpm2.so 경로를 확인합니다/etc/ssl/openssl.cnf.
2. 다음 명령을 실행하여 공급자 설치를 확인합니다.

```
openssl list -providers
```

3. tpm2-openssl 패키지가 제대로 설치되었는지 확인합니다.

다음 단계

이제 AWS IoT Greengrass 코어 디바이스를 TPM2와 성공적으로 통합했으므로 다음을 수행할 수 있습니다.

- 보안 AWS IoT Greengrass 디바이스에 구성 요소 배포
- TPM2 통합을 사용하여 추가 AWS IoT Greengrass 디바이스 설정

AWS IoT Greengrass 디바이스의 보안에 대한 자세한 내용은 섹션을 참조하세요 [의 보안 AWS IoT Greengrass](#).

자습서: SageMaker AI Edge Manager 시작하기

Important

SageMaker AI Edge Manager는 2024년 4월 26일에 중단되었습니다. 엣지 디바이스에 모델을 계속 배포하는 방법에 대한 자세한 내용은 [SageMaker AI Edge Manager 수명 종료](#)를 참조하세요.

Amazon SageMaker AI Edge Manager는 엣지 디바이스에서 실행되는 소프트웨어 에이전트입니다. SageMaker AI Edge Manager는 엣지 디바이스에 대한 모델 관리를 제공하므로 Greengrass 코어 디바이스에서 Amazon SageMaker AI Neo 컴파일 모델을 직접 패키징하고 사용할 수 있습니다. SageMaker AI Edge Manager를 사용하면 코어 디바이스에서 모델 입력 및 출력 데이터를 샘플링하고 모니터링 및 분석을 AWS 클라우드 위해 해당 데이터를 로 보낼 수도 있습니다. SageMaker AI Edge Manager가 Greengrass 코어 디바이스에서 작동하는 방식에 대한 자세한 내용은 [Greengrass 코어 디바이스에서 Amazon SageMaker AI Edge Manager 사용](#) 섹션을 참조하세요.

이 자습서에서는 기존 코어 디바이스에서 AWS제공 샘플 구성 요소와 함께 SageMaker AI Edge Manager 사용을 시작하는 방법을 보여줍니다. 이러한 샘플 구성 요소는 SageMaker AI Edge Manager 구성 요소를 종속성으로 사용하여 Edge Manager 에이전트를 배포하고 SageMaker AI Neo를 사용하여 컴파일된 사전 훈련 모델로 추론을 수행합니다. SageMaker AI Edge Manager 에이전트에 대한 자세한 내용은 Amazon SageMaker AI 개발자 안내서의 [SageMaker AI Edge Manager](#)를 참조하세요.

기존 Greengrass 코어 디바이스에서 SageMaker AI Edge Manager 에이전트를 설정하고 사용하려면 다음 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`
- 객체 감지
 - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
 - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

이 자습서에서는 샘플 구성 요소와 SageMaker AI Edge Manager 에이전트를 배포하는 방법을 보여줍니다.

주제

- [사전 조건](#)
- [SageMaker AI Edge Manager에서 Greengrass 코어 디바이스 설정](#)
- [샘플 구성 요소 생성](#)
- [샘플 이미지 분류 추론 실행](#)

사전 조건

이 튜토리얼을 완료하려면 다음 사전 조건을 충족해야 합니다.

- Amazon Linux 2, Debian 기반 Linux 플랫폼(x86_64 또는 Armv8) 또는 Windows(x86_64)에서 실행되는 Greengrass 코어 디바이스. 없으면 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요.
- [Python](#) 3.6 이상(사용하는 Python 버전에 대한 pip 포함)이 코어 디바이스에 설치되어 있어야 합니다.
- OpenGL API GLX 런타임(libgl1-mesa-glx)이 코어 디바이스에 설치되어 있어야 합니다.
- 관리자 권한이 있는 AWS Identity and Access Management (IAM) 사용자.
- 다음과 같은 요구 사항을 충족하는 인터넷이 활성화된 Windows, Mac 또는 Unix 계열 개발 컴퓨터:
 - [Python](#) 버전 3.6 이상 설치.
 - AWS CLI IAM 관리자 사용자 자격 증명으로 설치 및 구성됩니다. 자세한 내용은 [AWS CLI 설치](#) 및 [AWS CLI 구성](#)을 참조하세요.
- Greengrass 코어 디바이스 AWS 리전 와 동일한 AWS 계정 및에서 생성된 다음 S3 버킷:
 - 샘플 추론 및 모델 구성 요소에 포함된 아티팩트를 저장하는 S3 버킷입니다. 이 자습서에서는 amzn-s3-demo-bucket1을 사용하여 이 버킷을 참조합니다.
 - SageMaker AI 엣지 디바이스 플릿과 연결하는 S3 버킷입니다. SageMaker AI Edge Manager에는 엣지 디바이스 플릿을 생성하고 디바이스에서 추론을 실행하지 않는 샘플 데이터를 저장하는 S3 버킷이 필요합니다. 이 자습서에서는 amzn-s3-demo-bucket2를 사용하여 이 버킷을 참조합니다.

S3 버킷 생성에 대한 자세한 내용은 [Amazon S3 시작하기](#)를 참조하세요.

- 다음과 같이 구성된 [Greengrass 디바이스 역할](#):

- 다음 IAM 정책 예제와 같이 `credentials.iot.amazonaws.com` 및 `sagemaker.amazonaws.com`에서 역할을 수임하도록 허용하는 신뢰 관계.

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

- [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM 관리형 정책.
 - [AmazonSageMakerFullAccess](#) IAM 관리형 정책.
 - 다음 IAM 정책 예제와 같이 구성 요소 아티팩트가 포함된 S3 버킷에 대한 `s3:GetObject` 작업.
- JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

```
}
```

SageMaker AI Edge Manager에서 Greengrass 코어 디바이스 설정

SageMaker AI Edge Manager의 엣지 디바이스 플릿은 논리적으로 그룹화된 디바이스 컬렉션입니다. SageMaker AI Edge Manager를와 함께 사용하려면 SageMaker AI Edge Manager 에이전트를 배포하는 Greengrass 코어 디바이스와 동일한 AWS IoT 역할 별칭을 사용하는 엣지 디바이스 플릿을 생성해야 AWS IoT Greengrass합니다. 이후 코어 디바이스를 해당 플릿의 일부로 등록해야 합니다.

주제

- [엣지 디바이스 플릿 생성](#)
- [Greengrass 코어 디바이스 등록](#)

엣지 디바이스 플릿 생성

엣지 디바이스 플릿 생성(콘솔)

1. [Amazon SageMaker AI 콘솔](#)에서 Edge Manager를 선택한 다음 엣지 디바이스 플릿을 선택합니다.
2. 디바이스 플릿 페이지에서 디바이스 플릿 생성을 선택합니다.
3. 디바이스 플릿 속성에서 다음을 수행합니다.
 - 디바이스 플릿 이름에 디바이스 플릿의 이름을 입력합니다.
 - IAM 역할에 Greengrass 코어 디바이스를 설정할 때 지정한 AWS IoT 역할 별칭의 Amazon 리소스 이름(ARN)을 입력합니다.
 - IAM 역할 별칭 생성 토글을 비활성화합니다.
4. 다음을 선택합니다.
5. 출력 구성의 S3 버킷 URI에 디바이스 플릿과 연결할 S3 버킷의 URI를 입력합니다.
6. 제출을 선택합니다.

Greengrass 코어 디바이스 등록

Greengrass 코어 디바이스를 엣지 디바이스로 등록(콘솔)

1. [Amazon SageMaker AI 콘솔](#)에서 Edge Manager를 선택한 다음 엣지 디바이스를 선택합니다.

2. 디바이스 페이지에서 디바이스 등록을 선택합니다.
3. 디바이스 속성의 디바이스 플릿 이름에 생성한 디바이스 플릿의 이름을 입력하고, 다음을 선택합니다.
4. 다음을 선택합니다.
5. 디바이스 소스의 디바이스 이름에 Greengrass 코어 디바이스의 AWS IoT 사물 이름을 입력합니다.
6. 제출을 선택합니다.

샘플 구성 요소 생성

SageMaker AI Edge Manager 구성 요소 사용을 시작하는 데 도움이 되도록는 샘플 추론 및 모델 구성 요소를 생성하고 사용자를 AWS 클라우드 위해에 업로드하는 GitHub의 Python 스크립트를 AWS 제공합니다. 개발 컴퓨터에서 다음 단계를 완료합니다.

샘플 구성 요소 생성

1. GitHub의 [AWS IoT Greengrass 구성 요소 예제](#) 리포지토리를 개발 컴퓨터에 다운로드합니다.
2. 다운로드한 /machine-learning/sagemaker-edge-manager 폴더로 이동합니다.

```
cd download-directory/machine-learning/sagemaker-edge-manager
```

3. 다음 명령을 실행하여 샘플 구성 요소를 생성하고 AWS 클라우드에 업로드합니다.

```
python3 create_components.py -r region -b amzn-s3-demo-bucket
```

*region*을 Greengrass 코어 디바이스를 생성한 AWS 리전 로 바꾸고 amzn-s3-demo-bucket1을 구성 요소 아티팩트를 저장할 S3 버킷의 이름으로 바꿉니다.

Note

기본적으로 스크립트는 이미지 분류 및 객체 감지 추론 모두에 대한 샘플 구성 요소를 생성합니다. 특정 유형의 추론에 대한 구성 요소만 생성하려면 *-i ImageClassification | ObjectDetection* 인수를 지정합니다.

이제 SageMaker AI Edge Manager와 함께 사용할 샘플 추론 및 모델 구성 요소가 AWS 계정에 생성됩니다. [AWS IoT Greengrass 콘솔](#)에서 샘플 구성 요소를 보려면 구성 요소를 선택한 다음 내 구성 요소에서 다음 구성 요소를 검색합니다.

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

샘플 이미지 분류 추론 실행

AWS제공 샘플 구성 요소와 SageMaker AI Edge Manager 에이전트를 사용하여 이미지 분류 추론을 실행하려면 이러한 구성 요소를 코어 디바이스에 배포해야 합니다. 이러한 구성 요소를 배포하면 SageMaker AI Neo로 컴파일된 사전 훈련된 Resnet-50 모델을 다운로드하고 디바이스에 SageMaker AI Edge Manager 에이전트를 설치합니다. SageMaker AI Edge Manager 에이전트는 모델을 로드하고 `gg/sageMakerEdgeManager/image-classification` 주제에 대한 추론 결과를 게시합니다. 이러한 추론 결과를 보려면 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트를 사용하여이 주제를 구독합니다.

주제

- [알림 주제 구독](#)
- [샘플 구성 요소 배포](#)
- [추론 결과 보기](#)

알림 주제 구독

이 단계에서는 샘플 추론 구성 요소에서 게시한 AWS IoT MQTT 메시지를 볼 수 있도록 AWS IoT 콘솔에서 MQTT 클라이언트를 구성합니다. 기본적으로 구성 요소는 `gg/sageMakerEdgeManager/image-classification` 주제에 대한 추론 결과를 게시합니다. 구성 요소가 처음 실행될 때 추론 결과를 확인하려면 Greengrass 코어 디바이스에 구성 요소를 배포하기 전에 이 주제를 구독합니다.

기본 알림 주제를 구독하려면

1. [AWS IoT 콘솔](#) 탐색 메뉴에서 테스트, MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독의 주제 이름 상자에 `gg/sageMakerEdgeManager/image-classification`을 입력합니다.

3. 구독을 선택합니다.

샘플 구성 요소 배포

이 단계에서는 다음 구성 요소를 구성하고 코어 디바이스에 배포합니다.

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

구성 요소를 배포하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 배포를 선택한 다음 수정하려는 대상 디바이스의 배포를 선택합니다.
2. 배포 페이지에서 수정을 선택한 다음 배포 수정을 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. 퍼블릭 구성 요소에서 선택한 구성 요소만 표시 토글을 끈 다음 `aws.greengrass.SageMakerEdgeManager` 구성 요소를 선택합니다.
 - c. 다음을 선택합니다.
5. 구성 요소 구성 페이지에서 `aws.greengrass.SageMakerEdgeManager` 구성 요소를 선택하고 다음을 수행합니다.
 - a. 구성 요소 구성을 선택합니다.
 - b. 구성 업데이트 아래에 있는 병합할 구성에 다음 구성을 입력합니다.

```
{
  "DeviceFleetName": "device-fleet-name",
  "BucketName": "amzn-s3-demo-bucket"
}
```


*device-fleet-name*을 생성한 엣지 디바이스 플릿 이름으로 바꾸고, *amzn-s3-demo-bucket*을 디바이스 플릿과 연결된 Amazon S3 버킷 이름으로 바꿉니다.

- c. 확인을 선택하고 다음을 선택합니다.
6. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
7. 검토 페이지에서 배포를 선택합니다.

구성 요소를 배포하려면(AWS CLI)

1. 개발 컴퓨터에서 `deployment.json` 파일을 생성하여 SageMaker AI Edge Manager 구성 요소의 배포 구성을 정의합니다. 이 파일은 다음 예제와 비슷합니다.

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\": \"device-fleet-name\", \"BucketName\": \"amzn-s3-demo-bucket2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    }
  }
}
```

- `targetArn` 필드에서 *targetArn*을 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`

- merge 필드에서 *device-fleet-name*을 생성한 엣지 디바이스 플릿의 이름으로 바꿉니다. 이후 *amzn-s3-demo-bucket2*를 디바이스 플릿과 연결된 S3 버킷의 이름으로 바꿉니다.
- 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 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트에서 추론 결과를 볼 수 있습니다. 구성 요소에서 추론 결과를 게시하는 주제를 구독하려면 [알림 주제 구독](#) 섹션을 참조하세요.

- AWS IoT MQTT 클라이언트 - 추론 구성 요소가 [기본 알림 주제에](#) 게시하는 결과를 보려면 다음 단계를 완료합니다.

1. [AWS IoT 콘솔](#) 탐색 메뉴에서 테스트, MQTT 테스트 클라이언트를 선택합니다.
2. 구독에서 **gg/sageMakerEdgeManager/image-classification**을 선택합니다.

- 구성 요소 로그 - 구성 요소 로그에서 추론 결과를 보려면 Greengrass 코어 디바이스에서 다음 명령을 실행합니다.

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

구성 요소 로그 또는 MQTT 클라이언트에서 추론 결과를 볼 수 없는 경우 배포가 실패했거나 코어 디바이스에 연결하지 못한 것입니다. 이는 코어 디바이스가 인터넷에 연결되어 있지 않거나 구성 요소를 실행하는 데 적절한 권한이 없는 경우 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 배포 서비스의 로그가 포함됩니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

자세한 내용은 [기계 학습 추론 문제 해결](#) 단원을 참조하십시오.

자습서: TensorFlow Lite를 사용하여 샘플 이미지 분류 추론 수행

이 자습서에서는 [TensorFlow Lite 이미지 분류](#) 추론 구성 요소를 사용하여 Greengrass 코어 디바이스에서 샘플 이미지 분류 추론을 수행하는 방법을 보여줍니다. 이 구성 요소에는 다음 구성 요소 종속성이 포함됩니다.

- TensorFlow Lite 이미지 분류 모델 저장소 구성 요소
- TensorFlow Lite 런타임 구성 요소

이 구성 요소를 배포하면 사전 훈련된 MobileNet v1 모델이 다운로드되고 [TensorFlow Lite](#) 런타임과 그 종속성이 설치됩니다. 이 구성 요소는 ml/tflite/image-classification 주제에 대한 추론 결과를 게시합니다. 이러한 추론 결과를 보려면 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트를 사용하여 이 주제를 구독합니다.

이 자습서에서는 샘플 추론 구성 요소를 배포하여 AWS IoT Greengrass에서 제공하는 샘플 이미지에 대한 이미지 분류를 수행합니다. 이 자습서를 완료하면 Greengrass 코어 디바이스에서 로컬로 실행되는 카메라의 이미지에 대한 이미지 분류를 수행하도록 샘플 추론 구성 요소를 수정하는 방법을 보여주는 [자습서: TensorFlow Lite를 사용하여 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행](#)을 완료할 수 있습니다.

Greengrass 디바이스의 기계 학습에 대한 자세한 내용은 [기계 학습 추론 수행](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: 기본 알림 주제 구독](#)
- [2단계: TensorFlow Lite 이미지 분류 구성 요소 배포](#)
- [3단계: 추론 결과 보기](#)
- [다음 단계](#)

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- Linux Greengrass 코어 디바이스. 없으면 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요. 코어 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리\(glibc\)](#) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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 구성 요소에서 게시한 AWS IoT MQTT 메시지를 볼 수 있도록 AWS IoT 콘솔에서 MQTT 클라이언트를 구성합니다. 기본적으로 구성 요소는 `m1/tflite/image-classification` 주제에 대한 추론 결과를 게시합니다. 구성 요소가 처음 실행될 때 추론 결과를 확인하려면 Greengrass 코어 디바이스에 구성 요소를 배포하기 전에 이 주제를 구독합니다.

기본 알림 주제를 구독하려면

1. [AWS IoT 콘솔](#) 탐색 메뉴에서 테스트, MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독의 주제 이름 상자에 **ml/tflite/image-classification**을 입력합니다.
3. 구독을 선택합니다.

2단계: TensorFlow Lite 이미지 분류 구성 요소 배포

이 단계에서는 코어 디바이스에 TensorFlow Lite 이미지 분류 구성 요소를 배포합니다.

TensorFlow Lite 이미지 분류 구성 요소를 배포하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 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 퍼블릭 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 배포를 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 같습니다.

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```

- targetArn 필드에서 *targetArn*을 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 자습서는 구성 요소 버전 2.1.0을 사용합니다.
aws.greengrass.TensorFlowLiteObjectDetection 구성 요소 객체에서 *2.1.0*을 바꾸면 다른 버전의 TensorFlow Lite 객체 감지 구성 요소를 사용할 수 있습니다.

2. 다음 명령을 실행하여 TensorFlow Lite 이미지 분류 구성 요소를 디바이스에 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 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트에서 추론 결과를 볼 수 있습니다. 구성 요소에서 추론 결과를 게시하는 주제를 구독하려면 [1단계: 기본 알림 주제 구독](#) 섹션을 참조하세요.

- AWS IoT MQTT 클라이언트 - 추론 구성 요소가 [기본 알림 주제에](#) 게시하는 결과를 보려면 다음 단계를 완료합니다.
1. [AWS IoT 콘솔](#) 탐색 메뉴에서 테스트, MQTT 테스트 클라이언트를 선택합니다.
 2. 구독에서 **ml/tflite/image-classification**을 선택합니다.

다음 예제와 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 디바이스에서 다음 명령을 실행합니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

다음 예제와 비슷한 결과가 나타나야 합니다.

```
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 로그 또는 MQTT 클라이언트에서 추론 결과를 볼 수 없는 경우 배포가 실패했거나 코어 디바이스에 연결하지 못한 것입니다. 이는 코어 디바이스가 인터넷에 연결되어 있지 않거나 구성 요소를 실행하는 데 적절한 권한이 없는 경우 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 배포 서비스의 로그가 포함됩니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

자세한 내용은 [기계 학습 추론 문제 해결](#) 단원을 참조하십시오.

다음 단계

지원되는 카메라 인터페이스가 포함된 Greengrass 코어 디바이스가 있는 경우 로컬로 실행되는 카메라의 이미지에 대한 이미지 분류를 수행하도록 샘플 추론 구성 요소를 수정하는 방법을 보여주는 [자습서: TensorFlow Lite를 사용하여 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행](#)을 완료할 수 있습니다.

샘플 [TensorFlow Lite 이미지 분류](#) 추론 구성 요소의 구성을 자세히 알아보려면 다음을 시도하세요.

- InferenceInterval 구성 파라미터를 수정하여 추론 코드가 실행되는 빈도를 변경합니다.
- 추론 구성 요소 구성에서 ImageName 및 ImageDirectory 구성 파라미터를 수정하여 추론에 사용할 사용자 지정 이미지를 지정합니다.

퍼블릭 구성 요소의 구성을 사용자 지정하거나 사용자 지정 기계 학습 구성 요소를 생성하는 방법에 대한 자세한 내용은 [기계 학습 구성 요소 사용자 지정](#) 섹션을 참조하세요.

자습서: TensorFlow Lite를 사용하여 카메라의 이미지에 대한 샘플 이미지 분류 추론 수행

이 자습서에서는 [TensorFlow Lite 이미지 분류](#) 추론 구성 요소를 사용하여 Greengrass 코어 디바이스에서 로컬로 카메라의 이미지에 대한 샘플 이미지 분류 추론을 수행하는 방법을 보여줍니다. 이 구성 요소에는 다음 구성 요소 종속성이 포함됩니다.

- TensorFlow Lite 이미지 분류 모델 저장소 구성 요소
- TensorFlow Lite 런타임 구성 요소

Note

이 자습서에서는 [Raspberry Pi](#) 또는 [NVIDIA Jetson Nano](#) 디바이스용 카메라 모듈에 액세스하지만 Armv7l, Armv8 또는 x86_64 플랫폼에서 다른 디바이스를 AWS IoT Greengrass 지원하지 않습니다. 다른 디바이스의 카메라를 설정하려면 디바이스 관련 설명서를 참조하세요.

Greengrass 디바이스의 기계 학습에 대한 자세한 내용은 [기계 학습 추론 수행](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: 디바이스에서 카메라 모듈 구성](#)
- [2단계: 기본 알림 주제에 대한 구독 확인](#)
- [3단계: TensorFlow Lite 이미지 분류 구성 요소 구성 수정 및 배포](#)
- [4단계: 추론 결과 보기](#)
- [다음 단계](#)

사전 조건

이 자습서를 완료하려면 먼저 [자습서: TensorFlow Lite를 사용하여 샘플 이미지 분류 추론 수행](#) 섹션을 수행해야 합니다.

또한 다음 항목이 필요합니다.

- 카메라 인터페이스가 있는 Linux Greengrass 코어 디바이스. 이 자습서에서는 지원되는 다음 디바이스 중 하나의 카메라 모듈에 액세스합니다.
- [Raspberry Pi OS](#)(기존의 Raspbian)를 실행 중인 [Raspberry Pi](#)
- [NVIDIA Jetson Nano](#)

Greengrass 코어 디바이스 설정에 대한 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요.

코어 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.
- Raspberry Pi 또는 NVIDIA Jetson Nano 디바이스의 경우 [Raspberry Pi 카메라 모듈 V2 - 8메가픽셀, 1080p](#). 카메라 설정 방법을 알아보려면 Raspberry Pi 설명서의 [Connecting the camera](#)를 참조하세요.

1단계: 디바이스에서 카메라 모듈 구성

이 단계에서는 디바이스의 카메라 모듈을 설치하고 활성화합니다. 디바이스에서 다음 명령을 실행합니다.

Raspberry Pi (Armv7l)

1. 카메라 모듈의 picamera 인터페이스를 설치합니다. 다음 명령을 실행하여 이 자습서에 필요한 카메라 모듈 및 기타 Python 라이브러리를 설치합니다.

```
sudo apt-get install -y python3-picamera
```

2. Picamera가 설치되었는지 확인합니다.

```
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

출력에 오류가 포함되어 있지 않으면 검사가 성공한 것입니다.

Note

디바이스에 설치된 Python 실행 파일이 python3.7인 경우 이 자습서의 명령에 python3 대신 python3.7을 사용합니다. 종속성 오류를 피하기 위해 pip 설치가 올바른 python3.7 또는 python3 버전에 매핑되는지 확인하세요.

3. 장치를 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 장치 재부팅을 허용합니다.
6. 다음 명령을 실행하여 카메라 설정을 테스트합니다.

```
raspistill -v -o test.jpg
```

그러면 Raspberry Pi의 미리 보기 창이 열리고, 현재 디렉터리에 test.jpg라는 이름의 사진이 저장되며, 카메라에 대한 정보가 Raspberry Pi 터미널에 표시됩니다.

7. 다음 명령을 실행하여 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 하는 symlink를 생성합니다.

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

이 자습서에서 *MLRootPath*의 기본값은 */greengrass/v2/work/variant.TensorFlowLite/greengrass_ml*입니다. 이 위치의 greengrass_ml_tflite_venv 폴더는 [자습서: TensorFlow Lite를 사용하여 샘플 이미지 분류 추론 수행](#)에서 추론 구성 요소를 처음으로 배포할 때 생성합니다.

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 test.jpg라는 이미지가 캡처되어 현재 디렉터리에 저장됩니다.

2. (선택 사항) 디바이스를 재부팅합니다. 이전 단계에서 gst-launch 명령을 실행할 때 문제가 발생하는 경우 디바이스를 재부팅하면 이러한 문제가 해결될 수 있습니다.

```
sudo reboot
```

Note

Jetson Nano와 같은 Armv8(AArch64) 디바이스의 경우, 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 하기 위해 symlink를 생성할 필요가 없습니다.

2단계: 기본 알림 주제에 대한 구독 확인

에서 AWS IoT TensorFlow Lite 이미지 분류 구성 `ml/tflite/image-classification` 요소가 주제에 게시한 MQTT 메시지를 볼 수 있도록 AWS IoT 콘솔에서 MQTT 클라이언트를 구성 [자습서: TensorFlow Lite를 사용하여 샘플 이미지 분류 추론 수행](#)했습니다. AWS IoT 콘솔에서이 구독이 존재하는지 확인합니다. 없으면 Greengrass 코어 디바이스에 구성 요소를 배포하기 전에 [1단계: 기본 알림 주제 구독](#)의 단계에 따라 이 주제를 구독하세요.

3단계: TensorFlow Lite 이미지 분류 구성 요소 구성 수정 및 배포

이 단계에서는 TensorFlow Lite 이미지 분류 구성 요소를 구성하고 코어 디바이스에 배포합니다.

TensorFlow Lite 이미지 분류 구성 요소를 구성 및 배포하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 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 퍼블릭 구성 요소에서 `aws.greengrass.TensorFlowLiteImageClassification` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 다음을 수행합니다.
 - 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 사용하여 구성 요소는 디바이스의 카메라 모듈에 액세스하고 카메라로 찍은 이미지에 대한 추론을 수행합니다. 추론 코드는 60초마다 실행됩니다.

- c. 확인을 선택하고 다음을 선택합니다.
8. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
9. 검토 페이지에서 배포를 선택합니다.

TensorFlow Lite 이미지 분류 구성 요소를 구성하고 배포하려면(AWS CLI)

1. `deployment.json` 파일을 생성하여 TensorFlow Lite 이미지 분류 구성 요소의 배포 구성을 정의합니다. 이 파일은 다음과 같습니다.

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
    }
  }
}
```

- `targetArn` 필드에서 `targetArn`을 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 자습서는 구성 요소 버전 2.1.0을 사용합니다.
`aws.greengrass.TensorFlowLiteImageClassification` 구성 요소 객체에서 **2.1.0**을 바꾸면 다른 버전의 TensorFlow Lite 이미지 분류 구성 요소를 사용할 수 있습니다.

이 구성 업데이트를 사용하여 구성 요소는 디바이스의 카메라 모듈에 액세스하고 카메라로 찍은 이미지에 대한 추론을 수행합니다. 추론 코드는 60초마다 실행됩니다. 다음 값을 교체합니다.

2. 다음 명령을 실행하여 TensorFlow Lite 이미지 분류 구성 요소를 디바이스에 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 AWS IoT 콘솔의 AWS IoT MQTT 클라이언트에서 추론 결과를 볼 수 있습니다. 구성 요소에서 추론 결과를 게시하는 주제를 구독하려면 [2단계: 기본 알림 주제에 대한 구독 확인](#) 섹션을 참조하세요.

- AWS IoT MQTT 클라이언트 - 추론 구성 요소가 [기본 알림 주제에](#) 게시하는 결과를 보려면 다음 단계를 완료합니다.
 1. [AWS IoT 콘솔](#) 탐색 메뉴에서 테스트, MQTT 테스트 클라이언트를 선택합니다.
 2. 구독에서 `ml/tflite/image-classification`을 선택합니다.
- 구성 요소 로그 - 구성 요소 로그에서 추론 결과를 보려면 Greengrass 코어 디바이스에서 다음 명령을 실행합니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

구성 요소 로그 또는 MQTT 클라이언트에서 추론 결과를 볼 수 없는 경우 배포가 실패했거나 코어 디바이스에 연결하지 못한 것입니다. 이는 코어 디바이스가 인터넷에 연결되어 있지 않거나 구성 요소를 실행하는 데 필요한 권한이 없는 경우 발생할 수 있습니다. 코어 디바이스에서 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일을 확인합니다. 이 파일에는 Greengrass 코어 디바이스의 배포 서비스의 로그가 포함됩니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

자세한 내용은 [기계 학습 추론 문제 해결](#) 단원을 참조하십시오.

다음 단계

이 자습서에서는 TensorFlow Lite 이미지 분류 구성 요소를 사용자 지정 구성 옵션과 함께 사용하여 카메라로 촬영한 이미지에 대한 샘플 이미지 분류를 수행하는 방법을 보여줍니다.

퍼블릭 구성 요소의 구성을 사용자 지정하거나 사용자 지정 기계 학습 구성 요소를 생성하는 방법에 대한 자세한 내용은 [기계 학습 구성 요소 사용자 지정](#) 섹션을 참조하세요.

Components

AWS IoT Greengrass 구성 요소는 Greengrass 코어 디바이스에 배포하는 소프트웨어 모듈입니다. 구성 요소는 애플리케이션에서 실행할 애플리케이션, 런타임 설치 관리자, 라이브러리 또는 모든 코드를 나타낼 수 있습니다. 다른 구성 요소 종속되는 구성 요소를 정의할 수 있습니다. 예를 들어 Python을 설치하는 구성 요소를 정의한 다음 해당 구성 요소를 Python 애플리케이션을 실행하는 구성 요소의 종속성으로 정의할 수 있습니다. 구성 요소를 디바이스 플릿에 배포하면 Greengrass는 디바이스에 필요한 소프트웨어 모듈만 배포합니다.

주제

- [AWS 제공 구성 요소](#)
- [게시자 지원 구성 요소](#)
- [커뮤니티 구성 요소](#)
- [AWS IoT Greengrass 개발 도구](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)

AWS 제공 구성 요소

AWS IoT Greengrass는 디바이스에 배포할 수 있는 사전 구축된 구성 요소를 제공하고 유지 관리합니다. 이러한 구성 요소에는 기능(예: 스트림 관리자), AWS IoT Greengrass V1 커넥터(예: CloudWatch 지표) 및 로컬 개발 도구(예: AWS IoT Greengrass CLI)가 포함됩니다. 디바이스에 [이러한 구성 요소를 배포](#)하여 독립 실행형 기능으로 실행하거나 [사용자 지정 Greengrass 구성 요소](#)의 종속성으로 사용할 수 있습니다.

Note

AWS에서 제공하는 몇 가지 구성 요소는 Greengrass nucleus의 특정 마이너 버전에 따라 다릅니다. 이 종속성 때문에 Greengrass nucleus를 새로운 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 달라지는 특정 nucleus 버전에 대한 내용은 해당 구성 요소 주제를 참조하세요. nucleus 업데이트에 대한 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#) 섹션을 참조하세요.

구성 요소에 일반 및 Lambda 구성 요소 유형이 둘 다 있으면 구성 요소의 현재 버전은 일반 유형이고 구성 요소의 이전 버전은 Lambda 유형입니다.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
Greengrass nucleus	AWS IoT Greengrass 코어 소프트웨어의 nucleus입니다. 이 구성 요소를 사용하여 코어 디바이스에서 소프트웨어를 구성하고 업데이트합니다.	Nucleus	Linux, Windows	예	아니요
Greengrass nucleus lite	리소스가 제한된 디바이스용 경량 nucleus로, 저비용 엣지 디바이스와 대용량 애플리케이션에 최적화됨	NucleusLite	Linux	예	아니요
클라이언트 디바이스 인증	클라이언트 디바이스라는 로컬 IoT 디바이스를 활성화하여 코어 디바이	플러그인	Linux, Windows	예	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
	스에 연결합 니다.				
CloudWatch 지표	Amazon CloudWatch 에 사용자 지 정 지표를 게 시합니다.	일반, Lambda	Linux, Windows	예	예
AWS IoT Device Defender	관리자에게 Greengrass 코어 디바이 스의 상태 변 경을 알려 비 정상적인 동 작을 식별합 니다.	일반, Lambda	Linux, Windows	예	아니요
디스크 스펠러	Greengrass 코어 디바이 스에서 AWS IoT Core로 스플링되는 메시지에 대 한 영구 스토 리지 옵션을 활성화합니 다. 이 구성 요소에서는 이러한 아웃 바운드 메시 지를 디스크 에 저장합니 다.	플러그인	Linux, Windows	예	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
Docker 애플리케이션 관리자	AWS IoT Greengrass가 Docker Hub 또는 Amazon Elastic Container Registry(Amazon ECR)에서 Docker 이미지를 다운로드할 수 있게 합니다.	일반	Linux, Windows	아니요	아니요
Kinesis Video Streams용 엣지 커넥터	로컬 카메라에서 비디오 피드를 읽고, 스트림을 Kinesis Video Streams에 게시하고, AWS IoT TwinMaker를 사용하여 Grafana 대시보드에 스트림을 표시합니다.	일반	Linux	아니요	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
Greengrass CLI	로컬 배포 를 생성하고 Greengrass s 코어 디바 이스 및 해당 구성 요소와 연동하는 데 사용할 수 있 는 명령줄 인 터페이스를 제공합니다.	플러그인	Linux, Windows	예	아니요
IP 감지기	클라이언트 디바이스가 연결 방법을 검색할 수 있 도록 MQTT 연결 정보 를 AWS IoT Greengrass 에 보고합니 다.	플러그인	Linux, Windows	예	아니요
Firehose	Amazon Data Firehose 전 송 스트림을 통해 데이터 를 AWS 클 라우드의 대 상에 게시합 니다.	Lambda	Linux	아니요	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
Lambda 런처	Lambda 함수에 대한 프로세스 및 환경 구성을 처리합니다.	일반	Linux	아니요	아니요
Lambda 관리자	Lambda 함수에 대한 프로세스 간 통신 및 조정을 처리합니다.	플러그인	Linux	아니요	아니요
Lambda 런타임	각 Lambda 런타임에 대한 아티팩트를 제공합니다.	일반	Linux	아니요	아니요
레거시 구독 라우터	AWS IoT Greengrass V1에서 실행되는 Lambda 함수에 대한 구독을 관리합니다.	일반	Linux	아니요	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
로컬 디버그 콘솔	Greengrass 코어 디바이스 및 해당 구성 요소를 디버깅하고 관리하는 데 사용할 수 있는 로컬 콘솔을 제공합니다.	플러그인	Linux, Windows	예	아니요
로그 관리자	Greengrass 코어 디바이스에서 로그를 수집하고 업로드합니다.	플러그인	Linux, Windows	예	아니요
기계 학습 구성 요소	Greengrass 코어 디바이스에서 기계 학습 추론을 수행하는 데 사용할 수 있는 기계 학습 모델 및 샘플 추론 코드를 제공합니다.	기계 학습 구성 요소 을(를) 참조하세요.			아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
Modbus-RTU 프로토콜 어댑터	로컬 Modbus RTU 디바이스에서 정보를 폴링합니다.	Lambda	Linux	아니요	아니요
nucleus 원격 측정 이미터	nucleus에서 수집한 시스템 상태 원격 측정 데이터를 로컬 주제 또는 AWS IoT Core MQTT 주제에 게시합니다.	플러그인	Linux, Windows	예	아니요
MQTT 브리지	클라이언트 디바이스, 로컬 AWS IoT Greengrass 게시/구독 및 AWS IoT Core 간 MQTT 메시지를 릴레이합니다.	플러그인	Linux, Windows	예	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
MQTT 3.1.1 브로커(Moquette)	클라이언트 디바이스와 코어 디바이스 간 메시지를 처리하는 MQTT 3.1.1 브로커를 실행합니다.	플러그인	Linux, Windows	예	아니요
MQTT 5 브로커(EMQX)	클라이언트 디바이스와 코어 디바이스 간 메시지를 처리하는 MQTT 5 브로커를 실행합니다.	일반	Linux, Windows	아니요	아니요
PKCS#11 공급자	Greengrass 구성 요소가 HSM(하드웨어 보안 모듈)에 안전하게 저장한 프라이빗 키 및 인증서에 액세스할 수 있도록 합니다.	플러그인	Linux	예	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
보안 암호 관리자	Greengrass 코어 디바이스의 사용자 지정 구성 요소에서 보안 암호와 같은 자격 증명을 안전하게 사용할 수 있도록 AWS Secrets Manager 보안 암호의 보안 암호를 배포합니다.	플러그인	Linux, Windows	예	아니요
보안 터널링	제한된 방화벽 뒤에 있는 Greengrass 코어 디바이스와의 양방향 통신을 설정하는 데 사용할 수 있는 AWS IoT 보안 터널링 연결을 활성화합니다.	일반	Linux	아니요	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
새도 관리자	코어 디바이스에서 새도와 상호 작용을 활성화합니다. 새도 문서 스토리지는 물론 AWS IoT 디바이스 새도 서비스와 로컬 새도 상태의 동기화도 관리합니다.	플러그인	Linux, Windows	예	아니요
Amazon SNS	Amazon SNS 주제에 메시지를 게시합니다.	Lambda	Linux	아니요	아니요
스트림 관리자	로컬 소스에서 AWS 클라우드로 대용량 데이터를 스트리밍합니다.	일반	Linux, Windows	아니요	예
시스템 로그 전달자	systemd-journald 로그를 AWS 클라우드에 업로드합니다.	일반	Linux	예	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
Systems Manager Agent	디바이스 패치, 명 령 실행 등 을 할 수 있 는 AWS Systems Manager를 사용하여 코 어 디바이스 를 관리합니 다.	일반	Linux	예	아니요
토큰 교환 서비스	AWS 서비스 와 연동하는 데 사용할 수 있는 AWS 자격 증명을 제공합니다.	일반	Linux, Windows	아니요	아니요
IoT SiteWise OPC UA 수집기	OPC-UA 서 버에서 데이 터를 수집합 니다.	일반	Linux, Windows	아니요	아니요
IoT SiteWise OPC UA 데이터 소스 시뮬레이터	샘플 데이터 를 생성하는 로컬 OPC- UA 서버를 실행합니다.	일반	Linux, Windows	아니요	아니요
IoT SiteWise 게시자	AWS 클라우 드에 데이터 를 게시합니 다.	일반	Linux, Windows	아니요	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스	Nucleus lite 호환
IoT SiteWise 프로세서	Greengrass 코어 디바이 스에서 데이 터를 처리합 니다.	일반	Linux, Windows	아니요	아니요

Greengrass nucleus

Greengrass nucleus 구성 요소(`aws.greengrass.Nucleus`)는 필수 구성 요소이며 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 실행하기 위한 최소 요구 사항입니다. AWS IoT Greengrass 코어 소프트웨어를 원격으로 사용자 지정하고 업데이트하도록 구성 요소를 구성할 수 있습니다. 이 구성 요소를 배포하여 코어 디바이스에서 프록시, 디바이스 역할 및 AWS IoT 사물 구성과 같은 설정을 구성합니다.

Note

Greengrass 버전 2.14.0부터, 리소스가 제한된 엣지 디바이스를 위해 메모리 사용량이 최적화된 nucleus 디바이스 런타임 버전이 제공됩니다. 구성 및 사용에 대한 자세한 내용은 [Greengrass nucleus lite](#)를 참조하세요.

Important

nucleus 구성 요소의 버전이 변경되거나 특정 구성 파라미터를 변경하면 디바이스의 nucleus 및 기타 모든 구성 요소를 포함하는 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항이 적용됩니다.

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass

코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

주제

- [버전](#)
- [디바이스 요구 사항](#)
- [지원하는 플랫폼](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [다운로드 및 설치](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.16.x
- 2.15.x
- 2.14.x
- 2.13.x
- 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

디바이스 요구 사항

Note

AWS IoT Device Tester 용 AWS IoT Greengrass 를 사용하여 디바이스가 AWS IoT Greengrass 코어 소프트웨어를 실행하고와 통신할 수 있는지 확인할 수 있습니다 AWS 클라우드. 자세한 내용은 [AWS IoT Greengrass V2용 AWS IoT Device Tester 사용](#) 단원을 참조하십시오.

Linux

- 가 [AWS 리전](#) 지원하는의 사용 AWS IoT Greengrass V2. 지원되는 리전 목록은 AWS 일반 참조에서 [AWS IoT Greengrass V2 엔드포인트 및 할당량](#)을 참조하세요.
- AWS IoT Greengrass 코어 소프트웨어에 사용할 수 있는 최소 디스크 공간은 256MB입니다. 이 요구 사항에는 코어 디바이스에 배포된 구성 요소는 포함되지 않습니다.
- AWS IoT Greengrass 코어 소프트웨어에 할당된 최소 96MB RAM. 이 요구 사항에는 코어 디바이스에서 실행되는 구성 요소는 포함되지 않습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.
- Java 런타임 환경(JRE) 버전 8 이상. 디바이스의 [PATH](#) 환경 변수에서 Java를 사용할 수 있어야 합니다. Java를 사용하여 사용자 정의 구성 요소를 개발하려면 JDK(Java Development Kit)를 설치해야 합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.
- [GNU C Library](#)(glibc) 버전 2.25 이상.
- 루트 사용자로 AWS IoT Greengrass 코어 소프트웨어를 실행해야 합니다. 예를 들어 sudo를 사용합니다.
- 와 같은 AWS IoT Greengrass 코어 소프트웨어를 실행하는 루트 사용자는 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 root있어야 합니다. /etc/sudoers 파일은 이 사용자

에게 다른 그룹으로 `sudo`를 실행할 수 있는 권한을 부여해야 합니다. `/etc/sudoers`의 사용자에게 대한 권한은 다음 예제와 유사해야 합니다.

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

- 코어 디바이스는 엔드포인트 및 포트 세트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.
- `/tmp` 디렉터리는 `exec` 권한으로 탑재되어야 합니다.
- 다음 셸 명령 모두:
 - `ps -ax -o pid,ppid`
 - `sudo`
 - `sh`
 - `kill`
 - `cp`
 - `chmod`
 - `rm`
 - `ln`
 - `echo`
 - `exit`
 - `id`
 - `uname`
 - `grep`
- 디바이스에 다음 셸 명령(선택 사항)이 필요할 수도 있습니다.
 - (선택 사항) `systemctl`. 이 명령은 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정하는 데 사용됩니다.
 - (선택 사항) `useradd`, `groupadd`, `usermod`. 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 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.

Windows

- 가 [AWS 리전](#) 지원하는의 사용 AWS IoT Greengrass V2. 지원되는 리전 목록은 AWS 일반 참조에서 [AWS IoT Greengrass V2 엔드포인트 및 할당량](#)을 참조하세요.
- AWS IoT Greengrass 코어 소프트웨어에 사용할 수 있는 최소 디스크 공간은 256MB입니다. 이 요구 사항에는 코어 디바이스에 배포된 구성 요소는 포함되지 않습니다.
- AWS IoT Greengrass 코어 소프트웨어에 할당된 최소 160MB RAM. 이 요구 사항에는 코어 디바이스에서 실행되는 구성 요소는 포함되지 않습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.
- Java 런타임 환경(JRE) 버전 8 이상. 디바이스의 [PATH](#) 시스템 변수에서 Java를 사용할 수 있어야 합니다. Java를 사용하여 사용자 정의 구성 요소를 개발하려면 JDK(Java Development Kit)를 설치해야 합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.

Note

[Greengrass nucleus](#)의 버전 2.5.0을 사용하려면 Java 런타임 환경(JRE)의 64비트 버전을 사용해야 합니다. Greengrass nucleus 버전 2.5.1은 32비트 및 64비트 JRE를 지원합니다.

- AWS IoT Greengrass 코어 소프트웨어를 설치하는 사용자는 관리자여야 합니다.
- AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치해야 합니다. 소프트웨어를 설치할 때 `--setup-system-service true`를 지정합니다.
- 구성 요소 프로세스를 실행하는 각 사용자는 LocalSystem 계정에 있어야 하며 사용자의 이름과 암호는 LocalSystem 계정의 Credential Manager 인스턴스에 있어야 합니다. 지침에 따라 [AWS IoT Greengrass 코어 소프트웨어를 설치할 때 사용자 설정할 수 있습니다](#).
- 코어 디바이스는 엔드포인트 및 포트 세트에 대한 아웃바운드 요청을 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

지원하는 플랫폼

AWS IoT Greengrass 는 다음 플랫폼을 실행하는 디바이스를 공식적으로 지원합니다. 이 목록에 포함되지 않은 플랫폼이 있는 디바이스는 작동할 수 있지만 이러한 지정된 플랫폼에서만 AWS IoT Greengrass 테스트합니다.

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 디바이스에서 지원되지 않습니다. 자세한 내용은 [Greengrass 기능 호환성](#) 및 [기능 고려 사항](#) 섹션을 참조하세요.

기능 고려 사항

일부 AWS IoT Greengrass 기능은 현재 Windows 디바이스에서 지원되지 않습니다. 기능 차이를 검토하여 Windows 디바이스가 요구 사항을 충족하는지 확인합니다. 자세한 내용은 [Greengrass 기능 호환성](#) 단원을 참조하십시오.

사용자 지정 Linux 기반 운영 체제를 빌드하려면 [meta-aws 프로젝트](#)에서 AWS IoT Greengrass에 대한 BitBake 레시피를 사용하면 됩니다. meta-aws 프로젝트는 [OpenEmbedded](#) 및 Yocto 프로젝트 빌드 프레임워크로 빌드된 [임베디드 Linux](#) 시스템에서 AWS 엣지 소프트웨어 기능을 빌드하는 데 사용할 수 있는 레시피를 제공합니다. [Yocto Project](#)는 하드웨어 아키텍처에 관계없이 임베디드 애플리케이션

을 위한 사용자 지정 Linux 기반 시스템을 빌드할 수 있도록 지원하는 오픈 소스 협업 프로젝트입니다. [용 AWS IoT Greengrass BitBake 레시피](#)는 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 설치, 구성 및 자동으로 실행합니다.

Linux 플랫폼은 Docker 컨테이너 AWS IoT Greengrass 에서도 실행할 수 있습니다. 자세한 내용은 [Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

자세한 내용은 [지원하는 플랫폼](#) 단원을 참조하십시오.

요구 사항

Greengrass nucleus 및 AWS IoT Greengrass 코어 소프트웨어를 설치하고 실행하려면 디바이스가 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [디바이스 요구 사항](#) 단원을 참조하십시오.

VPC에서 Greengrass nucleus 구성 요소의 실행이 지원됩니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.

- Greengrass nucleus 구성 요소는 AWS IoT data, AWS IoT Credentials 및 Amazon S3에 연결되어 있어야 합니다.

종속성

Greengrass nucleus에는 구성 요소 종속성이 포함되지 않습니다. 하지만 AWS에서 제공하는 여러 구성 요소에는 종속성으로 nucleus가 포함됩니다. 자세한 내용은 [AWS 제공 구성 요소](#) 단원을 참조하십시오.

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

다운로드 및 설치

디바이스에 Greengrass nucleus 구성 요소를 설정하는 설치 관리자를 다운로드할 수 있습니다. 이 설치 관리자는 디바이스를 Greengrass 코어 디바이스로 설정합니다. 수행할 수 있는 설치에는 두 가지

유형이 있습니다. 즉, 필요한 AWS 리소스를 생성하는 빠른 설치 또는 AWS 리소스를 직접 생성하는 수동 설치가 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

자습서를 따라 Greengrass nucleus를 설치하고 Greengrass 구성 요소 개발을 살펴볼 수도 있습니다. 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#) 단원을 참조하십시오.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다. 일부 파라미터를 사용하려면 AWS IoT Greengrass 코어 소프트웨어를 다시 시작해야 합니다. 그 이유와 이 구성 요소를 구성하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 구성](#) 섹션을 참조하세요.

iotRoleAlias

토큰 교환 IAM AWS IoT 역할을 가리키는 역할 별칭입니다. 자격 AWS IoT 증명 공급자는 Greengrass 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 이 역할을 말합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

--provision true 옵션으로 AWS IoT Greengrass 코어 소프트웨어를 실행하면 소프트웨어가 역할 별칭을 프로비저닝하고 nucleus 구성 요소에서 해당 값을 설정합니다.

interpolateComponentConfiguration

(선택 사항) Greengrass nucleus가 구성 요소 구성에서 [구성 요소 레시피 변수](#)를 보간하고 [구성 업데이트를 병합](#)하도록 활성화할 수 있습니다. 코어 디바이스가 구성의 레시피 변수를 사용하는 Greengrass 구성 요소를 실행할 수 있도록 이 옵션을 true로 설정하는 것이 좋습니다.

이 기능은 이 구성 요소의 v2.6.0 이상에서 사용할 수 있습니다.

기본값: false

networkProxy

(선택 사항) 모든 연결에 사용할 네트워크 프록시입니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 단원을 참조하십시오.

Important

이 구성 파라미터에 변경 사항을 배포하면 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항이 적용됩니다.

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`

`scheme://userinfo@host:port` 형식으로 표시되는 프록시 서버의 URL입니다.

- `scheme` - 체계로, `http` 또는 `https`여야 합니다.

Important

Greengrass 코어 디바이스에서 HTTPS 프록시가 사용되려면 [Greengrass nucleus](#) v2.5.0 이상을 실행해야 합니다.

HTTPS 프록시를 구성하는 경우 코어 디바이스의 Amazon 루트 CA 인증서에 프록시 서버 CA 인증서를 추가해야 합니다. 자세한 내용은 [HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정](#) 단원을 참조하십시오.

- `userinfo` - (선택 사항) 사용자 이름 및 암호 정보입니다. `url`에서 이 정보를 지정하면 Greengrass 코어 디바이스에서는 `username` 및 `password` 필드를 무시합니다.
- `host` - 프록시 서버의 호스트 이름 또는 IP 주소입니다.
- `port` - (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 디바이스에서는 다음 기본값이 사용됩니다.
 - `http` - 80
 - `https` - 443

`username`

(선택 사항) 프록시 서버를 인증하는 사용자 이름입니다.

`password`

(선택 사항) 프록시 서버를 인증하는 암호입니다.

`mqtt`

(선택 사항) Greengrass 코어 디바이스의 MQTT 구성입니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 단원을 참조하십시오.

⚠ Important

이 구성 파라미터에 변경 사항을 배포하면 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항이 적용됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

port

(선택 사항) MQTT 연결에 사용할 포트입니다.

기본값: 8883

keepAliveTimeoutMs

(선택 사항) MQTT 연결을 유지하기 위해 클라이언트가 보내는 각 PING 메시지 사이의 시간(밀리초)입니다. 이 값은 pingTimeoutMs보다 커야 합니다.

기본값: 60000(60초)

pingTimeoutMs

(선택 사항) 클라이언트가 서버로부터 PINGACK 메시지를 수신하기 위해 대기하는 시간(밀리초)입니다. 대기 시간이 제한 시간을 초과하면 코어 디바이스가 닫히고 MQTT 연결을 다시 엽니다. 이 값은 keepAliveTimeoutMs보다 작아야 합니다.

기본값: 30000(30초)

operationTimeoutMs

(선택 사항) 클라이언트가 MQTT 작업(예: CONNECT 또는 PUBLISH)이 완료될 때까지 기다리는 시간(밀리초)입니다. 이 옵션은 MQTT PING 또는 연결 유지 메시지에는 적용되지 않습니다.

기본값: 30000(30초)

maxInFlightPublishes

(선택 사항) 동시에 이동할 수 있는 확인되지 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 크기입니다. 메시지가 이 크기를 초과하면 Greengrass nucleus가 오류와 함께 메시지를 거부합니다.

이 기능은 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

기본값: 131072(128KB)

유효 범위: 최대값 2621440(2.5MB)

`maxPublishRetry`

(선택 사항) 게시에 실패한 메시지의 최대 재시도 횟수입니다. -1을 지정하면 무제한으로 재시도할 수 있습니다.

이 기능은 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

기본값: 100

`spooler`

(선택 사항) Greengrass 코어 디바이스의 MQTT 스폴러 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`storageType`

메시지 저장을 위한 스토리지 유형입니다. `storageType`이 Disk로 설정된 경우 `pluginName`을 구성할 수 있습니다. Memory 또는 Disk를 지정할 수 있습니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.11.0 이상에서 사용할 수 있습니다.

 Important

MQTT 스폴러 `storageType`이 Disk로 설정되어 있고 Greengrass nucleus를 버전 2.11.x에서 이보다 낮은 버전으로 다운그레이드하려는 경우 구성을 다시 Memory로 변경해야 합니다. Greengrass nucleus 버전 2.10.x 이하에서 지원되는 `storageType`의 유일한 구성은 Memory입니다. 이 지침을 따르지 않으면 스폴러가 손상될 수 있습니다. 이렇게 하면 Greengrass 코어 디바이스가 AWS 클라우드에 MQTT 메시지를 보낼 수 없게 됩니다.

기본값: Memory

pluginName

(선택 사항) 플러그인 구성 요소 이름입니다. 이 구성 요소는 storageType이 Disk로 설정된 경우에만 사용됩니다. 이 옵션은 기본적으로 aws.greengrass.DiskSpooler로 설정되고 Greengrass에서 제공하는 [디스크 스푼러](#)를 사용합니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.11.0 이상에서 사용할 수 있습니다.

기본값: "aws.greengrass.DiskSpooler"

maxSizeInBytes

(선택 사항) 코어 디바이스가 처리되지 않은 MQTT 메시지를 메모리에 저장하는 캐시의 최대 크기입니다. 캐시가 가득 차면 새 메시지가 거부됩니다.

기본값: 2621440(2.5MB)

keepQos0WhenOffline

(선택 사항) 코어 디바이스가 오프라인 상태에서 수신하는 MQTT QoS 0 메시지를 스푼링할 수 있습니다. 이 옵션을 true로 설정하면 코어 디바이스는 오프라인 상태에서 보낼 수 없는 QoS 0 메시지를 스푼링합니다. 이 옵션을 false로 설정하면 코어 디바이스가 이러한 메시지를 폐기합니다. 코어 디바이스는 스푼이 가득 차지 않는 한 항상 QoS 1 메시지를 스푼링합니다.

기본값: false

version

(선택 사항) MQTT의 버전입니다. mqtt3 또는 mqtt5를 지정할 수 있습니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.10.0 이상에서 사용할 수 있습니다.

기본값: mqtt5

receiveMaximum

(선택 사항) 브로커가 보낼 수 있는 확인되지 않은 최대 QoS1 패킷 수입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.10.0 이상에서 사용할 수 있습니다.

기본값: 100

sessionExpirySeconds

(선택 사항) 세션이 IoT 코어에서 지속되도록 요청할 수 있는 시간(초)입니다. 기본값은에서 지원하는 최대 시간입니다 AWS IoT Core.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.10.0 이상에서 사용할 수 있습니다.

기본값: 604800 (7 days)

minimumReconnectDelaySeconds

(선택 사항) 재연결 동작 옵션입니다. MQTT가 다시 연결하는 데 걸리는 최소 시간(초)입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.10.0 이상에서 사용할 수 있습니다.

기본값: 1

maximumReconnectDelaySeconds

(선택 사항) 재연결 동작 옵션입니다. MQTT가 다시 연결하는 데 걸리는 최대 시간(초)입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.10.0 이상에서 사용할 수 있습니다.

기본값: 120

minimumConnectedTimeBeforeRetryResetSeconds

(선택 사항) 재연결 동작 옵션입니다. 재시도 지연이 최소로 재설정되기 전에 연결이 활성화되어야 하는 시간(초)입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.10.0 이상에서 사용할 수 있습니다.

기본값: 30

jvmOptions

(선택 사항) AWS IoT Greengrass 코어 소프트웨어를 실행하는 데 사용할 JVM 옵션입니다. AWS IoT Greengrass 코어 소프트웨어 실행을 위한 권장 JVM 옵션에 대한 자세한 내용은 섹션을 참조하세요 [JVM 옵션으로 메모리 할당 제어](#).

Important

이 구성 파라미터에 변경 사항을 배포하면 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항이 적용됩니다.

iotDataEndpoint

의 AWS IoT 데이터 엔드포인트입니다 AWS 계정.

--provision true 옵션으로 AWS IoT Greengrass 코어 소프트웨어를 실행하면 소프트웨어는 에서 데이터 및 자격 증명 엔드포인트를 가져 AWS IoT 와 nucleus 구성 요소에 설정합니다.

iotCredEndpoint

의 AWS IoT 자격 증명 엔드포인트입니다 AWS 계정.

--provision true 옵션으로 AWS IoT Greengrass 코어 소프트웨어를 실행하면 소프트웨어는 에서 데이터 및 자격 증명 엔드포인트를 가져 AWS IoT 와 nucleus 구성 요소에 설정합니다.

greengrassDataPlaneEndpoint

이 기능은 이 구성 요소의 v2.7.0 이상에서 사용할 수 있습니다.

자세한 내용은 [프라이빗 CA에서 서명한 디바이스 인증서 사용](#) 단원을 참조하십시오.

greengrassDataPlanePort

이 기능은 이 구성 요소의 v2.0.4 이상에서 사용할 수 있습니다.

(선택 사항) 데이터 플레인 연결에 사용할 포트입니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 단원을 참조하십시오.

Important

디바이스가 아웃바운드 연결을 수행할 수 있는 포트를 지정해야 합니다. 차단된 포트를 지정하면 디바이스가 연결하여 배포를 수신 AWS IoT Greengrass 할 수 없습니다.

다음 옵션 중 하나를 선택합니다.

- 443
- 8443

기본값: 8443

awsRegion

사용할 AWS 리전입니다.

runWithDefault

구성 요소 실행에 사용할 시스템 사용자.

Important

이 구성 파라미터에 변경 사항을 배포하면 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항이 적용됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

posixUser

코어 디바이스가 일반 및 Lambda 구성 요소를 실행하는 데 사용하는 시스템 사용자 및 선택적으로 시스템 그룹의 이름 또는 ID입니다. `user:group` 형식으로 사용자와 그룹을 콜론(:)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 사용자의 기본 그룹을 사용합니다. 예를 들어 `ggc_user` 또는 `ggc_user:ggc_group`을 지정할 수 있습니다. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 단원을 참조하십시오.

--component-default-user *ggc_user:ggc_group* 옵션을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 실행하면 소프트웨어가 nucleus 구성 요소에서 이 파라미터를 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. 사용자가 각 Windows 코어 디바이스에 있어야 하며, 해당 이름과 암호가 LocalSystem 계정의 Credentials Manager 인스턴스에 저장되어야 합니다. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 단원을 참조하십시오.

--component-default-user *ggc_user* 옵션을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 실행하면 소프트웨어가 nucleus 구성 요소에서 이 파라미터를 설정합니다.

systemResourceLimits

이 기능은 이 구성 요소의 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

기본적으로 일반 및 컨테이너화되지 않은 Lambda 구성 요소 프로세스에 적용할 시스템 리소스 제한. 배포를 생성할 때 개별 구성 요소에 대한 시스템 리소스 제한을 재정의할 수 있습니다. 자세한 내용은 [구성 요소에 대한 시스템 리소스 제한 구성](#) 단원을 참조하십시오.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

cpus

각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어, 4개의 CPU 코어가 있는 코어 디바이스에서 이 값을 2로 설정하여 각 구성 요소의 프로세스를 각 CPU 코어 사용량의 50%로 제한할 수 있습니다. 1개의 CPU 코어가 있는 디바이스에서 이 값을 0.25로 설정하여 각 구성 요소의 프로세스를 CPU 사용량의 25%로 제한할 수 있습니다. 이 값을 CPU 코어 수보다 큰 수로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소의 CPU 사용량을 제한하지 않습니다.

memory

각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 RAM(KB 단위)입니다.

s3EndpointType

(선택 사항) S3 엔드포인트 유형입니다. 이 파라미터는 미국 동부(버지니아 북부)(us-east-1) 리전에만 적용됩니다. 다른 리전에서 이 파라미터를 설정하면 무시됩니다. 다음 옵션 중 하나를 선택합니다.

- REGIONAL – S3 클라이언트 및 미리 서명된 URL이 리전 엔드포인트를 사용합니다.
- GLOBAL – S3 클라이언트 및 미리 서명된 URL이 레거시 엔드포인트를 사용합니다.
- DUALSTACK – S3 미리 서명된 URL은 듀얼 스택 엔드포인트를 사용합니다.

기본값: GLOBAL

fipsMode

(선택 사항) Greengrass가 FIPS 엔드포인트를 사용하게 됩니다. FIPS 엔드포인트를 활성화하는 방법에 대한 자세한 내용은 [FIPS 엔드포인트](#)를 참조하세요.

다음 옵션 중 하나를 선택합니다.

- true true로 설정하면 엔드포인트는 FIPS 엔드포인트를 사용합니다.
- false false일 경우 엔드포인트는 FIPS 엔드포인트를 사용하지 않습니다.

기본값: false

logging

(선택 사항) 코어 디바이스의 로깅 구성입니다. Greengrass 로그 구성 및 사용 방법에 대한 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 섹션을 참조하세요.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

level

(선택 사항) 출력할 로그 메시지의 최소 수준입니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

format

(선택 사항) 로그의 데이터 형식입니다. 다음 옵션 중 하나를 선택합니다.

- TEXT - 텍스트 형식으로 로그를 보려면 이 옵션을 선택합니다.
- JSON - [Greengrass CLI logs 명령](#)을 사용하여 로그를 보거나 프로그래밍 방식으로 로그와 상호 작용하려면 이 옵션을 선택합니다.

기본값: TEXT

outputType

(선택 사항) 로그의 출력 유형입니다. 다음 옵션 중 하나를 선택합니다.

- FILE - AWS IoT Greengrass 코어 소프트웨어에서 지정한 디렉터리의 파일에 로그를 출력합니다outputDirectory.
- CONSOLE - AWS IoT Greengrass 코어 소프트웨어에서 로그를 인쇄합니다stdout. 코어 디바이스에서 인쇄할 때 로그를 보려면 이 옵션을 선택합니다.

기본값: FILE

fileSizeKB

(선택 사항) 각 로그 파일의 최대 크기(KB)입니다. 로그 파일이 최대 파일 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어가 새 로그 파일을 생성합니다.

이 파라미터는 outputType에 대해 FILE을 지정한 경우에만 적용됩니다.

기본값: 1024

totalLogsSizeKB

(선택 사항) Greengrass nucleus를 포함하여 각 구성 요소에 대한 로그 파일의 최대 총 크기(KB)입니다. Greengrass nucleus의 로그 파일에는 [플러그인 구성 요소](#)의 로그도 포함됩니다. 구성 요소의 총 로그 파일 크기가 이 최대 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 해당 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 Greengrass nucleus(시스템) 및 각 구성 요소에 지정할 수 있는 [로그 관리자 구성 요소](#)의 [디스크 공간 제한](#) 파라미터(diskSpaceLimit)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값의 최소값을 Greengrass nucleus 및 각 구성 요소의 최대 총 로그 크기로 사용합니다.

이 파라미터는 outputType에 대해 FILE을 지정한 경우에만 적용됩니다.

기본값: 10240

outputDirectory

(선택 사항) 로그 파일의 출력 디렉터리입니다.

이 파라미터는 outputType에 대해 FILE을 지정한 경우에만 적용됩니다.

기본값: `/greengrass/v2/logs`, 여기서 `/greengrass/v2`는 AWS IoT Greengrass 루트 폴더입니다.

fleetstatus

이 파라미터는 이 구성 요소의 v2.1.0 이상에서 사용할 수 있습니다.

(선택 사항) 코어 디바이스의 플릿 상태 구성입니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

periodicStatusPublishIntervalSeconds

(선택 사항) 코어 디바이스가 AWS 클라우드에 디바이스 상태를 게시하는 간격의 시간(초)입니다.

최소: 86400(24시간)

기본값: 86400(24시간)

telemetry

(선택 사항) 코어 디바이스에 대한 시스템 상태 원격 측정 구성입니다. 원격 측정 지표 및 원격 측정 데이터를 활용하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집](#) 섹션을 참조하세요.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

enabled

(선택 사항) 원격 측정을 활성화 또는 비활성화할 수 있습니다.

기본값: true

periodicAggregateMetricsIntervalSeconds

(선택 사항) 코어 디바이스가 지표를 집계하는 간격(초)입니다.

이 값을 지원하는 최소값보다 낮게 설정하면 nucleus가 기본값을 대신 사용합니다.

최소: 3600

기본값: 3600

periodicPublishMetricsIntervalSeconds

(선택 사항) 코어 디바이스가 AWS 클라우드에 원격 측정 지표를 게시하는 간격의 시간(초)입니다.

이 값을 지원하는 최소값보다 낮게 설정하면 nucleus가 기본값을 대신 사용합니다.

최소: 86400

기본값: 86400

deploymentPollingFrequencySeconds

(선택 사항) 배포 알림을 위해 폴링할 기간(초)입니다.

기본값: 15

componentStoreMaxSizeBytes

(선택 사항) 구성 요소 레시피와 아티팩트로 구성된 구성 요소 스토어의 디스크 최대 크기입니다.

기본값: 10000000000(10GB)

platformOverride

(선택 사항) 코어 디바이스의 플랫폼을 식별하는 속성 사전입니다. 이를 사용하여 구성 요소 레시피가 구성 요소의 올바른 수명 주기 및 아티팩트를 식별하는 데 사용할 수 있는 사용자 지정 플랫폼 속성을 정의합니다. 예를 들어 하드웨어 기능 속성을 정의하여 구성 요소가 실행할 최소 아티팩트 세트만 배포할 수 있습니다. 자세한 내용은 구성 요소 레시피의 [매니페스트 플랫폼 파라미터](#)를 참조하세요.

이 파라미터를 사용하여 코어 디바이스의 os 및 architecture 플랫폼 속성을 재정의할 수도 있습니다.

httpClient

이 파라미터는 이 구성 요소의 v2.5.0 이상에서 사용할 수 있습니다.

(선택 사항) 코어 디바이스의 HTTP 클라이언트 구성입니다. 이러한 구성 옵션은 이 구성 요소의 모든 HTTP 요청에 적용됩니다. 코어 디바이스가 느린 네트워크에서 실행되는 경우 이러한 제한 시간을 늘려 HTTP 요청이 제한 시간에 도달하는 것을 방지할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

connectionTimeoutMs

(선택 사항) 연결 요청 시간이 초과되기 전에 연결이 열릴 때까지 대기하는 시간(밀리초)입니다.

기본값: 2000(2초)

socketTimeoutMs

(선택 사항) 연결 시간이 초과되기 전에 데이터가 열린 연결을 통해 전송될 때까지 대기하는 시간(밀리초)입니다.

기본값: 30000(30초)

deploymentConfigurationTimeSource

이 파라미터는 이 구성 요소의 v2.15.0 이상에서 사용할 수 있습니다.

(선택 사항) 배포를 처리할 때 사용할 타임스탬프입니다. 기본값은 deploymentCreationTime입니다.

이 객체에는 다음 값이 포함됩니다.

deploymentCreationTime

deploymentConfigurationTimeSource의 기본값입니다. 디바이스는 배포 생성 타임스탬프를 사용하여 처리 중에 구성 키 충돌을 해결합니다. 이 동작이 선택되면, nucleus가 보유한 로컬 디바이스 구성의 타임스탬프가 들어오는 배포보다 더 최신일 수 있으며, 이 경우 들어오는 구성 변경 사항은 오래된 것으로 간주되어 거부됩니다.

deploymentProcessingTime

디바이스는 배포 처리 중에 로컬 타임스탬프를 사용하여 구성 키 충돌을 해결합니다. 이 경우 디바이스는 배포 생성 타임스탬프가 아니라 처리 시점의 타임스탬프를 기준으로 구성을 업데이트합니다. 이 동작은 디바이스의 시계가 올바르게 보정되어 있다고 가정합니다.

새 디바이스가 처음 연결될 때부터 이 동작을 사용하도록 하려면 배포를 통해서가 아니라 초기 디바이스 이미지나 설치 과정에서 nucleus 설정을 구성합니다. 이 구성에는 nucleus classic 설치 관리자의 [--init-config](#) 옵션을 사용합니다.

이 초기 구성은 필수입니다. 디바이스가 여러 배포를 임의의 순서로 처리하기 때 문입니다. 적절한 초기 구성이 이루어지지 않으면, 디바이스가 nucleus 구성을 deploymentProcessingTime으로 지정하는 배포를 받기 전에 기본 동작인 deploymentCreationTime을 사용하여 배포를 처리할 수 있습니다.

Example예: 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.16.1	버그 수정 및 개선 사항 토큰 교환 서비스 실패 후 자격 증명 재시도 간격에 대한 구성을 추가합니다.

버전	변경 사항
2.16.0	새로운 기능 - Nucleus는 이제 cgroups V2를 지원합니다. 버그 수정 및 개선 사항 - 배포가 취소될 때 nucleus가 오래된 배포를 정리하지 않는 문제를 해결합니다.
2.15.1	버그 수정 및 개선 사항 부트스트랩 배포 중 30초 종료 제한 시간 이후에 nucleus가 구성 요소 프로세스를 정리하지 못하는 문제를 해결합니다.
2.15.0	새로운 기능 - CPU 및 OS 세부 정보와 같은 호스트 시스템 정보를 포함하는 원격 측정 기능을 추가합니다. 자세한 내용은 원격 측정 구성 요소 페이지를 참조하세요. - deploymentConfigurationTimeSource 구성을 추가합니다. 자세한 내용은 nucleus 구성 요소 구성을 참조하세요. 버그 수정 및 개선 사항 - 이제 Nucleus는 기본적으로 로컬 버전 및 아티팩트의 사용을 우선시하여, 배포의 일관성을 향상시키고 외부 종속성을 줄입니다. 사용자는 배포 문서에서 다른 기본 설정을 지정하여 이 동작을 재정의할 수 있습니다. - 또한 제공된 Windows 사용자가 Windows에서 일반적으로 허용되는 문자 집합과 일치하지 않을 경우 새로운 경고 로그가 추가됩니다.

버전	변경 사항
2.14.3	버그 수정 및 개선 사항 • 토큰 교환 서비스가 포트 구성 변경 시 재시작되도록 허용합니다. • 트리거되지 않은 고정되지 않은 Lambda의 경우, 플릿 상태 서비스가 구성 요소 상태 변경 메시지를 보내지 않는 문제를 수정합니다. • 새 구성 요소 버전 배포 시 구성 요소가 정상적으로 종료되지 않는 문제를 수정합니다. • 내장 플러그인의 주입 후크가 두 번 실행되어 불필요한 수명 주기 이벤트와 중복 로그가 발생하는 문제를 수정합니다. • Windows 디바이스에서 구성 요소 수명 주기 로그를 개선합니다.
2.14.2	버그 수정 및 개선 사항 • HTTP 클라이언트가 상호 인증으로 구성되지 않는 문제를 수정합니다.
2.14.1	버그 수정 및 개선 사항 • Greengrass를 새로 설치할 때 구성 요소가 제대로 중지되지 않는 문제를 수정합니다.

버전	변경 사항
2.14.0	 Warning 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 새로운 기능 • 새로운 듀얼 스택 엔드포인트 지원을 통해 IPv6 네트워크 통신이 가능합니다. • nucleus 재시작 실패 및 Launch 디렉터리 손상에 대한 복원력이 강화되었습니다. 버그 수정 및 개선 사항 • IPC PubSub 구독 종료 시 발생하던 메모리 누수 문제를 수정합니다. • skipif 조건이 true일 때 시작 시간 초과로 인해 구성 요소가 ERRORED 상태로 진입하는 실행 수명 주기 문제를 수정합니다. • TLS 정책이 TLS13_1_3_2022_10으로 설정된 경우 코어 디바이스가 AWS IoT Core 에 연결되지 않는 문제를 수정합니다.
2.13.0	새로운 기능 • Nucleus에서 FIPS 엔드포인트가 지원됩니다. 버그 수정 및 개선 사항 • 배포 개선 사항 취소: 이제 새 구성이 병합되는 동안과 서비스가 시작될 때까지 기다리는 동안 배포를 취소할 수 있습니다.
2.12.6	버그 수정 및 개선 사항 • Jetson Nano를 포함한 특정 ARMv8 프로세서에서 시작 시 충돌을 일으키는 문제를 해결합니다.

버전	변경 사항
2.12.5	버그 수정 및 개선 사항 - 하드 종속성으로 이전에 손상된 구성 요소를 롤백하는 동안 배포 롤백이 간헐적으로 중단되는 문제를 해결합니다. - 플릿 프로비저닝 후 nucleus가 상태 업데이트를 게시하지 않는 문제를 해결합니다. 404 오류 발생 이후 GetDeploymentConfiguration API에 대한 재시도가 추가됩니다.
2.12.4	버그 수정 및 개선 사항 일부 Linux 디바이스 시작 도중 nucleus가 교착 상태가 되는 문제를 해결합니다.
2.12.3	 Warning 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 사항 - nucleus가 다시 시작된 후 구성 요소 복구 중에 nucleus가 올바른 구성 요소 상태를 보고하지 않는 문제를 해결합니다. - 일반적인 버그 수정 및 개선 사항.
2.12.2	버그 수정 및 개선 사항 - 이전 로그가 제대로 정리되지 않은 문제를 해결합니다. - 일반적인 버그 수정 및 개선 사항.
2.12.1	버그 수정 및 개선 사항 nucleus가 배포 주제에 대한 MQTT 구독을 복제하여 추가 로깅 및 MQTT 게시가 발생할 수 있는 문제를 해결합니다.
2.12.0	새로운 기능 롤백 배포의 일부로 부트스트랩 수명 주기 단계를 실행할 수 있습니다.

버전	변경 사항
2.11.3	버그 수정 및 개선 사항 - 종속성이 실패할 때 구성 요소가 부적절하게 시작될 수 있는 nucleus 문제를 해결합니다. 새로운 기능 - 구성 가능한 S3 엔드포인트 유형이 추가됩니다.
2.11.2	버그 수정 및 개선 사항 - 많은 수(50개 초과)의 구독이 사용 중일 때 오프라인으로 표시될 수 있는 nucleus MQTT 5 클라이언트의 문제를 해결합니다. - Docker 다이얼 TCP 실패에 대한 재시도가 추가됩니다.
2.11.1	버그 수정 및 개선 사항 - 부트스트랩 작업이 실패하고 배포 메타데이터 파일이 손상된 경우 nucleus가 시작되지 않는 문제를 해결합니다. - 온디맨드 Lambda 구성 요소가 배포 상태 업데이트에 보고되지 않는 문제를 해결합니다. - 중복 권한 부여 정책 ID에 대한 지원을 추가합니다.
2.11.0	새로운 기능 - 로컬 배포를 취소할 수 있습니다. - 로컬 배포 실패 처리 정책을 구성할 수 있습니다. - 디스크 스펠러 플러그인에 대한 지원을 추가합니다.
2.10.3	버그 수정 및 개선 사항 PKCS#11 공급자를 사용할 때 Greengrass가 배포 알림을 구독하지 않는 문제를 해결합니다.
2.10.2	버그 수정 및 개선 사항 - 구성 요소 수명 주기의 대소문자를 구분하지 않는 구문 분석이 허용됩니다. - 환경 PATH 변수가 올바르게 다시 생성되지 않은 문제를 해결합니다. - 특수 문자가 있는 사용자 이름의 스트림 관리자를 포함한 구성 요소에 대한 프록시 URI 인코딩이 수정됩니다.

버전	변경 사항
2.10.1	버그 수정 및 개선 사항 • Jetson Nano를 포함한 특정 ARMv8 프로세서에서 시작 시 충돌을 일으키는 문제를 해결합니다. • Greengrass가 구성 요소의 표준에서 더 이상 종료되지 않고 동작을 2.10.0 이전 동작으로 되돌립니다.
2.10.0	새로운 기능 • 빈 정규식에 대한 <code>interpolateComponentConfiguration</code> 지원이 추가됩니다. 이제 Greengrass가 루트 구성 객체에서 보간합니다. • MQTT5에 대한 지원을 추가합니다. • 스캔 없이 플러그인 구성 요소를 빠르게 로드하는 메커니즘이 추가됩니다. • 사용하지 않는 Docker 이미지를 삭제하여 Greengrass가 디스크 공간을 절약할 수 있습니다. 버그 수정 및 개선 사항 • 롤백이 배포에서 특정 구성 값을 그대로 두는 문제를 해결합니다. • Greengrass nucleus가 사용자 지정 비AWS 자격 증명 및 데이터 엔드포인트의 AWS 도메인 시퀀스를 검증하는 문제를 해결합니다. • 다중 그룹 종속성 확인을 업데이트하여 활성 버전으로 잠그는 대신 AWS 클라우드 협상을 통해 모든 그룹 종속성을 다시 해결합니다. 이 업데이트를 통해 배포 오류 코드 <code>INSTALLED_COMPONENT_NOT_FOUND</code>도 제거됩니다. • Greengrass nucleus가 업데이트되어 로컬에 이미 있는 Docker 이미지의 다운로드를 건너뜁니다. • Greengrass nucleus가 업데이트되어 제한 시간이 만료되기 전에 구성 요소 설치 단계를 다시 시작합니다. • 사소한 추가 수정 및 개선 사항.

버전	변경 사항
2.9.6	버그 수정 및 개선 사항 LAUNCH_DIRECTORY_CORRUPTED 오류로 인해 Greengrass 배포가 실패하고 후속 디바이스 재부팅에서 Greengrass가 시작되지 않는 문제를 해결합니다. 이 오류는 Greengrass 디바이스를 Greengrass를 다시 시작해야 하는 배포가 있는 여러 사물 그룹 간에 이동할 때 발생할 수 있습니다.
2.9.5	새로운 기능 - Greengrass nucleus 소프트웨어 서명 확인에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - 로컬 레시피 메타데이터 리전이 Greengrass nucleus 시작 리전과 일치하지 않을 때 배포가 실패하는 문제를 해결합니다. 이러한 일이 발생할 때 Greengrass nucleus는 이제 클라우드와 재협상합니다. - MQTT 메시지 스폰서가 메시지를 채우고 메시지가 제거되지 않는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항.
2.9.4	버그 수정 및 개선 사항 - QOS 0 메시지를 삭제하기 전에 null 메시지를 확인합니다. 1,024자 제한을 초과하는 경우 작업 상태 세부 정보 값이 잘립니다. - Windows용 부트스트랩 스크립트가 업데이트되어 해당 경로에 공백이 포함된 경우 Greengrass 루트 경로를 올바르게 읽습니다. - 구독 응답이 전송되지 않은 경우 클라이언트 메시지를 삭제 AWS IoT Core 하도록 구독을 업데이트합니다. - 기본 구성 파일이 손상되거나 누락된 경우 nucleus가 백업 파일에서 구성을 로드하게 됩니다.

버전	변경 사항
2.9.3	버그 수정 및 개선 사항 • MQTT 클라이언트 ID가 중복되지 않게 됩니다. • 더 강력한 파일 읽기 및 쓰기 기능이 추가되어 손상 방지와 복구가 가능합니다. • 특정 네트워크 관련 오류 시 Docker 이미지 가져오기를 재시도합니다. • MQTT 연결에 대해 noProxyAddresses 옵션이 추가됩니다.
2.9.2	버그 수정 및 개선 사항 • interpolateComponentConfiguration 구성이 진행 중인 배포에 적용되지 않는 문제를 해결합니다. • OSHI를 사용하여 모든 하위 프로세스가 나열됩니다.
2.9.1	버그 수정 및 개선 사항 • 배포가 플러그인 구성 요소를 제거하는 경우 Greengrass가 다시 시작되는 수정 사항이 추가됩니다.
2.9.0	새로운 기능 • 더 작은 디바이스 하위 집합으로 배포를 재시도하는 하위 배포 생성 기능이 추가됩니다. 이 기능을 사용하면 보다 효율적으로 실패한 배포를 테스트하고 해결할 수 있습니다. 버그 수정 및 개선 사항 • useradd, groupadd 및 usermod가 없는 시스템에 대한 지원이 개선됩니다. • 사소한 추가 수정 및 개선 사항.
2.8.1	버그 수정 및 개선 사항 • Greengrass API 오류에서 배포 오류 코드가 올바르게 생성되지 않은 문제를 해결합니다. • 배포 중에 구성 요소가 ERRORED 상태에 도달할 때 플릿 상태 업데이트가 부정확한 정보를 전송하는 문제를 해결합니다. • Greengrass에 기존 구독이 50개보다 많을 때 배포를 완료할 수 없는 문제를 해결합니다.

버전	변경 사항
2.8.0	새로운 기능 - 코어 디바이스에 구성 요소를 배포하는 데 문제가 있는 경우 자세한 오류 코드가 포함된 배포 상태 응답을 보고하도록 Greengrass nucleus가 업데이트됩니다. 자세한 내용은 세부 배포 오류 코드 단원을 참조하십시오. - 구성 요소가 BROKEN 또는 ERRORED 상태가 될 때 자세한 오류 코드가 포함된 구성 요소 상태 응답을 보고하도록 Greengrass nucleus가 업데이트됩니다. 자세한 내용은 세부 구성 요소 상태 코드 단원을 참조하십시오. - 상태 메시지 필드가 확장되어 디바이스의 클라우드 가용성 정보가 개선됩니다. - 플릿 상태 서비스 견고성이 개선됩니다. 버그 수정 및 개선 사항 - 구성이 변경될 때 손상된 구성 요소를 다시 설치할 수 있습니다. - 부트스트랩 배포 중 nucleus 재시작으로 인해 배포가 실패하는 문제를 해결합니다. - Windows에서 루트 경로에 공백이 있을 때 설치가 실패하는 문제를 해결합니다. - 배포 중에 구성 요소가 종료되고 새 버전의 종료 스크립트를 사용하는 문제를 해결합니다. - 다양한 종료 개선 사항. - 사소한 추가 수정 및 개선 사항.

버전	변경 사항
2.7.0	새로운 기능 - 코어 디바이스가 로컬 배포를 적용할 때 AWS IoT Greengrass 클라우드로 상태 업데이트를 보내도록 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 iotdata로 설정할 수 있습니다. 자세한 내용은 프라이빗 CA에서 서명한 디바이스 인증서 사용 단원을 참조하십시오. 버그 수정 및 개선 사항 - nucleus가 중지되거나 다시 시작될 때 Greengrass nucleus가 특정 시나리오에서 배포를 롤백하는 문제를 해결합니다. 이제 nucleus가 다시 시작된 후 nucleus가 배포를 재개합니다. - 소프트웨어를 시스템 서비스로 설정하도록 지정할 때 --start 인수를 준수하도록 Greengrass 설치 관리자가 업데이트됩니다. - nucleus가 구성 요소를 업데이트한 이벤트에서 배포 ID를 설정하도록 SubscribeToComponentUpdates의 동작이 업데이트됩니다. - 사소한 추가 수정 및 개선 사항.

버전	변경 사항
2.6.0	새로운 기능 - 로컬 게시/구독 주제를 구독할 때 MQTT 와일드카드에 대한 지원을 추가합니다. 자세한 내용은 로컬 메시지 게시/구독 및 SubscribeToTopic을(를) 참조하세요. - 구성 요소 구성에서 <code>component_dependency_name</code> :configuration: <code>json_pointer</code> 레시피 변수 이외의 레시피 변수에 대한 지원을 추가합니다. 레시피에서 구성 요소의 DefaultConfiguration 을 정의하거나 배포에서 구성 요소를 구성할 때 이러한 레시피 변수를 사용할 수 있습니다. 이 기능을 활성화하려면 interpolateComponentConfiguration 구성 옵션을 true로 설정합니다. 자세한 내용은 레시피 변수 및 병합 업데이트에서 레시피 변수 사용을(를) 참조하세요. - IPC(프로세스 간 통신) 권한 부여 정책의 * 와일드카드에 대한 전체 지원이 추가됩니다. 이제 모든 문자 조합과 일치하도록 리소스 문자열에 * 문자를 지정할 수 있습니다. 자세한 내용은 권한 부여 정책의 와일드카드 단원을 참조하십시오. - 사용자 지정 구성 요소에서 Greengrass CLI가 사용하는 IPC 작업을 직접 호출할 수 있도록 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 참조하세요. 버그 수정 및 개선 사항 - 특정 시나리오에서 하드 종속성이 다시 시작되거나 상태가 변경될 때 종속 구성 요소가 반응하지 않는 문제를 해결합니다. - 배포가 실패할 때 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 보고하는 오류 메시지를 개선합니다. - nucleus가 다시 시작되는 특정 시나리오에서 Greengrass nucleus가 사물 배포를 두 번 적용하는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.

버전	변경 사항
2.5.6	새로운 기능 - ECC 키를 사용하는 하드웨어 보안 모듈에 대한 지원을 추가합니다. HSM(하드웨어 보안 모듈)을 사용하여 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 내용은 하드웨어 보안 통합 단원을 참조하십시오. 버그 수정 및 개선 사항 - 특정 시나리오에서 설치 스크립트가 손상된 구성 요소를 배포할 때 배포가 완료되지 않는 문제를 해결합니다. - 시작 중 성능이 개선됩니다. - 사소한 추가 수정 및 개선 사항.
2.5.5	새로운 기능 - 구성 요소에 대해 GG_ROOT_CA_PATH 환경 변수가 추가되어 사용자 지정 구성 요소의 루트 CA(인증 기관) 인증서에 액세스할 수 있습니다. 버그 수정 및 개선 사항 - 영어 이외의 표시 언어를 사용하는 Windows 디바이스에 대한 지원을 추가합니다. - Greengrass nucleus가 부울 설치 관리자 인수를 구문 분석하는 방법이 업데이트되어 부울 값 없이 부울 인수를 지정하여 true 값을 지정할 수 있습니다. 예를 들어 이제 --provision true 대신 --provision 을 지정하여 자동 리소스 프로비저닝을 사용하여 설치할 수 있습니다. - 또한 특정 시나리오에서 프로비저닝 후 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 상태를 보고하지 않는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항.
2.5.4	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.

버전	변경 사항
2.5.3	새로운 기능 - 하드웨어 보안 통합에 대한 지원을 추가합니다. HSM(하드웨어 보안 모듈)을 사용하여 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있습니다. 자세한 내용은 하드웨어 보안 통합 단원을 참조하십시오. 버그 수정 및 개선 사항 - nucleus가 AWS IoT Core와의 MQTT 연결을 설정하는 동안 런타임 예외 문제를 해결합니다.
2.5.2	버그 수정 및 개선 사항 Greengrass nucleus가 업데이트된 후 Windows 서비스가 중지하거나 디바이스를 재부팅한 후 다시 시작되지 않는 문제를 해결합니다.
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 사항 - Windows에서 Java 런타임 환경(JRE)의 32비트 버전에 대한 지원을 추가합니다. - AWS IoT 정책이 <code>greengrass:ListThingGroupsForCoreDevice</code> 권한을 부여하지 않는 코어 디바이스에 대한 사물 그룹 제거 동작이 변경됩니다. 이 버전을 사용하면 배포가 계속되고, 경고가 기록되며, 사물 그룹에서 코어 디바이스를 제거할 때 구성 요소가 제거되지 않습니다. 자세한 내용은 디바이스에 AWS IoT Greengrass 구성 요소 배포 단원을 참조하십시오. - Greengrass 구성 요소 프로세스에 Greengrass nucleus가 제공되도록 시스템 환경 변수 관련 문제를 해결합니다. 이제 구성 요소를 다시 시작하여 최신 시스템 환경 변수를 사용할 수 있습니다.

버전	변경 사항
2.5.0	새로운 기능 - Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. - 사물 그룹 제거 동작이 변경됩니다. 이 버전을 사용하면 사물 그룹에서 코어 디바이스를 제거하여 다음 배포에서 사물 그룹의 구성 요소를 제거할 수 있습니다. 이 변경으로 인해 코어 디바이스의 AWS IoT 정책에 <code>greengrass:ListThingGroupsForCoreDevice</code> 권한이 있어야 합니다. AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝한 경우 기본 AWS IoT 정책은 이 권한을 <code>greengrass:*</code> 포함하는 것을 허용합니다. 자세한 내용은 에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass 단원을 참조하십시오. - HTTPS 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 단원을 참조하십시오. - 새 <code>windowsUser</code> 구성 파라미터가 추가됩니다. 이 파라미터를 사용하여 Windows 코어 디바이스에서 구성 요소 실행에 사용할 기본 사용자를 지정할 수 있습니다. 자세한 내용은 구성 요소를 실행하는 사용자 구성 단원을 참조하십시오. - 느린 네트워크에서 성능을 개선하기 위해 HTTP 요청 제한 시간을 사용자 지정하는 데 사용할 수 있는 <code>httpClient</code> 구성 옵션이 새로 추가됩니다. 자세한 내용은 httpClient 구성 파라미터를 참조하세요. 버그 수정 및 개선 사항 - 부트스트랩 수명 주기 옵션이 수정되어 구성 요소에서 코어 디바이스를 다시 시작합니다. - 레시피 변수의 하이픈에 대한 지원을 추가합니다. - 온디맨드 Lambda 함수 구성 요소에 대한 IPC 권한 부여가 수정됩니다. - 로그 메시지가 개선되고 중요하지 않은 로그가 INFO에서 DEBUG 수준으로 변경되어 유용성이 증가합니다. 자동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 Greengrass nucleus가 생성하는 기본 토큰 교환 역할에서 <code>iot:DescribeCertificate</code> 권한을 제거합니다. 이 권한은 Greengrass nucleus에서 사용하지 않습니다.

버전	변경 사항
	• <code>iam:CreatePolicy</code> 를 동일한 정책에 사용할 수 있는 경우 자동 프로비저닝 스크립트에 <code>iam:GetPolicy</code> 권한이 필요하지 않도록 문제를 해결합니다. • 사소한 추가 수정 및 개선 사항.

2.4.0

새로운 기능

- 시스템 리소스 제한에 대한 지원을 추가합니다. 각 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 [구성 요소에 대한 시스템 리소스 제한 구성](#) 단원을 참조하십시오.
- 구성 요소를 일시 중지하고 재개하도록 IPC 작업이 추가됩니다. 자세한 내용은 [PauseComponent](#) 및 [ResumeComponent](#)을(를) 참조하세요.
- 플러그인 프로비저닝에 대한 지원을 추가합니다. 설치 중에 실행할 JAR 파일을 지정하여 Greengrass 코어 디바이스에 필요한 AWS 리소스를 프로비저닝할 수 있습니다. Greengrass nucleus에는 사용자 지정 프로비저닝 플러그인 개발을 구현할 수 있는 인터페이스가 포함되어 있습니다. 자세한 내용은 [사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.
- AWS IoT Greengrass 코어 소프트웨어 설치 관리자에 선택적 thing-name-policy 인수를 추가합니다. 자동 리소스 프로비저닝으로 코어 소프트웨어를 설치할 때 옵션을 사용하여 기존 또는 사용자 지정 AWS IoT 정책을 지정할 수 있습니다. [AWS IoT Greengrass](#)

버그 수정 및 개선 사항

- 시작 시 로깅 구성이 업데이트됩니다. 이렇게 하면 시작 시 로깅 구성이 적용되지 않는 문제를 해결합니다.
- 설치 중에 Greengrass 루트 폴더의 구성 요소 스토어를 가리키도록 nucleus 로더 심링크가 업데이트됩니다. 이 업데이트를 통해 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 다운로드하는 JAR 파일 및 기타 nucleus 아티팩트를 삭제할 수 있습니다.
- 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 [릴리스](#)를 참조하세요.

버전	변경 사항
2.3.0	새로운 기능 - 최대 10MB의 배포 구성 문서, 7KB(대상 사물 배포의 경우) 또는 31KB(대상 사물 그룹 배포의 경우)에 대한 지원을 추가합니다. 이 기능을 사용하려면 코어 디바이스의 AWS IoT 정책에서 <code>greengrass:GetDeploymentConfiguration</code> 권한을 허용해야 합니다. AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝한 경우 코어 디바이스의 AWS IoT 정책은 이 권한이 <code>greengrass:*</code> 포함된를 허용합니다. 자세한 내용은 에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass 단원을 참조하십시오. <code>iot:thingName</code> 레시피 변수가 추가됩니다. 이 레시피 변수를 사용하여 레시피에서 코어 디바이스의 AWS IoT 사물 이름을 가져올 수 있습니다. 자세한 내용은 레시피 변수 단원을 참조하십시오. 버그 수정 및 개선 사항 - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.
2.2.0	새로운 기능 - 로컬 새도 관리를 위한 IPC 작업이 추가됩니다. 버그 수정 및 개선 사항 - JAR 파일의 크기가 감소합니다. - 메모리 사용량이 감소합니다. - 경우에 따라 로그 구성이 업데이트되지 않는 문제가 해결됩니다. - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.

버전	변경 사항
2.1.0	새로운 기능 • Amazon ECR의 프라이빗 리포지토리에서 Docker 이미지 다운로드가 지원됩니다. • 다음 파라미터가 추가되어 코어 디바이스에서 MQTT 구성을 사용자 지정합니다. • <code>maxInFlightPublishes</code> - 동시에 이동할 수 있는 확인되지 않은 MQTT QoS 1 메시지의 최대 수입니다. • <code>maxPublishRetry</code> - 게시에 실패한 메시지의 최대 재시도 횟수입니다. • <code>fleetstatusservice</code> 구성 파라미터가 추가되어 코어 디바이스가 디바이스 상태를 AWS 클라우드에 게시하는 간격을 구성합니다. • 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요. 버그 수정 및 개선 사항 • <code>nucleus</code>가 다시 시작될 때 새도 배포가 중복되는 문제를 해결합니다. • 서비스 로드 예외가 발생했을 때 <code>nucleus</code>가 충돌하는 문제를 해결합니다. • 순환 종속성이 포함된 배포 실패에 대한 구성 요소 종속성 확인이 개선됩니다. • 해당 구성 요소가 이전에 코어 디바이스에서 제거된 경우 플러그인 구성 요소가 재배포되지 않는 문제를 해결합니다. • HOME 환경 변수가 Lambda 구성 요소 또는 루트로 실행되는 구성 요소의 <code>/greengrass/v2 /work</code> 디렉터리로 설정되는 문제가 해결됩니다. 이제 HOME 변수가 구성 요소를 실행하는 사용자의 홈 디렉터리로 올바르게 설정됩니다. • 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.

버전	변경 사항
2.0.5	버그 수정 및 개선 사항 - AWS제공된 구성 요소를 다운로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. - AWS 중국 리전에서 올바른 Greengrass 데이터 영역 엔드포인트를 사용합니다.
2.0.4	새로운 기능 - 포트 443을 통한 HTTPS 트래픽이 활성화됩니다. nucleus 구성 요소의 버전 2.0.4에 대한 새 greengrassDataPlanePort 구성 파라미터를 사용하여 기본 포트 8443 대신 포트 443을 통해 이동하도록 HTTPS 통신을 구성할 수 있습니다. 자세한 내용은 포트 443을 통해 HTTPS 구성 단원을 참조하십시오. - 작업 경로 레시피 변수가 추가됩니다. 이 레시피 변수를 사용하여 구성 요소의 작업 폴더 경로를 가져올 수 있고, 이 폴더를 사용하여 구성 요소와 해당 종속성 간에 파일을 공유할 수 있습니다. 자세한 내용은 작업 경로 레시피 변수를 참조하세요. 버그 수정 및 개선 사항 - 역할 정책이 이미 있는 경우 토큰 교환 AWS Identity and Access Management (IAM) 역할 정책의 생성을 방지합니다. 이 변경으로 인해 --provision true 로 실행할 때 설치 관리자에 iam:GetPolicy 및 sts:GetCallerIdentity 가 필요합니다. 자세한 내용은 설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책 단원을 참조하십시오. - 아직 성공적으로 등록되지 않은 배포의 취소가 올바르게 처리됩니다. - 배포를 롤백할 때 최신 타임스탬프가 있는 이전 항목을 제거하도록 구성이 업데이트됩니다. - 사소한 추가 수정 및 개선 사항. 자세한 내용은 GitHub의 릴리스를 참조하세요.
2.0.3	초기 버전입니다.

Greengrass nucleus lite

Greengrass nucleus lite(`aws.greengrass.NucleusLite`)는 최소 메모리 사용(5MB RAM 미만)에 최적화된 제한된 엣지 디바이스용 디바이스 런타임입니다. AWS IoT Greengrass 버전 2.14.0 릴리스에 도입되었으며 AWS IoT Greengrass 일반 구성 요소, Greengrass V2 API 및 SDK와 역호환됩니다.

Greengrass nucleus lite는 일반 [Greengrass nucleus\(`aws.greengrass.Nucleus`\)](#)를 대체하는 옵션으로 제공되며, 이기종 Greengrass 디바이스로 구성된 플릿에서도 사용할 수 있습니다.

주제

- [버전](#)
- [운영 체제](#)
- [요구 사항](#)
- [호환성](#)
- [다운로드 및 설치](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux(systemd를 사용한 배포)

자세한 내용은 [Greengrass nucleus](#)를 참조하세요.

요구 사항

AWS IoT Greengrass nucleus lite 및 AWS IoT Greengrass 코어 소프트웨어를 설치하고 실행하려면 디바이스가 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [설정 가이드](#)를 참조하세요.

- nucleus 런타임을 위한 5MB RAM 공간.
- 5MB 저장 공간(디스크/FLASH).

추가 시스템 의존성은 [설정 가이드](#)에 문서화되어 있습니다.

VPC에서 Greengrass nucleus 구성 요소의 실행이 지원됩니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.

- Greengrass nucleus는 AWS IoT 데이터, AWS IoT 자격 증명 및 Amazon S3에 연결되어 있어야 합니다.

호환성

AWS IoT Greengrass nucleus lite는 AWS IoT Greengrass v2 API(의 하위 세트) 및 지원되는 SDKs. 특정 언어 런타임/VM에 의존하지 않지만, 배포에 추가되는 구성 요소는 특정 런타임(Java JVM, Python 등)의 존재를 요구할 수 있습니다. nucleus lite에서 지원되는 기능에 대한 자세한 내용은 [Greengrass 기능 호환성](#) 섹션을 참조하세요.

다운로드 및 설치

apt 패키지를 다운로드하거나, [소스에서 빌드](#)하거나, [Yocto 계층을 사용](#)하거나, [사전 빌드된 Yocto 이미지를 다운로드](#)할 수 있습니다(예: Raspberry Pi). [AWS IoT Core 콘솔](#)에서는 디바이스에 필요한 모든 자격 증명과 초기 설정이 포함된 연결 키트를 다운로드할 수 있습니다. 각 배포 방법별 설치 지침이 포함되어 있습니다.

자습서에 따라 AWS IoT Greengrass nucleus lite를 설치하고 Greengrass 구성 요소 개발을 탐색할 수도 있습니다. 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#) 단원을 참조하십시오.

구성

nucleus는 다음 [구성](#) 파라미터를 지원합니다. 일부 파라미터를 사용하려면 AWS IoT Greengrass 코어 소프트웨어를 다시 시작해야 합니다.

iotRoleAlias

토큰 교환 IAM AWS IoT 역할을 가리키는 역할 별칭입니다. 자격 AWS IoT 증명 공급자는 Greengrass 코어 디바이스가 서비스와 상호 작용 AWS 할 수 있도록이 역할을 말합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용하도록 권한 부여를 참조하세요](#).

iotDataEndpoint

의 AWS IoT 데이터 엔드포인트입니다 AWS 계정.

iotCredEndpoint

의 AWS IoT 자격 증명 엔드포인트입니다 AWS 계정.

greengrassDataPlanePort

데이터 플레인 연결에 사용할 포트입니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#)을 참조하세요.

Important

디바이스가 아웃바운드 연결을 수행할 수 있는 포트를 지정해야 합니다. 차단된 포트를 지정하면 디바이스가 연결하여 배포를 수신 AWS IoT Greengrass 할 수 없습니다. 다음 옵션 중 하나를 선택합니다.

- 443
- 8443
- 기본값: 8443

awsRegion

사용할 AWS 리전입니다.

runWithDefault

구성 요소 실행에 사용할 시스템 사용자.

Important

이 구성 파라미터에 변경 사항을 배포하면 AWS IoT Greengrass 코어 소프트웨어가 다시 시작되어 변경 사항이 적용됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

posixUser

코어 디바이스가 일반 구성 요소를 실행하는 데 사용하는 시스템 사용자 및 선택적으로 시스템 그룹의 이름 또는 ID입니다. `user:group` 형식으로 사용자와 그룹을 콜론(:)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 사용자의 기본 그룹을 사용합니다. 예를 들어 `ggc_user` 또는 `ggc_user:ggc_group`을 지정할 수 있습니다. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 단원을 참조하십시오.

networkProxy

(선택 사항) 모든 연결에 사용할 네트워크 프록시입니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 단원을 참조하십시오.

Important

이 구성 파라미터에 대한 변경 사항을 배포하면 다음에 AWS IoT Greengrass 코어 소프트웨어를 다시 시작한 후 변경 사항이 적용됩니다.

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

`http://host:port` 형식으로 표시되는 프록시 서버의 URL입니다.

- `scheme` - 체계로, `http`여야 합니다.
- `host` - 프록시 서버의 호스트 이름 또는 IP 주소입니다.
- `port` - (선택 사항) 포트 번호입니다. 포트를 지정하지 않으면 Greengrass 코어 디바이스에서는 다음 기본값이 사용됩니다.
 - `http` - 80

로컬 로그 파일

메시지는 stdout에 로깅되고 로그 파일은 systemd에서 처리됩니다.

이 구성 요소의 로그를 보려면

- 로그를 확인하려면 `journalctl`을 사용합니다.

Changelog

버전	변경 사항
2.3.2	버그 수정 및 개선 사항 • 올바른 nucleus 버전 보고를 위해 버전 파일을 업데이트합니다.
2.3.1	버그 수정 및 개선 사항 • GG는 취소된 작업의 배포 상태를 업데이트하려고 시도하지 않습니다. • fdopendir의 DIR이 닫혀 있는지 확인합니다. • 플릿 프로비저닝은 이제 각 실행에서 인증서 파일 덮어쓰기를 트리거합니다. • 기타 마이너 버그 수정.
2.3.0	새로운 기능 • IoT Core MQTT 권한 부여를 위한 TPM 2.0 사용 지원. • 샘플 Apt 패키지는 이제 Ubuntu 22.04, Ubuntu 24.04, Debian 12, Debian 13 등 더 많은 운영 체제를 지원합니다. • 이제 RestartComponent IPC가 지원됩니다. 버그 수정 및 개선 사항 • 로컬 배포에는 더 이상 인터넷 액세스가 필요하지 않습니다. • Greengrass Nucleus 런타임 동작과 일치하도록 GetConfiguration이 업데이트되었습니다. (변경 위반) • 일반적인 버그 수정 및 개선 사항.

버전	변경 사항
2.2.2	버그 수정 및 개선 사항 • 변경되지 않은 구성 요소 버전이 포함된 수정 배포에서, 실행 가능한 아티팩트가 이미 실행 중인 경우 배포 실패 문제를 수정합니다. • 중첩된 하위 객체가 4개를 초과하는 경우 레시피 변수 보간 문제를 수정합니다. • 따옴표와 셸 특수 문자를 포함한 레시피 변수 보간을 수정합니다. • MQTT 연결 실패 시 발생하던 fds 누수 문제를 수정합니다.
2.2.1	버그 수정 및 개선 사항 • nucleus가 TES 자격 증명을 가져오지 못하는 문제를 수정합니다.
2.2.0	새로운 기능 • 컨테이너 이미지 아티팩트 URI에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 • 일반적인 버그 수정 및 개선 사항.
2.1.0	새로운 기능 • networkProxy 구성 옵션을 사용하여 설정 가능한 HTTP 프록시 지원을 추가합니다. 버그 수정 및 개선 사항 • 구버전 libcurl을 실행하는 장치를 위해 libcurl 요구 버전을 7.86에서 7.82로 낮췄습니다. • journalctl 로그를 recipe-runner 대신 구성요소에 귀속되도록 업데이트했습니다. • IPC 호출의 오류 응답을 개선합니다. • 일반 구성 요소 아티팩트의 S3 다운로드 시 재시도 기능을 추가합니다. • 사소한 버그를 수정했습니다.
2.0.2	버그 수정 및 개선 사항 • apt 패키지의 종속성에 cgroup-tools 를 포함하도록 수정합니다.

버전	변경 사항
2.0.1	버그 수정 및 개선 사항 • Greengrass 레시피의 timeout 섹션에 대한 레시피 변수 보간 지원을 추가합니다. • 스트림 관리자의 ValidateAuthorizationToken IPC 명령 지원을 추가합니다. • 플릿 프로비저닝 관련 경고를 수정합니다. • 작업 리스너에 재시도 및 백오프 기능을 추가합니다. • 일반적인 버그 수정 및 개선 사항.
2.0.0	초기 버전입니다.

클라이언트 디바이스 인증

클라이언트 디바이스 인증 구성 요소(`aws.greengrass.clientdevices.Auth`)에서는 클라이언트 디바이스가 인증되고 클라이언트 디바이스 작업이 승인됩니다.

Note

클라이언트 디바이스는 Greengrass 코어 디바이스에 연결되어 처리할 MQTT 메시지와 데이터를 보내는 로컬 IoT 디바이스입니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

Note

클라이언트 디바이스 인증 버전 2.3.0은 중단되었습니다. 클라이언트 디바이스 인증 버전 2.3.1 이상으로 업그레이드하는 것이 좋습니다.

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 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 nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- [Greengrass 서비스 역할](#)은 연결되어 AWS 계정 있어야 하며 `iot:DescribeCertificate` 권한을 허용해야 합니다.
- 코어 디바이스의 AWS IoT 정책은 다음 권한을 허용해야 합니다.
 - 이 구성 요소가 실행되는 코어 디바이스의 ARN이 리소스에 포함되어 있는 `greengrass:GetConnectivityInfo`
 - 코어 디바이스에 연결되는 각 클라이언트 디바이스의 Amazon 리소스 이름(ARN)이 리소스에 포함되어 있는 `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:PutCertificateAuthorities`
 - 다음 MQTT 주제의 ARN이 리소스에 포함되어 있는 `iot:Publish`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get`
 - 다음 MQTT 주제 필터의 ARN이 리소스에 포함되어 있는 `iot:Subscribe`
 - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
 - 다음 MQTT 주제의 ARN이 리소스에 포함되어 있는 `iot:Receive`
 - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

자세한 내용은 [AWS IoT 데이터 영역 작업에 대한 정책](#) 및 [클라이언트 디바이스를 지원하는 최소 AWS IoT 정책](#) 섹션을 참조하세요.

- (선택 사항) 오프라인 인증을 사용하려면 서비스에서 AWS IoT Greengrass 사용하는 AWS Identity and Access Management (IAM) 역할에 다음 권한이 포함되어야 합니다.
 - 코어 디바이스에서 사용하여 오프라인 인증 대상 클라이언트를 나열할 수 있는 `greengrass:ListClientDevicesAssociatedWithCoreDevice`입니다.
- VPC에서 클라이언트 디바이스 인증 구성 요소의 실행이 지원됩니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.
 - 클라이언트 디바이스 인증 구성 요소에는 AWS IoT data AWS IoT , 자격 증명 및 Amazon S3에 대한 연결이 있어야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	Port	필수	설명
iot. region .amazonaws.com	443	예	AWS IoT 사물 인증서에 대한 정보를 가져오는 데 사용됩니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.5.5

다음 표에는 이 구성 요소의 버전 2.5.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.17.0	소프트

2.5.4

다음 표에는 이 구성 요소의 버전 2.5.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.16.0	소프트

2.5.2 – 2.5.3

다음 표에는 이 구성 요소의 버전 2.5.2 및 2.5.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.15.0	소프트

2.5.1

다음 표에는 이 구성 요소의 버전 2.5.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.14.0	소프트

2.4.4 - 2.5.0

다음 표에는 이 구성 요소의 버전 2.4.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.13.0	소프트

2.4.3

다음 표에는 이 구성 요소의 버전 2.4.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.10.0	소프트

2.3.0

다음 표에는 이 구성 요소의 버전 2.3.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.10.0	소프트

2.2.3

다음 표에는 이 구성 요소의 버전 2.2.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <=2.9.0	소프트

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <=2.8.0	소프트

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.8.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.6.0 <2.7.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.7.0	소프트

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.2.0 < 2.5.0$	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.2.0 < 2.4.0$	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.2.0 < 2.3.0$	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

 Note

구독 권한은 로컬 MQTT 브로커에 대한 클라이언트 구독 요청 중에 평가됩니다. 클라이언트의 기존 구독 권한이 취소되면 클라이언트에서는 더는 주제를 구독할 수 없습니다. 그러나 이전에 구독한 주제의 메시지는 계속 수신됩니다. 이 동작을 방지하려면 강제로 클라이언트의 권한이 다시 부여되도록 구독 권한을 취소한 후 로컬 MQTT 브로커를 다시 시작해야 합니다. MQTT 5 브로커(EMQX) 구성 요소의 경우 `restartIdentifier` 구성을 업데이트하여 MQTT 5 브로커를 다시 시작합니다.

MQTT 3.1.1 브로커(Moquette) 구성 요소의 경우 서버 인증서가 변경되어 클라이언트에 강제로 다시 권한이 부여되면 기본적으로 매주 다시 시작됩니다. 코어 디바이스의 연결 정보(IP 주소)를 변경하거나 배포를 적용하여 브로커 구성 요소를 제거한 다음에 나중에 다시 배포하여 강제로 다시 시작할 수 있습니다.

v2.5.0 – 2.5.4

deviceGroups

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바

이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하거나 끝나는 클라이언트 디바이스와 일치하도록 사물 이름의 시작과 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 thingName: MyTeam\\:ClientDevice1를 지정하여 이름이 MyTeam:ClientDevice1인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- thingName - 클라이언트 디바이스 사물의 이름입니다 AWS IoT .

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 MyClientDevice1 또는 MyClientDevice2인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 MyClientDevice로 끝나는 클라이언트 디바이스가 일치됩니다.

```
thingName: *MyClientDevice
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. policies 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` - 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* - 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.
- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:`*mqttTopicFilter* - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 *mqttTopicFilter*를 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:`*mqttTopic*) 목록을 지정할 수 있습니다.

리소스 변수 내 임의의 위치에 와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 예를 들어 `mqtt:topic:my*`를 지정하여 해당 입력과 일치하는 리소스에 대한 액세스를 허용할 수 있습니다.

다음 리소스 변수가 지원됩니다.

- `mqtt:topic:${iot:Connection.Thing.ThingName}`

이렇게 하면 정책이 평가되는 AWS IoT Core 레지스트리의 사물 이름으로 확인됩니다. 디바이스가 인증할 때 제공하는 인증서를 AWS IoT Core 사용하여 연결을 확인하는 데 사용할 사물을 결정합니다. 이 정책 변수는 디바이스가 MQTT 또는 WebSocket 프로토콜을 통한 MQTT를 통해 연결하는 경우에만 사용할 수 있습니다.

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이 경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

performance

(선택 사항) 이 코어 디바이스의 성능 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

maxActiveAuthTokens

(선택 사항) 최대 활성 클라이언트 디바이스 권한 부여 토큰 수입니다. 이 수를 늘리면 다시 인증하지 않아도 더 많은 수의 클라이언트 디바이스가 단일 코어 디바이스에 연결될 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) AWS 클라우드에 보낼 최대 동시 요청 수입니다. 이 수를 늘려 많은 수의 클라이언트 디바이스를 연결하는 코어 디바이스의 인증 성능을 개선할 수 있습니다.

기본값: 1

certificateAuthority

(선택 사항) 코어 디바이스 중간 기관을 자신의 중간 인증 기관으로 바꾸는 인증 기관 구성 옵션입니다.

Note

사용자 지정 인증 기관(CA)으로 Greengrass 코어 디바이스를 구성하고 동일한 CA를 사용하여 클라이언트 디바이스 인증서를 발급하는 경우 Greengrass에서는 클라이언트 디바이스 MQTT 작업에 대한 권한 부여 정책 확인이 우회됩니다. 클라이언트 디바이스 인증 구성 요소에서는 사용하도록 구성된 CA에서 서명한 인증서를 사용하는 클라이언트가 완전히 신뢰됩니다.

사용자 지정 CA를 사용할 때 이 동작을 제한하려면 다른 CA 또는 중간 CA를 사용하여 클라이언트 디바이스를 생성하고 서명한 다음에 올바른 중간 CA를 가리키도록 `certificateUri` 및 `certificateChainUri` 필드를 조정합니다.

이 객체에는 다음 정보가 포함됩니다.

certificateUri

인증서의 위치입니다. 하드웨어 보안 모듈에 저장된 인증서를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

certificateChainUri

코어 디바이스 CA에 대한 인증서 체인의 위치입니다. 루트 CA로 돌아가는 전체 인증서 체인이어야 합니다. 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

privateKeyUri

코어 디바이스의 프라이빗 키 위치입니다. 하드웨어 보안 모듈에 저장된 인증서 프라이빗 키를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

security

(선택 사항) 이 코어 디바이스의 보안 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

clientDeviceTrustDurationMinutes

코어 디바이스로 재인증해야 하기 전에 클라이언트 디바이스의 인증 정보를 신뢰할 수 있는 기간(분)입니다. 기본값은 1입니다.

metrics

(선택 사항) 이 코어 디바이스의 지표 옵션입니다. 클라이언트 디바이스 인증에 오류가 있는 경우에만 오류 지표가 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

disableMetrics

`disableMetrics` 필드가 `true`로 설정된 경우 클라이언트 디바이스 인증에서는 지표가 수집되지 않습니다.

기본값: `false`

aggregatePeriodSeconds

클라이언트 디바이스에서 지표가 집계되어 원격 측정 에이전트로 보내지는 빈도가 결정되는 기간(초). 원격 측정 에이전트에서는 여전히 하루에 한 번씩 지표가 게시되기 때문에 지표가 게시되는 빈도는 변경되지 않습니다.

기본값: 3600

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 `ERRORRED`으로 변경됩니다.

기본값: 120

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

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

```

    }
  }
}

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}

```

Example예제: 구성 병합 업데이트(사물 이름 정책 사용)

다음 예제 구성에서는 클라이언트 디바이스에서 클라이언트 디바이스의 사물 이름으로 시작하고 문자열 topic으로 끝나는 주제에 게시할 수 있습니다.

```

{
  "deviceGroups": {

```

```

    "formatVersion": "2021-03-05",
    "definitions": {
      "myThing": {
        "selectionRule": "thingName: *",
        "policyName": "MyThingNamePolicy"
      }
    },
    "policies": {
      "MyThingNamePolicy": {
        "policyStatement": {
          "statementDescription": "mqtt publish",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:${iot:Connection.Thing.ThingName}/*/topic"
          ]
        }
      }
    }
  }
}

```

v2.4.5

deviceGroups

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스

에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하거나 끝나는 클라이언트 디바이스와 일치하도록 사물 이름의 시작과 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 thingName: MyTeam\\:ClientDevice1를 지정하여 이름이 MyTeam:ClientDevice1인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- `thingName` - 클라이언트 디바이스 사물의 이름입니다 AWS IoT .

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 `MyClientDevice1` 또는 `MyClientDevice2`인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 `MyClientDevice`로 끝나는 클라이언트 디바이스가 일치됩니다.

```
thingName: *MyClientDevice
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. `policies` 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` - 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* - 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.
- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:mqttTopicFilter` - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 `mqttTopicFilter`를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에 `mqtt:topicfilter:client/+/status` 리소스 구독을 허용하면 클라이언트 디바이스에서는 `client/+/status`가 구독될 수 있지만 `client/client1/status`은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:mqttTopic`) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 **"resources": "*"은 지정할 수 있지만, "resources": "mqtt:clientId:"은 지정할 수 없습니다.**

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이

경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

performance

(선택 사항) 이 코어 디바이스의 성능 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

maxActiveAuthTokens

(선택 사항) 최대 활성 클라이언트 디바이스 권한 부여 토큰 수입니다. 이 수를 늘리면 다시 인증하지 않아도 더 많은 수의 클라이언트 디바이스가 단일 코어 디바이스에 연결될 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) AWS 클라우드에 보낼 최대 동시 요청 수입니다. 이 수를 늘려 많은 수의 클라이언트 디바이스를 연결하는 코어 디바이스의 인증 성능을 개선할 수 있습니다.

기본값: 1

certificateAuthority

(선택 사항) 코어 디바이스 중간 기관을 자신의 중간 인증 기관으로 바꾸는 인증 기관 구성 옵션입니다.

Note

사용자 지정 인증 기관(CA)으로 Greengrass 코어 디바이스를 구성하고 동일한 CA를 사용하여 클라이언트 디바이스 인증서를 발급하는 경우 Greengrass에서는 클라이언트 디바이스 MQTT 작업에 대한 권한 부여 정책 확인이 우회됩니다. 클라이언트 디바이스 인

중 구성 요소에서는 사용하도록 구성된 CA에서 서명한 인증서를 사용하는 클라이언트가 완전히 신뢰됩니다.

사용자 지정 CA를 사용할 때 이 동작을 제한하려면 다른 CA 또는 중간 CA를 사용하여 클라이언트 디바이스를 생성하고 서명한 다음에 올바른 중간 CA를 가리키도록 `certificateUri` 및 `certificateChainUri` 필드를 조정합니다.

이 객체에는 다음 정보가 포함됩니다.

`certificateUri`

인증서의 위치입니다. 하드웨어 보안 모듈에 저장된 인증서를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

`certificateChainUri`

코어 디바이스 CA에 대한 인증서 체인의 위치입니다. 루트 CA로 돌아가는 전체 인증서 체인이어야 합니다. 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

`privateKeyUri`

코어 디바이스의 프라이빗 키 위치입니다. 하드웨어 보안 모듈에 저장된 인증서 프라이빗 키를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

`security`

(선택 사항) 이 코어 디바이스의 보안 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

`clientDeviceTrustDurationMinutes`

코어 디바이스로 재인증해야 하기 전에 클라이언트 디바이스의 인증 정보를 신뢰할 수 있는 기간(분)입니다. 기본값은 1입니다.

`metrics`

(선택 사항) 이 코어 디바이스의 지표 옵션입니다. 클라이언트 디바이스 인증에 오류가 있는 경우에만 오류 지표가 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`disableMetrics`

`disableMetrics` 필드가 `true`로 설정된 경우 클라이언트 디바이스 인증에서는 지표가 수집되지 않습니다.

기본값: false

aggregatePeriodSeconds

클라이언트 디바이스에서 지표가 집계되어 원격 측정 에이전트로 보내지는 빈도가 결정되는 기간(초). 원격 측정 에이전트에서는 여전히 하루에 한 번씩 지표가 게시되기 때문에 지표가 게시되는 빈도는 변경되지 않습니다.

기본값: 3600

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 ERRORED으로 변경됩니다.

기본값: 120

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

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

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```

{
  "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

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하는 클라이언트 디바이스와 일치하도록 사물 이름의 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 thingName: MyTeam\\:\\:ClientDevice1를 지정하여 이름이 MyTeam:ClientDevice1인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- thingName - 클라이언트 디바이스의 AWS IoT 사물 이름입니다.

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 MyClientDevice1 또는 MyClientDevice2인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. policies 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` - 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* - 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.
- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:`*mqttTopicFilter* - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 *mqttTopicFilter*를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에 `mqtt:topicfilter:client/+/status` 리소스 구독을 허용하면 클라이언트 디바이스에서는 `client/+/status`가 구독될 수 있지만 `client/client1/status`은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:mqttTopic`) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 `"resources": "*"` 는 지정할 수 있지만, `"resources": "mqtt:clientId:"`는 지정할 수 없습니다.

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이 경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

performance

(선택 사항) 이 코어 디바이스의 성능 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

maxActiveAuthTokens

(선택 사항) 최대 활성 클라이언트 디바이스 권한 부여 토큰 수입입니다. 이 수를 늘리면 다시 인증하지 않아도 더 많은 수의 클라이언트 디바이스가 단일 코어 디바이스에 연결될 수 있습니다.

기본값: 2500

cloudRequestQueueSize

(선택 사항)이 구성 요소가 AWS 클라우드 요청을 거부하기 전에 대기열에 추가할 최대 요청 수입입니다.

기본값: 100

maxConcurrentCloudRequests

(선택 사항) AWS 클라우드에 보낼 최대 동시 요청 수입입니다. 이 수를 늘려 많은 수의 클라이언트 디바이스를 연결하는 코어 디바이스의 인증 성능을 개선할 수 있습니다.

기본값: 1

certificateAuthority

(선택 사항) 코어 디바이스 중간 기관을 자신의 중간 인증 기관으로 바꾸는 인증 기관 구성 옵션입니다.

Note

사용자 지정 인증 기관(CA)으로 Greengrass 코어 디바이스를 구성하고 동일한 CA를 사용하여 클라이언트 디바이스 인증서를 발급하는 경우 Greengrass에서는 클라이언트 디바이스 MQTT 작업에 대한 권한 부여 정책 확인이 우회됩니다. 클라이언트 디바이스 인증 구성 요소에서는 사용하도록 구성된 CA에서 서명한 인증서를 사용하는 클라이언트가 완전히 신뢰됩니다.

사용자 지정 CA를 사용할 때 이 동작을 제한하려면 다른 CA 또는 중간 CA를 사용하여 클라이언트 디바이스를 생성하고 서명한 다음에 올바른 중간 CA를 가리키도록 `certificateUri` 및 `certificateChainUri` 필드를 조정합니다.

이 객체에는 다음 정보가 포함됩니다.

certificateUri

인증서의 위치입니다. 하드웨어 보안 모듈에 저장된 인증서를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

certificateChainUri

코어 디바이스 CA에 대한 인증서 체인의 위치입니다. 루트 CA로 돌아가는 전체 인증서 체인이어야 합니다. 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

privateKeyUri

코어 디바이스의 프라이빗 키 위치입니다. 하드웨어 보안 모듈에 저장된 인증서 프라이빗 키를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

security

(선택 사항) 이 코어 디바이스의 보안 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

clientDeviceTrustDurationMinutes

코어 디바이스로 재인증해야 하기 전에 클라이언트 디바이스의 인증 정보를 신뢰할 수 있는 기간(분)입니다. 기본값은 1입니다.

metrics

(선택 사항) 이 코어 디바이스의 지표 옵션입니다. 클라이언트 디바이스 인증에 오류가 있는 경우에만 오류 지표가 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

disableMetrics

`disableMetrics` 필드가 `true`로 설정된 경우 클라이언트 디바이스 인증에서는 지표가 수집되지 않습니다.

기본값: `false`

aggregatePeriodSeconds

클라이언트 디바이스에서 지표가 집계되어 원격 측정 에이전트로 보내지는 빈도가 결정되는 기간(초). 원격 측정 에이전트에서는 여전히 하루에 한 번씩 지표가 게시되기 때문에 지표가 게시되는 빈도는 변경되지 않습니다.

기본값: 3600

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 ERRORED으로 변경됩니다.

기본값: 120

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

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

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```

{
  "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

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하는 클라이언트 디바이스와 일치하도록 사물 이름의 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 `thingName: MyTeam\\:\\:ClientDevice1`를 지정하여 이름이 `MyTeam:ClientDevice1`인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- `thingName` - 클라이언트 디바이스의 AWS IoT 사물 이름입니다.

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 `MyClientDevice1` 또는 `MyClientDevice2`인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. `policies` 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` – 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* – 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.

- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:`*mqttTopicFilter* - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 *mqttTopicFilter*를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에 `mqtt:topicfilter:client/+/status` 리소스 구독을 허용하면 클라이언트 디바이스에서는 `client/+/status`가 구독될 수 있지만 `client/client1/status`은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:`*mqttTopic*) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 `"resources": "*"` 는 지정할 수 있지만, `"resources": "mqtt:clientId:"` 는 지정할 수 없습니다.

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이 경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

performance

(선택 사항) 이 코어 디바이스의 성능 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

maxActiveAuthTokens

(선택 사항) 최대 활성 클라이언트 디바이스 권한 부여 토큰 수입니다. 이 수를 늘리면 다시 인증하지 않아도 더 많은 수의 클라이언트 디바이스가 단일 코어 디바이스에 연결될 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) AWS 클라우드에 보낼 최대 동시 요청 수입니다. 이 수를 늘려 많은 수의 클라이언트 디바이스를 연결하는 코어 디바이스의 인증 성능을 개선할 수 있습니다.

기본값: 1

certificateAuthority

(선택 사항) 코어 디바이스 중간 기관을 자신의 중간 인증 기관으로 바꾸는 인증 기관 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

certificateUri

인증서의 위치입니다. 하드웨어 보안 모듈에 저장된 인증서를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

certificateChainUri

코어 디바이스 CA에 대한 인증서 체인의 위치입니다. 루트 CA로 돌아가는 전체 인증서 체인이어야 합니다. 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

privateKeyUri

코어 디바이스의 프라이빗 키 위치입니다. 하드웨어 보안 모듈에 저장된 인증서 프라이빗 키를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

security

(선택 사항) 이 코어 디바이스의 보안 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

clientDeviceTrustDurationMinutes

코어 디바이스로 재인증해야 하기 전에 클라이언트 디바이스의 인증 정보를 신뢰할 수 있는 기간(분)입니다. 기본값은 1입니다.

metrics

(선택 사항) 이 코어 디바이스의 지표 옵션입니다. 클라이언트 디바이스 인증에 오류가 있는 경우에만 오류 지표가 표시됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

disableMetrics

disableMetrics 필드가 true로 설정된 경우 클라이언트 디바이스 인증에서는 지표가 수집되지 않습니다.

기본값: false

aggregatePeriodSeconds

클라이언트 디바이스에서 지표가 집계되어 원격 측정 에이전트로 보내지는 빈도가 결정되는 기간(초). 원격 측정 에이전트에서는 여전히 하루에 한 번씩 지표가 게시되기 때문에 지표가 게시되는 빈도는 변경되지 않습니다.

기본값: 3600

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

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

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```

{
  "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

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하는 클라이언트 디바이스와 일치하도록 사물 이름의 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 `thingName: MyTeam\\:\\:ClientDevice1`를 지정하여 이름이 `MyTeam:ClientDevice1`인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- `thingName` - 클라이언트 디바이스의 AWS IoT 사물 이름입니다.

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 `MyClientDevice1` 또는 `MyClientDevice2`인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. `policies` 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` – 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* – 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.

- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:`*mqttTopicFilter* - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 *mqttTopicFilter*를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에 `mqtt:topicfilter:client/+/status` 리소스 구독을 허용하면 클라이언트 디바이스에서는 `client/+/status`가 구독될 수 있지만 `client/client1/status`은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:`*mqttTopic*) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 **"resources": "*"는 지정할 수 있지만, "resources": "mqtt:clientId:"는 지정할 수 없습니다.**

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이 경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

performance

(선택 사항) 이 코어 디바이스의 성능 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

maxActiveAuthTokens

(선택 사항) 최대 활성 클라이언트 디바이스 권한 부여 토큰 수입니다. 이 수를 늘리면 다시 인증하지 않아도 더 많은 수의 클라이언트 디바이스가 단일 코어 디바이스에 연결될 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) AWS 클라우드에 보낼 최대 동시 요청 수입니다. 이 수를 늘려 많은 수의 클라이언트 디바이스를 연결하는 코어 디바이스의 인증 성능을 개선할 수 있습니다.

기본값: 1

certificateAuthority

(선택 사항) 코어 디바이스 중간 기관을 자신의 중간 인증 기관으로 바꾸는 인증 기관 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

certificateUri

인증서의 위치입니다. 하드웨어 보안 모듈에 저장된 인증서를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

certificateChainUri

코어 디바이스 CA에 대한 인증서 체인의 위치입니다. 루트 CA로 돌아가는 전체 인증서 체인이어야 합니다. 하드웨어 보안 모듈에 저장된 인증서 체인을 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

privateKeyUri

코어 디바이스의 프라이빗 키 위치입니다. 하드웨어 보안 모듈에 저장된 인증서 프라이빗 키를 가리키는 파일 시스템 URI 또는 URI일 수 있습니다.

security

(선택 사항) 이 코어 디바이스의 보안 구성 옵션입니다. 이 객체에는 다음 정보가 포함됩니다.

clientDeviceTrustDurationMinutes

코어 디바이스로 재인증해야 하기 전에 클라이언트 디바이스의 인증 정보를 신뢰할 수 있는 기간(분)입니다. 기본값은 1입니다.

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```
{
  "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"
        ]
      }
    }
  }
}

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```
{
```

```

"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

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스

에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하는 클라이언트 디바이스와 일치하도록 사물 이름의 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 thingName: MyTeam\\\\\\\\:ClientDevice1를 지정하여 이름이 MyTeam:ClientDevice1인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- thingName - 클라이언트 디바이스의 AWS IoT 사물 이름입니다.

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 MyClientDevice1 또는 MyClientDevice2인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. policies 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` - 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* - 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.
- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:`*mqttTopicFilter* - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 *mqttTopicFilter*를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에

mqtt:topicfilter:client/+/status 리소스 구독을 허용하면 클라이언트 디바이스에서는 client/+/status가 구독될 수 있지만 client/client1/status은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 mqtt:publish 작업이 지정되는 정책에서 MQTT 주제 리소스(mqtt:topic:*mqttTopic*) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 **"resources": "*"는 지정할 수 있지만, "resources": "mqtt:clientId:"는 지정할 수 없습니다.**

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이 경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

performance

(선택 사항) 이 코어 디바이스의 성능 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

maxActiveAuthTokens

(선택 사항) 최대 활성 클라이언트 디바이스 권한 부여 토큰 수입니다. 이 수를 늘리면 다시 인증하지 않아도 더 많은 수의 클라이언트 디바이스가 단일 코어 디바이스에 연결될 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) AWS 클라우드에 보낼 최대 동시 요청 수입니다. 이 수를 늘려 많은 수의 클라이언트 디바이스를 연결하는 코어 디바이스의 인증 성능을 개선할 수 있습니다.

기본값: 1

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

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

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```

{
  "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

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 **groupNameKey**를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하는 클라이언트 디바이스와 일치하도록 사물 이름의 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 thingName: MyTeam\\:\\:ClientDevice1를 지정하여 이름이 MyTeam:ClientDevice1인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- thingName - 클라이언트 디바이스의 AWS IoT 사물 이름입니다.

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 MyClientDevice1 또는 MyClientDevice2인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 MyClientDevice로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. `policies` 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` - 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* - 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.
- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:`*mqttTopicFilter* - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 *mqttTopicFilter*를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에 `mqtt:topicfilter:client/+/status` 리소스 구독을 허용하면 클라이언트 디바이스에서는 `client/+/status`가 구독될 수 있지만 `client/client1/status`은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:mqttTopic`) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 **"resources": "*"는 지정할 수 있지만, "resources": "mqtt:clientId:"는 지정할 수 없습니다.**

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

certificates

(선택 사항) 이 코어 디바이스의 인증서 구성 옵션입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

serverCertificateValiditySeconds

(선택 사항) 로컬 MQTT 서버 인증서가 만료되는 시간(초)입니다. 이 옵션을 구성하여 클라이언트 디바이스와 코어 디바이스의 연결 해제 및 재연결 빈도를 사용자 지정할 수 있습니다.

이 구성 요소에서는 로컬 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. MQTT 브로커(예: [Moquette MQTT 브로커 구성 요소](#))에서는 새 인증서가 생성되어 다시 시작됩니다. 이 경우에는 이 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

기본값: 604800(7일)

최솟값: 172800(2일)

최댓값: 864000(10일)

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

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

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```
{
  "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

디바이스 그룹은 코어 디바이스와 연결하고 통신할 권한이 있는 클라이언트 디바이스 그룹입니다. 선택 규칙을 사용하여 클라이언트 디바이스 그룹을 식별하고 디바이스 그룹마다 권한을 지정하는 클라이언트 디바이스 권한 부여 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

formatVersion

이 구성 객체의 형식 버전입니다.

다음 옵션 중 하나를 선택합니다.

- 2021-03-05

definitions

이 코어 디바이스의 디바이스 그룹. 각 정의에서는 클라이언트 디바이스가 그룹의 멤버인지 평가할 선택 규칙이 지정됩니다. 각 정의에서는 선택 규칙과 일치하는 클라이언트 디바이스에 적용할 권한 정책도 지정됩니다. 클라이언트 디바이스가 여러 디바이스 그룹의 멤버인 경우 디바이스의 권한은 각 그룹의 권한 정책으로 구성됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

groupNameKey

이 디바이스 그룹의 이름입니다. 이 디바이스 그룹을 식별하는 데 도움이 되는 이름으로 *groupNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

selectionRule

이 디바이스 그룹의 멤버인 클라이언트 디바이스가 지정되는 쿼리입니다. 클라이언트 디바이스가 연결되면 코어 디바이스에서는 이 선택 규칙을 평가하여 클라이언트 디바이스가 이 디바이스 그룹의 멤버인지 결정됩니다. 클라이언트 디바이스가 멤버인 경우 코어 디바이스에서는 이 디바이스 그룹의 정책을 사용하여 클라이언트 디바이스의 작업에 권한이 부여됩니다.

각 선택 규칙은 클라이언트 디바이스와 일치할 수 있는 단일 표현식 쿼리인 하나 이상의 선택 규칙 절로 구성됩니다. 선택 규칙은 AWS IoT 플릿 인덱싱과 동일한 쿼리 구문을 사용합니다. 선택 규칙 구문에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 플릿 인덱싱 쿼리 구문](#)을 참조하세요.

여러 클라이언트 디바이스가 하나의 선택 규칙 절과 일치되도록 와일드카드(*)를 사용합니다. 지정하는 문자열로 이름이 시작하는 클라이언트 디바이스와 일치하도록 사물 이름의 끝에 이 와일드카드를 사용할 수 있습니다. 모든 클라이언트 디바이스가 일치되도록 이 와일드카드를 사용할 수도 있습니다.

Note

콜론 문자(:)가 있는 값을 선택하려면 백슬래시 문자(\\)로 콜론을 이스케이프합니다. JSON과 같은 형식에서는 백슬래시 문자를 이스케이프

해야 하므로 콜론 문자 앞에 백슬래시 문자 2개를 입력합니다. 예를 들어 `thingName: MyTeam\\:\\:ClientDevice1`를 지정하여 이름이 `MyTeam:ClientDevice1`인 사물을 선택합니다.

다음 선택기를 지정할 수 있습니다.

- `thingName` - 클라이언트 디바이스의 AWS IoT 사물 이름입니다.

Example선택 규칙 예제

다음 선택 규칙에서는 이름이 `MyClientDevice1` 또는 `MyClientDevice2`인 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example선택 규칙 예제(와일드카드 사용)

다음 선택 규칙에서는 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스가 일치됩니다.

```
thingName: MyClientDevice*
```

Example선택 규칙 예제(모든 디바이스 일치)

다음 선택 규칙에서는 모든 클라이언트 디바이스가 일치됩니다.

```
thingName: *
```

policyName

이 디바이스 그룹의 클라이언트 디바이스에 적용되는 권한 정책. `policies` 객체에서 정의하는 정책의 이름이 지정됩니다.

policies

코어 디바이스에 연결되는 클라이언트 디바이스에 대한 클라이언트 디바이스 권한 부여 정책입니다. 각 권한 부여 정책을 통해 작업 세트 및 클라이언트 디바이스에서 해당 작업을 수행할 수 있는 리소스가 지정됩니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

policyNameKey

이 권한 부여 정책의 이름. 이 권한 부여 정책을 식별하는 데 도움이 되는 이름으로 *policyNameKey*를 바꿉니다. 이 정책 이름을 사용하여 디바이스 그룹에 적용되는 정책을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

statementNameKey

이 정책 문의 이름입니다. 이 정책 문을 식별하는 데 도움이 되는 이름으로 *statementNameKey*를 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

operations

이 정책의 리소스에 허용되는 작업 목록입니다.

다음과 같은 작업을 임의로 포함할 수 있습니다.

- `mqtt:connect` - 코어 디바이스에 연결하는 권한을 부여합니다. 코어 디바이스에 연결하려면 클라이언트 디바이스에 이 권한이 있어야 합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:clientId:`*deviceClientId* - 클라이언트 디바이스에서 코어 디바이스의 MQTT 브로커에 연결하는 데 사용되는 클라이언트 ID에 따라 액세스를 제한합니다. 사용할 클라이언트 ID로 *deviceClientId*를 바꿉니다.
- `mqtt:publish` - 주제에 MQTT 메시지를 게시하는 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topic:`*mqttTopic* - 클라이언트 디바이스에서 메시지가 게시되는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제로 *mqttTopic*을 바꿉니다.

이 리소스에서는 MQTT 주제 와일드카드가 지원되지 않습니다.

- `mqtt:subscribe` - 메시지를 수신할 MQTT 주제 필터를 구독할 권한을 부여합니다.

이 작업에서는 다음과 같은 리소스가 지원됩니다.

- `mqtt:topicfilter:mqttTopicFilter` - 클라이언트 디바이스에서 메시지를 구독할 수 있는 MQTT 주제에 따라 액세스를 제한합니다. 사용할 주제 필터로 `mqttTopicFilter`를 바꿉니다.

이 리소스에서는 + 및 # MQTT 주제 와일드카드가 지원됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

클라이언트 디바이스에서는 사용자가 허용하는 정확한 주제 필터가 구독될 수 있습니다. 예를 들어 클라이언트 디바이스에 `mqtt:topicfilter:client/+/status` 리소스 구독을 허용하면 클라이언트 디바이스에서는 `client/+/status`가 구독될 수 있지만 `client/client1/status`은 구독될 수 없습니다.

와일드카드(*)를 지정하여 모든 작업에 대한 액세스를 허용할 수 있습니다.

resources

이 정책의 작업에 허용되는 리소스 목록입니다. 이 정책의 작업에 해당하는 리소스를 지정합니다. 예를 들어 `mqtt:publish` 작업이 지정되는 정책에서 MQTT 주제 리소스(`mqtt:topic:mqttTopic`) 목록을 지정할 수 있습니다.

와일드카드(*)를 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다. 부분 리소스 식별자와 일치하도록 와일드카드(*)를 사용할 수 있습니다. 예를 들어 **"resources": "*"는 지정할 수 있지만, "resources": "mqtt:clientId:"는 지정할 수 없습니다.**

statementDescription

(선택 사항) 이 정책 문에 대한 설명입니다.

Example예제: 구성 병합 업데이트(제한적 정책 사용)

다음 예제 구성에서는 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```
{
  "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"
        ]
      }
    }
  }
}

```

Example예제: 구성 병합 업데이트(허용적 정책 사용)

다음 예제 구성에서는 모든 클라이언트 디바이스의 연결 및 모든 주제 게시/구독이 허용되도록 지정됩니다.

```
{
```

```
"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 nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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


이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.5.5	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.5.4	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.5.3	버그 수정 및 개선 사항 클라이언트 인증서가 최신 상태가 아니어서 클라이언트 디바이스가 코어 디바이스에 연결하지 못하는 문제를 수정합니다.
2.5.2	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.5.1	버그 수정 및 개선 사항 FIPS 엔드포인트가 지원됩니다.
2.5.0	새로운 기능 정책 리소스에 대한 <code>\${iot:Connection.Thing.ThingName}</code> 변수 대체가 허용됩니다.

버전	변경 사항
	와일드카드가 포함된 정책 리소스(예: <code>mqtt:topic:my*</code>)가 허용됩니다.
2.4.5	새로운 기능 <code>selectionRule</code> 파라미터를 사용하여 사물 이름을 선택할 수 있도록 와일드카드 접두사에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 경우에 따라 인증서가 새 연결 정보로 업데이트되지 않는 문제를 해결합니다.
2.4.4	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.4.3	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.4.2	새로운 기능 새로운 <code>startupTimeoutSeconds</code> 구성 옵션이 추가됩니다.
2.4.1	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.4.0	새로운 기능 - 클라이언트 디바이스 인증에서 원격 측정 에이전트를 통해 게시되는 운영 지표 내보내기에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - 클라이언트 디바이스 인증에서 클라이언트 디바이스의 ID가 확인되는 데 10초 넘게 걸리는 문제를 해결합니다. - 사소한 추가 수정 및 개선 사항.
2.3.2	버그 수정 및 개선 사항 오프라인 상태에서 다시 시작할 때 구성 요소에서 인증서 주체가 올바르게 생성되도록 호스트 이름 정보 캐싱에 대한 지원을 추가합니다.
2.3.1	버그 수정 및 개선 사항 메모리 누수가 수정됩니다.

버전	변경 사항
2.3.0	 Warning 더는 이 버전을 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 새로운 기능 - 코어 디바이스가 인터넷에 연결되어 있지 않을 때 코어 디바이스에 계속 연결할 수 있도록 클라이언트 디바이스의 오프라인 인증에 대한 지원을 추가합니다. - 코어 디바이스에서 루트 인증서로 사용하여 MQTT 브로커 인증서를 생성하는 고객 제공 인증 기관에 대한 지원을 추가합니다.
2.2.3	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.2.2	버그 수정 및 개선 사항 로컬 MQTT 서버 인증서가 특정 시나리오에서 의도보다 자주 교체되는 문제를 해결합니다.
2.2.1	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.2.0	새로운 기능 - 사용자 지정 구성 요소에 대한 지원이 추가되어 IPC(프로세스 간 통신) 작업을 직접적으로 호출하여 클라이언트 디바이스를 인증하고 권한을 부여합니다. 예를 들어 사용자 지정 MQTT 브로커 구성 요소에서 이러한 작업을 사용할 수 있습니다. 자세한 내용은 IPC: 클라이언트 디바이스 인증 및 권한 부여를 참조하세요. <code>maxActiveAuthTokens</code>, <code>cloudQueueSize</code> 및 <code>threadPoolSize</code> 옵션이 추가되어 이를 구성하면 이 구성 요소의 성능을 조정할 수 있습니다.

버전	변경 사항
2.1.0	새로운 기능 - MQTT 브로커 서버 인증서가 만료되는 시기를 사용자 지정하도록 구성할 수 있는 <code>serverCertificateValiditySeconds</code> 옵션이 추가됩니다. 2~10일 후에 만료되도록 서버 인증서를 구성할 수 있습니다. 버그 수정 및 개선 사항 - 이 구성 요소에서 구성 재설정 업데이트가 처리되는 방식과 관련된 문제가 해결됩니다. - 로컬 MQTT 서버 인증서가 특정 시나리오에서 의도보다 자주 교체되는 문제를 해결합니다. 이 수정 사항을 적용하려면 Moquette MQTT 브로커 구성 요소의 v2.1.0 이상도 사용해야 합니다. - 인증서를 교체할 때 이 구성 요소의 로그에 기록되는 메시지가 개선됩니다. - Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	버그 수정 및 개선 사항 - 이제 코어 디바이스의 프라이빗 키를 교체하면 자격 증명이 새로 고쳐집니다. - 로그 메시지가 더 명확하도록 업데이트됩니다.
2.0.2	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.1	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.0	초기 버전입니다.

CloudWatch 지표

Amazon CloudWatch 지표 구성 요소(`aws.greengrass.Cloudwatch`)에서는 Greengrass 코어 디바이스의 사용자 지정 지표를 Amazon CloudWatch에 게시합니다. 구성 요소를 사용하면 Greengrass 코어 디바이스의 환경 모니터링 및 분석에 사용할 수 있는 CloudWatch 지표를 구성 요소에서 게시할 수

있습니다. 자세한 설명은 Amazon CloudWatch 사용자 가이드에서 [Amazon CloudWatch 지표 사용하기](#)를 참조하세요.

이 구성 요소로 CloudWatch 지표를 게시하려면 이 구성 요소에서 구독하는 주제에 메시지를 게시합니다. 기본적으로 이 구성 요소에서는 cloudwatch/metric/put [로컬 게시/구독](#) 주제를 구독합니다. 이 구성 요소를 배포할 때 AWS IoT Core MQTT 주제를 포함한 다른 주제를 지정할 수 있습니다.

이 구성 요소에서는 동일한 네임스페이스에 있는 지표를 배치 처리하며 정기적으로 CloudWatch에 게시합니다.

Note

이 구성 요소는 AWS IoT Greengrass V1의 CloudWatch 지표 커넥터와 유사한 기능을 제공합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [CloudWatch 지표 커넥터](#)를 참조하세요.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [입력 데이터](#)
- [출력 데이터](#)
- [라이선스](#)
- [로컬 로그 파일](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 3.2.x
- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

각 구성 요소 버전의 변경 사항에 대한 내용은 [changelog](#)를 참조하세요.

Type

v3.x

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

v2.x

이 구성 요소는 Lambda 구성 요소(`aws.greengrass.lambda`)입니다. [Greengrass nucleus](#)에서는 [Lambda 런처 구성 요소](#)를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

3.x

- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서는 `cloudwatch:PutMetricData` 작업이 허용되어야 합니다.

JSON

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

자세한 내용은 Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch 권한 참조](#)를 참조하세요.

2.x

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.
- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서는 `cloudwatch:PutMetricData` 작업이 허용되어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
```

```

        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}

```

자세한 내용은 Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch 권한 참조](#)를 참조하세요.

- 이 구성 요소의 출력 데이터를 수신하려면 이 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)(aws.greengrass.LegacySubscriptionRouter)에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성에서는 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"
    }
  }
}

```

- region*을 사용하는 AWS 리전 로 바꿉니다.

- **##**을 이 구성 요소에서 실행되는 Lambda 함수의 버전으로 바꿉니다. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소의 버전에 대한 레시피를 확인해야 합니다. [AWS IoT Greengrass 콘솔](#)에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 페어를 찾습니다. 이 키-값 페어에는 Lambda 함수의 이름과 버전이 포함되어 있습니다.

Important

이 구성 요소를 배포할 때마다 레거시 구독 라우터의 Lambda 함수 버전을 업데이트해야 합니다. 이를 통해 배포하는 구성 요소 버전에 맞는 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
monitoring. <i>region</i> .amazonaws.com	443	예	CloudWatch 지표를 업로드합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

3.2.0

다음 표에는 이 구성 요소의 버전 3.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	소프트
토큰 교환 서비스	>=0.0.0	하드

3.0.0 - 3.1.0

다음 표에는 이 구성 요소의 버전 3.0.0~3.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.1.4 - 2.1.9

다음 표에는 이 구성 요소의 버전 2.1.4~2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.4 - 2.1.8

다음 표에는 이 구성 요소의 버전 2.1.4 및 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.2 - 2.1.3

다음 표에는 이 구성 요소의 버전 2.1.2 및 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	하드
Lambda 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 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

v3.x

PublishInterval

(선택 사항) 주어진 네임스페이스에 대해 일괄 처리된 지표를 게시하기 전에 대기해야 하는 최대 시간(초)입니다. 지표를 받자마자 배치 처리 없이 게시하도록 구성 요소를 구성하려면 0을 지정합니다.

이 구성 요소에서는 동일한 네임스페이스의 지표 20개를 받은 후 또는 사용자가 지정하는 간격 후 CloudWatch에 게시합니다.

Note

구성 요소에서는 이벤트가 게시되는 순서를 지정되지 않습니다.

이 값은 최대 900초가 될 수 있습니다.

기본값: 10초

MaxMetricsToRetain

(선택 사항) 구성 요소에서 새 지표로 바꾸기 전에 메모리에 저장되는 모든 네임스페이스의 최대 지표 수입니다.

이 제한은 인터넷에 대한 연결이 코어 디바이스에 없어서 구성 요소에서 나중에 게시할 지표를 버퍼링할 때 적용됩니다. 버퍼가 가득 차면 구성 요소에서는 가장 오래된 지표를 최신 지표로 바꿉니다. 주어진 네임스페이스의 지표에서는 동일한 네임스페이스의 지표만 바꿉니다.

 Note

구성 요소에 대한 호스트 프로세스가 중단되면 구성 요소에서는 지표가 저장되지 않습니다. 예를 들어 배포 중이거나 코어 디바이스가 다시 시작할 때 이렇게 될 수 있습니다.

이 값의 지표는 2,000개 이상이어야 합니다.

기본값: 지표 5,000개

InputTopic

(선택 사항) 구성 요소에서 메시지 수신에 구독되는 주제입니다. PubSubToIoTCore에 true를 지정하는 경우 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

기본값: cloudwatch/metric/put

OutputTopic

(선택 사항) 구성 요소에서 상태 응답이 게시되는 주제입니다.

기본값: cloudwatch/metric/put/status

PubSubToIoTCore

(선택 사항) AWS IoT Core MQTT 주제에 게시하고 구독할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 true 및 false입니다.

기본값: false

LogLevel

(선택 사항) 구성 요소의 로깅 수준. 여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

기본값: INFO

UseInstaller

(선택 사항) 이 구성 요소의 설치 프로그램 스크립트를 사용하여 이 구성 요소의 SDK 종속성을 설치할지 여부를 정의하는 부울 값입니다.

사용자 지정 스크립트를 사용하여 종속성을 설치하려는 경우 또는 사전 구축된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 `false`로 설정합니다. 이 구성 요소를 사용하려면 다음 라이브러리를 설치하고(종속성 포함), 기본 Greengrass 시스템 사용자가 사용할 수 있도록 해야 합니다.

- [Python용AWS IoT Device SDK v2](#)
- [AWS SDK for Python \(Boto3\)](#)

기본값: true

PublishRegion

(선택 사항) CloudWatch 지표를 게시 AWS 리전 할 입니다. 이 값으로 코어 디바이스의 기본 리전이 재정의됩니다. 이 파라미터는 리전 간 지표에만 필요합니다.

accessControl

(선택 사항) 주어진 주제를 게시하고 구독하도록 구성 요소에 허용하는 [권한 부여 정책](#)이 있는 객체입니다. InputTopic 및 OutputTopic에 대한 사용자 지정 값을 지정하는 경우 이 객체의 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"
      ]
    }
  }
}

```

Example예: 구성 병합 업데이트

```

{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}

```

v2.x

Note

이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함되어 있습니다. 디바이스에서 다음 파라미터만 사용하여 이 구성 요소를 구성하는 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) 주어진 네임스페이스에 대해 일괄 처리된 지표를 게시하기 전에 대기해야 하는 최대 시간(초)입니다. 지표를 받자마자 배치 처리 없이 게시하도록 구성 요소를 구성하려면 0을 지정합니다.

이 구성 요소에서는 동일한 네임스페이스의 지표 20개를 받은 후 또는 사용자가 지정하는 간격 후 CloudWatch에 게시합니다.

Note

구성 요소에서는 이벤트가 게시되는 순서가 보장되지 않습니다.

이 값은 최대 900초가 될 수 있습니다.

기본값: 10초

MAX_METRICS_TO_RETAIN

(선택 사항) 구성 요소에서 새 지표로 바꾸기 전에 메모리에 저장되는 모든 네임스페이스의 최대 지표 수입니다.

이 제한은 인터넷에 대한 연결이 코어 디바이스에 없어서 구성 요소에서 나중에 게시할 지표를 버퍼링할 때 적용됩니다. 버퍼가 가득 차면 구성 요소에서는 가장 오래된 지표를 최신 지표로 바꿉니다. 주어진 네임스페이스의 지표에서는 동일한 네임스페이스의 지표만 바꿉니다.

Note

구성 요소에 대한 호스트 프로세스가 중단되면 구성 요소에서는 지표가 저장되지 않습니다. 예를 들어 배포 중이거나 코어 디바이스가 다시 시작할 때 이렇게 될 수 있습니다.

이 값의 지표는 2,000개 이상이어야 합니다.

기본값: 지표 5,000개

`PUBLISH_REGION`

(선택 사항) CloudWatch 지표를 게시 AWS 리전 할 입니다. 이 값으로 코어 디바이스의 기본 리전이 재정의됩니다. 이 파라미터는 리전 간 지표에만 필요합니다.

`containerMode`

(선택 사항) 이 구성 요소에 대한 컨테이너화 모드입니다. 다음 옵션 중 하나를 선택합니다.

- `NoContainer` – 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.
- `GreengrassContainer` - 구성 요소는 AWS IoT Greengrass 컨테이너 내의 격리된 런타임 환경에서 실행됩니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소에 대한 컨테이너 파라미터를 포함하는 객체입니다.

`containerMode`에 `GreengrassContainer`를 지정하면 구성 요소에서 이러한 파라미터를 사용합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당되는 메모리 양(킬로바이트)입니다.

기본값은 64MB(65,535KB)입니다.

`pubsubTopics`

(선택 사항) 구성 요소에서 메시지를 수신하려고 구독하는 주제를 포함하는 객체입니다. 각 주제와 구성 요소가 AWS IoT Core 또는 로컬 게시/구독 주제의 MQTT 주제를 구독하는지 여부를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

type

(선택 사항) 이 구성 요소에서 메시지 구독에 사용하는 게시/구독 메시징의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.
- IOT_CORE - AWS IoT Core MQTT 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본값: PUB_SUB

topic

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. type에 IotCore를 지정하면 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

Example예: 구성 병합 업데이트(컨테이너 모드)

```
{
  "containerMode": "GreengrassContainer"
}
```

Example예: 구성 병합 업데이트(컨테이너 없음 모드)

```
{
  "containerMode": "NoContainer"
}
```

입력 데이터

이 구성 요소에서는 다음 주제에 대한 지표가 수락되고 지표가 CloudWatch에 게시됩니다. 기본적으로 이 구성 요소는 로컬 게시/구독 메시지가 구독됩니다. 사용자 지정 구성 요소에서 이 구성 요소로 메시지를 게시하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

구성 요소 버전 v3.0.0부터 PubSubToIoTCore 구성 파라미터를 true로 설정하여 MQTT 주제를 구독하도록 이 구성 요소를 선택적으로 구성할 수 있습니다. 사용자 지정 구성 요소의 MQTT 주제로 메시지를 게시하는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본 주제: cloudwatch/metric/put

메시지는 다음 속성을 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

request

이 메시지의 지표입니다.

요청 객체에는 CloudWatch에 게시할 지표 데이터가 포함되어 있습니다. 지표 값에서는 [PutMetricData](#) 작업의 사양이 충족되어야 합니다.

유형: 다음 정보를 포함하는 object입니다.

namespace

이 요청의 지표 데이터에 대한 사용자 정의 네임스페이스입니다. CloudWatch는 네임스페이스를 지표 데이터 포인트의 컨테이너로 사용합니다.

Note

예약 문자어 AWS/로 시작하는 네임스페이스는 지정할 수 없습니다.

유형: string

유효한 패턴: [^:].*

metricData

지표에 대한 데이터.

유형: 다음 정보를 포함하는 object입니다.

metricName

지표의 이름.

유형: string

value

지표에 대한 값.

 Note

CloudWatch에서는 너무 작거나 큰 값을 거부합니다. 값은 $8.515920e-109 \sim 1.174271e+108$ (Base 10) 또는 $2e-360 \sim 2e360$ (Base 2)로 되어야 합니다. CloudWatch에서는 특수 값이 지원되지 않습니다(예: NaN, +Infinity 및 -Infinity).

유형: double

dimensions

(선택 사항) 지표의 차원입니다. 차원에서는 지표와 해당 데이터에 대한 추가 정보가 제공됩니다. 지표는 최대 10개 차원을 정의할 수 있습니다.

이 구성 요소에는 이름이 `coreName`인 차원이 자동으로 포함되어 있으며, 여기에서 값은 코어 디바이스의 이름입니다.

유형: 각각 다음 정보가 있는 객체의 array입니다.

name

(선택 사항) 차원 이름입니다.

유형: string

value

(선택 사항) 차원 값입니다.

유형: string

timestamp

(선택 사항) Unix 에포크 시간으로 초 단위로 표시되는 메트릭 데이터가 수신된 시각입니다.

기본값은 구성 요소에서 메시지를 받는 시각입니다.

유형: double

Note

이 구성 요소의 버전 2.0.3~2.0.7를 사용한다면 단일 소스의 여러 지표를 보낼 때 지표마다 타임스탬프를 개별적으로 검색하는 것이 좋습니다. 변수를 사용하여 타임스탬프를 저장하지 마세요.

unit

(선택 사항) 지표의 단위입니다.

유형: string

유효한 값: Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/Second, Terabits/Second, Count/Second, None

기본값은 None입니다.

Note

CloudWatch PutMetricData API에 적용되는 모든 할당량은 이 구성 요소로 게시하는 지표에 적용됩니다. 다음 할당량은 아주 중요합니다.

- API 페이로드에 대한 40KB 제한
- API 요청당 20개 지표
- PutMetricData API에 대한 150개의 초당 트랜잭션(TPS)

자세한 내용은 CloudWatch 사용 설명서의 [CloudWatch Service Quotas](#)를 참조하세요.

Example입력 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 다음 로컬 게시/구독 주제에 대한 출력 데이터로 응답을 게시합니다. 사용자 지정 구성 요소의 이 주제에 대한 메시지를 구독하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

PubSubToIoTCore 구성 파라미터를 true로 설정하여 MQTT 주제에 게시하도록 이 구성 요소를 선택적으로 구성할 수 있습니다. 사용자 지정 구성 요소의 MQTT 주제에 대한 메시지를 구독하는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

Note

구성 요소 버전 2.0.x에서는 기본적으로 응답이 MQTT 주제에 대한 출력 데이터로 게시됩니다. [레거시 구독 라우터 구성 요소](#)의 구성에서 주제를 subject로 지정해야 합니다.

기본 주제: cloudwatch/metric/put/status

Example출력 예: 성공

응답에는 지표 데이터의 네임스페이스 및 CloudWatch 응답의 RequestId 필드가 포함되어 있습니다.

```

{
  "response": {

```



```

    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}

```

Example출력 예: 실패

```

{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}

```

Note

구성 요소에서는 연결 오류와 같은 재시도할 수 있는 오류가 감지되면 다음 배치에서 게시를 다시 시도합니다.

라이선스

이 구성 요소에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- [AWS SDK for Python \(Boto3\)](#)/Apache 라이선스 2.0
- [botocore](#)/Apache 라이선스 2.0
- [dateutil](#)/PSF 라이선스
- [docutils](#)/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- [jmespath](#)/MIT 라이선스
- [s3transfer](#)/Apache 라이선스 2.0
- [urllib3](#)/MIT 라이선스

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. */greengrass/v2* 또는 *C:\greengrass\v2*를 AWS IoT Greengrass 루트 폴더의 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 사항
3.2.0	새로운 기능 Greengrass nucleus lite에 대한 레시피 지원 추가

버전	변경 사항
3.1.0	버그 수정 및 개선 사항 HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정 섹션을 참조하세요.
3.0.0	CloudWatch 지표 구성 요소의 이 버전에는 버전 2.x와 다른 구성 파라미터가 필요합니다. 버전 2.x에 기본값이 아닌 구성을 사용하고 v2.x에서 v3.x로 업그레이드하려면 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 CloudWatch 지표 구성 요소 구성 섹션을 참조하세요. 새로운 기능 - Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. - 구성 요소 유형을 Lambda 구성 요소에서 일반 구성 요소로 변경합니다. 이 구성 요소에서는 이제 구독 생성이 레거시 구독 라우터 구성 요소에 종속되지 않습니다. - 구성 요소가 메시지를 수신하기 위해 구독하는 주제를 지정하는 새 InputTopic 구성 파라미터를 추가합니다. - 구성 요소에서 상태 응답이 게시되는 주제를 지정하는 새 OutputTopic 구성 파라미터를 추가합니다. - AWS IoT Core MQTT 주제를 게시하고 구독할지 여부를 지정하는 새 PubSubToIoTCore 구성 파라미터를 추가합니다. - 구성 요소 종속성이 설치되는 설치 스크립트를 선택적으로 비활성화할 수 있는 새 UseInstaller 구성 파라미터를 추가합니다. 버그 수정 및 개선 사항 입력 데이터의 중복 타임스탬프에 대한 지원을 추가합니다.

v2.x

버전	변경 사항
2.1.8	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.3	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정을(를) 참조하세요.
2.0.8	버그 수정 및 개선 사항 • 입력 데이터의 중복 타임스탬프에 대한 지원을 추가합니다. • Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

다음 사항도 참조하세요.

- Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch 지표 사용](#)
- Amazon CloudWatch API 참조의 [PutMetricData](#)

AWS IoT Device Defender

AWS IoT Device Defender 구성 요소(`aws.greengrass.DeviceDefender`)는 Greengrass 코어 디바이스의 상태 변경에 대해 관리자에게 알립니다. 그러면 손상된 장치를 나타낼 수 있는 비정상적인 동작을 식별할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT Device Defender](#) 섹션을 참조하세요.

이 구성 요소에서는 코어 디바이스의 시스템 지표를 읽습니다. 그런 다음에 지표를 AWS IoT Device Defender에 게시합니다. 이 구성 요소에서 보고되는 지표를 읽고 해석하는 방법에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [디바이스 지표 문서 사양](#)을 참조하세요.

Note

이 구성 요소는 Device Defender 커넥터와 유사한 기능을 제공합니다 AWS IoT Greengrass V1. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [Device Defender 커넥터](#)를 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [입력 데이터](#)
- [출력 데이터](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 3.1.x
- 3.0.x
- 2.0.x

각 구성 요소 버전의 변경 사항에 대한 내용은 [changelog](#)를 참조하세요.

유형

v3.x

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

v2.x

이 구성 요소는 Lambda 구성 요소(`aws.greengrass.lambda`)입니다. [Greengrass nucleus](#)에서는 [Lambda 런처 구성 요소](#)를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

v3.x

- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- AWS IoT Device Defender Detect 기능을 사용하여 위반을 모니터링하도록 구성되었습니다. 자세한 정보는 AWS IoT Core 개발자 안내서의 [감지](#)를 참조하세요.

v2.x

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.
- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- AWS IoT Device Defender Detect 기능을 사용하여 위반을 모니터링하도록 구성되었습니다. 자세한 정보는 AWS IoT Core 개발자 안내서의 [감지](#)를 참조하세요.
- 코어 디바이스에 설치된 [psutil](#) 라이브러리입니다. 버전 5.7.0이 구성 요소와 연동하는 것으로 확인된 최신 버전입니다.
- 코어 디바이스에 설치된 [cbor](#) 라이브러리입니다. 버전 1.0.0이 구성 요소와 연동하는 것으로 확인된 최신 버전입니다.
- 이 구성 요소의 출력 데이터를 수신하려면 이 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)(aws.greengrass.LegacySubscriptionRouter)에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성에서는 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"
    }
  }
}
```

```
}
}
```

- **region**을 사용하는 AWS 리전 로 바꿉니다.
- **##**을 이 구성 요소에서 실행되는 Lambda 함수의 버전으로 바꿉니다. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소의 버전에 대한 레시피를 확인해야 합니다. [AWS IoT Greengrass 콘솔](#)에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 카-값 페어를 찾습니다. 이 카-값 페어에는 Lambda 함수의 이름과 버전이 포함되어 있습니다.

Important

이 구성 요소를 배포할 때마다 레거시 구독 라우터의 Lambda 함수 버전을 업데이트해야 합니다. 이를 통해 배포하는 구성 요소 버전에 맞는 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

3.1.1

다음 표에는 이 구성 요소의 버전 3.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	소프트
토큰 교환 서비스	>=0.0.0	하드

2.0.12 - 2.0.17

다음 표에는 이 구성 요소의 버전 2.0.12~2.0.17에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.12 - 2.0.16

다음 표에는 이 구성 요소의 버전 2.0.16에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.0.10 - 2.0.11

다음 표에는 이 구성 요소의 버전 2.0.10 및 2.0.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	하드
Lambda 런처	^2.0.0	하드
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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	하드
Lambda 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 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

v3.x

PublishRetryCount

게시가 재시도되는 시간 간격입니다. 이 기능은 버전 3.1.1에서 사용할 수 있습니다.

최소는 0입니다.

최대는 72입니다.

기본값: 5

SampleIntervalSeconds

(선택 사항) 구성 요소에서 지표를 수집하고 보고하는 각 주기 간 시간(초)입니다.

최소값은 300초(5분)입니다.

기본값: 300초

UseInstaller

(선택 사항) 이 구성 요소의 설치 프로그램 스크립트를 사용하여 이 구성 요소의 종속성을 설치할지 여부를 정의하는 부울 값입니다.

사용자 지정 스크립트를 사용하여 종속성을 설치하려는 경우 또는 사전 구축된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 `false`로 설정합니다. 이 구성 요소를 사용하려면 다음 라이브러리를 설치하고(종속성 포함), 기본 Greengrass 시스템 사용자가 사용할 수 있도록 해야 합니다.

- [Python용AWS IoT Device SDK v2](#)
- [cbor](#) 라이브러리입니다. 버전 1.0.0이 구성 요소와 연동하는 것으로 확인된 최신 버전입니다.
- [psutil](#) 라이브러리입니다. 버전 5.7.0이 구성 요소와 연동하는 것으로 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 `false`로 설정해야 합니다. 설치 프로그램 스크립트

에서는 이 구성 요소의 이러한 버전에서 HTTPS 프록시 이면의 작업을 지원하지 않습니다.

기본값: true

v2.x

Note

이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함되어 있습니다. 디바이스에서 다음 파라미터만 사용하여 이 구성 요소를 구성하는 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) /proc 폴더의 경로입니다.

- 이 구성 요소를 컨테이너에서 실행하려면 기본값(/host-proc)을 사용합니다. 기본적으로 컨테이너에서 구성 요소가 실행됩니다.
- 컨테이너 없음 모드에서 이 구성 요소를 실행하려면 이 파라미터에 /proc를 지정합니다.

기본값: /host-proc. 이 구성 요소에서 컨테이너에 /proc 폴더를 탑재하는 기본 경로입니다.

Note

이 구성 요소에는 이 폴더에 대한 읽기 전용 액세스 권한이 있습니다.

SAMPLE_INTERVAL_SECONDS

(선택 사항) 구성 요소에서 지표를 수집하고 보고하는 각 주기 간 시간 간격(초)입니다.

최소값은 300초(5분)입니다.

기본값: 300초

containerMode

(선택 사항) 이 구성 요소에 대한 컨테이너화 모드입니다. 다음 옵션 중 하나를 선택합니다.

- GreengrassContainer - 구성 요소는 AWS IoT Greengrass 컨테이너 내의 격리된 런타임 환경에서 실행됩니다.
- NoContainer - 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.

이 옵션을 지정하는 경우 PROCFS_PATH 환경 변수 파라미터에 /proc를 지정해야 합니다.

기본값: GreengrassContainer

containerParams

(선택 사항) 이 구성 요소에 대한 컨테이너 파라미터를 포함하는 객체입니다.

containerMode에 GreengrassContainer를 지정하면 구성 요소에서 이러한 파라미터를 사용합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

memorySize

(선택 사항) 구성 요소에 할당되는 메모리 양(킬로바이트)입니다.

기본값은 50,000KB입니다.

pubsubTopics

(선택 사항) 구성 요소에서 메시지를 수신하려고 구독하는 주제를 포함하는 객체입니다. 각 주제와 구성 요소가 AWS IoT Core의 MQTT 주제를 구독하는지 또는 로컬 게시/구독 주제를 구독하는지 여부를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 - 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

type

(선택 사항) 이 구성 요소에서 메시지 구독에 사용하는 게시/구독 메시징의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.
- IOT_CORE - AWS IoT Core MQTT 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본값: PUB_SUB

topic

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. type에 IotCore를 지정하면 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

Example예: 구성 병합 업데이트(컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCF_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example예: 구성 병합 업데이트(컨테이너 없음 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCF_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```


입력 데이터

이 구성 요소에서는 메시지를 입력 데이터로 수락하지 않습니다.

출력 데이터

이 구성 요소는 다음 예약 주제에 대한 보안 지표를 게시합니다 AWS IoT Device Defender. 이 구성 요소에서는 지표 게시될 때 *coreDeviceName*을 코어 디바이스의 이름으로 바꿉니다.

주제(AWS IoT Core MQTT): \$aws/things/*coreDeviceName*/defender/metrics/json

Example출력 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 요소에서 보고하는 지표에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [디바이스 지표 문서 사양](#)을 참조하세요.

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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

v3.x

버전	변경 사항
3.1.1	버그 수정 및 개선 사항 - 네트워크 중단 후 연결 복구 실패 시 클라이언트 연결 재시도가 추가되었습니다. - 구성 가능한 지표 게시 재시도가 추가되었습니다.

버전	변경 사항
3.1.0	버그 수정 및 개선 사항 HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정 섹션을 참조하세요.
3.0.1	구성 요소에서 지표에 대한 델타 값을 계산하는 방식과 관련한 문제가 해결되었습니다.
3.0.0	 Warning 더는 이 버전을 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 초기 버전입니다.

v2.x

버전	변경 사항
2.0.17	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.0.16	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.0.11	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.0.10	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.0.9	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.0.8	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.0.5	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

디스크 스폰러

디스크 스폰러 구성 요소(`aws.greengrass.DiskSpooler`)는 Greengrass 코어 디바이스에서 스폰링된 메시지를 위한 영구 스토리지 옵션을 제공합니다 AWS IoT Core. 이 구성 요소에서는 이러한 아웃바운드 메시지를 디스크에 저장합니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- 이 구성 요소를 사용하려면 storageType을 Disk로 설정해야 합니다. [Greengrass nucleus 구성](#)에서 이렇게 설정할 수 있습니다.
- 디바이스에서 사용할 수 있는 공간보다 크게 maxSizeInBytes를 구성해야 합니다. [Greengrass nucleus 구성](#)에서 이렇게 설정할 수 있습니다.
- VPC에서 디스크 스펠러 구성 요소의 실행이 지원됩니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

1.0.7

다음 표에는 이 구성 요소의 버전 1.0.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.11.0 <2.17.0	하드

1.0.

다음 표에는 이 구성 요소의 버전 1.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.11.0 <2.16.0	하드

1.0.5

다음 표에는 이 구성 요소의 버전 1.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.11.0 <2.15.0	하드

1.0.4

다음 표에는 이 구성 요소의 버전 1.0.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.11.0 <2.14.0	하드

1.0.1 – 1.0.3

다음 표에는 이 구성 요소의 버전 1.0.1~1.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.11.0 <2.13.0	하드

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.11.0 <2.12.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

사용법

디스크 스푼러 구성 요소를 사용하려면 `aws.greengrass.DiskSpooler`가 배포되어야 합니다.

이 구성 요소를 구성하고 사용하려면 `pluginName`을 `aws.greengrass.DiskSpooler`로 설정해야 합니다.

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
1.0.7	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
1.0.6	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
1.0.5	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
1.0.4	버그 수정 및 개선 사항 일반적인 버그가 수정되었습니다.
1.0.3	버그 수정 및 개선 사항 데이터베이스 연결을 재사용하여 성능을 개선합니다.
1.0.2	버그 수정 및 개선 사항 특정 사례에서 MQTT 메시지 형식이 지속되지 않은 문제를 해결합니다.
1.0.1	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
1.0.0	초기 버전입니다.

Docker 애플리케이션 관리자

Docker 애플리케이션 관리자 구성 요소(`aws.greengrass.DockerApplicationManager`) AWS IoT Greengrass 를 사용하면 Amazon Elastic Container Registry(Amazon ECR)에 호스팅된 퍼블릭 이미지 레지스트리 및 프라이빗 레지스트리에서 Docker 이미지를 다운로드할 수 있습니다. 또한 AWS IoT Greengrass 는 자격 증명을 자동으로 관리하여 Amazon ECR의 프라이빗 리포지토리에서 이미지를 안전하게 다운로드할 수 있습니다.

Docker 컨테이너가 실행되는 사용자 지정 구성 요소를 개발할 때 Docker 애플리케이션 관리자를 종속성으로 포함하여 구성 요소에서 아티팩트로 지정된 Docker 이미지를 다운로드합니다. 자세한 내용은 [Docker 컨테이너 실행](#) 단원을 참조하십시오.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.0.x

Type

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스에 [Docker Engine](#) 1.9.1 이상이 설치되어 있어야 합니다. 버전 20.10은 AWS IoT Greengrass 코어 소프트웨어에서 작동하는 것으로 확인된 최신 버전입니다. Docker 컨테이너를 실행하는 구성 요소를 배포하기 전에 코어 디바이스에 직접 Docker를 설치해야 합니다.
- 이 구성 요소를 배포하기 전에 Docker 대몬이 코어 디바이스에서 시작되어 실행 중입니다.
- 다음 지원되는 이미지 소스 중 하나에 저장된 Docker 이미지입니다.
 - Amazon Elastic Container Registry(Amazon ECR)의 퍼블릭 및 프라이빗 이미지 리포지토리
 - 퍼블릭 Docker Hub 리포지토리
 - 퍼블릭 Docker 신뢰할 수 있는 레지스트리
- 사용자 지정 Docker 컨테이너 구성 요소에 아티팩트로 포함된 Docker 이미지입니다. 다음 URI 형식을 사용하여 Docker 이미지를 지정합니다.
 - 프라이빗 Amazon ECR 이미지: `docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
 - 퍼블릭 Amazon ECR 이미지: `docker:public.ecr.aws/repository/image[:tag|@digest]`
 - 퍼블릭 Docker Hub 이미지: `docker:name[:tag|@digest]`

자세한 내용은 [Docker 컨테이너 실행](#) 단원을 참조하십시오.

Note

이미지에 대한 아티팩트 URI에서 이미지 태그 또는 이미지 다이제스트를 지정하지 않으면 사용자 지정 Docker 컨테이너 구성 요소를 배포할 때 Docker 애플리케이션 관리자에서는 해당 이미지의 사용 가능한 최신 버전을 가져옵니다. 모든 코어 디바이스에서 동일한 이미지 버전이 실행되도록 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 `docker` 그룹에 사용자를 추가하여 `sudo` 없이 `docker` 명령을 직접적으로 호출할 수 있습니다.
- Windows 디바이스에서는 `docker-users` 그룹에 사용자를 추가하여 관리자 권한 없이 `docker` 명령을 직접적으로 호출할 수 있습니다.

Linux or Unix

Docker 컨테이너 구성 요소를 실행하는 데 사용하는 `ggc_user` 또는 루트 사용자가 아닌 사용자를 `docker` 그룹에 추가하려면 다음 명령을 실행합니다.

```
sudo usermod -aG docker ggc_user
```

자세한 내용은 [루트 사용자가 아닌 사용자로 Docker 관리](#)를 참조하세요.

Windows Command Prompt (CMD)

`ggc_user` 또는 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자를 `docker-users` 그룹에 추가하려면 관리자로 다음 명령을 실행합니다.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

`ggc_user` 또는 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자를 `docker-users` 그룹에 추가하려면 관리자로 다음 명령을 실행합니다.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- [네트워크 프록시를 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성하는 경우 동일한 프록시 서버를 사용하도록 Docker를 구성](#)해야 합니다.
- Docker 이미지가 Amazon ECR 프라이빗 레지스트리에 저장되어 있는 경우 토큰 교환 서비스 구성 요소를 Docker 컨테이너 구성 요소에 종속성으로 포함해야 합니다. 또한 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서 `ecr:GetAuthorizationToken`, `ecr:BatchGetImage`, `ecr:GetDownloadUrlForLayer` 작업을 허용해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
    }
  ],
}
```

```

    "Resource": [
        "*"
    ],
    "Effect": "Allow"
}
]
}

```

- Docker 애플리케이션 관리자 구성 요소는 VPC에서 실행되도록 지원됩니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.
- 이미지를 다운로드할 연결이 Docker 애플리케이션 관리자 구성 요소에 있어야 합니다. 예를 들어, ECR을 사용하는 경우 다음 엔드포인트에 대한 연결이 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 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
ecr. region .amazonaws.com	443	아니요	Amazon ECR에서 Docker 이미지를 다운로드하는 경우에 필요합니다.
hub.docker.com registry.hub.docker.com/v1	443	아니요	Docker Hub에서 Docker 이미지를 다

엔드포인트	포트	필수	설명
			운로드하는 경우에 필요합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.0.15

다음 표에는 이 구성 요소의 버전 2.0.15에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.17.0	소프트

2.0.14

다음 표에는 이 구성 요소의 버전 2.0.14에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.16.0	소프트

2.0.13

다음 표에는 이 구성 요소의 버전 2.0.13에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.15.0	소프트

2.0.12

다음 표에는 이 구성 요소의 버전 2.0.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.14.0	소프트

2.0.11

다음 표에는 이 구성 요소의 버전 2.0.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.13.0	소프트

2.0.10

다음 표에는 이 구성 요소의 버전 2.0.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.12.0	소프트

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.11.0	소프트

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.10.0	소프트

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.9.0	소프트

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.8.0	소프트

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.7.0	소프트

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.6.0	소프트

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.5.0	소프트

2.0.2

다음 표에는 이 구성 요소의 버전 2.0.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.4.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.3.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.2.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. */greengrass/v2* 또는 *C:\greengrass\v2*를 AWS IoT Greengrass 루트 폴더의 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 사항
2.0.15	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.0.14	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.0.13	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.0.12	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.0.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.0.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.0.9	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.0.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.0	초기 버전입니다.

다음 사항도 참조하세요.

- [Docker 컨테이너 실행](#)

Kinesis Video Streams용 엣지 커넥터

Kinesis Video Streams 구성 요소용 엣지 커넥터(`aws.iot.EdgeConnectorForKVS`)는 로컬 카메라에서 비디오 피드를 읽고 스트림을 Kinesis Video Streams에 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 서버에서 대시보드를 설정하여 비디오 스트림을 모니터링하고 조작할 수 있습니다.

이 구성 요소를 AWS IoT TwinMaker와 통합하여 Grafana 대시보드에서 비디오 스트림을 표시하고 제어할 수 있습니다. AWS IoT TwinMaker는 물리적 시스템의 운영 디지털 트윈을 구축할 수 있는 AWS 서비스입니다. AWS IoT TwinMaker를 사용하여 센서, 카메라 및 엔터프라이즈 애플리케이션의 데이터를 시각화하여 물리적 공장, 건물 또는 산업 공장을 추적할 수 있습니다. 이 데이터를 사용하여 작업을 모니터링하고 오류를 진단하고 오류를 수정할 수도 있습니다. 자세한 내용은 [AWS IoT TwinMaker 사용 설명서의 What isTwinMaker?](#)를 참조하세요. AWS IoT TwinMaker

이 구성 요소는 산업 데이터를 모델링하고 저장하는 AWS 서비스 AWS IoT SiteWise인에 구성을 저장합니다. 에서 AWS IoT SiteWise자산은 디바이스, 장비 또는 다른 객체 그룹과 같은 객체를 나타냅니다. 이 구성 요소를 구성하고 사용하려면 각 Greengrass 코어 디바이스와 각 코어 디바이스에 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 AWS Secrets Manager 에서 카메라의 URL이 포함된 보안 암호를 생성합니다. 카메라에 인증이 필요한 경우 URL에 사용자 이름과 암호도 지정합니다. 그런 다음 IP 카메라의 자산 속성에서 해당 보안 암호를 지정합니다.

이 구성 요소는 각 카메라의 비디오 스트림을 Kinesis 비디오 스트림에 업로드합니다. 각 카메라의 AWS IoT SiteWise 자산 구성에서 대상 Kinesis 비디오 스트림의 이름을 지정합니다. Kinesis 비디오 스트림이 존재하지 않으면 이 구성 요소가 스트림을 생성합니다.

AWS IoT TwinMaker는 이러한 AWS IoT SiteWise 자산 및 Secrets Manager 보안 암호를 생성하기 위해 실행할 수 있는 스크립트를 제공합니다. 이러한 리소스를 생성하는 방법과 이 구성 요소를 설치, 구성 및 사용하는 방법에 대한 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 [AWS IoT TwinMaker video integration](#)을 참조하세요.

Note

Kinesis Video Streams 구성 요소용 엣지 커넥터는 다음 AWS 리전에서만 사용할 수 있습니다.

- 미국 동부(버지니아 북부)
- 미국 서부(오리건)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 아시아 태평양(싱가포르)
- 아시아 태평양(도쿄)
- 아시아 태평양(서울)
- 아시아 태평양(시드니)

- 아시아 태평양(뭄바이)
- 중국(베이징)

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [라이선스](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 이 구성 요소는 구성 요소 구성이 각 코어 디바이스에 고유해야 하므로 단일 코어 디바이스에만 배포할 수 있습니다. 이 구성 요소를 코어 디바이스 그룹에 배포할 수 없습니다.
- 코어 디바이스에 설치된 [GStreamer](#) 1.18.4 이상. 자세한 내용은 [Installing GStreamer](#)를 참조하세요.

apt가 있는 디바이스에서는 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 디바이스의 AWS IoT SiteWise 자산입니다. 이 AWS IoT SiteWise 자산은 코어 디바이스를 나타냅니다. 이 자산을 생성하는 방법에 대한 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 [AWS IoT TwinMaker video integration](#)을 참조하세요.
- 각 코어 디바이스에 연결하는 각 IP 카메라의 AWS IoT SiteWise 자산입니다. 이러한 AWS IoT SiteWise 자산은 각 코어 디바이스로 비디오를 스트리밍하는 카메라를 나타냅니다. 각 카메라의 자산은 카메라에 연결된 코어 디바이스의 자산에 연결되어 있어야 합니다. 카메라 자산에는 Kinesis 비디오 스트림, 인증 보안 암호 및 비디오 스트리밍 파라미터를 지정하기 위해 구성할 수 있는 속성이 있습니다. 카메라 자산을 생성하고 구성하는 방법에 대한 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 [AWS IoT TwinMaker video integration](#)을 참조하세요.
- 각 IP 카메라의 AWS Secrets Manager 보안 암호입니다. 이 보안 암호에서는 키-값 페어를 정의해야 하며, 여기서 키는 RTSPStreamUrl이고 값은 카메라의 URL입니다. 카메라에 인증이 필요한 경우 이 URL에 사용자 이름과 암호를 포함합니다. 이 구성 요소에 필요한 리소스를 생성할 때 스크립트를 사용하여 보안 암호를 생성할 수 있습니다. 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 [AWS IoT TwinMaker video integration](#)을 참조하세요.

Secrets Manager 콘솔 및 API를 사용하여 추가 보안 암호를 생성할 수도 있습니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 [보안 암호 생성](#)을 참조하세요.

- [Greengrass 토큰 교환 역할](#)은 다음 예제 IAM 정책에 표시된 대로 다음 AWS Secrets Manager AWS IoT SiteWise 및 Kinesis Video Streams 작업을 허용해야 합니다.

Note

이 예제 정책에서는 디바이스가 **IPCamera1Url** 및 **IPCamera2Url**이라는 보안 암호 값을 가져올 수 있도록 허용합니다. 각 IP 카메라를 구성할 때 해당 카메라의 URL이 포함된 보안 암호를 지정합니다. 카메라에 인증이 필요한 경우 URL에 사용자 이름과 암호도 지정합니다. 코어 디바이스의 토큰 교환 역할에서 연결할 각 IP 카메라의 보안 암호에 대한 액세스를 허용해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera1Url",
        "arn:aws:secretsmanager:us-east-1:123456789012: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 관리형 AWS Key Management Service 키를 사용하여 보안 암호를 암호화하는 경우 디바이스 역할도 kms:Decrypt 작업을 허용해야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
kinesisvideo. <i>region</i> .amazonaws.com	443	예	Kinesis Video Streams에 데이터를 업로드합니다.
data.iotsitewise. <i>region</i> .amazonaws.com	443	예	비디오 스트림 메타 데이터를 AWS IoT SiteWise에 게시합니다.
secretsmanager. <i>region</i> .amazonaws.com	443	예	카메라 URL 보안 암호를 코

엔드포인트	포트	필수	설명
			어 디바이스로 다운로드합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

다음 표에는 이 구성 요소의 버전 1.0.0~1.0.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=2.0.3	하드
스트림 관리자	>=2.0.9	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

SiteWiseAssetIdForHub

이 코어 디바이스를 나타내는 AWS IoT SiteWise 자산의 ID입니다. 이 자산을 생성하고 사용하여 이 구성 요소와 연동하는 방법에 대한 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 [AWS IoT TwinMaker video integration](#)을 참조하세요.

Example예: 구성 병합 업데이트

```
{
```

```
"SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

라이선스

이 구성 요소에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- [Quartz Job Scheduler](#)/Apache 라이선스 2.0
- [GStreamer용 Java 바인딩 1.x](#)/GNU Lesser General Public License v3.0

사용법

이 구성 요소를 구성하고 상호 작용하기 위해 코어 디바이스와 코어 디바이스가 연결되는 IP 카메라를 나타내는 AWS IoT SiteWise 자산에 속성을 설정할 수 있습니다. 또한 AWS IoT TwinMaker를 통해 Grafana 대시보드에서 비디오 스트림을 시각화하고 상호 작용할 수 있습니다. 자세한 내용은 AWS IoT TwinMaker 사용 설명서의 [AWS IoT TwinMaker video integration](#)을 참조하세요.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. 를 AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
1.0.5	일반적인 버그 수정 및 개선 사항.

버전	변경 사항
1.0.4	버그 수정 및 개선 사항 라이브 업로드가 중지되게 한 문제를 해결합니다.
1.0.3	일반적인 버그 수정 및 개선 사항.
1.0.1	일반적인 버그 수정 및 개선 사항.
1.0.0	초기 버전입니다.

다음 사항도 참조하세요.

- [AWS IoT TwinMaker 사용 설명서의TwinMaker란 무엇입니까?](#) AWS IoT TwinMaker
- [AWS IoT TwinMaker 사용 설명서의 TwinMaker 비디오 통합](#) AWS IoT TwinMaker
- AWS IoT SiteWise 사용 설명서의 [란 무엇입니까 AWS IoT SiteWise?](#)
- AWS IoT SiteWise 사용 설명서의 [속성 값 업데이트](#)
- AWS Secrets Manager 사용 설명서의 [AWS Secrets Manager란 무엇입니까?](#)
- AWS Secrets Manager 사용 설명서의 [보안 암호 생성 및 관리](#)

Greengrass CLI

Greengrass CLI 구성 요소(`aws.greengrass.Cli`)는 코어 디바이스에서 로컬로 구성 요소를 개발하고 디버깅하는 데 사용할 수 있는 로컬 명령줄 인터페이스를 제공합니다. 예를 들어 Greengrass CLI를 사용하면 코어 디바이스에서 로컬 배포를 생성하고 구성 요소를 다시 시작할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어를 설치할 때이 구성 요소를 설치할 수 있습니다. 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#) 단원을 참조하십시오.

Important

이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제공합니다. 필요한 코어 디바이스에만 이 구성 요소를 배포하여 최소 권한 원칙을 따르세요.

이 구성 요소를 설치한 후 다음 명령을 실행하여 도움말 설명서를 확인합니다. 이 구성 요소가 설치되면 `/greengrass/v2/bin` 폴더의 `greengrass-cli`에 기호 링크가 추가됩니다. 이 경로에서 Greengrass CLI를 실행하거나 PATH 환경 변수에 추가하여 절대 경로 없이 `greengrass-cli`를 실행할 수 있습니다.

Linux or Unix

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

Windows

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

예를 들어 다음 명령은 `com.example.HelloWorld`라는 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 내용은 [Greengrass 명령줄 인터페이스](#) 단원을 참조하십시오.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.16.x
- 2.15.x
- 2.14.x
- 2.13.x
- 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 nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- AWS IoT Greengrass 코어 소프트웨어와 상호 작용하려면 Greengrass CLI를 사용할 권한이 있어야 합니다. 다음 중 하나를 수행하여 Greengrass CLI 사용:
 - AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자를 사용합니다.
 - 루트 또는 관리 권한이 있는 사용자를 사용합니다. Linux 코어 디바이스에서 sudo를 사용하여 루트 권한을 얻을 수 있습니다.
 - 구성 요소를 배포할 때 AuthorizedPosixGroups 또는 AuthorizedWindowsGroups 구성 파라미터에 지정한 그룹에 있는 시스템 사용자를 사용합니다. 자세한 내용은 [Greengrass CLI 구성 요소 구성](#)을 참조하세요.
- VPC에서 Greengrass CLI 구성 요소의 실행이 지원됩니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.16.0

다음 표에는 이 구성 요소의 버전 2.16.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.12.0 <2.17.0	소프트

2.15.1

다음 표에는 이 구성 요소의 버전 2.15.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.12.0 <2.16.0	소프트

2.15.0

다음 표에는 이 구성 요소의 버전 2.15.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.12.0 <2.16.0	소프트

2.14.0 – 2.14.3

다음 표에는 이 구성 요소의 버전 2.14.0 및 2.14.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.12.0 <2.15.0	소프트

2.13.0

다음 표에는 이 구성 요소의 버전 2.13.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.12.0 <2.14.0	소프트

2.12.0 – 2.12.6

다음 표에는 이 구성 요소의 버전 2.12.0~2.12.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.9.0	소프트

2.7.0

다음 표에는 이 구성 요소의 버전 2.7.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.8.0	소프트

2.6.0

다음 표에는 이 구성 요소의 버전 2.6.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.6.0	소프트

2.4.0

다음 표에는 이 구성 요소의 버전 2.4.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.5.0	소프트

2.3.0

다음 표에는 이 구성 요소의 버전 2.3.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.4.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.1.0	소프트

 Note

Greengrass nucleus의 최소 호환 버전은 Greengrass CLI 구성 요소의 패치 버전과 일치합니다.

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.5.x - 2.14.x

AuthorizedPosixGroups

(선택 사항) 쉘표로 구분된 시스템 그룹 목록이 포함된 문자열. 이러한 시스템 그룹에 Greengrass CLI를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 상호 작용할 수 있는 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어 `group1,1002,group3`은 3개의 시스템 그룹(`group1`, `1002` 및 `group3`)에 Greengrass CLI 사용 권한을 부여합니다.

권한을 부여할 그룹을 지정하지 않으면 Greengrass CLI를 루트 사용자(`sudo`) 또는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자로 사용할 수 있습니다.

AuthorizedWindowsGroups

(선택 사항) 쉘표로 구분된 시스템 그룹 목록이 포함된 문자열. 이러한 시스템 그룹에 Greengrass CLI를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 상호 작용할 수 있는 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어 `group1,1002,group3`은 3개의 시스템 그룹(`group1`, `1002` 및 `group3`)에 Greengrass CLI 사용 권한을 부여합니다.

권한을 부여할 그룹을 지정하지 않으면 Greengrass CLI를 관리자 또는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자로 사용할 수 있습니다.

Example예: 구성 병합 업데이트

다음 예제 구성은 3개의 POSIX 시스템 그룹(`group1`, `1002` 및 `group3`)과 2개의 Windows 사용자 그룹(`Device Operators` 및 `QA Engineers`)에 Greengrass CLI 사용 권한을 부여하도록 지정합니다.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

AuthorizedPosixGroups

(선택 사항) 쉘표로 구분된 시스템 그룹 목록이 포함된 문자열. 이러한 시스템 그룹에 Greengrass CLI를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 상호 작용할

수 있는 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어 `group1,1002,group3`은 3개의 시스템 그룹(`group1`, `1002` 및 `group3`)에 Greengrass CLI 사용 권한을 부여합니다.

권한을 부여할 그룹을 지정하지 않으면 Greengrass CLI를 루트 사용자(`sudo`) 또는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자로 사용할 수 있습니다.

Example예: 구성 병합 업데이트

다음 예제 구성은 3개의 시스템 그룹(`group1`, `1002` 및 `group3`)에 Greengrass CLI 사용 권한을 부여하도록 지정합니다.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. */greengrass/v2* 또는 *C:\greengrass\v2*를 AWS IoT Greengrass 루트 폴더의 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 사항
2.16.1	Greengrass nucleus 버전 2.16.1 릴리스용으로 업데이트된 버전입니다.
2.16.0	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.15.1	Greengrass nucleus 버전 2.15.1 릴리스용으로 업데이트된 버전입니다.
2.15.0	버그 수정 및 개선 사항 - 인증 구성이 변경될 때 구성 요소가 다시 설치되지 않고 계속 다시 시작되도록 수정 사항을 추가합니다. - 일반적인 버그 수정 및 개선 사항.
2.14.3	Greengrass nucleus 버전 2.14.3 릴리스용으로 업데이트된 버전입니다.
2.14.2	Greengrass nucleus 버전 2.14.2 릴리스용으로 업데이트된 버전입니다.
2.14.1	Greengrass nucleus 버전 2.14.1 릴리스용으로 업데이트된 버전입니다.
2.14.0	 Warning 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 사항 cli 명령에서 배포 대상 파라미터를 검증합니다.
2.13.0	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.12.6	Greengrass nucleus 버전 2.12.6 릴리스용으로 업데이트된 버전입니다.
2.12.5	Greengrass nucleus 버전 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.12.4	Greengrass nucleus 버전 2.12.4 릴리스용으로 업데이트된 버전입니다.
2.12.3	 Warning 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. Greengrass nucleus 버전 2.12.3 릴리스용으로 업데이트된 버전입니다.
2.12.2	Greengrass nucleus 버전 2.12.2 릴리스용으로 업데이트된 버전입니다.
2.12.1	Greengrass nucleus 버전 2.12.1 릴리스용으로 업데이트된 버전입니다.
2.12.0	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.11.3	Greengrass nucleus 버전 2.11.3 릴리스용으로 업데이트된 버전입니다.
2.11.2	Greengrass nucleus 버전 2.11.2 릴리스용으로 업데이트된 버전입니다.
2.11.1	Greengrass nucleus 버전 2.11.1 릴리스용으로 업데이트된 버전입니다.
2.11.0	새로운 기능 - 로컬 배포를 취소할 수 있습니다. - 로컬 배포 실패 처리 정책을 구성할 수 있습니다. - 자세한 배포 상태 보고가 개선됩니다.
2.10.3	Greengrass nucleus 버전 2.10.3 릴리스용으로 업데이트된 버전입니다.
2.10.2	Greengrass nucleus 버전 2.10.2 릴리스용으로 업데이트된 버전입니다.
2.10.1	Greengrass nucleus 버전 2.10.1 릴리스용으로 업데이트된 버전입니다.
2.10.0	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.9.6	Greengrass nucleus 버전 2.9.6 릴리스용으로 업데이트된 버전입니다.
2.9.5	Greengrass nucleus 버전 2.9.5 릴리스용으로 업데이트된 버전입니다.
2.9.4	Greengrass nucleus 버전 2.9.4 릴리스용으로 업데이트된 버전입니다.
2.9.3	Greengrass nucleus 버전 2.9.3 릴리스용으로 업데이트된 버전입니다.
2.9.2	Greengrass nucleus 버전 2.9.2 릴리스용으로 업데이트된 버전입니다.
2.9.1	Greengrass nucleus 버전 2.9.1 릴리스용으로 업데이트된 버전입니다.
2.9.0	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.8.1	Greengrass nucleus 버전 2.8.1 릴리스용으로 업데이트된 버전입니다.
2.8.0	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.7.0	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.6.0	새로운 기능 - 사용자 지정 구성 요소에서 Greengrass CLI가 사용하는 IPC(프로세스 간 통신) 작업을 직접 호출할 수 있도록 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 참조하세요. 버그 수정 및 개선 사항 - 사소한 추가 수정 및 개선 사항.
2.5.6	Greengrass nucleus 버전 2.5.6 릴리스용으로 업데이트된 버전입니다.
2.5.5	Greengrass nucleus 버전 2.5.5 릴리스용으로 업데이트된 버전입니다.
2.5.4	Greengrass nucleus 버전 2.5.4 릴리스용으로 업데이트된 버전입니다.
2.5.3	Greengrass nucleus 버전 2.5.3 릴리스용으로 업데이트된 버전입니다.
2.5.2	Greengrass nucleus 버전 2.5.2 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.5.1	Greengrass nucleus 버전 2.5.1 릴리스용으로 업데이트된 버전입니다.
2.5.0	새로운 기능 - Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. - Windows 디바이스에서 시스템 그룹의 Greengrass CLI 사용 승인을 지정할 수 있도록 새 AuthorizedWindowsGroups 구성 파라미터가 추가됩니다. - 로컬 배포에 대해 windowsUser 파라미터가 추가됩니다. 이 파라미터를 사용하여 Windows 코어 디바이스에서 구성 요소 실행에 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 디바이스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 자세한 내용은 구성 요소에 대한 시스템 리소스 제한 구성 및 배포 생성 명령을 참조하세요.
2.3.0	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.2.0	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.0.5 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.0.4 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

IP 감지기

IP 감지기 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 Greengrass 클라우드 서비스에서 코어 디바이스의 연결 정보를 업데이트합니다.

클라이언트 디바이스는 Greengrass 클라우드 검색을 사용하여 관련 코어 디바이스의 연결 정보를 검색할 수 있습니다. 그런 다음 클라이언트 디바이스는 연결될 때까지 각 코어 디바이스에 대한 연결을 시도할 수 있습니다.

Note

클라이언트 디바이스는 Greengrass 코어 디바이스에 연결되어 처리할 MQTT 메시지와 데이터를 보내는 로컬 IoT 디바이스입니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

IP 감지기 구성 요소는 코어 디바이스의 기존 연결 정보를 감지하는 정보로 대체합니다. 이 구성 요소는 기존 정보를 제거하므로 IP 감지기 구성 요소를 사용하거나 연결 정보를 수동으로 관리할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- [Greengrass 서비스 역할](#)에 연결되어 `iot:GetThingShadow` 있어야 AWS 계정 하며 및 `iot:UpdateThingShadow` 권한을 허용해야 합니다.
- 코어 디바이스의 AWS IoT 정책은 `greengrass:UpdateConnectivityInfo` 권한을 허용해야 합니다. 자세한 내용은 [AWS IoT 데이터 영역 작업에 대한 정책](#) 및 [클라이언트 디바이스를 지원하는 최소 AWS IoT 정책](#) 섹션을 참조하세요.
- 기본 포트 8883 이외의 포트를 사용하도록 코어 디바이스의 MQTT 브로커 구성 요소를 구성하는 경우 IP 감지기 v2.1.0 이상을 사용해야 합니다. 브로커가 작동하는 포트를 보고하도록 구성합니다.
- 네트워크 설정이 복잡한 경우 IP 감지기 구성 요소에서 클라이언트 디바이스가 코어 디바이스에 연결할 수 있는 엔드포인트를 식별하지 못할 수 있습니다. IP 감지기 구성 요소가 엔드포인트를 관리할 수 없는 경우 대신 코어 디바이스 엔드포인트를 수동으로 관리해야 합니다. 예를 들어 코어 디바이스가 MQTT 브로커 포트를 전달하는 라우터 뒤에 있는 경우 라우터의 IP 주소를 코어 디바이스의 엔드포인트로 지정해야 합니다. 자세한 내용은 [코어 디바이스 엔드포인트 관리](#) 단원을 참조하십시오.
- IP 감지기 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.2.3

다음 표에는 이 구성 요소의 버전 2.2.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.17.0	소프트

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.16.0	소프트

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.15.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.14.0	소프트

2.1.8 – 2.1.9

다음 표에는 이 구성 요소의 버전 2.1.8 및 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.13.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.12.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.11.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.10.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.9.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.8.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.7.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.5.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.4.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.3.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.2.x

defaultPort

(선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다.

기본값: 8883

includeIPv4LoopbackAddr

(선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: localhost)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.

기본값: false

includeIPv4LinkLocalAddr

(선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.

기본값: false

includeIPv6LoopbackAddr

(선택 사항) 이 옵션을 활성화하여 IPv6 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: localhost)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다. 이 옵션을 사용하려면 includeIPv4Addr를 false, includeIPv6Addr를 true로 설정해야 합니다.

기본값: false

includeIPv6LinkLocalAddr

(선택 사항) 이 옵션을 활성화하여 IPv6 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다. 이 옵션을 사용하려면 includeIPv4Addr를 false, includeIPv6Addr를 true로 설정해야 합니다.

기본값: false

includeIPv4Addr

(선택 사항) 기본값은 true로 설정됩니다. 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv4 주소를 게시할 수 있습니다.

기본값: true

includeIPv6Addrs

(선택 사항) 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv6 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 includeIPv4Addrs를 false로 설정합니다.

기본값: false

2.1.x

defaultPort

(선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다.

기본값: 8883

includeIPv4LoopbackAddrs

(선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: localhost)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.

기본값: false

includeIPv4LinkLocalAddrs

(선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.

기본값: false

2.0.x

includeIPv4LoopbackAddrs

(선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: localhost)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.

기본값: false

includeIPv4LinkLocalAddr

(선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.

기본값: false

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.2.3	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.2.2	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.2.1	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.2.0	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다. 새로운 기능 IPv6 지원을 추가합니다. 이제 로컬 메시징에 IPv6를 사용할 수 있습니다.
2.1.9	버그 수정 및 개선 사항 디버그 로그 수준에서만 로그를 전송하도록 IP 획득 단계가 조정됩니다.
2.1.8	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	버그 수정 및 개선 사항 - 특정 시나리오에서 이 구성 요소가 기록하는 오류 메시지가 개선됩니다. - Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	개선 사항 - 기본값이 아닌 MQTT 브로커 포트를 사용할 수 있는 defaultPort 파라미터가 추가됩니다. - 로그 메시지가 더 명확하도록 업데이트됩니다.

버전	변경 사항
2.0.2	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.1	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.0	초기 버전입니다.

Firehose

Firehose 구성 요소(`aws.greengrass.KinesisFirehose`)는 Amazon Data Firehose 전송 스트림을 통해 Amazon S3, Amazon Redshift, Amazon OpenSearch Service 등의 대상에 데이터를 게시합니다. 자세한 내용은 Amazon Data Firehose 개발자 안내서의 [Amazon Data Firehose란?](#)을 참조하세요.

이 구성 요소를 사용하여 Kinesis 전송 스트림에 게시하려면 이 구성 요소가 구독하는 주제에 메시지를 게시합니다. 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 포함한 다른 주제를 지정할 수 있습니다.

Note

이 구성 요소는 Firehose 커넥터 in AWS IoT Greengrass V1과 유사한 기능을 제공합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [Firehose 커넥터](#)를 참조하세요.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [입력 데이터](#)
- [출력 데이터](#)
- [로컬 로그 파일](#)

- [라이선스](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

Type

이 구성 요소는 Lambda 구성 요소(`aws.greengrass.lambda`)입니다. [Greengrass nucleus](#)에서는 [Lambda 런처 구성 요소](#)를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.
- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서 `firehose:PutRecord` 및 `firehose:PutRecordBatch` 작업을 허용해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Action": [
    "firehose:PutRecord",
    "firehose:PutRecordBatch"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-name"
  ]
}
```

이 구성 요소의 입력 메시지 페이로드에서 기본 전송 스트림을 동적으로 재정의할 수 있습니다. 애플리케이션에서 이 기능을 사용하는 경우 IAM 정책에 모든 대상 스트림이 리소스로 포함되어야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해).

- 이 구성 요소의 출력 데이터를 수신하려면 이 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)(aws.greengrass.LegacySubscriptionRouter)에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성에서는 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"
  }
}
}

```

- **region**을 사용하는 AWS 리전 로 바꿉니다.
- **##**을 이 구성 요소에서 실행되는 Lambda 함수의 버전으로 바꿉니다. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소의 버전에 대한 레시피를 확인해야 합니다. [AWS IoT Greengrass 콘솔](#)에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 카-값 페어를 찾습니다. 이 카-값 페어에는 Lambda 함수의 이름과 버전이 포함되어 있습니다.

Important

이 구성 요소를 배포할 때마다 레거시 구독 라우터의 Lambda 함수 버전을 업데이트해야 합니다. 이를 통해 배포하는 구성 요소 버전에 맞는 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

- Firehose 구성 요소는 VPC에서 실행할 수 있습니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.
 - Firehose 구성 요소에는 VPC 엔드포인트가 `com.amazonaws.region.kinesis-firehose`인 `firehose.region.amazonaws.com`에 대한 연결이 있어야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
firehose. region .amazonaws.com	443	예	Firehose에 데이터를 업로드합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.16.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.15.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.14.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	하드
Lambda 런처	>=1.0.0	하드
Lambda 런타임	>=1.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=1.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

Note

이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함되어 있습니다. 디바이스에서 다음 파라미터만 사용하여 이 구성 요소를 구성하는 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

코어 디바이스 역할에서 모든 대상 전송 스트림에 대해 필요한 작업을 허용해야 합니다. 자세한 내용은 [요구 사항](#) 단원을 참조하십시오.

PUBLISH_INTERVAL

(선택 사항) 구성 요소가 Firehose에 일괄 처리된 데이터를 게시하기 전에 대기할 최대 시간 (초)입니다. 지표를 받자마자 배치 처리 없이 게시하도록 구성 요소를 구성하려면 0을 지정합니다.

이 값은 최대 900초가 될 수 있습니다.

기본값: 10초

DELIVERY_STREAM_QUEUE_SIZE

(선택 사항) 구성 요소가 동일한 전송 스트림에 대한 새 레코드를 거부하기 전에 메모리에 유지할 최대 레코드 수입니다.

이 값은 2,000개 이상 레코드여야 합니다.

기본값: 5,000개 레코드

containerMode

(선택 사항) 이 구성 요소에 대한 컨테이너화 모드입니다. 다음 옵션 중 하나를 선택합니다.

- NoContainer – 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.
- GreengrassContainer - 구성 요소는 AWS IoT Greengrass 컨테이너 내의 격리된 런타임 환경에서 실행됩니다.

기본값: GreengrassContainer

containerParams

(선택 사항) 이 구성 요소에 대한 컨테이너 파라미터를 포함하는 객체입니다. containerMode에 GreengrassContainer를 지정하면 구성 요소에서 이러한 파라미터를 사용합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

memorySize

(선택 사항) 구성 요소에 할당되는 메모리 양(킬로바이트)입니다.

기본값은 64MB(65,535KB)입니다.

pubsubTopics

(선택 사항) 구성 요소에서 메시지를 수신하려고 구독하는 주제를 포함하는 객체입니다. 각 주제와 구성 요소가 AWS IoT Core 또는 로컬 게시/구독 주제의 MQTT 주제를 구독하는지 여부를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

type

(선택 사항) 이 구성 요소에서 메시지 구독에 사용하는 게시/구독 메시징의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.
- IOT_CORE - AWS IoT Core MQTT 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본값: PUB_SUB

topic

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. type에 IotCore를 지정하면 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

Example예: 구성 병합 업데이트(컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example예: 구성 병합 업데이트(컨테이너 없음 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
```

```

    "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
  },
  "containerMode": "NoContainer"
}

```

입력 데이터

이 구성 요소는 다음 주제에 대한 스트림 콘텐츠를 수락하고 이 콘텐츠를 대상 전송 스트림에 전송합니다. 구성 요소는 두 가지 유형의 입력 데이터를 수락합니다.

- `kinesisfirehose/message` 주제에 대한 JSON 데이터.
- `kinesisfirehose/message/binary/#` 주제에 대한 이진 데이터.

JSON 데이터에 대한 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.

유형: 다음 정보를 포함하는 object입니다.

`data`

전송 스트림으로 보낸 데이터입니다.

유형: string

`delivery_stream_arn`

(선택 사항) 대상 Firehose 전송 스트림의 ARN입니다. 기본 전송 스트림을 재정의하려면 이 속성을 지정합니다.

유형: string

`id`

요청에 대한 임의의 ID입니다. 이 속성을 사용하여 입력 요청을 출력 응답에 매핑할 수 있습니다. 이 속성을 지정하면 구성 요소는 응답 객체의 `id` 속성을 해당 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"
}
```

이진 데이터에 대한 기본 주제(로컬 게시/구독): `kinesisfirehose/message/binary/#`

이 주제는 이진 데이터가 포함된 메시지를 보내는데 사용합니다. 이 구성 요소는 이진 데이터를 구문 분석하지 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. 반드시 후행 슬래시(/)를 포함해야 합니다.

출력 데이터

이 구성 요소는 기본적으로 다음 MQTT 주제에 대한 출력 데이터로 응답을 게시합니다. [레거시 구독 라우터 구성 요소](#)의 구성에서 이 주제를 `subject`로 지정해야 합니다. 사용자 지정 구성 요소에서 이 주제에 대한 메시지를 구독하는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

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 요소에서는 연결 오류와 같은 재시도할 수 있는 오류가 감지되면 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. 를 AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

라이선스

이 구성 요소에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- [AWS SDK for Python \(Boto3\)](#)/Apache 라이선스 2.0
- [botocore](#)/Apache 라이선스 2.0

- [dateutil](#)/PSF 라이선스
- [docutils](#)/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- [jmespath](#)/MIT 라이선스
- [s3transfer](#)/Apache 라이선스 2.0
- [urllib3](#)/MIT 라이선스

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.10	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정을(를) 참조하세요.

버전	변경 사항
2.0.8	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

다음 사항도 참조하세요.

- Amazon Kinesis Data Firehose 개발자 가이드의 [Amazon Kinesis Data Firehose란?](#)

Lambda 런처

Lambda 시작 관리자 구성 요소(`aws.greengrass.LambdaLauncher`)는 AWS IoT Greengrass 코어 디바이스에서 함수를 시작하고 중지 AWS Lambda 함수입니다. 또한 이 구성 요소는 컨테이너화를 설정하고 지정하는 사용자로 프로세스를 실행합니다.

Note

코어 디바이스에 Lambda 함수 구성 요소를 배포하면 배포에 이 구성 요소도 포함됩니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)

- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.
- Lambda 런처 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.0.11 – 2.0.13

다음 표에는 이 구성 요소의 버전 2.0.11~2.0.13에 대한 종속성이 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Lambda 관리자	>=2.0.3 <2.1.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 다음 로그 파일을 사용합니다.

```
/greengrass/v2/logs/LambdaFunctionComponentName.log
```

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `tail` AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꾸고 *LambdaFunctionComponentName*이 구성 요소가 시작하는 Lambda 함수 구성 요소의 이름으로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/LambdaFunctionComponentName.log
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.0.13	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
2.0.12	버그 수정 및 개선 사항 이전 프로세스가 제대로 중지되지 않은 경우 Lambda 런처에서 오류가 발생할 수 있는 문제를 해결합니다.
2.0.11	Lambda 관리자 2.3.0이 지원됩니다.
2.0.10	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
2.0.9	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.8	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.0.7	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	일반 성능 향상 및 버그 수정.
2.0.4	버그 수정 및 개선 사항 구성 요소가 AddGroupOwner 를 Lambda 함수 컨테이너로 올바르게 전달하지 않는 문제를 해결합니다.
2.0.3	초기 버전입니다.

Lambda 관리자

Lambda 관리자 구성 요소(`aws.greengrass.LambdaManager`)는 Greengrass 코어 디바이스에서 실행되는 AWS Lambda 함수에 대한 작업 항목 및 프로세스 간 통신을 관리합니다.

Note

코어 디바이스에 Lambda 함수 구성 요소를 배포하면 배포에 이 구성 요소도 포함됩니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.

주제

- [버전](#)
- [운영 체제](#)
- [유형](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.
- Lambda 관리자 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.3.7

다음 표에는 이 구성 요소의 버전 2.3.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.17.0	소프트

2.3.6

다음 표에는 이 구성 요소의 버전 2.3.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.16.0	소프트

2.3.5

다음 표에는 이 구성 요소의 버전 2.3.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.15.0	소프트

2.3.4

다음 표에는 이 구성 요소의 버전 2.3.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.14.0	소프트

2.3.2 and 2.3.3

다음 표에는 이 구성 요소의 버전 2.3.2 및 2.3.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

2.2.5

다음 표에는 이 구성 요소의 버전 2.2.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

2.2.4

다음 표에는 이 구성 요소의 버전 2.2.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

logHandlerMode

Note

Lambda 관리자 버전 2.3.0 이상에만 해당합니다.

사용할 Lambda 로그 관리자의 구현을 선택하는 데 사용됩니다. Lambda 로그를 읽는 데 더 적은 스레드를 사용하려면 값을 `optimized`로 설정합니다.

getResultTimeoutInSeconds

(선택 사항) 시간이 초과되기 전에 Lambda 함수가 실행될 수 있는 최대 시간(초)입니다.

기본값: 60

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

```
/greengrass/v2/logs/greengrass.log
```

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. 를 AWS IoT Greengrass 루트 폴더의 경로 `/greengrass/v2`로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.3.7	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.3.6	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.3.5	버그 수정 및 개선 사항 사용 가능한 경우 nio 대신 epoll을 사용하여 성능을 개선했습니다.
2.3.4	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.3.3	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
2.3.2	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.3.1	버그 수정 및 개선 사항 특정 오류에 대한 로그 수준이 조정됩니다.
2.3.0	새로운 기능 - CPU 부하를 줄이도록 로그 핸들러가 최적화되었습니다. 이 기능을 사용하려면 구성 옵션 logHandlerMode 를 optimized 로 설정합니다. 버그 수정 및 개선 사항 - 더 이상 WorkQueueFullException 에 대한 전체 스택 추적을 로깅하지 않아 로그와 성능이 향상됩니다. - 종료 제한 시간 초과를 방지하기 위해 lambda 종료 제한 시간이 15초에서 300초 사이로 설정됩니다. - 구성을 변경한 후 온디맨드 lambda가 다시 시작되지 않는 문제를 해결합니다.
2.2.11	버그 수정 및 개선 사항 Lambda 구성이 변경될 때 LegacySubscriptionRouter 구성이 업데이트되지 않는 문제를 해결합니다.
2.2.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.2.9	버그 수정 및 개선 사항 왜곡된 클럭으로 인해 포트 번호가 손상되는 문제를 해결합니다.
2.2.8	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.2.7	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.2.6	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.2.5	새로운 기능 - 로컬 게시/구독 메시지를 구독하는 이벤트 소스에서 MQTT 주제 와일드 카드에 대한 지원을 추가합니다. 이 기능을 사용하려면 Greengrass nucleus 구성 요소의 v2.6.0 이상이 필요합니다. - Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.2.4	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.2.3	버그 수정 및 개선 사항 Lambda 함수의 여러 인스턴스가 단일 cgroup을 공유하는 문제를 해결합니다. 이 구성 요소는 cgroup을 사용하여 Lambda 함수의 리소스 사용량을 관리합니다.
2.2.2	버그 수정 및 개선 사항 특정 시나리오에서 고정된 Lambda 함수 구성 요소가 예기치 않게 다시 시작되는 문제를 해결합니다.
2.2.1	버그 수정 및 개선 사항 종속성 확인 문제를 해결하기 위해 이 구성 요소의 Greengrass nucleus 종속성 버전 제약 조건이 변경됩니다.

버전	변경 사항
2.2.0	버그 수정 및 개선 사항 - 재시작 후 Lambda 함수가 로그를 쓸 수 없는 문제를 해결합니다. - 주제에 와일드카드가 있을 때 레거시 구독 라우터가 중복 메시지를 보내는 문제를 해결합니다. - 고정되지 않은 Lambda 함수가 AWS IoT Device SDK에서 Greengrass IPC(프로세스 간 통신) 라이브러리를 사용할 수 없는 문제를 해결합니다.
2.1.4	버그 수정 및 개선 사항 - NodeJS 런타임을 사용하는 Lambda 함수가 하나의 메시지만 처리하게 되는 문제를 해결합니다. - Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

Lambda 런타임

Lambda 런타임 구성 요소(`aws.greengrass.LambdaRuntimes`)는 Greengrass 코어 디바이스가 AWS Lambda 함수를 실행하는 데 사용하는 런타임을 제공합니다.

Note

코어 디바이스에 Lambda 함수 구성 요소를 배포하면 배포에 이 구성 요소도 포함됩니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 섹션을 참조하세요.

주제

- [버전](#)

- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [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.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 섹션을 참조하세요.
- Lambda 런타임 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

이 구성 요소에는 종속성이 없습니다.

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.0.9	버그 수정 및 개선 사항 Python 3.12를 사용하여 구문 경고 수정
2.0.8	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

레거시 구독 라우터

레거시 구독 라우터(`aws.greengrass.LegacySubscriptionRouter`)는 Greengrass 코어 디바이스에서 구독을 관리합니다. 구독은 Lambda 함수가 코어 디바이스에서 MQTT 메시징에 사용할 수 있는 주제를 정의하는 AWS IoT Greengrass V1의 기능입니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [MQTT 메시징 워크플로의 관리형 구독](#)을 참조하세요.

이 구성 요소를 사용하여 AWS IoT Greengrass 코어 SDK를 사용하는 커넥터 구성 요소 및 Lambda 함수 구성 요소에 대한 구독을 활성화할 수 있습니다.

Note

레거시 구독 라우터 구성 요소는 Lambda 함수가 AWS IoT Greengrass 코어 SDK의 `publish()` 함수를 사용하는 경우에만 필요합니다. Lambda 함수 코드를 업데이트하여 AWS IoT Device SDK V2의 프로세스 간 통신(IPC) 인터페이스를 사용하는 경우 레거시 구독 라우터 구성 요소를 배포할 필요가 없습니다. 자세한 내용은 다음 [프로세스 간 통신](#) 서비스를 참조하세요.

- [로컬 메시지 게시/구독](#)
- [AWS IoT Core MQTT 메시지 게시/구독](#)

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

Type

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소의 요구 사항은 다음과 같습니다.

- 레거시 구독 라우터는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.15

다음 표에는 이 구성 요소의 버전 2.1.15에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.17.0	소프트

2.1.14

다음 표에는 이 구성 요소의 버전 2.1.14에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.16.0	소프트

2.1.13

다음 표에는 이 구성 요소의 버전 2.1.13에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.15.0	소프트

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.14.0	소프트

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

v2.1.x

subscriptions

(선택 사항) 코어 디바이스에서 활성화할 구독입니다. 이는 객체이며, 여기서 각 키는 고유한 ID이고 각 값은 해당 커넥터의 구독을 정의하는 객체입니다. V1 커넥터 구성 요소 또는 AWS IoT Greengrass 코어 SDK를 사용하는 Lambda 함수를 배포할 때 구독을 구성해야 합니다.

각 구독 객체에는 다음 정보가 포함되어 있습니다.

id

이 구독의 고유한 ID입니다. 이 ID는 이 구독 객체의 키와 일치해야 합니다.

source

AWS IoT Greengrass 코어 SDK를 사용하여에서 지정한 주제에 MQTT 메시지를 게시하는 Lambda 함수입니다subject. 다음 중 하나를 지정하세요.

- 코어 디바이스에서 Lambda 함수 구성 요소의 이름입니다.
component:com.example.HelloWorldLambda와 같이 component: 접두사를 사용하여 구성 요소 이름을 지정합니다.
- 코어 디바이스에서 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

⚠ Important

Lambda 함수 버전이 변경되면 함수의 새 버전을 사용하여 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.

가져올 함수의 버전이 포함된 Amazon 리소스 이름(ARN)을 지정해야 합니다. \$LATEST같은 버전 별칭을 사용할 수 없습니다.

V1 커넥터 구성 요소에 대한 구독을 배포하려면 구성 요소의 이름 또는 커넥터 구성 요소의 Lambda 함수의 ARN을 지정합니다.

subject

소스와 대상이 메시지를 게시하고 수신할 수 있는 MQTT 주제 또는 주제 필터입니다. 이 값에서는 + 및 # 주제 와일드카드가 지원됩니다.

target

subject에서 지정하는 주제에 대한 MQTT 메시지를 수신하는 대상입니다. 구독은 source 함수가 코어 디바이스의 Lambda 함수에 MQTT 메시지를 게시 AWS IoT Core 하도록 지정합니다. 다음 중 하나를 지정하세요.

- `cloud.source` 함수는 MQTT 메시지에 게시합니다 AWS IoT Core.
- 코어 디바이스에서 Lambda 함수 구성 요소의 이름입니다.
component:com.example.HelloWorldLambda와 같이 `component:` 접두사를 사용하여 구성 요소 이름을 지정합니다.
- 코어 디바이스에서 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

⚠ Important

Lambda 함수 버전이 변경되면 함수의 새 버전을 사용하여 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.

가져올 함수의 버전이 포함된 Amazon 리소스 이름(ARN)을 지정해야 합니다. \$LATEST같은 버전 별칭을 사용할 수 없습니다.

기본값: 구독 없음

Example구성 업데이트 예제(에 대한 구독 정의 AWS IoT Core)

다음 예제에서는 `com.example.HelloWorldLambda` Lambda 함수 구성 요소가 `hello/world` 주제의 AWS IoT Core 에 MQTT 메시지를 게시하도록 지정합니다.

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example예제 구성 업데이트(다른 Lambda 함수 구독 정의)

다음 예제에서는 `com.example.HelloWorldLambda` Lambda 함수 구성 요소가 `hello/world` 주제에 대한 MQTT 메시지를 `com.example.MessageRelay` Lambda 함수 구성 요소에 게시하도록 지정합니다.

```
{
  "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 구독입니다. 이는 객체이며, 여기서 각 키는 고유한 ID 이고 각 값은 해당 커넥터의 구독을 정의하는 객체입니다. V1 커넥터 구성 요소 또는 AWS IoT Greengrass 코어 SDK를 사용하는 Lambda 함수를 배포할 때 구독을 구성해야 합니다.

각 구독 객체에는 다음 정보가 포함되어 있습니다.

`id`

이 구독의 고유한 ID입니다. 이 ID는 이 구독 객체의 키와 일치해야 합니다.

`source`

AWS IoT Greengrass 코어 SDK를 사용하여에서 지정한 주제에 MQTT 메시지를 게시하는 Lambda 함수입니다 `subject`. 다음을 지정합니다.

- 코어 디바이스에서 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

Important

Lambda 함수 버전이 변경되면 함수의 새 버전을 사용하여 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.

가져올 함수의 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 지정합니다.

`subject`

소스와 대상이 메시지를 게시하고 수신할 수 있는 MQTT 주제 또는 주제 필터입니다. 이 값에서는 + 및 # 주제 와일드카드가 지원됩니다.

`target`

`subject`에서 지정하는 주제에 대한 MQTT 메시지를 수신하는 대상입니다. 구독은 `source` 함수가 코어 디바이스의 Lambda 함수에 MQTT 메시지를 게시 AWS IoT Core 하도록 지정합니다. 다음 중 하나를 지정하세요.

- `cloud.source` 함수는 MQTT 메시지에 게시합니다 AWS IoT Core.
- 코어 디바이스에서 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.

Important

Lambda 함수 버전이 변경되면 함수의 새 버전을 사용하여 구독을 구성해야 합니다. 그렇지 않으면 이 구성 요소는 버전이 구독과 일치할 때까지 메시지를 라우팅하지 않습니다.

가져올 함수의 버전이 포함된 Amazon 리소스 이름(ARN)을 지정해야 합니다. \$LATEST같은 버전 별칭을 사용할 수 없습니다.

기본값: 구독 없음

Example구성 업데이트 예제(에 대한 구독 정의 AWS IoT Core)

다음 예제에서는 Greengrass_HelloWorld 함수가 hello/world 주제의 AWS IoT Core 에 MQTT 메시지를 게시하도록 지정합니다.

```
"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"
  }
}
```

Example예제 구성 업데이트(다른 Lambda 함수 구독 정의)

다음 예제에서는 Greengrass_HelloWorld 함수가 hello/world 주제에 대한 MQTT 메시지를 Greengrass_MessageRelay에 게시하도록 지정합니다.

```
"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 사항
2.1.15	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.1.14	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.1.13	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.1.12	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	버그 수정 및 개선 사항 source 및 target에 대한 ARN 대신 구성 요소 이름을 지정할 수 있도록 하는 지원이 추가됩니다. 구독의 구성 요소 이름을 지정하는 경우 Lambda 함수 버전이 변경될 때마다 구독을 재구성할 필요가 없습니다.

버전	변경 사항
2.0.3	초기 버전입니다.

로컬 디버그 콘솔

로컬 디버그 콘솔 구성 요소(`aws.greengrass.LocalDebugConsole`)는 AWS IoT Greengrass 코어 디바이스 및 해당 구성 요소에 대한 정보를 표시하는 로컬 대시보드를 제공합니다. 이 대시보드를 사용하여 코어 디바이스를 디버깅하고 로컬 구성 요소를 관리할 수 있습니다.

Important

이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제공합니다. 최소 권한 원칙에 따라 필요한 코어 디바이스에만 이 구성 요소를 배포하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 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 nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 `debug`가 제공됩니다. AWS IoT Greengrass CLI를 사용하여 코어 디바이스의 대시보드로 인증하는 임시 암호를 생성해야 합니다. 로컬 디버그 콘솔을 사용하려면 AWS IoT Greengrass CLI를 사용할 수 있어야 합니다. 자세한 내용은 [Greengrass CLI 요구 사항](#)을 참조하세요. 암호를 생성하고 로그인하는 방법에 대한 자세한 내용은 [로컬 디버그 콘솔 구성 요소 사용](#)을 참조하세요.
- 로컬 디버그 콘솔 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹

션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.4.6

다음 표에는 이 구성 요소의 버전 2.4.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.17.0	하드
Greengrass CLI	>=2.10.0 <2.17.0	하드

2.4.5

다음 표에는 이 구성 요소의 버전 2.4.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.16.0	하드
Greengrass CLI	>=2.10.0 <2.16.0	하드

2.4.4

다음 표에는 이 구성 요소의 버전 2.4.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.15.0	하드
Greengrass CLI	>=2.10.0 <2.15.0	하드

2.4.3

다음 표에는 이 구성 요소의 버전 2.4.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.14.0	하드
Greengrass CLI	>=2.10.0 <2.14.0	하드

2.4.1 – 2.4.2

다음 표에는 이 구성 요소의 버전 2.4.1~2.4.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.13.0	하드
Greengrass CLI	>=2.10.0 <2.13.0	하드

2.4.0

다음 표에는 이 구성 요소의 버전 2.4.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.12.0	하드
Greengrass 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.10.0 <2.12.0	하드
Greengrass CLI	>=2.10.0 <2.12.0	하드

2.2.9

다음 표에는 이 구성 요소의 버전 2.2.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.12.0	하드
Greengrass CLI	>=2.1.0 <2.12.0	하드

2.2.8

다음 표에는 이 구성 요소의 버전 2.2.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.11.0	하드
Greengrass CLI	>=2.1.0 <2.11.0	하드

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.10.0	하드
Greengrass CLI	>=2.1.0 <2.10.0	하드

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.9.0	하드

종속성	호환 버전	종속성 유형
Greengrass CLI	>=2.1.0 <2.9.0	하드

2.2.5

다음 표에는 이 구성 요소의 버전 2.2.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.8.0	하드
Greengrass CLI	>=2.1.0 <2.8.0	하드

2.2.4

다음 표에는 이 구성 요소의 버전 2.2.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.7.0	하드
Greengrass CLI	>=2.1.0 <2.7.0	하드

2.2.3

다음 표에는 이 구성 요소의 버전 2.2.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.6.0	하드
Greengrass CLI	>=2.1.0 <2.6.0	하드

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.5.0	하드
Greengrass CLI	>=2.1.0 <2.5.0	하드

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.4.0	하드
Greengrass CLI	>=2.1.0 <2.4.0	하드

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.3.0	하드
Greengrass CLI	>=2.1.0 <2.3.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.2.0	하드
Greengrass CLI	>=2.1.0 <2.2.0	하드

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트
Greengrass CLI	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

v2.1.x - v2.4.x

httpsEnabled

(선택 사항) 로컬 디버그 콘솔에 대해 HTTPS 통신을 활성화할 수 있습니다. HTTPS 통신을 활성화하면 로컬 디버그 콘솔에서 자체 서명된 인증서를 생성합니다. 웹 브라우저에서 자체 서명된 인증서를 사용하는 웹 사이트에 대해 보안 경고를 표시하므로 인증서를 수동으로 확인해야 합니다. 그러면 경고를 무시해도 됩니다. 자세한 내용은 [사용법](#) 단원을 참조하십시오.

기본값: true

port

(선택 사항) 로컬 디버그 콘솔을 제공할 포트입니다.

기본값: 1441

websocketPort

(선택 사항) 로컬 디버그 콘솔에 사용할 websocket 포트입니다.

기본값: 1442

bindHostname

(선택 사항) 로컬 디버그 콘솔에 사용할 호스트 이름입니다.

[Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행하는](#) 경우이 파라미터를 로 설정0.0.0.0하면 Docker 컨테이너 외부에서 로컬 디버그 콘솔을 열 수 있습니다.

기본값: localhost

Example예: 구성 병합 업데이트

다음 예제 구성에서는 기본이 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 websocket 포트입니다.

기본값: 1442

bindHostname

(선택 사항) 로컬 디버그 콘솔에 사용할 호스트 이름입니다.

[Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행하는](#) 경우이 파라미터를 로 설정0.0.0.0하면 Docker 컨테이너 외부에서 로컬 디버그 콘솔을 열 수 있습니다.

기본값: localhost

Example예: 구성 병합 업데이트

다음 예제 구성에서는 기본이 아닌 포트에서 로컬 디버그 콘솔을 열도록 지정합니다.

```
{
```

```
"port": "10441",
"websocketPort": "10442"
}
```

사용법

로컬 디버그 콘솔을 사용하려면 Greengrass CLI에서 세션을 생성합니다. 세션을 생성할 때 Greengrass CLI에서는 로컬 디버그 콘솔에 로그인하는 데 사용할 수 있는 사용자 이름과 임시 암호를 제공합니다.

다음 지침에 따라 코어 디바이스 또는 개발 컴퓨터에서 로컬 디버그 콘솔을 엽니다.

v2.1.x - v2.4.x

버전 2.1.0 이상에서는 로컬 디버그 콘솔이 기본적으로 HTTPS를 사용합니다. HTTPS가 활성화된 경우 로컬 디버그 콘솔은 연결을 보호하기 위해 자체 서명된 인증서를 생성합니다. 이 자체 서명된 인증서로 인해 로컬 디버그 콘솔을 열 때 웹 브라우저에서 보안 경고를 표시합니다. Greengrass CLI로 세션을 생성할 때 출력에 인증서의 지문이 포함되므로 인증서가 정상이고 연결이 안전한지 확인할 수 있습니다.

HTTPS를 비활성화할 수 있습니다. 자세한 내용은 [로컬 디버그 콘솔 구성](#)을 참조하세요.

로컬 디버그 콘솔을 열려면

1. (선택 사항) 개발 컴퓨터에서 로컬 디버그 콘솔을 표시하려면 SSH를 통해 콘솔의 포트를 전달할 수 있습니다. 하지만 먼저 코어 디바이스의 SSH 구성 파일에서 AllowTcpForwarding 옵션을 활성화해야 합니다. 이 옵션은 기본적으로 활성화되어 있습니다. 개발 컴퓨터에서 다음 명령을 실행하여 개발 컴퓨터의 localhost:1441에서 대시보드를 봅니다.

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

Note

기본 포트를 1441 및 1442에서 변경할 수 있습니다. 자세한 내용은 [로컬 디버그 콘솔 구성](#)을 참조하세요.

2. 로컬 디버그 콘솔을 사용할 세션을 생성합니다. 세션을 생성할 때 인증에 사용하는 암호를 생성합니다. 이 구성 요소를 사용하여 중요한 정보를 보고 코어 디바이스에서 작업을 수행할 수

있으므로 보안을 강화하기 위해 로컬 디버그 콘솔에 암호가 필요합니다. 구성 요소 구성에서 HTTPS를 활성화하는 경우에는 로컬 디버그 콘솔에서 연결을 보호하기 위해 인증서도 생성합니다. HTTPS는 기본적으로 활성화되어 있습니다.

AWS IoT Greengrass CLI를 사용하여 세션을 생성합니다. 이 명령은 8시간 후 만료되는 임의의 43자 암호를 생성합니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass V2 루트 폴더의 경로로 바꿉니다.

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 예제와 같습니다. 로컬 디버그 콘솔을 열 때 인증서 지문을 사용하여 연결이 안전한지 확인합니다.

```
Username: debug
Password: bEDp3M0Hdj8ou2w5de_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 보기 구성 요소는 8시간 지속되는 세션을 생성합니다. 이 시간 후에 로컬 디버그 콘솔을 다시 보려면 새 암호를 생성해야 합니다.

3. 대시보드를 열고 로그인합니다. SSH를 통해 포트를 전달하는 경우 Greengrass 코어 디바이스 또는 개발 컴퓨터에서 대시보드를 봅니다. 다음 중 하나를 수행하세요.
 - 로컬 디버그 콘솔에서 기본 설정인 HTTPS를 활성화한 경우 다음을 수행합니다.

- a. SSH를 통해 포트를 전달하는 경우 코어 디바이스 또는 개발 컴퓨터에서 `https://localhost:1441`을 엽니다.

브라우저에 잘못된 보안 인증서에 대한 보안 경고가 표시될 수 있습니다.

- b. 브라우저에 보안 경고가 표시되면 인증서가 정상인지 확인하고 보안 경고를 무시합니다. 해결 방법:
 - i. 인증서에 대한 SHA-256 또는 SHA-1 지문을 찾고 `get-debug-password` 명령이 이전에 출력한 SHA-256 또는 SHA-1 지문과 일치하는지 확인합니다. 브라우저에서 지문을 하나 또는 둘 다 제공할 수 있습니다. 브라우저의 설명서를 참조하여 인증서와 해당 지문을 확인합니다. 일부 브라우저에서는 인증서 지문을 지문(thumbprint)이라고 합니다.

Note

인증서 지문이 일치하지 않으면 [Step 2](#)로 이동하여 새 세션을 생성합니다. 그래도 인증서 지문이 일치하지 않는 경우 연결이 안전하지 않을 수 있습니다.

- ii. 인증서 지문이 일치하는 경우 브라우저의 보안 경고를 무시하고 로컬 디버그 콘솔을 엽니다. 브라우저의 설명서를 참조하여 브라우저 보안 경고를 우회합니다.
- c. `get-debug-password` 명령에서 이전에 출력한 사용자 이름과 암호를 사용하여 웹사이트에 로그인합니다.

로컬 디버그 콘솔이 열립니다.

- d. 로컬 디버그 콘솔에 실패한 TLS 핸드셰이크로 인해 WebSocket에 연결할 수 없다는 오류가 표시되면 WebSocket URL에 대한 자체 서명된 보안 경고를 우회해야 합니다.

Error connecting to WebSocket

The connection was closed due to a failure to perform a TLS handshake

Try opening <https://localhost:1442> and bypass any warnings, then reload this page. The WebSocket connection uses the same certificate as this page.

해결 방법:

- i. 로컬 디버그 콘솔을 연 동일한 브라우저에서 `https://localhost:1442`를 엽니다.
- ii. 인증서를 확인하고 보안 경고를 우회합니다.

경고를 우회하면 브라우저에 HTTP 404 페이지가 표시될 수 있습니다.

iii. `https://localhost:1441`을 다시 엽니다.

로컬 디버그 콘솔에 코어 디바이스에 대한 정보가 표시됩니다.

- 로컬 디버그 콘솔에서 HTTPS를 비활성화한 경우 다음을 수행합니다.
 - a. SSH를 통해 포트를 전달하는 경우 코어 디바이스 또는 개발 컴퓨터에서 `http://localhost:1441`을 엽니다.
 - b. `get-debug-password` 명령에서 이전에 출력한 사용자 이름과 암호를 사용하여 웹사이트에 로그인합니다.

로컬 디버그 콘솔이 열립니다.

v2.0.x

로컬 디버그 콘솔을 열려면

1. (선택 사항) 개발 컴퓨터에서 로컬 디버그 콘솔을 표시하려면 SSH를 통해 콘솔의 포트를 전달할 수 있습니다. 하지만 먼저 코어 디바이스의 SSH 구성 파일에서 `AllowTcpForwarding` 옵션을 활성화해야 합니다. 이 옵션은 기본적으로 활성화되어 있습니다. 개발 컴퓨터에서 다음 명령을 실행하여 개발 컴퓨터의 `localhost:1441`에서 대시보드를 봅니다.

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

Note

기본 포트를 1441 및 1442에서 변경할 수 있습니다. 자세한 내용은 [로컬 디버그 콘솔 구성](#)을 참조하세요.

2. 로컬 디버그 콘솔을 사용할 세션을 생성합니다. 세션을 생성할 때 인증에 사용하는 암호를 생성합니다. 이 구성 요소를 사용하여 중요한 정보를 보고 코어 디바이스에서 작업을 수행할 수 있으므로 보안을 강화하기 위해 로컬 디버그 콘솔에 암호가 필요합니다.

AWS IoT Greengrass CLI를 사용하여 세션을 생성합니다. 이 명령은 8시간 후 만료되는 임의의 43자 암호를 생성합니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass V2 루트 폴더의 경로로 바꿉니다.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

이 명령 출력은 다음 예제와 같습니다.

```
Username: debug
Password: bEDp3M0Hdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

디버그 보기 구성 요소는 4시간 지속되는 세션을 생성하며, 이 시간 이후 로컬 디버그 콘솔을 다시 보려면 새 암호를 생성해야 합니다.

3. SSH를 통해 포트를 전달하는 경우 코어 디바이스 또는 개발 컴퓨터에서 `http://localhost:1441`을 엽니다.
4. `get-debug-password` 명령에서 이전에 출력한 사용자 이름과 암호를 사용하여 웹 사이트에 로그인합니다.

로컬 디버그 콘솔이 열립니다.

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.4.6	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.4.5	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.4.4	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.4.3	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다. 버그 수정 및 개선 사항 STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH 가 초당 바이트 (Bps) 대신 초당 메가비트(Mbps)로 잘못 표시되는 문제를 해결합니다.
2.4.2	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
2.4.1	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.4.0	새로운 기능 스트림 관리자 디버깅 콘솔이 추가됩니다.
2.3.1	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.3.0	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다. 새로운 기능 PubSub 및 AWS IoT Core MQTT 디버그 클라이언트를 포함합니다.
2.2.7	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.2.6	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.2.5	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.2.4	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.2.3	버그 수정 및 개선 사항 - 구성 요소가 SSL 프라이빗 키를 보관하는 키 스토어를 복호화할 수 없을 때 시작되지 않는 문제를 해결합니다. - Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.2.2	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.2.1	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.2.0	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 - HTTPS를 사용하여 로컬 디버그 콘솔에 대한 연결을 보호합니다. HTTPS는 기본적으로 활성화되어 있습니다. 버그 수정 및 개선 사항 - 구성 편집기에서 플래시바 메시지를 해제할 수 있습니다.
2.0.3	초기 버전입니다.

로그 관리자

Warning

로그 관리자 v2.3.5 이상으로 업그레이드하는 것이 좋습니다. 버전 2.3.5에서는 로그 관리자 구성 쓰기를 최적화하여 IO 작업을 줄이고 로그 업로드 속도와 전체 디바이스 성능을 개선하고 디바이스 수명을 연장할 수 있습니다.

로그 관리자 구성 요소(`aws.greengrass.LogManager`)는 AWS IoT Greengrass 코어 디바이스의 로그를 Amazon CloudWatch Logs에 업로드합니다. Greengrass nucleus, 기타 Greengrass 구성 요소, Greengrass 구성 요소가 아닌 다른 애플리케이션 및 서비스에서 로그를 업로드할 수 있습니다. CloudWatch Logs 및 로컬 파일 시스템에서 로그를 모니터링하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 섹션을 참조하세요.

로그 관리자 구성 요소를 사용하여 CloudWatch Logs에 쓸 때는 다음 사항을 고려하세요.

- 로그 지연

로그 관리자 구성 요소 2.2.8 및 이전 버전은 교체된 로그 파일에서만 로그를 처리하고 업로드합니다. 기본적으로 AWS IoT Greengrass 코어 소프트웨어는 매시간 또는 1,024KB 후에 로그 파일을 교체합니다. 따라서 로그 관리자 구성 요소는 AWS IoT Greengrass 코어 소프트웨어 또는 Greengrass 구성 요소가 1,024KB 이상의 로그를 쓴 후에만 로그를 업로드합니다. 로그 파일 크기 제한을 더 낮게 구성하여 로그 파일이 더 자주 교체되도록 할 수 있습니다. 이렇게 하면 로그 관리자 구성 요소에서 CloudWatch Logs에 로그를 더 자주 업로드합니다.

로그 관리자 구성 요소 2.3.0 이상 버전은 모든 로그를 처리하고 업로드합니다. 새 로그를 쓸 때 로그 관리자 2.3.0 이상 버전은 해당 활성 로그 파일이 교체될 때까지 기다리지 않고 활성 로그를 처리하고 직접 업로드합니다. 즉, 새 로그를 5분 이내에 볼 수 있습니다.

로그 관리자 구성 요소에서는 새 로그를 주기적으로 업로드합니다. 기본적으로 로그 관리자 구성 요소는 5분마다 새 로그를 업로드합니다. 업로드 간격을 더 낮게 구성할 수 있으므로 `periodicUploadIntervalSec`을 구성하면 로그 관리자 구성 요소가 CloudWatch Logs에 로그를 더 자주 업로드합니다. 이 주기적 간격을 구성하는 방법에 대한 자세한 내용은 [구성](#)을 참조하세요.

동일한 Greengrass 파일 시스템에서 로그를 거의 실시간으로 업로드할 수 있습니다. 로그를 실시간으로 관찰해야 하는 경우 [파일 시스템 로그](#)를 사용하는 것이 좋습니다.

Note

다른 파일 시스템을 사용하여 로그를 쓰는 경우 로그 관리자가 로그 관리자 구성 요소 2.2.8 이하 버전의 동작으로 돌아갑니다. 파일 시스템 로그에 액세스하는 방법에 대한 자세한 내용은 [파일 시스템 로그 액세스](#)를 참조하세요.

클록 스큐

로그 관리자 구성 요소는 표준 Signature 버전 4 서명 프로세스를 사용하여 CloudWatch Logs에 대한 API 요청을 생성합니다. 코어 디바이스에서 시스템 시간이 15분 이상 동기화되지 않을 경우, CloudWatch Logs에서 요청을 거부합니다. 자세한 정보는 AWS 일반 참조의 [서명 버전 4 서명 프로세스](#)를 참조하세요.

이 구성 요소가 로그를 업로드하는 로그 그룹 및 로그 스트림에 대한 자세한 내용은 [사용법](#) 섹션을 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.3.x
- 2.2.x
- 2.1.x

- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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.

- 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서 `logs:CreateLogGroup`, `logs:CreateLogStream`, `logs:PutLogEvents` 및 `logs:DescribeLogStreams` 작업을 허용해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
```

```

    }
  ]
}
```

Note

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 생성하는 [Greengrass 디바이스 역할](#)에는 기본적으로 이 예제 정책의 권한이 포함됩니다.

자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 [CloudWatch Logs에 대한 자격 증명 기반 정책\(IAM 정책\) 사용](#)을 참조하세요.

- 로그 관리자 구성 요소는 VPC에서 실행할 수 있습니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.
- 로그 관리자 구성 요소에는 VPC 엔드포인트가 `com.amazonaws.us-east-1.logs`인 `logs.region.amazonaws.com`에 대한 연결이 있어야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
<code>logs.region.amazonaws.com</code>	443	아니요	CloudWatch Logs에 로그를 쓰는 경우 필요합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는

의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.3.11

다음 표에는 이 구성 요소의 버전 2.3.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.17.0	소프트

2.3.10

다음 표에는 이 구성 요소의 버전 2.3.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.16.0	소프트

2.3.9

다음 표에는 이 구성 요소의 버전 2.3.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.15.0	소프트

2.3.8

다음 표에는 이 구성 요소의 버전 2.3.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.14.0	소프트

2.3.7

다음 표에는 이 구성 요소의 버전 2.3.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.10.0	소프트

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.9.0	소프트

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.8.0	소프트

2.2.5

다음 표에는 이 구성 요소의 버전 2.2.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.1.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

v2.3.10

logsUploaderConfiguration

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그에 대한 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

systemLogsConfiguration

(선택 사항) [Greengrass nucleus](#) 및 [플러그인 구성 요소의](#) 로그를 포함하는 AWS IoT Greengrass 코어 소프트웨어 시스템 로그에 대한 구성입니다. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

uploadToCloudWatch

(선택 사항) 시스템 로그를 CloudWatch Logs에 업로드할 수 있습니다.

기본값: false

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 JSON 형식 로그를 출력하도록 [Greengrass nucleus 구성 요소](#)를 구성하는 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 [로깅 형식](#) 파라미터(JSON)에 logging.format을 지정합니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 Greengrass 시스템 로그 파일의 최대 총 크기입니다. Greengrass 시스템 로그 파일의 총 크기가 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogSizeKB)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

componentLogsConfigurationMap

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵입니다. 이 맵의 각 componentName 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 CloudWatch Logs에 업로드합니다.

Important

구성 요소당 단일 구성 키를 사용하는 것이 좋습니다. logFileRegex를 사용할 때 활발하게 기록되는 로그 파일이 하나만 있는 파일 그룹만을 대상으로 지정해야 합니다. 이 권장 사항을 따르지 않으면 중복 로그가 CloudWatch에 업로드될 수 있습니다. 단일 정규식으로 여러 활성 로그 파일을 대상으로 지정하는 경우에는 로그 관리자 v2.3.1 이상으로 업그레이드하고 [예제 구성](#)을 사용하여 구성을 변경하는 것이 좋습니다.

Note

v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 componentLogsConfigurationMap 대신 componentLogsConfiguration 목록을 계속 사용할 수 있습니다. 그러나 병합 및 재설정 업데이트를 사용하여 특

정 구성 요소에 대한 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration` 파라미터에 대한 자세한 내용은 이 구성 요소의 v2.1.x에 대한 구성 파라미터를 참조하세요.

componentName

componentName 구성 요소에 대한 로그 구성 또는 이 로그 구성에 대한 애플리케이션입니다. Greengrass 구성 요소의 이름 또는 이 로그 그룹을 식별하는 다른 값을 지정할 수 있습니다.

각 객체에는 다음 정보가 포함되어 있습니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 GitHub의 [AWS IoT Greengrass 로깅 모듈](#) 리포지토리에서 찾을 수 있는 특정 JSON 형식을 사용하는 경우에만 적용됩니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

`diskSpaceLimit`

(선택 사항) `diskSpaceLimitUnit`에서 지정하는 단위로 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기입니다. 이 구성 요소의 로그 파일의 총 크기가 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (`totalLogsSizeKB`)와 관련이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

`diskSpaceLimitUnit`

(선택 사항) `diskSpaceLimit`의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

logFileDirectoryPath

(선택 사항) 이 구성 요소의 로그 파일이 포함된 폴더의 경로입니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

기본값: */greengrass/v2/logs*.

logFileRegex

(선택 사항) 구성 요소 또는 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규식입니다. 로그 관리자 구성 요소는 이 정규식을 사용하여 logFileDirectoryPath의 폴더에서 로그 파일을 식별합니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

구성 요소 또는 애플리케이션에서 로그 파일을 교체하는 경우 교체된 로그 파일 이름과 일치하는 정규식을 지정합니다. 예를 들어 Hello World 애플리케이션의 로그를 업로드하려면 **hello_world\\\\w*.log**를 지정할 수 있습니다. **\\\\w*** 패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일과 일치합니다. 이 예제에서는 로그 관리자가 다음 로그 파일을 업로드합니다.

- hello_world.log - Hello World 애플리케이션의 최신 로그 파일입니다.
- hello_world_2020_12_15_17_0.log - Hello World 애플리케이션의 이전 로그 파일입니다.

기본값: *componentName\\\\w*.log*, 여기서 *componentName*은 이 로그 구성에 대한 구성 요소의 이름입니다.

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

multiLineStartPattern

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규식입니다. 정규식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 새 줄을 이전 줄의 로그 메시지에 추가합니다.

기본적으로 로그 관리자 구성 요소는 줄이 탭 또는 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자는 해당 줄을 새 로그 메시지로 처리합니다. 공백 문자로 시작하는 경우 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성 요소가 스택 추적과 같이 여러 줄에 걸쳐 있는 메시지를 분할하지 않도록 하기 위해서입니다.

periodicUploadIntervalSec

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간(초)입니다.

기본값: 300(5분)

최소값: 0.000001(1마이크로초)

updateToTlogIntervalSec

(선택 사항) nucleus가 Amazon CloudWatch Events 로그 업로드 이벤트 세부 정보를 로컬 트랜잭션 로그(config.tlog)에 기록하는 주기를 초 단위로 지정합니다. 기본값은 periodicUploadIntervalSec에 지정된 값입니다. 이 파라미터를 수정하여 기록 간격을 늘릴 수 있습니다.

기본값: periodicUploadIntervalSec

최소: periodicUploadIntervalSec

deprecatedVersionSupport

로그 관리자가 로그 관리자 v2.3.5에 도입된 로깅 속도 개선을 사용해야 하는지 여부를 나타냅니다. 개선 사항을 사용하려면 값을 false로 설정합니다.

로그 관리자 v2.3.1 또는 이전 버전에서 업그레이드할 때 이 값을 false로 설정하는 경우 중복 로그 항목이 업로드될 수 있습니다.

기본값은 true입니다.

Example예: 구성 병합 업데이트

다음 구성 예제에서는 시스템 로그와 com.example.HelloWorld 구성 요소 로그를 CloudWatch Logs에 업로드하도록 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"
}
```

Example예: 로그 관리자 v2.3.1을 사용하여 여러 활성 로그 파일을 업로드하는 구성

다음 예제 구성은 여러 활성 로그 파일을 대상으로 지정하려는 경우 권장되는 예제입니다. 이 예제 구성에서는 CloudWatch에 업로드할 활성 로그 파일을 지정합니다. 이 구성 예제를 사용하면 logFileRegex와 일치하는 교체된 파일도 업로드됩니다. 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"
}

```

v2.3.6 – v2.3.9

logsUploaderConfiguration

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그에 대한 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

systemLogsConfiguration

(선택 사항) [Greengrass nucleus](#) 및 [플러그인 구성 요소](#)의 로그를 포함하는 AWS IoT Greengrass 코어 소프트웨어 시스템 로그에 대한 구성입니다. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

uploadToCloudWatch

(선택 사항) 시스템 로그를 CloudWatch Logs에 업로드할 수 있습니다.

기본값: false

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 JSON 형식 로그를 출력하도록 [Greengrass nucleus 구성 요소](#)를 구성하는 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 [로깅 형식](#) 파라미터(JSON)에 `logging.format`를 지정합니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 Greengrass 시스템 로그 파일의 최대 총 크기입니다. Greengrass 시스템 로그 파일의 총 크기가 이 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소](#)의 [로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

componentLogsConfigurationMap

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵입니다. 이 맵의 각 componentName 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 CloudWatch Logs에 업로드합니다.

Important

구성 요소당 단일 구성 키를 사용하는 것이 좋습니다. logFileRegex를 사용할 때 활발하게 기록되는 로그 파일이 하나만 있는 파일 그룹만을 대상으로 지정해야 합니다. 이 권장 사항을 따르지 않으면 중복 로그가 CloudWatch에 업로드될 수 있습니다. 단일 정규식으로 여러 활성 로그 파일을 대상으로 지정하는 경우에는 로그 관리

자 v2.3.1 이상으로 업그레이드하고 [예제 구성](#)을 사용하여 구성을 변경하는 것이 좋습니다.

Note

v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 `componentLogsConfigurationMap` 대신 `componentLogsConfiguration` 목록을 계속 사용할 수 있습니다. 그러나 병합 및 재설정 업데이트를 사용하여 특정 구성 요소에 대한 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration` 파라미터에 대한 자세한 내용은 이 구성 요소의 v2.1.x에 대한 구성 파라미터를 참조하세요.

componentName

componentName 구성 요소에 대한 로그 구성 또는 이 로그 구성에 대한 애플리케이션입니다. Greengrass 구성 요소의 이름 또는 이 로그 그룹을 식별하는 다른 값을 지정할 수 있습니다.

각 객체에는 다음 정보가 포함되어 있습니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 GitHub의 [AWS IoT Greengrass 로깅 모듈](#) 리포지토리에서 찾을 수 있는 특정 JSON 형식을 사용하는 경우에만 적용됩니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

`diskSpaceLimit`

(선택 사항) `diskSpaceLimitUnit`에서 지정하는 단위로 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기입니다. 이 구성 요소의 로그 파일의 총 크기가 최대 총 크

기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 관련이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

logFileDirectoryPath

(선택 사항) 이 구성 요소의 로그 파일이 포함된 폴더의 경로입니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

기본값: */greengrass/v2/logs*.

logFileRegex

(선택 사항) 구성 요소 또는 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규식입니다. 로그 관리자 구성 요소는 이 정규식을 사용하여 logFileDirectoryPath의 폴더에서 로그 파일을 식별합니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

구성 요소 또는 애플리케이션에서 로그 파일을 교체하는 경우 교체된 로그 파일 이름과 일치하는 정규식을 지정합니다. 예를 들어 Hello World 애플리케이션의 로그를 업로드하려면 **hello_world\\\\w*.log**를 지정할 수 있습니다. **\\\\w*** 패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일과 일치합니다. 이 예제에서는 로그 관리자가 다음 로그 파일을 업로드합니다.

- `hello_world.log` - Hello World 애플리케이션의 최신 로그 파일입니다.
- `hello_world_2020_12_15_17_0.log` - Hello World 애플리케이션의 이전 로그 파일입니다.

기본값: `componentName\\\\w*.log`, 여기서 `componentName`은 이 로그 구성에 대한 구성 요소의 이름입니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규식입니다. 정규식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 새 줄을 이전 줄의 로그 메시지에 추가합니다.

기본적으로 로그 관리자 구성 요소는 줄이 탭 또는 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자는 해당 줄을 새 로그 메시지로 처리합니다. 공백 문자로 시작하는 경우 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성 요소가 스택 추적과 같이 여러 줄에 걸쳐 있는 메시지를 분할하지 않도록 하기 위해서입니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간(초)입니다.

기본값: `300`(5분)

최소값: `0.000001`(1마이크로초)

`deprecatedVersionSupport`

로그 관리자가 로그 관리자 v2.3.5에 도입된 로깅 속도 개선을 사용해야 하는지 여부를 나타냅니다. 개선 사항을 사용하려면 값을 `false`로 설정합니다.

로그 관리자 v2.3.1 또는 이전 버전에서 업그레이드할 때 이 값을 `false`로 설정하는 경우 중복 로그 항목이 업로드될 수 있습니다.

기본값은 `true`입니다.

Example예: 구성 병합 업데이트

다음 구성 예제에서는 시스템 로그와 com.example.HelloWorld 구성 요소 로그를 CloudWatch Logs에 업로드하도록 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"
}
```

Example예: 로그 관리자 v2.3.1을 사용하여 여러 활성 로그 파일을 업로드하는 구성

다음 예제 구성은 여러 활성 로그 파일을 대상으로 지정하려는 경우 권장되는 예제입니다. 이 예제 구성에서는 CloudWatch에 업로드할 활성 로그 파일을 지정합니다. 이 구성 예제를 사용하면 logFileRegex와 일치하는 교체된 파일도 업로드됩니다. 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"
}

```

v2.3.0 – 2.3.5

logsUploaderConfiguration

(선택 사항) 로그 관리자 구성 요소가 업로드하는 로그에 대한 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

systemLogsConfiguration

(선택 사항) [Greengrass nucleus](#) 및 [플러그인 구성 요소의](#) 로그를 포함하는 AWS IoT Greengrass 코어 소프트웨어 시스템 로그에 대한 구성입니다. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

uploadToCloudWatch

(선택 사항) 시스템 로그를 CloudWatch Logs에 업로드할 수 있습니다.

기본값: false

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 JSON 형식 로그를 출력하도록 [Greengrass nucleus 구성 요소](#)를 구성하는 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 [로깅 형식](#) 파라미터(JSON)에 logging.format을 지정합니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) `diskSpaceLimitUnit`에서 지정하는 단위로 Greengrass 시스템 로그 파일의 최대 총 크기입니다. Greengrass 시스템 로그 파일의 총 크기가 이 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (`totalLogsSizeKB`)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) `diskSpaceLimit`의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

componentLogsConfigurationMap

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵입니다. 이 맵의 각 `componentName` 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 CloudWatch Logs에 업로드합니다.

Important

구성 요소당 단일 구성 키를 사용하는 것이 좋습니다. `logFileRegex`를 사용할 때 활발하게 기록되는 로그 파일이 하나만 있는 파일 그룹만을 대상으로 지정해야 합니다. 이 권장 사항을 따르지 않으면 중복 로그가 CloudWatch에 업로드될 수 있습니다. 단일 정규식으로 여러 활성 로그 파일을 대상으로 지정하는 경우에는 로그 관리자 v2.3.1로 업그레이드하고 [예제 구성](#)을 사용하여 구성을 변경하는 것이 좋습니다.

 Note

v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 `componentLogsConfigurationMap` 대신 `componentLogsConfiguration` 목록을 계속 사용할 수 있습니다. 그러나 병합 및 재설정 업데이트를 사용하여 특정 구성 요소에 대한 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration` 파라미터에 대한 자세한 내용은 이 구성 요소의 v2.1.x에 대한 구성 파라미터를 참조하세요.

componentName

componentName 구성 요소에 대한 로그 구성 또는 이 로그 구성에 대한 애플리케이션입니다. Greengrass 구성 요소의 이름 또는 이 로그 그룹을 식별하는 다른 값을 지정할 수 있습니다.

각 객체에는 다음 정보가 포함되어 있습니다.

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 GitHub의 [AWS IoT Greengrass 로깅 모듈](#) 리포지토리에서 찾을 수 있는 특정 JSON 형식을 사용하는 경우에만 적용됩니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) `diskSpaceLimitUnit`에서 지정하는 단위로 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기입니다. 이 구성 요소의 로그 파일의 총 크기가 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 관련이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

logFileDirectoryPath

(선택 사항) 이 구성 요소의 로그 파일이 포함된 폴더의 경로입니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

기본값: */greengrass/v2/logs*.

logFileRegex

(선택 사항) 구성 요소 또는 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규식입니다. 로그 관리자 구성 요소는 이 정규식을 사용하여 logFileDirectoryPath의 폴더에서 로그 파일을 식별합니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

구성 요소 또는 애플리케이션에서 로그 파일을 교체하는 경우 교체된 로그 파일 이름과 일치하는 정규식을 지정합니다. 예를 들어 Hello World 애플리케이션의 로그를 업로드하려면 **hello_world\\\\w*.log**를 지정할 수 있습니다. \\\w* 패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일과 일치합니다. 이 예제에서는 로그 관리자가 다음 로그 파일을 업로드합니다.

- hello_world.log - Hello World 애플리케이션의 최신 로그 파일입니다.
- hello_world_2020_12_15_17_0.log - Hello World 애플리케이션의 이전 로그 파일입니다.

기본값: `componentName\\\\w*.log`, 여기서 `componentName`은 이 로그 구성에 대한 구성 요소의 이름입니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규식입니다. 정규식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 새 줄을 이전 줄의 로그 메시지에 추가합니다.

기본적으로 로그 관리자 구성 요소는 줄이 탭 또는 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자는 해당 줄을 새 로그 메시지로 처리합니다. 공백 문자로 시작하는 경우 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성 요소가 스택 추적과 같이 여러 줄에 걸쳐 있는 메시지를 분할하지 않도록 하기 위해서입니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간(초)입니다.

기본값: 300(5분)

최소값: 0.000001(1마이크로초)

Example예: 구성 병합 업데이트

다음 구성 예제에서는 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 CloudWatch Logs에 업로드하도록 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"
}

```

Example예: 로그 관리자 v2.3.1을 사용하여 여러 활성 로그 파일을 업로드하는 구성

다음 예제 구성은 여러 활성 로그 파일을 대상으로 지정하려는 경우 권장되는 예제입니다. 이 예제 구성에서는 CloudWatch에 업로드할 활성 로그 파일을 지정합니다. 이 구성 예제를 사용하면 `logFileRegex`와 일치하는 교체된 파일도 업로드됩니다. 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 로그에 대한 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

systemLogsConfiguration

(선택 사항) [Greengrass nucleus](#) 및 [플러그인 구성 요소](#)의 로그를 포함하는 AWS IoT Greengrass 코어 소프트웨어 시스템 로그에 대한 구성입니다. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

uploadToCloudWatch

(선택 사항) 시스템 로그를 CloudWatch Logs에 업로드할 수 있습니다.

기본값: false

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 JSON 형식 로그를 출력하도록 [Greengrass nucleus 구성 요소](#)를 구성하는 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 [로깅 형식](#) 파라미터(JSON)에 logging.format을 지정합니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 Greengrass 시스템 로그 파일의 최대 총 크기입니다. Greengrass 시스템 로그 파일의 총 크기가이 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소](#)의 [로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트

- MB – 메가바이트
- GB – 기가바이트

기본값: KB

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

`componentLogsConfigurationMap`

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 맵입니다. 이 맵의 각 `componentName` 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 CloudWatch Logs에 업로드합니다.

Note

v2.2.0 이전 버전의 로그 관리자에서 업그레이드하는 경우 `componentLogsConfigurationMap` 대신 `componentLogsConfiguration` 목록을 계속 사용할 수 있습니다. 그러나 병합 및 재설정 업데이트를 사용하여 특정 구성 요소에 대한 구성을 수정할 수 있도록 맵 형식을 사용하는 것이 좋습니다. `componentLogsConfiguration` 파라미터에 대한 자세한 내용은 이 구성 요소의 v2.1.x에 대한 구성 파라미터를 참조하세요.

componentName

componentName 구성 요소에 대한 로그 구성 또는 이 로그 구성에 대한 애플리케이션입니다. Greengrass 구성 요소의 이름 또는 이 로그 그룹을 식별하는 다른 값을 지정할 수 있습니다.

각 객체에는 다음 정보가 포함되어 있습니다.

`minimumLogLevel`

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 GitHub의 [AWS IoT Greengrass 로깅 모듈](#) 리포지토리에서 찾을 수 있는 특정 JSON 형식을 사용하는 경우에만 적용됩니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기입니다. 이 구성 요소의 로그 파일의 총 크기가 이 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 관련이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

logFileDirectoryPath

(선택 사항) 이 구성 요소의 로그 파일이 포함된 폴더의 경로입니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

기본값: */greengrass/v2/logs*.

logFileRegex

(선택 사항) 구성 요소 또는 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규식입니다. 로그 관리자 구성 요소는 이 정규식을 사용하여 logFileDirectoryPath의 폴더에서 로그 파일을 식별합니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

구성 요소 또는 애플리케이션에서 로그 파일을 교체하는 경우 교체된 로그 파일 이름과 일치하는 정규식을 지정합니다. 예를 들어 Hello World 애플리케이션의 로그를 업로드하려면 **hello_world\\\\\\w*.log**를 지정할 수 있습니다. \\\\w* 패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일과 일치합니다. 이 예제에서는 로그 관리자가 다음 로그 파일을 업로드합니다.

- hello_world.log - Hello World 애플리케이션의 최신 로그 파일입니다.
- hello_world_2020_12_15_17_0.log - Hello World 애플리케이션의 이전 로그 파일입니다.

기본값: **componentName\\\\\\w*.log**, 여기서 **componentName**은 이 로그 구성에 대한 구성 요소의 이름입니다.

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

multiLineStartPattern

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규식입니다. 정규식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 새 줄을 이전 줄의 로그 메시지에 추가합니다.

기본적으로 로그 관리자 구성 요소는 줄이 탭 또는 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자는 해당 줄을 새 로그 메시지로 처리합니다. 공백 문자로 시작하는 경우 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성 요소가 스택 추적과 같이 여러 줄에 걸쳐 있는 메시지를 분할하지 않도록 하기 위해서입니다.

periodicUploadIntervalSec

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간(초)입니다.

기본값: 300(5분)

최소값: 0.000001(1마이크로초)

Example예: 구성 병합 업데이트

다음 구성 예제에서는 시스템 로그와 com.example.HelloWorld 구성 요소 로그를 CloudWatch Logs에 업로드하도록 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 로그에 대한 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

systemLogsConfiguration

(선택 사항) [Greengrass nucleus](#) 및 [플러그인 구성 요소의](#) 로그를 포함하는 AWS IoT Greengrass 코어 소프트웨어 시스템 로그에 대한 구성입니다. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

uploadToCloudWatch

(선택 사항) 시스템 로그를 CloudWatch Logs에 업로드할 수 있습니다.

기본값: false

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 JSON 형식 로그를 출력하도록 [Greengrass nucleus 구성 요소](#)를 구성하는 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 [로깅 형식](#) 파라미터(JSON)에 logging.format을 지정합니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 Greengrass 시스템 로그 파일의 최대 총 크기입니다. Greengrass 시스템 로그 파일의 총 크기가 이 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

componentLogsConfiguration

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 목록입니다. 이 목록의 각 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 CloudWatch Logs에 업로드합니다.

각 객체에는 다음 정보가 포함되어 있습니다.

componentName

이 로그 구성에 대한 구성 요소 또는 애플리케이션의 이름입니다. Greengrass 구성 요소의 이름 또는 이 로그 그룹을 식별하는 다른 값을 지정할 수 있습니다.

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 GitHub의 [AWS IoT Greengrass 로깅 모듈](#) 리포지토리에서 찾을 수 있는 특정 JSON 형식을 사용하는 경우에만 적용됩니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기입니다. 이 구성 요소의 로그 파일의 총 크기가 이 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 관련이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트

- MB – 메가바이트
- GB – 기가바이트

기본값: KB

logFileDirectoryPath

(선택 사항) 이 구성 요소의 로그 파일이 포함된 폴더의 경로입니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

기본값: */greengrass/v2/logs*.

logFileRegex

(선택 사항) 구성 요소 또는 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규식입니다. 로그 관리자 구성 요소는 이 정규식을 사용하여 logFileDirectoryPath의 폴더에서 로그 파일을 식별합니다.

표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에는 이 파라미터를 지정할 필요가 없습니다.

구성 요소 또는 애플리케이션에서 로그 파일을 교체하는 경우 교체된 로그 파일 이름과 일치하는 정규식을 지정합니다. 예를 들어 Hello World 애플리케이션의 로그를 업로드하려면 **hello_world\\\\w*.log**를 지정할 수 있습니다. **\\\\w*** 패턴은 영숫자와 밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일과 일치합니다. 이 예제에서는 로그 관리자가 다음 로그 파일을 업로드합니다.

- hello_world.log - Hello World 애플리케이션의 최신 로그 파일입니다.
- hello_world_2020_12_15_17_0.log – Hello World 애플리케이션의 이전 로그 파일입니다.

기본값: *componentName\\\\w*.log*, 여기서 *componentName*은 이 로그 구성에 대한 구성 요소의 이름입니다.

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

multiLineStartPattern

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규식입니다. 정규식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 새 줄을 이전 줄의 로그 메시지에 추가합니다.

기본적으로 로그 관리자 구성 요소는 줄이 탭 또는 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자는 해당 줄을 새 로그 메시지로 처리합니다. 공백 문자로 시작하는 경우 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성 요소가 스택 추적과 같이 여러 줄에 걸쳐 있는 메시지를 분할하지 않도록 하기 위해서입니다.

periodicUploadIntervalSec

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간(초)입니다.

기본값: 300(5분)

최소값: 0.000001(1마이크로초)

Example예: 구성 병합 업데이트

다음 구성 예제에서는 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 CloudWatch Logs에 업로드하도록 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 로그에 대한 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

systemLogsConfiguration

(선택 사항) AWS IoT Greengrass 코어 소프트웨어 시스템 로그의 구성입니다. 로그 관리자 구성 요소가 시스템 로그를 관리할 수 있도록 하려면 이 구성을 지정합니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

uploadToCloudWatch

(선택 사항) 시스템 로그를 CloudWatch Logs에 업로드할 수 있습니다.

기본값: false

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 JSON 형식 로그를 출력하도록 [Greengrass nucleus 구성 요소](#)를 구성하는 경우에만 적용됩니다. JSON 형식 로그를 활성화하려면 [로깅 형식](#) 파라미터(JSON)에 `logging.format`을 지정합니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) `diskSpaceLimitUnit`에서 지정하는 단위로 Greengrass 시스템 로그 파일의 최대 총 크기입니다. Greengrass 시스템 로그 파일의 총 크기가 최대 총 크기를 초

과하면 AWS IoT Greengrass 코어 소프트웨어는 가장 오래된 Greengrass 시스템 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 동일합니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 최대 총 Greengrass 시스템 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

deleteLogFileAfterCloudUpload

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: false

componentLogsConfiguration

(선택 사항) 코어 디바이스의 구성 요소에 대한 로그 구성 목록입니다. 이 목록의 각 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다. 로그 관리자 구성 요소는 이러한 구성 요소 로그를 CloudWatch Logs에 업로드합니다.

각 객체에는 다음 정보가 포함되어 있습니다.

componentName

이 로그 구성에 대한 구성 요소 또는 애플리케이션의 이름입니다. Greengrass 구성 요소의 이름 또는 이 로그 그룹을 식별하는 다른 값을 지정할 수 있습니다.

minimumLogLevel

(선택 사항) 업로드할 로그 메시지의 최소 수준입니다. 이 최소 수준은 이 구성 요소의 로그가 GitHub의 [AWS IoT Greengrass 로깅 모듈](#) 리포지토리에서 찾을 수 있는 특정 JSON 형식을 사용하는 경우에만 적용됩니다.

여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- DEBUG

- INFO
- WARN
- ERROR

기본값: INFO

diskSpaceLimit

(선택 사항) diskSpaceLimitUnit에서 지정하는 단위로 이 구성 요소에 대한 모든 로그 파일의 최대 총 크기입니다. 이 구성 요소의 로그 파일의 총 크기가 최대 총 크기를 초과하면 AWS IoT Greengrass 코어 소프트웨어는 이 구성 요소의 가장 오래된 로그 파일을 삭제합니다.

이 파라미터는 [Greengrass nucleus 구성 요소의 로그 크기 제한](#) 파라미터 (totalLogsSizeKB)와 관련이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 두 값 중 최소값을 이 구성 요소의 최대 총 로그 크기로 사용합니다.

diskSpaceLimitUnit

(선택 사항) diskSpaceLimit의 단위입니다. 다음 옵션 중 하나를 선택합니다.

- KB – 킬로바이트
- MB – 메가바이트
- GB – 기가바이트

기본값: KB

logFileDirectoryPath

이 구성 요소의 로그 파일이 포함된 폴더의 경로입니다.

Greengrass 구성 요소의 로그를 업로드하려면 **/greengrass/v2/logs**를 지정하고 **/greengrass/v2**를 Greengrass 루트 폴더로 바꿉니다.

logFileRegex

구성 요소 또는 애플리케이션이 사용하는 로그 파일 이름 형식을 지정하는 정규식입니다. 로그 관리자 구성 요소는 이 정규식을 사용하여 logFileDirectoryPath의 폴더에서 로그 파일을 식별합니다.

Greengrass 구성 요소의 로그를 업로드하려면 교체된 로그 파일 이름과 일치하는 정규식을 지정합니다. 예를 들어 Hello World 구성 요소의 로그를 업로드하려면 **com.example.HelloWorld\\w*.log**를 지정할 수 있습니다. \\w* 패턴은 영숫자와

밑줄을 포함하는 0개 이상의 단어 문자와 일치합니다. 이 정규식은 이름에 타임스탬프가 있거나 없는 로그 파일과 일치합니다. 이 예제에서는 로그 관리자가 다음 로그 파일을 업로드합니다.

- `com.example.HelloWorld.log` - Hello World 구성 요소의 최신 로그 파일입니다.
- `com.example.HelloWorld_2020_12_15_17_0.log` - Hello World 구성 요소의 이전 로그 파일입니다. Greengrass nucleus는 로그 파일에 교체 타임스탬프를 추가합니다.

`deleteLogFileAfterCloudUpload`

(선택 사항) 로그 관리자 구성 요소가 로그를 CloudWatch Logs에 업로드한 후 로그 파일을 삭제할 수 있습니다.

기본값: `false`

`multiLineStartPattern`

(선택 사항) 새 줄의 로그 메시지가 새 로그 메시지인 경우를 식별하는 정규식입니다. 정규식이 새 줄과 일치하지 않는 경우 로그 관리자 구성 요소는 새 줄을 이전 줄의 로그 메시지에 추가합니다.

기본적으로 로그 관리자 구성 요소는 줄이 탭 또는 공백과 같은 공백 문자로 시작하는지 확인합니다. 그렇지 않으면 로그 관리자는 해당 줄을 새 로그 메시지로 처리합니다. 공백 문자로 시작하는 경우 해당 줄이 현재 로그 메시지에 추가됩니다. 이 동작은 로그 관리자 구성 요소가 스택 추적과 같이 여러 줄에 걸쳐 있는 메시지를 분할하지 않도록 하기 위해서입니다.

`periodicUploadIntervalSec`

(선택 사항) 로그 관리자 구성 요소가 업로드할 새 로그 파일을 확인하는 기간(초)입니다.

기본값: `300`(5분)

최소값: `0.000001`(1마이크로초)

Example예: 구성 병합 업데이트

다음 구성 예제에서는 시스템 로그와 `com.example.HelloWorld` 구성 요소 로그를 CloudWatch Logs에 업로드하도록 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"
}

```

사용법

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 이름에는 다음 변수를 사용합니다.

- **componentType** - 구성 요소의 유형으로, 다음 중 하나일 수 있습니다.
 - **GreengrassSystemComponent** - 이 로그 그룹에는 Greengrass nucleus와 동일한 JVM에서 실행되는 nucleus 및 플러그인 구성 요소에 대한 로그가 포함됩니다. 이 구성 요소는 [Greengrass nucleus](#)의 일부입니다.
 - **UserComponent** - 이 로그 그룹에는 디바이스의 일반 구성 요소, Lambda 구성 요소 및 기타 애플리케이션에 대한 로그가 포함됩니다. 이 구성 요소는 Greengrass nucleus의 일부가 아닙니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

- **region** - 코어 디바이스가 사용하는 AWS 리전입니다.
- **componentName** - 구성 요소의 이름입니다. 시스템 로그의 경우 이 값은 System입니다.

로그 스트림 이름

```
/date/thing/thingName
```

로그 스트림 이름에는 다음 변수를 사용합니다.

- **date** - 와 같은 로그 날짜입니다(예: 2020/12/15). 로그 관리자 구성 요소에서는 yyyy/MM/dd 형식을 사용합니다.
- **thingName** - 코어 디바이스 이름의 이름입니다.

Note

사물 이름에 콜론(:)이 포함된 경우 로그 관리자에서는 콜론을 더하기(+)로 바꿉니다.

2.0.x

로그 그룹 이름

```
/aws/greengrass/componentType/region/componentName
```

로그 그룹 이름에는 다음 변수를 사용합니다.

- **componentType** - 구성 요소의 유형으로, 다음 중 하나일 수 있습니다.
 - **GreengrassSystemComponent** - 이 구성 요소는 [Greengrass nucleus](#)의 일부입니다.
 - **UserComponent** - 이 구성 요소는 Greengrass nucleus의 일부가 아닙니다. 로그 관리자는 디바이스의 Greengrass 구성 요소 및 기타 애플리케이션에 이 유형을 사용합니다.
- **region** - 코어 디바이스가 사용하는 AWS 리전입니다.
- **componentName** - 구성 요소의 이름입니다. 시스템 로그의 경우 이 값은 System입니다.

로그 스트림 이름

```
/date/deploymentTargets/thingName
```

로그 스트림 이름에는 다음 변수를 사용합니다.

- `date` - 와 같은 로그 날짜입니다(예: 2020/12/15). 로그 관리자 구성 요소에서는 `yyyy/MM/dd` 형식을 사용합니다.
- `deploymentTargets` - 배포에 구성 요소를 포함하는 사물입니다. 로그 관리자 구성 요소는 각 대상을 슬래시로 구분합니다. 구성 요소가 로컬 배포의 결과로 코어 디바이스에서 실행되는 경우 이 값은 `LOCAL_DEPLOYMENT`입니다.

MyGreengrassCore라는 코어 디바이스가 있고 코어 디바이스에 다음 두 가지 배포가 있는 예를 생각해 보세요.

- 코어 디바이스 MyGreengrassCore를 대상으로 하는 배포
- 코어 디바이스가 포함된 MyGreengrassCoreGroup이라는 사물 그룹을 대상으로 하는 배포

이 코어 디바이스의 `deploymentTargets`는 `thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup`입니다.

- `thingName` - 코어 디바이스 이름의 이름입니다.

로그 항목의 형식

Greengrass nucleus는 로그 파일을 문자열 또는 JSON 형식으로 씁니다. 시스템 로그의 경우 `logging` 항목의 `format` 필드를 설정하여 형식을 제어합니다. `logging` 항목은 Greengrass nucleus 구성 요소의 구성 파일에서 찾을 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성](#)을 참조하세요.

텍스트 형식은 자유 형식이며 모든 문자열이 허용됩니다. 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 로그를 조작하려면 JSON 형식을 사용해야 합니다. 다음 예제에서는 JSON 형태를 간략하게 보여줍니다.

```
{
  "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 `minimumLogLevel` 구성 옵션을 사용할 수 있습니다. 이 옵션을 사용하려면 구성 요소가 로그 항목을 JSON 형식으로 써야 합니다. 시스템 로그 파일과 동일한 형식을 사용해야 합니다.

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.3.11	버그 수정 및 개선 사항 업로드된 로그 파일의 오래된 정보로 인해 Log Manager 런타임 구성이 무기한 증가하는 문제를 해결합니다.
2.3.10	새로운 기능 - 로그 업로드 이벤트 세부 정보를 로컬 트랜잭션 로그(<code>config.tllog</code>)에 저장하는 빈도를 제어할 수 있는 새로운 구성 키(<code>updateToTlogIntervalSec</code>)가 추가됩니다. 버그 수정 및 개선 사항 - 소켓 연결 오류 발생 시 CloudWatch 클라이언트를 새로 고치도록 로그 관리자 기능이 개선됩니다.
2.3.9	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.3.8	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.3.7	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.3.6	버그 수정 및 개선 사항 특정 오류에 대한 로그 수준이 조정됩니다.
2.3.5	개선 사항 로그 업로드 속도가 개선됩니다. Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.3.4	버그 수정 및 개선 사항 <code>periodicUploadIntervalSec</code> 파라미터를 분수 값으로 설정할 수 있도록 지원을 추가합니다. 최소값은 1마이크로초입니다. - 로그 관리자가 CloudWatch <code>putLogEvents</code> 제한을 준수하지 않는 문제를 해결합니다.
2.3.3	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.3.2	버그 수정 및 개선 사항 - 로그 파일이 업로드되기 전에 삭제되지 않도록 공간 관리가 개선됩니다. - 캐시 관리 문제가 해결됩니다. - 사소한 추가 버그 수정 및 개선 사항입니다.
2.3.1	버그 수정 및 개선 사항 - 활성 로그 파일이 여러 개 있는 파일 그룹을 대상으로 하는 경우 CloudWatch에 중복 항목이 업로드되는 문제를 해결합니다. - 사소한 추가 버그 수정 및 개선 사항입니다.
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 기다리는 대신 활성 로그 파일을 처리하고 직접 업로드하여 로그 지연이 감소합니다. 버그 수정 및 개선 사항 - 고유한 이름으로 파일을 교체할 때 로그 교체 지원이 개선됩니다. - 사소한 추가 버그 수정 및 개선 사항입니다.
2.2.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.2.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.2.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.2.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.2.4	버그 수정 및 개선 사항 • 잘못된 구성을 처리할 때의 안정성이 개선됩니다. • 사소한 추가 수정 및 개선 사항
2.2.3	버그 수정 및 개선 사항 • 구성 요소가 다시 시작되거나 오류가 발생하는 특정 시나리오에서 안정성이 개선됩니다. • 특정 시나리오에서 대용량 로그 메시지와 대용량 로그 파일이 업로드되지 않는 문제가 해결됩니다. • 이 구성 요소에서 구성 재설정 업데이트가 처리되는 방식과 관련된 문제가 해결됩니다. • <code>null diskSpaceLimit</code> 구성 값으로 인해 구성 요소가 배포되지 않는 문제를 해결합니다.
2.2.2	버그 수정 및 개선 사항 • 256킬로바이트보다 큰 로그 메시지에 대한 지원을 추가합니다. 로그 관리자 구성 요소는 이러한 대용량 로그 메시지를 로그 이벤트 타임스탬프가 동일한 여러 메시지로 분할합니다.
2.2.1	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.2.0	새로운 기능 • 구성 요소 로그 구성에 대한 맵 형식을 지원하는 <code>component LogsConfigurationMap</code> 구성 파라미터가 추가됩니다. 맵의 각 <code>componentName</code> 객체는 구성 요소 또는 애플리케이션에 대한 로그 구성을 정의합니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	버그 수정 및 개선 사항 • 경우에 따라 시스템 로그 구성이 업데이트되지 않는 문제를 해결합니다.

버전	변경 사항
2.1.0	버그 수정 및 개선 사항 - 표준 출력(stdout) 및 표준 오류(stderr)로 인쇄되는 Greengrass 구성 요소에 대해 작동하는 logFileDirectoryPath 및 logFileRegex 에 기본값을 사용합니다. - CloudWatch Logs에 로그를 업로드할 때 구성된 네트워크 프록시를 통해 트래픽을 올바르게 라우팅합니다. - 로그 스트림 이름에서 콜론 문자(:)가 올바르게 처리됩니다. CloudWatch Logs 로그 스트림 이름에 콜론을 포함할 수 없습니다. - 로그 스트림에서 사물 그룹 이름을 제거하여 로그 스트림 이름을 간소화합니다. - 정상적인 동작 중에 인쇄되는 오류 로그 메시지가 제거됩니다.
2.0.x	초기 버전입니다.

기계 학습 구성 요소

AWS IoT Greengrass 는 Amazon SageMaker AI로 훈련된 모델 또는 Amazon S3에 저장된 사전 훈련된 자체 모델을 사용하여 기계 [학습 추론을 수행하기 위해 지원되는 디바이스에 배포할 수 있는 다음과 같은 기계](#) 학습 구성 요소를 제공합니다.

AWS 는 다음과 같은 기계 학습 구성 요소 범주를 제공합니다.

- 모델 구성 요소 - 기계 학습 모델을 Greengrass 아티팩트로 포함합니다.
- 런타임 구성 요소 - Greengrass 코어 디바이스에서 기계 학습 프레임워크 및 그 종속성을 설치하는 스크립트를 포함합니다.
- 추론 구성 요소 - 추론 코드를 포함하고 기계 학습 프레임워크를 설치하고 사전 훈련된 기계 학습 모델을 다운로드하기 위한 구성 요소 종속성을 포함합니다.

AWS제공 기계 학습 구성 요소에서 샘플 추론 코드와 사전 훈련된 모델을 사용하여 DLR 및 TensorFlow Lite를 사용하여 이미지 분류 및 객체 감지를 수행할 수 있습니다. Amazon S3에 저장된 자체 모델을 사용하여 사용자 지정 기계 학습 추론을 수행하거나 다른 기계 학습 프레임워크를 사용하려면 이러한 퍼블릭 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 기계 학습 구성 요소를 생성할 수 있습니다. 자세한 내용은 [기계 학습 구성 요소 사용자 지정](#) 단원을 참조하십시오.

AWS IoT Greengrass에는 Greengrass 코어 디바이스에서 SageMaker AI Edge Manager 에이전트의 설치 및 수명 주기를 관리하는 AWS제공 구성 요소도 포함되어 있습니다. SageMaker AI Edge Manager를 사용하는 경우 코어 디바이스에서 Amazon SageMaker AI Neo 컴파일 모델을 직접 사용할 수 있습니다. 자세한 내용은 [Greengrass 코어 디바이스에서 Amazon SageMaker AI Edge Manager 사용](#) 단원을 참조하십시오.

다음 표에는에서 사용할 수 있는 기계 학습 구성 요소가 나열되어 있습니다 AWS IoT Greengrass.

Note

여러 AWS제공 구성 요소는 Greengrass nucleus의 특정 마이너 버전에 따라 달라집니다. 이 종속성 때문에 Greengrass nucleus를 새로운 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 달라지는 특정 nucleus 버전에 대한 내용은 해당 구성 요소 주제를 참조하세요. nucleus 업데이트에 대한 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#) 섹션을 참조하세요.

구성 요소에 일반 및 Lambda 구성 요소 유형이 둘 다 있으면 구성 요소의 현재 버전은 일반 유형이고 구성 요소의 이전 버전은 Lambda 유형입니다.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
SageMaker AI Edge Manager	Greengrass 코어 디바이스에 Amazon SageMaker AI Edge Manager 에이전트를 배포합니다.	일반	Linux, Windows	아니요
DLR 이미지 분류	DLR 이미지 분류 모델 저장소와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
	이미지 분류 모델을 다운로드 하고, 지원되는 디바이스에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.			
DLR 객체 감지	DLR 객체 감지 모델 저장소와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 객체 감지 모델을 다운로드하고, 지원되는 디바이스에서 객체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 이미지 분류 모델 저장소	샘플 ResNet-50 이미지 분류 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
DLR 객체 감지 모델 저장소	샘플 YOLOv3 객체 감지 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 런타임	Greengrass 코어 디바이스에 DLR 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트를 포함하는 런타임 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow Lite 이미지 분류	TensorFlow Lite 이미지 분류 모델 저장소와 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 디바이스에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow Lite 객체 감지	TensorFlow Lite 객체 감지 모델 저장소와 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 객체 감지 모델을 다운로드하고, 지원되는 디바이스에서 객체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow Lite 이미지 분류 모델 저장소	샘플 MobileNet v1 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow Lite 객체 감지 모델 저장소	샘플 SSD(Single Shot Detection) MobileNet 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow Lite 런타임	Greengrass 코어 디바이스에 TensorFlow Lite 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트를 포함하는 런타임 구성 요소입니다.	일반	Linux, Windows	아니요

SageMaker AI Edge Manager

Important

SageMaker AI Edge Manager는 2024년 4월 26일에 중단되었습니다. 엣지 디바이스에 모델을 계속 배포하는 방법에 대한 자세한 내용은 [SageMaker AI Edge Manager 수명 종료](#)를 참조하세요.

Amazon SageMaker AI Edge Manager 구성 요소(`aws.greengrass.SageMakerEdgeManager`)는 SageMaker AI Edge Manager 에이전트 바이너리를 설치합니다.

SageMaker AI Edge Manager는 엣지 디바이스에 대한 모델 관리 기능을 제공하므로 엣지 디바이스 플랫폼에서 기계 학습 모델을 최적화, 보호, 모니터링, 유지 관리할 수 있습니다. SageMaker AI Edge Manager 구성 요소는 코어 디바이스에 SageMaker AI Edge Manager 에이전트의 수명 주기를 설치하고 관리합니다. 또한 SageMaker AI Edge Manager를 사용하여 Greengrass 코어 디바이스에서 SageMaker AI Neo 컴파일 모델을 모델 구성 요소로 패키징하고 사용할 수 있습니다. 코어 디바이스에서 SageMaker AI Edge Manager 에이전트를 사용하는 방법에 대한 자세한 내용은 [Greengrass 코어 디바이스에서 Amazon SageMaker AI Edge Manager 사용](#) 섹션을 참조하세요.

SageMaker AI Edge Manager 구성 요소 v1.3.x는 Edge Manager 에이전트 바이너리 v1.20220822.836f3023을 설치합니다. Edge Manager 에이전트 바이너리 버전에 대한 자세한 내용은 [Edge Manager 에이전트](#)를 참조하세요.

Note

SageMaker AI Edge Manager 구성 요소는 AWS 리전다음에서만 사용할 수 있습니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(오리건)
- EU(프랑크푸르트)
- EU(아일랜드)
- 아시아 태평양(도쿄)

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2, Debian 기반 Linux 플랫폼(x86_64 또는 Armv8) 또는 Windows(x86_64)에서 실행되는 Greengrass 코어 디바이스. 없으면 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요.
- [Python](#) 3.6 이상(사용하는 Python 버전에 대한 pip 포함)이 코어 디바이스에 설치되어 있어야 합니다.
- 다음과 같이 구성된 [Greengrass 디바이스 역할](#):
 - 다음 IAM 정책 예제와 같이 `credentials.iot.amazonaws.com` 및 `sagemaker.amazonaws.com`에서 역할을 수임하도록 허용하는 신뢰 관계.

```
{
  "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"
  }
]
}

```

- [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM 관리형 정책.
- 다음 IAM 정책 예제와 같은 s3:PutObject 작업.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}

```

- Greengrass 코어 디바이스 AWS 리전 와 동일한 AWS 계정 밑에서 생성된 Amazon S3 버킷입니다. SageMaker AI Edge Manager에는 엣지 디바이스 플릿을 생성하고 디바이스에서 추론을 실행하지 않는 샘플 데이터를 저장하기 위한 S3 버킷이 필요합니다. S3 버킷 생성에 대한 자세한 내용은 [Amazon S3 시작하기](#)를 참조하세요.
- Greengrass 코어 디바이스와 동일한 AWS IoT 역할 별칭을 사용하는 SageMaker AI 엣지 디바이스 플릿입니다. 자세한 내용은 [엣지 디바이스 플릿 생성](#) 단원을 참조하십시오.

- SageMaker AI Edge 디바이스 플릿에 엣지 디바이스로 등록된 Greengrass 코어 디바이스. 엣지 디바이스 이름은 코어 디바이스의 AWS IoT 사물 이름과 일치해야 합니다. 자세한 내용은 [Greengrass 코어 디바이스 등록](#) 단원을 참조하십시오.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	Port	필수	설명
edge.sagemaker. <i>region</i> .amazonaws.com	443	예	디바이스 등록 상태를 확인하고 지표를 SageMaker AI로 전송합니다.
*.s3.amazonaws.com	443	예	지정한 S3 버킷에 캡처 데이터를 업로드합니다. *를 데이터를 업로드하는 각 버킷의 이름으로 바꿀 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

1.3.5 and 1.3.6

다음 표에는 이 구성 요소의 버전 1.3.5 및 1.3.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.13.0$	소프트
토큰 교환 서비스	$\geq 0.0.0$	하드

1.3.4

다음 표에는 이 구성 요소의 버전 1.3.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.12.0$	소프트
토큰 교환 서비스	$\geq 0.0.0$	하드

1.3.3

다음 표에는 이 구성 요소의 버전 1.3.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.11.0$	소프트
토큰 교환 서비스	$\geq 0.0.0$	하드

1.3.2

다음 표에는 이 구성 요소의 버전 1.3.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=0.0.0	하드

1.0.3

다음 표에는 이 구성 요소의 버전 1.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트
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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트
토큰 교환 서비스	>=0.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

Note

이 섹션에서는 구성 요소에서 설정하는 구성 파라미터에 대해 설명합니다. 해당 SageMaker AI Edge Manager 구성에 대한 자세한 내용은 Amazon SageMaker AI 개발자 안내서의 [Edge Manager 에이전트](#)를 참조하세요.

DeviceFleetName

Greengrass 코어 디바이스를 포함한 SageMaker AI Edge Manager 디바이스 플릿의 이름입니다.

이 구성 요소를 배포하는 경우 구성 업데이트에서 이 파라미터의 값을 지정해야 합니다.

BucketName

캡처된 추론 데이터를 업로드하는 S3 버킷의 이름. 버킷 이름이 sagemaker 문자열을 포함해야 합니다.

CaptureDataDestination을 Cloud로 설정하거나 CaptureDataPeriodicUpload를 true로 설정했다면 이 구성 요소를 배포하는 경우 구성 업데이트에서 이 파라미터의 값을 지정해야 합니다.

Note

캡처 데이터는 향후 분석을 위해 추론 입력, 추론 결과, 추가 추론 데이터를 S3 버킷 또는 로컬 디렉터리에 업로드하는 데 사용하는 SageMaker AI 기능입니다. SageMaker AI Edge Manager에서 캡처 데이터를 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker AI 개발자 안내서의 [모델 관리](#)를 참조하세요.

CaptureDataBatchSize

(선택 사항) 에이전트가 처리하는 캡처 데이터 요청의 배치 크기. 이 값은 CaptureDataBufferSize에서 지정한 버퍼 크기보다 작아야 합니다. 버퍼 크기의 절반을 초과하지 않는 것이 좋습니다.

에이전트는 버퍼에 있는 요청의 수가 `CaptureDataBatchSize` 수를 충족하거나 `CaptureDataPushPeriodSeconds` 간격이 경과하는 경우 둘 중 먼저 발생하는 요청 배치를 처리합니다.

기본값: 10

`CaptureDataBufferSize`

(선택 사항) 버퍼에 저장된 캡처 데이터 요청의 최대 수.

기본값: 30

`CaptureDataDestination`

(선택 사항) 캡처된 데이터를 저장하는 대상. 이 파라미터의 값은 다음과 같을 수 있습니다.

- Cloud - `BucketName`에서 지정한 S3 버킷에 캡처된 데이터를 업로드합니다.
- Disk - 캡처된 데이터를 구성 요소의 작업 디렉터리에 씁니다.

또한 Disk를 지정하는 경우 `CaptureDataPeriodicUpload`를 `true`로 설정하여 캡처된 데이터를 S3 버킷에 주기적으로 업로드하도록 선택할 수도 있습니다.

기본값: Cloud

`CaptureDataPeriodicUpload`

(선택 사항) 캡처된 데이터를 주기적으로 업로드할지 여부를 지정하는 문자열 값. 지원되는 값은 `true` 및 `false`입니다.

`CaptureDataDestination`을 Disk로 설정했고 에이전트가 캡처된 데이터를 S3 버킷에 주기적으로 업로드하도록 하려는 경우 이 파라미터를 `true`로 설정합니다.

기본값: false

`CaptureDataPeriodicUploadPeriodSeconds`

(선택 사항) SageMaker AI Edge Manager 에이전트가 캡처된 데이터를 S3 버킷에 업로드하는 간격(초). `CaptureDataPeriodicUpload`를 `true`로 설정한 경우 이 파라미터를 사용합니다.

기본값: 8

`CaptureDataPushPeriodSeconds`

(선택 사항) SageMaker AI Edge Manager 에이전트가 버퍼에서 캡처 데이터 요청의 배치를 처리하는 간격(초).

에이전트는 버퍼에 있는 요청의 수가 `CaptureDataBatchSize` 수를 충족하거나 `CaptureDataPushPeriodSeconds` 간격이 경과하는 경우 둘 중 먼저 발생하는 요청 배치를 처리합니다.

기본값: 4

`CaptureDataBase64EmbedLimit`

(선택 사항) SageMaker AI Edge Manager 에이전트가 업로드하는 캡처된 데이터의 최대 크기(바이트).

기본값: 3072

`FolderPrefix`

(선택 사항) 에이전트가 캡처된 데이터를 쓰는 폴더의 이름. `CaptureDataDestination`을 Disk로 설정하는 경우 에이전트는 `CaptureDataDiskPath`에서 지정한 디렉터리에 폴더를 생성합니다. `CaptureDataDestination`을 Cloud로 설정하거나 `CaptureDataPeriodicUpload`를 true로 설정한 경우 에이전트가 S3 버킷에 폴더를 생성합니다.

기본값: sme-capture

`CaptureDataDiskPath`

이 기능은 SageMaker AI Edge Manager 구성 요소의 v1.1.0 이상 버전에서 사용할 수 있습니다.

(선택 사항) 에이전트가 캡처된 데이터 폴더를 생성하는 폴더의 경로. `CaptureDataDestination`을 Disk로 설정하는 경우 에이전트가 이 디렉터리에 캡처된 데이터 폴더를 생성합니다. 이 값을 지정하지 않으면 에이전트가 구성 요소의 작업 디렉터리에 캡처된 데이터 폴더를 생성합니다. `FolderPrefix` 파라미터를 사용하여 캡처된 데이터 폴더의 이름을 지정합니다.

기본값: `/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager/capture`

`LocalDataRootPath`

이 기능은 SageMaker AI Edge Manager 구성 요소의 v1.2.0 이상 버전에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소가 코어 디바이스에 다음 데이터를 저장하는 경로:

- `DbEnable`을 true로 설정하는 경우 런타임 데이터에 대한 로컬 데이터베이스.
- `DeploymentEnable`을 true로 설정하는 경우 이 구성 요소가 자동으로 다운로드하는 SageMaker AI Neo 컴파일 모델.

기본값: `/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager`

DbEnable

(선택 사항) 이 구성 요소를 활성화하면 구성 요소가 실패하거나 디바이스 전원이 꺼지는 경우를 대비하여 로컬 데이터베이스에 런타임 데이터를 저장하여 데이터를 보존할 수 있습니다.

이 데이터베이스에는 코어 디바이스의 파일 시스템에 5MB의 스토리지가 필요합니다.

기본값: `false`

DeploymentEnable

이 기능은 SageMaker AI Edge Manager 구성 요소의 v1.2.0 이상 버전에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소를 활성화하여 Amazon S3에 업로드한 모델에서 SageMaker AI Neo 컴파일 모델을 자동으로 검색할 수 있습니다. Amazon S3에 새로운 모델을 업로드한 후 SageMaker AI Studio 또는 SageMaker AI API를 사용하여 이 코어 디바이스에 새로운 모델을 배포합니다. 이 기능을 활성화하면 AWS IoT Greengrass 배포를 생성할 필요 없이 코어 디바이스에 새로운 모델을 배포할 수 있습니다.

Important

이 기능을 사용하려면 DbEnable을 `true`로 설정해야 합니다. 이 기능은 로컬 데이터베이스를 사용하여 AWS 클라우드에서 검색하는 모델을 추적합니다.

기본값: `false`

DeploymentPollInterval

이 기능은 SageMaker AI Edge Manager 구성 요소의 v1.2.0 이상 버전에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소가 다운로드할 새로운 모델을 확인하는 데 걸리는 시간(분). 이 옵션은 DeploymentEnable을 `true`로 설정하는 경우 적용됩니다.

기본값: 1440(1일)

DLRBackendOptions

이 기능은 SageMaker AI Edge Manager 구성 요소의 v1.2.0 이상 버전에서 사용할 수 있습니다.

(선택 사항) 이 구성 요소가 사용하는 DLR 런타임에서 설정할 DLR 런타임 플래그. 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 할당합니다. 디렉터리 크기가 이 제한을 초과하여 커지면 가장 적게 사용되는 캐시된 엔진이 삭제됩니다. 기본 값은 512MB입니다.

예를 들어 이 파라미터를 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 값. 지원되는 값은 `true` 및 `false`입니다.

기본값: `false`

UnixSocketName

(선택 사항) 코어 디바이스에서의 SageMaker AI Edge Manager 소켓 파일 설명자의 위치.

기본값: `/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example예: 구성 병합 업데이트

다음 예제 구성은 코어 디바이스가 *MyEdgeDeviceFleet*의 일부이고 에이전트가 캡처 데이터를 디바이스와 S3 버킷 모두에 쓰도록 지정합니다. 이 구성은 디버그 로깅도 활성화합니다.

```
{
  "DeviceFleetName": "MyEdgeDeviceFleet",
  "BucketName": "amzn-s3-demo-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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
1.3.6	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
1.3.5	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
1.3.4	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
1.3.3	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
1.3.2	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
1.3.1	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
1.3.0	새로운 기능 - TensorRT 캐시 디스크 크기 관리에 대한 지원을 추가합니다. - DLRBackendOptions 파라미터에 TVM_TENSORRT_CACHE_DISK_SIZE_MB 플래그(선택 사항)를 추가하여 디스크에 캐시된 모델의 크기 제한을 설정합니다. 개선 사항 - 개선된 예측 동시성을 제공합니다. 이렇게 하면 GPU와 같은 디바이스 액셀러레이터 엔진을 더 잘 활용할 수 있습니다.
1.2.0	새로운 기능 - 이 구성 요소에 대한 지원을 추가하여 Amazon S3에 업로드한 SageMaker AI Neo 컴파일 모델을 자동으로 검색합니다. 이 기능을 활성화하면 배포를 생성할 필요 없이 코어 디바이스에 새 모델을 AWS IoT Greengrass 배포할 수 있습니다. - 구성 요소가 실패하거나 디바이스의 전원이 손실되는 경우를 대비하여 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 nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
1.1.0	새로운 기능 - Amazon Linux 2를 실행하는 Greengrass 코어 디바이스에 대한 지원을 추가합니다. - 새로운 CaptureDataDiskPath 구성 파라미터를 추가합니다. 이 파라미터를 사용하여 디바이스에서 캡처된 데이터 폴더의 경로를 지정할 수 있습니다. 버그 수정 및 개선 사항 - Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
1.0.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
1.0.2	버그 수정 및 개선 사항 구성 요소 수명 주기에서 설치 스크립트를 업데이트합니다. 이 구성 요소를 배포하기 전에 코어 디바이스에 Python 버전에 대한 pip를 포함하여 Python 3.6 이상이 설치되어 있어야 합니다.
1.0.1	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
1.0.0	초기 버전입니다.

DLR 이미지 분류

DLR 이미지 분류 구성 요소(`aws.greengrass.DLRImageClassification`)에는 [딥 러닝 런타임](#)과 resnet-50 모델이 이미지 분류 추론 수행에 사용되는 샘플 추론 코드가 있습니다. 이 구성 요소에서는 DLR과 샘플 모델 다운로드에 변형 [DLR 이미지 분류 모델 저장소](#) 및 [DLR 런타임](#) 구성 요소가 종속성으로 사용됩니다.

사용자 지정 훈련된 DLR 모델과 함께 이 추론 구성 요소를 사용하려면 종속 모델 저장소 구성 요소의 [사용자 지정 버전을 생성](#)합니다. 자체 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 [사용자 지정 추론 구성 요소를 생성](#)할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
 - 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.13 and 2.1.14

다음 표에는 이 구성 요소의 버전 2.1.13 및 2.1.14에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.4 - 2.1.5

다음 표에는 이 구성 요소의 버전 2.1.4~2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트
DLR 이미지 분류 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	2.0.0 이하	소프트
DLR 이미지 분류 모델 저장소	2.0.0 이하	하드
DLR	1.3.0 이하	소프트

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.1.x

accessControl

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 허용하는 [권한 부여 정책](#)이 있는 객체입니다.

기본값:

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.DLRIImageClassification: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 경우에는 사용자 지정 주제 이름과 일치하도록 accessControl 파라미터의 resources 값도 수정해야 합니다.

기본값: ml/dlr/image-classification

Accelerator

사용하려는 액셀러레이터입니다. 지원되는 값은 cpu 및 gpu입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 액셀러레이션만 지원합니다. 다른 사용자 지정 모델에서 GPU 액셀러레이션을 사용하려면 [사용자 지정 모델 구성 요소를 생성](#)하여 퍼블릭 모델 구성 요소를 재정의합니다.

기본값: cpu

ImageDirectory

(선택 사항) 추론 구성 요소가 이미지를 읽는 디바이스의 폴더 경로입니다. 이 값은 읽기/쓰기 액세스 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

ImageName

(선택 사항) 추론 구성 요소가 예측하기 위한 입력으로 사용하는 이미지의 이름입니다. 구성 요소는 ImageDirectory에 지정된 폴더에서 이미지를 찾습니다. 기본적으로 구성 요소는 기본 이미지 디렉터리의 샘플 이미지를 사용합니다.는 jpeg, jpg, 및 이미지 형식을 AWS IoT Greengrass 지원합니다pngnpy.

기본값: `cat.jpeg`

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

InferenceInterval

(선택 사항) 추론 코드가 예측한 각 예측 사이의 시간(초)입니다. 샘플 추론 코드는 무기한으로 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 퍼블릭 모델 구성 요소에 사용되는 모델입니다. 사용자 지정 구성 요소로 퍼블릭 모델 구성 요소를 재정의하는 경우에만 이 파라미터를 수정합니다.

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 디바이스에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 true 및 false입니다.

이 값을 true로 설정하면 샘플 추론 코드가 디바이스의 카메라에 액세스하여 캡처된 이미지에서 로컬로 추론을 실행합니다. ImageName 및 ImageDirectory 파라미터의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 촬영한 이미지를 저장하는 위치에 대한 읽기/쓰기 액세스 권한이 있는지 확인합니다.

기본값: false

Note

이 구성 요소의 레시피를 보면 기본 구성에는 UseCamera 구성 파라미터가 표시되지 않습니다. 그러나 구성 요소를 배포하는 경우 [구성 병합 업데이트](#)에서 이 파라미터의 값을 수정할 수 있습니다.

UseCamera를 true로 설정하면 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 심볼릭 링크도 생성해야 합니다. 샘플 추론 구성 요소와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

2.0.x

MLRootPath

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값은 해당 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/work/variant.DLR/greengrass_ml`

기본값: `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

Accelerator

사용하려는 액셀러레이터입니다. 지원되는 값은 cpu 및 gpu입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 액셀러레이션만 지원합니다. 다른 사용자 지정 모델에서 GPU 액셀러레이션을 사용하려면 [사용자 지정 모델 구성 요소를 생성](#)하여 퍼블릭 모델 구성 요소를 재정의합니다.

기본값: cpu

ImageName

(선택 사항) 추론 구성 요소가 예측하기 위한 입력으로 사용하는 이미지의 이름입니다. 구성 요소는 ImageDirectory에 지정된 폴더에서 이미지를 찾습니다. 기본 위치는 `MLRootPath/images`입니다. jpeg, jpg, 및 이미지 형식을 AWS IoT Greengrass 지원합니다. png, npy.

기본값: cat.jpeg

InferenceInterval

(선택 사항) 추론 코드가 예측한 각 예측 사이의 시간(초)입니다. 샘플 추론 코드는 무기한으로 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 퍼블릭 모델 구성 요소에 사용되는 모델입니다. 사용자 지정 구성 요소로 퍼블릭 모델 구성 요소를 재정의하는 경우에만 이 파라미터를 수정합니다.

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.DLRIImageClassification.log
```

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 경로로 바꿉니다.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRIImageClassification.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRIImageClassification.log -Tail 10 -Wait
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.14	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.13	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.12	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.7	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	구성 요소가 모두 릴리스되었습니다 AWS 리전.
2.1.4	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다. 유럽(런던)(eu-west-2)에서는 이 버전을 사용할 수 없습니다.
2.1.3	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	새로운 기능 • 딥 러닝 런타임 v1.6.0을 사용합니다. • Armv8(AArch64) 플랫폼에서 샘플 이미지 분류 지원이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다. • 샘플 추론을 위해 카메라 통합이 활성화됩니다. 새로운 UseCamera 구성 파라미터를 사용하여 샘플 추론 코드가 Greengrass 코어 디바이스의 카메라에 액세스하고 캡처된 이미지에서 로컬로 추론을 실행할 수 있습니다. • AWS 클라우드에 추론 결과 게시에 대한 지원이 추가됩니다. 새로운 PublishResultsOnTopic 구성 파라미터를 사용하여 결과를 게시하려는 주제를 지정합니다. • 새로운 ImageDirectory 구성 파라미터가 추가되어 추론을 수행할 이미지의 사용자 지정 디렉터리를 지정할 수 있습니다. 버그 수정 및 개선 사항 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과가 작성됩니다. • AWS IoT Greengrass 코어 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 로깅합니다. • AWS IoT Device SDK 를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다.

버전	변경 사항
2.0.4	초기 버전입니다.

DLR 객체 감지

DLR 객체 감지 구성 요소(`aws.greengrass.DLRObjectDetection`)에는 [딥 러닝 런타임](#) 및 샘플 사전 훈련된 모델을 사용하여 객체 감지 추론을 수행하는 샘플 추론 코드가 있습니다. 이 구성 요소에서는 DLR과 샘플 모델 다운로드에 변형 [DLR 객체 감지 모델 저장소](#) 및 [DLR 런타임](#) 구성 요소가 종속성으로 사용됩니다.

사용자 지정 훈련된 DLR 모델과 함께 이 추론 구성 요소를 사용하려면 종속 모델 저장소 구성 요소의 [사용자 지정 버전을 생성](#)합니다. 자체 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 [사용자 지정 추론 구성 요소를 생성](#)할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
 - 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.13 and 2.1.14

다음 표에는 이 구성 요소의 버전 2.1.13 및 2.1.14에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.4 - 2.1.5

다음 표에는 이 구성 요소의 버전 2.1.4~2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트
DLR 객체 감지 모델 저장소	2.1.0 이하	하드
DLR	1.6.0 이하	하드

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	2.0.0 이하	소프트
DLR 객체 감지 모델 저장소	2.0.0 이하	하드
DLR	1.3.0 이하	소프트

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.1.x

accessControl

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 허용하는 [권한 부여 정책](#)이 있는 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 경우에는 사용자 지정 주제 이름과 일치하도록 `accessControl` 파라미터의 `resources` 값도 수정해야 합니다.

기본값: `ml/dlr/object-detection`

Accelerator

사용하려는 액셀러레이터입니다. 지원되는 값은 `cpu` 및 `gpu`입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 액셀러레이션만 지원합니다. 다른 사용자 지정 모델에서 GPU 액셀러레이션을 사용하려면 [사용자 지정 모델 구성 요소를 생성](#)하여 퍼블릭 모델 구성 요소를 재정의합니다.

기본값: `cpu`

ImageDirectory

(선택 사항) 추론 구성 요소가 이미지를 읽는 디바이스의 폴더 경로입니다. 이 값은 읽기/쓰기 액세스 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/packages/artifacts-unarchived/component-name/object_detection/sample_images/`

Note

`UseCamera` 값을 `true`로 설정하는 경우 이 구성 파라미터는 무시됩니다.

ImageName

(선택 사항) 추론 구성 요소가 예측하기 위한 입력으로 사용하는 이미지의 이름입니다. 구성 요소는 `ImageDirectory`에 지정된 폴더에서 이미지를 찾습니다. 기본적으로 구성 요소는 기

본 이미지 디렉터리의 샘플 이미지를 사용합니다.는 jpeg, jpg, 및 이미지 형식을 AWS IoT Greengrass 지원합니다pngnpy.

기본값: objects.jpg

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

InferenceInterval

(선택 사항) 추론 코드가 예측한 각 예측 사이의 시간(초)입니다. 샘플 추론 코드는 무기한으로 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 퍼블릭 모델 구성 요소에 사용되는 모델입니다. 사용자 지정 구성 요소로 퍼블릭 모델 구성 요소를 재정의하는 경우에만 이 파라미터를 수정합니다.

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 디바이스에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 true 및 false입니다.

이 값을 true로 설정하면 샘플 추론 코드가 디바이스의 카메라에 액세스하여 캡처된 이미지에서 로컬로 추론을 실행합니다. ImageName 및 ImageDirectory 파라미터의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 촬영한 이미지를 저장하는 위치에 대한 읽기/쓰기 액세스 권한이 있는지 확인합니다.

기본값: false

Note

이 구성 요소의 레시피를 보면 기본 구성에는 UseCamera 구성 파라미터가 표시되지 않습니다. 그러나 구성 요소를 배포하는 경우 [구성 병합 업데이트](#)에서 이 파라미터의 값을 수정할 수 있습니다.

UseCamera를 true로 설정하면 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 심볼릭 링크도 생성해야 합니다. 샘플 추론 구성 요소와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

2.0.x**MLRootPath**

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값은 해당 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 디바이스의 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 모델이 CPU 액셀러레이터에 대해서만 컴파일되기 때문에 현재 액셀러레이터에 대해 지원되는 유일한 값은 cpu입니다.

ImageName

(선택 사항) 추론 구성 요소가 예측하기 위한 입력으로 사용하는 이미지의 이름입니다. 구성 요소는 ImageDirectory에 지정된 폴더에서 이미지를 찾습니다. 기본 위치는 `MLRootPath/images`입니다. jpeg, jpg, 및 이미지 형식을 AWS IoT Greengrass 지원합니다. png, npy.

기본값: `objects.jpg`

InferenceInterval

(선택 사항) 추론 코드가 예측한 각 예측 사이의 시간(초)입니다. 샘플 추론 코드는 무기한으로 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 퍼블릭 모델 구성 요소에 사용되는 모델입니다. 사용자 지정 구성 요소로 퍼블릭 모델 구성 요소를 재정의하는 경우에만 이 파라미터를 수정합니다.

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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.1.14	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.13	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.12	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	구성 요소가 모두 릴리스되었습니다 AWS 리전.
2.1.4	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다. 유럽(런던)(eu-west-2)에서는 이 버전을 사용할 수 없습니다.
2.1.3	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	버그 수정 및 개선 사항 샘플 DLR 객체 감지 추론 결과에서 잘못된 경계 상자가 발생하는 이미지 크기 조정 문제를 해결합니다.

버전	변경 사항
2.1.1	새로운 기능 • 딥 러닝 런타임 v1.6.0을 사용합니다. • Armv8(AArch64) 플랫폼에서 샘플 객체 감지에 대한 지원이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다. • 샘플 추론을 위해 카메라 통합이 활성화됩니다. 새로운 UseCamera 구성 파라미터를 사용하여 샘플 추론 코드가 Greengrass 코어 디바이스의 카메라에 액세스하고 캡처된 이미지에서 로컬로 추론을 실행할 수 있습니다. • AWS 클라우드에 추론 결과 게시에 대한 지원이 추가됩니다. 새로운 PublishResultsOnTopic 구성 파라미터를 사용하여 결과를 게시하려는 주제를 지정합니다. • 새로운 ImageDirectory 구성 파라미터가 추가되어 추론을 수행할 이미지의 사용자 지정 디렉터리를 지정할 수 있습니다. 버그 수정 및 개선 사항 • 별도의 추론 파일 대신 구성 요소 로그 파일에 추론 결과가 작성됩니다. • AWS IoT Greengrass 코어 소프트웨어 로깅 모듈을 사용하여 구성 요소 출력을 로깅합니다. • AWS IoT Device SDK 를 사용하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다.
2.0.4	초기 버전입니다.

DLR 이미지 분류 모델 저장소

DLR 이미지 분류 모델 저장소는 Greengrass 아티팩트로 사전 훈련된 ResNet-50 모델이 있는 기계 학습 모델 구성 요소입니다. 이 구성 요소에 사용되는 사전 훈련된 모델은 [GluonCV Model Zoo](#)에서 가져오며, SageMaker AI Neo [딥 러닝 런타임](#)을 사용하여 컴파일합니다.

[DLR 이미지 분류](#) 추론 구성 요소는 이 구성 요소를 모델 소스의 종속성으로 사용합니다. 사용자 지정 훈련된 DLR 모델을 사용하려면 이 모델 구성 요소의 [사용자 지정 버전을 생성](#)하고 사용자 지정 모델을 구성 요소 아티팩트로 포함합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 생성할 수 있습니다.

Note

DLR 이미지 분류 모델 저장소 구성 요소의 이름은 버전에 따라 다릅니다. 버전 2.1.x 이상의 구성 요소 이름은 `variant.DLR.ImageClassification.ModelStore`입니다. 버전 2.0.x의 구성 요소 이름은 `variant.ImageClassification.ModelStore`입니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 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 nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드 할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는

의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.12 - 2.1.14

다음 표에는 이 구성 요소의 버전 2.1.12 및 2.1.13에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	2.0.0 이하	소프트

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.13	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.12	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	새로운 기능 - Windows 코어 디바이스에 대한 샘플 이미지 분류 모델이 추가됩니다. - Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.1	새로운 기능 ArmV8(AArch64) 플랫폼에 대한 샘플 ResNet-50 이미지 분류 모델이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다.
2.0.4	초기 버전입니다.

DLR 객체 감지 모델 저장소

DLR 객체 감지 모델 저장소는 Greengrass 아티팩트로 사전 훈련된 YOLOv3 모델이 있는 기계 학습 모델 구성 요소입니다. 이 구성 요소에 사용되는 샘플 모델은 [GluonCV Model Zoo](#)에서 가져오며, SageMaker AI Neo [딥 러닝 런타임](#)을 사용하여 컴파일합니다.

[DLR 객체 감지](#) 추론 구성 요소는 이 구성 요소를 모델 소스의 종속성으로 사용합니다. 사용자 지정 훈련된 DLR 모델을 사용하려면 이 모델 구성 요소의 [사용자 지정 버전을 생성](#)하고 사용자 지정 모델을 구성 요소 아티팩트로 포함합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 생성할 수 있습니다.

Note

DLR 객체 감지 모델 저장소 구성 요소의 이름은 버전에 따라 다릅니다. 버전 2.1.x 이상의 구성 요소 이름은 `variant.DLR.ObjectDetection.ModelStore`입니다. 버전 2.0.x의 구성 요소 이름은 `variant.ObjectDetection.ModelStore`입니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
 - 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.13 and 2.1.14

다음 표에는 이 구성 요소의 버전 2.1.13 및 2.1.14에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트

2.1.12

다음 표에는 이 구성 요소의 버전 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	2.0.0 이하	소프트

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.14	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.13	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.12	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Armv8(AArch64) 디바이스의 문제가 해결되는 CPU 모델이 추가됩니다.
2.1.5	새로운 기능 - Windows 코어 디바이스에 대한 샘플 객체 감지 모델이 추가됩니다. 버그 수정 및 개선 사항 - Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	새로운 기능 Armv8(AArch64) 플랫폼에 대한 샘플 YOLOv3 객체 감지 모델이 추가됩니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다.
2.0.4	초기 버전입니다.

DLR 런타임

DLR 런타임 구성 요소(variant.DLR)에는 디바이스의 가상 환경에서 [딥 러닝 런타임\(DLR\)](#)과 해당 종속성을 설치하는 스크립트가 있습니다. [DLR 이미지 분류](#) 및 [DLR 객체 감지](#) 구성 요소에서는 이 구성

요소를 DLR 설치용 종속성으로 사용합니다. 구성 요소 버전 1.6.x에서는 DLR v1.6.0을 설치하고, 구성 요소 버전 1.3.x에서는 DLR v1.3.0을 설치합니다.

상이한 런타임을 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 [사용자 지정 기계 학습 구성 요소](#)를 생성할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.6.x
- 1.3.x

유형

이 구성 요소는 일반 구성 요소(aws.greengrass.generic)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
 - 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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 요소에 코어 디바이스에서 사용하는 플랫폼에 따라 apt, yum, brew, pip 명령을 사용하여 패키지를 설치하는 설치 관리자 스크립트를 사용합니다. 이 구성 요소는 설치 관리자 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 디바이스가 설치를 위해 연결되는 패키지 인덱스 및 리포지토리의 엔드포인트를 식별해야 합니다.

이 구성 요소의 설치 스크립트에 필요한 엔드포인트를 식별하는 경우 다음 사항을 고려하세요.

- 엔드포인트는 코어 디바이스의 플랫폼에 따라 다릅니다. 예를 들어 Ubuntu를 실행하는 코어 디바이스는 yum 또는 brew 대신 apt를 사용합니다. 또한 동일한 패키지 인덱스를 사용하는 디바이스의 소스 목록은 다를 수 있으므로 서로 다른 리포지토리에서 패키지를 검색할 수 있습니다.
- 각 디바이스에는 패키지를 검색할 위치를 정의하는 자체 소스 목록이 있으므로 동일한 패키지 인덱스를 사용하는 여러 디바이스 간에 엔드포인트가 다를 수 있습니다.
- 엔드포인트는 시간이 지남에 따라 변경될 수 있습니다. 각 패키지 인덱스는 패키지를 다운로드하는 리포지토리의 URL을 제공하며 패키지 소유자는 패키지 인덱스가 제공하는 URL을 변경할 수 있습니다.

이 구성 요소에서 설치하는 종속성에 대한 자세한 내용과 설치 프로그램 스크립트를 비활성화하는 방법은 [UseInstaller](#) 구성 파라미터를 참조하세요.

기본 작업에 필요한 엔드포인트와 포트에 대한 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 섹션을 참조하세요.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

1.6.11 - 1.6.16

다음 표에는 이 구성 요소의 버전 1.6.11~1.6.16에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	소프트

1.6.10

다음 표에는 이 구성 요소의 버전 1.6.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

1.6.9

다음 표에는 이 구성 요소의 버전 1.6.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

1.6.8

다음 표에는 이 구성 요소의 버전 1.6.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

1.6.3

다음 표에는 이 구성 요소의 버전 1.6.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

1.6.2

다음 표에는 이 구성 요소의 버전 1.6.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

1.6.1

다음 표에는 이 구성 요소의 버전 1.6.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

1.3.x

다음 표에는 이 구성 요소의 버전 1.3.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	2.0.0 이하	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

MLRootPath

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값은 해당 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/work/variant.DLR/greengrass_ml`

WindowsMLRootPath

이 기능은 이 구성 요소의 v1.6.6 이상에 사용할 수 있습니다.

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Windows 코어 디바이스의 폴더 경로입니다. 이 값은 해당 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `C:\greengrass\v2\work\variant.DLR\greengrass_ml`

UseInstaller

(선택 사항) 이 구성 요소의 설치 프로그램 스크립트를 사용하여 DLR와 해당 종속성을 설치할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

DLR 설치에 사용자 지정 스크립트를 사용하려는 경우 또는 사전 구축된 Linux 이미지에 런타임 종속성을 포함하려는 경우 이 값을 `false`로 설정합니다. 이 구성 요소를 AWS제공 DLR 추론 구성 요소와 함께 사용하려면 종속성을 포함하여 다음 라이브러리를 설치하고 ML 구성 요소를 `ggc_user` 실행하는와 같은 시스템 사용자가 사용할 수 있도록 합니다.

- [Python](#) 3.7 이상(사용하는 Python 버전에 대한 pip 포함)
- [딥 러닝 런타임](#) v1.6.0
- [NumPy](#).
- [OpenCV-Python](#).
- [AWS IoT Device SDK Python용 v2](#).
- [AWS 공통 런타임\(CRT\) Python](#).
- [Picamera](#)(Raspberry Pi 디바이스 전용).
- [awscam 모듈](#)(AWS DeepLens 디바이스용).
- `libGL`(Linux 디바이스용)

기본값: `true`

사용법

true로 설정된 UseInstaller 구성 파라미터와 함께 이 구성 요소를 사용하여 DLR과 해당 종속성을 디바이스에 설치합니다. 구성 요소에서는 DLR에 필요한 OpenCV 및 NumPy 라이브러리가 포함되어 있는 가상 환경을 디바이스에 설정합니다.

Note

이 구성 요소의 설치 프로그램 스크립트에서도 디바이스에 가상 환경을 구성하고 설치된 기계 학습 프레임워크를 사용하는 데 필요한 추가 시스템 라이브러리의 최신 버전을 설치합니다. 그러면 디바이스의 기존 시스템 라이브러리가 업그레이드될 수 있습니다. 지원되는 각 운영 체제 별로 이 구성 요소가 설치하는 라이브러리의 목록은 다음 표를 참조하세요. 이 설치 프로세스를 사용자 지정하려면 UseInstaller 구성 파라미터를 false로 설정하고 자체 설치 관리자 스크립트를 개발합니다.

플랫폼	디바이스 시스템에 설치된 라이브러리	가상 환경에 설치된 라이브러리
Armv7l	build-essential , cmake, ca-certificates , git	setuptools , wheel
Amazon Linux 2	mesa-libGL	없음
Ubuntu	wget	없음

추론 구성 요소를 배포할 때 이 런타임 구성 요소에서는 DLR과 해당 종속성이 디바이스에 이미 설치되어 있는지 먼저 확인한 다음에 없으면 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
1.6.16	Greengrass nucleus 버전 2.12.5용으로 업데이트된 버전입니다.
1.6.12	버그 수정 및 개선 사항 Windows OS 사용자를 위한 설치 스크립트를 수정합니다.
1.6.11	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
1.6.10	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
1.6.9	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
1.6.8	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
1.6.7	버그 수정 및 개선 사항 - 특정 Linux 플랫폼에서 기본적으로 사용할 수 없는 libGL을 설치하도록 UseInstaller 설치 스크립트를 업데이트합니다. - 이 구성 요소의 가상 환경에서 Python 3.9를 항상 사용하도록 UseInstaller 설치 스크립트를 업데이트합니다. 이 변경 사항은 다른 라이브러리와 호환성을 보장하는 데 도움이 됩니다.
1.6.6	새로운 기능 - Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. - Windows 코어 디바이스에서 추론 결과 폴더를 구성하는 데 사용할 수 있는 새로운 WindowsMLRootPath 구성 파라미터를 추가합니다.
1.6.5	새로운 기능 이 구성 요소에서 설치 스크립트를 비활성화하는 데 사용할 수 있는 새로운 UseInstaller 구성 파라미터를 추가합니다.
1.6.4	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
1.6.3	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
1.6.2	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
1.6.1	새로운 기능 딥 러닝 런타임 v1.6.0 및 해당 종속성을 설치합니다. - Armv8(AArch64) 플랫폼의 DLR 설치에 대한 지원을 추가합니다. 이로써 NVIDIA Jetson(예: Jetson Nano)을 실행하는 Greengrass 코어 디바이스에 대한 기계 학습 지원이 확장됩니다. 버그 수정 및 개선 사항 - 가상 환경에 AWS IoT Device SDK 를 설치하여 구성 요소 구성을 읽고 구성 변경 사항을 적용합니다. - 사소한 추가 버그 수정 및 개선 사항입니다.
1.3.2	초기 버전입니다. DLR v1.3.0을 설치합니다.

TensorFlow Lite 이미지 분류

TensorFlow Lite 이미지 분류 구성 요소

(aws.greengrass.TensorFlowLiteImageClassification)에는 [TensorFlow Lite](#) 런타임과 사전 훈련된 MobileNet 1.0 양자화 모델 샘플을 사용하여 이미지 분류 추론을 수행하는 샘플 추론 코드가 포함되어 있습니다. 이 구성 요소는 [TensorFlow Lite 이미지 분류 모델 저장소](#) 변형과 [TensorFlow Lite 런타임](#) 구성 요소를 종속성으로 사용하여 TensorFlow Lite 런타임과 샘플 모델을 다운로드합니다.

사용자 지정 훈련된 TensorFlow Lite 모델과 함께 이 추론 구성 요소를 사용하려면 종속 모델 저장소 구성 요소의 [사용자 지정 버전을 생성](#)합니다. 자체 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 [사용자 지정 추론 구성 요소를 생성](#)할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소(aws.greengrass.generic)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
 - 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.11 and 2.1.12

다음 표에는 이 구성 요소의 버전 2.1.11 및 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

accessControl

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 허용하는 [권한 부여 정책](#)이 있는 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 경우에는 사용자 지정 주제 이름과 일치하도록 accessControl 파라미터의 resources 값도 수정해야 합니다.

기본값: ml/tflite/image-classification

Accelerator

사용하려는 액셀러레이터입니다. 지원되는 값은 cpu 및 gpu입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 액셀러레이션만 지원합니다. 다른 사용자 지정 모델에서 GPU 액셀러레이션을 사용하려면 [사용자 지정 모델 구성 요소를 생성](#)하여 퍼블릭 모델 구성 요소를 재정의합니다.

기본값: cpu

ImageDirectory

(선택 사항) 추론 구성 요소가 이미지를 읽는 디바이스의 폴더 경로입니다. 이 값은 읽기/쓰기 액세스 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

ImageName

(선택 사항) 추론 구성 요소가 예측하기 위한 입력으로 사용하는 이미지의 이름입니다. 구성 요소는 ImageDirectory에 지정된 폴더에서 이미지를 찾습니다. 기본적으로 구성 요소는 기본 이미지 디렉터리의 샘플 이미지를 사용합니다.는 jpeg, jpg, 및 이미지 형식을 AWS IoT Greengrass 지원합니다. png, npy.

기본값: cat.jpeg

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

InferenceInterval

(선택 사항) 추론 코드가 예측한 각 예측 사이의 시간(초)입니다. 샘플 추론 코드는 무기한으로 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 퍼블릭 모델 구성 요소에 사용되는 모델입니다. 사용자 지정 구성 요소로 퍼블릭 모델 구성 요소를 재정의하는 경우에만 이 파라미터를 수정합니다.

기본값:

```
{
  "model": "TensorFlowLite-Mobilenet"
}
```

UseCamera

(선택 사항) Greengrass 코어 디바이스에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 true 및 false입니다.

이 값을 true로 설정하면 샘플 추론 코드가 디바이스의 카메라에 액세스하여 캡처된 이미지에서 로컬로 추론을 실행합니다. ImageName 및 ImageDirectory 파라미터의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 촬영한 이미지를 저장하는 위치에 대한 읽기/쓰기 액세스 권한이 있는지 확인합니다.

기본값: false

Note

이 구성 요소의 레시피를 보면 기본 구성에는 UseCamera 구성 파라미터가 표시되지 않습니다. 그러나 구성 요소를 배포하는 경우 [구성 병합 업데이트](#)에서 이 파라미터의 값을 수정할 수 있습니다.

UseCamera를 true로 설정하면 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 심볼릭 링크도 생성해야 합니다. 샘플 추론 구성 요소와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.1.12	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	초기 버전입니다.

TensorFlow Lite 객체 감지

TensorFlow Lite 객체 감지 구성 요소(`aws.greengrass.TensorFlowLiteObjectDetection`)에는 [TensorFlow Lite](#)를 사용하여 객체 감지 추론을 수행하기 위한 샘플 추론 코드와 사전 훈련된 샘플 Single Shot Detection(SSD) MobileNet 1.0 모델이 포함되어 있습니다. 이 구성 요소는 [TensorFlow Lite 객체 감지 모델 저장소](#) 변형과 [TensorFlow Lite 런타임](#) 구성 요소를 종속성으로 사용하여 TensorFlow Lite와 샘플 모델을 다운로드합니다.

사용자 지정 훈련된 TensorFlow Lite 모델과 함께 이 추론 구성 요소를 사용하려면 종속 모델 저장소 구성 요소의 [사용자 지정 버전을 생성](#)하면 됩니다. 자체 사용자 지정 추론 코드를 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 [사용자 지정 추론 구성 요소를 생성](#)합니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)

- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.

- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.11 and 2.1.12

다음 표에는 이 구성 요소의 버전 2.1.11 및 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드

종속성	호환 버전	종속성 유형
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

종속성	호환 버전	종속성 유형
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트
TensorFlow Lite 이미지 분류 모델 저장소	>=2.1.0 <2.2.0	하드
TensorFlow Lite	>=2.5.0 <2.6.0	하드

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

accessControl

(선택 사항) 구성 요소가 기본 알림 주제에 메시지를 게시할 수 있도록 허용하는 [권한 부여 정책](#)이 있는 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 경우에는 사용자 지정 주제 이름과 일치하도록 accessControl 파라미터의 resources 값도 수정해야 합니다.

기본값: ml/tflite/object-detection

Accelerator

사용하려는 액셀러레이터입니다. 지원되는 값은 cpu 및 gpu입니다.

종속 모델 구성 요소의 샘플 모델은 CPU 액셀러레이션만 지원합니다. 다른 사용자 지정 모델에서 GPU 액셀러레이션을 사용하려면 [사용자 지정 모델 구성 요소를 생성](#)하여 퍼블릭 모델 구성 요소를 재정의합니다.

기본값: cpu

ImageDirectory

(선택 사항) 추론 구성 요소가 이미지를 읽는 디바이스의 폴더 경로입니다. 이 값은 읽기/쓰기 액세스 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/packages/artifacts-unarchived/component-name/object_detection/sample_images/`

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

ImageName

(선택 사항) 추론 구성 요소가 예측하기 위한 입력으로 사용하는 이미지의 이름입니다. 구성 요소는 ImageDirectory에 지정된 폴더에서 이미지를 찾습니다. 기본적으로 구성 요소는 기본 이미지 디렉터리의 샘플 이미지를 사용합니다.는 jpeg, jpg, 및 이미지 형식을 AWS IoT Greengrass 지원합니다. png, npy.

기본값: `objects.jpg`

Note

UseCamera 값을 true로 설정하는 경우 이 구성 파라미터는 무시됩니다.

InferenceInterval

(선택 사항) 추론 코드가 예측한 각 예측 사이의 시간(초)입니다. 샘플 추론 코드는 무기한으로 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 퍼블릭 모델 구성 요소에 사용되는 모델입니다. 사용자 지정 구성 요소로 퍼블릭 모델 구성 요소를 재정의하는 경우에만 이 파라미터를 수정합니다.

기본값:

```
{
  "model": "TensorFlowLite-SSD"
}
```

UseCamera

(선택 사항) Greengrass 코어 디바이스에 연결된 카메라의 이미지를 사용할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 `true` 및 `false`입니다.

이 값을 `true`로 설정하면 샘플 추론 코드가 디바이스의 카메라에 액세스하여 캡처된 이미지에서 로컬로 추론을 실행합니다. `ImageName` 및 `ImageDirectory` 파라미터의 값은 무시됩니다. 이 구성 요소를 실행하는 사용자에게 카메라가 촬영한 이미지를 저장하는 위치에 대한 읽기/쓰기 액세스 권한이 있는지 확인합니다.

기본값: `false`

Note

이 구성 요소의 레시피를 보면 기본 구성에는 `UseCamera` 구성 파라미터가 표시되지 않습니다. 그러나 구성 요소를 배포하는 경우 [구성 병합 업데이트](#)에서 이 파라미터의 값을 수정할 수 있습니다.

`UseCamera`를 `true`로 설정하면 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 심볼릭 링크도 생성해야 합니다. 샘플 추론 구성 요소와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

Note

이 구성 요소의 레시피를 보면 기본 구성에는 `UseCamera` 구성 파라미터가 표시되지 않습니다. 그러나 구성 요소를 배포하는 경우 [구성 병합 업데이트](#)에서 이 파라미터의 값을 수정할 수 있습니다.

`UseCamera`를 `true`로 설정하면 추론 구성 요소가 런타임 구성 요소에 의해 생성된 가상 환경에서 카메라에 액세스할 수 있도록 심볼릭 링크도 생성해야 합니다. 샘플 추론 구성 요소와 함께 카메라를 사용하는 방법에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.1.12	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	버그 수정 및 개선 사항 샘플 TensorFlow Lite 객체 감지 추론 결과에서 잘못된 경계 상자가 발생하는 이미지 크기 조정 문제를 해결합니다.
2.1.0	초기 버전입니다.

TensorFlow Lite 이미지 분류 모델 저장소

TensorFlow Lite 이미지 분류 모델 저장소

(variant.TensorFlowLite.ImageClassification.ModelStore)는 사전 훈련된 MobileNet v1 모델을 Greengrass 아티팩트로 포함하는 기계 학습 모델 구성 요소입니다. 이 구성 요소에 사용되는 샘플 모델은 [TensorFlow Hub](#)에서 가져왔으며 [TensorFlow Lite](#)를 사용하여 구현되었습니다.

[TensorFlow Lite 이미지 분류](#) 추론 구성 요소는 이 구성 요소를 모델 소스의 종속성으로 사용합니다. 사용자 지정 훈련된 TensorFlow Lite 모델을 사용하려면 이 모델 구성 요소의 [사용자 지정 버전을 생성](#)하고 사용자 지정 모델을 구성 요소 아티팩트로 포함합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 생성할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)

- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```


- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드 할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.11 and 2.1.12

다음 표에는 이 구성 요소의 버전 2.1.11 및 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.12	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	초기 버전입니다.

TensorFlow Lite 객체 감지 모델 저장소

TensorFlow Lite 객체 감지 모델 저장소

(`variant.TensorFlowLite.ObjectDetection.ModelStore`)는 사전 훈련된 Single Shot Detection(SSD) MobileNet 모델을 Greengrass 아티팩트로 포함하는 기계 학습 모델 구성 요소입니다. 이 구성 요소에 사용되는 샘플 모델은 [TensorFlow Hub](#)에서 가져왔으며 [TensorFlow Lite](#)를 사용하여 구현되었습니다.

[TensorFlow Lite 객체 감지](#) 추론 구성 요소는 이 구성 요소를 모델 소스의 종속성으로 사용합니다. 사용자 지정 훈련된 TensorFlow Lite 모델을 사용하려면 이 모델 구성 요소의 [사용자 지정 버전을 생성](#)하고 사용자 지정 모델을 구성 요소 아티팩트로 포함합니다. 이 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 모델 구성 요소를 생성할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)

- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.

- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.11 and 2.1.12

다음 표에는 이 구성 요소의 버전 2.1.11 및 2.1.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	소프트

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

2.1.7

다음 표에는 이 구성 요소의 버전 2.1.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

구성

이 구성 요소에는 구성 파라미터가 없습니다.

로컬 로그 파일

이 구성 요소는 로그를 출력하지 않습니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.12	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.1.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.8	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	초기 버전입니다.

TensorFlow Lite 런타임

TensorFlow Lite 런타임 구성 요소(`variant.TensorFlowLite`)에는 디바이스의 가상 환경에 [TensorFlow Lite](#) 버전 2.5.0 및 해당 종속성을 설치하는 스크립트가 포함되어 있습니다. [TensorFlow Lite 이미지 분류](#) 및 [TensorFlow Lite 객체 감지](#) 구성 요소는 이 런타임 구성 요소를 TensorFlow Lite 설치를 위한 종속성으로 사용합니다.

Note

TensorFlow Lite 런타임 구성 요소 v2.5.6 이상은 TensorFlow Lite 런타임 및 해당 종속성의 기존 설치를 재설치합니다. 이 재설치는 코어 디바이스에서 TensorFlow Lite의 호환 버전과 해당 종속성을 실행하는 데 도움이 됩니다.

다른 런타임을 사용하려면 이 구성 요소의 레시피를 템플릿으로 사용하여 [사용자 지정 기계 학습 구성 요소](#)를 생성하면 됩니다.

주제

- [버전](#)

- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.5.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드 할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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 요소에 코어 디바이스에서 사용하는 플랫폼에 따라 apt, yum, brew, pip 명령을 사용하여 패키지를 설치하는 설치 관리자 스크립트를 사용합니다. 이 구성 요소는 설치 관리자 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 디바이스가 설치를 위해 연결되는 패키지 인덱스 및 리포지토리의 엔드포인트를 식별해야 합니다.

이 구성 요소의 설치 스크립트에 필요한 엔드포인트를 식별하는 경우 다음 사항을 고려하세요.

- 엔드포인트는 코어 디바이스의 플랫폼에 따라 다릅니다. 예를 들어 Ubuntu를 실행하는 코어 디바이스는 yum 또는 brew 대신 apt를 사용합니다. 또한 동일한 패키지 인덱스를 사용하는 디바이스의 소스 목록은 다를 수 있으므로 서로 다른 리포지토리에서 패키지를 검색할 수 있습니다.

- 각 디바이스에는 패키지를 검색할 위치를 정의하는 자체 소스 목록이 있으므로 동일한 패키지 인덱스를 사용하는 여러 디바이스 간에 엔드포인트가 다를 수 있습니다.
- 엔드포인트는 시간이 지남에 따라 변경될 수 있습니다. 각 패키지 인덱스는 패키지를 다운로드하는 리포지토리의 URL을 제공하며 패키지 소유자는 패키지 인덱스가 제공하는 URL을 변경할 수 있습니다.

이 구성 요소에서 설치하는 종속성에 대한 자세한 내용과 설치 프로그램 스크립트를 비활성화하는 방법은 [UseInstaller](#) 구성 파라미터를 참조하세요.

기본 작업에 필요한 엔드포인트와 포트에 대한 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 섹션을 참조하세요.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.5.14 and 2.5.15

다음 표에는 이 구성 요소의 버전 2.5.14 및 2.5.15에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	소프트

2.5.13

다음 표에는 이 구성 요소의 버전 2.5.13에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	소프트

2.5.12

다음 표에는 이 구성 요소의 버전 2.5.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.11.0$	소프트

2.5.11

다음 표에는 이 구성 요소의 버전 2.5.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.10.0$	소프트

2.5.10

다음 표에는 이 구성 요소의 버전 2.5.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.9.0$	소프트

2.5.9

다음 표에는 이 구성 요소의 버전 2.5.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	$\geq 2.0.0 < 2.8.0$	소프트

2.5.8

다음 표에는 이 구성 요소의 버전 2.5.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.5.2

다음 표에는 이 구성 요소의 버전 2.5.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.5.1

다음 표에는 이 구성 요소의 버전 2.5.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.5.0

다음 표에는 이 구성 요소의 버전 2.5.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

MLRootPath

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Linux 코어 디바이스의 폴더 경로입니다. 이 값은 해당 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

WindowsMLRootPath

이 기능은 이 구성 요소의 v1.6.6 이상에 사용할 수 있습니다.

(선택 사항) 추론 구성 요소가 이미지를 읽고 추론 결과를 쓰는 Windows 코어 디바이스의 폴더 경로입니다. 이 값은 해당 구성 요소를 실행하는 사용자에게 읽기/쓰기 권한이 있는 디바이스의 모든 위치로 수정할 수 있습니다.

기본값: `C:\greengrass\v2\work\variant.DLR\greengrass_ml`

UseInstaller

(선택 사항) 이 구성 요소의 설치 관리자 스크립트를 사용하여 TensorFlow Lite와 해당 종속성을 설치할지 여부를 정의하는 문자열 값입니다. 지원되는 값은 true 및 false입니다.

TensorFlow Lite 설치에 사용자 지정 스크립트를 사용하려는 경우 또는 사전 빌드된 Linux 이미지에 런타임 종속성을 포함하려는 경우에는 이 값을 false로 설정합니다. 이 구성 요소를 AWS제공 TensorFlow Lite 추론 구성 요소와 함께 사용하려면 종속성을 포함하여 다음 라이브러리를 설치하고 ML 구성 요소를 ggc_user실행하는와 같은 시스템 사용자가 사용할 수 있도록 합니다.

- [Python](#) 3.8 이상(사용 중인 Python 버전의 경우 pip 포함)
- [TensorFlow Lite](#) v2.5.0
- [NumPy](#)
- [OpenCV-Python](#)
- [AWS IoT Device SDK Python용 v2](#)
- [AWS 공통 런타임\(CRT\) Python](#)
- [Picamera](#)(Raspberry Pi 디바이스용)
- [awscam 모듈](#)(AWS DeepLens 디바이스용)
- [libGL](#)(Linux 디바이스용)

기본값: true

사용법

UseInstaller 구성 파라미터가 true로 설정된 이 구성 요소를 사용하여 디바이스에 TensorFlow Lite와 해당 종속성을 설치합니다. 이 구성 요소는 디바이스에 TensorFlow Lite에 필요한 OpenCV 및 NumPy 라이브러리를 포함하는 가상 환경을 설정합니다.

Note

이 구성 요소의 설치 프로그램 스크립트에서도 디바이스에 가상 환경을 구성하고 설치된 기계 학습 프레임워크를 사용하는 데 필요한 추가 시스템 라이브러리의 최신 버전을 설치합니다. 그러면 디바이스의 기존 시스템 라이브러리가 업그레이드될 수 있습니다. 지원되는 각 운영 체제 별로 이 구성 요소가 설치하는 라이브러리의 목록은 다음 표를 참조하세요. 이 설치 프로세스를 사용자 지정하려면 UseInstaller 구성 파라미터를 false로 설정하고 자체 설치 관리자 스크립트를 개발합니다.

플랫폼	디바이스 시스템에 설치된 라이브러리	가상 환경에 설치된 라이브러리
Armv7l	build-essential , cmake, ca-certificates , git	setuptools , wheel
Amazon Linux 2	mesa-libGL	없음

플랫폼	디바이스 시스템에 설치된 라이브러리	가상 환경에 설치된 라이브러리
Ubuntu	wget	없음

추론 구성 요소를 배포할 때 이 런타임 구성 요소는 먼저 디바이스에 이미 TensorFlow Lite 및 해당 종속성이 설치되어 있는지 확인합니다. 그렇지 않으면 런타임 구성 요소가 이를 대신 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.5.15	Greengrass nucleus 2.12.5 릴리스용으로 업데이트된 버전입니다.
2.5.14	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.5.13	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.5.12	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.5.11	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.5.10	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.5.9	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.5.8	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.5.7	버그 수정 및 개선 사항 - 특정 Linux 플랫폼에서 기본적으로 사용할 수 없는 libGL을 설치하도록 UseInstaller 설치 스크립트를 업데이트합니다. - 이 구성 요소의 가상 환경에서 Python 3.9를 항상 사용하도록 UseInstaller 설치 스크립트를 업데이트합니다. 이 변경 사항은 다른 라이브러리와 호환성을 보장하는 데 도움이 됩니다.
2.5.6	버그 수정 및 개선 사항 - 이 구성 요소를 업데이트하여 TensorFlow Lite 2.5.0(tflite-runtime-2.5.0.post1)의 최신 패치를 설치하면 Python 3.9에서 이 구성 요소를 사용할 수 있습니다. 이 구성 요소가 해당 패치를 설치하지 못하면 대신 tflite-runtime-2.5.0 을 설치합니다. - 이 구성 요소를 업데이트하여 기존에 설치한 TensorFlow Lite 및 해당 종속성을 다시 설치합니다. 이 변경 사항은 코어 디바이스가 호환 버전의 TensorFlow Lite 및 해당 종속성을 실행하도록 하는 데 도움이 됩니다.
2.5.5	새로운 기능 - Windows가 실행되는 코어 디바이스에 대한 지원을 추가합니다. - Windows 코어 디바이스에서 추론 결과 폴더를 구성하는 데 사용할 수 있는 새 WindowsMLRootPath 구성 파라미터를 추가합니다.

버전	변경 사항
2.5.4	새로운 기능 이 구성 요소에서 설치 스크립트를 비활성화할 수 있는 새 <code>UseInstaller</code> 구성 파라미터를 추가합니다.
2.5.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.5.2	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.5.1	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.5.0	초기 버전입니다.

Modbus-RTU 프로토콜 어댑터

Modbus-RTU 프로토콜 어댑터 구성 요소(`aws.greengrass.Modbus`)는 로컬 Modbus RTU 디바이스에서 정보를 폴링합니다.

이 구성 요소를 사용하여 로컬 Modbus RTU 디바이스에서 정보를 요청하려면 이 구성 요소가 구독하는 주제에 메시지를 게시합니다. 메시지에서 디바이스에 전송할 Modbus RTU 요청을 지정합니다. 그러면 이 구성 요소는 Modbus RTU 요청에 대한 결과가 포함된 응답을 게시합니다.

Note

이 구성 요소는 AWS IoT Greengrass V1의 Modbus RTU 프로토콜 어댑터 커넥터와 유사한 기능을 제공합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 가이드의 [Modbus RTU 프로토콜 어댑터 커넥터](#)를 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)

- [구성](#)
- [입력 데이터](#)
- [출력 데이터](#)
- [Modbus RTU 요청 및 응답](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

유형

이 구성 요소는 Lambda 구성 요소(`aws.greengrass.lambda`)입니다. [Greengrass nucleus](#)에서는 [Lambda 런처 구성 요소](#)를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.
- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- AWS IoT Greengrass 코어 디바이스와 Modbus 디바이스 간의 물리적 연결입니다. 코어 디바이스는 직렬 포트(예: USB 포트)를 통해 Modbus RTU 네트워크에 물리적으로 연결되어 있어야 합니다.

- 이 구성 요소의 출력 데이터를 수신하려면 이 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)(aws.greengrass.LegacySubscriptionRouter)에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성에서는 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"
    }
  }
}
```

- region**을 사용하는 AWS 리전 로 바꿉니다.
- ##**을 이 구성 요소에서 실행되는 Lambda 함수의 버전으로 바꿉니다. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소의 버전에 대한 레시피를 확인해야 합니다. [AWS IoT Greengrass 콘솔](#)에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 페어를 찾습니다. 이 키-값 페어에는 Lambda 함수의 이름과 버전이 포함되어 있습니다.

⚠ Important

이 구성 요소를 배포할 때마다 레거시 구독 라우터의 Lambda 함수 버전을 업데이트해야 합니다. 이를 통해 배포하는 구성 요소 버전에 맞는 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

- Modbus-RTU 프로토콜 어댑터는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.11

다음 표에는 이 구성 요소의 버전 2.1.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.16.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.15.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.14.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	하드
Lambda 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 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

Note

이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함되어 있습니다. 디바이스에서 다음 파라미터만 사용하여 이 구성 요소를 구성하는 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 디바이스에 있는 물리적 Modbus 직렬 포트의 절대 경로입니다(예: /dev/ttyS2).

컨테이너에서 이 구성 요소를 실행하려면 이 경로를 구성 요소가 액세스할 수 있는 시스템 디바이스(containerParams.devices에)로 정의해야 합니다. 이 구성 요소는 기본적으로 컨테이너에서 실행됩니다.

Note

이 구성 요소에는 디바이스에 대한 읽기/쓰기 액세스 권한이 있어야 합니다.

ModbusBaudRate

(선택 사항) 로컬 Modbus TCP 디바이스와의 직렬 통신을 위한 전송 속도를 지정하는 문자열 값입니다.

기본값: 9600

ModbusByteSize

(선택 사항) 로컬 Modbus TCP 디바이스와의 직렬 통신에서 바이트 크기를 지정하는 문자열 값입니다. 5, 6, 7 또는 8비트를 선택합니다.

기본값: 8

ModbusParity

(선택 사항) 로컬 Modbus TCP 디바이스와의 직렬 통신에서 데이터 무결성을 확인하는데 사용할 패리티 모드입니다.

- E - 짝수 패리티를 사용하여 데이터 무결성을 확인합니다.
- O - 홀수 패리티를 사용하여 데이터 무결성을 확인합니다.
- N - 데이터 무결성을 확인하지 않습니다.

기본값: N

ModbusStopBits

(선택 사항) 로컬 Modbus TCP 디바이스와의 직렬 통신에서 바이트의 끝을 나타내는 비트 수를 지정하는 문자열 값입니다.

기본값: 1

containerMode

(선택 사항) 이 구성 요소에 대한 컨테이너화 모드입니다. 다음 옵션 중 하나를 선택합니다.

- **GreengrassContainer** - 구성 요소는 AWS IoT Greengrass 컨테이너 내의 격리된 런타임 환경에서 실행됩니다.

이 옵션을 지정하는 경우 컨테이너에 Modbus 디바이스에 대한 액세스 권한을 부여할 시스템 디바이스(containerParams.devices에)를 지정해야 합니다.

- **NoContainer** - 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.

기본값: GreengrassContainer

containerParams

(선택 사항) 이 구성 요소에 대한 컨테이너 파라미터를 포함하는 객체입니다.

containerMode에 GreengrassContainer를 지정하면 구성 요소에서 이러한 파라미터를 사용합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

memorySize

(선택 사항) 구성 요소에 할당되는 메모리 양(킬로바이트)입니다.

기본값은 512MB(525,312KB)입니다.

devices

(선택 사항) 구성 요소가 컨테이너에서 액세스할 수 있는 시스템 디바이스를 지정하는 객체입니다.

Important

컨테이너에서 이 구성 요소를 실행하려면 ModbusLocalPort 환경 변수에서 구성하는 시스템 디바이스를 지정해야 합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

`path`

코어 디바이스의 시스템 디바이스 경로입니다. `ModbusLocalPort`에 대해 구성하는 값과 같은 값이어야 합니다.

`permission`

(선택 사항) 컨테이너에서 시스템 디바이스에 액세스할 수 있는 권한입니다. 이 값은 구성 요소가 시스템 디바이스에 대한 읽기/쓰기 액세스 권한을 갖도록 지정하는 `rw`여야 합니다.

기본값: `rw`

`addGroupOwner`

(선택 사항) 구성 요소를 실행하는 시스템 그룹을 시스템 디바이스의 소유자로 추가할지 여부입니다.

기본값: `true`

`pubsubTopics`

(선택 사항) 구성 요소에서 메시지를 수신하려고 구독하는 주제를 포함하는 객체입니다. 각 주제와 구성 요소가 AWS IoT Core 또는 로컬 게시/구독 주제의 MQTT 주제를 구독하는지 여부를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

`type`

(선택 사항) 이 구성 요소에서 메시지 구독에 사용하는 게시/구독 메시징의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `PUB_SUB` – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

- IOT_CORE - AWS IoT Core MQTT 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본값: PUB_SUB

topic

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. type에 IotCore를 지정하면 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

Example예: 구성 병합 업데이트(컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example예: 구성 병합 업데이트(컨테이너 없음 모드)

```
{
  "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 디바이스에 있는 물리적 Modbus 직렬 포트의 절대 경로입니다(예: `/dev/ttyS2`).

컨테이너에서 이 구성 요소를 실행하려면 이 경로를 구성 요소가 액세스할 수 있는 시스템 디바이스(`containerParams.devices`에)로 정의해야 합니다. 이 구성 요소는 기본적으로 컨테이너에서 실행됩니다.

 Note

이 구성 요소에는 디바이스에 대한 읽기/쓰기 액세스 권한이 있어야 합니다.

`containerMode`

(선택 사항) 이 구성 요소에 대한 컨테이너화 모드입니다. 다음 옵션 중 하나를 선택합니다.

- `GreengrassContainer` - 구성 요소는 AWS IoT Greengrass 컨테이너 내의 격리된 런타임 환경에서 실행됩니다.

이 옵션을 지정하는 경우 컨테이너에 Modbus 디바이스에 대한 액세스 권한을 부여할 시스템 디바이스(`containerParams.devices`에)를 지정해야 합니다.

- `NoContainer` - 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.

기본값: `GreengrassContainer`

`containerParams`

(선택 사항) 이 구성 요소에 대한 컨테이너 파라미터를 포함하는 객체입니다.

`containerMode`에 `GreengrassContainer`를 지정하면 구성 요소에서 이러한 파라미터를 사용합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`memorySize`

(선택 사항) 구성 요소에 할당되는 메모리 양(킬로바이트)입니다.

기본값은 512MB(525,312KB)입니다.

`devices`

(선택 사항) 구성 요소가 컨테이너에서 액세스할 수 있는 시스템 디바이스를 지정하는 객체입니다.

 Important

컨테이너에서 이 구성 요소를 실행하려면 `ModbusLocalPort` 환경 변수에서 구성하는 시스템 디바이스를 지정해야 합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`0` – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

`path`

코어 디바이스의 시스템 디바이스 경로입니다. `ModbusLocalPort`에 대해 구성하는 값과 같은 값이어야 합니다.

`permission`

(선택 사항) 컨테이너에서 시스템 디바이스에 액세스할 수 있는 권한입니다. 이 값은 구성 요소가 시스템 디바이스에 대한 읽기/쓰기 액세스 권한을 갖도록 지정하는 `rw`여야 합니다.

기본값: `rw`

`addGroupOwner`

(선택 사항) 구성 요소를 실행하는 시스템 그룹을 시스템 디바이스의 소유자로 추가할지 여부입니다.

기본값: `true`

pubsubTopics

(선택 사항) 구성 요소에서 메시지를 수신하려고 구독하는 주제를 포함하는 객체입니다. 각 주제와 구성 요소가 AWS IoT Core 또는 로컬 게시/구독 주제의 MQTT 주제를 구독하는지 여부를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

type

(선택 사항) 이 구성 요소에서 메시지 구독에 사용하는 게시/구독 메시징의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- PUB_SUB – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.
- IOT_CORE - AWS IoT Core MQTT 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본값: PUB_SUB

topic

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. type에 IotCore를 지정하면 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

Example예: 구성 병합 업데이트(컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
```

```

    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}

```

Example에: 구성 병합 업데이트(컨테이너 없음 모드)

```

{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}

```

입력 데이터

이 구성 요소는 다음 주제에 대한 Modbus RTU 요청 파라미터를 수락하고 Modbus RTU 요청을 디바이스에 전송합니다. 기본적으로 이 구성 요소는 로컬 게시/구독 메시지를 구독합니다. 사용자 지정 구성 요소에서 이 구성 요소로 메시지를 게시하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

기본 주제(로컬 게시/구독): modbus/adapter/request

메시지는 다음 속성을 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

request

전송할 Modbus RTU 요청의 파라미터입니다.

요청 메시지의 셰이프는 메시지가 나타내는 Modbus RTU 요청 유형에 따라 다릅니다. 다음은 모든 요청에 필수적인 속성입니다.

유형: 다음 정보를 포함하는 object입니다.

operation

실행할 작업의 이름입니다. 예를 들어 Modbus RTU 디바이스에서 코일을 읽으려면 `ReadCoilsRequest`를 지정합니다. 지원되는 작업에 대한 자세한 내용은 [Modbus RTU 요청 및 응답](#) 섹션을 참조하세요.

유형: string

device

요청의 대상 디바이스입니다.

이 값은 0에서 247 사이의 정수여야 합니다.

유형: integer

요청에 포함할 다른 파라미터는 작업에 따라 다릅니다. 이 구성 요소는 [순환 중복 검사\(CRC\)](#)를 처리하여 사용자의 데이터 요청을 확인합니다.

Note

요청에 `address` 속성이 포함된 경우 해당 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 요소는 응답 객체의 `id` 속성을 해당 값으로 설정합니다.

유형: string

Example입력 예: 코일 요청 읽기

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
}
```

```
"id": "MyRequest"
}
```

출력 데이터

이 구성 요소는 기본적으로 다음 MQTT 주제에 대한 출력 데이터로 응답을 게시합니다. [레거시 구독 라우터 구성 요소](#)의 구성에서 이 주제를 subject로 지정해야 합니다. 사용자 지정 구성 요소에서 이 주제에 대한 메시지를 구독하는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본 주제(AWS IoT Core MQTT): modbus/adapter/response

응답 메시지의 세이프는 요청 작업 및 응답 상태에 따라 달라집니다. 예시는 [예제 요청 및 응답](#) 섹션을 참조하세요.

모든 응답에는 다음 속성이 포함됩니다.

response

Modbus RTU 디바이스의 응답입니다.

유형: 다음 정보를 포함하는 object입니다.

status

요청 상태입니다. 상태는 다음 값 중 하나일 수 있습니다.

- Success – 요청이 유효하고 구성 요소에서 Modbus RTU 네트워크에 요청을 전송했으며 Modbus RTU 네트워크에서 응답을 반환했습니다.
- Exception - 요청이 유효하고 구성 요소에서 Modbus RTU 네트워크에 요청을 전송했으며 Modbus RTU 네트워크에서 예외를 반환했습니다. 자세한 내용은 [응답 상태: 예외](#) 단원을 참조하십시오.
- No Response - 요청이 잘못되었고, 구성 요소에서 Modbus RTU 네트워크에 요청을 보내기 전에 오류가 발견되었습니다. 자세한 내용은 [응답 상태: 응답 없음](#) 단원을 참조하십시오.

operation

구성 요소에서 요청한 작업입니다.

device

구성 요소에서 요청을 전송한 디바이스입니다.

payload

Modbus RTU 디바이스의 응답입니다. status가 No Response인 경우 이 객체에는 오류 설명이 있는 error 속성만 포함됩니다(예: [Input/Output] No Response received from the remote unit).

id

요청의 ID이며, 요청에 해당하는 응답을 식별하는 데 사용할 수 있습니다.

Note

쓰기 작업에 대한 응답은 단순히 요청의 에코일 뿐입니다. 쓰기 응답에 대해 의미 있는 정보가 없더라도 응답의 상태를 확인하여 요청의 성공 여부를 확인하는 것이 좋습니다.

Example출력 예: 성공

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example출력 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 예시는 [예제 요청 및 응답](#)를 참조합니다.

Modbus RTU 요청 및 응답

이 커넥터는 Modbus RTU 요청 파라미터를 [입력 데이터](#)로 수락하고 응답을 [출력 데이터](#)로 게시합니다.

지원되는 일반적인 작업은 다음과 같습니다.

요청 시 작업 이름	응답의 함수 코드
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

예제 요청 및 응답

다음은 지원되는 작업에 대한 예제 요청 및 응답입니다.

코일 읽기

요청 예:

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

응답 예:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

개별 입력 읽기

요청 예:

```
{
  "request": {
    "operation": "ReadDiscreteInputsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

응답 예:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadDiscreteInputsRequest",
    "payload": {
      "function_code": 2,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

보류 레지스터 읽기

요청 예:

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

응답 예:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadHoldingRegistersRequest",
    "payload": {
      "function_code": 3,
      "registers": [20,30]
    }
  },
  "id" : "TestRequest"
}
```

```
}
```

입력 레지스터 읽기

요청 예:

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "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 쓰기

요청 예:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

다중 코일 쓰기

요청 예:

```
{
  "request": {
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
  "id": "TestRequest"
}
```

응답 예:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleCoilsRequest",
    "payload": {
      "function_code": 15,
      "address": 1,
      "count": 4
    }
  }
}
```

```
},  
  "id" : "TestRequest"  
}
```

다중 레지스터 쓰기

요청 예:

```
{  
  "request": {  
    "operation": "WriteMultipleRegistersRequest",  
    "device": 1,  
    "address": 1,  
    "values": [20,30,10]  
  },  
  "id": "TestRequest"  
}
```

응답 예:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteMultipleRegistersRequest",  
    "payload": {  
      "function_code": 23,  
      "address": 1,  
      "count": 3  
    }  
  },  
  "id" : "TestRequest"  
}
```

레지스터 쓰기 마스크

요청 예:

```
{  
  "request": {  
    "operation": "MaskWriteRegisterRequest",  
    "device": 1,  
    "address": 1,
```

```

    "and_mask": 175,
    "or_mask": 1
  },
  "id": "TestRequest"
}

```

응답 예:

```

{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    }
  },
  "id" : "TestRequest"
}

```

다중 레지스터 쓰기 읽기

요청 예:

```

{
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
    "read_count": 2,
    "write_address": 3,
    "write_registers": [20,30,40]
  },
  "id": "TestRequest"
}

```

응답 예:

```

{
  "response": {

```



```

    "status": "success",
    "device": 1,
    "operation": "ReadWriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "registers": [10,20,10,20]
    }
  },
  "id" : "TestRequest"
}

```

Note

응답에는 구성 요소에서 읽는 레지스터가 포함됩니다.

응답 상태: 예외

요청 형식이 올바르지만 요청이 성공적으로 완료되지 않으면 예외가 발생합니다. 이 경우 응답에는 다음 정보가 포함됩니다.

- status는 Exception으로 설정됩니다.
- function_code는 요청의 코드 함수 + 128과 같습니다.
- exception_code에는 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 정보를 포함하는 응답을 반환합니다.

- status는 No Response으로 설정됩니다.
- error에는 오류 이유가 포함되어 있습니다.
- error_message에는 오류 메시지가 포함되어 있습니다.

예:

```
{
  "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 장치를 대상으로 하거나 Modbus RTU 네트워크가 작동하지 않는 경우 응답 없음 형식을 사용하는 ModbusIOException가 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. *를* AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
```

라이선스

이 구성 요소에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- [pymodbus](#)/BSD 라이선스
- [pyserial](#)/BSD 라이선스

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.11	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.8	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	버그 수정 및 개선 사항 ReadDiscreteInput 작업 문제를 해결합니다.
2.1.4	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 Modbus RTU 디바이스와의 직렬 통신을 구성하기 위해 지정할 수 있는 ModbusBaudRate , ModbusByteSize , ModbusParity 및 ModbusStopBits 옵션을 추가합니다.
2.0.8	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

MQTT 브리지

MQTT 브리지 구성 요소(`aws.greengrass.clientdevices.mqtt.Bridge`)는 클라이언트 디바이스, 로컬 Greengrass 게시/구독 및 간에 MQTT 메시지를 릴레이합니다 AWS IoT Core. 이 구성 요소를

사용하여 사용자 지정 구성 요소의 클라이언트 디바이스에서 보내는 MQTT 메시지에 대한 조치를 취하고 클라이언트 디바이스를 AWS 클라우드와 동기화할 수 있습니다.

Note

클라이언트 디바이스는 Greengrass 코어 디바이스에 연결되어 처리할 MQTT 메시지와 데이터를 보내는 로컬 IoT 디바이스입니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

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 디바이스에 있는 구성 요소 간의 메시지를 처리합니다. Greengrass 구성 요소에서 이러한 메시지를 조작하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.
- AWS IoT Core - AWS IoT Core MQTT 브로커는 IoT 디바이스와 AWS 클라우드 대상 간의 메시지를 처리합니다. Greengrass 구성 요소에서 이러한 메시지를 조작하는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)

- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 구성합니다.

- MQTT 브리지 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.3.2

다음 표에는 이 구성 요소의 버전 2.3.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.6.0	하드

2.3.0 and 2.3.1

다음 표에는 이 구성 요소의 버전 2.3.0 및 2.3.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	>=2.0.0 <2.1.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.3.0 – 2.3.2

`mqttTopicMapping`

브리지할 주제 매핑입니다. 이 구성 요소는 소스 주제에 대한 메시지를 구독하고 수신하는 메시지를 대상 주제에 게시합니다. 각 주제 매핑에서는 주제, 소스 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topicMappingNameKey`

이 주제 매핑의 이름입니다. *`topicMappingNameKey`*를 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`topic`

소스 브로커와 대상 브로커를 브리지할 주제 또는 주제 필터입니다.

+ 및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

 Note

Pubsub 소스 브로커와 함께 MQTT 주제 와일드카드를 사용하려면 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상을 사용해야 합니다.

`targetTopicPrefix`

이 구성 요소가 메시지를 릴레이할 때 대상 주제에 추가할 접두사입니다.

`source`

소스 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- `LocalMqtt` - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- `Pubsub` - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- `IotCore` - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

target

대상 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

mqtt5RouteOptions

(선택 사항) 소스 주제에서 대상 주제로 메시지를 브리징하기 위한 주제 매핑을 구성하는 옵션을 제공합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

mqtt5RouteOptionsNameKey

주제 매핑에 대한 경로 옵션의 이름입니다. *mqtt5RouteOptionsNameKey*를 *mqttTopicMapping* 필드에 정의된 일치하는 *topicMappingNameKey*로 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

noLocal

(선택 사항) 활성화하는 경우 브리지는 브리지 자체에서 게시한 주제에 대한 메시지를 전달하지 않습니다. 다음과 같은 루프를 방지하려면 이 옵션을 사용합니다.

```
{
  "mqtt5RouteOptions": {
    "toIoTCore": {
      "noLocal": true
    }
  },
  "mqttTopicMapping": {
    "toIoTCore": {
      "topic": "device",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "toLocal": {
      "topic": "device",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

*noLocal*는 *source*가 *LocalMqtt*인 경로에 대해서만 지원됩니다.

기본값: *false*

retainAsPublished

(선택 사항) 활성화하는 경우 브리지에서 전달하는 메시지에는 해당 경로에 대해 브로커에 게시된 메시지와 동일한 *retain* 플래그가 지정됩니다.

*retainAsPublished*는 *source*가 *LocalMqtt*인 경로에 대해서만 지원됩니다.

기본값: *false*

mqtt

(선택 사항) 로컬 브로커와 통신하기 위한 MQTT 프로토콜 설정입니다.

version

(선택 사항) 브리지에서 로컬 브로커와 통신하는 데 사용되는 MQTT 프로토콜 버전입니다. nucleus 구성에서 선택한 MQTT 버전과 동일해야 합니다.

다음 중에서 선택합니다.

- mqtt3
- mqtt5

mqttTopicMapping 객체의 source 또는 target 필드가 LocalMqtt로 설정된 경우 MQTT 브로커를 배포해야 합니다. mqtt5 옵션을 선택하는 경우 [MQTT 5 브로커\(EMQX\)](#)를 사용해야 합니다.

기본값: mqtt3

ackTimeoutSeconds

(선택 사항) 작업에 실패하기 전에 PUBACK, SUBACK 또는 UNSUBACK 패킷을 기다리는 시간 간격입니다.

기본값: 60

connAckTimeoutMs

(선택 사항) 연결을 종료하기 전에 CONNACK 패킷을 기다리는 시간 간격입니다.

기본값: 20000(20초)

pingTimeoutMs

(선택 사항) 브리지가 로컬 브로커에서 PINGACK 메시지를 수신하기 위해 대기하는 시간(밀리초)입니다. 대기 시간이 제한 시간을 초과하면 브리지가 닫히고 MQTT 연결을 다시 엽니다. 이 값은 keepAliveTimeoutSeconds보다 작아야 합니다.

기본값: 30000(30초)

keepAliveTimeoutSeconds

(선택 사항) MQTT 연결을 유지하기 위해 브리지가 보내는 각 PING 메시지 사이의 시간(초)입니다. 이 값은 pingTimeoutMs보다 커야 합니다.

기본값: 60

maxReconnectDelayMs

(선택 사항) MQTT가 다시 연결하는 데 걸리는 최대 시간(초)입니다.

기본값: 30000(30초)

minReconnectDelayMs

(선택 사항) MQTT가 다시 연결하는 데 걸리는 최소 시간(초)입니다.

receiveMaximum

(선택 사항) 브리지가 보낼 수 있는 확인되지 않은 최대 QoS1 패킷 수입니다.

기본값: 100

maximumPacketSize

클라이언트가 MQTT 패킷에 대해 허용할 최대 바이트 수입니다.

기본값: null(제한 없음)

sessionExpiryInterval

(선택 사항) 브리지와 로컬 브로커 간에 세션이 지속되도록 요청할 수 있는 시간(초)입니다.

기본값: 4294967295(세션이 만료되지 않음)

brokerUri

(선택 사항) 로컬 MQTT 브로커의 URI입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다. `ssl://localhost:port` 형식을 사용하고 *port*를 MQTT 브로커가 작동하는 포트로 바꿉니다.

기본값: `ssl://localhost:8883`

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 BROKEN으로 변경됩니다.

기본값: 120

Example예: 구성 병합 업데이트

아래에 나와 있는 구성 업데이트 예에서는 다음을 지정합니다.

- `clients/+/hello/world` 주제 필터와 일치하는 주제에 AWS IoT Core 대해 클라이언트 디바이스에서 로 메시지를 릴레이합니다.
- `clients/+/detections` 주제 필터와 일치하는 주제에 대한 메시지를 클라이언트 디바이스에서 로컬 게시/구독으로 릴레이하고 대상 주제에 `events/input/` 접두사를 추가합니다. 결과 대상 주제는 `events/input/clients/+/detections` 주제 필터와 일치합니다.
- `clients/+/status` 주제 필터와 일치하는 주제에 AWS IoT Core 대해 클라이언트 디바이스에서 로 메시지를 릴레이하고 대상 주제에 `$aws/rules/StatusUpdateRule/` 접두사를 추가합니다. 이 예제에서는 이러한 메시지를 StatusUpdateRule이라는 [AWS IoT 규칙](#)으로 직접 릴레이하여 [기본 수집](#)을 사용하는 비용을 절감합니다.

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

Example예: MQTT 5 구성

다음 예제 구성에서는 다음을 업데이트합니다.

- 브리지가 로컬 브로커와 함께 MQTT 5 프로토콜을 사용할 수 있도록 합니다.
- ClientDeviceHelloWorld 주제 매핑에 대한 게시된 설정으로 MQTT 유지를 구성합니다.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

mqttTopicMapping

브리지할 주제 매핑입니다. 이 구성 요소는 소스 주제에 대한 메시지를 구독하고 수신하는 메시지를 대상 주제에 게시합니다. 각 주제 매핑에서는 주제, 소스 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름입니다. *topicMappingNameKey*를 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topic

소스 브로커와 대상 브로커를 브리지할 주제 또는 주제 필터입니다.

+ 및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

Note

Pubsub 소스 브로커와 함께 MQTT 주제 와일드카드를 사용하려면 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상을 사용해야 합니다.

targetTopicPrefix

이 구성 요소가 메시지를 릴레이할 때 대상 주제에 추가할 접두사입니다.

source

소스 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

target

대상 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고

구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

brokerUri

(선택 사항) 로컬 MQTT 브로커의 URI입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다. `ssl://localhost:port` 형식을 사용하고 `port`를 MQTT 브로커가 작동하는 포트로 바꿉니다.

기본값: `ssl://localhost:8883`

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 BROKEN으로 변경됩니다.

기본값: 120

Example예: 구성 병합 업데이트

아래에 나와 있는 구성 업데이트 예에서는 다음을 지정합니다.

- `clients/+ /hello/world` 주제 필터와 일치하는 주제에 AWS IoT Core 대해 클라이언트 디바이스에서 로 메시지를 릴레이합니다.
- `clients/+ /detections` 주제 필터와 일치하는 주제에 대한 메시지를 클라이언트 디바이스에서 로컬 게시/구독으로 릴레이하고 대상 주제에 `events/input/` 접두사를 추가합니다. 결과 대상 주제는 `events/input/clients/+ /detections` 주제 필터와 일치합니다.
- `clients/+ /status` 주제 필터와 일치하는 주제에 AWS IoT Core 대해 클라이언트 디바이스에서 로 메시지를 릴레이하고 대상 주제에 `$aws/rules/StatusUpdateRule/` 접두사를 추가합니다. 이 예제에서는 이러한 메시지를 StatusUpdateRule이라는 [AWS IoT 규칙](#)으로 직접 릴레이하여 [기본 수집](#)을 사용하는 비용을 절감합니다.

```
{
  "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

브리지할 주제 매핑입니다. 이 구성 요소는 소스 주제에 대한 메시지를 구독하고 수신하는 메시지를 대상 주제에 게시합니다. 각 주제 매핑에서는 주제, 소스 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름입니다. *topicMappingNameKey*를 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topic

소스 브로커와 대상 브로커를 브리지할 주제 또는 주제 필터입니다.

+ 및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

Note

Pubsub 소스 브로커와 함께 MQTT 주제 와일드카드를 사용하려면 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상을 사용해야 합니다.

targetTopicPrefix

이 구성 요소가 메시지를 릴레이할 때 대상 주제에 추가할 접두사입니다.

source

소스 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

target

대상 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고

구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

brokerUri

(선택 사항) 로컬 MQTT 브로커의 URI입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다. `ssl://localhost:port` 형식을 사용하고 *port*를 MQTT 브로커가 작동하는 포트에 바꿉니다.

기본값: `ssl://localhost:8883`

Example예: 구성 병합 업데이트

아래에 나와 있는 구성 업데이트 예에서는 다음을 지정합니다.

- `clients/+ /hello/world` 주제 필터와 일치하는 주제에 AWS IoT Core 대해 클라이언트 디바이스에서 로 메시지를 릴레이합니다.
- `clients/+ /detections` 주제 필터와 일치하는 주제에 대한 메시지를 클라이언트 디바이스에서 로컬 게시/구독으로 릴레이하고 대상 주제에 `events/input/` 접두사를 추가합니다. 결과 대상 주제는 `events/input/clients/+ /detections` 주제 필터와 일치합니다.
- `clients/+ /status` 주제 필터와 일치하는 주제에 AWS IoT Core 대해 클라이언트 디바이스에서 로 메시지를 릴레이하고 대상 주제에 `$aws/rules/StatusUpdateRule/` 접두사를 추가합니다. 이 예제에서는 이러한 메시지를 StatusUpdateRule이라는 [AWS IoT 규칙](#)으로 직접 릴레이하여 [기본 수집](#)을 사용하는 비용을 절감합니다.

```
{
  "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

브리지할 주제 매핑입니다. 이 구성 요소는 소스 주제에 대한 메시지를 구독하고 수신하는 메시지를 대상 주제에 게시합니다. 각 주제 매핑에서는 주제, 소스 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름입니다. *topicMappingNameKey*를 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topic

소스 브로커와 대상 브로커를 브리지할 주제 또는 주제 필터입니다.

LocalMqtt 또는 IotCore 소스 브로커를 지정하는 경우 + 및 # MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

source

소스 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.

- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

 Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

target

대상 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub - 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

 Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

brokerUri

(선택 사항) 로컬 MQTT 브로커의 URI입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다. `ssl://localhost:port` 형식을 사용하고 *port*를 MQTT 브로커가 작동하는 포트로 바꿉니다.

기본값: `ssl://localhost:8883`

Example예: 구성 병합 업데이트

다음 예제 구성 업데이트는 `clients/MyClientDevice1/hello/world` 및 `clients/MyClientDevice2/hello/world` 주제에서 클라이언트 디바이스의 메시지를 AWS IoT Core 로 릴레이하도록 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

브리지할 주제 매핑입니다. 이 구성 요소는 소스 주제에 대한 메시지를 구독하고 수신하는 메시지를 대상 주제에 게시합니다. 각 주제 매핑에서는 주제, 소스 유형 및 대상 유형을 정의합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topicMappingNameKey

이 주제 매핑의 이름입니다. *topicMappingNameKey*를 이 주제 매핑을 식별하는 데 도움이 되는 이름으로 바꿉니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

topic

소스 브로커와 대상 브로커를 브리지할 주제 또는 주제 필터입니다.

`LocalMqtt` 또는 `IotCore` 소스 브로커를 지정하는 경우 `+` 및 `#` MQTT 주제 와일드카드를 사용하여 주제 필터와 일치하는 모든 주제에 대한 메시지를 릴레이할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT 주제](#)를 참조하세요.

source

소스 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub – 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

target

대상 메시지 브로커입니다. 다음 옵션 중 하나를 선택합니다.

- LocalMqtt - 클라이언트 디바이스가 통신하는 로컬 MQTT 브로커입니다.
- Pubsub – 로컬 Greengrass 게시/구독 메시지 브로커입니다.
- IotCore - AWS IoT Core MQTT 메시지 브로커입니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT 브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

source 및 target은 서로 달라야 합니다.

Example예: 구성 병합 업데이트

다음 예제 구성 업데이트는 `clients/MyClientDevice1/hello/world` 및 `clients/MyClientDevice2/hello/world` 주제에서 클라이언트 디바이스의 메시지를 AWS IoT Core 로 릴레이하도록 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 nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.3.2	클라이언트 디바이스 인증 버전 2.5.0 릴리스에 대한 버전이 업데이트되었습니다.
2.3.1	버그 수정 및 개선 사항 로컬 MQTT 클라이언트에서 연결 해제 루프가 발생하는 문제를 해결합니다.
2.3.0	새로운 기능 AWS IoT Core 및 로컬 MQTT 소스 간의 브리징에 대한 MQTT5 지원을 추가합니다.
2.2.6	새로운 기능 새로운 startupTimeoutSeconds 구성 옵션이 추가됩니다.
2.2.5	클라이언트 디바이스 인증 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
2.2.4	Greengrass 클라이언트 디바이스 인증 버전 2.3.0 릴리스에 대한 버전이 업데이트되었습니다.
2.2.3	이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.

버전	변경 사항
2.2.2	버그 수정 및 개선 사항 로깅 조정
2.2.1	버그 수정 및 개선 사항 MQTT 브리지가 MQTT 주제를 구독하지 못하게 될 수 있는 문제가 해결됩니다.
2.2.0	새로운 기능 - 로컬 게시/구독을 소스 메시지 브로커로 지정할 때 MQTT 주제 와일드카드(# 및 +)에 대한 지원을 추가합니다. 이 기능을 사용하려면 Greengrass nucleus 구성 요소의 v2.6.0 이상이 필요합니다. - 메시지를 릴레이할 때 대상 주제에 접두사를 추가하도록 MQTT 브리지를 구성하기 위해 지정할 수 있는 targetTopicPrefix 옵션이 추가됩니다.
2.1.1	버그 수정 및 개선 사항 - 이 구성 요소에서 구성 재설정 업데이트가 처리되는 방식과 관련된 문제가 해결됩니다. - 인증서 교체 시 MQTT 클라이언트 연결 해제 빈도가 줄어듭니다.
2.1.0	새로운 기능 기본값이 아닌 MQTT 브로커 포트를 사용할 수 있는 brokerUri 파라미터가 추가됩니다.
2.0.1	이 버전에는 버그 수정 및 개선 사항이 포함됩니다.
2.0.0	초기 버전입니다.

MQTT 3.1.1 브로커(Moquette)

EMQX MQTT 브로커 구성 요소(`aws.greengrass.clientdevices.mqtt.Moquette`)는 클라이언트 디바이스와 Greengrass 코어 디바이스 간의 MQTT 메시지를 처리합니다. 이 구성 요소는

[Moquette MQTT 브로커](#)의 수정된 버전을 제공합니다. 경량 MQTT 브로커를 실행하려면 이 MQTT 브로커를 배포합니다. MQTT 브로커를 선택하는 방법에 대한 자세한 내용은 [MQTT 브로커 선택](#) 섹션을 참조하세요.

이 브로커는 MQTT 3.1.1 프로토콜을 구현합니다. 이 프로토콜에는 QoS 0, QoS 1, QoS 2 보존된 메시지, 유언 메시지 및 영구 세션에 대한 지원이 포함됩니다.

 Note

클라이언트 디바이스는 Greengrass 코어 디바이스에 연결되어 처리할 MQTT 메시지와 데이터를 보내는 로컬 IoT 디바이스입니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 허용할 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 요소](#)를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 사용하도록 구성합니다.

다른 포트를 지정하고 [IP 감지기 구성 요소](#)를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 감지기 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 보고하도록 구성합니다.

- Moquette MQTT 브로커 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹

선에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.3.7

다음 표에는 이 구성 요소의 버전 2.3.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.6.0	하드

2.3.2 – 2.3.6

다음 표에는 이 구성 요소의 버전 2.3.2~2.3.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	$\geq 2.0.0 < 2.2.0$	하드

2.0.0 - 2.0.2

다음 표에는 이 구성 요소의 버전 2.0.0~2.0.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	$\geq 2.0.0 < 2.1.0$	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

moquette

(선택 사항) 사용할 [Moquette MQTT 브로커](#) 구성입니다. 이 구성 요소에서 Moquette 구성 옵션의 하위 집합을 구성할 수 있습니다. 자세한 내용은 [Moquette 구성 파일](#)의 인라인 설명을 참조하세요.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

ssl_port

(선택 사항) MQTT 브로커가 작동하는 포트입니다.

 Note

다른 포트를 지정하고 [MQTT 브리지 구성 요소](#)를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 사용하도록 구성합니다.

다른 포트를 지정하고 [IP 감지기 구성 요소](#)를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 감지기 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 보고하도록 구성합니다.

기본값: 8883

host

(선택 사항) MQTT 브로커가 바인딩되는 인터페이스입니다. 예를 들어 MQTT 브로커가 특정 로컬 네트워크에만 바인딩되도록 이 파라미터를 변경할 수 있습니다.

기본값: 0.0.0.0(모든 네트워크 인터페이스에 바인딩됨)

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 BROKEN으로 변경됩니다.

기본값: 120

Example예: 구성 병합 업데이트

다음 예제 구성에서는 포트 443에서 MQTT 브로커를 작동하도록 지정합니다.

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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


이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.3.7	클라이언트 디바이스 인증 버전 2.5.0 릴리스에 대한 버전이 업데이트되었습니다.
2.3.6	버그 수정 및 개선 사항 일반적인 버그 수정 및 개선 사항.
2.3.5	버그 수정 및 개선 사항 Moquette가 버전 0.17로 업데이트되었습니다.
2.3.4	버그 수정 및 개선 사항 클라이언트가 메시지를 보내거나 받을 때 중복된 클라이언트 ID로 인해 잘못된 세션 오류가 발생할 수 있는 문제를 해결합니다. 이 문제로 인해 클라이언트의 세션이 닫혔습니다.
2.3.3	새로운 기능 새로운 <code>startupTimeoutSeconds</code> 구성 옵션이 추가됩니다.

버전	변경 사항
2.3.2	클라이언트 디바이스 인증 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
2.3.1	버그 수정 및 개선 사항 다시 연결하려고 시도한 후 잘못된 세션으로 인해 클라이언트 연결이 해제될 수 있는 경쟁 조건이 해결됩니다.
2.3.0	인증서 체인에 대한 지원을 추가합니다.
2.2.0	클라이언트 디바이스 인증 버전 2.2.0 릴리스에 대한 버전이 업데이트되었습니다.
2.1.0	버그 수정 및 개선 사항 - 성능을 개선하고 몇 가지 기타 개선 사항을 포함하는 Moquette 버전 0.16을 사용하도록 이 구성 요소가 업데이트됩니다. - 로컬 MQTT 서버 인증서가 특정 시나리오에서 의도보다 자주 교체되는 문제를 해결합니다. 이 수정 사항을 적용하려면 클라이언트 디바이스 인증 구성 요소의 v2.1.0 이상을 사용해야 합니다.
2.0.2	버그 수정 및 개선 사항 - 최대 MQTT 메시지 크기가 8,092바이트에서 128킬로바이트로 증가됩니다. 메시지 크기 제한에 메시지 헤더가 포함되므로 유효한 MQTT 메시지 페이로드 제한은 약간 더 작습니다. - ssl_port 파라미터에서 정수 값에 대한 지원을 추가합니다.
2.0.1	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.0	초기 버전입니다.

MQTT 5 브로커(EMQX)

EMQX MQTT 브로커 구성 요소(aws.greengrass.clientdevices.mqtt.EMQX)는 클라이언트 디바이스와 Greengrass 코어 디바이스 간의 MQTT 메시지를 처리합니다. 이 구성 요소는 [EMQX MQTT](#)

[5.0 브로커](#)의 수정된 버전을 제공합니다. 클라이언트 디바이스와 코어 디바이스 간의 통신에서 MQTT 5 기능을 사용하려면 이 MQTT 브로커를 배포하세요. MQTT 브로커를 선택하는 방법에 대한 자세한 내용은 [MQTT 브로커 선택](#) 섹션을 참조하세요.

이 브로커는 MQTT 5.0 프로토콜을 구현합니다. 이 프로토콜에는 세션 및 메시지 만료 간격, 사용자 속성, 공유 구독, 주제 별칭 등에 대한 지원이 포함되어 있습니다. MQTT 5는 하위 버전 MQTT 3.1.1과 호환되므로 [Moquette MQTT 3.1.1 브로커](#)를 실행하는 경우 EMQX MQTT 5 브로커를 바꿀 수 있고 바꾸더라도 클라이언트 디바이스는 평소대로 계속 연결되고 작동할 수 있습니다.

Note

클라이언트 디바이스는 Greengrass 코어 디바이스에 연결되어 처리할 MQTT 메시지와 데이터를 보내는 로컬 IoT 디바이스입니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.0.x
- 1.2.x
- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 허용할 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 요소](#)를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 사용하도록 구성합니다.

다른 포트를 지정하고 [IP 감지기 구성 요소](#)를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 감지기 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 보고하도록 구성합니다.

- Linux 코어 디바이스에서는 다음과 같이 Docker가 코어 디바이스에 설치 및 구성되어 있습니다.
 - Greengrass 코어 디바이스에 [Docker Engine](#) 1.9.1 이상이 설치되어 있어야 합니다. 버전 20.10은 AWS IoT Greengrass 코어 소프트웨어에서 작동하는 것으로 확인된 최신 버전입니다. Docker 컨테이너를 실행하는 구성 요소를 배포하기 전에 코어 디바이스에 직접 Docker를 설치해야 합니다.
 - Docker 데몬이 이 구성 요소를 배포하기 전에 코어 디바이스에서 시작되어 실행 중입니다.
 - 이 구성 요소를 실행하는 시스템 사용자에게 루트 또는 관리자 권한이 있어야 합니다. 또는 `docker` 그룹의 시스템 사용자로 이 구성 요소를 실행하고 권한 없이 EMQX MQTT 브로커를 실행하도록 이 구성 요소의 `requiresPrivileges` 옵션을 `false`로 구성할 수 있습니다.
- EMQX MQTT 브로커 구성 요소는 VPC에서 실행할 수 있습니다.

- EMQX MQTT 브로커 구성 요소는 armv7 플랫폼에서 지원되지 않습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.0.2 – 2.0.3

다음 표에는 이 구성 요소의 버전 2.0.2 및 2.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.6.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.6.0	하드

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
클라이언트 디바이스 인증	>=2.2.0 <2.3.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

2.0.0 - 2.0.3

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

Important

MQTT 5 브로커(EMQX) 구성 요소의 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 Greengrass에서는 기본 구성을 사용합니다. 이 필드를 사용하여 수정하지 않는 한 이 구성이 사용됩니다.

다음 구성 설정을 수정하면 EMQX 브로커 구성 요소가 다시 시작됩니다. 다른 구성 변경 사항은 구성 요소를 다시 시작하지 않고 적용됩니다.

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

Note

`aws.greengrass.clientdevices.mqtt.EMQX`를 사용하면 보안상 중요한 옵션을 구성할 수 있습니다. 여기에는 TLS 설정, 인증 및 권한 부여 공급자가 포함됩니다. 상호 TLS 인증과 Greengrass 클라이언트 디바이스 인증 공급자를 사용하는 기본 구성을 사용하는 것이 좋습니다.

Example예: 기본 구성

다음 예제에서는 MQTT 5(EMQX) 브로커의 기본 설정을 보여줍니다. `emqxConfig` 구성 설정을 사용하여 이러한 설정을 재정의할 수 있습니다.

```
{
  "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"
}
}

```

authMode

(선택 사항) 브로커의 권한 부여 공급자를 설정합니다. 다음 값 중 하나일 수 있습니다.

- **enabled** – (기본값) Greengrass 인증 및 권한 부여 공급자를 사용합니다.
- **bypass_on_failure** – Greengrass 인증 공급자를 사용한 다음 Greengrass에서 인증 또는 권한 부여를 거부하는 경우 EMQX 공급자 체인의 나머지 인증 공급자를 사용합니다.
- **bypass** – Greengrass 공급자가 비활성화되었습니다. 인증 및 권한 부여가 EMQX 공급자 체인에 의해 처리됩니다.

requiresPrivilege

(선택 사항) Linux 코어 장치에서 루트 또는 관리자 권한 없이 EMQX MQTT 브로커를 실행하도록 지정할 수 있습니다. 이 옵션을 **false**로 설정하면 이 구성 요소를 실행하는 시스템 사용자가 **docker** 그룹의 구성원이어야 합니다.

기본값: **true**

startupTimeoutSeconds

(선택 사항) EMQX MQTT 브로커가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 BROKEN으로 변경됩니다.

기본값: 90

ipcTimeoutSeconds

(선택 사항) 구성 요소에서 Greengrass nucleus가 프로세스 간 통신(IPC) 요청에 응답할 때까지 기다리는 최대 시간(초)입니다. 이 구성 요소가 클라이언트 디바이스에 권한이 있는지 확인할 때 제한 시간 오류를 보고하는 경우 이 수를 늘립니다.

기본값: 5

crtLogLevel

(선택 사항) AWS 공통 런타임(CRT) 라이브러리의 로그 수준입니다.

기본값은 EMQX MQTT 브로커 로그 수준(emqx의 log.level)입니다.

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 해제할 수 있습니다.

dockerOptions

(선택 사항) Linux 운영 체제에서만 Docker 명령줄에 파라미터를 추가하려면 이 옵션을 구성합니다. 예를 들어 추가 포트를 매핑하려면 -p Docker 파라미터를 사용합니다.

```
"-p 1883:1883"
```

Example예: v1.x 구성 파일을 v2.x로 업데이트

다음 예제에서는 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"
  }
}

```

이와 동일한 v2의 구성 파일:

```

{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": "8883",
          "max_connections": "1024000",
          "max_conn_rate": "500",
          "ssl_options": {
            "handshake_timeout": "15s"
          }
        }
      }
    },
    "log": {
      "console": {
        "enable": true,
        "level": "warning"
      }
    }
  },
  "authMode": "enabled"
}

```

listener.ssl.external.rate_limit 구성 항목과 동일한 항목은 없습니다.
use_greengrass_managed_certificates 구성 옵션은 제거되었습니다.

Example예: 브로커의 새 포트 설정

다음 예제에서는 MQTT 브로커가 작동하는 포트를 기본값인 8883에서 포트 1234로 변경합니다.
Linux를 사용하는 경우 dockerOptions 필드를 포함합니다.

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example예: MQTT 브로커의 로그 수준 조정

다음 예제에서는 MQTT 브로커의 로그 수준을 debug로 변경합니다. 다음 로그 수준 중에서 선택할 수 있습니다.

- debug
- info
- notice
- warning
- error
- critical
- alert
- emergency

기본 로그 수준은 warning입니다.

```
{
  "emqxConfig": {
    "log": {
      "console": {
        "level": "debug"
      }
    }
  }
}
```

Example예: EMQX 대시보드 활성화

다음 예제에서는 브로커를 모니터링하고 관리할 수 있도록 EMQX 대시보드를 활성화합니다. Linux를 사용하는 경우 dockerOptions 필드를 포함합니다.

```
{
  "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 다음 구성 파라미터를 제공합니다.

emqx

(선택 사항) 사용할 [EMQX MQTT 브로커](#) 구성입니다. 이 구성 요소에서 EMQX 구성 옵션의 하위 집합을 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 요소](#)를 사용하여 MQTT 메시지를 다른 브로커에 릴레이하는 경우 MQTT 브리지 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 사용하도록 구성합니다.

다른 포트를 지정하고 [IP 감지기 구성 요소](#)를 사용하여 MQTT 브로커 엔드포인트를 관리하는 경우 IP 감지기 v2.1.0 이상을 사용해야 합니다. 이 구성 요소에서 MQTT 브로커가 작동하는 포트를 보고하도록 구성합니다.

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 제한입니다.

`bandwidth,duration`의 형식으로 대역폭과 해당 대역폭의 기간을 쉼표(,)로 구분하여 지정합니다. 예를 들어 5초마다 50KB(킬로바이트)의 데이터로 MQTT 브로커를 제한하려면 50KB,5s를 지정할 수 있습니다.

`listener.ssl.external.handshake_timeout`

(선택 사항) MQTT 브로커가 새 연결 인증이 완료될 때까지 대기하는 시간입니다.

기본값: 15s

`mqtt.max_packet_size`

(선택 사항) MQTT 메시지의 최대 크기입니다.

기본값: 268435455(256MB - 1)

`log.level`

(선택 사항) MQTT 브로커의 로그 수준입니다. 다음 옵션 중 하나를 선택합니다.

- debug
- info
- notice
- warning
- error
- critical
- alert

- emergency

기본 로그 수준은 warning입니다.

requiresPrivilege

(선택 사항) Linux 코어 장치에서 루트 또는 관리자 권한 없이 EMQX MQTT 브로커를 실행하도록 지정할 수 있습니다. 이 옵션을 false로 설정하면 이 구성 요소를 실행하는 시스템 사용자가 docker 그룹의 구성원이어야 합니다.

기본값: true

startupTimeoutSeconds

(선택 사항) EMQX MQTT 브로커가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 BROKEN으로 변경됩니다.

기본값: 90

ipcTimeoutSeconds

(선택 사항) 구성 요소에서 Greengrass nucleus가 프로세스 간 통신(IPC) 요청에 응답할 때까지 기다리는 최대 시간(초)입니다. 이 구성 요소가 클라이언트 디바이스에 권한이 있는지 확인할 때 제한 시간 오류를 보고하는 경우 이 수를 늘립니다.

기본값: 5

crtLogLevel

(선택 사항) AWS 공통 런타임(CRT) 라이브러리의 로그 수준입니다.

기본값은 EMQX MQTT 브로커 로그 수준(emqx의 log.level)입니다.

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 해제할 수 있습니다.

dockerOptions

(선택 사항) Linux 운영 체제에서만 Docker 명령줄에 파라미터를 추가하려면 이 옵션을 구성합니다. 예를 들어 추가 포트를 매핑하려면 -p Docker 파라미터를 사용합니다.

```
"-p 1883:1883"
```

mergeConfigurationFiles

(선택 사항) 지정된 EMQX 구성 파일의 기본값에 추가하거나 기본값을 재정의하려면 이 옵션을 구성합니다. 구성 파일 및 해당 형식에 대한 자세한 내용은 EMQX 4.0 Documentation의 [Configuration](#)을 참조하세요. 지정하는 값이 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 Greengrass에서는 EMQX에 대한 Greengrass 인증 플러그인을 구성하는 `etc/plugins/aws_greengrass_emqx_auth.conf`라는 파일도 지원합니다. 지원되는 두 가지 옵션은 `auth_mode` 및 `use_greengrass_managed_certificates`입니다. 다른 인증 공급자를 사용하려면 `auth_mode` 옵션을 다음 중 하나로 설정합니다.

- `enabled` – (기본값) Greengrass 인증 및 권한 부여 공급자를 사용합니다.
- `bypass_on_failure` – Greengrass 인증 공급자를 사용한 다음 Greengrass에서 인증 또는 권한 부여를 거부하는 경우 EMQX 공급자 체인의 나머지 인증 공급자를 사용합니다.
- `bypass` – Greengrass 공급자가 비활성화되었습니다. 그러면 인증 및 권한 부여는 EMQX 공급자 체인에 의해 처리됩니다.

`use_greengrass_managed_certificates`가 `true`이면 Greengrass가 브로커 TLS 인증서를 관리합니다. `false`인 경우에는 다른 소스를 통해 인증서를 제공합니다.

다음 예제에서는 `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.EMQX`를 사용하면 보안상 중요한 옵션을 구성할 수 있습니다. 여기에는 TLS 설정, 인증 및 권한 부여 공급자가 포함됩니다. 상호

TLS 인증과 Greengrass 클라이언트 디바이스 인증 공급자를 사용하는 기본 구성이 권장 구성입니다.

replaceConfigurationFiles

(선택 사항) 지정된 EMQX 구성 파일을 바꾸려면 이 옵션을 구성합니다. 지정하는 값으로 기존 구성 파일 전체가 대체됩니다. 이 섹션에서는 `etc/emqx.conf` 파일을 지정할 수 없습니다. `etc/emqx.conf`를 수정하려면 `mergeConfigurationFile`을 사용해야 합니다.

Example에: 구성 병합 업데이트

다음 예제 구성에서는 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 조건 - Microsoft Visual Studio Community 2022](#)에 따라 배포된 코드가 포함됩니다. 이 소프트웨어를 다운로드하면 해당 코드의 라이선스 조건에 동의하는 것입니다.

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

v2.x

버전	변경 사항
2.0.3	버그 수정 및 개선 사항 경로에 공백이 포함된 경우 Windows에서 EMQX가 시작되지 않는 문제를 수정합니다.
2.0.2	버그 수정 및 개선 사항 클라이언트 디바이스 인증 구성 요소가 준비되기 전에 EMQX가 먼저 시작되는 문제를 수정합니다.

버전	변경 사항
2.0.1	클라이언트 디바이스 인증 버전 2.5.0 릴리스에 대한 버전이 업데이트되었습니다.
2.0.0	이 버전의 MQTT 5 브로커(EMQX)에는 버전 1.x와 다른 구성 파라미터가 필요합니다. 버전 1.x에 기본값이 아닌 구성을 사용하고 있는 경우 2.x에 맞게 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 구성 단원을 참조하십시오. 새로운 기능 • MQTT 브로커가 EMQX 5.1.1로 업그레이드됩니다. • 구성 요소를 다시 시작하지 않고 브로커 구성 변경 사항이 활성화됩니다. 업데이트 • <code>emqx</code>, <code>mergeConfigurationFiles</code> 및 <code>replaceConfigurationFiles</code> 구성 필드를 대체하는 새 <code>emqxConfig</code> 구성 필드가 추가됩니다.

v1.x

버전	변경 사항
1.2.3	버그 수정 및 개선 사항 • 이전에 인증한 후 클라이언트가 EMQX와 연동할 수 없던 문제가 클라이언트를 연결 해제하고 다시 인증하여 해결됩니다.
1.2.2	클라이언트 디바이스 인증 버전 2.4.0 릴리스에 대한 버전이 업데이트되었습니다.
1.2.1	버그 수정 및 개선 사항 • Visual C++ 재배포 가능 패키지가 아직 없는 경우 Windows에서 구성 요소가 시작되지 않는 문제를 해결합니다. • EMQX가 버전 4.4.14로 업데이트됩니다.
1.2.0	인증서 체인에 대한 지원을 추가합니다.

버전	변경 사항
1.1.0	새로운 기능 - 브로커 옵션 및 플러그인을 포함한 EMQX 구성에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - EMQX가 버전 4.4.9로 업데이트됩니다.
1.0.1	TLS 핸드셰이크 중에 일부 MQTT 클라이언트가 연결되지 않는 문제를 해결합니다.
1.0.0	초기 버전입니다.

nucleus 원격 측정 이미터

nucleus 원격 측정 이미터 구성 요소(`aws.greengrass.telemetry.NucleusEmitter`)는 시스템 상태 원격 측정 데이터를 수집하여 로컬 주제 및 AWS IoT Core MQTT 주제에 지속적으로 게시합니다. 이 구성 요소를 사용하면 Greengrass 코어 디바이스에서 실시간 시스템 원격 측정 데이터를 수집할 수 있습니다. Amazon EventBridge에 시스템 원격 측정 데이터를 게시하는 Greengrass 원격 측정 에이전트에 대한 자세한 내용은 [AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집](#) 섹션을 참조하세요.

기본적으로 nucleus 원격 측정 이미터 구성 요소는 60초마다 다음 로컬 게시/구독 주제에 원격 측정 데이터를 게시합니다.

```
$local/greengrass/telemetry
```

nucleus 원격 측정 이미터 구성 요소는 기본적으로 AWS IoT Core MQTT 주제에 게시되지 않습니다. 배포 시 AWS IoT Core MQTT 주제에 게시하도록 이 구성 요소를 구성할 수 있습니다. MQTT 주제를 사용하여 데이터를 게시하는 데는 [AWS IoT Core 요금](#)이 AWS 클라우드 적용됩니다.

AWS IoT Greengrass 는 InfluxDB 및 Grafana를 사용하여 코어 디바이스에서 로컬로 원격 측정 데이터를 분석하고 시각화하는 데 도움이 되는 여러 [커뮤니티 구성 요소를](#) 제공합니다. 이러한 구성 요소는 nucleus 이미터 구성 요소의 원격 측정 데이터를 사용합니다. 자세한 내용은 [InfluxDB 게시자 구성 요소](#)에 대한 README를 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [종속성](#)
- [구성](#)
- [출력 데이터](#)
- [사용법](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

1.0.12

다음 표에는 이 구성 요소의 버전 1.0.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.17.0	소프트

1.0.11

다음 표에는 이 구성 요소의 버전 1.0.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.16.0	하드

1.0.10

다음 표에는 이 구성 요소의 버전 1.0.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.15.0	하드

1.0.9

다음 표에는 이 구성 요소의 버전 1.0.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.14.0	하드

1.0.8

다음 표에는 이 구성 요소의 버전 1.0.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.13.0	하드

1.0.7

다음 표에는 이 구성 요소의 버전 1.0.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.12.0	하드

1.0.6

다음 표에는 이 구성 요소의 버전 1.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.11.0	하드

1.0.5

다음 표에는 이 구성 요소의 버전 1.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.10.0	하드

1.0.4

다음 표에는 이 구성 요소의 버전 1.0.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.9.0	하드

1.0.3

다음 표에는 이 구성 요소의 버전 1.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.8.0	하드

1.0.2

다음 표에는 이 구성 요소의 버전 1.0.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.7.0	하드

1.0.1

다음 표에는 이 구성 요소의 버전 1.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.6.0	하드

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.4.0 <2.5.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

pubSubPublish

(선택 사항) 원격 측정 데이터를 \$local/greengrass/telemetry 주제에 게시할지 여부를 정의합니다. 지원되는 값은 true 및 false입니다.

기본값: true

mqttTopic

(선택 사항)이 구성 요소가 원격 측정 데이터를 게시하는 AWS IoT Core MQTT 주제입니다.

이 값을 원격 측정 데이터를 게시하려는 AWS IoT Core MQTT 주제로 설정합니다. 이 값이 비어 있으면 nucleus 이미터는 원격 측정 데이터를 AWS 클라우드에 게시하지 않습니다.

Note

MQTT 주제를 사용하여 데이터를 게시하는 AWS 클라우드 데는 [AWS IoT Core 요금](#)이 적용됩니다.

기본값: ""

telemetryPublishIntervalMs

(선택 사항) 구성 요소가 원격 측정 데이터를 게시하는 시간 간격(밀리초)입니다. 이 값을 지원하는 최소 값보다 낮게 설정하면 구성 요소에서 대신 최소값을 사용합니다.

Note

게시 간격이 짧으면 코어 디바이스의 CPU 사용량이 늘어납니다. 기본 게시 간격으로 시작하여 디바이스의 CPU 사용량에 따라 조정하는 것이 좋습니다.

최소: 500

기본값: 60000

Example에: 구성 병합 업데이트

다음 예제에서는 `$local/greengrass/telemetry` 주제 및 `greengrass/myTelemetry` AWS IoT Core MQTT 주제에 5초마다 원격 측정 데이터를 게시할 수 있는 샘플 구성 병합 업데이트를 보여줍니다.

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

출력 데이터

이 구성 요소는 원격 측정 지표를 다음 주제에 대한 JSON 배열로 게시합니다.

로컬 주제: `$local/greengrass/telemetry`

선택적으로 원격 측정 지표를 AWS IoT Core MQTT 주제에 게시하도록 선택할 수도 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT topics](#)를 참조하세요.

Example에시 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 다음 속성을 가진 지표 목록이 포함됩니다.

A

지표에 대한 집계 유형입니다.

CpuUsage 지표의 경우 지표의 게시된 값이 마지막 게시 이벤트 이후의 평균 CPU 사용량이므로 이 속성은 Average로 설정됩니다.

다른 모든 지표의 경우 nucleus 이미터에서 지표 값을 집계하지 않으며 이 속성은 Count로 설정됩니다.

N

지표의 이름.

NS

지표 네임스페이스입니다.

TS

데이터가 수집된 시기의 타임스탬프입니다.

U

지표 값의 단위.

V

지표 값.

nucleus 이미터는 다음 지표를 게시합니다.

명칭	설명	
시스템		
SystemMemUsage	운영 체제를 포함하여 Greengrass 코어 장치의 모든 애플리케이션에서 현재 사용 중인 메모리의 양입니다.	
CpuUsage	운영 체제를 포함하여 Greengrass 코어 장치의 모든	

명칭	설명	
	애플리케이션에서 현재 사용 중인 CPU의 양입니다.	
TotalNumberOfFDs	Greengrass 코어 장치의 운영 체제에 저장된 파일 디스크립터 수입입니다. 하나의 파일 디스크립터는 열려 있는 파일 하나를 고유하게 식별합니다.	
Greengrass nucleus		
NumberOfComponentsRunning	Greengrass 코어 디바이스에서 실행 중인 구성 요소의 수입입니다.	
NumberOfComponentsErrored	Greengrass 코어 디바이스에서 오류 상태에 있는 구성 요소의 수입입니다.	
NumberOfComponentsInstalled	Greengrass 코어 디바이스에 설치된 구성 요소의 수입입니다.	
NumberOfComponentsStarting	Greengrass 코어 디바이스에서 시작하는 구성 요소의 수입입니다.	
NumberOfComponentsNew	Greengrass 코어 디바이스의 새로운 구성 요소의 수입입니다.	
NumberOfComponentsStopping	Greengrass 코어 디바이스에서 중지하는 구성 요소의 수입입니다.	
NumberOfComponentsFinished	Greengrass 코어 디바이스에서 완료된 구성 요소의 수입입니다.	

명칭	설명	
NumberOfComponents Broken	Greengrass 코어 디바이스에서 손상된 구성 요소의 수입니다.	
NumberOfComponents Stateless	Greengrass 코어 디바이스에 있는 상태 비저장 구성 요소의 수입니다.	

사용법

시스템 상태 원격 측정 데이터를 사용하려면 nucleus 이미터가 원격 측정 데이터를 게시하는 주제를 구독하는 사용자 지정 구성 요소를 생성하고 필요에 따라 해당 데이터에 대응할 수 있습니다. nucleus 이미터 구성 요소에서 원격 측정 데이터를 로컬 주제에 게시하는 옵션을 제공하므로 해당 주제를 구독하고 게시된 데이터를 사용하여 코어 디바이스에서 로컬로 작업할 수 있습니다. 그러면 코어 디바이스는 클라우드에 대한 연결이 제한된 경우에도 원격 측정 데이터에 대응할 수 있습니다.

예를 들어 `$local/greengrass/telemetry` 주제에서 원격 측정 데이터를 수신하는 구성 요소를 구성하고 스트림 관리자 구성 요소에 데이터를 전송하여 데이터를 AWS 클라우드로 스트리밍할 수 있습니다. 이러한 구성 요소를 생성하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 및 [스트림 관리자 사용하는 사용자 지정 구성 요소 생성](#) 섹션을 참조하세요.

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
1.0.12	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
1.0.11	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
1.0.10	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
1.0.9	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
1.0.8	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
1.0.7	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
1.0.6	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
1.0.5	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
1.0.4	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
1.0.3	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
1.0.2	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
1.0.1	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
1.0.0	초기 버전입니다.

PKCS#11 공급자

PKCS#11 공급자 구성 요소(`aws.greengrass.crypto.Pkcs11Provider`)를 사용하면 [PKCS#11 인터페이스](#)를 통해 HSM(하드웨어 보안 모듈)을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 이 구성 요소를 사용하면 인증서와 프라이빗 키를 소프트웨어에 노출되거나 복제되지 않도록 안전하게 저장할 수 있습니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.

인증서와 프라이빗 키를 HSM에 저장하는 Greengrass 코어 디바이스를 프로비저닝하려면 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 이 구성 요소를 프로비저닝 플러그인으로 지정해야 합니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

AWS IoT Greengrass 는 설치 중에 프로비저닝 플러그인으로 지정하기 위해 다운로드할 수 있는 JAR 파일로 구성된 구성 요소를 제공합니다. 다음 URL(<https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>)에서 최신 버전의 구성 요소 JAR 파일을 다운로드할 수 있습니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 키를 사용하는 RSA 키가 지원되는 하드웨어 보안 모듈.

Note

ECC 키와 함께 하드웨어 보안 모듈을 사용하려면 [Greengrass nucleus](#) v2.5.6 이상을 사용해야 합니다.

하드웨어 보안 모듈 및 [보안 암호 관리자](#)를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

- AWS IoT Greengrass 코어 소프트웨어가 런타임에 로드(`libdl` 사용)하여 PKCS#11 함수를 호출할 수 있는 PKCS#11 공급자 라이브러리입니다. PKCS#11 공급자 라이브러리는 다음 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.
 - 프라이빗 키와 인증서를 동일한 슬롯의 HSM에 저장해야 하고, HSM이 객체 ID를 지원하는 경우 동일한 객체 레이블과 객체 ID를 사용해야 합니다.
 - 인증서와 프라이빗 키는 객체 레이블로 확인할 수 있어야 합니다.

- sign
- decrypt
- (선택 사항) [보안 암호 관리자 구성 요소](#)를 사용하려면 버전 2.1.0 이상을 사용해야 하며, 프라이빗 키에 다음 권한이 있어야 합니다.
- unwrap
- wrap
- (선택 사항) TPM2 라이브러리를 사용하고 Greengrass 코어를 서비스로 실행하는 경우 PKCS#11 스토어의 위치가 포함된 환경 변수를 제공해야 합니다. 다음 예제는 필수 환경 변수가 있는 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
```

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.0.11

다음 표에는 이 구성 요소의 버전 2.0.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.17.0	소프트

2.0.10

다음 표에는 이 구성 요소의 버전 2.0.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.16.0	소프트

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.15.0	소프트

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.14.0	소프트

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.13.0	소프트

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.12.0	소프트

2.0.5

다음 표에는 이 구성 요소의 버전 2.0.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.11.0	소프트

2.0.4

다음 표에는 이 구성 요소의 버전 2.0.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.10.0	소프트

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.9.0	소프트

2.0.2

다음 표에는 이 구성 요소의 버전 2.0.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.8.0	소프트

2.0.1

다음 표에는 이 구성 요소의 버전 2.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.7.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.3 <2.6.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

name

PKCS#11 구성의 이름입니다.

library

AWS IoT Greengrass 코어 소프트웨어가 libdl로 로드할 수 있는 PKCS#11 구현 라이브러리의 절대 파일 경로입니다.

slot

프라이빗 키와 디바이스 인증서가 포함된 슬롯의 ID입니다. 이 값은 슬롯 인덱스 또는 슬롯 레이블과 다릅니다.

userPin

솔루션 액세스에 사용할 사용자 PIN입니다.

Example에: 구성 병합 업데이트

```
{
  "name": "softsm_pkcs11",
  "library": "/usr/lib/softsm/libsoftsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.0.11	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.0.10	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.0.9	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.0.8	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.0.2	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.0.1	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.0.0	초기 버전입니다.

보안 암호 관리자

보안 암호 관리자 구성 요소(`aws.greengrass.SecretManager`)는 Greengrass 코어 디바이스에 AWS Secrets Manager의 보안 암호를 배포합니다. 이 구성 요소를 사용하면 Greengrass 코어 디바이스의 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 지정 Greengrass 구성 요소에서 이 구성 요소의 보안 암호에 액세스하려면 AWS IoT Device SDK에서 [GetSecretValue](#) 작업을 사용합니다. 자세한 내용은 [AWS IoT Device SDK 를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core](#) 및 [보안 암호 값 검색](#) 섹션을 참조하세요.

이 구성 요소는 코어 디바이스의 보안 암호를 암호화하여 자격 증명과 보안 암호를 사용해야 할 때까지 이를 안전하게 보관합니다. 코어 디바이스의 프라이빗 키를 사용하여 보안 암호를 암호화하고 해독합니다.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(aws.greengrass.plugin)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)은 `secretsmanager:GetSecretValue` 작업을 허용해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:MySecret"
      ]
    }
  ]
}
```

Note

고객 관리형 AWS Key Management Service 키를 사용하여 보안 암호를 암호화하는 경우 디바이스 역할도 `kms:Decrypt` 작업을 허용해야 합니다.

Secrets Manager의 IAM 정책에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서에서 다음을 참조하세요.

- [에 대한 인증 및 액세스 제어 AWS Secrets Manager](#)
- [Actions, resources, and context keys you can use in an IAM policy or secret policy for AWS Secrets Manager](#)
- 사용자 지정 구성 요소는 `aws.greengrass#GetSecretValue`가 이 구성 요소에 저장한 보안 암호를 가져올 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 이 권한 부여 정책에서 특정 보안 암호에 대한 구성 요소의 액세스를 제한할 수 있습니다. 자세한 내용은 [보안 암호 관리자 IPC 권한 부여](#)를 참조하세요.
- (선택 사항) 코어 디바이스의 프라이빗 키와 인증서를 [하드웨어 보안 모듈\(HSM\)](#)에 저장하는 경우 HSM은 RSA 키를 지원해야 하며 프라이빗 키에는 `unwrap` 권한이 있어야 하고 퍼블릭 키에는 `wrap` 권한이 있어야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
<code>secretsmanager. <i>region</i>.amazonaws.com</code>	443	예	코어 디바이스에 보안 암호를 다운로드합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.2.7

다음 표에는 이 구성 요소의 버전 2.2.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.17.0	소프트

2.2.6

다음 표에는 이 구성 요소의 버전 2.2.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.16.0	소프트

2.2.2 – 2.2.5

다음 표에는 이 구성 요소의 버전 2.2.2~2.2.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.15.0	소프트

2.2.0

다음 표에는 이 구성 요소의 버전 2.2.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.13.0 <2.14.0	소프트

2.1.7 – 2.1.8

다음 표에는 이 구성 요소의 버전 2.1.7 및 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.13.0	소프트

2.1.6

다음 표에는 이 구성 요소의 버전 2.1.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.12.0	소프트

2.1.5

다음 표에는 이 구성 요소의 버전 2.1.5에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.11.0	소프트

2.1.4

다음 표에는 이 구성 요소의 버전 2.1.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.10.0	소프트

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.9.0	소프트

2.1.2

다음 표에는 이 구성 요소의 버전 2.1.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.8.0	소프트

2.1.1

다음 표에는 이 구성 요소의 버전 2.1.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.7.0	소프트

2.1.0

다음 표에는 이 구성 요소의 버전 2.1.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.6.0	소프트

2.0.9

다음 표에는 이 구성 요소의 버전 2.0.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

2.0.7

다음 표에는 이 구성 요소의 버전 2.0.7에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

2.0.6

다음 표에는 이 구성 요소의 버전 2.0.6에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

periodicRefreshIntervalMin (선택 사항)

이 구성 요소가 코어 디바이스에 구성된 보안 암호를 AWS Secrets Manager 서비스의 최신 보안 암호 값과 동기화하는 분 단위 간격입니다. 이 간격이 구성되지 않은 경우 보안 암호 관리자가 구성된 보안 암호를 주기적으로 새로 고치지 않습니다.

```
{
  "cloudSecrets": [
```

```
{
  "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef"
},
"periodicRefreshIntervalMin" : 60
}
```

cloudSecrets

코어 디바이스에 배포할 Secrets Manager 보안 암호의 목록. 레이블을 지정하여 배포할 각 암호의 버전을 정의할 수 있습니다. 버전을 지정하지 않으면 이 구성 요소는 스테이징 레이블 `AWSCURRENT`가 연결된 버전을 배포합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 [Staging labels](#)를 참조하세요.

보안 암호 관리자 구성 요소는 보안 암호를 로컬에 캐시합니다. Secrets Manager에서 보안 암호 값이 변경되는 경우 이 구성 요소는 새로운 값을 자동으로 검색하지 않습니다. 로컬 복사본을 업데이트하려면 암호에 새 레이블을 지정하고 새 레이블로 식별된 암호를 검색하도록 이 구성 요소를 구성합니다.

각 객체에는 다음 정보가 포함되어 있습니다.

arn

배포할 암호의 ARN. 암호의 ARN은 전체 ARN 또는 부분 ARN일 수 있습니다. 부분 ARN이 아닌 전체 ARN을 지정하는 것이 좋습니다. 자세한 내용은 [Finding a secret from a partial ARN](#)을 참조하세요. 다음은 전체 ARN과 부분 ARN의 예제입니다.

- 전체 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 암호의 버전을 식별하기 위한 레이블의 목록.

각 레이블은 문자열이어야 합니다.

Example예: 구성 병합 업데이트

```
{
```



```
"cloudSecrets": [  
  {  
    "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-  
abcdef"  
  }  
]
```

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.2.7	Greengrass nucleus 버전 2.16.0 릴리스용으로 업데이트된 버전입니다.
2.2.6	버그 수정 및 개선 사항 TPM(Trusted Platform Module)이 느리거나 응답하지 않아 보안 암호 관리자가 비밀을 가져오지 못하는 문제를 수정합니다. Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.2.5	버그 수정 및 개선 사항 로컬 캐시에 보안 암호가 없는 AWS 클라우드 경우에서 보안 암호를 가져오지 않는 문제를 해결합니다.
2.2.4	버그 수정 및 개선 사항 로컬 암호 저장소에 쓰는 빈도를 줄입니다. 이제 보안 암호 관리자는 보안 암호가 업데이트될 때만 로컬 저장소에 기록합니다.
2.2.3	버그 수정 및 개선 사항 코어 디바이스가 오프라인 상태이고 디바이스의 보안 서비스(HSM 등)를 사용할 수 없을 때 보안 암호 관리자가 로컬에 저장된 비밀을 삭제하는 문제를 수정했습니다.
2.2.2	버그 수정 및 개선 사항 보안 암호 관리자가 부분 arn으로 구성된 보안 암호를 다운로드하지 않는 문제를 수정합니다.
2.2.1	버그 수정 및 개선 사항 nucleus 버전 2.5.0 이상에서 보안 암호 관리자를 지원합니다.
2.2.0	새로운 기능 새로운 구성 요소 구성 키를 통해 구성된 보안 암호를 주기적으로 새로 고치는 지원을 추가합니다.

버전	변경 사항
	요청당 보안 암호를 새로 고치기 위해 GetSecretValue IPC 요청에서 새로운 요청 파라미터에 대한 지원을 추가합니다.
2.1.8	버그 수정 및 개선 사항 보안 암호 관리자가 부분 ARN을 수락하지 않는 문제를 해결합니다.
2.1.7	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	버그 수정 및 개선 사항 보안 암호 관리자가 배포되고 Greengrass nucleus가 다시 시작될 때 캐시된 보안 암호가 제거되는 문제를 해결합니다. Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 - 하드웨어 보안 통합에 대한 지원을 추가합니다. 보안 암호 관리자 구성 요소는 하드웨어 보안 모듈(HSM)에 저장하는 프라이빗 키를 사용하여 보안 암호를 암호화하고 해독할 수 있습니다. 자세한 내용은 하드웨어 보안 통합 단원을 참조하십시오. 버그 수정 및 개선 사항 - Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.9	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.8	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.0.7	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	개선 사항 AWS 중국 리전 및 AWS GovCloud (US) 리전에 대한 지원을 추가합니다.
2.0.4	초기 버전입니다.

보안 터널링

`aws.greengrass.SecureTunneling` 구성 요소를 사용하면 제한된 방화벽 뒤에 있는 Greengrass 코어 디바이스와 안전한 양방향 통신을 설정할 수 있습니다.

예를 들어 방화벽 뒤에 들어오는 모든 연결을 금지하는 Greengrass 코어 디바이스가 있다고 가정해 보겠습니다. 보안 터널링은 MQTT를 사용하여 액세스 토큰을 디바이스로 전송한 다음 WebSockets를 사용하여 방화벽을 통해 디바이스에 SSH를 연결합니다. 이 AWS IoT 관리형 터널을 사용하면 디바이스에 필요한 SSH 연결을 열 수 있습니다. AWS IoT 보안 터널링을 사용하여 원격 디바이스에 연결하는 방법에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [AWS IoT 보안 터널링](#)을 참조하세요.

이 구성 요소는 `$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](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [사용법](#)
- [다음 사항도 참조하세요.](#)

- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

아키텍처:

- Armv71
- ARMv8(AArch64)
- x86_64

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- 보안 터널링 구성 요소에 사용할 수 있는 최소 디스크 공간은 32MB입니다. 이 요구 사항에는 동일한 디바이스에서 실행되는 Greengrass 코어 소프트웨어 또는 기타 구성 요소가 포함되지 않습니다.
- 보안 터널링 구성 요소에 사용할 수 있는 최소 RAM은 16MB입니다. 이 요구 사항에는 동일한 디바이스에서 실행되는 Greengrass 코어 소프트웨어 또는 기타 구성 요소가 포함되지 않습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.
- 보안 터널링 구성 요소 버전 1.0.12 이상에는 Linux 커널이 3.2 이상인 GNU C 라이브러리(glibc) 버전 2.25 이상이 필요합니다. 장기 지원 종료 날짜가 지난 운영 체제 및 라이브러리 버전은 지원되지 않습니다. 장기 지원과 함께 운영 체제 및 라이브러리를 사용해야 합니다.

- 운영 체제와 Java 런타임은 모두 64비트로 설치되어 있어야 합니다.
- Greengrass 코어 디바이스에 [Python](#) 버전 3.5 이상이 설치되고 PATH 환경 변수에 추가되어 있어야 합니다.
- `libcrypto.so.1.1`이 Greengrass 코어 디바이스에 설치되고 PATH 환경 변수에 추가되어 있어야 합니다.
- Greengrass 코어 디바이스의 포트 443에서 아웃바운드 트래픽을 엽니다.
- Greengrass 코어 디바이스와 통신하는 데 사용할 통신 서비스에 대한 지원을 엽니다. 예를 들어 디바이스에 대한 SSH 연결을 열려면 해당 디바이스에서 SSH를 켜야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	Port	필수	설명
<code>data.tunneling.iot</code> . <i>region</i> .amazonaws.com	443	예	보안 터널을 설정합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

1.0.19 – 1.1.3

다음 표에는 이 구성 요소의 버전 1.0.19~1.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	소프트

1.0.18

다음 표에는 이 구성 요소의 버전 1.0.18에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	소프트

1.0.10

다음 표에는 이 구성 요소의 버전 1.0.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	소프트

1.0.9

다음 표에는 이 구성 요소의 버전 1.0.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	소프트

1.0.8

다음 표에는 이 구성 요소의 버전 1.0.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트

1.0.4

다음 표에는 이 구성 요소의 버전 1.0.4에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	소프트

1.0.3

다음 표에는 이 구성 요소의 버전 1.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	소프트

1.0.2

다음 표에는 이 구성 요소의 버전 1.0.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

1.0.1

다음 표에는 이 구성 요소의 버전 1.0.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	소프트

1.0.0

다음 표에는 이 구성 요소의 버전 1.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

OS_DIST_INFO

(선택 사항) 코어 디바이스의 운영 체제. 기본적으로 구성 요소는 코어 디바이스에서 실행 중인 운영 체제를 자동으로 식별하려고 시도합니다. 구성 요소가 기본값으로 시작되지 않는 경우 이 값을 사용하여 운영 체제를 지정합니다. 이 구성 요소에 대해 지원되는 운영 체제의 목록은 [디바이스 요구 사항](#) 섹션을 참조하세요.

이 값은 auto, ubuntu, amzn2, raspberrypi 중 하나일 수 있습니다.

기본값: auto

accessControl

(선택 사항) 구성 요소가 보안 터널링 알림 주제에 메시지를 구독할 수 있도록 허용하는 [권한 부여 정책](#)이 있는 객체입니다.

 Note

배포가 사물 그룹을 대상으로 하는 경우 이 구성 파라미터를 수정하지 마세요. 배포가 개별 코어 디바이스를 대상으로 하고 해당 디바이스의 주제에 대한 구독을 제한하려는 경우 코어 디바이스의 사물 이름을 지정합니다. 디바이스의 권한 부여 정책의 resources 값에서 MQTT 주제 와일드카드를 디바이스의 사물 이름으로 바꿉니다.

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

Example예: 구성 병합 업데이트

다음 예제 구성은 이 구성 요소가 Ubuntu를 실행하는 **MyGreengrassCore**라는 코어 디바이스에서 보안 터널을 열 수 있도록 지정합니다.

```
{
  "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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. 를 AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

라이선스

이 구성 요소에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- [AWS IoT 디바이스 클라이언트](#)/Apache 라이선스 2.0

- [AWS IoT Device SDK for Java](#)/Apache 라이선스 2.0
- [gson](#)/Apache 라이선스 2.0
- [log4j](#)/Apache 라이선스 2.0
- [slf4j](#)/Apache 라이선스 2.0

사용법

디바이스에서 보안 터널링 구성 요소를 사용하려면 다음을 수행합니다.

1. 디바이스에 보안 터널링 구성 요소를 배포합니다.
2. [AWS IoT 콘솔](#)을 엽니다. 왼쪽 메뉴에서 원격 작업을 선택한 다음 보안 터널을 선택합니다.
3. Greengrass 디바이스에 대한 터널을 생성합니다.
4. 소스 액세스 토큰을 다운로드합니다.
5. 소스 액세스 토큰과 함께 로컬 프록시를 사용하여 대상에 연결합니다. 자세한 내용은 AWS IoT 개발자 안내서의 [How to use the local proxy](#)를 참조하세요.

다음 사항도 참조하세요.

- AWS IoT 개발자 안내서의 [AWS IoT 보안 터널링](#)
- AWS IoT 개발자 안내서의 [How to use the local proxy](#)

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
1.1.3	버그 수정 및 개선 사항 • 구성 요소가 호출한 기본 AWS IoT Device Client를 버전 1.10.0에서 버전 1.10.1로 업그레이드합니다. • 보안 터널링 구성 요소 v1.1.2에서 발생하던 GNU C 라이브러리(glibc) 호환성 문제를 수정했습니다.

버전	변경 사항
1.1.2	 Warning 이 버전은 더 이상 사용할 수 없습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 사항 - 구성 요소가 간접 호출하는 기본 AWS IoT Device Client를 버전 1.9.0에서 버전 1.10.0으로 업그레이드합니다. - Greengrass V2 코어 디바이스에서 소스 디바이스로 대용량 파일을 보안 터널을 통해 전달할 수 없던 페이로드 전송 문제를 수정했습니다.
1.1.1	버그 수정 및 개선 사항 Greengrass nucleus lite를 지원하기 위한 구성을 추가합니다.
1.1.0	새로운 기능 Greengrass nucleus lite용 레시피 지원을 추가했습니다.
1.0.19	버그 수정 및 개선 사항 - 구성 요소가 간접 호출하는 기본 AWS IoT Device Client를 버전 1.8.0에서 버전 1.9.0으로 업그레이드합니다. - 구성 요소 수준에서 동시 터널 제한을 20 터널로 늘립니다. - 기본 AWS IoT Greengrass 코어 IPC 제한 시간을 3초에서 10초로 늘립니다.  Warning 보안 터널링 로컬 프록시를 터널 소스 클라이언트로 사용하는 경우 로컬 프록시를 버전 3.1.1 이상으로 업그레이드할 때까지 이 버전으로 구성 요소를 업데이트하지 마세요.
1.0.18	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
1.0.17	버그 수정 및 개선 사항 • 사용자가 터널을 생성하지 못하도록 차단하는 스레드 정리 문제를 해결합니다. 이제 이 구성 요소는 CloseTunnel 신호를 수신하거나 터널이 12 시간 후에 만료되는 경우 스레드를 정리합니다.
1.0.16	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
1.0.15	버그 수정 및 개선 사항 • 디바이스에 홈 디렉터리가 없는 사용자의 시작 문제를 해결합니다. 이제 새도 문서용 디렉터리를 생성하지 않고도 보안 터널링 구성 요소가 시작됩니다.
1.0.14	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
1.0.13	버그 수정 및 개선 사항 • 고아 클라이언트 프로세스가 둘 이상의 터널이 디바이스를 대상으로 하지 못하도록 하는 문제를 해결합니다.
1.0.12	버그 수정 및 개선 사항 • Raspberry Pi OS에서 실행하는 경우 x86_64(AMD64) 및 ARMv8(Aarch64)에 대한 지원을 추가합니다.
1.0.11	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
1.0.10	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
1.0.9	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
1.0.8	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
1.0.7	버그 수정 및 개선 사항 • SCP를 통해 대용량 파일을 전송하는 경우 구성 요소의 연결이 끊기는 문제를 해결합니다.
1.0.6	이 버전에는 버그 수정이 포함되어 있습니다.
1.0.5	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
1.0.4	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
1.0.3	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
1.0.2	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
1.0.1	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
1.0.0	초기 버전입니다.

새도 관리자

새도 관리자 구성 요소(`aws.greengrass.ShadowManager`)는 코어 디바이스에서 로컬 새도 서비스를 활성화합니다. 로컬 새도 서비스를 사용하면 구성 요소가 프로세스 간 통신을 사용하여 [로컬 새도와 상호 작용](#)할 수 있습니다. 새도 관리자 구성 요소는 로컬 새도 문서의 스토리지를 관리하고 로컬 새도 상태를 AWS IoT 디바이스 새도 서비스와 동기화하는 작업도 처리합니다.

Greengrass 코어 디바이스가 새도와 상호 작용하는 방법에 대한 자세한 내용은 [디바이스 새도와 연동](#) 섹션을 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.3.x

- 2.2.x
- 2.1.x
- 2.0.x

유형

이 구성 요소는 플러그인 구성 요소(`aws.greengrass.plugin`)입니다. [Greengrass nucleus](#)는 이 구성 요소를 nucleus와 동일한 Java 가상 머신(JVM)에서 실행합니다. 코어 디바이스에서 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다.

이 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- (선택 사항) 새도우를 AWS IoT 디바이스 새도우 서비스에 동기화하려면 Greengrass 코어 디바이스의 AWS IoT 정책에 다음 AWS IoT Core 새도우 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 작업을 참조하십시오](#).

최소 AWS IoT 정책에 대한 자세한 내용은 섹션을 참조하십시오. [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#)

- 새도 관리자 구성 요소는 VPC에서 실행할 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.3.13

다음 표에는 이 구성 요소의 버전 2.3.13에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.17.0	소프트

2.3.12

다음 표에는 이 구성 요소의 버전 2.3.12에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.17.0	소프트

2.3.11

다음 표에는 이 구성 요소의 버전 2.3.11에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.16.0	소프트

2.3.10

다음 표에는 이 구성 요소의 버전 2.3.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.15.0	소프트

2.3.9

다음 표에는 이 구성 요소의 버전 2.3.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.14.0	소프트

2.3.5 – 2.3.8

다음 표에는 이 구성 요소의 버전 2.3.5~2.3.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.5.0 <2.12.0	소프트

2.3.2

다음 표에는 이 구성 요소의 버전 2.3.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <3.0.0	소프트

2.2.2

다음 표에는 이 구성 요소의 버전 2.2.2에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.9.0	소프트

2.2.1

다음 표에는 이 구성 요소의 버전 2.2.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.4.0	소프트

2.0.0

다음 표에는 이 구성 요소의 버전 2.0.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.2.0 <2.3.0	소프트

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.3.x

strategy

(선택 사항)이 구성 요소가 AWS IoT Core 와 코어 디바이스 간에 새도우를 동기화하는 데 사용하는 전략입니다.

이 객체에는 다음 정보가 포함됩니다.

type

(선택 사항) 이 구성 요소가 AWS IoT Core 와 코어 디바이스 간에 새도를 동기화하는 데 사용하는 전략의 유형. 다음 옵션 중 하나를 선택합니다.

- `realTime` - 새도우 업데이트가 발생할 AWS IoT Core 때마다 새도우를 동기화합니다.
- `periodic-delay` 구성 파라미터로 지정한 AWS IoT Core 정기적인 간격으로 새도우를 동기화합니다.

기본값: `realTime`

delay

(선택 사항) `periodic` 동기화 전략을 지정하는 경우 이 구성 요소가 새도를 AWS IoT Core 와 동기화하는 간격(초).

Note

이 파라미터는 `periodic` 동기화 전략을 지정하는 경우에 필요합니다.

synchronize

(선택 사항) 새도가 AWS 클라우드와 동기화되는 방법을 결정하는 동기화 설정.

Note

새도를 AWS 클라우드와 동기화하려면 이 속성으로 구성 업데이트를 생성해야 합니다.

이 객체에는 다음 정보가 포함됩니다.

`coreThing`

(선택 사항) 동기화할 코어 디바이스 새도. 이 객체에는 다음 정보가 포함됩니다.

`classic`

(선택 사항) 기본적으로 새도 관리자는 코어 디바이스에 대한 클래식 새도의 로컬 상태를 AWS 클라우드와 동기화합니다. 클래식 디바이스 새도를 동기화하지 않으려면 이를 `false`로 설정합니다.

기본값: `true`

`namedShadows`

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

Warning

AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment`명명된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명명된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

`shadowDocumentsMap`

(선택 사항) 동기화할 추가 디바이스 새도. 이 구성 파라미터를 사용하면 새도 문서를 더 간편하게 지정할 수 있습니다. 이 파라미터를 `shadowDocuments` 객체 대신 사용하는 것이 좋습니다.

Note

`shadowDocumentsMap` 객체를 지정하는 경우 `shadowDocuments` 객체를 지정해서는 안 됩니다.

각 객체에는 다음 정보가 포함되어 있습니다.

`thingName`

이 새도 구성의 *`thingName`*에 대한 새도 구성.

classic

(선택 사항) 클래식 디바이스 새도를 thingName 디바이스와 동기화하지 않으려면 이를 false로 설정합니다.

namedShadows

동기화하려는 이름이 지정된 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

shadowDocuments

(선택 사항) 동기화할 추가 디바이스 새도의 목록. shadowDocumentsMap 파라미터를 대신 사용하는 것이 좋습니다.

Note

shadowDocuments 객체를 지정하는 경우 shadowDocumentsMap 객체를 지정해서는 안 됩니다.

이 목록의 각 객체에는 다음 정보가 포함되어 있습니다.

thingName

새도를 동기화할 디바이스의 사물 이름.

classic

(선택 사항) 클래식 디바이스 새도를 thingName 디바이스와 동기화하지 않으려면 이를 false로 설정합니다.

기본값: true

namedShadows

(선택 사항) 동기화하려는 이름이 지정된 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

direction

(선택 사항) 로컬 새도 서비스와 AWS 클라우드간에 새도를 동기화하는 방향. AWS 클라우드에 대한 대역폭 및 연결을 줄이도록 이 옵션을 구성할 수 있습니다. 다음 옵션 중 하나를 선택합니다.

- betweenDeviceAndCloud - 로컬 새도 서비스와 AWS 클라우드간에 새도를 동기화합니다.

- `deviceToCloud` - 로컬 새도우 서비스에서 로 새도우 업데이트를 보내고에서 새도우 업데이트를 AWS 클라우드무시합니다 AWS 클라우드.
- `cloudToDevice` - AWS 클라우드에서 새도우 업데이트를 수신하고 새도우 업데이트를 로컬 새도우 서비스에서 AWS 클라우드로 전송하지 않습니다.

기본값: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(선택 사항) 새도우 서비스 요청의 속도 제한을 결정하는 설정.

이 객체에는 다음 정보가 포함됩니다.

`maxOutboundSyncUpdatesPerSecond`

(선택 사항) 디바이스가 전송하는 초당 최대 동기화 요청 수.

기본값: 초당 100개 요청

`maxTotalLocalRequestsRate`

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 초당 200개 요청

`maxLocalRequestsPerSecondPerThing`

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 각 사물에 대해 초당 20개 요청

Note

이러한 속도 제한 파라미터는 로컬 새도우 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT 디바이스 새도우 서비스에 대한 초당 최대 요청 수에는 따라 다릅니다 AWS 리전. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

`shadowDocumentSizeLimitBytes`

(선택 사항) 로컬 새도우에 대한 각 JSON 상태 문서의 최대 허용 크기.

이 값을 늘리면 클라우드 새도에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

기본값: 8,192바이트

최대: 30,720바이트

Example예: 구성 병합 업데이트

다음 예제에서는 새도 관리자 구성 요소에 사용 가능한 모든 구성 파라미터가 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 AWS IoT Core 와 코어 디바이스 간에 새도우를 동기화하는 데 사용하는 전략입니다.

이 객체에는 다음 정보가 포함됩니다.

type

(선택 사항) 이 구성 요소가 AWS IoT Core 와 코어 디바이스 간에 새도를 동기화하는 데 사용하는 전략의 유형. 다음 옵션 중 하나를 선택합니다.

- `realTime` - 새도우 업데이트가 발생할 AWS IoT Core 때마다 새도우를 동기화합니다.
- `periodic - delay` 구성 파라미터로 지정한 AWS IoT Core 정기적인 간격으로 새도우를 동기화합니다.

기본값: `realTime`

delay

(선택 사항) `periodic` 동기화 전략을 지정하는 경우 이 구성 요소가 새도를 AWS IoT Core 와 동기화하는 간격(초).

Note

이 파라미터는 `periodic` 동기화 전략을 지정하는 경우에 필요합니다.

synchronize

(선택 사항) 새도가 AWS 클라우드와 동기화되는 방법을 결정하는 동기화 설정.

Note

새도를 AWS 클라우드와 동기화하려면 이 속성으로 구성 업데이트를 생성해야 합니다.

이 객체에는 다음 정보가 포함됩니다.

coreThing

(선택 사항) 동기화할 코어 디바이스 새도. 이 객체에는 다음 정보가 포함됩니다.

classic

(선택 사항) 기본적으로 새도 관리자는 코어 디바이스에 대한 클래식 새도의 로컬 상태를 AWS 클라우드와 동기화합니다. 클래식 디바이스 새도를 동기화하지 않으려면 이를 `false`로 설정합니다.

기본값: `true`

namedShadows

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

Warning

AWS IoT Greengrass 서비스는 `AWSThingsV2Deployment` 명령된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명령된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

shadowDocumentsMap

(선택 사항) 동기화할 추가 디바이스 새도. 이 구성 파라미터를 사용하면 새도 문서를 더 간편하게 지정할 수 있습니다. 이 파라미터를 `shadowDocuments` 객체 대신 사용하는 것이 좋습니다.

Note

`shadowDocumentsMap` 객체를 지정하는 경우 `shadowDocuments` 객체를 지정해서는 안 됩니다.

각 객체에는 다음 정보가 포함되어 있습니다.

thingName

이 새도 구성의 *thingName*에 대한 새도 구성.

classic

(선택 사항) 클래식 디바이스 새도를 `thingName` 디바이스와 동기화하지 않으려면 이를 `false`로 설정합니다.

namedShadows

동기화하려는 이름이 지정된 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

shadowDocuments

(선택 사항) 동기화할 추가 디바이스 새도의 목록. shadowDocumentsMap 파라미터를 대신 사용하는 것이 좋습니다.

Note

shadowDocuments 객체를 지정하는 경우 shadowDocumentsMap 객체를 지정해서는 안 됩니다.

이 목록의 각 객체에는 다음 정보가 포함되어 있습니다.

thingName

새도를 동기화할 디바이스의 사물 이름.

classic

(선택 사항) 클래식 디바이스 새도를 thingName 디바이스와 동기화하지 않으려면 이를 false로 설정합니다.

기본값: true

namedShadows

(선택 사항) 동기화하려는 이름이 지정된 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

direction

(선택 사항) 로컬 새도 서비스와 AWS 클라우드간에 새도를 동기화하는 방향. AWS 클라우드에 대한 대역폭 및 연결을 줄이도록 이 옵션을 구성할 수 있습니다. 다음 옵션 중 하나를 선택합니다.

- **betweenDeviceAndCloud** - 로컬 새도 서비스와 AWS 클라우드간에 새도를 동기화합니다.
- **deviceToCloud** - 로컬 새도우 서비스에서 로 새도우 업데이트를 보내고에서 새도우 업데이트를 AWS 클라우드무시합니다 AWS 클라우드.

- `cloudToDevice` - AWS 클라우드에서 새도 업데이트를 수신하고 새도 업데이트를 로컬 새도 서비스에서 AWS 클라우드로 전송하지 않습니다.

기본값: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(선택 사항) 새도 서비스 요청의 속도 제한을 결정하는 설정.

이 객체에는 다음 정보가 포함됩니다.

`maxOutboundSyncUpdatesPerSecond`

(선택 사항) 디바이스가 전송하는 초당 최대 동기화 요청 수.

기본값: 초당 100개 요청

`maxTotalLocalRequestsRate`

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 초당 200개 요청

`maxLocalRequestsPerSecondPerThing`

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 각 사물에 대해 초당 20개 요청

Note

이러한 속도 제한 파라미터는 로컬 새도 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT 디바이스 새도우 서비스에 대한 초당 최대 요청 수에는 따라 다릅니다. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

`shadowDocumentSizeLimitBytes`

(선택 사항) 로컬 새도에 대한 각 JSON 상태 문서의 최대 허용 크기.

이 값을 늘리면 클라우드 새도에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

기본값: 8,192바이트

최대: 30,720바이트

Example예: 구성 병합 업데이트

다음 예제에서는 새도 관리자 구성 요소에 사용 가능한 모든 구성 파라미터가 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 AWS IoT Core 와 코어 디바이스 간에 새도우를 동기화하는 데 사용하는 전략입니다.

이 객체에는 다음 정보가 포함됩니다.

type

(선택 사항) 이 구성 요소가 AWS IoT Core 와 코어 디바이스 간에 새도를 동기화하는 데 사용하는 전략의 유형. 다음 옵션 중 하나를 선택합니다.

- `realTime` - 새도우 업데이트가 발생할 AWS IoT Core 때마다 새도우를와 동기화합니다.
- `periodic - delay` 구성 파라미터로 지정한 AWS IoT Core 정기적인 간격으로 새도우를와 동기화합니다.

기본값: `realTime`

delay

(선택 사항) `periodic` 동기화 전략을 지정하는 경우 이 구성 요소가 새도를 AWS IoT Core 와 동기화하는 간격(초).

Note

이 파라미터는 `periodic` 동기화 전략을 지정하는 경우에 필요합니다.

synchronize

(선택 사항) 새도가 AWS 클라우드와 동기화되는 방법을 결정하는 동기화 설정.

Note

새도를 AWS 클라우드와 동기화하려면 이 속성으로 구성 업데이트를 생성해야 합니다.

이 객체에는 다음 정보가 포함됩니다.

coreThing

(선택 사항) 동기화할 코어 디바이스 새도. 이 객체에는 다음 정보가 포함됩니다.

classic

(선택 사항) 기본적으로 새도 관리자는 코어 디바이스에 대한 클래식 새도의 로컬 상태를 AWS 클라우드와 동기화합니다. 클래식 디바이스 새도를 동기화하지 않으려면 이를 `false`로 설정합니다.

기본값: true

namedShadows

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

Warning

AWS IoT Greengrass 서비스는 AWSManagedGreengrassV2Deployment명명된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명명된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

shadowDocumentsMap

(선택 사항) 동기화할 추가 디바이스 새도. 이 구성 파라미터를 사용하면 새도 문서를 더 간편하게 지정할 수 있습니다. 이 파라미터를 shadowDocuments 객체 대신 사용하는 것이 좋습니다.

Note

shadowDocumentsMap 객체를 지정하는 경우 shadowDocuments 객체를 지정해서는 안 됩니다.

각 객체에는 다음 정보가 포함되어 있습니다.

thingName

이 새도 구성의 *thingName*에 대한 새도 구성.

classic

(선택 사항) 클래식 디바이스 새도를 thingName 디바이스와 동기화하지 않으려면 이를 false로 설정합니다.

namedShadows

동기화하려는 이름이 지정된 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

shadowDocuments

(선택 사항) 동기화할 추가 디바이스 새도의 목록. shadowDocumentsMap 파라미터를 대신 사용하는 것이 좋습니다.

Note

shadowDocuments 객체를 지정하는 경우 shadowDocumentsMap 객체를 지정해서는 안 됩니다.

이 목록의 각 객체에는 다음 정보가 포함되어 있습니다.

thingName

새도를 동기화할 디바이스의 사물 이름.

classic

(선택 사항) 클래식 디바이스 새도를 thingName 디바이스와 동기화하지 않으려면 이를 false로 설정합니다.

기본값: true

namedShadows

(선택 사항) 동기화하려는 이름이 지정된 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

rateLimits

(선택 사항) 새도 서비스 요청의 속도 제한을 결정하는 설정.

이 객체에는 다음 정보가 포함됩니다.

maxOutboundSyncUpdatesPerSecond

(선택 사항) 디바이스가 전송하는 초당 최대 동기화 요청 수.

기본값: 초당 100개 요청

maxTotalLocalRequestsRate

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 초당 200개 요청

maxLocalRequestsPerSecondPerThing

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 각 사물에 대해 초당 20개 요청

Note

이러한 속도 제한 파라미터는 로컬 새도 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT 디바이스 새도우 서비스에 대한 초당 최대 요청 수에는 따라 다릅니다 AWS 리전. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

shadowDocumentSizeLimitBytes

(선택 사항) 로컬 새도에 대한 각 JSON 상태 문서의 최대 허용 크기.

이 값을 늘리면 클라우드 새도에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

기본값: 8,192바이트

최대: 30,720바이트

Example예: 구성 병합 업데이트

다음 예제에서는 새도 관리자 구성 요소에 사용 가능한 모든 구성 파라미터가 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) 새도가 AWS 클라우드와 동기화되는 방법을 결정하는 동기화 설정.

Note

새도를 AWS 클라우드와 동기화하려면 이 속성으로 구성 업데이트를 생성해야 합니다.

이 객체에는 다음 정보가 포함됩니다.

coreThing

(선택 사항) 동기화할 코어 디바이스 새도. 이 객체에는 다음 정보가 포함됩니다.

classic

(선택 사항) 기본적으로 새도 관리자는 코어 디바이스에 대한 클래식 새도의 로컬 상태를 AWS 클라우드와 동기화합니다. 클래식 디바이스 새도를 동기화하지 않으려면 이를 `false`로 설정합니다.

기본값: `true`

namedShadows

(선택 사항) 동기화할 이름이 지정된 코어 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

Warning

AWS IoT Greengrass 서비스는 `AWSThingsV2Deployment` 명령된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명령된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

shadowDocumentsMap

(선택 사항) 동기화할 추가 디바이스 새도. 이 구성 파라미터를 사용하면 새도 문서를 더 간편하게 지정할 수 있습니다. 이 파라미터를 `shadowDocuments` 객체 대신 사용하는 것이 좋습니다.

Note

`shadowDocumentsMap` 객체를 지정하는 경우 `shadowDocuments` 객체를 지정해서는 안 됩니다.

각 객체에는 다음 정보가 포함되어 있습니다.

thingName

이 새도 구성의 *thingName*에 대한 새도 구성.

`classic`

(선택 사항) 클래식 디바이스 새도를 `thingName` 디바이스와 동기화하지 않으려면 이를 `false`로 설정합니다.

`namedShadows`

동기화하려는 이름이 지정된 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

`shadowDocuments`

(선택 사항) 동기화할 추가 디바이스 새도의 목록. `shadowDocumentsMap` 파라미터를 대신 사용하는 것이 좋습니다.

Note

shadowDocuments 객체를 지정하는 경우 shadowDocumentsMap 객체를 지정해서는 안 됩니다.

이 목록의 각 객체에는 다음 정보가 포함되어 있습니다.

thingName

새도를 동기화할 디바이스의 사물 이름.

classic

(선택 사항) 클래식 디바이스 새도를 thingName 디바이스와 동기화하지 않으려면 이를 false로 설정합니다.

기본값: true

namedShadows

(선택 사항) 동기화하려는 이름이 지정된 디바이스 새도의 목록. 새도의 정확한 이름을 지정해야 합니다.

rateLimits

(선택 사항) 새도 서비스 요청의 속도 제한을 결정하는 설정.

이 객체에는 다음 정보가 포함됩니다.

maxOutboundSyncUpdatesPerSecond

(선택 사항) 디바이스가 전송하는 초당 최대 동기화 요청 수.

기본값: 초당 100개 요청

maxTotalLocalRequestsRate

(선택 사항) 코어 디바이스로 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 초당 200개 요청

maxLocalRequestsPerSecondPerThing

(선택 사항) 연결된 각 IoT 사물에 대해 전송되는 초당 최대 로컬 IPC 요청 수.

기본값: 각 사물에 대해 초당 20개 요청

Note

이러한 속도 제한 파라미터는 로컬 새도 서비스에 대한 초당 최대 요청 수를 정의합니다. AWS IoT 디바이스 새도우 서비스에 대한 초당 최대 요청 수에는 따라 다릅니다 AWS 리전. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

shadowDocumentSizeLimitBytes

(선택 사항) 로컬 새도에 대한 각 JSON 상태 문서의 최대 허용 크기.

이 값을 늘리면 클라우드 새도에 대한 JSON 상태 문서의 리소스 제한도 늘려야 합니다. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS IoT Device Shadow Service API](#)에 대한 제한을 참조하세요.

기본값: 8,192바이트

최대: 30,720바이트

Example예: 구성 병합 업데이트

다음 예제에서는 새도 관리자 구성 요소에 사용 가능한 모든 구성 파라미터가 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 nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.3.13	버그 수정 및 개선 사항 - 동기화 방향 구성에서 허용하는 경우에도 로컬 새도우 업데이트가 클라우드에 간헐적으로 동기화되지 않는 문제를 해결합니다. - 사소한 버그 수정과 개선 사항입니다.
2.3.12	버그 수정 및 개선 사항 1,024개 이상의 디바이스 새도우가 구성된 경우 배포가 차단되는 문제를 해결합니다.
2.3.11	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.3.10	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.3.9	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.3.8	버그 수정 및 개선 사항 새도 관리자가 MQTT 클라이언트 연결 중에 교착 상태를 생성하는 문제를 해결합니다.
2.3.7	버그 수정 및 개선 사항 새도 관리자가 새도 관리자 동기화 중에 주기적으로 NullPointerException 오류를 기록하는 문제를 해결합니다.

버전	변경 사항
2.3.6	버그 수정 및 개선 사항 디바이스가 오프라인 상태인 동안 AWS 클라우드 업데이트를 통해 삭제되는 새도 속성이 연결이 복구된 후에도 로컬 새도에 계속 존재하는 문제를 해결합니다.
2.3.5	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.3.4	버그 수정 및 개선 사항 null 및 빈 새도 상태 문서에 대한 지원을 추가합니다.
2.3.3	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.3.2	버그 수정 및 개선 사항 - 로컬 새도 데이터베이스가 손상되었을 때 새도 관리자가 BROKEN 상태가 되는 문제를 해결합니다. - Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.3.1	버그 수정 및 개선 사항 - 클라우드 새도 업데이트가 동기화되지 않을 수 있는 조건이 수정됩니다. - 이름이 지정된 새도 동기화 구성에 대한 변경 사항이 이름이 지정된 새도 하나에만 적용되는 문제를 해결합니다.
2.3.0	버그 수정 및 개선 사항 Greengrass 디바이스 프라이빗 키가 하드웨어 보안 모듈에 저장되어 있는 경우 새도가 동기화되지 않을 수 있는 문제를 수정했습니다.
2.2.4	버그 수정 및 개선 사항 - 로컬 새도 문서를 업데이트할 때 새도 크기 검증이 클라우드와 일치하지 않는 문제를 해결합니다. - 배포가 구성 노드에서 RESET을 수행하는 경우 새도 관리자가 구성 업데이트 수신을 중지하는 문제를 해결합니다.
2.2.3	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.2.2	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.2.1	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.2.0	새로운 기능 - 로컬 게시/구독 인터페이스를 통해 로컬 새도 서비스에 대한 지원을 추가합니다. 이제 새도 MQTT 주제에 대해 로컬 게시/구독 메시지 브로커와 통신하여 코어 디바이스에서 새도를 가져오고 업데이트하고 삭제할 수 있습니다. 이 기능을 사용하면 MQTT 브리지를 사용하여 클라이언트 디바이스를 로컬 새도 서비스에 연결하고 클라이언트 디바이스와 로컬 게시/구독 인터페이스 간에 새도 주제에 대한 메시지를 릴레이할 수 있습니다. 이 기능을 사용하려면 Greengrass nucleus 구성 요소의 v2.6.0 이상이 필요합니다. 클라이언트 디바이스를 로컬 새도 서비스에 연결하려면 MQTT 브리지 구성 요소 v2.2.0 이상을 사용해야 합니다. - 로컬 새도 서비스와 AWS 클라우드간에 새도를 동기화하는 방향을 사용자 지정하도록 구성할 수 있는 <code>direction</code> 옵션을 추가합니다. AWS 클라우드에 대한 대역폭 및 연결을 줄이도록 이 옵션을 구성할 수 있습니다.
2.1.1	버그 수정 및 개선 사항 - JSON 디바이스 새도 상태 문서의 <code>desired</code> 및 <code>reported</code> 섹션의 최대 깊이가 5개 수준이 아닌 4개 수준인 문제를 수정합니다. - Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 주기적인 새도 동기화 간격에 대한 지원을 추가하여 대역폭 사용량과 요금을 줄이도록 코어 디바이스를 구성할 수 있습니다.
2.0.6	이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.
2.0.5	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	버그 수정 및 개선 사항 - 새도 관리자가 이전에 삭제된 새도의 새로 생성된 버전을 삭제하는 문제를 해결합니다. - 직접 호출 시 새도 버전을 늘리도록 <code>DeleteThingShadow</code> IPC 작업을 업데이트합니다.

버전	변경 사항
2.0.3	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.2	버그 수정 및 개선 사항 - AWS IoT Core에서 새도 상태를 동기화할 때 새도 관리자가 delta 속성을 인식하지 못하는 문제를 해결했습니다. - 때때로 새도에 대한 동기화 요청이 잘못 병합되는 문제를 해결했습니다.
2.0.1	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.0	초기 버전입니다.

Amazon SNS

Amazon SNS 구성 요소(`aws.greengrass.SNS`)는 Amazon Simple Notification Service(Amazon SNS) 주제에 메시지를 게시합니다. 이 구성 요소를 사용하여 Greengrass 코어 디바이스에서 웹 서버, 이메일 주소, 기타 메시지 구독자에게 이벤트를 전송할 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 게시합니다. 기본적으로 이 구성 요소에서는 `sns/message` [로컬 게시/구독](#) 주제를 구독합니다. 이 구성 요소를 배포할 때 AWS IoT Core MQTT 주제를 포함한 다른 주제를 지정할 수 있습니다.

사용자 지정 구성 요소에서 필터링 또는 형식 지정 논리를 구현하여 다른 소스의 메시지를 이 구성 요소에 게시하기 전에 처리하고 싶을 수 있습니다. 이를 통해 메시지 처리 논리를 단일 구성 요소로 중앙 집중화할 수 있습니다.

Note

이 구성 요소는 Amazon SNS 커넥터 in AWS IoT Greengrass V1과 유사한 기능을 제공합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [Amazon SNS 커넥터](#)를 참조하십시오.

주제

- [버전](#)

- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [입력 데이터](#)
- [출력 데이터](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

Type

이 구성 요소는 Lambda 구성 요소(`aws.greengrass.lambda`)입니다. [Greengrass nucleus](#)에서는 [Lambda 런처 구성 요소](#)를 사용하여 이 구성 요소의 Lambda 함수를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 단원을 참조하십시오.

- 코어 장치에 설치되고 PATH 환경 변수에 추가된 [Python](#) 버전 3.7입니다.
- Amazon SNS 주제. 자세한 설명은 Amazon Simple Notification Service 개발자 안내서에서 [Amazon SNS 주제 생성](#)을 참조하세요.
- 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서는 sns:Publish 작업이 허용되어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012:topic-name"
      ]
    }
  ]
}
```

이 구성 요소의 입력 메시지 페이로드에서 기본 주제를 동적으로 재정의할 수 있습니다. 애플리케이션에서 이 기능을 사용하는 경우 IAM 정책에 모든 대상 주제가 리소스로 포함되어야 합니다. 리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해).

- 이 구성 요소의 출력 데이터를 수신하려면 이 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)(aws.greengrass.LegacySubscriptionRouter)에 대한 다음 구성 업데이트를 병합해야 합니다. 이 구성에서는 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"
    }
  }
}
```

- *region* AWS 리전 을 사용하는 로 바꿉니다.
- *##*을 이 구성 요소에서 실행되는 Lambda 함수의 버전으로 바꿉니다. Lambda 함수 버전을 찾으려면 배포하려는 이 구성 요소의 버전에 대한 레시피를 확인해야 합니다. [AWS IoT Greengrass 콘솔](#)에서 이 구성 요소의 세부 정보 페이지를 열고 Lambda 함수 키-값 페어를 찾습니다. 이 키-값 페어에는 Lambda 함수의 이름과 버전이 포함되어 있습니다.

Important

이 구성 요소를 배포할 때마다 레거시 구독 라우터의 Lambda 함수 버전을 업데이트해야 합니다. 이를 통해 배포하는 구성 요소 버전에 맞는 올바른 Lambda 함수 버전을 사용할 수 있습니다.

자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

- Amazon SNS 구성 요소는 VPC에서 실행할 수 있습니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.
- Amazon SNS 구성 요소에는 VPC 엔드포인트가 `com.amazonaws.us-east-1.sns`인 `sns.region.amazonaws.com`에 대한 연결이 있어야 합니다.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
<code>sns.region.amazonaws.com</code>	443	예	Amazon SNS에 메시지 게시

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.10

다음 표에는 이 구성 요소의 버전 2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.16.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.9

다음 표에는 이 구성 요소의 버전 2.1.9에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.15.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트
토큰 교환 서비스	^2.0.0	하드

2.1.8

다음 표에는 이 구성 요소의 버전 2.1.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.14.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.13.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.12.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.11.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.10.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.1.3

다음 표에는 이 구성 요소의 버전 2.1.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.9.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.8.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.7.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.5.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.4.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	하드
Lambda 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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.2.0	하드
Lambda 런처	^2.0.0	하드
Lambda 런타임	^2.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	^2.0.0	하드

2.0.3

다음 표에는 이 구성 요소의 버전 2.0.3에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	하드
Lambda 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 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

Note

이 구성 요소의 기본 구성에는 Lambda 함수 파라미터가 포함되어 있습니다. 디바이스에서 다음 파라미터만 사용하여 이 구성 요소를 구성하는 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 ARN. 입력 메시지 페이로드의 `sns_topic_arn` 속성으로 대상 주제를 재정의할 수 있습니다.

containerMode

(선택 사항) 이 구성 요소에 대한 컨테이너화 모드. 다음 옵션 중 하나를 선택합니다.

- `NoContainer` – 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.
- `GreengrassContainer` – 구성 요소는 AWS IoT Greengrass 컨테이너 내의 격리된 런타임 환경에서 실행됩니다.

기본값: `GreengrassContainer`

containerParams

(선택 사항) 이 구성 요소에 대한 컨테이너 파라미터를 포함하는 객체입니다. `containerMode`에 `GreengrassContainer`를 지정하면 구성 요소에서 이러한 파라미터를 사용합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

memorySize

(선택 사항) 구성 요소에 할당되는 메모리 양(킬로바이트)입니다.

기본값은 512MB(525,312KB)입니다.

pubsubTopics

(선택 사항) 구성 요소에서 메시지를 수신하려고 구독하는 주제를 포함하는 객체입니다. 각 주제와 구성 요소가 AWS IoT Core 또는 로컬 게시/구독 주제의 MQTT 주제를 구독하는지 여부를 지정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

0 – 문자열 형태의 배열 인덱스입니다.

다음 정보를 포함하는 객체입니다.

type

(선택 사항) 이 구성 요소에서 메시지 구독에 사용하는 게시/구독 메시징의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- `PUB_SUB` – 로컬 게시/구독 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

- IOT_CORE - AWS IoT Core MQTT 메시지를 구독합니다. 이 옵션을 선택하면 주제에 MQTT 와일드카드를 포함할 수 없습니다. 이 옵션을 지정할 때 사용자 지정 구성 요소에서 메시지를 보내는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본값: PUB_SUB

topic

(선택 사항) 구성 요소가 메시지를 수신하기 위해 구독하는 주제입니다. type에 IotCore를 지정하면 이 주제에서 MQTT 와일드카드(+ 및 #)를 사용할 수 있습니다.

Example예: 구성 병합 업데이트(컨테이너 모드)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example예: 구성 병합 업데이트(컨테이너 없음 모드)

```
{
  "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 해당 메시지를 대상 Amazon SNS 주제에 그대로 게시합니다. 기본적으로 이 구성 요소는 로컬 게시/구독 메시지가 구독됩니다. 사용자 지정 구성 요소에서 이 구성 요소로 메시지를 게시하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

기본 주제(로컬 게시/구독): sns/message

메시지는 다음 속성을 수락합니다. 입력 메시지는 JSON 형식이어야 합니다.

request

Amazon SNS 주제로 전송할 메시지에 대한 정보.

유형: 다음 정보를 포함하는 object입니다.

message

문자열 형태의 메시지 콘텐츠.

JSON 객체를 전송하려면 이를 문자열로 직렬화하고 message_structure 속성에 json을 지정합니다.

유형: string

subject

(선택 사항) 메시지의 제목.

유형: string

제목은 ASCII 텍스트로 최대 100자까지 작성할 수 있습니다. 문자, 숫자 또는 구두점으로 시작해야 합니다. 줄 바꿈이나 제어 문자를 포함할 수 없습니다.

sns_topic_arn

(선택 사항) 이 구성 요소가 메시지를 게시하는 Amazon SNS 주제의 ARN. 기본 Amazon SNS 주제를 재정의하려면 이 속성을 지정합니다.

유형: string

message_structure

(선택 사항) 메시지의 구조. content 속성에서 문자열로 직렬화하는 JSON 메시지를 전송하려면 json을 지정합니다.

유형: string

유효한 값: json

id

요청에 대한 임의의 ID입니다. 이 속성을 사용하여 입력 요청을 출력 응답에 매핑할 수 있습니다. 이 속성을 지정하면 구성 요소는 응답 객체의 id 속성을 해당 값으로 설정합니다.

유형: string

 Note

메시지 크기는 최대 256KB까지 가능합니다.

Example입력 예: 문자열 메시지

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example입력 예: JSON 메시지

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

출력 데이터

이 구성 요소는 기본적으로 다음 MQTT 주제에 대한 출력 데이터로 응답을 게시합니다. [레거시 구독 라우터 구성 요소](#)의 구성에서 이 주제를 subject로 지정해야 합니다. 사용자 지정 구성 요소에서 이 주제에 대한 메시지를 구독하는 방법에 대한 자세한 내용은 [AWS IoT Core MQTT 메시지 게시/구독](#) 섹션을 참조하세요.

기본 주제(AWS IoT Core MQTT): sns/message/status

Example출력 예: 성공

```
{
```

```

"response": {
  "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
  "status": "success"
},
"id": "request123"
}

```

Example출력 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 실시간으로 이 구성 요소의 로그 파일을 봅니다. *를* AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
```

라이선스

이 구성 요소에는 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- [AWS SDK for Python \(Boto3\)](#)/Apache 라이선스 2.0
- [botocore](#)/Apache 라이선스 2.0
- [dateutil](#)/PSF 라이선스

- [docutils](#)/BSD 라이선스, GNU 일반 공개 라이선스(GPL), Python Software Foundation 라이선스, 퍼블릭 도메인
- [jmespath](#)/MIT 라이선스
- [s3transfer](#)/Apache 라이선스 2.0
- [urllib3](#)/MIT 라이선스

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.10	Greengrass nucleus 버전 2.15.0 릴리스용으로 업데이트된 버전입니다.
2.1.9	Greengrass nucleus 버전 2.14.0 릴리스용으로 업데이트된 버전입니다.
2.1.8	Greengrass nucleus 버전 2.13.0 릴리스용으로 업데이트된 버전입니다.
2.1.7	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.6	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.
2.1.5	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.3	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.
2.1.2	Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.1.0	새로운 기능 • HTTPS 네트워크 프록시 구성에 대한 지원을 추가합니다. 자세한 내용은 포트 443에서 또는 네트워크 프록시를 통해 연결 및 HTTPS 프록시가 신뢰될 수 있도록 코어 디바이스 설정(를) 참조하세요.

버전	변경 사항
2.0.8	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.6	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.5	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.4	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.3	초기 버전입니다.

스트림 관리자

스트림 관리자 구성 요소(`aws.greengrass.StreamManager`)를 사용하면 Greengrass 코어 디바이스 AWS 클라우드 에서 로 전송할 데이터 스트림을 처리할 수 있습니다.

사용자 지정 구성 요소에서 스트림 관리자를 구성하고 사용하는 방법에 대한 자세한 내용은 [Greengrass 코어 디바이스에서 데이터 스트림 관리](#) 섹션을 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.2.x

- 2.1.x
- 2.0.x

Note

스트림 관리자를 사용하여 클라우드로 데이터를 내보내는 경우에는 스트림 관리자 구성 요소의 버전 2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 수 없습니다. 스트림 관리자를 처음 배포하는 경우 최신 버전의 스트림 관리자 구성 요소를 배포하는 것이 좋습니다.

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- [토큰 교환 역할](#)은 스트림 관리자와 함께 사용하는 AWS 클라우드 대상에 대한 액세스를 허용해야 합니다. 자세한 내용은 다음을 참조하세요.
 - [the section called “AWS IoT 분석 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 실행할 수 있습니다. 이 구성 요소를 VPC에 배포하려면 다음이 필요합니다.

- 스트림 관리자 구성 요소는 데이터를 게시하는 AWS 서비스에 연결되어 있어야 합니다.
 - Amazon S3: `com.amazonaws.region.s3`
 - Amazon Kinesis Data Streams: `com.amazonaws.region.kinesis-streams`
 - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
- us-east-1 리전의 Amazon S3에 데이터를 게시하는 경우 이 구성 요소는 기본적으로 S3 전역 엔드포인트를 사용하려고 시도하지만 이 엔드포인트는 Amazon S3 VPC 인터페이스 엔드포인트를 통해서만 사용할 수 없습니다. 자세한 내용은 [Amazon S3에 AWS PrivateLink 대한 제한 및 제한을 참조하세요](#). 이 문제를 해결하려면 다음 옵션 중에서 선택할 수 있습니다.
 - JVM_ARGS에서 `-Daws.s3UseUsEast1RegionalEndpoint=regional`을 설정하여 us-east-1 리전의 지역 S3 엔드포인트를 사용하도록 스트림 관리자 구성 요소를 구성합니다.
 - Amazon S3 인터페이스 VPC 엔드포인트 대신 Amazon S3 게이트웨이 VPC 엔드포인트를 생성합니다. S3 게이트웨이 엔드포인트는 S3 전역 엔드포인트에 대한 액세스를 지원합니다. 자세한 내용은 [게이트웨이 엔드포인트 생성](#)을 참조하세요.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	Port	필수	설명
iotanalytics. region .amazonaws.com	443	아니요	데이터를 게시하는 경우 필요합니다 AWS IoT 분석.
kinesis. region .amazonaws.com	443	아니요	Firehose에 데이터를 게시하는 경우 필요합니다.

엔드포인트	Port	필수	설명
data.iots itewise. <i>region</i> .amazonaws.com	443	아니요	데이터를 에 게시하 는 경우 필 요합니다 AWS IoT SiteWise.
*.s3.amazonaws.com	443	아니요	S3 버킷에 데이터를 게시하는 경우 필요 합니다. *를 데이터 를 게시하 는 각 버킷 의 이름으 로 바꿀 수 있습니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.3 – 2.2.1

다음 표에는 이 구성 요소의 버전 2.1.3~2.2.1에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <3.0.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=0.0.0	하드

2.1.11 – 2.1.12

다음 표에는 이 구성 요소의 버전 2.1.11~2.1.10에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.6.0	소프트
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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.0 <2.3.0	소프트

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=0.0.0	하드

2.0.8

다음 표에는 이 구성 요소의 버전 2.0.8에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=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 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.0.3 <2.1.0	소프트
토큰 교환 서비스	>=0.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

STREAM_MANAGER_STORE_ROOT_DIR

(선택 사항) 스트림을 저장하는 데 사용되는 로컬 디렉터리의 절대 경로. 이 값은 슬래시로 시작해야 합니다(예: /data).

기존 폴더를 지정해야 하며 [스트림 관리자 구성 요소를 실행하는 시스템 사용자](#)에게 이 폴더를 읽고 쓸 권한이 있어야 합니다. 예를 들어 다음 명령을 실행하여 스트림 관리자 루트 폴더로 지정하는

폴더(/var/greengrass/streams)를 생성하고 구성할 수 있습니다. 이러한 명령을 사용하면 기본 시스템 사용자(ggc_user)가 이 폴더를 읽고 쓸 수 있습니다.

```
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 사용하는 로컬 포트 번호.

임의의 사용 가능한 포트를 사용하도록 0을 지정할 수 있습니다.

기본값: 8088

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(선택 사항) 클라이언트가 스트림 관리자와 상호 작용하기 전에 의무적으로 인증하도록 할 수 있습니다. 스트림 관리자 SDK는 클라이언트와 스트림 관리자 간의 상호 작용을 제어합니다. 이 파라미터는 스트림 관리자 SDK를 직접 호출하여 스트림 작업을 수행할 수 있는 클라이언트를 결정합니다. 자세한 내용은 [스트림 관리자 클라이언트 인증](#)을 참조하세요.

true를 지정하는 경우 스트림 관리자 SDK는 Greengrass 구성 요소만 클라이언트로 허용합니다.

false를 지정하는 경우 스트림 관리자 SDK는 코어 디바이스의 모든 프로세스를 클라이언트로 허용합니다.

기본값: true

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(선택 사항) 스트림 관리자가 데이터를 내보내는 데 사용할 수 있는 평균 최대 대역폭(초당 킬로비트).

기본값: 제한 없음

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(선택 사항) 스트림 관리자가 데이터를 내보내는 데 사용할 수 있는 최대 활성 스레드 수.

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.

니다. 코어 장치 하드웨어의 CPU 및 메모리는 제한적인 요소입니다. 시작을 위해 이 값을 장치의 프로세서 코어 수와 동일하게 설정해 볼 수 있습니다.

하드웨어가 지원할 수 있는 크기 보다 크게 설정하지 않도록 주의하세요. 각 스트림은 하드웨어 리소스를 사용하므로 제한된 디바이스에서는 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 스트림 `sizeThresholdForMultipartUploadBytes` 속성을 사용하여 Amazon S3로 단일 또는 멀티파트 업로드로 내보낼지 여부를 결정합니다. AWS IoT Greengrass 구성 요소는 Amazon S3로 내보내는 스트림을 생성할 때 이 임계값을 설정할 수 있습니다.

기본값: 5242880(5MB) 이는 또한 최소값입니다.

`LOG_LEVEL`

(선택 사항) 구성 요소의 로깅 수준. 여기에 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

기본값: INFO

`JVM_ARGS`

(선택 사항) 시작 시 스트림 관리자에게 전달할 사용자 지정 Java 가상 머신 인수. 여러 인수는 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.

startupTimeoutSeconds

(선택 사항) 구성 요소가 시작되는 최대 시간(초)입니다. 이 제한 시간이 초과되면 구성 요소의 상태가 ERRORED으로 변경됩니다.

기본값: 120

Example예: 구성 병합 업데이트

다음 예제 구성은 기본 포트가 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.2.1	버그 수정 및 개선 사항 - 스트림 관리자가 Kinesis 데이터 스트림 대상으로 메시지를 내보내지 못하는 문제를 수정했습니다. - Kinesis 데이터 스트림을 내보내는 스트림 관리자의 성능이 추가로 향상되었습니다..
2.2.0	새로운 기능 - 시작 시간 초과용 새 구성 키를 추가합니다. 기본값은 120초입니다. - Greengrass nucleus lite용 레시피 지원을 추가했습니다.
2.1.13	버그 수정 및 개선 사항 에 대한 FIPS 엔드포인트 지원 AWS IoT SiteWise
2.1.12	버그 수정 및 개선 사항 AWS 서비스 요청에 Greengrass 자격 증명을 선호하도록 자격 증명 사용 되는 순서를 업데이트합니다.
2.1.11	Greengrass nucleus 버전 2.12.0 릴리스용으로 업데이트된 버전입니다.
2.1.10	버그 수정 및 개선 사항 HTTPS 프록시 구성이 Greengrass 인증 기관(CA) 인증서 체인을 신뢰하지 않는 문제를 해결합니다.
2.1.9	Greengrass nucleus 버전 2.11.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.8	버그 수정 및 개선 사항 스트림 관리자가 <code>InvalidRequestException</code> 로 실패하여 SiteWise 내보내기를 무기한으로 재시도하는 문제를 해결합니다.
2.1.7	버그 수정 및 개선 사항 스트림 관리자가 프록시 구성을 올바르게 읽지 못하는 문제를 해결합니다.
2.1.6	버그 수정 및 개선 사항 Jetson Nano를 포함한 특정 ARMv8 프로세서에서 시작 시 충돌을 일으키는 문제를 해결합니다.
2.1.5	Greengrass nucleus 버전 2.10.0 릴리스용으로 업데이트된 버전입니다.
2.1.4	버그 수정 및 개선 사항 - 단일 배치 내에서 동일한 타임스탬프가 있는 동일한 속성 자산에 대한 항목이 SiteWise API에서 <code>ConflictingOperationException</code> 을 반환하여 스트림 관리자가 계속 재시도하는 문제를 해결합니다. - 기본 연결 제한 시간을 3초에서 1분으로 업데이트합니다.
2.1.3	버그 수정 및 개선 사항 시스템 사용자로 실행하는 경우 Windows OS의 시작 문제를 해결합니다.
2.1.2	버그 수정 및 개선 사항 - 영어 이외의 언어를 사용하는 Windows OS 문제를 해결합니다. - Greengrass nucleus 버전 2.9.0 릴리스용으로 업데이트된 버전입니다.
2.1.1	Greengrass nucleus 버전 2.8.0 릴리스용으로 업데이트된 버전입니다.

버전	변경 사항
2.1.0	새로운 기능 - 원격 측정 지표를 Amazon EventBridge에 자동으로 전송하도록 이 구성 요소가 업데이트됩니다. 자세한 내용은 AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집 단원을 참조하십시오. 이 기능은 Greengrass nucleus 구성 요소의 v2.7.0 이상에서 사용할 수 있습니다. - Greengrass nucleus 버전 2.7.0 릴리스용으로 업데이트된 버전입니다.
2.0.15	Greengrass nucleus 버전 2.6.0 릴리스용으로 업데이트된 버전입니다.
2.0.14	이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.
2.0.13	Greengrass nucleus 버전 2.5.0 릴리스용으로 업데이트된 버전입니다.
2.0.12	버그 수정 및 개선 사항 스트림 관리자 v2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 수 없었던 문제를 해결합니다. 스트림 관리자를 사용하여 클라우드로 데이터를 내보내는 경우 이제 v2.0.12로 업그레이드할 수 있습니다.
2.0.11	Greengrass nucleus 버전 2.4.0 릴리스용으로 업데이트된 버전입니다.
2.0.10	Greengrass nucleus 버전 2.3.0 릴리스용으로 업데이트된 버전입니다.
2.0.9	Greengrass nucleus 버전 2.2.0 릴리스용으로 업데이트된 버전입니다.
2.0.8	Greengrass nucleus 버전 2.1.0 릴리스용으로 업데이트된 버전입니다.
2.0.7	초기 버전입니다.

시스템 로그 전달자

시스템 로그 전달자(`aws.greengrass.SystemLogForwarder`)는 활성 시스템 로그를 CloudWatch HTTPS API를 사용해 Amazon CloudWatch로 직접 업로드합니다.

⚠ Important

이 구성 요소는 런타임 중에 생성된 systemd-journald 로그만 전달합니다. systemd-journald 로그에 대한 자세한 내용은 [systemd-journald](#) 및 [journalctl](#)을 참조하세요.

ℹ Note

이 구성 요소는 CloudWatch 로그 그룹과 스트림을 생성하고 관리할 수 있는 특정 권한이 필요합니다.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [엔드포인트 및 포트](#)
- [종속성](#)
- [구성](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.1.x
- 2.0.x

Type

이 구성 요소는 일반 구성 요소(aws.greengrass.generic)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 systemd 기반 Linux 시스템에 설치해야 합니다.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

이 구성 요소는 CloudWatch에서 로그 및 스트림 그룹을 생성할 수 있는 액세스 권한과 PutLogEvents HTTP 호출을 수행할 수 있는 권한이 필요합니다. 최소한, Greengrass 디바이스의 역할 별칭에 다음 정책 권한을 추가해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogGroup"],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/systemLogs:*"
    },
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogStream", "logs:PutLogEvents"],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/systemLogs:log-stream:${credentials-iot:ThingName}"
    }
  ]
}
```

Note

자세한 내용은 시스템 로그 전달자 [GitHub](#) 페이지를 참조하세요.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
logs. region .amazonaws.com	443	아니요	CloudWatch Logs에 로그를 쓰는 경우 필요합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

2.1.x

다음 표에는 이 구성 요소의 버전 2.1.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=2.0.0	하드
Greengrass nucleus lite	>=2.3.0	소프트

2.0.x

다음 표에는 이 구성 요소의 버전 2.0.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=2.0.0	하드

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

2.0.x-2.1.x

maxUploadIntervalSec

시스템 로그 전달자가 로그를 업로드하려고 시도하는 최대 주기입니다. 로그 전달자는 메모리가 가득 찼을 때도 로그를 업로드하므로, 구성된 최대 주기보다 더 자주 업로드될 수 있습니다.

maxRetriesCount

시스템 로그 전달자가 일시적인 HTTP 오류 발생 시 재시도하는 횟수입니다.

bufferCapacity

메모리 내 로그 저장용 링 버퍼의 크기입니다.

logGroup

CloudWatch에서의 로그 경로입니다.

logStream

CloudWatch logStream입니다.

filters

코어 디바이스에 대한 필터 구성 맵입니다.

services

시스템 로그 전달자가 로그 업로드 여부를 결정할 때 참조하는 서비스 이름 필터 목록입니다. 로그는 생성된 서비스가 이 목록의 필터 중 하나와 일치할 경우에만 업로드됩니다. 이 목록의 필터는 서비스 이름과 정확히 일치하는 문자열이거나, 뒤에 *가 붙어 접두사가 일치하면 되는 문자열일 수 있습니다.

기본값: [gg1.*]

⚠ Important

로그는 생성된 서비스가 이 목록의 필터 중 하나와 일치할 경우에만 업로드됩니다.

ℹ Note

* 값을 사용하면 사용 가능한 모든 서비스를 포함합니다.

Example구성 예제:

아래 예시는 Greengrass nucleus lite에 포함된 모든 서비스의 로그를 필터링하는 방법을 보여줍니다.

```
{
  "maxUploadIntervalSec": 300,
  "maxRetriesCount": 3,
  "bufferCapacity": 1048576,
  "logGroup": "greengrass/systemLogs",
  "logStream": "deviceName",
  "filters": {
    "services": ["ggl.*"]
  }
}
```

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
2.1.0	버그 수정 및 개선 사항 - Greengrass nucleus를 올바르게 지원하도록 구성 요소 레시피를 업데이트합니다. - 업로드할 로그가 없는 경우 로깅 출력이 개선되었습니다. - 일반적인 버그 수정 및 개선 사항.

버전	변경 사항
2.0.1	버그 수정 및 개선 사항 구성 요소 레시피가 aarch64(arm64) 시스템을 올바르게 지원하도록 업데이트되었습니다.
2.0.0	초기 버전입니다.

Systems Manager Agent

AWS Systems Manager 에이전트 구성 요소(`aws.greengrass.SystemsManagerAgent`)는 Systems Manager 에이전트를 설치하므로 Systems Manager를 사용하여 코어 디바이스를 관리할 수 있습니다. Systems Manager는 Amazon EC2 인스턴스 AWS, 온프레미스 서버 및 가상 머신(VMs), 엣지 디바이스를 포함하여 인프라를 보고 제어하는 데 사용할 수 있는 AWS 서비스입니다. Systems Manager를 사용하여 운영 데이터를 보고, 운영 태스크를 자동화하고, 보안 및 규정 준수를 유지 관리할 수 있습니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 Systems Manager 에이전트 [란 무엇입니까 AWS Systems Manager?](https://docs.aws.amazon.com/systems-manager/latest/userguide/prereqs-ssm-agent.html) 및 정보를 참조하세요. <https://docs.aws.amazon.com/systems-manager/latest/userguide/prereqs-ssm-agent.html>

Systems Manager 도구 및 기능을 기능이라고 합니다. Greengrass 코어 디바이스는 모든 Systems Manager 기능을 지원합니다. 이러한 기능과 Systems Manager를 사용하여 코어 디바이스를 관리하는 방법에 대한 자세한 내용은 AWS Systems Manager 사용 설명서의 [Systems Manager 기능](#)을 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [다음 사항도 참조하세요.](#)
- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

유형

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 Linux 코어 디바이스에만 설치할 수 있습니다.

요구 사항

이 구성 요소에는 다음과 같은 요구 사항이 있습니다.

- 64비트 Linux 플랫폼에서 실행되는 Greengrass 코어 디바이스: Armv8(AArch64) 또는 x86_64.
- Systems Manager가 수입할 수 있는 AWS Identity and Access Management (IAM) 서비스 역할이 있어야 합니다. 이 역할은 [AmazonSSMManagedInstanceCore](#) 관리형 정책 또는 동등한 권한을 정의하는 사용자 지정 정책을 포함해야 합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [오프지 디바이스에 대한 IAM 서비스 역할 생성](#)을 참조하세요.

이 구성 요소를 배포할 때 `SSMRegistrationRole` 구성 파라미터에 이 역할의 이름을 지정해야 합니다.

- [Greengrass 디바이스 역할](#)은 `ssm:AddTagsToResource` 및 `ssm:RegisterManagedInstance` 작업을 허용해야 합니다. 디바이스 역할은 이전 요구 사항을 충족하는 IAM 서비스 역할에 대한 `iam:PassRole` 작업도 허용해야 합니다. 다음 예제 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 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	Port	필수	설명
ec2messages. <i>region</i> .amazonaws.com	443	예	AWS 클라우드에서 Systems Manager 서비스와 통신합니다.
ssm. <i>region</i> .amazonaws.com	443	예	코어 디바이스를 Systems Manager

엔드포인트	Port	필수	설명
			관리형 노드로 등록합니다.
ssmmessages. <i>region</i> .amazonaws.com	443	예	AWS 클라우드에서 Systems Manager의 기능인 Session Manager와 통신합니다.

자세한 내용은 AWS Systems Manager 사용 설명서의 [참조: ec2messages, ssmmessages 및 기타 API 호출](#)을 참조하세요.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

다음 표에는 이 구성 요소의 버전 1.0.0~1.3.0에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=2.0.0 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

SSMRegistrationRole

Systems Manager가 수임할 수 있고 [AmazonSSMManagedInstanceCore](#) 관리형 정책 또는 동등한 권한을 정의하는 사용자 지정 정책을 포함하는 IAM 서비스 역할. 자세한 내용은 AWS Systems Manager 사용 설명서의 [엣지 디바이스에 대한 IAM 서비스 역할 생성](#)을 참조하세요.

SSMOverrideExistingRegistration

(선택 사항) 코어 디바이스가 하이브리드 활성화로 등록된 Systems Manager Agent를 이미 실행하는 경우 디바이스의 기존 Systems Manager Agent 등록을 재정의할 수 있습니다. 이 구성 요소가 제공하는 Systems Manager Agent를 사용하여 코어 디바이스를 관리형 노드로 등록하려면 이 옵션을 true로 설정합니다.

Note

이 옵션은 하이브리드 활성화로 등록된 디바이스에만 적용됩니다. 코어 디바이스가 Systems Manager Agent가 설치되어 있고 인스턴스 프로파일 역할이 구성된 Amazon EC2 인스턴스에서 실행되는 경우 Amazon EC2 인스턴스의 기존 관리형 노드 ID는 i-로 시작합니다. Systems Manager Agent 구성 요소를 설치하면 Systems Manager Agent는 ID가 i-대신 mi-로 시작하는 새 관리형 노드를 등록합니다. 그런 다음 ID가 mi-로 시작하는 관리형 노드를 사용하여 Systems Manager로 코어 디바이스를 관리할 수 있습니다.

기본값: false

SSMResourceTags

(선택 사항) 이 구성 요소가 코어 디바이스에 대해 생성하는 Systems Manager 관리형 노드에 추가할 태그. 이러한 태그를 사용하여 Systems Manager로 코어 디바이스 그룹을 관리할 수 있습니다. 예를 들어 지정한 태그가 있는 모든 디바이스에서 명령을 실행할 수 있습니다.

각 태그가 Key 및 Value를 사용하는 객체인 목록을 지정합니다. 예를 들어 SSMResourceTags에 대한 다음 값은 코어 디바이스의 관리형 노드에서 **Owner** 태그를 **richard-roe**로 설정하도록 이 구성 요소에 지시합니다.

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

관리형 노드가 이미 존재하고 `SSMOverrideExistingRegistration`이 `false`인 경우 이 구성 요소는 해당 태그를 무시합니다.

Example예: 구성 병합 업데이트

다음 예제 구성은 코어 디바이스가 Systems Manager에 등록하고 통신할 수 있도록 `SSMServiceRole`이라는 이름의 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 씁니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [Viewing Systems Manager Agent logs](#)를 참조하세요.

Systems Manager Agent 구성 요소는 셸 스크립트를 사용하여 Systems Manager Agent를 설치, 시작, 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 실시간으로 이 구성 요소의 로그 파일을 봅니다. 를 AWS IoT Greengrass 루트 폴더의 경로 */greengrass/v2*로 바꿉니다.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

다음 사항도 참조하세요.

- [AWS Systems Manager를 사용하여 Greengrass 코어 디바이스 관리](#)
- AWS Systems Manager 사용 설명서의 [AWS Systems Manager란 무엇입니까?](#)
- AWS Systems Manager 사용 설명서의 [Systems Manager Agent 정보](#)

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
1.3.0	버그 수정 및 개선 사항 에이전트 버전 3.3.1142.0을 가져오도록 이 구성 요소를 업데이트합니다.
1.2.4	버그 수정 및 개선 사항 Agent 버전 3.2.2303.0을 가져오도록 이 구성 요소를 업데이트합니다.
1.2.3	버그 수정 및 개선 사항 • Greengrass에서 스냅으로 Agent 구성 요소의 설치에 대한 재시도를 추가합니다. • Greengrass에서 온프레미스 ID만 사용하도록 Agent 구성 요소의 구성을 업데이트합니다. • 설치된 Agent 버전이 Greengrass SSM Agent 구성 요소의 버전과 일치하지 않는 경우에만 Agent를 업데이트하도록 이 구성 요소를 업데이트합니다.
1.1.0	이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.
1.0.0	초기 버전입니다.

토큰 교환 서비스

토큰 교환 서비스 구성 요소(`aws.greengrass.TokenExchangeService`)는 사용자 지정 구성 요소의 AWS 서비스와 상호 작용하는 데 사용할 수 있는 자격 증명을 제공합니다 AWS .

토큰 교환 서비스는 로컬 서버로 실행됩니다. 이 로컬 서버는 [Greengrass 코어 핵 구성](#) 요소에서 구성하는 AWS IoT 역할 별칭을 사용하여 AWS IoT 자격 증명 공급자에 연결합니다. 구성 요소는 두 가지 환경 변수 (AWS_CONTAINER_CREDENTIALS_FULL_URI 및 AWS_CONTAINER_AUTHORIZATION_TOKEN)를 제공합니다. AWS_CONTAINER_CREDENTIALS_FULL_URI는 이 로컬 서버에 대한 URI를 정의합니다. 구성 요소가 AWS SDK 클라이언트를 생성하면 클라이언트는 이 URI 환경 변수를 인식하고의 토큰 AWS_CONTAINER_AUTHORIZATION_TOKEN을 사용하여 토큰 교환 서비스에 연결하고 AWS 자격 증명을 검색합니다. 이렇게 하면 Greengrass 코어 디바이스가 AWS 서비스 작업을 직접 호출할 수 있습니다. 사용자 지정 구성 요소에서 이 구성 요소를 사용하는 방법에 대한 자세한 내용은 [AWS 서비스와 상호 작용](#) 섹션을 참조하세요.

Important

이러한 방식으로 AWS 자격 증명을 획득하는 지원이 2016년 7월 13일에 AWS SDKs에 추가되었습니다. 구성 요소는 해당 날짜 또는 그 이후에 생성된 AWS SDK 버전을 사용해야 합니다. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 [지원되는 AWS SDK 사용](#)을 참조하세요.

Note

AWS 기본 자격 증명 체인을 사용하는 구성 요소는 다른 자격 증명 소스가 있는 경우 이를 검색하고 사용할 수 있습니다. Amazon Elastic Container Service(Amazon ECS) 컨테이너 보안 인증 AWS_CONTAINER_CREDENTIALS_RELATIVE_URI 정보가 보다 우선합니다. 즉 AWS_CONTAINER_CREDENTIALS_FULL_URI, Amazon Elastic Container Service(Amazon ECS) 내에서는 토큰 교환 서비스 보안 인증 정보가 작동하지 않을 수 있습니다. 자세한 내용은 [표준화된 자격 증명 공급자](#)를 참조하세요.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)

- [Changelog](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 2.0.x

Type

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

운영 체제

이 구성 요소는 다음 운영 체제를 실행하는 코어 디바이스에 설치할 수 있습니다.

- Linux
- Windows

종속성

이 구성 요소에는 종속성이 없습니다.

구성

이 구성 요소는 구성 요소를 배포할 때 사용자 지정할 수 있는 다음 구성 파라미터를 제공합니다.

port

토큰 교환 서비스 연결에 사용할 포트입니다. 포트 구성 변경 후 토큰 교환 서비스가 재시작됩니다.

credentialRetryInSec

토큰 교환 서비스에서 자격 증명 요청 오류가 발생할 때 재시도 간격을 초 단위로 지정합니다.

clientError

클라이언트 오류(4xx HTTP 상태 코드)에 대한 초 단위의 재시도 간격입니다.

기본값: 120

유효한 값: 10 ~ 42900

serverError

서버 오류(5xx HTTP 상태 코드)에 대한 초 단위의 재시도 간격입니다.

기본값: 60

유효한 값: 10 ~ 42900

unknownError

알 수 없는 오류(연결 오류 및 4xx 및 5xx 범위를 벗어난 HTTP 상태 코드)에 대한 초 단위의 재시도 간격입니다.

기본값: 300

유효한 값: 10 ~ 42900

Example예: 구성 병합 업데이트

```
{
  "port": 2020,
  "credentialRetryInSec": {
    "clientError": 30,
    "serverError": 45,
    "unknownError": 60
  }
}
```

로컬 로그 파일

이 구성 요소는 [Greengrass nucleus](#) 구성요소와 동일한 로그 파일을 사용합니다.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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


이 구성 요소의 로그를 보려면

- 코어 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
2.0.3	초기 버전입니다.

IoT SiteWise OPC UA 수집기

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 게이트웨이가 여러 OPC UA 서버에 연결할 수 있습니다. AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서 [의 엣지 AWS IoT SiteWise 에서 사용을](#) 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)

- [종속성](#)
- [구성](#)
- [입력 데이터](#)
- [출력 데이터](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 3.1.x
- 3.0.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 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스가 다음 플랫폼 중 하나에서 실행되어야 합니다.

- OS: Ubuntu 20.04 이상

아키텍처: x86_64(AMD64) 또는 ARMv8(Aarch64)

- 운영체제: Red Hat Enterprise Linux (RHEL) 8

아키텍처: x86_64(AMD64) 또는 ARMv8(Aarch64)

- OS: Amazon Linux 2

아키텍처: x86_64(AMD64) 또는 ARMv8(Aarch64)

- OS: Debian 11

아키텍처: x86_64(AMD64) 또는 ARMv8(Aarch64)

- OS: Windows Server 2019 이상

아키텍처: x86_64(AMD64)

- Greengrass 코어 디바이스에서 OPC UA 서버에 대한 아웃바운드 네트워크 연결을 허용해야 합니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

다음 표에는 이 구성 요소의 모든 버전에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.3.0 <3.0.0	하드
스트림 관리자	>2.0.10<3.0.0	하드
보안 암호 관리자	>=2.0.8 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소에는 구성 파라미터가 없습니다.

AWS IoT SiteWise 콘솔 또는 API를 사용하여 IoT SiteWise OPC UA 수집기 구성 요소를 구성할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 [4단계: 데이터 소스 추가 - 선택 사항](#)을 참조하세요.

입력 데이터

이 구성 요소는 다음 형식의 데이터만 허용하며, 다른 모든 데이터는 무시되고 삭제됩니다. 아래 표에서는 OPC UA 데이터 유형과 이에 해당하는 SiteWise 데이터 유형을 매핑합니다.

SiteWise 데이터 유형	OPC UA 데이터 유형	설명
STRING	String	최대 길이가 1,024바이트인 문자열입니다.
	Guid	
	XmlElement	
INTEGER	SByte	범위가 -2,147,483,648 to 2,147,483,647 인 서명된 32비트 정수입니다.
	Byte	
	Int16	
	UInt16	
	Int32	

SiteWise 데이터 유형	OPC UA 데이터 유형	설명
DOUBLE	UInt32*	-10 ¹⁰⁰ to 10 ¹⁰⁰ 의 범위 및 IEEE 754의 배정밀도를 갖는 부동 소수점 숫자입니다.
	Int64*	
	UInt32*	
	Int64*	
	Float	
	Double	
BOOLEAN	Boolean	true 또는 false

* OPC UA 데이터 유형 UInt32 및 Int64의 경우 해당하는 SiteWise 데이터 유형은 SiteWise가 값을 나타낼 수 있는 INTEGER이고, 그렇지 않으면 DOUBLE입니다.

출력 데이터

이 구성 요소는 AWS IoT Greengrass 스트림 관리자에 BatchPutAssetPropertyValue 메시지를 씁니다. 자세한 내용은 AWS IoT SiteWise API 참조의 [BatchPutAssetPropertyValue](#)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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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
```

라이선스

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
3.1.0	새로운 기능 - 각 데이터 소스별로 노드 검색 프로세스를 구성할 수 있는 기능을 추가합니다. - 각 데이터 소스별 기본 데이터 변경 트리거를 구성할 수 있는 기능을 추가합니다. 버그 수정 및 개선 사항 - 보안 취약성을 수정합니다.
3.0.3	버그 수정 및 개선 사항 대용량 구성 동기화 시 파일이 잘리거나 동기화에 실패하는 문제를 수정합니다.

버전	변경 사항
	• AWS IoT Core 구독 지연으로 인해 시작 프로세스가 지연되는 문제를 수정합니다. • OPC UA 서버와 잠시 연결이 끊겼을 때 전체 검색이 트리거되는 문제를 수정합니다.
3.0.2	버그 수정 및 개선 사항 • <code>OpcUaCollector.IncomingValuesCount</code> 지표가 스냅샷 포인트를 계산에 정확히 포함하도록 수정합니다.
3.0.1	버그 수정 및 개선 사항 • 보안 취약성을 수정합니다. • OPC UA 타임스탬프 처리 문제를 수정합니다. 선호 타임스탬프 선택 기능을 추가하고, 선호 타임스탬프를 사용할 수 없을 때의 대체 로직을 개선합니다. • Classic 스트림 및 V2 게이트웨이 OPC UA 수집기 구성에서 대상이 지정되지 않은 경우 시스템 관리자로 기본 설정되지 않는 문제를 수정합니다.
3.0.0	새로운 기능 Classic 스트림 및 V2 게이트웨이(이전 명칭: 자체 호스팅 SiteWise Edge 게이트웨이)에 더해, MQTT 지원 V3 게이트웨이를 지원하도록 기능을 추가했습니다. 자세한 내용은 AWS IoT SiteWise Edge용 MQTT 지원 V3 게이트웨이를 참조하세요. • OPC UA 데이터를 AWS IoT Greengrass MQTT 5 브로커(EMQX)에 게시할 수 있습니다. 버그 수정 및 개선 사항 • 보안 취약성을 수정합니다.
2.6.0	새로운 기능 • AWS IoT SiteWise에서 수집이 활성화된 경우 Null 및 NaN 값의 수집이 지원되도록 기능이 추가되었습니다. AWS IoT SiteWise에서 Null 및 NaN 구성을 확인하거나 수정하려면 DescribeStorageConfiguration 및 PutStorageConfiguration API를 참조하세요.

버전	변경 사항
2.5.1	버그 수정 및 개선 사항 - 스냅샷 작업이 실행되는 동안 오류가 발생하면 향후 스냅샷 작업이 취소되는 버그를 수정합니다. - 데이터 소스의 OPC UA 서버 연결이 끊기면 OPC UA Collector를 재시작할 때까지 데이터 소스 구성 업데이트가 유지되지 않는 버그를 수정합니다.
2.5.0	새로운 기능 - 단순 배열 및 DateTime 값을 문자열로 변환하는 데이터 소스 옵션이 추가됩니다. - OPC UA 서버에서 데이터를 수집할 때 소스 또는 서버 타임스탬프를 선택하는 속성 그룹 옵션을 추가합니다. 버그 수정 및 개선 사항 - 소스 타임스탬프를 사용하도록 기본 폴링 구성이 재설정됩니다.
2.4.2	버그 수정 및 개선 사항 - OPC UA 서버 검색 중 노드가 여러 번 검색될 수 있는 문제가 해결됩니다. - 각 스냅샷 데이터 포인트에 대해 타임스탬프가 새로워지도록 스냅샷 기능이 수정됩니다.
2.4.1	버그 수정 및 개선 사항 - 프록시 지원과 관련된 문제가 해결됩니다. - 스레드 정리가 실패하여 데이터 차단이 발생한 문제가 해결됩니다.
2.4.0	새로운 기능 - 문제를 더 쉽게 식별하고 해결할 수 있도록 이벤트 로그가 추가됩니다. 버그 수정 및 개선 사항 - OPC UA 사양의 버전 1.05를 사용하는 OPC UA 서버에 연결할 때 인증서 오류를 일으키는 OPC UA 클라이언트 문제를 해결합니다.

버전	변경 사항
2.3.0	새로운 기능 - Linux에서 Greengrass nucleus HTTP 프록시 구성에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - 보안 문제(CVE-2019-19135)가 해결됩니다.
2.2.0	새로운 기능 - Linux ARMv8 아키텍처에 데이터 수집 팩 설치에 대한 지원을 추가합니다. - Linux ARMv8의 최소 요구 사항: - 메모리: 4GB - CPU: ARM Cortex-A72 또는 동등한 사양 버그 수정 및 개선 사항 - 노드 검색 프로세스의 지표에 대한 로깅이 개선됩니다. - 지원되지 않는 데이터 유형의 처리가 개선됩니다. - 데이터 스트림 오류의 로깅이 개선됩니다.
2.1.3	새로운 기능 - Windows Server 2019 이상에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 - 지원되지 않는 디바이스에 이 구성 요소를 배포할 때의 오류 메시지가 개선됩니다.

버전	변경 사항
2.1.1	새로운 기능 • 다음 구독 속성 구성에 대한 지원을 추가합니다. • DataChangeTrigger - 데이터 변경 알림을 시작하는 조건을 정의할 수 있습니다. • QueueSize - 모니터링되는 항목에 대한 알림이 대기열에 있는 특정 지표에 대한 OPC-UA 서버의 대기열 깊이입니다. • PublishingIntervalMilliseconds - 구독이 생성될 때 지정된 게시 주기 간격(밀리초)입니다. • SnapshotFrequencyMilliseconds - AWS IoT SiteWise Edge가 안정적인 데이터 스트림을 수집하도록 스냅샷 주파수 제한 시간 설정을 구성할 수 있습니다. • 이 버전은 BAD 품질 데이터 수집을 지원하고 다음 데이터 품질을 기반으로 데이터를 필터링합니다. • UNCERTAIN 품질 데이터 • BAD 품질 데이터 버그 수정 및 개선 사항 • 고객 지표가 개선됩니다. • 암호화가 활성화된 서버에 연결할 때 때때로 문제를 일으키는 보안 인코딩이 수정됩니다. • 속성 그룹을 업데이트하지 못하는 문제를 해결합니다.
2.0.3	버그 수정 및 개선 사항
2.0.2	엣지와 자산 우선 순위 동기화에 대한 버그 수정 및 개선 사항
2.0.1	초기 버전입니다.

다음 사항도 참조하세요.

- AWS IoT SiteWise 사용 설명서의 [란 무엇입니까 AWS IoT SiteWise?](#)

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 AWS IoT SiteWise 게이트웨이의 [IoT SiteWise OPC UA 수집기 구성](#) 요소에서 읽은 데이터 소스를 시뮬레이션합니다. 그런 다음이 샘플 데이터를 사용하여 AWS IoT SiteWise 기능을 탐색할 수 있습니다. AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서 [의 엣지 AWS IoT SiteWise 에서 사용을](#) 참조하세요.

주제

- [버전](#)
- [유형](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 1.0.x

유형

이 구성 요소는 일반 구성 요소(aws.greengrass.generic)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스가 로컬 호스트에서 포트 4840을 사용할 수 있어야 합니다. 이 구성 요소의 로컬 OPC UA 서버는 이 포트에서 실행됩니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

다음 표에는 이 구성 요소의 모든 버전에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.3.0 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소에는 구성 파라미터가 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 사항
1.0.0	초기 버전입니다. Windows Server 2016 이상에 대한 지원을 추가합니다.

다음 사항도 참조하세요.

- AWS IoT SiteWise 사용 설명서의 [란 무엇입니까 AWS IoT SiteWise?](#)

IoT SiteWise 게시자

IoT SiteWise 게시자 구성 요소(`aws.iot.SiteWiseEdgePublisher`)를 사용하면 AWS IoT SiteWise 게이트웨이가 엣지에서 로 데이터를 내보낼 수 있습니다 AWS 클라우드.

AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서 [의 엣지 AWS IoT SiteWise 에서 사용](#)을 참조하세요.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [입력 데이터](#)
- [로컬 로그 파일](#)
- [문제 해결 및 디버깅](#)
- [라이선스](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 4.1.x
- 4.0.x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.4.x
- 2.3.x

- 2.2.x
- 2.1.x
- 2.0.x

Type

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스가 다음 플랫폼 중 하나에서 실행되어야 합니다.

- OS: 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)

- OS: Amazon Linux 2

아키텍처: x86_64(AMD64) 또는 ARMv8(Aarch64)

- OS: Debian 11

아키텍처: x86_64(AMD64) 또는 ARMv8(Aarch64)

- OS: Windows Server 2019 이상

아키텍처: x86_64(AMD64)

- Greengrass 코어 디바이스를 인터넷에 연결해야 합니다.
- Greengrass 코어 디바이스에 `iotsitewise:BatchPutAssetPropertyValue` 작업을 수행할 권한이 있어야 합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용하도록 권한 부여](#)를 참조하세요.

Example 권한 정책

JSON

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

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
<code>data.iotsitewise. <i>region</i>.amazonaws.com</code>	443	예	데이터를 게시합니다 AWS IoT SiteWise.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는

의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

다음 표에는 이 구성 요소의 버전 2.0.x~2.2.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
Greengrass nucleus	>=2.3.0<3.0.0	하드
스트림 관리자	>=2.0.10<3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소에는 구성 파라미터가 없습니다.

AWS IoT SiteWise 콘솔 또는 API를 사용하여 IoT SiteWise 게시자 구성 요소를 구성할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 [3단계: 게시자 구성 - 선택 사항](#)을 참조하세요.

입력 데이터

이 구성 요소는 AWS IoT Greengrass 스트림 관리자에서 PutAssetPropertyValueEntry 메시지를 읽습니다. 자세한 내용은 AWS IoT SiteWise API 참조에서 [PutAssetPropertyValueEntry](#)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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 새로운 이벤트 로그가 포함되어 있습니다. 로그 파일은 로컬 로그 파일과 별개이며 다음 위치에 있습니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 들어 있습니다. 문제 해결 정보는 진단 및 문제 해결 방법 설명과 함께 제공되며, 추가 정보에 대한 링크가 제공되는 경우도 있습니다. 진단 정보에는 다음 사항이 포함됩니다.

- 심각도 수준
- 타임스탬프
- 추가 이벤트별 정보

Example로그 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 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
4.1.4	버그 수정 및 개선 사항 이제 IoT SiteWise 게시자는 IoT SiteWise 게시자의 메모리 및 CPU 사용량과 같은 시스템 상태 지표를 CloudWatch에 보고합니다.

버전	변경 사항
	시작 워크플로에 부분 장애가 발생한 경우 IoT SiteWise 게시자가 다시 시작 AWS IoT Greengrass 되도록 오류 보고를 로 업데이트했습니다.
4.1.3	버그 수정 및 개선 사항 - 데이터 체크포인팅 과정에서 발생하던 메모리 누수 문제가 수정됩니다. - 보안 취약성을 수정합니다.
4.1.2	버그 수정 및 개선 사항 - 대용량 구성 동기화 과정에서 발생하던 문제가 수정됩니다. - 에 대한 구독 지연은 더 이상 시작 프로세스에 영향을 주지 AWS IoT Core 않습니다. - 체크포인트 초기화 지연이 더 이상 시작 프로세스에 영향을 미치지 않습니다.
4.1.1	버그 수정 및 개선 사항 - 게시자를 다시 시작한 후 Amazon S3 대상을 사용하여 AWS IoT SiteWise 버퍼링된를 통해 데이터를 전송할 때 Classic 스트림, V2 게이트 웨이에서 CPU 및 메모리 사용량이 높은 문제를 해결합니다. - 또한 기본 구성 요소 로깅 동작을 재정의할 수 있도록 선택적 파일(/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/log4j2.xml)을 생성하는 기능이 추가됩니다.
4.1.0	새로운 기능 - 기본 제공 EMQX MQTT 브로커 사용자를 관리할 수 있는 명령줄 도구가 추가됩니다. 이 인터페이스는 사용자 계정 관리를 활성화하고, MQTT 토픽 게시 및 구독 권한에 대한 액세스 제어 정책을 구성합니다. 버그 수정 및 개선 사항 - 단일 버퍼링 수집 Parquet 파일에 포함될 수 있는 고유 별칭의 수를 결정하는 계산 방식이 수정됩니다.

버전	변경 사항
4.0.3	버그 수정 및 개선 사항 - 클라우드 한도 10,000과 일치하도록 AWS IoT SiteWise 버퍼링된 수집 파일당 고유 별칭의 최대값을 업데이트합니다. - 잘못된 서비스 상태를 잘못 보고하는 AWS IoT Greengrass 로깅 문제를 해결합니다. - Classic 스트림 및 V2 게이트웨이에서 Amazon S3 대상 구성 간 중복 스트림 이름을 방지하기 위한 검증이 추가됩니다. - 비표준 AWS 파티션에서 AWS IoT SiteWise 버퍼링된 수집과 관련된 호환성 문제를 해결합니다. - 중복 데이터 수집을 유발하던 문제가 수정됩니다.
4.0.2	버그 수정 및 개선 사항 - 배치 대기 시간이 만료된 후 데이터 업로드가 되지 않던 문제가 수정됩니다. - MQTT 지원 V3 게이트웨이 구성에서 각 대상이 고유한지 확인하는 검증이 추가됩니다.
4.0.1	버그 수정 및 개선 사항 - 최대 배치 대기 시간 구성을 비활성화 AWS IoT SiteWise 하면에 데이터가 업로드되지 않는 문제를 해결합니다. <code>IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker</code> 지표가 MQTT 브로커에 구독된 고유 토픽 수를 잘못 보고하던 문제가 수정됩니다.

버전	변경 사항
4.0.0	새로운 기능 Classic 스트림 및 V2 게이트웨이(이전 명칭: 자체 호스팅 SiteWise Edge 게이트웨이)에 더해, MQTT 지원 V3 게이트웨이를 지원하도록 기능이 추가됩니다. - AWS IoT Greengrass MQTT 5 브로커(EMQX)에서 데이터를 구독하고 수신하는 기능을 추가합니다. 자세한 내용은 AWS IoT SiteWise Edge용 MQTT 지원 V3 게이트웨이를 참조하세요. - 경로 필터를 사용하여 데이터 대상을 구성할 수 있는 옵션이 추가됩니다. 자세한 내용은 AWS IoT SiteWise 엣지 대상에 대한 경로 필터 이해를 참조하세요. 버그 수정 및 개선 사항 - PublishSuccessCount 지표가 음수 값을 보고하던 문제가 수정됩니다. - 게시자가 100초 내에 시작되지 않고 BROKEN 상태로 진입하던 문제가 수정됩니다. - 보안 취약성을 수정합니다.
3.2.0	새로운 기능 - AWS IoT SiteWise에서 수집이 활성화된 경우 Null 및 NaN 값의 수집이 지원되도록 기능이 추가되었습니다. AWS IoT SiteWise에서 Null 및 NaN 구성을 확인하거나 수정하려면 DescribeStorageConfiguration 및 PutStorageConfiguration API를 참조하세요. 버그 수정 및 개선 사항 - 체크포인트 데이터베이스 파일이 손상되는 문제가 수정됩니다. - 중복된 지표가 생성되는 문제가 수정됩니다.

버전	변경 사항
3.1.4	 Warning 버전 3.1.4는 2025년 2월 20일에 중단되었습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 사항 오프라인 후 예상보다 긴 시작 시간을 유발할 수 있는 문제가 수정됩니다.
3.1.3	버그 수정 및 개선 사항 <code>/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/IoTSiteWisePublisherEvents.log</code>에 있는 이벤트 로그 파일이 생성되었지만 이벤트가 기록되지 않던 문제가 수정됩니다. - MQTT 브로커와의 연결을 모니터링하기 위해 다음 CloudWatch 지표가 추가됩니다. <code>IoTSiteWisePublisher.IsConnectedToMqttBroker</code> <code>IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker</code> <code>IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived</code> <code>IoTSiteWisePublisher.MqttMessageReceivedSuccessCount</code> <code>IoTSiteWisePublisher.MqttReceivedSuccessBytes</code> 이러한 지표에 대한 자세한 내용은 AWS IoT Greengrass Version 2 게이트웨이 지표를 참조하세요. - parquet 파일을 S3에 업로드하는 데 실패할 때에도 BatchCreateJob API가 호출되던 문제가 수정됩니다.
3.1.2	버그 수정 및 개선 사항 버전 3.1.1에서 발생한 높은 CPU 사용량 문제를 수정합니다.

버전	변경 사항
3.1.1	 Warning 버전 3.1.1은 2024년 3월 12일에 중단되었습니다. 이 버전의 개선 사항은 이 구성 요소의 이후 버전에서 사용할 수 있습니다. 버그 수정 및 개선 사항 - 오류가 발생할 때 영향을 받는 데이터 별칭을 식별하는 로깅이 추가됩니다. - 수집된 데이터의 수명에 대한 AWS IoT SiteWise API 제한의 로컬 적용을 추가합니다. - Amazon S3 대상이 여러 개 있는 경우 게시자가 StreamManager 스트림의 체크포인트를 혼합하는 문제가 해결됩니다. - 게시자가 StreamManager 스트림에서 읽는 방식과 관련된 성능 병목 현상이 해결됩니다.
3.1.0	새로운 기능 - Amazon S3에 데이터를 parquet 파일로 게시할 수 있는 지원이 추가됩니다. - AWS IoT SiteWise 버퍼링된 수집에 대한 지원을 추가합니다.
3.0.0	버그 수정 및 개선 사항 - 프록시 지원과 관련된 문제가 해결됩니다. 새로운 기능 - MQTT 브로커에서 데이터 수집에 대한 지원이 활성화됩니다.
2.4.1	버그 수정 및 개선 사항 구성 요소가 Java Corretto 11 버전 11.0.20.8.1 이상과 연동되도록 합니다. 구성 요소 버전 2.4.0 및 2.3.3에서는 Java Corretto 버전 11.0.20.8.1과 함께 사용할 경우 "Could not find or load main class" 오류 메시지가 표시됩니다.

버전	변경 사항
2.4.0	새로운 기능 - 문제를 더 쉽게 식별하고 해결할 수 있도록 새 이벤트 로그가 추가됩니다. 버그 수정 및 개선 사항 - 게시자 체크포인트 복구가 개선됩니다.
2.3.3	버그 수정 및 개선 사항 높은 처리량을 지원할 수 있는 기능이 개선됩니다.
2.3.2	버그 수정 및 개선 사항 게시자 구성을 다운로드할 때 HTTP 프록시 지원이 수정됩니다.
2.3.1	새로운 기능 - Linux ARMv8 아키텍처에 데이터 수집 팩 설치에 대한 지원을 추가합니다. - Linux ARMv8의 최소 요구 사항: - 메모리: 4GB - CPU: ARM Cortex-A72 또는 동등한 사양
2.2.3	버그 수정 및 개선 사항 재시도 가능 예외 목록에 없는 일반 예외에 대한 재시도가 제거됩니다.
2.2.2	버그 수정 및 개선 사항 HTTP 프록시 서버를 AWS IoT SiteWise 통해 데이터 업로드 지원을 다시 소개합니다.
2.2.1	 Note 이 버전은 HTTP 프록시 구성을 지원하지 않습니다. 버전 2.2.2 이상에서 이 기능에 대한 지원이 다시 도입됩니다. 새로운 기능 AWS IoT SiteWise에 데이터를 업로드할 때 압축을 전환하도록 이 구성 요소에 대한 지원이 추가됩니다.

버전	변경 사항
2.2.0	 Note 이 버전은 HTTP 프록시 구성을 지원하지 않습니다. 버전 2.2.2 이상에서 이 기능에 대한 지원이 다시 도입됩니다. 새로운 기능 • 데이터를 AWS IoT SiteWise 서비스에 전송하기 전에 압축하도록 이 구성 요소가 업데이트됩니다. • 대부분의 경우 이 변경으로 인해 대역폭 사용량이 이 구성 요소의 이전 버전에 비해 75% 줄어듭니다. • 대부분의 경우 이 변경으로 인해 CPU 사용량이 최대 5% 증가합니다. 대량의 데이터를 처리하는 게이트웨이에서 이 변경으로 인해 CPU 사용량이 최대 15% 증가할 수 있습니다. • 이 변경 사항은 AWS IoT SiteWise 서비스 요금 또는 서비스 할당량 사용량에 영향을 주지 않습니다. • Windows Server 2019 이상에 대한 지원을 추가합니다. 버그 수정 및 개선 사항 • 체크포인트 파일이 손상되었을 때 이 구성 요소가 시작되지 않는 문제를 해결합니다.
2.1.4	버그 수정 및 개선 사항 • Java 버전 8과의 호환성이 수정됩니다.

버전	변경 사항
2.1.3	 Warning 이 버전은 미국 동부(오하이오), 캐나다(중부) 및 AWS GovCloud(미국 동부) 리전을 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 사항 - 지원되지 않는 디바이스에 이 구성 요소를 배포할 때의 오류 메시지가 개선됩니다. - 데이터 업로드가 실패하면 로그 오류가 업데이트됩니다.
2.1.2	버그 수정 및 개선 사항 데이터가 만료되는 즉시 만료된 데이터 내보내기 기능을 호출하도록 업데이트됩니다.
2.1.1	버그 수정 및 개선 사항
2.1.0	새로운 기능 - 최신 데이터를 클라우드에 먼저 게시하도록 지원을 추가합니다. - 만료된 데이터를 클라우드에 게시하지 않도록 지원을 추가합니다. - 만료된 데이터를 로컬에 저장하기 위한 지원을 추가합니다. 버그 수정 및 개선 사항 - 디스크 I/O 및 해당 지연 시간이 감소됩니다.
2.0.2	버그 수정 및 개선 사항
2.0.1	초기 버전입니다.

다음 사항도 참조하세요.

- AWS IoT SiteWise 사용 설명서의 [란 무엇입니까 AWS IoT SiteWise?](#)

IoT SiteWise 프로세서

IoT SiteWise 프로세서 구성 요소(`aws.iot.SiteWiseEdgeProcessor`)를 사용하면 AWS IoT SiteWise Classic 스트림, V2 게이트웨이가 엣지에서 데이터를 처리할 수 있습니다.

이 구성 요소를 사용하면 AWS IoT SiteWise 게이트웨이가 자산 모델과 자산을 사용하여 게이트웨이 디바이스의 데이터를 처리할 수 있습니다. AWS IoT SiteWise 게이트웨이에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서 [의 엣지 AWS IoT SiteWise 에서 사용을](#) 참조하세요.

Note

데이터 처리 팩(DPP) 기능은 2025년 11월 7일부터 신규 고객에게 더 이상 공개되지 않습니다. DPP를 사용하려면 해당 날짜 이전에 가입하세요. 기존 고객은 정상적으로 서비스를 계속 이용할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 [데이터 처리 팩 가용성 변경](#)을 참조하세요.

주제

- [버전](#)
- [Type](#)
- [운영 체제](#)
- [요구 사항](#)
- [종속성](#)
- [구성](#)
- [로컬 로그 파일](#)
- [라이선스](#)
- [Changelog](#)
- [다음 사항도 참조하세요.](#)

버전

이 구성 요소에는 다음과 같은 버전이 있습니다.

- 3.5.x
- 3.4.x
- 3.3.x

- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

Type

이 구성 요소는 일반 구성 요소(`aws.greengrass.generic`)입니다. [Greengrass nucleus](#)는 구성 요소의 수명 주기 스크립트를 실행합니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

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 디바이스가 다음 플랫폼 중 하나에서 실행되어야 합니다.
 - OS: Ubuntu 20.04 이상
아키텍처: x86_64(AMD64)
 - 운영체제: Red Hat Enterprise Linux (RHEL) 8
아키텍처: x86_64(AMD64)
 - OS: Amazon Linux 2
아키텍처: x86_64(AMD64)
 - OS: Windows Server 2019 이상
아키텍처: x86_64(AMD64)

- OS: Debian 11(Bullseye) 이상

아키텍처: x86_64(AMD64)

- Greengrass 코어 디바이스가 포트 443에서 인바운드 트래픽을 허용해야 합니다.
- Greengrass 코어 디바이스가 포트 443과 8883에서 아웃바운드 트래픽을 허용해야 합니다.
- 다음 포트는 AWS IoT SiteWise 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8086, 8445, 9000, 9500, 11080 및 50010에서 사용하도록 예약되어 있습니다. 예약된 포트를 트래픽용으로 사용하면 연결이 종료될 수 있습니다.

Note

포트 8087은 이 구성 요소의 버전 2.0.15 이상에만 필요합니다.

- [Greengrass 디바이스 역할에](#)는 AWS IoT Greengrass V2 디바이스에서 AWS IoT SiteWise 게이트웨이를 사용할 수 있는 권한이 있어야 합니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 [요구 사항](#)을 참조하세요.

엔드포인트 및 포트

이 구성 요소는 기본 작업에 필요한 엔드포인트 및 포트 외에 다음 엔드포인트 및 포트에 대한 아웃바운드 요청도 수행할 수 있어야 합니다. 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 단원을 참조하십시오.

엔드포인트	포트	필수	설명
<code>model.iotsitewise. <i>region</i>.amazonaws.com</code>	443	예	AWS IoT SiteWise 자산 및 자산 모델에 대한 정보를 가져옵니다.
<code>edge.iotsitewise. <i>region</i>.amazonaws.com</code>	443	예	코어 디바이스의 AWS IoT SiteWise

엔드포인트	포트	필수	설명
			게이트웨이 구성에 대한 정보를 가져옵니다.
<code>ecr.<i>region</i>.amazonaws.com</code>	443	예	Amazon AWS IoT SiteWise Elastic Container Registry에서 Edge 게이트웨이 Docker 이미지를 다운로드합니다.
<code>iot.<i>region</i>.amazonaws.com</code>	443	예	AWS 계정에 대한 디바이스 엔드포인트를 가져옵니다.
<code>sts.<i>region</i>.amazonaws.com</code>	443	예	의 ID를 가져옵니다 AWS 계정.

엔드포인트	포트	필수	설명
monitor.iotsitewis e. <i>region</i> .amazonaws.com	443	아니요	코어 디 바이스의 AWS IoT SiteWise Monitor 포 털에 액세스 하는 경 우 필요합 니다.

종속성

구성 요소를 배포할 때는 호환되는 버전의 종속성 AWS IoT Greengrass 도 배포합니다. 즉, 구성 요소를 성공적으로 배포하려면 구성 요소 및 모든 해당 종속성에 대한 요구 사항을 충족해야 합니다. 이 섹션에는 이 구성 요소의 [릴리스된 버전](#)에 대한 종속성과 각 종속성에 대한 구성 요소 버전을 정의하는 의미 체계 버전 제약 조건이 나열되어 있습니다. [AWS IoT Greengrass 콘솔](#)에서도 각 구성 요소 버전에 대한 종속성을 볼 수 있습니다. 구성 요소 세부 정보 페이지에서 종속성 목록을 찾습니다.

다음 표에는 이 구성 요소의 버전 2.0.x~2.1.x에 대한 종속성이 나열되어 있습니다.

종속성	호환 버전	종속성 유형
토큰 교환 서비스	>=2.0.3 <3.0.0	하드
스트림 관리자	>=2.0.10 <3.0.0	하드
Greengrass CLI	>=2.3.0 <3.0.0	하드

구성 요소 종속성에 대한 자세한 내용은 [구성 요소 레시피 참조](#)를 참조하세요.

구성

이 구성 요소에는 구성 파라미터가 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 디바이스에서 다음 명령을 실행하여 실시간으로 이 구성 요소의 로그 파일을 봅니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 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 다음 타사 소프트웨어/라이선스가 포함되어 있습니다.

- Apache-2.0
- MIT
- BSD-2-Clause
- BSD-3-Clause
- CDDL-1.0

- CDDL-1.1
- ISC
- ZLIB
- GPL-3.0-with-GCC-exception
- 퍼블릭 도메인
- Python-2.0
- Unicode-DFS-2015
- BSD-1-Clause
- OpenSSL
- EPL-1.0
- EPL-2.0
- GPL-2.0-with-classpath-exception
- MPL-2.0
- CC0-1.0
- JSON

이 구성 요소는 [Greengrass 코어 소프트웨어 라이선스 계약](#)에 따라 릴리스됩니다.

Changelog

다음 표에는 각 구성 요소 버전의 변경 사항이 설명되어 있습니다.

버전	변경 사항
3.5.1	새로운 기능 AWS IoT SiteWise에서 수집이 활성화된 경우 Null 및 NaN 값의 수집이 지원되도록 기능이 추가되었습니다. 에서 Null 및 NaN 구성을 보거나 수정하려면 DescribeStorageConfiguration 및 PutStorageConfiguration API를 AWS IoT SiteWise참조하세요. APIs 버그 수정 및 개선 사항 잠재적인 보안 취약성을 해결하기 위해 종속 항목을 업데이트했습니다.

버전	변경 사항
3.4.0	새로운 기능 • 사용자 지정 인증서가 필요한 프록시 서버를 통해 게이트웨이 통신을 활성화하는 선택적 HTTP 및 HTTPS 프록시 지원이 추가되었습니다. AWS IoT Greengrass 코어에 대한 HTTP 프록시 설정에 대한 자세한 내용은 섹션을 참조하세요 포트 443에서 또는 네트워크 프록시를 통해 연결. SiteWise Edge와 관련된 프록시 지원에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서의 Edge 프록시 지원을 위한 AWS IoT SiteWise 트러스트 스토어 관리를 참조하세요. • AWS OpsHub 및 SiteWise Edge APIs의 비활성 기간을 관리하기 위해 구성 가능한 세션 제한 시간 설정이 추가되었습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 AWS IoT SiteWise Edge에 대한 세션 제한 시간 구성을 참조하세요. 성능 개선 수신 데이터가 엣지 디바이스 스토리지에 도달하는 시간이 5초에서 1초 미만으로 단축되었습니다. 데이터 업로드의 지연 시간은 AWS IoT SiteWise 변경되지 않습니다.
3.3.1	새로운 기능 • 원본 간 리소스 공유 기능을 개선하는 SiteWise Edge API에 CORS 지원 (선택 사항)이 추가되었습니다. 이 기능은 API와 웹 애플리케이션 연동의 유연성을 향상합니다.

버전	변경 사항
3.3.0	성능 개선 • 새 자산 또는 업데이트된 자산의 항목만 새로 고쳐 AWS IoT SiteWise 자산 동기화 간의 I/O 사용량을 줄이도록 캐시 새로 고침 메커니즘이 최적화되었습니다. • 많은 수의 동기화된 자산 속성이 포함된 캐시를 유지하기 위한 메모리 공간이 줄었습니다. 버그 수정 및 개선 사항 • 수집 오류가 없는 경우 개별 속성 값 수집에 대한 로그가 표시되지 않아 수집 속도가 높을 때 로그 노이즈가 줄어듭니다. • 특정 로그 항목에 사람이 읽을 수 있는 형식을 사용하여 로그 가독성이 개선되었습니다. • Java 17 이상에 대한 지원이 추가되었습니다.
3.2.1	버그 수정 및 개선 사항 • AWS IoT SiteWise API 호출이 SiteWise Edge와 동기식으로 페이지 매김되지 않는 문제를 해결합니다. • 더 이상 <code>MessageRemaining.SiteWise_Edge_Stream</code> 지표를 게시하지 않도록 문제가 해결됩니다. • MQTT 브로커와의 연결을 모니터링하기 위해 다음 CloudWatch 지표가 추가되었습니다. • <code>IoTSiteWiseProcessor.IsConnectedToMqttBroker</code> • <code>IoTSiteWiseProcessor.NumberOfSubscriptionsToMqttBroker</code> • <code>IoTSiteWiseProcessor.NumberOfUniqueMqttTopicsReceived</code> • <code>IoTSiteWiseProcessor.MqttMessageReceivedSuccessCount</code> • <code>IoTSiteWiseProcessor.MqttReceivedSuccessBytes</code> 이러한 지표에 대한 자세한 내용은 AWS IoT Greengrass Version 2 게이트웨이 지표를 참조하세요.

버전	변경 사항
3.2.0	성능 개선 - 메모리 공간이 줄고 더 적은 디스크 공간에 설치되도록 API 서비스가 최적화됩니다. - 이에 따라 초기 메모리 사용량이 2GB 줄고(이제 시작 시 7.5GB의 메모리를 사용하지만 그래도 16GB 권장) 전체 구성 요소의 다운로드 크기가 500MB 줄어듭니다(현재 1.4GB를 다운로드해야 함). 새로운 기능 - GetAssetPropertyValueAggregates API는 이제 엣지에서 15분 집계 기간을 지원합니다. - 이 구성 요소가 올바르게 실행되기 위해 더 이상 포트 8081 및 8082를 사용할 필요가 없습니다. Note 와 같은 AWS IoT SiteWise 데이터 영역 APIs의 로컬 엔드포인트 <code>get-asset-property-value</code> 가에서 <code>http://localhost:8081</code> 로 변경됩니다 <code>http://localhost:11080/data</code> . 와 같은 AWS IoT SiteWise 컨트롤 플레인 APIs의 로컬 엔드포인트 <code>list-asset-models</code> 가에서 <code>http://localhost:11080</code> 로 변경됩니다 <code>http://localhost:11080/control</code> . AWS 항상 SiteWise Edge 게이트웨이 HTTPS 엔드포인트를 사용할 것을 권장합니다. 이러한 엔드포인트는 변경되지 않았습니다. 버그 수정 및 개선 사항 - 이제 이전 동기화가 중단된 경우에서 동기화 AWS IoT SiteWise 하면 리소스가 유효한 상태로 전환됩니다. 따라서 강제 재시작 후 일부 리소스가 손상되는 문제가 해결됩니다. - 동기화 중에 리소스가 수정된 경우 엣지에서 리소스가 손상될 수 있는 드문 조건이 수정됩니다. 이제 이 조건이 감지되면 동기화가 실패하고 리소스는 다음 동기화 때 다시 시도됩니다.

버전	변경 사항
	• API에 대한 HTTP 엔드포인트를 외부에서 직접 호출할 수 있도록 했던 문제를 해결합니다. 이제 로컬 루프백 주소 외부에서 API를 직접 호출할 때에는 HTTPS만 사용할 수 있습니다. • ListAssets API는 이제 엣지에 저장된 자산에 대한 자산 계층 구조를 표시합니다. • Windows에서 데이터 처리 팩을 다시 시작, 업그레이드 또는 다운그레이드하지 못한 문제를 해결합니다. • 고객이 자격 증명을 사용하여 MQTT 브로커에 연결할 수 없는 Windows OS용 데이터 처리 팩의 버그가 수정됩니다.
3.1.3	버그 수정 및 개선 사항 • 일부 리소스가 실제로 실패했을 때 데이터 처리 팩이 동기화 성공을 잘못 보고한 문제가 해결됩니다. • 상위 자산이 같지 않은 한 여러 자산의 이름이 같을 수 있도록 허용됩니다.
3.1.1	버그 수정 및 개선 사항 • 시간대 불일치로 인해 SigV4 요청이 실패하는 문제가 해결됩니다. • 다시 시작한 후 속성에 변환 및 지표 속성이 속성에 의존할 때 해당 속성의 계산이 중지되는 문제가 해결됩니다. • 사용자 지정 스트림 관리자 포트 구성에 대한 지원이 활성화됩니다. • 엣지에 동기화된 속성이 업데이트되지 않을 수 있는 문제가 해결됩니다.
3.1.0	버그 수정 및 개선 사항 • ListAssetModels API가 다음 토큰을 생성하지 못하는 문제가 해결됩니다.
3.0.0	새로운 기능 • MQTT 브로커에서 데이터 수집에 대한 지원이 활성화됩니다.

버전	변경 사항
2.2.1	버그 수정 및 개선 사항 - 컨트롤 플레인 데이터 스토리지가 클라우드 운영 방식에 더 잘 부합하도록 하기 위해 동기화 프로세스가 조정됩니다. 이에 따라 업그레이드에 약간의 영향을 미칩니다.  Note 버전 2.2.1 이상에서 동기화된 컨트롤 플레인 데이터는 이전 버전과 호환되지 않습니다. 이전 버전으로 다운그레이드하려면 새로 설치를 완료해야 합니다. 업그레이드에는 영향이 없으며 이전 버전에서 동기화된 데이터는 버전 2.2.1에서 작동합니다. - AWS 자격 증명 체인을 추가로 수정하여 AWS IoT Greengrass V2 자격 증명의 우선 순위를 지정합니다.
2.1.37	버그 수정 및 개선 사항 - 프로세스 통신에서의 리소스 사용량을 줄이기 위해 dependency-routing-service 프로세스가 사용 중지되고 해당 기능이 property-state-service 프로세스로 이동됩니다. - get-asset-property-value-history API의 최대 결과 제한을 20,000으로 늘려에서 사용하는 제한과 일치시킵니다 AWS IoT SiteWise. - 최대 결과 제한이 지정되지 않은 경우 get-asset-property-value-history API에 대한 페이지징된 결과에 다음 토큰이 제공되지 않는 문제가 해결됩니다.
2.1.35	버그 수정 및 개선 사항 - AWS 자격 증명 체인을 수정하여 자격 증명의 우선 순위를 AWS IoT Greengrass 지정합니다. - AWS IoT 사물 그룹의 일부로 배포할 때 계정 탐지 문제를 해결합니다.
2.1.34	버그 수정 및 개선 사항 - Linux에서 다중 스레딩을 사용하도록 지표/변환 계산이 조정됩니다. Windows에서는 호환성을 위해 단일 스레드 계산을 계속 실행합니다. - 일부 계산 기간에 지표 계산이 누락되는 문제를 해결합니다.

버전	변경 사항
2.1.33	버그 수정 및 개선 사항 • Greengrass 콘솔에 오류 상태 보고 관련 문제를 해결합니다.
2.1.32	버그 수정 및 개선 사항 • 사용자 지정된 사용자 이름 및 그룹에 대한 지원을 추가합니다.
2.1.31	버그 수정 및 개선 사항 • AWS IoT SiteWise에서 모델링된 데이터의 시간 가중치 평균과 시간 가중치 표준 편차를 계산할 수 있는 지원이 추가됩니다.
2.1.29	버그 수정 및 개선 사항 • 엣지 기능에 대해 자산을 필터링할 수 있는 지원이 추가됩니다.
2.1.28	버그 수정 및 개선 사항 • 리소스 동기화를 최적화하여 많은 수의 자산이에서 엣지 AWS 클라우드 로 동기화되도록 합니다.
2.1.24	버그 수정 및 개선 사항 • 리소스를 두 번째로 동기화할 때 대시보드가 사라지는 문제를 해결합니다.
2.1.23	버그 수정 및 개선 사항 • 인터넷 연결이 느린 경우 설치 실패를 방지하기 위해 <code>aws.iot.SiteWiseEdgeProcessor</code> 설치 프로세스에 대한 제한 시간이 추가되었습니다. • 클라우드와 엣지 간의 동기화 효율성을 개선하도록 리소스 동기화가 최적화되었습니다.

버전	변경 사항
2.1.21	 Warning 2.0.x에서 2.1.x로 업그레이드하면 로컬 데이터가 손실됩니다. 새로운 기능 - Windows Server 2019 이상에 대한 지원을 추가합니다. - Linux 기반 운영 체제용 도커가 제거됩니다.
2.0.16	이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.
2.0.15	버그 수정 및 개선 사항 - 이 구성 요소가 리소스 동기화 API 작업에 사용하는 포트가 8085에서 8087로 변경됩니다. 따라서 이제 이 구성 요소를 사용하려면 포트 8087을 사용할 수 있어야 합니다. 이 구성 요소를 사용하려면 포트 8085도 계속 사용할 수 있어야 합니다. - 사용자가 API 작업을 호출하려고 할 때가 아니라 로그인 중에 권한이 없는 사용자를 거부하도록 AWS OpsHub 인증을 업데이트합니다.
2.0.14	이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.
2.0.13	버그 수정 및 개선 사항 이 구성 요소가 Amazon CloudWatch 지표에 데이터를 보고할 때 모델링되지 않은 데이터를 올바르게 표시하도록 문제가 수정됩니다.
2.0.9	버그 수정 및 개선 사항 - 코어 디바이스에서 AWS IoT SiteWise 리소스를 생성하고 업데이트하는 신뢰성을 개선합니다. - 코어 디바이스에 설치된 구성 요소, 각 구성 요소의 버전 및 각 구성 요소의 상태를 모니터링하는 데 사용할 수 있는 추가 로컬 API 작업이 추가됩니다. 코어 디바이스의 애플리케이션 AWS OpsHub 용 AWS IoT SiteWise의 설정 탭에서이 정보를 볼 수 있습니다. - 이 구성 요소가 실행하는 Docker 컨테이너에 대한 상태가 추가됩니다. <code>docker ps</code> 명령을 실행하여 컨테이너의 상태를 볼 수 있습니다.

버전	변경 사항
2.0.7	버그 수정 및 개선 사항 코어 디바이스에서 AWS IoT SiteWise Monitor 포털을 볼 수 있는 지원이 수정되었습니다.
2.0.6	버그 수정 및 개선 사항 이 구성 요소가 코어 디바이스에서 계산하는 AWS IoT SiteWise <code>statetime()</code> <code>earliest()</code> , 및 <code>latest()</code> 함수를 수정합니다.
2.0.5	버그 수정 및 개선 사항 - 이 구성 요소가 코어 디바이스에서 계산하는 변환에서 AWS IoT SiteWise <code>pretrigger()</code> 함수에 대한 지원을 추가합니다. - 이 구성 요소가 인증을 위해 LDAP(Lightweight Directory Access Protocol) 구성을 저장하는 경로가 변경됩니다.
2.0.2	초기 버전입니다.

다음 사항도 참조하세요.

- AWS IoT SiteWise 사용 설명서의 [란 무엇입니까 AWS IoT SiteWise?](#)

게시자 지원 구성 요소

게시자 지원 구성 요소는 AWS IoT Greengrass의 미리 보기 릴리스에 포함되어 있으며 변경될 수 있습니다. 이러한 구성 요소는 AWS에서 지원되지 않습니다. 각 구성 요소에 문제가 있는 경우 게시자에게 문의해야 합니다.

Greengrass 게시자가 지원하는 구성 요소는 타사 구성 요소 공급업체에서 개발하고, 제안하고, 제공합니다. 타사 구성 요소 공급업체는 AWS Partner 디바이스 카탈로그, AWS Hero 또는 커뮤니티 공급업체 중 하나입니다. 이 카탈로그의 구성 요소는 타사 구성 요소 공급업체에 직접 문의하여 구매할 수 있습니다.

Greengrass 게시자 지원 구성 요소는 다음과 같습니다.

주제

- [AIShield.Edge](#)
- [AI EdgeLabs Sensor](#)
- [Greengrass S3 Ingestor](#)

AIShield.Edge

이 구성 요소는 Bosch에서 제공하는 AIShield에서 개발하고 지원합니다. AIShield.Edge로 AI 보안을 강화하세요. 이 구성 요소는 위협 정보를 기반으로 맞춤형 방어 시스템을 엣지 디바이스에 원활하게 구축하도록 설계되어 디바이스를 AI 공격으로부터 보호합니다.

이 구성 요소는 다음 이점을 제공합니다.

- AIShield AI Security를 통한 취약성 분석에서 AWS 내 강화된 엣지 방어로 원활하게 전환
- 여러 엣지 디바이스에 간편하게 맞춤형 방어 배포
- 다양한 모델 유형과 프레임워크를 지원하는 여러 가지 AI 설정에 맞춘 광범위한 보호
- Amazon SageMaker AI 및 Greengrass 워크플로에 원활하게 통합되어 최신 상태 유지
- AWS IoT Core에 직접 전달되는 데이터를 사용하여 잠재적인 위협에 대한 즉각적인 인사이트 확보
- AWS Marketplace의 AIShield AI Security에서 제공하는 엣지에서의 방어 배포를 위한 일관성 있는 AI 보안 경로

이 구성 요소는 다음 플랫폼에서 실행되어야 합니다.

- OS: linux

이 구성 요소를 구매하려면 Bosch Software and Digital Solutions(<AIShield.Contact@bosch.com>)에 문의하세요.

AI EdgeLabs Sensor

이 구성 요소는 AI EdgeLabs에서 개발하고 지원합니다. AI EdgeLabs Sensor는 AI 기반 위협 감지 및 방지 기능이 포함된 컨테이너 기반 애플리케이션입니다. AI Sensor는 Greengrass 구성 요소로 래핑되고 다른 Greengrass 구성 요소와 함께 코어 디바이스에 독립 실행형 컨테이너로 배포됩니다.

현재 구성 요소는 네트워크 통신을 지속적으로 확인하고 Edge Host나 IoT 게이트웨이에서 실행되는 소프트웨어의 위협 패턴을 찾는 컨테이너 기반 에이전트입니다. 이 구성 요소는 eBPF, 프로세스 대역

폭의 동작 검증, 호스트 기반 구성을 사용합니다. 이 구성 요소의 주요 기능은 NDR/IPS 및 EDR 기능을 기반으로 합니다.

이 구성 요소는 다음 이점을 제공합니다.

- 네트워크 공격 및 맬웨어에 대한 AI 기반 위협 감지(EDR/NDR)
- 자동화된 AI 기반 인시던트 대응(IPS)
- 외부로의 데이터-전송을 최소화하는 호스트-로컬 위협 인텔리전스
- Docker와 Greengrass를 사용한 경량 배포

이 구성 요소는 다음 플랫폼 중 하나에서 실행되어야 합니다.

- OS: linux

이 구성 요소를 구매하려면 AI EdgeLabs(<contact@edgelabs.ai>)에 문의하세요.

Greengrass S3 Ingestor

이 구성 요소는 Nathan Glover에서 개발하고 지원합니다. Greengrass S3 Ingestor 구성 요소는 [스트림 관리자 구성 요소](#)와 함께 사용하도록 설계되었습니다. 이 구성 요소는 스트림 관리자에서 줄로 구분된 JSON 메시지 스트림을 가져와서 GZIP 파일로 일괄 처리합니다. 이 구성 요소를 사용하면 추가 처리 또는 저장을 위해 Amazon S3로 데이터를 효율적으로 수집할 수 있습니다. 이 구성 요소는 AWS 클라우드의 실시간 데이터 전송을 지원하지 않습니다.

이 구성 요소는 다음 플랫폼 중 하나에서 실행되어야 합니다.

- OS: linux
- OS: Windows

이 구성 요소를 구매하려면 Nathan Glover(<nathan@glovers.id.au>)에 문의하세요.

커뮤니티 구성 요소

Greengrass Software Catalog는 Greengrass 커뮤니티에서 개발한 Greengrass 구성 요소의 인덱스입니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다. 카탈로그는 <https://github.com/aws-greengrass/aws-greengrass-software-catalog>에서 확인할 수 있습니다.

각 구성 요소에는 탐색할 수 있는 퍼블릭 GitHub 리포지토리가 있습니다. GitHub에서 Greengrass Software Catalog를 보면 커뮤니티 구성 요소의 전체 목록을 찾아볼 수 있습니다. 예를 들어 이 카탈로그에는 다음 구성 요소가 포함되어 있습니다.

- [Amazon Kinesis Video Streams](#)

이 구성 요소는 [RTSP\(실시간 스트리밍 프로토콜\)](#)를 사용하는 로컬 카메라에서 오디오 및 비디오 스트림을 수집합니다. 이후 오디오 및 비디오 스트림을 [Amazon Kinesis Video Streams](#)에 업로드합니다.

- [Bluetooth IoT 게이트웨이](#)

이 구성 요소는 Bluetooth LE(Low Energy) 디바이스와의 통신을 활성화하여 Bluetooth LE 클라이언트 인터페이스를 생성하는 [BluePy](#) 라이브러리를 사용합니다.

- [인증서 로테이터](#)

이 구성 요소는 AWS IoT Greengrass 코어 디바이스 인증서와 프라이빗 키를 폴릿 전체에서 대규모로 교체하는 수단을 제공합니다.

- [컨테이너화된 보안 터널링](#)

이 구성 요소는 특정 호스트 운영 체제에 의존하지 않는 재사용 가능한 레시피에서 모든 종속성과 일치하는 라이브러리를 사용하여 보안 터널링을 위한 Docker 컨테이너를 제공합니다.

- [Grafana](#)

이 구성 요소를 사용하면 Greengrass 코어 디바이스에서 [Grafana](#) 서버를 호스팅할 수 있습니다. Grafana 대시보드를 사용하여 코어 디바이스의 데이터를 시각화 및 관리할 수 있습니다.

- [GStreamer for Amazon Lookout for Vision](#)

이 구성 요소는 사용자 지정 GStreamer 파이프라인에서 Lookout for Vision 이상을 감지할 수 있도록 GStreamer 플러그인을 제공합니다.

- [홈 어시스턴트](#)

이 구성 요소를 사용하면 고객이 [Home Assistant](#)를 사용하여 스마트 홈 디바이스를 로컬로 제어할 수 있습니다. 엣지 및 클라우드의 AWS 서비스와 통합하여 Home Assistant를 확장하는 홈 자동화 솔루션을 제공합니다.

- [InfluxDBGrafana 대시보드](#)

이 구성 요소는 InfluxDB 및 Grafana 구성 요소를 설정하는 원클릭 환경을 제공합니다. InfluxDB를 Grafana에 연결하고 AWS IoT Greengrass 원격 측정을 실시간으로 렌더링하는 로컬 Grafana 대시보드 설정을 자동화합니다.

- [InfluxDB](#)

이 구성 요소는 Greengrass 코어 디바이스에서 [InfluxDB](#) 시계열 데이터베이스를 제공합니다. 이 구성 요소를 사용하여 IoT 센서의 데이터를 처리하고, 데이터를 실시간으로 분석하고, 엣지에서 작업을 모니터링할 수 있습니다.

- [InfluxDB 게시자](#)

이 구성 요소는 AWS IoT Greengrass 시스템 상태 원격 측정을 [nucleus 이미지 플러그인](#)에서 InfluxDB로 릴레이합니다. 이 구성 요소는 사용자 지정 원격 측정을 InfluxDB로 전달할 수도 있습니다.

- [IoT pubsub 프레임워크](#)

이 프레임워크는 AWS IoT Greengrass v2 사용자 지정 구성 요소를 사용하는 분산 이벤트 기반 IoT pubsub 애플리케이션의 코드 품질을 개선하는 데 도움이 되는 애플리케이션 아키텍처, 템플릿 코드 및 배포 가능한 예제를 제공합니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 생성](#) 섹션을 참조하세요.

- [Jupyter Labs](#)

이 구성 요소는 JupyterLab을 AWS IoT Greengrass 코어 디바이스에 배포합니다. Jupyter 환경은 AWS IoT Greengrass에서 설정한 프로세스 및 환경 변수 리소스에 액세스할 수 있으므로 Python으로 작성된 구성 요소를 테스트 및 개발하는 프로세스가 간소화됩니다.

- [로컬 웹 서버](#)

이 구성 요소를 사용하면 Greengrass 코어 디바이스에서 로컬 웹 사용자 인터페이스를 생성할 수 있습니다. 예를 들어 로컬 웹 사용자 인터페이스를 생성하여 디바이스 및 애플리케이션 설정을 구성하거나 디바이스를 모니터링할 수 있습니다.

- [LoRaWAN 프로토콜 어댑터](#)

이 구성 요소는 LPWAN(저전력 광역 네트워크) 프로토콜인 LoRaWAN 프로토콜을 사용하는 로컬 무선 디바이스에서 데이터를 수집합니다. 클라우드와 통신하지 않고도 로컬에서 데이터를 분석하고 조치를 취할 수 있습니다.

- [Modbus TCP](#)

이 구성 요소는 ModbusTCP 프로토콜을 사용하는 로컬 디바이스에서 데이터를 수집하고 선택한 데이터 스트림에 게시합니다.

- [Node-RED](#)

이 구성 요소는 NPM을 사용하여 AWS IoT Greengrass 코어 디바이스에 Node-RED를 설치합니다. 구성 요소는 명시적으로 배포 및 구성해야 하는 [Node-RED Auth](#) 구성 요소에 의존합니다. [Node-RED CLI for Greengrass](#)를 사용하여 AWS IoT Greengrass 디바이스에 Node-RED 흐름을 배포할 수 있습니다.

- [Node-RED Docker](#)

이 구성 요소는 공식 Node-RED Docker 컨테이너를 사용하여 AWS IoT Greengrass 코어 디바이스에 Node-RED를 설치합니다. 구성 요소는 명시적으로 배포 및 구성해야 하는 [Node-RED Auth](#) 구성 요소에 의존합니다. [Node-RED CLI for Greengrass](#)를 사용하여 AWS IoT Greengrass 디바이스에 Node-RED 흐름을 배포할 수 있습니다.

- [Node-RED Auth](#)

이 구성 요소는 AWS IoT Greengrass 코어 디바이스에서 실행되는 Node-RED 인스턴스를 보호하는 사용자 이름과 암호를 구성합니다.

- [OpenThread Border Router](#)

이 구성 요소는 OpenThread Border Router Docker 컨테이너를 배포합니다. 이 구성 요소는 스레드 경계 라우터가 포함된 Matter 디바이스를 구성하는 데 도움이 됩니다.

- [OSI Pi Streaming Data Connector](#)

이 구성 요소는 OSI Pi Data Archive에서 AWS의 최신 데이터 아키텍처로의 스트리밍 실시간 데이터 수집을 제공합니다. AWS IoT PubSub 메시징을 통해 중앙에서 관리되는 OSI Pi Asset Framework와 통합됩니다.

- [Parsec Provider](#)

이 구성 요소를 사용하면 AWS IoT Greengrass 디바이스가 [CNCF\(Cloud Native Computing Foundation\)](#)의 오픈 소스 [Parsec](#) 프로젝트를 사용하여 하드웨어 보안 솔루션을 통합할 수 있습니다.

- [PostgreSQL DB](#)

이 구성 요소는 엣지에서 [PostgreSQL](#) 관계형 데이터베이스를 지원합니다. 고객은 이 구성 요소를 사용하여 Docker 컨테이너 내에서 로컬 PostgreSQL 인스턴스를 프로비저닝 및 관리할 수 있습니다.

- [S3 파일 업로더](#)

이 구성 요소는 디렉터리에서 새 파일을 모니터링하여 Amazon Simple Storage Service(Amazon S3)에 업로드하고, 업로드 성공 후에는 삭제합니다.

- [Secrets Manager 클라이언트](#)

이 구성 요소는 레시피 수명 주기 스크립트의 Secrets Manager 구성 요소에서 보안 암호를 검색하는데 필요한 다른 구성 요소에서 사용할 수 있는 CLI 도구를 제공합니다.

- [컨테이너로 TES 라우팅](#)

이 구성 요소는 컨테이너에서 [토큰 교환 서비스](#) 구성 요소를 사용할 수 있도록 AWS IoT Greengrass 디바이스에서 nftable 또는 iptable을 구성합니다.

- [WebRTC](#)

이 구성 요소는 AWS IoT Greengrass 코어 디바이스에 연결된 RTSP 카메라에서 오디오 및 비디오 스트림을 수집합니다. 이후 오디오 및 비디오 스트림을 피어 투 피어 통신으로 변환하거나 Amazon Kinesis Video Streams를 통해 릴레이합니다.

기능을 요청하거나 버그를 보고하려면 해당 구성 요소의 리포지토리에서 GitHub 문제를 엽니다. AWS는 커뮤니티 구성 요소에 대한 지원을 제공하지 않습니다. 자세한 내용은 각 구성 요소의 리포지토리에 있는 CONTRIBUTING.md 파일을 참조하세요.

여러 AWS 제공 구성 요소도 오픈 소스입니다. 자세한 내용은 [오픈 소스 AWS IoT Greengrass 코어 소프트웨어](#) 섹션을 참조하세요.

AWS IoT Greengrass 개발 도구

AWS IoT Greengrass 개발 도구를 사용하여 사용자 지정 Greengrass 구성 요소를 생성, 테스트, 빌드, 게시 및 배포합니다.

- [Greengrass 개발 키트 CLI](#)

로컬 개발 환경에서 AWS IoT Greengrass 개발 키트 명령줄 인터페이스(GDK CLI)를 사용하여 [Greengrass 소프트웨어 카탈로그](#)의 템플릿 및 커뮤니티 구성 요소에서 구성 요소를 생성합니다. GDK CLI를 사용하여 구성 요소를 빌드하고 AWS IoT Greengrass 서비스에 AWS 계정의 프라이빗 구성 요소로 게시합니다.

- [Greengrass 명령줄 인터페이스](#)

Greengrass 코어 디바이스에서 Greengrass 명령줄 인터페이스(Greengrass CLI)를 사용하여 Greengrass 구성 요소를 배포 및 디버깅합니다. 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 통해 Greengrass 구성 요소를 배포 및 디버깅합니다. 로컬 디버그 콘솔은 코어 디바이스에 배포하여 로컬 배포를 생성하고 설치된 구성 요소에 대한 세부 정보를 볼 수 있는 구성 요소입니다.

AWS IoT Greengrass는 사용자 지정 Greengrass 구성 요소에 사용할 수 있는 다음 SDK도 제공합니다.

- IPC(프로세스 간 통신) 라이브러리가 포함된 AWS IoT Device SDK. 자세한 내용은 [AWS IoT Device SDK 를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core](#) 섹션을 참조하세요.
- 데이터 스트림을 AWS 클라우드로 전송하는 데 사용할 수 있는 스트림 관리자 SDK. 자세한 내용은 [Greengrass 코어 디바이스에서 데이터 스트림 관리](#) 섹션을 참조하세요.

주제

- [AWS IoT Greengrass 개발 키트 명령줄 인터페이스](#)
- [Greengrass 명령줄 인터페이스](#)
- [AWS IoT Greengrass 테스트 프레임워크 사용](#)

AWS IoT Greengrass 개발 키트 명령줄 인터페이스

AWS IoT Greengrass 개발 키트 CLI(GDK CLI)는 [사용자 지정 Greengrass 구성 요소](#)를 개발하는 데 도움이 되는 기능을 제공합니다. GDK CLI를 사용하여 사용자 지정 구성 요소를 생성, 빌드 및 게시할 수 있습니다. GDK CLI를 사용하여 구성 요소 리포지토리를 생성할 때 [Greengrass 소프트웨어 카탈로그](#)의 템플릿 또는 커뮤니티 구성 요소에서 시작할 수 있습니다. 이후 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 AWS IoT Greengrass 서비스에 게시할 수 있으므로 AWS IoT Greengrass 콘솔 또는 API를 사용하여 Greengrass 코어 디바이스에 구성 요소를 배포할 수 있습니다.

GDK CLI 없이 Greengrass 구성 요소를 개발할 때는 구성 요소의 새 버전을 생성할 때마다 [구성 요소 레시피 파일](#)에서 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 소스로 GitHub에서 사용할 수 있습니다. 구성 요소 개발 요구 사항에 맞춰 GDK CLI를 사용자 지정 및 확장할 수 있습니다. GitHub 리포지토리에서 문제를 열고 요청을 가져오는 것이 좋습니다. GDK CLI 소스를 찾을 수 있는 링크: <https://github.com/aws-greengrass/aws-greengrass-gdk-cli>.

사전 조건

Greengrass 개발 키트 CLI를 설치 및 사용하려면 다음이 필요합니다.

- AWS 계정. 없으면 [설정 AWS 계정](#) 단원을 참조하세요.
- 인터넷에 연결된 Windows, macOS 또는 Unix 계열 개발 컴퓨터.
- GDK CLI 버전 1.1.0 이상의 경우 [Python](#) 3.6 이상이 개발 컴퓨터에 설치되어야 합니다.

GDK CLI 버전 1.0.0의 경우 [Python](#) 3.8 이상이 개발 컴퓨터에 설치되어야 합니다.

- 개발 컴퓨터에 설치된 [Git](#).
- 자격 증명을 사용하여 개발 컴퓨터에 AWS Command Line Interface(AWS CLI)가 설치되어 있어야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 [AWS CLI 설치, 업데이트, 제거](#) 및 [AWS CLI 구성](#) 단원을 참조하세요.

Note

Raspberry Pi 또는 다른 32비트 ARM 디바이스를 사용하는 경우 AWS CLI V1을 설치합니다. AWS CLI 32비트 ARM 디바이스에서는 V2를 사용할 수 없습니다. 자세한 내용은 [AWS CLI 버전 1 설치, 업데이트 및 제거](#) 섹션을 참조하세요.

- GDK CLI를 사용하여 AWS IoT Greengrass 서비스에 구성 요소를 게시하려면 다음 권한이 있어야 합니다.
 - s3:CreateBucket
 - s3:GetBucketLocation
 - s3:PutObject
 - greengrass:CreateComponentVersion
 - greengrass:ListComponentVersions
- GDK CLI를 사용하여 아티팩트가 로컬 파일 시스템이 아닌 S3 버킷에 있는 구성 요소를 빌드하려면 다음 권한이 있어야 합니다.

- `s3:ListBucket`

이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

Changelog

다음 표에서는 GDK CLI의 각 버전에서 변경된 사항에 대해 설명합니다. 자세한 내용은 GitHub의 [GDK CLI Releases 페이지](#)를 참조하세요.

버전:	변경 사항
1.6.2	버그 수정 및 개선 사항 • 상대 경로로 인해 Windows gradlew.bat이 작동하지 않는 문제를 해결합니다. • 로깅, 테스트 및 패키징에 대한 사소한 개선 사항.
1.6.1	버그 수정 및 개선 사항 • CLI 인수 구문 분석에 대한 보안 수정 사항이 추가됩니다. • GDK가 최신 GTF(Greengrass Testing Framework) 릴리스 이름을 기본 GTF 버전으로 가져올 수 있습니다. • GDK가 이전 버전의 GTF를 사용하는 고객에게 최신 버전으로 업데이트할 것을 권장할 수 있습니다.
1.6.0	새로운 특성 • <code>component build</code> 및 <code>component publish</code> 명령 도중 Greengrass 레시피 스키마에 대한 레시피 검증 검사가 추가됩니다. 이 업데이트를 통해 개발자는 구성 요소 생성 프로세스 초기에 구성 요소 레시피 내에서 조치를 취할 수 있는 문제를 식별할 수 있습니다. • <code>test-e2e init</code> 명령으로 가져올 수 있는 템플릿에 신뢰도 테스트 제품군이 추가됩니다. 이 신뢰도 테스트 제품군에는 기본 구성 요소 테스트 요구 사항에 맞춰 사용하고 확장할 수 있는 8가지 일반 테스트가 포함되어 있습니다. 버그 수정 및 개선 사항 • <code>test-e2e</code> 명령에서 사용하는 기본 GTF(Greengrass Testing Framework) 버전이 1.2.0으로 업데이트됩니다.

버전:	변경 사항
1.5.0	버그 수정 및 개선 사항 <code>build_system</code> 가 <code>zip</code>인 경우 <code>excludes</code> 빌드 옵션에서 인식하는 패턴이 업데이트됩니다. 이제 이 버전에서 와일드카드 문자를 기반으로 경로 이름과 일치하는 <code>glob</code> 패턴을 인식합니다. 이를 통해 제외할 디렉터리를 사용자 지정할 수 있습니다.
1.4.0	새로운 특성 - 기존 GDK 구성 파일 내의 필드를 수정하는 대화형 프롬프트를 시작하는 <code>config</code> 명령이 새로 추가됩니다. - 계속하기 전에 레시피 크기가 Greengrass 요구 사항(16,000바이트 이하) 내에 있는지 확인하기 위해 <code>gdk component build</code> 및 <code>gdk component publish</code> 명령이 수정됩니다. 버그 수정 및 개선 사항 - 레시피 구문 오류로 인해 빌드 인식이 완료되지 못할 경우 <code>gdk component build</code> 명령 출력에 로깅이 추가됩니다. - Open Test Framework의 이름이 Greengrass Testing Framework로 변경되면서 <code>otf-options</code> 및 <code>otf-version</code> 의 이름이 <code>gtf-options</code> 및 <code>gtf-version</code> 으로 변경됩니다.
1.3.0	새로운 특성 - Open Test Framework를 사용하여 구성 요소의 종단 간 테스트를 지원하는 <code>test-e2e</code> 명령이 새로 추가됩니다. <code>zip</code> 빌드 시스템에서 구성 가능한 <code>zip</code> 파일 이름을 지원하기 위해 새 구성 옵션인 <code>zip_name</code>이 추가됩니다. - GDK 구성 파일의 <code>region</code> 속성이 선택 사항으로 설정됩니다. 버그 수정 및 개선 사항 <code>--name</code> 인수로 GDK 프로젝트를 초기화할 때 지정된 템플릿 또는 리포지토리가 존재하지 않더라도 새 디렉터리가 생성되는 문제를 해결합니다.

버전:	변경 사항
1.2.3	버그 수정 및 개선 사항 • 잘못된 오류 처리로 인해 버킷 생성이 실패하는 문제를 해결합니다. • 구성 요소 레시피의 목록 구조가 제거되는 문제를 해결합니다.
1.2.2	버그 수정 및 개선 사항 • 레시피 키가 더 이상 대/소문자를 구분하지 않습니다. • 새 버킷을 생성하기 전에 버킷이 AWS 리전에 있고 사용자가 액세스할 수 있는지 여부를 확인하는 검사가 추가됩니다. 사용자에게 GetBucket Location 권한이 있어야 합니다. • GDK CLI 구성 파일의 excludes 키워드 관련 문제를 해결합니다.
1.2.1	버그 수정 및 개선 사항 • gdk-config.json 파일의 리전 구성 항목에서 캐나다(중부)(ca-central-1) AWS 리전이 수락됩니다. • publish 명령에 대한 --region GDK CLI 인수 문제가 해결됩니다.
1.2.0	새로운 특성 • GDK CLI build 구성 파일의 구성에 options 항목이 추가됩니다. options에서 excludes가 지원되어 zip 빌드 시스템을 사용할 때 zip 아티팩트에서 특정 파일을 제외할 수 있습니다. • Gradle Wrapper를 사용하여 구성 요소를 빌드하는 gradlew 빌드 시스템이 추가됩니다. • gradle 빌드 옵션에 대해 Kotlin DSL 빌드 파일 지원을 추가합니다. • GDK CLI publish 구성 파일의 구성에 options 항목이 추가됩니다. Amazon S3에 파일을 업로드할 때 추가 인수를 제공하기 위해 options에서 file_upload_args 가 지원됩니다. 버그 수정 및 개선 사항 • build 명령을 실행하기 전에 Gradle 빌드가 정리되지 않은 문제를 해결합니다. • build 명령이 실패할 때 빌드가 종료되지 않는 문제를 해결합니다. • gdk component list 명령의 출력 형식이 개선됩니다.

버전:	변경 사항
1.1.0	새로운 특성 • Gradle 빌드 시스템에 대한 지원을 추가합니다. • Windows 디바이스의 Maven 빌드 시스템에 대한 지원을 추가합니다. • component publish 명령에 --bucket 인수가 추가됩니다. 이 인수를 사용하여 GDK CLI가 구성 요소 아티팩트를 업로드하는 정확한 버킷을 지정할 수 있습니다. • component init 명령에 --name 인수가 추가됩니다. 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 사용하면 기계 학습 모델과 같은 큰 구성 요소 아티팩트의 대역폭 비용을 줄일 수 있습니다. 버그 수정 및 개선 사항 • component publish 명령이 업데이트되어 구성 요소 게시 전에 구성 요소가 빌드되었는지 확인합니다. 구성 요소가 빌드되지 않은 경우 이제 이 명령을 사용하여 구성 요소가 빌드됩니다. • ZIP 파일 이름에 대문자가 포함된 경우 zip 빌드 시스템이 Windows 디바이스에 빌드되지 않는 문제를 해결합니다. • 로그 메시지 형식이 개선되고 Python 버전 3.8 이하를 실행하는 디바이스에서 기본 로그 수준이 INFO로 변경됩니다. • 최소 Python 버전 요구 사항이 Python 3.6으로 변경됩니다.
1.0.0	초기 버전입니다.

AWS IoT Greengrass 개발 키트 명령줄 인터페이스 설치 또는 업데이트

AWS IoT Greengrass 개발 키트 명령줄 인터페이스(GDK CLI)는 Python을 기반으로 빌드되므로 pip를 사용하여 개발 컴퓨터에 설치할 수 있습니다.

Tip

GDK CLI를 [venv](#) 와 같은 Python 가상 환경에 설치할 수도 있습니다. 자세한 내용은 Python 3 설명서의 [가상 환경 및 패키지](#)를 참조하세요.

GDK CLI 설치 또는 업데이트

1. 다음 명령을 실행하여 [GitHub 리포지토리](#)에서 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

GDK CLI의 특정 버전을 설치하려면 *versionTag*를 설치할 버전 태그로 바꿉니다. [GitHub 리포지토리](#)에서 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
```

gdk 명령을 찾을 수 없는 경우 해당 폴더를 PATH에 추가합니다.

- Linux 디바이스에서 PATH에 /home/*MyUser*/.local/bin을 추가하고 *MyUser*를 사용자 이름으로 바꿉니다.
- Windows 디바이스에서 PATH에 *PythonPath*\\Scripts를 추가하고 *PythonPath*를 디바이스의 Python 폴더 경로로 바꿉니다.

이제 GDK CLI를 사용하여 Greengrass 구성 요소를 생성, 빌드 및 게시할 수 있습니다. GDK CLI 사용 방법에 대한 자세한 내용은 [AWS IoT Greengrass 개발 키트 명령줄 인터페이스 명령](#) 섹션을 참조하세요.

AWS IoT Greengrass 개발 키트 명령줄 인터페이스 명령

AWS IoT Greengrass 개발 키트 명령줄 인터페이스(GDK CLI)는 개발 컴퓨터에 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 설치](#)하면 설치 관리자가 PATH에 gdk를 추가하여 명령줄에서 GDK CLI를 실행할 수 있습니다.

모든 명령에서 다음 인수를 사용할 수 있습니다.

- -h 또는 --help를 사용하면 GDK CLI 명령에 대한 정보를 확인합니다.
- -v 또는 --version을 사용하여 설치된 GDK CLI 버전을 확인합니다.
- -d 또는 --debug를 사용하면 GDK CLI 디버깅에 사용할 수 있는 세부 로그가 출력됩니다.

이 섹션에서는 GDK CLI 명령을 설명하고 각 명령의 예제를 제공합니다. 각 명령의 개요에는 인수와 용도가 표시됩니다. 선택적 인수는 대괄호에 표시됩니다.

가용 명령

- [구성 요소](#)
- [config](#)
- [test-e2e](#)

구성 요소

component 개발 키트 명령줄 인터페이스(GDK CLI)의 AWS IoT Greengrass 명령을 사용하여 사용자 지정 Greengrass 구성 요소를 생성, 빌드 및 게시합니다.

하위 명령

- [init](#)
- [빌드](#)
- [publish](#)
- [list](#)

init

구성 요소 템플릿 또는 커뮤니티 구성 요소에서 Greengrass 구성 요소 폴더를 초기화합니다.

GDK CLI는 [Greengrass 소프트웨어 카탈로그](#)에서 커뮤니티 구성 요소를 검색하고 [AWS IoT Greengrass GitHub의 구성 요소 템플릿 리포지토리](#)에서 구성 요소 템플릿을 검색합니다.

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 `--name` 인수를 지정하여 GDK CLI가 템플릿 또는 커뮤니티 구성 요소를 다운로드하는 폴더를 지정할 수 있습니다. 이 인수를 사용하는 경우 존재하지 않는 폴더를 지정해야 합니다. GDK CLI가 폴더를 생성합니다. 이 인수를 지정하지 않으면 GDK CLI에서 현재 폴더를 사용하고, 이 폴더는 비어 있어야 합니다.

구성 요소가 [zip 빌드 시스템](#)을 사용하는 경우 GDK CLI는 구성 요소의 폴더에 있는 특정 파일을 구성 요소 폴더와 이름이 동일한 zip 파일로 압축합니다. 예를 들어 구성 요소 폴더의 이름이 HelloWorld인 경우 GDK CLI는 HelloWorld.zip이라는 파일을 생성합니다. 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 이름이 다른 폴더로 초기화하는 경우 구성 요소 레시피에서 zip 아티팩트 이름을 변경해야 합니다. zip 파일 이름이 구성 요소 폴더의 이름과 일치하도록 Artifacts 및 Lifecycle 정의를 업데이트합니다. 다음 예제에서는 Artifacts 및 Lifecycle 정의의 zip 파일 이름이 강조 표시됩니다.

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 --language와 --template을 지정해야 합니다.
- -t, --template - 로컬 구성 요소 프로젝트에 사용할 구성 요소 템플릿입니다. 사용 가능한 템플릿을 보려면 [list](#) 명령을 사용합니다.
--repository 또는 --language와 --template을 지정해야 합니다.
- -n, --name - (선택 사항) GDK CLI가 구성 요소를 초기화하는 로컬 폴더의 이름입니다. 존재하지 않는 폴더를 지정해야 합니다. GDK CLI가 폴더를 생성합니다.

이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

인수(커뮤니티 구성 요소에서 초기화)

- `-r, --repository` - 로컬 폴더에서 확인할 커뮤니티 구성 요소입니다. 사용 가능한 커뮤니티 구성 요소를 보려면 [list](#) 명령을 사용합니다.
- `--repository` 또는 `--language`와 `--template`를 지정해야 합니다.
- `-n, --name` - (선택 사항) GDK CLI가 구성 요소를 초기화하는 로컬 폴더의 이름입니다. 존재하지 않는 폴더를 지정해야 합니다. GDK CLI가 폴더를 생성합니다.

이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

출력:

다음 예제는 Python Hello World 템플릿에서 구성 요소 폴더를 초기화하기 위해 이 명령을 실행할 때 생성된 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 커뮤니티 구성 요소에서 구성 요소 폴더를 초기화하기 위해 이 명령을 실행할 때 생성된 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.
```

빌드

구성 요소의 소스를 AWS IoT Greengrass 서비스에 게시할 수 있는 레시피와 아티팩트로 빌드합니다. GDK CLI는 [GDK CLI 구성 파일](#)인 `gdk-config.json`에서 지정한 빌드 시스템을 실행합니다. `gdk-config.json` 파일이 있는 곳과 동일한 폴더에서 이 명령을 실행해야 합니다.

이 명령을 실행하면 GDK CLI는 구성 요소 폴더의 `greengrass-build` 폴더에 레시피와 아티팩트를 생성합니다. GDK CLI는 레시피를 `greengrass-build/recipes` 폴더에 저장하고 아티팩트를 `greengrass-build/artifacts/componentName/componentVersion` 폴더에 저장합니다.

GDK CLI v1.1.0 이상을 사용하는 경우 구성 요소 레시피는 S3 버킷에는 있지만 로컬 구성 요소 빌드 폴더에 없는 아티팩트를 지정할 수 있습니다. 이 기능을 사용하면 기계 학습 모델과 같은 큰 아티팩트가 포함된 구성 요소를 개발할 때 대역폭 사용량을 줄일 수 있습니다.

구성 요소를 빌드한 후 다음 중 하나를 수행하여 Greengrass 코어 디바이스에서 테스트할 수 있습니다.

- AWS IoT Greengrass 코어 소프트웨어를 실행하는 디바이스가 아닌 다른 디바이스에서 개발하는 경우 구성 요소를 게시하여 Greengrass 코어 디바이스에 배포해야 합니다. AWS IoT Greengrass 서비스에 구성 요소를 게시하고 Greengrass 코어 디바이스에 배포합니다. 자세한 내용은 [publish](#) 명령과 [배포 만들기](#) 섹션을 참조하세요.
- AWS IoT Greengrass 코어 소프트웨어를 실행하는 것과 동일한 디바이스에서 개발하는 경우 구성 요소를 AWS IoT Greengrass 서비스에 게시하여 배포하거나 로컬 배포를 생성하여 구성 요소를 설치하고 실행할 수 있습니다. 로컬 배포를 생성하려면 Greengrass CLI를 사용합니다. 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 및 [로컬 배포를 사용한 AWS IoT Greengrass 구성 요소 테스트](#)(을)를 참조하세요. 로컬 배포를 생성할 때 `greengrass-build/recipes`를 레시피 폴더로 지정하고 `greengrass-build/artifacts`를 아티팩트 폴더로 지정합니다.

시놉시스

```
$ gdk component build
```

인수:

없음

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, 레시피에서 아티팩트 URI를 업데이트하고, 레시피에서 새 버전의 구성 요소를 생성합니다. GDK CLI는 [GDK CLI 구성 파일](#)인 gdk-config.json에서 지정한 S3 버킷 및 AWS 리전을 사용합니다. gdk-config.json 파일이 있는 곳과 동일한 폴더에서 이 명령을 실행해야 합니다.

GDK CLI v1.1.0 이상을 사용하는 경우 --bucket 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 수 있습니다. 이 인수를 지정하지 않으면 GDK CLI가 이름이 **bucket-region-accountId**인 S3 버킷에 업로드합니다. 여기서 **bucket**과 **region**은 gdk-config.json에서 지정하는 값이고, **accountId**는 AWS 계정 ID입니다. GDK CLI는 버킷이 없는 경우 버킷을 생성합니다.

GDK CLI v1.2.0 이상을 사용하는 경우 --region 파라미터를 사용하여 GDK CLI 구성 파일에 지정된 AWS 리전을 재정의할 수 있습니다. --options 파라미터를 사용하여 추가 옵션을 지정할 수도 있습니다. 사용 가능한 옵션의 목록은 [Greengrass 개발 키트 CLI 구성 파일](#) 섹션을 참조하세요.

이 명령을 실행하면 GDK CLI는 레시피에 지정한 버전으로 구성 요소를 게시합니다. NEXT_PATCH를 지정하면 GDK CLI는 아직 존재하지 않는 다음 패치 버전을 사용합니다. 의미 체계 버전은 메이저.마이너.패치 번호 시스템을 사용합니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

Note

GDK CLI v1.1.0 이상을 사용하는 경우 이 명령을 실행하면 GDK CLI가 구성 요소 빌드 여부를 확인합니다. 구성 요소가 빌드되지 않으면 GDK CLI는 구성 요소를 게시하기 전에 [구성 요소를 빌드](#)합니다.

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 CLI가 이름이 `bucket-region-accountId`인 S3 버킷에 업로드합니다. 여기서 `bucket`과 `region`은 `gdk-config.json`에서 지정하는 값이고, `accountId`는 AWS 계정 ID입니다. GDK CLI는 버킷이 없는 경우 버킷을 생성합니다.

GDK CLI는 버킷이 없는 경우 버킷을 생성합니다.

이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

- `-r, --region` – (선택 사항) 구성 요소가 생성될 때 AWS 리전의 이름을 지정합니다. 이 인수는 GDK CLI 구성의 리전 이름을 재정의합니다.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `-o, --options` (선택 사항) 구성 요소 게시를 위한 옵션 목록을 지정합니다. 인수는 유효한 JSON 문자열이거나 게시 옵션이 포함된 JSON 파일의 파일 경로여야 합니다. 이 인수는 GDK CLI 구성의 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 검색합니다.

GDK CLI는 [Greengrass 소프트웨어 카탈로그](#)에서 커뮤니티 구성 요소를 검색하고 [AWS IoT Greengrass GitHub의 구성 요소 템플릿 리포지토리](#)에서 구성 요소 템플릿을 검색합니다.

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) 이 인수를 지정하면 사용 가능한 구성 요소 템플릿이 나열됩니다. 이 명령은 *name-language* 형식으로 각 템플릿의 이름과 언어를 출력합니다. 예를 들어 HelloWorld-python에서 템플릿 이름은 HelloWorld이고 언어는 python입니다.
- -r, --repository – (선택 사항) 이 인수를 지정하면 사용 가능한 커뮤니티 구성 요소 리포지토리가 나열됩니다.

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 Greengrass 개발 키트 명령줄 인터페이스(GDK CLI)의 config 명령을 사용하여 구성 파일인 gdk-config.json에서 GDK에 대한 구성을 수정합니다.

하위 명령

- [업데이트](#)

업데이트

기존 GDK 구성 파일 내의 필드를 수정하는 대화형 프롬프트를 시작합니다.

시놉시스

```
$ gdk config update  
  [--component]
```

인수:

- -c, --component - gdk-config.json 파일의 구성 요소 관련 필드를 업데이트합니다. 이 인수는 유일한 옵션이므로 필수 사항입니다.

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...
```

test-e2e

AWS IoT Greengrass 개발 키트 명령줄 인터페이스(GDK CLI)의 test-e2e 명령을 사용하여 GDK 프로젝트에서 종단 간 테스트 모듈을 초기화, 빌드 및 실행합니다.

하위 명령

- [init](#)
- [빌드](#)
- [실행](#)

init

GTF(Greengrass Testing Framework)를 사용하는 테스트 모듈로 기존 GDK CLI 프로젝트를 초기화합니다.

기본적으로 GDK CLI는 [AWS IoT Greengrass GitHub의 구성 요소 템플릿 리포지토리](#)에서 Maven 모듈 템플릿을 검색합니다. 이 Maven 모듈은 `aws-greengrass-testing-standalone` JAR 파일에 대한 종속성과 함께 제공됩니다.

이 명령은 GDK 프로젝트 내에 `gg-e2e-tests`라는 새 디렉터리를 생성합니다. 테스트 모듈 디렉터리가 이미 있고 비어 있지 않은 경우 아무 작업도 실시하지 않고 명령이 종료됩니다. 이 `gg-e2e-tests` 폴더에는 Maven 프로젝트에 구조화된 Cucumber 기능과 단계 정의가 포함되어 있습니다.

기본적으로 이 명령은 GTF의 최신 릴리스 버전을 사용하려고 합니다.

시놉시스

```
$ gdk test-e2e init  
  [--gtf-version]
```

인수:

- `-ov, --gtf-version` - (선택 사항) GDK 프로젝트의 종단 간 테스트 모듈에 사용할 GTF 버전입니다. 이 값은 [releases](#)의 GTF 버전 중 하나여야 합니다. 이 인수는 GDK CLI 구성의 `gtf_version`을 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...
```

빌드

Note

종단 간 테스트 모듈을 빌드하기 전에 gdk component build를 실행하여 구성 요소를 빌드해야 합니다.

종단 간 테스트 모듈을 구축합니다. GDK CLI는 test-e2e 속성 아래에서 [GDK CLI 구성 파일](#)인 gdk-config.json에 지정한 빌드 시스템을 사용하여 테스트 모듈을 빌드합니다. gdk-config.json 파일이 있는 곳과 동일한 폴더에서 이 명령을 실행해야 합니다.

기본적으로 GDK CLI는 Maven 빌드 시스템을 사용하여 테스트 모듈을 빌드합니다. gdk test-e2e build 명령을 실행하려면 [Maven](#)이 필요합니다.

테스트 기능 파일에 GDK_COMPONENT_NAME 및 GDK_COMPONENT_RECIPE_FILE과 같이 보관할 변수가 있는 경우 테스트 모듈을 빌드하기 전에 gdk-component-build를 실행하여 구성 요소를 빌드해야 합니다.

이 명령을 실행하면 GDK CLI는 GDK 프로젝트 구성에서 모든 변수를 보관하고 gg-e2e-tests 모듈을 빌드하여 최종 테스트 JAR 파일을 생성합니다.

시놉시스

```
$ gdk test-e2e build
```

인수:

없음

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'
```

.....

실행

GDK 구성 파일의 테스트 옵션을 사용하여 테스트 모듈을 실행합니다.

Note

종단 간 테스트를 실행하기 전에 `gdk test-e2e build`를 실행하여 테스트 모듈을 빌드해야 합니다.

시놉시스

```
$ gdk test-e2e run
  [--gtf-options]
```

인수:

- `-oo`, `--gtf-options` - (선택 사항) 종단 간 테스트를 실행하기 위한 옵션 목록을 지정합니다. 인수는 유효한 JSON 문자열이거나 GTF 옵션이 포함된 JSON 파일의 파일 경로여야 합니다. 구성 파일에 제공된 옵션은 명령 인수에 제공된 옵션과 병합됩니다. 두 곳 모두에 옵션이 있는 경우 인수의 옵션이 구성 파일의 옵션보다 우선합니다.

이 명령에 `tags` 옵션이 지정되지 않은 경우 GDK는 태그에 `Sample`을 사용합니다. `ggc-archive`가 지정되지 않으면 GDK는 최신 버전의 Greengrass nucleus 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
.....
```

Greengrass 개발 키트 CLI 구성 파일

AWS IoT Greengrass 개발 키트 명령줄 인터페이스(GDK CLI)는 `gdk-config.json` 구성 파일로부터 읽어 구성 요소를 빌드 및 게시합니다. 이 구성 파일은 구성 요소 리포지토리의 루트에 있어야 합니다. GDK CLI [init 명령](#)을 사용하여 이 구성 파일로 구성 요소 리포지토리를 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 다음 정보를 JSON 형식으로 지정합니다.

`gdk_version`

이 구성 요소와 호환되는 GDK CLI의 최소 버전입니다. 이 값은 [releases](#)의 GDK CLI 버전 중 하나여야 합니다.

`component`

이 구성 요소의 구성입니다.

componentName

`author`

구성 요소의 작성자 또는 게시자입니다.

`version`

구성 요소의 버전입니다. 다음 중 하나를 지정하세요.

- `NEXT_PATCH` – 이 옵션을 선택하면 구성 요소를 게시할 때 GDK CLI에서 버전이 설정됩니다. GDK CLI에서는 AWS IoT Greengrass 서비스 쿼리를 통해 구성 요소의 최신 게시

버전이 식별됩니다. 그런 다음에 버전이 해당 버전 이후의 다음 패치 버전으로 설정됩니다. 이전에 구성 요소를 게시하지 않았으면 GDK CLI에서는 1.0.0 버전이 사용됩니다.

이 옵션을 선택하면 [Greengrass CLI](#)를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 실행하는 로컬 개발 컴퓨터에 구성 요소를 로컬로 배포하고 테스트할 수 없습니다. 로컬 배포를 활성화하려면 대신 의미 체계 버전을 지정해야 합니다.

- 의미 체계 버전(예: **1.0.0**). 의미 체계 버전에서는 major.minor.patch 번호 매기기 시스템이 사용됩니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

구성 요소를 배포하고 테스트하려는 Greengrass 코어 디바이스에서 구성 요소를 개발하는 경우 이 옵션을 선택합니다. [Greengrass CLI](#)를 사용하여 로컬 배포를 생성하려면 특정 버전으로 구성 요소를 빌드해야 합니다.

build

이 구성 요소의 소스를 아티팩트에 빌드하는 데 사용할 구성입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

build_system

사용할 빌드 시스템입니다. 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 JavaScript와 같이 해석된 프로그래밍 언어가 사용되는 구성 요소.
 - 기계 학습 모델 또는 기타 리소스와 같이 코드 이외의 파일이 패키징되는 구성 요소.

GDK CLI에서는 구성 요소의 폴더를 이름이 동일한 zip 파일로 압축됩니다. 예를 들어, 구성 요소 폴더의 이름이 HelloWorld라면 GDK CLI에서는 HelloWorld.zip이라는 파일이 생성됩니다.

Note

Windows 디바이스에서 GDK CLI 버전 1.0.0을 사용하는 경우 구성 요소 폴더 및 zip 파일 이름에는 소문자만 포함되어야 합니다.

GDK CLI에서는 구성 요소의 폴더가 zip 파일로 압축하면 다음 파일을 건너뛵니다.

- gdk-config.json 파일
- 레시피 파일(recipe.json 또는 recipe.yaml)

- 빌드 폴더(예: greengrass-build)
- `maven -mvn clean package` 명령을 실행하여 구성 요소의 소스를 아티팩트에 빌드합니다. Java 구성 요소와 같이 [Maven](#)을 사용하는 구성 요소의 경우 이 옵션을 선택합니다.

Windows 디바이스에서 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

- `gradle - gradle build` 명령을 실행하여 구성 요소의 소스를 아티팩트에 빌드합니다. [Gradle](#)을 사용하는 구성 요소의 경우 이 옵션을 선택합니다. 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

gradle 빌드 시스템에서는 빌드 파일로 Kotlin DSL이 지원됩니다. 이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `gradlew - gradlew` 명령을 실행하여 구성 요소의 소스를 아티팩트에 빌드합니다. [Gradle Wrapper](#)를 사용하는 구성 요소의 경우 이 옵션을 선택합니다.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `custom` - 사용자 지정 명령을 실행하여 구성 요소의 소스를 레시피 및 아티팩트에 빌드합니다. `custom_build_command` 파라미터에서 사용자 지정 명령을 지정합니다.

`custom_build_command`

(선택 사항) 사용자 지정 빌드 시스템에 대해 실행할 사용자 지정 빌드 명령입니다. `build_system`에 `custom`을 지정한 경우 이 파라미터를 지정해야 합니다.

Important

이 명령으로 구성 요소 폴더 내의 다음 폴더에 레시피와 아티팩트를 생성될 것입니다. GDK CLI는 [구성 요소 빌드 명령](#)을 실행할 때 이러한 폴더를 생성합니다.

- 레시피 폴더: `greengrass-build/recipes`
- 아티팩트 폴더: `greengrass-build/artifacts/componentName/componentVersion`

*componentName*을 구성 요소 이름으로 바꾸고 *componentVersion*을 구성 요소 버전 또는 NEXT_PATCH로 바꿉니다.

단일 문자열 또는 문자열 목록을 지정할 수 있으며, 여기서 각 문자열은 명령의 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 예제를 보려면 [GitHub의 aws.greengrass.labs.LocalWebServer community component](#)를 참조하세요.

options

(선택 사항) 구성 요소 빌드 프로세스 중에 사용되는 추가 구성 옵션입니다.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

excludes

zip 파일을 빌드할 때 구성 요소 디렉터리에서 제외할 파일을 정의하는 glob 패턴 목록입니다. build_system이 zip인 경우에만 유효합니다.

Note

GDK CLI 버전 1.4.0 이하에서는 제외 목록의 항목과 일치하는 모든 파일이 구성 요소의 모든 하위 디렉터리에서 제외됩니다. GDK CLI 버전 1.5.0 이상에서 동일한 동작을 하려면 제외 목록의 기존 항목 ****/*** 기호를 추가합니다. 예를 들어 *.txt는 디렉터리에서 텍스트 파일을 제외하고, ****/*.*.txt**는 모든 디렉터리 및 하위 디렉터리에서 텍스트 파일을 제외합니다.

GDK CLI 버전 1.5.0 이상에서는 excludes가 GDK 구성 파일에 정의되어 있을 때 구성 요소 빌드 도중 경고가 표시될 수 있습니다. 이 경고를 비활성화하려면 환경 변수 **GDK_EXCLUDES_WARN_IGNORE**를 true로 설정합니다.

GDK CLI는 항상 zip 파일에서 다음 파일을 제외합니다.

- gdk-config.json 파일
- 레시피 파일(recipe.json 또는 recipe.yaml)
- 빌드 폴더(예: greengrass-build)

다음 파일은 기본적으로 제외됩니다. 하지만 excludes 옵션으로 제외되는 파일을 제어할 수 있습니다.

- 접두사 "test"로 시작하는 모든 폴더(test*)
- 숨겨진 모든 파일

- `node_modules` 폴더

`excludes` 옵션을 지정하면 GDK CLI는 `excludes` 옵션으로 설정한 파일만 제외합니다. `excludes` 옵션을 지정하지 않으면 GDK CLI는 이전에 언급한 기본 파일 및 폴더를 제외합니다.

`zip_name`

빌드 프로세스 중에 zip 아티팩트를 생성할 때 사용할 zip 파일 이름입니다.

`build_system`이 zip인 경우에만 유효합니다. `build_system`이 비어 있으면 구성 요소 이름이 zip 파일 이름에 사용됩니다.

`publish`

이 구성 요소를 AWS IoT Greengrass 서비스에 게시하는 데 사용할 구성입니다.

GDK CLI v1.1.0 이상을 사용하는 경우 `--bucket` 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 수 있습니다. 이 인수를 지정하지 않으면 GDK CLI가 이름이 `bucket-region-accountId`인 S3 버킷에 업로드합니다. 여기서 `bucket`과 `region`은 `gdk-config.json`에서 지정하는 값이고, `accountId`는 AWS 계정 ID입니다. GDK CLI는 버킷이 없는 경우 버킷을 생성합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

`bucket`

구성 요소 아티팩트를 호스팅하는 데 사용할 S3 버킷 이름입니다.

`region`

GDK CLI가 이 구성 요소를 게시하는 AWS 리전입니다.

GDK CLI v1.3.0 이상을 사용하는 경우 이 속성은 선택 사항입니다.

`options`

(선택 사항) 구성 요소 버전 생성 중에 사용되는 추가 구성 옵션입니다.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

`file_upload_args`

메타데이터 및 암호화 메커니즘과 같이 버킷에 파일을 업로드하는 동안 Amazon S3로 전송된 인수가 포함된 JSON 구조입니다. 허용되는 인수 목록은 Boto3 설명서의 [S3Transfer](#) 클래스를 참조하세요.

test-e2e

(선택 사항) 구성 요소의 엔드 투 엔드 테스트에 사용할 구성입니다. 이 기능은 GDK CLI v1.3.0 이상에서 사용할 수 있습니다.

build

`build_system` - 사용할 빌드 시스템입니다. 기본값은 maven입니다. 다음 옵션 중 하나를 선택합니다.

- `maven - mvn package` 명령을 실행하여 테스트 모듈을 빌드합니다. 이 옵션을 선택하면 [Maven](#)을 사용하는 테스트 모듈을 빌드합니다.
- `gradle - gradle build` 명령을 실행하여 테스트 모듈을 빌드합니다. [Gradle](#)을 사용하는 테스트 모듈에 이 옵션을 선택합니다.

gtf_version

(선택 사항) GTF(Greengrass Testing Framework)로 GDK 프로젝트를 초기화할 때 엔드 투 엔드 테스트 모듈의 종속성으로 사용할 GTF의 버전입니다. 이 값은 [releases](#)의 GTF 버전 중 하나여야 합니다. 기본 버전은 GTF 1.1.0입니다.

gtf_options

(선택 사항) 구성 요소의 엔드 투 엔드 테스트 중에 사용되는 추가 구성 옵션입니다.

다음 목록에는 GTF 버전 1.1.0에서 사용할 수 있는 옵션이 있습니다.

- `additional-plugins` - (선택 사항) 추가 Cucumber 플러그인
- `aws-region` - AWS 서비스에 대한 특정 리전 엔드포인트를 대상으로 합니다. 기본값은 AWS SDK에서 검색되는 내용입니다.
- `credentials-path` - 선택적 AWS 프로파일 자격 증명 경로입니다. 기본값은 호스트 환경에서 검색된 자격 증명입니다.
- `credentials-path-rotation` - AWS 자격 증명의 선택적 교체 기간입니다. 기본값은 15 분 또는 PT15M입니다.
- `csr-path` - CSR에서 사용되어 디바이스 인증서가 생성되는 경로입니다.
- `device-mode` - 테스트 중인 대상 디바이스입니다. 기본값은 로컬 디바이스입니다.
- `env-stage` - Greengrass의 배포 환경을 대상으로 합니다. 기본값은 프로덕션입니다.
- `existing-device-cert-arn` - Greengrass의 디바이스 인증서로 사용하려는 기존 인증서의 arn입니다.

- `feature-path` – 추가 특성 파일이 포함된 파일 또는 디렉터리입니다. 기본적으로 추가 특성 파일은 사용되지 않습니다.
- `gg-cli-version` – Greengrass CLI의 버전을 재정의합니다. 기본값은 `ggc.version`에 있는 값입니다.
- `gg-component-bucket` - Greengrass 구성 요소가 보관되는 기존 Amazon S3 버킷의 이름입니다.
- `gg-component-overrides` – Greengrass 구성 요소 재정의 목록입니다.
- `gg-persist` - 테스트 실행 후 유지할 테스트 요소 목록입니다. 기본 동작은 아무것도 유지하지 않는 것입니다. 허용되는 값은 `aws.resources`, `installed.software` 및 `generated.files`입니다.
- `gg-runtime` - 테스트가 테스트 리소스와 상호 작용하는 방식에 영향을 미치는 값의 목록입니다. 이러한 값은 `gg.persist` 파라미터를 대체합니다. 기본값이 비어 있는 경우 설치된 Greengrass 런타임을 포함하여 모든 테스트 리소스가 테스트 사례로 관리된다고 가정합니다. 허용되는 값은 `aws.resources`, `installed.software` 및 `generated.files`입니다.
- `ggc-archive` – 보관된 Greengrass nucleus 구성 요소의 경로입니다.
- `ggc-install-root` – Greengrass nucleus 구성 요소를 설치하는 디렉터리입니다. 기본값은 `test.temp.path` 및 테스트 실행 폴더입니다.
- `ggc-log-level` – 테스트 실행에 대한 Greengrass nucleus 로그 수준을 설정합니다. 기본값은 'INFO'입니다.
- `ggc-tes-rolename` – AWS IoT Greengrass 코어가 AWS 서비스에 액세스하기 위해 맡을 IAM 역할입니다. 지정된 이름의 역할이 없는 경우 역할이 생성되고 기본 액세스 정책이 생성됩니다.
- `ggc-trusted-plugins` – Greengrass에 추가되어야 하는 신뢰할 수 있는 플러그인의 경로 (호스트)를 쉼표로 구분한 목록입니다. DUT 자체의 경로를 제공하려면 경로에 'dut:' 접두사를 붙입니다.
- `ggc-user-name` – Greengrass nucleus의 `user:group posixUser` 값입니다. 기본값은 로그인한 현재 사용자 이름입니다.
- `ggc-version` - 실행 중인 Greengrass nucleus 구성 요소의 버전을 재정의합니다. 기본값은 `ggc.archive`에 있는 값입니다.
- `log-level` – 테스트 실행의 로그 수준입니다. 기본값은 'INFO'입니다.
- `parallel-config` - 배치 인덱스 및 배치 수 세트입니다(JSON 문자열). 배치 인덱스의 기본값은 0이고 배치 수는 1입니다.
- `proxy-url` - 이 URL을 통해 트래픽을 라우팅하도록 모든 테스트를 구성합니다.

- `tags` - 특성 태그만 실행합니다. '&' 문자와 교차할 수 있습니다.
- `test-id-prefix` - AWS 리소스 이름 및 태그를 포함하여 모든 테스트별 리소스에 적용되는 일반 접두사입니다. 기본값은 'gg' 접두사입니다.
- `test-log-path` - 전체 테스트 실행의 결과를 포함하는 디렉터리입니다. 기본값은 'testResults'입니다.
- `test-results-json` - 결과로 나오는 Cucumber JSON 보고서가 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 true입니다.
- `test-results-log` - 콘솔 출력이 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 false입니다.
- `test-results-xml` - 결과로 나오는 JUnit XML 보고서가 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 true입니다.
- `test-temp-path` - 로컬 테스트 아티팩트를 생성하는 디렉터리입니다. 기본적으로 gg 테스트 접두사가 붙은 무작위 임시 디렉터리로 설정됩니다.
- `timeout-multiplier` - 모든 테스트 제한 시간에 제공되는 승수입니다. 기본값은 1.0.

GDK CLI 구성 파일 예제

다음 GDK CLI 구성 파일 예제를 참조하면 Greengrass 구성 요소 환경을 구성하는 데 도움이 됩니다.

Hello World(Python)

다음 GDK CLI 구성 파일은 Python 스크립트를 실행하는 Hello World 구성 요소를 지원합니다. 이 구성 파일은 zip 빌드 시스템을 사용하여 구성 요소의 Python 스크립트를 GDK CLI를 통해 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"
}

```

Hello World(Java)

다음 GDK CLI 구성 파일은 Java 애플리케이션을 실행하는 Hello World 구성 요소를 지원합니다. 이 구성 파일은 maven 빌드 시스템을 사용하여 구성 요소의 Java 소스 코드를 GDK CLI를 통해 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 카탈로그](#)의 여러 커뮤니티 구성 요소는 GDK CLI를 사용합니다. 이러한 구성 요소의 리포지토리에서 GDK CLI 구성 파일을 탐색할 수 있습니다.

커뮤니티 구성 요소의 GDK CLI 구성 파일 보기

1. 다음 명령을 실행하면 GDK CLI를 사용하는 커뮤니티 구성 요소가 나열됩니다.

```
gdk component list --repository
```

응답에는 GDK CLI를 사용하는 각 커뮤니티 구성 요소에 대한 GitHub 리포지토리 이름이 나열됩니다. 각 리포지토리는 awslabs 조직에 존재합니다.

```

[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 리포지토리를 엽니다. *community-component-name*을 이전 단계의 커뮤니티 구성 요소 이름으로 바꿉니다.

```
https://github.com/aws-labs/community-component-name
```

Greengrass 명령줄 인터페이스

Greengrass CLI(명령줄 인터페이스)를 사용하면 디바이스의 AWS IoT Greengrass 코어와 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.

[Greengrass CLI 구성 요소](#)(`aws.greengrass.Cli`)를 배포하여 코어 디바이스에 Greengrass CLI를 설치합니다.

Important

이 구성 요소는 프로덕션 환경이 아닌 개발 환경에서만 사용하는 것이 좋습니다. 이 구성 요소는 일반적으로 프로덕션 환경에서는 필요하지 않은 정보와 작업에 대한 액세스를 제공합니다. 필요한 코어 디바이스에만 이 구성 요소를 배포하여 최소 권한 원칙을 따르세요.

주제

- [Greengrass CLI 설치](#)
- [Greengrass CLI 명령](#)

Greengrass CLI 설치

다음 방법 중 하나를 사용하여 Greengrass CLI를 설치할 수 있습니다.

- 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 처음 설정할 때 `--deploy-dev-tools` 인수를 사용합니다. 이 인수를 적용하려면 `--provision true`를 지정해야 합니다.
- Greengrass CLI 구성 요소(`aws.greengrass.Cli`)를 디바이스에 배포합니다.

이 섹션에서는 Greengrass CLI 구성 요소 배포 단계를 설명합니다. 최초 설정 시 Greengrass CLI 설치에 대한 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#)의 내용을 참조하세요.

사전 조건

Greengrass CLI 구성 요소를 배포하려면 다음과 같은 요구 사항을 충족해야 합니다.

- AWS IoT Greengrass 코어 디바이스에 설치 및 구성된 코어 소프트웨어입니다. 자세한 내용은 [자습서: AWS IoT Greengrass V2 시작하기](#) 단원을 참조하십시오.
- AWS CLI 를 사용하여 Greengrass CLI를 배포하려면 설치하고 구성해야 합니다 AWS CLI. 자세한 내용은 AWS Command Line Interface 사용 설명서의 [AWS CLI구성](#)을 참조하세요.
- AWS IoT Greengrass 코어 소프트웨어와 상호 작용하려면 Greengrass CLI를 사용할 권한이 있어야 합니다. 다음 중 하나를 수행하여 Greengrass CLI 사용:
 - AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자를 사용합니다.
 - 루트 또는 관리 권한이 있는 사용자를 사용합니다. Linux 코어 디바이스에서 sudo를 사용하여 루트 권한을 얻을 수 있습니다.
 - 구성 요소를 배포할 때 AuthorizedPosixGroups 또는 AuthorizedWindowsGroups 구성 파라미터에 지정한 그룹에 있는 시스템 사용자를 사용합니다. 자세한 내용은 [Greengrass CLI 구성 요소 구성](#)을 참조하세요.

Greengrass CLI 구성 요소 배포

다음 단계를 완료하여 Greengrass CLI 구성 요소를 코어 디바이스에 배포:

Greengrass CLI 구성 요소 배포(콘솔)

1. [AWS IoT Greengrass 콘솔](#)에 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 대상에 있는 대상 이름 목록에서 배포할 Greengrass 그룹을 선택하고 다음을 선택합니다.
7. 구성 요소 선택 페이지에서 `aws.greengrass.Cli` 구성 요소가 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.

Greengrass CLI 구성 요소 배포(AWS CLI)

1. 디바이스에서 Greengrass CLI 구성 요소에 대한 배포 구성을 정의하는 `deployment.json` 파일을 생성합니다. 이 파일은 다음과 같습니다.

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.16.1",
      "configurationUpdate": {
        "merge": "{\\"AuthorizedPosixGroups\\":\\"<group1>,<group2>,...,<groupN>\\"",
        \\"AuthorizedWindowsGroups\\":\\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

- `target` 필드에서 *targetArn*을 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 객체에서 다음과 같이 값을 지정합니다.

`version`

Greengrass CLI 구성 요소의 버전.

`configurationUpdate.AuthorizedPosixGroups`

(선택 사항) 쉼표로 구분된 시스템 그룹 목록이 포함된 문자열. 이러한 시스템 그룹에 Greengrass CLI를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 상호 작용할 수 있는 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어 `group1,1002,group3`은 3개의 시스템 그룹(`group1`, `1002` 및 `group3`)에 Greengrass CLI 사용 권한을 부여합니다.

권한을 부여할 그룹을 지정하지 않으면 Greengrass CLI를 루트 사용자(`sudo`) 또는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자로 사용할 수 있습니다.

configurationUpdate.AuthorizedWindowsGroups

(선택 사항) 쉼표로 구분된 시스템 그룹 목록이 포함된 문자열. 이러한 시스템 그룹에 Greengrass CLI를 사용하여 AWS IoT Greengrass 코어 소프트웨어와 상호 작용할 수 있는 권한을 부여합니다. 그룹 이름 또는 그룹 ID를 지정할 수 있습니다. 예를 들어 group1,1002,group3은 3개의 시스템 그룹(group1, 1002 및 group3)에 Greengrass CLI 사용 권한을 부여합니다.

권한을 부여할 그룹을 지정하지 않으면 Greengrass CLI를 관리자 또는 AWS IoT Greengrass 코어 소프트웨어를 실행하는 시스템 사용자로 사용할 수 있습니다.

2. 다음 명령을 실행하여 디바이스에 Greengrass CLI 구성 요소를 배포합니다.

```
$ aws greengrassv2 create-deployment --cli-input-json file:///path/to/deployment.json
```

설치 도중 구성 요소는 디바이스의 */greengrass/v2/bin* 폴더에 있는 greengrass-cli에 심볼릭 링크를 추가하고 이 경로에서 Greengrass CLI를 실행합니다. 절대 경로 없이 Greengrass CLI를 실행하려면 */greengrass/v2/bin* 폴더를 PATH 변수에 추가합니다. 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.

greengrass-cli를 찾을 수 없는 경우 배포에서 Greengrass CLI를 설치하지 못했을 수 있습니다. 자세한 내용은 [문제 해결 AWS IoT Greengrass V2](#) 단원을 참조하십시오.

Greengrass CLI 명령

Greengrass CLI는 AWS IoT Greengrass 코어 디바이스와 로컬로 상호 작용할 수 있는 명령줄 인터페이스를 제공합니다. Greengrass CLI 명령은 다음 형식을 사용합니다.

```
$ greengrass-cli <command> <subcommand> [arguments]
```

기본적으로 */greengrass/v2/bin/* 폴더의 greengrass-cli 실행 파일은 */greengrass/v2* 폴더에서 실행되는 AWS IoT Greengrass 코어 소프트웨어의 버전과 상호 작용합니다. 이 위치에 없는 실행 파일을 직접 호출하거나 다른 위치의 AWS IoT Greengrass 코어 소프트웨어와 상호 작용하려면 다음 방법 중 하나를 사용하여 상호 작용하려는 AWS IoT Greengrass 코어 소프트웨어의 루트 경로를 명시적으로 지정해야 합니다.

- GGC_ROOT_PATH 환경 변수를 */greengrass/v2*로 설정합니다.
- 다음 예제와 같이 명령에 --ggcRootPath */greengrass/v2* 인수를 추가합니다.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

모든 명령에서 다음 인수를 사용할 수 있습니다.

- --help를 사용하면 특정 Greengrass CLI 명령에 관한 정보를 얻을 수 있습니다.
- --version을 사용하면 Greengrass CLI 버전에 관한 정보를 얻을 수 있습니다.

이 섹션에서는 Greengrass CLI 명령을 설명하고 이러한 명령의 예제를 제공합니다. 각 명령의 개요에는 인수와 용도가 표시됩니다. 선택적 인수는 대괄호에 표시됩니다.

가용 명령

- [구성 요소](#)
- [배포](#)
- [로그](#)
- [get-debug-password](#)

구성 요소

component 명령을 사용하여 코어 디바이스의 로컬 구성 요소와 상호 작용합니다.

하위 명령

- [details](#)
- [list](#)
- [재시작](#)
- [중단](#)

details

한 구성 요소의 버전, 상태 및 구성을 검색합니다.

시놉시스

```
greengrass-cli component details --name <component-name>
```

인수:

--name, -n. 구성 요소 이름.

출력:

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ 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 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ 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"}
```

재시작

구성 요소를 다시 시작합니다.

시놉시스

```
greengrass-cli component restart --names <component-name>,...
```

인수:

--names, -n. 구성 요소 이름. 하나 이상의 구성 요소 이름이 필요합니다. 추가 구성 요소 이름을 지정하고, 각 이름을 쉼표로 구분할 수 있습니다.

출력:

없음

종단

구성 요소 실행을 중지합니다.

시놉시스

```
greengrass-cli component stop --names <component-name>,...
```

인수:

--names, -n. 구성 요소 이름. 하나 이상의 구성 요소 이름이 필요합니다. 필요한 경우 추가 구성 요소 이름을 지정하고, 각 이름을 쉼표로 구분할 수 있습니다.

출력:

없음

배포

deployment 명령을 사용하여 코어 디바이스의 로컬 구성 요소와 상호 작용합니다.

로컬 배포 진행 상황을 모니터링하려면 status 하위 명령을 사용합니다. 콘솔을 사용하여 로컬 배포의 진행 상황을 모니터링할 수는 없습니다.

하위 명령

- [생성](#)
- [취소](#)
- [list](#)
- [status](#)

생성

지정된 구성 요소 레시피, 아티팩트 및 런타임 인수를 사용하여 로컬 배포를 생성하거나 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 포함된 폴더의 전체 경로.
- --artifactDir, -a. 배포에 포함할 아티팩트 파일이 포함된 폴더의 전체 경로. 아티팩트 폴더에는 다음 디렉터리 구조가 포함되어야 합니다.

```
/path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
```

- --update-config, -c. JSON 문자열 또는 JSON 파일로 제공되는 배포에 대한 구성 인수. JSON 문자열은 다음 형식이어야 합니다.

```
{ \
  "componentName": { \
    "MERGE": { "config-key": "config-value" }, \
    "RESET": [ "path/to/reset/" ] \
```

```
} \
}
```

MERGE 및 RESET은 대/소문자를 구분하며 대문자여야 합니다.

- --groupId, -g. 배포의 대상 사물 그룹.
- --merge, -m. 추가 또는 업데이트하려는 대상 구성 요소의 이름과 버전.
`<component>=<version>` 형식으로 구성 요소 정보를 제공해야 합니다. 각 추가 구성 요소에 대해 별도의 인수를 사용하여 지정합니다. 필요한 경우 --runWith 인수를 사용하여 구성 요소 실행에 대한 posixUser, posixGroup 및 windowsUser 정보를 제공합니다.
- --runWith. 일반 또는 Lambda 구성 요소 실행에 대한 posixUser, posixGroup 및 windowsUser 정보. `<component>:{posixUser|windowsUser}=<user>[:<posixGroup>]` 형식으로 이 정보를 제공해야 합니다. 예를 들어, **HelloWorld:posixUser=ggc_user:ggc_group** 또는 **HelloWorld:windowsUser=ggc_user**를 지정할 수 있습니다. 각 추가 옵션에 대해 별도의 인수를 사용하여 지정합니다.

자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 섹션을 참조하세요.

- --systemLimits. 코어 디바이스에서 일반 및 컨테이너화되지 않은 Lambda 구성 요소의 프로세스에 적용할 시스템 리소스 제한. 각 구성 요소의 프로세스에서 사용할 수 있는 최대 CPU 및 RAM 사용량을 구성할 수 있습니다. 직렬화된 JSON 객체 또는 JSON 파일의 파일 경로를 지정합니다. JSON 객체는 다음 형식이어야 합니다.

```
{ \
  "componentName": { \
    "cpus": cpuTimeLimit, \
    "memory": memoryLimitInKb \
  } \
}
```

각 구성 요소에 대해 다음 시스템 리소스 제한을 구성할 수 있습니다.

- cpus - 이 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 디바이스에서 이 값을 2로 설정하여 이 구성 요소 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 디바이스에서 이 값을 0.25로 설정하여 이 구성 요소 프로세스를 CPU의 25% 사용량으로 제한할 수 있습니다. 이 값을 CPU 코어 수보다 큰 수로 설정하면 AWS IoT Greengrass 코어 소프트웨어에서는 구성 요소의 CPU 사용량을 제한하지 않습니다.

- **memory** - 이 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 RAM(KB 단위)입니다.

자세한 내용은 [구성 요소에 대한 시스템 리소스 제한 구성](#) 섹션을 참조하세요.

이 기능은 Linux 코어 디바이스의 [Greengrass nucleus 구성 요소](#) 및 Greengrass CLI v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

- **--remove**. 로컬 배포에서 제거하려는 대상 구성 요소의 이름. 클라우드 배포에서 병합된 구성 요소를 제거하려면 다음 형식으로 대상 사물 그룹의 그룹 ID를 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**. 배포가 실패할 때 실시하는 작업 정의. 두 가지 작업을 지정할 수 있습니다.
 - ROLLBACK –
 - DO_NOTHING –

이 기능은 [Greengrass nucleus](#)의 v2.11.0 이상에서 사용할 수 있습니다.

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 식별자. 배포 ID는 create 명령의 출력에 반환됩니다.

출력

- 없음

list

마지막 10개의 로컬 배포 상태를 검색합니다.

시놉시스

```
greengrass-cli deployment list
```

인수:

- 없음

출력:

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다. 배포 상태에 따라 출력에 IN_PROGRESS, SUCCEEDED 또는 FAILED 중 하나의 상태 값이 표시됩니다.

```
$ 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. 배포의 ID.

출력:

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다. 배포 상태에 따라 출력에 IN_PROGRESS, SUCCEEDED 또는 FAILED 중 하나의 상태 값이 표시됩니다.

```
$ 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 분석합니다.

하위 명령

- [get](#)
- [list-keywords](#)
- [list-log-files](#)

get

Greengrass 로그 파일을 수집, 필터링 및 시각화합니다. 이 명령은 JSON 형식의 로그 파일만 지원합니다. nucleus 구성에서 [로깅 형식](#)을 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 디렉터리의 경로(예: `/greengrass/v2/logs`). `--syslog`와 함께 사용하지 않습니다. 각 추가 디렉터리에 대해 별도의 인수를 사용하여 지정합니다. `--log-dir` 또는 `--log-file` 중 하나 이상을 사용해야 합니다. 단일 명령에서 두 인수를 모두 사용할 수도 있습니다.
- `--log-file`, `-lf`. 사용하려는 로그 디렉터리의 경로. 각 추가 디렉터리에 대해 별도의 인수를 사용하여 지정합니다. `--log-dir` 또는 `--log-file` 중 하나 이상을 사용해야 합니다. 단일 명령에서 두 인수를 모두 사용할 수도 있습니다.
- `--follow`, `-fol`. 발생하는 로그 업데이트 표시. Greengrass CLI는 계속 실행되고 지정된 로그에서 읽습니다. 시간대를 지정하면 모든 시간대가 종료된 후 Greengrass CLI가 로그 모니터링을 중지합니다.
- `--filter`, `-f`. 필터로 사용할 키워드, 정규식 또는 키-값 페어. 이 값을 문자열, 정규식 또는 키-값 페어로 제공하세요. 각 추가 필터에 대해 별도의 인수를 사용하여 지정합니다.

평가 시 단일 인수에 지정된 여러 필터가 OR 연산자로 분리되고, 추가 인수에 지정된 필터가 AND 연산자와 결합됩니다. 예를 들어 명령에 `--filter "installed" --filter "name=alpha,name=beta"`가 포함된 경우 Greengrass CLI는 키워드 `installed`와 값이 `alpha` 또는 `beta`인 `name` 키가 모두 포함된 로그 메시지를 필터링하고 표시합니다.

- `--time-window`, `-t`. 로그 정보를 표시할 시간대. 정확한 타임스탬프와 상대 오프셋을 모두 사용할 수 있습니다. `<begin-time>`, `<end-time>` 형식으로 이 정보를 제공해야 합니다. 시작 시간 또는 종료 시간을 지정하지 않으면 해당 옵션의 값은 현재 시스템 날짜 및 시간으로 기본 설정됩니다. 각 추가 시간대에 대해 별도의 인수를 사용하여 지정합니다.

Greengrass CLI는 타임스탬프에 대해 다음 형식을 지원합니다.

- `yyyy-MM-DD`(예: `2020-06-30`). 이 형식을 사용할 때 시간은 기본적으로 `00:00:00`으로 설정됩니다.

yyyyMMDD(예: 20200630). 이 형식을 사용할 때 시간은 기본적으로 00:00:00으로 설정됩니다.

HH:mm:ss(예: 15:30:45). 이 형식을 사용할 때 날짜는 기본적으로 현재 시스템의 날짜로 설정됩니다.

HH:mm:ssSSS(예: 15:30:45). 이 형식을 사용할 때 날짜는 기본적으로 현재 시스템의 날짜로 설정됩니다.

YYYY-MM-DD'T'HH:mm:ss'Z'(예: 2020-06-30T15:30:45Z).

YYYY-MM-DD'T'HH:mm:ss(예: 2020-06-30T15:30:45).

yyyy-MM-dd'T'HH:mm:ss.SSS(예: 2020-06-30T15:30:45.250).

상대 오프셋은 현재 시스템 시간에서 시간 오프셋을 지정합니다. Greengrass CLI는 상대 오프셋에 대해 +|-<value>h|hr|hours[valuem|min|minutes][value]s|sec|seconds 형식을 지원합니다.

예를 들어 현재 시간에서 1시간~2시간 15분 사이 전의 시간대를 지정하는 인수는 --time-window -2h15min, -1hr입니다.

- --verbose. 로그 메시지의 모든 필드 표시. --syslog와 함께 사용하지 않습니다.
- --no-color, -nc. 색상 코딩 제거. 로그 메시지의 기본 색상 코딩은 굵은 빨간색 텍스트 사용입니다. ANSI 이스케이프 시퀀스를 사용하기 때문에 UNIX 계열 터미널만 지원합니다.
- --before, -b. 일치하는 로그 항목 앞에 표시할 줄 수. 기본값은 0.
- --after, -a. 일치하는 로그 항목 뒤에 표시할 줄 수. 기본값은 0.
- --syslog. RFC3164에서 정의한 syslog 프로토콜을 사용하여 모든 로그 파일 처리. --log-dir 및 --verbose와 함께 사용하지 않습니다. syslog 프로토콜에서 사용하는 형식: "<\$Priority>\$Timestamp \$Host \$Logger (\$Class): \$Message". 로그 파일을 지정하지 않으면 Greengrass CLI는 /var/log/messages, /var/log/syslog 또는 /var/log/system.log 위치에서 로그 메시지를 읽습니다.

AWS IoT Greengrass는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

- --max-log-queue-size, -m. 메모리에 할당할 최대 로그 항목 수. 이 옵션을 사용하여 메모리 사용량을 최적화합니다. 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
```

list-keywords

로그 파일을 필터링하는 데 사용할 수 있는 키워드 제안을 표시합니다.

시놉시스

```
greengrass-cli logs list-keywords [arguments]
```

인수:

없음

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 디렉터리의 경로.

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

get-debug-password 명령을 사용하여 [로컬 디버그 콘솔 구성 요소](#)(aws.greengrass.LocalDebugConsole)에 대해 무작위로 생성된 암호를 인쇄합니다. 암호는 생성되고 8시간 후에 만료됩니다.

시놉시스

```
greengrass-cli get-debug-password
```

인수:

없음

출력:

다음 예제는 이 명령을 실행할 때 생성되는 출력을 보여줍니다.

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEdp3M0Hdj8ou2w5de_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

GTF(Greengrass Testing Framework)는 고객의 관점에서 종단 간 자동화를 지원하는 구성 요소 컬렉션입니다. GTF는 [Cucumber](#)를 기능 드라이버로 사용합니다. 동일한 구성 요소를 AWS IoT Greengrass 사용하여 다양한 디바이스에서 소프트웨어 변경 사항을 검증합니다. 자세한 내용은 [Greengrass Testing Framework on Github](#)를 참조하세요.

GTF는 구성 요소의 BDD(동작 중심 개발)를 장려하기 위해 자동 테스트를 실행하는 데 사용되는 도구인 Cucumber를 사용하여 구현됩니다. Cucumber에서 이 시스템의 기능은 feature라는 특수 유형의 파일에 요약되어 있습니다. 각 기능은 사람이 읽을 수 있는 형식인 시나리오로 설명되고, 이는 자동 테스트로 변환할 수 있는 사양입니다. 각 시나리오는 Gherkin이라는 도메인별 언어를 사용하여 테스트 중인 이 시스템의 상호 작용 및 결과를 정의하는 일련의 단계로 설명됩니다. [Gherkin 단계](#)는 사양을 테스트 흐름에 하드 와이어링하는 단계 정의라는 메서드를 사용하여 프로그래밍 코드에 연결됩니다. GTF의 단계 정의는 Java로 구현됩니다.

주제

- [작동 방식](#)
- [Changelog](#)

- [Greengrass Testing Framework 구성 옵션](#)
- [자습서: Greengrass 테스트 프레임워크 및 Greengrass 개발 키트를 사용하여 엔드 투 엔드 테스트 실행](#)
- [자습서: 신뢰도 테스트 제품군의 신뢰도 테스트 사용](#)

작동 방식

AWS IoT Greengrass 는 GTF를 여러 Java 모듈로 구성된 독립 실행형 JAR로 배포합니다. 구성 요소의 종단 간 테스트에 GTF를 사용하려면 Java 프로젝트 내에서 테스트를 구현해야 합니다. Java 프로젝트에서 테스트용 JAR을 종속성으로 추가하면 GTF의 기존 기능을 사용하고 자체 사용자 지정 테스트 사례를 작성하면서 기능을 확장할 수 있습니다. 사용자 지정 테스트 사례를 실행하려면 Java 프로젝트를 빌드하고 [Greengrass Testing Framework 구성 옵션](#)에서 설명하는 구성 옵션을 사용하여 대상 JAR을 실행할 수 있습니다.

GTF 독립형 JAR

Greengrass는 Cloudfront를 [Maven](#) 리포지토리로 사용하여 다양한 버전의 GTF 독립형 JAR을 호스팅합니다. GTF 버전의 전체 목록은 [GTF 릴리스](#)를 참조하세요.

GTF 독립형 JAR에는 다음 모듈이 포함됩니다. 단, 이 모듈에만 국한되지 않습니다. 프로젝트에서 이러한 각 종속성을 개별적으로 선택하거나 [테스트 독립형 JAR 파일](#)에 모두 포함할 수 있습니다.

- `aws-greengrass-testing-resources`: 이 모듈은 테스트 과정에서 AWS 리소스의 수명 주기를 관리하기 위한 추상화를 제공합니다. GTF가 해당 AWS 리소스의 생성 및 제거를 처리할 수 있도록 `추ResourceSpec`상화를 사용하여 사용자 지정 리소스를 정의하는 데 사용할 수 있습니다.
- `aws-greengrass-testing-platform`: 이 모듈은 테스트 수명 주기 동안 테스트 중인 디바이스에 대한 플랫폼 수준 추상화를 제공합니다. 여기에는 플랫폼과 무관하게 OS와 상호 작용하는 데 사용되는 API가 포함되어 있으며 디바이스 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 구성됩니다.
- `aws-greengrass-testing-features`: 이 모듈은 재사용 가능한 일반 단계와 Greengrass 환경 내에서 테스트하는 데 사용되는 정의로 구성됩니다.

주제

- [Changelog](#)

- [Greengrass Testing Framework 구성 옵션](#)
- [자습서: Greengrass 테스트 프레임워크 및 Greengrass 개발 키트를 사용하여 엔드 투 엔드 테스트 실행](#)
- [자습서: 신뢰도 테스트 제품군의 신뢰도 테스트 사용](#)

Changelog

다음 표에서는 GTF의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다. 자세한 내용은 GitHub의 [GTF Releases 페이지](#)를 참조하세요.

버전	변경 사항
1.2.0	새로운 기능 • 테스트 도중 MQTT 및 인터넷 네트워크 연결을 구성하는 네트워크 관련 단계가 추가됩니다. • 시스템 지표 단계가 추가되어 디바이스 RAM 및 CPU 사용을 모니터링합니다. 버그 수정 및 개선 사항 • Greengrass CLI 로컬 배포 단계가 성공할 때까지 재시도됩니다. • 테스트에서 Greengrass nucleus를 종료하는 대신 정상적으로 중지합니다. • 사물 및 역할 별칭에 대한 AWS IoT 자격 증명을 검색할 수 있을 때까지 GTF가 자격 증명 엔드포인트를 폴링하는 개선 사항을 추가합니다. • 누락된 아티팩트 및 레시피 디렉터리가 수정됩니다. 또한 누락된 구성 요소 버전도 수정됩니다. • Docker 이미지가 없는 경우 Docker 이미지 정리 중에 GTF가 실패하는 문제를 해결합니다. • 구성 요소의 버전으로 CURRENT 키워드가 추가됩니다.
1.1.0	새로운 기능 • 구성으로 사용자 지정 구성 요소를 설치하는 기능이 추가됩니다. 이를 위해서는 사용자 지정 구성 요소에 대한 레시피가 필요합니다.

버전	변경 사항
	- 사용자 지정 구성으로 로컬 배포를 업데이트할 수 있는 기능이 추가됩니다. 버그 수정 및 개선 사항 - 로그 컨텍스트 GTF 버전 불일치 문제가 해결됩니다.
1.0.0	초기 버전입니다.

Greengrass Testing Framework 구성 옵션

JEG 구성 옵션

Greengrass Testing Framework(GTF)를 사용하면 엔드 투 엔드 테스트 프로세스를 시작하는 동안 특정 파라미터를 구성하여 테스트 흐름을 오케스트레이션할 수 있습니다. 이러한 구성 옵션을 GTF 독립 실행형 JAR에 대한 CLI 인수로 지정할 수 있습니다.

GTF 버전 1.1.0 이상에서는 다음 구성 옵션이 제공됩니다.

- `additional-plugins` - (선택 사항) 추가 Cucumber 플러그인
- `aws-region` - AWS 서비스에 대한 특정 리전 엔드포인트를 대상으로 합니다. 기본값은 AWS SDK에서 검색되는 내용입니다.
- `credentials-path` - 선택적 AWS 프로파일 자격 증명 경로입니다. 기본값은 호스트 환경에서 검색된 자격 증명입니다.
- `credentials-path-rotation` - AWS 자격 증명의 선택적 교체 기간입니다. 기본값은 15분 또는 PT15M입니다.
- `csr-path` - CSR에서 사용되어 디바이스 인증서가 생성되는 경로입니다.
- `device-mode` - 테스트 중인 대상 디바이스입니다. 기본값은 로컬 디바이스입니다.
- `env-stage` - Greengrass의 배포 환경을 대상으로 합니다. 기본값은 프로덕션입니다.
- `existing-device-cert-arn` - Greengrass의 디바이스 인증서로 사용하려는 기존 인증서의 arn입니다.
- `feature-path` - 추가 특성 파일이 포함된 파일 또는 디렉터리입니다. 기본적으로 추가 특성 파일은 사용되지 않습니다.
- `gg-cli-version` - Greengrass CLI의 버전을 재정의합니다. 기본값은 `ggc.version`에 있는 값입니다.

- `gg-component-bucket` - Greengrass 구성 요소가 보관되는 기존 Amazon S3 버킷의 이름입니다.
- `gg-component-overrides` - Greengrass 구성 요소 재정의 목록입니다.
- `gg-persist` - 테스트 실행 후 유지할 테스트 요소 목록입니다. 기본 동작은 아무것도 유지하지 않는 것입니다. 허용되는 값은 `aws.resources`, `installed.software` 및 `generated.files`입니다.
- `gg-runtime` - 테스트가 테스트 리소스와 상호 작용하는 방식에 영향을 미치는 값의 목록입니다. 이러한 값은 `gg.persist` 파라미터를 대체합니다. 기본값이 비어 있는 경우 설치된 Greengrass 런타임을 포함하여 모든 테스트 리소스가 테스트 사례로 관리된다고 가정합니다. 허용되는 값은 `aws.resources`, `installed.software` 및 `generated.files`입니다.
- `ggc-archive` - 보관된 Greengrass nucleus 구성 요소의 경로입니다.
- `ggc-install-root` - Greengrass nucleus 구성 요소를 설치하는 디렉터리입니다. 기본값은 `test.temp.path` 및 테스트 실행 폴더입니다.
- `ggc-log-level` - 테스트 실행에 대한 Greengrass nucleus 로그 수준을 설정합니다. 기본값은 'INFO'입니다.
- `ggc-tes-rolename` - AWS IoT Greengrass 코어가 AWS 서비스에 액세스하기 위해 맡을 IAM 역할입니다. 지정된 이름의 역할이 없는 경우 역할이 생성되고 기본 액세스 정책이 생성됩니다.
- `ggc-trusted-plugins` - Greengrass에 추가되어야 하는 신뢰할 수 있는 플러그인의 경로(호스트)를 쉼표로 구분한 목록입니다. DUT 자체의 경로를 제공하려면 경로에 'dut:' 접두사를 붙입니다.
- `ggc-user-name` - Greengrass nucleus의 `user:group posixUser` 값입니다. 기본값은 로그인한 현재 사용자 이름입니다.
- `ggc-version` - 실행 중인 Greengrass nucleus 구성 요소의 버전을 재정의합니다. 기본값은 `ggc.archive`에 있는 값입니다.
- `log-level` - 테스트 실행의 로그 수준입니다. 기본값은 'INFO'입니다.
- `parallel-config` - 배치 인덱스 및 배치 수 세트입니다(JSON 문자열). 배치 인덱스의 기본값은 0 이고 배치 수는 1입니다.
- `proxy-url` - 이 URL을 통해 트래픽을 라우팅하도록 모든 테스트를 구성합니다.
- `tags` - 특성 태그만 실행합니다. '&' 문자와 교차할 수 있습니다.
- `test-id-prefix` - AWS 리소스 이름 및 태그를 포함하여 모든 테스트별 리소스에 적용되는 일반 접두사입니다. 기본값은 'gg' 접두사입니다.
- `test-log-path` - 전체 테스트 실행의 결과를 포함하는 디렉터리입니다. 기본값은 'testResults'입니다.
- `test-results-json` - 결과로 나오는 Cucumber JSON 보고서가 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 true입니다.

- `test-results-log` - 콘솔 출력이 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 `false`입니다.
- `test-results-xml` - 결과로 나오는 JUnit XML 보고서가 디스크에 기록되는지 여부를 결정하는 플래그입니다. 기본값은 `true`입니다.
- `test-temp-path` - 로컬 테스트 아티팩트를 생성하는 디렉터리입니다. 기본적으로 `gg` 테스트 접두사가 붙은 무작위 임시 디렉터리로 설정됩니다.
- `timeout-multiplier` - 모든 테스트 제한 시간에 제공되는 승수입니다. 기본값은 1.0.

자습서: Greengrass 테스트 프레임워크 및 Greengrass 개발 키트를 사용하여 엔드 투 엔드 테스트 실행

AWS IoT Greengrass 테스트 프레임워크(GTF) 및 Greengrass 개발 키트(GDK)는 개발자에게 end-to-end 테스트를 실행할 수 있는 방법을 제공합니다. 이 자습서를 완료하여 구성 요소로 GDK 프로젝트를 초기화하고, 엔드 투 엔드 테스트 모듈로 GDK 프로젝트를 초기화하고, 사용자 지정 테스트 사례를 빌드할 수 있습니다. 사용자 지정 테스트 사례를 빌드하면 테스트를 실행할 수 있습니다.

이 자습서에서는 다음을 수행합니다.

1. 구성 요소로 GDK 프로젝트를 초기화합니다.
2. 엔드 투 엔드 테스트 모듈로 GDK 프로젝트를 초기화합니다.
3. 사용자 지정 테스트 사례를 빌드합니다.
4. 새 테스트 사례에 태그를 추가합니다.
5. 테스트 JAR을 빌드합니다.
6. 테스트를 실행합니다.

주제

- [사전 조건](#)
- [1단계: 구성 요소로 GDK 프로젝트 초기화](#)
- [2단계: 엔드 투 엔드 테스트 모듈로 GDK 프로젝트 초기화](#)
- [3단계: 사용자 지정 테스트 사례 빌드](#)
- [4단계: 새 테스트 사례에 태그 추가](#)
- [5단계: 테스트 JAR 빌드](#)
- [6단계: 테스트 실행](#)
- [예: 사용자 지정 테스트 사례 빌드](#)

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- GDK 버전 1.3.0 이상
- Java
- Maven
- Git

1단계: 구성 요소로 GDK 프로젝트 초기화

- GDK 프로젝트로 빈 폴더를 초기화합니다. 다음 명령을 실행하여 Python에서 구현된 HelloWorld 구성 요소를 다운로드합니다.

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

이 명령은 현재 디렉터리에 HelloWorld라는 이름의 디렉터를 생성합니다.

2단계: 엔드 투 엔드 테스트 모듈로 GDK 프로젝트 초기화

- GDK를 사용하면 기능 및 단계 구현으로 구성된 테스트 모듈 템플릿을 다운로드할 수 있습니다. 다음 명령을 실행하여 HelloWorld 디렉터를 열고 테스트 모듈을 사용하여 기존 GDK 프로젝트를 초기화합니다.

```
cd HelloWorld
gdk test-e2e init
```

이 명령은 HelloWorld 디렉터리 내에 gg-e2e-tests라는 이름의 디렉터를 생성합니다. 이 테스트 디렉터리는 Greengrass 테스트 독립 실행형 JAR에 종속된 [Maven](#) 프로젝트입니다.

3단계: 사용자 지정 테스트 사례 빌드

사용자 지정 테스트 사례 작성은 크게 테스트 시나리오가 포함된 기능 파일을 생성하고 단계 정의를 구현하는 두 단계로 구성됩니다. 사용자 지정 테스트 사례를 빌드하는 예제는 [예: 사용자 지정 테스트 사례 빌드](#) 섹션을 참조하세요. 다음 단계에 따라 사용자 지정 테스트 사례를 빌드합니다.

1. 테스트 시나리오를 사용하여 기능 파일 생성

기능은 일반적으로 테스트 중인 소프트웨어의 특정 기능을 설명합니다. Cucumber에서 각 기능은 제목, 자세한 설명, 시나리오라고 하는 특정 사례의 하나 이상의 예제를 포함하는 개별 기능 파일로 지정됩니다. 각 시나리오는 제목, 자세한 설명, 상호 작용, 예상 결과를 정의하는 일련의 단계로 구성됩니다. 시나리오는 “given”, “when”, “then” 키워드를 사용하여 구조화된 형식으로 작성됩니다.

2. 단계 정의 구현

단계 정의는 일반적인 언어로 작성된 [Gherkin 단계](#)를 프로그래밍 코드에 연결합니다. Cucumber가 시나리오에서 Gherkin 단계를 식별하면 실행할 일치하는 단계 정의를 찾습니다.

4단계: 새 테스트 사례에 태그 추가

- 기능 및 시나리오에 태그를 할당하여 테스트 프로세스를 구성할 수 있습니다. 태그를 사용하여 시나리오의 하위 세트를 분류하며 실행할 후크를 조건부로 선택할 수도 있습니다. 특성과 시나리오에는 공백으로 구분된 여러 태그가 있을 수 있습니다.

이 예제에서는 HelloWorld 구성 요소를 사용합니다.

기능 파일에서 @Sample 태그 옆에 @HelloWorld라는 이름의 새 태그를 추가합니다.

```
@Sample @HelloWorld
Scenario: As a developer, I can create a component and deploy it on my device
....
```

5단계: 테스트 JAR 빌드

- 구성 요소를 빌드합니다. 테스트 모듈을 빌드하기 전에 구성 요소를 빌드해야 합니다.

```
gdk component build
```

- 다음 명령을 사용하여 테스트 모듈을 빌드합니다. 이 명령은 greengrass-build 폴더에 테스트 JAR을 빌드합니다.

```
gdk test-e2e build
```

6단계: 테스트 실행

사용자 지정 테스트 사례를 실행하면 GTF는 테스트 중에 생성된 리소스를 관리하면서 테스트 수명 주기를 자동화합니다. 먼저 테스트 대상 디바이스(DUT)를 AWS IoT 사물로 프로비저닝하고 Greengrass 코어 소프트웨어를 설치합니다. 그러면 해당 경로에 지정된 레시피를 사용하여 HelloWorld라는 새 구성 요소가 생성됩니다. 그런 다음에 HelloWorld 구성 요소가 Greengrass 사물 배포를 통해 코어 디바이스에 배포됩니다. 그러면 배포되었는지 확인됩니다. 배포되면 3분 이내에 배포 상태가 COMPLETED로 변경됩니다.

1. 프로젝트 디렉터리의 gdk-config.json 파일로 이동하여 HelloWorld 태그로 테스트를 대상으로 지정합니다. 다음 명령을 사용하여 test-e2e 키를 업데이트합니다.

```
"test-e2e":{
  "gtf_options" : {
    "tags":"HelloWorld"
  }
}
```

2. 테스트를 실행하기 전에 호스트 디바이스에 AWS 자격 증명을 제공해야 합니다. GTF는 이러한 자격 증명을 사용하여 테스트 프로세스 중에 AWS 리소스를 관리합니다. 테스트에 포함된 필수 작업을 자동화할 수 있는 권한이 제공한 역할에 있는지 확인합니다.

다음 명령을 실행하여 AWS 자격 증명을 제공합니다.

- 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 최신 버전의 Greengrass nucleus를 다운로드하고 이를 사용하여 테스트를 실행합니다. 또한 이 명령에서는 HelloWorld 태그가 있는 시나리오만 대상이 되고 해당 시나리오에 대한 보고서가 생성됩니다. 이 테스트 중에 생성된 AWS 리소스는 테스트가 끝날 때 삭제됩니다.

예: 사용자 지정 테스트 사례 빌드

Example

GDK 프로젝트에서 다운로드한 테스트 모듈은 샘플 기능과 단계 구현 파일로 구성되어 있습니다.

다음 예제에서는 Greengrass 소프트웨어의 사물 배포 기능을 테스트하기 위한 기능 파일을 생성합니다. Greengrass AWS 클라우드를 통해 구성 요소를 배포하는 시나리오를 사용하여 이 특성의 기능을 부분적으로 테스트합니다. 이는 해당 사용 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다.

1. 기능 파일 생성

현재 디렉터리의 gg-e2e-tests/src/main/resources/greengrass/features 폴더로 이동합니다. 다음 예제처럼 보이는 component.feature 샘플을 찾을 수 있습니다.

이 기능 파일에서 Greengrass 소프트웨어의 사물 배포 기능을 테스트할 수 있습니다. Greengrass 클라우드를 통해 구성 요소를 배포하는 시나리오를 사용하여 이 특성의 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 And I call my custom step이라는 단계를 제외하고 다음 모든 단계의 단계 정의가 포함되어 있습니다.

2. 단계 정의 구현

GTF 독립 실행형 JAR에는 And I call my custom step이라는 한 단계를 제외하고 모든 단계의 단계 정의가 포함되어 있습니다. 이 단계는 테스트 모듈에서 구현할 수 있습니다.

테스트 파일의 소스 코드로 이동합니다. 다음 명령으로 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 ");
}
```

자습서: 신뢰도 테스트 제품군의 신뢰도 테스트 사용

AWS IoT Greengrass 테스트 프레임워크(GTF) 및 Greengrass 개발 키트(GDK)는 개발자에게 end-to-end 테스트를 실행할 수 있는 방법을 제공합니다. 이 자습서를 완료하면 구성 요소로 GDK 프로젝트를 초기화하고, 엔드 투 엔드 테스트 모듈로 GDK 프로젝트를 초기화하고, 신뢰도 테스트 제품군의 신뢰도 테스트를 사용할 수 있습니다. 사용자 지정 테스트 사례를 빌드하면 테스트를 실행할 수 있습니다.

신뢰도 테스트는 기본 구성 요소 동작이 검증되는 Greengrass에서 제공되는 일반 테스트입니다. 이러한 테스트는 더 구체적인 구성 요소 요구 사항에 알맞게 수정하거나 확장할 수 있습니다.

이 자습서에서는 HelloWorld 구성 요소를 사용하겠습니다. 다른 구성 요소를 사용한다면 해당 구성 요소로 HelloWorld 구성 요소를 바꿉니다.

이 자습서에서는 다음을 수행합니다.

1. 구성 요소로 GDK 프로젝트를 초기화합니다.
2. 엔드 투 엔드 테스트 모듈로 GDK 프로젝트를 초기화합니다.
3. 신뢰도 테스트 제품군의 테스트를 사용합니다.
4. 새 테스트 사례에 태그를 추가합니다.
5. 테스트 JAR을 빌드합니다.

6. 테스트를 실행합니다.

주제

- [사전 조건](#)
- [1단계: 구성 요소로 GDK 프로젝트 초기화](#)
- [2단계: 엔드 투 엔드 테스트 모듈로 GDK 프로젝트 초기화](#)
- [3단계: 신뢰도 테스트 제품군의 테스트를 사용합니다.](#)
- [4단계: 새 테스트 사례에 태그 추가](#)
- [5단계: 테스트 JAR 빌드](#)
- [6단계: 테스트 실행](#)
- [예제: 신뢰도 테스트 사용](#)

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- GDK 버전 1.6.0 이상
- Java
- Maven
- Git

1단계: 구성 요소로 GDK 프로젝트 초기화

- GDK 프로젝트로 빈 폴더를 초기화합니다. 다음 명령을 실행하여 Python에서 구현된 HelloWorld 구성 요소를 다운로드합니다.

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

이 명령은 현재 디렉터리에 HelloWorld라는 이름의 디렉터를 생성합니다.

2단계: 엔드 투 엔드 테스트 모듈로 GDK 프로젝트 초기화

- 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 HelloWorld 디렉터리 내에 gg-e2e-tests라는 이름의 디렉터를 생성합니다. 이 테스트 디렉터리는 Greengrass 테스트 독립 실행형 JAR에 종속된 [Maven](#) 프로젝트입니다.

3단계: 신뢰도 테스트 제품군의 테스트를 사용합니다.

신뢰도 테스트 사례 작성은 제공된 특성 파일 사용과 시나리오 수정(필요한 경우)으로 구성됩니다. 이러한 신뢰도 테스트 사용 예제는 [예: 사용자 지정 테스트 사례 빌드](#) 섹션을 참조하세요. 다음 단계를 사용하여 신뢰도 테스트를 사용합니다.

- 제공된 특성 파일을 사용합니다.

현재 디렉터리에서 gg-e2e-tests/src/main/resources/greengrass/features 폴더로 이동합니다. 신뢰도 테스트를 사용할 샘플 confidenceTest.feature 파일을 엽니다.

4단계: 새 테스트 사례에 태그 추가

- 기능 및 시나리오에 태그를 할당하여 테스트 프로세스를 구성할 수 있습니다. 태그를 사용하여 시나리오의 하위 세트를 분류하며 실행할 후크를 조건부로 선택할 수도 있습니다. 특성과 시나리오에는 공백으로 구분된 여러 태그가 있을 수 있습니다.

이 예제에서는 HelloWorld 구성 요소를 사용합니다.

각 시나리오에는 @ConfidenceTest라는 태그가 지정됩니다. 테스트 제품군의 하위 세트만 실행하려면 태그를 변경하거나 추가합니다. 각 테스트 시나리오는 각 신뢰도 테스트의 상단에 설명되어 있습니다. 시나리오는 각 테스트 사례의 상호 작용 및 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다. 원하는 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 GTF에서는 테스트의 수명 주기가 테스트 동안 생성된 리소스 관리와 함께 자동화됩니다. 먼저 테스트 대상 디바이스(DUT)를 AWS IoT 사물로 프로비저닝하고 Greengrass 코어 소프트웨어를 설치합니다. 그러면 해당 경로에 지정된 레시피를 사용하여 HelloWorld라는 새 구성 요소가 생성됩니다. 그런 다음에 HelloWorld 구성 요소가 Greengrass 사물 배포를 통해 코어 디바이스에 배포됩니다. 그러면 배포되었는지 확인됩니다. 배포되면 3분 이내에 배포 상태가 COMPLETED로 변경됩니다.

1. 프로젝트 디렉터리에서 ConfidenceTest 태그 또는 4단계에서 지정한 태그가 있는 테스트 대상인 gdk-config.json 파일로 이동합니다. 다음 명령을 사용하여 test-e2e 키를 업데이트합니다.

```
"test-e2e":{
  "gtf_options" : {
    "tags":"ConfidenceTest"
  }
}
```

2. 테스트를 실행하기 전에 호스트 디바이스에 AWS 자격 증명을 제공해야 합니다. GTF는 이러한 자격 증명을 사용하여 테스트 프로세스 중에 AWS 리소스를 관리합니다. 테스트에 포함된 필수 작업을 자동화할 수 있는 권한이 제공한 역할에 있는지 확인합니다.

다음 명령을 실행하여 AWS 자격 증명을 제공합니다.

- 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"
```

- 다음 명령을 사용하여 테스트를 실행합니다.

```
gdk test-e2e run
```

이 명령은 greengrass-build 폴더에 최신 버전의 Greengrass nucleus를 다운로드하고 이를 사용하여 테스트를 실행합니다. 또한 이 명령에서는 ConfidenceTest 태그가 있는 시나리오만 대상이 되고 해당 시나리오에 대한 보고서가 생성됩니다. 이 테스트 중에 생성된 AWS 리소스는 테스트가 끝날 때 삭제됩니다.

예제: 신뢰도 테스트 사용

Example

GDK 프로젝트에서 다운로드된 테스트 모듈은 제공된 특성 파일로 구성됩니다.

다음 예제에서는 Greengrass 소프트웨어의 사물 배포 특성 테스트에 특성 파일을 사용합니다. Greengrass AWS 클라우드를 통해 구성 요소를 배포하는 시나리오를 사용하여 이 특성의 기능을 부분적으로 테스트합니다. 이는 해당 사용 사례의 상호 작용과 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다.

- 제공된 특성 파일을 사용합니다.

현재 디렉터리에서 `gg-e2e-tests/src/main/resources/greengrass/features` 폴더로 이동합니다. 다음 예제처럼 보이는 `confidenceTest.feature` 샘플을 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 신뢰도 테스트의 상단에 설명되어 있습니다. 시나리오는 각 테스트 사례의 상호 작용 및 예상 결과를 이해하는 데 도움이 되는 일련의 단계입니다. 원하는 단계를 추가하거나 기존 단계를 수정하여 이러한 테스트를 확장할 수 있습니다. 각 시나리오에는 이러한 조정을 수행하는 데 도움이 되는 설명이 포함되어 있습니다.

AWS IoT Greengrass 구성 요소 개발

Greengrass 코어 디바이스에서 구성 요소를 개발하고 테스트할 수 있습니다. 따라서 AWS 클라우드와 상호 작용하지 않아도 AWS IoT Greengrass 소프트웨어를 생성하고 반복할 수 있습니다. 구성 요소의 버전을 마치면 클라우드의 AWS IoT Greengrass에 업로드할 수 있으므로 자신과 소속 팀이 구성 요소를 플릿의 다른 디바이스에 배포할 수 있습니다. 구성 요소를 배포하는 방법에 대한 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하세요.

모든 구성 요소는 레시피와 아티팩트 로 구성되어 있습니다.

- 레시피

모든 구성 요소에는 메타데이터가 정의되는 레시피 파일이 있습니다. 레시피에서는 구성 요소의 구성 파라미터, 구성 요소 종속성, 수명 주기 및 플랫폼 호환성도 지정됩니다. 구성 요소 수명 주기에서는 구성 요소 설치, 실행 및 종료 명령이 정의됩니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 섹션을 참조하세요.

레시피를 [JSON](#) 또는 [YAML](#) 형식으로 정의할 수 있습니다.

- 아티팩트

구성 요소의 구성 요소 바이너리인 아티팩트는 수량 제한이 없습니다. 아티팩트에는 구성 요소에서 사용되는 스크립트, 컴파일된 코드, 정적 리소스 및 기타 파일이 포함될 수 있습니다. 구성 요소에서는 구성 요소 종속성의 아티팩트도 사용될 수도 있습니다.

AWS IoT Greengrass에서는 애플리케이션에서 사용하고 디바이스에 배포할 수 있는 사전 빌드된 구성 요소가 제공됩니다. 예를 들어 스트림 관리자 구성 요소를 사용하여 다양한 AWS 서비스에 데이터를 업로드하거나 CloudWatch 지표 구성 요소를 사용하여 사용자 지정 지표를 Amazon CloudWatch에 게시할 수 있습니다. 자세한 내용은 [AWS 제공 구성 요소](#) 섹션을 참조하세요.

AWS IoT Greengrass에서는 Greengrass 소프트웨어 카탈로그라는 Greengrass 구성 요소의 인덱스가 큐레이팅됩니다. 이 카탈로그에서는 Greengrass 커뮤니티에서 개발하는 Greengrass 구성 요소가 추적됩니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다. 자세한 내용은 [커뮤니티 구성 요소](#) 섹션을 참조하세요.

AWS IoT Greengrass 코어 소프트웨어에서는 코어 디바이스에서 구성하는 시스템 사용자 및 그룹(예: `ggc_user` 및 `ggc_group`)으로 구성 요소가 실행됩니다. 즉, 구성 요소에 해당 시스템 사용자의 권한이 있습니다. 홈 디렉터리가 없는 시스템 사용자를 사용하는 경우에는 홈 디렉터리가 사용되는 실행 명령 또는 코드가 구성 요소에서 사용될 수 없습니다. 즉, `pip install some-library --user` 명령을 사용하여 예컨대 Python 패키지를 설치할 수 없습니다. [시작하기 자습서](#)에 따라 코어 디바이스를 설정했다면 시스템 사용자에게 홈 디렉터리가 없습니다. 구성 요소가 실행되는 사용자 및 그룹을 구성하는 방법에 대한 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 단원을 참조하세요.

Note

AWS IoT Greengrass에서는 구성 요소에 의미 체계 버전이 사용됩니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 1.0.0 버전은 구성 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

주제

- [구성 요소 수명 주기](#)
- [구성 요소 유형](#)
- [AWS IoT Greengrass 구성 요소 생성](#)
- [로컬 배포를 사용한 AWS IoT Greengrass 구성 요소 테스트](#)
- [코어 디바이스에 배포할 구성 요소를 게시하세요.](#)
- [AWS 서비스와 상호 작용](#)
- [Docker 컨테이너 실행](#)
- [AWS IoT Greengrass 구성 요소 레시피 참조](#)
- [구성 요소 환경 변수 참조](#)

구성 요소 수명 주기

구성 요소 수명 주기에서는 AWS IoT Greengrass 코어 소프트웨어의 구성 요소 설치 및 실행에 사용되는 스테이지가 정의됩니다. 각 스테이지에서는 구성 요소의 작동 방식이 지정되는 스크립트 및 기타 정보가 정의됩니다. 예를 들어 구성 요소를 설치할 때 AWS IoT Greengrass 코어 소프트웨어에서는 해당 구성 요소에 대한 `install` 수명 주기 스크립트가 실행됩니다. 코어 디바이스의 구성 요소에는 다음 수명 주기 상태가 있습니다.

- NEW - 코어 디바이스에서 구성 요소의 레시피와 아티팩트가 로드되지만, 구성 요소는 설치되지 않습니다. 구성 요소가 이 상태로 전환되면 [스크립트 설치](#)가 실행됩니다.
- INSTALLED - 코어 디바이스에서 구성 요소가 설치됩니다. 구성 요소가 [스크립트 설치](#)가 실행된 후 이 상태로 전환됩니다.
- STARTING - 코어 디바이스에서 구성 요소가 시작됩니다. [시작 스크립트](#)가 실행되면 구성 요소가 이 상태로 전환됩니다. 시작되면 구성 요소가 RUNNING 상태로 전환됩니다.
- RUNNING - 코어 디바이스에서 구성 요소가 실행됩니다. [스크립트 실행](#)이 실행되거나 시작 스크립트의 활성 백그라운드 프로세스가 있으면 구성 요소가 이 상태로 전환됩니다.
- FINISHED - 구성 요소가 실행되고 실행이 완료되었습니다.
- STOPPING - 구성 요소가 중지되고 있습니다. [스크립트 종료](#)가 실행되면 구성 요소가 이 상태로 전환됩니다.
- ERRORED - 구성 요소에서 오류가 발생했습니다. 구성 요소가 이 상태로 전환되면 [스크립트 복구](#)가 실행됩니다. 그런 다음에 구성 요소가 다시 시작되어 정상 사용으로 돌아가기가 시도됩니다. 구성 요소가 실행되지 않고 ERRORED 상태로 세 번 전환되면 구성 요소는 BROKEN으로 됩니다.

- **BROKEN** - 오류가 여러 번 발생하여 구성 요소를 복구할 수 없습니다. 오류를 해결하려면 구성 요소를 다시 배포해야 합니다.

구성 요소 유형

구성 요소 유형에서는 AWS IoT Greengrass 코어 소프트웨어에서 구성 요소가 실행되는 방식이 지정됩니다. 구성 요소에는 다음 유형이 있을 수 있습니다.

- **Nucleus**(aws.greengrass.nucleus)

Greengrass nucleus는 AWS IoT Greengrass 코어 소프트웨어의 최소 기능이 제공되는 구성 요소입니다. 자세한 내용은 [Greengrass nucleus](#) 섹션을 참조하세요.

- **플러그인**(aws.greengrass.plugin)

Greengrass nucleus에서는 플러그인 구성 요소가 nucleus와 동일한 Java 가상 머신(JVM)에서 실행됩니다. 코어 디바이스에서 플러그인 구성 요소의 버전을 변경하면 nucleus가 다시 시작됩니다. 플러그인 구성 요소를 설치하고 실행하려면 시스템 서비스로 실행되도록 Greengrass nucleus를 구성해야 합니다. 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 섹션을 참조하세요.

AWS에서 제공하는 여러 구성 요소는 Greengrass nucleus와 직접 인터페이스로 연결될 수 있는 플러그인 구성 요소입니다. 플러그인 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 섹션을 참조하세요.

- **일반**:(aws.greengrass.generic)

구성 요소에서 수명 주기가 정의되는 경우 Greengrass nucleus에서는 일반 구성 요소의 수명 주기 스크립트가 실행됩니다.

이 유형은 사용자 지정 구성 요소의 기본 유형입니다.

- **Lambda**(aws.greengrass.lambda)

Greengrass nucleus에서는 Lambda 함수 실행에 [Lambda 런처 구성 요소](#)가 사용됩니다.

Lambda 함수에서 구성 요소를 생성할 때 구성 요소에 이 유형이 있습니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 섹션을 참조하세요.

Note

레시피에서는 구성 요소 유형을 지정하지 않는 것이 좋습니다. 구성 요소를 생성할 때 자동으로 AWS IoT Greengrass에서 유형이 설정됩니다.

AWS IoT Greengrass 구성 요소 생성

로컬 개발 컴퓨터 또는 Greengrass 코어 디바이스에서 사용자 지정 AWS IoT Greengrass 구성 요소를 개발할 수 있습니다. 사전 정의된 구성 요소 템플릿 및 [커뮤니티 구성 요소](#)로부터 구성 요소를 생성, 빌드 및 게시하는 데 도움이 되는 [AWS IoT Greengrass 개발 키트 명령줄 인터페이스\(GDK CLI\)](#)가 AWS IoT Greengrass에서 제공됩니다. 기본 제공 셸 명령을 실행하여 구성 요소를 생성, 빌드 및 게시할 수도 있습니다. 다음 옵션 중에서 선택하여 사용자 지정 Greengrass 구성 요소를 생성합니다.

- Greengrass 개발 키트 CLI 사용

GDK CLI를 사용하여 로컬 개발 컴퓨터에서 구성 요소를 개발합니다. GDK CLI에서는 구성 요소 소스 코드가 빌드되어 AWS IoT Greengrass 서비스에 프라이빗 구성 요소로 게시할 수 있는 레시피 및 아티팩트에 패키징됩니다. 구성 요소를 게시할 때 구성 요소의 버전 및 아티팩트 URI가 자동으로 업데이트되도록 GDK CLI를 구성할 수 있으므로 매번 레시피를 업데이트할 필요가 없습니다. GDK CLI를 사용하여 구성 요소를 개발하려면 [Greengrass 소프트웨어 카탈로그](#)의 템플릿 또는 커뮤니티 구성 요소에서 시작할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 개발 키트 명령줄 인터페이스](#) 섹션을 참조하세요.

- 기본 제공 셸 명령 실행

기본 제공 셸 명령을 실행하여 로컬 개발 컴퓨터 또는 Greengrass 코어 디바이스에서 구성 요소를 개발할 수 있습니다. 셸 명령을 사용하여 구성 요소 소스 코드를 아티팩트에 복사하거나 빌드합니다. 새 구성 요소 버전을 생성할 때마다 새 구성 요소 버전으로 레시피를 생성하거나 업데이트해야 합니다. AWS IoT Greengrass 서비스에 구성 요소를 게시할 때 레시피의 각 구성 요소 아티팩트에 대해 URI를 업데이트해야 합니다.

주제

- [구성 요소 생성\(GDK CLI\)](#)
- [구성 요소 생성\(셸 명령\)](#)

구성 요소 생성(GDK CLI)

이 단원의 지침에 따라 GDK CLI를 사용하여 구성 요소를 생성하고 빌드합니다.

Greengrass 구성 요소(GDK CLI)를 개발하려면

1. 아직 설치하지 않았다면 개발 컴퓨터에 GDK CLI를 설치합니다. 자세한 내용은 [AWS IoT Greengrass 개발 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. 템플릿 또는 커뮤니티 구성 요소는 GDK CLI에서 다운로드되므로 작동 예제부터 시작할 수 있습니다. [구성 요소 목록](#) 명령을 사용하여 사용 가능한 템플릿 또는 커뮤니티 구성 요소 목록을 검색합니다.
 - 구성 요소 템플릿을 나열하려면 다음 명령을 실행합니다. 응답의 각 줄에 템플릿의 이름과 프로그래밍 언어가 포함되어 있습니다.

```
gdk component list --template
```

- 커뮤니티 구성 요소를 나열하려면 다음 명령을 실행합니다.

```
gdk component list --repository
```

4. GDK CLI에서 템플릿 또는 커뮤니티 구성 요소가 다운로드되는 구성 요소 폴더를 생성하여 변경합니다. 구성 요소의 이름 또는 이 구성 요소 폴더를 식별하는 데 도움이 되는 다른 이름으로 *HelloWorld*를 바꿉니다.

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. [component init](#) 명령을 사용합니다.

- 템플릿으로부터 구성 요소 폴더를 생성하려면 다음 명령을 실행합니다. 템플릿의 이름으로 *HelloWorld*를 바꾸고, 프로그래밍 언어의 이름으로 *python*을 바꿉니다.

```
gdk component init --template HelloWorld --language python
```

- 커뮤니티 구성 요소로부터 구성 요소 폴더를 생성하려면 다음 명령을 실행합니다. 커뮤니티 구성 요소의 이름으로 *ComponentName*을 바꿉니다.

```
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 `--name` 인수를 지정하여 GDK CLI가 템플릿 또는 커뮤니티 구성 요소를 다운로드하는 폴더를 지정할 수 있습니다. 이 인수를 사용하는 경우 존재하지 않는 폴더를 지정해야 합니다. GDK CLI가 폴더를 생성합니다. 이 인수를 지정하지 않으면 GDK CLI에서 현재 폴더를 사용하고, 이 폴더는 비어 있어야 합니다.

6. GDK CLI에서는 `gdk-config.json`라는 [GDK CLI 구성 파일](#)의 구성 요소 읽기, 빌드 및 게시가 진행됩니다. 이 구성 파일은 구성 요소 폴더의 루트에 있습니다. 이전 단계에서 이 파일이 자동으로 생성됩니다. 이 단계에서는 구성 요소에 대한 정보로 `gdk-config.json`을 업데이트합니다. 다음을 수행합니다.
- 텍스트 편집기에서 `gdk-config.json`을 엽니다.
 - (선택 사항) 구성 요소의 이름을 변경합니다. 구성 요소 이름은 `component` 객체에 있는 키입니다.
 - 구성 요소의 작성자를 변경합니다.
 - (선택 사항) 구성 요소의 버전을 변경합니다. 다음 중 하나를 지정하세요.
 - `NEXT_PATCH` – 이 옵션을 선택하면 구성 요소를 게시할 때 GDK CLI에서 버전이 설정됩니다. GDK CLI에서는 AWS IoT Greengrass 서비스 쿼리를 통해 구성 요소의 최신 게시 버전이 식별됩니다. 그런 다음에 버전이 해당 버전 이후의 다음 패치 버전으로 설정됩니다. 이전에 구성 요소를 게시하지 않았으면 GDK CLI에서는 `1.0.0` 버전이 사용됩니다.

이 옵션을 선택하면 [Greengrass CLI](#)를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 실행하는 로컬 개발 컴퓨터에 구성 요소를 로컬로 배포하고 테스트할 수 있습니다. 로컬 배포를 활성화하려면 대신 의미 체계 버전을 지정해야 합니다.

 - 의미 체계 버전(예: `1.0.0`). 의미 체계 버전에서는 `major.minor.patch` 번호 매기기 시스템이 사용됩니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.
- 구성 요소를 배포하고 테스트하려는 Greengrass 코어 디바이스에서 구성 요소를 개발하는 경우 이 옵션을 선택합니다. [Greengrass CLI](#)를 사용하여 로컬 배포를 생성하려면 특정 버전으로 구성 요소를 빌드해야 합니다.
- (선택 사항) 구성 요소의 빌드 구성을 변경합니다. 빌드 구성에서는 GDK CLI에서 구성 요소의 소스가 아티팩트에 빌드되는 방식이 정의됩니다. `build_system`의 경우 다음 옵션 중에서 선택합니다.
 - `zip` - 구성 요소의 폴더를 ZIP 파일로 패키징하여 구성 요소의 유일한 아티팩트로 정의합니다. 다음 유형의 구성 요소에 대해 이 옵션을 선택합니다.
 - Python 또는 JavaScript와 같이 해석된 프로그래밍 언어가 사용되는 구성 요소.
 - 기계 학습 모델 또는 기타 리소스와 같이 코드 이외의 파일이 패키징되는 구성 요소.

GDK CLI에서는 구성 요소의 폴더를 이름이 동일한 zip 파일로 압축됩니다. 예를 들어, 구성 요소 폴더의 이름이 `HelloWorld`라면 GDK CLI에서는 `HelloWorld.zip`이라는 파일이 생성됩니다.

Note

Windows 디바이스에서 GDK CLI 버전 1.0.0을 사용하는 경우 구성 요소 폴더 및 zip 파일 이름에는 소문자만 포함되어야 합니다.

GDK CLI에서는 구성 요소의 폴더가 zip 파일로 압축하면 다음 파일을 건너뛵니다.

- gdk-config.json 파일
- 레시피 파일(recipe.json 또는 recipe.yaml)
- 빌드 폴더(예: greengrass-build)
- `maven -mvn clean package` 명령을 실행하여 구성 요소의 소스를 아티팩트에 빌드합니다. Java 구성 요소와 같이 [Maven](#)을 사용하는 구성 요소의 경우 이 옵션을 선택합니다.

Windows 디바이스에서 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

- `gradle - gradle build` 명령을 실행하여 구성 요소의 소스를 아티팩트에 빌드합니다. [Gradle](#)을 사용하는 구성 요소의 경우 이 옵션을 선택합니다. 이 기능은 GDK CLI v1.1.0 이상에서 사용할 수 있습니다.

gradle 빌드 시스템에서는 빌드 파일로 Kotlin DSL이 지원됩니다. 이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `gradlew - gradlew` 명령을 실행하여 구성 요소의 소스를 아티팩트에 빌드합니다. [Gradle Wrapper](#)를 사용하는 구성 요소의 경우 이 옵션을 선택합니다.

이 기능은 GDK CLI v1.2.0 이상에서 사용할 수 있습니다.

- `custom` - 사용자 지정 명령을 실행하여 구성 요소의 소스를 레시피 및 아티팩트에 빌드합니다. `custom_build_command` 파라미터에서 사용자 지정 명령을 지정합니다.
- build_system에 custom을 지정하는 경우 build 객체에 custom_build_command를 추가합니다. custom_build_command에서 각 문자열은 명령의 단어인 단일 문자열 또는 문자열 목록을 지정합니다. 예를 들어, C++ 구성 요소에 대한 사용자 지정 빌드 명령을 실행하려면 `["cmake", "--build", "build", "--config", "Release"]`를 지정할 수 있습니다.
 - GDK CLI v1.1.0 이상을 사용하는 경우 --bucket 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 수 있습니다. 이 인수를 지정하지 않으면 GDK CLI가 이름이 `bucket-region-accountId`인 S3 버킷에 업로드합니다. 여기서 `bucket`과

region은 gdk-config.json에서 지정하는 값이고, **accountId**는 AWS 계정 ID입니다. GDK CLI는 버킷이 없는 경우 버킷을 생성합니다.

구성 요소의 게시 구성을 변경합니다. 다음을 수행합니다.

- i. 구성 요소 아티팩트를 호스팅하는 데 사용할 S3 버킷의 이름을 지정합니다.
- ii. GDK CLI에서 구성 요소가 게시되는 AWS 리전을 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. recipe.yaml 또는 recipe.json이라는 구성 요소 레시피 파일을 업데이트합니다. 다음을 수행합니다.
 - a. zip 빌드 시스템이 사용되는 템플릿 또는 커뮤니티 구성 요소를 다운로드한 경우 zip 아티팩트 이름이 구성 요소 폴더의 이름과 일치하는지 확인합니다. GDK CLI에서는 구성 요소 폴더를 이름이 동일한 zip 파일로 압축됩니다. 레시피의 구성 요소 아티팩트 목록과 zip 아티팩트의 파일이 사용되는 수명 주기 스크립트에 zip 아티팩트 이름이 있습니다. zip 파일 이름이 구성 요소 폴더의 이름과 일치하도록 Artifacts 및 Lifecycle 정의를 업데이트합니다. 다음 부분 레시피 예제에서는 Artifacts 및 Lifecycle 정의의 zip 파일 이름이 강조 표시됩니다.

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, 수명 주기 스크립트 및 플랫폼 지원을 업데이트합니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 섹션을 참조하세요.

이 단계를 완료하면 레시피 파일이 다음 예제와 비슷하게 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](#) 명령에서는 구성 요소 폴더의 greengrass-build 폴더에 레시피와 아티팩트가 생성됩니다. 다음 명령을 실행합니다.

```
gdk component build
```

구성 요소를 테스트할 준비가 되면 GDK CLI를 사용하여 AWS IoT Greengrass 서비스에 게시합니다. 그런 다음에 구성 요소를 다른 코어 디바이스에 배포할 수 있습니다. 자세한 내용은 [코어 디바이스에 배포할 구성 요소를 게시하세요](#) 섹션을 참조하세요.

구성 요소 생성(셸 명령)

이 단원의 지침에 따라 여러 구성 요소에 대한 소스 코드 및 아티팩트가 있는 레시피 및 아티팩트 폴더를 생성합니다.

Greengrass 구성 요소를 개발하려면(셸 명령)

1. 레시피 및 아티팩트의 하위 폴더로 구성 요소의 폴더를 생성합니다. Greengrass 코어 디바이스에서 다음 명령을 실행하여 이러한 폴더를 생성하고 구성 요소 폴더로 변경합니다. 로컬 개발에 사용할 폴더 경로로 `~/greengrassv2` 또는 `%USERPROFILE%\greengrassv2`를 바꿉니다.

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 구성 요소의 메타데이터, 파라미터, 종속성, 수명 주기 및 플랫폼 기능이 정의되는 레시피 파일을 생성합니다. 레시피에 반영되는 구성 요소 버전을 식별할 수 있도록 레시피 파일 이름에 구성 요소 버전을 포함합니다. 레시피에 YAML 또는 JSON 형식을 선택할 수 있습니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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 Greengrass에서는 구성 요소에 의미 체계 버전이 사용됩니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 1.0.0 버전은 구성 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

3. 구성 요소의 레시피를 정의합니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 섹션을 참조하세요.

레시피는 다음 Hello World 예제 레시피와 비슷하게 보일 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 레시피에서는 다음 예제 스크립트와 비슷하게 보일 수 있는 Hello World Python 스크립트가 실행됩니다.

```

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 버전의 폴더를 생성합니다. 각 구성 요소 버전의 아티팩트를 지 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. 레시피에서 지정하는 구성 요소 이름과 버전을 포함합니다.

```
artifacts/componentName/componentVersion/
```

- 이전 단계에서 생성한 폴더에서 구성 요소의 아티팩트를 생성합니다. 아티팩트에는 소프트웨어, 이미지, 구성 요소에서 사용되는 기타 바이너리가 포함될 수 있습니다.

구성 요소가 준비되면 [구성 요소를 테스트](#)합니다.

로컬 배포를 사용한 AWS IoT Greengrass 구성 요소 테스트

코어 디바이스에서 Greengrass 구성 요소를 개발하는 경우 로컬 배포를 생성하여 설치하고 테스트할 수 있습니다. 이 섹션의 단계에 따라 로컬 배포를 생성합니다.

다른 컴퓨터(예: 로컬 개발 컴퓨터)에서 구성 요소를 개발하는 경우 로컬 배포를 생성할 수 없습니다. 그 대신에 구성 요소를 AWS IoT Greengrass 서비스에 게시하여 Greengrass 코어 디바이스에 배포하고 테스트할 수 있습니다. 자세한 내용은 [코어 디바이스에 배포할 구성 요소를 게시하세요](#) 및 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)(을)를 참조하세요.

Greengrass 코어 디바이스에서 구성 요소를 테스트하려면

- 코어 디바이스는 구성 요소 업데이트와 같은 이벤트를 기록합니다. 이 로그 파일을 보고 잘못된 레시피와 같은 구성 요소 오류를 검색하고 해결할 수 있습니다. 이 로그 파일에는 구성 요소가 표준 출력(stdout)으로 출력하는 메시지도 표시됩니다. 새로운 로그 메시지를 실시간으로 관찰하려면 코어 디바이스에서 터미널 세션을 추가로 열어두는 것이 좋습니다. SSH 등을 통해 새 터미널 세션을 열고 다음 명령을 실행하여 로그를 확인합니다. `/greengrass/v2`를 AWS IoT Greengrass 루트 폴더의 경로로 바꿉니다.

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 파일을 볼 수도 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 디바이스를 업데이트합니다. `/greengrass/v2`를 AWS IoT Greengrass 루트 폴더의 경로로 바꾸고 `~/greengrassv2`를 로컬 개발 폴더의 경로로 바꿉니다.

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 파라미터 값을 설정할 수도 있습니다. 자세한 내용은 [생성](#) 섹션을 참조하세요.

3. greengrass-cli deployment status 명령을 사용하여 구성 요소의 배포 진행 상황을 모니터링합니다.

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 테스트합니다. 이 버전의 구성 요소를 완료하면 이를 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](#) 섹션을 참조하세요.

코어 디바이스에 배포할 구성 요소를 게시하세요.

구성 요소의 버전을 빌드하거나 완료한 후 AWS IoT Greengrass 서비스에 게시할 수 있습니다. 그런 다음 Greengrass 코어 디바이스에 배포할 수 있습니다.

[Greengrass 개발 키트 CLI\(GDK CLI\)](#)를 사용하여 [구성 요소를 개발하고 빌드하는 경우 GDK CLI를 사용하여 구성 요소를 AWS 클라우드에 게시할 수 있습니다.](#) 그렇지 않으면 [기본 제공 셸 명령과 AWS CLI를 사용하여 구성 요소를 게시합니다.](#)

또한 AWS CloudFormation을 사용하여 템플릿에서 구성 요소 및 기타 AWS 리소스를 생성할 수도 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 [AWS CloudFormation이란?](#) 및 [AWS::GreengrassV2::ComponentVersion](#)을 참조하세요.

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 게시합니다. GDK CLI는 빌드 아티팩트를 S3 버킷에 업로드하고, 레시피에서 아티팩트 URI를 업데이트하고, 레시피에서 구성 요소를 생성합니다. [GDK CLI 구성 파일](#)에서 사용할 S3 버킷 및 리전을 지정합니다.

GDK CLI v1.1.0 이상을 사용하는 경우 --bucket 인수를 지정하여 GDK CLI가 구성 요소의 아티팩트를 업로드하는 S3 버킷을 지정할 수 있습니다. 이 인수를 지정하지 않으면 GDK CLI가 이름이 **bucket-region-accountId**인 S3 버킷에 업로드합니다. 여기서 **bucket**과 **region**은 gdk-config.json에서 지정하는 값이고, **accountId**는 AWS 계정 ID입니다. GDK CLI는 버킷이 없는 경우 버킷을 생성합니다.

Important

코어 디바이스 역할에서는 기본적으로 S3 버킷에 대한 액세스를 허용하지 않습니다. 이 S3 버킷을 처음 사용하는 경우 코어 디바이스가 이 S3 버킷에서 구성 요소 아티팩트를 검색하도록 허용하는 권한을 역할에 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 섹션을 참조하세요.

Greengrass 구성 요소를 게시하려면(GDK CLI)

1. 명령 프롬프트 또는 터미널에서 구성 요소 폴더를 엽니다.
2. 아직 하지 않은 경우 Greengrass 구성 요소를 빌드합니다. [구성 요소 빌드](#) 명령에서는 구성 요소 폴더의 greengrass-build 폴더에 레시피와 아티팩트가 생성됩니다. 다음 명령을 실행합니다.

```
gdk component build
```

3. 구성 요소를 AWS 클라우드에 게시합니다. [component publish](#) 명령은 구성 요소의 아티팩트를 Amazon S3에 업로드하고 각 아티팩트의 URI로 구성 요소의 레시피를 업데이트합니다. 그런 다음 AWS IoT Greengrass 서비스에서 구성 요소를 생성합니다.

 Note

AWS IoT Greengrass는 구성 요소를 생성할 때 각 아티팩트의 다이제스트를 계산합니다. 따라서 구성 요소를 생성한 후에는 S3 버킷에서 아티팩트 파일을 수정할 수 없습니다. 수정하는 경우 파일 다이제스트가 일치하지 않게 되므로 이 구성 요소가 포함된 배포가 실패합니다. 아티팩트 파일을 수정하는 경우 구성 요소의 새 버전을 생성해야 합니다.

GDK CLI 구성 파일에서 구성 요소 버전에 NEXT_PATCH를 지정하는 경우 GDK CLI는 AWS IoT Greengrass 서비스에 아직 없는 다음 패치 버전을 사용합니다.

다음 명령을 실행합니다.

```
gdk component publish
```

출력에서는 GDK CLI가 생성한 구성 요소의 버전을 알려줍니다.

구성 요소를 게시한 후 코어 디바이스에 구성 요소를 배포할 수 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.

구성 요소 게시(셸 명령)

다음 절차에 따라 셸 명령과 AWS Command Line Interface(AWS CLI)를 사용하여 구성 요소를 게시합니다. 구성 요소를 게시할 때 다음을 수행합니다.

1. 구성 요소 아티팩트를 S3 버킷에 게시합니다.
2. 각 아티팩트의 Amazon S3 URI를 구성 요소 레시피에 추가합니다.
3. AWS IoT Greengrass에서 구성 요소 레시피를 통해 구성 요소 버전을 생성합니다.

Note

업로드하는 각 구성 요소 버전은 고유해야 합니다. 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 늘리고 파일을 저장합니다. 구성 요소에 대해 변경한 사항을 반영하는 새 버전을 선택합니다.

Note

AWS IoT Greengrass에서는 구성 요소에 의미 체계 버전이 사용됩니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 1.0.0 버전은 구성 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

2. 구성 요소에 아티팩트가 있는 경우 다음을 수행합니다.
 - a. 구성 요소의 아티팩트를 AWS 계정의 S3 버킷에 게시합니다.

Tip

S3 버킷의 아티팩트 경로에 구성 요소 이름과 버전을 포함하는 것이 좋습니다. 이 이름 지정 체계는 이전 버전의 구성 요소가 사용하는 아티팩트를 유지 관리하는 데 도움이 될 수 있으므로 이전 구성 요소 버전을 계속 지원할 수 있습니다.

다음 명령을 실행하여 아티팩트 파일을 S3 버킷에 게시합니다. amzn-s3-demo-bucket을 버킷 이름으로 바꾸고 `artifacts/com.example.HelloWorld/1.0.0/artifact.py`를 아티팩트 파일의 경로로 바꿉니다.

```
aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

⚠ Important

코어 디바이스 역할에서는 기본적으로 S3 버킷에 대한 액세스를 허용하지 않습니다. 이 S3 버킷을 처음 사용하는 경우 코어 디바이스가 이 S3 버킷에서 구성 요소 아티팩트를 검색하도록 허용하는 권한을 역할에 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 섹션을 참조하세요.

- b. 없는 경우 Artifacts라는 목록을 구성 요소 레시피에 추가합니다. Artifacts 목록은 구성 요소가 지원하는 각 플랫폼에서 구성 요소의 요구 사항(또는 모든 플랫폼에 대한 구성 요소의 기본 요구 사항)을 정의하는 각 매니페스트에 표시됩니다.
- c. 아티팩트 목록에 각 아티팩트를 추가하거나 기존 아티팩트의 URI를 업데이트합니다. Amazon S3 URI는 버킷 이름과 버킷의 아티팩트 객체 경로로 구성되어 있습니다. 아티팩트의 Amazon S3 URI는 다음 예와 비슷해야 합니다.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

이 단계를 완료한 후 레시피에는 다음 Artifacts 목록이 있어야 합니다.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
MyGreengrassComponent/1.0.0/artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
  ]
}
```

Note

ZIP 아티팩트에 대한 "Unarchive": "ZIP" 옵션을 추가하여 구성 요소가 배포될 때 아티팩트의 압축을 풀도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다.

YAML

```
...
Manifests:
  - Lifecycle:
      ...
      Artifacts:
        - URI: s3://amzn-s3-demo-bucket/artifacts/MyGreengrassComponent/1.0.0/artifact.py
          Unarchive: NONE
```

Note

Unarchive: ZIP 옵션을 사용하여 구성 요소가 배포될 때 ZIP 아티팩트의 압축을 풀도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 구성 요소에서 ZIP 아티팩트를 사용하는 방법에 대한 자세한 내용은 [artifacts:decompressedPath 레시피 변수](#)를 참조하세요.

레시피에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 단원을 참조하세요.

3. AWS IoT Greengrass 콘솔을 사용하여 레시피 파일에서 구성 요소를 생성합니다.

다음 명령을 실행하여 레시피 파일에서 구성 요소를 생성합니다. 이 명령은 구성 요소를 생성하고 AWS 계정의 프라이빗 AWS IoT Greengrass 구성 요소로 게시합니다. `path/to/recipeFile`을 레시피 파일의 경로로 바꿉니다.

```
aws greengrassv2 create-component-version --inline-recipe fileb://path/to/recipeFile
```

응답에서 arn를 복사하여 다음 단계에서 구성 요소의 상태를 확인합니다.

Note

AWS IoT Greengrass는 구성 요소를 생성할 때 각 아티팩트의 다이제스트를 계산합니다. 따라서 구성 요소를 생성한 후에는 S3 버킷에서 아티팩트 파일을 수정할 수 없습니다. 수정하는 경우 파일 다이제스트가 일치하지 않게 되므로 이 구성 요소가 포함된 배포가 실패합니다. 아티팩트 파일을 수정하는 경우 구성 요소의 새 버전을 생성해야 합니다.

4. AWS IoT Greengrass 서비스의 각 구성 요소에는 상태가 있습니다. 다음 명령을 실행하여 이 절차에서 게시하는 구성 요소 버전의 상태를 확인합니다. `com.example.HelloWorld` 및 `1.0.0`을 쿼리할 구성 요소 버전으로 바꿉니다. `arn`을 이전 단계의 ARN으로 바꿉니다.

```
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 상태와 해당하는 경우 오류를 포함하는 `status` 객체가 포함되어 있습니다.

구성 요소 상태가 `DEPLOYABLE`이면 구성 요소를 디바이스에 배포할 수 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.

AWS 서비스와 상호 작용

Greengrass 코어 디바이스는 X.509 인증서를 사용하여 TLS 상호 인증 프로토콜을 사용하는 AWS IoT Core에 연결합니다. 이러한 인증서를 사용하면 디바이스가 일반적으로 액세스 키 ID 및 비밀 액세스 키로 구성되는 AWS 자격 증명 없이 AWS IoT와 상호 작용할 수 있습니다. 다른 AWS 서비스는 서비스 엔드포인트에서 API 작업을 직접 호출하려면 X.509 인증서 대신 AWS 자격 증명이 필요합니다. AWS IoT Core에는 디바이스가 X.509 인증서를 사용하여 AWS 요청을 인증할 수 있도록 하는 자격 증명 공급자가 있습니다. AWS IoT 자격 증명 공급자는 X.509 인증서를 사용하여 디바이스를 인증하고 제한된 권한의 임시 보안 토큰 형태로 AWS 자격 증명을 발급합니다. 디바이스는 이 토큰을 사용하여 모든 AWS 요청을 서명하고 인증할 수 있습니다. 이렇게 하면 Greengrass 코어 디바이스에 AWS 자격 증명을 저장할 필요가 없습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스 직접 호출에 대한 권한 부여](#)를 참조하세요.

AWS IoT에서 자격 증명을 가져오려면 코어 디바이스는 IAM 역할을 가리키는 AWS IoT 역할 별칭을 사용합니다. 이 IAM 역할을 토큰 교환 역할이라고 합니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 때 역할 별칭과 토큰 교환 역할을 생성합니다. 코어 디바이스가 사용하는 역할 별칭을 지정하려면 [Greengrass nucleus](#)의 `iotRoleAlias` 파라미터를 구성합니다.

AWS IoT 자격 증명 공급자는 사용자를 대신하여 토큰 교환 역할을 수임하여 코어 디바이스에 AWS 자격 증명을 제공합니다. 이 역할에 적절한 IAM 정책을 연결하여 코어 디바이스가 S3 버킷의 구성 요소 아티팩트와 같은 AWS 리소스에 액세스할 수 있습니다. 토큰 교환 역할을 구성하는 방법에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

Greengrass 코어 디바이스는 AWS 자격 증명을 메모리에 저장하며 자격 증명은 기본적으로 1시간 후에 만료됩니다. AWS IoT Greengrass 코어 소프트웨어가 다시 시작되면 자격 증명을 다시 가져와야 합니다. [UpdateRoleAlias](#) 작업을 사용하여 자격 증명 유효 기간을 구성할 수 있습니다.

AWS IoT Greengrass는 AWS 서비스와 상호 작용할 사용자 지정 구성 요소의 종속성으로 정의할 수 있는 토큰 교환 서비스 구성 요소인 퍼블릭 구성 요소를 제공합니다. 토큰 교환 서비스는 AWS 자격 증명을 제공하는 로컬 서버에 대한 URI를 정의하는 환경 변수인 `AWS_CONTAINER_CREDENTIALS_FULL_URI`를 구성 요소에 제공합니다. AWS SDK 클라이언트를 생성할 때 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 내용은 [토큰 교환 서비스](#) 섹션을 참조하세요.

Important

이러한 방식으로 AWS 자격 증명을 획득하도록 하는 지원은 2016년 7월 13일에 AWS SDK에 추가되었습니다. 구성 요소는 이 날짜 이후에 생성된 AWS SDK 버전을 사용해야 합니다. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 [지원되는 AWS SDK 사용](#)을 참조하세요.

사용자 지정 구성 요소에서 AWS 자격 증명을 획득하려면 구성 요소 레시피에서 `aws.greengrass.TokenExchangeService`를 종속성으로 정의합니다. 다음 예제 레시피에서는 [boto3](#)를 설치하고 토큰 교환 서비스의 AWS 자격 증명을 사용하여 Amazon S3 버킷을 나열하는 Python 스크립트를 실행하는 구성 요소를 정의합니다.

Note

이 예제 구성 요소를 실행하려면 디바이스에 `s3:ListAllMyBuckets` 권한이 있어야 합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 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 나열하는 다음 Python 스크립트 `list_s3_buckets.py`를 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)
```

Docker 컨테이너 실행

다음 위치에 저장된 이미지에서 [Docker](#) 컨테이너를 실행하도록 AWS IoT Greengrass 구성 요소를 구성할 수 있습니다.

- Amazon Elastic Container Registry(Amazon ECR)의 퍼블릭 및 프라이빗 이미지 리포지토리
- 퍼블릭 Docker Hub 리포지토리
- 퍼블릭 Docker 신뢰할 수 있는 레지스트리
- S3 버킷

사용자 지정 컴포넌트에서 이미지를 검색하고 코어 디바이스에서 실행할 수 있도록 Docker 이미지 URI를 아티팩트로 포함하세요. Amazon ECR 및 Docker Hub 이미지의 경우 [Docker 애플리케이션 관리자](#) 구성 요소를 사용하여 이미지를 다운로드하고 프라이빗 Amazon ECR 리포지토리에 대한 자격 증명을 관리할 수 있습니다.

주제

- [요구 사항](#)
- [Amazon ECR 또는 Docker Hub의 퍼블릭 이미지에서 Docker 컨테이너 실행](#)
- [Amazon ECR의 프라이빗 이미지에서 Docker 컨테이너 실행](#)
- [Amazon S3의 이미지에서 Docker 컨테이너 실행](#)
- [Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용](#)
- [Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용\(Linux\)](#)
- [Docker 컨테이너 구성 요소에서 스트림 관리자 사용\(Linux\)](#)

요구 사항

구성 요소에서 Docker 컨테이너를 실행하려면 다음이 필요합니다.

- Greengrass 코어 디바이스. 없으면 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요.

- Greengrass 코어 디바이스에 [Docker Engine](#) 1.9.1 이상이 설치되어 있어야 합니다. 버전 20.10은 AWS IoT Greengrass 코어 소프트웨어에서 작동하는 것으로 확인된 최신 버전입니다. Docker 컨테이너를 실행하는 구성 요소를 배포하기 전에 코어 디바이스에 직접 Docker를 설치해야 합니다.

Tip

구성 요소가 설치될 때 코어 디바이스가 Docker Engine을 설치하도록 구성할 수도 있습니다. 예를 들어 다음 설치 스크립트는 Docker 이미지를 로드하기 전에 Docker Engine을 설치합니다. 이 설치 스크립트는 Ubuntu와 같은 Debian 기반 Linux 배포에서 작동합니다. 이 명령으로 Docker Engine을 설치하도록 구성하는 경우 설치 및 docker 명령을 실행하려면 수명 주기 스크립트에서 RequiresPrivilege를 true로 설정해야 할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 단원을 참조하십시오.

```
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 sudo 없이 docker 명령을 직접적으로 호출할 수 있습니다.
- Windows 디바이스에서는 docker-users 그룹에 사용자를 추가하여 관리자 권한 없이 docker 명령을 직접적으로 호출할 수 있습니다.

Linux or Unix

Docker 컨테이너 구성 요소를 실행하는 데 사용하는 ggc_user 또는 루트 사용자가 아닌 사용자를 docker 그룹에 추가하려면 다음 명령을 실행합니다.

```
sudo usermod -aG docker ggc_user
```

자세한 내용은 [루트 사용자가 아닌 사용자로 Docker 관리](#)를 참조하세요.

Windows Command Prompt (CMD)

ggc_user 또는 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자를 docker-users 그룹에 추가하려면 관리자로 다음 명령을 실행합니다.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

ggc_user 또는 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자를 docker-users 그룹에 추가하려면 관리자로 다음 명령을 실행합니다.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Docker 컨테이너에 [볼륨으로 탑재](#)된 Docker 컨테이너 구성 요소에서 액세스하는 파일입니다.
- [네트워크 프록시를 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성하는 경우 동일한 프록시 서버를 사용하도록 Docker를 구성](#)해야 합니다.

이러한 요구 사항 외에도 사용자 환경에 적용되는 경우 다음과 같은 요구 사항도 충족해야 합니다.

- [Docker Compose](#)를 사용하여 Docker 컨테이너를 생성하고 시작하려면 Greengrass 코어 디바이스에 Docker Compose를 설치하고 Docker Compose 파일을 S3 버킷에 업로드하세요. Compose 파일은 구성 AWS 리전 요소와 동일한 AWS 계정 및의 S3 버킷에 저장해야 합니다. 사용자 지정 컴포넌트에서 docker-compose up 명령을 사용하는 예제는 [Amazon ECR 또는 Docker Hub의 퍼블릭 이미지에서 Docker 컨테이너 실행](#) 섹션을 참조하세요.
- 네트워크 프록시 AWS IoT Greengrass 뒤에서를 실행하는 경우 [프록시 서버](#)를 사용하도록 Docker 데몬을 구성합니다.
- Docker 이미지가 Amazon ECR 또는 Docker Hub에 저장되어 있는 경우 Docker 컨테이너 구성 요소에 [Docker 구성 요소 관리자](#) 구성 요소를 종속성으로 포함합니다. 구성 요소를 배포하기 전에 코어 디바이스에서 Docker 데몬을 시작해야 합니다.

또한 이미지 URI를 구성 요소 아티팩트로 포함하세요. 이미지 URI는 다음 예제와 같이 `docker:registry/image[:tag@digest]` 형식이어야 합니다.

- 프라이빗 Amazon ECR 이미지: `docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag@digest]`
- 퍼블릭 Amazon ECR 이미지: `docker:public.ecr.aws/repository/image[:tag@digest]`
- 퍼블릭 Docker Hub 이미지: `docker:name[:tag@digest]`

퍼블릭 리포지토리에 저장된 이미지에서 Docker 컨테이너를 실행하는 방법에 대한 자세한 내용은 [Amazon ECR 또는 Docker Hub의 퍼블릭 이미지에서 Docker 컨테이너 실행](#) 섹션을 참조하세요.

- Docker 이미지가 Amazon ECR 프라이빗 리포지토리에 저장되어 있는 경우 토큰 교환 서비스 구성 요소를 Docker 컨테이너 구성 요소에 종속성으로 포함해야 합니다. 또한 다음 예제 IAM 정책에 표시

된 대로 [Greengrass 디바이스 역할](#)에서 `ecr:GetAuthorizationToken`, `ecr:BatchGetImage`, `ecr:GetDownloadUrlForLayer` 작업을 허용해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Amazon ECR 프라이빗 리포지토리에 저장된 이미지에서 Docker 컨테이너를 실행하는 방법에 대한 자세한 내용은 [Amazon ECR의 프라이빗 이미지에서 Docker 컨테이너 실행](#) 섹션을 참조하세요.

- Amazon ECR 프라이빗 리포지토리에 저장된 Docker 이미지를 사용하려면 프라이빗 리포지토리가 코어 디바이스 AWS 리전 와 동일한에 있어야 합니다.
- Docker 이미지 또는 Compose 파일이 S3 버킷에 저장되어 있는 경우 다음 예제 IAM 정책에 표시된 대로 [Greengrass 디바이스 역할](#)에서 코어 디바이스가 이미지를 구성 요소 아티팩트로 다운로드할 수 있도록 허용하는 `s3:GetObject` 권한을 허용해야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

```
    ],
    "Effect": "Allow"
  }
]
```

Amazon S3에 저장된 이미지에서 Docker 컨테이너를 실행하는 방법에 대한 자세한 내용은 [Amazon S3의 이미지에서 Docker 컨테이너 실행](#) 섹션을 참조하세요.

- Docker 컨테이너 구성 요소에서 프로세스 간 통신(IPC), AWS 자격 증명 또는 스트림 관리자를 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음을 참조하세요.
 - [Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용](#)
 - [Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용\(Linux\)](#)
 - [Docker 컨테이너 구성 요소에서 스트림 관리자 사용\(Linux\)](#)

Amazon ECR 또는 Docker Hub의 퍼블릭 이미지에서 Docker 컨테이너 실행

이 섹션에서는 Docker Compose를 사용하여 Amazon ECR 및 Docker Hub에 저장된 Docker 이미지에서 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 생성하는 방법을 설명합니다.

Docker Compose를 사용하여 Docker 컨테이너를 실행하려면

1. Docker Compose 파일을 생성하고 Amazon S3 버킷에 업로드합니다. [Greengrass 디바이스 역할](#)이 디바이스에서 Compose 파일에 액세스할 수 있는 s3:GetObject 권한을 허용하는지 확인합니다. 다음 예제에 표시된 Compose 파일 예제에는 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. AWS IoT Greengrass 코어 디바이스에서 [사용자 지정 구성 요소를 생성합니다](#). 다음 예제에 표시된 예제 레시피에는 다음 속성이 있습니다.

- 종속성으로서의 Docker 애플리케이션 관리자 구성 요소입니다. 이 구성 요소를 사용하면 AWS IoT Greengrass 가 퍼블릭 Amazon ECR 및 Docker Hub 리포지토리에서 이미지를 다운로드할 수 있습니다.
- 퍼블릭 Amazon ECR 리포지토리에서 Docker 이미지를 지정하는 구성 요소 아티팩트입니다.
- 퍼블릭 Docker Hub 리포지토리에서 Docker 이미지를 지정하는 구성 요소 아티팩트입니다.
- 실행하려는 Docker 이미지의 컨테이너를 포함하는 Docker Compose 파일을 지정하는 구성 요소 아티팩트입니다.
- 지정된 이미지에서 컨테이너를 생성하고 시작하기 위해 [docker-compose up](#)을 사용하는 수명 주기 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://amzn-s3-demo-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://amzn-s3-demo-bucket/folder/docker-compose.yaml"

```

Note

Docker 컨테이너 구성 요소에서 프로세스 간 통신(IPC), AWS 자격 증명 또는 스트림 관리자 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음을 참조하세요.

- [Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용](#)
- [Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용\(Linux\)](#)

- [Docker 컨테이너 구성 요소에서 스트림 관리자 사용\(Linux\)](#)

3. [구성 요소를 테스트](#)하여 예상대로 작동하는지 확인합니다.

Important

구성 요소를 배포하기 전에 Docker 대몬을 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 요소에 업로드 AWS IoT Greengrass 하여 다른 코어 디바이스에 배포합니다. 자세한 내용은 [코어 디바이스에 배포할 구성 요소를 게시하세요](#) 단원을 참조하십시오.

Amazon ECR의 프라이빗 이미지에서 Docker 컨테이너 실행

이 섹션에서는 Amazon ECR의 프라이빗 리포지토리에 저장된 Docker 이미지에서 Docker 컨테이너를 실행하는 사용자 지정 구성 요소를 만드는 방법을 설명합니다.

Docker 컨테이너를 실행하려면

1. AWS IoT Greengrass 코어 디바이스에서 [사용자 지정 구성 요소를 생성합니다](#). 다음 속성이 있는 다음 레시피 예제를 사용합니다.
 - 종속성으로서의 Docker 애플리케이션 관리자 구성 요소입니다. 이 구성 요소를 사용하면 AWS IoT Greengrass 가 프라이빗 리포지토리에 이미지를 다운로드하기 위한 자격 증명을 관리할 수 있습니다.
 - 종속성으로서의 토큰 교환 서비스 구성 요소입니다. 이 구성 요소를 사용하면 AWS IoT Greengrass 가 Amazon ECR과 상호 작용할 자격 AWS 증명을 검색할 수 있습니다.
 - 프라이빗 Amazon ECR 리포지토리에 Docker 이미지를 지정하는 구성 요소 아티팩트입니다.
 - 이미지에서 컨테이너를 생성하고 시작하기 위해 [docker run](#)을 사용하는 수명 주기 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 프로세스 간 통신(IPC), AWS 자격 증명 또는 스트림 관리자 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음을 참조하세요.

- [Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용](#)
- [Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용\(Linux\)](#)
- [Docker 컨테이너 구성 요소에서 스트림 관리자 사용\(Linux\)](#)

2. [구성 요소를 테스트](#)하여 예상대로 작동하는지 확인합니다.

Important

구성 요소를 배포하기 전에 Docker 대몬을 설치하고 시작해야 합니다.

구성 요소를 로컬에 배포한 후 [docker container ls](#) 명령을 실행하여 컨테이너가 실행되는지 확인할 수 있습니다.

```
docker container ls
```

3. 구성 요소에 업로드 AWS IoT Greengrass 하여 다른 코어 디바이스에 배포합니다. 자세한 내용은 [코어 디바이스에 배포할 구성 요소를 게시하세요](#) 단원을 참조하십시오.

Amazon S3의 이미지에서 Docker 컨테이너 실행

이 섹션에서는 Amazon S3에 저장된 Docker 이미지의 구성 요소에서 Docker 컨테이너를 실행하는 방법을 설명합니다.

Amazon S3의 이미지에서 구성 요소에서 Docker 컨테이너를 실행하려면

1. [docker save](#) 명령을 실행하여 Docker 컨테이너의 백업을 생성합니다. 이 백업을 구성 요소 아티팩트로 제공하여 AWS IoT Greengrass에서 컨테이너를 실행합니다. `hello-world`를 이미지 이름으로 바꾸고 `hello-world.tar`를 생성할 아카이브 파일 이름으로 바꿉니다.

```
docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
```

2. AWS IoT Greengrass 코어 디바이스에서 [사용자 지정 구성 요소를 생성합니다](#). 다음 속성이 있는 다음 레시피 예제를 사용합니다.
 - 아카이브에서 Docker 이미지를 로드하기 위해 [docker load](#)를 사용하는 수명 주기 설치 스크립트입니다.
 - 이미지에서 컨테이너를 생성하고 시작하기 위해 [docker run](#)을 사용하는 수명 주기 실행 스크립트입니다. `--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 프로세스 간 통신(IPC), AWS 자격 증명 또는 스트림 관리자를 사용하려면 Docker 컨테이너를 실행할 때 추가 옵션을 지정해야 합니다. 자세한 내용은 다음을 참조하세요.

- [Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용](#)
- [Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용\(Linux\)](#)

- [Docker 컨테이너 구성 요소에서 스트림 관리자 사용\(Linux\)](#)

3. [구성 요소를 테스트](#)하여 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 요소에 업로드 AWS IoT Greengrass 하여 다른 코어 디바이스에 배포할 수 있습니다. 자세한 내용은 [코어 디바이스에 배포할 구성 요소를 게시하세요](#) 단원을 참조하십시오.

완료되면 구성 요소 레시피는 다음 예제와 유사하게 보일 것입니다.

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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
        com.example.MyDockerComponent/1.0.0/hello-world.tar

```

Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용

의 Greengrass 프로세스 간 통신(IPC) 라이브러리를 사용하여 Greengrass nucleus, 기타 Greengrass 구성 요소 및와 통신 AWS IoT Device SDK 할 수 있습니다 AWS IoT Core. 자세한 내용은 [AWS IoT Device SDK 를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core](#) 단원을 참조하십시오.

Docker 컨테이너 구성 요소에서 IPC를 사용하려면 다음 파라미터를 사용하여 Docker 컨테이너를 실행해야 합니다.

- 컨테이너에 IPC 소켓을 탑재합니다. Greengrass nucleus는 `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` 환경 변수의 IPC 소켓 파일 경로를 제공합니다.

- SVCUID 및 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT 환경 변수를 Greengrass nucleus가 구성 요소에 제공하는 값으로 설정합니다. 구성 요소는 이러한 환경 변수를 사용하여 Greengrass nucleus에 대한 연결을 인증합니다.

Example레시피 예제:에 MQTT 메시지 게시 AWS IoT Core (Python)

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 AWS IoT Core 에 게시하도록 허용하는 권한 부여 정책(accessControl)입니다. 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 및 [AWS IoT Core MQTT IPC 권한 부여](#)를 참조하세요.
- Amazon S3에서 Docker 이미지를 TAR 아카이브로 지정하는 구성 요소 아티팩트입니다.
- TAR 아카이브에서 Docker 이미지를 로드하는 수명 주기 설치 스크립트입니다.
- 이미지에서 Docker 컨테이너를 실행하는 수명 주기 실행 스크립트입니다. [Docker 실행](#) 명령에는 다음 인수가 있습니다.
 - -v 인수는 컨테이너에 Greengrass IPC 소켓을 탑재합니다.
 - 처음 두 -e 인수는 Docker 컨테이너에서 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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
      com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar

```

Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용(Linux)

[토큰 교환 서비스 구성 요소](#)를 사용하여 Greengrass 구성 요소의 AWS 서비스와 상호 작용할 수 있습니다. 이 구성 요소는 로컬 컨테이너 서버를 사용하여 코어 디바이스의 [토큰 교환 역할](#)에서 AWS 자격 증명을 제공합니다. 자세한 내용은 [AWS 서비스와 상호 작용](#) 단원을 참조하십시오.

Note

이 섹션의 예제는 Linux 코어 디바이스에서만 작동합니다.

Docker 컨테이너 구성 요소에서 토큰 교환 서비스의 AWS 자격 증명을 사용하려면 다음 파라미터와 함께 Docker 컨테이너를 실행해야 합니다.

- `--network=host` 인수를 사용하여 호스트 네트워크에 대한 액세스를 제공합니다. 이 옵션을 사용하면 Docker 컨테이너가 로컬 토큰 교환 서비스에 연결하여 AWS 자격 증명을 검색할 수 있습니다. 이 인수는 Linux용 Docker에서만 작동합니다.

Warning

이 옵션을 사용하면 컨테이너가 호스트의 모든 로컬 네트워크 인터페이스에 액세스할 수 있으므로 호스트 네트워크에 대한 액세스 권한 없이 Docker 컨테이너를 실행하는 경우보다 안전하지 않습니다. 이 옵션을 사용하는 Docker 컨테이너 구성 요소를 개발하고 실행할 때 이 점을 고려하세요. 자세한 내용은 Docker 설명서에서 [네트워크: 호스트](#)를 참조하세요.

- `AWS_CONTAINER_CREDENTIALS_FULL_URI` 및 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 환경 변수를 Greengrass nucleus가 구성 요소에 제공하는 값으로 설정합니다. AWS SDKs 이러한 환경 변수를 사용하여 AWS 자격 증명을 검색합니다.

Example예제 레시피: Docker 컨테이너 구성 요소에 S3 버킷 나열(Python)

다음 레시피는 AWS 계정에 S3 버킷을 나열하는 예제 Docker 컨테이너 구성 요소를 정의합니다. 이 레시피의 속성은 다음과 같습니다.

- 종속성으로서의 토큰 교환 서비스 구성 요소입니다. 이 종속성을 사용하면 구성 요소가 자격 AWS 증명을 검색하여 다른 AWS 서비스와 상호 작용할 수 있습니다.
- Amazon S3에서 Docker 이미지를 TAR 아카이브로 지정하는 구성 요소 아티팩트입니다.
- TAR 아카이브에서 Docker 이미지를 로드하는 수명 주기 설치 스크립트입니다.
- 이미지에서 Docker 컨테이너를 실행하는 수명 주기 실행 스크립트입니다. [Docker 실행](#) 명령에는 다음 인수가 있습니다.
 - `--network=host` 인수는 컨테이너가 토큰 교환 서비스에 연결할 수 있도록 호스트 네트워크에 대한 컨테이너 액세스를 제공합니다.
 - `-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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
      com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar

```

Docker 컨테이너 구성 요소에서 스트림 관리자 사용(Linux)

[스트림 관리자 구성 요소](#)를 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.는 스트림 관리자 구성 요소와 상호 작용하는 데 사용하는 스트림 관리자 SDK를 AWS IoT Greengrass 제공합니다. 자세한 내용은 [Greengrass 코어 디바이스에서 데이터 스트림 관리](#) 단원을 참조하십시오.

Note

이 섹션의 예제는 Linux 코어 디바이스에서만 작동합니다.

Docker 컨테이너 구성 요소에서 스트림 관리자 SDK를 사용하려면 다음 파라미터를 사용하여 Docker 컨테이너를 실행해야 합니다.

- `--network=host` 인수를 사용하여 호스트 네트워크에 대한 액세스를 제공합니다. 이 옵션을 사용하면 Docker 컨테이너가 로컬 TLS 연결을 통해 스트림 관리자 구성 요소와 상호 작용할 수 있습니다. 이 인수는 Linux용 Docker에서만 작동합니다.

⚠ Warning

이 옵션을 사용하면 컨테이너가 호스트의 모든 로컬 네트워크 인터페이스에 액세스할 수 있으므로 호스트 네트워크에 대한 액세스 권한 없이 Docker 컨테이너를 실행하는 경우보다 안전하지 않습니다. 이 옵션을 사용하는 Docker 컨테이너 구성 요소를 개발하고 실행할 때 이 점을 고려하세요. 자세한 내용은 Docker 설명서에서 [네트워크: 호스트](#)를 참조하세요.

- 기본 동작인 인증을 요구하도록 스트림 관리자 구성 요소를 구성하는 경우에는 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 환경 변수를 Greengrass nucleus가 구성 요소에 제공하는 값으로 설정합니다. 자세한 내용은 [스트림 관리자 구성](#)을 참조하세요.
- 기본 포트가 아닌 포트를 사용하도록 스트림 관리자 구성 요소를 구성하는 경우 [프로세스 간 통신 \(IPC\)](#)을 사용하여 스트림 관리자 구성 요소 구성에서 포트를 가져옵니다. IPC를 사용하려면 추가 옵션이 포함된 Docker 컨테이너를 실행해야 합니다. 자세한 내용은 다음을 참조하세요.
 - [애플리케이션 코드에서 스트림 관리자에 연결](#)
 - [Docker 컨테이너 구성 요소에서 프로세스 간 통신 사용](#)

Example예제 레시피: Docker 컨테이너 구성 요소의 S3 버킷으로 파일 스트리밍(Python)

다음 레시피는 파일을 생성하여 S3 버킷으로 스트리밍하는 예제 Docker 컨테이너 컴포넌트를 정의합니다. 이 레시피의 속성은 다음과 같습니다.

- 종속성으로서의 스트림 관리자 구성 요소. 이러한 종속성을 통해 구성 요소는 스트림 관리자 SDK를 사용하여 스트림 관리자 구성 요소와 상호 작용할 수 있습니다.
- Amazon S3에서 Docker 이미지를 TAR 아카이브로 지정하는 구성 요소 아티팩트입니다.
- TAR 아카이브에서 Docker 이미지를 로드하는 수명 주기 설치 스크립트입니다.
- 이미지에서 Docker 컨테이너를 실행하는 수명 주기 실행 스크립트입니다. [Docker 실행](#) 명령에는 다음 인수가 있습니다.
 - `--network=host` 인수는 컨테이너가 스트림 관리자 구성 요소에 연결할 수 있도록 호스트 네트워크에 대한 컨테이너 액세스를 제공합니다.
 - 첫 번째 `-e` 인수는 Docker 컨테이너에서 필요한 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 환경 변수를 설정합니다.
 - 추가 `-e` 인수는 이 예제에서 사용하는 환경 변수를 설정합니다.
 - `-v` 인수는 컨테이너에 구성 요소의 [작업 폴더](#)를 탑재합니다. 이 예제에서는 작업 폴더에 파일을 생성하고 스트림 관리자를 사용하여 해당 파일을 Amazon S3에 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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
        com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar

```

AWS IoT Greengrass 구성 요소 레시피 참조

구성 요소 레시피는 구성 요소의 세부 정보, 종속성, 아티팩트 및 수명 주기가 정의되는 파일입니다. 예를 들어 구성 요소 수명 주기에서는 실행하여 구성 요소를 설치, 실행 및 종료하는 명령이 지정됩니다. AWS IoT Greengrass 코어는 레시피에서 정의한 수명 주기를 사용하여 구성 요소를 설치하고 실행합니다. 이 AWS IoT Greengrass 서비스는 구성 요소를 배포할 때 레시피를 사용하여 코어 디바이스에 배포할 종속성과 아티팩트를 식별합니다.

구성 요소에서 지원하는 각 플랫폼의 고유한 종속성과 수명 주기를 레시피에서 정의할 수 있습니다. 이 기능을 사용하여 요구 사항이 다른 여러 플랫폼이 있는 디바이스에 구성 요소를 배포할 수 있습니다. 이를 사용하여 AWS IoT Greengrass 가 이를 지원하지 않는 디바이스에 구성 요소를 설치하지 못하도록 할 수도 있습니다.

레시피마다 매니페스트 목록이 있습니다. 각 매니페스트에서는 플랫폼 요구 사항 세트와 플랫폼에서 해당 요구 사항을 충족하는 코어 디바이스에 사용할 수명 주기 및 아티팩트가 지정됩니다. 코어 디바이스에서는 디바이스에서 충족되는 플랫폼 요구 사항이 있는 첫 번째 매니페스트가 사용됩니다. 코어 디바이스와 일치하도록 플랫폼 요구 사항이 없는 매니페스트를 지정합니다.

매니페스트에 없는 글로벌 수명 주기를 지정할 수도 있습니다. 글로벌 수명 주기에서는 수명 주기의 하위 섹션을 식별하는 선택 키를 사용할 수 있습니다. 그런 다음에 매니페스트 내에서 이러한 선택 키를 지정하여 매니페스트의 수명 주기 외에 글로벌 수명 주기의 해당 섹션도 사용할 수 있습니다. 코어 디바이스에서는 매니페스트에서 수명 주기가 정의되지 않는 경우에만 매니페스트의 선택 키가 사용됩니다. 선택 키가 없는 글로벌 수명 주기의 섹션과 일치하도록 매니페스트의 `all` 선택을 사용할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어가 코어 디바이스와 일치하는 매니페스트를 선택하면 다음을 수행하여 사용할 수명 주기 단계를 식별합니다.

- 선택된 매니페스트에서 수명 주기가 정의되면 코어 디바이스에서는 해당 수명 주기가 사용됩니다.
- 선택된 매니페스트에서 수명 주기가 정의되지 않으면 코어 디바이스에서는 글로벌 수명 주기가 사용됩니다. 코어 디바이스에서는 다음이 수행되어 글로벌 수명 주기 중 사용할 섹션이 식별됩니다.
 - 매니페스트에서 선택 키가 정의되면 코어 디바이스에서는 매니페스트의 선택 키가 있는 글로벌 수명 주기의 섹션이 사용됩니다.
 - 매니페스트에서 선택 키가 정의되지 않으면 코어 디바이스에서는 선택 키가 없는 글로벌 수명 주기의 섹션이 사용됩니다. 이 동작은 `all` 선택이 정의되는 매니페스트와 동일합니다.

Important

구성 요소를 설치하려면 코어 디바이스가 매니페스트의 플랫폼 요구 사항 하나 이상과 일치해야 합니다. 코어 디바이스와 일치하는 매니페스트가 없으면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 설치하지 않고 배포가 실패합니다.

레시피를 [JSON](#) 또는 [YAML](#) 형식으로 정의할 수 있습니다. 레시피 예제 섹션에는 각 형식의 레시피가 포함되어 있습니다.

주제

- [레시피 검증](#)
- [레시피 형식](#)
- [레시피 변수](#)
- [레시피 예제](#)

레시피 검증

Greengrass에서는 구성 요소 버전 생성 시 JSON 또는 YAML 구성 요소 레시피가 검증됩니다. 이 레시피 검증에서는 잠재적인 배포 문제가 방지되도록 JSON 또는 YAML 구성 요소 레시피의 일반적인 오류가 확인됩니다. 검증에서는 레시피의 일반적인 오류(예: 심표, 종괄호 및 필드 누락)가 확인되고, 레시피가 제대로 구성되었는지 확인됩니다.

레시피 검증 오류 메시지가 표시되면 레시피에 누락된 심표, 종괄호 또는 필드가 있는지 확인합니다. [레시피 형식](#)을 조사하여 누락된 필드가 없는지 확인합니다.

레시피 형식

구성 요소에 대한 레시피를 정의할 때 레시피 문서에서 다음 정보를 지정합니다. 동일한 구조가 YAML 및 JSON 형식의 레시피에 적용됩니다.

RecipeFormatVersion

레시피의 템플릿 버전입니다. 다음 옵션을 선택합니다.

- 2020-01-25

ComponentName

이 레시피에서 정의되는 구성 요소의 이름입니다. 구성 요소 이름은 각 리전의 AWS 계정에서 고유해야 합니다.

팁

- 회사 내에서 이름 충돌을 방지하려면 역 도메인 이름 형식을 사용합니다. 예를 들어 회사에서 example.com을 소유하고 태양 에너지 프로젝트를 수행하는 경우 Hello World 구성 요소의 이름을 com.example.solar>HelloWorld로 지정할 수 있습니다. 그러면 회사 내 구성 요소 이름 충돌을 방지하는 데 도움이 됩니다.
- 구성 요소 이름에서 aws.greengrass 접두사는 피합니다. AWS IoT Greengrass에서 제공되는 [퍼블릭 구성 요소](#)에 이 접두사가 사용됩니다. 퍼블릭 구성 요소와 동일한 이름

을 선택하면 사용자가 선택한 구성 요소로 해당 구성 요소가 바뀝니다. 그런 다음 해당 퍼블릭 구성 요소에 종속된 구성 요소를 배포할 때 퍼블릭 구성 요소 대신 구성 요소를 AWS IoT Greengrass 제공합니다. 이 기능을 사용하면 퍼블릭 구성 요소의 동작을 재정의할 수 있지만, 퍼블릭 구성 요소를 재정의하지 않으려는 경우 다른 구성 요소가 중단될 수도 있습니다.

ComponentVersion

구성 요소의 버전입니다. 메이저, 마이너 및 패치 값의 최대값은 999,999입니다.

Note

AWS IoT Greengrass 는 구성 요소에 의미 체계 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 1.0.0 버전은 구성 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

ComponentDescription

(선택 사항) 구성 요소의 설명입니다.

ComponentPublisher

구성 요소의 게시자 또는 작성자입니다.

ComponentConfiguration

(선택 사항) 구성 요소의 구성 또는 파라미터가 정의되는 객체입니다. 기본 구성을 정의한 다음에 구성 요소를 배포할 때 구성 요소에 제공할 구성 객체를 지정할 수 있습니다. 구성 요소 구성에서는 중첩된 파라미터와 구조가 지원됩니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

DefaultConfiguration

구성 요소의 기본 구성이 정의되는 객체입니다. 이 객체의 구조를 정의합니다.

Note

AWS IoT Greengrass 는 구성 값에 JSON을 사용합니다. JSON에서는 숫자 유형이 지정되지만, 정수와 부동 소수점은 구분되지 않습니다. 따라서 구성 값은 AWS IoT Greengrass에서 부동 소수점으로 변환될 수 있습니다. 구성 요소에서 올바른 데이터 유형이 사용되도록 하려면 숫자 구성 값을 문자열로 정의하는 것이 좋습니다. 그런 다음에

구성 요소에서 이를 정수 또는 부동 소수점으로 구문 분석하도록 합니다. 이렇게 하면 구성 값이 구성과 코어 디바이스에서 동일한 유형이 됩니다.

ComponentDependencies

(선택 사항) 구성 요소에 대한 구성 요소 종속성이 각각 정의되는 객체의 사전입니다. 각 객체의 키는 구성 요소 종속성의 이름을 식별합니다. AWS IoT Greengrass 는 구성 요소가 구성 요소를 시작하기 전에 종속성이 시작되도록 installs. AWS IoT Greengrass waits할 때 구성 요소 종속성을 설치합니다. 각 객체에는 다음 정보가 포함되어 있습니다.

VersionRequirement

이 종속성에 대해 호환되는 구성 요소 버전이 정의되는 npm-style 스타일 의미 체계 버전 제약 조건입니다. 버전 또는 버전 범위를 지정할 수 있습니다. 자세한 내용은 [npm 의미 체계 버전 계산을](#) 참조하세요.

DependencyType

(선택 사항) 이 종속성의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- SOFT – 종속성 상태가 변경되면 구성 요소가 다시 시작되지 않습니다.
- HARD – 종속성 상태가 변경되면 구성 요소가 다시 시작됩니다.

기본값은 HARD입니다.

ComponentType

(선택 사항) 구성 요소의 유형입니다.

Note

구성 요소를 생성할 때 recipe. AWS IoT Greengrass sets에서 구성 요소 유형을 지정하지 않는 것이 좋습니다.

유형은 다음 중 하나일 수 있습니다.

- `aws.greengrass.generic` - 구성 요소에서 명령을 실행되거나 아티팩트가 제공됩니다.
- `aws.greengrass.lambda` - 구성 요소에서 [Lambda 런처 구성 요소](#)를 통해 Lambda 함수가 실행됩니다. ComponentSource 파라미터에서는 이 구성 요소에서 실행되는 Lambda 함수의 ARN이 지정됩니다.

이 옵션은 Lambda 함수에서 구성 요소를 생성할 AWS IoT Greengrass 때에서 설정하므로 사용하지 않는 것이 좋습니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.

- `aws.greengrass.plugin` – Greengrass nucleus와 동일한 Java 가상 머신(JVM)에서 이 구성 요소가 실행됩니다. 플러그인 구성 요소를 배포하거나 다시 시작하면 Greengrass nucleus가 다시 시작됩니다.

플러그인 구성 요소에서는 Greengrass nucleus와 동일한 로그 파일이 사용됩니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

구성 요소 레시피에는이 옵션을 사용하지 않는 것이 좋습니다. Greengrass nucleus와 직접 인터페이스하는 Java로 작성된 AWS제공 구성 요소를 위한 것이기 때문입니다. 플러그인인 구성 요소에 대한 자세한 내용은 [AWS 제공 구성 요소](#) 섹션을 참조하세요.

- `aws.greengrass.nucleus - nucleus` 구성 요소입니다. 자세한 내용은 [Greengrass nucleus](#) 단원을 참조하십시오.

구성 요소 레시피에서는 이 옵션을 사용하지 않는 것이 좋습니다. 이는 AWS IoT Greengrass 코어 소프트웨어의 최소 기능이 제공되는 Greengrass nucleus 구성 요소용 옵션입니다.

기본값은 레시피에서 구성 요소를 생성할 때는 `aws.greengrass.generic`이고, Lambda 함수에서 구성 요소를 생성할 때는 `aws.greengrass.lambda`입니다.

자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

ComponentSource

(선택 사항) 구성 요소에서 실행되는 Lambda 함수의 ARN입니다.

Lambda 함수에서 구성 요소를 생성할 때 `recipe`. AWS IoT Greengrass sets에서 구성 요소 소스를 지정하지 않는 것이 좋습니다. 자세한 내용은 [AWS Lambda 함수 실행](#) 단원을 참조하십시오.

Manifests

각각 구성 요소의 수명 주기, 파라미터 및 플랫폼 요구 사항이 정의되는 객체 목록입니다. 코어 디바이스가 여러 매니페스트의 플랫폼 요구 사항과 일치하는 경우는 코어 디바이스가 일치하는 첫 번째 매니페스트를 AWS IoT Greengrass 사용합니다. 코어 디바이스가 올바른 매니페스트를 사용하도록 하려면 더 엄격한 먼저 플랫폼 요구 사항으로 매니페스트를 정의합니다. 모든 플랫폼에 적용되는 매니페스트는 목록의 마지막 매니페스트여야 합니다.

⚠ Important

구성 요소를 설치하려면 코어 디바이스가 매니페스트의 플랫폼 요구 사항 하나 이상과 일치해야 합니다. 코어 디바이스와 일치하는 매니페스트가 없으면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 설치하지 않고 배포가 실패합니다.

각 객체에는 다음 정보가 포함되어 있습니다.

Name

(선택 사항) 이 매니페스트에서 정의되는 플랫폼의 친숙한 이름입니다.

이 파라미터를 생략하면가 플랫폼 os 및에서 이름을 AWS IoT Greengrass 생성합니다architecture.

Platform

(선택 사항) 이 매니페스트가 적용되는 플랫폼이 정의되는 객체입니다. 모든 플랫폼에 적용되는 매니페스트를 정의하려면 이 파라미터를 생략합니다.

이 객체에서는 코어 디바이스가 실행되는 플랫폼에 대한 키-값 페어가 지정됩니다. 이 구성 요소를 배포하면 AWS IoT Greengrass 코어 소프트웨어는 이러한 키-값 페어를 코어 디바이스의 플랫폼 속성과 비교합니다. AWS IoT Greengrass 코어 소프트웨어는 항상 os 및를 정의architecture하며 추가 속성을 정의할 수 있습니다. Greengrass nucleus 구성요소를 배포할 때 코어 디바이스에 대한 사용자 지정 플랫폼 속성을 지정할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성 요소의 플랫폼 재정의 파라미터](#)를 참조하세요.

키-값 페어마다 다음 값 중 하나를 지정할 수 있습니다.

- 정확한 값(linux 또는 windows). 각 값은 글자 또는 숫자로 시작해야 합니다.
- 모든 값과 일치하는 *. 값이 없는 경우에도 일치합니다.
- Java 스타일 정규식(예: /windows|linux/). 정규식은 슬래시 문자(/)로 시작하고 끝나야 합니다. 예를 들어 /.+/. 정규식은 비어 있지 않은 값과 일치합니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

runtime

이 매니페스트가 지원하는 플랫폼용 [Greengrass nucleus 런타임](#)입니다. 플랫폼 runtime을 이용해 여러 매니페스트를 정의할 때 레시피에서 지원되는 런타임 값은

`aws_nucleus_lite` 및 `*`뿐입니다. 클래식 디바이스를 대상으로 할 경우, 레시피에서 런타임 필드를 지정해서는 안 됩니다. 지원되는 Greengrass Nucleus 런타임에는 다음 값이 포함됩니다.

- `*`
- `aws_nucleus_lite`

`os`

(선택 사항) 이 매니페스트에서 지원되는 플랫폼의 운영 체제 이름입니다. 공통 플랫폼에는 다음 값이 포함되어 있습니다.

- `linux`
- `windows`
- `darwin(macOS)`

`architecture`

(선택 사항) 이 매니페스트에서 지원되는 플랫폼의 프로세서 아키텍처입니다. 공통 아키텍처에는 다음 값이 포함되어 있습니다.

- `amd64`
- `arm`
- `aarch64`
- `x86`

`architecture.detail`

(선택 사항) 이 매니페스트에서 지원되는 플랫폼의 프로세서 아키텍처 세부 정보입니다. 공통 아키텍처 세부 정보에는 다음 값이 포함되어 있습니다.

- `arm61`
- `arm71`
- `arm81`

key

(선택 사항) 이 매니페스트에 대해 정의하는 플랫폼 속성입니다. 플랫폼 속성의 이름으로 `#`를 바꿉니다. AWS IoT Greengrass 코어 소프트웨어는 이 플랫폼 속성을 Greengrass nucleus 구성 요소 구성에서 지정한 키-값 페어와 일치시킵니다. 자세한 내용은 [Greengrass nucleus 구성 요소의 플랫폼 재정의 파라미터](#)를 참조하세요.

i Tip

회사 내에서 이름 충돌을 방지하려면 역 도메인 이름 형식을 사용합니다. 예를 들어 회사에서 example.com을 소유하고 라디오 프로젝트에 착수한다면 사용자 지정 플랫폼 속성 com.example.radio.RadioModule의 이름을 지정할 수 있습니다. 그러면 회사 내 플랫폼 속성 이름 충돌을 방지하는 데 도움이 됩니다.

예를 들어 플랫폼 속성(com.example.radio.RadioModule)을 정의하여 코어 디바이스에서 사용할 수 있는 라디오 모듈에 따라 상이한 매니페스트를 지정할 수 있습니다. 각 매니페스트에는 상이한 하드웨어 구성에 적용되는 상이한 아티팩트가 포함될 수 있으므로 최소 소프트웨어 세트를 코어 디바이스에 배포할 수 있습니다.

Lifecycle

이 매니페스트에서 정의되는 플랫폼에서 구성 요소가 설치되고 실행되는 방법이 정의되는 객체 또는 문자열입니다. 모든 플랫폼에 적용되는 [글로벌 수명 주기](#)를 정의할 수도 있습니다. 코어 디바이스에서는 사용되는 매니페스트에서 수명 주기가 지정되지 않는 경우에만 글로벌 수명 주기가 사용됩니다.

i Note

매니페스트 내에 이 수명 주기를 정의합니다. 여기에서 지정하는 수명 주기 단계는 매니페스트에서 정의되는 플랫폼에만 적용됩니다. 모든 플랫폼에 적용되는 [글로벌 수명 주기](#)를 정의할 수도 있습니다.

다음 정보가 이 객체 또는 문자열에 있습니다.

Setenv

(선택 사항) 모든 수명 주기 스크립트에 제공되는 환경 변수의 사전입니다. 각 수명 주기 스크립트에서 Setenv로 이러한 환경 변수를 재정의할 수 있습니다.

install

(선택 사항) 구성 요소가 설치될 때 실행되는 스크립트가 정의되는 객체 또는 문자열입니다. AWS IoT Greengrass 코어 소프트웨어는 소프트웨어가 시작될 때마다 이 수명 주기 단계도 실행합니다.

install 스크립트가 성공 코드로 종료되면 구성 요소가 INSTALLED 상태로 전환됩니다.

다음 정보가 이 객체 또는 문자열에 있습니다.

Script

실행할 스크립트입니다.

RequiresPrivilege

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 `true`로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하도록 구성된 시스템 사용자 대신이 수명 주기 스크립트를 루트로 실행합니다. 기본값은 `false`입니다.

Skipif

(선택 사항) 스크립트 실행 여부가 결정되는 확인입니다. 실행 파일이 경로에 있는지 또는 파일이 있는지 확인되도록 정의할 수 있습니다. 출력이 `true`인 경우 AWS IoT Greengrass 코어 소프트웨어는 단계를 건너뜁니다. 다음 확인 사항 중 하나를 선택합니다.

- `onpath runnable` - 시스템 경로에 실행 가능성이 있는지 확인합니다. 예를 들어, Python 3를 사용할 수 있으면 `onpath python3`를 사용하여 이 수명 주기 단계를 건너뜁니다.
- `exists file` - 파일이 있는지 확인합니다. 예를 들어, `/tmp/my-configuration.db`가 있으면 `exists /tmp/my-configuration.db`를 사용하여 이 수명 주기 단계를 건너뜁니다.

Timeout

(선택 사항) AWS IoT Greengrass 코어 소프트웨어에서 프로세스가 종료되기 전에 스크립트가 실행될 수 있는 최대 시간(초)입니다.

기본값: 120초

Setenv

(선택 사항) 스크립트에 제공되는 환경 변수의 사전입니다. `Lifecycle.Setenv`에서 제공하는 변수가 이러한 환경 변수를 통해 재정의됩니다.

run

(선택 사항) 구성 요소가 시작될 때 실행되는 스크립트가 정의되는 객체 또는 문자열입니다.

이 수명 주기 단계가 실행되면 구성 요소가 `RUNNING` 상태로 전환됩니다. `run` 스크립트가 성공 코드로 종료되면 구성 요소가 `STOPPING` 상태로 전환됩니다. `shutdown` 스크립트가 지정되면 실행됩니다. 그렇지 않으면 구성 요소가 `FINISHED` 상태로 전환됩니다.

이 구성 요소에 종속되는 구성 요소는 이 수명 주기 단계가 실행될 때 시작됩니다. 종속된 구성 요소에서 사용되는 서비스와 같은 백그라운드 프로세스를 실행하려면 그 대신에 `startup` 수명 주기 단계를 사용합니다.

수명 주기가 `run`인 구성 요소를 배포하면 이 수명 주기 스크립트가 실행되는 즉시 코어 디바이스에서 배포가 완료로 보고될 수 있습니다. 따라서 `run` 수명 주기 스크립트가 실행 직후 실패하더라도 배포를 완료되고 성공할 수 있습니다. 구성 요소의 시작 스크립트 결과에 배포 상태를 종속시키려면 그 대신에 `startup` 수명 주기 단계를 사용합니다.

Note

하나의 `startup` 또는 `run` 수명 주기만 정의할 수 있습니다.

다음 정보가 이 객체 또는 문자열에 있습니다.

Script

실행할 스크립트입니다.

RequiresPrivilege

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 `true`로 설정하면 `true` AWS IoT Greengrass 코어 소프트웨어가 이 구성 요소를 실행하도록 구성된 시스템 사용자 대신이 수명 주기 스크립트를 루트로 실행합니다. 기본값은 `false`입니다.

Skipif

(선택 사항) 스크립트 실행 여부가 결정되는 확인입니다. 실행 파일이 경로에 있는지 또는 파일이 있는지 확인되도록 정의할 수 있습니다. 출력이 `true`라면 AWS IoT Greengrass 코어 소프트웨어에서는 단계를 건너뛵니다. 다음 확인 사항 중 하나를 선택합니다.

- `onpath runnable` - 시스템 경로에 실행 가능성이 있는지 확인합니다. 예를 들어, Python 3를 사용할 수 있으면 `onpath python3`를 사용하여 이 수명 주기 단계를 건너뛵니다.
- `exists file` - 파일이 있는지 확인합니다. 예를 들어, `/tmp/my-configuration.db`가 있으면 `exists /tmp/my-configuration.db`를 사용하여 이 수명 주기 단계를 건너뛵니다.

Timeout

(선택 사항) AWS IoT Greengrass 코어 소프트웨어가 프로세스를 종료하기 전에 스크립트가 실행할 수 있는 초 단위의 최대 시간입니다.

이 수명 주기 단계에는 기본적으로 제한 시간이 없습니다. 이 제한 시간을 생략하면 종료 될 때까지 `run` 스크립트가 실행됩니다.

Setenv

(선택 사항) 스크립트에 제공되는 환경 변수의 사전입니다. `Lifecycle.Setenv`에서 제공하는 변수가 이러한 환경 변수를 통해 재정의됩니다.

startup

(선택 사항) 구성 요소가 시작될 때 실행되는 백그라운드 프로세스가 정의되는 객체 또는 문자열입니다.

종속된 구성 요소가 시작될 수 있으려면 성공적으로 종료되거나 구성 요소의 상태가 `RUNNING`으로 업데이트되어야 하는 명령을 실행하려면 `startup`을 사용합니다. 종료되지 않는 스크립트가 구성 요소에서 시작될 때 구성 요소의 상태를 `RUNNING` 또는 `ERRORED`로 설정하려면 [UpdateState](#) IPC 작업을 사용합니다. 예를 들어 `/etc/init.d/mysqld start`로 MySQL 프로세스를 시작하는 `startup` 단계를 정의할 수 있습니다.

이 수명 주기 단계가 실행되면 구성 요소가 `STARTING` 상태로 전환됩니다. `startup` 스크립트가 성공 코드로 종료되면 구성 요소가 `RUNNING` 상태로 전환됩니다. 그러면 종속된 구성 요소가 시작될 수 있습니다.

수명 주기가 `startup`인 구성 요소를 배포하면 이 수명 주기 스크립트가 종료되거나 상태가 보고된 후 코어 디바이스에서 배포가 완료로 보고될 수 있습니다. 즉, 배포의 상태는 모든 구성 요소의 시작 스크립트가 종료되거나 상태가 보고될 때까지 `IN_PROGRESS`입니다.

Note

하나의 `startup` 또는 `run` 수명 주기만 정의할 수 있습니다.

다음 정보가 이 객체 또는 문자열에 있습니다.

Script

실행할 스크립트입니다.

RequiresPrivilege

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 `true`로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 이 구성 요소를 실행하도록 구성된 시스템 사용자 대신이 수명 주기 스크립트를 루트로 실행합니다. 기본값은 `false`입니다.

Skipif

(선택 사항) 스크립트 실행 여부가 결정되는 확인입니다. 실행 파일이 경로에 있는지 또는 파일이 있는지 확인되도록 정의할 수 있습니다. 출력이 true라면 AWS IoT Greengrass 코어 소프트웨어에서는 단계를 건너뜁니다. 다음 확인 사항 중 하나를 선택합니다.

- onpath **runnable** - 시스템 경로에 실행 가능이 있는지 확인합니다. 예를 들어, Python 3를 사용할 수 있으면 **onpath python3**를 사용하여 이 수명 주기 단계를 건너뜁니다.
- exists **file** - 파일이 있는지 확인합니다. 예를 들어, /tmp/my-configuration.db가 있으면 **exists /tmp/my-configuration.db**를 사용하여 이 수명 주기 단계를 건너뜁니다.

Timeout

(선택 사항) AWS IoT Greengrass 코어 소프트웨어에서 프로세스가 종료되기 전에 스크립트가 실행될 수 있는 최대 시간(초)입니다.

기본값: 120초

Setenv

(선택 사항) 스크립트에 제공되는 환경 변수의 사전입니다. Lifecycle.Setenv에서 제공하는 변수가 이러한 환경 변수를 통해 재정의됩니다.

shutdown

(선택 사항) 구성 요소가 종료될 때 실행되는 스크립트가 정의되는 객체 또는 문자열입니다. 종료 수명 주기를 사용하여 구성 요소가 STOPPING 상태에 있을 때 실행하려는 코드를 실행합니다. 종료 수명 주기는 startup 또는 run 스크립트에서 시작된 프로세스를 중지하는 데 사용할 수 있습니다.

startup에서 백그라운드 프로세스를 시작하는 경우 구성 요소가 종료될 때 shutdown 단계를 사용하여 해당 프로세스를 중지합니다. 예를 들어 /etc/init.d/mysqld stop로 MySQL 프로세스를 중지하는 shutdown 단계를 정의할 수 있습니다.

shutdown 스크립트는 구성 요소가 STOPPING 상태로 전환된 뒤 후 실행됩니다. 스크립트가 완료되면 구성 요소가 FINISHED 상태로 전환됩니다.

다음 정보가 이 객체 또는 문자열에 있습니다.

Script

실행할 스크립트입니다.

RequiresPrivilege

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 `true`로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하도록 구성된 시스템 사용자 대신이 수명 주기 스크립트를 루트로 실행합니다. 기본값은 `false`입니다.

Skipif

(선택 사항) 스크립트 실행 여부가 결정되는 확인입니다. 실행 파일이 경로에 있는지 또는 파일이 있는지 확인되도록 정의할 수 있습니다. 출력이 `true`라면 AWS IoT Greengrass 코어 소프트웨어에서는 단계를 건너뜁니다. 다음 확인 사항 중 하나를 선택합니다.

- `onpath runnable` - 시스템 경로에 실행 가능성이 있는지 확인합니다. 예를 들어, Python 3를 사용할 수 있으면 `onpath python3`를 사용하여 이 수명 주기 단계를 건너뜁니다.
- `exists file` - 파일이 있는지 확인합니다. 예를 들어, `/tmp/my-configuration.db`가 있으면 `exists /tmp/my-configuration.db`를 사용하여 이 수명 주기 단계를 건너뜁니다.

Timeout

(선택 사항) AWS IoT Greengrass 코어 소프트웨어가 프로세스를 종료하기 전에 스크립트가 실행할 수 있는 초 단위의 최대 시간입니다.

기본값: 15초

Setenv

(선택 사항) 스크립트에 제공되는 환경 변수의 사전입니다. `Lifecycle.Setenv`에서 제공하는 변수가 이러한 환경 변수를 통해 재정의됩니다.

recover

(선택 사항) 구성 요소에서 오류가 발생할 때 실행되는 스크립트가 정의되는 객체 또는 문자열입니다.

이 단계는 구성 요소가 `ERRORED` 상태로 전환되면 실행됩니다. 구성 요소가 복구되지 않고 세 번 `ERRORED`로 전환되면 구성 요소가 `BROKEN` 상태로 변경됩니다. `BROKEN` 구성 요소를 해결하려면 다시 구성 요소를 배포해야 합니다.

다음 정보가 이 객체 또는 문자열에 있습니다.

Script

실행할 스크립트입니다.

RequiresPrivilege

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 `true`로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하도록 구성된 시스템 사용자 대신이 수명 주기 스크립트를 루트로 실행합니다. 기본값은 `false`입니다.

Skipif

(선택 사항) 스크립트 실행 여부가 결정되는 확인입니다. 실행 파일이 경로에 있는지 또는 파일이 있는지 확인되도록 정의할 수 있습니다. 출력이 `true`라면 AWS IoT Greengrass 코어 소프트웨어에서는 단계를 건너뛵니다. 다음 확인 사항 중 하나를 선택합니다.

- `onpath runnable` - 시스템 경로에 실행 가능이 있는지 확인합니다. 예를 들어, Python 3를 사용할 수 있으면 `onpath python3`를 사용하여 이 수명 주기 단계를 건너뛵니다.
- `exists file` - 파일이 있는지 확인합니다. 예를 들어, `/tmp/my-configuration.db`가 있으면 `exists /tmp/my-configuration.db`를 사용하여 이 수명 주기 단계를 건너뛵니다.

Timeout

(선택 사항) AWS IoT Greengrass 코어 소프트웨어가 프로세스를 종료하기 전에 스크립트가 실행할 수 있는 초 단위의 최대 시간입니다.

기본값: 60초

Setenv

(선택 사항) 스크립트에 제공되는 환경 변수의 사전입니다. Lifecycle.Setenv에서 제공하는 변수가 이러한 환경 변수를 통해 재정의됩니다.

bootstrap

(선택 사항) AWS IoT Greengrass 코어 소프트웨어 또는 코어 디바이스가 다시 시작되어야 하는 스크립트가 재정의되는 객체 또는 문자열입니다. 그러면 예컨대 운영 체제 업데이트 또는 런타임 업데이트 설치 후 다시 시작이 수행되는 구성 요소를 개발할 수 있습니다.

Note

AWS IoT Greengrass 코어 소프트웨어 또는 디바이스를 다시 시작할 필요가 없는 업데이트 또는 종속성을 설치하려면 [설치 수명 주기](#)를 사용합니다.

이 수명 주기 단계는 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 배포하는 다음 경우 설치 수명 주기 단계 전에 실행됩니다.

- 처음으로 구성 요소가 코어 디바이스에 배포됩니다.
- 구성 요소 변경 사항입니다.
- 구성 요소 구성 업데이트의 결과로 부트스트랩 스크립트가 변경됩니다.

AWS IoT Greengrass 코어 소프트웨어가 배포에 부트스트랩 단계가 있는 모든 구성 요소에 대해 부트스트랩 단계를 완료하면 소프트웨어가 다시 시작됩니다.

 Important

AWS IoT Greengrass 코어 소프트웨어 또는 코어 디바이스를 다시 시작하려면 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 구성해야 합니다. AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 구성하지 않으면 소프트웨어가 다시 시작되지 않습니다. 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 단원을 참조하십시오.

다음 정보가 이 객체 또는 문자열에 있습니다.

BootstrapOnRollback

 Note

이 기능이 활성화되면 BootstrapOnRollback은 실패한 대상 배포의 일부로 부트스트랩 수명 주기 단계가 완료되었거나 실행이 시도되었던 구성 요소에 대해서만 실행됩니다. 이 기능은 Greengrass nucleus 버전 2.12.0 이상에 사용할 수 있습니다.

(선택 사항) 롤백 배포의 일부로 부트스트랩 수명 주기 단계를 실행할 수 있습니다. 이 옵션을 true로 설정하면 롤백 배포 내에 정의된 부트스트랩 수명 주기 단계가 실행됩니다. 배포에 실패하면 롤백 배포 동안 구성 요소 부트스트랩 수명 주기의 이전 버전이 다시 실행됩니다.

기본값은 false입니다.

Script

실행할 스크립트입니다. 이 스크립트의 종료 코드를 통해 재시작 지침이 정의됩니다. 다음 종료 코드를 사용합니다.

- 0 - AWS IoT Greengrass 코어 소프트웨어 또는 코어 디바이스를 다시 시작하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는 모든 구성 요소가 부트스트랩된 후에도 계속 다시 시작됩니다.
- 100 - AWS IoT Greengrass 코어 소프트웨어를 다시 시작하도록 요청합니다.
- 101 - 코어 디바이스 재시작을 요청합니다.

종료 코드 100~199는 특수 동작용으로 예비되어 있습니다. 기타 종료 코드는 스크립트 오류를 나타냅니다.

RequiresPrivilege

(선택 사항) 루트 권한으로 스크립트를 실행할 수 있습니다. 이 옵션을 `true`로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하도록 구성된 시스템 사용자 대신이 수명 주기 스크립트를 루트로 실행합니다. 기본값은 `false`입니다.

Timeout

(선택 사항) AWS IoT Greengrass 코어 소프트웨어에서 프로세스가 종료되기 전에 스크립트가 실행될 수 있는 최대 시간(초)입니다.

기본값: 120초

Setenv

(선택 사항) 스크립트에 제공되는 환경 변수의 사전입니다. `Lifecycle.Setenv`에서 제공하는 변수가 이러한 환경 변수를 통해 재정의됩니다.

Selections

(선택 사항) 이 매니페스트에 대해 실행할 [글로벌 수명 주기](#)의 섹션이 지정되는 선택 키 목록입니다. 글로벌 수명 주기에서는 수준과 관계없이 선택 키로 수명 주기 단계를 정의하여 수명 주기의 하위 섹션을 선택할 수 있습니다. 그러면 코어 디바이스에서는 이 매니페스트의 선택 키와 일치하는 섹션이 사용됩니다. 자세한 내용은 [글로벌 수명 주기 예제](#)를 참조하세요.

Important

코어 디바이스에서는 이 매니페스트에서 수명 주기가 지정되지 않는 경우에만 글로벌 수명 주기의 선택 사항이 사용됩니다.

all 선택 키를 지정하여 선택 키가 없는 글로벌 수명 주기의 섹션을 실행할 수 있습니다.

Artifacts

(선택 사항) 이 매니페스트에서 정의되는 플랫폼의 구성 요소에 대한 바이너리 아티팩트가 각각 정의되는 객체 목록입니다. 예를 들어 코드 또는 이미지를 아티팩트로 정의할 수 있습니다.

구성 요소가 배포되면 AWS IoT Greengrass 코어 소프트웨어는 코어 디바이스의 폴더에 아티팩트를 다운로드합니다. 다운로드 후 소프트웨어에서 추출되는 아카이브 파일로 아티팩트를 정의할 수도 있습니다.

[레시피 변수](#)를 사용하여 아티팩트가 코어 디바이스에 설치되는 폴더의 경로를 가져올 수 있습니다.

- 일반 파일 - [레시피 변수 artifacts:path](#)를 사용하여 아티팩트가 있는 폴더의 경로를 가져옵니다. 예를 들어 레시피에 {artifacts:path}/my_script.py를 지정하여 URI s3://amzn-s3-demo-bucket/path/to/my_script.py가 있는 아티팩트의 경로를 가져옵니다.
- 추출된 아카이브 - [레시피 변수 artifacts:decompressedPath](#)를 사용하여, 추출된 아카이브 아티팩트가 있는 폴더의 경로를 가져옵니다. AWS IoT Greengrass 코어 소프트웨어는 각 아카이브를 아카이브와 동일한 이름의 폴더로 추출합니다. 예를 들어 레시피에 {artifacts:decompressedPath}/my_archive/my_script.py를 지정하여 URI s3://amzn-s3-demo-bucket/path/to/my_archive.zip가 있는 아카이브 아티팩트에서 my_script.py의 경로를 가져옵니다.

Note

로컬 코어 디바이스의 아카이브 아티팩트로 구성 요소를 개발하면 해당 아티팩트에 대한 URI가 없을 수 있습니다. 아티팩트를 추출하는 Unarchive 옵션으로 구성 요소를 테스트하려면 파일 이름이 아카이브 아티팩트 파일의 이름과 일치하는 URI를 지정합니다. 아카이브 아티팩트를 업로드하려는 URI를 지정하거나 새 자리 표시자 URI를 지정할 수 있습니다. 예를 들어, 로컬 배포 동안 my_archive.zip 아티팩트를 추출하려면 s3://amzn-s3-demo-bucket/my_archive.zip을 지정할 수 있습니다.

각 객체에는 다음 정보가 포함되어 있습니다.

Uri

S3 버킷에 있는 아티팩트의 URI입니다. AWS IoT Greengrass 코어 소프트웨어는 아티팩트가 디바이스에 이미 존재하지 않는 한 구성 요소가 설치될 때 URI에서 아티팩트를 가져옵니다. 각 아티팩트에는 각 매니페스트 내에 고유한 파일 이름이 있어야 합니다.

Unarchive

(선택 사항) 압축을 풀 아카이브의 유형입니다. 다음 옵션 중 하나를 선택합니다.

- NONE - 파일은 압축을 풀 아카이브가 아닙니다. AWS IoT Greengrass 코어 소프트웨어에서 코어 디바이스의 폴더에 아티팩트가 설치됩니다. [artifacts:path 레시피 변수](#)를 사용하여 이 폴더의 경로를 가져올 수 있습니다.
- ZIP - 파일은 ZIP 아카이브입니다. AWS IoT Greengrass 코어 소프트웨어는 아카이브와 동일한 이름의 폴더로 아카이브를 추출합니다. [artifacts:decompressedPath 레시피 변수](#)를 사용하여 이 폴더가 있는 폴더의 경로를 가져올 수 있습니다.

기본값은 NONE입니다.

Permission

(선택 사항) 이 아티팩트 파일에 대해 설정할 액세스 권한이 정의되는 객체입니다. 읽기 권한과 실행 권한을 설정할 수 있습니다.

Note

AWS IoT Greengrass 코어 소프트웨어는 구성 요소가 아티팩트 폴더의 아티팩트 파일을 편집하도록 허용하지 않으므로 쓰기 권한을 설정할 수 없습니다. 구성 요소에서 아티팩트 파일을 편집하려면 다른 위치에 복사하거나 새 아티팩트 파일을 게시하고 배포합니다.

아티팩트를 압축을 풀 아카이브로 정의하면 AWS IoT Greengrass 코어 소프트웨어는 아카이브에서 압축을 풀 파일에 대해 이러한 액세스 권한을 설정합니다. AWS IoT Greengrass 코어 소프트웨어는 폴더의 액세스 권한을 Read 및에 ALL 대해 로 설정합니다Execute. 그러면 구성 요소를 통해 폴더에서 압축 해제된 파일을 볼 수 있습니다. 아카이브의 개별 파일에 대한 권한을 설정하려면 [설치 수명 주기 스크립트](#)에서 권한을 설정할 수 있습니다.

이 객체에는 다음 정보가 포함되어 있어야 합니다.

Read

(선택 사항) 이 아티팩트 파일에 대해 설정할 읽기 권한입니다. 이 구성 요소에 종속되는 구성 요소와 같이 기타 구성 요소에 이 아티팩트 액세스를 허용하려면 ALL을 지정합니다. 다음 옵션 중 하나를 선택합니다.

- NONE – 파일을 읽을 수 없습니다.
- OWNER - 이 구성 요소를 실행하도록 구성하는 시스템 사용자가 파일을 읽을 수 있습니다.
- ALL – 모든 사용자가 파일을 읽을 수 있습니다.

기본값은 OWNER입니다.

Execute

(선택 사항) 이 아티팩트 파일에 대해 설정할 실행 권한입니다. Execute 권한에는 Read 권한이 수반됩니다. 예를 들어, ALL에 대해 Execute을 지정하면 모든 사용자가 이 아티팩트 파일을 읽고 실행할 수 있습니다.

다음 옵션 중 하나를 선택합니다.

- NONE - 파일을 실행할 수 없습니다.
- OWNER - 구성 요소를 실행하도록 구성하는 시스템 사용자가 파일을 실행할 수 있습니다.
- ALL - 모든 사용자가 파일을 실행할 수 있습니다.

기본값은 NONE입니다.

Digest

(읽기 전용) 아티팩트의 암호화 다이제스트 해시입니다. 구성 요소를 생성할 때는 해시 알고리즘을 AWS IoT Greengrass 사용하여 아티팩트 파일의 해시를 계산합니다. 그런 다음에 구성 요소를 배포할 때 Greengrass nucleus에서는 다운로드된 아티팩트의 해시가 계산되고 해시가 이 다이제스트와 비교되어 설치 전에 아티팩트가 확인됩니다. 해시가 다이제스트와 일치하지 않으면 배포에 실패합니다.

이 파라미터를 설정하면 구성 요소를 생성할 때 설정한 값이 AWS IoT Greengrass 로 바뀝니다.

Algorithm

(읽기 전용)가 아티팩트의 다이제스트 해시를 계산하는 데 AWS IoT Greengrass 사용하는 해시 알고리즘입니다.

이 파라미터를 설정하면 구성 요소를 생성할 때 설정한 값이 AWS IoT Greengrass 로 바뀝니다.

Lifecycle

구성 요소를 설치하고 실행하는 방법이 정의되는 객체입니다. 코어 디바이스에서는 사용되는 [매니페스트](#)에서 수명 주기가 지정되지 않는 경우에만 글로벌 수명 주기가 사용됩니다.

Note

매니페스트 외부에 이 수명 주기를 정의합니다. 매니페스트와 일치하는 플랫폼에 적용되는 [매니페스트 수명 주기](#)도 정의할 수 있습니다.

글로벌 수명 주기에서 각 매니페스트에서 지정하는 특정 [선택 키](#)에 대해 실행되는 수명 주기를 지정할 수 있습니다. 선택 키는 각 매니페스트에 대해 실행할 글로벌 수명 주기의 섹션이 식별되는 문자열입니다.

all 선택 키는 선택 키가 없는 섹션에 대한 기본값입니다. 즉, 선택 키가 없는 글로벌 수명 주기의 섹션이 실행되도록 매니페스트의 all 선택 키를 지정할 수 있습니다. 글로벌 수명 주기에서는 all 선택 키를 지정할 필요가 없습니다.

매니페스트에서 수명 주기 또는 선택 키가 정의되지 않으면 코어 디바이스에서는 기본적으로 all 선택이 사용됩니다. 즉, 이 경우에는 선택 키가 사용되지 않는 글로벌 수명 주기의 섹션이 코어 디바이스에서 사용됩니다.

이 객체에는 [매니페스트 수명 주기](#)와 동일한 정보가 있지만, 수준과 관계없이 선택 키를 지정하여 수명 주기의 하위 섹션을 선택할 수 있습니다.

Tip

선택 키와 수명 주기 키가 충돌하지 않도록 각 선택 키에 소문자만 사용하는 것이 좋습니다. 수명 주기 키는 대문자로 시작합니다.

Example최상위 선택 키가 있는 글로벌 수명 주기 예제

```
Lifecycle:
  key1:
    install:
      SkipIf: either onpath executable or exists file
```

```

    Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3

```

Example최하위 선택 키가 있는 글로벌 수명 주기 예제

```

Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3

```

Example여러 수준의 선택 키가 있는 글로벌 수명 주기 예제

```

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 변수에서는 레시피에서 사용할 현재 구성 요소 및 nucleus의 정보가 노출됩니다. 예를 들어 레시피 변수를 사용하여 수명 주기 스크립트에서 실행하는 애플리케이션에 구성 요소 구성 파라미터를 전달할 수 있습니다.

구성 요소 레시피의 다음 섹션에서 레시피 변수를 사용할 수 있습니다.

- 수명 주기 정의입니다.
- [Greengrass nucleus](#) v2.6.0 이상을 사용하고 [interpolateComponentConfiguration](#) 구성 옵션을 `true`로 설정하는 경우의 구성 요소 구성 정의입니다. [구성 요소 구성 업데이트 배포](#) 시 레시피 변수를 사용할 수도 있습니다.

레시피 변수에는 `{recipe_variable}` 구문이 사용됩니다. 중괄호는 레시피 변수를 나타냅니다.

AWS IoT Greengrass 는 다음 레시피 변수를 지원합니다.

`component_dependency_name:configuration:json_pointer`

이 레시피에서 정의되는 구성 요소 또는 이 구성 요소가 종속되는 구성 요소에 대한 구성 파라미터의 값입니다.

이 변수를 사용하여 구성 요소 수명 주기에서 실행하는 스크립트에 파라미터를 제공할 수 있습니다.

Note

AWS IoT Greengrass 는 구성 요소 수명 주기 정의에서만 레시피 변수를 지원합니다.

이 레시피 변수에는 다음 입력이 있습니다.

- `component_dependency_name` – (선택 사항) 쿼리할 구성 요소 종속성의 이름입니다. 이 레시피에서 정의되는 구성 요소를 쿼리하려면 이 세그먼트를 생략합니다. 직접 종속성만 지정할 수 있습니다.
- `json_pointer` – 평가할 구성 값에 대한 JSON 포인터입니다. JSON 포인터는 슬래시(/)로 시작합니다. 중첩된 구성 요소 구성에서 값을 식별하려면 슬래시(/)를 사용하여 구성의 각 수준에 대한 키를 구분합니다. 숫자를 키로 사용하여 목록에서 인덱스를 지정할 수 있습니다. 자세한 내용은 [JSON 포인터 사양](#)을 참조하세요.

AWS IoT Greengrass 코어는 YAML 형식의 레시피에 JSON 포인터를 사용합니다.

JSON 포인터에서는 다음 노드 유형이 참조될 수 있습니다.

- `value node`. AWS IoT Greengrass Core는 레시피 변수를 값의 문자열 표현으로 대체합니다. Null 값은 문자열인 `null`로 변환됩니다.
- `object node`. AWS IoT Greengrass Core는 레시피 변수를 해당 객체의 직렬화된 JSON 문자열 표현으로 바꿉니다.

- No node. AWS IoT Greengrass Core는 레시피 변수를 대체하지 않습니다.

예를 들어 {configuration:/Message} 레시피 변수에서는 구성 요소 구성에 있는 Message 키 값이 검색됩니다. {com.example.MyComponentDependency:configuration:/server/port} 레시피 변수에서는 구성 요소 종속성의 server 구성 객체port에 있는 값이 검색됩니다.

component_dependency_name:artifacts:path

이 레시피에서 정의되는 구성 요소 또는 이 구성 요소가 종속되는 구성 요소에 대한 아티팩트의 루트 경로입니다.

구성 요소가 설치되면 이 변수가 노출하는 폴더에 구성 요소의 아티팩트를 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 디렉터리(\$PWD 또는 .)와 같은 다른 위치에 복사합니다. 그런 다음에 거기에서 파일을 수정합니다.

구성 요소 종속성에서 아티팩트를 읽거나 실행하려면 해당 아티팩트 Read 또는 Execute 권한이 ALL로 되어야 합니다. 자세한 내용은 구성 요소 레시피에서 정의하는 [아티팩트 권한](#)을 참조하세요.

이 레시피 변수에는 다음 입력이 있습니다.

- component_dependency_name – (선택 사항) 쿼리할 구성 요소 종속성의 이름입니다. 이 레시피에서 정의되는 구성 요소를 쿼리하려면 이 세그먼트를 생략합니다. 직접 종속성만 지정할 수 있습니다.

component_dependency_name:artifacts:decompressedPath

이 레시피에서 정의되는 구성 요소 또는 이 구성 요소가 종속되는 구성 요소에 대한 압축 해제된 아카이브 아티팩트의 루트 경로입니다.

구성 요소가 설치되면 이 변수가 노출하는 폴더에 구성 요소의 아카이브 아티팩트를 압축 AWS IoT Greengrass 해제합니다. 예를 들어 이 변수를 사용하여 구성 요소 수명 주기에서 실행할 스크립트의 위치를 식별할 수 있습니다.

각 아티팩트는 압축명을 제외한 아티팩트와 이름이 동일한 폴더인 압축 해제된 경로 내의 폴더에 압축이 풀립니다. 예를 들어 models.zip이라는 ZIP 아티팩트는 {artifacts:decompressedPath}/models 폴더에 압축이 풀립니다.

이 경로의 폴더는 읽기 전용입니다. 아티팩트 파일을 수정하려면 파일을 현재 작업 디렉터리(\$PWD 또는 .)와 같은 다른 위치에 복사합니다. 그런 다음에 거기에서 파일을 수정합니다.

구성 요소 종속성에서 아티팩트를 읽거나 실행하려면 해당 아티팩트 Read 또는 Execute 권한이 ALL로 되어야 합니다. 자세한 내용은 구성 요소 레시피에서 정의하는 [아티팩트 권한](#)을 참조하세요.

이 레시피 변수에는 다음 입력이 있습니다.

- `component_dependency_name` – (선택 사항) 쿼리할 구성 요소 종속성의 이름입니다. 이 레시피에서 정의되는 구성 요소를 쿼리하려면 이 세그먼트를 생략합니다. 직접 종속성만 지정할 수 있습니다.

`component_dependency_name:work:path`

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.0.4 이상에서 사용할 수 있습니다.

이 레시피에서 정의되는 구성 요소 또는 이 구성 요소가 종속되는 구성 요소에 대한 작업 경로입니다. 이 레시피 변수의 값은 구성 요소의 컨텍스트에서 실행될 때 \$PWD 환경 변수의 출력 및 [pwd](#) 명령과 동일합니다.

이 레시피 변수를 사용하여 구성 요소와 종속성 사이에 파일을 공유할 수 있습니다.

이 경로의 폴더는 이 레시피에서 정의되는 구성 요소 및 동일한 사용자 및 그룹으로 실행되는 다른 구성 요소에서 읽을 수 있고 쓸 수 있습니다.

이 레시피 변수에는 다음 입력이 있습니다.

- `component_dependency_name` – (선택 사항) 쿼리할 구성 요소 종속성의 이름입니다. 이 레시피에서 정의되는 구성 요소를 쿼리하려면 이 세그먼트를 생략합니다. 직접 종속성만 지정할 수 있습니다.

`kernel:rootPath`

AWS IoT Greengrass 코어 루트 경로입니다.

`iot:thingName`

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.3.0 이상에서 사용할 수 있습니다.

코어 디바이스 사물의 이름입니다 AWS IoT .

레시피 예제

다음 레시피 예제를 참조하면 구성 요소에 대한 레시피를 생성하는 데 도움이 될 수 있습니다.

AWS IoT Greengrass 는 Greengrass 소프트웨어 카탈로그라는 Greengrass 구성 요소의 인덱스를 큐레이션합니다. 이 카탈로그에서는 Greengrass 커뮤니티에서 개발하는 Greengrass 구성 요소가 추적

됩니다. 이 카탈로그에서 구성 요소를 다운로드, 수정 및 배포하여 Greengrass 애플리케이션을 생성할 수 있습니다. 자세한 내용은 [커뮤니티 구성 요소](#) 단원을 참조하십시오.

주제

- [Hello World 구성 요소 레시피](#)
- [Python 런타임 구성 요소 예제](#)
- [여러 가지 필드가 지정되는 구성 요소 레시피](#)

Hello World 구성 요소 레시피

다음 레시피에서는 Python 스크립트가 실행되는 Hello World 구성 요소가 설명됩니다. 이 구성 요소에서는 모든 플랫폼이 지원되고 Python 스크립트에 대한 인수로 AWS IoT Greengrass 에서 전달되는 Message 파라미터가 수락됩니다. 이는 [시작하기 자습서](#)의 Hello World 구성 요소에 대한 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 구성 요소 예제

다음 레시피에서는 Python이 설치되는 구성 요소가 설명됩니다. 이 구성 요소에서는 64비트 Linux 디바이스가 지원됩니다.

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 가지 필드가 지정되는 구성 요소 레시피

다음 구성 요소 레시피에서는 여러 가지 레시피 필드가 사용됩니다.

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://amzn-s3-demo-bucket/hello_world.zip",
        "Unarchive": "ZIP"
      },
      {
        "Uri": "s3://amzn-s3-demo-bucket/hello_world_linux.py"
      }
    ]
  },
  {
    "Lifecycle": {
      "install": {
```

```

        "Skipif": "onpath git",
        "Script": "sudo apt-get install git",
        "RequiresPrivilege": "true"
    },
    "Artifacts": [
        {
            "Uri": "s3://amzn-s3-demo-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://amzn-s3-demo-bucket/hello_world.zip'
  Unarchive: ZIP
- Uri: 's3://amzn-s3-demo-bucket/hello_world_linux.py'
- Lifecycle:
  install:
    SkipIf: onpath git
    Script: sudo apt-get install git
    RequiresPrivilege: 'true'
Artifacts:
- Uri: 's3://amzn-s3-demo-bucket/hello_world.py'

```

구성 요소 환경 변수 참조

AWS IoT Greengrass 코어 소프트웨어는 구성 요소에 대한 수명 주기 스크립트를 실행할 때 환경 변수를 설정합니다. 구성 요소에서 이러한 환경 변수를 가져와 사물 이름 AWS 리전 및 Greengrass nucleus 버전을 가져올 수 있습니다. 소프트웨어에서는 [프로세스 간 통신 SDK](#)를 사용하고 [AWS 서비스와 상호 작용](#)하려면 구성 요소에 필요한 환경 변수가 설정됩니다.

구성 요소의 수명 주기 스크립트에 대한 사용자 지정 환경 변수를 설정할 수도 있습니다. 자세한 내용은 [Setenv](#)를 참조하세요.

AWS IoT Greengrass 코어 소프트웨어는 다음과 같은 환경 변수를 설정합니다.

AWS_IOT_THING_NAME

이 Greengrass 코어 디바이스를 나타내는 AWS IoT 사물의 이름입니다.

AWS_REGION

이 Greengrass 코어 디바이스가 작동하는 AWS 리전입니다.

AWS SDKs 사용하여 사용할 기본 리전을 식별합니다. 이 변수는 AWS_DEFAULT_REGION과 동일합니다.

AWS_DEFAULT_REGION

이 Greengrass 코어 디바이스가 작동하는 AWS 리전입니다.

이 환경 변수를 AWS CLI 사용하여 사용할 기본 리전을 식별합니다. 이 변수는 AWS_REGION과 동일합니다.

GGC_VERSION

이 Greengrass 코어 디바이스에서 실행되는 [Greengrass nucleus 구성 요소](#)의 버전입니다.

GG_ROOT_CA_PATH

이 기능은 [Greengrass nucleus 구성 요소](#) v2.5.5 이상에 사용할 수 있습니다.

Greengrass nucleus에서 사용되는 루트 인증 기관(CA) 인증서의 경로입니다.

AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT

구성 요소가 AWS IoT Greengrass 코어 소프트웨어와 통신하는 데 사용하는 IPC 소켓의 경로입니다. 자세한 내용은 [AWS IoT Device SDK를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core](#) 단원을 참조하십시오.

SVCUID

구성 요소가 IPC 소켓에 연결하고 AWS IoT Greengrass 코어 소프트웨어와 통신하는 데 사용하는 보안 암호 토큰입니다. 자세한 내용은 [AWS IoT Device SDK를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core](#) 단원을 참조하십시오.

AWS_CONTAINER_AUTHORIZATION_TOKEN

구성 요소에서 [토큰 교환 서비스 구성 요소](#)의 자격 증명을 검색하는 데 사용되는 비밀 토큰입니다.

AWS_CONTAINER_CREDENTIALS_FULL_URI

구성 요소에서 [토큰 교환 서비스 구성 요소](#)의 자격 증명을 검색하려고 요청하는 URI입니다.

디바이스에 AWS IoT Greengrass 구성 요소 배포

AWS IoT Greengrass를 사용하여 디바이스 또는 디바이스 그룹에 구성 요소를 배포할 수 있습니다. 배포를 사용하여 디바이스로 전송되는 구성 요소 및 구성을 정의합니다. Greengrass 코어 디바이스를 나타내는 AWS IoT Greengrass 대상, AWS IoT 사물 또는 사물 그룹에 배포합니다. [AWS IoT Core 작업을](#) AWS IoT Greengrass 사용하여 코어 디바이스에 배포합니다. 작업이 디바이스로 돌아오는 방식을 구성할 수 있습니다.

코어 디바이스 배포

각 코어 디바이스는 해당 디바이스에 대한 배포의 구성 요소를 실행합니다. 동일한 대상에 대한 새 배포는 대상의 이전 배포를 덮어씁니다. 배포를 생성할 때 코어 디바이스의 기존 소프트웨어에 적용할 구성 요소 및 구성을 정의합니다.

대상에 대한 배포를 수정하면 이전 수정의 구성 요소를 새 수정의 구성 요소로 바꿉니다. 예를 들어 [로그 관리자](#) 및 [보안 암호 관리자](#) 구성 요소를 사물 그룹 TestGroup에 배포합니다. 그런 다음 보안 암호

호 관리자 구성 요소만 지정하는 TestGroup에 대한 다른 배포를 생성합니다. 결과적으로 해당 그룹의 코어 디바이스는 더 이상 로그 관리자를 실행하지 않습니다.

플랫폼 종속성 확인

코어 디바이스는 배포를 수신하면 구성 요소가 코어 디바이스와 호환되는지 확인합니다. 예를 들어 [Firehose](#)를 Windows 대상에 배포하면 배포가 실패합니다.

구성 요소 종속성 확인

구성 요소 배포 과정에서 코어 디바이스는 사물 그룹 내 모든 구성 요소의 종속성 및 버전 요구 사항 호환성을 확인합니다. 배포를 진행하기 전에 이 확인을 통해 모든 구성 요소와 그 종속성에 대한 버전 제약 조건이 충족되는지 확인합니다.

종속성 확인 프로세스는 먼저 해당 레시피에 종속성이 없는 대상 구성 요소를 식별하는 것부터 시작됩니다. 이후 시스템은 너비 우선 탐색(BFS)을 사용하여 종속성 트리를 구성합니다. 이 과정에서는 각 대상 노드를 체계적으로 탐색하며, 해당 노드의 종속성을 먼저 찾은 후에 다음 노드로 넘어갑니다. 각 노드는 대상 구성 요소를 키(key)로, 버전 요구 사항을 값(value)으로 포함합니다.

버전 요구 사항은 다음 세 가지 제약 조건을 결합하여 구성됩니다.

- 기존 사물 그룹에 이미 설정된 버전 요구 사항
- 배포에 필요한 구성 요소 버전. 배포를 생성하거나 업데이트할 때 구성 요소 버전을 선택해야 합니다.
- 레시피의 종속성 섹션 내에 정의된 구성 요소 버전 제약 조건.

구성 요소 종속성 확인

배포 중, Greengrass nucleus는 먼저 장치에서 현재 실행 중인 구성 요소 중 요구 사항을 충족하는 로컬 후보를 찾으려고 시도합니다. 실행 중인 구성 요소가 요구 사항을 충족하면, nucleus는 레시피 폴더에서 저장된 레시피 경로를 가져오고 로컬 저장소에서 최신 로컬 버전을 찾습니다.

AWS 클라우드 배포의 경우 nucleus는 [ResolveComponentCandidates API](#)를 호출합니다. 이 API는 사용 가능한 최신 버전부터 시작하여 해당 버전이 종속성과 요구 사항을 충족하는지 확인합니다. nucleus는 API 응답을 받으면 해당 최신 버전을 선택합니다. 요구 사항을 AWS 클라우드 충족하는에서 버전을 찾을 수 없는 경우 배포가 실패합니다. 디바이스가 오프라인 상태일 경우, nucleus는 처음에 찾았던 로컬 후보로 되돌아갑니다. 요구 사항을 충족하는 로컬 후보도 없으면 배포가 실패합니다.

로컬 배포의 경우 nucleus는 로컬 후보가 존재하고 협상 없이 요구 사항을 충족하는 경우에만 로컬 후보를 사용합니다 AWS 클라우드. 그런 후보가 없으면 배포가 실패합니다.

Note

모든 확인된 레시피는 이후 참고를 위해 로컬에 저장됩니다.

자세한 내용은 GitHub의 [종속성 확인](#) 섹션을 참조하세요.

Greengrass nucleus가 모든 구성 요소를 성공적으로 확인하면, nucleus 로그에 다음과 같은 문구가 표시됩니다.

```
resolve-all-group-dependencies-finish. Finish resolving all groups dependencies.
```

Example

다음은 nucleus가 구성 요소 요구 사항을 확인하는 방식의 예시입니다.

- ComponentC 버전 1.0.0-1.9.0을 이용하는 ComponentA를 배포합니다.
- 또한 ComponentC 버전 1.4.0-1.9.5를 이용하는 ComponentB를 배포합니다.

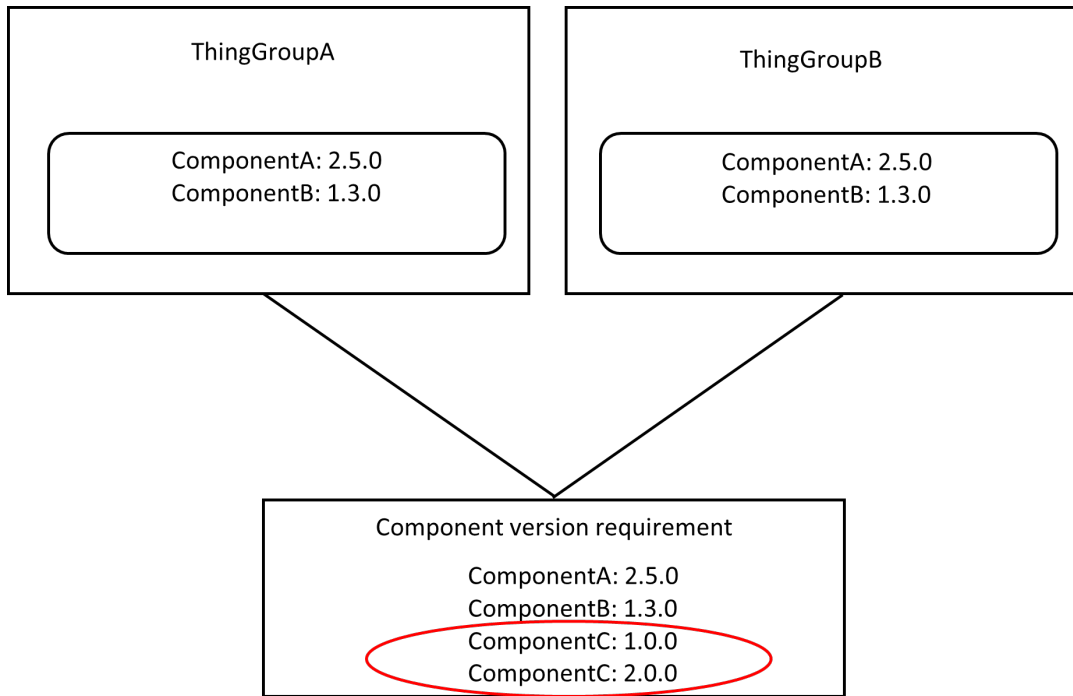
구성 요소 종속성 확인 과정에서 nucleus는 ComponentA와 ComponentB의 요구 사항을 모두 충족하는 ComponentC의 최신 버전을 배포합니다. 이 최신 버전의 ComponentC는 버전 1.9.0입니다.

일반적인 구성 요소 종속성 확인 실패

일반적인 구성 요소 종속성 확인 실패의 주된 원인 두 가지는 대상 버전 요구 사항 충돌과 구성 요소 종속성 요구 사항 충돌입니다.

시나리오 1: 대상 버전 요구 사항 충돌

- 하나의 사물이 이미 한 사물 그룹에 속해 있고 그 사물을 새 사물 그룹에 추가하려는 경우. 새 사물 그룹에서 다른 사물 버전을 요구하면 배포가 실패합니다.
- 또한 사물이 이미 사물 그룹에 속해 있는 상태에서 사물 배포를 통해 구성 요소 버전을 업데이트하려고 하는 경우에도 배포가 실패할 수 있습니다.



실패 로그 샘플:

```

2025-04-11T06:16:03.315Z [ERROR] (pool-3-thread-27)
com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
with cloud and no local version to fall back to. {componentName=ComponentC,
versionRequirement={thing/ABC==2.0.0, thinggroup/ThingGroupA==1.0.0}}
2025-04-11T06:16:03.316Z [ERROR] (pool-3-thread-26)
com.aws.greengrass.deployment.DeploymentService: Error occurred while
processing deployment. {deploymentId=fbac24de-4ef9-44b0-a685-fdc63b0f02b8,
serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentC version constraints:
thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
  
```

```

    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentC version constraints:
thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at
com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManager.
    at
com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)
    ... 4 more

```

로그에는 버전 충돌 오류가 표시됩니다. 이는 nucleus가 서로 충돌하는 두 가지 요구 사항을 동시에 충족하는 구성 요소 버전을 찾을 수 없기 때문입니다.

해결 방법

- 각 사물 그룹에 해당 구성 요소를 유지하려면 각 사물 그룹에서 동일한 버전의 구성 요소를 선택합니다.
- 배포 요구 사항을 충족하는 구성 요소 버전을 선택합니다.
- 두 사물 그룹의 요구 사항을 모두 충족하지 않는 구성 요소 버전을 사용하려면 사물 그룹 버전 요구 사항을 충족하는 구성 요소 버전을 선택하고 해당 구성 요소를 해당 사물 그룹에서만 사용합니다.

시나리오 2: 구성 요소 종속성 버전 요구 사항 충돌

하나의 구성 요소가 여러 구성 요소의 종속성이고 각 구성 요소가 해당 구성 요소의 서로 다른 버전 또는 버전 범위를 요구하는 경우, 모든 버전 요구 사항을 충족할 수 있는 사용 가능한 버전이 없을 가능성이 있습니다. 이러한 경우 배포가 실패합니다.

Example

ComponentA(v2.5.0), ComponentB(v1.3.0), ComponentC(v1.0.0)의 배포

- ComponentA에는 ComponentB 버전 1.0.0 이상이 필요합니다.

```
---
...
ComponentName: ComponentA
ComponentVersion: "2.5.0"
ComponentDependencies:
  ComponentB:
    VersionRequirement: ">=1.0.0"
    DependencyType: "HARD"
...
```

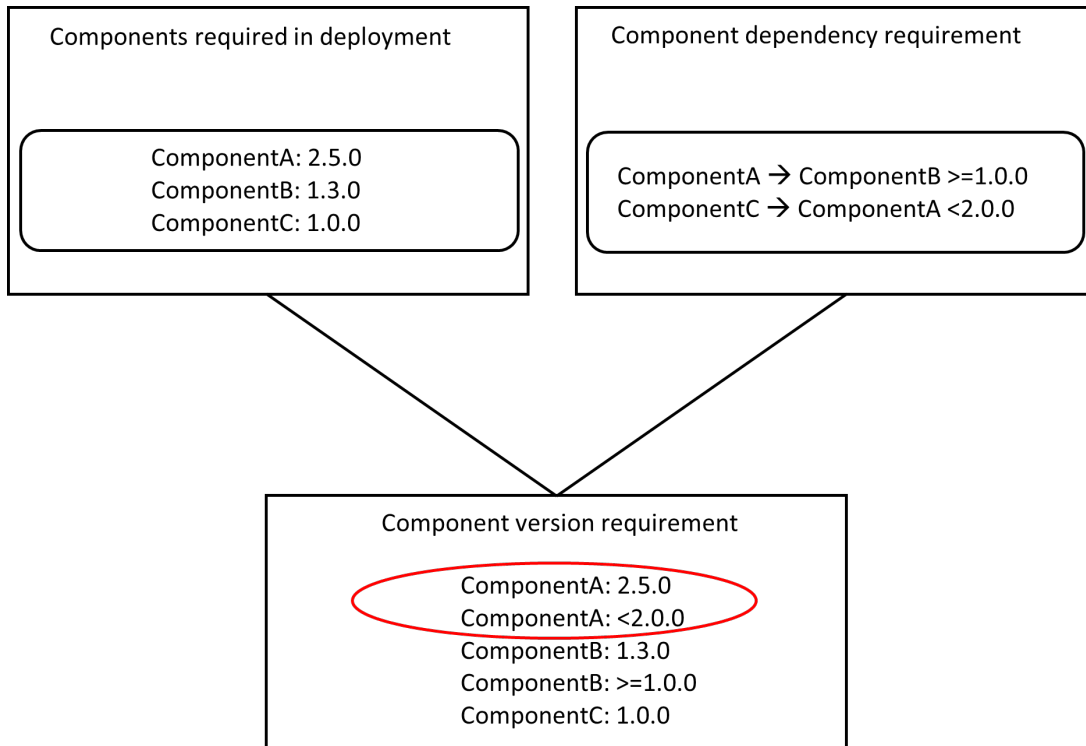
- ComponentC에는 ComponentA 버전 2.0.0 미만이 필요합니다.

```
---
...
ComponentName: ComponentC
ComponentVersion: "1.0.0"
ComponentDependencies:
  ComponentA:
    VersionRequirement: "<2.0.0"
    DependencyType: "HARD"
...
```

ComponentA에 대한 두 가지 요구 사항 간에 버전 충돌이 있습니다.

- 이번 배포에서 ComponentA에는 버전 2.5.0이 필요합니다.
- ComponentC에는 2.0.0 미만의 ComponentA 버전이 필요합니다.

이 두 요구 사항은 서로 모순되므로 nucleus가 두 요구 사항을 모두 충족하는 ComponentA 버전을 찾을 수 없습니다. 따라서 종속성 확인이 실패합니다.



실패 로그 샘플:

```

2025-04-11T06:07:18.291Z [ERROR] (pool-3-thread-25)
com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
with cloud and no local version to fall back to. {componentName=ComponentA,
versionRequirement={ComponentC=<2.0.0, thinggroup/ThingGroupA==2.5.0}}
2025-04-11T06:07:18.292Z [ERROR] (pool-3-thread-24)
com.aws.greengrass.deployment.DeploymentService: Error occurred while
processing deployment. {deploymentId=2ffac4df-1ac9-405c-8c11-28494a1b4382,
serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentA version constraints:
ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
  
```

```

    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentA version constraints:
ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at
com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManager
    at
com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)
    ... 4 more

```

로그에는 nucleus가 다음 요구 사항을 모두 충족하는 ComponentA 버전을 찾을 수 없다는 내용이 표시 됩니다.

- ComponentA가 정확히 버전 2.5.0이어야 한다는 요구 사항(ThingGroupA 기준)
- ComponentC와 호환되도록 ComponentA가 2.0.0 미만이어야 한다는 요구 사항

해결 방법

- 각 사물 그룹에 해당 구성 요소를 유지하려면 각 사물 그룹에서 동일한 버전의 구성 요소를 선택합 니다.
- 배포 요구 사항을 충족하는 구성 요소 버전을 선택합니다.
- 두 사물 그룹의 요구 사항을 모두 충족하지 않는 구성 요소 버전을 사용하려면 사물 그룹 버전 요구 사항을 충족하는 구성 요소 버전을 선택하고 해당 구성 요소를 해당 사물 그룹에서만 사용합니다.

i Tip

AWS 제공된 구성 요소에 오류가 표시되면 충돌하는 구성 요소를 최신 버전으로 업데이트하여 해결할 수 있습니다.

사물 그룹에서 디바이스 제거

사물 그룹에서 코어 디바이스를 제거할 때 구성 요소 배포 동작은 코어 디바이스가 실행하는 [Greengrass nucleus](#)의 버전에 따라 달라집니다.

2.5.1 and later

사물 그룹에서 코어 디바이스를 제거하면 AWS IoT 정책이 `greengrass:ListThingGroupsForCoreDevice` 권한을 부여하는지 여부에 따라 동작이 달라집니다. 코어 디바이스에 대한 권한 및 AWS IoT 정책에 대한 자세한 내용은 [섹션을 참조하세요](#) [요에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#).

- AWS IoT 정책이 권한을 부여하는 경우

사물 그룹에서 코어 디바이스를 제거하면 다음에 디바이스에 배포할 때 사물 그룹의 구성 요소가 AWS IoT Greengrass 제거됩니다. 디바이스의 구성 요소가 다음 배포에 포함되는 경우 해당 구성 요소는 디바이스에서 제거되지 않습니다.

- AWS IoT 정책이 권한을 부여하지 않는 경우

사물 그룹에서 코어 디바이스를 제거할 때 AWS IoT Greengrass 는 디바이스에서 해당 사물 그룹의 구성 요소를 삭제하지 않습니다.

디바이스에서 구성 요소를 제거하려면 Greengrass CLI의 [deployment create](#) 명령을 사용합니다. `--remove` 인수로 제거할 구성 요소를 지정하고 `--groupId` 인수로 사물 그룹을 지정합니다.

2.5.0

사물 그룹에서 코어 디바이스를 제거하면 다음에 디바이스에 배포할 때 사물 그룹의 구성 요소가 AWS IoT Greengrass 제거됩니다. 디바이스의 구성 요소가 다음 배포에 포함되는 경우 해당 구성 요소는 디바이스에서 제거되지 않습니다.

이 동작을 수행하려면 코어 디바이스의 AWS IoT 정책이 `greengrass:ListThingGroupsForCoreDevice` 권한을 부여해야 합니다. 코어 디바이스에 이

권한이 없는 경우 코어 디바이스는 배포를 적용하지 못합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.

2.0.x - 2.4.x

사물 그룹에서 코어 디바이스를 제거할 때 AWS IoT Greengrass 는 디바이스에서 해당 사물 그룹의 구성 요소를 삭제하지 않습니다.

디바이스에서 구성 요소를 제거하려면 Greengrass CLI의 [deployment create](#) 명령을 사용합니다. `--remove` 인수로 제거할 구성 요소를 지정하고 `--groupId` 인수로 사물 그룹을 지정합니다.

배포

배포는 지속됩니다. 배포를 생성하면가 온라인 상태인 대상 디바이스로 배포를 AWS IoT Greengrass 를아웃합니다. 대상 디바이스가 온라인 상태가 아닌 경우 다음에 연결할 때 배포를 수신합니다 AWS IoT Greengrass. 대상 사물 그룹에 코어 디바이스를 추가하면가 해당 사물 그룹에 대한 최신 배포를 디바이스에 AWS IoT Greengrass 전송합니다.

코어 디바이스는 구성 요소를 배포하기 전에 기본적으로 디바이스의 각 구성 요소에 알립니다. Greengrass 구성 요소는 알림에 응답하여 배포를 연기할 수 있습니다. 디바이스의 배터리 수준이 낮거나 중단될 수 없는 프로세스를 실행 중인 경우 배포를 연기할 수 있습니다. 자세한 내용은 [자습서: 구성 요소 업데이트가 지연되는 Greengrass 구성 요소 개발](#) 단원을 참조하십시오. 배포를 생성할 때 구성 요소에 알리지 않고 배포하도록 구성할 수 있습니다.

각 대상 사물 또는 사물 그룹은 한 번에 하나의 배포를 가질 수 있습니다. 즉, 대상에 대한 배포를 생성하면가 더 이상 해당 대상 배포의 이전 개정을 배포 AWS IoT Greengrass 하지 않습니다.

배포 옵션

배포에서는 업데이트를 수신하는 디바이스와 업데이트 배포 방법을 제어할 수 있는 몇 가지 옵션을 제공합니다. 배포를 생성할 때 다음 옵션을 구성할 수 있습니다.

- AWS IoT Greengrass 구성 요소

대상 디바이스에 설치하고 실행할 구성 요소를 정의합니다. AWS IoT Greengrass 구성 요소는 Greengrass 코어 디바이스에 배포하고 실행하는 소프트웨어 모듈입니다. 구성 요소에서 디바이스의 플랫폼을 지원하는 경우에만 디바이스에서 구성 요소를 수신합니다. 따라서 대상 디바이스가 여러 플랫폼에서 실행되더라도 디바이스 그룹에 배포할 수 있습니다. 구성 요소가 디바이스의 플랫폼을 지원하지 않는 경우 구성 요소는 디바이스에 배포되지 않습니다.

사용자 지정 구성 요소와 AWS제공 구성 요소를 디바이스에 배포할 수 있습니다. 구성 요소를 배포할 때 AWS IoT Greengrass 는 구성 요소 종속성을 식별하고 배포합니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#) 및 [AWS 제공 구성 요소](#) 섹션을 참조하세요.

각 구성 요소에 대해 배포할 버전 및 구성 업데이트를 정의합니다. 구성 업데이트는 코어 디바이스에서 구성 요소의 기존 구성을 수정하는 방법을 지정하거나 구성 요소가 코어 디바이스에 없는 경우 구성 요소의 기본 구성을 지정합니다. 기본값으로 재설정할 구성 값과 코어 디바이스에 병합할 새 구성 값을 지정할 수 있습니다. 코어 디바이스가 서로 다른 대상에 대한 배포를 수신하고 각 배포에서 호환되는 구성 요소 버전을 지정하는 경우 코어 디바이스는 배포를 생성할 때의 타임스탬프를 기반으로 구성 업데이트를 순서대로 적용합니다. 자세한 내용은 [구성 요소 구성 업데이트](#) 단원을 참조하십시오.

Important

구성 요소를 배포할 때는 해당 구성 요소의 모든 종속성에 대해 지원되는 최신 버전을 AWS IoT Greengrass 설치합니다. 따라서 사물 그룹에 새 디바이스를 추가하거나 해당 디바이스를 대상으로 하는 배포를 업데이트하면 AWS제공된 퍼블릭 구성 요소의 새 패치 버전이 코어 디바이스에 자동으로 배포될 수 있습니다. Nucleus 업데이트와 같은 일부 자동 업데이트로 인해 디바이스가 예기치 않게 다시 시작될 수 있습니다.

디바이스에서 실행 중인 구성 요소에 대한 의도하지 않은 업데이트를 방지하려면 [배포를 생성](#)할 때 해당 구성 요소의 기본 버전을 직접 포함하는 것이 좋습니다. AWS IoT Greengrass 코어 소프트웨어의 업데이트 동작에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#).

• 배포 정책

구성을 배포할 안전한 시기와 배포가 실패할 경우 수행할 작업을 정의합니다. 구성 요소가 업데이트할 수 있다고 보고할 때까지 기다릴지 여부를 지정할 수 있습니다. 디바이스에서 적용하는 배포가 실패하는 경우 디바이스를 이전 구성으로 롤백할지 여부를 지정할 수도 있습니다.

• 구성 중지

배포를 중지하는 시기와 방법을 정의합니다. 정의하는 기준이 충족되면 배포가 중지되고 실패합니다. 예를 들어, 디바이스 중 일부에서 최소 수의 디바이스가 배포를 수신한 후 해당 배포를 적용하지 못하는 경우 배포가 중지되도록 구성할 수 있습니다.

• 롤아웃 구성

배포가 대상 디바이스로 롤아웃되는 속도를 정의합니다. 최소 및 최대 속도 범위로 지수 속도 증가를 구성할 수 있습니다.

- 제한 시간 구성

각 디바이스가 배포를 적용해야 하는 최대 시간을 정의합니다. 디바이스가 지정한 기간을 초과하면 디바이스에서 배포를 적용하지 못합니다.

Important

사용자 지정 구성 요소는 S3 버킷에서 아티팩트를 정의할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 배포하면에서 구성 요소의 아티팩트가 다운로드됩니다. AWS 클라우드. 코어 디바이스 역할에서는 기본적으로 S3 버킷에 대한 액세스를 허용하지 않습니다. S3 버킷의 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스 역할에서 해당 버킷에서 아티팩트를 다운로드할 수 있는 권한을 부여해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오.

주제

- [배포 만들기](#)
- [하위 배포 생성](#)
- [배포 수정](#)
- [배포 취소](#)
- [배포 상태 확인](#)

배포 만들기

사물 또는 사물 그룹이 대상인 배포를 생성할 수 있습니다.

배포를 생성할 때 배포할 소프트웨어 구성 요소와 배포 작업이 대상 디바이스로 롤아웃되는 방식을 구성합니다. AWS CLI에 제공하는 JSON 파일에서 배포를 정의할 수 있습니다.

배포 대상에 따라 구성 요소를 실행할 디바이스가 결정됩니다. 하나의 코어 디바이스에 배포하려면 사물을 지정합니다. 여러 코어 디바이스에 배포하려면 해당 디바이스가 포함된 사물 그룹을 지정합니다. 사물 그룹을 구성하는 방법에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [정적 사물 그룹](#)과 [동적 사물 그룹](#)을 참조하세요.

이 단원의 단계에 따라 대상에 대한 배포를 생성합니다. 배포가 있는 대상에서 소프트웨어 구성 요소를 업데이트하는 방법에 대한 자세한 내용은 [배포 수정](#) 단원을 참조하세요.

Warning

[CreateDeployment](#) 작업에서는 구성 요소가 코어 디바이스에서 제거될 수 있습니다. 새 배포가 아닌 이전 배포에 구성 요소가 있으면 코어 디바이스에서는 해당 구성 요소가 제거됩니다. 구성 요소 제거를 방지하려면 먼저 [ListDeployments](#) 작업을 사용하여 배포 대상에 기존 배포가 이미 있는지 확인합니다. 그런 다음에 [GetDeployment](#) 작업을 사용하여 새 배포를 생성할 때 기존 배포에서 시작합니다.

배포를 생성하는 방법(AWS CLI)

1. deployment.json이라는 파일을 만들고 다음 JSON을 파일로 복사합니다. 배포 대상으로 지정할 AWS IoT 사물 또는 사물 그룹의 ARN으로 *targetArn*을 바꿉니다. 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 수행합니다.
 - a. 다음 명령을 실행하여 배포 대상의 배포를 나열합니다. 대상 AWS IoT 사물 또는 사물 그룹의 ARN으로 *targetArn*을 바꿉니다.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 응답이 비어 있는 경우 대상에 기존 배포가 없는 것이므로, [Step 3](#)로 건너뛸 수 있습니다. 그렇지 않으면 다음 단계에서 사용할 수 있도록 응답에서 deploymentId를 복사합니다.

Note

대상에 대한 최신 버전이 아닌 다른 배포를 수정할 수도 있습니다. 대상에 대한 모든 배포를 나열할 `--history-filter ALL` 인수를 지정합니다. 그런 다음 수정하려는 배포의 ID를 복사합니다.

- b. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. 이전 단계의 ID로 *deploymentId*를 바꿉니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

응답에는 배포의 세부 정보가 포함되어 있습니다.

- c. 이전 명령의 응답에서 다음 키-값 쌍을 임의로 `deployment.json`에 복사합니다. 새 배포에서 이러한 값을 변경할 수 있습니다.

- `deploymentName` - 배포의 이름입니다.
- `components` - 배포의 구성 요소입니다. 구성 요소를 제거하려면 이 개체에서 구성 요소를 제거하십시오.
- `deploymentPolicies` - 배포의 정책입니다.
- `iotJobConfiguration` - 배포의 작업 구성입니다.
- `tags` - 배포의 태그입니다.

3. (선택 사항) 배포의 이름을 정의합니다. 배포의 이름으로 *deploymentName*을 바꿉니다.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
}
```

4. 각 구성 요소를 추가하여 대상 디바이스를 배포합니다. 이렇게 하려면 키는 구성 요소 이름이고 값은 해당 구성 요소에 대한 세부 정보가 있는 객체인 키-값 페어를 `components` 객체에 추가합니다. 추가하는 구성 요소마다 다음 세부 정보를 지정합니다.

- `version` - 배포할 구성 요소 버전입니다.
- `configurationUpdate` - 배포할 [구성 업데이트](#)입니다. 업데이트는 각 대상 디바이스에 있는 구성 요소의 기존 구성 또는 대상 디바이스에 기존 구성이 없는 경우 구성 요소의 기본 구성이 수정되는 패치 작업입니다. 다음 구성 업데이트를 지정할 수 있습니다.

- 업데이트 재설정(reset) - (선택 사항) 대상 디바이스에서 기본값으로 재설정될 구성 값이 정의되는 JSON 포인터 목록입니다. AWS IoT Greengrass 코어 소프트웨어는 병합 업데이트를 적용하기 전에 재설정 업데이트를 적용합니다. 자세한 내용은 [재설정 업데이트](#) 섹션을 참조하세요.
- 병합 업데이트(merge) - (선택 사항) 대상 디바이스에 병합될 구성 값이 정의되는 JSON 문서입니다. JSON 문서를 문자열로 직렬화해야 합니다. 자세한 내용은 [병합 업데이트](#) 섹션을 참조하세요.
- runWith - (선택 사항) 코어 디바이스에서 이 구성 요소의 프로세스가 실행되도록 AWS IoT Greengrass 코어 소프트웨어에서 사용되는 시스템 프로세스 옵션입니다. runWith 객체에서 이 파라미터를 생략하면 [Greengrass nucleus 구성 요소](#)에서 구성하는 기본값이 AWS IoT Greengrass 코어 소프트웨어에서 사용됩니다.

다음 옵션을 임의로 지정할 수 있습니다.

- posixUser - Linux 코어 디바이스에서 이 구성 요소 실행에 사용되는 POSIX 시스템 사용자와 그룹(선택 사항)입니다. 사용자와 그룹(지정된 경우)이 각 Linux 코어 디바이스에 있어야 합니다. user:group 형식으로 사용자와 그룹을 콜론(:)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어에서는 사용자의 기본 그룹을 사용합니다. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 섹션을 참조하세요.
- windowsUser - Windows 코어 디바이스에서 이 구성 요소 실행에 사용할 Windows 사용자입니다. 사용자가 각 Windows 코어 디바이스에 있어야 하며, 해당 이름과 암호가 LocalSystem 계정의 Credentials Manager 인스턴스에 저장되어야 합니다. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 섹션을 참조하세요.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

- systemResourceLimits - 이 구성 요소의 프로세스에 적용되는 시스템 리소스 제한입니다. 기본적으로 일반 및 컨테이너화되지 않은 Lambda 구성 요소에 시스템 리소스 제한을 적용할 수 있습니다. 자세한 내용은 [구성 요소에 대한 시스템 리소스 제한 구성](#) 섹션을 참조하세요.

다음 옵션을 임의로 지정할 수 있습니다.

- cpus - 이 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 디바이스에서 이 값을 2로 설정하여 이 구성 요소 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 디바이스에서 이 값

을 0.25로 설정하여 이 구성 요소 프로세스를 CPU의 25% 사용량으로 제한할 수 있습니다. 이 값을 CPU 코어 수보다 큰 수로 설정하면 AWS IoT Greengrass 코어 소프트웨어에서는 구성 요소의 CPU 사용량을 제한하지 않습니다.

- memory - 이 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 RAM(KB 단위)입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

Example기본 구성 업데이트 예제

다음 예제 components 객체에서는 pythonVersion이라는 구성 파라미터가 예상되는 구성 요소 com.example.PythonRuntime 배포가 지정됩니다.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\": \"3.7\"}"
      }
    }
  }
}
```

Example재설정 및 병합 업데이트로 구성 업데이트 예제

기본 구성이 다음 산업 대시보드 구성 요소 예제(com.example.IndustrialDashboard)를 고찰합니다.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
```

```

    "http": 80,
    "https": 443
  },
},
"tags": []
}

```

다음 구성 업데이트에서는 다음 지침이 지정됩니다.

1. HTTPS 설정을 기본값(true)으로 재설정합니다.
2. 산업 태그의 목록을 빈 목록으로 재설정합니다.
3. 보일러 2대의 온도 및 압력 데이터 스트림이 식별되는 산업 태그 목록을 병합합니다.

```

{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
}

```

다음 components 객체 예제에서는 이 산업 대시보드 구성 요소 및 구성 업데이트 배포가 지정됩니다.

```

{
  "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) 배포에 대한 배포 정책을 정의합니다. 코어 디바이스에서 배포가 안전하게 적용될 수 있는 시점 또는 코어 디바이스에서 배포를 적용되지 못한 경우 수행할 작업을 구성할 수 있습니다. 이렇게 하려면 `deployment.json`에 `deploymentPolicies` 객체를 추가한 다음에 다음 중 하나를 수행합니다.

1. (선택 사항) 구성 요소 업데이트 정책(`componentUpdatePolicy`)을 지정합니다. 이 정책을 통해 구성 요소 업데이트가 준비될 때까지 업데이트 연기가 배포에서 허용되는지 여부가 정의됩니다. 예를 들어 구성 요소에서 업데이트 적용이 다시 시작될 수 있으려면 리소스가 정리되거나 중요한 작업이 마무리되어야 할 수 있습니다. 이 정책을 통해 구성 요소에서 업데이트 알림에 응답해야 하는 시간도 정의됩니다.

이 정책은 다음 파라미터가 있는 객체입니다.

- `action` - (선택 사항) 구성 요소에 알리고 업데이트 준비 완료 시 보고될 때까지 기다릴지 여부입니다. 다음 옵션 중 하나를 선택합니다.
 - `NOTIFY_COMPONENTS` - 배포는 해당 구성 요소를 중지하고 업데이트하기 전에 각 구성 요소에 알립니다. 구성 요소에서는 이러한 알림 수신에 [SubscribeToComponentUpdates](#) IPC 작업이 사용될 수 있습니다.
 - `SKIP_NOTIFY_COMPONENTS` - 배포는 구성 요소에 알리거나 업데이트해도 안전할 때까지 기다리지 않습니다.

기본값은 `NOTIFY_COMPONENTS`입니다.

- `timeoutInSeconds` 각 구성 요소에서 [DeferComponentUpdate](#) IPC 작업으로 업데이트 알림에 응답해야 하는 시간(초)입니다. 이 시간 내에 구성 요소에서 응답하지 않으면 코어 디바이스에서 배포가 진행됩니다.

기본값은 60초입니다.

2. (선택 사항) 구성 검증 정책(`configurationValidationPolicy`)을 지정합니다. 이 정책을 통해 배포에서 구성 업데이트를 검증하는 데 걸리는 각 구성 요소의 시간이 정의됩니다. 구성 요소에서는 [SubscribeToValidateConfigurationUpdates](#) IPC 작업을 사용하

여 자체 구성 업데이트에 대한 알림을 구독할 수 있습니다. 그런 다음에 구성 요소에서는 [SendConfigurationValidityReport](#) IPC 작업을 사용하여 구성 업데이트가 유효한지 AWS IoT Greengrass 코어 소프트웨어에 알릴 수 있습니다. 구성 업데이트가 유효하지 않으면 배포에 실패합니다.

이 정책은 다음 파라미터가 있는 객체입니다.

- `timeoutInSeconds` (선택 사항) 구성 요소에서 구성 업데이트가 검증되어야 하는 시간(초)입니다. 이 시간 내에 구성 요소에서 응답하지 않으면 코어 디바이스에서 배포가 진행됩니다.

기본값은 30초입니다.

3. (선택 사항) 장애 처리 정책(`failureHandlingPolicy`)을 지정합니다. 이 정책은 배포에 실패할 경우 디바이스 롤백 여부가 정의되는 문자열입니다. 다음 옵션 중 하나를 선택합니다.

- `ROLLBACK` - 코어 디바이스에서 배포에 실패하면 AWS IoT Greengrass 코어 소프트웨어에서는 해당 코어 디바이스가 이전 구성으로 롤백됩니다.
- `DO_NOTHING` - 코어 디바이스에서 배포에 실패하면 AWS IoT Greengrass 코어 소프트웨어에서는 새 구성이 유지됩니다. 따라서 새 구성이 유효하지 않으면 구성 요소가 손상될 수 있습니다.

기본값은 `ROLLBACK`입니다.

`deployment.json`의 배포는 다음 예제와 비슷하게 보일 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) 배포의 중지, 롤아웃 또는 제한 시간을 정의합니다. AWS IoT Greengrass에서는 코어 디바이스로 배포를 보내는 데 AWS IoT Core 작업이 사용되므로 이러한 옵션은 AWS IoT Core 작업의 구성 옵션과 동일합니다. 자세한 내용은 AWS IoT 개발자 안내서의 [작업 롤아웃 및 중단 구성](#)을 참조하세요.

작업 옵션을 정의하려면 deployment.json에 iotJobConfiguration 객체를 추가합니다. 그런 다음에 구성할 옵션을 정의합니다.

deployment.json의 배포는 다음 예제와 비슷하게 보일 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 대한 태그(tags)를 추가합니다. 자세한 내용은 [AWS IoT Greengrass Version 2 리소스에 태그 지정](#) 섹션을 참조하세요.
8. 다음 명령을 실행하여 deployment.json에서 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

이 배포가 식별되는 deploymentId가 응답에 포함되어 있습니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 [배포 상태 확인](#) 섹션을 참조하세요.

구성 요소 구성 업데이트

구성 요소 구성은 각 구성 요소에 대한 파라미터를 정의하는 JSON 객체입니다. 각 구성 요소의 레시피는 구성 요소를 코어 디바이스에 배포하는 경우에 수정하는 기본 구성을 정의합니다.

배포를 생성할 때 각 구성 요소에 적용할 구성 업데이트를 지정할 수 있습니다. 구성 업데이트는 패치 작업입니다. 이는 업데이트가 코어 디바이스에 존재하는 구성 요소 구성을 수정함을 의미합니다. 코어 디바이스에 구성 요소가 없는 경우 구성 업데이트는 해당 배포에 대한 기본 구성을 수정하고 적용합니다.

구성 업데이트는 재설정 업데이트와 병합 업데이트를 정의합니다. 재설정 업데이트는 어떤 구성 값을 기본값으로 재설정하거나 제거할지 정의합니다. 병합 업데이트는 구성 요소에 대해 설정할 새 구성 값을 정의합니다. 구성 업데이트를 배포하면 AWS IoT Greengrass 코어 소프트웨어가 병합 업데이트 전에 재설정 업데이트를 실행합니다.

구성 요소는 배포하는 구성 업데이트를 검증할 수 있습니다. 구성 요소는 배포가 구성을 변경할 때 알림을 수신하도록 구독하며 지원하지 않는 구성은 거부할 수 있습니다. 자세한 내용은 [구성 요소 구성과 연동](#) 단원을 참조하십시오.

주제

- [재설정 업데이트](#)
- [병합 업데이트](#)
- [예제](#)

재설정 업데이트

재설정 업데이트는 코어 디바이스에서 어떤 구성 값을 기본값으로 재설정할지 정의합니다. 구성 값에 기본값이 없는 경우 재설정 업데이트는 구성 요소의 구성에서 해당 값을 제거합니다. 이는 잘못된 구성으로 인해 중단되는 구성 요소를 수정하는 데 도움이 될 수 있습니다.

JSON 포인터 목록을 사용하여 어떤 구성 값을 재설정할지 정의합니다. JSON 포인터는 슬래시(/)로 시작합니다. 중첩된 구성 요소 구성에서 값을 식별하려면 슬래시(/)를 사용하여 구성의 각 수준에 대한 키를 구분하세요. 자세한 내용은 [JSON 포인터 사양](#)을 참조하세요.

Note

전체 목록만 기본값으로 재설정할 수 있습니다. 재설정 업데이트를 사용하여 목록의 개별 요소를 재설정할 수 없습니다.

구성 요소의 전체 구성을 기본값으로 재설정하려면 빈 문자열 하나를 재설정 업데이트로 지정합니다.

```
"reset": [""]
```

병합 업데이트

병합 업데이트는 코어의 구성 요소 구성에 삽입할 구성 값을 정의합니다. 병합 업데이트는 재설정 업데이트에서 지정한 경로의 값을 재설정 후 AWS IoT Greengrass 코어 소프트웨어가 병합하는 JSON 객체입니다. AWS CLI 또는 AWS SDKs를 사용하는 경우이 JSON 객체를 문자열로 직렬화해야 합니다.

구성 요소의 기본 구성에 없는 키-값 페어를 병합할 수 있습니다. 동일한 키가 포함된 값과 다른 유형의 키-값 페어를 병합할 수도 있습니다. 그러면 기존 값이 새 값으로 바뀝니다. 이는 구성 객체의 구조를 변경할 수 있음을 의미합니다.

Null 값과 빈 문자열, 목록, 객체를 병합할 수 있습니다.

Note

목록에 요소를 삽입하거나 추가하기 위한 목적으로 병합 업데이트를 사용할 수 없습니다. 전체 목록을 바꾸거나 각 요소에 고유한 키가 있는 객체를 정의할 수 있습니다.

AWS IoT Greengrass 는 구성 값에 JSON을 사용합니다. JSON에서는 숫자 유형이 지정되지만, 정수와 부동 소수점은 구분되지 않습니다. 따라서 구성 값은 AWS IoT Greengrass에서 부동 소수점으로 변환될 수 있습니다. 구성 요소에서 올바른 데이터 유형이 사용되도록 하려면 숫자 구성 값을 문자열로 정의하는 것이 좋습니다. 그런 다음에 구성 요소에서 이를 정수 또는 부동 소수점으로 구문 분석하도록 합니다. 이렇게 하면 구성 값이 구성과 코어 디바이스에서 동일한 유형이 됩니다.

병합 업데이트에서 레시피 변수 사용

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다.

Greengrass nucleus의 [interpolateComponentConfiguration](#) 구성 옵션을 true로 설정한 경우 병합 업데이트에서 `component_dependency_name:configuration:json_pointer` 레시피 변수 이외의 레시피 변수를 사용할 수 있습니다. 예를 들어 병합 업데이트에서 {iot:thingName} 레시피 변수를 사용하여 [프로세스 간 통신\(IPC\) 권한 부여 정책과](#) 같은 구성 요소 구성 값에 코어 디바이스의 AWS IoT 사물 이름을 포함할 수 있습니다.

예제

다음 예제는 다음 기본 구성이 있는 대시보드 구성 요소에 대한 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
```

Example예제 1: 병합 업데이트

병합 업데이트는 지정하지만 재설정 업데이트는 지정하지 않는 다음 구성 업데이트를 적용하는 배포를 생성합니다. 이 구성 업데이트는 구성 요소에 두 보일러의 데이터가 포함된 HTTP 포트 8080에서 대시보드를 표시하도록 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 명령은 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file:///dashboard-deployment.json
```

dashboard-deployment.json 파일에는 다음 JSON 문서가 포함되어 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
```

dashboard-configuration.json 파일에는 다음 JSON 문서가 포함되어 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 요소의 구성은 다음과 같습니다.

```
{
  "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"
  ]
}
```

Example예제 2: 재설정 및 병합 업데이트

그런 다음 재설정 업데이트와 병합 업데이트를 지정하는 다음 구성 업데이트를 적용하는 배포를 생성합니다. 해당 업데이트는 다른 보일러의 데이터와 포함된 대시보드를 기본 HTTPS 포트에 표시하도록 지정합니다. 이러한 업데이트는 이전 예제의 구성 업데이트에서 발생하는 구성을 수정합니다.

Console

재설정 경로

```
[
  "/network/useHttps",
  "/tags"
]
```

병합할 구성

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

다음 명령은 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

dashboard-deployment2.json 파일에는 다음 JSON 문서가 포함되어 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
```

dashboard-configuration2.json 파일에는 다음 JSON 문서가 포함되어 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 요소의 구성은 다음과 같습니다.

```
{  
  "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

하위 배포 특성은 Greengrass nucleus 버전 2.9.0 이상에서 사용할 수 있습니다. Greengrass nucleus의 이전 구성 요소 버전을 사용하여 하위 배포에 구성을 배포할 수는 없습니다.

하위 배포는 상위 배포 내 더 작은 디바이스 하위 집합이 대상이 되는 배포입니다. 하위 배포를 사용하여 더 작은 디바이스 하위 집합에 구성을 배포할 수 있습니다. 상위 배포에서 하나 이상의 디바이스가 실패할 때 성공하지 못한 해당 상위 배포를 다시 시도할 하위 배포를 생성할 수도 있습니다. 이 특성으로 해당 상위 배포에서 실패한 디바이스를 선택하고 하위 배포를 생성하여 하위 배포가 성공할 때까지 구성을 테스트할 수 있습니다. 하위 배포에 성공하면 해당 구성을 상위 배포에 다시 배포할 수 있습니다.

이 단원의 단계에 따라 하위 배포를 생성하고 상태를 확인합니다. 배포를 생성하는 방법에 대한 자세한 내용은 [배포 생성](#)을 참조하세요.

하위 배포를 생성하는 방법(AWS CLI)

1. 다음 명령을 실행하여 사물 그룹에 대한 최신 배포를 검색합니다. 쿼리할 사물 그룹의 ARN으로 명령의 ARN을 바꿉니다. `--history-filter`를 **LATEST_ONLY**로 설정하여 해당 사물 그룹의 최신 배포를 참조합니다.

```
aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
```

2. 응답 중에서 deploymentId를 다음 단계에서 사용할 list-deployments 명령에 복사합니다.
3. 다음 명령을 실행하여 배포의 상태를 검색합니다. 쿼리할 배포의 ID로 *deploymentId*를 바꿉니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

4. 응답 중에서 iotJobId를 다음 단계에서 사용할 get-deployment 명령에 복사합니다.

5. 다음 명령을 실행하여 지정된 작업에 대한 작업 실행 목록을 검색합니다. 이전 단계의 `iotJobId`로 *jobID*를 바꿉니다. 필터링하려는 상태로 *##*를 바꿉니다. 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 대해 새 AWS IoT 사물 그룹을 생성하거나 기존 사물 그룹을 사용합니다. 그런 다음에 AWS IoT 사물을 이 사물 그룹에 추가합니다. 사물 그룹을 사용하여 Greengrass 코어 디바이스 플릿을 관리합니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포가 있는 사물 그룹에 디바이스를 추가할 수 있습니다. 추가하면 해당 사물 그룹의 소프트웨어 구성 요소를 해당 디바이스에 배포할 수 있습니다.

새 사물 그룹을 생성하고 디바이스를 추가하려면 다음을 수행합니다.

- a. AWS IoT 사물 그룹을 생성합니다. 새 사물 그룹의 이름으로 *MyGreengrassCoreGroup*을 바꿉니다. 사물 그룹 이름에 콜론(:)은 사용할 수 없습니다.

 Note

하나의 `parentTargetArn`으로 하위 배포의 사물 그룹을 사용하는 경우 다른 상위 플릿에서 재사용할 수 없습니다. 사물 그룹이 이미 다른 플릿에 대한 하위 배포 생성에 사용된 경우 API에서 오류가 반환됩니다.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 응답은 다음 예제와 비슷하게 보입니다.

```
{
  "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 명령을 이러한 파라미터와 함께 실행합니다.

- 프로비저닝된 Greengrass 코어의 이름으로 *MyGreengrassCore*를 바꿉니다.
- 사물 그룹의 이름으로 *MyGreengrassCoreGroup*를 바꿉니다.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 명령에 출력이 없습니다.

7. `deployment.json`이라는 파일을 만들고 다음 JSON을 파일로 복사합니다. 하위 배포 대상으로 지정할 AWS IoT 사물 그룹의 ARN으로 *targetArn*을 바꿉니다. 사물 그룹만 하위 배포 대상이 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 원래 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. [Step 1](#)의 ID로 *deploymentId*를 바꿉니다. 이 배포 구성을 사용하여 하위 배포를 구성하고 필요에 따라 변경할 수 있습니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

응답에는 배포의 세부 정보가 포함되어 있습니다. `get-deployment` 명령의 응답에서 다음 키-값 쌍을 임의로 `deployment.json`에 복사합니다. 하위 배포에 대한 이러한 값을 변경할 수 있습니다. 이 명령의 세부 정보에 대한 자세한 내용은 [GetDeployment](#)를 참조하세요.

- `components` - 배포의 구성 요소입니다. 구성 요소를 제거하려면 이 개체에서 구성 요소를 제거하십시오.
- `deploymentName` - 배포의 이름입니다.
- `deploymentPolicies` - 배포의 정책입니다.
- `iotJobConfiguration` - 배포의 작업 구성입니다.
- `parentTargetArn` - 상위 배포의 대상입니다.
- `tags` - 배포의 태그입니다.

9. 다음 명령을 실행하여 `deployment.json`에서 하위 배포를 생성합니다. 하위 배포의 이름으로 *subdeploymentName*을 바꿉니다.

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-json file:///deployment.json
```

이 하위 배포가 식별되는 `deploymentId`가 응답에 포함되어 있습니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 [배포 상태 확인](#)을 참조하세요.

10. 하위 배포에 성공하면 해당 구성을 사용하여 상위 배포를 수정할 수 있습니다. 이전 단계에서 사용한 `deployment.json`을 복사합니다. 상위 배포의 ARN으로 JSON 파일의 `targetArn`을 바꾸고 다음 명령을 실행하여 이 새 구성으로 상위 배포를 생성합니다.

Note

상위 플릿의 새 배포 수정을 생성하면 해당 상위 배포의 모든 배포 수정 및 하위 배포가 바뀝니다. 자세한 내용은 [배포 수정](#)을 참조하세요.

```
aws greengrassv2 create-deployment --cli-input-json file:///deployment.json
```

이 배포가 식별되는 `deploymentId`가 응답에 포함되어 있습니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 [배포 상태 확인](#) 섹션을 참조하세요.

배포 수정

각 대상 사물 또는 사물 그룹은 한 번에 하나의 활성 배포를 가질 수 있습니다. 이미 배포가 있는 대상에 대한 배포를 생성하면 새 배포의 소프트웨어 구성 요소가 이전 배포의 소프트웨어 구성 요소를 대체합니다. 새 배포가 이전 배포가 정의한 구성 요소를 정의하지 않는 경우 AWS IoT Greengrass 코어 소프트웨어는 대상 코어 디바이스에서 해당 구성 요소를 제거합니다. 기존 배포를 수정하여 코어 디바이스에서 실행되는 구성 요소를 이전 배포에서 대상으로 제거하지 않도록 할 수 있습니다.

배포를 수정하려면 이전 배포에 있는 것과 동일한 구성 요소와 구성에서 시작하는 배포를 생성합니다. [배포를 생성](#)하는 데 사용하는 것과 동일한 [CreateDeployment](#) 작업을 사용합니다.

배포를 수정하려면(AWS CLI)

1. 다음 명령을 실행하여 배포 대상의 배포를 나열합니다. 대상 AWS IoT 사물 또는 사물 그룹의 ARN으로 *targetArn*을 바꿉니다.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 다음 단계에서 사용할 *deploymentId*를 응답에서 복사합니다.

Note

대상에 대한 최신 버전이 아닌 다른 배포를 수정할 수도 있습니다. 대상에 대한 모든 배포를 나열할 `--history-filter ALL` 인수를 지정합니다. 그런 다음 수정하려는 배포의 ID를 복사합니다.

2. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. 이전 단계의 ID로 *deploymentId*를 바꿉니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

응답에는 배포의 세부 정보가 포함되어 있습니다.

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`

[CreateDeployment](#) 작업은 다음 구조의 페이로드를 예상합니다.

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. `deployment.json`에서 다음을 수행합니다.

- 배포의 이름(`deploymentName`)을 변경합니다.
- 배포의 구성 요소(`components`)를 변경합니다.
- 배포의 정책(`deploymentPolicies`)을 변경합니다.
- 배포의 작업 구성(`iotJobConfiguration`)을 변경합니다.
- 배포의 태그(`tags`)를 변경합니다.

이러한 배포 세부 정보를 정의하는 방법에 대한 자세한 내용은 [배포 만들기](#) 섹션을 참조하세요.

6. 다음 명령을 실행하여 `deployment.json`에서 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

이 배포가 식별되는 `deploymentId`가 응답에 포함되어 있습니다. 배포 ID를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 [배포 상태 확인](#) 섹션을 참조하세요.

배포 취소

활성 배포를 취소하여 해당 소프트웨어 구성 요소가 AWS IoT Greengrass 코어 디바이스에 설치되지 않도록 할 수 있습니다. 대상이 사물 그룹인 배포를 취소하면 그룹에 추가하는 코어 디바이스에서는 해

당 지속적 배포를 받지 않습니다. 코어 디바이스에서 이미 배포가 실행되는 경우에 배포를 취소하면 해당 디바이스의 구성 요소는 변경되지 않습니다. 취소된 배포를 받은 코어 디바이스에서 실행되는 구성 요소를 수정하려면 [새 배포를 생성](#)하거나 [배포를 수정](#)해야 합니다.

배포를 취소하는 방법(AWS CLI)

1. 다음 명령을 실행하여 대상에 대한 최신 배포 수정의 ID를 찾습니다. 새 수정을 생성할 때 이전 배포가 취소되므로 최신 수정이 대상에 대해 활성화될 수 있는 유일한 배포입니다. 대상 AWS IoT 사물 또는 사물 그룹의 ARN으로 *targetArn*을 바꿉니다.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 다음 단계에서 사용할 *deploymentId*를 응답에서 복사합니다.

2. 다음 명령을 실행하여 배포를 취소합니다. 이전 단계의 ID로 *deploymentId*를 바꿉니다.

```
aws greengrassv2 cancel-deployment --deployment-id deploymentId
```

작업에 성공하면 배포 상태가 CANCELED로 변경됩니다.

배포 상태 확인

AWS IoT Greengrass에서 생성하는 배포의 상태를 확인할 수 있습니다. 각 코어 디바이스에 배포를 롤아웃하는 AWS IoT 작업의 상태를 확인할 수도 있습니다. 배포가 활성화된 동안 AWS IoT 작업의 상태는 IN_PROGRESS입니다. 배포의 새 수정을 생성하면 이전 수정의 AWS IoT 작업 상태가 CANCELLED로 변경됩니다.

주제

- [배포 상태 확인](#)
- [디바이스 배포 상태 확인](#)

배포 상태 확인

대상 또는 ID로 식별하는 배포의 상태를 확인할 수 있습니다.

대상별 배포 상태 확인 방법(AWS CLI)

- 다음 명령을 실행하여 대상에 대한 최신 배포의 상태를 검색합니다. 배포의 대상인 AWS IoT 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)으로 *targetArn*을 바꿉니다.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 이 배포 객체에는 배포의 상태가 포함되어 있습니다.

ID별 배포 상태 확인 방법(AWS CLI)

- 다음 명령을 실행하여 배포의 상태를 검색합니다. 쿼리할 배포의 ID로 *deploymentId*를 바꿉니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

배포의 상태가 응답에 있습니다.

디바이스 배포 상태 확인

개별 코어 디바이스에 적용되는 배포 작업의 상태를 확인할 수 있습니다. 사물 그룹 배포에 대한 배포 작업의 상태를 확인할 수도 있습니다.

코어 디바이스에 대한 배포 작업 상태를 확인하는 방법(AWS CLI)

- 다음 명령을 실행하여 코어 디바이스에 대한 모든 배포 작업의 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 *coreDeviceName*을 바꿉니다.

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

응답에 코어 디바이스에 대한 배포 작업의 목록이 포함되어 있습니다. *deploymentId* 또는 *targetArn*으로 배포에 대한 작업을 식별할 수 있습니다. 코어 디바이스의 작업 상태가 각 배포 작업에 있습니다.

사물 그룹의 배포 상태를 확인하는 방법(AWS CLI)

1. 다음 명령을 실행하여 기존 배포의 ID를 검색합니다. 대상 사물 그룹의 ARN으로 *targetArn*을 바꿉니다.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 다음 단계에서 사용할 *deploymentId*를 응답에서 복사합니다.

Note

대상에 대한 최신 배포 이외의 배포를 나열할 수도 있습니다. 대상에 대한 모든 배포를 나열할 `--history-filter ALL` 인수를 지정합니다. 그런 다음에 상태를 확인하려는 배포의 ID를 복사합니다.

2. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. 이전 단계의 ID로 *deploymentID*를 바꿉니다.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

배포에 대한 정보가 응답에 있습니다. 다음 단계에서 사용할 *iotJobId*를 응답에서 복사합니다.

3. 배포에 대한 코어 디바이스의 작업 실행이 설명되는 다음 명령을 실행합니다. 이전 단계의 작업 ID와 상태를 확인하려는 코어 디바이스로 *iotJobId*와 *coreDeviceThingName*을 바꿉니다.

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

코어 디바이스의 배포 작업 실행 및 상태에 대한 세부 정보가 응답에 있습니다. 다음과 같은 정보가 *detailsMap*에 있습니다.

- `detailed-deployment-status` - 다음 값 중 하나일 수 있는 배포 결과 상태입니다.
 - `SUCCESSFUL` - 배포에 성공했습니다.
 - `FAILED_NO_STATE_CHANGE` - 코어 디바이스에서 배포를 적용할 준비가 되는 동안 배포에 실패했습니다.
 - `FAILED_ROLLBACK_NOT_REQUESTED` - 배포에 실패했으며, 이전 작업 구성으로 롤백하도록 배포에서 지정되지 않아서 코어 디바이스가 올바르게 작동하지 않을 수 있습니다.

- **FAILED_ROLLBACK_COMPLETE** - 배포에 실패했으며, 코어 디바이스가 이전 작업 구성으로 롤백되었습니다.
- **FAILED_UNABLE_TO_ROLLBACK** - 배포에 실패했으며, 코어 디바이스에서 이전 작업 구성으로 롤백하는 데 실패하여 코어 디바이스가 올바르게 작동하지 않을 수 있습니다.

배포에 실패하면 `deployment-failure-cause` 값과 코어 디바이스의 로그 파일을 확인하여 문제를 식별합니다. 코어 디바이스의 로그 파일에 액세스하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하세요.

- `deployment-failure-cause` - 작업 실행에 실패한 이유에 대한 추가 세부 정보가 제공되는 오류 메시지입니다.

응답 모양은 다음 예제와 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 Greengrass에서 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](#)
- [AWS IoT Greengrass 로그 모니터링](#)
- [를 사용하여 AWS IoT Greengrass V2 API 호출 로깅 AWS CloudTrail](#)
- [AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집](#)
- [배포 및 구성 요소 상태 알림 받기](#)
- [Greengrass 코어 디바이스 상태 확인](#)

모니터링 도구

AWS는 를 모니터링하는 데 사용할 수 있는 도구를 제공합니다. AWS IoT Greengrass 이러한 도구 중 일부를 구성하여 모니터링을 수행할 수 있습니다. 일부 도구는 수동 개입이 필요합니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

다음과 같은 자동 모니터링 도구를 사용하여 AWS IoT Greengrass를 모니터링하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch Logs – AWS CloudTrail 또는 기타 소스의 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 [로그 파일 모니터링](#)을 참조하세요.

- AWS CloudTrail 로그 모니터링 - 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs에 전송하여 실시간으로 모니터링하며, Java에서 로그 처리 애플리케이션을 작성하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 [CloudTrail 로그 파일 작업](#)을 참조하십시오.
- Greengrass 시스템 상태 텔레메트리 — Greengrass 코어에서 전송된 텔레메트리 데이터를 수신하려면 구독하십시오. 자세한 내용은 [the section called “시스템 상태 원격 측정 데이터 수집”](#) 섹션을 참조하세요.
- 디바이스 상태 알림 배포 및 구성 요소에 대한 상태 업데이트를 수신하려면 Amazon EventBridge를 사용하여 이벤트를 생성합니다. 자세한 내용은 [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 Greengrass 콘솔에서도 플릿 상태 정보를 볼 수 있습니다. 자세한 내용은 [Greengrass 코어 디바이스 상태 확인](#) 섹션을 참조하세요.

AWS IoT Greengrass 로그 모니터링

AWS IoT Greengrass은(는) 클라우드 서비스와 AWS IoT Greengrass 코어 소프트웨어로 구성됩니다. AWS IoT Greengrass 코어 소프트웨어는 Amazon CloudWatch Logs 및 코어 장치의 로컬 파일 시스템에 로그를 기록할 수 있습니다. 코어 디바이스에서 실행되는 Greengrass 구성 요소는 CloudWatch Logs 및 로컬 파일 시스템에 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 코어 소프트웨어는 로컬 파일 시스템에만 로그를 기록합니다. 파일 시스템 로그를 실시간으로 볼 수 있으므로 개발 및 배포하는 Greengrass 구성 요소를 디버깅할 수 있습니다. 또한 코어 디바이스가 CloudWatch Logs에 로그를 기록하도록 구성하여 로컬 파일 시스템에 액세스하지 않고도 코어 디바이스의 문제를 해결할 수도 있습니다. 자세한 내용은 [CloudWatch Logs에 로깅 활성화](#) 섹션을 참조하세요.

주제

- [파일 시스템 로그 액세스](#)
- [CloudWatch Logs 액세스](#)
- [시스템 서비스 로그 액세스](#)
- [CloudWatch Logs에 로깅 활성화](#)
- [AWS IoT Greengrass의 로깅 구성](#)

- [AWS CloudTrail 로그](#)

파일 시스템 로그 액세스

AWS IoT Greengrass 코어 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 코어 소프트웨어 로그 파일입니다. 이 로그 파일을 사용하면 구성 요소 및 배포에 대한 실시간 정보를 볼 수 있습니다. 이 로그 파일에는 AWS IoT Greengrass 코어 소프트웨어의 코어인 Greengrass nucleus에 대한 로그와 [로그 관리자](#) 및 [보안 암호 관리자](#)와 같은 플러그인 구성 요소가 포함됩니다.
- `ComponentName.log` – Greengrass 구성 요소 로그 파일입니다. 구성 요소 로그 파일을 사용하면 코어 디바이스에서 실행되는 Greengrass 구성 요소에 대한 실시간 정보를 볼 수 있습니다. 일반 구성 요소와 Lambda 구성 요소에서는 이러한 로그 파일에 표준 출력(stdout)과 표준 오류(stderr)를 기록합니다.
- `main.log` - 구성 요소 수명 주기를 처리하는 main 서비스에 대한 로그 파일입니다. 이 로그 파일은 항상 비어 있습니다.

플러그인, 일반 및 Lambda 구성 요소의 차이점에 대한 자세한 내용은 [구성 요소 유형](#) 섹션을 참조하세요.

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 교체

AWS IoT Greengrass 코어 소프트웨어에서는 매시간 또는 파일 크기 제한이 초과될 때 로그 파일을 교체합니다. 교체된 로그 파일의 파일 이름에는 타임스탬프가 포함됩니다. 예를 들어 교체된 AWS IoT Greengrass 코어 소프트웨어 로그 파일의 이름은 `greengrass_2021_09_14_15_0.log`일 수 있습니다. 기본 파일 크기 제한은 1,024KB(1MB)입니다. [Greengrass nucleus 구성 요소](#)에서 파일 크기 제한을 구성할 수 있습니다.

- 로그 파일 삭제

AWS IoT Greengrass 코어 소프트웨어는 교체된 로그 파일을 포함하여 AWS IoT Greengrass 코어 소프트웨어 로그 파일 또는 Greengrass 구성 요소 로그 파일의 크기가 디스크 공간 제한을 초과하는 경우 이전 로그 파일을 정리합니다. AWS IoT Greengrass 코어 소프트웨어 로그 및 각 구성 요소 로그의 기본 디스크 공간 제한은 10,240KB(10MB)입니다. [Greengrass nucleus 구성 요소](#) 또는 [로그 관리자 구성 요소](#)에서 AWS IoT Greengrass 코어 소프트웨어 로그 디스크 공간 제한을 구성할 수 있습니다. [로그 관리자 구성 요소](#)에서는 각 구성 요소의 로그 디스크 공간 제한을 구성할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어 로그 파일을 보려면

- 로그 파일을 실시간으로 보려면 다음 명령을 실행합니다. `/greengrass/v2`를 AWS IoT Greengrass 루트 폴더의 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
```

type 명령에서는 파일의 내용이 터미널이 기록됩니다. 이 명령을 여러 번 실행하여 파일의 변경 사항을 관찰합니다.

PowerShell

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

구성 요소의 로그 파일을 보려면

- 로그 파일을 실시간으로 보려면 다음 명령을 실행합니다. `/greengrass/v2` 또는 `C:\greengrass\v2`를 AWS IoT Greengrass 루트 폴더의 경로로 바꾸고 `com.example>HelloWorld`를 구성 요소 이름으로 바꿉니다.

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
```

[Greengrass CLI](#)의 `logs` 명령을 사용하여 코어 디바이스에서 Greengrass 로그를 분석할 수도 있습니다. `logs` 명령을 사용하려면 [Greengrass nucleus가](#) JSON 형식 로그 파일을 출력하도록 구성해야 합니다. 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 및 [로그\(을\)](#)를 참조하세요.

CloudWatch Logs 액세스

[로그 관리자 구성 요소](#)를 배포하여 CloudWatch Logs에 쓰도록 코어 디바이스를 구성할 수 있습니다. 자세한 내용은 [CloudWatch Logs에 로깅 활성화](#) 섹션을 참조하세요. 그런 다음 Amazon CloudWatch 콘솔의 로그 페이지에서나 CloudWatch Logs API를 사용하여 로그를 볼 수 있습니다.

로그 그룹 이름

```
/aws/greengrass/componentType/region/componentName
```

로그 그룹 이름에는 다음 변수를 사용합니다.

- `componentType` - 구성 요소의 유형으로, 다음 중 하나일 수 있습니다.
 - `GreengrassSystemComponent` - 이 로그 그룹에는 Greengrass nucleus와 동일한 JVM에서 실행되는 nucleus 및 플러그인 구성 요소에 대한 로그가 포함됩니다. 이 구성 요소는 [Greengrass nucleus](#)의 일부입니다.
 - `UserComponent` - 이 로그 그룹에는 디바이스의 일반 구성 요소, Lambda 구성 요소 및 기타 애플리케이션에 대한 로그가 포함됩니다. 이 구성 요소는 Greengrass nucleus의 일부가 아닙니다.

자세한 내용은 [구성 요소 유형](#) 섹션을 참조하세요.

- `region` - 코어 디바이스에서 사용하는 AWS 리전입니다.
- `componentName` - 구성 요소의 이름입니다. 시스템 로그의 경우 이 값은 `System`입니다.

로그 스트림 이름

```
/date/thing/thingName
```

로그 스트림 이름에는 다음 변수를 사용합니다.

- `date` - 와 같은 로그 날짜입니다(예: 2020/12/15). 로그 관리자 구성 요소에서는 `yyyy/MM/dd` 형식을 사용합니다.
- `thingName` - 코어 디바이스 이름의 이름입니다.

Note

사물 이름에 콜론(:)이 포함된 경우 로그 관리자에서는 콜론을 더하기(+)로 바꿉니다.

로그 관리자 구성 요소를 사용하여 CloudWatch Logs에 쓸 때는 다음 사항을 고려하세요.

- 로그 지연

Note

교체된 로그 파일과 활성 로그 파일에 대한 로그 지연을 줄이는 로그 관리자 버전 2.3.0으로 업그레이드하는 것이 좋습니다. 로그 관리자 2.3.0으로 업그레이드할 때는 Greengrass nucleus도 2.9.1로 업그레이드하는 것이 좋습니다.

로그 관리자 구성 요소 2.2.8 및 이전 버전은 교체된 로그 파일에서만 로그를 처리하고 업로드합니다. 기본적으로 AWS IoT Greengrass 코어 소프트웨어는 로그 파일을 매시간 교체하거나 1,024KB가 넘으면 교체합니다. 따라서 로그 관리자 구성 요소는 AWS IoT Greengrass 코어 소프트웨어 또는 Greengrass 구성 요소에서 1,024KB 이상의 로그를 쓴 후에만 로그를 업로드합니다. 로그 파일 크기 제한을 더 낮게 구성하여 로그 파일이 더 자주 교체되도록 할 수 있습니다. 이렇게 하면 로그 관리자 구성 요소에서 CloudWatch Logs에 로그를 더 자주 업로드합니다.

로그 관리자 구성 요소 2.3.0 이상 버전은 모든 로그를 처리하고 업로드합니다. 새 로그를 쓸 때 로그 관리자 2.3.0 이상 버전은 해당 활성 로그 파일이 교체될 때까지 기다리지 않고 활성 로그를 처리하고 직접 업로드합니다. 즉, 새 로그를 5분 이내에 볼 수 있습니다.

로그 관리자 구성 요소에서는 새 로그를 주기적으로 업로드합니다. 기본적으로 로그 관리자 구성 요소는 5분마다 새 로그를 업로드합니다. 업로드 간격을 더 낮게 구성할 수 있으므로 `periodicUploadIntervalSec`을 구성하면 로그 관리자 구성 요소가 CloudWatch Logs에 로그를 더 자주 업로드합니다. 이 주기적 간격을 구성하는 방법에 대한 자세한 내용은 [구성](#)을 참조하세요.

동일한 Greengrass 파일 시스템에서 로그를 거의 실시간으로 업로드할 수 있습니다. 로그를 실시간으로 관찰해야 하는 경우 [파일 시스템 로그](#)를 사용하는 것이 좋습니다.

Note

다른 파일 시스템을 사용하여 로그를 쓰는 경우 로그 관리자가 로그 관리자 구성 요소 2.2.8 이하 버전의 동작으로 돌아갑니다. 파일 시스템 로그에 액세스하는 방법에 대한 자세한 내용은 [파일 시스템 로그 액세스](#)를 참조하세요.

• 클록 스큐

로그 관리자 구성 요소는 표준 Signature 버전 4 서명 프로세스를 사용하여 CloudWatch Logs에 대한 API 요청을 생성합니다. 코어 디바이스에서 시스템 시간이 15분 이상 동기화되지 않을 경우, CloudWatch Logs에서 요청을 거부합니다. 자세한 정보는 AWS 일반 참조의 [서명 버전 4 서명 프로세스](#)를 참조하십시오.

시스템 서비스 로그 액세스

[AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 구성](#)하는 경우 시스템 서비스 로그를 확인하여 소프트웨어가 시작되지 않음과 같은 문제를 해결할 수 있습니다.

시스템 서비스 로그를 보려면(CLI)

1. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 시스템 서비스 로그를 봅니다.

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 디바이스에서 AWS IoT Greengrass 코어 소프트웨어는 시스템 서비스 오류에 대한 로그 파일을 별도로 생성합니다. 다음 명령을 실행하여 시스템 서비스 오류 로그를 봅니다.

Windows Command Prompt (CMD)

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

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.err.log
```

Windows 디바이스에서는 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. 애플리케이션을 선택하여 애플리케이션 서비스 로그를 봅니다.
4. 소스가 greengrass인 이벤트 로그를 찾아 엽니다.

CloudWatch Logs에 로깅 활성화

[로그 관리자 구성 요소](#)를 배포하여 CloudWatch Logs에 로그를 쓰도록 코어 디바이스를 구성할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어 로그에 대해 CloudWatch Logs를 활성화하고 특정 Greengrass 구성 요소에 대해 CloudWatch Logs를 활성화할 수 있습니다.

Note

다음 예제 IAM 정책에 표시된 대로 Greengrass 코어 디바이스의 토큰 교환 역할에서 코어 디바이스가 CloudWatch Logs에 쓸 수 있도록 허용해야 합니다. [자동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치](#)한 경우 코어 디바이스에 이러한 권한이 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

AWS IoT Greengrass 코어 소프트웨어 로그를 CloudWatch Logs에 쓰도록 코어 디바이스를 구성하려면 `aws.greengrass.LogManager` 구성 요소에 대해 `uploadToCloudWatch`를 `true`로 설정하는 구성 업데이트를 지정하는 [배포를 생성합니다](#). AWS IoT Greengrass 코어 소프트웨어 로그에는 [Greengrass nucleus](#) 및 [플러그인 구성 요소](#)에 대한 로그가 포함됩니다.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Greengrass 구성 요소의 로그를 CloudWatch Logs에 쓰도록 코어 디바이스를 구성하려면 구성 요소 로깅 구성 목록에 구성 요소를 추가하는 구성 업데이트를 지정하는 [배포를 생성](#)합니다. 이 목록에 구성 요소를 추가하면 로그 관리자 구성 요소는 그 구성 요소의 로그를 CloudWatch Logs에 씁니다. 구성 요소 로그에는 [일반 구성 요소 및 Lambda 구성 요소](#)에 대한 로그가 포함됩니다.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
```

```

    }
  }
}
}

```

로그 관리자 구성 요소를 배포할 때 디스크 공간 제한과 코어 디바이스에서 CloudWatch Logs에 로그를 쓴 후 로그 파일을 삭제하는지 여부도 구성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass의 로깅 구성](#) 섹션을 참조하세요.

AWS IoT Greengrass의 로깅 구성

다음 옵션을 구성하여 Greengrass 코어 디바이스에 대한 로깅을 사용자 지정할 수 있습니다. 이러한 옵션을 구성하려면 Greengrass nucleus 또는 로그 관리자 구성 요소에 대한 구성 업데이트를 지정하는 [배포를 생성](#)합니다.

- CloudWatch Logs에 로그 쓰기

코어 디바이스 문제를 원격으로 해결하려면 AWS IoT Greengrass 코어 소프트웨어 및 구성 요소 로그를 CloudWatch Logs에 쓰도록 코어 디바이스를 구성할 수 있습니다. 이렇게 하려면 [로그 관리자 구성 요소](#)를 배포하고 구성합니다. 자세한 내용은 [CloudWatch Logs에 로깅 활성화](#) 섹션을 참조하세요.

- 업로드된 로그 파일 삭제

디스크 공간 사용량을 줄이기 위해 CloudWatch Logs에 로그 파일을 쓴 후 로그 파일을 삭제하도록 코어 디바이스를 구성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 로그 및 구성 요소 로그](#)에 지정할 수 있는 로그 관리자 구성 요소의 deleteLogFileAfterCloudUpload 파라미터를 참조하세요.

- 로그 디스크 공간 제한

디스크 공간 사용량을 제한하려면 코어 디바이스에서 교체된 로그 파일을 포함하여 각 로그에 대한 최대 디스크 공간을 구성할 수 있습니다. 예를 들어 greengrass.log 및 교체된 greengrass.log 파일의 최대 합산 디스크 공간을 구성할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 로그 및 구성 요소 로그](#)에 지정할 수 있는 Greengrass nucleus 구성 요소의 logging.totalLogsSizeKB 파라미터 및 로그 관리자 구성 요소의 diskSpaceLimit 파라미터를 참조하세요.

- 로그 파일 크기 제한

각 로그 파일의 최대 파일 크기를 구성할 수 있습니다. 로그 파일이 이 파일 크기 제한을 초과하면 AWS IoT Greengrass 코어 소프트웨어에서 새 로그 파일을 생성합니다. [로그 관리자 구성 요소](#) 2.28

이하 버전에서는 교체된 로그 파일만 CloudWatch Logs에 쓰므로 CloudWatch Logs에 로그를 더 자주 쓰도록 파일 크기 제한을 더 낮게 지정할 수 있습니다. 로그 관리자 구성 요소 2.3.0 이상 버전에서는 로그가 교체될 때까지 기다리지 않고 모든 로그를 처리하고 업로드합니다. 자세한 내용은 Greengrass nucleus 구성 요소의 [로그 파일 크기 제한 파라미터](#)(logging.fileSizeKB)를 참조하세요.

- 최소 로그 수준

Greengrass nucleus 구성 요소가 파일 시스템 로그에 쓰는 최소 로그 수준을 구성할 수 있습니다. 예를 들어 문제 해결에 도움이 되는 DEBUG 수준 로그를 지정하거나 코어 디바이스가 생성하는 로그의 양을 줄이기 위해 ERROR 수준 로그를 지정할 수 있습니다. 자세한 내용은 Greengrass nucleus 구성 요소의 [로그 수준 파라미터](#)(logging.level)를 참조하세요.

로그 관리자 구성 요소가 CloudWatch Logs에 쓰는 최소 로그 수준을 구성할 수도 있습니다. 예를 들어 [로깅 비용](#)을 줄이기 위해 더 높은 로그 수준을 지정할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 로그 및 구성 요소 로그](#)에 지정할 수 있는 로그 관리자 구성 요소의 minimumLogLevel 파라미터를 참조하세요.

- CloudWatch Logs에 쓸 로그를 확인하는 간격

로그 관리자 구성 요소가 CloudWatch Logs에 로그를 쓰는 빈도를 늘리거나 줄이려면 쓸 새 로그 파일을 확인하는 간격을 구성할 수 있습니다. 예를 들어 CloudWatch Logs에서 로그를 기본 5분 간격보다 더 빨리 보도록 간격을 더 낮게 지정할 수 있습니다. 간격을 더 높게 지정하여 비용을 줄일 수도 있습니다. 이 경우 로그 관리자 구성 요소에서 로그 파일을 더 적은 요청으로 일괄 처리하기 때문입니다. 자세한 내용은 로그 관리자 구성 요소의 [업로드 간격 파라미터](#)(periodicUploadIntervalSec)를 참조하세요.

- 로그 형식

AWS IoT Greengrass 코어 소프트웨어에서 로그를 텍스트 또는 JSON 형식 중 어떤 형식으로 쓰는지 선택할 수 있습니다. 로그를 읽는 경우에는 텍스트 형식을 선택하고, 애플리케이션을 사용하여 로그를 읽거나 구문 분석하는 경우에는 JSON 형식을 선택합니다. 자세한 내용은 Greengrass nucleus 구성 요소의 [로그 형식 파라미터](#)(logging.format)를 참조하세요.

- 로컬 파일 시스템 로그 폴더

로그 폴더를 `/greengrass/v2/logs`에서 코어 디바이스의 다른 폴더로 변경할 수 있습니다. 자세한 내용은 Greengrass nucleus 구성 요소의 [출력 디렉터리 파라미터](#)(logging.outputDirectory)를 참조하세요.

AWS CloudTrail 로그

AWS IoT Greengrass는 AWS IoT Greengrass에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. 자세한 내용은 [를 사용하여 AWS IoT Greengrass V2 API 호출 로깅 AWS CloudTrail](#) 섹션을 참조하세요.

를 사용하여 AWS IoT Greengrass V2 API 호출 로깅 AWS CloudTrail

AWS IoT Greengrass V2 는 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다 AWS IoT Greengrass Version 2. CloudTrail은에 대한 모든 API 호출을 이벤트 AWS IoT Greengrass 로 캡처합니다. 캡처되는 호출에는 AWS IoT Greengrass 콘솔의 호출과 AWS IoT Greengrass API 작업에 대한 코드 호출이 포함됩니다.

추적을 생성하면 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 S3 버킷에 배포할 수 있습니다 AWS IoT Greengrass. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 수행된 요청, 요청이 수행된 AWS IoT Greengrass IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용자 안내서](#)를 참조하세요.

주제

- [AWS IoT Greengrass V2 CloudTrail의 정보](#)
- [AWS IoT Greengrass CloudTrail의 데이터 이벤트](#)
- [AWS IoT Greengrass CloudTrail의 관리 이벤트](#)
- [AWS IoT Greengrass V2 로그 파일 항목 이해](#)

AWS IoT Greengrass V2 CloudTrail의 정보

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화됩니다. 에서 활동이 발생하면 AWS IoT Greengrass 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 [CloudTrail 이벤트 기록을 사용하여 이벤트 보기](#)를 참조하세요.

에 대한 이벤트를 AWS 계정포함하여에 이벤트를 지속적으로 기록하려면 추적을 AWS IoT Greengrass 생성합니다. 추적은 CloudTrail이 S3 버킷으로 로그 파일을 전송할 수 있도록 합니다. 기본

적으로 콘솔에서 추적을 생성하면 추적이 모든 AWS 리전에 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기](#) 및 [여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 AWS IoT Greengrass V2 작업은 CloudTrail에서 로깅되며 [AWS IoT Greengrass V2 API 참조](#)에 문서화됩니다. 예를 들어 CreateComponentVersion, CreateDeployment, CancelDeployment 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에 의해 이루어졌는지 여부입니다.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하세요.

AWS IoT Greengrass CloudTrail의 데이터 이벤트

[데이터 이벤트](#)는 리소스에 대해 또는 리소스에서 수행된 리소스 작업에 대한 정보를 제공합니다(예: 구성 요소 버전 또는 배포에 대한 구성 가져오기). 이를 데이터 플레인 작업이라고도 합니다. 데이터 이벤트는 흔히 대량 활동입니다. 기본적으로 CloudTrail은 데이터 이벤트를 로깅하지 않습니다. CloudTrail 이벤트 기록은 데이터 이벤트를 기록하지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 관한 자세한 내용은 [AWS CloudTrail 요금](#)을 참조하세요.

CloudTrail 콘솔 AWS CLI 또는 CloudTrail API 작업을 사용하여 AWS IoT Greengrass 리소스 유형에 대한 데이터 이벤트를 로깅할 수 있습니다. 이 섹션의 [표](#)에는 사용 가능한 리소스 유형이 나와 있습니다 AWS IoT Greengrass.

- CloudTrail 콘솔을 사용하여 데이터 이벤트를 로깅하려면 [추적](#) 또는 [이벤트 데이터 스토어](#)를 생성하여 데이터 이벤트를 기록하거나 [기존 추적 또는 이벤트 데이터 스토어를 업데이트](#)하여 데이터 이벤트를 기록합니다.
 1. 데이터 이벤트를 로깅하려면 데이터 이벤트를 선택합니다.
 2. 데이터 이벤트 유형에서 데이터 이벤트를 로깅할 리소스 유형을 선택합니다.
 3. 사용할 템플릿에서 로그 선택기를 선택합니다. 리소스 유형에 대한 모든 데이터 이벤트를 로그하거나, 모든 readOnly 이벤트를 로그하거나, 모든 writeOnly 이벤트를 로그하거나, 사용자 지정 로그 선택기 템플릿을 생성하여 readOnly, eventName 및 resources.ARN 필드를 필터링할 수 있습니다.
- 를 사용하여 데이터 이벤트를 로깅하려면 eventCategory 필드를 로 Data 설정하고 resources.type 필드를 리소스 유형 값과 로 설정하도록 --advanced-event-selectors 파라미터를 AWS CLI 구성합니다([표 참조](#)). 조건을 추가하여 readOnly, eventName 및 resources.ARN 필드의 값을 필터링할 수 있습니다.
 - 데이터 이벤트를 로깅하도록 추적을 구성하려면 [put-event-selectors](#) 명령을 실행합니다. 자세한 내용은 [AWS CLI로 추적에 대한 데이터 이벤트 로깅](#)을 참조하세요.
 - 데이터 이벤트를 로깅하도록 이벤트 데이터 스토어를 구성하려면 [create-event-data-store](#) 명령을 실행하여 데이터 이벤트를 로깅할 새 이벤트 데이터 스토어를 생성하거나 [update-event-data-store](#) 명령을 실행하여 기존 이벤트 데이터 스토어를 업데이트합니다. 자세한 내용은 [AWS CLI로 데이터 이벤트 스토어에 대한 데이터 이벤트 로깅](#)을 참조하세요.

다음 표에는 AWS IoT Greengrass 리소스 유형이 나열되어 있습니다. 데이터 이벤트 유형(콘솔) 옆에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 값이 표시됩니다. resources.type 값 옆에는 AWS CLI 또는 CloudTrail APIs를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 resources.type 값이 표시됩니다. CloudTrail에 로깅되는 데이터 API 옆에는 리소스 유형에 대해 CloudTrail에 로깅된 API 직접 호출이 표시됩니다.

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
IoT 인증서	AWS::IoT::Certificate	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
IoT Greengrass 구성 요소 버전	AWS::GreengrassV2::ComponentVersion	• ResolveComponentCandidates
IoT Greengrass 배포	AWS::GreengrassV2::Deployment	• GetDeploymentConfiguration
IoT 사물	AWS::IoT::Thing	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

eventName, readOnly 및 resources.ARN 필드를 필터링하여 중요한 이벤트만 로깅하도록 고급 이벤트 선택기를 구성할 수 있습니다.

eventName에 필터를 추가하여 특정 데이터 API를 포함하거나 제외합니다.

필드에 대한 자세한 내용은 [AdvancedFieldSelector](#) 섹션을 참조하세요.

다음 예제에서는 AWS CLI를 사용하여 고급 선택기를 구성하는 방법을 보여줍니다. *TrailName* 및 *region*을 자신의 정보로 바꿉니다.

Example- IoT 사물에 대한 데이터 이벤트 로깅

```
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 API 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 로깅

```
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 CloudTrail의 관리 이벤트

[관리 이벤트](#)는 AWS 계정의 리소스에 대해 수행되는 관리 작업에 대한 정보를 제공합니다. 이를 컨트롤 플레인 작업이라고도 합니다. 기본적으로 CloudTrail은 관리 이벤트를 로깅합니다.

AWS IoT Greengrass 는 모든 AWS IoT Greengrass 컨트롤 플레인 작업을 관리 이벤트로 기록합니다. CloudTrail에 AWS IoT Greengrass 로깅하는 AWS IoT Greengrass 컨트롤 플레인 작업 목록은 [AWS IoT Greengrass API 참조 버전 2](#)를 참조하세요.

AWS IoT Greengrass V2 로그 파일 항목 이해

추적이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있도록 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터 단일 요청을 나타냅니다. 요청된 작업, 모든 파라미터, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 CreateDeployment 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예시입니다.

```
{
  "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 Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집

시스템 상태 원격 측정 데이터는 Greengrass 코어 장치의 중요 작업 성능을 모니터링하는 데 도움이 되는 진단 데이터입니다. 엣지 장치에서 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 디바이스에서 원격 측정 데이터를 수집할 수 있습니다.

- nucleus 원격 측정 이미터 구성 요소 - Greengrass 코어 디바이스의 [nucleus 원격 측정 이미터 구성 요소](#)(aws.greengrass.telemetry.NucleusEmitter)는 기본적으로 원격 측정 데이터를 \$local/greengrass/telemetry 주제에 게시합니다. 이 주제에 게시된 데이터를 사용하여 디바이스의 클라우드 연결이 제한되어 있는 경우에도 코어 디바이스에서 로컬로 작업할 수 있습니다. 선택적으로 선택한 AWS IoT Core MQTT 주제에 원격 측정 데이터를 게시하도록 구성 요소를 구성할 수도 있습니다.

원격 측정 데이터를 게시하려면 코어 디바이스에 nucleus 이미터 구성 요소를 배포해야 합니다. 원격 측정 데이터를 로컬 주제에 게시하는 데는 비용이 들지 않습니다. 그러나 MQTT 주제를 사용하여 데이터를 게시하는 데 AWS 클라우드 는 [AWS IoT Core 요금](#)이 적용됩니다.

AWS IoT Greengrass 는 InfluxDB 및 Grafana를 사용하여 코어 디바이스에서 로컬로 원격 측정 데이터를 분석하고 시각화하는 데 도움이 되는 여러 [커뮤니티 구성 요소](#)를 제공합니다. 이러한 구성 요소는 nucleus 이미터 구성 요소의 원격 측정 데이터를 사용합니다. 자세한 내용은 [InfluxDB 게시자 구성 요소](#)에 대한 README를 참조하세요.

- 원격 측정 에이전트 - Greengrass 코어 디바이스의 원격 측정 에이전트는 로컬 원격 측정 데이터를 수집하여 고객과 상호 작용할 필요 없이 Amazon EventBridge에 게시합니다. 코어 장치는 최선의 노력으로 EventBridge에 원격 측정 데이터를 게시합니다. 예를 들어, 코어 장치는 오프라인 상태에서 원격 측정 데이터를 제공하지 못할 수 있습니다.

원격 측정 에이전트 기능은 모든 Greengrass 코어 디바이스에서 기본적으로 활성화되어 있습니다. Greengrass 코어 디바이스를 설정하는 즉시 자동으로 데이터 수신을 시작합니다. 데이터 링크 비용 외에도 코어 디바이스에서 로 데이터를 AWS IoT Core 무료로 전송할 수 있습니다. 이는 에이전트가 AWS 예약된 주제에 게시하기 때문입니다. 하지만 사용 사례에 따라 데이터를 받거나 처리할 때 비용이 발생할 수 있습니다.

Note

Amazon EventBridge는 애플리케이션을 Greengrass 코어 디바이스와 같은 다양한 소스의 데이터와 연결하는 데 사용할 수 있는 이벤트 버스 서비스입니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 [Amazon EventBridge란?](#) 섹션을 참조하세요.

AWS IoT Greengrass 코어 소프트웨어가 제대로 작동하는지 확인하기 위해서는 개발 및 품질 개선을 위해 데이터를 AWS IoT Greengrass 사용합니다. 또한 이 기능은 새롭고 향상된 엣지 기능을 알리는 데 도움이 됩니다.는 최대 7일 동안 원격 측정 데이터를 AWS IoT Greengrass 유지합니다.

이 섹션에서는 원격 측정 에이전트를 구성하고 사용하는 방법에 대해 설명합니다. nucleus 원격 측정 이미터 구성 요소 구성에 대한 자세한 내용은 [nucleus 원격 측정 이미터](#) 섹션을 참조하세요.

주제

- [원격 측정 지표](#)
- [원격 측정 에이전트 설정](#)
- [EventBridge에서 원격 측정 데이터 구독](#)

원격 측정 지표

다음 표에서는 원격 측정 에이전트에서 게시하는 지표를 설명합니다.

이름	설명	
시스템		
SystemMemUsage	운영 체제를 포함하여 Greengrass 코어 장치의 모든 애플리케이션에서 현재 사용 중인 메모리의 양입니다.	
CpuUsage	운영 체제를 포함하여 Greengrass 코어 장치의 모든 애플리케이션에서 현재 사용 중인 CPU의 양입니다.	
TotalNumberOfFDs	Greengrass 코어 장치의 운영 체제에 저장된 파일 디스크립터 수입니다. 하나의 파일 디스크립터는 열려 있는 파일 하나를 고유하게 식별합니다.	
Greengrass nucleus		
NumberOfComponentsRunning	Greengrass 코어 디바이스에서 실행 중인 구성 요소의 수입니다.	

이름	설명	
NumberOfComponents Errored	Greengrass 코어 디바이스에서 오류 상태에 있는 구성 요소의 수입니다.	
NumberOfComponents Installed	Greengrass 코어 디바이스에 설치된 구성 요소의 수입니다.	
NumberOfComponents Starting	Greengrass 코어 디바이스에서 시작하는 구성 요소의 수입니다.	
NumberOfComponents New	Greengrass 코어 디바이스의 새로운 구성 요소의 수입니다.	
NumberOfComponents Stopping	Greengrass 코어 디바이스에서 중지하는 구성 요소의 수입니다.	
NumberOfComponents Finished	Greengrass 코어 디바이스에서 완료된 구성 요소의 수입니다.	
NumberOfComponents Broken	Greengrass 코어 디바이스에서 손상된 구성 요소의 수입니다.	
NumberOfComponents Stateless	Greengrass 코어 디바이스에 있는 상태 비저장 구성 요소의 수입니다.	
클라이언트 디바이스 인증 - 이 기능을 사용하려면 클라이언트 디바이스 인증 구성 요소의 v2.4.0 이상이 필요합니다.		
VerifyClientDevice Identity.Success	클라이언트 디바이스 ID가 성공했는지 확인하는 횟수.	

이름	설명	
VerifyClientDeviceIdentity.Failure	클라이언트 디바이스 ID가 실패했는지 확인하는 횟수.	
AuthorizeClientDeviceActions.Success	클라이언트 디바이스에 요청된 작업을 완료할 수 있는 권한이 부여된 횟수.	
AuthorizeClientDeviceActions.Failure	클라이언트 디바이스에 요청된 작업을 완료할 수 있는 권한이 부여되지 않은 횟수.	
GetClientDeviceAuthToken.Success	클라이언트 디바이스가 성공적으로 인증된 횟수.	
GetClientDeviceAuthToken.Failure	클라이언트 디바이스가 인증되지 못한 횟수.	
SubscribeToCertificateUpdates.Success	인증서 업데이트에 대한 성공적인 구독의 수.	
SubscribeToCertificateUpdates.Failure	인증서 업데이트를 구독하려다 실패한 시도 횟수.	
ServiceError	클라이언트 디바이스 인증에서 처리되지 않은 내부 오류의 수.	
스트림 관리자 - 이 기능은 Greengrass nucleus 구성 요소의 v2.7.0 이상에서 사용할 수 있습니다.		
BytesAppended	스트림 관리자에 추가된 데이터의 바이트 수입니다.	
BytesUploadedToIoTAnalytics	스트림 관리자가 채널로 보내는 데이터의 바이트 수입니다 AWS IoT Analytics.	

이름	설명	
BytesUploadedToKinesis	스트림 관리자가 Amazon Kinesis Data Streams의 스트림으로 내보내는 데이터의 바이트 수입니다.	
BytesUploadedToIoTSiteWise	스트림 관리자가 자산 속성으로 내보내는 데이터의 바이트 수입니다 AWS IoT SiteWise.	
BytesUploadedToS3	스트림 관리자가 Amazon S3의 객체로 내보내는 데이터의 바이트 수입니다.	
시스템 지표 – 이 기능은 Greengrass nucleus 구성 요소 버전 2.15.0 이상이 필요합니다.		
CPUArchitecture	디바이스 중앙 처리 장치 아키텍처.	
Family	디바이스 운영 체제 계열 (Windows만 해당).	
KernelVersion	디바이스 커널 버전(Unix만 해당).	
KnowledgeBaseArticles	디바이스에 설치된 지식 베이스 문서(Windows만 해당).	
OSBuildMajor	운영 체제 버전의 메이저 빌드 번호(Windows만 해당).	
OSBuildMinor	운영 체제 버전의 마이너 빌드 번호(Windows만 해당).	
OSName	디바이스 운영 체제의 이름.	

이름	설명
OSVersion	디바이스의 운영 체제의 마케팅 버전.

원격 측정 에이전트 설정

원격 측정 에이전트는 다음 기본 설정을 사용합니다.

- 원격 측정 에이전트는 1시간마다 원격 측정 데이터를 집계합니다.
- 원격 측정 에이전트는 24시간마다 원격 측정 메시지를 게시합니다.

원격 측정 에이전트는 서비스 품질(QoS) 수준이 0인 MQTT 프로토콜을 사용하여 데이터를 게시합니다. 이는 전송을 확인하거나 게시 시도를 다시 시도하지 않음을 의미합니다. 원격 측정 메시지는 AWS IoT Core 대상으로 하는 구독에 대한 다른 메시지와 MQTT 연결을 공유합니다.

데이터 링크 비용 외에도 코어에서 로 데이터를 AWS IoT Core 무료로 전송할 수 있습니다. 이는 에이전트가 AWS 예약된 주제에 게시하기 때문입니다. 하지만 사용 사례에 따라 데이터를 받거나 처리할 때 비용이 발생할 수 있습니다.

각 Greengrass 코어 디바이스에 대해 원격 측정 에이전트 기능을 활성화하거나 비활성화할 수 있습니다. 코어 디바이스가 데이터를 집계하고 게시하는 간격을 구성할 수도 있습니다. 원격 측정을 구성하려면 [Greengrass nucleus 구성 요소](#)를 배포할 때 [원격 측정 구성 파라미터](#)를 사용자 지정합니다.

EventBridge에서 원격 측정 데이터 구독

Amazon EventBridge에서 Greengrass 코어 디바이스의 원격 측정 에이전트에서 게시된 원격 측정 데이터를 처리하는 방법을 정의하는 규칙을 생성할 수 있습니다. EventBridge는 데이터를 수신하면 규칙에 정의된 대상 작업을 간접적으로 호출합니다. 예를 들어 알림을 보내거나, 이벤트 정보를 저장하거나, 교정 작업을 수행하거나, 기타 이벤트를 간접적으로 호출하는 이벤트 규칙을 생성할 수 있습니다.

원격 측정 이벤트

원격 측정 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"
    }
  ]
}
{
  "TS": 1750104449426,
  "NS": "SystemMetrics",
  "M": [
    {
      "N": "KernelVersion",
      "Sum": 1,
      "U": "6.1.140-154.222.amzn2023.x86_64"
    },
    {
      "N": "OSVersion",
      "Sum": 1,
      "U": "2023.7.20250609"
    }
  ]
}
```

```

    },
    {
      "N": "OSName",
      "Sum": 1,
      "U": "Amazon Linux"
    },
    {
      "N": "CPUArchitecture",
      "Sum": 1,
      "U": "Broadwell (Server)"
    },
  ],
}
]
}
}
}

```

ADP 배열에는 다음 속성을 가진 집계된 데이터 포인트 목록이 포함됩니다.

TS

데이터가 수집된 시기의 타임스탬프입니다.

NS

지표 네임스페이스입니다.

M

지표 목록 이 지표는 다음 속성을 포함하고 있습니다.

N

지표의 이름.

Sum

이 원격 측정 이벤트의 지표 값의 합계.

U

지표 값의 단위.

각 지표에 대한 자세한 내용은 [원격 측정 지표](#) 섹션을 참조하세요.

EventBridge 규칙을 생성하기 위한 사전 조건

에 대한 EventBridge 규칙을 생성하기 전에 다음을 수행해야 AWS IoT Greengrass합니다.

- Eventbridge의 이벤트, 규칙, 대상을 숙지해야 합니다.
- EventBridge 규칙에 의해 호출되는 [대상](#)을 생성하고 구성해야 합니다. 규칙은 Amazon Kinesis 스트림, AWS Lambda 함수, Amazon SNS 주제 및 Amazon SQS 대기열과 같은 다양한 유형의 대상을 호출할 수 있습니다.

EventBridge 규칙 및 연결된 대상은 Greengrass 리소스를 생성한 AWS 리전 에 있어야 합니다. 자세한 내용은 AWS 일반 참조의 [서비스 엔드포인트 및 할당량](#)을 참조하세요.

자세한 내용은 Amazon EventBridge 사용 설명서의 [Amazon EventBridge란?](#) 및 [Amazon EventBridge 시작하기](#) 섹션을 참조하세요.

원격 측정 데이터를 가져오기 위한 이벤트 규칙 생성(콘솔)

다음 단계를 사용하여 Greengrass 코어 디바이스에서 게시한 원격 측정 데이터를 수신하는 EventBridge 규칙을 AWS Management Console 생성합니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다. 자세한 내용은 Amazon [EventBridge 사용 설명서의 AWS 리소스에서 이벤트를 트리거하는 EventBridge 규칙 생성](#)을 참조하세요. EventBridge

1. [Amazon EventBridge 콘솔](#)을 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 선택하세요.
 - c. 서비스 공급자(Service provider)에 AWS를 선택합니다.
 - d. 서비스 이름에서 Greengrass를 선택합니다.
 - e. 이벤트 유형에서 Greengrass 원격 측정 데이터를 선택합니다.
4. 이벤트 버스 선택에서 기본 이벤트 버스 옵션을 유지합니다.
5. 대상 선택에서 대상을 구성합니다. 다음 예제는 Amazon SQS 대기열을 사용하지만 다른 대상 유형을 구성할 수도 있습니다.
 - a. 대상에서 SQS 대기열을 선택합니다.
 - b. 대기열*에서 대상 대기열을 선택합니다.

6. 태그 - 선택 사항에서 규칙에 대한 태그를 정의하거나 필드를 비워 둡니다.
7. 생성(Create)을 선택합니다.

원격 측정 데이터(CLI) 를 가져오기 위한 이벤트 규칙 생성

다음 단계에 따라 AWS CLI 를 사용하여 Greengrass 코어 디바이스에서 게시한 원격 측정 데이터를 수신하는 EventBridge 규칙을 생성합니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다.

1. 규칙을 생성합니다.
 - *thing-name*을 코어 디바이스의 사물 이름으로 바꿉니다.

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 ARN으로 대체하세요.

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

Amazon EventBridge가 대상 주제를 간접적으로 호출하도록 허용하려면 대기열에 리소스 기반 정책을 추가해야 합니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 [Amazon SQS 권한](#) 섹션을 참조하세요.

자세한 내용은 Amazon EventBridge 사용 설명서에서 [EventBridge의 이벤트 및 이벤트 패턴](#) 섹션을 참조하세요.

배포 및 구성 요소 상태 알림 받기

Amazon EventBridge 이벤트 규칙에서는 디바이스에 수신된 Greengrass 배포 및 디바이스에 설치된 구성 요소의 상태 변경에 대한 알림이 제공됩니다. EventBridge는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. 이러한 이벤트를 최선을 다해 EventBridge에 AWS IoT Greengrass 전송합니다. 즉, 모든 이벤트를 EventBridge로 전송하려고

AWS IoT Greengrass 시도하지만 경우에 따라 이벤트가 전달되지 않을 수 있습니다. 또한 AWS IoT Greengrass 는 지정된 이벤트의 사본을 여러 개 전송할 수 있습니다. 즉, 이벤트 리스너가 이벤트가 발생한 순서대로 이벤트를 수신하지 못할 수 있습니다.

Note

Amazon EventBridge는 애플리케이션을 [Greengrass 코어 디바이스](#)와 배포 및 구성 요소 알림과 같은 다양한 소스의 데이터와 연결하는 데 사용할 수 있는 이벤트 버스 서비스입니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 [Amazon EventBridge란?](#) 섹션을 참조하세요.

주제

- [배포 상태 변경 이벤트](#)
- [구성 요소 상태 변경 이벤트](#)
- [EventBridge 규칙 생성을 위한 사전 조건](#)
- [디바이스 상태 알림 구성\(콘솔\)](#)
- [디바이스 상태 알림 구성\(CLI\)](#)
- [디바이스 상태 알림 구성\(CloudFormation\)](#)
- [다음 사항도 참조하세요.](#)

배포 상태 변경 이벤트

AWS IoT Greengrass 는 배포가 FAILED, , COMPLETED, SUCCEDEDREJECTED, 상태가 되면 이벤트를 내보냅니다CANCELED. 모든 상태 전환 또는 지정한 상태로의 전환에 대해 실행하는 EventBridge 규칙을 생성할 수 있습니다. 배포가 규칙을 트리거하는 상태로 전환되면 EventBridge는 규칙에 정의된 대상 작업을 간접 호출합니다. 이렇게 하면 알림을 전송하고, 이벤트 정보를 캡처하고, 적절한 조치를 취하거나, 상태 변경에 대응하는 기타 이벤트를 시작할 수 있습니다. 예를 들어 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 [이벤트](#)에서 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|REJECTED|CANCELED",
    "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/
1uLCYANGq08RIH+x2H+hEKc=)"
  }
}
```

배포의 상태에 대한 업데이트가 제공되는 규칙과 이벤트를 생성할 수 있습니다. 배포가 FAILED, SUCCEEDED, COMPLETED, REJECTED 또는 CANCELED로 완료되면 이벤트가 시작됩니다. 코어 디바이스에서 배포에 실패하면 배포 실패 이유가 설명되는 자세한 응답을 수신됩니다. 배포 오류 코드에 대한 자세한 내용은 [세부 배포 오류 코드](#) 섹션을 참조하세요.

배포 상태

- FAILED. 배포가 실패했습니다.
- SUCCEEDED. 사물 그룹이 대상인 배포가 완료되었습니다.
- COMPLETED. 사물이 대상인 배포가 완료되었습니다.
- REJECTED. 배포가 거부되었습니다. 자세한 내용은 `statusDetails` 필드를 참조하세요.
- CANCELED. 사용자가 배포를 취소했습니다.

이벤트가 중복되거나 이벤트 순서가 잘못되었을 수 있습니다. 이벤트 순서를 정하려면 `time` 속성을 사용하세요.

`errorStacks` 및 `errorTypes`의 오류 코드 전체 목록은 [세부 배포 오류 코드](#) 및 [세부 구성 요소 상태 코드](#) 섹션을 참조하세요.

구성 요소 상태 변경 이벤트

AWS IoT Greengrass 버전 2.12.2 이하의 경우 구성 요소가 `ERRORED` 및 상태가 되면 Greengrass에서 이벤트를 내보냅니다. `BROKEN`. Greengrass nucleus 버전 2.12.3 이상의 경우 구성 요소가 `BROKEN` 및 `ERRORED`, `RUNNING`, `FINISHED` 상태로 전환되면 Greengrass에서 이벤트가 발생합니다. 배포가 완료 되어도 Greengrass에서 이벤트가 발생합니다. 모든 상태 전환 또는 지정한 상태로의 전환에 대해 실행 하는 EventBridge 규칙을 생성할 수 있습니다. 설치된 구성 요소가 규칙이 시작되는 상태로 전환되면 EventBridge에서는 규칙에 정의된 대상 작업이 간접 호출됩니다. 이렇게 하면 알림을 전송하고, 이벤트 정보를 캡처하고, 적절한 조치를 취하거나, 상태 변경에 대응하는 기타 이벤트를 시작할 수 있습니다.

구성 요소 상태 변경에 대한 [이벤트](#)에서 사용되는 형식은 다음과 같습니다.

Greengrass nucleus v2.12.2 and earlier

<title>구성 요소 상태: `ERRORED` 또는 `BROKEN`</title>

```
{
  "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."
    }
  ]
}
}

```

Greengrass nucleus v2.12.3 and later

<title>구성 요소 상태: ERRORED 또는 BROKEN</title>

```

{
  "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."
      }
    ]
  }
}

```

<title>구성 요소 상태: RUNNING 또는 FINISHED</title>

```

{
  "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": "RUNNING|FINISHED",
          "lifecycleStateDetails": null
        }
      ]
    }
  }
}

```

설치된 구성 요소의 상태에 대한 업데이트가 제공되는 규칙과 이벤트를 생성할 수 있습니다. 디바이스에서 구성 요소 상태가 변경되면 이벤트가 시작됩니다. 구성 요소에 오류가 있거나 연결이 끊어진 이유가 설명되는 자세한 응답이 수신됩니다. 실패 사유를 나타내는 상태 코드도 수신됩니다. 구성 요소 상태 코드에 대한 자세한 내용은 [세부 구성 요소 상태 코드](#) 섹션을 참조하세요.

EventBridge 규칙 생성을 위한 사전 조건

에 대한 EventBridge 규칙을 생성하기 전에 다음을 AWS IoT Greengrass 수행합니다.

- Eventbridge의 이벤트, 규칙, 대상을 숙지해야 합니다.
- EventBridge 규칙에 의해 간접 호출되는 대상을 생성하고 구성해야 합니다. 규칙은 다음을 비롯한 다양한 유형의 대상을 간접 호출할 수 있습니다.
 - Amazon Simple Notification Service(Amazon SNS)
 - AWS Lambda 함수
 - Amazon Kinesis Video Streams
 - Amazon Simple Queue Service(Amazon SQS) 대기열

자세한 내용은 Amazon EventBridge 사용 설명서의 [Amazon EventBridge란?](#) 및 [Amazon EventBridge 시작하기](#) 섹션을 참조하세요.

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 사용 설명서의 AWS 리소스에서 이벤트를 트리거하는 EventBridge 규칙 생성](#)을 참조하세요. EventBridge

1. [Amazon EventBridge 콘솔](#)을 엽니다.
2. 탐색 창에서 규칙을 선택합니다.
3. 규칙 생성을 선택합니다.
4. 규칙에 대해 이름과 설명을 입력하세요.

규칙은 동일한 리전과 동일한 이벤트 버스의 다른 규칙과 동일한 이름을 가질 수 없습니다.

5. 이벤트 버스에서 이 규칙과 연결할 이벤트 버스를 선택합니다. 이 규칙이 자신의 계정에서 발생하는 이벤트와 일치하도록 하려면 AWS 기본 이벤트 버스(default event bus)를 선택합니다. 계정의 AWS 서비스가 이벤트를 내보내면 항상 계정의 기본 이벤트 버스로 이동합니다.
6. 규칙 유형(Rule type)에서 이벤트 패턴이 있는 규칙(Rule with an event pattern)을 생성합니다.
7. 다음을 선택합니다.
8. 이벤트 소스에서 AWS 이벤트를 선택합니다.
9. 이벤트 패턴의 경우, AWS 서비스를 선택합니다.
10. AWS 서비스 이름에서 Greengrass를 선택합니다.
11. 이벤트 유형의 경우 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 AWS 서비스를 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 구성(CLI)

다음 단계를 사용하여 Greengrass 상태 변경 이벤트가 있을 때 Amazon SNS 주제가 게시되는 EventBridge 규칙을 생성합니다. 이렇게 하면 웹 서버, 이메일 주소 및 기타 주제 구독자가 이벤트에 대응할 수 있습니다.

1. 규칙을 생성합니다.

- 그룹 배포 상태 변경 이벤트의 경우.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":  
  [\"Greengrass V2 Effective Deployment Status Change\"]}"
```

- 구성 요소 상태 변경 이벤트의 경우.

```
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

Amazon EventBridge가 대상 주제를 호출하도록 허용하려면 주제에 리소스 기반 정책을 추가해야 합니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 [Amazon SNS 권한](#) 섹션을 참조하세요.

자세한 내용은 Amazon EventBridge 사용 설명서에서 [EventBridge의 이벤트 및 이벤트 패턴](#) 섹션을 참조하세요.

디바이스 상태 알림 구성(CloudFormation)

CloudFormation 템플릿을 사용하여 Greengrass 그룹 배포의 상태 변경에 대한 알림을 보내는 EventBridge 규칙을 생성합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 [Amazon EventBridge 리소스 유형 참조](#)를 참조하세요.

다음 사항도 참조하세요.

- [디바이스 배포 상태 확인](#)
- Amazon EventBridge 사용 설명서의 [Amazon EventBridge란?](#) 섹션을 참조하세요.

Greengrass 코어 디바이스 상태 확인

Greengrass 코어 디바이스는 소프트웨어 구성 요소의 상태들에 보고합니다 AWS IoT Greengrass. 각 디바이스의 상태 요약 확인할 수 있으며, 각 디바이스의 각 구성 요소 상태를 확인할 수 있습니다.

코어 디바이스에는 다음 상태가 있습니다.

- HEALTHY - AWS IoT Greengrass 코어 소프트웨어 및 모든 구성 요소는 코어 디바이스에서 문제 없이 실행됩니다.
- UNHEALTHY - AWS IoT Greengrass 코어 소프트웨어 또는 구성 요소가 코어 디바이스에서 오류 상태입니다.

Note

AWS IoT Greengrass 는 개별 디바이스를 사용하여 상태 업데이트를 전송합니다 AWS 클라우드. AWS IoT Greengrass 코어 소프트웨어가 디바이스에서 실행되지 않거나 디바이스가

연결되지 않은 경우 해당 디바이스의 AWS 클라우드보고된 상태가 현재 상태를 반영하지 않을 수 있습니다. 상태 타임스탬프는 디바이스 상태가 마지막으로 업데이트된 시점을 나타냅니다. 코어 디바이스에서는 다음 시점에 상태 업데이트를 보냅니다.

- AWS IoT Greengrass 코어 소프트웨어가 시작될 때
- 코어 디바이스에서 배포를 수신하는 경우 AWS 클라우드
- Greengrass nucleus 2.12.2 이하의 경우 구성 요소의 상태가 **ERRORED** 또는 **BROKEN**으로 되면 코어 디바이스에서 상태 업데이트 발송
- Greengrass nucleus 2.12.3 이상의 경우 구성 요소의 상태가 **ERRORED**, **BROKEN**, **RUNNING** 또는 **FINISHED**로 되면 코어 디바이스에서 상태 업데이트 발송
- [구성할 수 있는 정기적인 간격](#) 시(기본값 24시간)

AWS IoT Greengrass 코어 v2.7.0 이상의 경우 로컬 배포 및 클라우드 배포가 발생할 때 코어 디바이스가 상태 업데이트를 전송합니다.

주제

- [코어 디바이스 상태 확인](#)
- [코어 디바이스 그룹 상태 확인](#)
- [코어 디바이스 구성 요소 상태 확인](#)

코어 디바이스 상태 확인

개별 코어 디바이스의 상태를 확인할 수 있습니다.

코어 디바이스 상태 확인 방법(AWS CLI)

- 다음 명령을 실행하여 디바이스의 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 *coreDeviceName*을 바꿉니다.

```
aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
```

상태를 포함한 코어 디바이스에 대한 정보가 응답에 있습니다.

코어 디바이스 그룹 상태 확인

코어 디바이스 그룹(사물 그룹)의 상태를 확인할 수 있습니다.

디바이스 그룹 상태 확인 방법(AWS CLI)

- 다음 명령을 실행하여 여러 코어 디바이스의 상태를 검색합니다. 쿼리할 사물 그룹의 ARN으로 명령의 ARN을 바꿉니다.

```
aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
```

응답에 사물 그룹의 코어 디바이스 목록이 있습니다. 목록의 각 항목에 코어 디바이스의 상태가 있습니다.

코어 디바이스 구성 요소 상태 확인

코어 디바이스의 소프트웨어 구성 요소 상태(예: 수명 주기 상태)를 확인할 수 있습니다. 구성 요소 수명 주기 상태에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#) 섹션을 참조하세요.

코어 디바이스의 구성 요소 상태 확인 방법(AWS CLI)

- 다음 명령을 실행하여 코어 디바이스의 구성 요소 상태를 검색합니다. 쿼리할 코어 디바이스의 이름으로 `coreDeviceName`을 바꿉니다.

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

응답에 코어 디바이스에서 실행되는 구성 요소 목록이 있습니다. 목록의 각 항목에 데이터의 현재 상태 및 특정 구성 요소가 들어 있는 메시지를 마지막으로 Greengrass 코어 디바이스에서 클라우드로 보낸 시점을 포함한 구성 요소 수명 주기 상태가 있습니다. 구성 요소를 Greengrass 코어 디바이스로 가져온 최종 배포 소스도 응답에 포함되어 있습니다.

Note

이 명령에서는 Greengrass 코어 디바이스에서 실행되는 페이지가 매겨진 구성 요소 목록을 검색합니다. 기본적으로 다른 구성 요소의 종속성으로 배포된 구성 요소는 이 목록에 포함되지 않습니다. `topologyFilter` 파라미터를 ALL로 설정하여 종속성을 응답에 포함할 수 있습니다.

AWS Lambda 함수 실행

Note

AWS IoT Greengrass는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

AWS IoT Greengrass 코어 디바이스에서 실행되는 구성 요소로 AWS Lambda 함수를 가져올 수 있습니다. 다음 경우에 이를 수행할 수 있습니다.

- 코어 디바이스에 배포하려는 Lambda 함수에 애플리케이션 코드가 있습니다.
- AWS IoT Greengrass V2 코어 디바이스에서 실행하려는 AWS IoT Greengrass V1 애플리케이션이 있습니다. 자세한 내용은 [2단계: AWS IoT Greengrass V1 애플리케이션 마이그레이션을 위한 AWS IoT Greengrass V2 구성 요소 생성 및 배포](#) 섹션을 참조하세요.

Lambda 함수에는 다음 구성 요소에 대한 종속성이 포함되어 있습니다. 함수를 가져오는 경우 이러한 구성 요소를 종속성으로 정의할 필요는 없습니다. Lambda 함수 구성 요소를 배포하는 경우 배포에는 이러한 Lambda 구성 요소 종속성이 포함되어 있습니다.

- [Lambda 런처 구성 요소](#)(aws.greengrass.LambdaLauncher)는 프로세스와 환경 구성을 처리합니다.
- [Lambda 관리자 구성 요소](#)(aws.greengrass.LambdaManager)는 프로세스 간 통신 및 스케일링을 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 컨테이너화 구성](#)
- [Lambda 함수를 구성 요소로 가져오기\(콘솔\)](#)
- [Lambda 함수를 구성 요소로 가져오기\(AWS CLI\)](#)

요구 사항

AWS IoT Greengrass 코어 소프트웨어에서 함수를 실행하려면 코어 디바이스와 Lambda 함수가 다음과 같은 요구 사항을 충족해야 합니다.

- Lambda 함수를 실행하려면 코어 디바이스가 요구 사항을 충족해야 합니다. 코어 디바이스에서 컨테이너화된 Lambda 함수를 실행하려면 디바이스가 요구 사항을 충족해야 합니다. 자세한 내용은 [Lambda 함수 요구 사항](#) 섹션을 참조하세요.
- 코어 디바이스에 Lambda 함수가 사용하는 프로그래밍 언어를 설치해야 합니다.

Tip

프로그래밍 언어를 설치하는 구성 요소를 생성한 다음 해당 구성 요소를 Lambda 함수 구성 요소의 종속성으로 지정할 수 있습니다. Greengrass는 Python, Node.js, Java 런타임의 모든 Lambda 지원 버전을 지원합니다. Greengrass는 더 이상 사용되지 않는 Lambda 런타임 버전에 추가 제한 사항을 적용하지 않습니다. AWS IoT Greengrass에서는 이러한 더 이상 사용되지 않는 런타임을 사용하는 Lambda 함수를 실행할 수 있지만 AWS Lambda에서 이를 생성할 수는 없습니다. Lambda 런타임용 AWS IoT Greengrass 지원에 대한 자세한 내용은 [AWS Lambda 함수 실행](#) 섹션을 참조하세요.

Lambda 함수 수명 주기 구성

Lambda 함수 수명 주기에 따라 함수가 시작되는 시점과 컨테이너를 생성 및 사용하는 방법이 결정됩니다. 수명 주기에 따라 AWS IoT Greengrass 코어 소프트웨어가 함수 핸들러 외부에 있는 변수와 사전 처리 논리를 유지하는 방법도 결정됩니다.

AWS IoT Greengrass는 온디맨드(기본값)와 수명이 긴 수명 주기를 지원합니다.

- 온디맨드 함수는 호출되면 시작되고 실행할 태스크가 남아 있지 않을 때 종료됩니다. 함수를 간접적으로 호출할 때마다 샌드박스라고도 하는 별도의 컨테이너가 생성되어 간접 호출을 처리합니다. 단, 기존 컨테이너를 재사용할 수 있는 경우는 예외입니다. 모든 컨테이너는 함수에 전송하는 데이터를 처리할 수 있습니다.

온디맨드 함수의 다중 호출은 동시에 실행할 수 있습니다.

함수 핸들러 외부에서 정의한 변수 및 사전 처리 논리는 새 컨테이너가 생성될 때 유지되지 않습니다.

- 수명이 긴(또는 고정된) 함수는 AWS IoT Greengrass 코어 소프트웨어가 시작될 때 시작되며 단일 컨테이너에서 실행됩니다. 동일한 컨테이너가 함수에 전송하는 모든 데이터를 처리합니다.

AWS IoT Greengrass 코어 소프트웨어가 이전 호출을 실행할 때까지 다중 호출은 대기 상태가 됩니다.

함수 핸들러 외부에서 정의한 변수 및 사전 처리 논리는 모든 핸들러 호출 시 유지됩니다.

초기 입력 없이 작업을 시작해야 하는 경우에는 수명이 긴 Lambda 함수를 사용합니다. 예를 들어 수명이 긴 함수는 기계 학습 모델을 로드하고 처리하기 시작하여 함수가 디바이스 데이터를 수신하는 경우 준비된 상태가 되도록 할 수 있습니다.

Note

수명이 긴 함수는 핸들러 호출할 때마다 적용되는 제한 시간이 있습니다. 무기한으로 실행되는 코드를 호출하려면 핸들러 외부에서 시작해야 합니다. 핸들러 외부에 함수 초기화를 방지할 수 있는 차단 코드가 없는지 확인하세요.

이러한 함수는 배포 도중 또는 재부팅과 같이 AWS IoT Greengrass 코어 소프트웨어가 중지되지 않는 한 실행됩니다. 이러한 함수는 함수에 잡히지 않은 예외가 발생하거나 메모리 제한을 초과하거나 핸들러 제한 시간과 같은 오류 상태가 발생하는 경우에는 실행되지 않습니다.

컨테이너 재사용에 대한 자세한 내용은 AWS 컴퓨팅 블로그의 [AWS Lambda에서 컨테이너 재사용 이해하기](#)를 참조하세요.

Lambda 함수 컨테이너화 구성

기본적으로 Lambda 함수는 AWS IoT Greengrass 컨테이너의 내부에서 실행됩니다. Greengrass 컨테이너는 함수와 호스트 사이를 격리합니다. 이러한 격리는 호스트와 컨테이너의 함수 모두에 대한 보안을 강화합니다.

컨테이너화 없이 실행해야 하는 사용 사례가 아니라면 Lambda 함수를 Greengrass 컨테이너에서 실행하는 것이 좋습니다. Greengrass 컨테이너에서 Lambda 함수를 실행하면 리소스에 대한 액세스를 제한하는 방법을 더 잘 제어할 수 있습니다.

다음 경우에는 컨테이너화하지 않고 Lambda 함수를 실행할 수 있습니다.

- 컨테이너 모드를 지원하지 않는 디바이스에서 AWS IoT Greengrass를 실행하려고 합니다. 특수한 Linux 배포를 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 시점에 경로를 결정할 수 없거나 배포 이후 경로가 변경될 수 있는 로컬 리소스에 액세스해야 합니다. 해당 리소스의 예로 플러그형 디바이스를 들 수 있습니다.
- 프로세스로 작성된 이전 애플리케이션이 있고 이를 Greengrass 컨테이너에서 실행하는 경우 문제가 발생합니다.

컨테이너화의 차이점

컨테이너화	Notes
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. 다시 말해 Lambda 함수는 상호 격리가 강화된 상태로 실행됩니다. • AWS IoT Greengrass 코어 소프트웨어는 Lambda 함수와 동일한 컨테이너에서 모든 하위 프로세스를 실행하므로 Lambda 함수가 중지되면 하위 프로세스도 중지됩니다.
컨테이너 없음	• 컨테이너화되지 않은 Lambda 함수에서는 다음 기능을 사용할 수 없습니다. • Lambda 함수 메모리 제한. • 로컬 장치 및 볼륨 리소스. 이러한 리소스는 Lambda 함수 리소스가 아닌 코어 디바이스의 파일 경로를 사용하여 액세스해야 합니다. • 컨테이너화되지 않은 Lambda 함수가 기계 학습 리소스에 액세스하는 경우 리소스 소유자

컨테이너화	Notes
	를 식별하고 Lambda 함수가 아닌 리소스에 대한 액세스 권한을 설정해야 합니다. 컨테이너화되지 않은 Lambda 함수에는 동일한 시스템 그룹으로 실행되는 다른 Lambda 함수의 배포된 코드에 읽기 전용 액세스 권한이 있습니다.

Lambda 함수를 배포할 때 컨테이너화를 변경하면 함수가 예상대로 작동하지 않을 수 있습니다. Lambda 함수가 새로운 컨테이너화 설정에서 더 이상 사용할 수 없는 로컬 리소스를 사용하는 경우 배포가 실패합니다.

- Lambda 함수를 Greengrass 컨테이너에서 실행하는 방식에서 컨테이너화 없이 실행하는 방식으로 변경하면 해당 함수의 메모리 제한이 삭제됩니다. 연결된 로컬 리소스를 사용하기보다 파일 시스템에 직접 액세스해야 합니다. Lambda 함수를 배포하기 전에 연결된 모든 리소스를 제거해야 합니다.
- Lambda 함수를 컨테이너화 없이 실행하는 방식에서 컨테이너에서 실행하는 방식으로 변경하면 Lambda 함수는 파일 시스템에 대한 직접 액세스 권한을 상실합니다. 각 함수에 대한 메모리 제한을 정의하거나 기본 16MB 메모리 제한을 수락해야 합니다. 배포 시 각 Lambda 함수에 대해 이러한 설정을 구성할 수 있습니다.

Lambda 함수 구성 요소의 컨테이너화 설정을 변경하려면 구성 요소 배포 시 `containerMode` 구성 파라미터의 값을 다음 옵션 중 하나로 설정합니다.

- `NoContainer` – 구성 요소가 격리된 런타임 환경에서 실행되지 않습니다.
- `GreengrassContainer` – 구성 요소가 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 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

Lambda 함수를 구성 요소로 가져오기(콘솔)

[AWS IoT Greengrass 콘솔](#)을 사용하여 Lambda 함수 구성 요소를 생성할 때 기존 AWS Lambda 함수를 가져온 다음 Greengrass 디바이스에서 실행되는 구성 요소를 생성하도록 구성합니다.

시작하기 전에 Greengrass 디바이스에서 Lambda 함수 실행 [요구 사항](#)을 검토하세요.

작업

- [1단계: 가져올 Lambda 함수 선택](#)
- [2단계: Lambda 함수 파라미터 구성](#)
- [3단계: \(선택 사항\) Lambda 함수에 지원되는 플랫폼 지정](#)
- [4단계: \(선택 사항\) Lambda 함수의 구성 요소 종속성 지정](#)
- [5단계: \(선택 사항\) 컨테이너에서 Lambda 함수 실행](#)
- [6단계: Lambda 함수 구성 요소 생성](#)

1단계: 가져올 Lambda 함수 선택

1. [AWS IoT Greengrass 콘솔](#) 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 검색하고 선택합니다.

AWS IoT Greengrass 는 Lambda 함수의 이름으로 구성 요소를 생성합니다.

5. Lambda 함수 버전에서 가져올 버전을 선택합니다. \$LATEST 등과 같은 Lambda 별칭은 선택할 수 없습니다.

AWS IoT Greengrass 는 Lambda 함수 버전을 유효한 의미 체계 버전으로 사용하여 구성 요소를 생성합니다. 예를 들어 함수 버전이 3인 경우 구성 요소 버전은 3.0.0가 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. 이벤트 소스를 지정하여이 함수가 로컬 게시/구독 메시지 및 AWS IoT Core MQTT 메시지를 구독하도록 할 수 있습니다. Lambda 함수는 이벤트 소스에서 메시지를 수신할 때 직접 호출됩니다.

Note

이 함수를 다른 Lambda 함수 또는 구성 요소의 메시지에 구독하려면 이 Lambda 함수 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)를 배포합니다. 레거시 구독 라우터 구성 요소를 배포하는 경우 Lambda 함수가 사용하는 구독을 지정합니다.

이벤트 소스에서 다음을 수행하여 이벤트 소스 추가:

a. 추가하는 각 이벤트 소스에 대해 다음 옵션 지정:

- 주제 - 메시지에 대해 구독하는 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 주제에서 MQTT 주제 와일드카드(+ 및 #)를 사용할 수 있습니다.

- AWS IoT Core MQTT - AWS IoT Core MQTT 메시지를 구독합니다.

이 유형을 지정할 때 주제에서 MQTT 주제 와일드카드(+ 및 #)를 사용할 수 있습니다.

b. 다른 이벤트 소스를 추가하려면 이벤트 소스 추가를 선택한 다음 이전 단계를 반복합니다. 이벤트 소스를 제거하려면 제거하려는 이벤트 소스 옆에 있는 제거를 선택합니다.

- 제한 시간(초)에 고정되지 않은 Lambda 함수가 제한 시간이 되기 전에 먼저 실행될 수 있는 최대 시간을 초 단위로 입력합니다. 기본값은 3초입니다.
- 고정에서 Lambda 함수 구성 요소가 고정되었는지 여부를 선택합니다. 기본값은 True입니다.
 - 고정(또는 수명이 긴) Lambda 함수는 AWS IoT Greengrass 시작되어 자체 컨테이너에서 계속 실행될 때 시작됩니다.
 - 고정되지 않은(또는 온디맨드) Lambda 함수는 작업 항목을 수신한 경우에만 시작되고 지정된 최대 유휴 시간 동안 유휴 상태로 유지된 후에 종료됩니다. 함수에 여러 작업 항목이 있는 경우 AWS IoT Greengrass 코어 소프트웨어는 함수의 여러 인스턴스를 생성합니다.
- (선택 사항) 추가 파라미터에서 다음 Lambda 함수 파라미터를 설정합니다.
 - 상태 제한 시간(초) - Lambda 함수 구성 요소가 상태 업데이트를 Lambda 관리자 구성 요소로 전송하는 간격(초)입니다. 이 파라미터는 고정된 함수에만 적용됩니다. 기본값은 60초입니다.

- 최대 대기열 크기 - Lambda 함수 구성 요소에 대한 메시지 대기열의 최대 크기입니다. AWS IoT Greengrass 코어 소프트웨어는 Lambda 함수를 실행하여 각 메시지를 사용할 수 있을 때까지 메시지를 FIFO(선입선출) 대기열에 저장합니다. 기본값은 1,000개 메시지입니다.
- 최대 인스턴스 수 - 고정되지 않은 Lambda 함수가 동시에 실행할 수 있는 최대 인스턴스 수입니다. 기본값은 인스턴스 100개입니다.
- 최대 유휴 시간(초) - AWS IoT Greengrass 코어 소프트웨어가 프로세스를 중지하기 전에 고정되지 않은 Lambda 함수가 유휴 상태가 될 수 있는 초 단위의 최대 시간입니다. 기본값은 60초입니다.
- 인코딩 유형 - Lambda 함수가 지원하는 페이로드의 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 전달할 인수를 입력합니다.
 - c. 인수를 제거하려면 제거하려는 인수 옆에 있는 제거를 선택합니다.
6. (선택 사항) Lambda 함수가 실행될 때 사용할 수 있는 환경 변수를 지정합니다. 환경 변수를 사용하면 함수 코드를 변경할 필요 없이 구성 설정을 저장하고 업데이트할 수 있습니다.
 - a. 추가 파라미터, 환경 변수에서 환경 변수 추가를 선택합니다.
 - b. 추가하는 각 환경 변수에 대해 다음 옵션을 지정합니다.
 - 키 - 변수 이름.
 - 값 - 이 변수의 기본값.
 - c. 환경 변수를 제거하려면 제거하려는 환경 변수 옆에 있는 제거를 선택합니다.

3단계: (선택 사항) Lambda 함수에 지원되는 플랫폼 지정

모든 코어 디바이스에는 운영 체제 및 아키텍처에 대한 속성이 있습니다. Lambda 함수 구성 요소를 배포할 때 AWS IoT Greengrass 코어 소프트웨어는 지정한 플랫폼 값을 코어 디바이스의 플랫폼 속성과 비교하여 Lambda 함수가 해당 디바이스에서 지원되는지 여부를 결정합니다.

Note

Greengrass nucleus 구성 요소를 코어 디바이스에 배포할 때 사용자 지정 플랫폼 속성을 지정할 수 있습니다. 자세한 내용은 [Greengrass nucleus 구성 요소의 플랫폼 재정의 파라미터](#)를 참조하세요.

Lambda 함수 구성, 추가 파라미터, 플랫폼에서 다음을 수행하여 이 Lambda 함수가 지원하는 플랫폼을 지정합니다.

1. 각 플랫폼에 대해 다음 옵션을 지정합니다.

- 운영 체제 - 플랫폼의 운영 체제 이름입니다. 현재 지원되는 값은 linux입니다.
- 아키텍처 - 플랫폼의 프로세서 아키텍처입니다. 지원되는 값은 다음과 같습니다.
 - amd64
 - arm
 - aarch64
 - x86

2. 다른 플랫폼을 추가하려면 플랫폼 추가를 선택하고 이전 단계를 반복합니다. 지원되는 플랫폼을 제거하려면 제거할 플랫폼 옆에 있는 제거를 선택합니다.

4단계: (선택 사항) Lambda 함수의 구성 요소 종속성 지정

구성 요소 종속성은 함수가 사용하는 추가 AWS제공 구성 요소 또는 사용자 지정 구성 요소를 식별합니다. Lambda 함수 구성 요소를 배포할 때 배포에는 함수에서 실행하는 이러한 종속성이 포함됩니다.

Important

AWS IoT Greengrass V1에서 실행하기 위해 생성한 Lambda 함수를 가져오려면 보안 암호, 로컬 새도우, 스트림 관리자 등 함수가 사용하는 기능에 대한 개별 구성 요소 종속성을 정의해야 합니다. 종속성 상태가 변경되는 경우 Lambda 함수 구성 요소가 다시 시작되도록 이러한 구성 요소를 [하드 종속성](#)으로 정의합니다. 자세한 내용은 [V1 Lambda 함수 가져오기](#) 단원을 참조하십시오.

Lambda 함수 구성, 추가 파라미터, 구성 요소 종속성에서 다음 단계를 완료하여 Lambda 함수의 구성 요소 종속성을 지정합니다.

1. 종속성 추가를 선택합니다.
2. 추가하는 각 구성 요소 종속성에 대해 다음 옵션을 지정합니다.
 - 구성 요소 이름 - 구성 요소 이름입니다. 예를 들어 **aws.greengrass.StreamManager**를 입력하여 [스트림 관리자 구성 요소](#)를 포함합니다.
 - 버전 요구사항 - 구성 요소 종속성의 호환 버전을 식별하는 npm 스타일 의미 체계 버전 제약 조건입니다. 단일 버전 또는 여러 버전 범위를 지정할 수 있습니다. 예를 들어 **^1.0.0**을 입력하여 이 Lambda 함수가 스트림 관리자 구성 요소의 첫 번째 메이저 버전에 따라 달라지도록 지정합니다. 의미 체계 버전 제약 조건에 대한 자세한 내용은 [npm semver 계산기](#)를 참조하세요.
 - 유형 - 종속성의 유형입니다. 다음 옵션 중 하나를 선택합니다.
 - 하드 - 종속성 상태가 변경되면 Lambda 함수 구성 요소가 다시 시작됩니다. 이는 기본 선택입니다.
 - 소프트 - 종속성 상태가 변경되면 Lambda 함수 구성 요소가 다시 시작되지 않습니다.
3. 구성 요소 종속성을 제거하려면 구성 요소 종속성 옆에 있는 제거를 선택합니다.

5단계: (선택 사항) 컨테이너에서 Lambda 함수 실행

기본적으로 Lambda 함수는 AWS IoT Greengrass 코어 소프트웨어 내의 격리된 런타임 환경에서 실행됩니다. 격리 없이(컨테이너 없음 모드) Lambda 함수를 프로세스로 실행하도록 선택할 수도 있습니다.

Linux 프로세스 구성의 격리 모드에서 다음 옵션 중에서 선택하여 Lambda 함수의 컨테이너화 선택:

- Greengrass 컨테이너 - Lambda 함수가 컨테이너에서 실행됩니다. 이는 기본 선택입니다.
- 컨테이너 없음 - Lambda 함수가 격리 없이 프로세스로 실행됩니다.

컨테이너에서 Lambda 함수를 실행하는 경우 다음 단계를 완료하여 Lambda 함수에 대한 프로세스 구성을 구성합니다.

1. 컨테이너에서 사용할 수 있는 메모리의 양과 시스템 리소스(예: 볼륨 및 디바이스)를 구성할 수 있습니다.

컨테이너 파라미터에서 다음을 수행합니다.

- a. 메모리 크기에 컨테이너에 할당할 메모리 크기를 입력합니다. 메모리 크기를 MB 또는 KB 단위로 지정할 수 있습니다.
 - b. 읽기 전용 sys 폴더에서 컨테이너가 디바이스의 /sys 폴더에서 정보를 읽을 수 있는지 여부를 선택합니다. 기본값은 False입니다.
2. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 볼륨을 구성합니다. 볼륨을 정의하면 AWS IoT Greengrass 코어 소프트웨어가 소스 파일을 컨테이너 내부의 대상에 탑재합니다.
 - a. 볼륨에서 볼륨 추가를 선택합니다.
 - b. 추가하는 각 볼륨에 대해 다음 옵션 지정:
 - 물리적 볼륨 - 코어 디바이스의 소스 폴더 경로입니다.
 - 논리적 볼륨 - 컨테이너의 대상 폴더 경로입니다.
 - 권한 - (선택 사항) 컨테이너에서 소스 폴더에 액세스할 수 있는 권한입니다. 다음 옵션 중 하나를 선택합니다.
 - 읽기 전용 - Lambda 함수에 소스 폴더에 대한 읽기 전용 액세스 권한이 있습니다. 이는 기본 선택입니다.
 - 읽기-쓰기 - Lambda 함수에 소스 폴더에 대한 읽기-쓰기 액세스 권한이 있습니다.
 - 그룹 소유자 추가 - (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 소스 폴더의 소유자로 추가할지 여부입니다. 기본값은 False입니다.
 - c. 볼륨을 제거하려면 제거하려는 볼륨 옆에 있는 제거를 선택합니다.
3. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 시스템 디바이스를 구성합니다.
 - a. 디바이스에서 디바이스 추가를 선택합니다.
 - b. 추가하는 각 디바이스에 대해 다음 옵션 지정:
 - 탑재 경로 - 코어 디바이스의 시스템 디바이스 경로입니다.
 - 권한 - (선택 사항) 컨테이너에서 시스템 디바이스에 액세스할 수 있는 권한입니다. 다음 옵션 중 하나를 선택합니다.
 - 읽기 전용 - Lambda 함수에 시스템 디바이스에 대한 읽기 전용 액세스 권한이 있습니다. 이는 기본 선택입니다.
 - 읽기-쓰기 - Lambda 함수에 소스 폴더에 대한 읽기-쓰기 액세스 권한이 있습니다.
 - 그룹 소유자 추가 - (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 시스템 디바이스의 소유자로 추가할지 여부입니다. 기본값은 False입니다.

6단계: Lambda 함수 구성 요소 생성

Lambda 함수 구성 요소에 대한 설정을 구성한 후 생성을 선택하여 새 구성 요소 생성을 완료합니다.

코어 디바이스에서 Lambda 함수를 실행하기 위해 코어 디바이스에 새 구성 요소를 배포할 수 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

Lambda 함수를 구성 요소로 가져오기(AWS CLI)

[CreateComponentVersion](#) 작업을 사용하여 Lambda 함수에서 구성 요소를 생성합니다. 이 작업을 직접적으로 호출할 때 `lambdaFunction`을 지정하여 Lambda 함수를 가져옵니다.

작업

- [1단계: Lambda 함수 구성 정의](#)
- [2단계: Lambda 함수 구성 요소 생성](#)

1단계: Lambda 함수 구성 정의

1. `lambda-function-component.json`이라는 파일을 만들고 다음 JSON을 파일로 복사합니다. `lambdaArn`을 가져올 Lambda 함수의 ARN으로 바꿉니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
  }
}
```

Important

가져올 함수의 버전이 포함된 ARN을 지정해야 합니다. `$LATEST`같은 버전 별칭을 사용할 수 없습니다.

2. (선택 사항) 구성 요소의 이름(`componentName`)을 지정합니다. 이 파라미터를 생략하면가 Lambda 함수의 이름으로 구성 요소를 AWS IoT Greengrass 생성합니다.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
```

```

    "componentName": "com.example.HelloWorldLambda"
  }
}

```

3. (선택 사항) 구성 요소의 버전(componentVersion)을 지정합니다. 이 파라미터를 생략하면는 Lambda 함수 버전을 유효한 의미 체계 버전으로 사용하여 구성 요소를 AWS IoT Greengrass 생성합니다. 예를 들어 함수 버전이 3인 경우 구성 요소 버전은 3.0.0가 됩니다.

Note

업로드하는 각 구성 요소 버전은 고유해야 합니다. 업로드한 후에는 편집할 수 없으므로 올바른 구성 요소 버전을 업로드해야 합니다.

AWS IoT Greengrass 는 구성 요소에 의미 체계 버전을 사용합니다. 시맨틱 버전은 메이저.마이너.패치 번호 시스템을 따릅니다. 예를 들어 1.0.0 버전은 구성 요소의 첫 번째 주요 릴리스를 나타냅니다. 자세한 내용은 [의미 체계 버전 사양](#)을 참조하세요.

```

{
  "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 코어 디바이스에는 운영 체제(os) 및 아키텍처(architecture)에 대한 속성이 있습니다. AWS IoT Greengrass 코어 소프트웨어는 다른 플랫폼 속성을 추가할 수 있습니다. [Greengrass nucleus 구성 요소](#)를 코어 디바이스에 배포할 때 사용자 지정 플랫폼 속성을 지정할 수 있습니다. 해결 방법:
 - a. lambda-function-component.json의 Lambda 함수에 플랫폼 목록(componentPlatforms)을 추가합니다.

```

{
  "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 식별할 수 있는 친숙한 name 및 속성 맵이 있습니다. 다음 예제에서는 이 함수가 Linux를 실행하는 x86 디바이스를 지원하도록 지정합니다.

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

lambda-function-component.json에는 다음 예제와 유사한 문서가 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"
        }
      }
    ]
  }
}
```

5. (선택 사항) Lambda 함수의 구성 요소 종속성을 지정합니다. Lambda 함수 구성 요소를 배포할 때 배포에는 함수에서 실행하는 이러한 종속성이 포함됩니다.

⚠ Important

AWS IoT Greengrass V1에서 실행하기 위해 생성한 Lambda 함수를 가져오려면 보안 암호, 로컬 새도우, 스트림 관리자 등 함수가 사용하는 기능에 대한 개별 구성 요소 종속성을 정의해야 합니다. 종속성 상태가 변경되는 경우 Lambda 함수 구성 요소가 다시 시작되도록 이러한 구성 요소를 [하드 종속성](#)으로 정의합니다. 자세한 내용은 [V1 Lambda 함수 가져오기](#) 단원을 참조하십시오.

해결 방법:

- a. `lambda-function-component.json`의 Lambda 함수에 구성 요소 종속성 맵 (`componentDependencies`)을 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": {

    }
  }
}
```

- b. 맵에 각 구성 요소 종속성을 추가합니다. 구성 요소 이름을 키로 지정하고 다음 파라미터를 사용하여 객체 지정:
 - `versionRequirement` – 구성 요소 종속성의 호환 버전을 식별하는 npm 스타일 의미 체계 버전 제약 조건입니다. 단일 버전 또는 여러 버전 범위를 지정할 수 있습니다. 의미 체계 버전 제약 조건에 대한 자세한 내용은 [npm semver 계산기](#)를 참조하세요.

- `dependencyType` – (선택 사항) 종속성의 유형입니다. 다음 중에서 선택합니다.
 - `SOFT` – 종속성 상태가 변경되면 Lambda 함수 구성 요소가 다시 시작되지 않습니다.
 - `HARD` – 종속성 상태가 변경되면 Lambda 함수 구성 요소가 다시 시작됩니다.

기본값은 `HARD`입니다.

다음 예제에서는 이 Lambda 함수가 [스트림 관리자 구성 요소](#)의 첫 번째 메이저 버전에 따라 달라지도록 지정합니다. 스트림 관리자가 다시 시작하거나 업데이트하면 Lambda 함수 구성 요소가 다시 시작됩니다.

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

`lambda-function-component.json`에는 다음 예제와 유사한 문서가 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) 함수 실행에 사용할 Lambda 함수 파라미터를 구성합니다. 환경 변수, 메시지 이벤트 소스, 제한 시간 및 컨테이너 설정과 같은 옵션을 구성할 수 있습니다. 해결 방법:

- a. Lambda 파라미터 객체(componentLambdaParameters)를 `lambda-function-component.json`의 Lambda 함수에 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": {

  }
}
```

- b. (선택 사항) Lambda 함수가 작업 메시지를 구독하는 이벤트 소스를 지정합니다. 이벤트 소스를 지정하여이 함수가 로컬 게시/구독 메시지 및 AWS IoT Core MQTT 메시지를 구독하도록 할 수 있습니다. Lambda 함수는 이벤트 소스에서 메시지를 수신할 때 직접 호출됩니다.

Note

이 함수를 다른 Lambda 함수 또는 구성 요소의 메시지에 구독하려면 이 Lambda 함수 구성 요소를 배포할 때 [레거시 구독 라우터 구성 요소](#)를 배포합니다. 레거시 구독 라우터 구성 요소를 배포하는 경우 Lambda 함수가 사용하는 구독을 지정합니다.

해결 방법:

- i. Lambda 함수 파라미터에 이벤트 소스 목록(eventSources)을 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. 목록에 각 이벤트 소스를 추가합니다. 각 이벤트 소스에는 다음 파라미터가 있습니다.

- `topic` - 메시지에 대해 구독하는 주제입니다.
- `type` - 이벤트 소스의 유형입니다. 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 `topic`에서 MQTT 주제 와일드카드(+ 및 #)를 사용할 수 있습니다.

- `IOT_CORE` - AWS IoT Core MQTT 메시지를 구독합니다.

이 유형을 지정할 때 `topic`에서 MQTT 주제 와일드카드(+ 및 #)를 사용할 수 있습니다.

다음 예시에서는 주제 필터와 일치하는 `hello/world/+` 주제에 대해 AWS IoT Core MQTT를 구독합니다.

```
{
  "topic": "hello/world/",
  "type": "IOT_CORE"
}
```

`lambda-function-component.json`의 경우 다음 예제와 비슷할 것입니다.

```
{
  "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 객체에서 다음 파라미터 중 하나를 지정합니다.

- `environmentVariables` - Lambda 함수가 실행될 때 사용할 수 있는 환경 변수의 맵입니다.
- `execArgs` - Lambda 함수가 실행될 때 전달할 인수 목록입니다.
- `inputPayloadEncodingType` - Lambda 함수가 지원하는 페이로드의 유형입니다. 다음 옵션 중 하나를 선택합니다.
 - `json`
 - `binary`

기본값: `json`

- `pinned` - Lambda 함수가 고정되었는지 여부입니다. 기본값은 `true`입니다.
 - 고정(또는 수명이 긴) Lambda 함수는 AWS IoT Greengrass 시작되어 자체 컨테이너에서 계속 실행될 때 시작됩니다.
 - 고정되지 않은(또는 온디맨드) Lambda 함수는 작업 항목을 수신한 경우에만 시작되고 지정된 최대 유휴 시간 동안 유휴 상태로 유지된 후에 종료됩니다. 함수에 여러 작업 항목이 있는 경우 AWS IoT Greengrass 코어 소프트웨어는 함수의 여러 인스턴스를 생성합니다.

`maxIdleTimeInSeconds`를 사용하여 함수의 최대 유휴 시간을 설정합니다.

- `timeoutInSeconds` - 제한 시간이 되기 전까지 Lambda 함수를 실행할 수 있는 최대 시간(초)입니다. 기본값은 3초입니다.

- `statusTimeoutInSeconds` - Lambda 함수 구성 요소가 상태 업데이트를 Lambda 관리자 구성 요소로 전송하는 간격(초)입니다. 이 파라미터는 고정된 함수에만 적용됩니다. 기본값은 60초입니다.
- `maxIdleTimeInSeconds` - AWS IoT Greengrass 코어 소프트웨어가 프로세스를 중지하기 전에 고정되지 않은 Lambda 함수가 유휴 상태가 될 수 있는 초 단위의 최대 시간입니다. 기본값은 60초입니다.
- `maxInstancesCount` - 고정되지 않은 Lambda 함수가 동시에 실행할 수 있는 최대 인스턴스 수입니다. 기본값은 인스턴스 100개입니다.
- `maxQueueSize` - Lambda 함수 구성 요소에 대한 메시지 대기열의 최대 크기입니다. AWS IoT Greengrass 코어 소프트웨어는 Lambda 함수를 실행하여 각 메시지를 사용할 수 있을 때까지 메시지를 FIFO(first-in-first-out) 대기열에 저장합니다. 기본값은 1,000개 메시지입니다.

`lambda-function-component.json`에는 다음 예제와 유사한 문서가 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"
      }
    },
    "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 AWS IoT Greengrass 코어 소프트웨어 내의 격리된 런타임 환경에서 실행됩니다. 격리 없이 Lambda 함수를 프로세스로 실행하도록 선택할 수도 있습니다. 컨테이너에서 Lambda 함수를 실행하면 컨테이너의 메모리 크기와 Lambda 함수에서 사용할 수 있는 시스템 리소스를 구성합니다. 해결 방법:
- i. Linux 프로세스 파라미터 객체(`linuxProcessParams`)를 `lambda-function-component.json`의 Lambda 파라미터 객체에 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": {
      }
    }
  }
}
}

```

- ii. (선택 사항) Lambda 함수가 컨테이너에서 실행되는지 여부를 지정합니다. 프로세스 파라미터 객체에 `isolationMode` 파라미터를 추가하고, 다음 옵션 중에서 선택합니다.

- `GreengrassContainer` - Lambda 함수가 컨테이너에서 실행됩니다.
- `NoContainer` - Lambda 함수가 격리 없이 프로세스로 실행됩니다.

기본값은 `GreengrassContainer`입니다.

- iii. (선택 사항) 컨테이너에서 Lambda 함수를 실행하는 경우 컨테이너에서 사용할 수 있는 메모리의 양과 시스템 리소스(예: 볼륨 및 디바이스)를 구성할 수 있습니다. 해결 방법:

- A. 컨테이너 파라미터 객체(containerParams)를 lambda-function-component.json의 Linux 프로세스 파라미터 객체에 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": {

            }
        }
    }
}
}
}

```

- B. (선택 사항) `memorySizeInKB` 파라미터를 추가하여 컨테이너의 메모리 크기를 지정합니다. 기본값은 16,384KB(16MB)입니다.
- C. (선택 사항) `mountROSysfs` 파라미터를 추가하여 컨테이너가 디바이스의 `/sys` 폴더에서 정보를 읽을 수 있는지 여부를 지정합니다. 기본값은 `false`입니다.
- D. (선택 사항) 컨테이너화된 Lambda 함수가 액세스할 수 있는 로컬 볼륨을 구성합니다. 볼륨을 정의하면 AWS IoT Greengrass 코어 소프트웨어가 소스 파일을 컨테이너 내부의 대상에 탑재합니다. 해결 방법:

- I. 컨테이너 파라미터에 볼륨 목록(volumes)을 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. 목록에 각 볼륨을 추가합니다. 각 볼륨에는 다음 파라미터가 있습니다.

- **sourcePath** - 코어 디바이스의 소스 폴더 경로입니다.
- **destinationPath** - 컨테이너의 대상 폴더 경로입니다.
- **permission** - (선택 사항) 컨테이너에서 소스 폴더에 액세스할 수 있는 권한입니다. 다음 옵션 중 하나를 선택합니다.
 - **ro** - Lambda 함수에 소스 폴더에 대한 읽기 전용 액세스 권한이 있습니다.

- `rw` - Lambda 함수에 소스 폴더에 대한 읽기-쓰기 액세스 권한이 있습니다.

기본값은 `ro`입니다.

- `addGroupOwner` - (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 소스 폴더의 소유자로 추가할지 여부입니다. 기본값은 `false`입니다.

`lambda-function-component.json`에는 다음 예제와 유사한 문서가 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"
      }
    },
    "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,
        "mountR0Sysfs": 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. 해결 방법:

I. 컨테이너 파라미터에 시스템 디바이스 목록(devices)을 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. 목록에 각 시스템 디바이스를 추가합니다. 각 시스템 디바이스에는 다음 파라미터가 있습니다.

- path - 코어 디바이스의 시스템 디바이스 경로입니다.
- permission - (선택 사항) 컨테이너에서 시스템 디바이스에 액세스할 수 있는 권한입니다. 다음 옵션 중 하나를 선택합니다.
 - ro - Lambda 함수에 시스템 디바이스에 대한 읽기 전용 액세스 권한이 있습니다.
 - rw - Lambda 함수에 시스템 디바이스에 대한 읽기-쓰기 액세스 권한이 있습니다.

기본값은 ro입니다.

- addGroupOwner - (선택 사항) Lambda 함수 구성 요소를 실행하는 시스템 그룹을 시스템 디바이스의 소유자로 추가할지 여부입니다. 기본값은 false입니다.

lambda-function-component.json에는 다음 예제와 유사한 문서가 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"
  }
},
"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 대한 태그(tags)를 추가합니다. 자세한 내용은 [AWS IoT Greengrass Version 2 리소스에 태그 지정](#) 단원을 참조하십시오.

2단계: Lambda 함수 구성 요소 생성

1. 다음 명령을 실행하여 `lambda-function-component.json`에서 Lambda 함수 구성 요소를 생성합니다.

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

요청에 성공하는 경우 응답은 다음 예제와 비슷합니다.

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

출력에서 `arn`을 복사하여 다음 단계에서 구성 요소의 상태를 확인합니다.

2. 구성 요소를 생성할 때 구성 요소의 상태는 `REQUESTED`입니다. 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. `arn`을 이전 단계의 ARN으로 바꿉니다.

```
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 DEPLOYABLE임을 나타냅니다.

```
{  
  "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 DEPLOYABLE이면 Lambda 함수를 코어 디바이스에 배포할 수 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

AWS IoT Device SDK 를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core

코어 디바이스에서 실행되는 구성 요소는 AWS IoT Greengrass 코어 프로세스 간 통신(IPC) 라이브러리를 사용하여 AWS IoT Greengrass nucleus 및 기타 Greengrass 구성 요소와 통신 AWS IoT Device SDK 할 수 있습니다. IPC를 사용하는 사용자 지정 구성 요소를 개발하고 실행하려면 AWS IoT Device SDK 를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 연결하고 IPC 작업을 수행해야 합니다.

IPC 인터페이스는 두 가지 유형의 작업을 지원합니다.

- 요청/응답

구성 요소가 IPC 서비스에 요청을 보내고 요청 결과가 포함된 응답을 수신합니다.

- Subscription

구성 요소가 IPC 서비스에 구독 요청을 보내고 응답으로 이벤트 메시지 스트림을 예상합니다. 구성 요소에서 이벤트 메시지, 오류 및 스트림 종료를 처리하는 구독 핸들러를 제공합니다. 이는 각 IPC 작업에 대한 올바른 응답 및 이벤트 유형이 있는 핸들러 인터페이스가 AWS IoT Device SDK 포함되어 있습니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

주제

- [IPC 클라이언트 버전](#)
- [프로세스 간 통신을 위해 지원되는 SDK](#)
- [AWS IoT Greengrass 코어 IPC 서비스에 연결](#)
- [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#)
- [IPC 이벤트 스트림 구독](#)
- [IPC 모범 사례](#)
- [로컬 메시지 게시/구독](#)
- [AWS IoT Core MQTT 메시지 게시/구독](#)
- [구성 요소 수명 주기와 연동](#)
- [구성 요소 구성과 연동](#)
- [보안 암호 값 검색](#)

- [로컬 새도와 연동](#)
- [로컬 배포 및 구성 요소 관리](#)
- [클라이언트 디바이스 인증 및 권한 부여](#)

IPC 클라이언트 버전

Java 및 Python SDKs의 이후 버전에서는 IPC 클라이언트 V2라는 향상된 버전의 IPC 클라이언트를 AWS IoT Greengrass 제공합니다. IPC 클라이언트 V2의 특징은 다음과 같습니다.

- IPC 작업을 사용하기 위해 작성해야 하는 코드의 양이 감소하고 IPC 클라이언트 V1에서 발생할 수 있는 일반적인 오류를 방지하는 데 도움이 됩니다.
- 구독 핸들러 콜백을 별도의 스레드로 호출하므로 이제 구독 핸들러 콜백에서 추가 IPC 함수 호출이 포함된 차단 코드를 실행할 수 있습니다. IPC 클라이언트 V1에서는 동일한 스레드를 사용하여 IPC 서버와 통신하고 구독 핸들러 콜백을 직접 호출합니다.
- Lambda 표현식(Java) 또는 함수(Python)를 사용하여 구독 작업을 직접적으로 호출할 수 있습니다. IPC 클라이언트 V1에서는 구독 핸들러 클래스를 정의해야 합니다.
- 각 IPC 작업의 동기 및 비동기 버전을 제공합니다. IPC 클라이언트 V1에서는 각 작업의 비동기 버전만 제공합니다.

이러한 개선 사항을 활용하려면 IPC 클라이언트 V2를 사용하는 것이 좋습니다. 그러나 이 설명서와 일부 온라인 콘텐츠의 많은 예제에서는 IPC 클라이언트 V1을 사용하는 방법만 보여줍니다. 다음 예제와 자습서를 사용하여 IPC 클라이언트 V2를 사용하는 샘플 구성 요소를 확인할 수 있습니다.

- [PublishToTopic 예제](#)
- [SubscribeToTopic 예제](#)
- [자습서: 구성 요소 업데이트가 지연되는 Greengrass 구성 요소 개발](#)
- [자습서: MQTT를 통해 로컬 IoT 디바이스와 상호 작용](#)

현재 C++ v2 AWS IoT Device SDK 용은 IPC 클라이언트 V1만 지원합니다.

프로세스 간 통신을 위해 지원되는 SDK

AWS IoT Greengrass 코어 IPC 라이브러리는 다음 AWS IoT Device SDK 버전에 포함되어 있습니다.

SDK	최소 버전	사용법
AWS IoT Device SDK Java v2용	v1.6.0	Java v2 AWS IoT Device SDK 용 사용 (IPC 클라이언트 V2) 섹션을 참조하세요
AWS IoT Device SDK Python v2용	v1.9.0	Python v2(IPC 클라 이언트 V2) AWS IoT Device SDK 에 사용 섹션을 참조하세요
AWS IoT Device SDK C++ v2용	v1.17.0	C++ v2 AWS IoT Device SDK 에 사용 섹션을 참조하세요
AWS IoT Device SDK JavaScript v2용	v1.12.0	JavaScript v2 AWS IoT Device SDK 용 사용(IPC 클라이언트 V1) 섹션을 참조하세 요

AWS IoT Greengrass 코어 IPC 서비스에 연결

사용자 지정 구성 요소에서 프로세스 간 통신을 사용하려면 AWS IoT Greengrass 코어 소프트웨어가 실행하는 IPC 서버 소켓에 대한 연결을 생성해야 합니다. 다음 작업을 완료하여 원하는 언어로 AWS IoT Device SDK 를 다운로드하고 사용합니다.

Java v2 AWS IoT Device SDK 용 사용(IPC 클라이언트 V2)

AWS IoT Device SDK for Java v2(IPC 클라이언트 V2)를 사용하려면

1. [AWS IoT Device SDK for Java v2](#)(v1.6.0 이상)를 다운로드합니다.
2. 다음 중 하나를 수행하여 구성 요소에서 사용자 지정 코드를 실행합니다.
 - 구성 요소가 포함된 JAR 파일로 빌드 AWS IoT Device SDK하고 구성 요소 레시피에서이 JAR 파일을 실행합니다.

- AWS IoT Device SDK JAR을 구성 요소 아티팩트로 정의하고 구성 요소 레시피에서 애플리케이션을 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 AWS IoT Device SDK 용을 사용하려면(IPC 클라이언트 V2)

1. [AWS IoT Device SDK for Python](#)(v1.9.0 이상)을 다운로드합니다.
2. 구성 요소 레시피의 설치 수명 주기에 SDK의 [설치 단계](#)를 추가합니다.
3. AWS IoT Greengrass 코어 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++ v2 AWS IoT Device SDK 에 사용

C++용 AWS IoT Device SDK v2를 빌드하려면 디바이스에 다음 도구가 있어야 합니다.

- C++ 11 이상
- CMake 3.1 이상

- 다음 컴파일러 중 하나입니다.
 - GCC 4.8 이상
 - Clang 3.9 이상
 - MSVC 2015 이상

C++ v2 AWS IoT Device SDK 옹를 사용하려면

1. [AWS IoT Device SDK for C++ v2\(v1.17.0 이상\)](#)를 다운로드합니다.
2. [README의 설치 지침에 따라 소스에서 C++ v2옹](#) 를 빌드합니다. AWS IoT Device SDK
3. C++ 빌드 도구에서 이전 단계에서 빌드한 Greengrass IPC 라이브러리 `AWS::GreengrassIpc-cpp`를 연결합니다. 다음 `CMakeLists.txt` 예제에서는 Greengrass IPC 라이브러리를 CMake로 빌드하는 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 코어 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 예제에서는 IPC 클라이언트와 IPC 클라이언트가 연결 및 연결 해제하고 오류가 발생할 때 출력하는 IPC 연결 수명 주기 핸들러를 생성합니다.

```
#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 실행합니다. AWS IoT Greengrass 코어 소프트웨어가 이진 아티팩트를 실행할 수 OWNER 있도록 아티팩트의 Execute 권한을 로 설정합니다.

구성 요소 레시피의 Manifests 섹션은 다음 예제와 같을 수 있습니다.

JSON

```

{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
      }
    }
  ]
}

```

```

    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
    Permission:
      Execute: OWNER

```

JavaScript v2 AWS IoT Device SDK 용 사용(IPC 클라이언트 V1)

NodeJS에서 사용할 AWS IoT Device SDK for JavaScript v2를 빌드하려면 디바이스에 다음 도구가 있어야 합니다.

- NodeJS 10.0 이상
 - `node -v`를 실행하여 노드 버전을 확인합니다.
- CMake 3.1 이상

AWS IoT Device SDK for JavaScript v2(IPC 클라이언트 V1)를 사용하려면

1. [AWS IoT Device SDK for JavaScript v2](#)(v1.12.10 이상)를 다운로드합니다.
2. [README의 설치 지침에](#) 따라 소스에서 for JavaScript v2 AWS IoT Device SDK 를 빌드합니다.

3. AWS IoT Greengrass 코어 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 구성 요소와 Greengrass nucleus 사이의 연결을 설정합니다.

```
await client.connect();
```

구성 요소에 IPC 작업을 수행할 수 있는 권한 부여

사용자 지정 구성 요소가 일부 IPC 작업을 사용하도록 허용하려면 구성 요소가 특정 리소스에서 작업을 수행하도록 허용하는 권한 부여 정책을 정의해야 합니다. 각 권한 부여 정책에서는 정책에서 허용하는 작업 목록과 리소스 목록을 정의합니다. 예를 들어 메시징 IPC 서비스 게시/구독에서는 주제 리소스에 대한 게시 및 구독 작업을 정의합니다. * 와일드카드를 사용하여 모든 작업 또는 모든 리소스에 대한 액세스를 허용할 수 있습니다.

구성 요소 레시피에서나 구성 요소를 배포할 때 설정할 수 있는 `accessControl` 구성 파라미터로 권한 부여 정책을 정의합니다. `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 정책 ID가 있으며, 이는 모든 구성 요소에서 고유해야 합니다.

Tip

고유한 정책 ID를 생성하려면 구성 요소 이름, IPC 서비스 이름 및 카운터를 결합할 수 있습니다. 예를 들어 `com.example.HelloWorld`라는 구성 요소에서는 다음 ID를 사용하여 두 개의 게시/구독 권한 부여 정책을 정의할 수 있습니다.

- `com.example.HelloWorld:pubsub:1`
- `com.example.HelloWorld:pubsub:2`

권한 부여 정책에서는 다음 형식을 사용합니다. 이 객체는 `accessControl` 구성 파라미터입니다.

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 정책의 `resources` 요소에 * 와일드카드를 사용하여 단일 권한 부여 정책에서 여러 리소스에 대한 액세스를 허용할 수 있습니다.

- 모든 버전의 [Greengrass nucleus](#)에서 단일 * 문자를 리소스로 지정하여 모든 리소스에 대한 액세스를 허용할 수 있습니다.
- [Greengrass nucleus](#) v2.6.0 이상에서는 임의의 문자 조합과 일치하도록 * 문자를 리소스에 지정할 수 있습니다. 예를 들어 `factory/1/devices/Thermostat*/status`를 지정하여 각 디바이스의

이름이 Thermostat로 시작되는 공장의 모든 온도조절 디바이스의 상태 주제에 대한 액세스를 허용할 수 있습니다.

AWS IoT Core MQTT IPC 서비스에 대한 권한 부여 정책을 정의할 때 MQTT 와일드카드(+ 및 #)를 사용하여 여러 리소스와 일치시킬 수도 있습니다. 자세한 내용은 [MQTT IPC 권한 부여 정책의 AWS IoT Core MQTT 와일드카드를 참조하세요](#).

권한 부여 정책의 레시피 변수

[Greengrass nucleus](#) v2.6.0 이상을 사용하고 Greengrass nucleus의 [interpolateComponentConfiguration](#) 구성 옵션을 true로 설정한 경우 권한 부여 정책에서 {iot:thingName} [레시피 변수](#)를 사용할 수 있습니다. MQTT 주제 또는 디바이스 새도 등에 대한 코어 디바이스의 이름이 포함된 권한 부여 정책이 필요한 경우 이 레시피 변수를 사용하여 코어 디바이스 그룹에 대한 단일 권한 부여 정책을 구성할 수 있습니다. 예를 들어 새도 IPC 작업을 위해 다음 리소스에 대한 구성 요소 액세스를 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 코어 소프트웨어에 캐릭터의 특별한 의미 대신 리터럴 값을 사용하도록 지시합니다. 예를 들어 * 문자는 임의의 문자 조합과 일치하는 [와일드카드](#)입니다.

리터럴 문자	이스케이프 시퀀스	참고
*	<code>\${*}</code>	
?	<code>\${?}</code>	AWS IoT Greengrass 는 현재 단일 문자와 일치하는 ? 와일드카드를 지원하지 않습니다.
\$	<code>\${\$}</code>	<code>\${</code> 이 포함된 리소스와 일치시키려면 이스케이프 시퀀스를 사용합니다. 예를 들어 <code>\${resourceName}</code> 이라는 리소스와 일치시키려면 <code>\${\$}</code>

리터럴 문자	이스케이프 시퀀스	참고
		{resourceName} 을 지정해야 합니다. 그렇지 않으면 \$가 포함된 리소스와 일치시키려면 리터럴 \$를 사용할 수 있습니다. 예를 들어 \$aws로 시작하는 주제에 대한 액세스를 허용할 수 있습니다.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example 권한 부여 정책이 있는 구성 요소 레시피 예제

다음 예제 구성 요소 레시피에는 권한 부여 정책을 정의하는 `accessControl` 객체가 포함되어 있습니다. 이 정책은 `com.example>HelloWorld` 구성 요소에 `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 정책이 있는 구성 요소 구성 업데이트 예제

배포의 다음 예제 구성 업데이트에서는 권한 부여 정책을 정의하는 `accessControl` 객체로 구성 요소를 구성하도록 지정합니다. 이 정책은 `com.example.HelloWorld` 구성 요소에 `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 명령은 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-  
deployment.json
```

hello-world-deployment.json 파일에는 다음 JSON 문서가 포함되어 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
```

hello-world-configuration.json 파일에는 다음 JSON 문서가 포함되어 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 스트림 구독

IPC 작업을 사용하여 Greengrass 코어 디바이스에서 이벤트 스트림을 구독할 수 있습니다. 구독 작업을 사용하려면 구독 핸들러를 정의하고 IPC 서비스에 대한 요청을 생성합니다. 그런 다음 IPC 클라이언트는 코어 디바이스가 구성 요소에 이벤트 메시지를 스트리밍할 때마다 구독 핸들러의 함수를 실행합니다.

구독을 닫아 이벤트 메시지 처리를 중지할 수 있습니다. 이렇게 하려면 구독을 여는 데 사용한 구독 작업 객체에 대해 `closeStream()`(Java), `close()`(Python) 또는 `Close()`(C++)를 직접 호출합니다.

AWS IoT Greengrass 코어 IPC 서비스는 다음 구독 작업을 지원합니다.

- [SubscribeToTopic](#)
- [SubscribeToIoTCore](#)
- [SubscribeToComponentUpdates](#)
- [SubscribeToConfigurationUpdate](#)
- [SubscribeToValidateConfigurationUpdates](#)

주제

- [구독 핸들러 정의](#)
- [구독 핸들러 예제](#)

구독 핸들러 정의

구독 핸들러를 정의하려면 이벤트 메시지, 오류 및 스트림 종료를 처리하는 콜백 함수를 정의합니다. IPC 클라이언트 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 생성하지 않고도 이러한 함수를 정의할 수 있습니다.

Java

IPC 클라이언트 V1을 사용하는 경우 일반

`software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<StreamEventType>` 인터페이스를 구현해야 합니다. *StreamEventType*은 구독 작업에 대한 이벤트 메시지의 유형입니다. 이벤트 메시지, 오류 및 스트림 종료를 처리하는 다음 함수를 정의합니다.

IPC 클라이언트 V2를 사용하는 경우 구독 핸들러 클래스 외부에서 이러한 함수를 정의하거나 [lambda 표현식](#)을 사용할 수 있습니다.

```
void onStreamEvent(StreamEventType event)
```

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 직접 호출하는 콜백입니다.

```
boolean onStreamError(Throwable error)
```

스트림 오류가 발생할 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

오류로 인해 구독 스트림을 닫으려면 true를 반환하고 스트림을 열린 상태로 유지하려면 false를 반환합니다.

```
void onStreamClosed()
```

스트림이 닫힐 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

Python

IPC 클라이언트 V1을 사용하는 경우 구독 작업에 해당하는 스트림 응답 핸들러 클래스를 확장해야 합니다. 예는 각 구독 작업에 대한 구독 핸들러 클래스가 AWS IoT Device SDK 포함되어 있습니다. *StreamEventType*은 구독 작업에 대한 이벤트 메시지의 유형입니다. 이벤트 메시지, 오류 및 스트림 종료를 처리하는 다음 함수를 정의합니다.

IPC 클라이언트 V2를 사용하는 경우 구독 핸들러 클래스 외부에서 이러한 함수를 정의하거나 [lambda 표현식](#)을 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 직접 호출하는 콜백입니다.

```
def on_stream_error(self, error: Exception) -> bool
```

스트림 오류가 발생할 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

오류로 인해 구독 스트림을 닫으려면 true를 반환하고 스트림을 열린 상태로 유지하려면 false를 반환합니다.

```
def on_stream_closed(self) -> None
```

스트림이 닫힐 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

C++ (IPC client V1)

구독 작업에 해당하는 스트림 응답 핸들러 클래스에서 파생되는 클래스를 구현합니다. 예는 각 구독 작업에 대한 구독 핸들러 기본 클래스가 AWS IoT Device SDK 포함되어 있습니다.

*StreamEventType*은 구독 작업에 대한 이벤트 메시지의 유형입니다. 이벤트 메시지, 오류 및 스트림 종료를 처리하는 다음 함수를 정의합니다.

```
void OnStreamEvent(StreamEventType *event)
```

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 직접 호출하는 콜백입니다.

```
bool OnStreamError(OnError *error)
```

스트림 오류가 발생할 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

오류로 인해 구독 스트림을 닫으려면 true를 반환하고 스트림을 열린 상태로 유지하려면 false를 반환합니다.

```
void OnStreamClosed()
```

스트림이 닫힐 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

JavaScript

구독 작업에 해당하는 스트림 응답 핸들러 클래스에서 파생되는 클래스를 구현합니다. 예는 각 구독 작업에 대한 구독 핸들러 기본 클래스가 AWS IoT Device SDK 포함되어 있습니다.

*StreamEventType*은 구독 작업에 대한 이벤트 메시지의 유형입니다. 이벤트 메시지, 오류 및 스트림 종료를 처리하는 다음 함수를 정의합니다.

```
on(event: 'ended', listener: StreamingOperationEndedListener)
```

스트림이 닫힐 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

```
on(event: 'streamError', listener: StreamingRpcErrorListener)
```

스트림 오류가 발생할 때 IPC 클라이언트가 직접 호출하는 콜백입니다.

오류로 인해 구독 스트림을 닫으려면 true를 반환하고 스트림을 열린 상태로 유지하려면 false를 반환합니다.

```
on(event: 'message', listener: (message: InboundMessageType) => void)
```

IPC 클라이언트가 MQTT 메시지 또는 구성 요소 업데이트 알림과 같은 이벤트 메시지를 수신할 때 직접 호출하는 콜백입니다.

구독 핸들러 예제

다음 예제에서는 [SubscribeToTopic](#) 작업 및 구독 핸들러를 사용하여 로컬 게시/구독 메시지를 구독하는 방법을 보여줍니다.

Java (IPC client V2)

Example예: 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)

Example예: 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++ (IPC client V1)

Example예: 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

Example예: 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();

```

Rust

Example예: 로컬 게시/구독 메시지 구독

```
use gg_sdk::{Sdk, SubscribeToTopicPayload};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";

    let callback = |topic: &str, payload: SubscribeToTopicPayload| match payload
    {
        SubscribeToTopicPayload::Binary(message) => {
            let message = String::from_utf8_lossy(message);
            println!("Received new message on topic {topic}: {message}");
        }
        SubscribeToTopicPayload::Json(_) => {
            println!("Received new message on topic {topic}: (JSON message)");
        }
    };

    let _sub = sdk
        .subscribe_to_topic(topic, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

C

Example예: 로컬 게시/구독 메시지 구독

```
#include <assert.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
```

```
#include <gg/sdk.h>
#include <gg/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    if (gg_obj_type(payload) == GG_TYPE_BUF) {
        GgBuffer message = gg_obj_into_buf(payload);
        printf(
            "Received new message on topic %.*s: %.*s\n",
            (int) topic.len,
            topic.data,
            (int) message.len,
            message.data
        );
    } else {
        assert(gg_obj_type(payload) == GG_TYPE_MAP);
        printf(
            "Received new message on topic %.*s: (JSON message)\n",
            (int) topic.len,
            topic.data
        );
    }
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_topic(
```

```

        topic, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the stream.
    ggipc_close_subscription(handle);
}

```

C++ (Component SDK)

Example예: 로컬 게시/구독 메시지 구독

```

#include <gg/ipc/client.hpp>
#include <gg/object.hpp>
#include <unistd.h>
#include <cassert>
#include <iostream>

class ResponseHandler : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        if (payload.index() == GG_TYPE_BUF) {

```

```

        std::cout << "Received new message on topic " << topic << ": "
                  << get<gg::Buffer>(payload) << "\n";
    } else {
        assert(payload.index() == GG_TYPE_MAP);
        std::cout << "Received new message on topic " << topic
                  << ": (JSON message)\n";
    }
}
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";

    static ResponseHandler handler;
    error = client.subscribe_to_topic(topic, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}

```

IPC 모범 사례

사용자 지정 구성 요소에서 IPC를 사용하는 모범 사례는 IPC 클라이언트 V1과 IPC 클라이언트 V2 간에 다릅니다. 사용하는 IPC 클라이언트 버전에 대한 모범 사례를 따르세요.

IPC client V2

IPC 클라이언트 V2는 별도의 스레드에서 콜백 함수를 실행하므로 IPC 클라이언트 V1에 비해 IPC 를 사용하고 구독 핸들러 함수 작성할 때 따라야 할 지침이 더 적습니다.

- 하나의 IPC 클라이언트 재사용

IPC 클라이언트를 생성한 후 계속 열어 두고 모든 IPC 작업에 재사용합니다. 여러 클라이언트를 생성하면 추가 리소스가 사용되어 리소스가 누출될 수 있습니다.

- 예외 처리

IPC 클라이언트 V2는 구독 핸들러 함수의 catch되지 않은 예외를 기록합니다. 코드에서 발생하는 오류를 처리하려면 핸들러 함수의 예외를 catch해야 합니다.

IPC client V1

IPC 클라이언트 V1은 IPC 서버와 통신하고 구독 핸들러를 직접 호출하는 단일 스레드를 사용합니다. 구독 핸들러 함수를 작성할 때는 이 동기 동작을 고려해야 합니다.

- 하나의 IPC 클라이언트 재사용

IPC 클라이언트를 생성한 후 계속 열어 두고 모든 IPC 작업에 재사용합니다. 여러 클라이언트를 생성하면 추가 리소스가 사용되어 리소스가 누출될 수 있습니다.

- 비동기적으로 차단 코드 실행

IPC 클라이언트 V1은 스레드가 차단된 동안 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 및 완료하는 데 시간이 걸리는 동기 I/O 요청이 포함됩니다.

- 비동기적으로 새 IPC 요청 전송

IPC 클라이언트 V1은 구독 핸들러 함수 내에서 새 요청을 보낼 수 없습니다. 응답을 기다리는 경우 요청에서 핸들러 함수를 차단하기 때문입니다. 핸들러 함수에서 실행하는 별도의 스레드에서 IPC 요청을 보내야 합니다.

- 예외 처리

IPC 클라이언트 V1은 구독 핸들러 함수의 catch되지 않은 예외를 기록합니다. 핸들러 함수에서 예외를 throw하면 구독이 종료되고 구성 요소 로그에 예외가 표시되지 않습니다. 구독을 열린 상태로 유지하고 코드에서 발생하는 오류를 기록하려면 핸들러 함수의 예외를 catch해야 합니다.

로컬 메시지 게시/구독

게시/구독(pubsub) 메시징을 사용하면 주제에 메시지를 보내고 주제에서 메시지를 받을 수 있습니다. 구성 요소가 주제에 메시지를 게시하여 다른 구성 요소에 메시지를 보낼 수 있습니다. 그러면 해당 주제를 구독하는 구성 요소가 수신하는 메시지에 대한 작업을 수행할 수 있습니다.

Note

이 게시/구독 IPC 서비스를 사용하여 AWS IoT Core MQTT를 게시하거나 구독할 수 없습니다. AWS IoT Core MQTT와 메시지를 교환하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Core MQTT 메시지 게시/구독](#).

주제

- [최소 SDK 버전](#)
- [권한 부여](#)
- [PublishToTopic](#)
- [SubscribeToTopic](#)
- [예제](#)

최소 SDK 버전

다음 표에는 로컬 주제에 메시지를 게시하고 구독하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.2.10	
AWS IoT Device SDK Python v2용	v1.5.3	
AWS IoT Device SDK C++ v2용	v1.17.0	

SDK	최소 버전	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 로컬 게시/구독 메시징을 사용하려면 구성 요소가 주제에 대한 메시지를 보내고 받을 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

게시/구독 메시징에 대한 권한 부여 정책에는 다음 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ipc.pubsub`

연산	설명	리소스
<code>aws.greengrass#PublishToTopic</code>	구성 요소가 지정한 주제에 메시지를 게시할 수 있도록 허용합니다.	<code>test/topic</code> 과 같은 주제 문자열입니다. 주제의 임의의 문자 조합과 일치시키려면 <code>*</code>를 사용합니다. 이 주제 문자열에서는 MQTT 주제 와일드카드(<code>#</code> 및 <code>+</code>)를 지원하지 않습니다.
<code>aws.greengrass#SubscribeToTopic</code>	구성 요소가 지정하는 주제에 대한 메시지를 구독할 수 있도록 허용합니다.	<code>test/topic</code> 과 같은 주제 문자열입니다. 주제의 임의의 문자 조합과 일치시키려면 <code>*</code>를 사용합니다. Greengrass nucleus v2.6.0 이상에서는 MQTT 주제 와일드카드(<code>#</code> 및 <code>+</code>)가 포함된 주제를 구독할 수 있습니다. 이 주제 문자열은 MQTT 주제 와일드카드를 리터럴 문자로 지원합니다. 예를 들어 구성 요소의 권한 부

연산	설명	리소스
		여 정책에서 test/topic/#에 대한 액세스 권한을 부여하는 경우 구성 요소는 test/topic/# 는 구독할 수 있지만 test/topic/filter 는 구독할 수 없습니다.
*	구성 요소가 지정하는 주제에 대한 메시지를 게시하고 구독할 수 있도록 허용합니다.	test/topic 과 같은 주제 문자열입니다. 주제의 임의의 문자 조합과 일치시키려면 *를 사용합니다. 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 정책에서 test/topic/#에 대한 액세스 권한을 부여하는 경우 구성 요소는 test/topic/# 는 구독할 수 있지만 test/topic/filter 는 구독할 수 없습니다.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example 권한 부여 정책 예제

다음 권한 부여 정책 예제에서는 구성 요소가 모든 주제를 게시하고 구독할 수 있도록 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 요청에서는 다음 파라미터를 사용합니다.

topic

메시지를 게시할 주제입니다.

publishMessage(Python: publish_message)

게시할 메시지입니다. 이 객체 PublishMessage에는 다음 정보가 포함됩니다. jsonMessage 및 binaryMessage 중 하나를 지정해야 합니다.

jsonMessage(Python: json_message)

(선택 사항) JSON 메시지입니다. 이 객체 JsonMessage에는 다음 정보가 포함됩니다.

message

객체인 JSON 메시지입니다.

context

메시지가 게시된 주제와 같은 메시지의 컨텍스트입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하기 위해 사용해야 하는 AWS IoT Device SDK의 최소 버전이 나열되어 있습니다.

SDK	최소 버전
AWS IoT Device SDK Java v2용	v1.9.3
AWS IoT Device SDK Python v2용	v1.11.3
AWS IoT Device SDK C++ v2용	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

 Note

AWS IoT Greengrass 코어 소프트웨어는 PublishToTopic 및 SubscribeToTopic 작업에서 동일한 메시지 객체를 사용합니다. AWS IoT Greengrass 코어 소프트웨어는 구독 시 메시지에서이 컨텍스트 객체를 설정하고 게시하는 메시지에서이 컨텍스트 객체를 무시합니다.

이 객체 MessageContext에는 다음 정보가 포함됩니다.

topic

메시지가 게시된 주제입니다.

binaryMessage(Python: binary_message)

(선택 사항) 이진 메시지입니다. 이 객체 BinaryMessage에는 다음 정보가 포함됩니다.

message

Blob인 이진 메시지입니다.

context

메시지가 게시된 주제와 같은 메시지의 컨텍스트입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하기 위해 사용해야 하는 AWS IoT Device SDK의 최소 버전이 나열되어 있습니다.

SDK	최소 버전
AWS IoT Device SDK Java v2용	v1.9.3
AWS IoT Device SDK Python v2용	v1.11.3
AWS IoT Device SDK C++ v2용	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

Note

AWS IoT Greengrass 코어 소프트웨어는 PublishToTopic 및 SubscribeToTopic 작업에서 동일한 메시지 객체를 사용합니다. AWS IoT Greengrass 코어 소프트웨어는 구독 시 메시지에서이 컨텍스트 객체를 설정하고 게시하는 메시지에서이 컨텍스트 객체를 무시합니다.

이 객체 MessageContext에는 다음 정보가 포함됩니다.

topic

메시지가 게시된 주제입니다.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example예: 이진 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)

Example예: 이진 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++ (IPC client V1)

Example예: 이진 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

Example예: 이진 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();

```

Rust

Example예: 이진 메시지 게시

```

use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";

    sdk.publish_to_topic_binary(topic, message)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}

```

C

Example예: 이진 메시지 게시

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");

    err = ggipc_publish_to_topic_binary(topic, message);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}
```

C++ (Component SDK)

Example예: 이진 메시지 게시

```
#include <gg/ipc/client.hpp>
```

```
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";

    error = client.publish_to_topic(topic, message);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}
```

SubscribeToTopic

주제에 대한 메시지를 구독합니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 종료를 처리하는 함수를 사용하여 스트림 응답 핸들러를 정의합니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

이벤트 메시지 유형: SubscriptionResponseMessage

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

topic

구독할 주제입니다.

Note

[Greengrass nucleus](#) v2.6.0 이상에서 이 주제는 MQTT 주제 와일드카드(# 및 +)를 지원합니다.

`receiveMode(Python: receive_mode)`

(선택 사항) 구성 요소가 자체에서 메시지를 수신하는지 여부를 지정하는 동작입니다. 이 동작을 변경하여 구성 요소가 자체 메시지에 대해 조치하도록 허용할 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 때 기본 옵션입니다.

- RECEIVE_MESSAGES_FROM_OTHERS - 구독하는 구성 요소의 메시지를 제외하고 주제와 일치하는 모든 메시지를 수신합니다.

이 모드는 MQTT 와일드카드가 포함된 주제를 구독할 때 기본 옵션입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다. 다음 표에는 수신 모드를 설정하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK for Java v2	v1.9.3	
AWS IoT Device SDK Python v2용	v1.11.3	
AWS IoT Device SDK for C++ v2	v1.18.4	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

messages

메시지 스트림입니다. 이 객체 SubscriptionResponseMessage에는 다음 정보가 포함됩니다. 각 메시지에는 jsonMessage 또는 binaryMessage가 포함됩니다.

jsonMessage(Python: json_message)

(선택 사항) JSON 메시지입니다. 이 객체 JsonMessage에는 다음 정보가 포함됩니다.

message

객체인 JSON 메시지입니다.

context

메시지가 게시된 주제와 같은 메시지의 컨텍스트입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하기 위해 사용해야 하는 AWS IoT Device SDK 의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.9.3	
AWS IoT Device SDK Python v2용	v1.11.3	
AWS IoT Device SDK C++ v2용	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Note

AWS IoT Greengrass 코어 소프트웨어는 PublishToTopic 및 SubscribeToTopic 작업에서 동일한 메시지 객체를 사용합니다. AWS IoT Greengrass 코어 소프트웨어는 구독 시 메시지에서이 컨텍스트 객체를 설정하고 게시하는 메시지에서이 컨텍스트 객체를 무시합니다.

이 객체 MessageContext에는 다음 정보가 포함됩니다.

`topic`

메시지가 게시된 주제입니다.

`binaryMessage(Python: binary_message)`

(선택 사항) 이진 메시지입니다. 이 객체 BinaryMessage에는 다음 정보가 포함됩니다.

`message`

Blob인 이진 메시지입니다.

`context`

메시지가 게시된 주제와 같은 메시지의 컨텍스트입니다.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다. 다음 표에는 메시지 컨텍스트에 액세스하기 위해 사용해야 하는 AWS IoT Device SDK의 최소 버전이 나열되어 있습니다.

SDK	최소 버전
AWS IoT Device SDK Java v2용	v1.9.3
AWS IoT Device SDK Python v2용	v1.11.3
AWS IoT Device SDK C++ v2용	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

Note

AWS IoT Greengrass 코어 소프트웨어는 PublishToTopic 및 SubscribeToTopic 작업에서 동일한 메시지 객체를 사용합니다. AWS IoT Greengrass 코어 소프트웨어는 구독 시 메시지에서이 컨텍스트 객체를 설정하고 게시하는 메시지에서이 컨텍스트 객체를 무시합니다.

이 객체 MessageContext에는 다음 정보가 포함됩니다.

topic

메시지가 게시된 주제입니다.

topicName(Python: topic_name)

메시지가 게시된 주제입니다.

Note

이 속성은 현재 사용되지 않습니다. [Greengrass nucleus](#) v2.6.0 이상에서는 SubscriptionResponseMessage에서 (jsonMessage | binaryMessage).context.topic 값을 가져와 메시지가 게시된 주제를 가져올 수 있습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example예: 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)

Example예: 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++ (IPC client V1)

Example예: 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

Example예: 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();

```

Rust

Example예: 로컬 게시/구독 메시지 구독

```

use gg_sdk::{Sdk, SubscribeToTopicPayload};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");
}

```

```

let topic = "my/topic";

let callback = |topic: &str, payload: SubscribeToTopicPayload| match payload
{
    SubscribeToTopicPayload::Binary(message) => {
        let message = String::from_utf8_lossy(message);
        println!("Received new message on topic {topic}: {message}");
    }
    SubscribeToTopicPayload::Json(_) => {
        println!("Received new message on topic {topic}: (JSON message)");
    }
};

let _sub = sdk
    .subscribe_to_topic(topic, &callback)
    .expect("Failed to subscribe to topic");

println!("Successfully subscribed to topic: {topic}");

// Keep the main thread alive, or the process will exit.
loop {
    thread::sleep(Duration::from_secs(10));
}
}

```

C

Example예: 로컬 게시/구독 메시지 구독

```

#include <assert.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <gg/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;

```

```
(void) handle;

if (gg_obj_type(payload) == GG_TYPE_BUF) {
    GgBuffer message = gg_obj_into_buf(payload);
    printf(
        "Received new message on topic %.*s: %.*s\n",
        (int) topic.len,
        topic.data,
        (int) message.len,
        message.data
    );
} else {
    assert(gg_obj_type(payload) == GG_TYPE_MAP);
    printf(
        "Received new message on topic %.*s: (JSON message)\n",
        (int) topic.len,
        topic.data
    );
}
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_topic(
        topic, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }
}
```

```

}

printf(
    "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
);

// Keep the main thread alive, or the process will exit.
while (1) {
    sleep(10);
}

// To stop subscribing, close the stream.
ggipc_close_subscription(handle);
}

```

C++ (Component SDK)

Example예: 로컬 게시/구독 메시지 구독

```

#include <gg/ipc/client.hpp>
#include <gg/object.hpp>
#include <unistd.h>
#include <cassert>
#include <iostream>

class ResponseHandler : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        if (payload.index() == GG_TYPE_BUF) {
            std::cout << "Received new message on topic " << topic << ": "
                << get<gg::Buffer>(payload) << "\n";
        } else {
            assert(payload.index() == GG_TYPE_MAP);
            std::cout << "Received new message on topic " << topic
                << ": (JSON message)\n";
        }
    }
};

```

```

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";

    static ResponseHandler handler;
    error = client.subscribe_to_topic(topic, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}

```

예제

다음 예제를 사용하여 구성 요소에서 게시/구독 IPC 서비스를 사용하는 방법을 알아봅니다.

예제 게시/구독 게시자(Java, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

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 애플리케이션에서는 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);
}
}
}

```

예제 게시/구독 구독자(Java, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

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 애플리케이션에서는 게시/구독 IPC 서비스를 사용하여 다른 구성 요소의 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.");
    }
}
}
}

```

예제 게시/구독 게시자(Python, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

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 애플리케이션에서는 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)

```

예제 게시/구독 구독자(Python, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

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 애플리케이션에서는 게시/구독 IPC 서비스를 사용하여 다른 구성 요소의 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++, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
  com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher
Permission:
Execute: OWNER
```

다음 예제 C++ 애플리케이션에서는 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;
}

```

구독자 게시/구독 예제(C++, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
      Permission:
        Execute: OWNER

```

다음 예제 C++ 애플리케이션에서는 게시/구독 IPC 서비스를 사용하여 다른 구성 요소의 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;
}

```

게시/구독 게시자 예제(Rust)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherRust",

```

```

"ComponentVersion": "1.0.0",
"ComponentDescription": "A component that publishes messages.",
"ComponentPublisher": "Amazon",
"ComponentConfiguration": {
  "DefaultConfiguration": {
    "accessControl": {
      "aws.greengrass.ipc.pubsub": {
        "com.example.PubSubPublisherRust:pubsub:1": {
          "policyDescription": "Allows access to publish to all topics.",
          "operations": ["aws.greengrass#PublishToTopic"],
          "resources": ["*"]
        }
      }
    }
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "{artifacts:path}/publish_to_topic"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherRust/1.0.0/publish_to_topic",
        "Permission": {
          "Execute": "OWNER"
        }
      }
    ]
  }
]
}

```

다음 Rust 애플리케이션 예제에서는 게시/구독 IPC 서비스를 사용하여 메시지를 다른 구성 요소에 게시하는 방법을 보여줍니다.

```
use gg_sdk::Sdk;
```

```
fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";

    sdk.publish_to_topic_binary(topic, message)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}
```

구독자 게시/구독 예제(Rust)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberRust",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberRust:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/subscribe_to_topic"
```

```

    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberRust/1.0.0/subscribe_to_topic",
        "Permission": {
          "Execute": "OWNER"
        }
      }
    ]
  }
}

```

다음 Rust 애플리케이션 예제에서는 게시/구독 IPC 서비스를 사용하여 다른 구성 요소의 메시지를 구독하는 방법을 보여줍니다.

```

use gg_sdk::{Sdk, SubscribeToTopicPayload};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";

    let callback = |topic: &str, payload: SubscribeToTopicPayload| match payload
    {
        SubscribeToTopicPayload::Binary(message) => {
            let message = String::from_utf8_lossy(message);
            println!("Received new message on topic {topic}: {message}");
        }
        SubscribeToTopicPayload::Json(_) => {
            println!("Received new message on topic {topic}: (JSON message)");
        }
    };

    let _sub = sdk
        .subscribe_to_topic(topic, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");
}

```



```
// Keep the main thread alive, or the process will exit.
loop {
    thread::sleep(Duration::from_secs(10));
}
}
```

게시/구독 게시자 예제(C)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherC",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherC:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_publish_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.PubSubPublisherC/1.0.0/sample_publish_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

```

    }
  }
]
}
]
}

```

다음 예제 C 애플리케이션은 게시/구독 IPC 서비스를 사용하여 메시지를 다른 구성 요소에 게시하는 방법을 보여줍니다.

```

#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");

    err = ggipc_publish_to_topic_binary(topic, message);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}

```

구독자 게시/구독 예제(C)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberC",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberC:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_subscribe_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.PubSubSubscriberC/1.0.0/sample_subscribe_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

```
}
```

다음 예제 C 애플리케이션은 게시/구독 IPC 서비스를 사용하여 다른 구성 요소의 메시지를 구독하는 방법을 보여줍니다.

```
#include <assert.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <gg/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    if (gg_obj_type(payload) == GG_TYPE_BUF) {
        GgBuffer message = gg_obj_into_buf(payload);
        printf(
            "Received new message on topic %.*s: %.*s\n",
            (int) topic.len,
            topic.data,
            (int) message.len,
            message.data
        );
    } else {
        assert(gg_obj_type(payload) == GG_TYPE_MAP);
        printf(
            "Received new message on topic %.*s: (JSON message)\n",
            (int) topic.len,
            topic.data
        );
    }
}

int main(void) {
    gg_sdk_init();
```

```

GgError err = ggipc_connect();
if (err != GG_ERR_OK) {
    fprintf(stderr, "Failed to establish IPC connection.\n");
    exit(-1);
}

GgBuffer topic = GG_STR("my/topic");

GgIpcSubscriptionHandle handle;
err = ggipc_subscribe_to_topic(
    topic, on_subscription_response, NULL, &handle
);
if (err != GG_ERR_OK) {
    fprintf(
        stderr,
        "Failed to subscribe to topic: %.*s\n",
        (int) topic.len,
        topic.data
    );
    exit(-1);
}

printf(
    "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
);

// Keep the main thread alive, or the process will exit.
while (1) {
    sleep(10);
}

// To stop subscribing, close the stream.
ggipc_close_subscription(handle);
}

```

게시/구독 게시자 예제(C++, 구성 요소 SDK)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

```

{
    "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": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_cpp_publish_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherCpp/1.0.0/sample_cpp_publish_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}

```

다음 예제 C++ 애플리케이션에서는 게시/구독 IPC 서비스를 사용하여 메시지를 다른 구성 요소에 게시하는 방법을 보여줍니다.

```

#include <gg/ipc/client.hpp>
#include <iostream>

int main() {

```

```

auto &client = gg::ipc::Client::get();

auto error = client.connect();
if (error) {
    std::cerr << "Failed to establish IPC connection.\n";
    exit(-1);
}

std::string_view message = "Hello, World";
std::string_view topic = "my/topic";

error = client.publish_to_topic(topic, message);
if (error) {
    std::cerr << "Failed to publish to topic: " << topic << "\n";
    exit(-1);
}

std::cout << "Successfully published to topic: " << topic << "\n";
}

```

구독자 게시/구독 예제(C++, 구성 요소 SDK)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

```

{
  "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": [

```

```

{
  "Platform": {
    "os": "linux",
    "runtime": "*"
  },
  "Lifecycle": {
    "run": "{artifacts:path}/sample_cpp_subscribe_to_topic"
  },
  "Artifacts": [
    {
      "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/sample_cpp_subscribe_to_topic",
      "Permission": {
        "Execute": "OWNER"
      }
    }
  ]
}

```

다음 예제 C++ 애플리케이션은 게시/구독 IPC 서비스를 사용하여 다른 구성 요소의 메시지를 구독하는 방법을 보여줍니다.

```

#include <gg/ipc/client.hpp>
#include <gg/object.hpp>
#include <unistd.h>
#include <cassert>
#include <iostream>

class ResponseHandler : public gg::ipc::LocalTopicCallback {
public:
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        if (payload.index() == GG_TYPE_BUF) {
            std::cout << "Received new message on topic " << topic << ": "
                      << get<gg::Buffer>(payload) << "\n";
        } else {
            assert(payload.index() == GG_TYPE_MAP);
            std::cout << "Received new message on topic " << topic

```



```

        << " : (JSON message)\n";
    }
}
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";

    static ResponseHandler handler;
    error = client.subscribe_to_topic(topic, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}

```

AWS IoT Core MQTT 메시지 게시/구독

AWS IoT Core MQTT 메시징 IPC 서비스를 사용하면 MQTT 메시지를 주고 받을 수 있습니다 AWS IoT Core. 구성 요소는 메시지를 게시 AWS IoT Core 하고 주제를 구독하여 다른 소스의 MQTT 메시지에 대해 조치를 취할 수 있습니다. MQTT AWS IoT Core 구현에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [MQTT](#)를 참조하세요.

Note

이 MQTT 메시징 IPC 서비스를 사용하면와 메시지를 교환할 수 있습니다 AWS IoT Core. 구성 요소 간에 메시지를 교환하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

주제

- [최소 SDK 버전](#)
- [권한 부여](#)
- [PublishToIoTCore](#)
- [SubscribeToIoTCore](#)
- [예제](#)

최소 SDK 버전

다음 표에는 MQTT 메시지를 게시하고 구독하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전 이 나열되어 있습니다 AWS IoT Core.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.2.10	
AWS IoT Device SDK Python v2용	v1.5.3	
AWS IoT Device SDK C++ v2용	v1.17.0	
AWS IoT Device SDK JavaScript v2용	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 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

AWS IoT Core MQTT 메시징에 대한 권한 부여 정책에는 다음과 같은 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ipc.mqttproxy`

연산	설명	리소스
<code>aws.greengrass#PublishToIoTCore</code>	구성 요소가 지정한 MQTT 주제에 대해 AWS IoT Core 에 메시지를 게시하도록 허용합니다.	모든 주제에 대한 액세스를 허용하는 주제 문자열(<code>test/topic</code> 또는 <code>*</code>)입니다. MQTT 주제 와일드카드(<code>#</code> 및 <code>+</code>)를 사용하여 여러 리소스와 일치시킬 수 있습니다.
<code>aws.greengrass#SubscribeToIoTCore</code>	구성 요소가 지정한 주제에 AWS IoT Core 대해의 메시지를 구독하도록 허용합니다.	모든 주제에 대한 액세스를 허용하는 주제 문자열(<code>test/topic</code> 또는 <code>*</code>)입니다. MQTT 주제 와일드카드(<code>#</code> 및 <code>+</code>)를 사용하여 여러 리소스와 일치시킬 수 있습니다.
<code>*</code>	구성 요소가 지정한 주제에 대한 AWS IoT Core MQTT 메시지를 게시하고 구독하도록 허용합니다.	모든 주제에 대한 액세스를 허용하는 주제 문자열(<code>test/topic</code> 또는 <code>*</code>)입니다. MQTT 주제 와일드카드(<code>#</code> 및 <code>+</code>)를 사용하여 여러 리소스와 일치시킬 수 있습니다.

MQTT 권한 부여 정책의 AWS IoT Core MQTT 와일드카드

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 정책에서 `test/topic/#`에 대한 액세스 권한을 부여하는 경우 구성 요소는 `test/topic/#`을 구독할 수 있으며 `test/topic/filter`를 게시하고 구독할 수 있습니다.

AWS IoT Core MQTT 권한 부여 정책의 레시피 변수

[Greengrass nucleus](#)의 v2.6.0 이상을 사용하는 경우 권한 부여 정책에서 `{iot:thingName}` 레시피 변수를 사용할 수 있습니다. 이 기능을 사용하면 코어 디바이스 그룹에 대해 단일 권한 부여 정책을 구성할 수 있고, 이 경우 각 코어 디바이스는 자체 이름이 포함된 주제에만 액세스할 수 있습니다. 예를 들어 다음 주제 리소스에 대한 구성 요소 액세스를 허용할 수 있습니다.

```
devices/{iot:thingName}/messages
```

자세한 내용은 [레시피 변수](#) 및 [병합 업데이트에서 레시피 변수 사용](#)을(를) 참조하세요.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example제한 없는 액세스를 사용하는 권한 부여 정책 예제

다음 권한 부여 정책 예제에서는 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:
        - "*"

```

Example제한된 액세스를 사용하는 권한 부여 정책 예제

다음 권한 부여 정책 예제에서는 구성 요소가 `factory/1/events` 및 `factory/1/actions`라는 두 가지 주제를 게시하고 구독할 수 있도록 허용합니다.

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

```

Example코어 디바이스 그룹에 대한 권한 부여 정책 예제

Important

이 예제에서는 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있는 기능을 사용합니다. Greengrass nucleus v2.6.0에서는 구성 요소 구성에서 {iot:thingName}과 같은 대부분의 [레시피 변수](#)에 대한 지원을 추가합니다.

다음 권한 부여 정책 예제에서는 구성 요소가 구성 요소를 실행하는 코어 디바이스의 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

주제의 AWS IoT Core 에 MQTT 메시지를 게시합니다.

MQTT 메시지에 게시할 때 초당 100개의 트랜잭션 할당 AWS IoT Core량이 있습니다. 이 할당량을 초과하면 메시지가 Greengrass 디바이스에서 처리를 위해 대기열에 대기됩니다. 또한 초당 512Kb의 데이터 할당량과 초당 20,000개의 게시(일부에서는 2,000개)의 계정 전체 할당량도 있습니다 AWS 리전. AWS IoT Core에서 MQTT 메시지 브로커 한도에 대한 자세한 내용은 [AWS IoT Core 메시지 브로커와 프로토콜 한도 및 할당량](#)을 참조하세요.

이러한 할당량을 초과하면 Greengrass 디바이스가 메시지 게시를 제한합니다 AWS IoT Core. 메시지는 메모리의 스펙러에 저장됩니다. 기본적으로 스펙러에 할당된 메모리는 2.5Mb입니다. 스펙러가 채워지면 새 메시지가 거부됩니다. 스펙러의 크기를 늘릴 수 있습니다. 자세한 내용은 [Greengrass nucleus](#) 설명서의 [구성](#)(를) 참조하세요. 스펙러가 채워지고 할당된 메모리를 늘려야 하는 상황을 방지하려면 게시 요청을 초당 100개 이하의 요청으로 제한합니다.

애플리케이션에서 메시지를 더 높은 속도로 보내거나 더 큰 메시지를 보내야 하는 경우 [스트림 관리자](#)를 사용하여 Kinesis Data Streams로 메시지를 보내는 것이 좋습니다. 스트림 관리자 구성 요소는 AWS 클라우드로 대용량 데이터를 전송하기 위한 것입니다. 자세한 내용은 [Greengrass 코어 디바이스에서 데이터 스트림 관리](#) 단원을 참조하십시오.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

topicName(Python: topic_name)

메시지를 게시할 주제입니다.

qos

사용할 MQTT QoS입니다. 이 열거형 QoS의 값은 다음과 같습니다.

- AT_MOST_ONCE – QoS 0. MQTT 메시지가 최대 한 번 전송됩니다.
- AT_LEAST_ONCE – QoS 1. MQTT 메시지가 한 번 이상 전송됩니다.

payload

(선택 사항) Blob인 메시지 페이로드입니다.

MQTT 5를 사용할 때 [Greengrass nucleus](#)의 v2.10.0 이상에서는 다음 기능을 사용할 수 있습니다. MQTT 3.1.1을 사용하는 경우 이러한 기능은 무시됩니다. 다음 표에는 이러한 기능에 액세스하는 데 사용해야 하는 AWS IoT 디바이스 SDK의 최소 버전이 나열되어 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 형식입니다. payloadFormat을 설정하지 않으면 유형은 BYTES인 것으로 가정합니다. 이 열거형의 값은 다음과 같습니다.

- BYTES – 페이로드의 콘텐츠는 이진 blob입니다.
- UTF8 – 페이로드의 콘텐츠는 UTF8 문자열입니다.

retain

(선택 사항) 게시할 때 MQTT 유지 옵션을 true로 설정할지 여부를 나타냅니다.

userProperties

(선택 사항) 전송할 애플리케이션별 UserProperty 객체의 목록입니다. UserProperty 객체는 다음과 같이 정의됩니다.


```
UserProperty:
  key: string
  value: string
```

messageExpiryIntervalSeconds

(선택 사항) 메시지가 만료되고 서버에 의해 삭제되기 전까지의 시간(초)입니다. 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) 메시지의 콘텐츠 유형에 대한 애플리케이션별 식별자입니다.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example예: 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 (GreengrassCoreIPCClietV2 ipcClientV2 =
GreengrassCoreIPCClietV2.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)

Example예: 메시지 게시

Note

이 예제에서는 Python v2 AWS IoT Device SDK 용의 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.GreengrassCoreIPCClietV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()

```

Java (IPC client V1)

Example에: 메시지 게시

Note

이 예제에서는 IPCUtils 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```
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)

Example예: 메시지 게시

Note

이 예제에서는 Python v2 AWS IoT Device SDK 옹의 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++ (IPC client V1)

Example예: 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

Example예: 메시지 게시

```

import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} 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
            }

            this.ipcClient = await getIpClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpClient(){
    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();

```

Rust

Example예: 메시지 게시

```
use gg_sdk::{Qos, Sdk};
```



```
fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;

    sdk.publish_to_iot_core(topic, message, qos)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}
```

C

Example예: 메시지 게시

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    err = ggipc_publish_to_iot_core(topic, message, qos);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
```

```

        (int) topic.len,
        topic.data
    );
    exit(-1);
}

printf(
    "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
);
}

```

C++ (Component SDK)

Example예: 메시지 게시

```

#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    error = client.publish_to_iot_core(topic, message, qos);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}

```

SubscribeToIoTCore

주제 또는 주제 필터 AWS IoT Core 에서의 MQTT 메시지를 구독합니다. AWS IoT Greengrass 코어 소프트웨어는 구성 요소가 수명 주기 종료에 도달하면 구독을 제거합니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 종료를 처리하는 함수를 사용하여 스트림 응답 핸들러를 정의합니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

이벤트 메시지 유형: `IoTCoreMessage`

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`topicName`(Python: `topic_name`)

구독할 주제입니다. MQTT 주제 와일드카드(# 및 +)를 사용하여 여러 주제를 구독할 수 있습니다.

`qos`

사용할 MQTT QoS입니다. 이 열거형 QoS의 값은 다음과 같습니다.

- `AT_MOST_ONCE` – QoS 0. MQTT 메시지가 최대 한 번 전송됩니다.
- `AT_LEAST_ONCE` – QoS 1. MQTT 메시지가 한 번 이상 전송됩니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

MQTT 메시지의 스트림입니다. 이 객체 `IoTCoreMessage`에는 다음 정보가 포함됩니다.

`message`

MQTT 메시지입니다. 이 객체 `MQTTMessage`에는 다음 정보가 포함됩니다.

`topicName`(Python: `topic_name`)

메시지가 게시된 주제입니다.

`payload`

(선택 사항) Blob인 메시지 페이로드입니다.

MQTT 5를 사용할 때 [Greengrass nucleus](#)의 v2.10.0 이상에서는 다음 기능을 사용할 수 있습니다. MQTT 3.1.1을 사용하는 경우 이러한 기능은 무시됩니다. 다음 표에는 이러한 기능에 액세스하는 데 사용해야 하는 AWS IoT 디바이스 SDK의 최소 버전이 나열되어 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 형식입니다. payloadFormat을 설정하지 않으면 유형은 BYTES인 것으로 가정합니다. 이 열거형의 값은 다음과 같습니다.

- BYTES – 페이로드의 콘텐츠는 이진 blob입니다.
- UTF8 – 페이로드의 콘텐츠는 UTF8 문자열입니다.

retain

(선택 사항) 게시할 때 MQTT 유지 옵션을 true로 설정할지 여부를 나타냅니다.

userProperties

(선택 사항) 전송할 애플리케이션별 UserProperty 객체의 목록입니다. UserProperty 객체는 다음과 같이 정의됩니다.

```
UserProperty:
  key: string
  value: string
```

messageExpiryIntervalSeconds

(선택 사항) 메시지가 만료되고 서버에 의해 삭제되기 전까지의 시간(초)입니다. 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) 메시지의 콘텐츠 유형에 대한 애플리케이션별 식별자입니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V2)

Example예: 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 (GreengrassCoreIPCCliientV2 ipcClientV2 =
GreengrassCoreIPCCliientV2.builder().build()) {
    SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
        .withTopicName(topic)
        .withQos(qos);

    GreengrassCoreIPCCliientV2.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)

Example에: 메시지 구독

Note

이 예제에서는 Python v2 AWS IoT Device SDK 용의 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)

Example에: 메시지 구독

Note

이 예제에서는 IPCUtils 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```

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(GreengrassCoreIPCCClient greengrassCoreIPCCClient, String topic,
QoS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
    SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
    subscribeToIoTCoreRequest.setTopicName(topic);
    subscribeToIoTCoreRequest.setQos(qos);
    return
greengrassCoreIPCCClient.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)

Example예: 메시지 구독

Note

이 예제에서는 Python v2 AWS IoT Device SDK 용의 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++ (IPC client V1)

Example예: 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

Example예: 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();

```

Rust

Example예: 메시지 구독

```

use gg_sdk::{Qos, Sdk};
use std::{thread, time::Duration};

fn main() {

```



```

let sdk = Sdk::init();
sdk.connect().expect("Failed to establish IPC connection");

let topic = "my/topic";
let qos = Qos::AtLeastOnce;

let callback = |topic: &str, payload: &[u8]| {
    let message = String::from_utf8_lossy(payload);
    println!("Received new message on topic {topic}: {message}");
};

let _sub = sdk
    .subscribe_to_iot_core(topic, qos, &callback)
    .expect("Failed to subscribe to topic");

println!("Successfully subscribed to topic: {topic}");

// Keep the main thread alive, or the process will exit.
loop {
    thread::sleep(Duration::from_secs(10));
}
}

```

C

Example예: 메시지 구독

```

#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    printf(
        "Received new message on topic %.*s: %.*s\n",
        (int) topic.len,
        topic.data,

```

```
        (int) payload.len,
        payload.data
    );
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_iot_core(
        topic, qos, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the subscription handle.
    ggipc_close_subscription(handle);
}
```

C++ (Component SDK)

Example예: 메시지 구독

```
#include <gg/ipc/client.hpp>
#include <unistd.h>
#include <iostream>

class ResponseHandler : public gg::ipc::IotTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Buffer payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        std::cout << "Received new message on topic " << topic << ": "
            << payload << "\n";
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    static ResponseHandler handler;
    error = client.subscribe_to_iot_core(topic, qos, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}
```

```
}
}
```

예제

다음 예제를 사용하여 구성 요소에서 AWS IoT Core MQTT IPC 서비스를 사용하는 방법을 알아봅니다.

예제 AWS IoT Core MQTT 게시자(C++, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

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://amzn-s3-demo-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://amzn-s3-demo-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 Core MQTT 구독자 예제(C++, IPC 클라이언트 V1)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

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://amzn-s3-demo-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://amzn-s3-demo-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;
}

```

AWS IoT Core 예제 MQTT 게시자(Rust)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherRust",
  "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.IoTCorePublisherRust:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  }
}

```

```

},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "{artifacts:path}/publish_to_iot_core"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherRust/1.0.0/publish_to_iot_core",
        "Permission": {
          "Execute": "OWNER"
        }
      }
    ]
  }
]
}

```

다음 Rust 애플리케이션 예제에서는 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 게시하는 방법을 보여줍니다 AWS IoT Core.

```

use gg_sdk::{Qos, Sdk};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;

    sdk.publish_to_iot_core(topic, message, qos)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}

```

AWS IoT Core MQTT 구독자 예제(Rust)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberRust",
  "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.IoTCoreSubscriberRust:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/subscribe_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberRust/1.0.0/subscribe_to_iot_core",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

```
}
```

다음 Rust 애플리케이션 예제에서는 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 구독하는 방법을 보여줍니다 AWS IoT Core.

```
use gg_sdk::{Qos, Sdk};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;

    let callback = |topic: &str, payload: &[u8]| {
        let message = String::from_utf8_lossy(payload);
        println!("Received new message on topic {topic}: {message}");
    };

    let _sub = sdk
        .subscribe_to_iot_core(topic, qos, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

예제 AWS IoT Core MQTT 게시자(C)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherC",
  "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.IoTCorePublisherC:mqttproxy:1": {
        "policyDescription": "Allows access to publish to all topics.",
        "operations": ["aws.greengrass#PublishToIoTCore"],
        "resources": ["*"]
      }
    }
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "{artifacts:path}/sample_publish_to_iot_core"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherC/1.0.0/sample_publish_to_iot_core",
        "Permission": {
          "Execute": "OWNER"
        }
      }
    ]
  }
]
}

```

다음 예제 C 애플리케이션은 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 게시하는 방법을 보여줍니다 AWS IoT Core.

```

#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

```



```

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    err = ggipc_publish_to_iot_core(topic, message, qos);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}

```

AWS IoT Core 예제 MQTT 구독자(C)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberC",
  "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.IoTCoreSubscriberC:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["*"]
        }
    },
    "Manifests": [
        {
            "Platform": {
                "os": "linux",
                "runtime": "*"
            },
            "Lifecycle": {
                "run": "{artifacts:path}/sample_subscribe_to_iot_core"
            },
            "Artifacts": [
                {
                    "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberC/1.0.0/sample_subscribe_to_iot_core",
                    "Permission": {
                        "Execute": "OWNER"
                    }
                }
            ]
        }
    ]
}

```

다음 예제 C 애플리케이션은 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 구독하는 방법을 보여줍니다 AWS IoT Core.

```

#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle

```

```
) {
    (void) ctx;
    (void) handle;

    printf(
        "Received new message on topic %.*s: %.*s\n",
        (int) topic.len,
        topic.data,
        (int) payload.len,
        payload.data
    );
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_iot_core(
        topic, qos, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
```

```

while (1) {
    sleep(10);
}

// To stop subscribing, close the subscription handle.
ggipc_close_subscription(handle);
}

```

예제 AWS IoT Core MQTT 게시자(C++, 구성 요소 SDK)

다음 예제 레시피에서는 구성 요소가 모든 주제에 게시할 수 있도록 허용합니다.

```

{
  "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": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_cpp_publish_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/sample_cpp_publish_to_iot_core",

```

```

        "Permission": {
            "Execute": "OWNER"
        }
    }
]
}
]
}

```

다음 예제 C++ 애플리케이션은 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 게시하는 방법을 보여줍니다 AWS IoT Core.

```

#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    error = client.publish_to_iot_core(topic, message, qos);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}

```

예제 AWS IoT Core MQTT 구독자(C++, 구성 요소 SDK)

다음 예제 레시피에서는 구성 요소가 모든 주제를 구독하도록 허용합니다.

```

{

```

```

"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": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "{artifacts:path}/sample_cpp_subscribe_to_iot_core"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/sample_cpp_subscribe_to_iot_core",
        "Permission": {
          "Execute": "OWNER"
        }
      }
    ]
  }
]
}

```

다음 예제 C++ 애플리케이션은 AWS IoT Core MQTT IPC 서비스를 사용하여 메시지를 구독하는 방법을 보여줍니다 AWS IoT Core.

```
#include <gg/ipc/client.hpp>
#include <unistd.h>
#include <iostream>

class ResponseHandler : public gg::ipc::IotTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Buffer payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        std::cout << "Received new message on topic " << topic << ": "
            << payload << "\n";
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    static ResponseHandler handler;
    error = client.subscribe_to_iot_core(topic, qos, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}
```

구성 요소 수명 주기와 연동

구성 요소 수명 주기 IPC 서비스를 사용하여 다음을 수행할 수 있습니다.

- 코어 디바이스의 구성 요소 상태를 업데이트합니다.
- 구성 요소 상태 업데이트를 구독합니다.
- 배포 중에 업데이트를 적용할 구성 요소를 nucleus가 중지하지 못하게 합니다.
- 구성 요소 프로세스 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 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.2.10	
AWS IoT Device SDK Python v2용	v1.5.3	
AWS IoT Device SDK C++ v2용	v1.17.0	

SDK	최소 버전	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 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 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

구성 요소 수명 주기 관리를 위한 권한 부여 정책에는 다음 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ipc.lifecycle`

연산	설명	리소스
<code>aws.greengrass#PauseComponent</code>	구성 요소가 지정하는 구성 요소를 일시 중지하도록 허용합니다.	구성 요소 이름 또는 모든 구성 요소에 대한 액세스를 허용하려면 *
<code>aws.greengrass#ResumeComponent</code>	구성 요소가 지정하는 구성 요소를 다시 시작하도록 허용합니다.	구성 요소 이름 또는 모든 구성 요소에 대한 액세스를 허용하려면 *
*	구성 요소가 지정하는 구성 요소를 일시 중지 및 다시 시작하도록 허용합니다.	구성 요소 이름 또는 모든 구성 요소에 대한 액세스를 허용하려면 *

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example 권한 부여 정책 예제

다음 예제 권한 부여 정책에서는 구성 요소가 모든 구성 요소를 일시 중지 및 다시 시작할 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 디바이스에서 구성 요소의 상태를 업데이트합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

state

설정할 상태입니다. 이 열거형 LifecycleState의 값은 다음과 같습니다.

- RUNNING
- ERRORED

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Rust

Example에: 상태 업데이트

```
use gg_sdk::{ComponentState, Sdk};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Update component state to RUNNING
    sdk.update_state(ComponentState::Running)
        .expect("Failed to update component state");

    println!("Successfully updated component state to RUNNING.");
}
```

C

Example에: 상태 업데이트

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Update component state to RUNNING
    err = ggipc_update_state(GG_COMPONENT_STATE_RUNNING);
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to update component state.\n");
        exit(-1);
    }

    printf("Successfully updated component state to RUNNING.\n");
}
```

}

C++ (Component SDK)

Example예: 상태 업데이트

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Update component state to RUNNING
    error = client.update_component_state(GG_COMPONENT_STATE_RUNNING);
    if (error) {
        std::cerr << "Failed to update component state.\n";
        exit(-1);
    }

    std::cout << "Successfully updated component state to RUNNING.\n";
}
```

SubscribeToComponentUpdates

AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 업데이트하기 전에 알림을 수신하도록 구독합니다. 알림에서는 업데이트의 일부로 nucleus가 다시 시작되는지 여부를 지정합니다.

배포의 구성 요소 업데이트 정책에서 구성 요소에 알리도록 지정하는 경우에만 nucleus가 업데이트 알림을 보냅니다. 기본 동작은 구성 요소에 알리는 것입니다. 자세한 내용은 [배포 만들기](#) 섹션 및 [CreateDeployment](#) 작업을 직접 호출할 때 제공할 수 있는 [DeploymentComponentUpdatePolicy](#) 객체를 참조하세요.

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 종료를 처리하는 함수를 사용하여 스트림 응답 핸들러를 정의합니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

이벤트 메시지 유형: ComponentUpdatePolicyEvents

Tip

자습서에 따라 조건부로 구성 요소 업데이트를 연기하는 구성 요소를 개발하는 방법을 알아볼 수 있습니다. 자세한 내용은 [자습서: 구성 요소 업데이트가 지연되는 Greengrass 구성 요소 개발](#) 단원을 참조하십시오.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

messages

알림 메시지의 스트림입니다. 이 객체 ComponentUpdatePolicyEvents에는 다음 정보가 포함됩니다.

preUpdateEvent(Python: pre_update_event)

(선택 사항) Nucleus가 구성 요소를 업데이트하려고 함을 나타내는 이벤트입니다. 구성 요소가 다시 시작될 준비가 될 때까지 [DeferComponentUpdate](#) 작업에 응답하여 업데이트를 승인하거나 연기할 수 있습니다. 이 객체 PreComponentUpdateEvent에는 다음 정보가 포함됩니다.

deploymentId(Python: deployment_id)

구성 요소를 업데이트하는 AWS IoT Greengrass 배포의 ID입니다.

isGgcRestarting(Python: is_ggc_restarting)

업데이트를 적용하기 위해 nucleus를 다시 시작해야 하는지 여부입니다.

postUpdateEvent(Python: post_update_event)

(선택 사항) Nucleus가 구성 요소를 업데이트했음을 나타내는 이벤트입니다. 이 객체 PostComponentUpdateEvent에는 다음 정보가 포함됩니다.

deploymentId(Python: deployment_id)

구성 요소를 업데이트한 AWS IoT Greengrass 배포의 ID입니다.

Note

이 기능은 Greengrass nucleus 구성 요소의 v2.7.0 이상에서 사용할 수 있습니다.

DeferComponentUpdate

[SubscribeToComponentUpdates](#)로 검색하는 구성 요소 업데이트를 승인하거나 연기합니다. 구성 요소가 구성 요소 업데이트를 진행할 준비가 되었는지 nucleus가 다시 확인하기 전까지 기다릴 시간을 지정합니다. 이 작업을 사용하여 구성 요소가 업데이트 준비가 되었음을 nucleus에 알릴 수도 있습니다.

구성 요소가 구성 업데이트 알림에 응답하지 않으면 nucleus는 배포의 구성 업데이트 정책에 지정한 시간을 기다립니다. 이 제한 시간이 지나면 nucleus는 배포를 진행합니다. 기본 구성 요소 업데이트 제한 시간은 60초입니다. 자세한 내용은 [배포 만들기](#) 섹션 및 [CreateDeployment](#) 작업을 직접 호출할 때 제공할 수 있는 [DeploymentComponentUpdatePolicy](#) 객체를 참조하세요.

Tip

자습서에 따라 조건부로 구성 요소 업데이트를 연기하는 구성 요소를 개발하는 방법을 알아볼 수 있습니다. 자세한 내용은 [자습서: 구성 요소 업데이트가 지연되는 Greengrass 구성 요소 개발](#) 단원을 참조하십시오.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

deploymentId(Python: deployment_id)

연기할 AWS IoT Greengrass 배포의 ID입니다.

message

(선택 사항) 업데이트를 연기할 구성 요소의 이름입니다.

기본값은 요청을 하는 구성 요소의 이름입니다.

`recheckAfterMs`(Python: `recheck_after_ms`)

업데이트를 연기할 시간(밀리초)입니다. Nucleus는 이 시간 동안 기다린 다음 [SubscribeToComponentUpdates](#)로 검색할 수 있는 다른 `PreComponentUpdateEvent`를 보냅니다.

업데이트를 승인하려면 0을 지정합니다. 이렇게 하면 구성 요소의 업데이트가 준비되었음을 nucleus에 알립니다.

기본값은 0밀리초이며, 업데이트를 승인하는 것을 나타냅니다.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

PauseComponent

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

코어 디바이스에서 구성 요소의 프로세스를 일시 중지합니다. 구성 요소를 다시 시작하려면 [ResumeComponent](#) 작업을 사용합니다.

일반 구성 요소만 일시 중지할 수 있습니다. 다른 유형의 구성 요소를 일시 중지하려고 하면 이 작업에서 `InvalidRequestError`를 throw합니다.

Note

이 작업은 Docker 컨테이너와 같은 컨테이너화된 프로세스를 일시 중지할 수 없습니다. Docker 컨테이너를 일시 중지하고 다시 시작하려면 [도커 일시 중지](#) 및 [도커 일시 중지 해제](#) 명령을 사용할 수 있습니다.

이 작업은 구성 요소 종속성 또는 일시 중지하는 구성 요소에 종속된 구성 요소를 일시 중지하지 않습니다. 이 동작은 다른 구성 요소의 종속성인 구성 요소를 일시 중지할 때 사용하는 것이 좋습니다. 해당 종속성이 일시 중지되면 종속 구성 요소에 문제가 발생할 수 있기 때문입니다.

일시 중지된 구성 요소를 배포 등을 통해 다시 시작하거나 종료하면 Greengrass nucleus는 구성 요소를 다시 시작하고 해당 종료 수명 주기를 실행합니다. 구성 요소 다시 시작에 대한 자세한 내용은 [RestartComponent](#) 섹션을 참조하세요.

⚠ Important

이 작업을 사용하려면 이 작업을 사용할 수 있는 권한을 부여하는 권한 부여 정책을 정의해야 합니다. 자세한 내용은 [권한 부여](#) 단원을 참조하십시오.

최소 SDK 버전

다음 표에는 구성 요소를 일시 중지하고 재개하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.4.3	
AWS IoT Device SDK Python v2용	v1.6.2	
AWS IoT Device SDK C++ v2용	v1.13.1	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`componentName`(Python: `component_name`)

일시 중지할 구성 요소의 이름이며, 일반 구성 요소여야 합니다. 자세한 내용은 [구성 요소 유형](#) 단원을 참조하십시오.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

ResumeComponent

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.4.0 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

코어 디바이스에서 구성 요소의 프로세스를 다시 시작합니다. 구성 요소를 일시 중지하려면 [PauseComponent](#) 작업을 사용합니다.

일시 중지된 구성 요소만 다시 시작할 수 있습니다. 일시 중지되지 않은 구성 요소를 다시 시작하려고 하면 이 작업은 `InvalidRequestError`를 발생시킵니다.

Important

이 작업을 사용하려면 이 작업을 수행할 수 있는 권한을 부여하는 권한 부여 정책을 정의해야 합니다. 자세한 내용은 [권한 부여](#) 단원을 참조하십시오.

최소 SDK 버전

다음 표에는 구성 요소를 일시 중지하고 재개하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전
AWS IoT Device SDK Java v2용	v1.4.3
AWS IoT Device SDK Python v2용	v1.6.2
AWS IoT Device SDK C++ v2용	v1.13.1
AWS IoT Device SDK JavaScript v2용	v1.12.0

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

componentName(Python: component_name)

다시 시작할 구성 요소의 이름입니다.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

구성 요소 구성과 연동

구성 요소 구성 IPC 서비스를 사용하여 다음을 수행할 수 있습니다.

- 구성 요소 구성 파라미터를 가져오고 설정합니다.
- 구성 요소 구성 업데이트를 구독합니다.
- nucleus에서 구성 요소 구성 업데이트를 적용하기 전에 검증합니다.

주제

- [최소 SDK 버전](#)
- [GetConfiguration](#)
- [UpdateConfiguration](#)
- [SubscribeToConfigurationUpdate](#)
- [SubscribeToValidateConfigurationUpdates](#)
- [SendConfigurationValidityReport](#)

최소 SDK 버전

다음 표에는 구성 요소 구성과 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2	v1.2.10	
용		

SDK	최소 버전	
AWS IoT Device SDK Python v2용	v1.5.3	
AWS IoT Device SDK C++ v2용	v1.17.0	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

GetConfiguration

코어 디바이스의 구성 요소에 대한 구성 값을 가져옵니다. 구성 값을 가져올 키 경로를 지정합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`componentName`(Python: `component_name`)

(선택 사항) 구성 요소의 이름입니다.

기본값은 요청을 하는 구성 요소의 이름입니다.

`keyPath`(Python: `key_path`)

구성 값의 키 경로입니다. 각 항목이 구성 객체에서 단일 수준에 대한 키인 목록을 지정합니다. 예를 들어 다음 구성에서 `port`의 값을 가져오려면 `["mqtt", "port"]`를 지정합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

구성 요소의 전체 구성을 가져오려면 빈 목록을 지정합니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

componentName(Python: component_name)

구성 요소의 이름입니다.

value

객체인 요청된 구성입니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Rust

Example예: 구성 가져오기

```
use core::mem::MaybeUninit;
use gg_sdk::{Sdk, UnpackedObject};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Get a configuration value at key path ["mqtt", "port"]
    let mut buf = [MaybeUninit::uninit(); 1024];

    let value = sdk
        .get_config(&["mqtt", "port"], None, &mut buf)
        .expect("Failed to get configuration");

    if let UnpackedObject::I64(port) = value.unpack() {
        println!("Configuration value: {port}");
    }
}
```

C

Example예: 구성 가져오기

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
```

```

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Get a configuration value at key path ["mqtt", "port"]
    uint8_t response_mem[1024];
    GgObject value;

    err = ggipc_get_config(
        GG_BUF_LIST(GG_STR("mqtt"), GG_STR("port")),
        NULL, // component_name (NULL = current component)
        GG_BUF(response_mem),
        &value
    );
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to get configuration.\n");
        exit(-1);
    }

    if (gg_obj_type(value) == GG_TYPE_I64) {
        printf("Configuration value: %" PRIi64 "\n", gg_obj_into_i64(value));
    } else if (gg_obj_type(value) == GG_TYPE_BUF) {
        GgBuffer buf = gg_obj_into_buf(value);
        printf("Configuration value: %.*s\n", (int) buf.len, buf.data);
    } else {
        printf("Configuration value is of unexpected type.\n");
    }
}

```

C++ (Component SDK)

Example예: 구성 가져오기

```
#include <gg/ipc/client.hpp>
```

```
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Get a configuration value at key path ["mqtt", "port"]
    std::array key_path = { gg::Buffer { "mqtt" }, gg::Buffer { "port" } };
    int64_t value = 0;

    error = client.get_config(key_path, std::nullopt, value);
    if (error) {
        std::cerr << "Failed to get configuration.\n";
        exit(-1);
    }

    std::cout << "Configuration value: " << value << "\n";
}
```

UpdateConfiguration

코어 디바이스에서 이 구성 요소에 대한 구성 값을 업데이트합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

keyPath(Python: key_path)

(선택 사항) 업데이트할 컨테이너 노드(객체)의 키 경로입니다. 각 항목이 구성 객체에서 단일 수준에 대한 키인 목록을 지정합니다. 예를 들어 다음 구성에서 port의 값을 설정하려면 키 경로 ["mqtt"] 및 병합 값 { "port": 443 }을 지정합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

```
}

```

키 경로는 구성의 컨테이너 노드(객체)를 지정해야 합니다. 노드가 구성 요소의 구성에 없는 경우 이 작업은 노드를 생성하고 해당 값을 `valueToMerge`의 객체로 설정합니다.

기본값은 구성 객체의 루트입니다.

`timestamp`

현재 Unix epoch 시간(밀리초)입니다. 이 작업은 이 타임스탬프를 사용하여 키에 대한 동시 업데이트를 해결합니다. 구성 요소 구성의 키의 타임스탬프가 요청의 타임스탬프보다 크면 요청이 실패합니다.

`valueToMerge`(Python: `value_to_merge`)

`keyPath`에서 지정하는 위치에서 병합할 구성 객체입니다. 자세한 내용은 [구성 요소 구성 업데이트](#) 단원을 참조하십시오.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Rust

Example예: 구성 업데이트

```
use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Update configuration value at key path ["mqtt", "port"] to 443
    sdk.update_config(&["mqtt", "port"], None, 443)
        .expect("Failed to update configuration");

    println!("Successfully updated configuration.");
}
```

C

Example예: 구성 업데이트

```

#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Update configuration value at key path ["mqtt", "port"] to 443
    err = ggipc_update_config(
        GG_BUF_LIST(GG_STR("mqtt"), GG_STR("port")),
        NULL, // timestamp (NULL = current time)
        gg_obj_i64(443)
    );
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to update configuration.\n");
        exit(-1);
    }

    printf("Successfully updated configuration.\n");
}

```

C++ (Component SDK)

Example예: 구성 업데이트

```

#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();
}

```



```

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Update configuration value at key path ["mqtt", "port"] to 443
    std::array key_path = { gg::Buffer { "mqtt" }, gg::Buffer { "port" } };

    error = client.update_config(key_path, 443);
    if (error) {
        std::cerr << "Failed to update configuration.\n";
        exit(-1);
    }

    std::cout << "Successfully updated configuration.\n";
}

```

SubscribeToConfigurationUpdate

구성 요소의 구성이 업데이트될 때 알림을 수신하려면 구독합니다. 키를 구독하는 경우 해당 키의 하위 키가 업데이트되면 알림을 받습니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 종료를 처리하는 함수를 사용하여 스트림 응답 핸들러를 정의합니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

이벤트 메시지 유형: ConfigurationUpdateEvents

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

componentName(Python: component_name)

(선택 사항) 구성 요소의 이름입니다.

기본값은 요청을 하는 구성 요소의 이름입니다.

keyPath(Python: key_path)

구독할 구성 값의 키 경로입니다. 각 항목이 구성 객체에서 단일 수준에 대한 키인 목록을 지정합니다. 예를 들어 다음 구성에서 port의 값을 가져오려면 ["mqtt", "port"]를 지정합니다.

```
{
  "mqtt": {
    "port": 443
  }
}
```

구성 요소 구성에서 모든 값에 대한 업데이트를 구독하려면 빈 목록을 지정합니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

messages

알림 메시지의 스트림입니다. 이 객체 `ConfigurationUpdateEvents`에는 다음 정보가 포함됩니다.

`configurationUpdateEvent`(Python: `configuration_update_event`)

구성 업데이트 이벤트입니다. 이 객체 `ConfigurationUpdateEvent`에는 다음 정보가 포함됩니다.

`componentName`(Python: `component_name`)

구성 요소의 이름입니다.

`keyPath`(Python: `key_path`)

업데이트된 구성 값의 키 경로입니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Rust

Example예: 구성 업데이트 구독

```
use gg_sdk::Sdk;
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
```

```

    sdk.connect().expect("Failed to establish IPC connection");

    // Subscribe to configuration updates for key path ["mqtt"]
    let callback = |component_name: &str, key_path: &[&str]| {
        println!(
            "Received configuration update for component: {component_name}"
        );
        println!("Key path: {key_path:?}");
    };

    let _sub = sdk
        .subscribe_to_configuration_update(None, &["mqtt"], &callback)
        .expect("Failed to subscribe to configuration updates");

    println!("Successfully subscribed to configuration updates.");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}

```

C

Example예: 구성 업데이트 구독

```

#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx,
    GgBuffer component_name,
    GgList key_path,
    GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    printf(

```

```

        "Received configuration update for component: %.*s\n",
        (int) component_name.len,
        component_name.data
    );

    printf("Key path: [");
    for (size_t i = 0; i < key_path.len; i++) {
        if (i > 0) {
            printf(", ");
        }
        GgObject *obj = &key_path.items[i];
        if (gg_obj_type(*obj) == GG_TYPE_BUF) {
            GgBuffer key = gg_obj_into_buf(*obj);
            printf("\">%.*s\<", (int) key.len, key.data);
        }
    }
    printf("]\n");
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Subscribe to configuration updates for key path ["mqtt"]
    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_configuration_update(
        NULL, // component_name (NULL = current component)
        GG_BUF_LIST(GG_STR("mqtt")),
        on_subscription_response,
        NULL,
        &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to subscribe to configuration updates.\n");
        exit(-1);
    }

    printf("Successfully subscribed to configuration updates.\n");
}

```

```
// Keep the main thread alive, or the process will exit.
while (1) {
    sleep(10);
}

// To stop subscribing, close the stream.
ggipc_close_subscription(handle);
}
```

C++ (Component SDK)

Example예: 구성 업데이트 구독

```
#include <gg/ipc/client.hpp>
#include <unistd.h>
#include <iostream>

class ResponseHandler : public gg::ipc::ConfigurationUpdateCallback {
    void operator()(
        std::string_view component_name,
        gg::List key_path,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        std::cout << "Received configuration update for component: "
                    << component_name << "\n";
        std::cout << "Key path: [";
        for (size_t i = 0; i < key_path.size(); i++) {
            if (i > 0) {
                std::cout << ", ";
            }
            std::cout << "\"" << get<gg::Buffer>(key_path[i]) << "\"";
        }
        std::cout << "]\n";
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
    }
}
```

```

        exit(-1);
    }

    // Subscribe to configuration updates for key path ["mqtt"]
    std::array key_path = { gg::Buffer { "mqtt" } };

    static ResponseHandler handler;
    error = client.subscribe_to_configuration_update(
        key_path, std::nullopt, handler
    );
    if (error) {
        std::cerr << "Failed to subscribe to configuration updates.\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to configuration updates.\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}

```

SubscribeToValidateConfigurationUpdates

이 구성 요소의 구성이 업데이트되기 전에 알림을 수신하려면 구독합니다. 그러면 구성 요소가 자체 구성에 대한 업데이트를 검증할 수 있습니다. [SendConfigurationValidityReport](#) 작업을 사용하면 구성이 유효한지 여부를 nucleus에 알립니다.

Important

로컬 배포에서는 업데이트에 대해 구성 요소에 알리지 않습니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 종료를 처리하는 함수를 사용하여 스트림 응답 핸들러를 정의합니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

이벤트 메시지 유형: `ValidateConfigurationUpdateEvents`

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

messages

알림 메시지의 스트림입니다. 이 객체 `ValidateConfigurationUpdateEvents`에는 다음 정보가 포함됩니다.

```
validateConfigurationUpdateEvent(Python:  
validate_configuration_update_event)
```

구성 업데이트 이벤트입니다. 이 객체 `ValidateConfigurationUpdateEvent`에는 다음 정보가 포함됩니다.

```
deploymentId(Python: deployment_id)
```

구성 요소를 업데이트하는 AWS IoT Greengrass 배포의 ID입니다.

```
configuration
```

새 구성을 포함하는 객체입니다.

SendConfigurationValidityReport

이 구성 요소에 대한 구성 업데이트가 유효한지 여부를 `nucleus`에 알립니다. 새 구성이 유효하지 않고 `nucleus`에 알리면 배포가 실패합니다. [SubscribeToValidateConfigurationUpdates](#) 작업을 사용하면 구성 업데이트 검증을 구독할 수 있습니다.

구성 요소가 구성 업데이트 검증 알림에 응답하지 않으면 `nucleus`는 배포의 구성 검증 정책에 지정한 시간을 기다립니다. 이 제한 시간이 지나면 `nucleus`는 배포를 진행합니다. 기본 구성 요소 검증 제한 시간은 20초입니다. 자세한 내용은 [배포 만들기](#) 섹션 및 [CreateDeployment](#) 작업을 직접 호출할 때 제공할 수 있는 [DeploymentConfigurationValidationPolicy](#) 객체를 참조하세요.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

configurationValidityReport(Python: configuration_validity_report)

구성 업데이트가 유효한지 여부를 nucleus에 알려주는 보고서입니다. 이 객체 ConfigurationValidityReport에는 다음 정보가 포함됩니다.

status

유효성 상태입니다. 이 열거형 ConfigurationValidityStatus의 값은 다음과 같습니다.

- ACCEPTED - 구성이 유효하며 nucleus가 이 구성 요소에 적용할 수 있습니다.
- REJECTED - 구성이 유효하지 않고 배포가 실패합니다.

deploymentId(Python: deployment_id)

구성 업데이트를 요청한 AWS IoT Greengrass 배포의 ID입니다.

message

(선택 사항) 구성이 유효하지 않은 이유를 보고하는 메시지입니다.

응답

이 작업의 응답에는 어떠한 정보도 제공하지 않습니다.

보안 암호 값 검색

보안 암호 관리자 IPC 서비스를 사용하여 코어 디바이스의 보안 암호에서 보안 암호 값을 검색할 수 있습니다. [보안 암호 관리자 구성 요소](#)를 사용하여 암호화된 보안 암호를 코어 디바이스에 배포합니다. 그런 다음 IPC 작업을 사용하여 보안 암호를 복호화하고 사용자 지정 구성 요소에서 해당 값을 사용할 수 있습니다.

주제

- [최소 SDK 버전](#)
- [권한 부여](#)
- [GetSecretValue](#)
- [예제](#)

최소 SDK 버전

다음 표에는 코어 디바이스의 보안 암호에서 보안 암호 값을 검색하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.2.10	
AWS IoT Device SDK Python v2용	v1.5.3	
AWS IoT Device SDK C++ v2용	v1.17.0	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 보안 암호 관리자를 사용하려면 구성 요소가 코어 디바이스에 저장하는 보안 암호의 값을 가져올 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

보안 암호 관리자에 대한 권한 부여 정책에는 다음 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.SecretManager`

연산	설명	리소스
<code>aws.greengrass#GetSecretValue</code> 또는 *	구성 요소가 코어 디바이스에서 암호화된 보안 암호의 값을 가져올 수 있도록 허용합니다.	Secrets Manager 보안 암호 ARN 또는 모든 보안 암호에 대한 액세스를 허용하려면 *입니다.

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example 권한 부여 정책 예제

다음 예제 권한 부여 정책에서는 구성 요소가 코어 디바이스에서 보안 암호 값을 가져올 수 있도록 허용합니다.

Note

프로덕션 환경에서는 구성 요소가 사용하는 보안 암호만 검색하도록 권한 부여 정책의 범위를 줄이는 것이 좋습니다. 구성 요소를 배포할 때 * 와일드카드를 보안 암호 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 저장하는 보안 암호의 값을 가져옵니다.

이 작업은 AWS 클라우드에서 보안 암호의 값을 가져오는 데 사용할 수 있는 Secrets Manager 작업과 유사합니다. 자세한 내용은 AWS Secrets Manager API 참조의 [GetSecretValue](#)를 참조하세요.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

refresh(Python: refresh)

(선택 사항): 요청된 보안 암호를 AWS Secrets Manager 서비스의 최신 값과 동기화할지 여부입니다.

true로 설정하면 보안 암호 관리자가 지정된 보안 암호 레이블의 최신 값을 AWS Secrets Manager 서비스에 요청하고 해당 값을 응답으로 반환합니다. 그렇지 않으면 로컬에 저장된 보안 암호 값이 반환됩니다.

이 파라미터는 요청의 `versionId` 파라미터와 함께 사용할 수 없습니다. 이 파라미터는 Nucleus 2.13.0 이상과 함께 사용할 때 작동합니다.

secretId(Python: secret_id)

가져올 보안 암호의 이름입니다. Amazon 리소스 이름(ARN) 또는 보안 암호의 표시 이름을 지정할 수 있습니다.

versionId(Python: version_id)

(선택 사항) 가져올 버전의 ID입니다.

`versionId` 또는 `versionStage`를 지정할 수 있습니다.

`versionId` 또는 `versionStage`를 지정하지 않으면 이 작업의 기본값은 `AWSCURRENT` 레이블이 있는 버전입니다.

versionStage(Python: version_stage)

(선택 사항) 가져올 버전의 스테이징 레이블입니다.

`versionId` 또는 `versionStage`를 지정할 수 있습니다.

`versionId` 또는 `versionStage`를 지정하지 않으면 이 작업의 기본값은 `AWSCURRENT` 레이블이 있는 버전입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

secretId(Python: secret_id)

보안 암호의 ID입니다.

`versionId`(Python: `version_id`)

이 보안 암호 버전의 ID입니다.

`versionStage`(Python: `version_stage`)

이 보안 암호 버전에 연결된 스테이징 레이블의 목록입니다.

`secretValue`(Python: `secret_value`)

이 보안 암호 버전의 값입니다. 이 객체 `SecretValue`에는 다음 정보가 포함됩니다.

`secretString`(Python: `secret_string`)

Secrets Manager에 문자열로 제공한 보호된 보안 암호 정보의 복호화된 부분입니다.

`secretBinary`(Python: `secret_binary`)

(선택 사항) Secrets Manager에 바이트 배열 형식의 이진 데이터로 제공한 보호된 보안 암호 정보의 복호화된 부분입니다. 이 속성에는 base64로 인코딩된 문자열인 이진 데이터가 포함됩니다.

Secrets Manager 콘솔에서 보안 암호를 생성한 경우 이 속성은 사용되지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example예: 보안 암호 값 가져오기

Note

이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```
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)

Example예: 보안 암호 값 가져오기

Note

이 예제에서는 Python v2 AWS IoT Device SDK 용의 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

Example예: 보안 암호 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 getIpClient(){
    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 보안 암호 관리자 IPC 서비스를 사용하는 방법을 알아봅니다.

예: 보안 암호 출력(Python, IPC 클라이언트 V1)

이 예제 구성 요소에서는 코어 디바이스에 배포하는 보안 암호의 값을 출력합니다.

Important

이 예제 구성 요소는 보안 암호의 값을 출력하므로 테스트 데이터를 저장하는 보안 암호로만 사용하세요. 중요한 정보를 저장하는 보안 암호의 값을 출력하기 위해 이 구성 요소를 사용하지 마세요.

주제

- [방법](#)
- [아티팩트](#)
- [사용법](#)

방법

다음 예제 레시피에서는 보안 암호 ARN 구성 파라미터를 정의하고 구성 요소가 코어 디바이스에서 임의의 보안 암호 값을 가져올 수 있도록 허용합니다.

Note

프로덕션 환경에서는 구성 요소가 사용하는 보안 암호만 검색하도록 권한 부여 정책의 범위를 줄이는 것이 좋습니다. 구성 요소를 배포할 때 * 와일드카드를 보안 암호 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 awsiodsdk
        Run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
- Platform:
    os: windows
    Lifecycle:
        install: py -3 -m pip install --user awsiodsdk
        Run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"

```

아티팩트

다음 예제 Python 애플리케이션에서는 보안 암호 관리자 IPC 서비스를 사용하여 코어 디바이스에서 보안 암호 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)

```

사용법

이 예제 구성 요소를 [보안 암호 관리자 구성 요소](#)와 함께 사용하여 코어 디바이스의 보안 암호 값을 배포하고 출력할 수 있습니다.

테스트 보안 암호를 생성, 배포 및 출력하려면

1. 테스트 데이터를 사용하여 Secrets Manager 보안 암호를 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 보안 암호의 ARN을 저장합니다.

자세한 내용은 AWS Secrets Manager 사용 설명서의 [보안 암호 생성](#)을 참조하세요.

2. 다음 구성 병합 업데이트를 사용하여 [보안 암호 관리자 구성 요소](#)(aws.greengrass.SecretManager)를 배포합니다. 이전에 생성한 보안 암호의 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 deployment 명령](#)을 참조하세요.

3. 다음 구성 병합 업데이트를 사용하여 이 섹션의 예제 구성 요소를 생성하고 배포합니다. 이전에 생성한 보안 암호의 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](#) 섹션을 참조하세요.

4. AWS IoT Greengrass 코어 소프트웨어 로그를 보고 배포가 성공했는지 확인하고 구성 `com.example.PrintSecret` 요소 로그를 보고 인쇄된 보안 암호 값을 확인합니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

로컬 새도와 연동

새도 IPC 서비스를 사용하여 디바이스의 로컬 새도와 연동합니다. 연동할 디바이스로 코어 디바이스나 연결된 클라이언트 디바이스를 선택할 수 있습니다.

이러한 IPC 작업을 사용하려면 [새도 관리자 구성 요소](#)를 사용자 지정 구성 요소의 종속성으로 포함합니다. 그런 다음 사용자 지정 구성 요소에서 IPC 작업을 사용하여 새도 관리자를 통해 디바이스의 로컬 새도와 연동할 수 있습니다. 사용자 지정 구성 요소가 로컬 새도 상태 변경에 대응할 수 있도록 하려면

게시/구독 IPC 서비스를 사용하여 새도 이벤트를 구독할 수도 있습니다. 게시/구독 서비스 사용에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.

Note

코어 디바이스가 클라이언트 디바이스 새도와 연동할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정](#)을 참조하세요.

주제

- [최소 SDK 버전](#)
- [권한 부여](#)
- [GetThingShadow](#)
- [UpdateThingShadow](#)
- [DeleteThingShadow](#)
- [ListNamedShadowsForThing](#)

최소 SDK 버전

다음 표에는 로컬 새도와 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.4.0	
AWS IoT Device SDK Python v2용	v1.6.0	
AWS IoT Device SDK C++ v2용	v1.17.0	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 새도 IPC 서비스를 사용하려면 구성 요소가 새도와 연동할 수 있도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

새도 연동에 대한 권한 부여 정책에는 다음 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.ShadowManager`

연산	설명	리소스
<code>aws.greengrass#GetThingShadow</code>	구성 요소가 사물의 새도를 검색할 수 있도록 허용합니다.	다음 문자열 중 하나입니다. <code>\$aws/thinggs/ <i>thingName</i> /shadow/</code> - 클래식 디바이스 새도에 대한 액세스를 허용합니다. <code>\$aws/thinggs/ <i>thingName</i> /shadow/name/ <i>shadowName</i></code> - 이름 지정된 새도에 대한 액세스를 허용합니다. <code>*</code> - 모든 새도에 대한 액세스를 허용합니다.
<code>aws.greengrass#UpdateThingShadow</code>	구성 요소가 사물의 새도를 업데이트할 수 있도록 허용합니다.	다음 문자열 중 하나입니다. <code>\$aws/thinggs/ <i>thingName</i> /shadow/</code> - 클래식 디바이스 새도에 대한 액세스를 허용합니다. <code>\$aws/thinggs/ <i>thingName</i> /shadow/n</code>

연산	설명	리소스
		ame/ <i>shadowName</i> - 이름 지정된 새도에 대한 액세스를 허용합니다. * - 모든 새도에 대한 액세스를 허용합니다.
aws.greengrass#DeleteThingShadow	구성 요소가 사물의 새도를 삭제할 수 있도록 허용합니다.	다음 문자열 중 하나입니다. \$aws/thinggs/ <i>thingName</i> /shadow/ - 클래식 디바이스 새도에 대한 액세스를 허용합니다. \$aws/thinggs/ <i>thingName</i> /shadow/name/ <i>shadowName</i> - 이름 지정된 새도에 대한 액세스를 허용합니다. * - 모든 새도에 대한 액세스를 허용합니다.
aws.greengrass#ListNamedShadowsForThing	구성 요소가 사물의 이름 지정된 새도 목록을 검색할 수 있도록 허용합니다.	새도를 사물에 액세스하여 해당 새도를 나열하도록 허용하는 사물 이름 문자열입니다. 모든 사물에 대한 액세스를 허용하려면 *를 사용합니다.

IPC 서비스 식별자: aws.greengrass.ipc.pubsub

연산	설명	리소스
<code>aws.greengrass#SubscribeToTopic</code>	구성 요소가 에서 지정하는 주제에 대한 메시지를 구독할 수 있도록 허용합니다.	다음 주제 문자열 중 하나입니다. <code>shadowTopicPrefix / get/accepted</code> <code>shadowTopicPrefix / get/rejected</code> <code>shadowTopicPrefix / delete/accepted</code> <code>shadowTopicPrefix / delete/rejected</code> <code>shadowTopicPrefix / update/accepted</code> <code>shadowTopicPrefix / update/delta</code> <code>shadowTopicPrefix / update/rejected</code> 주제 접두사 <code>shadowTopicPrefix</code> 의 값은 색도 유형에 따라 달라집니다. - 클래식 색도: <code>\$aws/things/ thingName /shadow</code> - 이름 지정된 색도: <code>\$aws/things/ thingName /shadow/name/ shadowName</code> 모든 주제에 대한 액세스를 허용하려면 *를 사용합니다.

연산	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 정책에서 test/topic/#에 대한 액세스 권한을 부여하는 경우 구성 요소는 test/topic/# 는 구독할 수 있지만 test/topic/filter 는 구독할 수 없습니다.

로컬 새도 권한 부여 정책의 레시피 변수

[Greengrass nucleus](#) v2.6.0 이상을 사용하고 Greengrass nucleus의 [interpolateComponentConfiguration](#) 구성 옵션을 true로 설정한 경우 권한 부여 정책에서 {iot:thingName} [레시피 변수](#)를 사용할 수 있습니다. 이 기능을 사용하면 코어 디바이스 그룹에 대해 단일 권한 부여 정책을 구성할 수 있고, 이 경우 각 코어 디바이스는 자체 새도만 액세스할 수 있습니다. 예를 들어 새도 IPC 작업을 위해 다음 리소스에 대한 구성 요소 액세스를 허용할 수 있습니다.

```
$aws/things/{iot:thingName}/shadow/
```

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example예: 코어 디바이스 그룹이 로컬 새도와 연동하도록 허용

Important

이 예제에서는 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있는 기능을 사용합니다. Greengrass nucleus v2.6.0에서는 구성 요소 구성에서 {iot:thingName}과 같은 대부분의 [레시피 변수](#)에 대한 지원을 추가합니다. 이 기능을 활성화하려면 Greengrass nucleus

의 [interpolateComponentConfiguration](#) 구성 옵션을 `true`로 설정합니다. Greengrass nucleus의 모든 버전에서 작동하는 예제는 [단일 코어 디바이스에 대한 예제 권한 부여 정책](#)을 참조하세요.

다음 예제 권한 부여 정책에서는 구성 요소 `com.example.MyShadowInteractionComponent`가 구성 요소를 실행하는 코어 디바이스의 클래식 디바이스 새도 및 이름 지정된 새도 `myNamedShadow`와 연동할 수 있도록 허용합니다. 또한 이 정책에서는 이 구성 요소가 이러한 새도의 로컬 주제에 대한 메시지를 수신하도록 허용합니다.

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

```

Example예: 코어 디바이스 그룹이 클라이언트 디바이스 새도와 연동하도록 허용

Important

이 기능에는 [Greengrass nucleus](#) v2.6.0 이상, [새도 관리자](#) v2.2.0 이상 및 [MQTT 브리지](#) v2.2.0 이상이 필요합니다. [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 MQTT 브리지를 구성해야 합니다.](#)

다음 예제 권한 부여 정책에서는 구성 요소 `com.example.MyShadowInteractionComponent`가 이름이 `MyClientDevice`로 시작하는 클라이언트 디바이스의 모든 디바이스 새도와 연동할 수 있도록 허용합니다.

Note

코어 디바이스가 클라이언트 디바이스 새도와 연동할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정](#)을 참조하세요.

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*

```

Example에: 단일 코어 디바이스가 로컬 새도와 연동하도록 허용

다음 예제 권한 부여 정책에서는 구성 요소 `com.example.MyShadowInteractionComponent`가 디바이스 `MyThingName`의 클래식 디바이스 새도 및 이름 지정된 새도 `myNamedShadow`와 연동할 수 있도록 허용합니다. 또한 이 정책에서는 이 구성 요소가 이러한 새도의 로컬 주제에 대한 메시지를 수신하도록 허용합니다.

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

```

Example에: 코어 디바이스 그룹이 로컬 새도 상태 변경에 대응하도록 허용

Important

이 예제에서는 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있는 기능을 사용합니다. Greengrass nucleus v2.6.0에서는 구성 요소 구성에서 {iot:thingName}과 같은 대부분의 [레시피 변수](#)에 대한 지원을 추가합니다. 이 기능을 활성화하려면 Greengrass nucleus의 [interpolateComponentConfiguration](#) 구성 옵션을 true로 설정합니다. Greengrass nucleus의 모든 버전에서 작동하는 예제는 [단일 코어 디바이스에 대한 예제 권한 부여 정책](#)을 참조하세요.

다음 예제 액세스 제어 정책에서는 구성 요소 com.example.MyShadowReactiveComponent가 구성 요소를 실행하는 각 코어 디바이스에서 클래식 디바이스 새도 및 이름 지정된 새도 myNamedShadow의 /update/delta 주제에 대한 메시지를 수신할 수 있도록 허용합니다.

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
```

Example예: 단일 코어 디바이스가 로컬 새도 상태 변경에 대응하도록 허용

다음 예제 액세스 제어 정책에서는 구성 요소 `com.example.MyShadowReactiveComponent`가 디바이스 `MyThingName`의 클래식 디바이스 새도 및 이름 지정된 새도 `myNamedShadow`의 `/update/delta` 주제에 대한 메시지를 수신할 수 있도록 허용합니다.

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 사물의 새도를 가져옵니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`thingName`(Python: `thing_name`)

사물의 이름입니다.

유형: `string`

shadowName(Python: shadow_name)

새도의 이름입니다. 사물의 클래식 새도를 지정하려면 이 파라미터를 빈 문자열("")로 설정합니다.

 Warning

AWS IoT Greengrass 서비스는 AWSManagedGreengrassV2Deployment명명된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명명된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

유형: string

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

payload

Blob인 응답 상태 문서입니다.

유형: 다음 정보를 포함하는 object입니다.

state

상태 정보입니다.

이 객체에는 다음 정보가 포함됩니다.

desired

디바이스에서 업데이트하도록 요청된 상태 속성 및 값입니다.

유형: 키-값 페어의 map

reported

디바이스에서 보고한 상태 속성 및 값입니다.

유형: 키-값 페어의 map

delta

상태 속성 및 값의 원하는 상태와 보고된 상태 사이의 차이입니다. 이 속성은 `desired` 및 `reported` 상태가 다른 경우에만 있습니다.

유형: 키-값 페어의 map

metadata

언제 상태가 업데이트되었는지 확인할 수 있는 `desired` 및 `reported` 섹션의 각 속성에 대한 타임스탬프입니다.

유형: string

timestamp

응답이 생성된 epoch 날짜 및 시간입니다.

유형: integer

clientToken(Python: clientToken)

요청과 해당 응답을 일치시키는 데 사용되는 토큰입니다.

유형: string

version

로컬 새도 문서의 버전입니다.

유형: integer

오류

이 작업은 다음 오류를 반환할 수 있습니다.

InvalidArgumentsError

로컬 새도 서비스에서 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON 또는 지원되지 않는 문자가 포함된 경우 발생할 수 있습니다.

ResourceNotFoundError

요청된 로컬 새도 문서를 찾을 수 없습니다.

ServiceError

내부 서비스 오류가 발생했거나, IPC 서비스에 대한 요청 수가 새도 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 파라미터에 지정된 제한을 초과했습니다.

UnauthorizedError

구성 요소의 권한 부여 정책에 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example에: 사물 새도 가져오기

Note

이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```
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()) {
            GreengrassCoreIPCCClient ipcClient =
                new GreengrassCoreIPCCClient(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)

Example에: 사물 새도 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

Example예: 사물 새도 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 cases
        throw err
    }
}

const startScript = new GetThingShadow();

```

UpdateThingShadow

지정된 사물의 새도를 업데이트합니다. 새도가 존재하지 않을 경우 새도가 생성됩니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`thingName(Python: thing_name)`

사물의 이름입니다.

유형: `string`

`shadowName(Python: shadow_name)`

새도의 이름입니다. 사물의 클래식 새도를 지정하려면 이 파라미터를 빈 문자열("")로 설정합니다.

 Warning

AWS IoT Greengrass 서비스는 `AWSManagedGreengrassV2Deployment` 명령된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명령된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

유형: `string`

`payload`

Blob인 요청 상태 문서입니다.

유형: 다음 정보를 포함하는 `object`입니다.

`state`

업데이트할 상태 정보입니다. 이 IPC 작업은 지정된 필드에만 영향을 미칩니다.

이 객체에는 다음 정보가 포함됩니다. 일반적으로 `desired` 또는 `reported` 속성 중 하나를 사용하지만 동일한 요청에서는 둘 다 사용하지는 않습니다.

`desired`

디바이스에서 업데이트하도록 요청된 상태 속성 및 값입니다.

유형: 키-값 페어의 `map`

`reported`

디바이스에서 보고한 상태 속성 및 값입니다.

유형: 키-값 페어의 `map`

`clientToken`(Python: `client_token`)

(선택 사항) 클라이언트 토큰으로 요청과 해당 응답을 일치시키는 데 사용되는 토큰입니다.

유형: `string`

`version`

(선택 사항) 업데이트할 로컬 새도 문서의 버전입니다. 새도 서비스에서는 지정된 버전이 서비스에서 보유하는 최신 버전과 일치하는 경우에만 업데이트를 처리합니다.

유형: `integer`

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`payload`

Blob인 응답 상태 문서입니다.

유형: 다음 정보를 포함하는 `object`입니다.

`state`

상태 정보입니다.

이 객체에는 다음 정보가 포함됩니다.

`desired`

디바이스에서 업데이트하도록 요청된 상태 속성 및 값입니다.

유형: 키-값 페어의 `map`

`reported`

디바이스에서 보고한 상태 속성 및 값입니다.

유형: 키-값 페어의 `map`

`delta`

디바이스에서 보고한 상태 속성 및 값입니다.

유형: 키-값 페어의 `map`

metadata

언제 상태가 업데이트되었는지 확인할 수 있는 `desired` 및 `reported` 섹션의 각 속성에 대한 타임스탬프입니다.

유형: `string`

timestamp

응답이 생성된 epoch 날짜 및 시간입니다.

유형: `integer`

clientToken(Python: `client_token`)

요청과 해당 응답을 일치시키는 데 사용되는 토큰입니다.

유형: `string`

version

업데이트가 완료된 후 로컬 새도 문서의 버전입니다.

유형: `integer`

오류

이 작업은 다음 오류를 반환할 수 있습니다.

ConflictError

업데이트 작업 중에 로컬 새도 서비스에서 버전 충돌이 발생했습니다. 요청 페이로드의 버전이 사용 가능한 최신 로컬 새도 문서의 버전과 일치하지 않을 경우 발생합니다.

InvalidArgumentsError

로컬 새도 서비스에서 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON 또는 지원되지 않는 문자가 포함된 경우 발생할 수 있습니다.

유효한 `payload`의 속성은 다음과 같습니다.

- `state` 노드가 있고 `desired` 또는 `reported` 상태 정보를 포함하는 객체입니다.
- `desired` 및 `reported` 노드는 객체 또는 `null`입니다. 이러한 객체 중 하나 이상에 유효한 상태 정보가 포함되어 있어야 합니다.
- `desired` 및 `reported` 객체의 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 새도 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 파라미터에 지정된 제한을 초과했습니다.

UnauthorizedError

구성 요소의 권한 부여 정책에 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example예: 사물 새도 업데이트

Note

이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```
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)

Example예: 사물 새도 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

Example예: 사물 새도 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 getIpClient() {  
    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 새도를 삭제합니다.

새도 관리자 v2.0.4부터 새도를 삭제하면 버전 번호가 증가합니다. 예를 들어 버전 1에서 새도 MyThingShadow를 삭제하면 삭제된 새도의 버전은 2입니다. 그런 다음 이름이 MyThingShadow인 새도를 다시 생성하면 해당 새도의 버전은 3입니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

thingName(Python: thing_name)

사물의 이름입니다.

유형: string

shadowName(Python: shadow_name)

새도의 이름입니다. 사물의 클래식 새도를 지정하려면 이 파라미터를 빈 문자열("")로 설정합니다.

Warning

AWS IoT Greengrass 서비스는 AWSManagedGreengrassV2Deployment명명된 새도우를 사용하여 개별 코어 디바이스를 대상으로 하는 배포를 관리합니다. 이 명명된 새도우는 AWS IoT Greengrass 서비스에서 사용하도록 예약되어 있습니다. 이 이름이 지정된 새도를 업데이트하거나 삭제하지 마세요.

유형: string

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

payload

빈 응답 상태 문서입니다.

오류

이 작업은 다음 오류를 반환할 수 있습니다.

InvalidArgumentsError

로컬 새도 서비스에서 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON 또는 지원되지 않는 문자가 포함된 경우 발생할 수 있습니다.

ResourceNotFoundError

요청된 로컬 새도 문서를 찾을 수 없습니다.

ServiceError

내부 서비스 오류가 발생했거나, IPC 서비스에 대한 요청 수가 새도 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 파라미터에 지정된 제한을 초과했습니다.

UnauthorizedError

구성 요소의 권한 부여 정책에 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example예: 사물 새도 삭제

Note

이 예제에서는 `IPCUtils` 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```
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)

Example예: 사물 새도 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

Example예: 사물 새도 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

지정한 사물에 대한 이름 지정된 새도를 나열합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`thingName`(Python: `thing_name`)

사물의 이름입니다.

유형: `string`

`pageSize`(Python: `page_size`)

(선택 사항) 각 호출에서 반환할 새도 이름의 수입니다.

유형: `integer`

기본값: 25

최대: 100

`nextToken`(Python: `next_token`)

(선택 사항) 다음 결과 집합을 검색하기 위한 토큰입니다. 이 값은 페이지징된 결과에서 반환되며 다음 페이지를 반환하는 호출에 사용됩니다.

유형: `string`

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`results`

새도 이름 목록입니다.

유형: `array`

`timestamp`

(선택 사항) 응답이 생성된 날짜 및 시간입니다.

유형: `integer`

`nextToken`(Python: `next_token`)

(선택 사항) 시퀀스의 다음 페이지를 검색하기 위해 페이지징된 요청에 사용할 토큰 값입니다. 반환할 새도 이름이 더 이상 없는 경우에는 이 속성이 없습니다.

유형: `string`

 Note

요청된 페이지 크기가 응답의 새도 이름 수와 정확히 일치하면 이 토큰이 있지만, 사용 시 빈 목록을 반환합니다.

오류

이 작업은 다음 오류를 반환할 수 있습니다.

`InvalidArgumentsError`

로컬 새도 서비스에서 요청 파라미터를 검증할 수 없습니다. 요청에 잘못된 형식의 JSON 또는 지원되지 않는 문자가 포함된 경우 발생할 수 있습니다.

`ResourceNotFoundError`

요청된 로컬 새도 문서를 찾을 수 없습니다.

`ServiceError`

내부 서비스 오류가 발생했거나, IPC 서비스에 대한 요청 수가 새도 관리자 구성 요소의 `maxLocalRequestsPerSecondPerThing` 및 `maxTotalLocalRequestsRate` 구성 파라미터에 지정된 제한을 초과했습니다.

`UnauthorizedError`

구성 요소의 권한 부여 정책에 이 작업에 필요한 권한이 포함되어 있지 않습니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Java (IPC client V1)

Example예: 사물의 이름 지정된 새도 나열

Note

이 예제에서는 IPCUtils 클래스를 사용하여 AWS IoT Greengrass 코어 IPC 서비스에 대한 연결을 생성합니다. 자세한 내용은 [AWS IoT Greengrass 코어 IPC 서비스에 연결](#) 단원을 참조하십시오.

```
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)

Example예: 사물의 이름 지정된 새도 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

Example예: 사물의 이름 지정된 새도 나열

```

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 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다.

Greengrass CLI IPC 서비스를 사용하여 코어 디바이스에서 로컬 배포 및 Greengrass 구성 요소를 관리할 수 있습니다.

이러한 IPC 작업을 사용하려면 [Greengrass CLI 구성 요소](#) 버전 2.6.0 이상을 사용자 지정 구성 요소에 종속성으로 포함합니다. 그런 다음 사용자 지정 구성 요소에서 IPC 작업을 사용하여 다음을 수행할 수 있습니다.

- 로컬 배포를 생성하여 코어 디바이스에서 Greengrass 구성 요소를 수정하고 구성합니다.
- 코어 디바이스에서 Greengrass 구성 요소를 다시 시작하고 중지합니다.
- [로컬 디버그 콘솔](#)에 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 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.2.10	
AWS IoT Device SDK Python v2용	v1.5.3	
AWS IoT Device SDK C++ v2용	v1.17.0	
AWS IoT Device SDK JavaScript v2용	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 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

Greengrass CLI에 대한 권한 부여 정책에는 다음 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.Cli`

연산	설명	리소스
<code>aws.greengrass#CreateLocalDeployment</code>	구성 요소가 코어 디바이스에서 로컬 배포를 생성할 수 있도록 허용합니다.	*
<code>aws.greengrass#ListLocalDeployments</code>	구성 요소가 코어 디바이스의 로컬 배포를 나열할 수 있도록 허용합니다.	*

연산	설명	리소스
<code>aws.greengrass#GetLocalDeploymentStatus</code>	구성 요소가 코어 디바이스에서 로컬 배포 상태를 가져올 수 있도록 허용합니다.	로컬 배포 ID 또는 모든 로컬 배포에 대한 액세스를 허용하려면 *입니다.
<code>aws.greengrass#ListComponents</code>	구성 요소가 코어 디바이스의 구성 요소를 나열할 수 있도록 허용합니다.	*
<code>aws.greengrass#GetComponentDetails</code>	구성 요소가 코어 디바이스의 구성 요소에 대한 세부 정보를 가져올 수 있도록 허용합니다.	구성 요소 이름(예: <code>com.example.HelloWorld</code>) 또는 모든 구성 요소에 대한 액세스를 허용하려면 *입니다.
<code>aws.greengrass#RestartComponent</code>	구성 요소가 코어 디바이스에서 구성 요소를 다시 시작할 수 있도록 허용합니다.	구성 요소 이름(예: <code>com.example.HelloWorld</code>) 또는 모든 구성 요소에 대한 액세스를 허용하려면 *입니다.
<code>aws.greengrass#StopComponent</code>	구성 요소가 코어 디바이스에서 구성 요소를 중지할 수 있도록 허용합니다.	구성 요소 이름(예: <code>com.example.HelloWorld</code>) 또는 모든 구성 요소에 대한 액세스를 허용하려면 *입니다.
<code>aws.greengrass#CreateDebugPassword</code>	구성 요소가 로컬 디버그 콘솔 구성 요소 에 로그인하는 데 사용할 암호를 생성할 수 있도록 허용합니다.	*

Example권한 부여 정책 예제

다음 권한 부여 정책 예제에서는 구성 요소가 로컬 배포를 생성하고, 모든 로컬 배포 및 구성 요소를 보고, `com.example.HelloWorld`라는 구성 요소를 다시 시작하고 중지할 수 있도록 허용합니다.

```
{
  "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 생성하거나 업데이트합니다.

이 작업은 Greengrass CLI의 [deployment create 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`recipeDirectoryPath`(Python: `recipe_directory_path`)

(선택 사항) 구성 요소 레시피 파일이 포함된 폴더의 절대 경로입니다.

`artifactDirectoryPath`(Python: `artifact_directory_path`)

(선택 사항) 배포에 포함할 아티팩트 파일이 포함된 폴더의 절대 경로입니다. 아티팩트 폴더에는 다음 폴더 구조를 포함해야 합니다.

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd`(Python: `root_component_versions_to_add`)

(선택 사항) 코어 디바이스에 설치할 구성 요소 버전입니다. 이 객체 `ComponentToVersionMap`는 다음 키-값 페어를 포함하는 맵입니다.

`key`

구성 요소의 이름입니다.

`value`

구성 요소의 버전입니다.

`rootComponentsToRemove`(Python: `root_components_to_remove`)

(선택 사항) 코어 디바이스에서 제거할 구성 요소입니다. 각 항목이 구성 요소의 이름인 목록을 지정합니다.

`componentToConfiguration`(Python: `component_to_configuration`)

(선택 사항) 배포의 각 구성 요소에 대한 구성 업데이트입니다. 이 객체 `ComponentToConfiguration`는 다음 키-값 페어를 포함하는 맵입니다.

`key`

구성 요소의 이름입니다.

`value`

구성 요소에 대한 구성 업데이트 JSON 객체입니다. JSON 객체는 다음 형식이어야 합니다.

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

구성 업데이트에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

`componentToRunWithInfo(Python: component_to_run_with_info)`

(선택 사항) 배포의 각 구성 요소에 대한 런타임 구성입니다. 이 구성에는 각 구성 요소의 프로세스를 소유하는 시스템 사용자와 각 구성 요소에 적용할 시스템 제한이 포함됩니다. 이 객체 `ComponentToRunWithInfo`는 다음 키-값 페어를 포함하는 맵입니다.

`key`

구성 요소의 이름입니다.

`value`

구성 요소의 런타임 구성입니다. 런타임 구성 파라미터를 생략하면 AWS IoT Greengrass 코어 소프트웨어는 [Greengrass nucleus](#)에서 구성하는 기본값을 사용합니다. 이 객체 `RunWithInfo`에는 다음 정보가 포함됩니다.

`posixUser(Python: posix_user)`

(선택 사항) Linux 코어 디바이스에서 이 구성 요소를 실행하는 데 사용할 POSIX 시스템 사용자와 그룹(선택 사항)입니다. 사용자와 그룹(지정된 경우)이 각 Linux 코어 디바이스에 있어야 합니다. `user:group` 형식으로 사용자와 그룹을 콜론(:)으로 구분하여 지정합니다. 그룹은 선택 사항입니다. 그룹을 지정하지 않으면 AWS IoT Greengrass 코어 소프트웨어는 사용자의 기본 그룹을 사용합니다. 자세한 내용은 단원을 참조하십시오 [구성 요소를 실행하는 사용자 구성](#).

`windowsUser(Python: windows_user)`

(선택 사항) Windows 코어 디바이스에서 이 구성 요소를 실행하는 데 사용할 Windows 사용자입니다. 사용자가 각 Windows 코어 디바이스에 있어야 하며, 해당 이름과 암호가 LocalSystem 계정의 Credentials Manager 인스턴스에 저장되어야 합니다. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 섹션을 참조하세요.

systemResourceLimits(Python: system_resource_limits)

(선택 사항) 이 구성 요소의 프로세스에 적용할 시스템 리소스 제한입니다. 기본적으로 일반 및 컨테이너화되지 않은 Lambda 구성 요소에 시스템 리소스 제한을 적용할 수 있습니다. 자세한 내용은 단원을 참조하십시오 [구성 요소에 대한 시스템 리소스 제한 구성](#).

AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

이 객체 SystemResourceLimits에는 다음 정보가 포함됩니다.

cpus

(선택 사항) 이 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 CPU 시간입니다. 코어 디바이스의 총 CPU 시간은 디바이스의 CPU 코어 수와 같습니다. 예를 들어 CPU 코어가 4개인 코어 디바이스에서 이 값을 2로 설정하여 이 구성 요소 프로세스를 각 CPU 코어의 50% 사용량으로 제한할 수 있습니다. CPU 코어가 1개인 디바이스에서 이 값을 0.25로 설정하여 이 구성 요소 프로세스를 CPU의 25% 사용량으로 제한할 수 있습니다. 이 값을 CPU 코어 수보다 큰 수로 설정하면 AWS IoT Greengrass 코어 소프트웨어가 구성 요소의 CPU 사용량을 제한하지 않습니다.

memory

(선택 사항) 이 구성 요소의 프로세스가 코어 디바이스에서 사용할 수 있는 최대 RAM(KB 단위)입니다.

groupName(Python: group_name)

(선택 사항) 이 배포에서 대상으로 지정할 사물 그룹의 이름입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

deploymentId(Python: deployment_id)

요청이 생성한 로컬 배포의 ID입니다.

ListLocalDeployments

마지막 10개 로컬 배포의 상태를 가져옵니다.

이 작업은 Greengrass CLI의 [deployment list 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`localDeployments(Python: local_deployments)`

로컬 배포 목록입니다. 이 목록의 각 객체는 `LocalDeployment` 객체이며, 여기에는 다음 정보가 포함됩니다.

`deploymentId(Python: deployment_id)`

로컬 배포의 ID입니다.

`status`

로컬 배포의 상태입니다. 이 열거형 `DeploymentStatus`의 값은 다음과 같습니다.

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED

GetLocalDeploymentStatus

로컬 배포의 상태를 가져옵니다.

이 작업은 Greengrass CLI의 [deployment status 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`deploymentId(Python: deployment_id)`

가져올 로컬 배포의 ID입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

deployment

로컬 배포입니다. 이 객체 LocalDeployment에는 다음 정보가 포함됩니다.

deploymentId(Python: deployment_id)

로컬 배포의 ID입니다.

status

로컬 배포의 상태입니다. 이 열거형 DeploymentStatus의 값은 다음과 같습니다.

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED

ListComponents

코어 디바이스에서 각 루트 구성 요소의 이름, 버전, 상태 및 구성을 가져옵니다. 루트 구성 요소는 배포에서 지정하는 구성 요소입니다. 이 응답에는 다른 구성 요소의 종속성으로 설치된 구성 요소는 포함되지 않습니다.

이 작업은 Greengrass CLI의 [component list 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

components

코어 디바이스의 루트 구성 요소 목록입니다. 이 목록의 각 객체는 ComponentDetails 객체이며, 여기에는 다음 정보가 포함됩니다.

componentName(Python: component_name)

구성 요소의 이름입니다.

version

구성 요소의 버전입니다.

state

구성 요소의 상태입니다. 이 상태는 다음 중 하나일 수 있습니다.

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

configuration

JSON 객체인 구성 요소의 구성입니다.

GetComponentDetails

코어 디바이스에서 구성 요소의 버전, 상태 및 구성을 가져옵니다.

이 작업은 Greengrass CLI의 [component details 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

componentName(Python: component_name)

가져올 구성 요소의 이름입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

componentDetails(Python: component_details)

구성 요소의 세부 정보입니다. 이 객체 ComponentDetails에는 다음 정보가 포함됩니다.

componentName(Python: component_name)

구성 요소의 이름입니다.

version

구성 요소의 버전입니다.

state

구성 요소의 상태입니다. 이 상태는 다음 중 하나일 수 있습니다.

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

configuration

JSON 객체인 구성 요소의 구성입니다.

RestartComponent

코어 디바이스에서 구성 요소를 다시 시작합니다.

Note

어느 구성 요소든 다시 시작할 수 있지만 [일반 구성 요소](#)만 다시 시작하는 것이 좋습니다.

이 작업은 Greengrass CLI의 [component restart 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`componentName`(Python: `component_name`)

구성 요소의 이름입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`restartStatus`(Python: `restart_status`)

다시 시작 요청의 상태입니다. 요청 상태는 다음 중 하나일 수 있습니다.

- SUCCEEDED
- FAILED

`message`

요청이 실패한 경우 구성 요소를 다시 시작하지 못한 이유에 대한 메시지입니다.

예제

다음 예제에서는 사용자 지정 구성 요소 코드에서 이 작업을 직접 호출하는 방법을 보여줍니다.

Rust

Example예: 구성 요소 다시 시작

```
use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let component_name = "com.example.HelloWorld";

    sdk.restart_component(component_name)
        .expect("Failed to restart component");
}
```

```
    println!("Successfully requested restart for component: {component_name}");  
}
```

C

Example예: 구성 요소 다시 시작

```
#include <gg/error.h>  
#include <gg/ipc/client.h>  
#include <gg/sdk.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void) {  
    gg_sdk_init();  
  
    GgError err = ggipc_connect();  
    if (err != GG_ERR_OK) {  
        fprintf(stderr, "Failed to establish IPC connection.\n");  
        exit(-1);  
    }  
  
    GgBuffer component_name = GG_STR("com.example.HelloWorld");  
  
    err = ggipc_restart_component(component_name);  
    if (err != GG_ERR_OK) {  
        fprintf(  
            stderr,  
            "Failed to restart component: %.*s\n",  
            (int) component_name.len,  
            component_name.data  
        );  
        exit(-1);  
    }  
  
    printf(  
        "Successfully requested restart for component: %.*s\n",  
        (int) component_name.len,  
        component_name.data  
    );  
}
```

C++ (Component SDK)

Example예: 구성 요소 다시 시작

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view component_name = "com.example.HelloWorld";

    error = client.restart_component(component_name);
    if (error) {
        std::cerr << "Failed to restart component: " << component_name << "\n";
        exit(-1);
    }

    std::cout << "Successfully requested restart for component: "
              << component_name << "\n";
}
```

StopComponent

코어 디바이스에서 구성 요소의 프로세스를 중지합니다.

Note

어느 구성 요소든 중지할 수 있지만 [일반 구성 요소](#)만 중지하는 것이 좋습니다.

이 작업은 Greengrass CLI의 [component stop 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

componentName(Python: component_name)

구성 요소의 이름입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

stopStatus(Python: stop_status)

중지 요청의 상태입니다. 요청 상태는 다음 중 하나일 수 있습니다.

- SUCCEEDED
- FAILED

message

요청이 실패한 경우 구성 요소를 중지하지 못한 이유에 대한 메시지입니다.

CreateDebugPassword

[로컬 디버그 콘솔 구성 요소](#)에 로그인하는 데 사용할 수 있는 임의 암호를 생성합니다. 암호는 생성되고 8시간 후에 만료됩니다.

이 작업은 Greengrass CLI의 [get-debug-password 명령](#)과 동일한 기능을 제공합니다.

요청

이 작업의 요청에는 파라미터가 없습니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

username

로그인에 사용할 사용자 이름입니다.

password

로그인에 사용할 암호입니다.

passwordExpiration(Python: password_expiration)

암호가 만료되는 시간입니다.

certificateSHA256Hash(Python: certificate_sha256_hash)

HTTPS가 활성화된 경우 로컬 디버그 콘솔에서 사용하는 자체 서명된 인증서의 SHA-256 지문입니다. 로컬 디버그 콘솔을 열 때 이 지문을 사용하여 인증서가 합법적이고 연결이 안전한지 확인합니다.

certificateSHA1Hash(Python: certificate_sha1_hash)

HTTPS가 활성화된 경우 로컬 디버그 콘솔에서 사용하는 자체 서명된 인증서의 SHA-1 지문입니다. 로컬 디버그 콘솔을 열 때 이 지문을 사용하여 인증서가 합법적이고 연결이 안전한지 확인합니다.

클라이언트 디바이스 인증 및 권한 부여

Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.6.0 이상에서 사용할 수 있습니다.

클라이언트 디바이스 인증 IPC 서비스를 사용하여 클라이언트 디바이스와 같은 로컬 IoT 디바이스가 연결할 수 있는 사용자 지정 로컬 브로커 구성 요소를 개발할 수 있습니다.

이러한 IPC 작업을 사용하려면 사용자 지정 구성 요소에서 [클라이언트 디바이스 인증 구성 요소](#) 버전 2.2.0 이상을 종속성으로 포함합니다. 그런 다음 사용자 지정 구성 요소에서 IPC 작업을 사용하여 다음을 수행할 수 있습니다.

- 코어 디바이스에 연결하는 클라이언트 디바이스의 ID를 확인합니다.
- 클라이언트 디바이스가 코어 디바이스에 연결할 세션을 생성합니다.
- 클라이언트 디바이스에 작업을 수행할 권한이 있는지 확인합니다.
- 코어 디바이스의 서버 인증서가 교체되면 알림을 받습니다.

주제

- [최소 SDK 버전](#)
- [권한 부여](#)
- [VerifyClientDeviceIdentity](#)
- [GetClientDeviceAuthToken](#)

- [AuthorizeClientDeviceAction](#)
- [SubscribeToCertificateUpdates](#)

최소 SDK 버전

다음 표에는 클라이언트 디바이스 인증 IPC 서비스와 상호 작용하는 데 사용해야 AWS IoT Device SDK 하는의 최소 버전이 나열되어 있습니다.

SDK	최소 버전	
AWS IoT Device SDK Java v2용	v1.9.3	
AWS IoT Device SDK Python v2용	v1.11.3	
AWS IoT Device SDK C++ v2용	v1.18.3	
AWS IoT Device SDK JavaScript v2용	v1.12.0	

권한 부여

사용자 지정 구성 요소에서 클라이언트 디바이스 인증 IPC 서비스를 사용하려면 구성 요소가 이러한 작업을 수행하도록 허용하는 권한 부여 정책을 정의해야 합니다. 권한 부여 정책 정의에 대한 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 섹션을 참조하세요.

클라이언트 디바이스 인증 및 권한 부여에 대한 권한 부여 정책에는 다음 속성이 있습니다.

IPC 서비스 식별자: `aws.greengrass.clientdevices.Auth`

연산	설명	리소스
<code>aws.greengrass#VerifyClientDeviceIdentity</code>	구성 요소가 클라이언트 디바이스의 ID를 확인할 수 있도록 합니다.	*

연산	설명	리소스
<code>aws.greengrass#GetClientDeviceAuthToken</code>	구성 요소가 클라이언트 디바이스의 자격 증명을 검증하고 해당 클라이언트 디바이스에 대한 세션을 생성할 수 있도록 합니다.	*
<code>aws.greengrass#AuthorizeClientDeviceAction</code>	구성 요소가 클라이언트 디바이스에 작업을 수행할 수 있는 권한이 있는지 확인할 수 있도록 합니다.	*
<code>aws.greengrass#SubscribeToCertificateUpdates</code>	코어 디바이스의 서버 인증서가 교체될 때 구성 요소가 알림을 받을 수 있도록 합니다.	*
*	구성 요소가 모든 클라이언트 디바이스 인증 IPC 서비스 작업을 수행할 수 있도록 합니다.	*

권한 부여 정책 예제

다음 권한 부여 정책 예제를 참조하면 구성 요소의 권한 부여 정책을 구성하는 데 도움이 됩니다.

Example 권한 부여 정책 예제

다음 예제 권한 부여 정책에서는 구성 요소가 모든 클라이언트 디바이스 인증 IPC 작업을 수행할 수 있도록 합니다.

```
{
  "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 디바이스가 유효한 AWS IoT 사물인지 확인합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

credential

클라이언트 디바이스의 자격 증명입니다. 이 객체 ClientDeviceCredential에는 다음 정보가 포함됩니다.

clientDeviceCertificate(Python: client_device_certificate)

클라이언트 디바이스의 X.509 디바이스 인증서입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

isValidClientDevice(Python: is_valid_client_device)

클라이언트 디바이스의 ID가 유효한지 여부입니다.

GetClientDeviceAuthToken

클라이언트 디바이스의 자격 증명을 검증하고 클라이언트 디바이스에 대한 세션을 생성합니다. 이 작업은 이후 [클라이언트 디바이스 작업에 권한을 부여](#)하는 요청에서 사용할 수 있는 세션 토큰을 반환합니다.

클라이언트 디바이스를 연결하려면 [클라이언트 디바이스 인증 구성 요소](#)에서 클라이언트 디바이스가 사용하는 클라이언트 ID에 대한 `mqtt:connect` 권한을 부여해야 합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`credential`

클라이언트 디바이스의 자격 증명입니다. 이 객체 `CredentialDocument`에는 다음 정보가 포함됩니다.

`mqttCredential(Python: mqtt_credential)`

클라이언트 디바이스의 MQTT 자격 증명입니다. 클라이언트 디바이스가 연결에 사용하는 클라이언트 ID 및 인증서를 지정합니다. 이 객체 `MQTTCredential`에는 다음 정보가 포함됩니다.

`clientId(Python: client_id)`

연결하는 데 사용할 Client ID입니다.

`certificatePem(Python: certificate_pem)`

연결하는 데 사용할 X.509 디바이스 인증서입니다.

`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 토큰입니다. 이후 이 클라이언트 디바이스의 작업에 권한을 부여하는 요청에서 이 세션 토큰을 사용할 수 있습니다.

AuthorizeClientDeviceAction

클라이언트 디바이스가 리소스에서 작업을 수행할 권한이 있는지 확인합니다. 클라이언트 디바이스 권한 부여 정책은 클라이언트 디바이스가 코어 디바이스에 연결되는 동안 수행할 수 있는 권한을 지정합니다. [클라이언트 디바이스 인증 구성 요소](#)를 구성할 때 클라이언트 디바이스 권한 부여 정책을 정의합니다.

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`clientDeviceAuthToken(Python: client_device_auth_token)`

클라이언트 디바이스의 세션 토큰입니다.

`operation`

권한을 부여할 작업입니다.

`resource`

클라이언트 디바이스가 작업을 수행하는 리소스입니다.

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`isAuthorized(Python: is_authorized)`

클라이언트 디바이스가 리소스에서 작업을 수행할 수 있는 권한이 있는지 여부입니다.

SubscribeToCertificateUpdates

코어 디바이스가 교체될 때마다 코어 디바이스의 새 서버 인증서를 수신하도록 구독합니다. 서버 인증서가 교체되면 새 서버 인증서를 사용하여 브로커를 다시 로드해야 합니다.

[클라이언트 디바이스 인증 구성 요소](#)는 기본적으로 7일마다 서버 인증서를 교체합니다. 교체 간격을 2~10일로 구성할 수 있습니다.

이 작업은 이벤트 메시지 스트림을 구독하는 구독 작업입니다. 이 작업을 사용하려면 이벤트 메시지, 오류 및 스트림 종료를 처리하는 함수를 사용하여 스트림 응답 핸들러를 정의합니다. 자세한 내용은 [IPC 이벤트 스트림 구독](#) 단원을 참조하십시오.

이벤트 메시지 유형: `CertificateUpdateEvent`

요청

이 작업의 요청에서는 다음 파라미터를 사용합니다.

`certificateOptions`(Python: `certificate_options`)

구독할 인증서 업데이트의 유형입니다. 이 객체 `CertificateOptions`에는 다음 정보가 포함됩니다.

`certificateType`(Python: `certificate_type`)

구독할 인증서 업데이트의 유형입니다. 다음 옵션을 선택합니다.

- `SERVER`

응답

이 작업의 응답에는 다음 정보가 포함됩니다.

`messages`

메시지 스트림입니다. 이 객체 `CertificateUpdateEvent`에는 다음 정보가 포함됩니다.

`certificateUpdate`(Python: `certificate_update`)

새 인증서에 대한 정보입니다. 이 객체 `CertificateUpdate`에는 다음 정보가 포함됩니다.

`certificate`

인증서입니다.

`privateKey`(Python: `private_key`)

인증서의 프라이빗 키입니다.

`publicKey`(Python: `public_key`)

인증서의 퍼블릭 키입니다.

`caCertificates`(Python: `ca_certificates`)

인증서의 CA 인증서 체인에 있는 CA(인증 기관) 인증서 목록입니다.

로컬 IoT 디바이스와 상호 작용

클라이언트 디바이스는 MQTT를 통해 Greengrass 코어 디바이스에 연결하고 통신하는 로컬 IoT 디바이스입니다. 클라이언트 디바이스를 코어 디바이스에 연결하여 다음을 수행할 수 있습니다.

- Greengrass 구성 요소의 MQTT 메시지와 연동합니다.
- 클라이언트 디바이스와 AWS IoT Core 간에 메시지와 데이터를 릴레이합니다.
- Greengrass 구성 요소의 클라이언트 디바이스 새도와 상호 작용합니다.
- 클라이언트 디바이스 새도를 AWS IoT Core와 동기화합니다.
- 로컬 메시징에 IPv6를 사용합니다.

코어 디바이스에 연결하기 위해 클라이언트 디바이스는 클라우드 검색을 사용할 수 있습니다. 클라이언트 디바이스는 AWS IoT Greengrass 클라우드 서비스에 연결하여 연결할 수 있는 코어 디바이스에 대한 정보를 검색합니다. 그런 다음 코어 디바이스에 연결하여 메시지를 처리하고 데이터를 AWS IoT Core 클라우드 서비스와 동기화할 수 있습니다.

코어 디바이스를 구성하여 AWS IoT 사물과 연결하고 통신하는 방법을 안내하는 자습서를 따를 수 있습니다. 이 자습서에서는 클라이언트 디바이스와 상호 작용하는 사용자 지정 Greengrass 구성 요소를 개발하는 방법도 살펴봅니다. 자세한 내용은 [자습서: MQTT를 통해 로컬 IoT 디바이스와 상호 작용](#) 섹션을 참조하세요.

주제

- [AWS에서 제공하는 클라이언트 디바이스 구성 요소](#)
- [코어 디바이스에 클라이언트 디바이스 연결](#)
- [클라이언트 디바이스와 간에 MQTT 메시지 릴레이 AWS IoT Core](#)
- [구성 요소의 클라이언트 디바이스와 상호 작용](#)
- [클라이언트 디바이스 새도와 상호 작용 및 동기화](#)
- [로컬 메시징에 IPv6 사용](#)
- [클라이언트 디바이스 문제 해결](#)

AWS에서 제공하는 클라이언트 디바이스 구성 요소

AWS IoT Greengrass에서는 코어 디바이스에 배포할 수 있는 다음 퍼블릭 구성 요소를 제공합니다. 이러한 구성 요소를 통해 클라이언트 디바이스에서 코드 디바이스와 연결하여 통신할 수 있습니다.

Note

AWS에서 제공하는 몇 가지 구성 요소는 Greengrass nucleus의 특정 마이너 버전에 따라 다릅니다. 이 종속성 때문에 Greengrass nucleus를 새로운 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 달라지는 특정 nucleus 버전에 대한 내용은 해당 구성 요소 주제를 참조하세요. nucleus 업데이트에 대한 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#) 단원을 참조하세요.

구성 요소에 일반 및 Lambda 구성 요소 유형이 둘 다 있으면 구성 요소의 현재 버전은 일반 유형이고 구성 요소의 이전 버전은 Lambda 유형입니다.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
클라이언트 디바이스 인증	클라이언트 디바이스라는 로컬 IoT 디바이스를 활성화하여 코어 디바이스에 연결합니다.	플러그인	Linux, Windows	예
IP 감지기	클라이언트 디바이스가 연결 방법을 검색할 수 있도록 MQTT 연결 정보를 AWS IoT Greengrass에 보고합니다.	플러그인	Linux, Windows	예
MQTT 브리지	클라이언트 디바이스, 로컬 AWS IoT Greengrass 게시/구독 및	플러그인	Linux, Windows	예

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
	AWS IoT Core 간 MQTT 메시 지를 릴레이합 니다.			
MQTT 3.1.1 브로커(Moquette)	클라이언트 디 바이스와 코어 디바이스 간 메 시지를 처리하 는 MQTT 3.1.1 브로커를 실행 합니다.	플러그인	Linux, Windows	예
MQTT 5 브로커(EMQX)	클라이언트 디 바이스와 코어 디바이스 간 메 시지를 처리하 는 MQTT 5 브 로커를 실행합 니다.	일반	Linux, Windows	아니요
샤도 관리자	코어 디바이스 에서 샤도와 상 호 작용을 활성 화합니다. 샤도 문서 스토리지 는 물론 AWS IoT 디바이스 샤도 서비스와 로컬 샤도 상태 의 동기화도 관 리합니다.	플러그인	Linux, Windows	예

코어 디바이스에 클라이언트 디바이스 연결

코어 디바이스에 클라이언트 디바이스가 연결되도록 클라우드 검색을 구성할 수 있습니다. 클라우드 검색을 구성하면 클라이언트 디바이스가 AWS IoT Greengrass 클라우드 서비스에 연결하여 연결할 수 있는 코어 디바이스에 대한 정보를 검색할 수 있습니다. 그러면 클라이언트 디바이스에서는 연결될 때까지 각 코어 디바이스에 대한 연결이 시도될 수 있습니다.

클라우드 검색을 사용하려면 다음을 수행해야 합니다.

- 연결될 수 있는 코어 디바이스에 클라이언트 디바이스를 연결합니다.
- 클라이언트 디바이스가 각 코어 디바이스에 연결될 수 있는 MQTT 브로커 엔드포인트를 지정합니다.
- 클라이언트 디바이스에 대한 지원이 활성화되는 구성 요소를 코어 디바이스에 배포합니다.

선택적 구성 요소를 배포하여 다음을 수행할 수도 있습니다.

- 클라이언트 디바이스, Greengrass 구성 요소 및 AWS IoT Core 클라우드 서비스 간에 메시지를 릴레이합니다.
- 코어 디바이스 MQTT 브로커 엔드포인트를 자동으로 관리합니다.
- 로컬 클라이언트 디바이스 새도우를 관리하고 새도우를 AWS IoT Core 클라우드 서비스와 동기화합니다.

또한 코어 디바이스의 AWS IoT 정책을 검토하고 업데이트하여 클라이언트 디바이스를 연결하는 데 필요한 권한이 있는지 확인해야 합니다. 자세한 내용은 [요구 사항](#) 단원을 참조하십시오.

클라우드 검색을 구성하면 클라이언트 디바이스와 코어 디바이스 간 통신을 테스트할 수 있습니다. 자세한 내용은 [클라이언트 디바이스 통신 테스트](#) 단원을 참조하십시오.

주제

- [요구 사항](#)
- [클라이언트 디바이스 지원용 Greengrass 구성 요소](#)
- [클라우드 검색 구성\(콘솔\)](#)
- [클라우드 검색 구성\(AWS CLI\)](#)
- [클라이언트 디바이스 연결](#)
- [오프라인 상태에서 클라이언트 인증](#)
- [코어 디바이스 엔드포인트 관리](#)

- [MQTT 브로커 선택](#)
- [MQTT 브로커를 사용하여 AWS IoT Greengrass 코어 디바이스에 클라이언트 디바이스 연결](#)
- [클라이언트 디바이스 통신 테스트](#)
- [Greengrass Discovery RESTful API](#)

요구 사항

코어 디바이스에 클라이언트 디바이스를 연결하려면 다음이 필요합니다.

- 코어 디바이스에서 [Greengrass nucleus](#) v2.2.0 이상이 실행되어야 합니다.
- 코어 디바이스가 작동하는 AWS 리전 AWS 계정 의에 AWS IoT Greengrass 대해와 연결된 Greengrass 서비스 역할입니다. 자세한 내용은 [Greengrass 서비스 역할 구성](#) 단원을 참조하십시오.
- 코어 디바이스의 AWS IoT 정책은 다음 권한을 허용해야 합니다.
 - greengrass:PutCertificateAuthorities
 - greengrass:VerifyClientDeviceIdentity
 - greengrass:VerifyClientDeviceIoTCertificateAssociation
 - greengrass:GetConnectivityInfo
 - greengrass:UpdateConnectivityInfo – (선택 사항)이 권한은 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 [IP 감지기 구성 요소](#)를 사용하는 데 필요합니다.
 - iot:GetThingShadow, iot:UpdateThingShadow 및 iot:DeleteThingShadow – (선택 사항) 이러한 권한은 [새도 관리자 구성 요소](#)를 사용하여 클라이언트 디바이스 새도와 동기화하는 데 필요합니다 AWS IoT Core. 이 특성에는 [Greengrass nucleus](#) v2.6.0 이상, 새도 관리자 v2.2.0 이상 및 [MQTT 브리지](#) v2.2.0 이상이 필요합니다.

자세한 내용은 [AWS IoT 사물 정책 구성](#) 단원을 참조하십시오.

Note

[AWS IoT Greengrass 코어 소프트웨어를 설치할](#) 때 기본 AWS IoT 정책을 사용한 경우 코어 디바이스에는 모든 AWS IoT Greengrass 작업()에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다 greengrass:*

- AWS IoT 클라이언트 디바이스로 연결할 수 있는 사물입니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 리소스 생성](#)을 참조하세요.

- 클라이언트 ID를 사용하여 클라이언트 디바이스를 연결해야 합니다. 클라이언트 ID는 사물 이름입니다. 기타 클라이언트 ID는 수락되지 않습니다.
- 각 클라이언트 디바이스의 AWS IoT 정책은 `greengrass:Discover` 권한을 허용해야 합니다. 자세한 내용은 [클라이언트 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.

주제

- [Greengrass 서비스 역할 구성](#)
- [AWS IoT 사물 정책 구성](#)

Greengrass 서비스 역할 구성

Greengrass 서비스 역할은가 사용자를 대신하여 서비스의 리소스에 액세스할 AWS IoT Greengrass 수 있도록 권한을 부여하는 AWS Identity and Access Management (IAM) AWS 서비스 역할입니다. 이 역할을 사용하면 AWS IoT Greengrass 가 클라이언트 디바이스의 ID를 확인하고 코어 디바이스 연결 정보를 관리할 수 있습니다.

이전에이 리전에서 [Greengrass 서비스 역할을](#) 설정하지 않은 경우이 리전 AWS 계정 의에 AWS IoT Greengrass 대해 Greengrass 서비스 역할을와 연결해야 합니다.

[AWS IoT Greengrass 콘솔](#)에서 코어 디바이스 검색 구성 페이지를 사용하면 AWS IoT Greengrass 가 Greengrass 서비스 역할을 설정합니다. 그렇지 않으면 [AWS IoT 콘솔](#) 또는 AWS IoT Greengrass API 를 사용하여 수동으로 설정할 수 있습니다.

이 섹션에서는 Greengrass 서비스 역할이 설정되었는지 확인합니다. 설정되지 않은 경우이 리전 AWS 계정 의에 AWS IoT Greengrass 대해와 연결할 새 Greengrass 서비스 역할을 생성합니다.

Greengrass 서비스 역할 구성(콘솔)

1. Greengrass 서비스 역할이이 리전의 AWS 계정 에 AWS IoT Greengrass 대해와 연결되어 있는지 확인합니다. 해결 방법:
 - a. [AWS IoT 콘솔](#)로 이동합니다.
 - b. 탐색 창에서 설정을 선택합니다.
 - c. Greengrass 서비스 역할 섹션에서 현재 서비스 역할을 찾아 Greengrass 서비스 역할이 연결되었는지 참조합니다.

연결된 Greengrass 서비스 역할이 있으면 IP 감지기 구성 요소를 사용할 이 요구 사항을 충족하는 것입니다. [AWS IoT 사물 정책 구성](#)로 이동하세요.

2. Greengrass 서비스 역할이이 리전의 AWS 계정 에 AWS IoT Greengrass 대해와 연결되지 않은 경우 Greengrass 서비스 역할을 생성하고 연결합니다. 해결 방법:
 - a. [IAM 콘솔](#)로 이동합니다.
 - b. [Roles]를 선택합니다.
 - c. 역할 생성을 선택합니다.
 - d. 역할 생성 페이지에서 다음을 수행합니다.
 - i. 신뢰할 수 있는 엔터티 유형에서 AWS 서비스를 선택합니다.
 - ii. 사용 사례, 기타 AWS 서비스사용 사례에서 Greengrass를 선택하고, Greengrass를 선택합니다. 이 옵션은롤이 역할을 수임할 수 있는 신뢰할 수 있는 엔 AWS IoT Greengrass 터티로 추가하도록 지정합니다.
 - iii. 다음을 선택합니다.
 - iv. 권한 정책에서 역할에 연결할 AWSGreengrassResourceAccessRolePolicy를 선택합니다.
 - v. 다음을 선택합니다.
 - vi. 역할 이름에 역할의 이름(예: **Greengrass_ServiceRole**)을 입력합니다.
 - vii. 역할 생성을 선택합니다.
 - e. [AWS IoT 콘솔](#)로 이동합니다.
 - f. 탐색 창에서 설정을 선택합니다.
 - g. Greengrass 서비스 역할 섹션에서 역할 연결을 선택합니다.
 - h. Greengrass 서비스 역할 업데이트 모달에서, 생성한 IAM 역할을 선택한 다음에 역할 연결을 선택합니다.

Greengrass 서비스 역할 구성(AWS CLI)

1. Greengrass 서비스 역할이이 리전의 AWS 계정 에 AWS IoT Greengrass 대해와 연결되어 있는지 확인합니다.

```
aws greengrassv2 get-service-role-for-account
```

Greengrass 서비스 역할이 연결되어 있으면 역할에 대한 정보가 있는 응답이 작업에서 반환됩니다.

연결된 Greengrass 서비스 역할이 있으면 IP 감지기 구성 요소를 사용할 이 요구 사항을 충족하는 것입니다. [AWS IoT 사물 정책 구성](#)로 이동하세요.

2. Greengrass 서비스 역할이이 리전의 AWS 계정 에 AWS IoT Greengrass 대해와 연결되지 않은 경우 Greengrass 서비스 역할을 생성하고 연결합니다. 해결 방법:
 - a. 가 역할을 AWS IoT Greengrass 수임하도록 허용하는 신뢰 정책을 사용하여 역할을 생성합니다. 이 예제에서는 Greengrass_ServiceRole라는 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 대리인 방지](#)를 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 역할의 AWS IoT Greengrass 와 연결합니다 AWS 계정. 서비스 역할의 ARN으로 *role-arn*을 바꿉니다.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

작업에 성공하면 다음 응답이 반환됩니다.

```
{
  "associatedAt": "timestamp"
}
```

AWS IoT 사물 정책 구성

코어 디바이스에서는 AWS에 대한 연결 권한 부여에 X.509 디바이스 인증서가 사용됩니다. 디바이스 인증서에 AWS IoT 정책을 연결하여 코어 디바이스에 대한 권한을 정의합니다. 자세한 내용은 [AWS IoT 데이터 영역 작업에 대한 정책](#) 및 [클라이언트 디바이스를 지원하는 최소 AWS IoT 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)이 권한은 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 [IP 감지기 구성 요소를](#) 사용하는 데 필요합니다.
- `iot:GetThingShadow`, `iot:UpdateThingShadow` 및 `iot:DeleteThingShadow` – (선택 사항) 이러한 권한은 [새도 관리자 구성 요소를](#) 사용하여 클라이언트 디바이스 새도와 동기화하는 데 필요합니다 AWS IoT Core. 이 특성에는 [Greengrass nucleus](#) v2.6.0 이상, 새도 관리자 v2.2.0 이상 및 [MQTT 브리지](#) v2.2.0 이상이 필요합니다.

이 섹션에서는 코어 디바이스에 대한 AWS IoT 정책을 검토하고 누락된 필수 권한을 추가합니다. [AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 코어

디바이스에는 모든 AWS IoT Greengrass 작업()에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다. `greengrass:*`. 이 경우 디바이스 새도를 동기화하기 위해 새도 관리자 구성 요소를 배포하려는 경우에만 AWS IoT 정책을 업데이트해야 합니다. AWS IoT Core. 그렇지 않으면 이 섹션을 건너뛰어도 됩니다.

AWS IoT 사물 정책 구성(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 코어 디바이스를 선택합니다.
2. 업데이트할 코어 디바이스를 코어 디바이스 페이지에서 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 코어 디바이스의 사물에 대한 링크를 선택합니다. 이 링크를 통해 사물 세부 정보 페이지가 AWS IoT 콘솔에서 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 검토하고 업데이트할 AWS IoT 정책을 선택합니다. 코어 디바이스의 활성 인증서에 연결된 원하는 정책에 필수 권한을 추가할 수 있습니다.

Note

[AWS IoT Greengrass 코어 소프트웨어 설치 관리자](#)를 사용하여 리소스를 프로비저닝한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 `GreengrassV2IoTThingPolicy`인 정책을 선택하는 것이 좋습니다(있는 경우). 빠른 설치 프로그램으로 생성한 코어 디바이스에서는 기본적으로 이 정책 이름이 사용됩니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 코어 디바이스에도 이러한 권한을 부여하는 것입니다.

8. 정책 개요에서 활성 버전 편집을 선택합니다.
9. 필수 권한에 대한 정책을 검토하고 누락된 필수 권한이 있으면 추가합니다.
 - `greengrass:PutCertificateAuthorities`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:GetConnectivityInfo`
 - `greengrass:UpdateConnectivityInfo` – (선택 사항)이 권한은 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 [IP 감지기 구성 요소](#)를 사용하는 데 필요합니다.

- `iot:GetThingShadow`, `iot:UpdateThingShadow` 및 `iot:DeleteThingShadow`- (선택 사항) 이러한 권한은 [새도 관리자 구성 요소를](#) 사용하여 클라이언트 디바이스 새도와 동기화하는 데 필요합니다 AWS IoT Core. 이 특성에는 [Greengrass nucleus](#) v2.6.0 이상, 새도 관리자 v2.2.0 이상 및 [MQTT 브리지](#) v2.2.0 이상이 필요합니다.
10. (선택 사항) 코어 디바이스가 새도우를 동기화하도록 허용하려면 정책에 다음 문을 AWS IoT Core 추가합니다. 클라이언트 디바이스 새도우와 상호 작용하지만 동기화하지 않으려는 경우 이 단계를 AWS IoT Core 건너뛴니다. 사용하는 리전 및 AWS 계정 숫자로 `##` 및 `account-id`를 바꿉니다.
- 이 예제 문에서는 모든 사물의 디바이스 새도에 대한 액세스가 허용됩니다. 모범 보안 사례를 따르려면 코어 디바이스에 연결하는 클라이언트 디바이스와 코어 디바이스만으로 액세스를 제한할 수 있습니다. 자세한 내용은 [클라이언트 디바이스를 지원하는 최소 AWS IoT 정책](#) 단원을 참조하십시오.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

이 문을 추가하면 정책 문서가 다음 예제와 비슷하게 보일 수 있습니다.

JSON

```
{
  "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:us-east-1:123456789012: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 IoT 사물 정책 구성(AWS CLI)

1. 코어 디바이스 사물의 보안 주체를 나열합니다 AWS IoT . 사물 보안 주체는 X.509 디바이스 인증서 또는 기타 ID일 수 있습니다. 다음 명령을 실행하고 *MyGreengrassCore*를 코어 디바이스의 이름으로 바꿉니다.

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

코어 디바이스의 사물 보안 주체가 나열되는 응답이 작업에서 반환됩니다.

```

{
  "principals": [
    "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
  ]
}

```

2. 코어 디바이스의 활성 인증서를 식별합니다. 다음 명령을 실행하고, 활성 인증서를 찾을 때까지 이전 단계의 각 인증서 ID로 *certificateId*를 바꿉니다. 인증서 ID는 인증서 ARN 끝에 있는 16진수 문자열입니다. --query 인수에서는 인증서의 상태만 출력으로 지정됩니다.

```
aws iot describe-certificate --certificate-id certificateId --query
'certificateDescription.status'
```

인증서 상태가 문자열로 작업에서 반환됩니다. 예를 들어, 인증서가 활성이라면 이 작업에서는 "ACTIVE"가 출력됩니다.

3. 인증서에 연결된 AWS IoT 정책을 나열합니다. 다음 명령을 실행하고 인증서의 ARN으로 인증서 ARN을 바꿉니다.

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

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 Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝한](#) 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 GreengrassV2IoTThingPolicy인 정책을 선택하는 것이 좋습니다(있는 경우). 빠른 설치 프로그램으로 생성한 코어 디바이스에서는 기본적으로 이 정책 이름이 사용됩니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 코어 디바이스에도 이러한 권한을 부여하는 것입니다.

5. 정책의 문서를 가져옵니다. 다음 명령을 실행하고 정책의 이름으로 *GreengrassV2IoTThingPolicy*를 바꿉니다.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

정책의 문서와 정책에 대한 기타 정보가 있는 응답이 작업에서 반환됩니다. 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 변환기 또는 기타 도구를 사용하여 정책 문서 문자열을 JSON 객체로 변환한 다음에 *iot-policy.json*이라는 파일에 저장합니다.

예를 들어, 설치된 [jq](#) 도구가 있으면 다음 명령을 실행하여 정책 문서를 가져와서 JSON 객체로 변환하고 정책 문서를 JSON 객체로 저장할 수 있습니다.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
'policyDocument' | jq fromjson >> iot-policy.json
```

7. 필수 권한에 대한 정책을 검토하고 누락된 필수 권한이 있으면 추가합니다.

예를 들어 Linux 기반 시스템에서 GNU nano를 사용하여 파일을 열도록 다음 명령을 실행할 수 있습니다.

```
nano iot-policy.json
```

- `greengrass:PutCertificateAuthorities`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:GetConnectivityInfo`
 - `greengrass:UpdateConnectivityInfo` – (선택 사항)이 권한은 코어 디바이스의 네트워크 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 보고하는 [IP 감지기 구성 요소](#)를 사용하는 데 필요합니다.
 - `iot:GetThingShadow`, `iot:UpdateThingShadow` 및 `iot:DeleteThingShadow` – (선택 사항) 이러한 권한은 [새도 관리자 구성 요소](#)를 사용하여 클라이언트 디바이스 새도와 동기화하는 데 필요합니다. AWS IoT Core. 이 특성에는 [Greengrass nucleus](#) v2.6.0 이상, 새도 관리자 v2.2.0 이상 및 [MQTT 브리지](#) v2.2.0 이상이 필요합니다.
8. 변경 사항을 정책의 새 버전으로 저장합니다. 다음 명령을 실행하고 정책의 이름으로 *GreengrassV2IoTThingPolicy*를 바꿉니다.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

작업에 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 디바이스 지원용 Greengrass 구성 요소

 Important

클라이언트 디바이스가 지원되려면 코어 디바이스에서 [Greengrass nucleus](#) v2.2.0 이상이 실행되어야 합니다.

코어 디바이스와 연결되어 통신하도록 클라이언트 디바이스를 활성화하려면 다음 Greengrass 구성 요소를 코어 디바이스에 배포합니다.

- 클라이언트 디바이스 인증 (aws.greengrass.clientdevices.Auth)

클라이언트 디바이스를 인증하고 클라이언트 디바이스 작업을 승인할 클라이언트 디바이스 인증 구성 요소를 배포합니다. 이 구성 요소를 사용하면 AWS IoT 사물을 코어 디바이스에 연결할 수 있습니다.

이 구성 요소를 사용하려면 몇 가지 구성이 필요합니다. 클라이언트 디바이스의 그룹과 각 그룹에서 수행하도록 승인되는 작업(예: MQTT를 통한 연결 및 통신)을 지정해야 합니다. 자세한 내용은 [클라이언트 디바이스 인증 구성 요소 구성](#)을 참조하세요.

- MQTT 3.1.1 브로커(Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

경량 MQTT 브로커를 실행할 Moquette MQTT 브로커 구성 요소를 배포합니다. Moquette MQTT 브로커에서는 MQTT 3.1.1을 준수하며 QoS 0, QoS 1, QoS 2, 보존된 메시지, Last Will 메시지 및 영구 구독에 대한 로컬 지원이 포함되어 있습니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다. 그러나 이 구성 요소에서 MQTT 브로커를 운영하는 포트는 구성해도 됩니다. 기본적으로 포트 8883이 사용됩니다.

- [MQTT 5 브로커\(EMQX\)](#) (aws.greengrass.clientdevices.mqtt.EMQX)

Note

EMQX MQTT 5를 사용하려면 [Greengrass nucleus](#) v2.6.0 이상 및 클라이언트 디바이스 인종 v2.2.0 이상을 사용해야 합니다.

클라이언트 디바이스와 코어 디바이스 간 통신에서 MQTT 5.0 특성이 사용되도록 EMQX MQTT 브로커 구성 요소를 배포합니다. EMQX MQTT 브로커에서는 MQTT 5.0 규정이 준수되며, 세션 및 메시지 만료 간격, 사용자 속성, 공유 구독, 주제 별칭 등에 대한 지원이 포함되어 있습니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다. 그러나 이 구성 요소에서 MQTT 브로커를 운영하는 포트는 구성해도 됩니다. 기본적으로 포트 8883이 사용됩니다.

- [MQTT 브리지](#) (aws.greengrass.clientdevices.mqtt.Bridge)

(선택 사항) MQTT 브리지 구성 요소를 배포하여 클라이언트 디바이스(로컬 MQTT), 로컬 게시/구독 및 AWS IoT Core MQTT 간에 메시지를 릴레이합니다. 클라이언트 디바이스와 동기화 AWS IoT Core 하고 Greengrass 구성 요소의 클라이언트 디바이스와 상호 작용하도록 이 구성 요소를 구성합니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다. 이 구성 요소에서 메시지를 릴레이하는 주제 매핑을 지정해야 합니다. 자세한 내용은 [MQTT 브리지 구성 요소 구성](#)을 참조하세요.

- [IP 감지기](#) (aws.greengrass.clientdevices.IPDetector)

(선택 사항) 코어 디바이스의 MQTT 브로커 엔드포인트를 AWS IoT Greengrass 클라우드 서비스에 자동으로 보고하도록 IP 감지기 구성 요소를 배포합니다. 네트워크 설정이 복잡한 경우(예: 라우터에서 MQTT 브로커 포트를 코어 디바이스에 전달하는 경우) 이 구성 요소를 사용할 수 없습니다.

이 구성 요소는 구성하지 않아도 사용할 수 있습니다.

- [새도 관리자](#) (aws.greengrass.ShadowManager)

Note

클라이언트 디바이스 새도를 관리하려면 [Greengrass nucleus](#) v2.6.0 이상, 새도 관리자 v2.2.0 이상 및 [MQTT 브리지](#) v2.2.0 이상을 사용해야 합니다.

(선택 사항) 새도 관리자 구성 요소를 배포하여 코어 디바이스에서 클라이언트 디바이스 새도를 관리합니다. Greengrass 구성 요소에서는 클라이언트 디바이스와 상호 작용할 클라이언트 디바이스 새도를 가져오고 업데이트하고 삭제할 수 있습니다. 클라이언트 디바이스 새도를 AWS IoT Core 클라우드 서비스와 동기화하도록 새도 관리자 구성 요소를 구성할 수도 있습니다.

클라이언트 디바이스 새도와 함께 이 구성 요소를 사용하려면 로컬 게시/구독이 사용되는 클라이언트 디바이스와 새도 관리자 간 메시지가 릴레이되도록 MQTT 브리지 구성 요소를 구성해야 합니다. 그렇지 않으면 이 구성 요소에는 사용할 구성은 필요하지 않지만, 디바이스 새도를 동기화할 구성은 필요합니다.

Note

MQTT 브로커 구성 요소를 하나만 배포하는 것이 좋습니다. [MQTT 브리지](#) 및 [IP 탐지기](#) 구성 요소는 한 번에 단 하나의 MQTT 브로커 구성 요소와 연동합니다. 여러 MQTT 브로커 구성 요소를 배포하는 경우 다른 포트가 사용되도록 구성해야 합니다.

클라우드 검색 구성(콘솔)

AWS IoT Greengrass 콘솔을 사용하여 클라이언트 디바이스를 연결하고, 코어 디바이스 엔드포인트를 관리하고, 구성 요소를 배포하여 클라이언트 디바이스 지원을 활성화할 수 있습니다. 자세한 내용은 [2 단계: 클라이언트 디바이스 지원 활성화](#) 단원을 참조하십시오.

클라우드 검색 구성(AWS CLI)

AWS Command Line Interface (AWS CLI)를 사용하여 클라이언트 디바이스를 연결하고, 코어 디바이스 엔드포인트를 관리하고, 구성 요소를 배포하여 클라이언트 디바이스 지원을 활성화할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결 관리\(AWS CLI\)](#)
- [코어 디바이스 엔드포인트 관리](#)

- [AWS에서 제공하는 클라이언트 디바이스 구성 요소](#)
- [배포 만들기](#)

클라이언트 디바이스 연결

클라우드 검색을 사용하려면 코어 디바이스를 검색할 수 있도록 클라이언트 디바이스를 코어 디바이스와 연결합니다. 그러면 연결된 코어 디바이스에 대한 연결 정보와 인증서를 검색하는 데 [Greengrass 검색 API](#)가 사용될 수 있습니다.

마찬가지로, 코어 디바이스 검색을 중지시키려면 클라이언트 디바이스로부터 코어 디바이스 연결을 해제합니다.

주제

- [클라이언트 디바이스 연결 관리\(콘솔\)](#)
- [클라이언트 디바이스 연결 관리\(AWS CLI\)](#)
- [클라이언트 디바이스 연결 관리\(API\)](#)

클라이언트 디바이스 연결 관리(콘솔)

AWS IoT Greengrass 콘솔을 사용하여 클라이언트 디바이스 연결을 확인, 추가 및 삭제할 수 있습니다.

코어 디바이스에 대한 클라이언트 디바이스 연결을 보는 방법(콘솔)

1. [AWS IoT Greengrass 콘솔](#)로 이동합니다.
2. 코어 디바이스를 선택합니다.
3. 관리할 코어 디바이스를 선택합니다.
4. 코어 디바이스의 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. 연결된 클라이언트 디바이스 섹션에서 코어 디바이스와 연결된 클라이언트 디바이스(AWS IoT 사물)를 확인할 수 있습니다.

클라이언트 디바이스를 코어 디바이스와 연결하는 방법(콘솔)

1. [AWS IoT Greengrass 콘솔](#)로 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 수행하십시오.
 - a. 클라이언트 디바이스로 연결할 AWS IoT 사물의 이름을 입력합니다.
 - b. 추가를 선택합니다.
7. 연결을 선택합니다.

연결한 클라이언트 디바이스는 이제 Greengrass 검색 API를 사용하여 이 코어 디바이스를 검색할 수 있습니다.

코어 디바이스로부터 클라이언트 디바이스 연결을 해제하는 방법(콘솔)

1. [AWS IoT Greengrass 콘솔](#)로 이동합니다.
2. 코어 디바이스를 선택합니다.
3. 관리할 코어 디바이스를 선택합니다.
4. 코어 디바이스의 세부 정보 페이지에서 클라이언트 디바이스 탭을 선택합니다.
5. 연결을 해제할 각 클라이언트 디바이스를 연결된 클라이언트 디바이스 섹션에서 선택합니다.
6. 연결 해제를 선택합니다.
7. 확인 모달에서 연결 해제를 선택합니다.

연결 해제한 클라이언트 디바이스에서 더는 Greengrass 검색 API를 사용하여 이 코어 디바이스를 검색할 수 있습니다.

클라이언트 디바이스 연결 관리(AWS CLI)

AWS Command Line Interface (AWS CLI)를 사용하여 코어 디바이스에 대한 클라이언트 디바이스 연결을 관리할 수 있습니다.

코어 디바이스에 대한 클라이언트 디바이스 연결을 보는 방법(AWS CLI)

- [list-client-devices-associated-with-core-device](#) 명령을 사용합니다.

클라이언트 디바이스를 코어 디바이스와 연결하는 방법(AWS CLI)

- [batch-associate-client-device-with-core-device](#) 명령을 사용합니다.

코어 디바이스로부터 클라이언트 디바이스 연결을 해제하는 방법(AWS CLI)

- [batch-disassociate-client-device-from-core-device](#) 명령을 사용합니다.

클라이언트 디바이스 연결 관리(API)

AWS API를 사용하여 코어 디바이스에 대한 클라이언트 디바이스 연결을 관리할 수 있습니다.

코어 디바이스에 대한 클라이언트 디바이스 연결을 보려면(AWS API)

- [ListClientDevicesAssociatedWithCoreDevice](#) 작업을 사용합니다.

클라이언트 디바이스를 코어 디바이스와 연결하려면(AWS API)

- [BatchAssociateClientDeviceWithCoreDevice](#) 작업을 사용합니다.

코어 디바이스에서 클라이언트 디바이스의 연결을 해제하려면(AWS API)

- [BatchDisassociateClientDeviceFromCoreDevice](#) 작업을 사용합니다.

오프라인 상태에서 클라이언트 인증

오프라인 인증을 사용하면 AWS IoT Greengrass 코어 디바이스가 클라우드에 연결되지 않은 경우에도 클라이언트 디바이스가 코어 디바이스에 연결할 수 있도록 코어 디바이스를 구성할 수 있습니다. 오프라인 인증을 사용하는 경우 Greengrass 디바이스가 부분적으로 오프라인 환경에서 계속 작동할 수 있습니다.

클라우드에 연결된 클라이언트 디바이스에 대해 오프라인 인증을 사용하려면 다음이 필요합니다.

- [클라이언트 디바이스 인증](#) 구성 요소가 배포된 AWS IoT Greengrass 코어 디바이스입니다. 오프라인 인증에는 버전 2.3.0 이상을 사용해야 합니다.
- 클라이언트 디바이스의 초기 연결 중 코어 디바이스에 대한 클라우드 연결.

클라이언트 자격 증명 저장

클라이언트 디바이스가 코어 디바이스에 처음 연결되면 코어 디바이스는 AWS IoT Greengrass 서비스를 호출합니다. 직접 호출 시 Greengrass는 클라이언트 디바이스의 등록을 AWS IoT 사물로 확인합니다. 또한 디바이스에 유효한 인증서가 있는지 확인합니다. 그런 다음 코어 디바이스는 이 정보를 로컬에 저장합니다.

다음에 디바이스가 연결될 때 Greengrass 코어 디바이스는 AWS IoT Greengrass 서비스로 클라이언트 디바이스를 검증하려고 시도합니다. 연결할 수 없는 경우 코어 디바이스는 로컬 AWS IoT Greengrass에 저장된 디바이스 정보를 사용하여 클라이언트 디바이스를 검증합니다.

Greengrass 코어 디바이스가 자격 증명을 저장하는 기간을 구성할 수 있습니다. [클라이언트 디바이스 인증 구성 요소 구성](#)에서 `clientDeviceTrustDurationMinutes` 구성 옵션을 설정하여 제한 시간을 1분에서 2,147,483,647분으로 설정할 수 있습니다. 기본값인 1분을 설정하면 오프라인 인증이 실질적으로 해제됩니다. 이 제한 시간을 설정할 때는 보안 요구 사항을 고려하는 것이 좋습니다. 또한 클라우드와 연결이 끊긴 상태에서 코어 디바이스를 실행해야 하는 기간도 고려해야 합니다.

코어 디바이스는 자격 증명 스토리지를 세 번 업데이트합니다.

1. 디바이스가 코어 디바이스에 처음으로 연결할 때
2. 코어 디바이스가 클라우드에 연결된 경우 클라이언트 디바이스가 코어 디바이스에 다시 연결할 때
3. 코어 디바이스가 클라우드에 연결된 경우 하루에 한 번 전체 자격 증명 저장소를 새로 고칠 때

Greengrass 코어 디바이스가 보안 인증 저장소를 새로 고칠 때에는

[ListClientDevicesAssociatedWithCoreDevice](#) 작업을 사용합니다. Greengrass는 이 작업에서 반환되는 디바이스만 새로 고칩니다. 클라이언트 디바이스를 코어 디바이스와 연결하려면 [클라이언트 디바이스 연결](#) 섹션을 참조하세요.

`ListClientDevicesAssociatedWithCoreDevice` 작업을 사용하려면 실행되는 AWS 계정 연결된 AWS Identity and Access Management (IAM) 역할에 작업에 대한 권한을 추가해야 합니다. AWS IoT Greengrass. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

코어 디바이스 엔드포인트 관리

클라우드 검색을 사용할 때 코어 디바이스의 MQTT 브로커 엔드포인트를 AWS IoT Greengrass 클라우드 서비스에 저장합니다. 클라이언트 디바이스는 AWS IoT Greengrass에 연결하여 연결된 코어 디바이스에 대한 이러한 엔드포인트 및 기타 정보를 검색합니다.

각 코어 디바이스에 대해 엔드포인트를 자동 또는 수동으로 관리할 수 있습니다.

- IP 감지기로 엔드포인트 자동 관리

클라이언트 디바이스가 코어 디바이스와 동일한 네트워크에 있는 경우와 같이 네트워크 설정이 복잡하지 않은 경우 [IP 감지기 구성 요소](#)를 배포하여 코어 디바이스 엔드포인트를 자동으로 관리할 수 있습니다. 예를 들어 코어 디바이스가 MQTT 브로커 포트를 코어 디바이스로 전달하는 라우터 뒤에 있는 경우 IP 감지기 구성 요소를 사용할 수 없습니다.

IP 감지기 구성 요소는 사물 그룹에 배포하는 경우에도 사물 그룹의 모든 코어 디바이스에 대한 엔드포인트를 관리하기 때문에 유용합니다. 자세한 내용은 [IP 감지기를 사용하여 엔드포인트 자동 관리](#) 단원을 참조하십시오.

- 엔드포인트 수동 관리

IP 감지기 구성 요소를 사용할 수 없는 경우 코어 디바이스 엔드포인트를 수동으로 관리해야 합니다. 콘솔 또는 API를 사용하여 이러한 엔드포인트를 업데이트할 수 있습니다. 자세한 내용은 [엔드포인트 수동 관리](#) 단원을 참조하십시오.

주제

- [IP 감지기를 사용하여 엔드포인트 자동 관리](#)
- [엔드포인트 수동 관리](#)

IP 감지기를 사용하여 엔드포인트 자동 관리

코어 디바이스와 동일한 네트워크에 있는 클라이언트 디바이스와 같이 네트워크 설정이 간단한 경우에는 [IP 감지기 구성 요소](#)를 배포하여 다음을 수행할 수 있습니다.

- Greengrass 코어 디바이스의 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 보고합니다.

IP 감지기 구성 요소는 수동으로 설정한 엔드포인트를 덮어씁니다.

⚠ Important

코어 디바이스의 AWS IoT 정책은 IP 감지기 구성 요소를 사용할 수 있는 `greengrass:UpdateConnectivityInfo` 권한을 허용해야 합니다. 자세한 내용은 [AWS IoT 데이터 영역 작업에 대한 정책](#) 및 [AWS IoT 사물 정책 구성](#) 섹션을 참조하세요.

다음 중 하나를 수행하여 IP 감지기 구성 요소를 배포할 수 있습니다.

- 콘솔의 검색 구성 페이지를 사용합니다. 자세한 내용은 [클라우드 검색 구성\(콘솔\)](#) 단원을 참조하십시오.
- IP 감지기를 포함하도록 배포를 생성 및 수정합니다. 콘솔, AWS CLI 또는 AWS API를 사용하여 배포를 관리할 수 있습니다. 자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

IP 감지기 구성 요소 배포(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 퍼블릭 구성 요소 탭을 선택한 다음 `aws.greengrass.clientdevices.IPDetector`를 선택합니다.
3. `aws.greengrass.clientdevices.IPDetector` 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포 생성을 선택하고 다음을 선택합니다.
5. 새 배포 생성을 선택한 경우 배포의 대상 코어 디바이스 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 디바이스 또는 사물 그룹을 선택하고 다음을 선택합니다.
6. 구성 요소 선택 페이지에서 `aws.greengrass.clientdevices.IPDetector` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 `aws.greengrass.clientdevices.IPDetector`를 선택하고 다음을 수행합니다.
 - a. 구성 요소 구성을 선택합니다.
 - b. `aws.greengrass.clientdevices.IPDetector` 구성 모달의 구성 업데이트 아래의 병합할 구성에서 구성 업데이트를 입력하여 IP 감지기 구성 요소를 구성할 수 있습니다. 다음 구성 옵션을 지정할 수 있습니다.
 - `defaultPort` – (선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다.

- `includeIPv4LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.
- `includeIPv4LinkLocalAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.
- `includeIPv6LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv6 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`, `includeIPv6Addrs`를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv6LinkLocalAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv6 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`, `includeIPv6Addrs`를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv4Addrs` – (선택 사항) 기본값은 `True`로 설정됩니다. 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv4 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv6Addrs` – (선택 사항) 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv6 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`로 설정합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

구성 업데이트는 다음 예제와 유사할 수 있습니다.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false
}
```

- 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 1분 정도 걸릴 수 있습니다.

IP 감지기 구성 요소 배포(AWS CLI)

IP 감지기 구성 요소를 배포하려면 components 객체에

`aws.greengrass.clientdevices.IPDetector`를 포함하는 배포 문서를 생성하고 구성 요소에 대한 구성 업데이트를 지정합니다. [배포 만들기](#)의 지침에 따라 새 배포를 생성하거나 기존 배포를 수정합니다.

배포 문서를 생성하는 경우 다음 옵션 중 하나를 지정하여 IP 감지기 구성 요소를 구성할 수 있습니다.

- `defaultPort` – (선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다.
- `includeIPv4LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.
- `includeIPv4LinkLocalAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.
- `includeIPv6LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv6 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`, `includeIPv6Addrs`를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv6LinkLocalAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv6 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`, `includeIPv6Addrs`를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv4Addrs` – (선택 사항) 기본값은 `True`로 설정됩니다. 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv4 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

- `includeIPv6Addrs` – (선택 사항) 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv6 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`로 설정합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

다음 부분 배포 문서 예제에서는 포트 8883을 MQTT 브로커 포트로 보고하도록 지정합니다.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\"}"
      }
    }
  }
}
```

엔드포인트 수동 관리

코어 디바이스의 MQTT 브로커 엔드포인트를 수동으로 관리할 수 있습니다.

각 MQTT 브로커 엔드포인트에 대한 정보는 다음과 같습니다.

엔드포인트(HostAddress)

클라이언트 디바이스가 코어 디바이스의 MQTT 브로커에 연결할 수 있는 IP 주소 또는 DNS 주소입니다.

포트 (PortNumber)

코어 디바이스에서 MQTT 브로커가 작동하는 포트입니다.

기본적으로 포트 8883을 사용하는 [Moquette MQTT 브로커 구성 요소](#)에서 이 포트를 구성할 수 있습니다.

메타데이터(Metadata)

이 엔드포인트에 연결하는 클라이언트 디바이스에 제공할 추가 메타데이터입니다.

주제

- [엔드포인트 관리\(콘솔\)](#)
- [엔드포인트 관리\(AWS CLI\)](#)
- [엔드포인트 관리\(API\)](#)

엔드포인트 관리(콘솔)

AWS IoT Greengrass 콘솔을 사용하여 코어 디바이스의 엔드포인트를 확인, 업데이트 및 제거할 수 있습니다.

코어 디바이스의 엔드포인트를 관리하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#)로 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 볼 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 디바이스의 엔드포인트를 관리할 수 있습니다.

Note

의 클라이언트 디바이스 지원 AWS IoT Greengrass V2 은 이전 버전과 호환되므로 AWS IoT Greengrass V2 또는 AWS IoT Greengrass V1 API 작업을 사용하여 코어 디바이스 엔드포인트를 관리할 AWS IoT Greengrass V1 수 있습니다.

코어 디바이스의 엔드포인트를 가져오려면(AWS CLI)

- 다음 명령 중 하나를 실행합니다.
 - [greengrassv2: get-connectivity-info](#)

- [greengrass: get-connectivity-info](#)

코어 디바이스의 엔드포인트를 업데이트하려면(AWS CLI)

- 다음 명령 중 하나를 실행합니다.
 - [greengrassv2: update-connectivity-info](#)
 - [greengrass: update-connectivity-info](#)

엔드포인트 관리(API)

AWS API를 사용하여 코어 디바이스의 엔드포인트를 관리할 수 있습니다.

 Note

의 클라이언트 디바이스 지원 AWS IoT Greengrass V2 은 이전 버전과 호환되므로 AWS IoT Greengrass V2 또는 AWS IoT Greengrass V1 API 작업을 사용하여 코어 디바이스 엔드포인트를 관리할 AWS IoT Greengrass V1수 있습니다.

코어 디바이스의 엔드포인트를 가져오려면(AWS API)

- 다음 작업 중 하나를 사용합니다.
 - [V2: GetConnectivityInfo](#)
 - [V1: GetConnectivityInfo](#)

코어 디바이스의 엔드포인트를 업데이트하려면(AWS API)

- 다음 작업 중 하나를 사용합니다.
 - [V2: UpdateConnectivityInfo](#)
 - [V1: UpdateConnectivityInfo](#)

MQTT 브로커 선택

AWS IoT Greengrass에서는 코어 디바이스에서 실행될 로컬 MQTT 브로커를 선택할 수 있는 옵션이 제공됩니다. 코어 디바이스에서 실행되는 MQTT 브로커에 클라이언트 디바이스가 연결되므로 연결하려는 클라이언트 디바이스와 호환되는 MQTT 브로커를 선택합니다.

Note

MQTT 브로커 구성 요소를 하나만 배포하는 것이 좋습니다. [MQTT 브리지](#) 및 [IP 탐지기](#) 구성 요소는 한 번에 단 하나의 MQTT 브로커 구성 요소와 연동합니다. 여러 MQTT 브로커 구성 요소를 배포하는 경우 다른 포트가 사용되도록 구성해야 합니다.

다음과 같은 MQTT 브로커 중에서 선택할 수 있습니다.

- [MQTT 3.1.1 broker\(Moquette\)](#) - `aws.greengrass.clientdevices.mqtt.Moquette`

MQTT 3.1.1 표준이 준수되는 경량 MQTT 브로커에 이 옵션을 선택합니다. AWS IoT Core MQTT 브로커와 AWS IoT Device SDK에서는 MQTT 3.1.1 표준도 준수되므로 이러한 특성을 사용하여 디바이스와 AWS 클라우드에 걸쳐 MQTT 3.1.1가 사용되는 애플리케이션을 생성할 수 있습니다.

- [MQTT 5 브로커\(EMQX\)](#) - `aws.greengrass.clientdevices.mqtt.EMQX`

코어 디바이스와 클라이언트 디바이스 간 통신에서 MQTT 5 특성을 사용하려면 이 옵션을 선택합니다. 이 구성 요소에서는 Moquette MQTT 3.1.1 브로커보다 많은 리소스가 사용되며, Linux 코어 디바이스에서는 Docker가 필요합니다.

MQTT 5는 MQTT 3.1.1과 역호환되므로 MQTT 3.1.1이 사용되는 클라이언트 디바이스를 이 브로커에 연결할 수 있습니다. Moquette MQTT 3.1.1 브로커를 실행하는 경우 EMQX MQTT 5로 이 브로커를 바꿀 수 있으며, 클라이언트 디바이스는 평소대로 계속 연결되어 작동할 수 있습니다.

- 사용자 지정 브로커 구현

클라이언트 디바이스와 통신할 사용자 지정 로컬 브로커 구성 요소를 생성하려면 이 옵션을 선택합니다. MQTT 이외의 프로토콜이 사용되는 사용자 지정 로컬 브로커를 생성할 수 있습니다. AWS IoT Greengrass에서는 클라이언트 디바이스를 인증하고 권한을 부여하는 데 사용할 수 있는 구성 요소 SDK가 제공됩니다. 자세한 내용은 [AWS IoT Device SDK를 사용하여 Greengrass nucleus, 기타 구성 요소 및 AWS IoT Core 및 클라이언트 디바이스 인증 및 권한 부여\(을\)](#)를 참조하세요.

MQTT 브로커를 사용하여 AWS IoT Greengrass 코어 디바이스에 클라이언트 디바이스 연결

AWS IoT Greengrass 코어 디바이스에서 MQTT 브로커를 사용하는 경우 디바이스는 디바이스에 고유한 코어 디바이스 인증 기관(CA)을 사용하여 클라이언트와 상호 TLS 연결을 만들기 위해 브로커에 인증서를 발급합니다.

AWS IoT Greengrass 는 코어 디바이스 CA를 자동 생성하거나 사용자가 직접 제공할 수 있습니다. 코어 디바이스 CA는 [클라이언트 디바이스 인증](#) 구성 요소가 연결될 AWS IoT Greengrass 때에 등록됩니다. 자동 생성된 코어 디바이스 CA는 영구적이며, 클라이언트 디바이스 인증 구성 요소가 구성되어 있는 한 동일한 CA가 디바이스에서 계속 사용됩니다.

MQTT 브로커가 시작되면 인증서가 요청됩니다. 코어 디바이스 CA를 사용하여 클라이언트 디바이스 인증 구성 요소에서 X.509 인증서가 발급합니다. 브로커가 시작되거나 인증서가 만료되거나 IP 주소와 같은 연결 정보가 변경되면 인증서가 교체됩니다. 자세한 내용은 [로컬 MQTT 브로커의 인증서 교체](#) 단원을 참조하십시오.

클라이언트를 MQTT 브로커에 연결하려면 다음이 필요합니다.

- 클라이언트 디바이스에는 AWS IoT Greengrass 코어 디바이스 CA가 있어야 합니다. 클라우드 검색을 통하거나 CA를 수동으로 제공하여 이 CA를 가져올 수 있습니다. 자세한 내용은 [자체 인증 기관 사용](#) 단원을 참조하십시오.
- 코어 디바이스 CA에서 발급된 브로커 인증서에 코어 디바이스의 정규화된 도메인 이름(FQDN) 또는 IP 주소가 있어야 합니다. [IP 감지기](#) 구성 요소를 사용하거나 IP 주소를 수동으로 구성하면 확인됩니다. 자세한 내용은 [코어 디바이스 엔드포인트 관리](#) 단원을 참조하십시오.
- 클라이언트 디바이스 인증 구성 요소에서는 Greengrass 코어 디바이스에 연결할 권한이 클라이언트 디바이스에 부여되어야 합니다. 자세한 내용은 [클라이언트 디바이스 인증](#) 단원을 참조하십시오.

자체 인증 기관 사용

클라이언트 디바이스에서 클라우드에 액세스하여 코어 디바이스를 검색할 수 없으면 코어 디바이스 인증 기관(CA)을 제공할 수 있습니다. Greengrass 코어 디바이스에서는 MQTT 브로커에 대한 인증서 발급에 코어 디바이스 CA가 사용됩니다. 코어 디바이스를 구성하고 CA를 통해 클라이언트 디바이스를 프로비저닝하면 클라이언트 디바이스에서는 엔드포인트에 연결되어 코어 디바이스 CA(자체 제공하거나 자동 생성된 CA)를 통해 TLS 핸드셰이크가 확인될 수 있습니다.

코어 디바이스 CA가 사용되도록 [클라이언트 디바이스 인증](#) 구성 요소를 구성하려면 구성 요소를 배포할 때 certificateAuthority 구성 파라미터를 설정합니다. 구성 중 다음 세부 정보를 제공해야 합니다.

- 코어 디바이스 CA 인증서의 위치.
- 코어 디바이스 CA 인증서의 프라이빗 키.
- (선택 사항) 코어 디바이스 CA가 중간 CA인 경우 루트 인증서에 대한 인증서 체인입니다.

코어 디바이스 CA를 제공하는 경우는 클라우드에 CA를 AWS IoT Greengrass 등록합니다.

인증서를 하드웨어 보안 모듈 또는 파일 시스템에 저장할 수 있습니다. 다음 예제에서는 HSM/TPM을 사용하여 저장된 중간 CA의 certificateAuthority 구성을 보여줍니다. 참고: 인증서 체인은 디스크에만 저장할 수 있습니다.

```
"certificateAuthority": {
  "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
  "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
  "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
}
```

이 예제에서 certificateAuthority 구성 파라미터에서는 파일 시스템의 중간 CA이 사용되도록 클라이언트 디바이스 인증 구성 요소가 구성됩니다.

```
"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",
}
```

디바이스를 AWS IoT Greengrass 코어 디바이스에 연결하려면 다음을 수행합니다.

1. 조직의 루트 CA를 사용하여 Greengrass 코어 디바이스에 대한 중간 인증 기관(CA)을 생성합니다. 중간 CA를 보안 모듈 사례로 사용하는 것이 좋습니다.
2. 루트 CA에 대한 중간 CA 인증서, 프라이빗 키 및 인증서 체인을 Greengrass 코어 디바이스에 제공합니다. 자세한 내용은 [클라이언트 디바이스 인증](#) 단원을 참조하십시오. 중간 CA는 Greengrass 코어 디바이스의 코어 디바이스 CA가 되고 디바이스는 CA를 등록합니다 AWS IoT Greengrass.
3. 클라이언트 디바이스를 AWS IoT 사물로 등록합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사물 객체 생성](#)을 참조하세요. 프라이빗 키, 퍼블릭 키, 디바이스 인증서 및 루트 CA 인증서를 클라이언트 디바이스에 추가합니다. 정보를 추가하는 방법은 디바이스와 소프트웨어에 따라 다릅니다.

디바이스를 구성하면 인증서와 퍼블릭 키 체인을 사용하여 Greengrass 코어 디바이스에 연결할 수 있습니다. 소프트웨어에서 코어 디바이스 엔드포인트 찾기를 담당합니다. 코어 디바이스에 대해 엔드포인트를 수동으로 설정할 수 있습니다. 자세한 내용은 [엔드포인트 수동 관리](#) 단원을 참조하십시오.

클라이언트 디바이스 통신 테스트

클라이언트 디바이스를 사용하여 코어 디바이스를 검색, 연결 및 통신 AWS IoT Device SDK 할 수 있습니다. 에서 Greengrass 검색 클라이언트 AWS IoT Device SDK 를 사용하여 클라이언트 디바이스가 연결할 수 있는 코어 디바이스에 대한 정보를 반환하는 [Greengrass 검색 API](#)를 사용할 수 있습니다. API 응답은 연결할 MQTT 브로커 엔드포인트와 각 코어 디바이스의 ID를 확인하는 데 사용할 인증서를 포함합니다. 그런 다음 클라이언트 디바이스는 코어 디바이스에 성공적으로 연결될 때까지 각 엔드포인트를 시도할 수 있습니다.

클라이언트 디바이스는 연결한 코어 디바이스만 검색할 수 있습니다. 클라이언트 디바이스와 코어 디바이스 간의 통신을 테스트하기 전에 클라이언트 디바이스를 코어 디바이스에 연결해야 합니다. 자세한 내용은 [클라이언트 디바이스 연결](#) 단원을 참조하십시오.

Greengrass 검색 API는 사용자가 지정한 코어 디바이스 MQTT 브로커 엔드포인트를 반환합니다. [IP 감지기 구성 요소](#)를 사용하여 이러한 엔드포인트를 관리할 수도 있고 각 코어 디바이스에 대해 수동으로 관리할 수도 있습니다. 자세한 내용은 [코어 디바이스 엔드포인트 관리](#) 단원을 참조하십시오.

Note

Greengrass 검색 API를 사용하려면 클라이언트 디바이스에 `greengrass:Discover` 권한이 있어야 합니다. 자세한 내용은 [클라이언트 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.

AWS IoT Device SDK 는 여러 프로그래밍 언어로 제공됩니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 디바이스 SDK](#) 섹션을 참조하세요.

주제

- [통신 테스트\(Python\)](#)
- [통신 테스트\(C++\)](#)
- [통신 테스트\(JavaScript\)](#)
- [통신 테스트\(Java\)](#)

통신 테스트(Python)

이 섹션에서는 [Python용AWS IoT Device SDK v2](#)의 Greengrass 검색 샘플을 사용하여 클라이언트 디바이스와 코어 디바이스 간의 통신을 테스트합니다.

⚠ Important

AWS IoT Device SDK v2 for Python을 사용하려면 디바이스가 Python 3.6 이상을 실행해야 합니다.

통신을 테스트하려면(Python용AWS IoT Device SDK v2)

1. 클라이언트 디바이스로 연결할 AWS IoT 사물에 [AWS IoT Device SDK Python용 v2](#)를 다운로드 하여 설치합니다.

클라이언트 디바이스에서 다음을 수행합니다.

- a. AWS IoT Device SDK v2 for Python 리포지토리를 복제하여 다운로드합니다.

```
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. AWS IoT Device SDK v2 for Python의 샘플 폴더로 변경합니다.

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

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 `clients/MyClientDevice1/hello/world` 주제에 Hello World 메시지를 전송합니다.

- `MyClientDevice1`을 클라이언트 디바이스의 사물 이름으로 바꿉니다.
- `~/certs/AmazonRootCA1.pem`을 클라이언트 디바이스의 Amazon 루트 CA 인증서 경로로 바꿉니다.
- `~/certs/device.pem.crt`를 클라이언트 디바이스의 디바이스 인증서 경로로 바꿉니다.
- `~/certs/private.pem.key`를 클라이언트 디바이스의 프라이빗 키 파일 경로로 바꿉니다.
- `us-east-1`을 클라이언트 디바이스와 코어 디바이스가 작동하는 AWS 리전으로 바꿉니다.

```
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 10회 전송하고 연결을 해제합니다. 또한 메시지를 게시하는 주제와 동일한 주제를 구독합니다. 출력에서 애플리케이션이 해당 주제에 대한 MQTT 메시지를 수신했음을 나타내는 경우 클라이언트 디바이스는 코어 디바이스와 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}'
```

그 대신에 애플리케이션에서 오류가 출력되면 [Greengrass 검색 문제 해결](#)을 참조하세요.

코어 디바이스에서 Greengrass 로그를 보고 클라이언트 디바이스가 메시지를 성공적으로 연결하고 전송하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

통신 테스트(C++)

이 섹션에서는 [AWS IoT Device SDK v2 for C++](#)의 Greengrass 검색 샘플을 사용하여 클라이언트 디바이스와 코어 디바이스 간의 통신을 테스트합니다.

C++용 AWS IoT Device SDK v2를 빌드하려면 디바이스에 다음 도구가 있어야 합니다.

- C++ 11 이상
- CMake 3.1 이상
- 다음 컴파일러 중 하나입니다.
 - GCC 4.8 이상
 - Clang 3.9 이상
 - MSVC 2015 이상

통신을 테스트하려면(C++용AWS IoT Device SDK v2)

1. 클라이언트 디바이스로 연결할 AWS IoT 사물에 [AWS IoT Device SDK C++용 v2](#)를 다운로드하고 빌드합니다.

클라이언트 디바이스에서 다음을 수행합니다.

- a. C++ 워크스페이스용 AWS IoT Device SDK v2에 대한 폴더를 생성하고 변경합니다.

```
cd  
mkdir iot-device-sdk-cpp  
cd iot-device-sdk-cpp
```

- b. AWS IoT Device SDK v2 for C++ 리포지토리를 복제하여 다운로드합니다. `--recursive` 플래그는 하위 모듈을 다운로드하도록 지정합니다.

```
git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
```

- c. C++용 AWS IoT Device SDK v2 빌드 출력에 대한 폴더를 생성하고 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 빌드합니다.

```
cmake -DCMAKE_INSTALL_PREFIX="/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
cmake --build . --target install
```

2. C++용 AWS IoT Device SDK v2에서 Greengrass 검색 샘플 애플리케이션을 빌드합니다. 해결 방법:

- a. 를 C++용 AWS IoT Device SDK v2의 Greengrass 검색 샘플 폴더로 변경합니다.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

- b. Greengrass 검색 샘플 빌드 출력을 위한 폴더를 생성하고 해당 폴더로 변경합니다.

```
mkdir build
cd build
```

- c. Greengrass 검색 샘플 애플리케이션을 빌드합니다.

```
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 예제에서는 `clients/MyClientDevice1/hello/world` 주제를 구독하고 명령줄에 입력한 메시지를 동일한 주제에 게시합니다.

- `MyClientDevice1`을 클라이언트 디바이스의 사물 이름으로 바꿉니다.

- `~/certs/AmazonRootCA1.pem`을 클라이언트 디바이스의 Amazon 루트 CA 인증서 경로로 바꿉니다.
- `~/certs/device.pem.crt`를 클라이언트 디바이스의 디바이스 인증서 경로로 바꿉니다.
- `~/certs/private.pem.key`를 클라이언트 디바이스의 프라이빗 키 파일 경로로 바꿉니다.
- `us-east-1`을 클라이언트 디바이스와 코어 디바이스가 작동하는 AWS 리전으로 바꿉니다.

```
./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 애플리케이션이 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.
```

그 대신에 애플리케이션에서 오류가 출력되면 [Greengrass 검색 문제 해결](#)을 참조하세요.

4. 메시지를 입력합니다(예: **Hello World!**).

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
and press enter. Enter 'exit' to exit this program.
Hello World!
```

출력에서 애플리케이션이 해당 주제에 대한 MQTT 메시지를 수신했음을 나타내는 경우 클라이언트 디바이스는 코어 디바이스와 성공적으로 통신할 수 있습니다.

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
```

```
Hello World!
```

코어 디바이스에서 Greengrass 로그를 보고 클라이언트 디바이스가 메시지를 성공적으로 연결하고 전송하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

통신 테스트(JavaScript)

이 섹션에서는 [JavaScript용 AWS IoT Device SDK v2](#)의 Greengrass 검색 샘플을 사용하여 클라이언트 디바이스와 코어 디바이스 간의 통신을 테스트합니다.

Important

JavaScript용 AWS IoT Device SDK v2를 사용하려면 디바이스에서 노드 v10.0 이상을 실행해야 합니다.

통신을 테스트하려면(Javascript용AWS IoT Device SDK v2)

1. [AWS IoT Device SDK v2 for JavaScript](#)를 다운로드하여 클라이언트 디바이스로 연결할 AWS IoT 사물에 설치합니다.

클라이언트 디바이스에서 다음을 수행합니다.

- a. AWS IoT Device SDK v2 for JavaScript 리포지토리를 복제하여 다운로드합니다.

```
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

- b. JavaScript용 AWS IoT Device SDK v2를 설치합니다.

```
cd aws-iot-device-sdk-js-v2
npm install
```

2. AWS IoT Device SDK v2 for JavaScript에서 Greengrass 검색 샘플 폴더로 변경합니다.

```
cd samples/node/greengrass/basic_discovery
```

3. Greengrass 검색 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 `clients/MyClientDevice1/hello/world` 주제에 Hello World 메시지를 전송합니다.

- `MyClientDevice1`을 클라이언트 디바이스의 사물 이름으로 바꿉니다.
- `~/certs/AmazonRootCA1.pem`을 클라이언트 디바이스의 Amazon 루트 CA 인증서 경로로 바꿉니다.
- `~/certs/device.pem.crt`를 클라이언트 디바이스의 디바이스 인증서 경로로 바꿉니다.
- `~/certs/private.pem.key`를 클라이언트 디바이스의 프라이빗 키 파일 경로로 바꿉니다.
- `us-east-1`을 클라이언트 디바이스와 코어 디바이스가 작동하는 AWS 리전으로 바꿉니다.

```
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 10회 전송하고 연결을 해제합니다. 또한 메시지를 게시하는 주제와 동일한 주제를 구독합니다. 출력에서 애플리케이션이 해당 주제에 대한 MQTT 메시지를 수신했음을 나타내는 경우 클라이언트 디바이스는 코어 디바이스와 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":""}]}],"certificate":["-----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!
```

그 대신에 애플리케이션에서 오류가 출력되면 [Greengrass 검색 문제 해결](#)을 참조하세요.

코어 디바이스에서 Greengrass 로그를 보고 클라이언트 디바이스가 메시지를 성공적으로 연결하고 전송하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

통신 테스트(Java)

이 섹션에서는 [Java용 AWS IoT Device SDK v2](#)의 Greengrass 검색 샘플을 사용하여 클라이언트 디바이스와 코어 디바이스 간의 통신을 테스트합니다.

Important

Java용 AWS IoT Device SDK v2를 빌드하려면 디바이스에 다음 도구가 있어야 합니다.

- Java 8 이상(Java 폴더를 가리키는 JAVA_HOME 포함)
- Apache Maven

통신을 테스트하려면(Java용 AWS IoT Device SDK v2)

1. [AWS IoT Device SDK Java용 v2](#)를 다운로드하여 클라이언트 디바이스로 연결할 AWS IoT 사물에 빌드합니다.

클라이언트 디바이스에서 다음을 수행합니다.

- a. AWS IoT Device SDK v2 for Java 리포지토리를 복제하여 다운로드합니다.

```
git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
```

- b. 를 Java용 AWS IoT Device SDK v2 폴더로 변경합니다.
- c. Java용 AWS IoT Device SDK v2를 빌드합니다.

```
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 예제에서는 `clients/MyClientDevice1/hello/world` 주제를 구독하고 명령줄에 입력한 메시지를 동일한 주제에 게시합니다.

- `MyClientDevice1`의 두 인스턴스 모두 클라이언트 디바이스의 사물 이름으로 바꿉니다.
- `$HOME/certs/AmazonRootCA1.pem`을 클라이언트 디바이스의 Amazon 루트 CA 인증서 경로로 바꿉니다.

- `$HOME/certs/device.pem.crt`를 클라이언트 디바이스의 디바이스 인증서 경로로 바꿉니다.
- `$HOME/certs/private.pem.key`를 클라이언트 디바이스의 프라이빗 키 파일 경로로 바꿉니다.
- `us-east-1`을 클라이언트 디바이스와 코어 디바이스가 작동하는 AWS 리전 로 바꿉니다.

```
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/Discovery \
-Dexec.mainClass=greengrass.BasicDiscovery \
-Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

검색 샘플 애플리케이션이 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:
```

그 대신에 애플리케이션에서 오류가 출력되면 [Greengrass 검색 문제 해결](#)을 참조하세요.

3. 메시지를 입력합니다(예: **Hello World!**).

```
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!
```

출력에서 애플리케이션이 해당 주제에 대한 MQTT 메시지를 수신했음을 나타내는 경우 클라이언트 디바이스는 코어 디바이스와 성공적으로 통신할 수 있습니다.

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

코어 디바이스에서 Greengrass 로그를 보고 클라이언트 디바이스가 메시지를 성공적으로 연결하고 전송하는지 확인할 수도 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

Greengrass Discovery RESTful API

AWS IoT Greengrass 는 클라이언트 디바이스가 연결할 수 있는 Greengrass 코어 디바이스를 식별하는 데 사용할 수 있는 Discover API 작업을 제공합니다. 클라이언트 디바이스는 이 데이터 플레인 작업을 사용하여 [BatchAssociateClientDeviceWithCoreDevice](#) API 작업과 관련된 Greengrass 코어 디바이스에 연결하는 데 필요한 정보를 검색합니다. 클라이언트 디바이스가 온라인 상태가 되면 AWS IoT Greengrass 클라우드 서비스에 연결하고 검색 API를 사용하여 다음을 찾을 수 있습니다.

- 연결된 각 Greengrass 코어 디바이스의 IP 주소 및 포트.
- 클라이언트 디바이스가 Greengrass 코어 디바이스 인증에 사용할 수 있는 클라이언트 디바이스 CA 인증서.

Note

클라이언트 디바이스는의 검색 클라이언트를 사용하여 Greengrass 코어 디바이스의 연결 정보를 AWS IoT Device SDK 검색할 수도 있습니다. Discovery 클라이언트는 Discovery API를 사용합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 통신 테스트](#)
- AWS IoT Greengrass Version 1 개발자 안내서의 [Greengrass Discovery RESTful API](#).

이 API를 사용하려면 Greengrass 데이터 플레인 엔드포인트의 Discovery API 엔드포인트에 HTTP 요청을 전송합니다. 이 API 엔드포인트의 형식은 다음과 같습니다.

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

AWS IoT Greengrass 검색 API에 대해 지원되는 AWS 리전 및 엔드포인트 목록은의 [AWS IoT Greengrass V2 엔드포인트 및 할당량을 참조하세요](#) 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

검색 API는 AWS IoT Greengrass V1 및에 대해 동일합니다 AWS IoT Greengrass V2. AWS IoT Greengrass V1 코어에 연결하는 클라이언트 디바이스가 있는 경우 클라이언트 디바이스의 코드를 변경하지 않고 AWS IoT Greengrass V2 코어 디바이스에 연결할 수 있습니다. 자세한 내용은 AWS IoT Greengrass Version 1 개발자 안내서의 [Greengrass Discovery RESTful API](#)를 참조하세요.

주제

- [Discovery 인증 및 권한 부여](#)
- [요청](#)
- [응답](#)
- [cURL을 사용하여 Discovery API 테스트](#)

Discovery 인증 및 권한 부여

Discovery API를 사용하여 연결 정보를 검색하려면 클라이언트 디바이스가 X.509 클라이언트 인증서와 함께 TLS 상호 인증을 사용하여 인증해야 합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [X.509 클라이언트 인증서](#)를 참조하세요.

클라이언트 디바이스에는 `greengrass:Discover` 작업을 수행할 수 있는 권한도 있어야 합니다. 다음 예제 AWS IoT 정책은 이름이 `인`인 AWS IoT 사물이 `Discover` 자체적으로 수행MyClientDevice1되도록 허용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

```
]
}
```

⚠ Important

코어 디바이스 또는 Greengrass 데이터 플레인 작업의 AWS IoT 정책에 대해 [사물 정책 변수](#)(`iot:Connection.Thing.*`)가 지원되지 않습니다. 그 대신에 이름이 비슷한 여러 디바이스와 일치하는 와일드카드를 사용할 수 있습니다. 예를 들어 `MyGreengrassDevice1`, `MyGreengrassDevice2` 등과 일치하도록 `MyGreengrassDevice*`를 지정할 수 있습니다.

자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT Core 정책](#)을 참조하세요.

요청

다음 예제와 같이 요청은 표준 HTTP 헤더를 포함하고 있으며, Greengrass Discovery 엔드포인트로 전송됩니다.

포트 번호는 코어 디바이스가 포트 8443 또는 포트 443을 통해 HTTPS 트래픽을 보내도록 구성되었는지 여부에 따라 다릅니다. 자세한 내용은 [the section called “포트 443에서 또는 네트워크 프록시를 통해 연결”](#) 단원을 참조하십시오.

ℹ Note

이 예제에서는 권장되는 ATS 루트 CA 인증서에 사용되는 ATS(Amazon Trust Services) 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 ProtocolNameList에 x-amzn-http-ca를 ProtocolName으로 전달해야 합니다. 자세한 정보는 AWS IoT 개발자 안내서의 [프로토콜](#)을 참조하세요.

응답

성공 시 응답 헤더에는 HTTP 200 상태 코드가 포함되고 응답 본문에는 검색 응답 문서가 포함됩니다.

Note

AWS IoT Greengrass V2 는와 동일한 검색 API를 사용하므로 응답 AWS IoT Greengrass V1 은 Greengrass 그룹과 같은 AWS IoT Greengrass V1 개념에 따라 정보를 구성합니다. 응답에는 Greengrass 그룹 목록이 포함됩니다. 에서 AWS IoT Greengrass V2 각 코어 디바이스는 자체 그룹에 있으며, 그룹에는 해당 코어 디바이스와 연결 정보만 포함됩니다.

검색 응답 문서 예제

다음 문서는 하나의 Greengrass 코어 디바이스에 연결된 클라이언트 디바이스에 대한 응답을 보여줍니다. 코어 디바이스에는 하나의 엔드포인트 하나의 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-description"
            }
          ]
        }
      ]
    }
  ],
}
```

```

    "CAs": [
      "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
    ]
  }
]
}

```

다음 문서는 두 개의 코어 디바이스에 연결된 클라이언트 디바이스에 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 Discovery API 테스트

cURL을 설치한 경우 Discovery API를 테스트할 수 있습니다. 다음 예제에서는 Greengrass Discovery API 엔드포인트에 대한 요청을 인증하기 위해 클라이언트 디바이스의 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 사용하여 오류를 식별할 수 있습니다.

요청에 성공하는 경우 이 명령에서 다음 예제와 비슷한 응답이 출력됩니다.

```

{
  "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 Discovery 문제 해결](#)을 참조하세요.

클라이언트 디바이스와 간에 MQTT 메시지 릴레이 AWS IoT Core

클라이언트 디바이스와 AWS IoT Core간에 MQTT 메시지 및 기타 데이터를 전달할 수 있습니다. 클라이언트 디바이스는 코어 디바이스에서 실행되는 MQTT 브로커 구성 요소에 연결합니다. 기본적으로 코어 디바이스는 클라이언트 디바이스와 간에 MQTT 메시지 또는 데이터를 릴레이하지 않습니다 AWS IoT Core. 클라이언트 디바이스는 기본적으로 MQTT를 통해서만 서로 통신할 수 있습니다.

클라이언트 디바이스와 간에 MQTT 메시지를 릴레이하려면 다음을 수행하도록 [MQTT 브리지 구성 요소를](#) AWS IoT Core구성합니다.

- 클라이언트 디바이스에서 로 메시지를 릴레이합니다 AWS IoT Core.
- 에서 클라이언트 디바이스 AWS IoT Core 로 메시지를 릴레이합니다.

Note

MQTT 브리지는 클라이언트 디바이스가 QoS 0을 사용하여 로컬 MQTT 브로커를 게시하고 구독 AWS IoT Core하는 경우에도 QoS 1을 사용하여 게시하고 구독합니다. 따라서 로컬 MQTT

브로커의 클라이언트 디바이스에서 MQTT 메시지를 릴레이할 때 추가 지연 시간이 관찰될 수 있습니다 AWS IoT Core. 코어 디바이스의 MQTT 구성에 대한 자세한 내용은 [MQTT 제한 시간 및 캐시 설정 구성](#) 섹션을 참조하세요.

주제

- [MQTT 브리지 구성 요소 구성 및 배포](#)
- [MQTT 메시지 전달](#)

MQTT 브리지 구성 요소 구성 및 배포

MQTT 브리지 구성 요소는 각각 메시지 소스와 메시지 대상을 지정하는 주제 매핑의 목록을 사용합니다. 클라이언트 디바이스와 간에 메시지를 릴레이하려면 MQTT 브리지 구성 요소를 AWS IoT Core 배포하고 구성 요소 구성에서 각 소스 및 대상 주제를 지정합니다.

코어 디바이스 또는 코어 디바이스 그룹에 MQTT 브리지 구성 요소를 배포하려면 `aws.greengrass.clientdevices.mqtt.Bridge` 구성 요소가 포함된 [배포를 생성](#)합니다. 배포의 MQTT 브리지 구성 요소 구성에서 주제 매핑(`mqttTopicMapping`)을 지정합니다.

다음 예제에서는 클라이언트 디바이스에서 `clients/+/hello/world` 주제 필터와 일치하는 토픽에 대한 메시지를 AWS IoT Core로 전달하도록 MQTT 브리지 구성 요소를 구성하는 배포를 정의합니다. `merge` 구성 업데이트에는 직렬화된 JSON 객체가 필요합니다. 자세한 내용은 [구성 요소 구성 업데이트](#) 단원을 참조하십시오.

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\"}}}"
        }
      }
      ...
    }
  }
}

```

MQTT 메시지 전달

클라이언트 디바이스와 간에 MQTT 메시지를 릴레이하려면 MQTT 브리지 구성 요소를 AWS IoT Core 구성 및 배포하고 릴레이할 주제를 지정합니다. ???

Example예: 클라이언트 디바이스에서 주제의 메시지를 로 릴레이 AWS IoT Core

다음 MQTT 브리지 구성 요소 구성은 클라이언트 디바이스에서 `clients/+hello/world/event` 주제 필터와 일치하는 주제에 대한 메시지를 AWS IoT Core로 전달하도록 지정합니다.

```

{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}

```

Example예: 주제에 대한 메시지를에서 클라이언트 디바이스 AWS IoT Core 로 릴레이

다음 MQTT 브리지 구성 요소 구성은 AWS IoT Core 에서 `clients/+hello/world/event/response` 주제 필터와 일치하는 주제에 대한 메시지를 클라이언트 디바이스로 전달하도록 지정합니다.

```

{
  "mqttTopicMapping": {

```

```
"HelloWorldEventConfirmation": {  
  "topic": "clients/+/hello/world/event/response",  
  "source": "IotCore",  
  "target": "LocalMqtt"  
}  
}  
}
```

구성 요소의 클라이언트 디바이스와 상호 작용

코어 디바이스에 연결된 클라이언트 디바이스와 상호 작용하는 사용자 지정 Greengrass 구성 요소를 개발할 수 있습니다. 예를 들어 다음을 수행하는 구성 요소를 개발할 수 있습니다.

- 클라이언트 디바이스의 MQTT 메시지에 대한 작업을 수행하고 AWS 클라우드 대상에 데이터를 전송합니다.
- 클라이언트 디바이스에 MQTT 메시지를 전송하여 작업을 시작합니다.

클라이언트 디바이스는 코어 디바이스에서 실행되는 MQTT 브로커 구성 요소를 통해 코어 디바이스와 연결하고 통신합니다. 기본적으로 클라이언트 디바이스는 MQTT를 통해서만 서로 통신할 수 있으며, Greengrass 구성 요소는 이러한 MQTT 메시지를 수신하거나 클라이언트 디바이스에 메시지를 보낼 수 없습니다.

Greengrass 구성 요소는 [로컬 게시/구독 인터페이스](#)를 사용하여 코어 디바이스에서 통신합니다. Greengrass 구성 요소의 클라이언트 디바이스와 통신하려면 다음을 수행하도록 [MQTT 브리지 구성 요소](#)를 구성합니다.

- MQTT 메시지를 클라이언트 디바이스에서 로컬 게시/구독으로 릴레이합니다.
- MQTT 메시지를 로컬 게시/구독에서 클라이언트 디바이스로 릴레이합니다.

또한 Greengrass 구성 요소의 클라이언트 디바이스 새도와 상호 작용할 수 있습니다. 자세한 내용은 [클라이언트 디바이스 새도와 상호 작용 및 동기화](#) 단원을 참조하십시오.

주제

- [MQTT 브리지 구성 요소 구성 및 배포](#)
- [클라이언트 디바이스에서 MQTT 메시지 수신](#)
- [클라이언트 디바이스로 MQTT 메시지 전송](#)

MQTT 브리지 구성 요소 구성 및 배포

MQTT 브리지 구성 요소는 각각 메시지 소스와 메시지 대상을 지정하는 주제 매핑의 목록을 사용합니다. 클라이언트 디바이스와 통신하려면 MQTT 브리지 구성 요소를 배포하고 구성 요소 구성에서 각 소스 및 대상 주제를 지정합니다.

코어 디바이스 또는 코어 디바이스 그룹에 MQTT 브리지 구성 요소를 배포하려면 `aws.greengrass.clientdevices.mqtt.Bridge` 구성 요소가 포함된 [배포를 생성](#)합니다. 배포의 MQTT 브리지 구성 요소 구성에서 주제 매핑(`mqttTopicMapping`)을 지정합니다.

다음 예제에서는 클라이언트 디바이스에서 로컬 게시/구독 브로커로 `clients/MyClientDevice1/hello/world` 주제를 릴레이하도록 MQTT 브리지 구성 요소를 구성하는 배포를 정의합니다. `merge` 구성 업데이트에는 직렬화된 JSON 객체가 필요합니다. 자세한 내용은 [구성 요소 구성 업데이트](#) 단원을 참조하십시오.

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 사용하면 소스 브로커가 로컬 게시/구독인 경우 주제 필터에서 MQTT 주제 와일드카드를 사용할 수 있습니다. 자세한 내용은 [MQTT 브리지 구성 요소 구성](#)을 참조하세요.

클라이언트 디바이스에서 MQTT 메시지 수신

MQTT 브리지 구성 요소가 클라이언트 디바이스의 메시지를 수신하도록 구성하는 로컬 게시/구독 주제를 구독할 수 있습니다.

사용자 지정 구성 요소의 클라이언트 디바이스에서 MQTT 메시지를 수신하려면

1. 클라이언트 디바이스가 게시하는 MQTT 주제의 메시지를 로컬 게시/구독 주제로 릴레이하도록 [MQTT 브리지 구성 요소를 구성하고 배포](#)합니다.
2. 로컬 게시/구독 IPC 인터페이스를 사용하여 MQTT 브리지가 메시지를 릴레이하는 주제를 구독합니다. 자세한 내용은 [로컬 메시지 게시/구독](#) 및 [SubscribeToTopic](#) 섹션을 참조하세요.

[클라이언트 디바이스 연결 및 테스트 자습서](#)에는 클라이언트 디바이스의 메시지를 구독하는 구성 요소를 개발하는 섹션이 포함되어 있습니다. 자세한 내용은 [4단계: 클라이언트 디바이스와 통신하는 구성 요소를 개발합니다](#). 단원을 참조하십시오.

클라이언트 디바이스로 MQTT 메시지 전송

MQTT 브리지 구성 요소가 클라이언트 디바이스로 메시지를 보내도록 구성하는 로컬 게시/구독 주제에 게시할 수 있습니다.

사용자 지정 구성 요소의 클라이언트 디바이스에 MQTT 메시지를 게시하려면

1. 로컬 게시/구독 주제의 메시지를 클라이언트 디바이스가 구독하는 MQTT 주제로 릴레이하도록 [MQTT 브리지 구성 요소를 구성하고 배포](#)합니다.
2. 로컬 게시/구독 IPC 인터페이스를 사용하여 MQTT 브리지가 메시지를 릴레이하는 주제에 게시합니다. 자세한 내용은 [로컬 메시지 게시/구독](#) 및 [PublishToTopic](#)을(를) 참조하세요.

클라이언트 디바이스 새도와 상호 작용 및 동기화

[새도 관리자 구성 요소](#)를 사용하여 클라이언트 디바이스 새도를 포함한 로컬 새도를 관리할 수 있습니다. 새도 관리자를 사용하여 다음을 수행할 수 있습니다.

- Greengrass 구성 요소의 클라이언트 디바이스 새도와 상호 작용합니다.

- 클라이언트 디바이스 새도우를와 동기화합니다 AWS IoT Core.

Note

새도 관리자 구성 요소는 AWS IoT Core 기본적으로 새도우를와 동기화하지 않습니다. 동기화할 클라이언트 디바이스 새도를 지정하려면 새도 관리자 구성 요소를 구성해야 합니다.

주제

- [사전 조건](#)
- [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정](#)
- [구성 요소의 클라이언트 디바이스 새도와 상호 작용](#)
- [클라이언트 디바이스 새도우를와 동기화 AWS IoT Core](#)

사전 조건

클라이언트 디바이스 새도와 상호 작용하고 클라이언트 디바이스 새도와 동기화하려면 코어 디바이스 AWS IoT Core가 다음 요구 사항을 충족해야 합니다.

- 코어 디바이스는 [클라이언트 디바이스 지원을 위한 Greengrass 구성 요소](#) 외에도 다음 구성 요소를 실행해야 합니다.
 - [Greengrass nucleus](#) v2.6.0 이상
 - [새도 관리자](#) v2.2.0 이상
 - [MQTT 브리지](#) v2.2.0 이상
- [클라이언트 디바이스 인증](#) 구성 요소는 클라이언트 디바이스가 [디바이스 새도 주제](#)에서 통신할 수 있도록 구성해야 합니다.

새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정

기본적으로 새도 관리자 구성 요소는 클라이언트 디바이스 새도를 관리하지 않습니다. 이 기능을 활성화하려면 클라이언트 디바이스와 새도 관리자 구성 요소 간에 MQTT 메시지를 전달해야 합니다. 클라이언트 디바이스는 MQTT 메시지를 사용하여 디바이스 새도 업데이트를 수신하고 전송합니다. 새도 관리자 구성 요소는 로컬 Greengrass 게시/구독 인터페이스를 구독하므로 [디바이스 새도 주제](#)에 대한 MQTT 메시지를 전달하도록 [MQTT 브리지 구성 요소](#)를 구성할 수 있습니다.

MQTT 브리지 구성 요소는 각각 메시지 소스와 메시지 대상을 지정하는 주제 매핑의 목록을 사용합니다. 새도 관리자 구성 요소가 클라이언트 디바이스 새도를 관리할 수 있도록 하려면 MQTT 브리지 구성 요소를 배포하고 클라이언트 디바이스 새도에 대한 새도 주제를 지정합니다. 로컬 MQTT와 로컬 게시/구독 간에 양방향으로 메시지를 전달하도록 브리지를 구성해야 합니다.

코어 디바이스 또는 코어 디바이스 그룹에 MQTT 브리지 구성 요소를 배포하려면 `aws.greengrass.clientdevices.mqtt.Bridge` 구성 요소가 포함된 [배포를 생성](#)합니다. 배포의 MQTT 브리지 구성 요소 구성에서 주제 매핑(`mqttTopicMapping`)을 지정합니다.

다음 예제를 사용하여 클라이언트 디바이스와 새도 관리자 구성 요소 간의 통신을 활성화하도록 MQTT 브리지 구성 요소를 구성합니다.

Note

AWS IoT Greengrass 콘솔에서 이러한 구성 예제를 사용할 수 있습니다. AWS IoT Greengrass API를 사용하는 경우 merge 구성 업데이트에는 직렬화된 JSON 객체가 필요하므로 다음 JSON 객체를 문자열로 직렬화해야 합니다. 자세한 내용은 [구성 요소 구성 업데이트](#) 단원을 참조하십시오.

Example예: 모든 클라이언트 디바이스 새도 관리

다음 MQTT 브리지 구성 예제를 사용하면 새도 관리자가 모든 클라이언트 디바이스에 대한 모든 새도를 관리할 수 있습니다.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```


Example예: 클라이언트 디바이스에 대한 새도 관리

다음 MQTT 브리지 구성 예제를 사용하면 새도 관리자가 MyClientDevice라는 이름의 클라이언트 디바이스에 대한 모든 새도를 관리할 수 있습니다.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example예: 모든 클라이언트 디바이스에 대해 이름이 지정된 새도 관리

다음 MQTT 브리지 구성 예제를 사용하면 새도 관리자가 모든 클라이언트 디바이스에 대해 DeviceConfiguration이라는 이름의 새도를 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"
    }
  }
}
```

Example예: 모든 클라이언트 디바이스의 이름이 지정되지 않은 새도 관리

다음 MQTT 브리지 구성 예제를 사용하면 새도 관리자가 모든 클라이언트 디바이스에 대해 이름이 지정되지 않은 새도를 관리할 수 있지만 이름이 지정된 새도는 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 디바이스 새도와 상호 작용

로컬 새도 서비스를 사용하여 클라이언트 디바이스의 로컬 새도 문서를 읽고 수정하는 사용자 지정 구성 요소를 개발할 수 있습니다. 자세한 내용은 [구성 요소에서 새도와 연동](#) 단원을 참조하십시오.

클라이언트 디바이스 새도우들과 동기화 AWS IoT Core

로컬 클라이언트 디바이스 새도 상태를 동기화하도록 새도 관리자 구성 요소를 구성할 수 있습니다 AWS IoT Core. 자세한 내용은 [로컬 디바이스 새도우들과 동기화 AWS IoT Core](#) 단원을 참조하십시오.

로컬 메시징에 IPv6 사용

IPv6를 사용하여 로컬 메시지를 전송하도록 IP 감지기 구성 요소를 구성할 수 있습니다.

Note

IPv6를 사용하여 로컬 메시지를 전송하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

[IP 감지기 구성 요소](#)를 배포하여 IPv6 주소를 감지하고 사용할 수 있습니다. IPv4 대신 IPv6를 사용하려면 IP 감지기 구성 요소의 구성을 업데이트해야 합니다. 자세한 내용은 [IP 감지기를 사용하여 엔드포인트 자동 관리](#) 단원을 참조하십시오.

주제

- [IPv6를 사용하도록 IP 감지기 구성](#)

IPv6를 사용하도록 IP 감지기 구성

클라이언트 디바이스가 코어 디바이스와 동일한 네트워크에 있는 것처럼 네트워크 설정이 간단한 경우에는 [IP 감지기 구성 요소](#)를 배포하여 로컬 메시징에 IPv6를 사용할 수 있습니다.

IP 감지기 구성 요소는 수동으로 설정한 엔드포인트를 덮어씁니다.

Important

코어 디바이스의 AWS IoT 정책은 IP 감지기 구성 요소를 사용할 수 있는 `greengrass:UpdateConnectivityInfo` 권한을 허용해야 합니다. 자세한 내용은 [AWS IoT 데이터 영역 작업에 대한 정책](#) 및 [AWS IoT 사물 정책 구성](#) 섹션을 참조하세요.

다음 중 하나를 수행하여 IP 감지기 구성 요소를 배포할 수 있습니다.

- 콘솔의 검색 구성 페이지를 사용합니다. 자세한 내용은 [클라우드 검색 구성\(콘솔\)](#) 단원을 참조하십시오.
- IP 감지기를 포함하도록 배포를 생성 및 수정합니다. 콘솔, AWS CLI 또는 AWS API를 사용하여 배포를 관리할 수 있습니다. 자세한 내용은 [배포 만들기](#) 단원을 참조하십시오.

IP 감지기 구성 요소 배포(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 퍼블릭 구성 요소 탭을 선택한 다음 `aws.greengrass.clientdevices.IPDetector`를 선택합니다.
3. `aws.greengrass.clientdevices.IPDetector` 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포 생성을 선택하고 다음을 선택합니다.
5. 새 배포 생성을 선택한 경우 배포의 대상 코어 디바이스 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 디바이스 또는 사물 그룹을 선택하고 다음을 선택합니다.
6. 구성 요소 선택 페이지에서 `aws.greengrass.clientdevices.IPDetector` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 `aws.greengrass.clientdevices.IPDetector`를 선택하고 다음을 수행합니다.
 - a. 구성 요소 구성을 선택합니다.
 - b. `aws.greengrass.clientdevices.IPDetector` 구성 모달의 구성 업데이트 아래의 병합할 구성에서 구성 업데이트를 입력하여 IP 감지기 구성 요소를 구성할 수 있습니다. 다음 구성 옵션 중 하나를 지정할 수 있습니다. 이 경우 `includeIPv4Addrs`을 `false`로, `includeIPv6Addrs`를 `true`으로 설정합니다. 그런 다음 다른 IPv6 구성 옵션을 업데이트할 수 있습니다.
 - `defaultPort` – (선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다.
 - `includeIPv4LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.

- `includeIPv4LinkLocalAddr`s – (선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.
- `includeIPv6LoopbackAddr`s – (선택 사항) 이 옵션을 활성화하여 IPv6 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addr`s를 `false`, `includeIPv6Addr`s를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv6LinkLocalAddr`s – (선택 사항) 이 옵션을 활성화하여 IPv6 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addr`s를 `false`, `includeIPv6Addr`s를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv4Addr`s – (선택 사항) 기본값은 `True`로 설정됩니다. 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv4 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv6Addr`s – (선택 사항) 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv6 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 `includeIPv4Addr`s를 `false`로 설정합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

구성 업데이트는 다음 예제와 유사할 수 있습니다.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddr": false,
  "includeIPv4LinkLocalAddr": false,
  "includeIPv6LoopbackAddr": true,
  "includeIPv6LinkLocalAddr": true,
  "includeIPv4Addr": false,
  "includeIPv6Addr": true
}
```

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 1분 정도 걸릴 수 있습니다.

IP 감지기 구성 요소 배포(AWS CLI)

IP 감지기 구성 요소를 배포하려면 components 객체에

`aws.greengrass.clientdevices.IPDetector`를 포함하는 배포 문서를 생성하고 구성 요소에 대한 구성 업데이트를 지정합니다. [배포 만들기](#)의 지침에 따라 새 배포를 생성하거나 기존 배포를 수정합니다.

배포 문서를 생성하는 경우 다음 옵션 중 하나를 지정하여 IP 감지기 구성 요소를 구성할 수 있습니다.

- `defaultPort` – (선택 사항) 이 구성 요소가 IP 주소를 감지할 때 보고할 MQTT 브로커 포트입니다. 기본 포트 8883이 아닌 다른 포트를 사용하도록 MQTT 브로커를 구성하는 경우 이 파라미터를 지정해야 합니다.
- `includeIPv4LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다.
- `includeIPv4LinkLocalAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv4 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다.
- `includeIPv6LoopbackAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv6 루프백 주소를 감지하고 보고할 수 있습니다. 이들은 디바이스가 자체적으로 통신할 수 있는 IP 주소(예: `localhost`)입니다. 코어 디바이스와 클라이언트 디바이스가 동일한 시스템에서 실행되는 테스트 환경에서 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`, `includeIPv6Addrs`를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv6LinkLocalAddrs` – (선택 사항) 이 옵션을 활성화하여 IPv6 [링크-로컬 주소](#)를 감지하고 보고할 수 있습니다. 코어 디바이스의 네트워크에 DHCP(Dynamic Host Configuration Protocol) 또는 정적 할당 IP 주소가 없는 경우 이 옵션을 사용합니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`, `includeIPv6Addrs`를 `true`로 설정해야 합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.
- `includeIPv4Addrs` – (선택 사항) 기본값은 `True`로 설정됩니다. 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv4 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

- `includeIPv6Addrs` – (선택 사항) 이 옵션을 활성화하여 코어 디바이스에서 찾은 IPv6 주소를 게시할 수 있습니다. 이 옵션을 사용하려면 `includeIPv4Addrs`를 `false`로 설정합니다. 이 옵션을 사용하려면 IP 감지기 v2.2.0 이상이 있어야 합니다.

다음 예제 부분 배포 문서는 IPv6를 사용하도록 지정합니다.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

클라이언트 디바이스 문제 해결

이 섹션의 문제 해결 정보 및 솔루션을 사용하여 Greengrass 클라이언트 디바이스 및 클라이언트 디바이스 구성 요소와 관련된 문제를 해결하세요.

주제

- [Greengrass 검색 문제](#)
- [MQTT 연결 문제](#)

Greengrass 검색 문제

다음 정보를 활용하여 Greengrass 검색 관련 문제를 해결하세요. 이러한 문제는 클라이언트 디바이스가 연결할 수 있는 Greengrass 코어 디바이스를 식별하기 위해 [Greengrass 검색 API](#)를 사용할 때 발생할 수 있습니다.

주제

- [Greengrass 검색 문제\(HTTP API\)](#)
- [Greengrass 검색 문제\(Python용AWS IoT Device SDK v2\)](#)

- [Greengrass 검색 문제\(C++의 경우AWS IoT Device SDK v2\)](#)
- [Greengrass 검색 문제\(JavaScript용AWS IoT Device SDK v2\)](#)
- [Greengrass 검색 문제\(Java용AWS IoT Device SDK v2\)](#)

Greengrass 검색 문제(HTTP API)

다음 정보를 활용하여 Greengrass 검색 관련 문제를 해결하세요. [cURL로 검색 API를 테스트](#)하면 이 오류가 발생할 수 있습니다.

주제

- [curl: \(52\) Empty reply from server](#)
- [HTTP 403: {"message":null,"traceld":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}](#)
- [HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}](#)

curl: (52) Empty reply from server

요청에 비활성 AWS IoT 인증서를 지정하면 이 오류가 표시될 수 있습니다.

클라이언트 디바이스에 연결된 인증서가 있고 해당 인증서가 활성화되어 있는지 확인합니다. 자세한 내용은 AWS IoT Core 개발자 가이드에서 [클라이언트 인증서에 사물 또는 정책 연결](#)과 [클라이언트 인증서 활성화 또는 비활성화](#)를 참조하세요.

HTTP 403: {"message":null,"traceld":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

클라이언트 디바이스에 greengrass:Discover를 직접 호출할 수 있는 권한이 없는 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스의 인증서에 greengrass:Discover를 허용하는 정책이 있는지 확인합니다. 권한에는 Resource 섹션의 [사물 정책 변수](#)(iot:Connection.Thing.*)를 사용할 수 없습니다. 자세한 내용은 [Discovery 인증 및 권한 부여](#) 섹션을 참조하세요.

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

다음 경우에 이 오류가 발생할 수 있습니다.

- 클라이언트 디바이스는 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에 연결되어 있지 않습니다.

- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 중 클라이언트 [디바이스 인증 구성 요소](#)를 실행하는 디바이스는 없습니다.

클라이언트 디바이스가 연결하려는 코어 디바이스에 연결되어 있는지 확인합니다. 그런 다음 코어 디바이스가 [클라이언트 디바이스 인증 구성 요소](#)를 실행하고 MQTT 브로커 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라우드 검색 구성\(콘솔\)](#)

Greengrass 검색 문제(Python용AWS IoT Device SDK v2)

다음 정보를 활용하여 [Python용AWS IoT Device SDK v2](#)에서 Greengrass 검색 관련 문제를 해결하세요.

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.

요청에 비활성 AWS IoT 인증서를 지정하면 오류가 표시될 수 있습니다.

클라이언트 디바이스에 연결된 인증서가 있고 해당 인증서가 활성화되어 있는지 확인합니다. 자세한 내용은 AWS IoT Core 개발자 가이드에서 [클라이언트 인증서에 사물 또는 정책 연결](#)과 [클라이언트 인증서 활성화 또는 비활성화](#)를 참조하세요.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

클라이언트 디바이스에 greengrass:Discover를 직접 호출할 수 있는 권한이 없는 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스의 인증서에 greengrass:Discover를 허용하는 정책이 있는지 확인합니다. 이 권한에는 Resource 섹션의 [사물 정책 변수](#)(iot:Connection.Thing.*)를 사용할 수 없습니다. 자세한 내용은 [Discovery 인증 및 권한 부여](#) 섹션을 참조하세요.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

다음 경우에 이 오류가 발생할 수 있습니다.

- 클라이언트 디바이스는 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에 연결되어 있지 않습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 중 클라이언트 [디바이스 인증 구성 요소](#)를 실행하는 디바이스는 없습니다.

클라이언트 디바이스가 연결하려는 코어 디바이스에 연결되어 있는지 확인합니다. 그런 다음 코어 디바이스가 [클라이언트 디바이스 인증 구성 요소](#)를 실행하고 MQTT 브로커 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라우드 검색 구성\(콘솔\)](#)

Greengrass 검색 문제(C++의 경우AWS IoT Device SDK v2)

다음 정보를 활용하여 [AWS IoT Device SDK v2 for C++](#)에서 Greengrass 검색 관련 문제를 해결하세요.

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.

요청에 비활성 AWS IoT 인증서를 지정하면 오류가 표시될 수 있습니다.

클라이언트 디바이스에 연결된 인증서가 있고 해당 인증서가 활성화되어 있는지 확인합니다. 자세한 내용은 AWS IoT Core 개발자 가이드에서 [클라이언트 인증서에 사물 또는 정책 연결](#)과 [클라이언트 인증서 활성화 또는 비활성화](#)를 참조하세요.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

클라이언트 디바이스에 greengrass:Discover를 직접 호출할 수 있는 권한이 없는 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스의 인증서에 greengrass:Discover를 허용하는 정책이 있는지 확인합니다. 이 권한에는 Resource 섹션의 [사물 정책 변수](#)(iot:Connection.Thing.*)를 사용할 수 없습니다. 자세한 내용은 [Discovery 인증 및 권한 부여](#) 섹션을 참조하세요.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

다음 경우에 이 오류가 발생할 수 있습니다.

- 클라이언트 디바이스는 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에 연결되어 있지 않습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 중 클라이언트 [디바이스 인증 구성 요소](#)를 실행하는 디바이스는 없습니다.

클라이언트 디바이스가 연결하려는 코어 디바이스에 연결되어 있는지 확인합니다. 그런 다음 코어 디바이스가 [클라이언트 디바이스 인증 구성 요소](#)를 실행하고 MQTT 브로커 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라우드 검색 구성\(콘솔\)](#)

Greengrass 검색 문제(JavaScript용AWS IoT Device SDK v2)

다음 정보를 활용하여 [JavaScript용AWS IoT Device SDK v2](#)에서 Greengrass 검색 관련 문제를 해결하세요.

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.

요청에 비활성 AWS IoT 인증서를 지정하면이 오류가 표시될 수 있습니다.

클라이언트 디바이스에 연결된 인증서가 있고 해당 인증서가 활성화되어 있는지 확인합니다. 자세한 내용은 AWS IoT Core 개발자 가이드에서 [클라이언트 인증서에 사물 또는 정책 연결](#)과 [클라이언트 인증서 활성화 또는 비활성화](#)를 참조하세요.

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

클라이언트 디바이스에 greengrass:Discover를 직접 호출할 수 있는 권한이 없는 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스의 인증서에 greengrass:Discover를 허용하는 정책이 있는지 확인합니다. 이 권한에는 Resource 섹션의 [사물 정책 변수](#)(iot:Connection.Thing.*)를 사용할 수 없습니다. 자세한 내용은 [Discovery 인증 및 권한 부여](#) 섹션을 참조하세요.

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

다음 경우에 이 오류가 발생할 수 있습니다.

- 클라이언트 디바이스는 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에 연결되어 있지 않습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.

- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 중 클라이언트 [디바이스 인증 구성 요소](#)를 실행하는 디바이스는 없습니다.

클라이언트 디바이스가 연결하려는 코어 디바이스에 연결되어 있는지 확인합니다. 그런 다음 코어 디바이스가 [클라이언트 디바이스 인증 구성 요소](#)를 실행하고 MQTT 브로커 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라우드 검색 구성\(콘솔\)](#)

Error: Discovery failed (headers: [object Object])

Greengrass 검색 샘플을 실행할 때 HTTP 응답 코드 없이 이 오류가 발생할 수 있습니다. 해당 오류는 여러 가지 이유로 발생할 수 있습니다.

- 클라이언트 디바이스에 greengrass:Discover를 직접 호출할 수 있는 권한이 없는 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스의 인증서에 greengrass:Discover를 허용하는 정책이 있는지 확인합니다. 이 권한에는 Resource 섹션의 [사물 정책 변수](#)(iot:Connection.Thing.*)를 사용할 수 없습니다. 자세한 내용은 [Discovery 인증 및 권한 부여](#) 섹션을 참조하세요.

- 다음 경우에 이 오류가 발생할 수 있습니다.
 - 클라이언트 디바이스는 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에 연결되어 있지 않습니다.
 - 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.
 - 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 중 클라이언트 [디바이스 인증 구성 요소](#)를 실행하는 디바이스는 없습니다.

클라이언트 디바이스가 연결하려는 코어 디바이스에 연결되어 있는지 확인합니다. 그런 다음 코어 디바이스가 [클라이언트 디바이스 인증 구성 요소](#)를 실행하고 MQTT 브로커 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라우드 검색 구성\(콘솔\)](#)

Greengrass 검색 문제(Java용AWS IoT Device SDK v2)

다음 정보를 활용하여 [Java용AWS IoT Device SDK v2](#)에서 Greengrass 검색 관련 문제를 해결하세요.

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.)

요청에 비활성 AWS IoT 인증서를 지정하면이 오류가 표시될 수 있습니다.

클라이언트 디바이스에 연결된 인증서가 있고 해당 인증서가 활성화되어 있는지 확인합니다. 자세한 내용은 AWS IoT Core 개발자 가이드에서 [클라이언트 인증서에 사물 또는 정책 연결](#)과 [클라이언트 인증서 활성화 또는 비활성화](#)를 참조하세요.

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

클라이언트 디바이스에 greengrass:Discover를 직접 호출할 수 있는 권한이 없는 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스의 인증서에 greengrass:Discover를 허용하는 정책이 있는지 확인합니다. 이 권한에는 Resource 섹션의 [사물 정책 변수](#)(iot:Connection.Thing.*)를 사용할 수 없습니다. 자세한 내용은 [Discovery 인증 및 권한 부여](#) 섹션을 참조하세요.

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

다음 경우에 이 오류가 발생할 수 있습니다.

- 클라이언트 디바이스는 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에 연결되어 있지 않습니다.
- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 또는 AWS IoT Greengrass V1 그룹에는 MQTT 브로커 엔드포인트가 없습니다.

- 클라이언트 디바이스의 연결된 Greengrass 코어 디바이스 중 클라이언트 [디바이스 인증 구성 요소](#)를 실행하는 디바이스는 없습니다.

클라이언트 디바이스가 연결하려는 코어 디바이스에 연결되어 있는지 확인합니다. 그런 다음 코어 디바이스가 [클라이언트 디바이스 인증 구성 요소](#)를 실행하고 MQTT 브로커 엔드포인트가 하나 이상 있는지 확인합니다. 자세한 내용은 다음을 참조하세요.

- [클라이언트 디바이스 연결](#)
- [코어 디바이스 엔드포인트 관리](#)
- [클라우드 검색 구성\(콘솔\)](#)

MQTT 연결 문제

다음 정보를 활용하여 클라이언트 디바이스 MQTT 연결 관련 문제를 해결하세요. 이러한 문제는 클라이언트 디바이스가 MQTT를 통해 코어 디바이스에 연결하려고 시도할 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 디바이스가 권한이 없는 MQTT 주제를 구독하려고 할 때 Greengrass 로그에 이 오류가 발생할 수 있습니다. 오류 메시지에는 주제가 포함됩니다.

[클라이언트 디바이스 인증 구성 요소](#)의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 주제에 대한 mqtt:subscribe 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

MQTT 연결 문제(Python)

다음 정보를 활용하여 [Python용AWS IoT Device SDK v2](#)를 사용하는 경우 클라이언트 디바이스 MQTT 연결 관련 문제를 해결하세요.

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 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 mqtt:connect 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

[클라이언트 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 `mqtt:connect` 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

MQTT 연결 문제(C++)

다음 정보를 활용하여 [AWS IoT Device SDK v2 for C++](#)를 사용하는 경우 클라이언트 디바이스 MQTT 연결 관련 문제를 해결하세요.

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 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 `mqtt:connect` 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

[클라이언트 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 mqtt:connect 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

MQTT 연결 문제(Java)

다음 정보를 활용하여 [Java용AWS IoT Device SDK v2](#)를 사용하는 경우 클라이언트 디바이스 MQTT 연결 관련 문제를 해결하세요.

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 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 mqtt:connect 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

[클라이언트 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 mqtt:connect 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

MQTT 연결 문제(JavaScript)

다음 정보를 활용하여 [JavaScript용AWS IoT Device SDK v2](#)를 사용하는 경우 클라이언트 디바이스 MQTT 연결 관련 문제를 해결하세요.

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 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 mqtt:connect 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

[클라이언트 디바이스 인증 구성 요소](#)가 클라이언트 디바이스에 연결 권한을 부여하는 클라이언트 디바이스 권한 부여 정책을 정의하지 않은 경우 이 오류가 발생할 수 있습니다.

클라이언트 디바이스 인증 구성 요소의 구성에 다음이 포함되어 있는지 확인합니다.

- 클라이언트 디바이스와 일치하는 디바이스 그룹입니다.
- 클라이언트 디바이스에 대한 mqtt:connect 권한을 부여하는 해당 디바이스 그룹에 대한 클라이언트 디바이스 권한 부여 정책입니다.

클라이언트 디바이스 인증 구성 요소를 배포하고 구성하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- [클라우드 검색 구성\(콘솔\)](#)
- [클라이언트 디바이스 인증](#)
- [배포 만들기](#)

디바이스 새도와 연동

Greengrass 코어 디바이스는 구성 요소를 사용하여 [AWS IoT 디바이스 새도](#)와 연동할 수 있습니다. 새도는 AWS IoT 사물에 대한 현재 또는 원하는 상태 정보가 저장되는 JSON 문서입니다. 디바이스가 AWS IoT에 연결되어 있는지 여부에 관계없이 새도를 통해 디바이스의 상태를 다른 AWS IoT Greengrass 구성 요소에 제공할 수 있습니다. 각 AWS IoT 디바이스에는 자체의 명명되지 않은 클래식 새도가 있습니다. 또한 각 디바이스에 대해 여러 개의 이름 지정된 새도를 생성할 수도 있습니다.

디바이스 및 서비스는 MQTT 및 [예약된 MQTT 새도 주제](#), [디바이스 새도 REST API](#)를 사용한 HTTP 및 [AWS IoT의 AWS CLI](#)를 사용하여 클라우드 새도를 생성, 업데이트 및 삭제할 수 있습니다.

[새도 관리자](#) 구성 요소를 사용하면 Greengrass 구성 요소가 [로컬 새도 서비스](#)와 로컬 게시/구독 새도 주제를 사용하여 로컬 새도를 생성, 업데이트 및 삭제할 수 있습니다. 또한 새도 관리자는 코어 디바이스에서 이러한 로컬 새도 문서의 스토리지를 관리하고 클라우드 새도와 새도 상태 정보 동기화를 처리합니다.

새도 관리자 구성 요소를 사용하여 코어 디바이스에 연결하는 [클라이언트 디바이스](#)의 로컬 새도를 관리할 수도 있습니다. 새도 관리자가 클라이언트 디바이스 새도를 관리할 수 있도록 하려면 로컬 MQTT 브로커와 로컬 게시/구독 서비스 간에 메시지를 릴레이하도록 [MQTT 브리지 구성 요소](#)를 구성합니다. 자세한 내용은 [클라이언트 디바이스 새도와 상호 작용 및 동기화](#) 섹션을 참조하세요.

AWS IoT 디바이스 새도에 대한 자세한 내용은 AWS IoT 개발자 가이드의 [AWS IoT Device Shadow service](#)를 참조하세요.

주제

- [구성 요소에서 새도와 연동](#)
- [로컬 디바이스 새도우들과 동기화 AWS IoT Core](#)

구성 요소에서 새도와 연동

로컬 새도 서비스를 사용하여 로컬 새도 문서 및 클라이언트 디바이스 새도 문서를 읽고 수정하는 Lambda 함수 구성 요소를 포함하는 사용자 지정 구성 요소를 개발할 수 있습니다.

사용자 지정 구성 요소는 AWS IoT Greengrass 코어 IPC 라이브러리를 사용하여 로컬 새도우 서비스와 상호 작용합니다 AWS IoT Device SDK. [새도 관리자](#) 구성 요소는 코어 디바이스에서 로컬 새도 서비스를 활성화합니다.

Greengrass 코어 디바이스에 새도 관리자 구성 요소를 배포하려면 `aws.greengrass.ShadowManager` 구성 요소가 포함된 [배포를 생성](#)합니다.

Note

기본적으로 새도 관리자 구성 요소를 배포하면 로컬 새도 작업만 활성화됩니다. AWS IoT Greengrass 가 코어 디바이스 새도우에 대한 새도 상태 정보 또는 클라이언트 디바이스에 대한 새도우를의 해당 클라우드 새도 문서에 동기화하도록 하려면 `synchronize` 파라미터가 포함된 새도 관리자 구성 요소에 대한 구성 업데이트를 생성 AWS IoT Core해야 합니다. 자세한 내용은 [로컬 디바이스 새도우들과 동기화 AWS IoT Core](#) 단원을 참조하십시오.

주제

- [새도 상태 검색 및 수정](#)
- [새도 상태 변경에 대응](#)

새도 상태 검색 및 수정

새도 IPC 작업은 로컬 새도 문서에서 상태 정보를 검색하고 업데이트합니다. 새도 관리자 구성 요소는 코어 디바이스에서 이러한 새도 문서의 스토리지를 처리합니다.

로컬 새도 상태를 수정하려면

1. 구성 요소가 로컬 새도 주제에 대한 메시지를 수신할 수 있도록 허용하는 권한 부여 정책을 사용자 지정 구성 요소의 레시피에 추가합니다.

권한 부여 정책의 예는 [로컬 새도 IPC 권한 부여 정책 예제](#)를 참조하세요.

2. 새도 IPC 작업을 사용하여 새도 상태 정보를 검색하고 수정합니다. 구성 요소 코드에서 새도 IPC 작업을 사용하는 방법에 대한 자세한 내용은 [로컬 새도와 연동](#) 섹션을 참조하세요.

Note

코어 디바이스가 클라이언트 디바이스 새도와 연동할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정](#)을 참조하세요.

새도 상태 변경에 대응

Greengrass 구성 요소는 로컬 게시/구독 인터페이스를 사용하여 코어 디바이스에서 통신합니다. 사용자 지정 구성 요소가 새도 상태 변경에 대응할 수 있도록 하려면 로컬 게시/구독 주제를 구독할 수 있습니다. 이렇게 하면 구성 요소가 로컬 새도 주제에 대한 메시지를 수신한 다음 해당 메시지에 대한 작업을 수행할 수 있습니다.

로컬 새도 주제는 AWS IoT 디바이스 새도 MQTT 주제와 동일한 형식을 사용합니다. 자세한 내용은 AWS IoT 개발자 안내서의 [Device Shadow MQTT topics](#)를 참조하세요.

로컬 새도 상태 변경에 대응하려면

1. 구성 요소가 로컬 새도 주제에 대한 메시지를 수신할 수 있도록 허용하는 액세스 제어 정책을 사용자 지정 구성 요소의 레시피에 추가합니다.

권한 부여 정책의 예는 [로컬 새도 IPC 권한 부여 정책 예제](#)를 참조하세요.

2. 구성 요소에서 사용자 지정 작업을 시작하려면 SubscribeToTopic IPC 작업을 사용하여 메시지를 수신하려는 새도 주제를 구독합니다. 구성 요소 코드에서 로컬 게시/구독 IPC 작업을 사용하는 방법에 대한 자세한 내용은 [로컬 메시지 게시/구독](#) 섹션을 참조하세요.
3. Lambda 함수를 호출하려면 이벤트 소스 구성을 사용하여 새도 주제의 이름을 제공하고 로컬 게시/구독 주제를 지정합니다. Lambda 함수 구성 요소 생성에 대한 자세한 내용은 [AWS Lambda 함수 실행](#) 섹션을 참조하세요.

Note

코어 디바이스가 클라이언트 디바이스 새도와 연동할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정](#)을 참조하세요.

로컬 디바이스 새도우를 동기화 AWS IoT Core

새도 관리자 구성 요소를 사용하면 로컬 디바이스 새도 상태들과 동기화 AWS IoT Greengrass 할 수 있습니다 AWS IoT Core. 구성 파라미터를 포함하도록 새도 관리자 구성 요소의 synchronization 구성을 수정하고 디바이스의 AWS IoT 사물 이름과 동기화하려는 새도우를 지정해야 합니다.

새도 관리자를 구성하여 새도를 동기화하면 로컬 새도 문서나 클라우드 새도 문서에서 변경 사항이 발생하는지 여부에 관계없이 지정된 새도의 모든 상태 변경 사항이 동기화됩니다.

새도 관리자 구성 요소가 새도를 실시간으로 동기화할지 아니면 주기적으로 동기화할지 지정할 수도 있습니다. 기본적으로 새도 관리자 구성 요소는 새도우를 실시간으로 동기화하므로 코어 디바이스는 각 업데이트가 발생할 AWS IoT Core 때 새도우 업데이트를 주고 받습니다. 주기적인 간격을 구성하여 대역폭 사용량과 요금을 줄일 수 있습니다.

주제

- [사전 조건](#)
- [새도 관리자 구성 요소 구성](#)
- [로컬 새도 동기화](#)
- [새도 병합 충돌 동작](#)

사전 조건

로컬 새도와 동기화하려면 다음 AWS IoT Core 새도 정책 작업을 허용하도록 Greengrass 코어 디바이스의 AWS IoT 정책을 구성 AWS IoT Core해야 합니다.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

자세한 내용은 다음을 참조하세요.

- AWS IoT 개발자 안내서의 [AWS IoT Core 정책 작업](#)
- [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#)
- [코어 디바이스의 AWS IoT 정책 업데이트](#)

새도 관리자 구성 요소 구성

새도 관리자에게는 로컬 새도 문서의 새도 상태 정보를 AWS IoT Core의 클라우드 새도 문서와 동기화하기 위한 새도 이름 매핑의 목록이 필요합니다.

새도 상태를 동기화하려면 `aws.greengrass.ShadowManager` 구성 요소를 포함하는 [배포를 생성](#) [설](#)하고 배포의 새도 관리자 구성의 `synchronize` 구성 파라미터에서 동기화할 새도를 지정합니다.

Note

코어 디바이스가 클라이언트 디바이스 새도와 연동할 수 있도록 하려면 MQTT 브리지 구성 요소도 구성하고 배포해야 합니다. 자세한 내용은 [새도 관리자가 클라이언트 디바이스와 통신할 수 있도록 설정](#)을 참조하세요.

다음 예제 구성 업데이트는 새도 관리자 구성 요소가 다음 새도와 동기화하도록 지시합니다. AWS IoT Core

- 코어 디바이스에 대한 클래식 새도
- 코어 디바이스에 대해 이름이 지정된 MyCoreShadow
- MyDevice2라는 이름의 IoT 사물의 클래식 새도
- MyDevice1이라는 이름의 IoT 사물에 대해 이름이 지정된 새도 MyShadowA 및 MyShadowB

이 구성 업데이트는 새도우를 AWS IoT Core 와 실시간으로 동기화하도록 지정합니다. 새도 관리자 v2.1.0 이상을 사용하는 경우 주기적으로 새도를 동기화하도록 새도 관리자 구성 요소를 구성할 수 있습니다. 이 기능을 구성하려면 동기화 전략을 `periodic`으로 변경하고 간격을 `delay(초)`로 지정합니다. 자세한 내용은 새도 관리자 구성 요소의 [전략 구성 파라미터](#)를 참조하세요.

이 구성 업데이트는 AWS IoT Core 와 코어 디바이스 간에 양방향으로 새도를 동기화하도록 지정합니다. 새도 관리자 v2.2.0 이상을 사용하는 경우 한 방향으로만 새도를 동기화하도록 새도 관리자 구성 요소를 구성할 수 있습니다. 이 기능을 구성하려면 동기화 `direction`을 `deviceToCloud` 또는 `cloudToDevice`로 변경합니다. 자세한 내용은 새도 관리자 구성 요소의 [방향 구성 파라미터](#)를 참조하세요.

```
{
  "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 새도 동기화

Greengrass 코어 디바이스가 AWS IoT 클라우드에 연결되면 새도 관리자가 구성 요소 구성에서 지정한 새도에 대해 다음 작업을 수행합니다. 동작은 지정하는 새도 동기화 방향 구성 옵션에 따라 달라집니다. 기본적으로 새도 관리자는 `betweenDeviceAndCloud` 옵션을 사용하여 양방향으로 새도를 동기화합니다. 새도 관리자 v2.2.0 이상을 사용하는 경우 한 방향으로만 새도를 동기화하도록 코어 디바이스를 구성할 수 있으며 이는 `cloudToDevice` 또는 `deviceToCloud`입니다.

- 새도 동기화 방향 구성이 `betweenDeviceAndCloud` 또는 `cloudToDevice`인 경우 새도 관리자는 AWS IoT Core의 클라우드 새도 문서에서 보고된 상태 정보를 검색합니다. 그런 다음 로컬로 저장된 새도 문서를 업데이트하여 디바이스 상태를 동기화합니다.
- 새도 동기화 방향 구성이 `betweenDeviceAndCloud` 또는 `deviceToCloud`인 경우 새도 관리자는 디바이스의 현재 상태를 클라우드 새도 문서에 게시합니다.

새도 병합 충돌 동작

코어 디바이스가 인터넷에서 연결 해제되는 경우와 같은 경우 새도 관리자가 변경 사항을 동기화하기 전에 로컬 새도 서비스와 AWS IoT 클라우드에서 새도우가 변경될 수 있습니다. 따라서 원하는 상태와 보고된 상태는 로컬 새도우 서비스와 AWS IoT 클라우드 간에 다릅니다.

새도 관리자가 새도를 동기화하면 다음 동작에 따라 변경 사항을 병합합니다.

- v2.2.0 이전 버전의 새도 관리자를 사용하거나 `betweenDeviceAndCloud` 새도 동기화 방향을 지정하는 경우 다음 동작이 적용됩니다.
 - 새도우의 원하는 상태에서 병합 충돌이 발생하면 새도 관리자가 로컬 새도우 문서의 충돌하는 섹션을 AWS IoT 클라우드의 값으로 덮어씁니다.
 - 새도우의 보고된 상태에서 병합 충돌이 있는 경우 새도우 관리자는 AWS IoT 클라우드에 있는 새도우의 충돌하는 섹션을 로컬 새도우 문서의 값으로 덮어씁니다.
- `deviceToCloud` 새도우 동기화 방향을 지정하면 새도 관리자가 AWS IoT 클라우드에 있는 새도우의 충돌 섹션을 로컬 새도우 문서의 값으로 덮어씁니다.
- `cloudToDevice` 새도 동기화 방향을 지정하면 새도 관리자가 로컬 새도 문서의 충돌 섹션을 AWS IoT 클라우드의 값으로 덮어씁니다.

Greengrass 코어 디바이스에서 데이터 스트림 관리

AWS IoT Greengrass 스트림 관리자를 사용하면 대용량 IoT 데이터를 로 더 효율적이고 안정적으로 전송할 수 있습니다 AWS 클라우드. 스트림 관리자는 AWS IoT Greengrass 코어에서 데이터 스트림을 처리한 후 로 내보냅니다 AWS 클라우드. 스트림 관리자는 AWS IoT Greengrass 코어 디바이스가 데이터를 AWS 클라우드 또는 로컬 스토리지 대상으로 내보내기 전에 데이터를 처리하고 분석하는 기계 학습(ML) 추론과 같은 일반적인 엣지 시나리오와 통합됩니다.

스트림 관리자는 사용자 지정 스트림 관리 기능을 구축할 필요가 없도록 사용자 지정 구성 요소 개발을 간소화하는 공통 인터페이스를 제공합니다. 구성 요소에서는 표준화된 메커니즘을 사용하여 대용량 스트림을 처리하고 로컬 데이터 보존 정책을 관리할 수 있습니다. 스트림별로 스토리지 유형, 크기 및 데이터 보존에 대한 정책을 정의하여 스트림 관리자가 데이터를 처리하고 내보내는 방식을 제어할 수 있습니다.

스트림 관리자는 연결이 간헐적이거나 제한된 환경에서 작동합니다. 대역폭 사용, 제한 시간 동작 및 AWS IoT Greengrass 코어가 연결되거나 연결이 해제될 때 스트림 데이터를 처리하는 방법을 정의할 수 있습니다. 또한 우선 순위를 설정하여 AWS IoT Greengrass 코어가 스트림을 AWS 클라우드로 내보내는 순서를 제어할 수 있습니다. 이를 통해 다른 데이터보다 중요한 데이터를 더 빨리 처리할 수 있습니다.

스토리지 또는 추가 처리 및 분석을 위해 로 데이터를 자동으로 내보내도록 스트림 관리자 AWS 클라우드 를 구성할 수 있습니다. 스트림 관리자는 다음 AWS 클라우드 대상으로의 내보내기를 지원합니다.

- 채널 in AWS IoT 분석. AWS IoT 분석 lets 데이터에 대한 고급 분석을 수행하여 비즈니스 결정을 내리고 기계 학습 모델을 개선하는 데 도움이 됩니다. 자세한 내용은 [AWS IoT 분석이란?](#)을 참조하세요.
- Amazon Kinesis Data Streams의 스트림. Kinesis Data Streams를 사용하여 대용량 데이터를 집계하고 데이터 웨어하우스나 MapReduce 클러스터에 로드할 수 있습니다. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 [Amazon Kinesis Data Streams이란 무엇입니까?](#)를 참조하세요.
- AWS IoT SiteWise의 자산 속성 AWS IoT SiteWise 은 산업 장비에서 대규모로 데이터를 수집, 구성 및 분석할 수 있도록 해줍니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 [What is AWS IoT SiteWise?](#)를 참조하세요.
- Amazon Simple Storage Service(Amazon S3)의 객체. Amazon S3를 사용하여 대량의 데이터를 저장 및 검색할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 [Amazon S3란 무엇인가요?](#)를 참조하세요.

스트림 관리 워크플로우

IoT 애플리케이션은 스트림 관리자 SDK를 통해 스트림 관리자와 연동합니다.

간단한 워크플로에서 AWS IoT Greengrass 코어의 구성 요소는 시계열 온도 및 압력 지표와 같은 IoT 데이터를 사용합니다. 이 구성 요소는 데이터를 필터링 또는 압축한 다음 스트림 관리자 SDK를 직접적으로 호출하여 스트림 관리자의 스트림에 데이터를 기록할 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 수도 있습니다.

IoT 애플리케이션에는 스트림을 읽거나 스트림에 쓰는 여러 사용자 지정 구성 요소가 포함될 수 있습니다. 이러한 구성 요소는 스트림에 읽고 쓸 수 있어 AWS IoT Greengrass 코어 디바이스의 데이터를 필터링, 집계 및 분석할 수 있습니다. 이렇게 하면 코어에서 AWS 클라우드 또는 로컬 대상으로 데이터를 전송하기 전에 로컬 이벤트에 빠르게 응답하고 중요한 정보를 추출할 수 있습니다.

시작하려면 스트림 관리자 구성 요소를 AWS IoT Greengrass 코어 디바이스에 배포합니다. 배포에서 스트림 관리자 구성 요소 파라미터를 구성하여 Greengrass 코어 디바이스의 모든 스트림에 적용되는 설정을 정의합니다. 이러한 파라미터를 사용하여 비즈니스 요구 사항 및 환경 제약에 따라 스트림 관리자가 스트림을 저장, 처리 및 내보내는 방법을 제어합니다.

스트림 관리자를 구성한 후 IoT 애플리케이션을 생성하고 배포할 수 있습니다. 일반적으로 이들 애플리케이션은 스트림 관리자 SDK에서 `StreamManagerClient`를 사용하여 스트림을 생성하고 조작하는 사용자 지정 구성 요소입니다. 스트림을 생성할 때 내보내기 대상, 우선순위, 지속성과 같은 스트림 별 정책을 정의할 수 있습니다.

요구 사항

스트림 관리자 사용을 위해 다음 요구 사항이 적용됩니다.

- 스트림 관리자에는 AWS IoT Greengrass 코어 소프트웨어 외에 최소 70MB RAM이 필요합니다. 총 메모리 요구 사항은 워크로드에 따라 다릅니다.
- AWS IoT Greengrass 구성 요소는 스트림 관리자 SDK를 사용하여 스트림 관리자와 상호 작용해야 합니다. 스트림 관리자 SDK는 다음 언어로 제공됩니다.
 - [Java용 스트림 관리자 SDK](#)(v1.1.0 이상)
 - [Node.js용 스트림 관리자 SDK](#)(v1.1.0 이상)
 - [Python용 스트림 관리자 SDK](#)(v1.1.0 이상)
- AWS IoT Greengrass 구성 요소는 스트림 관리자 구성 요소(`aws.greengrass.StreamManager`)를 레시피의 종속성으로 지정해야 스트림 관리자를 사용할 수 있습니다.

Note

스트림 관리자를 사용하여 클라우드로 데이터를 내보내는 경우에는 스트림 관리자 구성 요소의 버전 2.0.7을 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드할 수 없습니다. 스트림 관리자를 처음 배포하는 경우 최신 버전의 스트림 관리자 구성 요소를 배포하는 것이 좋습니다.

- 스트림의 AWS 클라우드 내보내기 대상을 정의하는 경우 내보내기 대상을 생성하고 [Greengrass 디바이스 역할](#)에서 액세스 권한을 부여해야 합니다. 대상에 따라 다른 요구 사항도 적용될 수 있습니다. 자세한 내용은 다음을 참조하세요.
 - [the section called “AWS IoT 분석 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 유지 관리하는 것은 사용자의 책임입니다.

데이터 보안

스트림 관리자를 사용할 때는 다음과 같은 보안 고려 사항에 유의하세요.

로컬 데이터 보안

AWS IoT Greengrass 는 코어 디바이스의 로컬 구성 요소 간에 저장 중이거나 전송 중인 스트림 데이터를 암호화하지 않습니다.

- 저장 시 데이터. 스트림 데이터는 스토리지 디렉터리에 로컬로 저장됩니다. 데이터 보안을 위해 AWS IoT Greengrass 는 활성화된 경우 파일 권한과 전체 디스크 암호화를 사용합니다. 선택적 [STREAM_MANAGER_STORE_ROOT_DIR](#) 파라미터를 사용하여 스토리지 디렉터리를 지정할 수 있습니다. 나중에 다른 스토리지 디렉터리를 사용하도록 이 파라미터를 변경하는 경우는 이전 스토리지 디렉터리 또는 해당 콘텐츠를 삭제하지 AWS IoT Greengrass 않습니다.
- 로컬로 전송 중인 데이터.는 데이터 소스, AWS IoT Greengrass 구성 요소, 스트림 관리자 SDK 및 스트림 관리자 간에 로컬로 전송 중인 스트림 데이터를 암호화하지 AWS IoT Greengrass 않습니다.
- 로 전송 중인 데이터입니다 AWS 클라우드. 스트림 관리자가 로 내보낸 데이터 스트림은 전송 계층 보안(TLS)을 통한 표준 AWS 서비스 클라이언트 암호화를 AWS 클라우드 사용합니다.

클라이언트 인증

스트림 관리자 클라이언트는 스트림 관리자 SDK를 사용하여 스트림 관리자와 통신합니다. 클라이언트 인증이 활성화된 경우 Greengrass 구성 요소만 스트림 관리자의 스트림과 연동할 수 있습니다. 클라이언트 인증이 비활성화된 경우에는 Greengrass 코어 디바이스에서 실행 중인 모든 프로세스가 스트림 관리자의 스트림과 연동할 수 있습니다. 비즈니스 사례에 필요한 경우에만 인증을 비활성화해야 합니다.

[STREAM_MANAGER_AUTHENTICATE_CLIENT](#) 파라미터를 사용하여 클라이언트 인증 모드를 설정합니다. 이 파라미터는 스트림 관리자 구성 요소를 코어 디바이스에 배포할 때 구성할 수 있습니다.

	활성화됨	비활성화됨
파라미터값	true(기본값 및 권장)	false
허용된 클라이언트	코어 디바이스의 Greengrass 구성 요소	코어 디바이스의 Greengrass 구성 요소 Greengrass 코어 장치에서 실행 중인 기타 프로세스

다음 사항도 참조하세요.

- [the section called “스트림 관리자 구성”](#)
- [the section called “StreamManagerClient를 사용하여 스트림 작업”](#)
- [the section called “지원되는 클라우드 대상에 대한 구성 내보내기”](#)

AWS IoT Greengrass 스트림 관리자 구성

Greengrass 코어 디바이스에서는 스트림 관리자를 통해 IoT 디바이스 데이터를 저장, 처리하고 내보낼 수 있습니다. 스트림 관리자에서는 런타임 설정을 구성하는 데 사용하는 파라미터가 제공됩니다. 이러한 설정은 Greengrass 코어 디바이스의 모든 스트림에 적용됩니다. 구성 요소를 배포할 때 AWS IoT Greengrass 콘솔 또는 API를 사용하여 스트림 관리자 설정을 구성할 수 있습니다. 배포 완료 후 변경 사항이 적용됩니다.

스트림 관리자 파라미터

스트림 관리자에서는 코어 디바이스에 구성 요소를 배포할 때 구성할 수 있는 다음 파라미터가 제공됩니다. 모든 파라미터는 선택 사항입니다.

스토리지 디렉터리

파라미터 이름: `STREAM_MANAGER_STORE_ROOT_DIR`

스트림 저장에 사용되는 로컬 폴더의 절대 경로. 이 값은 슬래시로 시작해야 합니다(예: `/data`).

기존 폴더를 지정해야 하며 [스트림 관리자 구성 요소를 실행하는 시스템 사용자](#)에게 이 폴더를 읽고 쓸 권한이 있어야 합니다. 예를 들어 다음 명령을 실행하여 스트림 관리자 루트 폴더로 지정하는 폴더(`/var/greengrass/streams`)를 생성하고 구성할 수 있습니다. 이러한 명령을 사용하면 기본 시스템 사용자(`ggc_user`)가 이 폴더를 읽고 쓸 수 있습니다.

```
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`

서버 포트:

파라미터 이름: `STREAM_MANAGER_SERVER_PORT`

스트림 관리자와 통신하는 데 사용되는 로컬 포트 번호입니다. 기본값은 8088입니다.

임의의 사용 가능한 포트를 사용하도록 0을 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 스트림 관리자 SDK를 통해 제어됩니다. 이 파라미터는 스트림 관리자 SDK를 직접 호출하여 스트림 작업을 수행할 수 있는 클라이언트를 결정합니다. 자세한 내용은 [the section called “클라이언트 인증”](#) 섹션을 참조하세요.

유효한 값은 `true` 또는 `false`입니다. 기본값은 `true`입니다(권장).

- `true`. Greengrass 구성 요소만 클라이언트로 허용됩니다. 구성 요소에서는 스트림 관리자 SDK를 통해 인증되는 데 AWS IoT Greengrass 코어 프로토콜이 사용됩니다.
- `false`. AWS IoT Greengrass 코어에서 실행되는 모든 프로세스가 클라이언트로 허용됩니다. 비즈니스 사례에 필요한 경우가 아니면 값을 `false`로 설정하지 마세요. 예를 들어 코어 디바이스의 구성 요소 외 프로세스에서 스트림 관리자와 직접 통신해야 하는 경우에만 `false`를 사용합니다.

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 장치 하드웨어의 CPU 및 메모리는 제한적인 요소입니다. 시작을 위해 이 값을 장치의 프로세서 코어 수와 동일하게 설정해 볼 수 있습니다.

하드웨어가 지원할 수 있는 크기 보다 크게 설정하지 않도록 주의하십시오. 각 스트림은 하드웨어 리소스를 사용하므로 제한된 디바이스에서는 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 수준 순서대로 나열된 다음 로그 수준 중에서 선택합니다.

- `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 요소에서는 Amazon S3로 내보내는 스트림이 생성될 때 이 임계값이 설정됩니다. 기본 임계값 크기는 5MB입니다.

다음 사항도 참조하세요.

- [Greengrass 코어 디바이스에서 데이터 스트림 관리](#)
- [StreamManagerClient를 사용하여 스트림 작업](#)
- [지원되는 AWS 클라우드 대상에 대한 구성 내보내기](#)

스트림 관리자를 사용하는 사용자 지정 구성 요소 생성

사용자 지정 Greengrass 구성 요소의 스트림 관리자를 사용하여 IoT 디바이스 데이터를 저장하고 처리하고 내보낼 수 있습니다. 이 섹션의 절차와 예제를 사용하여 스트림 관리자와 함께 작동하는 구성 요소 레시피, 아티팩트, 애플리케이션을 생성합니다. 구성 요소 개발 및 테스트 방법에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 생성](#) 섹션을 참조하세요.

주제

- [스트림 관리자를 사용하는 구성 요소 레시피 정의](#)
- [애플리케이션 코드에서 스트림 관리자에 연결](#)

스트림 관리자를 사용하는 구성 요소 레시피 정의

사용자 지정 구성 요소에서 스트림 관리자를 사용하려면 `aws.greengrass.StreamManager` 구성 요소를 종속성으로 정의해야 합니다. 스트림 관리자 SDK도 제공해야 합니다. 다음 태스크를 완료하여 원하는 언어로 스트림 관리자 SDK를 다운로드하고 사용하세요.

Java용 스트림 관리자 SDK 사용

Java용 스트림 관리자 SDK는 구성 요소를 컴파일하는 데 사용할 수 있는 JAR 파일로 제공됩니다. 그런 다음 스트림 관리자 SDK가 포함된 애플리케이션 JAR을 생성하고, 애플리케이션 JAR을 구성 요소 아티팩트로 정의하고, 구성 요소 수명 주기에서 애플리케이션 JAR을 실행할 수 있습니다.

Java용 스트림 관리자 SDK를 사용하려면

1. [Java JAR용 스트림 관리자 SDK 파일](#)을 다운로드합니다.
2. 다음 중 하나를 수행하여 Java 애플리케이션과 스트림 관리자 SDK JAR 파일에서 구성 요소 아티팩트를 생성합니다.
 - 애플리케이션을 스트림 관리자 SDK JAR이 포함된 JAR 파일로 빌드하고 구성 요소 레시피에서 해당 JAR 파일을 실행합니다.
 - 스트림 관리자 SDK JAR을 구성 요소 아티팩트로 정의합니다. 구성 요소 레시피에서 애플리케이션을 실행할 때 해당 아티팩트를 클래스 경로에 추가합니다.

구성 요소 레시피는 다음 예제와 유사할 수 있습니다. 이 구성 요소는 `StreamManagerS3.jar`에 스트림 관리자 SDK JAR이 포함된 [StreamManagerS3.java](#) 예제의 수정된 버전을 실행합니다.

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://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar

```

구성 요소 개발 및 테스트 방법에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 생성](#) 섹션을 참조하세요.

Python용 스트림 관리자 SDK 사용

Python용 스트림 관리자 SDK는 구성 요소에 포함할 수 있는 소스 코드로 사용할 수 있습니다. 스트림 관리자 SDK의 ZIP 파일을 생성하고 ZIP 파일을 구성 요소 아티팩트로 정의한 다음 구성 요소 수명 주기에 SDK의 요구 사항을 설치합니다.

Python용 스트림 관리자 SDK를 사용하려면

1. [aws-greengrass-stream-manager-sdk-python](#) 리포지토리를 복제하거나 다운로드합니다.

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-python.git
```

2. Python용 스트림 관리자 SDK의 소스 코드를 포함하는 `stream_manager` 폴더를 포함한 ZIP 파일을 생성합니다. 이 ZIP 파일을 구성 요소를 설치할 때 AWS IoT Greengrass 코어 소프트웨어가 압축을 풀 수 있는 구성 요소 아티팩트로 제공할 수 있습니다. 해결 방법:

- a. 이전 단계에서 복제하거나 다운로드한 리포지토리가 포함된 폴더를 엽니다.

```
cd aws-greengrass-stream-manager-sdk-python
```

- b. `stream_manager` 폴더를 `stream_manager_sdk.zip`이라는 이름의 ZIP 파일로 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 콘텐츠가 포함되어 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
```

출력은 다음과 유사합니다.

```
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 관리자 SDK 아티팩트를 구성 요소의 아티팩트 폴더에 복사합니다. 구성 요소는 스트림 관리자 SDK ZIP 파일 외에도 SDK의 `requirements.txt` 파일을 사용하여 스트림 관리자 SDK의 종속성을 설치합니다. `~/greengrass-components`를 로컬 개발에 사용하는 폴더의 경로로 바꿉니다.

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. 구성 요소 레시피를 생성합니다. 레시피에서 다음을 수행합니다.
 - a. stream_manager_sdk.zip 및 requirements.txt를 아티팩트로 정의합니다.
 - b. Python 애플리케이션을 아티팩트로 정의합니다.
 - c. 설치 수명 주기에서 requirements.txt의 스트림 관리자 SDK 요구 사항을 설치합니다.
 - d. 실행 수명 주기에서 스트림 관리자 SDK를 PYTHONPATH에 추가하고 Python 애플리케이션을 실행합니다.

구성 요소 레시피는 다음 예제와 유사할 수 있습니다. 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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
      "Unarchive": "ZIP"
    },
    {
      "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
    },
    {
      "URI": "s3://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
      "Unarchive": "ZIP"
    },
    {
      "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
    },
    {
      "URI": "s3://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
```

```
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

구성 요소 개발 및 테스트 방법에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 생성](#) 섹션을 참조하세요.

JavaScript용 스트림 관리자 SDK 사용

JavaScript용 스트림 관리자 SDK는 구성 요소에 포함할 수 있는 소스 코드로 사용할 수 있습니다. 스트림 관리자 SDK의 ZIP 파일을 생성하고, ZIP 파일을 구성 요소 아티팩트로 정의한 다음, 구성 요소 수명 주기에 SDK를 설치합니다.

JavaScript용 스트림 관리자 SDK를 사용하려면

1. [aws-greengrass-stream-manager-sdk-js](#) 리포지토리를 복제하거나 다운로드합니다.

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
```

2. JavaScript용 스트림 관리자 SDK의 소스 코드를 포함하는 aws-greengrass-stream-manager-sdk 폴더를 포함한 ZIP 파일을 생성합니다. 이 ZIP 파일을 구성 요소를 설치할 때 AWS IoT Greengrass 코어 소프트웨어가 압축을 풀 수 있는 구성 요소 아티팩트로 제공할 수 있습니다. 해결 방법:

- a. 이전 단계에서 복제하거나 다운로드한 리포지토리가 포함된 폴더를 엽니다.

```
cd aws-greengrass-stream-manager-sdk-js
```

- b. aws-greengrass-stream-manager-sdk 폴더를 stream-manager-sdk.zip이라는 이름의 ZIP 파일로 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 콘텐츠가 포함되어 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
```

출력은 다음과 유사합니다.

```
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 관리자 SDK 아티팩트를 구성 요소의 아티팩트 폴더에 복사합니다. `~/greengrass-components`를 로컬 개발에 사용하는 폴더의 경로로 바꿉니다.

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. 구성 요소 레시피를 생성합니다. 레시피에서 다음을 수행합니다.
 - a. stream-manager-sdk.zip을 아티팩트로 정의합니다.
 - b. JavaScript 애플리케이션을 아티팩트로 정의합니다.
 - c. 설치 수명 주기에서 stream-manager-sdk.zip 아티팩트의 스트림 관리자 SDK를 설치합니다. 이 `npm install` 명령은 스트림 관리자 SDK와 해당 종속 항목을 포함한 `node_modules` 폴더를 생성합니다.
 - d. 실행 수명 주기에서 `node_modules` 폴더를 `NODE_PATH`에 추가하고 JavaScript 애플리케이션을 실행합니다.

구성 요소 레시피는 다음 예제와 유사할 수 있습니다. 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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-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://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {

```

```

        "URI": "s3://amzn-s3-demo-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 요소 개발 및 테스트 방법에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 생성](#) 섹션을 참조하세요.

애플리케이션 코드에서 스트림 관리자에 연결

애플리케이션의 스트림 관리자에 연결하려면 스트림 관리자 SDK에서 `StreamManagerClient`의 인스턴스를 생성합니다. 이 클라이언트는 기본 포트 8088의 스트림 관리자 구성 요소 또는 지정된 포트에 연결합니다. 인스턴스를 생성한 후 `StreamManagerClient`를 사용하는 방법에 대한 자세한 내용은 [StreamManagerClient를 사용하여 스트림 작업](#) 섹션을 참조하세요.

Example예: 기본 포트를 사용하여 스트림 관리자에 연결

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.
}
```

Example예: 기본값이 아닌 포트를 사용하여 스트림 관리자에 연결

기본 포트가 아닌 포트로 스트림 관리자를 구성하는 경우 [프로세스 간 통신](#)을 사용하여 구성 요소 구성에서 포트를 검색해야 합니다.

Note

port 구성 파라미터에는 스트림 관리자를 배포할 때 STREAM_MANAGER_SERVER_PORT에서 지정하는 값이 포함되어 있습니다.

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
    IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCCClient greengrassCoreIPCCClient = new
    GreengrassCoreIPCCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
        greengrassCoreIPCCClient.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.

```

StreamManagerClient를 사용하여 스트림 작업

Greengrass 코어 디바이스에서 실행되는 사용자 정의 Greengrass 구성 요소는 스트림 관리자 SDK의 StreamManagerClient 객체를 사용하여 [스트림 관리자](#)에서 스트림을 생성한 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 전송하려면 구성 요소가 데이터를 스트림에 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 코어에서 실행되는 구성 요소가 아닌 프로세스(예: Docker 컨테이너)가 스트림 관리자와 상호 작용하도록 허용할 수 있습니다. 자세한 내용은 [the section called “클라이언트 인증”](#) 단원을 참조하십시오.

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 함수가 함수 핸들러 StreamManagerClient 외부에서 인스턴스화되어야 합니다. 핸들러에서 인스턴스화된 함수는 호출될 때마다 스트림 관리자에 대한 client 및 연결을 만듭니다.

Note

핸들러에서 StreamManagerClient을(를) 인스턴스한 경우, client이(가) 작업을 완료하면 사용자가 close() 메서드를 명시적으로 호출해야 합니다. 그렇지 않으면 client은(는) 연결을 열어두고 스크립트가 종료될 때까지 다른 스레드를 실행합니다.

StreamManagerClient에서는 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 구성 요소가 생성 메서드를 직접적으로 호출하고 `MessageStreamDefinition` 객체를 전달합니다. 이 객체는 스트림의 고유한 이름을 지정하고 최대 스트림 크기에 도달했을 때 스트림 관리자가 새 데이터를 처리하는 방법을 정의합니다. `MessageStreamDefinition` 및 해당 데이터 유형(예: `ExportDefinition`, `StrategyOnFull` 및 `Persistence`)을 사용하여 다른 스트림 속성을 정의할 수 있습니다. 다음이 포함됩니다.

- 자동 내보내기를 위한 대상 AWS IoT SiteWise, AWS IoT 분석 Kinesis Data Streams 및 Amazon S3 대상입니다. 자세한 내용은 [the section called “지원되는 클라우드 대상에 대한 구성 내보내기”](#) 단원을 참조하십시오.
- 내보내기 우선 순위. 스트림 관리자는 우선 순위가 낮은 스트림보다 우선 순위가 높은 스트림을 먼저 내보냅니다.
- AWS IoT 분석, Kinesis Data Streams 및 AWS IoT SiteWise 대상의 최대 배치 크기 및 배치 간격입니다. 스트림 관리자는 두 조건 중 하나가 충족되면 메시지를 내보냅니다.
- TTL(Time-to-Live). 스트림 데이터를 처리에 사용할 수 있도록 보장하는 시간입니다. 이 기간 내에 데이터를 사용할 수 있는지 확인해야 합니다. 이는 삭제 정책이 아닙니다. TTL 기간 직후에는 데이터가 삭제되지 않을 수 있습니다.
- 스트림 지속성. 스트림을 파일 시스템에 저장하여 코어 재시작 시 데이터를 유지하거나 메모리에 스트림을 저장하도록 선택합니다.
- 시작 시퀀스 번호 내보내기에서 시작 메시지로 사용할 메시지의 시퀀스 번호를 지정합니다.

`MessageStreamDefinition`에 대한 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하세요.

- Java SDK의 [MessageStreamDefinition](#)
- Node.js SDK의 [MessageStreamDefinition](#)
- Python SDK의 [MessageStreamDefinition](#)

Note

StreamManagerClient는 스트림을 HTTP 서버로 내보내는 데 사용할 수 있는 대상을 제공합니다. 이 대상은 테스트 목적으로만 사용됩니다. 이 대상은 안정적이지 않으며 프로덕션 환경에서 사용할 수 없습니다.

스트림이 생성되면 Greengrass 구성 요소가 스트림에 [메시지를 추가](#)하여 내보내기용 데이터를 전송하고 로컬 처리를 위해 스트림에서 [메시지를 읽을 수 있습니다](#). 생성하는 스트림 수는 하드웨어 기능 및 비즈니스 사례에 따라 다릅니다. 한 전략은 스트림에 대해 여러 대상을 정의할 수 있지만 AWS IoT 분석 또는 Kinesis 데이터 스트림의 각 대상 채널에 대해 스트림을 생성하는 것입니다. 스트림은 안정적인 수명을 가지고 있습니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 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: [create_message_stream](#) | [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.
}

```

Java SDK 참조: [createMessageStream](#) | [MessageStreamDefinition](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
  try {
    await client.createMessageStream(
      new MessageStreamDefinition()
        .WithName("StreamName") // Required.
        .withMaxSize(268435456) // Default is 256 MB.
        .withStreamSegmentSize(16777216) // Default is 16 MB.
        .withTimeToLiveMillis(null) // By default, no TTL is enabled.
        .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) // Required.
        .withPersistence(Persistence.File) // Default is File.
        .withFlushOnWrite(false) // Default is false.
        .withExportDefinition( // Optional. Choose where/how the stream is exported
to the AWS #####.
          new ExportDefinition()
            .withKinesis(null)
            .withIotAnalytics(null)
            .withIotSiteWise(null)
            .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: [createMessageStream](#) | [MessageStreamDefinition](#)

내보내기 대상 구성에 대한 자세한 내용은 [the section called “지원되는 클라우드 대상에 대한 구성 내보내기”](#) 섹션을 참조하세요.

메시지 추가

내보내기를 위해 스트림 관리자로 데이터를 전송하려면 Greengrass 구성 요소가 대상 스트림에 데이터를 추가합니다. 내보내기 대상은 이 메소드에 전달할 데이터 유형을 결정합니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

예제

AWS IoT 분석 또는 Kinesis Data Streams 내보내기 대상

다음 코드 조각은 StreamName이라는 스트림에 메시지를 추가합니다. AWS IoT 분석 또는 Kinesis Data Streams 대상의 경우 Greengrass 구성 요소는 데이터 블록을 추가합니다.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 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 대상

다음 코드 조각은 StreamName이라는 스트림에 메시지를 추가합니다. AWS IoT SiteWise 대상의 경우 Greengrass 구성 요소는 직렬화된 PutAssetPropertyValueEntry 객체를 추가합니다. 자세한 내용은 [the section called “로 내보내기 AWS IoT SiteWise”](#) 단원을 참조하십시오.

Note

데이터를 전송할 때 AWS IoT SiteWise 데이터는 BatchPutAssetPropertyValue 작업의 요구 사항을 충족해야 합니다. 자세한 내용은 AWS IoT SiteWise API 참조의 [BatchPutAssetPropertyValue](#)를 참조하세요.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 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: [append_message](#) | [PutAssetPropertyValueEntry](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
    in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>();

    // IoTSiteWise requires unique timestamps in all messages 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.
}

```

Java SDK 참조: [appendMessage](#) | [PutAssetPropertyValueEntry](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
Math.floor(Math.random() * maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    }
}

```

```

        const putAssetPropertyValueEntry = new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
        const sequenceNumber = await client.appendMessage("StreamName",
            util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});

```

Node.js SDK 참조: [appendMessage](#) | [PutAssetPropertyValueEntry](#)

Amazon S3 대상

다음 코드 조각은 StreamName이라는 스트림에 내보내기 작업을 추가합니다. Amazon S3 대상의 경우 Greengrass 구성 요소가 소스 입력 파일 및 대상 Amazon S3 객체에 대한 정보를 포함하는 직렬화된 S3ExportTaskDefinition 객체를 추가합니다. 지정된 객체가 없는 경우 Stream Manager가 자동으로 생성합니다. 자세한 내용은 [the section called “Amazon S3로 내보내기”](#) 단원을 참조하십시오.

이 코드 조각에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 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: [append_message](#) | [S3ExportTaskDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
        ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 참조: [appendMessage](#) | [S3ExportTaskDefinition](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        // Append an Amazon S3 export task definition and print the sequence number.
        const taskDefinition = new S3ExportTaskDefinition()
            .withBucket("BucketName")
            .withKey("KeyName")
            .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
            util.validateAndSerializeToJsonBytes(taskDefinition));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
```

```
});
```

Node.js SDK 참조: [appendMessage](#) | [S3ExportTaskDefinition](#)

메시지 읽기

스트림의 메시지 읽기

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

예제

다음 코드 조각은 StreamName이라는 스트림에서 메시지를 읽습니다. 읽기 메서드는 읽기를 시작할 시퀀스 번호, 읽기를 수행할 최소 및 최대 숫자, 메시지 읽기 시간 제한을 지정하는 선택적 ReadMessagesOptions 객체를 가져옵니다.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
        the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
            0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
            NotEnoughMessagesException is raised. By default, this is 1.
            max_message_count=100,    # Accept up to 100 messages. By default this
            is 1.
            read_timeout_millis=5000
```

```

        # Try to wait at most 5 seconds for the min_message_count to be
        fulfilled. By default, this is 0, which immediately returns the messages or an
        exception.
    )
)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.

```

Python SDK 참조: [read_messages](#) | [ReadMessagesOptions](#)

Java

```

try (final StreamManagerClient client =
    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_message_count to
            be fulfilled. By default, this is 0, which immediately returns the messages or an
            exception.
            .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Java SDK 참조: [readMessages](#) | [ReadMessagesOptions](#)

Node.js

```
const client = new StreamManagerClient();
```

```
client.onConnected(async () => {
  try {
    const messages = await client.readMessages("StreamName",
      // By default, if no options are specified, it tries to read one message
      from the beginning of the stream.
      new ReadMessagesOptions()
        // Try to read from sequence number 100 or greater. By default this
        is 0.
        .withDesiredStartSequenceNumber(100)
        // Try to read 10 messages. If 10 messages are not available, then
        NotEnoughMessagesException is thrown. By default, this is 1.
        .withMinMessageCount(10)
        // Accept up to 100 messages. By default this is 1.
        .withMaxMessageCount(100)
        // Try to wait at most 5 seconds for the minMessageCount to be
        fulfilled. By default, this is 0, which immediately returns the messages or an
        exception.
        .withReadTimeoutMillis(5 * 1000)
    );
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: [readMessages](#) | [ReadMessagesOptions](#)

스트림 나열

스트림 매니저에서 스트림 목록을 가져옵니다.

요구 사항

이 작업에는 다음과 같은 요구 사항이 있습니다.

- 최소 스트림 관리자 SDK 버전: Python: 1.1.0 | Java: 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 관리자 SDK 버전: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

예제

다음 코드 조각은 스트림의 정의, 크기 및 내보내기 상태를 포함하여 StreamName라는 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.
}
```

Java SDK 참조: [describeMessageStream](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
```

```
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 참조: [describeMessageStream](#)

메시지 스트림 업데이트

기존 스트림의 속성을 업데이트합니다. 스트림이 생성된 후 요구 사항이 변경되면 스트림을 업데이트하는 것이 좋습니다. 예제:

- AWS 클라우드 대상에 대한 새 [내보내기 구성](#)을 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 간략한 프로세스를 따라 스트림을 업데이트합니다.

1. [스트림에 대한 설명을 가져옵니다.](#)
2. 해당 MessageStreamDefinition 및 하위 객체의 대상 속성을 업데이트합니다.
3. 업데이트된 MessageStreamDefinition을(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 관리자 SDK 버전: Python: 1.1.0 | Java: 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: [updateMessageStream](#) | [MessageStreamDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
            Strategy on full to reject new data.
            // Max Size update should be greater than initial Max Size defined in
            Create Message Stream request
```

```

        .withMaxSize(536870912L) // Update Max Size to 512 MB.
        .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
        .withFlushOnWrite(true) // Update flush on write to true.
        .withPersistence(Persistence.Memory) // Update the persistence to
Memory.

        .withTimeToLiveMillis(3600000L) // Update TTL to 1 hour.
        .withExportDefinition(
            // Optional. Choose where/how the stream is exported to the AWS ###
#.
            messageStreamInfo.getDefinition().getExportDefinition().
            // Updating Export definition to add a Kinesis Stream
configuration.

            .withKinesis(new ArrayList<KinesisConfig>() {{
                add(new KinesisConfig()
                    .withIdentifier(EXPORT_IDENTIFIER)
                    .withKinesisStreamName("test"));
            })
        );
    } catch (StreamManagerException e) {
        // Properly handle exception.
    }

```

Java SDK 참조: [update_message_stream](#) | [MessageStreamDefinition](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
            // Max Size update should be greater than initial Max Size defined
in Create Message Stream request
            .withMaxSize(536870912) // Default is 256 MB. Updating Max Size
to 512 MB.
            .withStreamSegmentSize(33554432) // Default is 16 MB. Updating
Segment Size to 32 MB.
            .withTimeToLiveMillis(3600000) // By default, no TTL is enabled.
Update TTL to 1 hour.
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required.
Updating Strategy on full to reject new data.
            .withPersistence(Persistence.Memory) // Default is File. Update
the persistence to Memory

```

```

        .withFlushOnWrite(true)    // Default is false. Updating to true.
        .withExportDefinition(
            // Optional. Choose where/how the stream is exported to the AWS
#####.
            messageStreamInfo.definition.exportDefinition
            // Updating Export definition to add a Kinesis Stream
configuration.
            .withKinesis([new
KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
        )
    );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});

```

[Node.js SDK 참조: updateMessageStream | MessageStreamDefinition](#)

스트림 업데이트에 대한 제약조건

스트림을 업데이트할 때는 다음 제약조건이 적용됩니다. 다음 목록에 명시되어 있지 않는 한 업데이트는 즉시 적용됩니다.

- 스트림의 지속성을 업데이트할 수 없습니다. 이 동작을 변경하려면 [스트림을 삭제하고](#) 새 지속성 정책을 정의하는 [스트림을 생성합니다](#).
- 다음과 같은 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 [업데이트](#)하려면 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 관리자 SDK 버전: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

예제

다음 코드 조각은 StreamName라는 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.
});
```

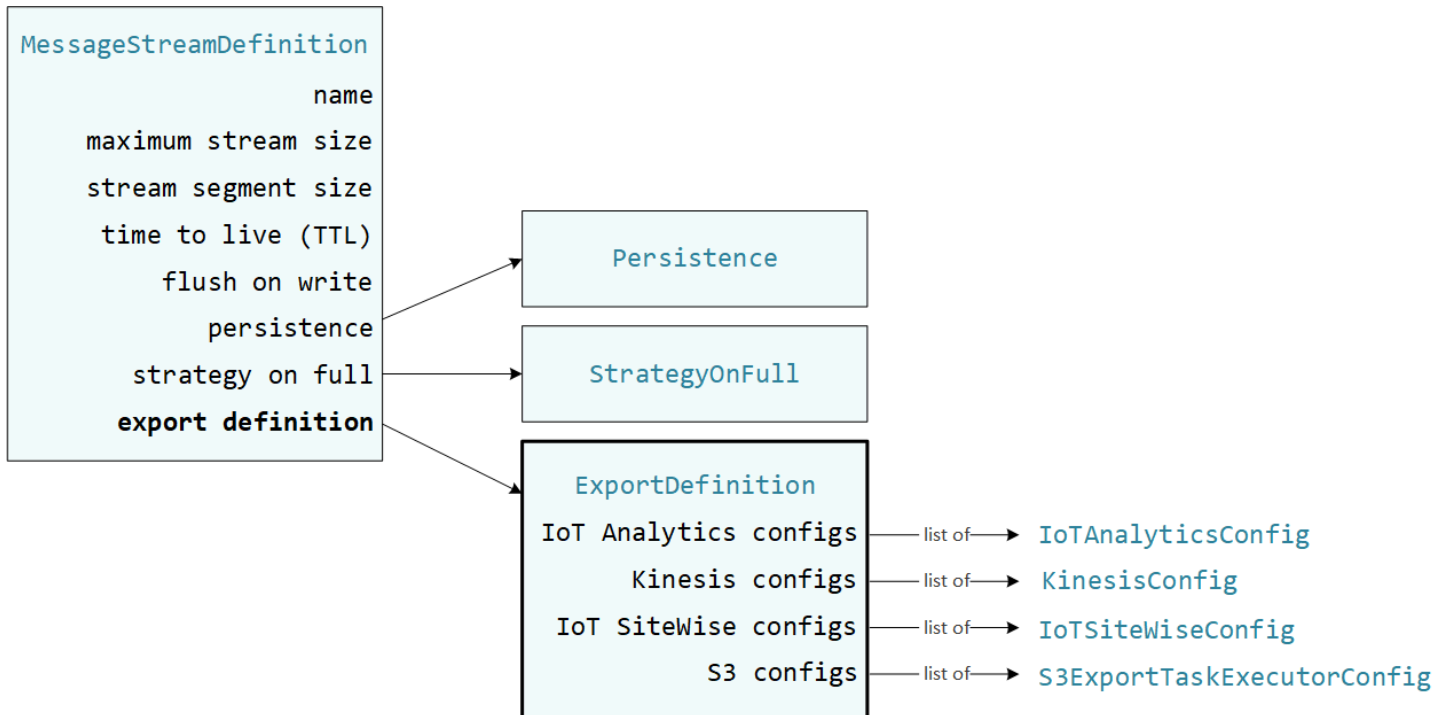
Node.js SDK 참조: [deleteMessageStream](#)

다음 사항도 참조하세요.

- [Greengrass 코어 디바이스에서 데이터 스트림 관리](#)
- [AWS IoT Greengrass 스트림 관리자 구성](#)
- [지원되는 AWS 클라우드 대상에 대한 구성 내보내기](#)
- 스트림 관리자 SDK 참조의 StreamManagerClient:
 - [Python](#)
 - [Java](#)
 - [Node.js](#)

지원되는 AWS 클라우드 대상에 대한 구성 내보내기

사용자 정의 Greengrass 구성 요소는 스트림 관리자 SDK의 `StreamManagerClient`를 사용하여 스트림 관리자와 상호 작용합니다. 구성 요소는 [스트림을 생성](#)하거나 [스트림을 업데이트](#)하는 경우 내보내기 정의를 포함하여 스트림 속성을 나타내는 `MessageStreamDefinition` 객체를 전달합니다. `ExportDefinition` 객체에는 스트림에 대해 정의된 내보내기 구성이 포함됩니다. 스트림 관리자는 이러한 내보내기 구성을 사용하여 스트림을 내보내는 위치와 방법을 결정합니다.



단일 대상 유형에 대한 여러 내보내기 구성을 포함하여 스트림에 0개 이상의 내보내기 구성을 정의할 수 있습니다. 예를 들어 하나의 스트림을 2개의 AWS IoT 분석 채널과 하나의 Kinesis 데이터 스트림으로 내보낼 수 있습니다.

내보내기 시도가 실패한 경우 스트림 관리자는 최대 5분 간격으로 AWS 클라우드 로 데이터 내보내기를 계속 재시도합니다. 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 분석 채널](#)
- [Amazon Kinesis Data Streams](#)
- [AWS IoT SiteWise 자산 속성](#)
- [Amazon S3 객체](#)

이러한 AWS 클라우드 리소스를 유지 관리하는 것은 사용자의 책임입니다.

AWS IoT 분석 채널

스트림 관리자는 로 자동 내보내기를 지원합니다 AWS IoT 분석. AWS IoT 분석 를 사용하면 데이터에 대한 고급 분석을 수행하여 비즈니스 결정을 내리고 기계 학습 모델을 개선할 수 있습니다. 자세한 내용은 AWS IoT Analytics 사용 설명서의 [What is AWS IoT Analytics?](#)를 참조하세요.

스트림 관리자 SDK에서 Greengrass 구성 요소는 IoTAnalyticsConfig를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하세요.

- Python SDK의 [IoTAnalyticsConfig](#)
- 자바 SDK의 [IoTAnalyticsConfig](#)
- Node.js SDK의 [IoTAnalyticsConfig](#)

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 의 대상 채널은 Greengrass 코어 디바이스 AWS 리전 와 동일한 AWS 계정 및에 AWS IoT 분석 있어야 합니다.
- [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#)은 채널을 타겟팅할 수 있는 `iotanalytics:BatchPutMessage` 권한을 허용해야 합니다. 예제:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ]
    }
  ]
}
```

```

    "Resource": [
      "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
      "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
    ]
  }
}

```

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 [IAM 정책 추가 및 제거](#)를 참조하세요.

로 내보내기 AWS IoT 분석

를 내보내는 스트림을 생성하기 위해 AWS IoT 분석 Greengrass 구성 요소는 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 디바이스로부터 데이터를 수신할 때 대상 스트림에 데이터 블록을 포함하는 [메시지를 추가](#)합니다.

그런 다음 스트림 관리자는 스트림의 내보내기 구성에 정의된 배치 설정 및 우선순위에 따라 데이터를 내보냅니다.

Amazon Kinesis Data Streams

스트림 관리자는 Amazon Kinesis Data Streams로의 자동 내보내기를 지원합니다. Kinesis Data Streams는 일반적으로 대용량 데이터를 종합하여 데이터 웨어하우스나 MapReduce 클러스터에 로드하는 데 사용됩니다. 자세한 내용은 Amazon Kinesis 개발자 안내서의 [Amazon Kinesis Data Streams이란 무엇입니까?](#)를 참조하세요.

스트림 관리자 SDK에서 Greengrass 구성 요소는 KinesisConfig를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하세요.

- Python SDK의 [KinesisConfig](#)
- 자바 SDK의 [KinesisConfig](#)
- Node.js SDK의 [KinesisConfig](#)

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- Kinesis Data Streams의 대상 스트림은 AWS 리전 Greengrass 코어 디바이스 AWS 계정 와 동일한 밑에 있어야 합니다.
- (권장) 스트림 관리자 버전 2.2.1은 스트림을 Kinesis Data Streams 대상으로 내보내는 성능을 향상합니다. 최신 버전의 개선 사항을 사용하려면 [스트림 관리자 구성 요소](#)를 v2.2.1로 업그레이드하고 [Greengrass 토큰 교환 역할](#)에서 `kinesis:ListShards` 정책을 사용합니다.
- [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#)은 데이터 스트림을 타겟팅할 수 있는 `kinesis:PutRecords` 권한을 허용해야 합니다. 예제:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:123456789012:stream/stream_1_name",
        "arn:aws:kinesis:us-east-1:123456789012:stream/stream_2_name"
      ]
    }
  ]
}
```

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 [IAM 정책 추가 및 제거](#)를 참조하세요.

Kinesis Data Streams로 내보내기

Kinesis Data Streams로 내보내는 스트림을 생성하려면 Greengrass 구성 요소가 하나 이상의 `KinesisConfig` 객체를 포함하는 내보내기 정의가 있는 [스트림을 생성](#)합니다. 이 객체는 대상 데이터 스트림, 배치 크기, 배치 간격, 우선 순위와 같은 내보내기 설정을 정의합니다.

Greengrass 구성 요소는 디바이스로부터 데이터를 수신할 때 대상 스트림에 데이터 블록을 포함하는 [메시지를 추가](#)합니다. 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 임의의 UUID를 생성합니다.

AWS IoT SiteWise 자산 속성

스트림 관리자는 로 자동 내보내기를 지원합니다 AWS IoT SiteWise. AWS IoT SiteWise 를 사용하면 산업 장비에서 대규모로 데이터를 수집, 구성 및 분석할 수 있습니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 [What is AWS IoT SiteWise?](#)를 참조하세요.

스트림 관리자 SDK에서 Greengrass 구성 요소는 `IoTSiteWiseConfig`를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하세요.

- Python SDK에서의 [IoTSiteWiseConfig](#)
- Java SDK의 [IoTSiteWiseConfig](#)
- Node.js SDK의 [IoTSiteWiseConfig](#)

Note

AWS 는 OPC-UA 소스에서 데이터를 스트리밍하는 데 사용할 수 있는 사전 구축된 솔루션을 제공하는 AWS IoT SiteWise 구성 요소도 제공합니다. 자세한 내용은 [IoT SiteWise OPC UA 수 집기](#) 단원을 참조하십시오.

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 의 대상 자산 속성은 Greengrass 코어 디바이스와 동일한 AWS 계정 및 AWS 리전 에 AWS IoT SiteWise 있어야 합니다.

Note

에서 AWS 리전 AWS IoT SiteWise 지원하는 목록은 AWS 일반 참조의 [AWS IoT SiteWise 엔드포인트 및 할당량을 참조하세요.](#)

- [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#)은 대상 자산 속성에 `iotsitewise:BatchPutAssetPropertyValue` 권한을 허용해야 합니다. 다음 예제의 정책은 `iotsitewise:assetHierarchyPath` 조건 키를 사용하여 대상 루트 자산과 그 하위 자산에 대한

액세스 권한을 부여합니다. 정책Condition에서를 제거하여 모든 AWS IoT SiteWise 자산에 대한 액세스를 허용하거나 개별 자산의 ARNs 지정할 수 있습니다.

JSON

```
{
  "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 제거](#)를 참조하세요.

중요한 보안 정보는 AWS IoT SiteWise 사용 설명서의 [BatchPutAssetPropertyValue 권한 부여](#)를 참조하세요.

로 내보내기 AWS IoT SiteWise

를 내보내는 스트림을 생성하기 위해 AWS IoT SiteWise Greengrass 구성 요소는 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 요소는 디바이스로부터 자산 속성을 수신할 때 대상 스트림에 데이터를 포함하는 메시지를 추가합니다. 메시지는 하나 이상의 자산 속성에 대한 속성 값을 포함하는 JSON 직렬화 PutAssetPropertyValueEntry 객체입니다. 자세한 내용은 AWS IoT SiteWise 내보내기 대상에 대한 [메시지 추가](#)를 참조하세요.

Note

데이터를 전송할 때 AWS IoT SiteWise 데이터는 BatchPutAssetPropertyValue 작업의 요구 사항을 충족해야 합니다. 자세한 내용은 AWS IoT SiteWise API 참조의 [BatchPutAssetPropertyValue](#)를 참조하세요.

그런 다음 스트림 관리자는 스트림의 내보내기 구성에 정의된 배치 설정 및 우선순위에 따라 데이터를 내보냅니다.

스트림 관리자 설정과 Greengrass 구성 요소 논리를 조정하여 내보내기 전략을 설계할 수 있습니다. 예제:

- 거의 실시간으로 내보내려면 배치 크기 및 간격을 낮게 설정하고 스트림이 수신되면 스트림에 데이터를 추가하세요.
- 배치 처리를 최적화하고 대역폭 제약을 완화하거나 비용을 최소화하기 위해 Greengrass 구성 요소는 스트림에 데이터를 추가하기 전에 단일 자산 속성에 대해 수신한 타임스탬프-품질-값(TQV) 데이터 포인트를 풀링할 수 있습니다. 한 가지 전략은 동일한 속성에 대해 둘 이상의 항목을 보내는 대신 최대 10개의 서로 다른 속성-자산 조합 또는 속성 별칭에 대한 항목을 하나의 메시지로 일괄 처리하는 것입니다. 이렇게 하면 스트림 관리자가 [AWS IoT SiteWise 할당량](#) 이내로 유지할 수 있습니다.

Amazon S3 객체

스트림 관리자는 Amazon S3로의 자동 내보내기를 지원합니다. Amazon S3를 사용하여 대량의 데이터를 저장 및 검색할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 [Amazon S3란 무엇인가요?](#)를 참조하세요.

스트림 관리자 SDK에서 Greengrass 구성 요소는 S3ExportTaskExecutorConfig를 사용하여 이 대상 유형에 대한 내보내기 구성을 정의합니다. 자세한 내용은 대상 언어에 대한 SDK 참조를 확인하세요.

- Python SDK의 [S3ExportTaskExecutorConfig](#)
- Java SDK의 [S3ExportTaskExecutorConfig](#)
- Node.js SDK의 [S3ExportTaskExecutorConfig](#)

요구 사항

이 내보내기 대상은 다음 요구 사항을 충족해야 합니다.

- 대상 Amazon S3 버킷은 Greengrass 코어 디바이스 AWS 계정 와 동일한에 있어야 합니다.
- Greengrass 컨테이너 모드에서 실행되는 Lambda 함수가 입력 파일 디렉터리에 입력 파일을 쓰는 경우 해당 디렉터리를 쓰기 권한이 있는 컨테이너의 볼륨으로 탑재해야 합니다. 이렇게 하면 파일이 루트 파일 시스템에 기록되고 컨테이너 외부에서 실행되는 스트림 관리자 구성 요소에 표시됩니다.
- Docker 컨테이너 구성 요소가 입력 파일 디렉터리에 입력 파일을 쓰는 경우 해당 디렉터리를 쓰기 권한이 있는 컨테이너의 볼륨으로 탑재해야 합니다. 이렇게 하면 파일이 루트 파일 시스템에 기록되고 컨테이너 외부에서 실행되는 스트림 관리자 구성 요소에 표시됩니다.
- [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#)은 대상 버킷에 다음 권한을 허용해야 합니다. 예제:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-1-name/*",
        "arn:aws:s3:::bucket-2-name/*"
      ]
    }
  ]
}
```

리소스에 대한 세부적 또는 조건부 액세스 권한을 부여할 수 있습니다(예: 와일드카드 * 이름 지정 스키마를 사용해). 자세한 내용은 IAM 사용 설명서의 [IAM 정책 추가 및 제거](#)를 참조하세요.

Amazon S3로 내보내기

Amazon S3로 내보내는 스트림을 생성하려면 Greengrass 구성 요소에서 S3ExportTaskExecutorConfig 객체를 사용하여 내보내기 정책을 구성합니다. 정책은 멀티파트 업로드 임계값 및 우선 순위와 같은 내보내기 설정을 정의합니다. Amazon S3 내보내기의 경우 스트림 관리자는 코어 장치의 로컬 파일에서 읽은 데이터를 업로드합니다. 업로드를 시작하려면 Greengrass

구성 요소가 대상 스트림에 내보내기 태스크를 추가합니다. 내보내기 작업에는 입력 파일 및 대상 Amazon S3 객체에 대한 정보가 포함됩니다. 스트림 관리자는 스트림에 추가된 순서대로 태스크를 실행합니다.

Note

대상 버킷에는 이미 있어야 합니다 AWS 계정. 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}` 자리표시자에 유효한 [Java DateTimeFormatter](#) 문자열을 포함할 수 있습니다. 이러한 타임스탬프 자리표시자를 사용하여 입력 파일 데이터가 업로드된 시간을 기준으로 Amazon S3의 데이터를 분할할 수 있습니다. 예를 들어, 다음 키 이름은 `my-key/2020/12/31/data.txt`와 같은 값으로 해석됩니다.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

Note

스트림의 내보내기 상태를 모니터링하려면 먼저 상태 스트림을 만든 다음 이를 사용하도록 내보내기 스트림을 구성하세요. 자세한 내용은 [the section called “내보내기 작업 모니터링”](#) 단원을 참조하십시오.

입력 데이터 관리

IoT 애플리케이션이 입력 데이터의 라이프사이클을 관리하는 데 사용하는 코드를 작성할 수 있습니다. 다음 예제 워크플로는 Greengrass 구성 요소를 사용하여 이 데이터를 관리하는 방법을 보여줍니다.

1. 로컬 프로세스는 장치 또는 주변 기기로부터 데이터를 수신한 다음 코어 장치의 디렉터리에 있는 파일에 데이터를 씁니다. 스트림 관리자용 입력 파일입니다.
2. Greengrass 구성 요소는 디렉터리를 스캔하고 새 파일이 생성되는 경우 대상 스트림에 [내보내기 태스크를 추가](#)합니다. 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 이미 있어야 합니다 AWS 계정. 지정된 키의 객체가 존재하지 않는 경우 스트림 관리자가 대신 객체를 생성합니다.
4. Greengrass 구성 요소는 상태 스트림에서 [메시지를 읽어](#) 내보내기 상태를 모니터링합니다. 내보내기 태스크가 완료되면 Greengrass 구성 요소는 해당 입력 파일을 삭제할 수 있습니다. 자세한 내용은 [the section called “내보내기 작업 모니터링”](#) 단원을 참조하십시오.

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. 예제:

- 상태 메시지를 저장하지 않으려는 경우 Persistence를 Memory로 설정합니다.
- 새 상태 메시지가 손실되지 않도록 StrategyOnFull을 OverwriteOldestData로 설정합니다.

그런 다음, 상태 스트림을 사용하도록 내보내기 스트림을 생성하거나 업데이트하세요. 특히 스트림의 S3ExportTaskExecutorConfig 내보내기 구성의 상태 구성 속성을 설정하세요. 이 설정은 스트림 관리자가 내보내기 태스크에 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 INFO입니다.

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

다음 예제 워크플로는 Greengrass 구성 요소가 상태 스트림을 사용하여 내보내기 상태를 모니터링하는 방법을 보여줍니다.

1. 이전 워크플로에서 설명한 대로 Greengrass 구성 요소는 내보내기 태스크에 대한 상태 메시지를 상태 스트림에 기록하도록 구성된 스트림에 [내보내기 태스크를 추가](#)합니다. 추가 작업은 작업 ID를 나타내는 시퀀스 번호를 반환합니다.
2. Greengrass 구성 요소는 상태 스트림에서 순차적으로 [메시지를 읽은](#) 다음 스트림 이름과 태스크 ID 또는 메시지 컨텍스트의 내보내기 태스크 속성을 기준으로 메시지를 필터링합니다. 예를 들어 Greengrass 구성 요소는 메시지 컨텍스트에서 S3ExportTaskDefinition 객체에 의해 표시되는 내보내기 태스크의 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 태스크의 TTL(Time-to-Live) 기간이 만료되어 태스크가 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 업로드 정리에 실패하면 경고가 반환됩니다.

3. 내보내기 태스크가 완료되면 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: [read_messages](#) | [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.
}
}

```

Java SDK 참조: [readMessages](#) | [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: [readMessages](#) | [StatusMessage](#)

기계 학습 추론 수행

를 AWS IoT Greengrass 사용하면 클라우드 훈련 모델을 사용하여 로컬에서 생성된 데이터의 엣지 디바이스에서 기계 학습(ML) 추론을 수행할 수 있습니다. 이를 통해 로컬 추론 실행의 낮은 지연 시간과 비용 절감이라는 이점을 얻을 수 있습니다. 그러면서도 모델 훈련 및 복잡한 처리에 필요한 클라우드 컴퓨팅 능력을 활용할 수 있습니다.

AWS IoT Greengrass 는 추론을 수행하는 데 필요한 단계를 보다 효율적으로 만듭니다. 어디서나 추론 모델을 훈련하고 기계 학습 구성 요소로 로컬에 배포할 수 있습니다. 예를 들어 [Amazon SageMaker AI](#)에서 딥 러닝 모델을 구축하고 훈련할 수 있습니다. 그런 다음 이러한 모델을 [Amazon S3](#) 버킷에 저장할 수 있으므로 이러한 모델을 구성 요소의 아티팩트로 사용하여 코어 디바이스에서 추론을 수행할 수 있습니다.

주제

- [AWS IoT Greengrass ML 추론 작동 방식](#)
- [AWS IoT Greengrass 버전 2의 차이점은 무엇인가요?](#)
- [요구 사항](#)
- [지원되는 모델 소스](#)
- [지원되는 기계 학습 런타임](#)
- [AWS제공 기계 학습 구성 요소](#)
- [Greengrass 코어 디바이스에서 Amazon SageMaker AI Edge Manager 사용](#)
- [기계 학습 구성 요소 사용자 지정](#)
- [기계 학습 추론 문제 해결](#)

AWS IoT Greengrass ML 추론 작동 방식

AWS 는 디바이스에서 [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 생성할 수도 있습니다.

AWS 는 다음과 같은 기계 학습 구성 요소 범주를 제공합니다.

- 모델 구성 요소 - 기계 학습 모델을 Greengrass 아티팩트로 포함합니다.
- 런타임 구성 요소 - Greengrass 코어 디바이스에서 기계 학습 프레임워크 및 그 종속성을 설치하는 스크립트를 포함합니다.

- 추론 구성 요소 - 추론 코드를 포함하고 기계 학습 프레임워크를 설치하고 사전 훈련된 기계 학습 모델을 다운로드하기 위한 구성 요소 종속성을 포함합니다.

기계 학습 추론을 수행하기 위해 생성하는 각 배포는 추론 애플리케이션을 실행하고, 기계 학습 프레임워크를 설치하고, 기계 학습 모델을 다운로드하는 하나 이상의 구성 요소로 구성됩니다. AWS제공 구성 요소를 사용하여 샘플 추론을 수행하려면 해당 모델 및 런타임 구성 요소를 종속성으로 자동으로 포함하는 추론 구성 요소를 코어 디바이스에 배포합니다. 배포를 사용자 지정하려면 샘플 모델 구성 요소를 사용자 지정 모델 구성 요소로 연결하거나 교체하거나 AWS제공 구성 요소의 구성 요소 레시피를 템플릿으로 사용하여 사용자 지정 추론, 모델 및 런타임 구성 요소를 생성할 수 있습니다.

사용자 지정 구성 요소를 사용하여 기계 학습 추론을 수행하려면

1. 모델 구성 요소를 생성합니다. 이 구성 요소에는 추론을 수행하는 데 사용할 기계 학습 모델이 포함되어 있습니다.는 사전 훈련된 DLR 및 TensorFlow Lite 모델 샘플을 AWS 제공합니다. 사용자 지정 모델을 사용하려면 모델 구성 요소를 직접 생성합니다.
2. 런타임 구성 요소를 생성합니다. 이 구성 요소에는 모델에 기계 학습 런타임을 설치하는 데 필요한 스크립트가 포함되어 있습니다.는 DLR(Deep [Learning Runtime](#)) 및 [TensorFlow Lite](#)용 샘플 런타임 구성 요소를 AWS 제공합니다. 사용자 지정 모델 및 추론 코드가 포함된 다른 런타임을 사용하려면 런타임 구성 요소를 직접 생성합니다.
3. 추론 구성 요소를 생성합니다. 이 구성 요소에는 추론 코드가 포함되어 있으며 모델 및 런타임 구성 요소를 종속성으로 포함합니다.는 DLR 및 TensorFlow Lite를 사용하여 이미지 분류 및 객체 감지를 위한 샘플 추론 구성 요소를 AWS 제공합니다. 다른 유형의 추론을 수행하거나 사용자 지정 모델 및 런타임을 사용하려면 추론 구성 요소를 직접 생성합니다.
4. 추론 구성 요소를 배포합니다. 이 구성 요소를 배포하면 도 모델 및 런타임 구성 요소 종속성을 AWS IoT Greengrass 자동으로 배포합니다.

AWS제공 구성 요소를 시작하려면 섹션을 참조하세요 [the section called “샘플 이미지 분류 추론 수행”](#).

사용자 지정 기계 학습 구성 요소를 생성하는 방법에 대한 자세한 내용은 [기계 학습 구성 요소 사용자 지정](#) 섹션을 참조하세요.

AWS IoT Greengrass 버전 2의 차이점은 무엇인가요?

AWS IoT Greengrass 는 모델, 런타임 및 추론 코드와 같은 기계 학습용 기능 단위를 구성 요소로 통합하여 1단계 프로세스를 사용하여 기계 학습 런타임을 설치하고, 훈련된 모델을 다운로드하고, 디바이스에서 추론을 수행할 수 있습니다.

AWS제공 기계 학습 구성 요소를 사용하면 샘플 추론 코드 및 사전 훈련된 모델을 사용하여 기계 학습 추론 수행을 유연하게 시작할 수 있습니다. 사용자 지정 모델 구성 요소를 연결하여가 AWS 제공하는 추론 및 런타임 구성 요소와 함께 사용자 지정 훈련 모델을 사용할 수 있습니다. 완전히 사용자 지정된 기계 학습 솔루션이 필요한 경우 퍼블릭 구성 요소를 템플릿으로 사용하여 사용자 지정 구성 요소를 생성하고 원하는 런타임, 모델 또는 추론 유형을 사용할 수 있습니다.

요구 사항

기계 학습 구성 요소를 생성하고 사용하려면 다음이 있어야 합니다.

- Greengrass 코어 디바이스. 없으면 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요.
- AWS제공 샘플 기계 학습 구성 요소를 사용하기 위한 최소 500MB의 로컬 스토리지 공간.

지원되는 모델 소스

AWS IoT Greengrass 는 Amazon S3에 저장된 사용자 지정 훈련된 기계 학습 모델 사용을 지원합니다. 또한 Amazon SageMaker AI 엣지 패키징 작업을 사용하여 SageMaker AI Neo로 컴파일된 모델의 모델 구성 요소를 직접 생성할 수도 있습니다. 에서 SageMaker AI Edge Manager를 사용하는 방법에 대한 자세한 내용은 섹션을 AWS IoT Greengrass참조하세요 [Greengrass 코어 디바이스에서 Amazon SageMaker AI Edge Manager 사용](#).

모델이 포함되는 S3 버킷은 다음과 같은 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 보호](#)를 참조하세요.
- 이름에 마침표(.)를 포함하면 안 됩니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [버킷 이름 지정 규칙](#)에서 SSL과 함께 가상 호스팅 스타일 버킷을 사용하는 것과 관련된 규칙을 참조하세요.
- 모델 소스를 저장하는 S3 버킷은 AWS 리전 기계 학습 구성 요소와 동일한 AWS 계정 밑에 있어야 합니다.
- AWS IoT Greengrass 에는 모델 소스에 대한 read 권한이 있어야 합니다. AWS IoT Greengrass 가 S3 버킷에 액세스할 수 있도록 하려면 [Greengrass 디바이스 역할](#)이 s3:GetObject 작업을 허용해야 합니다. 디바이스 역할에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

지원되는 기계 학습 런타임

AWS IoT Greengrass 를 사용하면 사용자 지정 구성 요소를 생성하여 원하는 기계 학습 런타임을 사용하여 사용자 지정 훈련된 모델로 기계 학습 추론을 수행할 수 있습니다. 사용자 지정 기계 학습 구성 요소를 생성하는 방법에 대한 자세한 내용은 [기계 학습 구성 요소 사용자 지정](#) 섹션을 참조하세요.

기계 학습을 더 효율적으로 시작하는 프로세스를 만들기 위해서는 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 Lite](#) v2.5.0

AWS제공 기계 학습 구성 요소

다음 표에는 기계 학습에 사용되는 AWS제공 구성 요소가 나열되어 있습니다.

Note

여러 AWS제공 구성 요소는 Greengrass nucleus의 특정 마이너 버전에 따라 달라집니다. 이 종속성 때문에 Greengrass nucleus를 새로운 마이너 버전으로 업데이트할 때 이러한 구성 요소를 업데이트해야 합니다. 각 구성 요소가 달라지는 특정 nucleus 버전에 대한 내용은 해당 구성 요소 주제를 참조하세요. nucleus 업데이트에 대한 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어\(OTA\) 업데이트](#) 섹션을 참조하세요.

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
SageMaker AI Edge Manager	Greengrass 코어 디바이스에 Amazon SageMaker AI Edge Manager 에이전트를 배포합니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
DLR 이미지 분류	DLR 이미지 분류 모델 저장소와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 디바이스에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 객체 감지	DLR 객체 감지 모델 저장소와 DLR 런타임 구성 요소를 종속성으로 사용하여 DLR을 설치하고, 샘플 객체 감지 모델을 다운로드하고, 지원되는 디바이스에서 객체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
DLR 이미지 분류 모델 저장소	샘플 ResNet-50 이미지 분류 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 객체 감지 모델 저장소	샘플 YOLOv3 객체 감지 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
DLR 런타임	Greengrass 코어 디바이스에 DLR 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트를 포함하는 런타임 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow Lite 이미지 분류	TensorFlow Lite 이미지 분류 모델 저장소와 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 이미지 분류 모델을 다운로드하고, 지원되는 디바이스에서 이미지 분류 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow Lite 객체 감지	TensorFlow Lite 객체 감지 모델 저장소와 TensorFlow Lite 런타임 구성 요소를 종속성으로 사용하여 TensorFlow Lite를 설치하고, 샘플 객체 감지 모델을 다운로드하고, 지원되는 디바이스에서 객체 감지 추론을 수행하는 추론 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow Lite 이미지 분류 모델 저장소	샘플 MobileNet v1 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요

구성 요소	설명	구성 요소 유형	지원되는 OS	오픈 소스
TensorFlow Lite 객체 감지 모델 저장소	샘플 SSD(Single Shot Detection) MobileNet 모델을 Greengrass 아티팩트로 포함하는 모델 구성 요소입니다.	일반	Linux, Windows	아니요
TensorFlow Lite 런타임	Greengrass 코어 디바이스에 TensorFlow Lite 및 해당 종속성을 설치하는 데 사용되는 설치 스크립트를 포함하는 런타임 구성 요소입니다.	일반	Linux, Windows	아니요

Greengrass 코어 디바이스에서 Amazon SageMaker AI Edge Manager 사용

Important

SageMaker AI Edge Manager는 2024년 4월 26일에 중단되었습니다. 엣지 디바이스에 모델을 계속 배포하는 방법에 대한 자세한 내용은 [SageMaker AI Edge Manager 수명 종료](#)를 참조하세요.

Amazon SageMaker AI Edge Manager는 엣지 디바이스에서 실행되는 소프트웨어 에이전트입니다. SageMaker AI Edge Manager는 엣지 디바이스에 대한 모델 관리를 제공하므로 Greengrass 코

어 디바이스에서 Amazon SageMaker AI Neo 컴파일 모델을 직접 패키징하고 사용할 수 있습니다. SageMaker AI Edge Manager를 사용하면 코어 디바이스에서 모델 입력 및 출력 데이터를 샘플링하고 모니터링 및 분석을 AWS 클라우드 위해 해당 데이터를 로 보낼 수도 있습니다. SageMaker AI Edge Manager는 SageMaker AI Neo를 사용하여 대상 하드웨어에 맞게 모델을 최적화하므로 디바이스에 DLR 런타임을 직접 설치할 필요가 없습니다. Greengrass 디바이스에서 SageMaker AI Edge Manager는 로컬 AWS IoT 인증서를 로드하거나 AWS IoT 자격 증명 공급자 엔드포인트를 직접 호출하지 않습니다. 대신 SageMaker AI Edge Manager는 [토큰 교환 서비스](#)를 사용하며 TES 엔드포인트에서 임시 자격 증명을 가져옵니다.

이 섹션에서는 Greengrass 코어 디바이스에서 SageMaker AI Edge Manager가 작동하는 방법을 설명합니다.

Greengrass 디바이스에서 SageMaker AI Edge Manager가 작동하는 방식

SageMaker AI Edge Manager 에이전트를 코어 디바이스에 배포하려면 `aws.greengrass.SageMakerEdgeManager` 구성 요소가 포함된 배포를 생성합니다.는 디바이스에서 Edge Manager 에이전트의 설치 및 수명 주기를 AWS IoT Greengrass 관리합니다. 새 버전의 에이전트 바이너리를 사용할 수 있는 경우 `aws.greengrass.SageMakerEdgeManager` 구성 요소의 업데이트된 버전을 배포하여 디바이스에 설치된 에이전트의 버전을 업그레이드합니다.

SageMaker AI Edge Manager와 함께 사용하는 경우 AWS IoT Greengrass 워크플로에는 다음과 같은 상위 수준 단계가 포함됩니다.

1. SageMaker AI Neo로 모델을 컴파일합니다.
2. SageMaker AI 엣지 패키징 작업을 사용하여 SageMaker AI Neo 컴파일 모델을 패키징합니다. 모델에 대한 엣지 패키징 작업을 실행하는 경우 패키징된 모델을 Greengrass 코어 디바이스에 배포할 수 있는 아티팩트로 포함하는 모델 구성 요소를 생성하도록 선택할 수 있습니다.
3. 사용자 지정 추론 구성 요소를 생성합니다. 이 추론 구성 요소를 사용하여 Edge Manager 에이전트와 상호 작용하고 코어 디바이스에서 추론을 수행합니다. 이러한 작업은 모델 로드, 추론 실행을 위한 예측 요청 호출, 구성 요소 종료 시 모델 언로드를 포함합니다.
4. SageMaker AI Edge Manager 구성 요소, 패키징된 모델 구성 요소, 추론 구성 요소를 배포하여 디바이스의 SageMaker AI 추론 엔진(Edge Manager 에이전트)에서 모델을 실행합니다.

SageMaker AI Edge Manager와 함께 작동하는 엣지 패키징 작업 및 추론 구성 요소 생성에 대한 자세한 내용은 Amazon SageMaker AI 개발자 안내서의 [Deploy Model Package and Edge Manager Agent with AWS IoT Greengrass](#)를 참조하세요.

이 [자습서: SageMaker AI Edge Manager 시작하기](#) 자습서에서는 샘플 추론 및 모델 구성 요소를 생성하는 데 사용할 수 있는 AWS제공 예제 코드를 사용하여 기존 Greengrass 코어 디바이스에서 SageMaker AI Edge Manager 에이전트를 설정하고 사용하는 방법을 보여줍니다.

Greengrass 코어 디바이스에서 SageMaker AI Edge Manager를 사용하는 경우 데이터 캡처 기능을 사용하여 샘플 데이터를 AWS 클라우드에 업로드할 수도 있습니다. 캡처 데이터는 향후 분석을 위해 추론 입력, 추론 결과, 추가 추론 데이터를 S3 버킷 또는 로컬 디렉터리에 업로드하는 데 사용하는 SageMaker AI 기능입니다. SageMaker AI Edge Manager에서 캡처 데이터를 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker AI 개발자 안내서의 [모델 관리](#)를 참조하세요.

요구 사항

Greengrass 코어 디바이스에서 SageMaker AI Edge Manager 에이전트를 사용하려면 다음과 같은 요구 사항을 충족해야 합니다.

- Amazon Linux 2, Debian 기반 Linux 플랫폼(x86_64 또는 Armv8) 또는 Windows(x86_64)에서 실행되는 Greengrass 코어 디바이스. 없으면 [자습서: AWS IoT Greengrass V2 시작하기](#) 섹션을 참조하세요.
- [Python](#) 3.6 이상(사용하는 Python 버전에 대한 pip 포함)이 코어 디바이스에 설치되어 있어야 합니다.
- 다음과 같이 구성된 [Greengrass 디바이스 역할](#):
 - 다음 IAM 정책 예제와 같이 `credentials.iot.amazonaws.com` 및 `sagemaker.amazonaws.com`에서 역할을 수임하도록 허용하는 신뢰 관계.

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

```
    }
  ]
}
```

- [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM 관리형 정책.
- 다음 IAM 정책 예제와 같은 s3:PutObject 작업.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Greengrass 코어 디바이스 AWS 리전 와 동일한 AWS 계정 밑에서 생성된 Amazon S3 버킷입니다. SageMaker AI Edge Manager에는 엣지 디바이스 플릿을 생성하고 디바이스에서 추론을 실행하지 않는 샘플 데이터를 저장하기 위한 S3 버킷이 필요합니다. S3 버킷 생성에 대한 자세한 내용은 [Amazon S3 시작하기](#)를 참조하세요.
- Greengrass 코어 디바이스와 동일한 AWS IoT 역할 별칭을 사용하는 SageMaker AI 엣지 디바이스 플릿입니다. 자세한 내용은 [엣지 디바이스 플릿 생성](#) 단원을 참조하십시오.
- SageMaker AI Edge 디바이스 플릿에 엣지 디바이스로 등록된 Greengrass 코어 디바이스. 엣지 디바이스 이름은 코어 디바이스의 AWS IoT 사물 이름과 일치해야 합니다. 자세한 내용은 [Greengrass 코어 디바이스 등록](#) 단원을 참조하십시오.

SageMaker AI Edge Manager 시작하기

자습서를 완료하여 SageMaker AI Edge Manager 사용을 시작할 수 있습니다. 이 자습서에서는 기존 코어 디바이스에서 AWS제공 샘플 구성 요소와 함께 SageMaker AI Edge Manager 사용을 시작하는 방법을 보여줍니다. 이러한 샘플 구성 요소는 SageMaker AI Edge Manager 구성 요소를 종속성으로 사용하여 Edge Manager 에이전트를 배포하고 SageMaker AI Neo를 사용하여 컴파일된 사전 훈련 모

델로 추론을 수행합니다. 자세한 내용은 [자습서: SageMaker AI Edge Manager 시작하기](#) 단원을 참조하십시오.

기계 학습 구성 요소 사용자 지정

에서는 추론 AWS IoT Greengrass, 모델 및 런타임 [구성 요소를 구성 요소로 사용하여 디바이스에서 기계 학습 추론을 수행하는 방법을 사용자 지정](#)하도록 샘플 기계 학습 구성 요소를 구성할 수 있습니다. AWS IoT Greengrass 또한 샘플 구성 요소를 템플릿으로 사용하고 필요에 따라 사용자 지정 구성 요소를 생성할 수 있는 유연성을 제공합니다. 다음과 같이 이 모듈식 접근 방식을 혼합하여 기계 학습 추론 구성 요소를 사용자 지정할 수 있습니다.

샘플 추론 구성 요소 사용

- 추론 구성 요소를 배포할 때 해당 구성 요소의 구성을 수정합니다.
- 샘플 모델 저장소 구성 요소를 사용자 지정 모델 구성 요소로 대체하여 샘플 추론 구성 요소와 함께 사용자 지정 모델을 사용합니다. 사용자 지정 모델은 샘플 런타임을 샘플 모델로 사용하여 훈련시켜야 합니다.

사용자 지정 추론 구성 요소 사용

- 퍼블릭 모델 구성 요소 및 런타임 구성 요소를 사용자 지정 추론 구성 요소의 종속성으로 추가하여 샘플 모델 및 런타임과 함께 사용자 지정 추론 코드를 사용합니다.
- 사용자 지정 모델 구성 요소 또는 런타임 구성 요소를 생성하고 사용자 지정 추론 구성 요소의 종속성으로 추가합니다. 에서 샘플 구성 요소를 제공하지 AWS IoT Greengrass 않는 사용자 지정 추론 코드 또는 런타임을 사용하려면 사용자 지정 구성 요소를 사용해야 합니다.

주제

- [퍼블릭 추론 구성 요소의 구성 수정](#)
- [샘플 추론 구성 요소와 함께 사용자 지정 모델 사용](#)
- [사용자 지정 기계 학습 구성 요소 생성](#)
- [사용자 지정 추론 구성 요소 생성](#)

퍼블릭 추론 구성 요소의 구성 수정

[AWS IoT Greengrass 콘솔](#)에서 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"
  }
}
```

퍼블릭 추론 구성 요소를 배포할 때 기본 구성을 수정하여 배포를 사용자 지정할 수 있습니다. 각 퍼블릭 추론 구성 요소에 사용 가능한 구성 파라미터에 대한 자세한 내용은 [AWS제공 기계 학습 구성 요소](#)의 구성 요소 주제를 참조하세요.

이 섹션에서는 AWS IoT Greengrass 콘솔에서 수정된 구성 요소를 배포하는 방법을 설명합니다. 를 사용하여 구성 요소를 배포하는 방법에 대한 자세한 내용은 섹션을 AWS CLI참조하세요 [배포 만들기](#).

수정된 퍼블릭 추론 구성 요소를 배포하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#)에 로그인합니다.
2. 탐색 메뉴에서 구성 요소를 선택합니다.
3. 구성 요소 페이지의 퍼블릭 구성 요소 탭에서 배포할 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 퍼블릭 구성 요소에서 구성이 수정된 추론 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
 8. 구성 요소 구성 페이지에서 다음을 수행합니다.
 - 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 요소에 `/custom-ml-inference/images/` 폴더에서 `custom.jpg`라는 이미지를 찾으도록 지시합니다.
- ```
{
 "InferenceInterval": "15",
 "ImageName": "custom.jpg",
 "ImageDirectory": "/custom-ml-inference/images/"
}
```
- 구성 요소의 전체 구성을 기본값으로 재설정하려면 경로 재설정 상자에 빈 문자열 하나 `""`를 지정합니다.
- c. 확인을 선택하고 다음을 선택합니다.
  9. 고급 설정 구성 페이지에서 기본 구성 설정을 유지하고 다음을 선택합니다.
  10. 검토 페이지에서 배포를 선택합니다.

## 샘플 추론 구성 요소와 함께 사용자 지정 모델 사용

샘플 런타임 구성 요소를 AWS IoT Greengrass 제공하는 런타임에 자체 기계 학습 모델과 함께 샘플 추론 구성 요소를 사용하려면 해당 모델을 아티팩트로 사용하는 구성 요소로 퍼블릭 모델 구성 요소를 재정의해야 합니다. 대략적으로 다음 단계를 완료하여 사용자 지정 모델을 샘플 추론 구성 요소와 함께 사용합니다.

1. S3 버킷에서 사용자 지정 모델을 아티팩트로 사용하는 모델 구성 요소를 생성합니다. 사용자 지정 모델은 교체하려는 모델과 동일한 런타임을 사용하여 훈련해야 합니다.
2. 사용자 지정 모델을 사용하도록 추론 구성 요소의 `ModelResourceKey` 구성 파라미터를 수정합니다. 추론 구성 요소의 구성 업데이트에 대한 자세한 내용은 [퍼블릭 추론 구성 요소의 구성 수정](#) 섹션을 참조하세요.

추론 구성 요소를 배포할 때는 구성 요소 종속성의 최신 버전을 AWS IoT Greengrass 찾습니다. 구성 요소의 이후 사용자 지정 버전이 동일한 AWS 계정 및에 있는 경우 종속 퍼블릭 모델 구성 요소를 재정의합니다 AWS 리전.

### 사용자 지정 구성 요소 생성(콘솔)

1. S3 버킷에 모델을 업로드합니다. S3 버킷에 모델 업로드에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [Amazon S3 버킷을 사용한 작업](#)을 참조하세요.

#### Note

구성 요소와 동일한 AWS 계정 및에 있는 S3 버킷에 아티팩트 AWS 리전 를 저장해야 합니다. 가 이러한 아티팩트 AWS IoT Greengrass 에 액세스할 수 있도록 하려면 [Greengrass 디바이스 역할](#)이 `s3:GetObject` 작업을 허용해야 합니다. 디바이스 역할에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

2. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
3. 퍼블릭 모델 저장소 구성 요소의 구성 요소 레시피를 검색합니다.
  - a. 구성 요소 페이지의 퍼블릭 구성 요소 탭에서 새 버전을 생성할 퍼블릭 모델 구성 요소를 찾아 선택합니다. 예를 들어 `variant.DLR.ImageClassification.ModelStore`입니다.
  - 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 업데이트합니다.

- `ComponentVersion`: 구성 요소의 부 버전을 늘립니다.



사용자 지정 구성 요소를 생성하여 퍼블릭 모델 구성 요소를 재정의하는 경우 기존 구성 요소 버전의 부 버전만 업데이트해야 합니다. 예를 들어 퍼블릭 구성 요소 버전이 2.1.0인 경우 버전 2.1.1로 사용자 지정 구성 요소를 생성할 수 있습니다.

- Manifests.Artifacts.Uri: 각 URI 값을 사용하려는 모델의 Amazon S3 URI로 업데이트합니다.

#### Note

구성 요소의 이름은 변경하지 마세요.

### 8. 구성 요소 생성을 선택합니다.

#### 사용자 지정 모델 구성 요소 생성(AWS CLI)

1. S3 버킷에 모델을 업로드합니다. S3 버킷에 모델 업로드에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [Amazon S3 버킷을 사용한 작업](#)을 참조하세요.

#### Note

구성 요소와 동일한 AWS 계정 및에 있는 S3 버킷에 아티팩트 AWS 리전 를 저장해야 합니다. 가 이러한 아티팩트 AWS IoT Greengrass 에 액세스할 수 있도록 하려면 [Greengrass 디바이스 역할](#)이 s3:GetObject 작업을 허용해야 합니다. 디바이스 역할에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

2. 다음 명령을 실행하여 퍼블릭 구성 요소의 구성 요소 레시피를 검색합니다. 이 명령은 명령에서 제공하는 출력 파일에 구성 요소 레시피를 씁니다. 필요에 따라 검색된 base64 인코딩 문자열을 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 파일의 이름을 **<component-name>-<component-version>**로 업데이트합니다. 여기서 구성 요소 버전은 새 구성 요소의 대상 버전입니다. 예를 들어 `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`입니다.
4. 레시피에서 다음 값을 업데이트합니다.

- `ComponentVersion`: 구성 요소의 부 버전을 늘립니다.

사용자 지정 구성 요소를 생성하여 퍼블릭 모델 구성 요소를 재정의하는 경우 기존 구성 요소 버전의 부 버전만 업데이트해야 합니다. 예를 들어 퍼블릭 구성 요소 버전이 2.1.0인 경우 버전 2.1.1로 사용자 지정 구성 요소를 생성할 수 있습니다.

- `Manifests.Artifacts.Uri`: 각 URI 값을 사용하려는 모델의 Amazon S3 URI로 업데이트합니다.

### Note

구성 요소의 이름은 변경하지 마세요.

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 단계에서는의 AWS IoT Greengrass 서비스에 구성 요소를 생성합니다 AWS 클라우드. Greengrass CLI를 사용하여 구성 요소를 클라우드에 업로드하기 전에 로컬에서 개발, 테스트 및 배포할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#) 단원을 참조하십시오.

구성 요소를 생성하는 작업에 관한 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#) 섹션을 참조하십시오.

## 사용자 지정 기계 학습 구성 요소 생성

에서 샘플 구성 요소를 제공하지 AWS IoT Greengrass 애플리케이션 사용자 지정 추론 코드 또는 런타임을 사용하려면 사용자 지정 구성 요소를 생성해야 합니다. 사용자 지정 추론 코드를 AWS제공 샘플 기계 학습 모델 및 런타임과 함께 사용하거나 자체 모델 및 런타임으로 완전히 사용자 지정된 기계 학습 추론 솔루션을 개발할 수 있습니다. 모델이 샘플 런타임 구성 요소를 AWS IoT Greengrass 제공하는 런타임을 사용하는 경우 해당 런타임 구성 요소를 사용할 수 있으며 추론 코드와 사용하려는 모델에 대해서만 사용자 지정 구성 요소를 생성해야 합니다.

### 주제

- [퍼블릭 구성 요소의 레시피 검색](#)
- [샘플 구성 요소 아티팩트 검색](#)
- [S3 버킷에 구성 요소 아티팩트 업로드](#)
- [사용자 지정 구성 요소 생성](#)

### 퍼블릭 구성 요소의 레시피 검색

기존 퍼블릭 기계 학습 구성 요소의 레시피를 템플릿으로 사용하여 사용자 지정 구성 요소를 생성할 수 있습니다. 퍼블릭 구성 요소의 최신 버전에 대한 구성 요소 레시피를 보려면 다음과 AWS CLI 같이 콘솔 또는를 사용합니다.

- 콘솔 사용

1. 구성 요소 페이지의 퍼블릭 구성 요소 탭에서 퍼블릭 구성 요소를 찾아 선택합니다.
2. 구성 요소 페이지에서 레시피 보기를 선택합니다.

- 사용 AWS CLI

다음 명령을 실행하여 퍼블릭 변형 구성 요소의 구성 요소 레시피를 검색합니다. 이 명령은 명령에서 제공하는 JSON 또는 YAML 레시피 파일에 구성 요소 레시피를 씁니다.

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)입니다.
- **<recipe-format>**. 레시피 파일을 생성할 형식입니다. 지원되는 값은 JSON 및 YAML입니다.
- **<recipe-file>**. **<component-name>-<component-version>** 형식의 레시피 이름입니다.

## 샘플 구성 요소 아티팩트 검색

퍼블릭 기계 학습 구성 요소에 사용되는 아티팩트를 템플릿으로 사용하여 추론 코드 또는 런타임 설치 스크립트와 같은 사용자 지정 구성 요소 아티팩트를 생성할 수 있습니다.

퍼블릭 기계 학습 구성 요소에 포함된 샘플 아티팩트를 보려면 퍼블릭 추론 구성 요소를 배포한 다음 `/greengrass/v2/packages/artifacts-unarchived/component-name/component-version` 폴더의 디바이스에서 아티팩트를 확인합니다.

## S3 버킷에 구성 요소 아티팩트 업로드

사용자 지정 구성 요소를 생성하려면 먼저 구성 요소 아티팩트를 S3 버킷에 업로드하고 구성 요소 레시피에 S3 URI를 사용해야 합니다. 예를 들어, 추론 구성 요소에 사용자 지정 추론 코드를 사용하려면 코드를 S3 버킷에 업로드합니다. 그런 다음 추론 코드의 Amazon S3 URI를 구성 요소의 아티팩트로 사용할 수 있습니다.

S3 버킷에 콘텐츠 업로드에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [Amazon S3 버킷을 사용한 작업](#)을 참조하세요.

### Note

구성 요소와 동일한 AWS 계정 및에 있는 S3 버킷에 아티팩트 AWS 리전 를 저장해야 합니다. 가 이러한 아티팩트 AWS IoT Greengrass 에 액세스할 수 있도록 하려면 [Greengrass 디바이스 역할](#)이 `s3:GetObject` 작업을 허용해야 합니다. 디바이스 역할에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

## 사용자 지정 구성 요소 생성

검색한 아티팩트와 레시피를 사용하여 사용자 지정 기계 학습 구성 요소를 생성할 수 있습니다. 예제는 [사용자 지정 추론 구성 요소 생성](#) 섹션을 참조하세요.

구성 요소를 생성하고 Greengrass 디바이스에 배포하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#) 및 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.

## 사용자 지정 추론 구성 요소 생성

이 섹션에서는 DLR 이미지 분류 구성 요소를 템플릿으로 사용하여 사용자 지정 추론 구성 요소를 생성하는 방법을 보여줍니다.

## 주제

- [Amazon S3 버킷에 추론 코드 업로드](#)
- [추론 구성 요소에 대한 레시피 생성](#)
- [추론 구성 요소 생성](#)

## Amazon S3 버킷에 추론 코드 업로드

추론 코드를 생성한 다음 S3 버킷에 업로드합니다. S3 버킷에 콘텐츠 업로드에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [Amazon S3 버킷을 사용한 작업](#)을 참조하세요.

 Note

구성 요소와 동일한 AWS 계정 및에 있는 S3 버킷에 아티팩트 AWS 리전 를 저장해야 합니다. 가 이러한 아티팩트 AWS IoT Greengrass 에 액세스할 수 있도록 하려면 [Greengrass 디바이스 역할](#)이 s3:GetObject 작업을 허용해야 합니다. 디바이스 역할에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

## 추론 구성 요소에 대한 레시피 생성

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 씁니다.

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.DLRIImageClassification:versions
 `
 --recipe-output-format JSON | YAML `
 --query recipe `
 --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

*<recipe-file>*을 *<component-name>-<component-version>* 형식의 레시피 이름으로 바꿉니다.

- 레시피의 ComponentDependencies 객체에서 사용하려는 모델 및 런타임 구성 요소에 따라 다음 중 하나 이상을 수행합니다.
  - DLR 컴파일 모델을 사용하려면 DLR 구성 요소 종속성을 유지합니다. 다음 예제와 같이 사용자 지정 런타임 구성 요소의 종속성으로 교체할 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 유지하여에서 AWS 제공하는 사전 훈련된 ResNet-50 모델을 사용하거나 사용자 지정 모델 구성 요소를 사용하도록 수정합니다. 퍼블릭 모델 구성 요소에 대한 종속성을 포함하는 경우 구성 요소의 이후 사용자 지정 버전이 동일한에 있는 경우 AWS 계정 AWS 리전추론 구성 요소는 해당 사용자 지정 구성 요소를 사용합니다. 다음 예제와 같이 모델 구성 요소 종속성을 지정합니다.

### 퍼블릭 모델 구성 요소

#### JSON

```
{
 "variant.DLR.ImageClassification.ModelStore": {
 "VersionRequirement": "<version>",
 "DependencyType": "HARD"
 }
}
```

#### YAML

```
variant.DLR.ImageClassification.ModelStore:
 VersionRequirement: "<version>"
 DependencyType: HARD
```

### 사용자 지정 모델 구성 요소

#### JSON

```
{
 "<custom-model-component>": {
 "VersionRequirement": "<version>",
 "DependencyType": "HARD"
 }
}
```

#### YAML

```
<custom-model-component>:
 VersionRequirement: "<version>"
 DependencyType: HARD
```



3. ComponentConfiguration 객체에서 이 구성 요소의 기본 구성을 추가합니다. 나중에 구성 요소를 배포할 때 이 구성을 수정할 수 있습니다. 다음 발췌문에서는 DLR 이미지 분류 구성 요소의 구성 요소 구성을 보여줍니다.

예를 들어 사용자 지정 모델 구성 요소를 사용자 지정 추론 구성 요소의 종속성으로 사용하는 경우 ModelResourceKey를 수정하여 사용 중인 모델의 이름을 제공합니다.

## 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. Manifests 객체에서 구성 요소가 다른 플랫폼에 배포될 때 사용되는 아티팩트 및 이 구성 요소의 구성에 대한 정보와 구성 요소를 성공적으로 실행하는 데 필요한 기타 정보를 제공합니다. 다음 발췌문에서는 DLR 이미지 분류 구성 요소에서 Linux 플랫폼에 대한 Manifests 객체의 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 생성에 대한 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 섹션을 참조하세요.

## 추론 구성 요소 생성

AWS IoT Greengrass 콘솔 또는 AWS CLI 를 사용하여 방금 정의한 레시피를 사용하여 구성 요소를 생성합니다. 구성 요소를 생성한 후 배포하여 디바이스에서 추론을 수행할 수 있습니다. 추론 구성 요소

를 배포하는 방법의 예는 [자습서: TensorFlow Lite를 사용하여 샘플 이미지 분류 추론 수행](#) 섹션을 참조하세요.

### 사용자 지정 추론 구성 요소 생성(콘솔)

1. [AWS IoT Greengrass 콘솔](#)에 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 JSON으로 레시피 입력 또는 YAML로 레시피 입력을 구성 요소 소스로 선택합니다.
5. 레시피 상자에 생성한 사용자 지정 레시피를 입력합니다.
6. 구성 요소 생성을 클릭합니다.

### 사용자 지정 추론 구성 요소 생성(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 단계에서는의 AWS IoT Greengrass 서비스에 구성 요소를 생성합니다 AWS 클라우드. Greengrass CLI를 사용하여 구성 요소를 클라우드에 업로드하기 전에 로컬에서 개발, 테스트 및 배포할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 구성 요소 개발](#) 단원을 참조하십시오.

## 기계 학습 추론 문제 해결

이 섹션의 문제 해결 정보와 해결 방법을 사용하여 기계 학습 구성 요소 관련 문제를 해결하세요. 퍼블릭 기계 학습 추론 구성 요소에 대해서는 다음 구성 요소 로그의 오류 메시지를 참조하세요.

### 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 못함](#)
- [Cannot open shared object file](#)
- [Error: ModuleNotFoundError: No module named '<library>'](#)
- [CUDA 지원 디바이스가 감지되지 않음](#)
- [해당 파일 또는 디렉터리 없음](#)
- [RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>](#)
- [picamera.exc.PiCameraError: Camera is not enabled](#)
- [메모리 오류](#)
- [디스크 공간 오류](#)
- [제한 시간 오류](#)

## 라이브러리를 가져오지 못함

다음 오류는 설치 관리자 스크립트에서 Raspberry Pi 디바이스에 배포하는 동안 필요한 라이브러리를 다운로드하지 못할 때 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]
```

`sudo apt-get update`을 실행하고 구성 요소를 다시 배포합니다.

## Cannot open shared object file

설치 관리자 스크립트에서 Raspberry Pi 디바이스에 배포하는 동안 `opencv-python`에 필요한 종속성을 다운로드하지 못할 때 다음과 유사한 오류가 발생할 수 있습니다.

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

다음 명령을 실행하여 `opencv-python`에 대한 종속성을 수동으로 설치합니다.

```
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 올바르게 설치되지 않은 경우 ML 런타임 구성 요소 로그 (`variant.DLR.log` 또는 `variant.TensorFlowLite.log`)에 이 오류가 표시될 수 있습니다. 이 오류는 다음 경우에 발생할 수 있습니다.

- 기본적으로 활성화되는 `UseInstaller` 옵션을 사용하는 경우 이 오류는 ML 런타임 구성 요소가 런타임 또는 해당 종속성을 설치하지 못했음을 나타냅니다. 해결 방법:
  - `UseInstaller` 옵션을 비활성화하도록 ML 런타임 구성 요소를 구성합니다.
  - ML 런타임과 해당 종속성을 설치하고 ML 구성 요소를 실행하는 시스템 사용자가 사용할 수 있도록 합니다. 자세한 내용은 다음을 참조하세요.
    - [DLR 런타임 UseInstaller 옵션](#)
    - [TensorFlow Lite 런타임 UseInstaller 옵션](#)
- `UseInstaller` 옵션을 사용하지 않는 경우 이 오류는 ML 런타임 또는 해당 종속 항목이 ML 구성 요소를 실행하는 시스템 사용자에게 설치되지 않았음을 나타냅니다. 해결 방법:
  - ML 구성 요소를 실행하는 시스템 사용자에게 대해 라이브러리가 설치되어 있는지 확인합니다. `ggc_user`를 시스템 사용자의 이름으로 바꾸고 `tflite_runtime`을 확인할 라이브러리의 이름으로 바꿉니다.

## 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\""
```

- 라이브러리가 설치되지 않은 경우 해당 사용자에게 대해 라이브러리를 설치합니다. *ggc\_user*를 시스템 사용자의 이름으로 바꾸고 *tflite\_runtime*을 라이브러리의 이름으로 바꿉니다.

## 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 참조하세요.

- [DLR 런타임 UseInstaller 옵션](#)
- [TensorFlow Lite 런타임 UseInstaller 옵션](#)

- 문제가 지속되는 경우 다른 사용자에게 대해 라이브러리를 설치하여 이 디바이스가 라이브러리를 설치할 수 있는지 확인합니다. 예를 들어 사용자는 사용자, 루트 사용자 또는 관리 사용자일 수 있습니다. 어떤 사용자에게 대해서도 라이브러리를 설치할 수 없는 경우 디바이스가 라이브러리를 지원하지 않을 수 있습니다. 라이브러리 설명서를 참조하여 요구 사항을 검토하고 설치 문제를 해결합니다.

## CUDA 지원 디바이스가 감지되지 않음

GPU 가속화를 사용하면 다음 오류가 발생할 수 있습니다. 다음 명령을 실행하여 Greengrass 사용자의 GPU 액세스를 활성화합니다.

```
sudo usermod -a -G video ggc_user
```

## 해당 파일 또는 디렉터리 없음

다음 오류는 런타임 구성 요소가 가상 환경을 올바르게 설정할 수 없음을 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 확인합니다. 설치 관리자 스크립트에 서 설치한 라이브러리에 대한 자세한 내용은 다음 주제를 참조하세요.

- [DLR 런타임](#)
- [TensorFlow Lite 런타임](#)

기본적으로 `MLRootPath`는 `/greengrass/v2/work/component-name/greengrass_ml`로 설정됩니다. 이 위치를 변경하려면 배포에 [DLR 런타임](#) 또는 [TensorFlow Lite 런타임](#) 런타임 구성 요소를 직접 포함시키고 구성 병합 업데이트에서 `MLRootPath` 파라미터에 수정된 값을 지정합니다. 구성 요소 구성에 대한 자세한 내용은 [구성 요소 구성 업데이트](#) 섹션을 참조하세요.

### Note

DLR 구성 요소 v1.3.x의 경우 추론 구성 요소의 구성에서 `MLRootPath` 파라미터를 설정하며 기본값은 `$HOME/greengrass_ml`입니다.

## 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 버전보다 이전 버전의 NumPy가 포함되어 있기 때문에 발생합니다. 이 문제를 해결하려면 다음 명령을 실행하여 NumPy를 최신 버전으로 업그레이드합니다.

```
pip3 install --upgrade numpy
```

## picamera.exc.PiCameraError: Camera is not enabled

Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi에서 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 활성화합니다.

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 디바이스에 메모리가 충분하지 않고 구성 요소 프로세스가 중단될 때 발생합니다.

- `stderr. Killed.`
- `exitCode=137`

퍼블릭 기계 학습 추론 구성 요소를 배포하려면 최소 500MB의 메모리를 사용하는 것이 좋습니다.

## 디스크 공간 오류

no space left on device 오류는 일반적으로 디바이스에 스토리지가 부족할 때 발생합니다. 구성 요소를 다시 배포하기 전에 디바이스에 사용 가능한 디스크 공간이 충분한지 확인합니다. 퍼블릭 기계 학습 추론 구성 요소를 배포하려면 최소 500MB의 빈 디스크 공간을 사용하는 것이 좋습니다.

## 제한 시간 오류

퍼블릭 기계 학습 구성 요소는 200MB보다 큰 기계 학습 모델 파일을 다운로드합니다. 배포 중에 다운로드 시간이 초과되면 인터넷 연결 속도를 확인하고 배포를 다시 시도하세요.

# AWS Systems Manager를 사용하여 Greengrass 코어 디바이스 관리

## Note

AWS IoT Greengrass는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

Systems Manager는 Amazon EC2 인스턴스, 온프레미스 서버 및 가상 머신(VM), 엣지 디바이스를 포함하여 AWS에서 인프라를 보고 제어하는 데 사용할 수 있는 AWS 서비스입니다. Systems Manager를 사용하여 운영 데이터를 보고, 운영 태스크를 자동화하고, 보안 및 규정 준수를 유지 관리할 수 있습니다. Systems Manager에 등록된 시스템을 관리형 노드라고 합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [AWS Systems Manager란 무엇입니까?](#) 섹션을 참조하세요.

AWS Systems Manager 에이전트(Systems Manager 에이전트)는 Systems Manager에서 디바이스를 업데이트, 관리 및 구성할 수 있도록 하기 위해 디바이스에 설치할 수 있는 소프트웨어입니다. Greengrass 코어 디바이스에 Systems Manager 에이전트를 설치하려면 [Systems Manager 에이전트 구성 요소](#)를 배포합니다. Systems Manager 에이전트를 처음 배포하면 이 에이전트는 코어 디바이스를 Systems Manager 관리형 노드로 등록합니다. Systems Manager 에이전트는 디바이스에서 실행되어 AWS 클라우드의 Systems Manager 서비스와 통신할 수 있게 합니다. Systems Manager 에이전트 구성 요소를 설치하고 구성하는 방법에 대한 자세한 내용은 [AWS Systems Manager 에이전트 설치](#) 섹션을 참조하세요.

Systems Manager 도구 및 기능을 기능이라고 합니다. Greengrass 코어 디바이스는 모든 Systems Manager 기능을 지원합니다. 이러한 기능과 Systems Manager를 사용하여 코어 디바이스를 관리하는 방법에 대한 자세한 내용은 AWS Systems Manager 사용 설명서의 [Systems Manager 기능](#)을 참조하세요.

AWS Systems Manager는 Systems Manager 관리형 노드에 대한 표준 인스턴스 티어 및 고급 인스턴스 티어를 제공합니다. Systems Manager를 처음 사용하는 경우 표준 인스턴스 티어에서 시작합니다. 표준 인스턴스 티어에서는 AWS 계정에 AWS 리전당 최대 1,000개의 관리형 노드를 등록할 수 있습니다. 단일 계정 및 리전에 1,000개 이상의 관리형 노드를 등록해야 하거나 [Session Manager 기능](#)을 사용해야 하는 경우 고급 인스턴스 티어를 사용합니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [인스턴스 티어 구성](#)을 참조하세요.

## 주제

- [AWS Systems Manager 에이전트 설치](#)

- [AWS Systems Manager 에이전트 제거](#)

## AWS Systems Manager 에이전트 설치

AWS Systems Manager 에이전트(Systems Manager 에이전트)는 Systems Manager가 Greengrass 코어 디바이스, Amazon EC2 인스턴스 및 기타 리소스를 업데이트, 관리 및 구성할 수 있도록 설치한 Amazon 소프트웨어입니다. 이 에이전트는 AWS 클라우드에서 Systems Manager 서비스의 요청을 처리하고 실행합니다. 그런 다음 에이전트는 상태 및 런타임 정보를 Systems Manager 서비스로 다시 보냅니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [Systems Manager Agent](#)를 참조하세요.

AWS 는 Systems Manager Agent를 Greengrass 코어 디바이스에 배포하여 Systems Manager로 관리할 수 있는 Greengrass 구성 요소로 제공합니다. [Systems Manager 에이전트 구성 요소](#)는 Systems Manager 에이전트 소프트웨어를 설치하고 코어 디바이스를 Systems Manager에 관리형 노드로 등록합니다. 이 페이지의 단계에 따라 사전 조건을 완료하고 Systems Manager 에이전트 구성 요소를 코어 디바이스 또는 코어 디바이스 그룹에 배포하세요.

### 주제

- [1단계: 일반 Systems Manager 설정 단계 완료](#)
- [2단계: Systems Manager에 대한 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 수행하지 않은 경우에 대한 일반 설정 단계를 완료합니다 AWS Systems Manager. 자세한 내용은 AWS Systems Manager 사용 설명서의 [일반 Systems Manager 설정 단계 완료](#)를 참조하세요.

### 2단계: Systems Manager에 대한 IAM 서비스 역할 생성

Systems Manager Agent는 AWS Identity and Access Management (IAM) 서비스 역할을 사용하여와 통신합니다 AWS Systems Manager. Systems Manager는 이 역할을 수임하여 각 코어 디바이스에서 Systems Manager 기능을 활성화합니다. 또한 Systems Manager Agent 구성 요소는 구성 요소를 배포할 때 이 역할을 사용하여 코어 디바이스를 Systems Manager 관리형 노드로 등록합니다. 아직 하지 않은 경우 Systems Manager 에이전트 구성 요소가 사용할 Systems Manager 서비스 역할을 생성합니다

다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [엣지 디바이스에 대한 IAM 서비스 역할 생성](#)을 참조하세요.

### 3단계: 토큰 교환 역할에 권한 추가

Greengrass 코어 디바이스는 토큰 교환 역할이라는 IAM 서비스 역할을 사용하여 AWS 서비스와 상호 작용합니다. 각 코어 디바이스에는 [AWS IoT Greengrass 코어 소프트웨어를 설치할](#) 때 생성하는 토큰 교환 역할이 있습니다. Systems Manager 에이전트와 같은 많은 Greengrass 구성 요소에는 이 역할에 추가 권한이 필요합니다. Systems Manager 에이전트 구성 요소에는 [2단계: Systems Manager에 대한 IAM 서비스 역할 생성](#)에서 생성한 역할을 사용할 수 있는 권한을 포함하는 다음 권한이 필요합니다.

```
{
 "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 그렇게 하지 않은 경우 코어 디바이스의 토큰 교환 역할에 이러한 권한을 추가하여 Systems Manager 에이전트가 작동할 수 있도록 합니다. 새 정책을 토큰 교환 역할에 추가하여 이 권한을 부여할 수 있습니다.

토큰 교환 역할에 권한을 추가하려면(콘솔)

1. [IAM 콘솔](#) 탐색 메뉴에서 역할을 선택합니다.

2. AWS IoT Greengrass 코어 소프트웨어를 설치할 때 토큰 교환 역할로 설정한 IAM 역할을 선택합니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 때 토큰 교환 역할의 이름을 지정하지 않은 경우 라는 역할이 생성되었습니다 `GreengrassV2TokenExchangeRole`.
3. 권한에서 권한 추가, 정책 연결을 차례로 선택합니다.
4. 정책 생성을 선택합니다. 새 브라우저 탭에서 정책 생성 페이지가 열립니다.
5. 정책 생성 페이지에서 다음을 수행합니다.
  - a. JSON을 선택하여 JSON 편집기를 엽니다.
  - b. 다음 정책을 JSON 편집기에 붙여넣습니다. `SSMServiceRole`을 [2단계: Systems Manager에 대한 IAM 서비스 역할 생성](#)에서 생성한 서비스 역할의 이름으로 바꿉니다.

```
{
 "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. 정책의 이름을 입력합니다(예: **GreengrassSSMAgentComponentPolicy**).
- f. 정책 생성을 선택합니다.
- g. 토큰 교환 역할을 열어 둔 이전 브라우저 탭으로 전환합니다.

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 코어 디바이스는 이제 Systems Manager 서비스와 상호 작용할 수 있는 권한이 있습니다.

#### 토큰 교환 역할에 권한을 추가하려면(AWS CLI)

Systems Manager를 사용할 수 있는 권한을 부여하는 정책을 추가하려면

1. `ssm-agent-component-policy.json`이라는 파일을 만들고 다음 JSON을 파일로 복사합니다. *SSMSERVICE\_ROLE*을 [2단계: Systems Manager에 대한 IAM 서비스 역할 생성](#)에서 생성한 서비스 역할의 이름으로 바꿉니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Action": [
 "iam:PassRole"
],
 "Effect": "Allow",
 "Resource": [
 "arn:aws:iam::account-id:role/SSMSERVICE_ROLE"
]
 },
 {
 "Action": [
 "ssm:AddTagsToResource",
 "ssm:RegisterManagedInstance"
],
 "Effect": "Allow",
 "Resource": "*"
 }
]
}
```

2. 다음 명령을 실행하여 `ssm-agent-component-policy.json`의 정책 문서에서 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 정책 Amazon 리소스 이름(ARN)을 복사합니다. 다음 단계에서 이 ARN을 사용하여 이 정책을 코어 디바이스에 연결합니다.

3. 다음 명령을 실행하여 정책을 토큰 교환 역할에 연결합니다.

- *GreengrassV2TokenExchangeRole*을 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 지정한 토큰 교환 역할의 이름으로 바꿉니다. AWS IoT Greengrass 코어 소프트웨어를 설치할 때 토큰 교환 역할의 이름을 지정하지 않은 경우 라는 역할이 생성되었습니다GreengrassV2TokenExchangeRole.
- 정책 ARN을 이전 단계의 ARN으로 바꿉니다.

## 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
```

명령의 출력이 없으면 성공한 것입니다. 이 토큰 교환 역할을 사용하는 코어 디바이스는 이제 Systems Manager 서비스와 상호 작용할 수 있는 권한이 있습니다.

## 4단계: Systems Manager 에이전트 구성 요소 배포

Systems Manager 에이전트 구성 요소를 배포하고 구성하려면 다음 단계를 완료합니다. 구성 요소를 단일 코어 디바이스나 코어 디바이스 그룹에 배포할 수 있습니다.

Systems Manager 에이전트 구성 요소를 배포하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
2. 구성 요소 페이지의 퍼블릭 구성 요소 탭을 선택한 다음 `aws.greengrass.SystemsManagerAgent`를 선택합니다.
3. `aws.greengrass.SystemsManagerAgent` 페이지에서 배포를 선택합니다.
4. 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포 생성을 선택하고 다음을 선택합니다.
5. 새 배포 생성을 선택한 경우 배포의 대상 코어 디바이스 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 디바이스 또는 사물 그룹을 선택하고 다음을 선택합니다.
6. 구성 요소 선택 페이지에서 `aws.greengrass.SystemsManagerAgent` 구성 요소가 선택되었는지 확인하고 다음을 선택합니다.
7. 구성 요소 구성 페이지에서 `aws.greengrass.SystemsManagerAgent`를 선택하고 다음을 수행합니다.

- a. 구성 요소 구성을 선택합니다.
- b. 구성 `aws.greengrass.SystemsManagerAgent` 모달에서 구성 업데이트 아래의 병합할 구성에 다음 구성 업데이트를 입력합니다. ***SSMServiceRole***을 [2단계: Systems Manager에 대한 IAM 서비스 역할 생성](#)에서 생성한 서비스 역할의 이름으로 바꿉니다.

```
{
 "SSMRegistrationRole": "SSMServiceRole",
 "SSMOverrideExistingRegistration": false
}
```

#### Note

코어 디바이스가 하이브리드 활성화로 등록된 Systems Manager 에이전트를 이미 실행하고 있는 경우 `SSMOverrideExistingRegistration`을 `true`로 변경합니다. 이 파라미터는 Systems Manager 에이전트가 하이브리드 활성화를 사용하는 디바이스에서 이미 실행 중일 때 Systems Manager 에이전트 구성 요소가 코어 디바이스를 등록하는지 여부를 지정합니다.

또한 태그(`SSMResourceTags`)를 지정하여 Systems Manager 에이전트 구성 요소가 코어 디바이스에 대해 생성하는 Systems Manager 관리형 노드에 추가할 수도 있습니다. 자세한 내용은 [Systems Manager Agent component configuration](#)을 참조하세요.

- 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 1분 정도 걸릴 수 있습니다.

Systems Manager 에이전트 구성 요소를 배포하려면(AWS CLI)

Systems Manager 에이전트 구성 요소를 배포하려면 `components` 객체에

`aws.greengrass.SystemsManagerAgent`를 포함하는 배포 문서를 생성하고 구성 요소에 대한 구성 업데이트를 지정합니다. [배포 만들기](#)의 지침에 따라 새 배포를 생성하거나 기존 배포를 수정합니다.

다음의 부분 배포 문서 예제에서는 `SSMServiceRole`이라는 서비스 역할을 사용하도록 지정합니다. ***SSMServiceRole***을 [2단계: Systems Manager에 대한 IAM 서비스 역할 생성](#)에서 생성한 서비스 역할의 이름으로 바꿉니다.

```
{
 ...,
 "components": {
 ...,
 "aws.greengrass.SystemsManagerAgent": {
 "componentVersion": "1.0.0",
 "configurationUpdate": {
 "merge": "{\\"SSMRegistrationRole\\":\\"SSMSERVICERole\\",
\\"SSMOverrideExistingRegistration\\":false}"
 }
 }
 }
}
```

### Note

코어 디바이스가 하이브리드 활성화로 등록된 Systems Manager 에이전트를 이미 실행하고 있는 경우 SSMOverrideExistingRegistration을 true로 변경합니다. 이 파라미터는 Systems Manager 에이전트가 하이브리드 활성화를 사용하는 디바이스에서 이미 실행 중일 때 Systems Manager 에이전트 구성 요소가 코어 디바이스를 등록하는지 여부를 지정합니다. 또한 태그(SSMResourceTags)를 지정하여 Systems Manager 에이전트 구성 요소가 코어 디바이스에 대해 생성하는 Systems Manager 관리형 노드에 추가할 수도 있습니다. 자세한 내용은 [Systems Manager Agent component configuration](#)을 참조하세요.

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS IoT Greengrass 서비스를 사용하여 배포 상태를 확인할 수 있으며 코어 AWS IoT Greengrass 소프트웨어 로그 및 Systems Manager 에이전트 구성 요소 로그를 확인하여 Systems Manager 에이전트가 성공적으로 실행되는지 확인할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- [배포 상태 확인](#)
- [AWS IoT Greengrass 로그 모니터링](#)
- AWS Systems Manager 사용 설명서의 [Systems Manager 에이전트 로그 보기](#)

배포가 실패하거나 Systems Manager 에이전트가 실행되지 않는 경우 각 코어 디바이스에서 배포 문제를 해결할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- [문제 해결 AWS IoT Greengrass V2](#)

- AWS Systems Manager 사용 설명서의 [Systems Manager 에이전트 문제 해결](#)

## 5단계: Systems Manager를 사용하여 코어 디바이스 등록 확인

Systems Manager 에이전트 구성 요소가 실행되면 코어 디바이스를 Systems Manager의 관리형 노드로 등록합니다. AWS IoT Greengrass 콘솔, Systems Manager 콘솔 및 Systems Manager API를 사용하여 코어 디바이스가 관리형 노드로 등록되었는지 확인할 수 있습니다. 관리형 노드는 AWS 콘솔 및 API의 일부분에서 인스턴스라고도 합니다.

코어 디바이스 등록을 확인하려면(AWS IoT Greengrass 콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 코어 디바이스를 선택합니다.
2. 확인할 코어 디바이스를 선택합니다.
3. 코어 디바이스의 세부 정보 페이지에서 AWS Systems Manager 인스턴스 속성을 찾습니다. 이 속성이 있고 Systems Manager 콘솔에 대한 링크를 표시하면 코어 디바이스가 관리형 노드로 등록된 것입니다.

또한 AWS Systems Manager ping 상태 속성을 찾아 코어 디바이스에서 Systems Manager 에이전트의 상태를 확인할 수 있습니다. 상태가 온라인인 경우 Systems Manager를 사용하여 코어 디바이스를 관리할 수 있습니다.

코어 디바이스 등록을 확인하려면(Systems Manager 콘솔)

1. [Systems Manager 콘솔](#) 탐색 메뉴에서 Fleet Manager를 선택합니다.
2. 관리형 노드에서 다음을 수행합니다.
  - a. 소스 유형이 AWS::IoT::Thing인 필터를 추가합니다.
  - b. 소스 ID가 확인할 코어 디바이스의 이름인 필터를 추가합니다.
3. 관리형 노드 테이블에서 코어 디바이스를 찾습니다. 코어 디바이스가 테이블에 있는 경우 관리형 노드로 등록된 것입니다.

Systems Manager 에이전트 ping 상태 속성을 찾아 코어 디바이스에서 Systems Manager 에이전트의 상태를 확인할 수도 있습니다. 상태가 온라인인 경우 Systems Manager를 사용하여 코어 디바이스를 관리할 수 있습니다.

## 코어 디바이스 등록을 확인하려면(AWS CLI)

- [DescribeInstanceInformation](#) 작업을 사용하여 지정하는 필터와 일치하는 관리형 노드 목록을 가져옵니다. 다음 명령을 실행하여 코어 디바이스가 관리형 노드로 등록되어 있는지 확인합니다. *MyGreengrassCore*를 확인할 코어 디바이스의 이름으로 바꿉니다.

```
aws ssm describe-instance-information --filter
 Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
```

응답에는 필터와 일치하는 관리형 노드 목록이 포함됩니다. 목록에 관리형 노드가 포함된 경우 코어 디바이스는 관리형 노드로 등록된 것입니다. 응답에서 코어 디바이스의 관리형 노드에 대한 다른 정보도 찾을 수 있습니다. PingStatus 속성이 Online인 경우 Systems Manager를 사용하여 코어 디바이스를 관리할 수 있습니다.

코어 디바이스가 Systems Manager에서 관리형 노드로 등록되어 있는지 확인한 후 Systems Manager 콘솔 및 API를 사용하여 해당 코어 디바이스를 관리할 수 있습니다. Greengrass 코어 디바이스를 관리하는 데 사용할 수 있는 Systems Manager 기능에 대한 자세한 내용은 AWS Systems Manager 사용 설명서의 [Systems Manager 기능](#)을 참조하세요.

## AWS Systems Manager 에이전트 제거

더 이상 로 Greengrass 코어 디바이스를 관리하지 않으려면 Systems Manager에서 코어 디바이스의 등록을 취소하고 디바이스에서 AWS Systems Manager 에이전트(Systems Manager Agent)를 제거할 AWS Systems Manager 수 있습니다.

코어 디바이스는 언제든지 다시 등록할 수 있습니다. 이렇게 하려면 Systems Manager Agent 구성 요소를 다시 배포하여 코어 디바이스가 설치될 때 Systems Manager에 등록합니다. Systems Manager는 등록 취소된 코어 디바이스에 대한 명령 기록을 30일 동안 저장합니다.

### 주제

- [1단계: Systems Manager에서 코어 디바이스 등록 취소](#)
- [2단계: Systems Manager Agent 구성 요소 제거](#)
- [3단계: Systems Manager Agent 소프트웨어 제거](#)

## 1단계: Systems Manager에서 코어 디바이스 등록 취소

Systems Manager 콘솔 또는 API를 사용하여 코어 디바이스의 등록을 취소할 수 있습니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [Deregistering managed nodes](#)를 참조하세요.

## 2단계: Systems Manager Agent 구성 요소 제거

코어 디바이스의 등록을 취소한 후 디바이스에서 [Systems Manager Agent 구성 요소](#)를 제거합니다. Greengrass 코어 디바이스에서 구성 요소를 제거하려면 구성 요소를 설치한 배포를 수정하고 배포에서 구성 요소를 제거합니다. AWS IoT Greengrass 코어 디바이스의 배포에서 해당 구성 요소를 지정하지 않으면 코어 소프트웨어가 구성 요소를 제거합니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

Systems Manager Agent 구성 요소를 제거하려면(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 코어 디바이스를 선택합니다.
2. Systems Manager Agent 구성 요소를 제거하려는 코어 디바이스를 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 배포 탭을 선택합니다.
4. Systems Manager Agent 구성 요소를 코어 디바이스에 배포하는 배포를 선택합니다.
5. 배포 세부 정보 페이지에서 수정을 선택합니다.
6. 배포 수정 모달에서 배포 수정을 선택합니다.
7. 1단계: 대상 지정에서 다음을 선택합니다.
8. 2단계: 구성 요소 선택에서 `aws.greengrass.SystemsManagerAgent` 구성 요소에 대한 선택을 취소하고 다음을 선택합니다.
9. 3단계: 구성 요소 구성에서 다음을 선택합니다.
10. 4단계: 고급 설정 구성에서 다음을 선택합니다.
11. 5단계: 검토에서 배포를 선택합니다.

Systems Manager Agent 구성 요소를 제거하려면(CLI)

Systems Manager Agent 구성 요소를 제거하려면 해당 구성 요소를 배포하는 배포를 수정하고 이를 배포에서 제거합니다. 자세한 내용은 [배포 수정](#) 단원을 참조하십시오.

배포를 완료하는 데 몇 분 정도 걸릴 수 있습니다. AWS IoT Greengrass 서비스를 사용하여 배포 상태를 확인할 수 있습니다. 자세한 내용은 [배포 상태 확인](#) 단원을 참조하십시오.

## 3단계: Systems Manager Agent 소프트웨어 제거

Systems Manager Agent 구성 요소를 제거한 후에도 Systems Manager Agent 소프트웨어는 코어 디바이스에서 계속 실행됩니다. Systems Manager Agent 소프트웨어를 제거하려면 코어 디바이스에서 명령을 실행하면 됩니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 [Uninstall Systems Manager Agent from Linux instances](#)를 참조하세요.

# 의 보안 AWS IoT Greengrass

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS와 사용자 간의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다 AWS 클라우드.는 안전하게 사용할 수 있는 서비스 AWS 도 제공합니다. 타사 감사자는 [AWS 규정 준수 프로그램](#) 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 [AWS 제공 범위 내 서비스규정 준수 프로그램](#) 제공 범위 내 서비스를 AWS IoT Greengrass참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

를 사용할 때 디바이스 AWS IoT Greengrass, 로컬 네트워크 연결 및 프라이빗 키를 보호할 책임도 있습니다.

이 설명서는를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 AWS IoT Greengrass. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS IoT Greengrass 를 구성하는 방법을 보여줍니다. 또한 AWS IoT Greengrass 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

## 주제

- [의 데이터 보호 AWS IoT Greengrass](#)
- [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#)
- [에 대한 자격 증명 및 액세스 관리 AWS IoT Greengrass](#)
- [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#)
- [의 규정 준수 검증AWS IoT Greengrass](#)
- [FIPS 엔드포인트](#)
- [AWS IoT Greengrass의 복원성](#)
- [의 인프라 보안 AWS IoT Greengrass](#)
- [의 구성 및 취약성 분석 AWS IoT Greengrass](#)
- [AWS IoT Greengrass V2의 코드 무결성](#)



- [AWS IoT Greengrass 및 인터페이스 VPC 엔드포인트\(AWS PrivateLink\)](#)
- [에 대한 보안 모범 사례 AWS IoT Greengrass](#)

## 의 데이터 보호 AWS IoT Greengrass

AWS [공동 책임 모델](#)의 데이터 보호에 적용됩니다 AWS IoT Greengrass. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보안 구성 과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 [데이터 프라이버시 FAQ](#)를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 [AWS 공동 책임 모델 및 GDPR](#) 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 [CloudTrail 추적 작업을 참조하세요](#).
- 내의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [연방 정보 처리 표준\(FIPS\) 140-3](#)을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 AWS IoT Greengrass 또는 기타 AWS 서비스 에서 콘솔, API AWS CLI또는 AWS SDKs를 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 증명을 URL에 포함해서는 안 됩니다.

에서 민감한 정보를 보호하는 방법에 대한 자세한 내용은 섹션을 AWS IoT Greengrass 참조하세요 [the section called “민감한 정보를 기록하지 않음”](#).

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 [AWS 공동 책임 모델 및 GDPR](#) 블로그 게시물을 참조하십시오.

주제

- [데이터 암호화](#)
- [하드웨어 보안 통합](#)

## 데이터 암호화

AWS IoT Greengrass 는 암호화를 사용하여 전송 중(인터넷 또는 로컬 네트워크를 통해) 및 저장 중(에 저장) 데이터를 보호합니다 AWS 클라우드.

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 암호화](#)
- [저장 시 암호화](#)
- [Greengrass 코어 장치를 위한 키 관리](#)

## 전송 중 암호화

AWS IoT Greengrass에서는 두 가지 통신 모드로 데이터를 전송할 수 있습니다.

- [the section called “인터넷을 통해 전송 중인 데이터”](#). 인터넷을 통한 Greengrass 코어와 AWS IoT Greengrass 간 통신은 암호화됩니다.
- [the section called “코어 장치의 데이터”](#). Greengrass 코어 장치의 구성 요소 간 통신은 암호화되지 않습니다.

### 인터넷을 통해 전송 중인 데이터

AWS IoT Greengrass는 전송 계층 보안(TLS)을 사용하여 인터넷을 통한 모든 통신을 암호화합니다. AWS 클라우드로 전송되는 모든 데이터는 MQTT 또는 HTTPS 프로토콜을 사용하여 TLS 연결을 통해

전송되므로 기본적으로 안전합니다. AWS IoT Greengrass는 AWS IoT 전송 보안 모델을 사용합니다. 자세한 정보는 AWS IoT Core 개발자 안내서의 [전송 보안](#)을 참조하세요.

## 코어 장치의 데이터

데이터가 장치를 떠나지 않기 때문에 AWS IoT Greengrass는 Greengrass 코어 장치에서 로컬로 교환되는 데이터를 암호화하지 않습니다. 여기에는 사용자 정의 구성 요소, AWS IoT 디바이스 SDK 및 퍼블릭 구성 요소(예: 스트림 관리자) 간의 통신이 포함됩니다.

## 저장 시 암호화

AWS IoT Greengrass가 데이터를 저장합니다.

- [the section called “AWS 클라우드에 저장된 데이터”](#). 이 데이터는 암호화됩니다.
- [the section called “Greengrass 코어에 저장된 데이터”](#). 이 데이터는 암호화되지 않습니다(보안 암호의 로컬 사본 제외).

## AWS 클라우드에 저장된 데이터

AWS IoT Greengrass는 AWS 클라우드에 저장된 고객 데이터를 암호화합니다. 이 데이터는 AWS KMS에서 관리하는 AWS IoT Greengrass 키를 사용하여 보호됩니다.

## Greengrass 코어에 저장된 데이터

AWS IoT Greengrass는 Unix 파일 권한 및 전체 디스크 암호화(활성화된 경우)를 사용하여 코어에 저장된 유틸리티 상태의 데이터를 보호합니다. 파일 시스템 및 장치의 보안을 유지하는 것은 사용자의 책임입니다.

그러나 AWS IoT Greengrass는 AWS Secrets Manager에서 검색한 보안 암호의 로컬 복사본을 암호화합니다. 자세한 내용은 [보안 암호 관리자](#) 구성 요소를 참조하세요.

## Greengrass 코어 장치를 위한 키 관리

Greengrass 코어 장치에 암호화(퍼블릭 및 프라이빗) 키를 안전하게 저장하는 것은 고객의 책임입니다. AWS IoT Greengrass는 다음 시나리오에 퍼블릭 키와 프라이빗 키를 사용합니다.

- IoT 클라이언트 키는 IoT 인증서와 함께 사용되어 Greengrass 코어가 AWS IoT Core에 연결될 때 TLS(전송 계층 보안) 핸드셰이크를 인증합니다. 자세한 내용은 [the section called “장치 인증 및 권한 부여”](#) 섹션을 참조하세요.

**Note**

키와 인증서를 코어 프라이빗 키와 코어 장치 인증서라고도 합니다.

Greengrass 코어 디바이스는 파일 시스템 권한 또는 [하드웨어 보안 모듈](#)을 사용하여 프라이빗 키 스토리지를 지원합니다. 파일 시스템 기반 프라이빗 키를 사용하는 경우, 코어 장치의 보안 스토리지에 대한 책임은 사용자에게 있습니다.

## 하드웨어 보안 통합

**Note**

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

[PKCS#11 인터페이스](#)를 통해 하드웨어 보안 모듈(HSM)을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있습니다. 이 기능을 사용하면 디바이스의 프라이빗 키와 인증서를 소프트웨어에 노출되거나 복제되지 않도록 안전하게 저장할 수 있습니다. 프라이빗 키와 인증서를 HSM 또는 TPM(Trusted Platform Module)과 같은 하드웨어 모듈에 저장할 수 있습니다.

AWS IoT Greengrass 코어 소프트웨어는 프라이빗 키와 X.509 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 연결을 인증합니다. [보안 암호 관리자 구성 요소](#)는 이 프라이빗 키를 사용하여 Greengrass 코어 디바이스에 배포하는 보안 암호를 안전하게 암호화 및 복호화합니다. HSM을 사용하도록 코어 디바이스를 구성하면 이러한 구성 요소는 HSM에 저장한 프라이빗 키와 인증서를 사용합니다.

[Moquette MQTT 브로커 구성 요소](#) 또한 로컬 MQTT 서버 인증서의 프라이빗 키를 저장합니다. 이 구성 요소는 구성 요소의 작업 폴더에 있는 디바이스의 파일 시스템에 프라이빗 키를 저장합니다. 현재 AWS IoT Greengrass 는 이 프라이빗 키 또는 인증서를 HSM에 저장하는 기능을 지원하지 않습니다.

**Tip**

[AWS 파트너 디바이스 카탈로그](#)에서 이 기능을 지원하는 디바이스를 검색하세요.

## 주제

- [요구 사항](#)
- [하드웨어 보안 모범 사례](#)
- [하드웨어 보안으로 AWS IoT Greengrass 코어 소프트웨어 설치](#)
- [기존 코어 디바이스에서 하드웨어 보안 구성](#)
- [PKCS#11 지원 기능이 없는 하드웨어 사용](#)
- [다음 사항도 참조하세요.](#)

## 요구 사항

Greengrass 코어 디바이스에서 HSM을 사용하려면 다음과 같은 요구 사항을 충족해야 합니다.

- [Greengrass nucleus](#) v2.5.3 이상이 코어 디바이스에 설치되어야 합니다. 코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 호환되는 버전을 선택할 수 있습니다.
- [PKCS#11 공급자 구성 요소](#)가 코어 디바이스에 설치되어야 합니다. 코어 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 구성 요소를 다운로드하여 설치할 수 있습니다.
- [PKCS#1 v1.5](#) 서명 체계와 RSA-2048 키 크기(또는 그 이상) 또는 ECC 키를 사용하는 RSA 키가 지원되는 하드웨어 보안 모듈.

### Note

ECC 키와 함께 하드웨어 보안 모듈을 사용하려면 [Greengrass nucleus](#) v2.5.6 이상을 사용해야 합니다.  
하드웨어 보안 모듈 및 [보안 암호 관리자](#)를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

- AWS IoT Greengrass 코어 소프트웨어가 런타임에 로드(libdl 사용)하여 PKCS#11 함수를 호출할 수 있는 PKCS#11 공급자 라이브러리입니다. PKCS#11 공급자 라이브러리는 다음 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.
- 프라이빗 키와 인증서를 동일한 슬롯의 HSM에 저장해야 하고, HSM이 객체 ID를 지원하는 경우 동일한 객체 레이블과 객체 ID를 사용해야 합니다.
- 인증서와 프라이빗 키는 객체 레이블로 확인할 수 있어야 합니다.
- 프라이빗 키에 다음 권한이 있어야 합니다.
  - sign
  - decrypt
- (선택 사항) [보안 암호 관리자 구성 요소](#)를 사용하려면 버전 2.1.0 이상을 사용해야 하며, 프라이빗 키에 다음 권한이 있어야 합니다.

- wrap

## 하드웨어 보안 모범 사례

Greengrass 코어 디바이스에서 하드웨어 보안을 구성할 때 다음 모범 사례를 고려하세요.

- 내부 하드웨어 난수 생성기를 사용하여 HSM에서 직접 프라이빗 키를 생성합니다. 프라이빗 키는 HSM 내에 유지되므로 이 접근 방식은 다른 곳에서 생성한 프라이빗 키를 가져오는 것보다 더 안전합니다.
- 변경이 불가능하고 내보내기를 금지하도록 프라이빗 키를 구성합니다.
- HSM 하드웨어 공급업체가 하드웨어 보호 프라이빗 키를 사용하여 인증서 서명 요청(CSR)을 생성하도록 권장하는 프로비저닝 도구를 사용한 다음 AWS IoT 콘솔 또는 API를 사용하여 클라이언트 인증서를 생성합니다.

### Note

HSM에서 프라이빗 키를 생성할 때에는 키 교체 관련 모범 사례가 적용되지 않습니다.

## 하드웨어 보안으로 AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 HSM에서 생성한 프라이빗 키를 사용하도록 구성할 수 있습니다. 이 접근 방식은 [보안 모범 사례](#)를 따라 HSM에서 프라이빗 키를 생성하므로 프라이빗 키가 HSM 내에 유지됩니다.

하드웨어 보안으로 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 다음을 수행합니다.

1. HSM에서 프라이빗 키를 생성합니다.
2. 프라이빗 키에서 CSR(인증서 서명 요청)을 생성합니다.
3. CSR에서 인증서를 생성합니다. AWS IoT 또는 다른 루트 인증 기관(CA)에서 서명한 인증서를 생성할 수 있습니다. 다른 루트 CA를 사용하는 방법에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 [자체 클라이언트 인증서 생성](#)을 참조하세요.
4. AWS IoT 인증서를 다운로드하여 HSM으로 가져옵니다.
5. PKCS#11 공급자 구성 요소와 HSM의 프라이빗 키 및 인증서를 사용하도록 지정하는 구성 파일에서 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

다음 설치 옵션 중 하나를 선택하여 하드웨어 보안이 적용된 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다.

- 수동 설치

필요한 AWS 리소스를 수동으로 생성하고 하드웨어 보안을 구성하려면이 옵션을 선택합니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

- 사용자 지정 프로비저닝을 사용한 설치

필요한 AWS 리소스를 자동으로 생성하고 하드웨어 보안을 구성하는 사용자 지정 Java 애플리케이션을 개발하려면이 옵션을 선택합니다. 자세한 내용은 [사용자 지정 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

현재 AWS IoT Greengrass 는 자동 리소스 프로비저닝 또는 [AWS IoT 플릿 프로비저닝](#)으로 설치할 때 하드웨어 보안으로 AWS IoT Greengrass 코어 소프트웨어 설치를 지원하지 않습니다. ???

## 기존 코어 디바이스에서 하드웨어 보안 구성

코어 디바이스의 프라이빗 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 디바이스에서 하드웨어 보안을 구성하려면 다음을 수행합니다.

1. HSM을 초기화합니다.
2. [PKCS#11 공급자 구성 요소](#)를 코어 디바이스에 배포합니다.
3. AWS IoT Greengrass 코어 소프트웨어를 중지합니다.
4. 코어 디바이스의 프라이빗 키와 인증서를 HSM으로 가져옵니다.
5. HSM에서 프라이빗 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 소프트웨어의 구성 파일을 업데이트합니다.
6. AWS IoT Greengrass 코어 소프트웨어를 시작합니다.



## 1단계: 하드웨어 보안 모듈 초기화

다음 단계를 완료하여 코어 디바이스에서 HSM을 초기화합니다.

### 하드웨어 보안 모듈 초기화

- HSM에서 PKCS#11 토큰을 초기화하고, 토큰의 슬롯 ID와 사용자 PIN을 저장합니다. HSM의 설명서를 확인하여 토큰을 초기화하는 방법을 알아보세요. 나중에 PKCS#11 공급자 구성 요소를 배포 및 구성할 때 슬롯 ID와 사용자 PIN을 사용합니다.

## 2단계: PKCS#11 공급자 구성 요소 배포

다음 단계를 완료하여 [PKCS#11 공급자 구성 요소](#)를 배포 및 구성합니다. 구성 요소를 하나 이상의 코어 디바이스에 배포할 수 있습니다.

### PKCS#11 공급자 구성 요소 배포(콘솔)

- [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 구성 요소를 선택합니다.
- 구성 요소 페이지의 퍼블릭 구성 요소 탭을 선택한 다음 `aws.greengrass.crypto.Pkcs11Provider`를 선택합니다.
- `aws.greengrass.crypto.Pkcs11Provider` 페이지에서 배포를 선택합니다.
- 배포에 추가에서 수정할 기존 배포를 선택하거나 새 배포 생성을 선택하고 다음을 선택합니다.
- 새 배포 생성을 선택한 경우 배포의 대상 코어 디바이스 또는 사물 그룹을 선택합니다. 대상 지정 페이지의 배포 대상에서 코어 디바이스 또는 사물 그룹을 선택하고 다음을 선택합니다.
- 구성 요소 선택 페이지의 퍼블릭 구성 요소에서 `aws.greengrass.crypto.Pkcs11Provider`를 선택하고 다음을 선택합니다.
- 구성 요소 구성 페이지에서 `aws.greengrass.crypto.Pkcs11Provider`를 선택하고 다음을 수행합니다.
  - 구성 요소 구성을 선택합니다.
  - 구성 `aws.greengrass.crypto.Pkcs11Provider` 모달에서 구성 업데이트 아래의 병합할 구성에 다음 구성 업데이트를 입력합니다. 다음 구성 파라미터를 대상 코어 디바이스의 값으로 업데이트합니다. 앞서 PKCS#11 토큰을 초기화한 슬롯 ID와 사용자 PIN을 지정합니다. 나중에 프라이빗 키와 인증서를 HSM의 이 슬롯으로 가져옵니다.

name

PKCS#11 구성의 이름입니다.

## library

AWS IoT Greengrass 코어 소프트웨어가 libdl로 로드할 수 있는 PKCS#11 구현 라이브러리의 절대 파일 경로입니다.

## slot

프라이빗 키와 디바이스 인증서가 포함된 슬롯의 ID입니다. 이 값은 슬롯 인덱스 또는 슬롯 레이블과 다릅니다.

## userPin

슬롯 액세스에 사용할 사용자 PIN입니다.

```
{
 "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 1분 정도 걸릴 수 있습니다.

## PKCS#11 공급자 구성 요소 배포(AWS CLI)

PKCS#11 공급자 구성 요소를 배포하려면 components 객체에 `aws.greengrass.crypto.Pkcs11Provider`를 포함하는 배포 문서를 생성하고 구성 요소에 대한 구성 업데이트를 지정합니다. [배포 만들기](#)의 지침에 따라 새 배포를 생성하거나 기존 배포를 수정합니다.

다음 예제 부분 배포 문서를 통해 PKCS#11 공급자 구성 요소를 배포하고 구성하도록 지정합니다. 다음 구성 파라미터를 대상 코어 디바이스의 값으로 업데이트합니다. 프라이빗 키와 인증서를 HSM으로 가져올 때 나중에 사용할 슬롯 ID와 사용자 PIN을 저장합니다.

## name

PKCS#11 구성의 이름입니다.

## library

AWS IoT Greengrass 코어 소프트웨어가 libdl로 로드할 수 있는 PKCS#11 구현 라이브러리의 절대 파일 경로입니다.

## slot

프라이빗 키와 디바이스 인증서가 포함된 슬롯의 ID입니다. 이 값은 슬롯 인덱스 또는 슬롯 레이블과 다릅니다.

## userPin

슬롯 액세스에 사용할 사용자 PIN입니다.

```
{
 "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 Greengrass 서비스를 사용하여 배포 상태를 확인할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어 로그를 확인하여 PKCS#11 공급자 구성 요소가 성공적으로 배포되었는지 확인할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- [배포 상태 확인](#)
- [AWS IoT Greengrass 로그 모니터링](#)

배포에 실패하면 각 코어 디바이스에서 배포 문제를 해결할 수 있습니다. 자세한 내용은 [문제 해결 AWS IoT Greengrass V2](#) 단원을 참조하십시오.

### 3단계: 코어 디바이스에서 구성 업데이트

AWS IoT Greengrass 코어 소프트웨어는 디바이스 작동 방식을 지정하는 구성 파일을 사용합니다. 이 구성 파일에는 디바이스가 AWS 클라우드에 연결하는 데 사용하는 프라이빗 키와 인증서를 찾을 수 있는 위치가 포함됩니다. 다음 단계를 완료하여 코어 디바이스의 프라이빗 키 및 인증서를 HSM으로 가져오고 HSM을 사용하도록 구성 파일을 업데이트합니다.

하드웨어 보안을 사용하도록 코어 디바이스의 구성 업데이트

1. AWS IoT Greengrass 코어 소프트웨어를 중지합니다. `systemd`를 사용하여 [AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 구성](#)한 경우 다음 명령을 실행하여 소프트웨어를 중지할 수 있습니다.

```
sudo systemctl stop greengrass.service
```

2. 코어 디바이스의 프라이빗 키 및 인증서 파일을 찾습니다.
  - [자동 프로비저닝](#) 또는 [플릿 프로비저닝](#)과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치한 경우 프라이빗 키는 `/greengrass/v2/privKey.key`에 존재하고 인증서는 `/greengrass/v2/thingCert.crt`에 존재합니다.
  - [수동 프로비저닝](#)으로 AWS IoT Greengrass 코어 소프트웨어를 설치한 경우 프라이빗 키는 `/greengrass/v2/private.pem.key`에 기본적으로 존재하고 인증서는 기본적으로 `/greengrass/v2/device.pem.crt`에 존재합니다.

`/greengrass/v2/config/effectiveConfig.yaml`의 `system.privateKeyPath` 및 `system.certificateFilePath` 속성을 확인하여 이러한 파일의 위치를 찾을 수도 있습니다.

3. 프라이빗 키와 인증서를 HSM으로 가져옵니다. 프라이빗 키와 인증서를 가져오는 방법을 알아보면 HSM 설명서를 확인하세요. 앞서 PKCS#11 토큰을 초기화한 슬롯 ID와 사용자 PIN을 사용하여 프라이빗 키와 인증서를 가져옵니다. 프라이빗 키와 인증서에 동일한 객체 레이블과 객체 ID를 사용해야 합니다. 각 파일을 가져올 때 지정한 객체 레이블을 저장합니다. 나중에 HSM에서 프라이빗 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 소프트웨어 구성을 업데이트할 때 이 레이블을 사용합니다.
4. HSM에서 프라이빗 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 구성을 업데이트합니다. 구성을 업데이트하려면 AWS IoT Greengrass 코어 구성 파일을 수정하고 업데이트된 구성 파일로 AWS IoT Greengrass 코어 소프트웨어를 실행하여 새 구성을 적용합니다.

## 해결 방법:

- a. AWS IoT Greengrass 코어 구성 파일의 백업을 생성합니다. 하드웨어 보안을 구성할 때 문제가 발생하면 이 백업을 사용하여 코어 디바이스를 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 사용해 파일을 편집할 수 있습니다. `/greengrass/v2`를 Greengrass 루트 폴더의 경로로 바꿉니다.

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

- c. `system.privateKeyPath`의 값을 HSM의 프라이빗 키에 대한 PKCS#11 URI로 바꿉니다. `iotdevicekey`를 앞서 프라이빗 키와 인증서를 가져온 객체 레이블로 바꿉니다.

```
pkcs11:object=iotdevicekey;type=private
```

- d. `system.certificateFilePath`의 값을 HSM의 인증서에 대한 PKCS#11 URI로 바꿉니다. `iotdevicekey`를 앞서 프라이빗 키와 인증서를 가져온 객체 레이블로 바꿉니다.

```
pkcs11:object=iotdevicekey;type=cert
```

이 단계를 완료한 후 AWS IoT Greengrass 코어 구성 파일의 `system` 속성은 다음 예제와 비슷해야 합니다.

```
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. `--init-config` 파라미터로 `Greengrass.jar`를 실행하여 `effectiveConfig.yaml`에 구성을 적용합니다. `/greengrass/v2`를 Greengrass 루트 폴더의 경로로 바꿉니다.

```
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 Greengrass 코어 소프트웨어를 시작합니다. [systemd를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 구성](#)한 경우 다음 명령을 실행하여 소프트웨어를 시작할 수 있습니다.

```
sudo systemctl start greengrass.service
```

자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

7. AWS IoT Greengrass 코어 소프트웨어 로그를 확인하여 소프트웨어가 시작되고에 연결되어 있는지 확인합니다 AWS 클라우드. AWS IoT Greengrass 코어 소프트웨어는 프라이빗 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 연결합니다.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

다음 INFO 수준 로그 메시지는 AWS IoT Greengrass 코어 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 코어 소프트웨어가 HSM의 프라이빗 키 및 인증서와 함께 작동하는지 확인한 후 디바이스의 파일 시스템에서 프라이빗 키 및 인증서 파일을 삭제합니다. 다음 명령을 실행하고 파일 경로를 프라이빗 키 및 인증서 파일의 경로로 바꿉니다.

```
sudo rm /greengrass/v2/privKey.key
sudo rm /greengrass/v2/thingCert.crt
```

## PKCS#11 지원 기능이 없는 하드웨어 사용

PKCS#11 라이브러리는 일반적으로 하드웨어 공급업체가 제공하고 오픈 소스입니다. 예를 들어, 표준 호환 하드웨어(예: TPM1.2)를 사용하면 기존 오픈 소스 소프트웨어를 사용할 수 있습니다. 그러나 하드웨어에 해당하는 PKCS#11 라이브러리 구현이 없는 경우 또는 사용자 지정 PKCS#11 공급자를 쓰려는 경우 통합과 관련해 궁금한 점이 있으면 Amazon Web Services Enterprise Support 담당자에게 문의해야 합니다.

다음 사항도 참조하세요.

- [PKCS #11 암호화 토큰 인터페이스 사용 설명서 버전 2.4.0](#)
- [RFC 7512](#)
- [PKCS #1: RSA 암호화 버전 1.5](#)

## 에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass

AWS IoT Greengrass 환경의 디바이스는 인증에 X.509 인증서를 사용하고 권한 부여에 AWS IoT 정책을 사용합니다. 인증서 및 정책을 통해 장치끼리, 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 세트를 정의합니다. 특히 MQTT 메시지 게시 및 디바이스 새도우 검색과 같은 AWS IoT Core 및 AWS IoT Greengrass 데이터 영역 작업에 대한 액세스를 허용하고 거부합니다.

모든 디바이스에는 AWS IoT Core 레지스트리의 항목과 AWS IoT 정책이 연결된 활성화된 X.509 인증서가 필요합니다. 장치는 두 가지 범주로 나뉩니다.

- Greengrass 코어 디바이스

Greengrass 코어 디바이스는 인증서와 AWS IoT 정책을 사용하여 AWS IoT Core 및에 연결합니다 AWS IoT Greengrass. 인증서 및 정책은가 코어 디바이스 AWS IoT Greengrass 에 구성 요소 및 구성을 배포하도록 허용합니다.

- 클라이언트 디바이스

MQTT 클라이언트 디바이스는 인증서와 정책을 사용하여 AWS IoT Core 및 AWS IoT Greengrass 서비스에 연결합니다. 이를 통해 클라이언트 디바이스는 AWS IoT Greengrass 클라우드 검색을 사용하여 Greengrass 코어 디바이스를 찾고 연결할 수 있습니다. 클라이언트 디바이스에서는 AWS IoT Core 클라우드 서비스 및 코어 디바이스 연결에 동일한 인증서가 사용됩니다. 또한 클라이언트 장치는 코어 장치와의 상호 인증을 위해 검색 정보를 사용합니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

## X.509 인증서

코어 디바이스와 클라이언트 디바이스 간의 통신과 디바이스와 AWS IoT Core 또는 간의 통신은 인증되어야 AWS IoT Greengrass 합니다. 이 상호 인증은 등록된 X.509 장치 인증서와 암호화 키를 기반으로 수행됩니다.

환경에서 AWS IoT Greengrass 디바이스는 다음 전송 계층 보안(TLS) 연결에 퍼블릭 및 프라이빗 키가 있는 인증서를 사용합니다.

- 인터넷을 통해 AWS IoT Core 및 AWS IoT Greengrass 에 연결하는 Greengrass 코어 디바이스의 AWS IoT 클라이언트 구성 요소입니다.
- 코어 디바이스를 검색하기 위해 인터넷을 AWS IoT Greengrass 통해 연결하는 클라이언트 디바이스입니다.
- 로컬 네트워크를 통해 그룹의 Greengrass 디바이스에 연결되어 있는 Greengrass 코어의 MQTT 브로커 구성 요소입니다.

AWS IoT Greengrass 코어 디바이스는 Greengrass 루트 폴더에 인증서를 저장합니다.

### CA(인증 기관) 인증서

Greengrass 코어 디바이스 및 클라이언트 디바이스는 AWS IoT Core 및 AWS IoT Greengrass 서비스를 사용한 인증에 사용되는 루트 CA 인증서를 다운로드합니다. [Amazon Root CA 1](#) 같은 Amazon Trust Services(ATS) 루트 CA 인증서를 사용하는 것이 좋습니다. [서버 인증용 CA 인증서](#)에 대한 자세한 내용은 AWS IoT Core 개발자 설명서의 인증(IoT)를 참조하세요.

클라이언트 디바이스에서는 Greengrass 코어 디바이스 CA 인증서도 다운로드됩니다. 상호 인증 중 Greengrass 코어의 MQTT 서버 인증서 검증에 이 인증서가 사용됩니다.

### 로컬 MQTT 브로커의 인증서 교체

[클라이언트 디바이스 지원을 활성화](#)하면 Greengrass 코어 디바이스에서는 클라이언트 디바이스에서 상호 인증에 사용되는 로컬 MQTT 서버 인증서가 생성됩니다. 이 인증서는 코어 디바이스가 AWS IoT Greengrass 클라우드에 저장하는 코어 디바이스 CA 인증서로 서명됩니다. 클라이언트 디바이스에서는 코어 디바이스가 검색될 때 코어 디바이스 CA 인증서가 검색됩니다. 코어 디바이스에 연결될 때 코어 디바이스의 MQTT 서버 인증서 확인에 코어 디바이스 CA 인증서가 사용됩니다. 코어 디바이스 CA 인증서는 5년이 지나면 만료됩니다.



MQTT 서버 인증서는 기본적으로 7일마다 만료되며, 이 기간을 2~10일로 구성할 수 있습니다. 이 제한된 기간은 보안 모범 사례를 기반으로 합니다. 이 교체는 공격자가 Greengrass 코어 디바이스로 가장하려고 MQTT 서버 인증서와 프라이빗 키를 도용하는 위협을 완화하는 데 도움이 됩니다.

Greengrass 코어 디바이스에서는 MQTT 서버 인증서가 만료 24시간 전에 교체됩니다. Greengrass 코어 디바이스에서 새 인증서가 생성되고 로컬 MQTT 브로커가 다시 시작됩니다. 이렇게 되면 Greengrass 코어 디바이스에 연결된 모든 클라이언트 디바이스의 연결이 해제됩니다. 짧은 기간 뒤에 Greengrass 코어 디바이스에 클라이언트 디바이스가 다시 연결될 수 있습니다.

## AWS IoT 데이터 영역 작업에 대한 정책

AWS IoT 정책을 사용하여 AWS IoT Core 및 AWS IoT Greengrass 데이터 영역에 대한 액세스를 승인합니다. AWS IoT Core 데이터 플레인에서 디바이스, 사용자 및 애플리케이션에 대한 작업이 제공됩니다. 이러한 작업에는 주제에 연결하고 AWS IoT Core 주제를 구독하는 기능이 포함됩니다. AWS IoT Greengrass 데이터 영역은 Greengrass 디바이스에 대한 작업을 제공합니다. 자세한 내용은 [AWS IoT Greengrass V2 정책 작업](#) 단원을 참조하십시오. 이러한 작업에는 구성 요소 종속성이 해결되고 퍼블릭 구성 요소 아티팩트가 다운로드되는 기능이 포함되어 있습니다.

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 와일드카드 문자\*로 지원하고 MQTT 와일드카드 문자(+ 및 #)를 리터럴 문자열로 취급합니다. \* 와일드카드에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [리소스 ARN에서 와일드카드 사용](#)을 참조하십시오.

자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT 정책](#) 및 [AWS IoT 정책 조치](#)를 참조하십시오.

### Important

코어 디바이스 또는 Greengrass 데이터 플레인 작업의 AWS IoT 정책에 대해 [사물 정책 변수\(iot:Connection.Thing.\\*\)](#)가 지원되지 않습니다. 그 대신에 이름이 비슷한 여러 디바이스와 일치하는 와일드카드를 사용할 수 있습니다. 예를 들어 MyGreengrassDevice1, MyGreengrassDevice2 등과 일치하도록 MyGreengrassDevice\*를 지정할 수 있습니다.

**Note**

AWS IoT Core 를 사용하면 사물 그룹에 AWS IoT 정책을 연결하여 디바이스 그룹에 대한 권한을 정의할 수 있습니다. 사물 그룹 정책은 AWS IoT Greengrass 데이터 영역 작업에 대한 액세스를 허용하지 않습니다. AWS IoT Greengrass 데이터 영역 작업에 대한 사물 액세스를 허용하려면 사물의 인증서에 연결하는 AWS IoT 정책에 권한을 추가합니다.

## AWS IoT Greengrass V2 정책 작업

AWS IoT Greengrass V2 는 Greengrass 코어 디바이스 및 클라이언트 디바이스가 정책에 사용할 수 있는 다음과 같은 AWS IoT 정책 작업을 정의합니다. 정책 작업에 리소스를 지정하려면 리소스의 Amazon 리소스 이름(ARN)을 사용해야 합니다.

### 코어 디바이스 작업

#### `greengrass:GetComponentVersionArtifact`

퍼블릭 구성 요소 아티팩트 또는 Lambda 구성 요소 아티팩트가 다운로드되도록 미리 서명된 URL 을 가져오는 권한을 부여합니다.

이 권한은 아티팩트가 있는 퍼블릭 구성 요소 또는 Lambda가 지정되는 배포가 코어 디바이스에 수신될 때 평가됩니다. 코어 디바이스에 이미 아티팩트가 있으면 아티팩트가 다시 다운로드되지 않습니다.

리소스 유형: `componentVersion`

리소스 ARN 형식: `arn:aws:greengrass:region:account-id:components:component-name:versions:component-version`

#### `greengrass:ResolveComponentCandidates`

배포의 구성 요소, 버전 및 플랫폼 요구 사항이 충족되는 구성 요소의 목록을 식별하는 권한을 부여합니다. 요구 사항이 충돌하거나 요구 사항이 충족되는 구성 요소가 없으면 이 작업에서는 오류가 반환되고 디바이스에서 배포에 실패합니다.

이 권한은 아티팩트가 있는 구성 요소가 지정되는 배포가 코어 디바이스에 수신될 때 평가됩니다.

리소스 유형: 없음

리소스 ARN 형식: \*

## greengrass:GetDeploymentConfiguration

큰 배포 문서가 다운로드되는 미리 서명된 URL을 가져오는 권한을 부여합니다.

이 권한은 7KB(배포의 대상이 사물인 경우) 또는 31KB(배포의 대상이 사물 그룹인 경우)보다 큰 배포 문서가 지정되는 배포가 코어 디바이스에 수신될 때 평가됩니다. 배포 문서에는 구성 요소 구성, 배포 정책 및 배포 메타데이터가 포함되어 있습니다. 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 단원을 참조하십시오.

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.3.0 이상에서 사용할 수 있습니다.

리소스 유형: 없음

리소스 ARN 형식: \*

## greengrass:ListThingGroupsForCoreDevice

코어 디바이스의 사물 그룹 계층 구조를 가져오는 권한을 부여합니다.

코어 디바이스가 배포를 수신하면 이 권한이 확인됩니다 AWS IoT Greengrass. 코어 디바이스에서는 마지막 배포 이후 사물 그룹에서 제거되었는지 여부를 식별하는 데 이 작업이 사용됩니다. 코어 디바이스가 사물 그룹에서 제거되었고, 해당 사물 그룹이 코어 디바이스에 대한 배포의 대상인 경우 코어 디바이스에서는 해당 배포에 따라 설치된 구성 요소가 제거됩니다.

이 특성은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용됩니다.

리소스 유형: thing(코어 디바이스)

리소스 ARN 형식: arn:aws:iot:*region*:*account-id*:thing/*core-device-thing-name*

## greengrass:VerifyClientDeviceIdentity

코어 디바이스에 연결되는 클라이언트 디바이스의 ID를 확인하는 권한을 부여합니다.

이 권한은 코어 디바이스에서 [클라이언트 디바이스 인증 구성 요소](#)가 실행되고 클라이언트 디바이스의 MQTT 연결이 수신될 때 평가됩니다. 클라이언트 디바이스에서는 AWS IoT 디바이스 인증서가 제공됩니다. 그런 다음에서는 코어 디바이스에서는 클라이언트 디바이스의 ID를 확인할 디바이스 인증서가 AWS IoT Greengrass 클라우드 서비스로 발송됩니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

리소스 유형: 없음

리소스 ARN 형식: \*

## greengrass:VerifyClientDeviceIoTCertificateAssociation

클라이언트 디바이스가 AWS IoT 인증서와 연결되어 있는지 확인하는 권한을 부여합니다.

이 권한은 코어 디바이스에서 [클라이언트 디바이스 인증 구성 요소](#)가 실행되고 MQTT를 통한 연결 권한이 클라이언트 디바이스에 부여될 때 평가됩니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

### Note

코어 디바이스가 이 작업을 사용하려면 [Greengrass 서비스 역할을](#) AWS 계정에 연결하고 `iot:DescribeCertificate` 권한을 허용해야 합니다.

리소스 유형: thing(클라이언트 디바이스)

리소스 ARN 형식: `arn:aws:iot:region:account-id:thing/client-device-thing-name`

## greengrass:PutCertificateAuthorities

클라이언트 디바이스에 다운로드되어 코어 디바이스가 확인될 수 있는 인증 기관(CA) 인증서를 업로드하는 권한을 부여합니다.

이 권한은 코어 디바이스가 설치되고 [클라이언트 디바이스 인증 구성 요소](#)가 실행될 때 평가됩니다. 이 구성 요소에서는 로컬 인증 기관이 생성되며 CA 인증서 업로드에 이 작업이 사용됩니다. 클라이언트 디바이스에서는 연결될 수 있는 코어 디바이스 찾기에 [검색](#) 작업이 사용될 때 이러한 CA 인증서가 다운로드됩니다. 클라이언트 디바이스에서는 코어 디바이스의 MQTT 브로커에 연결되면 코어 디바이스의 ID 확인에 이러한 CA 인증서가 사용됩니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

리소스 유형: 없음

ARN 형식: \*

## greengrass:GetConnectivityInfo

코어 디바이스에 대한 연결 정보를 가져오는 권한을 부여합니다. 이 정보에는 클라이언트 디바이스가 코어 디바이스에 연결되는 방식이 설명되어 있습니다.

이 권한은 코어 디바이스가 설치되고 [클라이언트 디바이스 인증 구성 요소](#)가 실행될 때 평가됩니다. 이 구성 요소는 연결 정보를 사용하여 [PutCertificateAuthorities](#) 작업을 통해 AWS IoT

Greengrass 클라우드 서비스에 업로드할 유효한 CA 인증서를 생성합니다. 클라이언트 디바이스에서는 코어 디바이스의 ID 확인에 이러한 CA 인증서가 사용됩니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

AWS IoT Greengrass 컨트롤 플레인에서 이 작업을 사용하여 코어 디바이스의 연결 정보를 볼 수도 있습니다. 자세한 내용은 AWS IoT Greengrass V1 API 참조의 [GetConnectivityInfo](#)를 참조하세요.

리소스 유형: thing(코어 디바이스)

리소스 ARN 형식: `arn:aws:iot:region:account-id:thing/core-device-thing-name`  
`greengrass:UpdateConnectivityInfo`

코어 디바이스에 대한 연결 정보를 업데이트하는 권한을 부여합니다. 이 정보에는 클라이언트 디바이스가 코어 디바이스에 연결되는 방식이 설명되어 있습니다.

이 권한은 코어 디바이스에서 [IP 감지기 구성 요소](#)가 실행될 때 평가됩니다. 이 구성 요소에서는 클라이언트 디바이스가 로컬 네트워크의 코어 디바이스에 연결되는 데 필요한 정보가 식별됩니다. 그런 다음 이 구성 요소는 이 작업을 사용하여 연결 정보를 AWS IoT Greengrass 클라우드 서비스에 업로드하므로 클라이언트 디바이스가 [검색](#) 작업으로 이 정보를 검색할 수 있습니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

AWS IoT Greengrass 컨트롤 플레인에서 이 작업을 사용하여 코어 디바이스의 연결 정보를 수동으로 업데이트할 수도 있습니다. 자세한 내용은 AWS IoT Greengrass V1 API 참조의 [UpdateConnectivityInfo](#)를 참조하세요.

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 디바이스가 연결될 수 있는 코어 디바이스에 대한 연결 정보를 검색하는 권한을 부여합니다. 이 정보에는 클라이언트 디바이스가 코어 디바이스에 연결되는 방식이 설명되어 있습니다. 클라이언트 디바이스에서는 [BatchAssociateClientDeviceWithCoreDevice](#) 작업을 사용하여 연결한 코어 디바이스만 검색될 수 있습니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.

리소스 유형: thing(클라이언트 디바이스)

리소스 ARN 형식: `arn:aws:iot:region:account-id:thing/client-device-thing-name`

## 코어 디바이스의 AWS IoT 정책 업데이트

AWS IoT Greengrass 및 AWS IoT 콘솔 또는 AWS IoT API를 사용하여 코어 디바이스의 AWS IoT 정책을 보고 업데이트할 수 있습니다.

### Note

[AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 코어 디바이스에는 모든 AWS IoT Greengrass 작업()에 대한 액세스를 허용하는 AWS IoT 정책이 있습니다. `greengrass:*`. 다음과 같은 단계에 따라 코어 디바이스에서 사용되는 작업으로만 액세스를 제한할 수 있습니다.

### 코어 디바이스의 AWS IoT 정책 검토 및 업데이트(콘솔)

1. [AWS IoT Greengrass 콘솔](#) 탐색 메뉴에서 코어 디바이스를 선택합니다.
2. 업데이트할 코어 디바이스를 코어 디바이스 페이지에서 선택합니다.
3. 코어 디바이스 세부 정보 페이지에서 코어 디바이스의 사물에 대한 링크를 선택합니다. 이 링크를 통해 사물 세부 정보 페이지가 AWS IoT 콘솔에서 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 검토하고 업데이트할 AWS IoT 정책을 선택합니다. 코어 디바이스의 활성 인증서에 연결된 원하는 정책에 필수 권한을 추가할 수 있습니다.

### Note

[AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝](#)한 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 `GreengrassV2IoTThingPolicy`인 정책을 선택하는 것이 좋습니다(있는 경우). 빠른 설치 프로그램으로 생성한 코어 디바이스에서는 기본적으로 이 정책 이름이 사용됩니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 코어 디바이스에도 이러한 권한을 부여하는 것입니다.

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.

## 코어 디바이스의 AWS IoT 정책 검토 및 업데이트(AWS CLI)

1. 코어 디바이스 사물의 보안 주체를 나열합니다 AWS IoT . 사물 보안 주체는 X.509 디바이스 인증서 또는 기타 ID일 수 있습니다. 다음 명령을 실행하고 *MyGreengrassCore*를 코어 디바이스의 이름으로 바꿉니다.

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

코어 디바이스의 사물 보안 주체가 나열되는 응답이 작업에서 반환됩니다.

```
{
 "principals": [
 "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
]
}
```

2. 코어 디바이스의 활성 인증서를 식별합니다. 다음 명령을 실행하고, 활성 인증서를 찾을 때까지 이전 단계의 각 인증서 ID로 *certificateId*를 바꿉니다. 인증서 ID는 인증서 ARN 끝에 있는 16진수 문자열입니다. --query 인수에서는 인증서의 상태만 출력으로 지정됩니다.

```
aws iot describe-certificate --certificate-id certificateId --query
'certificateDescription.status'
```

인증서 상태가 문자열로 작업에서 반환됩니다. 예를 들어, 인증서가 활성이라면 이 작업에서는 "ACTIVE"가 출력됩니다.

3. 인증서에 연결된 AWS IoT 정책을 나열합니다. 다음 명령을 실행하고 인증서의 ARN으로 인증서 ARN을 바꿉니다.

```
aws iot list-principal-policies --principal arn:aws:iot:us-west-2:123456789012:cert/certificateId
```

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 Greengrass 코어 소프트웨어 설치 관리자를 사용하여 리소스를 프로비저닝한](#) 경우 두 가지 AWS IoT 정책이 있습니다. 이름이 GreengrassV2IoTThingPolicy인 정책을 선택하는 것이 좋습니다(있는 경우). 빠른 설치 프로그램으로 생성한 코어 디바이스에서 기본적으로 이 정책 이름이 사용됩니다. 이 정책에 권한을 추가하면 이 정책을 사용하는 다른 코어 디바이스에도 이러한 권한을 부여하는 것입니다.

5. 정책의 문서를 가져옵니다. 다음 명령을 실행하고 정책의 이름으로 *GreengrassV2IoTThingPolicy*를 바꿉니다.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

정책의 문서와 정책에 대한 기타 정보가 있는 응답이 작업에서 반환됩니다. 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"\\
 }

```

- 온라인 변환기 또는 기타 도구를 사용하여 정책 문서 문자열을 JSON 객체로 변환한 다음에 `iot-policy.json`이라는 파일에 저장합니다.

예를 들어, 설치된 [jq](#) 도구가 있으면 다음 명령을 실행하여 정책 문서를 가져와서 JSON 객체로 변환하고 정책 문서를 JSON 객체로 저장할 수 있습니다.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
'policyDocument' | jq fromjson >> iot-policy.json
```

- 정책 문서를 검토하고 필요에 따라 권한을 추가, 제거 또는 편집합니다.

예를 들어 Linux 기반 시스템에서 GNU nano를 사용하여 파일을 열도록 다음 명령을 실행할 수 있습니다.

```
nano iot-policy.json
```

완료되면 정책 문서가 [코어 디바이스에 대한 최소 AWS IoT 정책](#)과 비슷할 수 있습니다.

- 변경 사항을 정책의 새 버전으로 저장합니다. 다음 명령을 실행하고 정책의 이름으로 *GreengrassV2IoTThingPolicy*를 바꿉니다.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-document file://iot-policy.json --set-as-default
```

작업에 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\\\": [\\
 \\\"iot:Connect\\\",\\
 \\\"iot:Publish\\\",\\
 \\\"iot:Subscribe\\\",\\
 \\\"iot:Receive\\\",\\
 \\\"greengrass:*\\\"\\
],\\
 \\\"Resource\\\": \\\"*\\\"\\
 }\\
]\\
 }\",
 "policyVersionId": "2",
 "isDefaultVersion": true
}
```

## AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책

### Important

이후 버전의 [Greengrass nucleus 구성 요소](#)에는 최소 AWS IoT 정책에 대한 추가 권한이 필요합니다. [코어 디바이스의 AWS IoT 정책을 업데이트](#)하여 추가 권한을 부여해야 할 수도 있습니다.

- Greengrass nucleus v2.5.0 이상이 실행되는 코어 디바이스에서는 사물 그룹에서 코어 디바이스를 제거할 때 구성 요소 제거에 `greengrass:ListThingGroupsForCoreDevice` 권한이 사용됩니다.

- Greengrass nucleus v2.3.0 이상이 실행되는 코어 디바이스에서는 큰 배포 구성 문서 지원에 `greengrass:GetDeploymentConfiguration` 권한이 사용됩니다.

다음 예제 정책에는 코어 장치용 기본 Greengrass 기능을 지원하는 데 필요한 최소 작업 세트가 포함됩니다.

- Connect 정책에는 코어 디바이스 사물 이름 뒤에 \* 와일드카드가 포함되어 있습니다(예: `core-device-thing-name*`). 코어 디바이스는 동일한 디바이스 인증서를 사용하여 여러 동시 구독을 만들 AWS IoT Core지만 연결의 클라이언트 ID가 코어 디바이스 사물 이름과 정확히 일치하지 않을 수 있습니다. 50개까지 구독 후 코어 디바이스에서는 추가 50개 구독마다 `number`개씩 증분되는 `core-device-thing-name#number`가 클라이언트 ID로 사용됩니다. 예를 들어, MyCoreDevice라는 코어 디바이스에서 150개의 동시 구독이 생성되면 다음 클라이언트 ID가 사용됩니다.
  - 구독 1~50개: MyCoreDevice
  - 구독 51~100개: MyCoreDevice#2
  - 구독 101~150개: MyCoreDevice#3

접미사가 있는 이러한 클라이언트 ID가 사용될 때 와일드카드를 통해 코어 디바이스가 연결될 수 있습니다.

- 이 정책은 코어 장치가 새도 상태에 사용되는 주제를 포함하여 메시지를 게시하고 구독하고 메시지를 수신할 수 있는 MQTT 정책과 주제 필터를 나열합니다. AWS IoT Core, Greengrass 구성 요소 및 클라이언트 디바이스 간의 메시지 교환을 지원하려면 허용할 주제 및 주제 필터를 지정합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [게시/구독 정책 예제](#)를 참조하십시오.
- 이 정책에서는 원격 측정 데이터에 대한 다음 주제에 게시하는 권한이 부여됩니다.

```
$aws/things/core-device-thing-name/greengrass/health/json
```

원격 측정을 비활성화하는 코어 디바이스의 이 권한을 제거할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 디바이스에서 시스템 상태 원격 측정 데이터 수집](#) 단원을 참조하십시오.

- 이 정책은 역할 별칭을 통해 IAM AWS IoT 역할을 수입할 수 있는 권한을 부여합니다. 코어 디바이스는 토큰 교환 역할이라는 이 역할을 사용하여 AWS 요청을 인증하는 데 사용할 수 있는 AWS 자격 증명을 획득합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어를 설치할 때이 권한만 포함된 두 번째 AWS IoT 정책을 생성하고 연결합니다. 코어 디바이스의 기본 AWS IoT 정책에이 권한을 포함하면 다른 AWS IoT 정책을 분리하고 삭제할 수 있습니다.

## JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "iot:Connect"
],
 "Resource": "arn:aws:iot:us-east-1:123456789012:client/core-device-thing-name*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "iot:Receive",
 "iot:Publish"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-device-thing-name/greengrass/health/json",
 "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-device-thing-name/greengrassv2/health/json",
 "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-device-thing-name/jobs/*",
 "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-device-thing-name/shadow/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "iot:Subscribe"
],
 "Resource": [
```

```

 "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name/jobs/*",
 "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name/shadow/*"
],
 {
 "Effect": "Allow",
 "Action": "iot:AssumeRoleWithCertificate",
 "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/token-
exchange-role-alias-name"
 },
 {
 "Effect": "Allow",
 "Action": [
 "greengrass:GetComponentVersionArtifact",
 "greengrass:ResolveComponentCandidates",
 "greengrass:GetDeploymentConfiguration",
 "greengrass:ListThingGroupsForCoreDevice"
],
 "Resource": "*"
 }
]
}

```

## 클라이언트 디바이스를 지원하는 최소 AWS IoT 정책

다음 예제 정책에는 코어 디바이스의 클라이언트 디바이스와 상호 작용 지원에 필요한 최소 작업 세트가 포함되어 있습니다. 클라이언트 디바이스를 지원하려면 코어 디바이스에 기본 작업에 대한 최소 AWS IoT 정책 외에도 이 정책의 권한이 있어야 합니다. [AWS IoT](#)

- 이 정책을 통해 코어 디바이스의 자체 연결 정보 업데이트가 허용됩니다. [IP 감지기 구성 요소](#)를 코어 디바이스에 배포하는 경우에만 이 권한(`greengrass:UpdateConnectivityInfo`)이 필요합니다.

### JSON

```

{
 "Version": "2012-10-17",
 "Statement": [

```

```
{
 "Effect": "Allow",
 "Action": [
 "iot:Publish"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/get"
]
},
{
 "Effect": "Allow",
 "Action": [
 "iot:Subscribe"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name-gci/shadow/update/delta",
 "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name-gci/shadow/get/accepted"
]
},
{
 "Effect": "Allow",
 "Action": [
 "iot:Receive"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/update/delta",
 "arn:aws:iot:us-east-1:123456789012: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:us-east-1:123456789012:thing/*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "greengrass:GetConnectivityInfo",
 "greengrass:UpdateConnectivityInfo"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:thing/core-device-thing-name"
]
 }
]
}

```

## 클라이언트 디바이스에 대한 최소 AWS IoT 정책

다음 예제 정책에는 MQTT를 통해 연결되고 통신할 코어 디바이스를 클라이언트 디바이스에서 검색하는 데 필요한 최소 작업 세트가 포함되어 있습니다. 클라이언트 디바이스의 AWS IoT 정책에는 디바이스가 연결된 Greengrass 코어 디바이스의 연결 정보를 검색할 수 있도록 허용하는 `greengrass:Discover` 작업이 포함되어야 합니다. Resource 섹션에서 Greengrass 코어 디바이스의 Amazon 리소스 이름(ARN)이 아닌 클라이언트 디바이스의 ARN을 지정합니다.

- 모든 MQTT 주제에 대한 통신이 정책에서 허용됩니다. 모범 보안 사례를 따르려면 `iot:Publish`, `iot:Subscribe` 및 `iot:Receive` 권한을 클라이언트 디바이스의 사용 사례에 필요한 최소 주제 집합으로 제한합니다.
- 이 정책은 사물이 모든 AWS IoT 사물의 코어 디바이스를 검색하도록 허용합니다. 모범 보안 사례를 따르려면 클라이언트 디바이스의 AWS IoT 사물 또는 AWS IoT 사물 집합과 일치하는 와일드카드로 `greengrass:Discover` 권한을 제한합니다.

### Important

코어 디바이스 또는 Greengrass 데이터 플레인 작업의 AWS IoT 정책에 대해 [사물 정책 변수](#)(`iot:Connection.Thing.*`)가 지원되지 않습니다. 그 대신에 이름이 비슷한 여러 디바이스와 일치하는 와일드카드를 사용할 수 있습니다. 예를 들어 `MyGreengrassDevice1`,

MyGreengrassDevice2 등과 일치하도록 MyGreengrassDevice\*를 지정할 수 있습니다.

- Greengrass 코어 디바이스가 클라이언트 디바이스에 대한 새도 동기화 작업을 처리하기 때문에 클라이언트 디바이스의 AWS IoT 정책에는 일반적으로 `iot:GetThingShadow`, `iot:UpdateThingShadow`, 또는 `iot:DeleteThingShadow` 작업에 대한 권한이 필요하지 않습니다. 코어 디바이스가 클라이언트 디바이스 새도우를 처리할 수 있도록 하려면 코어 디바이스의 AWS IoT 정책이 이러한 작업을 허용하는지, 그리고 Resource 섹션에 클라이언트 디바이스의 ARNs이 포함되어 있는지 확인합니다.

## JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "iot:Connect"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "iot:Publish"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topic/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "iot:Subscribe"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topicfilter/*"
]
 }
]
}
```



```

 {
 "Effect": "Allow",
 "Action": [
 "iot:Receive"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:topic/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "greengrass:Discover"
],
 "Resource": [
 "arn:aws:iot:us-east-1:123456789012:thing/*"
]
 }
]
}

```

## 에 대한 자격 증명 및 액세스 관리 AWS IoT Greengrass

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 되는 AWS 서비스입니다. IAM 관리자는 누가 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. 에서 지원하는 IAM 기능에 대한 자세한 내용은 섹션을 AWS IoT Greengrass참조하세요 [the section called “AWS IoT Greengrass 에서 IAM을 사용하는 방법”](#).

## 대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청([참조에 대한 자격 증명 및 액세스 문제 해결 AWS IoT Greengrass](#))
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출([AWS IoT Greengrass 에서 IAM을 사용하는 방법 참조](#))
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성([에 대한 자격 증명 기반 정책 예제 AWS IoT Greengrass 참조](#))

## ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [AWS 계정에 로그인하는 방법](#) 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해서는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 [API 요청용 AWS Signature Version 4](#) 섹션을 참조하세요.

## AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자가 필요한 작업 목록은 IAM 사용자 설명서의 [루트 사용자 자격 증명이 필요한 작업](#)을 참조하세요.

## IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명에 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여 액세스하도록 인간 사용자에게 요구](#)하기를 참조하세요.

[IAM 그룹](#)은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자 사용 사례](#) 섹션을 참조하세요.

## IAM 역할

[IAM 역할](#)은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. [사용자에서 IAM 역할\(콘솔\)로 전환하거나 또는 API 작업을 호출하여 역할을](#) 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 [역할 수임 방법](#)을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 [교차 계정 리소스 액세스](#)를 참조하세요.

## 정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#) 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

## ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 [고객 관리형 정책으로 사용자 지정 IAM 권한 정의](#)를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책 및 인라인 정책 중에서 선택](#) 섹션을 참조하세요.

## 리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한

정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다.

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 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF 및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 [액세스 제어 목록\(ACL\) 개요](#)를 참조하세요.

## 기타 정책 타입

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 [IAM 엔터티의 권한 범위](#)를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [서비스 제어 정책](#)을 참조하세요.
- 리소스 제어 정책(RCP) - 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [리소스 제어 정책\(RCP\)](#)을 참조하세요.
- 세션 정책 - 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하세요.

## 여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하세요.

다음 사항도 참조하세요.

- [the section called “AWS IoT Greengrass 에서 IAM을 사용하는 방법”](#)
- [the section called “ID 기반 정책 예시”](#)

- [the section called “자격 증명 및 액세스 문제 해결”](#)

## AWS IoT Greengrass 에서 IAM을 사용하는 방법

IAM을 사용하여 액세스를 관리하기 전에 사용할 수 있는 IAM 기능을 이해해야 AWS IoT Greengrass 합니다 AWS IoT Greengrass.

| IAM 특성                                    | Greengrass에서 지원합니까? |
|-------------------------------------------|---------------------|
| <a href="#">리소스 수준 권한이 있는 자격 증명 기반 정책</a> | 예                   |
| <a href="#">리소스 기반 정책</a>                 | 아니요                 |
| <a href="#">액세스 제어 목록(ACL)</a>            | 아니요                 |
| <a href="#">태그 기반 승인</a>                  | 예                   |
| <a href="#">임시 보안 인증</a>                  | 예                   |
| <a href="#">서비스 연결 역할</a>                 | 아니요                 |
| <a href="#">서비스 역할</a>                    | 예                   |

다른 AWS 서비스에서 IAM을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 [AWS IAM 으로 작업하는 서비스](#)를 참조하세요.

### 에 대한 자격 증명 기반 정책 AWS IoT Greengrass

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부된 작업 및 리소스와 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다.는 특정 작업, 리소스 및 조건 키를 AWS IoT Greengrass 지원합니다. 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

#### 작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

에 대한 정책 작업은 작업 앞에 `greengrass:` 접두사를 AWS IoT Greengrass 사용합니다. 예를 들어 API `ListCoreDevices` 작업을 사용하여 코어 디바이스를 나열하도록 허용하려면 해당 정책에 `greengrass:ListCoreDevices` 작업을 AWS 계정포함합니다. 정책 설명에는 Action 또는 NotAction 요소가 포함되어야 합니다. 이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 AWS IoT Greengrass 정의합니다.

명령문 하나에 여러 작업을 지정하려면 다음과 같이 괄호 안에 나열한 후([ ]) 각 작업을 쉼표로 구분합니다.

```
"Action": [
 "greengrass:action1",
 "greengrass:action2",
 "greengrass:action3"
]
```

와일드카드(\*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, List라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "greengrass:List*"
```

#### Note

와일드카드를 사용하여 서비스에 대해 사용 가능한 모든 작업을 지정하는 것은 권장하지 않습니다. 가장 좋은 방법은 정책에 최소한의 권한을 부여하고 권한을 줄이는 것입니다. 자세한 내용은 [the section called “가능한 최소 권한 부여”](#) 단원을 참조하십시오.

AWS IoT Greengrass 작업의 전체 목록은 IAM 사용 설명서의 [에서 정의한 작업을 AWS IoT Greengrass](#) 참조하세요.

## 리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 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. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(\*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"

```

다음 표에는 정책 설명의 Resource 요소에 사용할 수 있는 AWS IoT Greengrass 리소스 ARN이 나와 있습니다. AWS IoT Greengrass 작업에 지원되는 리소스 수준 권한의 매핑은 IAM 사용 설명서의 [에서 정의한 작업을 AWS IoT Greengrass](#) 참조하세요.

일부 AWS IoT Greengrass 작업(예: 일부 목록 작업)은 특정 리소스에서 수행할 수 없습니다. 이러한 경우 와일드카드만 사용해야 합니다.

```
"Resource": "*"

```

한 명령에 여러 리소스 ARN을 지정하려면 다음과 같이 괄호 사이에 나열하고([ ]) 쉼표로 구분합니다.

```
"Resource": [
 "resource-arn1",
 "resource-arn2",
 "resource-arn3"
]

```

ARN 형식에 대한 자세한 내용은 [Amazon 리소스 이름\(ARNs\) 및 AWS 서비스 네임스페이스](#)를 참조하세요Amazon Web Services 일반 참조.

## 조건 키

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소는 정의된 기준에 따라 문이 실행되는 시기를 지정합니다. 같음(equals) 또는 미만(less than)과 같은 [조건 연산자](#)를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 [AWS 전역 조건 컨텍스트 키](#)를 참조하세요.

## 예제

자격 AWS IoT Greengrass 증명 기반 정책의 예를 보려면 섹션을 참조하세요[the section called “ID 기반 정책 예시”](#).

## 에 대한 리소스 기반 정책 AWS IoT Greengrass

AWS IoT Greengrass 는 [리소스 기반 정책](#)을 지원하지 않습니다.

## 액세스 제어 목록(ACL)

AWS IoT Greengrass 는 [ACLs](#) 지원하지 않습니다.

## AWS IoT Greengrass 태그 기반 권한 부여

지원되는 AWS IoT Greengrass 리소스에 태그를 연결하거나 요청에 태그를 전달할 수 있습니다 AWS IoT Greengrass. 태그를 기반으로 액세스를 제어하려면 `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공합니다. 자세한 내용은 [리소스에 태그 지정](#) 단원을 참조하십시오.

## 에 대한 IAM 역할 AWS IoT Greengrass

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS 계정 내 엔터티입니다.

### 에서 임시 자격 증명 사용 AWS IoT Greengrass

임시 보안 인증 정보는 페더레이션을 통해 로그인하거나, IAM 역할을 수임하거나, 교차 계정 역할을 수임하는 데 사용됩니다. [AssumeRole](#) 또는 [GetFederationToken](#) 같은 AWS STS API 작업을 호출하여 임시 자격 증명을 가져옵니다.

Greengrass 코어에서는 [디바이스 역할](#)에 대한 임시 자격 증명을 Greengrass 구성 요소에 제공합니다. 구성 요소가 AWS SDK를 사용하는 경우 AWS SDK가 이를 대신 수행하므로 자격 증명을 얻기 위해 로직을 추가할 필요가 없습니다.

### 서비스 연결 역할

AWS IoT Greengrass 는 [서비스 연결 역할](#)을 지원하지 않습니다.

### 서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 [서비스 역할](#)을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

AWS IoT Greengrass 코어 디바이스는 서비스 역할을 사용하여 Greengrass 구성 요소 및 Lambda 함수가 사용자를 대신하여 일부 AWS 리소스에 액세스할 수 있도록 허용합니다. 자세한 내용은 [the section called “코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여”](#) 단원을 참조하십시오.



AWS IoT Greengrass 는 서비스 역할을 사용하여 사용자를 대신하여 일부 AWS 리소스에 액세스합니다. 자세한 내용은 [Greengrass 서비스 역할](#) 단원을 참조하십시오.

## 에 대한 자격 증명 기반 정책 예제 AWS IoT Greengrass

기본적으로 IAM 사용자 및 역할은 AWS IoT Greengrass 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 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

자격 증명 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. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 관리형 정책](#) 또는 [AWS 직무에 대한 관리형 정책](#)을 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 [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. AWS 서비스와 같은 특정을 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 [IAM Access Analyzer에서 정책 검증](#)을 참조하세요.

- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정칩니다. API 작업을 직접적으로 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 [MFA를 통한 보안 API 액세스](#)를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 [IAM의 보안 모범 사례](#)를 참조하세요.

## 정책 예시

다음 예제의 고객 정의 정책은 일반적인 시나리오에 대한 권한을 부여합니다.

### 예제

- [사용자가 자신의 고유한 권한을 볼 수 있도록 허용](#)

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 [JSON 탭에서 정책 생성](#)을 참조하세요.

### 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "ViewOwnUserInfo",
 "Effect": "Allow",
 "Action": [
 "iam:GetUserPolicy",
 "iam:ListGroupsForUser",
 "iam:ListAttachedUserPolicies",
 "iam:ListUserPolicies",
 "iam:GetUser"
],
 "Resource": ["arn:aws:iam::*:user/${aws:username}"]
 },
 {
 "Sid": "NavigateInConsole",
 "Effect": "Allow",

```

```

 "Action": [
 "iam:GetGroupPolicy",
 "iam:GetPolicyVersion",
 "iam:GetPolicy",
 "iam:ListAttachedGroupPolicies",
 "iam:ListGroupPolicies",
 "iam:ListPolicyVersions",
 "iam:ListPolicies",
 "iam:ListUsers"
],
 "Resource": "*"
 }
}

```

## 코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여

AWS IoT Greengrass 코어 디바이스는 AWS IoT Core 자격 증명 공급자를 사용하여 서비스에 대한 호출을 AWS 승인합니다. 자격 AWS IoT Core 증명 공급자를 사용하면 디바이스가 X.509 인증서를 고유한 디바이스 자격 증명으로 사용하여 AWS 요청을 인증할 수 있습니다. 따라서 AWS IoT Greengrass 코어 디바이스에 AWS 액세스 키 ID와 보안 액세스 키를 저장할 필요가 없습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스에 대한 직접 호출 권한 부여](#)를 참조하세요.

AWS IoT Greengrass 코어 소프트웨어를 실행할 때 코어 디바이스에 필요한 AWS 리소스를 프로비저닝하도록 선택할 수 있습니다. 여기에는 코어 디바이스가 AWS IoT Core 자격 증명 공급자를 통해 수입하는 AWS Identity and Access Management (IAM) 역할이 포함됩니다. --provision true 인수를 사용하여 코어 디바이스가 임시 AWS 자격 증명을 가져오도록 허용하는 역할 및 정책을 구성합니다. 또한 이 인수는 이 IAM AWS IoT 역할을 가리키는 역할 별칭을 구성합니다. 사용할 IAM 역할 및 AWS IoT 역할 별칭의 이름을 지정할 수 있습니다. 이러한 이름 파라미터 없이 --provision true를 지정하면 Greengrass 코어 디바이스에서는 다음 기본 리소스를 생성하여 사용합니다.

- IAM 역할: GreengrassV2TokenExchangeRole

역할에는 이름이 GreengrassV2TokenExchangeRoleAccess인 정책과 `credentials.iot.amazonaws.com`에 역할 수입이 허용되는 신뢰 관계가 있습니다. 코어 디바이스에 대한 최소 권한이 정책에 포함되어 있습니다.

**⚠ Important**

이 정책에는 S3 버킷의 파일에 대한 액세스 권한이 포함되어 있지 않습니다. 역할에 권한을 추가하여 코어 디바이스에서 S3 버킷의 구성 요소 아티팩트를 검색하도록 허용해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오.

- AWS IoT 역할 별칭: GreengrassV2TokenExchangeRoleAlias

이 역할 별칭에서는 IAM 역할이 참조됩니다.

자세한 내용은 [3단계: AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

기존 코어 디바이스에 대한 역할 별칭도 설정할 수 있습니다. 그렇게 하려면 [Greengrass nucleus 구성 요소](#)의 `iotRoleAlias` 구성 파라미터를 구성합니다.

이 IAM 역할에 대한 임시 AWS 자격 증명을 획득하여 사용자 지정 구성 요소에서 AWS 작업을 수행할 수 있습니다. 자세한 내용은 [AWS 서비스와 상호 작용](#) 단원을 참조하십시오.

## 주제

- [코어 디바이스에 대한 서비스 역할 권한](#)
- [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#)

## 코어 디바이스에 대한 서비스 역할 권한

역할에서는 다음 서비스의 역할 수임이 허용됩니다.

- `credentials.iot.amazonaws.com`

AWS IoT Greengrass 코어 소프트웨어를 사용하여이 역할을 생성하는 경우 다음 권한 정책을 사용하여 코어 디바이스가 연결하고 로그를 전송할 수 있도록 허용합니다 AWS. 정책의 이름 기본값은 `Access`로 끝나는 IAM 역할의 이름입니다. 예를 들어, 기본 IAM 역할 이름을 사용하는 경우에는 이 정책의 이름이 `GreengrassV2TokenExchangeRoleAccess`입니다.

## Greengrass nucleus v2.5.0 and later

### JSON

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "logs:CreateLogGroup",
 "logs:CreateLogStream",
 "logs:PutLogEvents",
 "logs:DescribeLogStreams",
 "s3:GetBucketLocation"
],
 "Resource": "*"
 }
]
}
```

## v2.4.x

### JSON

```
{
 "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

JSON

```
{
 "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 버킷 액세스 허용

기본 코어 디바이스 역할에서는 코어 디바이스에 S3 버킷 액세스를 허용하지 않습니다. S3 버킷에 아티팩트가 있는 구성 요소를 배포하려면 코어 디바이스에 구성 요소 아티팩트 다운로드를 허용하는 `s3:GetObject` 권한을 추가해야 합니다. 새 정책을 코어 디바이스에 추가하여 이 권한을 부여할 수 없습니다.

## Amazon S3의 구성 요소 아티팩트에 대한 액세스를 허용하는 정책을 추가하는 방법

1. `component-artifact-policy.json`이라는 파일을 생성하고 다음 JSON을 파일로 복사합니다. 이 정책에서는 S3 버킷의 모든 파일에 대한 액세스를 허용합니다. `amzn-s3-demo-bucket`을 S3 버킷의 이름으로 바꾸고 액세스를 코어 디바이스에 허용합니다.

### JSON

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

2. 다음 명령을 사용하여 `component-artifact-policy.json`의 정책 문서에서 정책을 생성합니다.

### 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 정책 Amazon 리소스 이름(ARN)을 복사합니다. 이 ARN을 사용하여 다음 단계에서 이 정책을 코어 디바이스 역할에 연결합니다.

3. 다음 명령을 실행하여 정책을 코어 디바이스 역할에 연결합니다.

*GreengrassV2TokenExchangeRole*을 AWS IoT Greengrass 코어 소프트웨어를 실행할 때 지정한 역할의 이름으로 바꿉니다. 그런 다음에 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 성공한 것이며, 이 S3 버킷에 업로드하는 아티팩트에 코어 디바이스에서 액세스할 수 있습니다.

## 설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책

AWS IoT Greengrass 코어 소프트웨어를 설치할 때 디바이스의 AWS IoT 사물 및 IAM 역할과 같은 필수 AWS 리소스를 프로비저닝할 수 있습니다. 로컬 개발 도구도 디바이스에 배포할 수 있습니다. 설치



관리자에서 이러한 작업을 수행할 수 있도록 AWS 자격 증명이 필요합니다 AWS 계정. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

다음 예제 정책에는 설치 관리자가 이러한 리소스를 프로비저닝하기 위해 수행해야 하는 최소 작업 세트가 포함되어 있습니다. 이러한 권한은 설치 관리자에 --provision 인수를 지정하는 경우 필요합니다. *account-id*를 AWS 계정 ID로 바꾸고 *GreengrassV2TokenExchangeRole*을 --tes-role-name [설치 관리자 인수](#)에 지정하는 토큰 교환 역할의 이름으로 바꿉니다.

#### Note

DeployDevTools 정책 문은 설치 관리자에 --deploy-dev-tools 인수를 지정하는 경우에만 필요합니다.

Greengrass nucleus v2.5.0 and later

JSON

```
{
 "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::123456789012:role/GreengrassV2TokenExchangeRole",
 "arn:aws:iam::123456789012: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": "*"
 }
]
}

```

## Earlier than v2.5.0

## JSON

```
{
 "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::123456789012:role/GreengrassV2TokenExchangeRole",
 "arn:aws:iam::123456789012: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 역할은가 사용자를 대신하여 서비스의 리소스에 액세스할 AWS IoT Greengrass 수 있도록 권한을 부여하는 AWS Identity and Access Management (IAM) AWS 서비스 역할입니다. 이 역할을 사용하면 AWS IoT Greengrass 가 클라이언트 디바이스의 ID를 확인하고 코어 디바이스 연결 정보를 관리할 수 있습니다.

### Note

AWS IoT Greengrass V1 는이 역할을 사용하여 필수 작업을 수행합니다. 자세한 정보는 AWS IoT Greengrass V1 개발자 안내서에서 [Greengrass 서비스 역할](#)을 참조하세요.

가 리소스 AWS IoT Greengrass 에 액세스하도록 허용하려면 Greengrass 서비스 역할을 AWS 계정 와 연결하고를 신뢰할 수 있는 엔터 AWS IoT Greengrass 티로 지정해야 합니다. 역할에는 [AWSGreengrassResourceAccessRolePolicy](#) 관리형 정책 또는 사용하는 AWS IoT Greengrass 기능에 대해 동등한 권한을 정의하는 사용자 지정 정책이 포함되어야 합니다.이 정책은가 AWS 리소스에 액세스하는 데 AWS IoT Greengrass 사용하는 권한 세트를 정의하는이 정책을 AWS 유지합니다. 자세한 내용은 [AWS 관리형 정책: AWSGreengrassResourceAccessRolePolicy](#) 단원을 참조하십시오.

동일한 Greengrass 서비스 역할을 재사용할 수 AWS 리전 있지만 사용하는 모든 AWS 리전 의 계정과 연결해야 합니다 AWS IoT Greengrass. 서비스 역할이 현재에 구성되지 않은 경우 AWS 리전코어 디바이스는 클라이언트 디바이스를 확인하지 못하고 연결 정보를 업데이트하지 못합니다.

다음 섹션에서는 AWS Management Console 또는를 사용하여 Greengrass 서비스 역할을 생성하고 관리하는 방법을 설명합니다 AWS CLI.

#### 주제

- [Greengrass 서비스 역할 관리\(콘솔\)](#)
- [Greengrass 서비스 역할 관리\(CLI\)](#)
- [다음 사항도 참조하세요.](#)

#### Note

서비스 수준 액세스 권한을 부여하는 서비스 역할 외에도 토큰 교환 역할을 Greengrass 코어 디바이스에 할당합니다. 토큰 교환 역할은 코어 디바이스의 Greengrass 구성 요소 및 Lambda 함수가 AWS 서비스에 액세스하는 방법을 제어하는 별도의 IAM 역할입니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

## Greengrass 서비스 역할 관리(콘솔)

AWS IoT 콘솔을 사용하면 Greengrass 서비스 역할을 쉽게 관리할 수 있습니다. 예를 들어, 코어 디바이스에 대한 클라이언트 디바이스 검색을 구성할 때 콘솔은 AWS 계정 이 현재 AWS 리전의 Greengrass 서비스 역할에 연결되어 있는지 확인합니다. 연결이 되어 있지 않으면 콘솔이 사용자를 대신하여 서비스 역할을 생성 및 구성할 수 있습니다. 자세한 내용은 [the section called “Greengrass 서비스 역할 생성”](#) 단원을 참조하십시오.

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 단계에 따라 현재에서가 AWS IoT Greengrass 사용하는 서비스 역할을 찾습니다 AWS 리전.

1. [AWS IoT 콘솔](#)로 이동합니다.
2. 탐색 창에서 설정을 선택합니다.
3. Greengrass service role(Greengrass 서비스 역할) 섹션으로 스크롤하여 서비스 역할과 그 정책을 확인합니다.

서비스 역할이 표시되지 않으면 콘솔에서 역할을 생성 또는 구성할 수 있습니다. 자세한 내용은 [Greengrass 서비스 역할 생성](#) 단원을 참조하십시오.

## Greengrass 서비스 역할 생성(콘솔)

콘솔이 사용자를 대신하여 기본 Greengrass 서비스 역할을 생성 및 구성할 수 있습니다. 이 역할에는 다음 속성이 있습니다.

| 속성            | 값                                                     |
|---------------|-------------------------------------------------------|
| 이름            | Greengrass_ServiceRole                                |
| 신뢰할 수 있는 엔터티. | AWS service: greengrass                               |
| 정책            | <a href="#">AWSGreengrassResourceAccessRolePolicy</a> |

**Note**

[AWS IoT Greengrass V1 디바이스 설정 스크립트](#)로 이 역할을 생성하는 경우 역할 이름은 GreengrassServiceRole\_*random-string*입니다.

코어 디바이스에 대한 클라이언트 디바이스 검색을 구성할 때 콘솔은 Greengrass 서비스 역할이 현재 AWS 계정 의와 연결되어 있는지 확인합니다 AWS 리전. 그렇지 않은 경우 콘솔에서가 사용자를 대신 하여 AWS 서비스를 읽고 쓸 수 AWS IoT Greengrass 있도록 허용하라는 메시지가 표시됩니다.

권한을 부여하는 경우, 콘솔은 Greengrass\_ServiceRole 이라는 이름의 역할이 AWS 계정계정에 존재하는지 여부를 확인합니다.

- 역할이 있는 경우 콘솔은 현재의 AWS 계정 에 서비스 역할을 연결합니다 AWS 리전.
- 역할이 없는 경우 콘솔은 기본 Greengrass 서비스 역할을 생성하여 현재의 AWS 계정 에 연결합니다 AWS 리전.

### Note

사용자 지정 역할 정책을 사용하여 서비스를 생성하려는 경우에는 IAM 콘솔을 사용하여 역할을 생성하거나 수정하세요. 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 권한을 위임할 역할 생성](#) 또는 [역할 수정을 참조하세요](#). 역할이 사용하는 기능 및 리소스에 대한 AWSGreengrassResourceAccessRolePolicy 관리형 정책과 동일한 권한을 부여하는 지 확인합니다. 혼동된 대리자 보안 문제를 방지하려면 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키도 포함하는 것이 좋습니다. 조건 컨텍스트 키는 지정된 계정 및 Greengrass 작업 영역에서 들어오는 요청만 허용하도록 액세스를 제한합니다. 혼동된 대리자 문제에 대한 자세한 내용은, [교차 서비스 혼동된 대리인 방지](#)를 참조하세요. 서비스 역할을 생성하는 경우 AWS IoT 콘솔로 돌아가 역할에 연결합니다 AWS 계정. 설정 페이지의 Greengrass 서비스 역할에서 이 작업을 수행할 수 있습니다.

## Greengrass 서비스 역할 변경(콘솔)

다음 절차에 따라 콘솔 AWS 계정 에서 AWS 리전 현재 선택한의에 연결할 다른 Greengrass 서비스 역할을 선택합니다.

1. [AWS IoT 콘솔](#)로 이동합니다.
2. 탐색 창에서 설정을 선택합니다.
3. Greengrass 서비스 역할에서 역할 선택을 선택합니다.

Greengrass 서비스 역할 업데이트 대화 상자가 열리고에서 신뢰할 수 있는 엔터 AWS IoT Greengrass 티로 AWS 계정 정의하는 IAM 역할이 표시됩니다.

4. 연결할 Greengrass 서비스 역할을 선택합니다.

## 5. 역할 연결을 선택합니다.

### Greengrass 서비스 역할 분리(콘솔)

다음 절차에 따라 현재의 AWS 계정에서 Greengrass 서비스 역할을 분리합니다 AWS 리전. 이렇게 하면 AWS IoT Greengrass 가 현재의 AWS 서비스에 액세스할 수 있는 권한이 취소됩니다 AWS 리전.

#### Important

서비스 역할을 분리하면 진행 중인 작업에 방해가 될 수 있습니다.

1. [AWS IoT 콘솔](#)로 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 프로파일 삭제](#)를 참조하세요.

다른 역할은 리소스에 액세스 AWS IoT Greengrass 하도록 허용할 수 있습니다. AWS IoT Greengrass 이(가) 사용자를 대신하여 권한을 위임할 수 있는 모든 역할을 찾으려면 IAM 콘솔의 역할 페이지의 신뢰할 수 있는 엔터티 열에서 AWS 서비스: greengrass가 포함된 역할을 찾아보세요.

### Greengrass 서비스 역할 관리(CLI)

다음 절차에서는 AWS Command Line Interface 가 설치되고를 사용하도록 구성되어 있다고 가정합니다 AWS 계정. 자세한 내용은 AWS Command Line Interface 사용 설명서의 [AWS CLI설치, 업데이트, 제거](#) 및 [AWS CLI구성](#) 섹션을 참조하세요.

다음 역할 관리 작업에 AWS CLI 를 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 계정 과 연결되어 있는지 알아냅니다.

- 서비스 역할을 가져옵니다. *region*을 로 바꿉니다 AWS 리전 (예: us-west-2).

```
aws greengrassv2 get-service-role-for-account --region region
```

Greengrass 서비스 역할이 계정과 이미 연결되어 있는 경우 요청에서 다음 역할 메타데이터가 반환됩니다.

```
{
 "associatedAt": "timestamp",
 "roleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

요청에서 역할 메타데이터가 반환되지 않는 경우 서비스 역할을 생성하고(없는 경우) 해당 역할을 AWS 리전의 계정과 연결해야 합니다.

## Greengrass 서비스 역할 생성(CLI)

다음 단계에 따라 역할을 생성하고 AWS 계정과 연결합니다.

IAM을 사용하여 서비스 역할을 생성하려면

1. 가 역할을 AWS IoT Greengrass 수임하도록 허용하는 신뢰 정책을 사용하여 역할을 생성합니다. 이 예제에서는 Greengrass\_ServiceRole라는 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 대리인 방지](#)를 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

```

## 서비스 역할을와 연결하려면 AWS 계정

- 역할을 계정과 연결합니다. *role-arn*을 서비스 역할 ARN으로 바꾸고 *region*을 로 바꿉니다 AWS 리전 (예: us-west-2).

```

aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region

```

성공하면 요청에서 다음과 비슷한 응답이 반환됩니다.

```

{
 "associatedAt": "timestamp"
}

```

## Greengrass 서비스 역할 제거(CLI)

다음 단계를 사용하여 AWS 계정에서 Greengrass 서비스 역할의 연결을 해제합니다.

- 계정에서 서비스 역할의 연결을 해제합니다. *region*을 로 바꿉니다 AWS 리전 (예: us-west-2).

```
aws greengrassv2 disassociate-service-role-from-account --region region
```

성공하면 다음 응답이 반환됩니다.

```
{
 "disassociatedAt": "timestamp"
}
```

### Note

서비스 역할을 사용하지 않는 경우 삭제해야 합니다 AWS 리전. 먼저 [delete-role-policy](#)를 사용하여 역할에서 AWSGreengrassResourceAccessRolePolicy 관리형 정책을 연결 해제하고, [delete-role](#)을 사용하여 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서에서 [역할 또는 인스턴스 프로파일 삭제](#)를 참조하세요.

다음 사항도 참조하세요.

- IAM 사용 설명서의 [AWS 서비스에 권한을 위임하는 역할 생성](#)
- [IAM 사용자 설명서](#)에서 역할 수정
- 자세한 내용은 IAM 사용 설명서에서 [역할 또는 인스턴스 프로파일 삭제](#)를 참조하세요.
- AWS IoT Greengrass 명령 AWS CLI 참조의 명령
  - [associate-service-role-to-account](#)
  - [disassociate-service-role-from-account](#)
  - [get-service-role-for-account](#)
- AWS CLI 명령 참조에서 IAM 명령
  - [attach-role-policy](#)
  - [create-role](#)
  - [delete-role](#)
  - [delete-role-policy](#)

## AWS 에 대한 관리형 정책 AWS IoT Greengrass

AWS 관리형 정책은에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 [고객 관리형 정책](#)을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 관리형 정책에 정의된 권한을 AWS 업데이트하는 AWS 경우 업데이트는 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 줍니다. AWS AWS 서비스 는 새가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용자 가이드의 [AWS 관리형 정책](#)을 참조하세요.

### 주제

- [AWS 관리형 정책: AWSGreengrassFullAccess](#)
- [AWS 관리형 정책: AWSGreengrassReadOnlyAccess](#)
- [AWS 관리형 정책: AWSGreengrassResourceAccessRolePolicy](#)
- [AWS IoT Greengrass AWS 관리형 정책에 대한 업데이트](#)

### AWS 관리형 정책: AWSGreengrassFullAccess

AWSGreengrassFullAccess 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 위탁자에게 모든 AWS IoT Greengrass 작업에 대한 전체 액세스 권한을 허용하는 관리 권한을 부여합니다.

### 권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- greengrass – 위탁자에게 모든 AWS IoT Greengrass 작업에 대한 전체 액세스 권한을 허용합니다.

## JSON

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

## AWS 관리형 정책: AWSGreengrassReadOnlyAccess

AWSGreengrassReadOnlyAccess 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 위탁자가 AWS IoT Greengrass의 정보를 볼 수 있지만 수정할 수는 없는 읽기 전용 권한을 부여합니다. 예를 들어 이러한 권한이 있는 위탁자는 Greengrass 코어 디바이스에 배포된 구성 요소 목록을 볼 수 있지만 해당 디바이스에서 실행되는 구성 요소를 변경하기 위한 배포를 생성할 수는 없습니다.

## 권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `greengrass` - 위탁자가 항목의 목록이나 항목에 대한 세부 정보를 반환하는 작업을 수행할 수 있도록 허용합니다. 이는 `List` 또는 `Get`으로 시작하는 API 작업을 포함합니다.

## JSON

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

```

 "greengrass:List*",
 "greengrass:Get*"
],
 "Resource": "*"
}
]
}

```

## AWS 관리형 정책: AWSGreengrassResourceAccessRolePolicy

AWSGreengrassResourceAccessRolePolicy 정책을 IAM 엔터티에 연결할 수 있습니다. AWS IoT Greengrass 는가 사용자를 대신하여 작업을 AWS IoT Greengrass 수행하도록 허용하는 서비스 역할에도이 정책을 연결합니다. 자세한 내용은 [Greengrass 서비스 역할](#) 단원을 참조하십시오.

이 정책은가 Lambda 함수 검색, AWS IoT 디바이스 새도 관리, Greengrass 클라이언트 디바이스 확인과 같은 필수 작업을 AWS IoT Greengrass 수행할 수 있도록 허용하는 관리 권한을 부여합니다.

### 권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- **greengrass** – Greengrass 리소스를 관리합니다.
- **iot (\*Shadow)** - 이름에 다음과 같은 특수 식별자가 있는 AWS IoT 새도우를 관리합니다. 이러한 권한은 AWS IoT Greengrass 가 코어 디바이스와 통신하는 데 필요합니다.
  - **\*-gci** - 클라이언트 디바이스가 코어 디바이스를 검색하고 연결할 수 있도록이 새도우를 AWS IoT Greengrass 사용하여 코어 디바이스 연결 정보를 저장합니다.
  - **\*-gcm** – AWS IoT Greengrass V1은이 새도를 사용하여 코어 디바이스에 Greengrass 그룹의 인증 기관(CA) 인증서가 교체되었음을 알립니다.
  - **\*-gda** – AWS IoT Greengrass V1은이 새도우를 사용하여 코어 디바이스에 배포를 알립니다.
  - **GG\_\*** - 사용되지 않습니다.
- **iot (DescribeThing 및 DescribeCertificate)** - AWS IoT 사물 및 인증서에 대한 정보를 검색합니다. 가 코어 디바이스에 연결하는 클라이언트 디바이스를 AWS IoT Greengrass 확인하려면 이러한 권한이 필요합니다. 자세한 내용은 [로컬 IoT 디바이스와 상호 작용](#) 단원을 참조하십시오.
- **lambda** - AWS Lambda 함수에 대한 정보를 검색합니다. 이 권한은 AWS IoT Greengrass V1이 Greengrass 코어에 Lambda 함수를 배포할 수 있도록 하기 위해 필요합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서 [의 AWS IoT Greengrass 코어에서 Lambda 함수 실행](#)을 참조하십시오.

- **secretsmanager** - 이름이 로 시작하는 AWS Secrets Manager 보안 암호의 값을 검색합니다. 이 권한은 AWS IoT Greengrass V1이 Secrets Manager 보안 암호를 Greengrass 코어에 배포할 수 있도록 하기 위해 필요합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [AWS IoT Greengrass 코어에 보안 암호 배포](#)를 참조하세요.
- **s3** - 이름에 greengrass 또는 sagemaker를 포함하는 S3 버킷에서 파일 객체를 검색합니다. 이러한 권한은 AWS IoT Greengrass V1이 S3 버킷에 저장하는 기계 학습 리소스를 배포할 수 있도록 하기 위해 필요합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [기계 학습 리소스를](#) 참조하세요.
- **sagemaker** - Amazon SageMaker AI 기계 학습 추론 모델에 대한 정보를 검색합니다. 이 권한은 AWS IoT Greengrass V1이 Greengrass 코어에 ML 모델을 배포할 수 있도록 하기 위해 필요합니다. 자세한 내용은 AWS IoT Greengrass V1 개발자 안내서의 [기계 학습 추론 수행을](#) 참조하세요.

## JSON

```
{
 "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 IoT Greengrass AWS 관리형 정책에 대한 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 시점부터 AWS IoT Greengrass의 AWS 관리형 정책 업데이트에 대한 세부 정보를 볼 수 있습니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 [AWS IoT Greengrass V2 문서 기록 페이지에서](#) RSS 피드를 구독하세요.

| 변경                               | 설명                                                   | Date        |
|----------------------------------|------------------------------------------------------|-------------|
| AWS IoT Greengrass에서 변경 내용 추적 시작 | AWS IoT Greengrass가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다. | 2021년 7월 2일 |

## 교차 서비스 혼동된 대리인 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 에서 AWS교차 서비스 가장은 혼동된 대리자 문제를 초래할 수 있습니다. 교차 서비스 가장은 한 서비스(직접 호출하는 서비스)가 다른 서비스(직접 호출되는 서비스)를 직접 호출할 때 발생할 수 있습니다. 직접 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해 AWS 에서 는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 위탁자를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다.

리소스 정책에서 [aws:SourceArn](#) 및 [aws:SourceAccount](#) 전역 조건 컨텍스트 키를 사용하여 리소스에 다른 서비스를 AWS IoT Greengrass 제공하는 권한을 제한하는 것이 좋습니다. 두 전역 조건 컨텍스트 키를 모두 사용하는 경우 `aws:SourceAccount` 값과 `aws:SourceArn` 값의 계정은 동일한 정책 문에서 사용할 경우 동일한 계정 ID를 사용해야 합니다.

`aws:SourceArn`의 값은 `sts:AssumeRole` 요청과 관련된 Greengrass 고객 리소스여야 합니다.

혼동된 대리인 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`전역 조건 컨텍스트 키를 사용합니다. 예제: `arn:aws:greengrass::account-id:*`.

`aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키가 사용되는 정책의 예제는 [Greengrass 서비스 역할 생성](#) 섹션을 참조하세요.

## 에 대한 자격 증명 및 액세스 문제 해결 AWS IoT Greengrass

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS IoT Greengrass 하고 수정할 수 있습니다.

### 문제

- [에서 작업을 수행할 권한이 없음 AWS IoT Greengrass](#)
- [iam:PassRole을 수행하도록 인증되지 않음](#)
- [관리자인데 다른 사람에게 액세스하도록 허용하려고 함 AWS IoT Greengrass](#)
- [내 외부의 사람이 내 AWS IoT Greengrass 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.](#)

일반적인 문제 해결 도움말은 [문제 해결](#) 섹션을 참조하십시오.

## 에서 작업을 수행할 권한이 없음 AWS IoT Greengrass

작업을 수행할 권한이 없다는 오류가 수신되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 mateojackson IAM 사용자가 코어 디바이스에 대한 세부 정보를 보려고 하지만 greengrass:GetCoreDevice 권한이 없는 경우에 발생합니다.

```
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 작업을 사용하여 greengrass:GetCoreDevice 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

다음은 AWS IoT Greengrass에서 작업할 때 발생할 수 있는 일반적인 IAM 문제입니다.

### iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS IoT Greengrass에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 AWS IoT Greengrass에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

이 경우, Mary가 iam:PassRole 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

## 관리자인데 다른 사람 이에 액세스하도록 허용하려고 함 AWS IoT Greengrass

다른 사용자에게 액세스를 허용하려면 액세스 권한이 필요한 사용자 또는 애플리케이션에 권한을 부여해야 AWS IoT Greengrass합니다. AWS IAM Identity Center 를 사용하여 사용자 및 애플리케이션을

관리하는 경우 사용자 또는 그룹에 권한 세트를 할당하여 액세스 수준을 정의합니다. 권한 세트는 IAM 정책을 자동으로 생성하고 사용자 또는 애플리케이션과 연결된 IAM 역할에 할당합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 [권한 세트](#)를 참조하세요.

IAM Identity Center를 사용하지 않는 경우 액세스가 필요한 사용자 또는 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 그런 다음 AWS IoT Greengrass에 대한 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다. 권한이 부여되면 사용자 또는 애플리케이션 개발자에게 자격 증명을 제공합니다. 이들은 이 자격 증명을 사용하여 AWS에 액세스합니다. IAM 사용자, 그룹, 정책 및 권한 생성에 대해 자세히 알아보려면 IAM 사용자 설명서의 [IAM 자격 증명](#)과 [IAM의 권한 및 정책](#)을 참조하세요.

내 외부의 사람이 내 AWS IoT Greengrass 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사용자가 AWS 리소스에 액세스하는 데 사용할 수 있는 IAM 역할을 생성할 수 있습니다. 역할을 수입할 신뢰할 수 있는 사람을 지정할 수 있습니다. 자세한 내용은 [IAM 사용 설명서의 AWS 계정 자신이 소유한 다른의 IAM 사용자에게 액세스 권한 제공 및 타사가 소유한에 액세스 권한 제공을 참조 AWS 계정](#)하세요.

AWS IoT Greengrass 는 리소스 기반 정책 또는 액세스 제어 목록(ACLs)을 기반으로 하는 교차 계정 액세스를 지원하지 않습니다.

## 프록시 또는 방화벽을 통해 디바이스 트래픽 허용

Greengrass 코어 디바이스 및 Greengrass 구성 요소는 AWS 서비스 및 기타 웹 사이트에 대한 아웃바운드 요청을 수행합니다. 보안 조치로 아웃바운드 트래픽을 엔드포인트와 포트의 작은 범위로 제한할 수 있습니다. 엔드포인트와 포트에 대한 다음 정보를 사용하여 프록시, 방화벽 또는 [Amazon VPC 보안 그룹](#)을 통한 디바이스 트래픽을 제한할 수 있습니다. 프록시를 사용할 코어 디바이스를 구성하는 방법에 대한 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결](#) 섹션을 참조하세요.

### 주제

- [기본 작업을 위한 엔드포인트](#)
- [자동 프로비저닝으로 설치하기 위한 엔드포인트](#)
- [AWS제공 구성 요소에 대한 엔드포인트](#)

## 기본 작업을 위한 엔드포인트

Greengrass 코어 디바이스에서는 다음과 같은 엔드포인트와 포트가 기본 작업에 사용됩니다.

## AWS IoT 엔드포인트 검색

의 AWS IoT 엔드포인트를 가져 AWS 계정와 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. 해결 방법:

1. 의 AWS IoT 데이터 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 의 AWS IoT 자격 증명 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "endpointAddress": "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
}
```

| 엔드포인트                                                | 포트          | 필수 | 설명                                                                     |
|------------------------------------------------------|-------------|----|------------------------------------------------------------------------|
| greengrass-ats.iot<br>. <i>region</i> .amazonaws.com | 8443 또는 443 | 예  | 데이터 폴<br>레인 작업<br>에 사용됩<br>니다(예: 배<br>포 설치 및<br>클라이언트<br>디바이스와<br>연동). |

| 엔드포인트                                                               | 포트                                          | 필수 | 설명                                                                |
|---------------------------------------------------------------------|---------------------------------------------|----|-------------------------------------------------------------------|
| <i>device-data-prefix</i> -ats.iot.<br><i>region</i> .amazonaws.com | MQTT: 8883 또는 443<br><br>HTTPS: 8443 또는 443 | 예  | 디바이스 관리를 위한 데이터 플레인 작업에 사용됩니다(예: MQTT 통신 및 AWS IoT Core와 새도 동기화). |

| 엔드포인트                                                                                       | 포트  | 필수 | 설명                                                                                                                                                            |
|---------------------------------------------------------------------------------------------|-----|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>device-credentials-prefix</code> .credentials.iot. <code>region</code> .amazonaws.com | 443 | 예  | 코어 디바이스가 Amazon S3에서 구성 요소 아티팩트를 다운로드하고 다른 작업을 수행하는 데 사용하는 AWS 자격 증명을 획득하는 데 사용됩니다. 자세한 내용은 <a href="#">코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여 단원을 참조하십시오</a> . |



| 엔드포인트                                                                                                       | 포트  | 필수 | 설명                                                                       |
|-------------------------------------------------------------------------------------------------------------|-----|----|--------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>*.s3.amazonaws.com</li><li>*.s3.<i>region</i>.amazonaws.com</li></ul> | 443 | 예  | 배포에 사용됩니다. 엔드포인트 접두사는 내부적으로 제어되고 언제든지 변경될 수 있으므로 이 형식에는 * 문자가 포함되어 있습니다. |

| 엔드포인트                                  | 포트  | 필수  | 설명                                                                                                                                                                                                            |
|----------------------------------------|-----|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| data.iot. <i>region</i> .amazonaws.com | 443 | 아니요 | 코어 디바이스에 v2.4.0 보다 이전 버전의 <a href="#">Greengrass nucleus</a> 가 실행되고 네트워크 프록시를 사용하도록 구성된 경우에 필요합니다. 코어 디바이스는 프록시 뒤에 있을 AWS IoT Core 때와 MQTT 통신에이 엔드포인트를 사용합니다. 자세한 내용은 <a href="#">네트워크 프록시 구성</a> 단원을 참조하십시오. |

## 자동 프로비저닝으로 설치하기 위한 엔드포인트

Greengrass 코어 디바이스는 [자동 리소스 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치할](#) 때 다음 엔드포인트와 포트를 사용합니다.

| 엔드포인트                                    | 포트  | 필수  | 설명                                                     |
|------------------------------------------|-----|-----|--------------------------------------------------------|
| iot. <i>region</i> .amazonaws.com        | 443 | 예   | AWS IoT 리소스를 생성하고 기존 AWS IoT 리소스에 대한 정보를 검색하는 데 사용됩니다. |
| iam.amazonaws.com                        | 443 | 예   | IAM 리소스를 생성하고 기존 IAM 리소스에 대한 정보를 검색하는 데 사용됩니다.         |
| sts. <i>region</i> .amazonaws.com        | 443 | 예   | 의 ID를 가져오는 데 사용됩니다<br>AWS 계정.                          |
| greengrass. <i>region</i> .amazonaws.com | 443 | 아니요 | --<br>deploy-dev-tools<br>인수를 사용하여 Greengrass          |

| 엔드포인트 | 포트 | 필수 | 설명                                    |
|-------|----|----|---------------------------------------|
|       |    |    | s CLI 구성 요소를 코어 디바이스에 배포하는 경우에 필요합니다. |

## AWS제공 구성 요소에 대한 엔드포인트

Greengrass 코어 디바이스에서는 실행하는 소프트웨어 구성 요소에 따라 추가 엔드포인트가 사용됩니다. 이 개발자 안내서의 각 구성 요소 페이지에 있는 요구 사항 섹션에서 각 AWS제공 구성 요소에 필요한 엔드포인트를 찾을 수 있습니다. 자세한 내용은 [AWS 제공 구성 요소](#) 단원을 참조하십시오.

## 의 규정 준수 검증AWS IoT Greengrass

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하고 관심 있는 규정 준수 프로그램을 선택하십시오. 일반적인 정보는 [AWS 규정 준수 프로그램](#)을 참조하세요.

AWS Artifact(을)를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 [AWS Artifact에서 보고서 다운로드](#)를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률 및 규정에 따라 결정됩니다. AWS 서비스 사용 시 규정 준수 책임에 대한 자세한 내용은 [AWS 보안 설명서](#)를 참조하세요.

## FIPS 엔드포인트

AWS IoT Greengrass 는 FIPS([Federal Information Processing Standard](#) 140-2) 엔드포인트 사용을 지원합니다. FIPS 모드가 활성화되면 HTTP 및 MQTT 프로토콜을 포함하여 AWS 클라우드 서비스에 대한 모든 데이터 전송이 해당 FIPS 준수 엔드포인트([FIPS - Amazon Web Services\(AWS\)](#))를 호출하고 연결을 설정해야 합니다.

IoT 데이터플레인 FIPS 엔드포인트(FIPS 엔드포인트에 연결 -) 및 개발된 FIPS 호환 암호화 라이브러리 aws AWS-Ic를 AWS IoT 활용하기 위한 MQTT 통신입니다.[AWS IoTAWS IoT Core](#)

## Greengrass의 HTTP 통신:

- nucleus 및 플러그인 구성 요소의 경우 모든 SDK HTTP 클라이언트는 시스템 속성 `AWS_USE_FIPS_ENDPOINT`를 `true`로 설정하여 FIPS 엔드포인트로 구성됩니다.
- 일반 구성 요소의 경우 모든 구성 요소는 시스템 속성 `AWS_USE_FIPS_ENDPOINT`가 `true`로 설정된 상태에서 시작됩니다. 이 프로세스를 통해 이러한 일반 구성 요소에 사용되는 SDK HTTP 클라이언트가 FIPS 호환 엔드포인트로 요청을 보낼 수 있습니다.

### Note

스트림 관리자의 경우 nucleus는 환경 변수 `AWS_GG_FIPS_MODE`를 전달합니다. 이 환경 변수를 사용하면 스트림 관리자 내에서 사용되는 HTTP 클라이언트가 해당 FIPS 준수 엔드포인트를 식별 및 연결할 수 있습니다.

AWS IoT Greengrass 는 프로비저닝과 배포라는 두 가지 방법으로 FIPS 모드를 활성화합니다. FIPS 모드를 활성화하기 위해 구성 파라미터 `fipsMode`를 `true`로 설정해야 하고, 이후 nucleus는 시스템 속성 `AWS_USE_FIPS_ENDPOINT`를 `true`로 설정하고 환경 변수로 다른 모든 구성 요소로 전파합니다. 또한 AWS IoT Greengrass 는 루트 CA 인증서(CA3)를 다운로드하여 기존 `rootCA.pem`(또는 `AmazonRootCA1.pem`) 파일에 추가합니다. 새 배포를 통해 FIPS를 활성화하면 FIPS 모드를 활성화한 후 시스템 속성이 적용되도록 nucleus가 재시작됩니다.

`fipsMode` 파라미터 구성 외에도 `iotDataEndpoint`, `iotCredEndpoint` 및 `greengrassDataEndpoint` 파라미터도 구성해야 합니다. 자세한 내용은 아래의 관련 문서를 참조하세요.

## 배포를 통해 FIPS 엔드포인트 활성화

의 AWS IoT 엔드포인트를 가져 AWS 계정와 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. `iotDataEndpoint` 및 `iotCredEndpoint`라는 두 엔드포인트가 있습니다. 해결 방법:

1. FIPS 데이터 [AWS IoT Core 데이터 플레인 엔드포인트](#)에서 리전의 FIPS 데이터 엔드포인트를 가져옵니다. 의 FIPS 데이터 엔드포인트는 다음과 같아 AWS 계정 야 합니다. `data.iot-fips.us-west-2.amazonaws.com`
2. FIPS 데이터 [AWS IoT Core 데이터 플레인 엔드포인트](#)에서 리전의 FIPS 자격 증명 엔드포인트를 가져옵니다. 의 FIPS 자격 증명 엔드포인트는 다음과 같아 AWS 계정 야 합니다. `data.credentials.iot-fips.us-west-2.amazonaws.com`

이후 배포로 FIPS를 활성화하려면 다음 구성을 nucleus에 적용해야 합니다. 배포 시 병합할 구성은 다음과 같습니다.

## Console

### 병합할 구성

```
{
 "fipsMode": "true",
 "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
 "greengrassDataPlaneEndpoint": "iotData",
 "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"
}
```

## AWS CLI

다음 명령은 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

dashboard-deployment.json 파일에는 다음 JSON 문서가 포함되어 있습니다.

```
{
 "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
 "deploymentName": "Deployment for MyGreengrassCore",
 "components": {
 "aws.greengrass.Nucleus": {
 "componentVersion": "2.13.0",
 "configurationUpdate": {
 "merge": {"fipsMode\\":\\"true\\",\\"iotDataEndpoint\\":\\"data.iot-fips.us-west-2.amazonaws.com\\",\\"greengrassDataPlaneEndpoint\\":\\"iotData\\",\\"iotCredEndpoint\\":\\"data.credentials.iot-fips.us-west-2.amazonaws.com\\"}"
 }
 }
 }
}
```

## Greengrass CLI

다음 [Greengrass CLI](#) 명령은 코어 디바이스에 로컬 배포를 생성합니다.

```
sudo greengrass-cli deployment create \
 --recipeDir recipes \
 --artifactDir artifacts \
 --merge "aws.greengrass.Nucleus=2.13.0" \
 --update-config dashboard-configuration.json
```

dashboard-configuration.json 파일에는 다음 JSON 문서가 포함됩니다.

```
{
 "aws.greengrass.Nucleus": {
 "MERGE": {
 "fipsMode": "true",
 "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
 "greengrassDataPlaneEndpoint": "iotData",
 "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"
 }
 }
}
```

## 수동 리소스 프로비저닝을 사용하여 FIPS 엔드포인트로 nucleus 설치

FIPS 엔드포인트를 사용하여 AWS IoT Greengrass V2 코어 디바이스에 대한 AWS 리소스를 수동으로 프로비저닝

### Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인합니다.

### 주제

- [AWS IoT 엔드포인트 검색](#)
- [AWS IoT 사물 생성](#)
- [사물 인증서 생성](#)
- [토큰 교환 역할 생성](#)
- [디바이스에 인증서 다운로드](#)
- [디바이스 환경 설정](#)

- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)

## AWS IoT 엔드포인트 검색

의 AWS IoT 엔드포인트를 가져 AWS 계정와 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. `iotDataEndpoint` 및 `iotCredEndpoint`라는 두 엔드포인트가 있습니다. 해결 방법:

1. FIPS 데이터 [AWS IoT Core 데이터 플레인 엔드포인트](#)에서 리전의 FIPS 데이터 엔드포인트를 가져옵니다. 의 FIPS 데이터 엔드포인트는 다음과 같아 AWS 계정 야 합니다. *`data.iot-fips.us-west-2.amazonaws.com`*
2. FIPS 데이터 [AWS IoT Core 데이터 플레인 엔드포인트](#)에서 리전의 FIPS 자격 증명 엔드포인트를 가져옵니다. 의 FIPS 자격 증명 엔드포인트는 다음과 같아 AWS 계정 야 합니다. *`data.credentials.iot-fips.us-west-2.amazonaws.com`*

## AWS IoT 사물 생성

AWS IoT 사물에는 연결하는 디바이스 및 논리적 엔터티를 나타냅니다 AWS IoT. Greengrass 코어 디바이스는 AWS IoT 사물입니다. 디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다 AWS.

이 섹션에서는 디바이스를 나타내는 AWS IoT 사물을 생성합니다.

AWS IoT 사물을 생성하려면

1. 디바이스에 대한 AWS IoT 사물을 생성합니다. 개발 컴퓨터에서 다음 명령을 실행합니다.
  - *`MyGreengrassCore`*를 사용할 사물 이름으로 바꿉니다. 이 이름은 Greengrass 코어 디바이스의 이름이기도 합니다.

### Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

```
aws iot create-thing --thing-name MyGreengrassCore
```



요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "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 또는 기존 사물 그룹에 사물을 추가합니다. 사물 그룹을 사용하여 Greengrass 코어 디바이스 플릿을 관리합니다. 소프트웨어 구성 요소를 디바이스에 배포할 때 개별 디바이스 또는 디바이스 그룹을 대상으로 지정할 수 있습니다. 활성 Greengrass 배포가 있는 사물 그룹에 디바이스를 추가하여 사물 그룹의 소프트웨어 구성 요소를 디바이스에 배포할 수 있습니다. 해결 방법:

- a. (선택 사항) AWS IoT 사물 그룹을 생성합니다.

- *MyGreengrassCoreGroup*을 생성할 사물 그룹의 이름으로 바꿉니다.

 Note

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "thingGroupName": "MyGreengrassCoreGroup",
 "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
 "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. AWS IoT 사물을 사물 그룹에 추가합니다.

- *MyGreengrassCore*를 AWS IoT 사물의 이름으로 바꿉니다.
- *MyGreengrassCoreGroup*을 사물 그룹의 이름으로 바꿉니다.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

요청에 성공하는 경우 명령에 출력이 없습니다.

## 사물 인증서 생성

디바이스를 AWS IoT 사물로 등록하면 해당 디바이스는 디지털 인증서를 사용하여 인증할 수 있습니다 AWS. 이 인증서를 사용하면 디바이스가 AWS IoT 및와 통신할 수 있습니다 AWS IoT Greengrass.

이 섹션에서는 디바이스가 AWS에 연결하는 데 사용할 수 있는 인증서를 생성하고 다운로드합니다.

하드웨어 보안 모듈(HSM)을 사용하여 프라이빗 키와 인증서를 안전하게 저장하도록 AWS IoT Greengrass 코어 소프트웨어를 구성하려면 단계에 따라 HSM의 프라이빗 키에서 인증서를 생성합니다. 그렇지 않으면 단계에 따라 AWS IoT 서비스에서 인증서와 프라이빗 키를 생성합니다. 하드웨어 보안 기능은 Linux 디바이스에서만 사용할 수 있습니다. 하드웨어 보안 및 이를 사용하는 데 필요한 요구 사항에 대한 자세한 내용은 [하드웨어 보안 통합](#) 섹션을 참조하세요.

### AWS IoT 서비스에서 인증서 및 프라이빗 키 생성

#### 사물 인증서를 생성하려면

1. AWS IoT 사물에 대한 인증서를 다운로드할 폴더를 생성합니다.

```
mkdir greengrass-v2-certs
```

2. AWS IoT 사물에 대한 인증서를 생성하고 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
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
 "certificateId": "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
```

```

"certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBIDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXRhbnQ2I2sYWMxHZAAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2I2sYWMx
HZAAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEibb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
 "keyPair": {
 "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQ0CAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEUuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJT2o
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUSTzEcCyNCx2EXAMPLEv9mQ0UXP6plfgxwKRX2fEXAMPLEda\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLE1nI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
 "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
 }
}

```

이후 인증서 구성에 사용할 인증서의 Amazon 리소스 이름(ARN)을 저장합니다.

## HSM의 프라이빗 키에서 인증서 생성

**Note**

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

## 사물 인증서를 생성하려면

1. 코어 디바이스에서 HSM의 PKCS#11 토큰을 초기화하고, 프라이빗 키를 생성합니다. 프라이빗 키는 RSA-2048 키 크기(또는 그 이상)의 RSA 키나 ECC 키여야 합니다.

**Note**

ECC 키와 함께 하드웨어 보안 모듈을 사용하려면 [Greengrass nucleus](#) v2.5.6 이상을 사용해야 합니다.

하드웨어 보안 모듈 및 [보안 암호 관리자](#)를 사용하려면 RSA 키가 있는 하드웨어 보안 모듈을 사용해야 합니다.

HSM의 설명서를 확인하여 토큰을 초기화하고 프라이빗 키를 생성하는 방법을 알아봅니다. HSM 이 객체 ID를 지원하는 경우 프라이빗 키를 생성할 때 객체 ID를 지정합니다. 토큰을 초기화하고 프라이빗 키를 생성할 때 지정한 슬롯 ID, 사용자 PIN, 객체 레이블, 객체 ID(HSM에서 이를 사용하는 경우)를 저장합니다. 나중에 사물 인증서를 HSM으로 가져오고 AWS IoT Greengrass 코어 소프트웨어를 구성할 때 이러한 값을 사용합니다.

2. 프라이빗 키에서 CSR(인증서 서명 요청)을 생성합니다. AWS IoT 는 이 CSR을 사용하여 HSM에서 생성한 프라이빗 키의 사물 인증서를 생성합니다. 프라이빗 키에서 CSR을 생성하는 방법에 대한 자세한 내용은 HSM 설명서를 참조하세요. CSR은 `iotdevicekey.csr`과 같은 파일입니다.
3. 디바이스에서 개발 컴퓨터로 CSR을 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 `scp` 명령을 사용하여 CSR을 전송할 수 있습니다. `device-ip-address`를 디바이스의 IP 주소로 바꾸고, `~/iotdevicekey.csr`을 디바이스의 CSR 파일 경로로 바꿉니다.

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 개발 컴퓨터에서 AWS IoT 사물에 대한 인증서를 다운로드할 폴더를 생성합니다.

```
mkdir greengrass-v2-certs
```

5. CSR 파일을 사용하여 개발 컴퓨터에 AWS IoT 사물에 대한 인증서를 생성 및 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
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
 "certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
 "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xZDASBgNVBASTC01BTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXR0Q21sYWMxHjAdBgkqhkiG9w0BCQEWEG5vb25lQGftYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xZDASBgNVBASTC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWMx
HjAdBgkqhkiG9w0BCQEWEG5vb25lQGftYXpvi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

이후 인증서 구성에 사용할 인증서의 ARN을 저장합니다.

그런 다음 사물 인증서를 구성합니다. 자세한 내용은 [사물 인증서 구성](#) 단원을 참조하십시오.

## 토큰 교환 역할 생성

Greengrass 코어 디바이스는 토큰 교환 역할이라는 IAM 서비스 역할을 사용하여 AWS 서비스에 대한 호출을 승인합니다. 디바이스는 자격 AWS IoT 증명 공급자를 사용하여이 역할에 대한 임시 AWS 자격 증명을 가져오므로 디바이스가 상호 작용하고 AWS IoT, Amazon CloudWatch Logs로 로그를 전송하고, Amazon S3에서 사용자 지정 구성 요소 아티팩트를 다운로드할 수 있습니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

AWS IoT Anrole 별칭을 사용하여 Greengrass 코어 디바이스에 대한 토큰 교환 역할을 구성합니다. 역할 별칭을 사용하면 디바이스의 토큰 교환 역할을 변경할 수 있지만 디바이스 구성은 동일하게 유지할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS 서비스 직접 호출에 대한 권한 부여](#)를 참조하세요.

이 섹션에서는 토큰 교환 IAM 역할과 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다. 이미 Greengrass 코어 디바이스를 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 생성하려면

1. 디바이스가 토큰 교환 역할로 사용할 수 있는 IAM 역할을 생성합니다. 해결 방법:
  - a. 토큰 교환 역할에 필요한 신뢰 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-trust-policy.json
```

다음 JSON을 파일로 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.

- *GreengrassV2TokenExchangeRole*을 생성할 IAM 역할의 이름으로 바꿉니다.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "Role": {
 "Path": "/",
 "RoleName": "GreengrassV2TokenExchangeRole",
 "RoleId": "AR0AZ2YMUHYHK50KM77FB",
 "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 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano device-role-access-policy.json
```

다음 JSON을 파일로 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 S3에서 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오.

구성 요소 아티팩트에 대한 S3 버킷이 아직 없는 경우 버킷을 생성한 후 권한을 추가할 수 있습니다.

d. 정책 문서에서 IAM 정책을 생성합니다.

- *GreengrassV2TokenExchangeRoleAccess*를 생성할 IAM 정책의 이름으로 바꿉니다.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "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.

- *GreengrassV2TokenExchangeRole*을 IAM 역할의 이름으로 바꿉니다.
- 정책 ARN을 이전 단계에서 생성한 IAM 정책의 ARN으로 변경합니다.

```

aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess

```

요청에 성공하는 경우 명령에 출력이 없습니다.

2. 토큰 교환 AWS IoT 역할을 가리키는 역할 별칭을 생성합니다.

- *GreengrassCoreTokenExchangeRoleAlias*를 생성할 역할 별칭의 이름으로 바꿉니다.
- 역할 ARN을 이전 단계에서 생성한 IAM 역할의 ARN으로 변경합니다.

```

aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole

```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```

{
 "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
 "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}

```

**Note**

역할 별칭을 생성하려면 토큰 교환 IAM 역할을 AWS IoT에 전달할 권한이 있어야 합니다. 역할 별칭을 생성하려고 할 때 오류 메시지가 표시되면 AWS 사용자에게 권한이 있는지 확인합니다. 자세한 내용은 사용 AWS Identity and Access Management 설명서의 [AWS 서비스에 역할을 전달할 수 있는 사용자 권한 부여](#)를 참조하세요.

3. Greengrass 코어 디바이스가 역할 별칭을 사용하여 토큰 교환 역할을 수임하도록 허용하는 AWS IoT 정책을 생성하고 연결합니다. 이전에 Greengrass 코어 디바이스를 설정한 경우 새 디바이스를 생성하는 대신 역할 별칭 AWS IoT 정책을 연결할 수 있습니다. 해결 방법:

- a. (선택 사항) 역할 별칭에 필요한 AWS IoT 정책 문서가 포함된 파일을 생성합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano greengrass-v2-iot-role-alias-policy.json
```

다음 JSON을 파일로 복사합니다.

- 리소스 ARN을 역할 별칭의 ARN으로 바꿉니다.

```
{
 "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.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*를 생성할 AWS IoT 정책의 이름으로 바꿉니다.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "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 사물의 인증서에 AWS IoT 정책을 연결합니다.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*를 역할 별칭 AWS IoT 정책의 이름으로 바꿉니다.
- 대상 ARN을 AWS IoT 사물 인증서의 ARN으로 바꿉니다.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

요청에 성공하는 경우 명령에 출력이 없습니다.

## 디바이스에 인증서 다운로드

앞서 디바이스의 인증서를 개발 컴퓨터에 다운로드했습니다. 이 섹션에서는 인증서를 코어 디바이스에 복사하여 AWS IoT에 연결하는 데 사용하는 인증서로 디바이스를 설정합니다. Amazon 루트 CA(인증 기관) 인증서도 다운로드합니다. HSM을 사용하는 경우 이 섹션의 HSM으로 인증서 파일도 가져옵니다.

- 이전에 AWS IoT 서비스에서 사물 인증서와 프라이빗 키를 생성한 경우 단계에 따라 프라이빗 키와 인증서 파일이 있는 인증서를 다운로드합니다.
- 먼저 HSM(하드웨어 보안 모듈)의 프라이빗 키에서 사물 인증서를 생성한 경우 단계에 따라 프라이빗 키와 HSM의 인증서가 포함된 인증서를 다운로드합니다.

### 프라이빗 키 및 인증서 파일이 포함된 인증서 다운로드

#### 디바이스에 인증서를 다운로드하려면

1. 개발 컴퓨터에서 디바이스로 AWS IoT 사물 인증서를 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 `scp` 명령을 사용하여 인증서를 전송할 수 있습니다. `device-ip-address`를 디바이스의 IP 주소로 바꿉니다.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중 에이 폴더에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

#### Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: C:\greengrass\v2 또는 D:\greengrass\v2)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

### Linux or Unix

- `/greengrass/v2`를 사용할 폴더로 바꿉니다.

```
sudo mkdir -p /greengrass/v2
```

## Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

## PowerShell

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 권한을 설정합니다.

- `/greengrass`를 루트 폴더의 상위로 바꿉니다.

```
sudo chmod 755 /greengrass
```

4. AWS IoT 사물 인증서를 Greengrass 루트 폴더에 복사합니다.

## Linux or Unix

- `/greengrass/v2`를 Greengrass 루트 폴더로 바꿉니다.

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

## Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
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 CA(인증 기관) 인증서를 다운로드합니다. AWS IoT 인증서는 기본적으로 Amazon의 루트 CA 인증서와 연결됩니다. CA1 인증서와 [CA3 인증서](#)를 다운로드합니다.

#### Linux or Unix

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
greengrass/v2/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
```

#### HSM에서 프라이빗 키 및 인증서 파일이 포함된 인증서 다운로드

##### Note

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서이 기능을 지원하지 않습니다.

## 디바이스에 인증서를 다운로드하려면

1. 개발 컴퓨터에서 디바이스로 AWS IoT 사물 인증서를 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 `scp` 명령을 사용하여 인증서를 전송할 수 있습니다. `device-ip-address`를 디바이스의 IP 주소로 바꿉니다.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중 에이 폴더에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

### Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: C:\greengrass\v2 또는 D:\greengrass\v2)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

## Linux or Unix

- `/greengrass/v2`를 사용할 폴더로 바꿉니다.

```
sudo mkdir -p /greengrass/v2
```

## Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

## PowerShell

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 권한을 설정합니다.

- `/greengrass`를 루트 폴더의 상위로 바꿉니다.

```
sudo chmod 755 /greengrass
```

4. 사물 인증서 파일인 `~/greengrass-v2-certs/device.pem.crt`를 HSM으로 가져옵니다. 인증서를 가져오는 방법을 알아보려면 HSM 설명서를 확인하세요. 이전에 HSM에서 프라이빗 키를 생성한 것과 동일한 토큰, 슬롯 ID, 사용자 PIN, 객체 레이블 및 객체 ID(HSM에서 사용하는 경우)를 사용하여 인증서를 가져옵니다.

#### Note

앞서 객체 ID 없이 프라이빗 키를 생성했고, 인증서에 객체 ID가 있는 경우 프라이빗 키의 객체 ID를 인증서와 동일한 값으로 설정합니다. 프라이빗 키 객체의 객체 ID를 설정하는 방법을 알아보려면 HSM의 설명서를 확인하세요.

5. (선택 사항) HSM에만 존재하도록 사물 인증서 파일을 삭제합니다.

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. Amazon 루트 CA(인증 기관) 인증서를 다운로드합니다. AWS IoT 인증서는 기본적으로 Amazon의 루트 CA 인증서와 연결됩니다. CA1 및 [CA3 인증서](#)를 모두 다운로드합니다.

#### Linux or Unix

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
greengrass/v2/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 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

### Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.
  - a. 다음 명령을 실행하여 /etc/sudoers 파일을 엽니다.

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgrouops](#) v1을 활성화하고 memory 및 devices cgrouops를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgrouops 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

## Windows 디바이스 설정

### Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
  - a. Windows 키를 눌러 시작 메뉴를 엽니다.
  - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
  - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
  - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
  - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
  - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.

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

- g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
  4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

### Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고

만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- wmic 명령이 더는 사용되지 않는 Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

#### Note

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

## AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

**Note**

다음 위치에서 특정 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

## AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다. `greengrass-nucleus-latest.zip`.

### 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 계약](#)에 동의하는 것입니다.

2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

**Note**

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.

- a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6\_10*을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6\_10*을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.
  - i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

- ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. -verify 및 -certs 옵션과 함께 Jarsigner -verbose 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.
3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다.  
*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

## 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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

### Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후이 폴더를 제거할 수 있습니다.

## AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자를 실행합니다.

- 이전에 생성한 AWS 리소스와 인증서를 사용하도록 지정하는 부분 구성 파일에서 설치합니다. AWS IoT Greengrass 코어 소프트웨어는 디바이스에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 관리자는 제공된 부분 구성 파일에서 전체 구성 파일을 생성합니다.

- 코어 디바이스에서 소프트웨어 구성 요소를 실행하는 데 `ggc_user` 시스템 사용자가 사용되도록 지정합니다. Linux 디바이스에서 이 명령은 `ggc_group` 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

### Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

### Note

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 `nucleus` 구성 요소의 `jvmOptions` 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

- 이전에 AWS IoT 서비스에서 사물 인증서와 프라이빗 키를 생성한 경우 단계에 따라 프라이빗 키와 인증서 파일이 있는 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.
- 이전에 하드웨어 보안 모듈(HSM)의 프라이빗 키에서 사물 인증서를 생성한 경우 단계에 따라 프라이빗 키와 인증서가 있는 AWS IoT Greengrass 코어 소프트웨어를 HSM에 설치합니다.

프라이빗 키 및 인증서 파일을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.

- ***GreengrassInstaller***를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```



## 2. 텍스트 편집기를 사용하여 config.yaml 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 내용을 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터와 Greengrass nucleus 파라미터를 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: "2.16.1"
 configuration:
 awsRegion: "us-west-2"
 iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
 fipsMode: "true"
 iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
 greengrassDataPlaneEndpoint: "iotData"
 iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

뒤이어 다음과 같이 하세요.

- */greengrass/v2*의 각 인스턴스를 Greengrass 루트 폴더로 바꿉니다.
- *MyGreengrassCore*를 AWS IoT 사물 이름으로 바꿉니다.
- *2.16.1*을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.
- *us-west-2*를 리소스를 생성한 AWS 리전 으로 바꿉니다.
- *GreengrassCoreTokenExchangeRoleAlias*를 토큰 교환 역할 별칭의 이름으로 바꿉니다.
- *iotDataEndpoint*를 AWS IoT 데이터 엔드포인트로 바꿉니다.
- *iotCredEndpoint*를 자격 AWS IoT 증명 엔드포인트로 바꿉니다.

## 3. 설치 관리자를 실행하고, --init-config를 지정하여 구성 파일을 제공합니다.

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.
- `GreengrassInstaller`의 각 인스턴스를 설치 관리자를 압축 해제한 폴더로 바꿉니다.

## 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 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

**Note**

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

#### 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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

#### HSM에서 프라이빗 키와 인증서를 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치

**Note**

이 기능은 [Greengrass nucleus 구성 요소의](#) v2.5.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 코어 디바이스에서 이 기능을 지원하지 않습니다.

#### AWS IoT Greengrass 코어 소프트웨어를 설치하려면

##### 1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.

- *GreengrassInstaller*를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. AWS IoT Greengrass 코어 소프트웨어가 HSM에서 프라이빗 키와 인증서를 사용하도록 하려면 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 [PKCS#11 공급자 구성 요소를](#) 설치합니다. PKCS#11 공급자 구성 요소는 설치 도중 구성할 수 있는 플러그인입니다. 다음 위치에서 PKCS#11 공급자 구성 요소의 최신 버전을 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>

PKCS#11 공급자 플러그인을 `aws.greengrass.crypto.Pkcs11Provider.jar`이라는 파일에 다운로드합니다. *GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

```
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 계약](#)에 동의하는 것입니다.

3. 텍스트 편집기를 사용하여 `config.yaml` 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 내용을 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터, Greengrass nucleus 파라미터 및 PKCS#11 공급자 파라미터를 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: "2.16.1"
 configuration:
 awsRegion: "us-west-2"
 iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
 fipsMode: "true"
 iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
 greengrassDataPlaneEndpoint: "iotData"
 iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

뒤이어 다음과 같이 하세요.

- PKCS#11 URI의 각 *iotdevicekey* 인스턴스를 프라이빗 키를 생성하고 인증서를 가져온 객체 레이블로 바꿉니다.
  - */greengrass/v2*의 각 인스턴스를 Greengrass 루트 폴더로 바꿉니다.
  - *MyGreengrassCore*를 AWS IoT 사물 이름으로 바꿉니다.
  - *2.16.1*을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.
  - *us-west-2*를 리소스를 생성한 AWS 리전 으로 바꿉니다.
  - *GreengrassCoreTokenExchangeRoleAlias*를 토큰 교환 역할 별칭의 이름으로 바꿉니다.
  - 를 AWS IoT 데이터 엔드포인트 *iotDataEndpoint*로 바꿉니다.
  - *iotCredEndpoint*를 AWS IoT 자격 증명 엔드포인트로 바꿉니다.
  - *aws.greengrass.crypto.Pkcs11Provider* 구성 요소의 구성 파라미터를 코어 디바이스의 HSM 구성 값으로 바꿉니다.
4. 설치 관리자를 실행하고, *--init-config*를 지정하여 구성 파일을 제공합니다.
- */greengrass/v2*를 Greengrass 루트 폴더로 바꿉니다.
  - *GreengrassInstaller*의 각 인스턴스를 설치 관리자를 압축 해제한 폴더로 바꿉니다.

```
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 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

**i Note**

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

**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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass 참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

## 플릿 프로비저닝을 사용하여 FIPS 엔드포인트 설치

이 기능은 [Greengrass nucleus 구성 요소](#) v2.4.0 이상에서 사용할 수 있습니다.

AWS IoT Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝을 사용하여 코어 소프트웨어에 FIPS 엔드포인트를 설치합니다.

### Note

플릿 프로비저닝 플러그인은 현재 HSM(하드웨어 보안 모듈)에 프라이빗 키 및 인증서 파일을 저장하는 것을 지원하지 않습니다. HSM을 사용하려면 [수동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치합니다](#).

AWS IoT 플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 Greengrass 코어 디바이스를 프로비저닝 AWS 계정 하는 데 AWS IoT 사용하는 리소스에 설정해야 합니다. 이러한 리소스에는 프로비저닝 템플릿, 클레임 인증서 및 [토큰 교환 IAM 역할](#)이 포함됩니다. 이러한 리소스를 생성한 후 재사용하여 플릿에 여러 코어 디바이스를 프로비저닝할 수 있습니다. 자세한 내용은 [Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝 설정](#) 단원을 참조하십시오.

### Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0을 설치하고 실행하기 위한 [요구 사항](#)을 충족하는지 확인합니다.

## 주제

- [사전 조건](#)
- [AWS IoT 엔드포인트 검색](#)

- [디바이스에 인증서 다운로드](#)
- [디바이스 환경 설정](#)
- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT 플릿 프로비저닝 플러그인 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)

## 사전 조건

AWS IoT 플릿 프로비저닝과 함께 AWS IoT Greengrass 코어 소프트웨어를 설치하려면 먼저 [Greengrass 코어 디바이스에 대한 AWS IoT 플릿 프로비저닝을 설정해야](#) 합니다. 이 단계를 한 번 완료한 후 플릿 프로비저닝을 사용하여 원하는 수의 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치할 수 있습니다.

## AWS IoT 엔드포인트 검색

에 대한 FIPS 엔드포인트를 가져 AWS 계정과 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. 해결 방법:

1. FIPS 데이터 [AWS IoT Core 데이터 플레인 엔드포인트](#)에서 리전의 FIPS 데이터 엔드포인트를 가져옵니다. 의 FIPS 데이터 엔드포인트는 다음과 같아 AWS 계정 야 합니다. `data.iot-fips.us-west-2.amazonaws.com`
2. FIPS 데이터 [AWS IoT Core 데이터 플레인 엔드포인트](#)에서 리전의 FIPS 자격 증명 엔드포인트를 가져옵니다. 의 FIPS 자격 증명 엔드포인트는 다음과 같아 AWS 계정 야 합니다. `data.credentials.iot-fips.us-west-2.amazonaws.com`

## 디바이스에 인증서 다운로드

디바이스는 클레임 인증서와 프라이빗 키를 사용하여 AWS 리소스를 프로비저닝하고 X.509 디바이스 인증서를 획득하기 위한 요청을 인증합니다. 제조 중에 클레임 인증서와 프라이빗 키를 디바이스에 내장하거나 설치 중에 인증서와 키를 디바이스에 복사할 수 있습니다. 이 섹션에서는 클레임 인증서와 프라이빗 키를 디바이스에 복사합니다. 또한 Amazon 루트 CA(인증 기관) 인증서를 디바이스에 다운로드할 수도 있습니다.

### Important

프로비저닝 클레임 프라이빗 키는 Greengrass 코어 디바이스 등 어디에서든 항상 보안 유지되어야 합니다. Amazon CloudWatch 지표 및 로그를 사용하여 디바이스 프로비저닝을 위한 클레



임 인증서의 무단 사용과 같은 오용 징후를 모니터링하는 것이 좋습니다. 오용이 감지되면 프로비저닝 클레임 인증서를 디바이스 프로비저닝에 사용할 수 없도록 비활성화합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [AWS IoT모니터링](#)를 참조하세요.

디바이스 수와에 자신을 등록하는 디바이스를 더 잘 관리할 수 있도록 AWS 계정폴릿 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크를 지정할 수 있습니다. 사전 프로비저닝 후크는 디바이스가 등록 중에 제공하는 템플릿 파라미터를 검증하는 AWS Lambda 함수입니다. 예를 들어 디바이스 ID를 데이터베이스와 비교하여 디바이스에 프로비저닝 권한이 있는지 확인하는 사전 프로비저닝 후크를 생성할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 [사전 프로비저닝 후크](#)를 참조하세요.

## 디바이스에 클레임 인증서 다운로드

1. 클레임 인증서와 프라이빗 키를 디바이스에 복사합니다. 개발 컴퓨터와 디바이스에서 SSH 및 SCP가 활성화된 경우 개발 컴퓨터에서 scp 명령을 사용하여 클레임 인증서와 프라이빗 키를 전송할 수 있습니다. 다음 예제 명령은 개발 컴퓨터의 claim-certs 폴더에서 이러한 파일을 디바이스로 전송합니다. *device-ip-address*를 디바이스의 IP 주소로 바꿉니다.

```
scp -r claim-certs/ device-ip-address:~
```

2. 디바이스에 Greengrass 루트 폴더를 생성합니다. 나중 에이 폴더에 AWS IoT Greengrass 코어 소프트웨어를 설치합니다.

### Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 루트 폴더(예: C:\greengrass\v2 또는 D:\greengrass\v2)를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

## Linux or Unix

- */greengrass/v2*를 사용할 폴더로 바꿉니다.

```
sudo mkdir -p /greengrass/v2
```

## Windows Command Prompt

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

## PowerShell

- `C:\greengrass\v2`를 사용할 폴더로 바꿉니다.

```
mkdir C:\greengrass\v2
```

3. (Linux만 해당) Greengrass 루트 폴더의 상위 권한을 설정합니다.

- `/greengrass`를 루트 폴더의 상위로 바꿉니다.

```
sudo chmod 755 /greengrass
```

4. 클레임 인증서를 Greengrass 루트 폴더로 옮깁니다.

- `/greengrass/v2` 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.

## 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. CA1 인증서와 [CA3 인증서](#)를 모두 다운로드합니다.

## Linux or Unix

```
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /greengrass/v2/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 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

### Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

## Windows 디바이스 설정

### Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

## 용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
  - a. Windows 키를 눌러 시작 메뉴를 엽니다.
  - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
  - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
  - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
  - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
  - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.

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

- g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

#### Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- [wmic 명령이 더는 사용되지 않는](#) Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

**Note**

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

## AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

**Note**

다음 위치에서 특정 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

## AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다greengrass-nucleus-latest.zip.

### 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 계약](#)에 동의하는 것입니다.

### 2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

#### Note

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.

#### a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6\_10*을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6\_10*을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -verify -certs -verbose greengrass-nucleus-latest.zip
```

#### b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.



- i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

- ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. `-verify` 및 `-certs` 옵션과 함께 `Jarsigner -verbose` 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다.  
*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

#### 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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./ GreengrassInstaller/lib/Greengrass.jar --version
```

#### Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후이 폴더를 제거할 수 있습니다.

## AWS IoT 플릿 프로비저닝 플러그인 다운로드

다음 위치에서 최신 버전의 AWS IoT 플릿 프로비저닝 플러그인을 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar>

### Note

다음 위치에서 특정 버전의 AWS IoT 플릿 프로비저닝 플러그인을 다운로드할 수 있습니다. **version**을 다운로드할 버전으로 바꿉니다. 플릿 프로비저닝 플러그인의 각 버전에 대한 자세한 내용은 [AWS IoT 플릿 프로비저닝 플러그인 변경 로그](#) 섹션을 참조하세요.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

플릿 프로비저닝 플러그인은 오픈 소스입니다. 소스 코드를 보려면 GitHub의 [AWS IoT 플릿 프로비저닝 플러그인](#)을 참조하세요.

AWS IoT 플릿 프로비저닝 플러그인을 다운로드하려면

- 디바이스에서 AWS IoT 플릿 프로비저닝 플러그인을 라는 파일에 다운로드합니다  
다aws.greengrass.FleetProvisioningByClaim.jar. **GreengrassInstaller**를 사용하여려는 폴더로 바꿉니다.

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 계약](#)에 동의하는 것입니다.

## AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자를 실행합니다.

- 플릿 프로비저닝 플러그인을 사용하여 AWS 리소스를 프로비저닝하도록 지정하는 부분 구성 파일에서 설치를 실행합니다. AWS IoT Greengrass 코어 소프트웨어는 디바이스에 있는 모든 Greengrass 구성 요소의 구성을 지정하는 구성 파일을 사용합니다. 설치 관리자는 사용자가 제공하는 부분 구성 파일과 플릿 프로비저닝 플러그인이 생성하는 AWS 리소스에서 전체 구성 파일을 생성합니다.
- ggc\_user 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 ggc\_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.
- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

### Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

**Note**

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 nucleus 구성 요소의 jvmOptions 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. AWS IoT Greengrass 코어 소프트웨어의 버전을 확인합니다.

- *GreengrassInstaller*를 소프트웨어가 포함된 폴더의 경로로 바꿉니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 텍스트 편집기를 사용하여 config.yaml 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 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 [AWS IoT 플릿 프로비저닝 플러그인 구성](#) 섹션을 참조하세요.

Linux or Unix

```

services:
 aws.greengrass.Nucleus:
 version: "2.16.1"
 configuration:
 fipsMode: "true"
 greengrassDataPlaneEndpoint: "iotData"
 aws.greengrass.FleetProvisioningByClaim:
 configuration:
```

```

rootPath: "/greengrass/v2"
awsRegion: "us-west-2"
iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
iotCredEndpoint: "data.credentials.iot-fips.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: "2.16.1"
 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"

```

뒤이어 다음과 같이 하세요.

- 2.16.1을 AWS IoT Greengrass 코어 소프트웨어 버전으로 바꿉니다.

- `/greengrass/v2`의 각 인스턴스 또는 `C:\greengrass\v2`를 Greengrass 루트 폴더로 바꿉니다.

#### Note

Windows 디바이스에서는 경로 구분자를 이중 백슬래시(\\)로 지정해야 합니다(예: `C:\\greengrass\\v2`).

- `us-west-2`를 프로비저닝 템플릿 및 기타 리소스를 생성한 AWS 리전으로 바꿉니다.
  - `iotDataEndpoint`를 AWS IoT 데이터 엔드포인트로 바꿉니다.
  - `iotCredentialEndpoint`를 AWS IoT 자격 증명 엔드포인트로 바꿉니다.
  - `GreengrassCoreTokenExchangeRoleAlias`를 토큰 교환 역할 별칭의 이름으로 바꿉니다.
  - `GreengrassFleetProvisioningTemplate`을 플릿 프로비저닝 템플릿의 이름으로 바꿉니다.
  - `claimCertificatePath`를 디바이스의 클레임 인증서 경로로 바꿉니다.
  - `claimCertificatePrivateKeyPath`를 디바이스의 클레임 인증서 프라이빗 키 경로로 바꿉니다.
  - 템플릿 파라미터(`templateParameters`)를 디바이스 프로비저닝에 사용하는 값으로 바꿉니다. 이 예제는 `ThingName` 및 `ThingGroupName` 파라미터를 정의하는 [예제 템플릿](#)을 참조합니다.
3. 설치 관리자를 실행합니다. `--trusted-plugin`을 지정하여 플릿 프로비저닝 플러그인을 제공하고, `--init-config`를 지정하여 구성 파일을 제공합니다.
- `/greengrass/v2`를 Greengrass 루트 폴더로 바꿉니다.
  - `GreengrassInstaller`의 각 인스턴스를 설치 관리자를 압축 해제한 폴더로 바꿉니다.

## 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 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

--setup-system-service true를 지정하면 설치 관리자는 소프트웨어를 시스템 서비스로 설정하고 실행한 경우 Successfully set up Nucleus as a system service를 출력합니다. 그렇지 않은 경우 소프트웨어가 성공적으로 설치되면 설치 관리자에서 메시지가 출력되지 않습니다.

### Note

--provision true 인수 없이 설치 관리자를 실행할 때는 deploy-dev-tools 인수를 사용하여 로컬 개발 도구를 배포할 수 없습니다. Greengrass CLI를 디바이스에 직접 배포하는 방법에 대한 자세한 내용은 [Greengrass 명령줄 인터페이스](#) 섹션을 참조하세요.

## 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 폴더에 config, packages 및 logs와 같은 여러 폴더가 포함됩니다.

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass 참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

## 자동 리소스 프로비저닝을 사용하여 FIPS 엔드포인트 설치

AWS IoT Greengrass 코어 소프트웨어에는 디바이스를 Greengrass 코어 디바이스로 설정하는 설치 프로그램이 포함되어 있습니다. 디바이스를 빠르게 설정하기 위해 설치 관리자는 코어 디바이스가 작동하는 데 필요한 AWS IoT 사물, AWS IoT 사물 그룹, IAM 역할 및 AWS IoT 역할 별칭을 프로비저닝할 수 있습니다. 설치 관리자는 또한 로컬 개발 도구를 코어 디바이스에 배포할 수 있으므로 디바이스를 사용하여 사용자 지정 소프트웨어 구성 요소를 개발하고 테스트할 수 있습니다. 설치 관리자에서 이러한 리소스를 프로비저닝하고 배포를 생성하려면 AWS 자격 증명이 필요합니다.

디바이스에 AWS 자격 증명을 제공할 수 없는 경우 코어 디바이스가 AWS 작동하는 데 필요한 리소스를 프로비저닝할 수 있습니다. 개발 도구를 코어 디바이스에 배포하여 개발 디바이스로 사용할 수도 있



습니다. 이렇게 하면 설치 관리자 실행 시 디바이스에 더 적은 권한을 제공할 수 있습니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

#### Important

AWS IoT Greengrass 코어 소프트웨어를 다운로드하기 전에 코어 디바이스가 AWS IoT Greengrass 코어 소프트웨어 v2.0 설치 및 실행 [요구 사항](#)을 충족하는지 확인합니다.

## 주제

- [디바이스 환경 설정](#)
- [디바이스에 AWS 자격 증명 제공](#)
- [AWS IoT Greengrass 코어 소프트웨어 다운로드](#)
- [AWS IoT Greengrass 코어 소프트웨어 설치](#)

## 디바이스 환경 설정

이 섹션의 단계에 따라 AWS IoT Greengrass 코어 디바이스로 사용할 Linux 또는 Windows 디바이스를 설정합니다.

### Linux 디바이스 설정

용 Linux 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 Java 런타임을 설치합니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다. 다음 명령은 디바이스에 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
```

- Amazon Linux 2023의 경우:

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

설치가 완료되면 다음 명령을 실행하여 Linux 디바이스에서 Java가 실행되는지 확인합니다.

```
java -version
```

디바이스에서 실행되는 Java 버전이 명령을 통해 인쇄됩니다. 예를 들어 Debian 기반 배포의 경우 출력이 다음 샘플과 유사할 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) 디바이스에서 구성 요소를 실행하는 기본 시스템 사용자와 그룹을 생성합니다. 설치 관리자 인수를 사용하여 설치하는 동안 AWS IoT Greengrass 코어 소프트웨어 --component-default-user 설치 관리자가 사용자와 그룹을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [설치 프로그램 인수](#) 단원을 참조하십시오.

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

3. AWS IoT Greengrass 코어 소프트웨어(일반적으로 root)를 실행하는 사용자에게 모든 사용자 및 모든 그룹에서 실행할 수 sudo 있는 권한이 있는지 확인합니다.

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

```
sudo visudo
```

- b. 사용자에게 대한 권한이 다음 예제와 같은지 확인합니다.

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

4. (선택 사항) [컨테이너화된 Lambda 함수를 실행](#)하려면 [cgroups](#) v1을 활성화하고 memory 및 devices cgroups를 활성화하고 탑재해야 합니다. 컨테이너화된 Lambda 함수를 실행할 계획이 없는 경우 이 단계를 건너뛸 수 있습니다.

이러한 cgroups 옵션을 활성화하려면 다음 Linux 커널 파라미터로 디바이스를 부팅합니다.

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

디바이스의 커널 파라미터를 보고 설정하는 방법에 대한 자세한 내용은 운영 체제 및 부트 로더 설명서를 참조하세요. 지침에 따라 커널 파라미터를 영구적으로 설정합니다.

5. [디바이스 요구 사항](#)의 요구 사항 목록에 따라 디바이스에 기타 모든 필수 종속성을 설치합니다.

## Windows 디바이스 설정

### Note

이 기능은 [Greengrass nucleus 구성 요소](#)의 v2.5.0 이상에서 사용할 수 있습니다.

## 용 Windows 디바이스를 설정하려면 AWS IoT Greengrass V2

1. AWS IoT Greengrass 코어 소프트웨어가 실행해야 하는 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 추가합니다. LocalSystem 계정은 AWS IoT Greengrass 코어 소프트웨어를 실행하므로 사용자의 PATH 사용자 변수 대신 PATH 시스템 변수에 Java를 추가해야 합니다. 해결 방법:
  - a. Windows 키를 눌러 시작 메뉴를 엽니다.
  - b. 시작 메뉴에서 **environment variables**를 입력하여 시스템 옵션을 검색합니다.
  - c. 시작 메뉴 검색 결과에서 시스템 환경 변수 편집을 선택하여 시스템 속성 창을 엽니다.
  - d. 환경 변수...를 선택하여 환경 변수 창을 엽니다.
  - e. 시스템 변수에서 경로를 선택하고 편집을 선택합니다. 환경 변수 편집 창에서 각 경로를 별도의 줄로 볼 수 있습니다.
  - f. Java 설치 bin 폴더의 경로가 있는지 확인합니다. 경로는 다음 예제와 유사할 수 있습니다.

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

- g. 경로에서 Java 설치 bin 폴더가 누락된 경우 새로 만들기를 선택하여 추가한 다음 확인을 선택합니다.
3. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
4. Windows 디바이스의 LocalSystem 계정에 기본 사용자를 생성합니다. ##를 안전한 암호로 바꿉니다.

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

#### Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- [wmic 명령이 더는 사용되지 않는](#) Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

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

5. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
6. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다. 이전에 설정한 사용자의 암호로 ##를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

**Note**

Windows 디바이스에서 LocalSystem 계정은 Greengrass nucleus를 실행하고 PsExec 유틸리티를 사용하여 기본 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

## 디바이스에 AWS 자격 증명 제공

설치 관리자가 필요한 AWS 리소스를 프로비저닝할 수 있도록 디바이스에 AWS 자격 증명을 제공합니다. 필요한 권한에 대한 자세한 정보는 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

디바이스에 AWS 자격 증명을 제공하려면

- 설치 관리자가 코어 디바이스에 대한 AWS IoT 및 IAM 리소스를 프로비저닝할 수 있도록 디바이스에 AWS 자격 증명을 제공합니다. 보안을 강화하려면 프로비저닝에 필요한 최소 권한만 허용하는 IAM 역할에 대한 임시 자격 증명을 받는 것이 좋습니다. 자세한 내용은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

**Note**

설치 관리자에서는 자격 증명을 저장하거나 저장하지 않습니다.

디바이스에서 다음 중 하나를 수행하여 자격 증명을 검색하고 AWS IoT Greengrass 코어 소프트웨어 설치 관리자가 사용할 수 있도록 합니다.

- (권장)의 임시 자격 증명 사용 AWS IAM Identity Center
  - a. IAM Identity Center에서 액세스 키 ID, 비밀 액세스 키, 세션 토큰을 제공합니다. 자세한 내용은 IAM Identity Center 사용 설명서의 [임시 자격 증명 가져오기 및 새로 고침](#)에서 수동 자격 증명 새로 고침을 참조하세요.
  - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

## Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

## PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- IAM 역할에서 임시 보안 인증을 사용합니다.
  - a. 수입하는 IAM 역할에서 액세스 키 ID, 비밀 액세스 키, 세션 토큰을 제공합니다. 이러한 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서에서 [임시 보안 인증 요청](#)을 참조하세요.
  - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

## Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

## Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

## PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- IAM 사용자의 장기 자격 증명 사용:
  - a. IAM 사용자의 액세스 키 ID 및 비밀 액세스 키를 제공합니다. 나중에 삭제하는 프로비저닝을 위한 IAM 사용자를 생성할 수 있습니다. 사용자에게 제공할 IAM 정책은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요. 장기 자격 증명을 검색하는 방법에 대한 자세한 내용은 IAM 사용 설명서에서 [IAM 사용자의 액세스 키 관리](#)를 참조하세요.
  - b. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어에 자격 증명을 제공합니다.

#### 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 사용자를 삭제합니다.
- d. (선택 사항) 기존 IAM 사용자의 액세스 키 ID 및 비밀 액세스 키를 사용한 경우 해당 키가 더 이상 유효하지 않도록 사용자의 키를 업데이트합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [액세스 키 업데이트](#)를 참조하세요.

## AWS IoT Greengrass 코어 소프트웨어 다운로드

다음 위치에서 최신 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다.

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

**Note**

다음 위치에서 특정 버전의 AWS IoT Greengrass 코어 소프트웨어를 다운로드할 수 있습니다. 다운로드할 버전으로 *version*을 바꿉니다.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

**AWS IoT Greengrass 코어 소프트웨어를 다운로드하려면**

1. 코어 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다greengrass-nucleus-latest.zip.

**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 계약](#)에 동의하는 것입니다.

2. (선택 사항) Greengrass nucleus 소프트웨어 서명을 확인하려면

**Note**

이 기능은 Greengrass nucleus 버전 2.9.5 이상에서 사용할 수 있습니다.



- a. 다음 명령을 사용하여 Greengrass nucleus 아티팩트의 서명을 확인합니다.

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6\_10*을 설치한 JDK 버전으로 바꿉니다.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

파일 이름은 설치하는 JDK 버전에 따라 다르게 보일 수 있습니다. *jdk17.0.6\_10*을 설치한 JDK 버전으로 바꿉니다.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 간접 호출 시 확인 결과를 나타내는 출력이 생성됩니다.

- i. Greengrass nucleus zip 파일이 서명되면 출력에 다음 문이 포함됩니다.

```
jar verified.
```

- ii. Greengrass nucleus zip 파일이 서명되지 않으면 출력에 다음 문이 포함됩니다.

```
jar is unsigned.
```

- c. -verify 및 -certs 옵션과 함께 Jarsigner -verbose 옵션을 제공한 경우 출력에는 자세한 서명자 인증서 정보도 포함됩니다.

3. 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다.

*GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

## 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 코어 소프트웨어의 버전을 확인합니다.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

### Important

v2.4.0 이전 버전의 Greengrass nucleus를 설치하는 경우 AWS IoT Greengrass 코어 소프트웨어를 설치한 후에는이 폴더를 제거하지 마십시오. AWS IoT Greengrass 코어 소프트웨어는이 폴더의 파일을 사용하여 실행합니다.

최신 버전의 소프트웨어를 다운로드한 경우 v2.4.0 이상을 설치하고 AWS IoT Greengrass 코어 소프트웨어를 설치한 후이 폴더를 제거할 수 있습니다.

## AWS IoT Greengrass 코어 소프트웨어 설치

다음 작업을 지정하는 인수를 사용하여 설치 관리자 실행:

- 코어 디바이스가 작동하는 데 필요한 AWS 리소스를 생성합니다.
- ggc\_user 시스템 사용자를 사용하도록 지정하여 코어 디바이스에서 소프트웨어 구성 요소를 실행합니다. Linux 디바이스에서 이 명령은 ggc\_group 시스템 그룹을 사용하도록 지정하고 설치 관리자가 시스템 사용자와 그룹을 생성합니다.

- AWS IoT Greengrass 코어 소프트웨어를 부팅 시 실행되는 시스템 서비스로 설정합니다. Linux 디바이스에서는 [Systemd](#) init 시스템이 필요합니다.

#### Important

Windows 코어 디바이스에서는 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정해야 합니다.

로컬 개발 도구를 사용하여 개발 디바이스를 설정하려면 `--deploy-dev-tools true` 인수를 지정합니다. 설치 완료 후 로컬 개발 도구를 배포하는 데 최대 1분이 걸릴 수 있습니다.

지정할 수 있는 인수에 대한 자세한 내용은 [설치 프로그램 인수](#) 섹션을 참조하세요.

#### Note

메모리가 제한된 디바이스 AWS IoT Greengrass 에서 실행하는 경우 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 메모리 양을 제어할 수 있습니다. 메모리 할당을 제어하기 위해 nucleus 구성 요소의 `jvmOptions` 구성 파라미터에서 JVM 힙 크기 옵션을 설정할 수 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

## AWS IoT Greengrass 코어 소프트웨어를 설치하려면

1. 텍스트 편집기를 사용하여 `config.yaml` 구성 파일을 생성하고 설치 관리자에 제공합니다.

예를 들어 Linux 기반 시스템에서 다음 명령을 실행하면 GNU nano를 사용하여 파일을 생성할 수 있습니다.

```
nano GreengrassInstaller/config.yaml
```

다음 YAML 내용을 파일에 복사합니다. 이 부분 구성 파일은 시스템 파라미터와 Greengrass nucleus 파라미터를 지정합니다.

```

services:
 aws.greengrass.Nucleus:
 configuration:
 fipsMode: "true"
```

```
iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
greengrassDataPlaneEndpoint: "iotData"
```

- *us-west-2*를 리소스를 생성한 AWS 리전 으로 바꿉니다.
  - *iotDataEndpoint*를 AWS IoT 데이터 엔드포인트로 바꿉니다.
  - *iotCredEndpoint*를 AWS IoT 자격 증명 엔드포인트로 바꿉니다.
2. AWS IoT Greengrass 코어 설치 관리자를 실행합니다. 다음과 같이 명령의 인수 값을 바꿉니다.

#### Note

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 C:\greengrass\v2 또는 D:\greengrass\v2와 같은 루트 폴더를 사용하여 Greengrass 구성 요소 경로를 260자 제한 미만으로 유지합니다.

- /greengrass/v2* 또는 *C:\greengrass\v2*: AWS IoT Greengrass 코어 소프트웨어를 설치하는 데 사용할 루트 폴더의 경로입니다.
- GreengrassInstaller*. AWS IoT Greengrass 코어 소프트웨어 설치 관리자를 압축 해제한 폴더의 경로입니다.
- region*. 리소스를 찾거나 생성할 AWS 리전 입니다.
- MyGreengrassCore*. Greengrass 코어 디바이스에 대한 AWS IoT 사물의 이름입니다. 사물이 존재하지 않는 경우 설치 관리자가 이를 생성합니다. 설치 관리자는 인증서를 다운로드하여 AWS IoT 사물로 인증합니다. 자세한 내용은 [에 대한 디바이스 인증 및 권한 부여 AWS IoT Greengrass](#) 단원을 참조하십시오.

#### Note

사물 이름에는 콜론(:) 문자를 포함할 수 없습니다.

- MyGreengrassCoreGroup*. Greengrass 코어 디바이스의 AWS IoT 사물 그룹 이름입니다. 사물 그룹이 존재하지 않는 경우 설치 관리자가 이를 생성하고 사물을 추가합니다. 사물 그룹이 존재하고 활성 배포가 있는 경우 코어 디바이스는 배포에서 지정한 소프트웨어를 다운로드하고 실행합니다.

**Note**

사물 그룹 이름에는 콜론(:) 문자를 포함할 수 없습니다.

- f. ***GreengrassV2IoTThingPolicy***. Greengrass 코어 디바이스가 AWS IoT 및와 통신할 수 있도록 허용하는 AWS IoT 정책의 이름입니다 AWS IoT Greengrass. 정책이 없는 경우 AWS IoT 설치 관리자는이 이름으로 허용 AWS IoT 정책을 생성합니다. 사용 사례에 대해 이 정책의 권한을 제한할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.
- g. ***GreengrassV2TokenExchangeRole***. Greengrass 코어 디바이스가 임시 AWS 자격 증명을 가져오도록 허용하는 IAM 역할의 이름입니다. 역할이 없는 경우 설치 관리자는 이를 생성하고 ***GreengrassV2TokenExchangeRoleAccess***라는 정책을 생성 및 연결합니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.
- h. ***GreengrassCoreTokenExchangeRoleAlias***. Greengrass 코어 디바이스가 나중에 임시 자격 증명을 가져올 수 있도록 허용하는 IAM 역할의 별칭입니다. 역할 별칭이 없는 경우 설치 관리자에서 이를 생성하고 지정한 IAM 역할로 가리킵니다. 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

## 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 \
--init-config ./GreengrassInstaller/config.yaml \
--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 AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설정--setup-system-service true하도록 지정해야 합니다.

설치 관리자가 성공하면 다음 메시지가 인쇄됩니다.

- --provision을 지정하면 리소스가 성공적으로 구성된 경우 설치 관리자에서 Successfully configured Nucleus with provisioned resource details가 인쇄됩니다.
- --deploy-dev-tools를 지정하면 배포가 성공적으로 생성된 경우 설치 관리자에서 Configured Nucleus to deploy aws.greengrass.Cli component가 인쇄됩니다.

- `--setup-system-service true`를 지정하면 소프트웨어가 설정되고 서비스로 실행되면 설치 관리자에서 `Successfully set up Nucleus as a system service`가 인쇄됩니다.
  - `--setup-system-service true`를 지정하지 않으면 소프트웨어가 실행되면 설치 관리자에서 `Launched Nucleus successfully`가 인쇄됩니다.
3. [Greengrass nucleus](#) v2.0.4 이상을 설치한 경우 이 단계를 건너뛰니다. 최신 버전의 소프트웨어를 다운로드한 경우 v2.0.4 이상을 설치한 것입니다.

다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어 루트 폴더에 필요한 파일 권한을 설정합니다. `/greengrass/v2`를 설치 명령에서 지정한 루트 폴더로 바꾸고 `/greengrass`를 루트 폴더의 상위 폴더로 바꿉니다.

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 설치한 경우 설치 관리자가 소프트웨어를 실행합니다. 그렇지 않으면 소프트웨어를 수동으로 실행해야 합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

#### Note

기본적으로 설치 관리자가 생성하는 IAM 역할은 S3 버킷의 구성 요소 아티팩트에 대한 액세스를 허용하지 않습니다. Amazon S3에서 아티팩트를 정의하는 사용자 지정 구성 요소를 배포하려면 코어 디바이스가 구성 요소 아티팩트를 검색할 수 있도록 역할에 권한을 추가해야 합니다. 자세한 내용은 [구성 요소 아티팩트에 대한 S3 버킷 액세스 허용](#) 단원을 참조하십시오. 구성 요소 아티팩트에 대한 S3 버킷이 아직 없는 경우 버킷을 생성한 후 권한을 추가할 수 있습니다.

소프트웨어를 구성하고 사용하는 방법에 대한 자세한 내용은 다음을 AWS IoT Greengrass참조하세요.

- [AWS IoT Greengrass 코어 소프트웨어 구성](#)
- [AWS IoT Greengrass 구성 요소 개발](#)
- [디바이스에 AWS IoT Greengrass 구성 요소 배포](#)
- [Greengrass 명령줄 인터페이스](#)

## FIPS 규정 준수 자사 구성 요소

|                                      |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                      |                                                                                                                                                                                                                                                                                                                                                                           |
| aws.greengrass.Nucleus               | data.iot-fips. <i>us-east-1</i> .amazonaws.com                                                                                                                                                                                                                                                                                                                            |
|                                      | greengrass-fips. <i>us-east-1</i> .amazonaws.com                                                                                                                                                                                                                                                                                                                          |
|                                      | data.credentials.iot-fips. <i>us-east-1</i> .amazonaws.com                                                                                                                                                                                                                                                                                                                |
| aws.greengrass.TokenExchangeService  | data.credentials.iot-fips. <i>us-east-1</i> .amazonaws.com                                                                                                                                                                                                                                                                                                                |
| aws.greengrass.Cli                   |                                                                                                                                                                                                                                                                                                                                                                           |
| aws.greengrass.StreamManager         | <ul style="list-style-type: none"> <li>• kinesis-fips. <i>us-east-1</i> .amazonaws.com</li> <li>• data.iotsitewise-fips. <i>us-east-1</i> .amazonaws.com</li> <li>• s3-fips. <i>us-east-1</i> .amazonaws.com</li> </ul> <div>  Note<br/>스트림 관리자가 AWS IoT 분석 FIPS 엔드포인트를 지원하지 않음 </div> |
| aws.greengrass.LogManager            | logs-fips. <i>us-east-1</i> .amazonaws.com                                                                                                                                                                                                                                                                                                                                |
| aws.greengrass.crypto.Pkcs11Provider |                                                                                                                                                                                                                                                                                                                                                                           |
| aws.greengrass.ShadowManager         |                                                                                                                                                                                                                                                                                                                                                                           |



|                                         |                                             |
|-----------------------------------------|---------------------------------------------|
|                                         |                                             |
| aws.greengrass.DockerApplicationManager | ecr-fips.us-east-1.amazonaws.com            |
| aws.greengrass.SecretManager            | secretsmanager-fips.us-east-1.amazonaws.com |
| aws.greengrass.telemetry.NucleusEmitter |                                             |
| aws.greengrass.clientdevices.IPDetector |                                             |
| aws.greengrass.DiskSpooler              |                                             |

## AWS IoT Greengrass의 복원성

AWS 글로벌 인프라는 Amazon Web Services Regions 및 가용 영역을 중심으로 구축되어 있습니다. 각 AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 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 인프라](#)를 참조하세요.

AWS 글로벌 인프라뿐만 아니라 AWS IoT Greengrass도 데이터 복원력과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

- 로컬 파일 시스템 및 CloudWatch Logs에 로그를 기록하도록 Greengrass 코어 디바이스를 구성할 수 있습니다. 코어 디바이스에서는 연결이 끊어지는 경우 파일 시스템에 계속 메시지를 로깅할 수 있습니다. 다시 연결되면 로그 메시지를 CloudWatch Logs에 씁니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 섹션을 참조하세요.
- 배포 중 코어 디바이스의 전원이 꺼지면 AWS IoT Greengrass 코어 소프트웨어가 다시 시작된 후 배포를 재개합니다.
- 코어 디바이스의 연결이 끊어지는 경우 Greengrass 클라이언트 디바이스에서는 로컬 네트워크를 통해 계속 통신할 수 있습니다.

- [스트림 관리자](#) 스트림을 읽고 로컬 스토리지 대상으로 데이터를 보내는 Greengrass 구성 요소를 작성할 수 있습니다.

## 의 인프라 보안 AWS IoT Greengrass

관리형 서비스인 [Amazon Web Services: 보안 프로세스 개요](#) 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 AWS IoT Greengrass 보호됩니다.

AWS 에서 게시한 API 호출을 사용하여 네트워크를 AWS IoT Greengrass 통해 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.

환경에서 AWS IoT Greengrass 디바이스는 X.509 인증서와 암호화 키를 사용하여 연결하고 인증합니다. 자세한 내용은 [the section called “장치 인증 및 권한 부여”](#) 단원을 참조하십시오.

## 의 구성 및 취약성 분석 AWS IoT Greengrass

IoT 환경은 다양한 기능을 수행하고 장기적으로 사용되며 지리적으로 분산된 다수의 장치로 구성될 수 있습니다. 이러한 특성으로 인해 장치 설정이 복잡해지고 오류가 발생하기 쉬워집니다. 장치가 컴퓨팅 파워, 메모리 및 스토리지 기능에서 제한되는 경우가 있으므로 장치 자체에서 암호화 및 다른 보안 형태의 사용이 제한됩니다. 또한 장치에서 알려진 취약성이 있는 소프트웨어를 사용하는 경우도 있습니다. 이러한 요소로 인해 IoT 장치가 해커의 매력적인 대상이 되며, 장치를 지속적으로 보호하기 어렵게 됩니다.

AWS IoT Device Defender 는 보안 문제와 모범 사례와의 편차를 식별하는 도구를 제공하여 이러한 문제를 해결합니다. AWS IoT Device Defender 를 사용하여 연결된 디바이스를 분석, 감사 및 모니터링하여 비정상적인 동작을 감지하고 보안 위험을 완화할 수 있습니다. AWS IoT Device Defender 는 디바이스를 감사하여 보안 모범 사례를 준수하고 디바이스에서 비정상적인 동작을 감지할 수 있도록 합니다. 따라서 장치 전반에 걸쳐 일관된 보안 정책을 시행하고 장치가 손상된 경우 신속하게 대응할 수 있습니다. 자세한 정보는 다음 주제를 참조하세요.

- [Device Defender 구성 요소](#)

- AWS IoT Core 개발자 안내서의 [AWS IoT Device Defender](#)

AWS IoT Greengrass 환경에서는 다음 고려 사항에 유의해야 합니다.

- 물리적 장치, 장치의 파일 시스템 및 로컬 네트워크를 보호하는 것은 사용자의 책임입니다.
- AWS IoT Greengrass 는 Greengrass 컨테이너에서 실행되는지 여부에 관계없이 사용자 정의 Greengrass 구성 요소에 대해 네트워크 격리를 적용하지 않습니다. 따라서 Greengrass 구성 요소는 네트워크를 통해 시스템 또는 외부에서 실행 중인 다른 프로세스와 통신할 수 있습니다.

## AWS IoT Greengrass V2의 코드 무결성

AWS IoT Greengrass에서는 AWS IoT Greengrass 코어 소프트웨어가 실행되는 디바이스로 AWS 클라우드의 소프트웨어 구성 요소가 배포됩니다. 이러한 소프트웨어 구성 요소에는 [AWS에서 제공하는 구성 요소](#)와 AWS 계정에 업로드하는 [사용자 지정 구성 요소](#)가 포함되어 있습니다. 모든 구성 요소는 레시피로 구성되어 있습니다. 레시피에서는 구성 요소의 메타데이터와 아티팩트(컴파일된 코드 및 정적 리소스와 같은 구성 요소 바이너리) 수가 정의됩니다. 구성 요소 아티팩트는 Amazon S3에 저장됩니다.

Greengrass 구성 요소를 개발하고 배포할 때 AWS 계정 및 디바이스에서 구성 요소 아티팩트로 작업하는 다음과 같은 기본 단계를 따릅니다.

1. 아티팩트를 생성하고 S3 버킷에 업로드합니다.
2. 각 아티팩트의 [암호화 해시](#)가 계산되는 구성 요소를 AWS IoT Greengrass 서비스의 레시피 및 아티팩트에서 생성합니다.
3. 각 아티팩트의 무결성이 다운로드되고 확인되는 Greengrass 코어 디바이스에 구성 요소를 배포합니다.

AWS에서는 Greengrass 코어 디바이스에 구성 요소를 배포하는 경우를 포함하여 S3 버킷에 아티팩트를 업로드한 후 아티팩트의 무결성 유지 관리를 담당합니다. S3 버킷에 아티팩트를 업로드하기 전에 소프트웨어 아티팩트를 보호할 책임은 사용자에게 있습니다. 구성 요소 아티팩트를 업로드하는 S3 버킷을 포함하여 AWS 계정의 리소스에 대한 액세스 권한을 보호할 책임도 사용자에게 있습니다.

### Note

Amazon S3에서는 S3 버킷 AWS 계정의 구성 요소 아티팩트가 변경되지 않도록 보호하는데 사용할 수 있는 S3 Object Lock이라는 특성이 제공됩니다. S3 Object Lock을 사용하여 구

성 요소 아티팩트 삭제 또는 덮어쓰기를 방지할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [Amazon S3 Object Lock](#)을 참조하세요.

AWS에서 퍼블릭 구성 요소를 게시할 때와 사용자가 사용자 지정 구성 요소를 업로드할 때 AWS IoT Greengrass에서는 각 구성 요소 아티팩트에 대한 암호화 다이제스트가 계산됩니다. AWS IoT Greengrass에서는 해당 다이제스트 계산에 사용되는 각 아티팩트의 다이제스트와 해시 알고리즘이 포함되도록 구성 요소 레시피가 업데이트됩니다. 아티팩트가 AWS 클라우드에서나 다운로드 중에 변경되면 파일 다이제스트가 AWS IoT Greengrass를 통해 구성 요소 레시피에 저장되는 다이제스트와 일치하지 않기 때문에 이 다이제스트를 통해 아티팩트의 무결성이 보장됩니다. 자세한 내용은 [구성 요소 레시피 참조의 아티팩트](#)를 참조하세요.

코어 디바이스에 구성 요소를 배포하면 AWS IoT Greengrass 코어 소프트웨어에서는 구성 요소 레시피와 레시피를 통해 정의되는 각 구성 요소 아티팩트가 다운로드됩니다. AWS IoT Greengrass 코어 소프트웨어에서는 다운로드된 각 아티팩트 파일의 다이제스트가 계산되고 레시피에 있는 해당 아티팩트의 다이제스트와 비교됩니다. 다이제스트가 일치하지 않으면 배포에 실패하고, AWS IoT Greengrass 코어 소프트웨어에서는 디바이스의 파일 시스템에서 다운로드된 아티팩트가 삭제됩니다. 코어 디바이스와 간 연결과 AWS IoT Greengrass의 보호 방식에 대한 자세한 내용은 [전송 중 암호화](#) 단원을 참조하세요.

코어 디바이스의 파일 시스템에서 구성 요소 아티팩트 파일을 보호할 책임은 사용자에게 있습니다. AWS IoT Greengrass 코어 소프트웨어에서는 Greengrass 루트 폴더의 packages 폴더에 아티팩트가 저장됩니다. AWS IoT Device Defender를 사용하여 코어 디바이스를 분석하고, 감사하고, 모니터링할 수 있습니다. 자세한 내용은 [의 구성 및 취약성 분석 AWS IoT Greengrass](#) 섹션을 참조하세요.

## AWS IoT Greengrass 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 생성하여 VPC와 AWS IoT Greengrass 컨트롤 플레인 간에 프라이빗 연결을 설정할 수 있습니다. 이 엔드포인트를 사용하여 AWS IoT Greengrass 서비스의 구성 요소, 배포 및 코어 디바이스를 관리할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이 [AWS PrivateLink](#), NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 비공개로 AWS IoT Greengrass APIs에 액세스할 수 있는 기술로 구동됩니다. VPC의 인스턴스는 AWS IoT Greengrass APIs. VPC와 간의 트래픽 AWS IoT Greengrass 은 Amazon 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 [Elastic Network Interfaces](#)로 표현됩니다.

자세한 내용은 Amazon VPC 사용 설명서의 [인터페이스 VPC 엔드포인트\(AWS PrivateLink\)](#)를 참조하세요.

## 주제

- [AWS IoT Greengrass VPC 엔드포인트에 대한 고려 사항](#)
- [AWS IoT Greengrass 컨트롤 플레인 작업을 위한 인터페이스 VPC 엔드포인트 생성](#)
- [에 대한 VPC 엔드포인트 정책 생성 AWS IoT Greengrass](#)
- [VPC에서 AWS IoT Greengrass 코어 디바이스 작동](#)

## AWS IoT Greengrass VPC 엔드포인트에 대한 고려 사항

에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 [인터페이스 엔드포인트 속성 및 제한 사항](#)을 AWS IoT Greengrass 검토하세요. 추가적으로, 다음 사항을 고려하세요.

- AWS IoT Greengrass 는 VPC에서 모든 컨트롤 플레인 API 작업을 호출할 수 있도록 지원합니다. 컨트롤 플레인은 [CreateDeployment](#) 및 [ListEffectiveDeployments](#)와 같은 작업을 포함합니다. 컨트롤 플레인은 데이터 플레인 작업인 [ResolveComponentCandidates](#) 및 [Discover](#)와 같은 작업을 포함하지 않습니다.
- 에 대한 VPC 엔드포인트 AWS IoT Greengrass 는 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 ()를 사용하여 AWS IoT Greengrass 컨트롤 플레인의 VPC 엔드포인트를 생성할 수 있습니다AWS CLI. 자세한 내용은 Amazon VPC 사용 설명서의 [인터페이스 엔드포인트 생성](#)을 참조하세요.

다음 서비스 이름을 AWS IoT Greengrass 사용하여 VPC 엔드포인트를 생성합니다.

- `com.amazonaws.region.greengrass`

엔드포인트에 대해 프라이빗 DNS를 활성화하는 경우 리전의 기본 DNS 이름, 예를 들어 `aws-iot-greengrass.us-east-1.amazonaws.com`. 프라이빗 DNS는 기본적으로 활성화되어 있습니다.

자세한 내용은 Amazon VPC 사용 설명서의 [인터페이스 엔드포인트를 통해 서비스 액세스](#)를 참조하세요.

## 에 대한 VPC 엔드포인트 정책 생성 AWS IoT Greengrass

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 제어](#)를 참조하세요.

Example: AWS IoT Greengrass 작업에 대한 VPC 엔드포인트 정책

다음은에 대한 엔드포인트 정책의 예입니다 AWS IoT Greengrass. 엔드포인트에 연결되면이 정책은 모든 리소스의 모든 보안 주체에 대해 나열된 AWS IoT Greengrass 작업에 대한 액세스 권한을 부여합니다.

```
{
 "Statement": [
 {
 "Principal": "*",
 "Effect": "Allow",
 "Action": [
 "greengrass:CreateDeployment",
 "greengrass:ListEffectiveDeployments"
],
 "Resource": "*"
 }
]
}
```

## VPC에서 AWS IoT Greengrass 코어 디바이스 작동

퍼블릭 인터넷 액세스 없이도 Greengrass 코어 디바이스를 작동하고 VPC에서 배포를 수행할 수 있습니다. 최소한 다음 VPC 엔드포인트를 해당 DNS 별칭으로 설정해야 합니다. VPC 엔드포인트 생성 및 사용 방법에 대한 자세한 내용은 Amazon VPC 사용 설명서에서 [VPC 엔드포인트 생성](#)을 참조하세요.

**Note**

DNS 레코드를 자동으로 생성하기 위한 VPC 기능은 AWS IoT data 및 AWS IoT 자격 증명에 대해 비활성화됩니다. 이러한 엔드포인트에 연결하려면 프라이빗 DNS 레코드를 수동으로 생성해야 합니다. 자세한 내용은 [Private DNS for interface endpoints](#)를 참조하세요. AWS IoT Core VPC 제한 사항에 대한 자세한 내용은 [VPC 엔드포인트의 제한을](#) 참조하세요.

## 사전 조건

- 수동 프로비저닝 단계를 사용하여 AWS IoT Greengrass 코어 소프트웨어를 설치해야 합니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

## 제한 사항

- 중국 리전 및 AWS GovCloud (US) Regions에서는 VPC에서 Greengrass 코어 디바이스를 작동할 수 없습니다.
- AWS IoT data 및 AWS IoT 자격 증명 공급자 VPC 엔드포인트의 제한 사항에 대한 자세한 내용은 [제한을 참조하세요](#).

## Greengrass 코어 디바이스가 VPC에서 작동하도록 설정

1. 의 AWS IoT 엔드포인트를 가져 AWS 계정와 나중에 사용할 수 있도록 저장합니다. 디바이스에서는 이러한 엔드포인트를 사용하여 AWS IoT에 연결합니다. 해결 방법:
  - a. 의 AWS IoT 데이터 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

- b. 의 AWS IoT 자격 증명 엔드포인트를 가져옵니다 AWS 계정.

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

요청에 성공하는 경우 응답은 다음 예제와 유사합니다.

```
{
 "endpointAddress": "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
}
```

## 2. AWS IoT data 및 AWS IoT 자격 증명 엔드포인트에 대한 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 영역에 대한 VPC 엔드포인트를 생성하는 경우 해당 리전의 AWS IoT Core 데이터 영역 API 엔드포인트를 선택합니다. 엔드포인트 형식은 `com.amazonaws.region.iot.data`이(가) 될 것입니다.

AWS IoT Core 자격 증명 공급자에 대한 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 영역에 대한 서비스 이름은 형식입니다 `cn.com.amazonaws.region.iot.data`. 중국 리전에서는 AWS IoT Core 자격 증명 공급자에 대한 VPC 엔드포인트 생성이 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 영역이나 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 생성한 후 엔드포인트의 세부 정보 탭에 DNS 이름 목록이 표시됩니다. 이러한 DNS 이름 중 하나를 사용하여 [프라이빗 호스팅 영역을 구성](#)할 수 있습니다.
  4. Amazon S3 엔드포인트를 생성합니다. 자세한 내용은 [Create a VPC endpoint for Amazon S3](#)를 참조하세요.
  5. [AWS제공 Greengrass 구성 요소](#)를 사용하는 경우 추가 엔드포인트 및 구성이 필요할 수 있습니다. 엔드포인트 요구 사항을 보려면 AWS제공 구성 요소의 목록에서 구성 요소를 선택하고 요구 사항 섹션을 살펴보세요. 예를 들어 [로그 관리자 구성 요소 요구 사항](#)에서는 이 구성 요소가 엔드포인트 logs.**region**.amazonaws.com에 대한 아웃바운드 요청을 수행할 수 있어야 한다고 조언합니다.

자체 구성 요소를 사용하는 경우 종속성을 검토하고 추가 테스트를 수행하여 추가 엔드포인트가 필요한지 확인해야 할 수 있습니다.

6. Greengrass nucleus 구성에서 greengrassDataPlaneEndpoint는 **iotdata**로 설정해야 합니다. 자세한 내용은 [Greengrass nucleus 구성](#)을 참조하세요.
7. 사용자가 us-east-1 리전에 있는 경우 Greengrass nucleus 구성에서 구성 파라미터 s3EndpointType을 **REGIONAL**로 설정합니다. 이 기능은 Greengrass nucleus 버전 2.11.3 이상에서 사용할 수 있습니다.

Example예: 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 표에서는 해당 사용자 지정 프라이빗 DNS 별칭에 대한 정보를 제공합니다.

| 서비스           | VPC 엔드포인트 서비스 이름                              | VPC 엔드포인트 유형 | 사용자 지정 프라이빗 DNS 별칭                           | 참고                                                                                                      |
|---------------|-----------------------------------------------|--------------|----------------------------------------------|---------------------------------------------------------------------------------------------------------|
| AWS IoT data  | com.amazonaws. <i>region</i> .iot.data        | 인터페이스        | <i>prefix</i> -ats.iot. <i>region</i> .s.com | 프라이빗 DNS 레코드는 계정의 AWS IoT data 엔드포인트와 일치해야 합니다aws iot describe-endpoint -- endpoint-type iot:Data-ATS . |
| AWS IoT 자격 증명 | com.amazonaws. <i>region</i> .iot.credentials | 인터페이스        | <i>prefix</i> .credentials.iot.s.com         | 프라이빗 DNS 레코드는 계정의 AWS IoT 자격 증명 엔드포인트와 일치해야 합니다aws iot                                                  |

| 서비스       | VPC 엔드포인트 서비스 이름                 | VPC 엔드포인트 유형 | 사용자 지정 프라이빗 DNS 별칭 | 참고                                                          |
|-----------|----------------------------------|--------------|--------------------|-------------------------------------------------------------|
|           |                                  |              |                    | describe-endpoint -- endpoint-type iot:CredentialProvider . |
| Amazon S3 | com.amazonaws. <i>region</i> .s3 | 인터페이스        |                    | DNS 레코드가 자동으로 생성됩니다.                                        |

## 에 대한 보안 모범 사례 AWS IoT Greengrass

이 주제에는에 대한 보안 모범 사례가 포함되어 있습니다 AWS IoT Greengrass.

### 가능한 최소 권한 부여

구성 요소를 권한 없는 사용자로 실행하여 구성 요소에 대한 최소 권한 원칙을 따릅니다. 꼭 필요한 경우가 아니라면 구성 요소를 루트로 실행해서는 안 됩니다.

IAM 역할에서 최소한의 권한 세트를 사용합니다. IAM 정책의 Action 및 Resource 속성에 대한 \* 와 일드카드 사용을 제한합니다. 대신, 가능한 경우 한정된 작업과 리소스를 선언합니다. 최소 권한 및 기타 정책 모범 사례에 대한 자세한 내용은 [the section called “정책 모범 사례”](#)을 참조하세요.

최소 권한 모범 사례는 Greengrass 코어에 연결하는 AWS IoT 정책에도 적용됩니다.

### Greengrass 구성 요소에서 자격 증명을 하드 코딩하지 마세요.

사용자 정의 Greengrass 구성 요소에서 자격 증명을 하드 코딩하지 마세요. 자격 증명에 대한 보호를 강화하려면 다음을 수행하십시오.

- AWS 서비스와 상호 작용하려면 [Greengrass 코어 디바이스 서비스 역할](#)에서 특정 작업 및 리소스에 대한 권한을 정의합니다.
- [보안 암호 관리자 구성 요소](#)를 사용하여 자격 증명을 저장합니다. 또는 함수가 AWS SDK를 사용하는 경우 기본 자격 증명 공급자 체인의 자격 증명을 사용합니다.

## 민감한 정보를 기록하지 않음

자격 증명 및 기타 개인 식별 정보(PII)의 로깅을 방지해야 합니다. 코어 장치의 로컬 로그에 액세스하기 위해 루트 권한이 필요하고 CloudWatch Logs에 대한 액세스를 위해 IAM 권한이 필요한 경우에도 다음과 같은 보호 조치를 구현하는 것이 좋습니다.

- MQTT 주제 경로에는 민감한 정보를 사용하지 마십시오.
- AWS IoT Core 레지스트리의 장치(사물) 이름, 유형 및 속성에 민감한 정보를 사용하지 마십시오.
- 사용자 정의 Greengrass 구성 요소나 Lambda 함수에 민감한 정보를 기록하지 마세요.
- Greengrass 리소스의 이름과 ID에 민감한 정보를 사용하지 마십시오.
  - 코어 디바이스
  - 구성 요소
  - 배포
  - Loggers

## 장치의 시계를 동기화 상태로 유지

장치에서는 정확한 시간을 유지하는 것이 중요합니다. X.509 인증서에는 만료 날짜와 시간이 있습니다. 장치의 시계는 서버 인증서가 여전히 유효한지 확인하는 데 사용됩니다. 장치 시계는 시간이 지나 드리프트 상태가 되거나 배터리가 방전될 수 있습니다.

자세한 내용은 AWS IoT Core 개발자 안내서의 [장치 시계를 동기화된 상태로 유지](#) 모범 사례를 참조하십시오.

## Cipher Suite 권장 사항

Greengrass 기본값은 디바이스에서 사용할 수 있는 최신 TLS Cipher Suite를 선택합니다. 디바이스에서 레거시 암호 제품군 사용을 비활성화하는 것이 좋습니다. 예: CBC 암호 제품군.

자세한 내용은 [Java Cryptography Configuration](#)을 참조하세요.

다음 사항도 참조하세요.

- AWS IoT 개발자 안내서의 [AWS IoT Core내 보안 모범 사례](#)
- AWS 공식 블로그의 사물 인터넷에서 [산업용 IoT 솔루션에 대한 10가지 보안 황금률](#)

# AWS IoT Greengrass V2용 AWS IoT Device Tester 사용

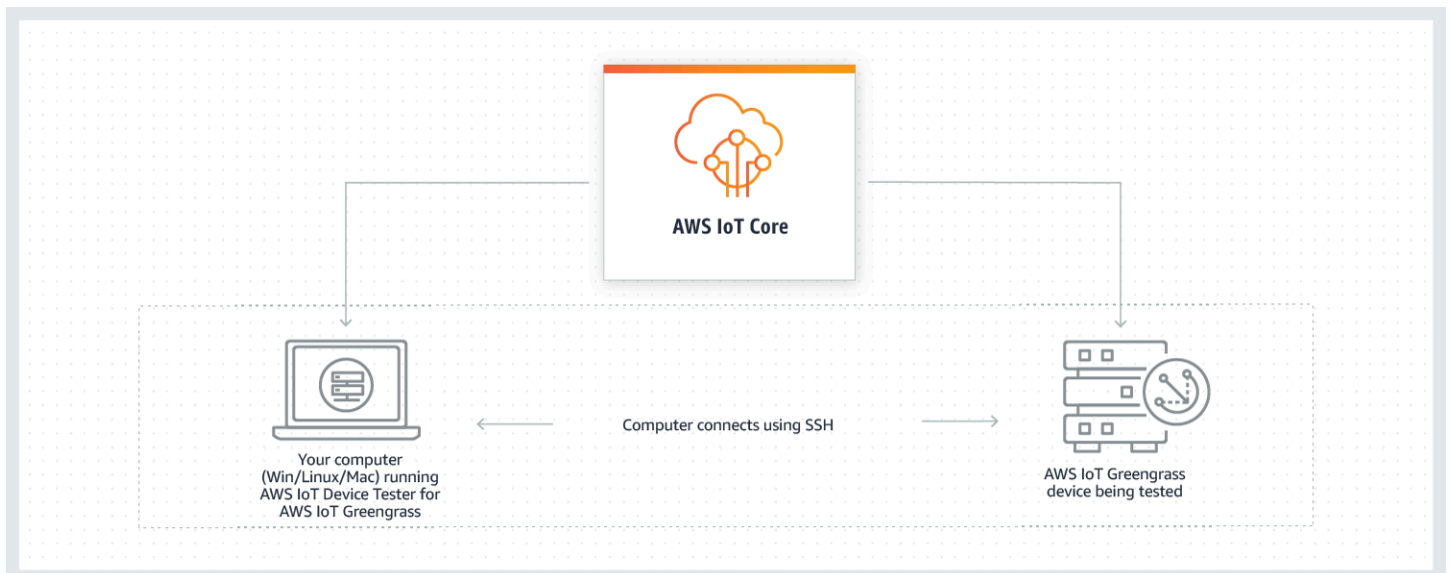
AWS IoT Device Tester(IDT)는 IoT 장치를 검증할 수 있는 다운로드 가능한 테스트 프레임워크입니다. AWS IoT Greengrass용 IDT를 사용하여 AWS IoT Greengrass 검증 제품군을 실행하고, 디바이스용 사용자 지정 테스트 제품군을 생성하고 실행할 수 있습니다.

AWS IoT Greengrass용 IDT는 테스트 대상 장치에 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 Greengrass V2용 AWS IoT Device Tester를 사용하면 AWS IoT Greengrass 코어 소프트웨어가 하드웨어에서 실행되고 AWS 클라우드와 통신할 수 있는지 확인할 수 있습니다. 또한 이 IDT는 AWS IoT Core와의 엔드 투 엔드 테스트를 수행합니다. 예를 들어 디바이스에서 구성 요소를 배포하고 업그레이드할 수 있는지 확인합니다.

AWS Partner Device Catalog에 하드웨어를 추가하려면 AWS IoT Greengrass 검증 제품군을 실행하여 AWS IoT에 제출할 수 있는 테스트 보고서를 생성합니다. 자세한 내용은 [AWS Device Qualification Program](#)을 참조하세요.



AWS IoT Greengrass V2용 IDT는 테스트 제품군 및 테스트 그룹의 개념을 사용하여 테스트를 구성합니다.

- 테스트 제품군은 장치가 AWS IoT Greengrass의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 구성 요소 배포와 같은 특정 특성과 관련된 개별 테스트 집합입니다.

자세한 내용은 [IDT를 사용하여 AWS IoT Greengrass 검증 제품군 실행](#) 섹션을 참조하세요.

## 사용자 지정 테스트 제품군

IDT v4.0.1부터 AWS IoT Greengrass V2용 IDT는 표준화된 구성 설정 및 결과 형식을 디바이스 및 디바이스 소프트웨어에 대한 사용자 지정 테스트 제품군을 개발할 수 있는 테스트 제품군 환경과 결합합니다. 사용자 지정 테스트를 자체 내부 검증을 위해 추가하거나 디바이스 검증을 위해 고객에게 제공할 수 있습니다.

테스트 작성자가 사용자 지정 테스트 제품군을 구성하는 방법에 따라 사용자 지정 테스트 제품군을 실행하는 데 필요한 설정 구성이 결정됩니다. 자세한 내용은 [IDT를 사용하여 자체 테스트 제품군 개발 및 실행](#) 섹션을 참조하세요.

## AWS IoT Device Tester for AWS IoT Greengrass V2 지원 버전

이 주제에서는 지원되는 IDT for AWS IoT Greengrass V2 버전을 나열합니다. 가장 좋은 방법은 대상 버전의 AWS IoT Greengrass V2를 지원하는 최신 버전의 IDT for AWS IoT Greengrass V2를 사용하는 것입니다. 의 새 릴리스에서는 새 버전의 IDT for AWS IoT Greengrass V2를 다운로드해야 할 AWS IoT Greengrass 수 있습니다. IDT for AWS IoT Greengrass V2가 사용 중인 버전과 호환되지 않는 경우 테스트 실행을 시작할 때 알림을 받게 AWS IoT Greengrass 됩니다.

소프트웨어를 다운로드하면 [AWS IoT Device Tester IDT 라이선스 계약](#)에 동의하는 것입니다.

### 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.

## AWS IoT Greengrass V2의 최신 IDT 버전

이 버전의 IDT for AWS IoT Greengrass V2를 여기에 나열된 AWS IoT Greengrass 버전과 함께 사용할 수 있습니다.

## 용 IDT v4.9.4 AWS IoT Greengrass

지원되는 AWS IoT Greengrass 버전:

- [Greengrass nucleus](#) v2.12.0, v2.11.0, v2.10.0 및 v2.9.5

IDT 소프트웨어 다운로드:

- [Linux](#)용 테스트 제품군 GGV2Q\_2.5.4가 있는 IDT v4.9.4
- [macOS](#)용 테스트 제품군 GGV2Q\_2.5.4가 있는 IDT v4.9.4
- [Windows](#)용 테스트 제품군 GGV2Q\_2.5.4가 있는 IDT v4.9.4

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.12.0, 2.11.0, 2.10.0 및 2.9.5를 실행하는 디바이스에 대해 디바이스 검증 및 검증을 활성화합니다.
- 스트림 관리자 및 기계 학습 테스트 그룹을 제거합니다.

추가 참고 사항

- 디바이스가 HSM을 사용하고 nucleus 2.10.x를 사용하는 경우 Greengrass nucleus 버전 2.11.0 이상으로 마이그레이션합니다.

테스트 제품군 버전:

GGV2Q\_2.5.4

- 2024년 5월 3일 릴리스

## 용 이전 IDT 버전 AWS IoT Greengrass

다음 이전 버전의 IDT for AWS IoT Greengrass V2도 지원됩니다.

### 용 IDT v4.9.3 AWS IoT Greengrass

지원되는 AWS IoT Greengrass 버전:

- [Greengrass nucleus](#) v2.12.0, v2.11.0, v2.10.0 및 v2.9.5

IDT 소프트웨어 다운로드:

- [Linux](#)용 테스트 제품군 GGV2Q\_2.5.3이 있는 IDT v4.9.3
- [macOS](#)용 테스트 제품군 GGV2Q\_2.5.3이 있는 IDT v4.9.3
- [Windows](#)용 테스트 제품군 GGV2Q\_2.5.3이 있는 IDT v4.9.3

릴리스 정보:

- Windows 호스트에서 Linux 디바이스를 테스트할 때 또는 그 반대의 경우 구성 요소 테스트의 문제를 해결합니다.



- component 테스트 그룹에서 localcomponent 테스트 사례를 제거합니다. 이 테스트 사례는 이제 검증에 필요하지 않습니다.

#### 추가 참고 사항

- 디바이스가 HSM을 사용하고 nucleus 2.10.x를 사용하는 경우 Greengrass nucleus 버전 2.11.0 이상으로 마이그레이션합니다.

#### 테스트 제품군 버전:

GGV2Q\_2.5.3

- 2024년 4월 5일 릴리스됨

## 지원되지 않는 AWS IoT Device Tester for AWS IoT Greengrass V2 버전

이 주제에서는 지원되지 않는 IDT for AWS IoT Greengrass V2 버전을 나열합니다. 지원되지 않는 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다. 자세한 내용은 [the section called “AWS IoT Device Tester용 AWS IoT Greengrass에 대한 지원 정책”](#) 단원을 참조하십시오.

### 용 IDT v4.9.2 AWS IoT Greengrass

#### 릴리스 정보:

- Java 8이 더는 사용되지 않아서 Lambda 테스트 제품군이 실패하는 문제를 해결합니다.

#### 테스트 제품군 버전:

GGV2Q\_2.5.2

- 2024년 3월 18일 릴리스됨

### 용 IDT v4.9.1 AWS IoT Greengrass

#### 릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.12.0, 2.11.0, 2.10.0 및 2.9.5가 실행되는 디바이스를 검증할 수 있습니다.
- 사소한 버그를 수정했습니다.

#### 테스트 제품군 버전:

GGV2Q\_2.5.1

- 2023년 10월 5일 릴리스됨

### 용 IDT v4.7.0 AWS IoT Greengrass

#### 지원되는 AWS IoT Greengrass 버전:

- [Greengrass nucleus](#) v2.11.0, v2.10.0 및 v2.9.5

**릴리스 정보:**

- AWS IoT Greengrass 코어 소프트웨어 버전 2.11.0, 2.10.0 및 2.9.5가 실행되는 디바이스를 검증할 수 있습니다.
- AWS Systems Manager Parameter Store에 IDT 사용자 데이터 값을 저장하고 자리 표시자 구문을 사용하여 구성으로 가져오는 지원을 추가합니다.
- 사소한 버그를 수정했습니다.

**테스트 제품군 버전:**

GGV2Q\_2.5.0

- 2022년 12월 13일 릴리스됨

**용 IDT v4.5.11 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, 2.6.0이 실행되는 디바이스를 검증할 수 있습니다.
- 코어 디바이스에서 PreInstalled Greengrass를 테스트하는 지원을 추가합니다.
- 사소한 버그를 수정했습니다.

**테스트 제품군 버전:**

GGV2Q\_2.4.1

- 2022년 10월 13일 릴리스됨

**용 IDT v4.5.8 AWS IoT Greengrass****릴리스 정보:**

- AWS IoT Greengrass 코어 소프트웨어 버전 2.7.0, 2.6.0 및 2.5.6이 실행되는 디바이스를 검증할 수 있습니다.
- 코어 디바이스에서 PreInstalled Greengrass로 테스트할 수 있습니다.
- 사소한 버그를 수정했습니다.

**테스트 제품군 버전:**

GGV2Q\_2.4.0

- 2022년 8월 12일 릴리스됨

**용 IDT v4.5.3 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 2.5.3이 실행되는 디바이스를 검증할 수 있습니다.

- ECR 기반 Docker 이미지가 사용되도록 DockerApplicationManager 테스트를 업데이트합니다.
- 사소한 버그를 수정했습니다.

테스트 제품군 버전:

GGV2Q\_2.3.1

- 2022년 4월 15일 릴리스됨

용 IDT v4.5.1 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 v2.5.3이 실행되는 디바이스를 검증할 수 있습니다.
- AWS IoT Greengrass 코어 소프트웨어에서 사용되는 프라이빗 키 및 인증서 저장에 하드웨어 보안 모듈(HSM)이 사용되는 리눅스 기반 디바이스 검증에 대한 지원을 추가합니다.
- 사용자 지정 테스트 제품군을 구성하기 위한 새로운 IDT 테스트 오케스트레이터를 구현합니다. 자세한 내용은 [IDT 테스트 오케스트레이터 구성](#) 단원을 참조하십시오.
- 추가 사소한 버그 수정 사항.

테스트 제품군 버전:

GGV2Q\_2.3.0

- 2022년 1월 11일 릴리스됨

용 IDT v4.4.1 AWS IoT Greengrass

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 v2.5.2가 실행되는 디바이스를 검증할 수 있습니다.
- 테스트 중인 디바이스가 AWS 리소스와 상호 작용하기 위해 수입하는 토큰 교환 역할로 사용자 정의 IAM 역할을 사용할 수 있는 지원을 추가합니다.

[userdata.json 파일](#)에서 IAM 역할을 지정할 수 있습니다. 사용자 지정 역할을 지정하면 IDT에서는 테스트 실행 중 기본 토큰 교환 역할이 생성되지 않고 해당 역할이 사용됩니다.

- 추가 사소한 버그 수정 사항.

테스트 제품군 버전:

GGV2Q\_2.2.1

- 2021년 12월 12일 릴리스됨

## 용 IDT v4.4.0 AWS IoT Greengrass

### 릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 v2.5.0이 실행되는 디바이스를 검증할 수 있습니다.
- Windows에서 AWS IoT Greengrass 코어 소프트웨어를 실행하는 디바이스의 검증 및 검증에 대한 지원을 추가합니다.
- Secure Shell(SSH) 디바이스 연결에 대한 퍼블릭 키 검증 사용을 지원합니다.
- 보안 모범 사례로 IDT 권한 IAM 정책을 개선합니다.
- 추가 사소한 버그 수정 사항.

### 테스트 제품군 버전:

GGV2Q\_2.1.0

- 2021년 11월 19일 릴리스됨

## 용 IDT v4.2.0 AWS IoT Greengrass

### 릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 v2.2.0 이상을 실행하는 디바이스에서 다음 기능의 검증에 대한 지원을 포함합니다.
- Docker - Amazon Elastic Container Registry(Amazon ECR)의 Docker 컨테이너 이미지를 디바이스에서 다운로드할 수 있는지 검증합니다.
- 기계 학습 - 디바이스의 기계 학습(ML) 추론 수행에 [딥 러닝 런타임](#) 또는 [TensorFlow Lite](#) ML 프레임워크가 사용될 수 있는지 검증합니다.
- 스트림 관리자 - 디바이스가 AWS IoT Greengrass 스트림 관리자를 다운로드, 설치 및 실행할 수 있는지 확인합니다.
- AWS IoT Greengrass 코어 소프트웨어 버전 v2.4.0, v2.3.0, v2.2.0 및 v2.1.0이 실행되는 디바이스를 검증할 수 있습니다.
- 각 테스트 사례에 대한 테스트 로그를 `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` 디렉터리 내 별도의 `<test-case-id>` 폴더에 그룹화합니다.
- 추가 사소한 버그 수정 사항.

### 테스트 제품군 버전:

GGV2Q\_2.0.1

- 2021년 8월 31일 릴리스됨

## 용 IDT v4.1.0 AWS IoT Greengrass

### 릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 v2.3.0, v2.2.0, v2.1.0 및 v2.0.5가 실행되는 디바이스를 검증할 수 있습니다.
- GreengrassNucleusVersion 및 GreengrassCLIVersion 속성 지정에 대한 요구 사항을 제거하여 userdata.json 구성을 개선합니다.
- AWS IoT Greengrass 코어 소프트웨어 v2.1.0 이상 버전에 대한 Lambda 및 MQTT 기능 검증 지원이 포함되어 있습니다. 이제 IDT for AWS IoT Greengrass V2를 사용하여 코어 디바이스가 Lambda 함수를 실행할 수 있고 디바이스가 AWS IoT Core MQTT 주제를 게시하고 구독할 수 있는지 확인할 수 있습니다.
- 로깅 기능을 개선합니다.
- 추가 사소한 버그 수정 사항.

### 테스트 제품군 버전:

GGV2Q\_1.1.1

- 2021년 6월 18일 릴리스됨

## 용 IDT v4.0.2 AWS IoT Greengrass

### 릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 v2.1.0이 실행되는 디바이스를 검증할 수 있습니다.
- AWS IoT Greengrass 코어 소프트웨어 v2.1.0 이상 버전에 대한 Lambda 및 MQTT 기능 검증에 대한 지원을 추가합니다. 이제 IDT for AWS IoT Greengrass V2를 사용하여 코어 디바이스가 Lambda 함수를 실행할 수 있고 디바이스가 AWS IoT Core MQTT 주제를 게시하고 구독할 수 있는지 확인할 수 있습니다.
- 로깅 기능을 개선합니다.
- 추가 사소한 버그 수정 사항.

### 테스트 제품군 버전:

GGV2Q\_1.1.1

- 2021년 5월 5일 릴리스됨

## 용 IDT v4.0.1 AWS IoT Greengrass

### 릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2 소프트웨어가 실행되는 디바이스를 검증할 수 있습니다.

- 옹를 사용하여 사용자 지정 테스트 제품군을 개발하고 실행할 AWS IoT Device Tester 수 있습니다 AWS IoT Greengrass. 자세한 내용은 [IDT를 사용하여 자체 테스트 제품군 개발 및 실행](#) 단원을 참조하십시오.
- macOS 및 Windows용 코드 서명 IDT 애플리케이션을 제공합니다. macOS에서는 IDT에 대한 보안 예외를 부여해야 할 수도 있습니다. 자세한 내용은 [macOS에서의 보안 예외](#) 단원을 참조하십시오.

테스트 제품군 버전:

GGV2Q\_1.0.0

- 2020년 12월 22일 릴리스됨
- features 배열에서 해당 value를 yes로 설정하지 않으면 테스트 제품군에서는 검증에 필요한 테스트만 실행됩니다.

## AWS IoT Greengrass V2용 IDT 다운로드

이 주제에서는 AWS IoT Greengrass V2용 AWS IoT Device Tester를 다운로드하는 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 IDT 라이선스 계약](#)에 동의하는 것입니다.

### 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 주제에서는 AWS IoT Greengrass V2용 IDT의 지원되는 버전을 나열합니다. 가장 좋은 방법은 AWS IoT Greengrass V2의 대상 버전을 지원하는 AWS IoT Greengrass V2용 IDT의 최신 버전을 사용하는

것입니다. AWS IoT Greengrass의 새 릴리스를 사용하려면 새 버전의 AWS IoT Greengrass V2용 IDT를 다운로드해야 합니다. AWS IoT Greengrass V2용 IDT가 사용 중인 AWS IoT Greengrass 버전과 호환되지 않을 경우 테스트 실행을 시작할 때 알림이 제공됩니다.

## AWS IoT Greengrass용 IDT v4.9.4

지원되는 AWS IoT Greengrass 버전

- [Greengrass nucleus](#) v2.12.0, v2.11.0, v2.10.0 및 v2.9.5

IDT 소프트웨어 다운로드:

- [Linux](#)용 테스트 제품군 GGV2Q\_2.5.4가 있는 IDT v4.9.4
- [macOS](#)용 테스트 제품군 GGV2Q\_2.5.4가 있는 IDT v4.9.4
- [Windows](#)용 테스트 제품군 GGV2Q\_2.5.4가 있는 IDT v4.9.4

릴리스 정보:

- AWS IoT Greengrass 코어 소프트웨어 버전 2.12.0, 2.11.0, 2.10.0 및 2.9.5를 실행하는 디바이스에 대해 디바이스 검증이 활성화됩니다.
- 스트림 관리자 및 기계 학습 테스트 그룹을 제거합니다.

추가 참고 사항

- 디바이스가 HSM을 사용하고 nucleus 2.10.x를 사용하는 경우 Greengrass nucleus 버전 2.11.0 이상으로 마이그레이션합니다.

테스트 제품군 버전:

GGV2Q\_2.5.4

- 2024년 5월 3일 릴리스

## 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 iot-device-tester를 서비스 이름으로 사용합니다.

## API 요청

HostOS - 호스트 시스템의 운영 체제입니다. 다음 옵션 중 하나를 선택합니다.

- mac
- linux
- windows

TestSuiteType - 테스트 제품군의 유형입니다. 다음 옵션을 선택합니다.

GGV2 - AWS IoT Greengrass V2용 IDT

ProductVersion

(선택 사항) Greengrass nucleus의 버전. 이 서비스는 해당 버전의 Greengrass nucleus와 호환되는 최신 버전의 IDT를 반환합니다. 이 옵션을 지정하지 않으면 서비스가 최신 버전의 IDT를 반환합니다.

## API 응답

API 응답은 다음 형식을 갖습니다. DownloadURL에는 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
 }
}
```

## 예시

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.

Example예제: 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 [jq](#)를 사용하여 응답에서 다운로드 URL을 파싱합니다.



**⚠ Warning**

aws-sigv4 플래그에는 curl GET 요청의 쿼리 파라미터가 HostOs/ProductVersion/TestSuiteType 또는 HostOs/TestSuiteType의 순서로 되어 있어야 합니다. 이 순서를 준수하지 않으면 API Gateway에서 표준 문자열에 대해 일치하지 않는 서명을 가져오는 오류가 발생합니다.

선택적 파라미터 ProductVersion이 포함된 경우 [AWS IoT Greengrass V2용 AWS IoT Device Tester의 지원되는 버전](#)에 있는 지원되는 제품 버전을 사용해야 합니다.

- **us-west-2**를 해당 AWS 리전으로 바꿉니다. 리전 코드의 목록은 [리전 엔드포인트](#)를 참조하세요.
- **linux**를 호스트 시스템의 운영 체제로 바꿉니다.
- **2.5.3**을 AWS IoT Greengrass nucleus의 버전으로 바꿉니다.

```
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
```

Example예제: 이전 버전의 cURL을 사용하여 다운로드(Mac 및 Linux)

사용자가 서명 및 계산한 AWS 서명과 함께 다음 cURL 명령을 사용할 수 있습니다. AWS 서명을 서명 및 계산하는 방법에 대한 자세한 내용은 [AWS API 요청 서명](#)을 참조하세요.

- **linux**를 호스트 시스템의 운영 체제로 바꿉니다.
- **Timestamp**를 날짜 및 시간(예: **20220210T004606Z**)으로 바꿉니다.
- **Date**를 날짜(예: **20220210**)로 바꿉니다.
- **AWSRegion**을 해당 AWS 리전으로 바꿉니다. 리전 코드의 목록은 [리전 엔드포인트](#)를 참조하세요.
- **AAWSSignature**를 사용자가 생성한 [AWS 서명](#)으로 바꿉니다.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
Host0s=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'
```

Example예제: Python 스크립트를 사용하여 다운로드

이 예제에서는 Python [요청](#) 라이브러리를 사용합니다. 이 예제는 Python 예제에서 AWS 일반 참조의 [AWS API 요청 서명](#)으로 수정되었습니다.

- **us-west-2**를 해당 리전으로 바꿉니다. 리전 코드의 목록은 [리전 엔드포인트](#)를 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 = 'Host0s=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 for AWS IoT Greengrass V2를 사용하여 AWS IoT Greengrass 코어 소프트웨어가 하드웨어에서 실행되고와 통신할 수 있는지 확인할 수 있습니다 AWS 클라우드. 또한를 사용하여 end-to-end 테스트를 수행합니다 AWS IoT Core. 예를 들어 디바이스에서 구성 요소를 배포하고 업그레이드할 수 있는지 확인합니다.

디바이스 테스트 외에도 IDT for AWS IoT Greengrass V2는 검증 프로세스를 용이하게 AWS 계정 하기 위해에 리소스(예: AWS IoT 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 실행할 때 다음 단계를 수행합니다.

1. 디바이스 및 자격 증명 구성을 로드하고 검증합니다.
2. 필수 로컬 및 클라우드 리소스를 사용하여 선택한 테스트를 수행합니다.
3. 로컬 및 클라우드 리소스를 정리합니다.
4. 보드에서 검증에 필요한 테스트를 통과했는지를 나타내는 테스트 보고서를 생성합니다.

## 테스트 제품군 버전

IDT for AWS IoT Greengrass V2는 테스트를 테스트 제품군과 테스트 그룹으로 구성합니다.

- 테스트 제품군은 장치가 AWS IoT Greengrass의 특정 버전에서 작동하는지 확인하는 데 사용되는 테스트 그룹 집합입니다.
- 테스트 그룹은 구성 요소 배포와 같은 특정 특성과 관련된 개별 테스트 집합입니다.

테스트 제품군은 *major.minor.patch* 형식을 사용하여 버전이 지정됩니다(예: GGV2Q\_1.0.0). IDT를 다운로드하면 패키지에 최신 Greengrass 검증 제품군 버전이 포함됩니다.

### Important

지원되지 않는 테스트 제품군 버전의 테스트는 장치 검증에 유효하지 않습니다. IDT는 지원되지 않는 버전에 대한 검증 보고서를 인쇄하지 않습니다. 자세한 내용은 [the section called “AWS IoT Device Tester용 AWS IoT Greengrass에 대한 지원 정책”](#) 단원을 참조하십시오. `list-supported-products`를 사용하여 현재 버전의 IDT에서 지원하는 AWS IoT Greengrass 및 테스트 제품군의 버전을 나열할 수 있습니다.

## 테스트 그룹 설명

### 코어 검증을 위한 필수 테스트 그룹

이러한 테스트 그룹은 AWS Partner Device Catalog에 대한 AWS IoT Greengrass V2 디바이스를 검증하는 데 필요합니다.

#### 코어 종속성

디바이스가 AWS IoT Greengrass 코어 소프트웨어에 대한 모든 소프트웨어 및 하드웨어 요구 사항을 충족하는지 여부를 확인합니다. 이 테스트 그룹에는 다음 테스트 사례가 포함됩니다.

#### Java 버전

테스트 중인 디바이스에 필요한 Java 버전이 설치되어 있는지 확인합니다. Java 8 이상이 AWS IoT Greengrass 필요합니다.

#### 사전 테스트 검증

디바이스가 테스트 실행을 위한 소프트웨어 요구 사항을 충족하는지 확인합니다.

- Linux 기반 디바이스의 경우 이 테스트는 디바이스가 다음 Linux 명령을 실행할 수 있는지 확인합니다.

`chmod, cp, echo, grep, kill, ln, mkinfo, ps, rm, sh, uname`

- Windows 기반 디바이스의 경우 이 테스트는 디바이스에 다음 Microsoft 소프트웨어가 설치되어 있는지 확인합니다.

[Powershell](#) v5.1 이상, [.NET](#) v4.6.1 이상, [Visual C++](#) 2017 이상, [PsExec 유틸리티](#)

## 버전 검사기

AWS IoT Greengrass 제공된 버전이 사용 중인 AWS IoT 디바이스 테스터 버전과 호환되는지 확인합니다.

## 구성 요소

디바이스가 구성 요소를 배포하고 업그레이드할 수 있는지 확인합니다. 이 테스트 그룹에는 다음 테스트가 포함됩니다.

### 클라우드 구성 요소

클라우드 구성 요소에 대한 디바이스 기능을 검증합니다.

### 로컬 구성 요소

로컬 구성 요소에 대한 디바이스 기능을 검증합니다.

## Lambda

이 테스트는 Windows 기반 디바이스에 적용되지 않습니다.

디바이스가 Java 런타임을 사용하는 Lambda 함수 구성 요소를 배포할 수 있고 Lambda 함수가 AWS IoT Core MQTT 주제를 작업 메시지의 이벤트 소스로 사용할 수 있는지 확인합니다.

## MQTT

디바이스가 AWS IoT Core MQTT 주제를 구독하고 게시할 수 있는지 확인합니다.

## 선택적 테스트 그룹

### Note

이러한 테스트 그룹은 선택 사항으로, Linux 기반 Greengrass 코어 디바이스를 검증하는 데만 사용됩니다. 선택적 테스트를 받기로 선택하면 디바이스 카탈로그에 추가 기능이 있는 AWS Partner 디바이스가 나열됩니다.

## Docker 종속성

디바이스가 AWS제공 Docker 애플리케이션 관리자 (`aws.greengrass.DockerApplicationManager`) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

## Docker 애플리케이션 관리자 검증

디바이스가 Amazon ECR에서 Docker 컨테이너 이미지를 다운로드할 수 있는지 확인합니다.

## 기계 학습 종속성

### Note

기계 학습 선택적 테스트 그룹은 IDT v4.9.3에서만 지원됩니다.

디바이스가 AWS제공 기계 학습(ML) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

## 기계 학습 추론 테스트

### Note

기계 학습 선택적 테스트 그룹은 IDT v4.9.3에서만 지원됩니다.

디바이스가 [딥 러닝 런타임](#) 및 [TensorFlow Lite](#) ML 프레임워크를 사용하여 ML 추론을 수행할 수 있는지 확인합니다.

## 스트림 관리자 종속성

### Note

스트림 관리자 선택적 테스트 그룹은 IDT v4.9.3에서만 지원됩니다.

디바이스가 [AWS IoT Greengrass 스트림 관리자](#)를 다운로드, 설치 및 실행할 수 있는지 확인합니다.

## HSI(하드웨어 보안 통합)

### Note

이 테스트는 Linux 기반 디바이스에 대해서만 IDT v4.9.3 이상에서 사용할 수 있습니다. AWS IoT Greengrass 는 현재 Windows 디바이스에 대한 하드웨어 보안 통합을 지원하지 않습니다.

디바이스가 하드웨어 보안 모듈(HSM)에 저장된 프라이빗 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 연결을 인증할 수 있는지 확인합니다. 또한 이 테스트



는 AWS제공 [PKCS#11 공급자 구성 요소](#)가 공급업체 제공 PKCS#11 라이브러리를 사용하여 HSM과 인터페이스할 수 있는지 확인합니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.

## AWS IoT Greengrass 검증 제품군을 실행하기 위한 사전 조건

이 섹션에서는 AWS IoT Device Tester 에 (IDT)를 사용하기 위한 사전 조건을 설명합니다 AWS IoT Greengrass.

### 용의 최신 버전 다운로드 AWS IoT Device Tester AWS IoT Greengrass

[최신 버전](#)의 IDT를 다운로드하여 읽기/쓰기 권한이 있는 파일 시스템의 위치(<*device-tester-extract-location*>)에 소프트웨어의 압축을 풉니다.

#### Note

여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하는 것은 지원되지 않습니다. 로컬 드라이브에 IDT 패키지의 압축을 풀고 로컬 워크스테이션에서 IDT 바이너리를 실행하는 것이 좋습니다.

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 경로를 260자 제한 아래로 유지하도록 IDT 압축을 C:\ 또는 D:\ 같은 루트 디렉터리에 풉니다.

## AWS IoT Greengrass 소프트웨어 다운로드

IDT for AWS IoT Greengrass V2는 디바이스의 특정 버전과의 호환성을 테스트합니다 AWS IoT Greengrass. 다음 명령을 실행하여 AWS IoT Greengrass 코어 소프트웨어를 라는 파일에 다운로드합니다 `aws.greengrass.nucleus.zip`. IDT 버전에 [지원되는 nucleus 구성 요소 버전](#)으로 ##을 바꿉니다.

### 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 `<device-tester-extract-location>/products/` 폴더에 배치합니다.

### Note

동일한 운영 체제 및 아키텍처에 대해 이 디렉터리에 여러 파일을 배치하지 마세요.

## 생성 및 구성 AWS 계정

AWS IoT Device Tester for AWS IoT Greengrass V2를 사용하려면 먼저 다음 단계를 수행해야 합니다.

1. [를 설정합니다 AWS 계정.](#) 가 이미 있는 경우 2단계로 AWS 계정건너뛵니다.
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

대다수 테스트는 [AWS 프리 티어](#) 대상으로 적합하지만, AWS 계정에 가입할 때 신용카드를 등록해야 합니다. 자세한 내용은 [계정에 프리 티어가 적용되는데 결제 방법이 필요한 이유는 무엇입니까?](#)를 참조하세요.

## 1단계: 설정 AWS 계정

이 단계에는 AWS 계정을 생성하고 구성합니다. 이미 AWS 계정이 있으면 [the section called “2단계: IDT에 대한 권한 구성”](#) 섹션으로 건너뛰니다.

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. <https://portal.aws.amazon.com/billing/signup>을 엽니다.
2. 온라인 지시 사항을 따르세요.

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

에 가입하면 AWS 계정AWS 계정 루트 사용자의 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 수행하는 것입니다.

다음 옵션 중 하나를 선택하여 관리 사용자를 생성합니다.

| 관리자를 관리하는 방법 한 가지 선택          | 목적                                                                                                                             | By                                                             | 다른 방법                                                                                                                 |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| IAM Identity Center에서<br>(권장) | 단기 보안 인증 정보를 사용하여 AWS에 액세스합니다.<br><br>이는 보안 모범 사례와 일치합니다. 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 <a href="#">IAM의 보안 모범 사례</a> 를 참조하세요. | AWS IAM Identity Center 사용 설명서의 <a href="#">시작하기</a> 지침을 따릅니다. | AWS Command Line Interface 사용 설명서에서 <a href="#">사용하도록 AWS CLI를 구성 AWS IAM Identity Center</a> 하여 프로그래밍 방식 액세스를 구성합니다. |

| 관리자를 관리하는 방법 한 가지 선택 | 목적                             | By                                                                  | 다른 방법                                                                         |
|----------------------|--------------------------------|---------------------------------------------------------------------|-------------------------------------------------------------------------------|
| IAM에서<br>(권장되지 않음)   | 장기 보안 인증 정보를 사용하여 AWS에 액세스합니다. | IAM 사용 설명서의 <a href="#">비상 액세스를 위한 IAM 사용자 생성</a> 에 나와 있는 지침을 따르세요. | IAM 사용 설명서에 나온 <a href="#">IAM 사용자의 액세스 키 관리</a> 를 수행하여 프로그래밍 방식의 액세스를 구성합니다. |

## 2단계: IDT에 대한 권한 구성

이 단계에서는 IDT for AWS IoT Greengrass V2가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. [AWS Management Console](#) 또는 [AWS Command Line Interface \(AWS CLI\)](#)를 사용하여 IDT에 대한 IAM 정책 및 테스트 사용자를 생성한 다음에 정책을 사용자에게 연결할 수 있습니다. IDT에 대한 테스트 사용자를 이미 생성했으면 [IDT 테스트를 실행하도록 장치 구성](#)으로 건너뛰십시오.

IDT에 대한 권한을 구성하려면(콘솔)

1. [IAM 콘솔](#)에 로그인합니다.
2. 특정 권한으로 역할을 생성하는 권한을 부여하는 고객 관리형 정책을 만듭니다.
  - a. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
  - b. PreInstalled를 사용하지 않는다면 JSON 탭에서 다음 정책으로 자리 표시자 내용을 바꿉니다. PreInstalled를 사용한다면 다음 단계로 진행합니다.

```
{
 "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. PreInstalled를 사용한다면 JSON 탭에서 다음 정책으로 자리 표시자 내용을 바꿉니다. 다음을 확인합니다.
- 테스트 중 디바이스(DUT)에 Greengrass가 설치되는 동안 생성된 사물 이름과 사물 그룹으로 `iotResources` 문의 *thingName*과 *thingGroup*을 바꾸어 권한을 추가합니다.
  - DUT에 Greengrass가 설치되는 동안 생성된 역할로 `roleAliasResources` 문과 `passRoleForResources` 문의 *passRole*과 *roleAlias*를 바꿉니다.

```

{
 "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**

테스트 중인 디바이스의 [토큰 교환 역할로 사용자 지정 IAM 역할](#)을 사용하려면 사용자 지정 IAM 역할 리소스가 허용되도록 정책의 `roleAliasResources` 문과 `passRoleForResources` 문을 업데이트해야 합니다.

- 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 따르세요.
    - 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 요약を 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 장치](#)를 구성합니다.

## IDT에 대한 권한을 구성하려면(AWS CLI)

1. 아직 설치되지 않은 AWS CLI 경우 컴퓨터에 설치하고 구성합니다. AWS Command Line Interface 사용 설명서 [AWS CLI설치](#) 단계를 따르세요.

**Note**

AWS CLI 는 명령줄 셸의 서비스와 상호 작용하는 데 사용할 수 있는 AWS 오픈 소스 도구입니다.

2. IDT 및 AWS IoT Greengrass 역할을 관리할 수 있는 권한을 부여하는 고객 관리형 정책을 만듭니다.
  - a. PreInstalled를 사용하지 않는다면 텍스트 편집기를 열고 다음 정책 내용을 JSON 파일에 저장합니다. PreInstalled를 사용한다면 다음 단계로 진행합니다.

```
{
 "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. PreInstalled를 사용한다면 텍스트 편집기를 열고 다음 정책 내용을 JSON 파일에 저장합니다. 다음을 확인합니다.

- 테스트 중 디바이스(DUT)에 Greengrass가 설치되는 동안 `iotResources` 문의 *thingName*과 *thingGroup*을 바꾸어 권한을 추가합니다.
- DUT에 Greengrass가 설치되는 동안 생성된 역할로 `roleAliasResources` 문과 `passRoleForResources` 문의 *passRole*와 *roleAlias*를 바꿉니다.



```
{
 "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/idx-*",
 "arn:aws:iot:*:*:cert/*",
 "arn:aws:iot:*:*:topic/idx-*",
 "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

테스트 중인 디바이스의 [토큰 교환 역할로 사용자 지정 IAM 역할](#)을 사용하려면 사용자 지정 IAM 역할 리소스가 허용되도록 정책의 `roleAliasResources` 문과 `passRoleForResources` 문을 업데이트해야 합니다.

- c. 다음 명령을 실행하여 IDTGreengrassIAMPermissions라는 고객 관리형 정책을 생성합니다. 이전 단계에서 생성한 JSON 파일의 전체 경로로 *policy.json*을 바꿉니다.

```
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 IDTGreengrassUser입니다.

```
aws iam create-user --user-name IDTGreengrassUser
```

- b. 2단계에서 생성한 IDGreengrassIAMPermissions 정책을 IAM 사용자에게 연결합니다. 명령의 `<account-id>`를 ID로 바꿉니다 AWS 계정.

```
aws iam attach-user-policy --user-name IDGreengrassUser --policy-arn
arn:aws:iam::<account-id>:policy/IDGreengrassIAMPermissions
```

4. 사용자에게 대한 비밀 액세스 키를 만듭니다.

```
aws iam create-access-key --user-name IDGreengrassUser
```

출력을 안전한 위치에 저장합니다. 나중에이 정보를 사용하여 AWS 자격 증명 파일을 구성합니다.

5. 다음 단계: [물리적 장치](#)를 구성합니다.

## AWS IoT Device Tester 권한

다음 정책은 AWS IoT Device Tester 권한을 설명합니다.

AWS IoT Device Tester에서는 버전 확인 및 자동 업데이트 기능에 이러한 권한이 필요합니다.

- `iot-device-tester:SupportedVersion`

지원되는 제품, 테스트 제품군 및 IDT 버전 목록을 가져올 수 있는 AWS IoT Device Tester 권한을 부여합니다.

- `iot-device-tester:LatestIdt`

다운로드할 수 있는 최신 IDT 버전을 가져올 수 있는 AWS IoT Device Tester 권한을 부여합니다.

- `iot-device-tester:CheckVersion`

IDT, 테스트 제품군 및 제품의 버전 호환성을 확인할 수 있는 AWS IoT Device Tester 권한을 부여합니다.

- `iot-device-tester:DownloadTestSuite`

테스트 제품군 업데이트를 다운로드할 수 있는 AWS IoT Device Tester 권한을 부여합니다.

AWS IoT Device Tester는 선택적 지표 보고에도 다음 권한을 사용합니다.

- `iot-device-tester:SendMetrics`

AWS IoT Device Tester 내부 사용에 대한 지표 AWS 를 수집할 수 있는 권한을 부여합니다. 이 권한을 생략하면 이러한 지표가 수집되지 않습니다.

## IDT 테스트를 실행하도록 장치 구성

IDT에서 디바이스 자격 검증을 위한 테스트를 실행되도록 하려면 디바이스에 액세스하도록 호스트 컴퓨터를 구성하고 디바이스에 대한 사용자 권한을 구성해야 합니다.

### 호스트 컴퓨터의 Java 설치

IDT v4.2.0부터에 대한 선택적 검증 테스트를 실행하려면 Java가 AWS IoT Greengrass 필요합니다.

Java 버전 8 이상을 사용할 수 있습니다. [Amazon Corretto](#) 또는 [OpenJDK](#) 장기 지원 버전을 사용하는 것이 좋습니다. 버전 8 이상이 필요합니다.

### 테스트 대상 장치에 액세스하도록 호스트 컴퓨터 구성

IDT는 호스트 컴퓨터에서 실행되며, SSH를 사용하여 장치에 연결할 수 있어야 합니다. IDT가 테스트 대상 장치에 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 장치에 로그인할 수 있도록 권한을 부여합니다.
2. `device.json` 파일의 각 장치에 사용자 이름 및 암호를 제공합니다. 자세한 내용은 [device.json 구성](#) 단원을 참조하십시오.

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 장치에 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 장치에 연결할 수 없습니다. 또한 원하는 사용자만 원격 장치에 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>\.ssh입니다.

메시지가 표시되면 SSH 키를 보호하기 위한 키 구문을 입력합니다. 자세한 내용은 [새 SSH 키 생성](#)을 참조하십시오.

2. 테스트 대상 장치에 권한 있는 SSH 키를 추가합니다,

IDT는 SSH 프라이빗 키를 사용하여 테스트 대상 장치에 로그인해야 합니다. 테스트 대상 장치에 로그인하도록 SSH 프라이빗 키를 승인하려면 호스트 컴퓨터의 ssh-copy-id 명령을 사용합니다. 이 명령은 테스트 대상 장치에서 ~/.ssh/authorized\_keys 파일에 퍼블릭 키를 추가합니다. 예제:

```
$ ssh-copy-id <remote-ssh-user>@<remote-device-ip>
```

여기서 *remote-ssh-user*는 테스트 대상 장치에 로그인하는 데 사용하는 사용자 이름이고, *remote-device-ip*는 테스트를 실행할 테스트 대상 장치의 IP 주소입니다. 예제:

```
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.pub로 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-ID](#)를 참조하십시오.

## PuTTYgen을 사용하여 SSH 키를 생성하려면(Windows만 해당)

1. 테스트 대상 장치에 OpenSSH 서버 및 클라이언트가 설치되어 있는지 확인합니다. 자세한 내용은 [OpenSSH](#)를 참조하십시오.
2. 테스트 대상 장치에 [PuTTYgen](#)을 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 장치에서 /home/<user>/.ssh/authorized\_keys 파일에 퍼블릭 키를 추가합니다.
  - a. PuTTYgen 창에서 퍼블릭 키 텍스트를 복사합니다.
  - b. PuTTY를 사용하여 테스트 대상 장치에서 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 장치의 암호를 입력합니다.
    - iii. vi 또는 다른 텍스트 편집기를 사용하여 테스트 대상 장치의 /home/<user>/.ssh/authorized\_keys 파일에 퍼블릭 키를 추가합니다.
7. 각 테스트 대상 장치에 대해 사용자 이름, IP 주소, 방금 호스트 컴퓨터에 저장한 프라이빗 키 파일의 경로로 device.json 파일을 업데이트합니다. 자세한 내용은 [the section called “device.json 구성”](#) 단원을 참조하십시오. 프라이빗 키에 전체 경로 및 파일 이름을 제공하고 슬래시('/')를 사용해야 합니다. 예를 들어 Windows 경로 C:\DT\privatekey.pem의 경우 device.json 파일에 C:/DT/privatekey.pem을 사용합니다.

## Windows 디바이스에 대한 사용자 자격 증명 구성

Windows 기반 디바이스를 검증하려면 다음 사용자에게 대해 테스트 중인 디바이스의 LocalSystem 계정에서 사용자 자격 증명을 구성해야 합니다.

- 기본 Greengrass 사용자(ggc\_user).
- 테스트 중인 디바이스에 연결하는 데 사용하는 사용자입니다. [device.json 파일](#)에서 이 사용자를 구성합니다.



테스트 중인 디바이스의 LocalSystem 계정에서 각 사용자를 생성한 다음에 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장해야 합니다.

### Windows 디바이스에서 사용자를 구성하는 방법

1. 관리자 권한으로 Windows 명령 프롬프트(cmd.exe)를 엽니다.
2. Windows 디바이스의 LocalSystem 계정에서 사용자를 생성합니다. 생성하려는 각 사용자에 대해 다음 명령을 실행합니다. 기본 Greengrass 사용자의 경우 *user-name*을 ggc\_user로 바꿉니다. *##*를 안전한 암호로 바꿉니다.

```
net user /add user-name password
```

3. Microsoft에서 [PsExec 유틸리티](#)를 다운로드하여 디바이스에 설치합니다.
4. PsExec 유틸리티를 사용하여 기본 사용자의 사용자 이름과 암호를 LocalSystem 계정의 Credential Manager 인스턴스에 저장합니다.

Credential Manager에서 구성하려는 각 사용자에 대해 다음 명령을 실행합니다. 기본 Greengrass 사용자의 경우 *user-name*을 ggc\_user로 바꿉니다. 이전에 설정한 사용자의 암호로 *##*를 바꿉니다.

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

PsExec License Agreement가 열리면 Accept를 선택하여 라이선스에 동의하고 명령을 실행합니다.

#### Note

Windows 디바이스에서는 LocalSystem 계정에서 Greengrass nucleus가 실행되며, PsExec 유틸리티를 사용하여 사용자 정보를 LocalSystem 계정에 저장해야 합니다. Credential Manager 애플리케이션을 사용하면 이 정보가 LocalSystem 계정 대신 현재 로그인한 사용자의 Windows 계정에 저장됩니다.

### 장치에서 사용자 권한 구성

IDT는 테스트 대상 장치에서 다양한 디렉터리와 파일에 대해 작업을 수행합니다. 이러한 작업 중 일부는 승격된 권한(sudo 사용)이 필요합니다. 이러한 작업을 자동화하려면 IDT for AWS IoT Greengrass V2가 암호를 입력하라는 메시지 없이 sudo로 명령을 실행할 수 있어야 합니다.

암호 입력 메시지 없이 sudo 액세스를 허용하려면 테스트 대상 장치에서 다음 단계를 수행합니다.

#### Note

username은 IDT가 테스트 대상 장치에 액세스하는 데 사용하는 SSH 사용자를 나타냅니다.

sudo 그룹에 사용자를 추가하려면

1. 테스트 대상 장치에서 `sudo usermod -aG sudo <username>`을 실행합니다.
2. 변경 사항을 적용하려면 로그아웃했다가 다시 로그인하십시오.
3. 사용자 이름이 성공적으로 추가되었는지 확인하려면 `sudo echo test`를 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 역할 구성

테스트 중인 디바이스가 AWS 리소스와 상호 작용하기 위해 수입하는 토큰 교환 역할로 사용자 지정 IAM 역할을 사용하도록 선택할 수 있습니다. IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 역할 생성](#) 섹션을 참조하세요.

IDT에 사용자 지정 IAM 역할 사용을 허용하려면 다음과 같은 요구 사항을 충족해야 합니다. 이 역할에 필요한 최소 정책 작업만 추가하는 것이 좋습니다.

- [userdata.json](#) 구성 파일을 업데이트하여 `GreengrassV2TokenExchangeRole` 파라미터를 `true`로 설정해야 합니다.
- 사용자 지정 IAM 역할은 다음 최소 신뢰 정책으로 구성되어야 합니다.

JSON

```
{
 "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 다음 최소 권한 정책으로 구성되어야 합니다.

JSON

```

{
 "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 권한에서 지정하는 IAM 역할 리소스와 일치해야 합니다. 기본적으로 [테스트 사용자 정책](#)에서는 역할 이름에 idt- 접두사가 있는 IAM

역할에 대한 액세스가 허용됩니다. IAM 역할 이름에 이 접두사가 사용되지 않으면 다음 예제와 같이 테스트 사용자 정책의 `roleAliasResources` 문과 `passRoleForResources` 문에 `arn:aws:iam::*:role/custom-iam-role-name` 리소스를 추가합니다.

### 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 장치 구성

이 섹션에는 선택적 Docker 및 기계 학습(ML) 특성에 대한 IDT 테스트 실행에 대한 디바이스 요구 사항이 설명되어 있습니다. ML 특성은 IDT v4.9.3에서만 지원됩니다. 이러한 특성을 테스트하려는 경우에만 디바이스에서 이러한 요구 사항이 충족되는지 확인해야 합니다. 그렇지 않은 경우 [the section called “IDT 설정 구성”](#)를 계속 진행합니다.

### 주제

- [Docker 검증 요구 사항](#)
- [ML 검증 요구 사항](#)
- [HSM 검증 요구 사항](#)

### Docker 검증 요구 사항

IDT for AWS IoT Greengrass V2는 디바이스가 AWS제공 Docker [애플리케이션 관리자 구성 요소를 사용하여 사용자 지정 Docker](#) 컨테이너 구성 요소를 사용하여 실행할 수 있는 Docker 컨테이너 이미지를 다운로드할 수 있는지 검증하는 Docker 검증 테스트를 제공합니다. 사용자 지정 Docker 구성 요소 생성에 대한 내용은 [Docker 컨테이너 실행](#) 섹션을 참조하세요.

Docker 검증 테스트를 실행하려면 테스트 중인 디바이스에서 Docker 애플리케이션 관리자 구성 요소가 배포되도록 다음과 같은 요구 사항이 충족되어야 합니다.

- Greengrass 코어 디바이스에 [Docker Engine](#) 1.9.1 이상이 설치되어 있어야 합니다. 버전 20.10은 AWS IoT Greengrass 코어 소프트웨어에서 작동하는 것으로 확인된 최신 버전입니다. Docker 컨테이너를 실행하는 구성 요소를 배포하기 전에 코어 디바이스에 직접 Docker를 설치해야 합니다.
- 이 구성 요소를 배포하기 전에 Docker 대몬이 코어 디바이스에서 시작되어 실행되고 있습니다.
- 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 sudo 없이 docker 명령을 직접적으로 호출할 수 있습니다.
- Windows 디바이스에서는 docker-users 그룹에 사용자를 추가하여 관리자 권한 없이 docker 명령을 직접적으로 호출할 수 있습니다.

### Linux or Unix

Docker 컨테이너 구성 요소를 실행하는 데 사용하는 ggc\_user 또는 루트 사용자가 아닌 사용자를 docker 그룹에 추가하려면 다음 명령을 실행합니다.

```
sudo usermod -aG docker ggc_user
```

자세한 내용은 [루트 사용자가 아닌 사용자로 Docker 관리](#)를 참조하세요.

### Windows Command Prompt (CMD)

*ggc\_user* 또는 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자를 `docker-users` 그룹에 추가하려면 관리자로 다음 명령을 실행합니다.

```
net localgroup docker-users ggc_user /add
```

### Windows PowerShell

*ggc\_user* 또는 Docker 컨테이너 구성 요소를 실행하는 데 사용하는 사용자를 `docker-users` 그룹에 추가하려면 관리자로 다음 명령을 실행합니다.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

## ML 검증 요구 사항

### Note

기계 학습 특성은 IDT v4.9.3에서만 지원됩니다.

IDT for AWS IoT Greengrass V2는 디바이스가 AWS제공 [기계 학습 구성 요소](#)를 사용하여 [딥 러닝 런타임](#) 또는 [TensorFlow Lite](#) ML 프레임워크를 사용하여 로컬에서 ML 추론을 수행할 수 있는지 검증하는 ML 검증 테스트를 제공합니다. Greengrass 디바이스의 ML 추론 실행에 대한 자세한 내용은 [기계 학습 추론 수행](#) 섹션을 참조하세요.

ML 검증 테스트를 실행하려면 테스트 중인 디바이스에서 기계 학습 구성 요소가 배포되도록 다음과 같은 요구 사항이 충족되어야 합니다.

- Amazon Linux 2 또는 Ubuntu 18.04를 실행 중인 Greengrass 코어 디바이스의 경우 [GNU C 라이브러리](#)(glibc) 버전 2.27 이상이 디바이스에 설치되어 있어야 합니다.
- Raspberry Pi와 같은 Armv7l 디바이스에는 디바이스에 OpenCV-Python에 대한 종속성이 설치되어 있어야 합니다. 다음 명령을 실행하여 종속성을 설치합니다.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Raspberry Pi OS Bullseye를 실행하는 Raspberry Pi 디바이스는 다음과 같은 요구 사항을 충족해야 합니다.
- 디바이스에 NumPy 1.22.4 이상이 설치되어 있어야 합니다. Raspberry Pi OS Bullseye에는 이전 버전의 NumPy가 포함되어 있으므로 다음 명령을 실행하여 디바이스에서 NumPy를 업그레이드 할 수 있습니다.

```
pip3 install --upgrade numpy
```

- 디바이스에서 레거시 카메라 스택이 활성화되어 있어야 합니다. Raspberry Pi OS Bullseye에는 기본적으로 활성화되어 있지만 호환되지 않는 새 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 검증 요구 사항

AWS IoT Greengrass 는 디바이스의 [PKCS 하드웨어 보안 모듈\(HSM\)과 통합하기 위한 PKCS#11 공급자 구성 요소](#)를 제공합니다. HSM 설정은 디바이스 및 선택한 HSM 모듈에 따라 다릅니다. [IDT 구성 설정](#)에 문서화되어 있는 대로 예상 HSM 구성이 제공되는 한 이 선택적 특성 검증 테스트 실행에 필요한 정보가 IDT에 있습니다.

## AWS IoT Greengrass 검증 제품군을 실행하도록 IDT 설정 구성

테스트를 실행하기 전에 호스트 컴퓨터에서 AWS 자격 증명 및 디바이스에 대한 설정을 구성해야 합니다.

## config.json에서 AWS 자격 증명 구성

`<device_tester_extract_location>`/configs/config.json 파일에서 IAM 사용자 보안 인증을 구성해야 합니다. 에서 생성된 IDT for AWS IoT Greengrass V2 사용자의 자격 증명을 사용합니다. [the section called “생성 및 구성 AWS 계정”](#). 두 가지 방법 중 하나로 자격 증명을 지정할 수 있습니다.

- 자격 증명 파일에서
- 환경 변수로

### AWS 자격 증명 파일을 사용하여 자격 증명 구성

IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 [구성 및 자격 증명 파일](#)을 참조하십시오.

자격 증명 파일의 위치는 사용하는 운영 체제에 따라 달라집니다.

- macOS, Linux의 경우: `~/.aws/credentials`
- Windows: `C:\Users\UserName\.aws\credentials`

다음 형식으로 자격 AWS 증명을 credentials 파일에 추가합니다.

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

credentials 파일의 AWS 자격 증명을 사용하도록 IDT for AWS IoT Greengrass V2를 구성하려면 다음과 같이 config.json 파일을 편집합니다.

```
{
 "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 프로필](#)을 참조하십시오.

**환경 변수를 사용하여 AWS 자격 증명 구성**

환경 변수는 운영 체제에서 유지 관리하고 시스템 명령에서 사용하는 변수입니다. 이들은 SSH 세션을 닫으면 저장되지 않습니다. IDT for AWS IoT Greengrass V2는 AWS\_ACCESS\_KEY\_ID 및 AWS\_SECRET\_ACCESS\_KEY 환경 변수를 사용하여 AWS 자격 증명을 저장할 수 있습니다.

Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 export를 사용합니다.

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Windows에서 이러한 변수를 설정하려면 set을 사용합니다.

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

환경 변수를 사용하도록 IDT를 구성하려면 config.json 파일에서 auth 섹션을 편집합니다. 다음 예를 참조하세요

```
{
 "awsRegion": "region",
 "auth": {
 "method": "environment"
 }
}
```

**device.json 구성****Note**

IDT v4.9.3은 m1, docker, streamManagement 기능 테스트를 지원합니다. IDT v4.9.4 이상 버전은 docker 테스트를 지원합니다. 이러한 기능을 테스트하지 않으려면 해당 값을 no로 설정합니다.

자격 AWS 증명 외에도 IDT for AWS IoT Greengrass V2에는 테스트가 실행되는 디바이스에 대한 정보가 필요합니다. IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처가 예제 정보가 될 수 있습니다.

`<device_tester_extract_location>/configs/device.json`에 있는 `device.json` 템플릿을 사용하여 이 정보를 제공해야 합니다.

IDT v4.9.3

```
[
 {
 "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

method가 pki로 설정된 경우에만 privKeyPath를 지정합니다.  
method가 password로 설정된 경우에만 password를 지정합니다.

여기에 설명된 대로 값을 포함하는 모든 속성은 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 장치를 고유하게 식별하는 영숫자 값입니다. SKU는 적격 보드를 추적하는 데 사용됩니다.

#### Note

AWS Partner 디바이스 카탈로그에 디바이스를 나열하려면 여기에서 지정하는 SKU가 나열 프로세스에서 사용하는 SKU와 일치해야 합니다.

## features

장치의 지원되는 기능이 포함된 배열입니다. 모든 기능이 필요합니다.

### arch

테스트 실행이 검증하는 지원되는 운영 체제 아키텍처. 유효값은 다음과 같습니다.

- x86\_64
- armv6l
- armv7l
- aarch64

### ml

디바이스가 AWS제공 기계 학습(ML) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

또한 이 기능을 활성화하면 디바이스가 [딥 러닝 런타임](#) 및 [TensorFlow Lite](#) ML 프레임워크를 사용하여 ML 추론을 수행할 수 있는지 검증합니다.

유효한 값은 dlr 및 tensorflowlite 또는 no의 모든 조합입니다.

### docker

디바이스가 AWS제공 Docker 애플리케이션 관리자(aws.greengrass.DockerApplicationManager) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

또한 이 기능을 활성화하면 디바이스가 Amazon ECR에서 Docker 컨테이너 이미지를 다운로드할 수 있는지 검증합니다.

유효한 값은 yes 또는 no의 모든 조합입니다.

### streamManagement

디바이스가 [AWS IoT Greengrass 스트림 관리자](#)를 다운로드, 설치 및 실행할 수 있는지 확인합니다.

유효한 값은 yes 또는 no의 모든 조합입니다.

### hsi

디바이스가 하드웨어 보안 모듈(HSM)에 저장된 프라이빗 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 연결을 인증할 수 있는지 확인합니다. 또한 이 테스트

트는 AWS제공 [PKCS#11 공급자 구성 요소가](#) 공급업체 제공 PKCS#11 라이브러리를 사용하여 HSM과 인터페이스할 수 있는지 확인합니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.

유효한 값은 hsm 또는 no입니다.

 Note

hsi 테스트는 IDT v4.9.3 이상 버전에서만 사용할 수 있습니다.

`devices.id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`devices.operatingSystem`

디바이스 운영 체제. 지원되는 값은 Linux 및 Windows입니다.

`connectivity.protocol`

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 현재 지원되는 유일한 값은 물리적 디바이스에 대한 ssh 값입니다.

`connectivity.ip`

테스트 대상 디바이스의 IP입니다.

이 속성은 `connectivity.protocol`이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.publicKeyPath`

선택 사항. 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 주소가 있는 항목이 여러 개 있는 경우 IDT에서 사용하는 키 유형에 대한 항목이 파일의 다른 항목보다 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.

## IDT v4.9.4

```
[
 {
 "id": "<pool-id>",
```

```

"sku": "<sku>",
"features": [
 {
 "name": "arch",
 "value": "x86_64 | armv6l | armv7l | aarch64"
 },
 {
 "name": "docker",
 "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

method가 pki로 설정된 경우에만 privKeyPath를 지정합니다.  
 method가 password로 설정된 경우에만 password를 지정합니다.

여기에 설명된 대로 값을 포함하는 모든 속성은 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 장치를 고유하게 식별하는 영숫자 값입니다. SKU는 적격 보드를 추적하는 데 사용됩니다.

### Note

AWS Partner 디바이스 카탈로그에 디바이스를 나열하려면 여기에서 지정하는 SKU가 나열 프로세스에서 사용하는 SKU와 일치해야 합니다.

## features

장치의 지원되는 기능이 포함된 배열입니다. 모든 기능이 필요합니다.

## arch

테스트 실행이 검증하는 지원되는 운영 체제 아키텍처. 유효값은 다음과 같습니다.

- x86\_64
- armv6l
- armv7l
- aarch64

## docker

디바이스가 AWS제공 Docker 애플리케이션 관리자(`aws.greengrass.DockerApplicationManager`) 구성 요소를 사용하는 데 필요한 모든 기술 종속성을 충족하는지 확인합니다.

또한 이 기능을 활성화하면 디바이스가 Amazon ECR에서 Docker 컨테이너 이미지를 다운로드할 수 있는지 검증합니다.



유효한 값은 yes 또는 no의 모든 조합입니다.

hsi

디바이스가 하드웨어 보안 모듈(HSM)에 저장된 프라이빗 키와 인증서를 사용하여 AWS IoT 및 AWS IoT Greengrass 서비스에 대한 연결을 인증할 수 있는지 확인합니다. 또한 이 테스트는 AWS제공 [PKCS#11 공급자 구성 요소가](#) 공급업체 제공 PKCS#11 라이브러리를 사용하여 HSM과 인터페이스할 수 있는지 확인합니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.

유효한 값은 hsm 또는 no입니다.

 Note

hsi 테스트는 IDT v4.9.3 이상 버전에서만 사용할 수 있습니다.

devices.id

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

devices.operatingSystem

디바이스 운영 체제. 지원되는 값은 Linux 및 Windows입니다.

connectivity.protocol

이러한 디바이스와 통신하는 데 사용되는 통신 프로토콜입니다. 현재 지원되는 유일한 값은 물리적 디바이스에 대한 ssh 값입니다.

connectivity.ip

테스트 대상 디바이스의 IP 입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

connectivity.port

선택 사항. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

## connectivity.publicKeyPath

선택 사항. 테스트 대상 디바이스에 대한 연결을 인증하는 데 사용되는 퍼블릭 키의 전체 경로입니다.

publicKeyPath를 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 주소가 있는 항목이 여러 개 있는 경우 IDT에서 사용하는 키 유형에 대한 항목이 파일의 다른 항목보다 앞에 있어야 합니다.

## 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.

## userdata.json 구성

또한 IDT for AWS IoT Greengrass V2에는 테스트 아티팩트 및 AWS IoT Greengrass 소프트웨어의 위치에 대한 추가 정보가 필요합니다.

`<device_tester_extract_location>/configs/userdata.json`에 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"
 }
}
```

여기에 설명된 대로 값을 포함하는 모든 속성은 필수입니다.

## TempResourcesDirOnDevice

테스트 아티팩트를 저장할 테스트 대상 디바이스의 임시 폴더 전체 경로. 이 디렉터리에 쓰는 데 `sudo` 권한이 필요하지 않은지 확인합니다.

### Note

IDT는 테스트 실행이 완료되면 이 폴더의 콘텐츠를 삭제합니다.

## InstallationDirRootOnDevice

AWS IoT Greengrass를 설치할 디바이스의 폴더의 전체 경로. 사전 설치된 Greengrass의 경우 이는 Greengrass 설치 디렉터리의 경로입니다.

이 폴더에 필요한 파일 권한을 설정해야 합니다. 설치 경로의 각 폴더에 대해 다음 명령을 실행합니다.

```
sudo chmod 755 folder-name
```

## GreengrassNucleusZip

호스트 컴퓨터의 Greengrass nucleus ZIP(greengrass-nucleus-latest.zip) 파일의 전체 경로. 이 필드는 사전 설치된 Greengrass를 사용한 테스트에는 필요하지 않습니다.

### Note

용 IDT용 Greengrass nucleus의 지원되는 버전에 대한 자세한 내용은 섹션을 AWS IoT Greengrass참조하세요 [AWS IoT Greengrass V2의 최신 IDT 버전](#). 최신 Greengrass 소프트웨어를 다운로드하려면 [AWS IoT Greengrass 소프트웨어 다운로드](#)를 참조하세요.

## PreInstalled

이 기능은 Linux 디바이스의 IDT v4.5.8 이상 버전에서만 사용할 수 있습니다.

(선택 사항) 값이 **Yes**인 경우 IDT는 InstallationDirRootOnDevice에 언급된 경로를 Greengrass가 설치된 디렉터리라고 가정합니다.

디바이스에 Greengrass를 설치하는 방법에 대한 자세한 내용은 [자동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 섹션을 참조하세요. [수동 프로비저닝을 사용하여를 설치하는](#) 경우 AWS IoT 사물을 수동으로 생성할 때 “새 사물 또는 기존 사물 그룹에 [AWS IoT 사물](#) 추가” 단계를 포함합니다. IDT는 설치 설정 중에 사물 그룹이 생성된다고 가정합니다. 이러한 값이 effectiveConfig.yaml 파일에 반영되어 있는지 확인합니다. IDT는 <InstallationDirRootOnDevice>/config/effectiveConfig.yaml에서 effectiveConfig.yaml 파일을 확인합니다.

HSM을 사용하여 테스트를 실행하려면 aws.greengrass.crypto.Pkcs11Provider 필드가 effectiveConfig.yaml에서 업데이트되었는지 확인합니다.

## GreengrassV2TokenExchangeRole

(선택 사항) 테스트 중인 디바이스가 AWS 리소스와 상호 작용하기 위해 수입하는 토큰 교환 역할로 사용할 사용자 지정 IAM 역할.

### Note

IDT는 테스트 실행 중에 기본 토큰 교환 역할을 생성하는 대신 이 사용자 지정 IAM 역할을 사용합니다. 사용자 지정 역할을 사용하는 경우 [테스트 사용자에게 대한 IAM 권한](#)을 업데이트하여 사용자가 IAM 역할 및 정책을 생성하고 삭제할 수 있도록 허용하는 `iamResourcesUpdate` 문을 제외할 수 있습니다.

사용자 지정 IAM 역할을 토큰 교환 역할로 생성하는 방법에 대한 자세한 내용은 [사용자 지정 토큰 교환 역할 구성](#) 섹션을 참조하세요.

## hsm

이 기능은 IDT v4.5.1 이상에서 사용할 수 있습니다.

(선택 사항) AWS IoT Greengrass 하드웨어 보안 모듈(HSM)을 사용하여 테스트하기 위한 구성 정보. 그렇지 않으면 `hsm` 속성을 생략해야 합니다. 자세한 내용은 [하드웨어 보안 통합](#) 단원을 참조하십시오.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

### Warning

하드웨어 보안 모듈이 IDT와 다른 시스템 간에 공유되는 경우 HSM 구성은 민감한 데이터로 간주될 수 있습니다. 이러한 경우 구성 값을 AWS Parameter Store SecureString 파라미터에 저장하고 테스트 실행 중에 가져오도록 IDT를 구성하여 일반 텍스트로 보호하지 않을 수 있습니다. 자세한 내용은 [???](#) 섹션을 참조하세요.

## hsm.greengrassPkcsPluginJar

IDT 호스트 시스템에 다운로드하는 [PKCS#11 공급자 구성 요소](#)의 전체 경로. AWS IoT Greengrass는 이 구성 요소를 설치하는 중 프로비저닝 플러그인으로 지정하기 위해 다운로드할 수 있는 JAR 파일로 제공합니다. 다음 URL(<https://d2s8p88vqu9w66.cloudfront.net/releases/>

[Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar](#))에서 최신 버전의 구성 요소 JAR 파일을 다운로드할 수 있습니다.

`hsm.pkcs11ProviderLibrary`

하드웨어 보안 모듈(HSM) 공급업체가 HSM과 상호 작용하기 위해 제공하는 PKCS#11 라이브러리의 전체 경로.

`hsm.slotId`

키와 인증서를 로드하는 HSM 슬롯을 식별하는 데 사용되는 슬롯 ID.

`hsm.slotLabel`

키와 인증서를 로드하는 HSM 슬롯을 식별하는 데 사용되는 슬롯 레이블.

`hsm.slotUserPin`

IDT가 AWS IoT Greengrass 코어 소프트웨어를 HSM에 인증하는 데 사용하는 사용자 PIN입니다.

 Note

보안 모범 사례로 프로덕션 디바이스에서 동일한 사용자 PIN을 사용하지 마세요.

`hsm.keyLabel`

하드웨어 모듈에서 키를 식별하는 데 사용되는 레이블입니다. 키와 인증서 모두 동일한 키 레이블을 사용해야 합니다.

`hsm.preloadedCertificateArn`

AWS IoT 클라우드에 업로드한 디바이스 인증서의 Amazon 리소스 이름(ARN).

이전에 HSM의 키를 사용하여 인증서를 생성하고 HSM으로 가져온 다음 AWS IoT 클라우드에 업로드해야 합니다. 인증서 생성 및 가져오기에 대한 자세한 내용은 HSM에 대한 설명서를 참조하세요.

인증서를 [config.json](#)에 제공한 것과 동일한 계정 및 리전에 인증서를 업로드해야 합니다. 인증서에 업로드하는 방법에 대한 자세한 내용은 AWS IoT 개발자 안내서의 [수동으로 클라이언트 인증서 등록](#)을 AWS IoT참조하세요.

## hsm.rootCAPath

(선택 사항) 인증서를 서명한 루트 인증 기관(CA)에 대한 IDT 호스트 컴퓨터의 전체 경로. 이는 생성한 HSM의 인증서가 Amazon 루트 CA에 의해 서명하지 않은 경우에 필요합니다.

## AWS 파라미터 스토어에서 구성 가져오기

AWS IoT 디바이스 테스터(IDT)에는 [AWS Systems Manager 파라미터 스토어](#)에서 구성 값을 가져오는 선택적 기능이 포함되어 있습니다. AWS 파라미터 스토어는 안전하고 암호화된 구성 저장을 허용합니다. 구성된 경우 IDT는 파라미터를 userdata.json 파일 내부의 일반 텍스트로 저장하는 대신 파라미터 AWS 스토어에서 파라미터를 가져올 수 있습니다. 이는 암호, PIN, 기타 보안 암호와 같이 암호화된 상태로 저장해야 하는 민감한 데이터에 유용합니다.

1. 이 기능을 사용하려면 [IDT 사용자](#)를 생성하는 데 사용되는 권한을 업데이트하여 IDT가 사용하도록 구성된 파라미터에 대한 GetParameter 작업을 허용해야 합니다. 다음은 IDT 사용자에게 추가할 수 있는 권한 문의 예제입니다. 자세한 내용은 [AWS Systems Manager 사용 설명서](#)를 참조하세요.

```
{
 "Sid": "parameterStoreResources",
 "Effect": "Allow",
 "Action": [
 "ssm:GetParameter"
],
 "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
}
```

위 권한은 와일드카드 문자(IDT)를 사용하여 이름이 \*로 시작하는 모든 파라미터를 가져올 수 있도록 구성되었습니다. 사용 중인 파라미터의 이름에 따라 IDT가 구성되어 있는 파라미터를 가져올 수 있도록 필요에 따라 이를 사용자 지정해야 합니다.

2. AWS Paramater Store 내에 구성 값을 저장해야 합니다. 이 작업은 AWS 콘솔 또는 AWS CLI에서 수행할 수 있습니다. AWS Parameter Store를 사용하면 암호화되거나 암호화되지 않은 스토리지를 선택할 수 있습니다. 보안 암호, PIN 등 민감한 값을 저장하려면 SecureString의 파라미터 유형인 암호화된 옵션을 사용해야 합니다. AWS CLI를 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 해독된 SecureString 파라미터를 가져옵니다.

```
aws ssm get-parameter --name IDT-example-name
```

AWS CLI를 사용하면 현재 CLI 사용자의 AWS 리전에 파라미터가 업로드되고 IDT는에 구성된 리전에서 파라미터를 가져옵니다 `config.json`. AWS CLI에서 리전을 확인하려면 다음을 사용합니다.

```
aws configure get region
```

3. AWS 클라우드에 구성 값이 있으면 IDT 구성 내의 모든 값을 업데이트하여 AWS 클라우드에서 가져올 수 있습니다. 이렇게 하려면 양식의 IDT 구성에 자리 표시자를 사용하여 파라미터 스토어에서 해당 이름으로 AWS 파라미터를 `{{AWS.Parameter.parameter_name}}` 가져옵니다.

예를 들어 2단계의 IDT-example-name 파라미터를 HSM 구성의 HSM keyLabel로 사용하려는 경우를 가정해 보겠습니다. 이렇게 하려면 다음과 같이 `userdata.json`을 업데이트하면 됩니다.

```
"hsm": {
 "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
 [...]
}
```

IDT는 2단계에서 IDT-example-value로 설정된 런타임에서 이 파라미터의 값을 가져옵니다. 이 구성은 설정과 유사 `"keyLabel": "IDT-example-value"` 하지만 대신 해당 값은 AWS 클라우드에 암호화된 상태로 저장됩니다.

## AWS IoT Greengrass 검증 제품군 실행

[필수 구성을 설정](#)한 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` 명령을 사용하여 테스트 제품군을 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-idx y/n \\
--userdata userdata.json
```

모든 옵션은 선택 사항입니다. 예를 들어 device.json 파일에 정의된 동일한 디바이스의 세트인 디바이스 풀이 하나만 있는 경우 pool-id를 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 내용은 [the section called “테스트 제품군 버전”](#) 단원을 참조하십시오.

## 자격 제품군을 실행하는 예제 명령

다음 명령줄 예제에서는 디바이스 풀에 대한 자격 테스트를 실행하는 방법을 설명합니다. run-suite 및 기타 IDT 명령에 대한 자세한 내용은 [the section called “IDT 명령”](#) 섹션을 참조하십시오.

다음 명령을 사용하여 지정된 테스트 제품군에 있는 모든 테스트 그룹을 실행합니다. list-suites 명령은 tests 폴더에 있는 테스트 제품군을 나열합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
--suite-id GGV2Q_1.0.0 \
--pool-id <pool-id> \
--userdata userdata.json
```

다음 명령을 사용하여 테스트 제품군의 특정 테스트 그룹을 실행합니다. list-groups 명령은 테스트 제품군의 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 명령을 사용하여 테스트 그룹의 특정 테스트 사례를 실행합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
```

```
--group-id <group-id> \
--test-id <test-id> \
--userdata userdata.json
```

다음 명령을 사용하여 테스트 그룹의 여러 테스트 사례를 실행합니다.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
--group-id <group-id> \
--test-id <test-id1>,<test-id2>
--userdata userdata.json
```

다음 명령을 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 자격 테스트 제품군을 실행하는 것이 좋습니다. 특정 테스트 그룹을 실행하기로 선택한 경우 관련 테스트 그룹을 실행하기 전에 먼저 종속성 검사기 테스트 그룹을 실행하여 모든 Greengrass 종속성이 설치되어 있는지 확인하는 것이 좋습니다. 예제:

- 코어 자격 테스트 그룹을 실행하기 전에 `coredependencies`를 실행합니다.

## IDT for AWS IoT Greengrass V2 명령

IDT 명령은 `<device-tester-extract-location>/bin` 디렉터리에 있습니다. 테스트 제품군을 실행하려면 다음 형식으로 명령을 입력합니다.

`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 및 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

장치의 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(선택으로 구분된 목록). 지정되지 않은 경우 IDT는 `device.json`에서 구성된 설정에 따라 테스트 제품군에서 모든 적절한 테스트 그룹을 실행합니다. IDT는 구성된 설정에 따라 디바이스가 지원하지 않는 테스트 그룹이 `group-id` 목록에 지정되어 있더라도 테스트 그룹을 실행하지 않습니다.
- `test-id`. 실행할 테스트 케이스(선택으로 구분된 목록). 지정된 경우, `group-id`은(는) 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 디바이스 풀. `device.json` 파일에 여러 장치 풀이 정의되어 있는 경우 하나의 풀을 지정해야 합니다.
- `stop-on-first-failure`. 첫 번째 실패 시 IDT가 실행을 중지하도록 구성합니다. 지정된 테스트 그룹을 디버깅하려는 경우 `group-id`와 함께 이 옵션을 사용합니다. 전체 테스트 제품군을 실행하여 검증 보고서를 생성할 때는 이 옵션을 사용하지 마시기 바랍니다.
- `update-idt`. IDT를 업데이트하라는 프롬프트에 대한 응답을 설정합니다. IDT에서 최신 버전이 있음을 감지하면 Y 응답이 테스트 실행을 중지합니다. N 응답은 테스트 실행을 계속합니다.
- `userdata`. 테스트 아티팩트 경로에 대한 정보가 포함되어 있는 `userdata.json` 파일의 전체 경로입니다. 이 옵션은 `run-suite` 명령에 필요합니다. `userdata.json` 파일은 `devicetester_extract_location/devicetester_ggv2_[win|mac|linux]` 디렉터리에 있어야 합니다.

`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.

오류를 해결하려면 [IDT for AWS IoT Greengrass V2 문제 해결](#) 섹션을 참조하세요.

## 결과 보기

실행하는 동안 IDT는 콘솔, 로그 파일 및 테스트 보고서에 오류를 작성합니다. IDT는 자격 테스트 제품군을 완료한 후 두 개의 테스트 보고서를 생성합니다. 이러한 보고서는 `<device-tester-extract-location>/results/<execution-id>/`에 있습니다. 두 보고서 모두 적격성 테스트 제품군을 실행하여 얻은 결과를 캡처합니다.

`awsiotdevicetester_report.xml`은 AWS Partner Device Catalog에 장치를 등록하기 위해 AWS에 제출하는 자격 테스트 보고서입니다. 보고서에는 다음 요소가 포함됩니다.

- IDT 버전
- 테스트한 AWS IoT Greengrass 버전
- `device.json` 파일에 지정된 SKU 및 장치 풀 이름
- `device.json` 파일에 지정된 장치 풀의 기능
- 테스트 결과의 집계 요약
- 로컬 리소스 액세스, 새도, MQTT 등 디바이스 기능을 기반으로 테스트한 라이브러리별 테스트 결과 분석입니다.

`GGV2Q_Result.xml` 보고서는 [JUnit XML 형식](#)입니다. [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 Tester 결과 해석

`awsiotdevicetester_report.xml` 또는 `awsiotdevicetester_report.xml`의 보고서 섹션에는 실행된 테스트 및 결과가 나열됩니다.

첫 번째 XML 태그(`<testsuites>`)에는 테스트 실행의 요약이 포함되어 있습니다. 예:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0" disabled="0">
```

## **<testsuites>** 태그에 사용되는 속성

### name

테스트 제품군의 이름입니다.

### time

자격 제품군을 실행하는 데 걸린 시간(초)입니다.

### tests

실행된 테스트 수입니다.

### failures

실행되었지만 통과하지 못한 테스트의 수입니다.

### errors

IDT에서 실행하지 못한 테스트의 수입니다.

### disabled

이 속성을 무시합니다. 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. required로 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 장치는 AWS IoT Greengrass를 실행하기 위한 기술 요구 사항을 충족하며 AWS IoT 서비스와 상호 작용할 수 있습니다. AWS Partner Device Catalog에 장치를 나열하려는 경우 이 보고서를 자격 증명으로 사용할 수 있습니다.

테스트 실패 또는 오류의 경우 <testsuites> XML 태그를 검토하여 실패한 테스트를 식별할 수 있습니다. <testsuite> 태그 내부의 <testsuites> XML 태그는 테스트 그룹에 대한 테스트 결과 요약 을 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 안에는 테스트 그룹에 대해 실행된 각 테스트에 대한 <testcase> 태그가 있습니다. 예:

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security
Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled
and following changes are made:Add CIS conn info and Add another CIS conn info"
attempts="1"></testcase>>
```

## <testcase> 태그에 사용되는 속성

### name

테스트의 이름입니다.

### attempts

IDT가 테스트 사례를 실행한 횟수.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
<failure type="Failure">Reason for the test failure</failure>
<error>Reason for the test execution error</error>
```

```
</testcase>
```

## 로그 보기

IDT는 `<devicetester-extract-location>/results/<execution-id>/logs`의 테스트 실행에서 로그를 생성합니다. 두 개의 로그 세트가 생성됩니다.

`test_manager.log`

AWS IoT Device Tester의 Test Manager 구성 요소(예: 구성 관련 로그, 테스트 시퀀싱, 보고서 생성)에서 생성된 로그입니다.

`<test-case-id>.log` (for example, `lambdaDeploymentTest.log`)

테스트 중인 디바이스의 로그를 포함하여 테스트 그룹 내 테스트 사례의 로그입니다. IDT v4.2.0 부터는 각 테스트 사례에 대한 테스트 로그를 `<devicetester-extract-location>/results/<execution-id>/logs/<test-group-id>/` 디렉터리 내의 별도의 `<test-case-id>` 폴더에 그룹화합니다.

## IDT를 사용하여 자체 테스트 제품군 개발 및 실행

IDT v4.0.1부터 IDT for AWS IoT Greengrass V2는 표준화된 구성 설정 및 결과 형식을 디바이스 및 디바이스 소프트웨어에 대한 사용자 지정 테스트 제품군을 개발할 수 있는 테스트 제품군 환경과 결합합니다. 사용자 지정 테스트를 자체 내부 검증을 위해 추가하거나 디바이스 검증을 위해 고객에게 제공할 수 있습니다.

IDT를 사용하여 다음과 같이 사용자 지정 테스트 도구 모음을 개발하고 실행하십시오.

사용자 지정 테스트 도구 모음을 개발하려면

- 테스트하려는 Greengrass 장치에 대해 사용자 지정 테스트 로직이 포함된 테스트 도구 모음을 만드세요.
- 테스트 러너에게 사용자 지정 테스트 도구 모음과 IDT를 제공하세요. 테스트 도구 모음의 특정 설정 구성에 대한 정보를 포함해야 합니다.

사용자 지정 테스트 도구 모음을 실행하려면

- 테스트하려는 디바이스를 설정합니다.
- 사용하려는 테스트 도구 모음의 필요에 따라 설정 구성을 구현하십시오.
- IDT를 사용하여 사용자 지정 테스트 도구 모음을 실행하세요.

- IDT에서 실행한 테스트의 테스트 결과 및 실행 로그를 확인합니다.

## 용의 최신 버전 다운로드 AWS IoT Device Tester AWS IoT Greengrass

[최신 버전](#)의 IDT를 다운로드하여 읽기/쓰기 권한이 있는 파일 시스템의 위치(<*device-tester-extract-location*>)에 소프트웨어의 압축을 풉니다.

### Note

여러 사용자가 NFS 디렉터리 또는 Windows 네트워크 공유 폴더와 같은 공유 위치에서 IDT를 실행하는 것은 지원되지 않습니다. 로컬 드라이브에 IDT 패키지의 압축을 풀고 로컬 워크스테이션에서 IDT 바이너리를 실행하는 것이 좋습니다.

Windows의 경우 260자의 경로 길이 제한이 있습니다. Windows를 사용하는 경우 경로를 260자 제한 아래로 유지하도록 IDT 압축을 C:\ 또는 D:\ 같은 루트 디렉터리에 풉니다.

## 테스트 도구 모음 생성 워크플로

테스트 도구 모음은 세 가지 유형의 파일로 구성됩니다.

- 테스트 제품군 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 생성](#)합니다.
2. 테스트 제품군의 테스트 로직이 포함된 [테스트 사례 실행 파일을 생성](#)합니다.
3. 테스트 도구 모음을 실행하는 데 [테스트 러너에게 필요한 구성 정보](#)를 확인하고 문서화하세요.
4. IDT가 예상대로 테스트 도구 모음을 실행하고 [테스트 결과](#)를 생성할 수 있는지 확인하십시오.

샘플 사용자 지정 도구 모음을 빠르게 구축하고 실행하려면 [튜토리얼: 샘플 IDT 테스트 도구 모음 구축 및 실행](#)의 지침을 따르십시오.

Python으로 사용자 지정 테스트 도구 모음 생성을 시작하려면 [튜토리얼: 간단한 IDT 테스트 도구 모음 개발](#)(를) 참조하십시오.



## 튜토리얼: 샘플 IDT 테스트 도구 모음 구축 및 실행

AWS IoT Device Tester 다운로드에는 샘플 테스트 제품군의 소스 코드가 포함됩니다. 이 자습서를 완료하여 샘플 테스트 제품군을 빌드하고 실행하여 용 IDT를 사용하여 사용자 지정 테스트 제품군 AWS IoT Greengrass 을 실행하는 방법을 이해할 수 있습니다.

이 자습서에서는 다음 단계를 완료합니다.

1. [샘플 테스트 도구 모음 구축](#)
2. [IDT를 사용하여 샘플 테스트 도구 모음을 실행하세요.](#)

### 사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - 의 최신 버전 AWS IoT Device Tester
  - [Python](#) 3.7 이상

컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

```
python3 --version
```

Windows에서 이 명령 사용시 오류가 반환되면 python --version을(를) 대신 사용하십시오. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 python 명령의 별칭으로 python3을(를) 설정합니다.

```
Set-Alias -Name "python3" -Value "python"
```

버전 정보가 반환되지 않았거나 버전 번호가 3.7 미만이면 [Python 다운로드](#)의 지침에 따라 Python 3.7 이상을 설치합니다. 자세한 내용은 [Python 설명서](#)를 참조하세요.

- [urllib3](#)

urllib3이 제대로 설치되었는지 확인하려면 다음 명령을 실행합니다.

```
python3 -c 'import urllib3'
```

urllib3가 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 디바이스입니다.

Raspberry Pi OS와 함께 [Raspberry Pi](#)를 사용하는 것이 좋습니다. Raspberry Pi에 원격으로 연결하려면 Pi에서 [SSH](#)를 설정해야 합니다.

## IDT용 디바이스 정보 구성

IDT가 테스트를 실행할 수 있도록 디바이스 정보를 구성하십시오. *<device-tester-extract-location>*/configs 폴더에 있는 device.json 템플릿을 다음 정보로 업데이트해야 합니다.

```
[
 {
 "id": "pool",
 "sku": "N/A",
 "devices": [
 {
 "id": "<device-id>",
 "connectivity": {
 "protocol": "ssh",
 "ip": "<ip-address>",
 "port": "<port>",
 "auth": {
 "method": "pki | password",
 "credentials": {
 "user": "<user-name>",
 "privKeyPath": "/path/to/private/key",
 "password": "<password>"
 }
 }
 }
 }
]
 }
]
```

devices 개체에 다음 정보를 제공하시기 바랍니다.

`id`

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

`connectivity.ip`

디바이스의 IP 주소입니다.

`connectivity.port`

선택 사항. 디바이스에 SSH 연결에 사용할 포트 번호입니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.user`

디바이스에 로그인하는 데 사용되는 사용자 이름.

`connectivity.auth.credentials.privKeyPath`

디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`devices.connectivity.auth.credentials.password`

디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

#### Note

`method`가 `pki`로 설정된 경우에만 `privKeyPath`를 지정합니다.  
`method`가 `password`로 설정된 경우에만 `password`를 지정합니다.

## 샘플 테스트 도구 모음 구축

`<device-tester-extract-location>/samples/python` 폴더에는 제공된 구축 스크립트를 사용하여 테스트 도구 모음에 결합할 수 있는 샘플 구성 파일, 소스 코드 및 IDT Client SDK가 포함되어 있습니다. 다음 디렉터리 트리는 이러한 샘플 파일의 위치를 보여줍니다.

```
<device-tester-extract-location>
...
tests
samples
...
python
configuration
src
build-scripts
build.sh
build.ps1
sdks
...
python
idt_client
```

테스트 도구 모음을 구축하려면 호스트 컴퓨터에서 다음 명령을 실행합니다.

### Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

### Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
```

```
./build.sh
```

그러면 IDTSampleSuitePython\_1.0.0 폴더 내 *<device-tester-extract-location>*/tests 폴더에 샘플 테스트 도구 모음이 만들어집니다. IDTSampleSuitePython\_1.0.0 폴더의 파일을 검토하여 샘플 테스트 도구 모음이 구조화되는 방식을 이해하고 테스트 사례 실행 파일 및 테스트 구성 JSON 파일의 다양한 예제를 참조하세요.

#### Note

샘플 테스트 제품군에는 python 소스 코드가 포함되어 있습니다. 테스트 제품군 코드에 민감한 정보를 포함하지 마십시오.

다음 단계: IDT를 사용하여 생성한 [샘플 테스트 제품군을 실행](#)합니다.

IDT를 사용하여 샘플 테스트 도구 모음을 실행하세요.

샘플 테스트 도구 모음을 실행하려면 호스트 컴퓨터에서 다음 명령을 실행하세요.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT는 샘플 테스트 도구 모음을 실행하고 결과를 콘솔로 스트리밍합니다. 테스트 실행이 완료되면 다음 정보가 표시됩니다.

```
===== Test Summary =====
Execution Time: 5s
Tests Completed: 4
Tests Passed: 4
Tests Failed: 0
Tests Skipped: 0

Test Groups:
 sample_group: PASSED

Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
```

```
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

## 문제 해결

다음 정보를 사용하면 튜토리얼 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않습니다

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 이 튜토리얼의 모든 [사전 요구](#) 사항을 충족하는지 확인하세요.

테스트 중인 디바이스에 연결할 수 없습니다.

다음을 확인합니다.

- device.json 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 장치에 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 자습서에서는 AWS IoT Greengrass 용 IDT를 사용하여 단일 테스트 사례가 포함된 Python 테스트 제품군을 개발하는 방법을 보여줍니다. 이 자습서에서는 다음 단계를 완료합니다.

1. [테스트 도구 모음 디렉터리 생성](#)
2. [구성 파일 생성](#)
3. [테스트 사례 실행 파일 생성](#)
4. [테스트 도구 모음 실행](#)

## 사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- 호스트 컴퓨터 요구 사항
  - 의 최신 버전 AWS IoT Device Tester
  - [Python](#) 3.7 이상

컴퓨터에 설치된 Python 버전 번호를 확인하려면 인스턴스에서 다음 명령을 실행합니다.

```
python3 --version
```

Windows에서 이 명령 사용시 오류가 반환되면 `python --version`을(를) 대신 사용하십시오. 반환된 버전 번호가 3.7 이상인 경우 Powershell 터미널에서 다음 명령을 실행하여 `python` 명령의 별칭으로 `python3`을(를) 설정합니다.

```
Set-Alias -Name "python3" -Value "python"
```

버전 정보가 반환되지 않았거나 버전 번호가 3.7 미만이면 [Python 다운로드](#)의 지침에 따라 Python 3.7 이상을 설치합니다. 자세한 내용은 [Python 설명서](#)를 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 디바이스입니다.

Raspberry Pi OS와 함께 [Raspberry Pi](#)를 사용하는 것이 좋습니다. Raspberry Pi에 원격으로 연결하려면 Pi에서 [SSH](#)를 설정해야 합니다.

## 테스트 도구 모음 디렉터리 생성

IDT는 각 테스트 도구 모음 내의 테스트 그룹에 테스트 사례를 논리적으로 분리합니다. 각 테스트 사례는 테스트 그룹 내에 있어야 합니다. 이 자습서에서는 MyTestSuite\_1.0.0이라는 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 파일](#)이 포함되어야 합니다.

### 필수 구성 파일

#### suite.json

테스트 제품군 정보가 포함되어 있습니다. [suite.json 구성](#)(를) 참조하세요.

#### group.json

테스트 그룹에 대한 정보를 포함합니다. 테스트 도구 모음의 각 테스트 그룹에 대한 group.json 파일을 만들어야 합니다. [group.json을 구성하십시오](#)(를) 참조하세요.

#### test.json

테스트 케이스에 대한 정보가 들어 있습니다. 테스트 도구 모음의 각 테스트 케이스에 대한 test.json 파일을 만들어야 합니다. [test.json을 구성하십시오](#)(를) 참조하세요.

1. MyTestSuite\_1.0.0/suite 폴더에서 다음 폴더 구조로 suite.json 파일을 안에 생성합니다.

```
{
 "id": "MyTestSuite_1.0.0",
 "title": "My Test Suite",
 "details": "This is my test suite.",
 "userDataRequired": false
}
```

2. MyTestSuite\_1.0.0/myTestGroup 폴더에서 다음 폴더 구조로 group.json 파일을 안에 생성합니다.



```
{
 "id": "MyTestGroup",
 "title": "My Test Group",
 "details": "This is my test group.",
 "optional": false
}
```

3. MyTestSuite\_1.0.0/myTestGroup/myTestCase 폴더에서 다음 폴더 구조로 test.json 파일을 안에 생성합니다.

```
{
 "id": "MyTestCase",
 "title": "My Test Case",
 "details": "This is my test case.",
 "execution": {
 "timeout": 300000,
 "linux": {
 "cmd": "python3",
 "args": [
 "myTestCase.py"
]
 },
 "mac": {
 "cmd": "python3",
 "args": [
 "myTestCase.py"
]
 },
 "win": {
 "cmd": "python3",
 "args": [
 "myTestCase.py"
]
 }
 }
}
```

이제 MyTestSuite\_1.0.0 폴더의 디렉터리 트리가 다음과 같이 표시되어야 합니다.

```
MyTestSuite_1.0.0
suite
```

```
suite.json
myTestGroup
 ### group.json
 ### myTestCase
 ### test.json
```

## IDT 클라이언트 SDK 다운로드

[IDT 클라이언트 SDK](#)를 사용하여 IDT가 테스트 대상 디바이스와 상호 작용하고 테스트 결과를 보고할 수 있도록 합니다. 이 튜토리얼에서는 Python 버전의 SDK를 사용합니다.

`<device-tester-extract-location>/sdks/python/` 폴더에서 `idt_client` 폴더를 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 폴더로 복사합니다.

SDK가 성공적으로 복사되었는지 확인하려면 다음 명령을 실행합니다.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

## 테스트 사례 실행 파일 생성

테스트 사례 실행 파일에는 실행하려는 테스트 로직이 포함되어 있습니다. 테스트 도구 모음에는 여러 테스트 사례 실행 파일이 포함될 수 있습니다. 이 튜토리얼에서는 하나의 테스트 사례 실행 파일을 생성합니다.

1. 테스트 도구 모음 파일을 만드세요.

`MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 폴더 안에 다음 내용으로 `myTestCase.py`라는 파일을 만듭니다.

```
from idt_client import *

def main():
 # Use the client SDK to communicate with IDT
 client = Client()

if __name__ == "__main__":
 main()
```

2. 클라이언트 SDK 함수를 사용하여 `myTestCase.py` 파일에 다음 테스트 로직을 추가합니다.

- a. 테스트 중인 디바이스에서 SSH 명령어를 실행합니다.

```
from idt_client import *

def main():
 # Use the client SDK to communicate with IDT
 client = Client()

 # Create an execute on device request
 exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
world'"))

 # Run the command
 exec_resp = client.execute_on_device(exec_req)

 # Print the standard output
 print(exec_resp.stdout)

if __name__ == "__main__":
 main()
```

b. 테스트 결과를 IDT로 전송합니다.

```
from idt_client import *

def main():
 # Use the client SDK to communicate with IDT
 client = Client()

 # Create an execute on device request
 exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
world'"))

 # Run the command
 exec_resp = client.execute_on_device(exec_req)

 # Print the standard output
 print(exec_resp.stdout)

 # Create a send result request
 sr_req = SendResultRequest(TestResult(passed=True))

 # Send the result
 client.send_result(sr_req)
```

```
if __name__ == "__main__":
 main()
```

## IDT용 디바이스 정보 구성

IDT가 테스트를 실행할 수 있도록 디바이스 정보를 구성하십시오. *<device-tester-extract-location>*/configs 폴더에 있는 device.json 템플릿을 다음 정보로 업데이트해야 합니다.

```
[
 {
 "id": "pool",
 "sku": "N/A",
 "devices": [
 {
 "id": "<device-id>",
 "connectivity": {
 "protocol": "ssh",
 "ip": "<ip-address>",
 "port": "<port>",
 "auth": {
 "method": "pki | password",
 "credentials": {
 "user": "<user-name>",
 "privKeyPath": "/path/to/private/key",
 "password": "<password>"
 }
 }
 }
 }
]
 }
]
```

devices 개체에 다음 정보를 제공하시기 바랍니다.

id

테스트 대상 디바이스의 고유한 사용자 정의 식별자입니다.

connectivity.ip

디바이스의 IP 주소입니다.

## `connectivity.port`

선택 사항. 디바이스에 SSH 연결에 사용할 포트 번호입니다.

## `connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

### `connectivity.auth.method`

지정된 연결 프로토콜을 통해 디바이스에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- `pki`
- `password`

### `connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

#### `connectivity.auth.credentials.user`

디바이스에 로그인하는 데 사용되는 사용자 이름.

#### `connectivity.auth.credentials.privKeyPath`

디바이스에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

#### `devices.connectivity.auth.credentials.password`

디바이스에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

#### Note

`method`가 `pki`로 설정된 경우에만 `privKeyPath`를 지정합니다.  
`method`가 `password`로 설정된 경우에만 `password`를 지정합니다.

## 테스트 도구 모음 실행

테스트 도구 모음을 만든 후에는 예상대로 작동하는지 확인해야 합니다. 이를 위해 기존 디바이스 플로 테스트 도구 모음을 실행하려면 다음 단계를 완료하세요.

1. MyTestSuite\_1.0.0 폴더를 *<device-tester-extract-location>*/tests에 복사하세요.
2. 다음 명령을 실행합니다.

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT는 테스트 도구 모음을 실행하고 결과를 콘솔로 스트리밍합니다. 테스트 실행이 완료되면 다음 정보가 표시됩니다.

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

===== Test Summary =====
Execution Time: 1s
Tests Completed: 1
Tests Passed: 1
Tests Failed: 0
Tests Skipped: 0

Test Groups:
 myTestGroup: PASSED

Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
```

```
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

## 문제 해결

다음 정보를 사용하면 튜토리얼 완료와 관련된 문제를 해결하는 데 도움이 됩니다.

테스트 케이스가 성공적으로 실행되지 않습니다

테스트가 성공적으로 실행되지 않을 경우 IDT는 오류 로그를 콘솔로 스트리밍하여 테스트 실행 문제를 해결하는 데 도움을 줍니다. 오류 로그를 확인하기 전에 다음 사항을 확인합니다.

- IDT 클라이언트 SDK는 [이 단계](#)에서 설명한 대로 올바른 폴더에 있습니다.
- 이 튜토리얼의 모든 [사전 요구 사항](#)을 충족합니다.

테스트 중인 디바이스에 연결할 수 없습니다.

다음을 확인합니다.

- device.json 파일에는 올바른 IP 주소, 포트 및 인증 정보가 들어 있습니다.
- 호스트 컴퓨터에서 SSH를 통해 장치에 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](#)(를) 참조하세요.

group.json

테스트 그룹에 대한 정보를 포함합니다. 테스트 도구 모음의 각 테스트 그룹에 대한 group.json 파일을 만들어야 합니다. [group.json을 구성하십시오](#)(를) 참조하세요.

test.json

테스트 케이스에 대한 정보가 들어 있습니다. 테스트 도구 모음의 각 테스트 케이스에 대한 test.json 파일을 만들어야 합니다. [test.json을 구성하십시오](#)(를) 참조하세요.

## 선택적 구성 파일

test\_orchestrator.yaml or 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 파일](#)의 스키마를 정의합니다. 이 userdata.json 파일은 테스트를 실행하는 데 필요하지만 device.json 파일에는 없는 추가 구성 정보에 사용됩니다. [userdata\\_schema.json 구성](#)(를) 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 도구 모음을 실행하는 데 사용자 데이터가 필요한지 여부를 결정합니다. 다음 템플릿을 사용하여 *<custom-test-suite-folder>/suite/suite.json* 파일을 구성하십시오.

```
{
```



```

 "id": "<suite-name>_<suite-version>",
 "title": "<suite-title>",
 "details": "<suite-details>",
 "userDataRequired": true | false,
 "environmentVariables": [
 {
 "key": "<name>",
 "value": "<value>",
 },
 ...
 {
 "key": "<name>",
 "value": "<value>",
 }
]
 }
}

```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

## id

테스트 도구 모음의 고유한 사용자 정의 ID. id의 값은 suite.json 파일이 있는 테스트 도구 모음 폴더의 이름과 일치해야 합니다. 세트 이름과 세트 버전도 다음 요구 사항을 충족해야 합니다.

- **<suite-name>**은(는) 언더바를 포함할 수 없습니다.
- **<suite-version>**은(는) 다음과 같이 **x.x.x**(으)로 표시됩니다. 여기서 x은(는) 숫자입니다.

ID는 IDT에서 생성한 테스트 보고서에 표시됩니다.

## title

이 테스트 도구 모음에서 테스트 중인 제품 또는 기능의 사용자 정의 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

## details

테스트 도구 모음의 용도에 대한 짧은 설명입니다.

## userDataRequired

테스트 실행기가 userdata.json 파일에 사용자 지정 정보를 포함해야 하는지 여부를 정의합니다. 이 값을 true(로) 설정하는 경우, 테스트 도구 모음 폴더에도 [userdata\\_schema.json 파일](#)을 포함해야 합니다.

## environmentVariables

선택 사항입니다. 이 테스트 도구 모음에 설정할 환경 변수 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

## group.json을 구성하십시오.

`group.json` 파일은 테스트 그룹이 필수인지 옵션인지 여부를 정의합니다. 다음 템플릿을 사용하여 `<custom-test-suite-folder>/suite/<test-group>/group.json` 파일을 구성하십시오.

```
{
 "id": "<group-id>",
 "title": "<group-title>",
 "details": "<group-details>",
 "optional": true | false,
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

### id

테스트 그룹의 고유한 사용자 정의 ID입니다. `id`의 값은 `group.json` 파일이 있는 테스트 그룹 폴더의 이름과 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 사항입니다. IDT에서 필수 테스트 실행을 완료한 후 이 테스트 그룹을 선택적 그룹으로 표시하도록 `true`(으)로 설정합니다. 기본값은 `false`입니다.

## test.json을 구성하십시오.

test.json 파일은 테스트 케이스 실행 파일 및 테스트 케이스에서 사용되는 환경 변수를 결정합니다. 테스트 케이스 실행 파일 생성에 대한 자세한 내용은 [IDT 테스트 케이스 실행 파일 생성](#) 단원을 참조하세요.

다음 템플릿을 사용하여 `<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json` 파일을 구성하십시오.

```
{
 "id": "<test-id>",
 "title": "<test-title>",
 "details": "<test-details>",
 "requireDUT": true | false,
 "requiredResources": [
 {
 "name": "<resource-name>",
 "features": [
 {
 "name": "<feature-name>",
 "version": "<feature-version>",
 "jobSlots": <job-slots>
 }
]
 }
],
 "execution": {
 "timeout": <timeout>,
 "mac": {
 "cmd": "<path/to/executable>",
 "args": [
 "<argument>"
],
 },
 "linux": {
 "cmd": "<path/to/executable>",
 "args": [
 "<argument>"
],
 },
 "win": {
 "cmd": "<path/to/executable>",
 "args": [
```

```

 "<argument>"
]
}
},
"environmentVariables": [
 {
 "key": "<name>",
 "value": "<value>",
 }
]
}

```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

#### id

테스트 사례의 고유한 사용자 정의 ID입니다. id의 값은 test.json 파일이 있는 테스트 사례 폴더의 이름과 일치해야 하며 밑줄(\_)을 포함할 수 없습니다. ID는 IDT에서 생성한 테스트 보고서에 사용됩니다.

#### title

테스트 케이스를 나타내는 서술형 이름입니다. 테스트 실행기의 IDT CLI에 이름이 표시됩니다.

#### details

테스트 케이스의 용도에 대한 짧은 설명입니다.

#### requireDUT

선택 사항입니다. 이 테스트를 실행하는 데 기기가 필요한 경우, true(으)로 설정하고, 그렇지 않으면 false(으)로 설정합니다. 기본값은 true입니다. 테스트 실행기는 device.json 파일에서 테스트를 실행하는 데 사용할 기기를 구성합니다.

#### requiredResources

선택 사항입니다. 이 테스트를 실행하는 데 필요한 리소스 기기에 대한 정보를 제공하는 배열입니다.

##### requiredResources.name

이 테스트를 실행할 때 리소스 기기에 부여하는 고유한 이름입니다.

##### requiredResources.features

사용자 정의 리소스 장치 기능의 배열.

`requiredResources.features.name`

기능의 이름입니다. 이 장치를 사용하려는 장치 기능. 이 이름은 테스트 실행기가 `resource.json` 파일에 제공한 기능 이름과 일치합니다.

`requiredResources.features.version`

선택 사항입니다. 기능의 버전. 이 값은 테스트 실행기가 `resource.json` 파일에서 제공한 기능 버전과 일치합니다. 버전이 제공되지 않으면 기능이 확인되지 않습니다. 기능에 버전 번호가 필요하지 않은 경우, 이 필드를 비워 둡니다.

`requiredResources.features.jobSlots`

선택 사항입니다. 이 기능이 지원할 수 있는 동시 테스트 수입니다. 기본값은 1입니다. IDT에서 개별 기능에 대해 서로 다른 기기를 사용하도록 하려면 이 값을 1(으)로 설정하는 것이 좋습니다.

`execution.timeout`

IDT가 테스트 실행이 완료될 때까지 기다리는 시간(밀리초)입니다. 이 값 설정에 대한 자세한 내용을 알아보려면 [IDT 테스트 케이스 실행 파일 생성](#) 섹션을 참조하세요.

`execution.os`

IDT를 실행하는 호스트 컴퓨터의 운영 체제에 따라 실행할 테스트 케이스 실행 파일입니다. 지원되는 값은 linux, mac 및 win입니다.

`execution.os.cmd`

지정된 운영 체제에서 실행하려는 테스트 케이스 실행 파일의 경로입니다. 이 위치는 시스템 경로에 있어야 합니다.

`execution.os.args`

선택 사항입니다. 테스트 케이스 실행 파일을 실행하기 위해 제공할 인수입니다.

`environmentVariables`

선택 사항입니다. 이 테스트 케이스에 설정된 환경 변수 배열입니다.

`environmentVariables.key`

환경 변수의 이름입니다.

`environmentVariables.value`

환경 변수의 값입니다.

**Note**

test.json 파일과 suite.json 파일에서 동일한 환경 변수를 지정하는 경우, test.json 파일의 값이 우선합니다.

## 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 도구 모음에 device.json 파일에 없는 정보가 필요한 경우, 사용자 데이터가 필요합니다. 예를 들어 테스트에는 Wi-Fi 네트워크 자격 증명, 특정 오픈 포트 또는 사용자가 제공해야 하는 인증서가 필요할 수 있습니다. 이 정보는 userdata(이)라는 입력 파라미터로 IDT에 제공될 수 있습니다. 이때 값은 `<device-tester-extract-location>/config` 폴더에 생성한 userdata.json 파일입니다. userdata.json 파일 형식은 테스트 도구 모음에 포함된 userdata\_schema.json 파일을 기반으로 합니다.

테스트 실행기가 userdata.json 파일을 제공해야 함을 나타내려면:

1. suite.json 파일에서 userDataRequired을(를) true(으)로 설정합니다.
2. *<custom-test-suite-folder>*에서 userdata\_schema.json 파일을 생성하십시오.
3. userdata\_schema.json 파일을 편집하여 유효한 [IETF 초안 v4 JSON 스키마](#)를 생성하십시오.

IDT는 테스트 제품군을 실행하면 자동으로 스키마를 읽고 이를 사용하여 테스트 실행기가 제공한 userdata.json 파일을 검증합니다. 유효한 경우, userdata.json 파일의 내용은 [IDT 컨텍스트](#)와 [테스트 오케스트레이터 컨텍스트](#) 모두에서 사용할 수 있습니다.

## IDT 테스트 오케스트레이터 구성

IDT v4.5.1부터 IDT에는 새로운 테스트 오케스트레이터 구성 요소가 포함됩니다. 테스트 오케스트레이터는 테스트 제품군 실행 흐름을 제어하고 IDT가 모든 테스트 실행을 완료한 후 테스트 보고서를 생성하는 IDT 구성 요소입니다. 테스트 오케스트레이터는 사용자 정의 규칙을 기반으로 테스트 선택 및 테스트 실행 순서를 결정합니다.

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.

테스트 오케스트레이터는 IDT 테스트 오케스트레이터를 대체합니다. 테스트 제품군을 개발할 때 IDT 테스트 오케스트레이터 대신 테스트 오케스트레이터를 사용하는 것이 좋습니다. 테스트 오케스트레이터는 다음과 같은 향상된 기능을 제공합니다.

- IDT 상태 시스템이 사용하는 명령형 형식이 아닌 선언적 형식을 사용합니다. 이를 통해 실행할 테스트 및 실행 시기를 지정할 수 있습니다.
- 특정 그룹 처리, 보고서 생성, 오류 처리 및 결과 추적을 관리하므로 이러한 작업을 수동으로 관리할 필요가 없습니다.
- 기본적으로 주석을 지원하는 YAML 형식을 사용합니다.

- 동일한 워크플로를 정의하는 데 테스트 오케스트레이터보다 80% 적은 디스크 공간이 필요합니다.
- 사전 테스트 검증을 추가하여 워크플로 정의에 잘못된 테스트 ID 또는 순환 종속성이 포함되어 있지 않은지 확인합니다.

## 테스트 오케스트레이터 형식

다음 템플릿을 사용하여 `<custom-test-suite-folder>/suite/test_orchestrator.yaml` 파일을 직접 구성할 수 있습니다.

```
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 사항입니다. 컨텍스트 표현식에 매핑되는 사용자 정의 문자열입니다. 별칭을 사용하면 테스트 오케스트레이터 구성의 컨텍스트 표현식을 식별하기 위한 표시 이름을 생성할 수 있습니다. 이는 복잡한 컨텍스트 표현식이나 여러 위치에서 사용하는 표현식을 생성할 때 특히 유용합니다.

컨텍스트 표현식을 사용하여 다른 IDT 구성의 데이터에 액세스할 수 있는 컨텍스트 쿼리를 저장할 수 있습니다. 자세한 내용은 [컨텍스트에서 데이터 액세스](#) 섹션을 참조하세요.



## Example예시

### Aliases:

```
FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

## ConditionalTests

선택 사항입니다. 조건 목록 및 각 조건이 충족될 때 실행되는 해당 테스트 사례입니다. 각 조건에는 여러 테스트 사례가 있을 수 있지만 특정 테스트 사례를 하나의 조건에만 할당할 수 있습니다.

기본적으로 IDT는 이 목록의 조건에 할당되지 않은 테스트 사례를 모두 실행합니다. 이 섹션을 지정하지 않으면 IDT가 테스트 제품군의 모든 테스트 그룹을 실행합니다.

ConditionalTests 목록의 각 항목에는 다음 파라미터가 포함되어 있습니다.

### Condition

부울 값으로 평가되는 컨텍스트 표현식입니다. 평가된 값이 true인 경우 IDT는 Tests 파라미터에 지정된 테스트 사례를 실행합니다.

### Tests

테스트 설명자의 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
```

## Example예시

다음 예제에서는 Aliases와 같이 정의할 수 있는 일반 컨텍스트 표현식을 사용합니다.

```
ConditionalTests:
 - Condition: "{{$aliases.Condition1}}"
 Tests:
 - GroupId: A
 - GroupId: B
 - Condition: "{{$aliases.Condition2}}"
 Tests:
 - GroupId: D
```

```
- Condition: "{{${aliases.Condition1}}} || {{${aliases.Condition2}}}"
Tests:
 - GroupId: C
```

IDT는 정의된 조건에 따라 다음과 같이 테스트 그룹을 선택합니다.

- Condition1이 true인 경우 IDT는 테스트 그룹 A, B 및 C의 테스트를 실행합니다.
- Condition2이 true인 경우 IDT는 테스트 그룹 C 및 D의 테스트를 실행합니다.

## Order

선택 사항입니다. 테스트를 실행하는 순서입니다. 테스트 순서는 테스트 그룹 수준에서 지정합니다. 이 섹션을 지정하지 않으면 IDT는 해당하는 모든 테스트 그룹을 무작위 순서로 실행합니다. Order의 값은 그룹 설명자 목록의 목록입니다. Order 목록에 없는 모든 테스트 그룹은 나열된 다른 테스트 그룹과 병렬로 실행할 수 있습니다.

각 그룹 설명자 목록에는 하나 이상의 그룹 설명자가 포함되며 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*] - 서로 임의의 순서로 실행할 수 있는 테스트 그룹의 목록입니다.
- "\*" - 와일드카드입니다. 이는 현재 그룹 설명자 목록에 아직 지정되지 않은 모든 테스트 그룹의 목록과 동일합니다.

Order의 값은 다음 요구 사항도 충족해야 합니다.

- 그룹 설명자에서 지정하는 테스트 그룹 ID가 테스트 제품군에 있어야 합니다.
- 각 그룹 설명자 목록에는 테스트 그룹이 하나 이상 포함되어야 합니다.
- 각 그룹 설명자 목록에는 고유한 그룹 ID가 포함되어야 합니다. 개별 그룹 설명자 내에서 테스트 그룹 ID를 반복해서 사용할 수 없습니다.
- 그룹 설명자 목록에는 와일드카드 그룹 설명자가 최대 하나만 포함될 수 있습니다. 와일드카드 그룹 설명자는 목록의 첫 번째 또는 마지막 항목이어야 합니다.

## Example예시

테스트 그룹 A, B, C, D, E가 포함된 테스트 제품군의 경우 다음 예제 목록은 IDT가 먼저 테스트 그룹 A 및 테스트 그룹 B를 순서대로 실행한 다음 테스트 그룹 C, D, E를 순서에 상관없이 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 사항입니다. IDT가 `awsiotdevicetester_report.xml` 파일에 추가하려는 제품 기능의 목록입니다. 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 디바이스가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. `awsiotdevicetester_report.xml`에서는 지정된 테스트의 통과 여부에 따라 제품 기능이 `supported`, `not-supported` 또는 사용자 정의 값으로 설정됩니다.

Features 목록의 각 항목은 다음 파라미터로 구성됩니다.

Name

기능의 이름입니다.

Value

선택 사항입니다. 보고서에서 `supported` 대신 사용할 사용자 지정 값입니다. 이 값을 지정하지 않으면 IDT가 테스트 결과에 따라 기능 값을 `supported` 또는 `not-supported`로 설정합니다. 동일한 기능을 다른 조건으로 테스트하는 경우, Features 목록에 있는 기능의 각 인스턴스에 대해 사용자 지정 값을 사용할 수 있으며, IDT는 지원되는 조건에 대해 기능 값을 연결합니다. 자세한 내용을 알아보려면 다음 섹션을 참조하세요.

Condition

부울 값으로 평가되는 컨텍스트 표현식입니다. 평가된 값이 true인 경우 IDT는 테스트 제품군 실행을 완료한 후 테스트 보고서에 기능을 추가합니다. 평가된 값이 false인 경우 테스트가 보고서에 포함되지 않습니다.

Tests

선택 사항입니다. 테스트 설명자의 목록입니다. 기능이 지원되려면 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
```

Features 목록의 각 기능에 대해 Tests 또는 OneOfTests 중 하나를 지정해야 합니다.

OneOfTests

선택 사항입니다. 테스트 설명자의 목록입니다. 기능이 지원되려면 이 목록에 지정된 테스트를 하나 이상 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
```

Features 목록의 각 기능에 대해 Tests 또는 OneOfTests 중 하나를 지정해야 합니다.

IsRequired

테스트 보고서에서 기능이 필요한지 여부를 정의하는 부울 값입니다. 기본값은 false입니다.

Example

테스트 오케스트레이터 컨텍스트

테스트 오케스트레이터 컨텍스트는 실행 중에 테스트 오케스트레이터가 사용할 수 있는 데이터가 포함된 읽기 전용 JSON 문서입니다. 테스트 오케스트레이터 컨텍스트는 테스트 오케스트레이터에서만

액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어 `userdata.json` 파일에서 테스트 실행기가 구성한 정보를 사용하여 특정 테스트 실행이 필요한지 여부를 결정할 수 있습니다.

테스트 오케스트레이터 컨텍스트는 다음 형식을 사용합니다.

```
{
  "pool": {
    <device-json-pool-element>
  },
  "userData": {
    <userdata-json-content>
  },
  "config": {
    <config-json-content>
  }
}
```

pool

테스트 실행을 위해 선택한 디바이스 풀에 대한 정보입니다. 선택한 장치 풀의 경우 이 정보는 `device.json` 파일에 정의된 해당 최상위 장치 풀 배열 요소에서 검색됩니다.

userData

`userdata.json` 파일에 있는 정보입니다.

config

`config.json` 파일에 있는 정보입니다.

JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의의 JSONPath 쿼리 구문은 `{{query}}`입니다. 테스트 오케스트레이터 컨텍스트에서 데이터에 액세스할 때는 각 값이 문자열, 숫자 또는 부울로 평가되어야 합니다.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 [IDT 컨텍스트 사용](#) 섹션을 참조하십시오.

IDT 상태 머신 구성

⚠ Important

IDT v4.5.1부터 이 상태 시스템은 더 이상 사용되지 않습니다. 새 테스트 오케스트레이터를 사용하는 것이 가장 좋습니다. 자세한 내용은 [IDT 테스트 오케스트레이터 구성](#) 단원을 참조하십시오.

상태 시스템은 테스트 제품군 실행 흐름을 제어하는 구조입니다. 테스트 도구 모음의 시작 상태를 결정하고, 사용자 정의 규칙을 기반으로 상태 전환을 관리하며, 최종 상태에 도달할 때까지 해당 상태를 계속 전환합니다.

테스트 제품군에 사용자 정의 상태 머신이 포함되지 않은 경우 IDT가 상태 머신을 자동으로 생성합니다. 기본 상태 머신은 다음 기능을 수행합니다.

- 테스트 실행기에 전체 테스트 제품군 대신 특정 테스트 그룹을 선택하고 실행할 수 있는 기능을 제공합니다.
- 특정 테스트 그룹을 선택하지 않은 경우, 테스트 제품군의 모든 테스트 그룹을 무작위 순서로 실행합니다.
- 보고서를 생성하고 각 테스트 그룹 및 테스트 사례의 테스트 결과를 보여주는 콘솔 요약을 인쇄합니다.

IDT 테스트 제품군의 상태 머신은 다음 기준을 충족해야 합니다.

- 각 상태는 테스트 그룹 실행 또는 보고서 파일 생성과 같이 IDT가 취할 수 있는 작업에 해당합니다.
- 상태로 전환하면 상태와 관련된 작업이 실행됩니다.
- 각 상태는 다음 상태의 전환 규칙을 정의합니다.
- 종료 상태는 Succeed 또는 Fail 중 하나여야 합니다.

상태 머신 형식

다음 템플릿을 사용하여 `<custom-test-suite-folder>/suite/state_machine.json` 파일을 직접 구성할 수 있습니다.

```
{
  "Comment": "<description>",
```

```

"StartAt": "<state-name>",
"States": {
  "<state-name>": {
    "Type": "<state-type>",
    // Additional state configuration
  }

  // Required states
  "Succeed": {
    "Type": "Succeed"
  },
  "Fail": {
    "Type": "Fail"
  }
}
}

```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Comment

상태 머신의 설명입니다.

StartAt

IDT가 테스트 제품군을 실행하기 시작하는 상태의 이름. StartAt의 값은 States 객체에 나열된 상태 중 하나로 설정해야 합니다.

States

사용자 정의 상태 이름을 유효한 IDT 상태에 매핑하는 객체입니다. 각 상태 *state-name* 객체에는 상태 *state-name*에 매핑된 유효한 상태의 정의가 포함됩니다.

States 객체에는 Succeed 및 Fail 상태가 포함되어야 합니다. 유효한 상태에 대한 자세한 내용은 [유효한 상태 및 상태 정의](#) 섹션을 참조하세요.

유효한 상태 및 상태 정의

이 섹션에서는 IDT 상태 머신에서 사용할 수 있는 모든 유효한 상태의 상태 정의를 설명합니다. 다음 상태 중 일부는 테스트 케이스 수준의 구성을 지원합니다. 하지만 꼭 필요한 경우가 아니면 테스트 케이스 수준 대신 테스트 그룹 수준에서 상태 전환 규칙을 구성하는 것이 좋습니다.

상태 정의

- [RunTask](#)
- [Choice](#)
- [Parallel](#)
- [AddProductFeatures](#)
- [Report](#)
- [LogMessage](#)
- [SelectGroup](#)
- [Fail](#)
- [Succeed](#)

RunTask

RunTask 상태는 테스트 제품군에 정의된 테스트 그룹에서 테스트 케이스를 실행합니다.

```
{
  "Type": "RunTask",
  "Next": "<state-name>",
  "TestGroup": "<group-id>",
  "TestCases": [
    "<test-id>"
  ],
  "ResultVar": "<result-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

TestGroup

선택 사항. 실행할 테스트 그룹의 ID. 이 값을 지정하지 않으면 IDT는 테스트 실행기가 선택한 테스트 그룹을 실행합니다.

TestCases

선택 사항. TestGroup에서 지정한 그룹의 테스트 케이스 ID 배열. IDT는 TestGroup 및 TestCases 값을 기반으로 다음과 같이 테스트 실행 동작을 결정합니다.

- TestGroup과 TestCases가 모두 지정된 경우 IDT는 테스트 그룹에서 지정된 테스트 케이스를 실행합니다.
- TestCases가 지정되었지만 TestGroup이 지정되지 않은 경우 IDT는 지정된 테스트 케이스를 실행합니다.
- TestGroup이 지정되었지만 TestCases가 지정되지 않은 경우 IDT는 지정된 테스트 그룹 내의 모든 테스트 케이스를 실행합니다.
- TestGroup 또는 TestCases 둘 다 지정되지 않은 경우 IDT는 테스트 실행기가 IDT CLI에서 선택한 테스트 그룹에서 모든 테스트 케이스를 실행합니다. 테스트 실행기에 대한 그룹 선택을 활성화하려면 state_machine.json 파일에 RunTask 및 Choice 상태를 모두 포함해야 합니다. 작동 방식에 대한 예제는 [상태 시스템 예시: 사용자가 선택한 테스트 그룹 실행](#)을 참조하십시오.

테스트 실행기의 IDT CLI 명령 활성화에 대한 자세한 내용은 [the section called “IDT CLI 명령을 활성화합니다.”](#) 섹션을 참조하세요.

ResultVar

테스트 실행 결과와 함께 설정할 컨텍스트 변수의 이름. TestGroup에 대한 값을 지정하지 않은 경우 이 값을 지정하지 마십시오. IDT는 다음에 따라 ResultVar에서 true 또는 false로 정의한 변수 값을 설정합니다.

- 변수 이름의 `text_text_passed` 형식인 경우 값은 첫 번째 테스트 그룹의 모든 테스트를 통과했는지 아니면 건너뛰었는지로 설정됩니다.
- 다른 모든 경우에는 값이 모든 테스트 그룹의 모든 테스트를 통과했는지 아니면 건너뛰었는지로 설정됩니다.

일반적으로 RunTask 상태를 사용하여 개별 테스트 케이스 ID를 지정하지 않고 테스트 그룹 ID를 지정하면 IDT가 지정된 테스트 그룹의 모든 테스트 케이스를 실행하게 됩니다. 이 상태에서 실행되는 모든 테스트 케이스는 무작위 순서로 병렬로 실행됩니다. 그러나 모든 테스트 케이스를 실행하는 데 디바이스가 필요하고 사용 가능한 디바이스가 하나뿐인 경우에는 테스트 케이스가 대신 순차적으로 실행됩니다.

오류 처리

지정된 테스트 그룹 또는 테스트 케이스 ID가 유효하지 않은 경우 이 상태에서는 RunTaskError 실행 오류가 발생합니다. 상태에서 실행 오류가 발생하는 경우 상태 머신 컨텍스트의 hasExecutionError 변수도 true로 설정됩니다.

Choice

Choice 상태를 사용하면 사용자 정의 조건에 따라 전환할 다음 상태를 동적으로 설정할 수 있습니다.

```
{
  "Type": "Choice",
  "Default": "<state-name>",
  "FallthroughOnError": true | false,
  "Choices": [
    {
      "Expression": "<expression>",
      "Next": "<state-name>"
    }
  ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Default

Choices에 정의된 표현식을 true로 평가할 수 없는 경우 전환할 기본 상태입니다.

FallthroughOnError

선택 사항. 상태에서 표현식을 평가할 때 오류가 발생할 경우의 동작을 지정합니다. 평가 결과 오류가 발생할 경우 표현식을 건너뛰려면 true로 설정합니다. 일치하는 표현식이 없는 경우 상태 머신은 해당 Default 상태로 전환됩니다. false 값이 지정하지 않은 경우 기본값은 FallthroughOnError입니다.

Choices

현재 상태에서 동작을 실행한 후 전환할 상태를 결정하는 표현식 및 상태의 배열입니다.

Choices.Expression

부울 값으로 평가되어야 하는 표현식 문자열입니다. 표현식이 true로 평가되면 상태 머신은 Choices.Next에 정의된 상태로 전환됩니다. 표현식 문자열은 상태 시스템 컨텍스트에서 값을 검색한 다음 해당 값에 대한 연산을 수행하여 부울 값에 도달합니다. 상태 머신 컨텍스트 액세스에 대한 자세한 내용은 [상태 머신 컨텍스트](#) 섹션을 참조하세요.

Choices.Next

Choices.Expression에 정의된 표현식이 true로 평가되면 전환할 상태의 이름.

오류 처리

Choice 상태는 다음과 같은 경우에 오류 처리를 요구할 수 있습니다.

- 선택 표현식의 일부 변수는 상태 머신 컨텍스트에 존재하지 않습니다.
- 표현식의 결과는 부울 값이 아닙니다.
- JSON 조회 결과는 문자열, 숫자 또는 부울이 아닙니다.

이 상태에서는 Catch 블록을 사용하여 오류를 처리할 수 없습니다. 오류가 발생했을 때 상태 머신 실행을 중지하려면 FallthroughOnError를 false로 설정해야 합니다. 하지만 사용 사례에 따라 다음 중 하나를 수행하도록 FallthroughOnError를 true로 설정하는 것이 좋습니다.

- 액세스하는 변수가 존재하지 않을 것으로 예상되는 경우가 있다면, Default의 값과 추가 Choices 블록을 사용하여 다음 상태를 지정하십시오.
- 액세스 중인 변수가 항상 존재해야 하는 경우 Default 상태를 Fail로 설정하십시오.

Parallel

Parallel 상태를 사용하면 새 상태 머신을 서로 병렬로 정의하고 실행할 수 있습니다.

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Branches": [
    <state-machine-definition>
  ]
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

Branches

실행할 상태 머신 정의의 배열입니다. 각 스테이트 머신 정의에는 고유한 StartAt, Succeed, Fail 상태가 포함되어야 합니다. 이 배열의 상태 머신 정의는 자체 정의 이외의 상태를 참조할 수 없습니다.

Note

각 브랜치 상태 머신은 동일한 상태 머신 컨텍스트를 공유하므로 한 브랜치에서 변수를 설정한 다음 다른 브랜치에서 해당 변수를 읽으면 예상치 못한 동작이 발생할 수 있습니다.

브랜치 상태 머신을 모두 실행한 후에만 Parallel 상태가 다음 상태로 이동합니다. 디바이스가 필요한 각 상태는 디바이스를 사용할 수 있을 때까지 실행을 대기합니다. 여러 디바이스를 사용할 수 있는 경우 이 상태는 여러 그룹의 테스트 케이스를 병렬로 실행합니다. 사용할 수 있는 디바이스가 충분하지 않으면 테스트 케이스가 순차적으로 실행됩니다. 테스트 케이스는 병렬로 실행될 때 무작위 순서로 실행되므로 동일한 테스트 그룹에서 테스트를 실행하는 데 여러 디바이스가 사용될 수 있습니다.

오류 처리

실행 오류를 처리하려면 브랜치 상태 머신과 부모 상태 머신이 모두 해당 Fail 상태로 전환되는지 확인하십시오.

브랜치 상태 머신은 부모 상태 머신에 실행 오류를 전송하지 않으므로 Catch 블록을 사용하여 브랜치 상태 머신의 실행 오류를 처리할 수 없습니다. 대신 공유 상태 머신 컨텍스트의 `hasExecutionErrors` 값을 사용하십시오. 이렇게 하는 방법의 예는 [상태 머신 예제: 두 테스트 그룹을 병렬로 실행](#) 섹션을 참조하세요.

AddProductFeatures

AddProductFeatures 상태에서는 IDT에서 생성한 `awsiotdevicetester_report.xml` 파일에 제품 기능을 추가할 수 있습니다.

제품 기능은 디바이스가 충족할 수 있는 특정 기준에 대한 사용자 정의 정보입니다. 예를 들어, MQTT 제품 기능은 디바이스가 MQTT 메시지를 올바르게 게시하도록 지정할 수 있습니다. 보고서에서 제품 기능은 지정된 테스트의 통과 여부에 따라 `supported`, `not-supported` 또는 사용자 지정 값으로 설정됩니다.

Note

AddProductFeatures 상태는 자체적으로 보고서를 생성하지 않습니다. 보고서를 생성하려면 이 상태를 [Report 상태로](#) 전환해야 합니다.

```
{
```

```

    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
      {
        "Feature": "<feature-name>",
        "Groups": [
          "<group-id>"
        ],
        "OneOfGroups": [
          "<group-id>"
        ],
        "TestCases": [
          "<test-id>"
        ],
        "IsRequired": true | false,
        "ExecutionMethods": [
          "<execution-method>"
        ]
      }
    ]
  }
}

```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

Features

awsiotdevicetester_report.xml 파일에 표시할 제품 기능의 배열.

Feature

기능의 이름입니다.

FeatureValue

선택 사항. 보고서에서 supported 대신 사용할 사용자 지정 값입니다. 이 값을 지정하지 않으면 테스트 결과에 따라 기능 값이 supported 또는 not-supported로 설정됩니다.

FeatureValue에 사용자 지정 값을 사용하면 동일한 기능을 다른 조건으로 테스트할 수 있으며, IDT는 지원되는 조건에 대한 기능 값을 연결합니다. 예를 들어, 다음 발췌문은 두 개의 개별 기능 값이 있는 MyFeature 기능을 보여줍니다.

```

...
{
  "Feature": "MyFeature",
  "FeatureValue": "first-feature-supported",
  "Groups": ["first-feature-group"]
},
{
  "Feature": "MyFeature",
  "FeatureValue": "second-feature-supported",
  "Groups": ["second-feature-group"]
},
...

```

두 테스트 그룹이 모두 통과하면 기능 값이 `first-feature-supported`, `second-feature-supported`로 설정됩니다.

Groups

선택 사항. 테스트 그룹 ID의 배열입니다. 기능을 지원하려면 지정된 각 테스트 그룹 내의 모든 테스트를 통과해야 합니다.

OneOfGroups

선택 사항. 테스트 그룹 ID의 배열입니다. 기능이 지원되려면 지정된 테스트 그룹 중 하나 이상의 모든 테스트를 통과해야 합니다.

TestCases

선택 사항. 테스트 케이스 ID의 배열입니다. 이 값을 지정하면 다음이 적용됩니다.

- 기능이 지원되려면 지정된 모든 테스트 케이스를 통과해야 합니다.
- `Groups`은 테스트 그룹 ID는 하나만 포함해야 합니다.
- `OneOfGroups`은 지정하지 않아야 합니다.

IsRequired

선택 사항. 보고서에서 이 기능을 선택적 기능으로 표시하려면 `false`로 설정합니다. 기본값은 `true`입니다.

ExecutionMethods

선택 사항. `device.json` 파일에 지정된 `protocol` 값과 일치하는 실행 메서드의 배열. 이 값을 지정하는 경우 테스트 실행기는 이 배열의 값 중 하나와 일치하는 `protocol` 값을 지정하여 보고서에 기능을 포함해야 합니다. 이 값을 지정하지 않으면 기능이 항상 보고서에 포함됩니다.

AddProductFeatures 상태를 사용하려면 RunTask 상태의 ResultVar 값을 다음 값 중 하나로 설정해야 합니다.

- 개별 테스트 케이스 ID를 지정한 경우 ResultVar을(를) *group-id_test-id_passed*(으)로 설정합니다.
- 개별 테스트 케이스 ID를 지정하지 않은 경우 ResultVar을(를) *group-id_passed*(으)로 설정합니다.

AddProductFeatures 상태에서는 다음과 같은 방식으로 테스트 결과를 확인합니다.

- 테스트 케이스 ID를 지정하지 않은 경우 각 테스트 그룹의 결과는 상태 머신 컨텍스트의 *group-id_passed* 변수 값에 따라 결정됩니다.
- 테스트 케이스 ID를 지정한 경우 각 테스트의 결과는 상태 머신 컨텍스트의 *group-id_test-id_passed* 변수 값을 기반으로 결정됩니다.

오류 처리

이 상태에서 제공된 그룹 ID가 유효한 그룹 ID가 아닌 경우 이 상태로 인해 AddProductFeaturesError 실행 오류가 발생합니다. 상태에서 실행 오류가 발생하는 경우 상태 머신 컨텍스트의 hasExecutionErrors 변수도 true로 설정됩니다.

Report

Report 상태는 *suite-name_Report.xml* 및 *awsiotdevicetester_report.xml* 파일을 생성합니다. 또한 이 상태는 보고서를 콘솔로 스트리밍합니다.

```
{
  "Type": "Report",
  "Next": "<state-name>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

테스트 실행기가 테스트 결과를 볼 수 있도록 테스트 실행 흐름이 끝날 무렵에는 항상 Report 상태로 전환해야 합니다. 일반적으로 이 상태 이후의 다음 상태는 Succeed입니다.

오류 처리

이 상태에서 보고서를 생성하는 데 문제가 발생하면 ReportError 실행 오류가 발생합니다.

LogMessage

LogMessage 상태는 test_manager.log 파일을 생성하고 로그 메시지를 콘솔로 스트리밍합니다.

```
{
  "Type": "LogMessage",
  "Next": "<state-name>"
  "Level": "info | warn | error"
  "Message": "<message>"
}
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

Level

로그 메시지를 생성할 때의 오류 수준입니다. 유효하지 않은 수준을 지정하면 이 상태가 오류 메시지를 생성하고 삭제합니다.

Message

로그할 메시지.

SelectGroup

SelectGroup 상태는 상태 머신 컨텍스트를 업데이트하여 선택된 그룹을 나타냅니다. 이 상태에서 설정된 값은 이후의 모든 Choice 상태에서 사용됩니다.

```
{
  "Type": "SelectGroup",
  "Next": "<state-name>"
  "TestGroups": [
    "<group-id>"
  ]
}
```


여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Next

현재 상태에서 작업을 실행한 후 전환할 상태의 이름입니다.

TestGroups

선택된 것으로 표시될 테스트 그룹 배열입니다. 이 배열의 각 테스트 그룹 ID에 대해 *group-id_selected* 변수는 컨텍스트에서 true로 설정됩니다. IDT는 지정된 그룹이 존재하는지 여부를 확인하지 않으므로 유효한 테스트 그룹 ID를 제공해야 합니다.

Fail

Fail 상태는 상태 머신이 제대로 실행되지 않았음을 나타냅니다. 이는 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
  "Type": "Fail"
}
```

Succeed

Succeed 상태는 상태 머신이 올바르게 실행되었음을 나타냅니다. 이는 상태 머신의 종료 상태이며 각 상태 머신 정의에는 이 상태가 포함되어야 합니다.

```
{
  "Type": "Succeed"
}
```

상태 머신 컨텍스트

상태 머신 컨텍스트는 실행 중에 상태 머신이 사용할 수 있는 데이터를 포함하는 읽기 전용 JSON 문서입니다. 상태 머신 컨텍스트는 상태 머신에서만 액세스할 수 있으며 테스트 흐름을 결정하는 정보를 포함합니다. 예를 들어 `userdata.json` 파일에서 테스트 실행기가 구성한 정보를 사용하여 특정 테스트 실행이 필요한지 여부를 결정할 수 있습니다.

상태 머신 컨텍스트는 다음 형식을 사용합니다.

```
{
```

```

    "pool": {
      <device-json-pool-element>
    },
    "userData": {
      <userdata-json-content>
    },
    "config": {
      <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
      "<group-id>"
    ],
    "specificTestCases": [
      "<test-id>"
    ],
    "hasExecutionErrors": true
  }

```

pool

테스트 실행을 위해 선택한 디바이스 풀에 대한 정보. 선택한 디바이스 풀의 경우 이 정보는 device.json 파일에 정의된 해당 최상위 디바이스 풀 배열 요소에서 검색됩니다.

userData

userdata.json 파일의 정보.

config

정보는 config.json 파일을 정의합니다.

suiteFailed

값은 상태 머신이 시작될 때 false로 설정됩니다. 테스트 그룹이 RunTask 상태로 실패하는 경우 이 값은 상태 머신의 남은 실행 기간 동안으로 true로 설정됩니다.

specificTestGroups

테스트 실행기가 전체 테스트 제품군 대신 실행할 특정 테스트 그룹을 선택하면 이 키가 생성되고 여기에는 특정 테스트 그룹 ID 목록이 포함됩니다.

specificTestCases

테스트 실행기가 전체 테스트 제품군 대신 실행할 특정 테스트 케이스를 선택하면 이 키가 생성되고 여기에는 특정 테스트 케이스 ID 목록이 포함됩니다.

hasExecutionErrors

상태 머신이 시작될 때 종료되지 않습니다. 어떤 상태에서도 실행 오류가 발생하는 경우 이 변수가 생성되고 남은 상태 머신 실행 기간 동안 `true`로 설정됩니다.

JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. 상태 정의의 JSONPath 쿼리 구문은 `{{$.query}}`입니다. 일부 상태에서는 JSONPath 쿼리를 자리 표시자 문자열로 사용할 수 있습니다. IDT는 자리 표시자 문자열을 컨텍스트에서 평가된 JSONPath 쿼리의 값으로 대체합니다. 다음 값에 자리 표시자를 사용할 수 있습니다.

- RunTask 상태의 TestCases 값.
- Expression 값 Choice 상태.

상태 머신 컨텍스트에서 데이터에 액세스할 때 다음 조건이 충족되는지 확인하십시오.

- JSON 경로는 \$.로 시작해야 합니다.
- 각 값은 문자열, 숫자 또는 부울로 평가되어야 합니다.

JSONPath 표기법을 사용하여 컨텍스트에서 데이터에 액세스하는 방법에 대한 자세한 내용은 [IDT 쿼리 텍스트 사용](#) 섹션을 참조하십시오.

실행 오류

실행 오류는 상태 머신이 상태를 실행할 때 발생하는 상태 머신 정의의 오류입니다. IDT는 `test_manager.log` 파일의 각 오류에 대한 정보를 기록하고 로그 메시지를 콘솔로 스트리밍합니다.

다음 방법을 사용하여 실행 오류를 처리할 수 있습니다.

- 상태 정의에 [Catch 블록](#)을 추가합니다.
- 상태 머신 컨텍스트에서 [hasExecutionErrors](#) 값의 값을 확인합니다.

Catch

Catch를 사용하려면 상태 정의에 다음을 추가하십시오.

```
"Catch": [
  {
    "ErrorEquals": [
```

```

        "<error-type>"
    ]
    "Next": "<state-name>"
}
]
```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Catch.ErrorEquals

캐치할 오류 유형의 배열입니다. 실행 오류가 지정된 값 중 하나와 일치하면 상태 머신이 Catch.Next에서 지정된 상태로 전환됩니다. 발생하는 오류 유형에 대한 자세한 내용은 각 상태 정의를 참조하십시오.

Catch.Next

현재 상태에서 Catch.ErrorEquals에서 지정한 값 중 하나와 일치하는 실행 오류가 발생하는 경우 전환할 다음 상태입니다.

Catch 블록은 둘 중 하나가 일치할 때까지 순차적으로 처리됩니다. Catch 블록에 나열된 오류와 일치하는 오류가 없으면 상태 머신이 계속 실행됩니다. 실행 오류는 잘못된 상태 정의로 인해 발생하므로 상태에 실행 오류가 발생하면 Fail 상태로 전환하는 것이 좋습니다.

hasExecutionError

일부 상태에서는 실행 오류가 발생하는 경우 오류가 발생하는 것 외에도 상태 시스템 컨텍스트에서 hasExecutionError 값을 true로 설정합니다. 이 값을 사용하여 오류 발생 시기를 감지한 다음 Choice 상태를 사용하여 상태 머신을 해당 Fail 상태로 전환할 수 있습니다.

이 메서드는 다음과 특징이 있습니다.

- 상태 머신은 hasExecutionError에 할당된 어떤 값으로도 시작되지 않으며 특정 상태가 값을 설정하기 전까지는 이 값을 사용할 수 없습니다. 즉, 실행 오류가 발생하지 않을 경우 상태 머신이 중지되지 않도록 이 값에 액세스하는 Choice 상태에 대해 명시적으로 FallthroughOnError를 false로 설정해야 합니다.
- true로 설정한 후에는 hasExecutionError가 false로 설정되거나 컨텍스트에서 제거되지 않습니다. 즉, 이 값은 true로 처음 설정할 때만 유용하며 이후의 모든 상태에서는 의미 있는 값을 제공하지 않습니다.
- hasExecutionError 값은 Parallel 상태의 모든 브랜치 상태 머신과 공유되므로 액세스 순서에 따라 예상치 못한 결과가 발생할 수 있습니다.

이러한 특성 때문에 Catch 블록을 대신 사용할 수 있는 경우에는 이 방법을 사용하지 않는 것이 좋습니다.

상태 머신 예제

이 섹션에서는 몇 가지 상태 시스템 구성의 예제를 제공합니다.

예제

- [상태 머신 예제: 단일 테스트 그룹 실행](#)
- [상태 머신 예제: 사용자가 선택한 테스트 그룹 실행](#)
- [상태 머신 예제: 제품 기능이 포함된 단일 테스트 그룹 실행](#)
- [상태 머신 예제: 두 테스트 그룹을 병렬로 실행](#)

상태 머신 예제: 단일 테스트 그룹 실행

이 상태 머신:

- ID GroupA를 사용하여 테스트 그룹을 실행하며 이 ID는 group.json 파일에 있어야 합니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

```
{
  "Comment": "Runs a single group and then generates a report.",
  "StartAt": "RunGroupA",
  "States": {
    "RunGroupA": {
      "Type": "RunTask",
      "Next": "Report",
      "TestGroup": "GroupA",
      "Catch": [
        {
          "ErrorEquals": [
            "RunTaskError"
          ],
          "Next": "Fail"
        }
      ]
    },
    "Report": {
```

```

        "Type": "Report",
        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Fail"
            }
        ],
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}

```

상태 머신 예제: 사용자가 선택한 테스트 그룹 실행

이 상태 머신:

- 테스트 실행기가 특정 테스트 그룹을 선택했는지 확인합니다. 테스트 실행기가 테스트 그룹을 선택하지 않으면 테스트 케이스를 선택할 수 없기 때문에 상태 머신은 특정 테스트 케이스를 확인하지 않습니다.
- 테스트 그룹을 선택한 경우:
 - 선택한 테스트 그룹 내에서 테스트 케이스를 실행합니다. 이를 위해 상태 머신은 RunTask 상태의 테스트 그룹이나 테스트 케이스를 명시적으로 지정하지 않습니다.
 - 모든 테스트를 실행한 후 보고서를 생성하고 종료합니다.
- 테스트 그룹을 선택하지 않은 경우:
 - 테스트 그룹 GroupA에서 테스트를 실행합니다.
 - 보고서를 생성하고 종료합니다.

```

{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise runs GroupA.",
    "StartAt": "SpecificGroupsCheck",

```

```
"States": {
  "SpecificGroupsCheck": {
    "Type": "Choice",
    "Default": "RunGroupA",
    "FallthroughOnError": true,
    "Choices": [
      {
        "Expression": "{{$.specificTestGroups[0]}} != ''",
        "Next": "RunSpecificGroups"
      }
    ]
  },
  "RunSpecificGroups": {
    "Type": "RunTask",
    "Next": "Report",
    "Catch": [
      {
        "ErrorEquals": [
          "RunTaskError"
        ],
        "Next": "Fail"
      }
    ]
  },
  "RunGroupA": {
    "Type": "RunTask",
    "Next": "Report",
    "TestGroup": "GroupA",
    "Catch": [
      {
        "ErrorEquals": [
          "RunTaskError"
        ],
        "Next": "Fail"
      }
    ]
  },
  "Report": {
    "Type": "Report",
    "Next": "Succeed",
    "Catch": [
      {
        "ErrorEquals": [
          "ReportError"
        ]
      }
    ]
  }
}
```

```

        ],
        "Next": "Fail"
      }
    ]
  },
  "Succeed": {
    "Type": "Succeed"
  },
  "Fail": {
    "Type": "Fail"
  }
}
}

```

상태 머신 예제: 제품 기능이 포함된 단일 테스트 그룹 실행

이 상태 머신:

- 테스트 그룹 GroupA를 실행합니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다.
- awsiotdevicetester_report.xml 파일에 FeatureThatDependsOnGroupA 기능을 추가합니다.
 - GroupA가 통과하면 기능이 supported로 설정됩니다.
 - 이 기능은 보고서에서 선택 사항으로 표시되어 있지 않습니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

```

{
  "Comment": "Runs GroupA and adds product features based on GroupA",
  "StartAt": "RunGroupA",
  "States": {
    "RunGroupA": {
      "Type": "RunTask",
      "Next": "AddProductFeatures",
      "TestGroup": "GroupA",
      "ResultVar": "GroupA_passed",
      "Catch": [
        {
          "ErrorEquals": [
            "RunTaskError"
          ],

```



```

        "Next": "Fail"
      }
    ]
  },
  "AddProductFeatures": {
    "Type": "AddProductFeatures",
    "Next": "Report",
    "Features": [
      {
        "Feature": "FeatureThatDependsOnGroupA",
        "Groups": [
          "GroupA"
        ],
        "IsRequired": true
      }
    ]
  },
  "Report": {
    "Type": "Report",
    "Next": "Succeed",
    "Catch": [
      {
        "ErrorEquals": [
          "ReportError"
        ],
        "Next": "Fail"
      }
    ]
  },
  "Succeed": {
    "Type": "Succeed"
  },
  "Fail": {
    "Type": "Fail"
  }
}

```

상태 머신 예제: 두 테스트 그룹을 병렬로 실행

이 상태 머신:

- GroupA 및 GroupB 테스트 그룹을 병렬로 실행합니다. 브랜치 상태 머신의 RunTask 상태에 의해 컨텍스트에 저장된 ResultVar 변수는 AddProductFeatures 상태에서 사용할 수 있습니다.
- 실행 오류가 있는지 확인하고 오류가 있는 경우, Fail로 전환합니다. 이 상태 머신은 Catch 블록을 사용하지 않습니다. 해당 메서드는 브랜치 상태 머신의 실행 오류를 감지하지 못하기 때문입니다.
- 통과한 그룹을 기반으로 awsiotdevicetester_report.xml 파일에 기능을 추가합니다.
 - GroupA가 통과하면 기능이 supported로 설정됩니다.
 - 이 기능은 보고서에서 선택 사항으로 표시되어 있지 않습니다.
- 보고서를 생성하고 오류가 없는 경우 Succeed로 전환하고 그렇지 않으면 Fail로 전환합니다.

디바이스 풀에 두 개의 디바이스가 구성된 경우 GroupA 및 GroupB 둘 다 동시에 실행할 수 있습니다. 그러나 GroupA 또는 GroupB 둘 중 하나에 여러 테스트가 포함된 경우에는 두 디바이스 모두 해당 테스트에 할당될 수 있습니다. 장치가 하나만 구성된 경우 테스트 그룹은 순차적으로 실행됩니다.

```
{
  "Comment": "Runs GroupA and GroupB in parallel",
  "StartAt": "RunGroupAAndB",
  "States": {
    "RunGroupAAndB": {
      "Type": "Parallel",
      "Next": "CheckForErrors",
      "Branches": [
        {
          "Comment": "Run GroupA state machine",
          "StartAt": "RunGroupA",
          "States": {
            "RunGroupA": {
              "Type": "RunTask",
              "Next": "Succeed",
              "TestGroup": "GroupA",
              "ResultVar": "GroupA_passed",
              "Catch": [
                {
                  "ErrorEquals": [
                    "RunTaskError"
                  ],
                  "Next": "Fail"
                }
              ]
            },
            "Succeed": {
```

```

        "Type": "Succeed"
    },
    "Fail": {
        "Type": "Fail"
    }
},
{
    "Comment": "Run GroupB state machine",
    "StartAt": "RunGroupB",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Succeed",
            "TestGroup": "GroupB",
            "ResultVar": "GroupB_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
],
"CheckForErrors": {
    "Type": "Choice",
    "Default": "AddProductFeatures",
    "FallthroughOnError": true,
    "Choices": [
        {
            "Expression": "{{$.hasExecutionErrors}} == true",
            "Next": "Fail"
        }
    ]
}

```

```

    ]
  },
  "AddProductFeatures": {
    "Type": "AddProductFeatures",
    "Next": "Report",
    "Features": [
      {
        "Feature": "FeatureThatDependsOnGroupA",
        "Groups": [
          "GroupA"
        ],
        "IsRequired": true
      },
      {
        "Feature": "FeatureThatDependsOnGroupB",
        "Groups": [
          "GroupB"
        ],
        "IsRequired": true
      }
    ]
  },
  "Report": {
    "Type": "Report",
    "Next": "Succeed",
    "Catch": [
      {
        "ErrorEquals": [
          "ReportError"
        ],
        "Next": "Fail"
      }
    ]
  },
  "Succeed": {
    "Type": "Succeed"
  },
  "Fail": {
    "Type": "Fail"
  }
}

```

IDT 테스트 케이스 실행 파일 생성

다음과 같은 방법으로 테스트 세트 폴더에 테스트 케이스 실행 파일을 만들고 배치할 수 있습니다.

- `test.json` 파일의 인수나 환경 변수를 사용하여 실행할 테스트를 결정하는 테스트 세트의 경우, 전체 테스트 세트에 대해 단일 테스트 사례 실행 파일을 만들거나 테스트 세트의 각 테스트 그룹에 대해 테스트 실행 파일을 만들 수 있습니다.
- 지정된 명령을 기반으로 특정 테스트를 실행하려는 테스트 세트의 경우, 테스트 세트의 각 테스트 사례에 대해 테스트 사례 실행 파일을 하나씩 만듭니다.

테스트 작성자는 사용 사례에 적합한 접근 방식을 결정하고 그에 따라 테스트 사례 실행 파일을 구성할 수 있습니다. 각 `test.json` 파일에 올바른 테스트 케이스 실행 파일 경로를 제공하고 지정된 실행 파일이 올바르게 실행되는지 확인합니다.

모든 장치에서 테스트 케이스를 실행할 준비가 되면 IDT는 다음 파일을 읽습니다.

- 선택한 테스트 사례에 대한 `test.json`에 따라 시작할 프로세스와 설정할 환경 변수가 결정됩니다.
- 테스트 제품군용 `suite.json`에 따라 설정할 환경 변수가 결정됩니다.

IDT는 `test.json` 파일에 지정된 명령과 인수를 기반으로 필수 테스트 실행 파일 프로세스를 시작하고 필요한 환경 변수를 프로세스에 전달합니다.

IDT 클라이언트 SDK 사용

IDT 클라이언트 SDK를 사용하면 IDT 및 테스트 대상 장치와 상호 작용하는 데 사용할 수 있는 API 명령을 사용하여 테스트 실행 파일에 테스트 로직을 작성하는 방법을 간소화할 수 있습니다. IDT는 현재 다음과 같은 SDK를 제공합니다.

- Python용 IDT 클라이언트 SDK
- Go용 IDT 클라이언트 SDK
- Java용 IDT 클라이언트 SDK

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 상호작용](#)
- [IDT 상호 작용](#)
- [호스트 상호작용](#)

장치 상호작용

다음 명령을 사용하면 추가 장치 상호 작용 및 연결 관리 기능을 구현하지 않고도 테스트 중인 장치와 통신할 수 있습니다.

ExecuteOnDevice

테스트 세트가 SSH 또는 Docker 셸 연결을 지원하는 장치에서 셸 명령을 실행할 수 있습니다.

CopyToDevice

테스트 세트가 IDT를 실행하는 호스트 컴퓨터의 로컬 파일을 SSH 또는 Docker 셸 연결을 지원하는 장치의 지정된 위치로 복사할 수 있습니다.

ReadFromDevice

테스트 세트가 UART 연결을 지원하는 장치의 직렬 포트에서 읽을 수 있도록 허용합니다.

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. 하지만 이러한 명령이 테스트 사례 요구 사항을 충족하지 않는 경우, IDT 컨텍스트에서 장치 액세스 정보를 검색하고 이를 사용하여 테스트 세트에서 장치에 직접 연결할 수 있습니다. 직접 연결하려면 테스트 중인 장치와 리소스 장치에 대해 각각 `device.connectivity` 및 `resource.devices.connectivity` 필드에서 정보를 검색합니다. IDT 컨텍스트 사용에 관한 자세한 내용은 [IDT 컨텍스트 사용](#) 섹션을 참조합니다.

IDT 상호 작용

다음 명령을 사용하면 테스트 세트가 IDT와 통신할 수 있습니다.

PollForNotifications

테스트 세트에서 IDT의 알림을 확인할 수 있습니다.

GetContextValue 'and'GetContextString

테스트 세트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 내용은 [IDT 컨텍스트 사용](#) 섹션을 참조하세요.

SendResult

테스트 세트에서 테스트 사례 결과를 IDT에 보고할 수 있습니다. 이 명령은 테스트 세트의 각 테스트 케이스 끝에서 직접적으로 호출해야 합니다.

호스트 상호작용

다음 명령을 사용하면 테스트 세트가 호스트 머신과 통신할 수 있습니다.

PollForNotifications

테스트 세트에서 IDT의 알림을 확인할 수 있습니다.

GetContextValue 'and'GetContextString

테스트 세트가 IDT 컨텍스트에서 값을 검색할 수 있도록 합니다. 자세한 내용은 [IDT 컨텍스트 사용](#) 섹션을 참조하세요.

ExecuteOnHost

테스트 세트가 로컬 머신에서 명령을 실행할 수 있도록 하고 IDT가 테스트 케이스 실행 수명 주기를 관리할 수 있도록 합니다.

IDT CLI 명령을 활성화합니다.

run-suite 명령 IDT CLI는 테스트 실행기가 테스트 실행을 사용자 지정할 수 있는 몇 가지 옵션을 제공합니다. 테스트 실행기가 이러한 옵션을 사용하여 사용자 지정 테스트 세트를 실행할 수 있도록 하려면 IDT CLI에 대한 지원을 구현해야 합니다. 지원을 구현하지 않는 경우, 테스트 실행기는 여전히 테스트를 실행할 수 있지만 일부 CLI 옵션은 제대로 작동하지 않습니다. 이상적인 고객 경험을 제공하려면 IDT CLI에서 run-suite 명령에 대한 다음 인수에 대한 지원을 구현하는 것이 좋습니다.

timeout-multiplier

테스트를 실행하는 동안 모든 제한 시간에 적용할 1.0보다 큰 값을 지정합니다.

테스트 실행기는 이 인수를 사용하여 실행하려는 테스트 사례의 제한 시간을 늘릴 수 있습니다. 테스트 실행기가 `run-suite` 명령에서 이 인수를 지정하면 IDT는 이 인수를 사용하여 `IDT_TEST_TIMEOUT` 환경 변수의 값을 계산하고 IDT 컨텍스트에서 `config.timeoutMultiplier` 필드를 설정합니다. 이 인수를 뒷받침하려면 다음을 수행하여야 합니다.

- `test.json` 파일의 제한 시간 값을 직접 사용하는 대신 `IDT_TEST_TIMEOUT` 환경 변수를 읽고 올바르게 계산된 제한 시간 값을 구합니다.
- IDT 컨텍스트에서 `config.timeoutMultiplier` 값을 검색하여 장기 실행 제한 시간에 적용합니다.

제한 시간 이벤트로 인한 조기 종료에 대한 자세한 내용은 [종료 동작을 지정합니다](#).을(를) 참조하세요.

stop-on-first-failure

장애가 발생할 경우, IDT에서 모든 테스트 실행을 중지하도록 지정합니다.

테스트 실행기가 `run-suite` 명령에 이 인수를 지정하면 IDT는 장애가 발생하는 즉시 테스트 실행을 중단합니다. 그러나 테스트 케이스가 병렬로 실행되는 경우, 예상치 못한 결과가 발생할 수 있습니다. 지원을 구현하려면 IDT에서 이 이벤트가 발생할 경우, 테스트 로직에서 실행 중인 모든 테스트 사례를 중지하고, 임시 리소스를 정리하고, 테스트 결과를 IDT에 보고하도록 지시해야 합니다. 실패 시 조기 종료에 대한 자세한 내용은 [종료 동작을 지정합니다](#). 섹션을 참조하세요.

group-id'and'test-id

IDT가 선택한 테스트 그룹 또는 테스트 케이스만 실행하도록 지정합니다.

테스트 실행기는 `run-suite` 명령과 함께 이러한 인수를 사용하여 다음과 같은 테스트 실행 동작을 지정할 수 있습니다.

- 지정된 테스트 제품군에 있는 모든 테스트 그룹을 실행합니다.
- 지정된 테스트 그룹 내에서 엄선된 테스트를 실행합니다.

이러한 인수가 지원되려면 테스트 오케스트레이터의 특정 `RunTask` 및 `Choice` 상태 세트가 테스트 제품군에 대한 테스트 오케스트레이터에 포함되어야 합니다. 사용자 지정 상태 머신을 사용하지 않는 경우에는 기본 IDT 테스트 오케스트레이터에 필수 상태가 포함되므로 추가 조치를 취할 필요가 없습니다. 그러나 사용자 지정 테스트 오케스트레이터를 사용하는 경우에는 [상태 머신 예제: 사용자가 선택한 테스트 그룹 실행](#)를 샘플로 사용하여 테스트 오케스트레이터에서 필수 상태를 추가합니다.

IDT CLI 명령에 대한 자세한 내용은 [사용자 지정 테스트 도구 모음 디버그 및 실행](#) 단원을 참조합니다.

이벤트 로그 작성

테스트가 실행되는 동안 stdout 및 stderr에 데이터를 보내 콘솔에 이벤트 로그와 오류 메시지를 기록합니다. 콘솔 메시지의 형식에 대한 자세한 정보는 [콘솔 메시지 형식](#)에서 확인하세요.

IDT가 테스트 세트 실행을 마치면 `<devicetester-extract-location>/results/<execution-id>/logs` 폴더에 있는 `test_manager.log` 파일에서도 이 정보를 확인할 수 있습니다.

테스트 대상 장치의 로그를 포함하여 테스트 실행의 로그를 `<device-tester-extract-location>/results/<execution-id>/logs` 폴더에 있는 `<group-id>_<test-id>` 파일에 기록하도록 각 테스트 사례를 구성할 수 있습니다. 이렇게 하려면 `testData.logFilePath` 쿼리를 사용하여 IDT 컨텍스트에서 로그 파일의 경로를 검색하고 해당 경로에 파일을 만든 다음 원하는 콘텐츠를 작성해야 합니다. IDT는 실행 중인 테스트 케이스를 기반으로 경로를 자동으로 업데이트합니다. 테스트 사례에 대한 로그 파일을 만들지 않기로 선택하면 해당 테스트 사례에 대한 파일이 생성되지 않습니다.

필요에 따라 `<device-tester-extract-location>/logs` 폴더에 추가 로그 파일을 생성하도록 텍스트 실행 파일을 설정할 수도 있습니다. 파일을 덮어쓰지 않도록 로그 파일 이름에 고유한 접두사를 지정하는 것이 좋습니다.

결과를 IDT에 보고합니다.

IDT는 테스트 결과를 `awsiotdevicetester_report.xml` 및 `<suite-name>_report.xml` 파일에 기록합니다. 이 보고서 파일은 `<device-tester-extract-location>/results/<execution-id>/`에 위치합니다. 두 보고서 모두 테스트 세트의 실행 결과를 캡처합니다. IDT에서 이러한 보고서에 사용하는 스키마에 대한 자세한 내용은 [IDT 테스트 결과 및 로그 검토](#)을(를) 참조합니다.

`<suite-name>_report.xml` 파일 내용을 채우려면 테스트 실행이 완료되기 전에 `SendResult` 명령을 사용하여 테스트 결과를 IDT에 보고해야 합니다. IDT에서 테스트 결과를 찾을 수 없는 경우, 테스트 사례에 오류가 발생합니다. 다음 Python 발췌문은 테스트 결과를 IDT로 보내는 명령을 보여줍니다.

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

API를 통해 결과를 보고하지 않는 경우, IDT는 테스트 아티팩트 폴더에서 테스트 결과를 찾습니다. 이 폴더의 경로는 IDT 컨텍스트의 `testData.testArtifactsPath` 파일에 저장됩니다. 이 폴더에서 IDT는 찾은 첫 번째 알파벳순으로 정렬된 XML 파일을 테스트 결과로 사용합니다.

테스트 로직이 JUnit XML 결과를 생성하는 경우, 결과를 파싱한 다음 API를 사용하여 IDT에 제출하는 대신 아티팩트 폴더의 XML 파일에 테스트 결과를 기록하여 결과를 IDT에 직접 제공할 수 있습니다.

이 방법을 사용하는 경우, 테스트 로직이 테스트 결과를 정확하게 요약하고 결과 파일의 형식을 `suite-name_report.xml` 파일과 동일한 형식으로 지정해야 합니다. IDT는 제공된 데이터에 대한 검증은 수행하지 않습니다. 단, 다음과 같은 경우는 예외입니다.

- IDT는 `testsuites` 태그의 모든 속성을 무시합니다. 대신 보고된 다른 테스트 그룹 결과에서 태그 속성을 계산합니다.
- 하나 이상의 `testsuite` 태그가 `testsuites` 내에 있어야 합니다.

IDT는 모든 테스트 사례에 동일한 아티팩트 폴더를 사용하고 테스트 실행 사이에 결과 파일을 삭제하지 않기 때문에 IDT에서 잘못된 파일을 읽을 경우, 이 방법을 사용하면 잘못된 보고가 발생할 수도 있습니다. 생성된 XML 결과 파일은 모든 테스트 사례에서 동일한 이름을 사용하여 각 테스트 사례의 결과를 덮어쓰고 IDT에서 사용할 수 있는 올바른 결과가 있는지 확인하는 것이 좋습니다. 테스트 세트의 보고에는 혼합된 접근 방식을 사용할 수 있습니다. 즉, 일부 테스트 사례에는 XML 결과 파일을 사용하고 다른 테스트 사례에는 API를 통해 결과를 제출하는 등 혼합된 접근 방식을 사용할 수 있지만 이 방법은 권장되지 않습니다.

종료 동작을 지정합니다.

테스트 케이스에서 실패 또는 오류 결과가 보고되더라도 텍스트 실행 파일이 항상 종료 코드 0으로 종료되도록 구성합니다. 0이 아닌 종료 코드는 테스트 케이스가 실행되지 않았음을 나타내거나 테스트 케이스 실행 파일이 IDT에 결과를 전달할 수 없는 경우에만 사용합니다. IDT가 0이 아닌 종료 코드를 받으면 테스트 케이스에 오류가 발생하여 테스트 케이스를 실행할 수 없게 된 것으로 표시합니다.

IDT는 다음 이벤트에서 테스트 케이스가 완료되기 전에 테스트 케이스의 실행을 중단하도록 요청하거나 예상할 수 있습니다. 이 정보를 사용하여 테스트 케이스에서 이러한 각 이벤트를 감지하도록 테스트 케이스 실행 파일을 구성합니다.

제한 시간

테스트 케이스가 `test.json` 파일에 지정된 제한 시간 값보다 오래 실행될 때 발생합니다. 테스트 실행기가 `timeout-multiplier` 인수를 사용하여 제한 시간 승수를 지정한 경우, IDT는 승수로 제한 시간 값을 계산합니다.

이 이벤트를 탐지하려면 `IDT_TEST_TIMEOUT` 환경 변수를 사용합니다. 테스트 실행기가 테스트를 시작하면 IDT는 `IDT_TEST_TIMEOUT` 환경 변수의 값을 계산된 제한 시간 값(초)으로 설정하고 변수를 테스트 케이스 실행 파일로 전달합니다. 변수 값을 읽어 적절한 타이머를 설정할 수 있습니다.

인터럽트

테스트 러너가 IDT를 인터럽트할 때 발생합니다. 예를 들어, Ctrl+C 키를 누르면 됩니다.

터미널은 신호를 모든 하위 프로세스에 전파하므로 인터럽트 신호를 감지하도록 테스트 케이스에 신호 처리기를 구성하기만 하면 됩니다.

또는 정기적으로 API를 폴링하여 PollForNotifications API 응답의 CancellationRequested 부울 값을 확인할 수도 있습니다. IDT는 인터럽트 신호를 수신하면 CancellationRequested 부울 값을 true(으)로 설정합니다.

첫 번째 실패 시 중지

현재 테스트 케이스와 병렬로 실행 중인 테스트 케이스가 실패하고 테스트 실행기가 stop-on-first-failure 인수를 사용하여 IDT에 오류가 발생할 경우, 중지하도록 지정하는 경우, 발생합니다.

이 이벤트를 탐지하기 위해 주기적으로 API를 폴링하여 PollForNotifications API 응답의 CancellationRequested 부울 값을 확인할 수 있습니다. IDT에서 장애가 발생하고 첫 번째 실패 시 중지되도록 구성된 경우, IDT는 CancellationRequested 부울 값을 true(으)로 설정합니다.

이러한 이벤트가 발생하면 IDT는 현재 실행 중인 테스트 케이스의 실행이 완료될 때까지 5분 동안 기다립니다. 실행 중인 모든 테스트 케이스가 5분 내에 종료되지 않으면 IDT는 각 프로세스를 강제로 중지합니다. 프로세스가 종료되기 전에 IDT에서 테스트 결과를 받지 못한 경우, 테스트 케이스가 제한 시간이 초과된 것으로 표시됩니다. 가장 좋은 방법은 테스트 케이스에서 이벤트 중 하나가 발생할 때 다음 작업을 수행하도록 하는 것입니다.

1. 일반 테스트 로직 실행을 중단합니다.
2. 테스트 대상 장치의 테스트 아티팩트와 같은 임시 리소스를 모두 정리합니다.
3. 테스트 실패 또는 오류와 같은 테스트 결과를 IDT에 보고합니다.
4. 종료

IDT 컨텍스트 사용

IDT가 테스트 도구 모음을 실행할 때, 테스트 도구 모음은 각 테스트 실행 방식을 결정하는 데 사용할 수 있는 데이터 세트에 액세스할 수 있습니다. 이 데이터를 IDT 컨텍스트라고 합니다. 예를 들어 테스트 러너가 userdata.json 파일로 제공한 사용자 데이터 구성은 IDT 컨텍스트의 테스트 도구 모음에서 사용할 수 있도록 만들어졌습니다.

IDT 컨텍스트는 읽기 전용 JSON 문서로 간주할 수 있습니다. 테스트 도구 모음은 객체, 배열, 숫자 등과 같은 표준 JSON 데이터 유형을 사용하여 컨텍스트에서 데이터를 검색하고, 컨텍스트에 데이터를 쓸 수 있습니다.

컨텍스트 스키마

IDT 컨텍스트는 다음 형식을 사용합니다.

```
{
  "config": {
    "config-json-content"
    "timeoutMultiplier": timeout-multiplier
  },
  "device": {
    "device-json-device-element"
  },
  "devicePool": {
    "device-json-pool-element"
  },
  "resource": {
    "devices": [
      {
        "resource-json-device-element"
        "name": "<resource-name>"
      }
    ]
  },
  "testData": {
    "awsCredentials": {
      "awsAccessKeyId": "<access-key-id>",
      "awsSecretAccessKey": "<secret-access-key>",
      "awsSessionToken": "<session-token>"
    },
    "logFilePath": "/path/to/log/file"
  },
  "userData": {
    "userdata-json-content"
  }
}
```

config

[config.json 파일](#)의 정보. config 필드에는 다음과 같은 추가 필드도 포함됩니다.

`config.timeoutMultiplier`

테스트 도구 모음에서 사용하는 모든 시간 제한 값의 승수입니다. 이 값은 IDT CLI에서 테스트 러너에 의해 지정됩니다. 기본값은 1입니다.

`device`

테스트 실행을 위해 선택한 장치에 대한 정보. 이 정보는 선택한 장치의 [device.json 파일](#)에 있는 `devices` 배열 요소와 동일합니다.

`devicePool`

테스트 실행을 위해 선택한 장치 풀에 대한 정보. 이 정보는 선택한 장치 풀에 대해 `device.json` 파일에 정의된 최상위 장치 풀 배열 요소와 동일합니다.

`resource`

`resource.json` 파일의 리소스 장치에 대한 정보.

`resource.devices`

이 정보는 `resource.json` 파일에 정의된 `devices` 배열과 동일합니다. 각 `devices` 요소에는 다음과 같은 추가 필드가 포함됩니다.

`resource.device.name`

리소스 장치의 이름입니다. 이 값은 `test.json` 파일의 `requiredResource.name` 값으로 설정됩니다.

`testData.awsCredentials`

테스트에서 AWS 클라우드에 연결하는 데 사용하는 AWS 보안 인증입니다. 이 정보는 `config.json` 파일에서 가져옵니다.

`testData.logFilePath`

테스트 사례가 로그 메시지를 기록하는 로그 파일의 경로입니다. 이 파일이 존재하지 않는 경우 테스트 도구 모음에서 해당 파일을 생성합니다.

`userData`

테스트 러너가 [userdata.json 파일](#)에서 제공한 정보.

컨텍스트에서 데이터 액세스

JSON 파일 및 `GetContextValue` 및 `GetContextString` API로 실행 가능한 텍스트 파일에서 JSONPath 표기법을 사용하여 컨텍스트를 쿼리할 수 있습니다. IDT 컨텍스트에 액세스하기 위한 JSONPath 문자열의 구문은 다음과 같이 다양합니다.

- `suite.json`와(과) `test.json`에서는 `{{query}}`을(를) 사용합니다. 즉, 루트 요소 `$`을(를) 사용하여 표현식을 시작하지 마십시오.
- `test_orchestrator.yaml`에서는 `{{query}}`을(를) 사용합니다.

더 이상 사용되지 않는 상태 시스템을 사용하는 경우 `state_machine.json`에서 `{{$.query}}`를 사용합니다.

- API 명령에서는 명령에 따라 `query` 또는 `{{$.query}}`을(를) 사용합니다. 자세한 내용을 알아보려면 SDK에서 인라인 설명서를 참조하세요.

다음 표에서는 일반적인 JSONPath 표현식의 연산자를 설명합니다.

Operator	Description
\$	The root element. Because the top-level context value for IDT is an object, you will typically use <code>\$.</code> to start your queries.
.childName	Accesses the child element with name <code>childName</code> 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 <code>awsRegion</code> value in the <code>config</code> object is <code>\$.config.awsRegion</code> .
[start:end]	Filters elements from an array, retrieving items beginning from the <code>##</code> index and going up to the end index, both inclusive.
[index1, index2, ... , indexN]	Filters elements from an array, retrieving items from only the specified indices.

Operator	Description
[?(expr)]	Filters elements from an array using the expr expression. This expression must evaluate to a boolean value.

필터 표현식을 만들려면 다음 구문을 사용하십시오.

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

이 구문에서:

- jsonpath은(는) 표준 JSON 구문을 사용하는 JSONPath입니다.
- value은(는) 표준 JSON 구문을 사용하는 모든 사용자 지정 값입니다.
- operator는 다음 작업 중 하나를 호출합니다.
 - < (미만)
 - <= (이하)
 - == (같음)

표현식의 JSONPath 또는 값이 배열, 부울 또는 객체 값인 경우 이것이 사용할 수 있는 유일하게 지원되는 바이너리 연산자입니다.

- >= (이상)
- > (초과)
- =~(정규 표현식 일치). 필터 표현식에서 이 연산자를 사용하려면 표현식 왼쪽의 JSONPath 또는 값이 문자열로 평가되어야 하고, 오른쪽은 [RE2 구문](#)을 따르는 패턴 값이어야 합니다.

{{*query*}} 형식의 JSONPath 쿼리를 test.json 파일의 args 및 environmentVariables 필드, suite.json 파일의 environmentVariables 필드 내에서 자리 표시자 문자열로 사용할 수 있습니다. IDT는 컨텍스트 조회를 수행하고 쿼리의 평가된 값으로 필드를 채웁니다. 예를 들어 suite.json 파일에서 자리 표시자 문자열을 사용하여 각 테스트 사례에 따라 변경되는 환경 변수 값을 지정할 수 있으며, IDT는 환경 변수를 각 테스트 사례에 맞는 값으로 채웁니다. 하지만 test.json 및 suite.json 파일에서 자리 표시자 문자열을 사용하는 경우 쿼리에 다음 사항을 고려해야 합니다.

- 쿼리에서 나타나는 devicePool 키는 모두 소문자로 입력해야 합니다. 즉, devicepool을(를) 대신 사용하십시오.

- 배열의 경우 문자열 배열만 사용할 수 있습니다. 또한 배열은 비표준 `item1, item2, ..., itemN` 형식을 사용합니다. 배열에 요소가 하나만 포함된 경우 이 배열은 `item(으)`로 직렬화되어 문자열 필드와 구분할 수 없게 됩니다.
- 자리 표시자를 사용하여 컨텍스트에서 객체를 검색할 수 없습니다.

이러한 고려 사항 때문에 가능하면 `test.json` 및 `suite.json` 파일의 자리 표시자 문자열 대신 API를 사용하여 테스트 로직의 컨텍스트에 액세스하는 것이 좋습니다. 하지만 경우에 따라 JSONPath 자리 표시자를 사용하여 단일 문자열을 가져와서 환경 변수로 설정하는 것이 더 편리할 수 있습니다.

테스트 러너를 위한 설정 구성

사용자 지정 테스트 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 도구 모음을 디버깅](#)하도록 이러한 파일을 구성해야 합니다. 테스트 러너가 테스트 도구 모음을 실행하는 데 필요한 대로 다음 설정을 구성할 수 있도록 지침을 제공해야 합니다.

device.json 구성

`device.json` 파일은 테스트가 실행되는 장치에 대한 정보가 필요합니다(예: IP 주소, 로그인 정보, 운영 체제, CPU 아키텍처).

테스트 러너는 `<device-tester-extract-location>/configs/` 폴더에 있는 다음 템플릿 `device.json` 파일을 사용하여 이 정보를 제공할 수 있습니다.

```
[
  {
    "id": "<pool-id>",
    "sku": "<pool-sku>",
    "features": [
      {
        "name": "<feature-name>",
        "value": "<feature-value>",
        "configs": [
          {
            "name": "<config-name>",
            "value": "<config-value>"
          }
        ]
      }
    ]
  }
]
```



```

        }
    ],
}
],
"devices": [
{
    "id": "<device-id>",
    "connectivity": {
        "protocol": "ssh | uart | docker",
        // ssh
        "ip": "<ip-address>",
        "port": <port-number>,
        "auth": {
            "method": "pki | password",
            "credentials": {
                "user": "<user-name>",
                // pki
                "privKeyPath": "/path/to/private/key",

                // password
                "password": "<password>",
            }
        },
        // uart
        "serialPort": "<serial-port>",

        // docker
        "containerId": "<container-id>",
        "containerUser": "<container-user-name>",
    }
}
]
}
]

```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

id

장치 풀이라고 하는 장치 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 장치의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 장치는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 장치가 사용됩니다.

sku

테스트 대상 장치를 고유하게 식별하는 영숫자 값입니다. SKU는 정규화된 장치를 추적하는 데 사용됩니다.

Note

AWS Partner 장치 카탈로그에 보드를 등록하려면 여기서 지정하는 SKU가 목록 등록 프로세스에 사용하는 SKU와 일치해야 합니다.

features

선택 사항입니다. 장치의 지원되는 기능이 포함된 배열입니다. 장치 기능은 테스트 도구 모음에서 구성한 사용자 정의 값입니다. `device.json` 파일에 포함할 기능 이름 및 값에 대한 정보를 테스트 러너에게 제공해야 합니다. 예를 들어, 다른 장치의 MQTT 서버 역할을 하는 장치를 테스트하려는 경우 이름이 `MQTT_QOS`로 지정된 기능에 대해 지원되는 특정 수준을 검증하도록 테스트 로직을 구성할 수 있습니다. 테스트 러너는 이 기능 이름을 제공하고 기능 값을 장치에서 지원하는 QOS 수준으로 설정합니다. 제공된 정보는 `devicePool.features` 쿼리를 사용하여 [IDT 컨텍스트](#)에서 검색하거나 `pool.features` 쿼리를 사용하여 [테스트 오케스트레이터 컨텍스트](#)에서 검색할 수 있습니다.

`features.name`

기능의 이름입니다.

`features.value`

지원되는 기능 값.

`features.configs`

필요한 경우 기능에 대한 구성 설정.

`features.config.name`

구성 설정의 이름입니다.

`features.config.value`

지원되는 설정값.

devices

테스트할 풀의 장치 배열입니다. 최소 1개 이상의 장치가 필요합니다.

`devices.id`

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

`connectivity.protocol`

이러한 장치와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 장치는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 값은 uart 물리적 장치의 경우 및, docker Docker 컨테이너의 경우 ssh 입니다.

`connectivity.ip`

테스트 대상 장치의 IP 입니다.

이 속성은 `connectivity.protocol`이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.port`

선택 사항입니다. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.auth`

연결에 대한 인증 정보입니다.

이 속성은 `connectivity.protocol`이 ssh로 설정된 경우에만 적용됩니다.

`connectivity.auth.method`

지정된 연결 프로토콜을 통해 장치에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.password`

테스트 대상 장치에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.privKeyPath`

테스트 대상 장치에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.user`

테스트 대상 장치에 로그인하기 위한 사용자 이름입니다.

`connectivity.serialPort`

선택 사항입니다. 장치가 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 사항입니다. 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 4입니다.

이 속성은 `connectivity.protocol`이 `ssh`로 설정된 경우에만 적용됩니다.

Note

테스트 러너가 테스트에 대해 잘못된 디바이스 연결을 구성했는지 확인하려면 테스트 오케스트레이터 컨텍스트에서 `pool.Devices[0].Connectivity.Protocol`을 검색하고 이를 Choice 상태의 예상 값과 비교할 수 있습니다. 잘못된 프로토콜이 사용된 경우 `LogMessage` 상태를 사용하여 메시지를 인쇄하고 `Fail` 상태로 전환하세요. 또는 오류 처리 코드를 사용하여 잘못된 장치 유형에 대한 테스트 실패를 보고할 수 있습니다.

(선택 사항) userdata.json 구성

userdata.json 파일에는 테스트 도구 모음에 필요하지만 device.json 파일에 지정되지 않은 모든 추가 정보가 들어 있습니다. 이 파일의 형식은 테스트 도구 모음에 정의된 [userdata_scheme.json 파일](#)에 따라 달라집니다. 테스트 작성자인 경우, 이 정보를 작성한 테스트 도구 모음을 실행할 사용자에게 제공해야 합니다.

(선택 사항) resource.json 구성

resource.json 파일에는 리소스 장치로 사용될 모든 장치에 대한 정보가 들어 있습니다. 리소스 장치는 테스트 대상 장치의 특정 기능을 테스트하는 데 필요한 장치입니다. 예를 들어 장치의 Bluetooth 기능을 테스트하려면 리소스 장치를 사용하여 장치가 제대로 연결될 수 있는지 테스트할 수 있습니다. 리소스 장치는 선택 사항이며 필요한 만큼 리소스 장치를 요청할 수 있습니다. 테스트 작성자는 [test.json 파일](#)을 사용하여 테스트에 필요한 리소스 장치 기능을 정의합니다. 그런 다음 테스트 러너는 resource.json 파일을 사용하여 필수 기능을 갖춘 리소스 장치 풀을 제공합니다. 이 정보를 작성한 테스트 도구 모음을 실행할 사용자에게 제공해야 합니다.

테스트 러너는 `<device-tester-extract-location>/configs/` 폴더에 있는 다음 템플릿 resource.json 파일을 사용하여 이 정보를 제공할 수 있습니다.

```
[
  {
    "id": "<pool-id>",
    "features": [
      {
        "name": "<feature-name>",
        "version": "<feature-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

장치 풀이라고 하는 장치 모음을 고유하게 식별하는 사용자 정의 영숫자 ID입니다. 풀에 속한 장치의 하드웨어는 서로 동일해야 합니다. 테스트 제품군을 실행할 때 풀에 있는 장치는 워크로드를 병렬화하는 데 사용됩니다. 다양한 테스트를 실행하기 위해 여러 장치가 사용됩니다.

features

선택 사항입니다. 장치의 지원되는 기능이 포함된 배열입니다. 이 필드에 필요한 정보는 테스트 도구 모음의 [test.json 파일](#)에 정의되어 있으며, 실행할 테스트와 이 테스트를 실행하는 방법을 결정합니다. 테스트 도구 모음에 기능이 필요하지 않은 경우에는 이 필드가 필요하지 않습니다.

features.name

기능의 이름입니다.

features.version

기능 버전.

features.jobSlots

장치를 동시에 사용할 수 있는 테스트 수를 나타내는 설정입니다. 기본값은 1입니다.

devices

테스트할 풀의 장치 배열입니다. 최소 1개 이상의 장치가 필요합니다.

devices.id

테스트 대상 장치의 고유한 사용자 정의 식별자입니다.

connectivity.protocol

이러한 장치와 통신하는 데 사용되는 통신 프로토콜입니다. 풀의 각 장치는 동일한 프로토콜을 사용해야 합니다.

현재 지원되는 값은 uart 물리적 장치의 경우 및, docker Docker 컨테이너의 경우 ssh 입니다.

connectivity.ip

테스트 대상 장치의 IP 입니다.

이 속성은 connectivity.protocol이 ssh로 설정된 경우에만 적용됩니다.

connectivity.port

선택 사항입니다. SSH 연결하는 데 사용하는 포트 번호입니다.

기본값은 4입니다.

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 장치에 액세스하는 데 사용되는 인증 방법입니다.

지원되는 값은 다음과 같습니다.

- pki
- password

`connectivity.auth.credentials`

인증에 사용되는 자격 증명입니다.

`connectivity.auth.credentials.password`

테스트 대상 장치에 로그인하기 위해 사용하는 암호입니다.

이 값은 `connectivity.auth.method`가 `password`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.privKeyPath`

테스트 대상 장치에 로그인하는 데 사용하는 프라이빗 키의 전체 경로입니다.

이 값은 `connectivity.auth.method`가 `pki`로 설정된 경우에만 적용됩니다.

`connectivity.auth.credentials.user`

테스트 대상 장치에 로그인하기 위한 사용자 이름입니다.

`connectivity.serialPort`

선택 사항입니다. 장치가 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 사항입니다. 컨테이너 내부 사용자의 사용자 이름입니다. 기본값은 Dockerfile에 제공된 사용자입니다.

기본값은 4입니다.

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 수

정할 필요가 없습니다. 필요한 권한이 있는 AWS 보안 인증 정보가 제공되면 AWS IoT Device Tester 는 사용량 지표를 수집하여 AWS에 제출합니다. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. 자세한 내용은 [IDT 사용량 지표](#) 섹션을 참조하세요.

테스트 러너는 다음 방법 중 하나로 AWS 보안 인증을 구성할 수 있습니다.

- 보안 인증 파일

IDT는 AWS CLI와 동일한 자격 증명 파일을 사용합니다. 자세한 내용은 [구성 및 자격 증명 파일](#)을 참조하십시오.

자격 증명 파일의 위치는 사용하는 운영 체제에 따라 달라집니다.

- macOS, Linux의 경우: ~/.aws/credentials
- Windows: C:\Users*UserName*\.aws\credentials
- 환경 변수:

환경 변수는 운영 체제에서 유지 관리하고 시스템 명령에서 사용하는 변수입니다. SSH 세션 중에 정의된 변수는 해당 세션이 종료된 후에는 사용할 수 없습니다. IDT는 AWS_ACCESS_KEY_ID 및 AWS_SECRET_ACCESS_KEY 환경 변수를 사용하여 AWS 보안 인증을 저장할 수 있습니다.

Linux, macOS 또는 Unix에서 이러한 변수를 설정하려면 export를 사용합니다.

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Windows에서 이러한 변수를 설정하려면 set을 사용합니다.

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

IDT용 AWS 보안 인증을 구성하기 위해 테스트 러너는 *<device-tester-extract-location>*/configs/ 폴더에 있는 config.json 파일에서 auth 섹션을 편집합니다.

```
{
  "log": {
    "location": "logs"
  },
  "configFiles": {
    "root": "configs",
```

```

    "device": "configs/device.json"
  },
  "testPath": "tests",
  "reportPath": "results",
  "awsRegion": "<region>",
  "auth": {
    "method": "file | environment",
    "credentials": {
      "profile": "<profile-name>"
    }
  }
}
]

```

여기 설명된 것처럼 값이 포함된 모든 필드는 필수입니다.

Note

이 파일의 모든 경로는 *<device-tester-extract-location>*을 기준으로 정의됩니다.

log.location

*<device-tester-extract-location>*에 있는 로그 폴더의 경로.

configFiles.root

구성 파일이 포함된 폴더 경로입니다.

configFiles.device

device.json 파일 경로입니다.

testPath

테스트 도구 모음이 들어 있는 폴더의 경로.

reportPath

IDT에서 테스트 도구 모음을 실행한 후 테스트 결과를 포함할 폴더의 경로입니다.

awsRegion

선택 사항입니다. 테스트 도구 모음에서 사용할 AWS 리전. 설정하지 않으면 테스트 도구 모음은 각 테스트 도구 모음에 지정된 기본 리전을 사용합니다.

auth.method

IDT가 AWS 보안 인증을 검색하는 데 사용하는 메서드입니다. 지원되는 값은 보안 인증 파일에서 보안 인증을 검색하는 file와(과) 환경 변수를 사용하여 보안 인증을 검색하는 environment입니다.

auth.credentials.profile

보안 인증 파일에서 사용할 보안 인증 프로필. 이 속성은 auth.method이 file로 설정된 경우에만 적용됩니다.

사용자 지정 테스트 도구 모음 디버그 및 실행

[필요한 구성](#)이 설정되면 IDT에서 테스트 도구 모음을 실행할 수 있습니다. 전체 테스트 도구 모음의 런타임은 하드웨어와 테스트 도구 모음의 구성에 따라 달라집니다. 참조를 위해, Raspberry Pi 3B에서 전체 AWS IoT Greengrass 자격 테스트 도구 모음을 완료하는 데 약 30분이 걸립니다.

테스트 도구 모음을 작성할 때 IDT를 사용하여 테스트 도구 모음을 디버그 모드에서 실행하여 코드를 실행하기 전에 검사하거나 테스트 실행기에 제공할 수 있습니다.

IDT를 디버그 모드에서 실행합니다.

테스트 도구 모음은 장치와 상호 작용하고, 컨텍스트를 제공하고, 결과를 받기 위해 IDT를 사용하기 때문에 IDT 상호 작용 없이 IDE에서 테스트 도구 모음을 간단히 디버깅할 수는 없습니다. 이를 위해 IDT CLI는 IDT를 디버그 모드에서 실행할 수 있는 debug-test-suite 명령을 제공합니다. 다음 명령을 실행하여 debug-test-suite에 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.json에서 이 정보를 읽지 않습니다.
- 리소스 장치를 선택하기 위한 인수. IDT는 test.json에서 이 정보를 읽지 않습니다.

테스트 도구 모음을 디버깅하려면 다음 단계를 완료하세요.

1. 테스트 도구 모음을 실행하는 데 필요한 설정 구성 파일을 생성합니다. 예를 들어, 테스트 도구 모음에 `device.json`, `resource.json`, 및 `user data.json`이(가) 필요한 경우, 필요에 따라 모두 구성해야 합니다.
2. 다음 명령을 실행하여 IDT를 디버그 모드로 설정하고 테스트를 실행하는 데 필요한 장치를 선택합니다.

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

이 명령을 실행하면 IDT는 테스트 도구 모음의 요청을 기다린 다음 요청에 응답합니다. 또한 IDT는 IDT Client SDK의 케이스 프로세스에 필요한 환경 변수를 생성합니다.

3. IDE에서 `run` 또는 `debug` 구성을 사용하여 다음을 수행합니다.
 - a. IDT에서 생성한 환경 변수의 값을 설정합니다.
 - b. `test.json` 및 `suite.json` 파일에 지정한 모든 환경 변수 또는 인수의 값을 설정합니다.
 - c. 필요에 따라 중단점을 설정합니다.
4. IDE에서 테스트 도구 모음을 실행합니다.

필요한 횟수만큼 테스트 도구 모음을 디버깅하고 다시 실행할 수 있습니다. 디버그 모드에서는 IDT가 타임아웃되지 않습니다.

5. 디버깅을 완료한 후 IDT를 중단하여 디버그 모드를 종료하십시오.

테스트를 실행하기 위한 IDT CLI 명령

다음 단원에서는 IDT CLI 명령에 대해 설명합니다.

IDT v4.0.0

`help`

지정된 명령에 대한 정보를 나열합니다.

`list-groups`

지정된 테스트 제품군에 있는 그룹을 나열합니다.

`list-suites`

사용 가능한 테스트 제품군을 나열합니다.

list-supported-products

IDT 버전에 대해 지원되는 제품(이 경우, AWS IoT Greengrass 버전)과 현재 IDT 버전에 대한 AWS IoT Greengrass 자격 테스트 제품군 버전을 나열합니다.

list-test-cases

주어진 테스트 그룹의 테스트 사례를 나열합니다. 다음 옵션이 지원됩니다.

- `group-id`. 검색할 테스트 그룹입니다. 이 옵션은 필수이며 단일 그룹을 지정해야 합니다.

run-suite

장치의 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(선택으로 구분된 목록). 지정하지 않으면 IDT는 테스트 제품군의 모든 테스트 그룹을 실행합니다.
- `test-id`. 실행할 테스트 케이스(선택으로 구분된 목록). 지정된 경우, `group-id`은(는) 단일 그룹을 지정해야 합니다.
- `pool-id`. 테스트할 장치 풀. `device.json` 파일에 여러 장치 풀이 정의되어 있는 경우, 테스트 실행기는 하나의 풀을 지정해야 합니다.
- `timeout-multiplier`. 사용자 정의 승수를 사용하여 테스트에 대해 `test.json` 파일에 지정된 테스트 실행 제한 시간을 수정하도록 IDT를 구성합니다.
- `stop-on-first-failure`. 첫 번째 실패 시 실행을 중지하도록 IDT를 구성합니다. 이 옵션은 지정된 테스트 그룹을 디버깅하는 데 `group-id`와(과) 함께 사용해야 합니다.
- `userdata`. 테스트 도구 모음을 실행하는 데 필요한 사용자 데이터 정보가 포함된 파일을 설정합니다. 이는 테스트 도구 모음 `suite.json` 파일에서 `userdataRequired`이(가) `true`로 설정된 경우에만 필요합니다.

`run-suite` 옵션에 대한 자세한 내용은 다음 `help` 옵션을 사용하십시오.

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

debug-test-suite

테스트 도구 모음을 디버그 모드에서 실행합니다. 자세한 내용은 [IDT를 디버그 모드에서 실행합니다](#) 섹션을 참조하세요.

IDT 테스트 결과 및 로그 검토

이 섹션에서는 IDT가 콘솔 로그와 테스트 보고서를 생성하는 형식을 설명합니다.

콘솔 메시지 형식

AWS IoT Device Tester는 테스트 제품군을 시작할 때 콘솔에 메시지를 인쇄하는 데 표준 형식을 사용합니다. 다음 발췌문은 IDT에서 생성한 콘솔 메시지의 한 가지 예를 보여줍니다.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

대부분의 콘솔 메시지는 다음 필드로 구성되어 있습니다.

time

로깅된 이벤트의 전체 ISO 8601 타임스탬프.

level

로깅된 이벤트의 메시지 수준. 일반적으로 로깅된 메시지 수준은 `info`, `warn` 또는 `error` 중 하나입니다. IDT는 예상 이벤트가 발생하여 이벤트가 일찍 종료되는 경우 `fatal` 또는 `panic` 메시지를 표시합니다.

msg

로깅된 메시지.

executionId

현재 IDT 프로세스의 고유 ID 문자열입니다. 이 ID는 개별 IDT 실행을 구분하는 데 사용됩니다.

테스트 제품군에서 생성된 콘솔 메시지는 테스트 대상 장치와 테스트 제품군, 테스트 그룹 및 IDT가 실행하는 테스트 케이스에 대한 추가 정보를 제공합니다. 다음 발췌문은 테스트 제품군에서 생성한 콘솔 메시지의 한 가지 예를 보여줍니다:

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

콘솔 메시지의 테스트 제품군 관련 부분에는 다음 필드가 포함됩니다.

suiteId

현재 실행 중인 테스트 제품군의 이름.

groupId

현재 실행 중인 테스트 그룹의 ID.

testCaseId

현재 실행 중인 테스트 케이스의 ID.

deviceId

현재 테스트 케이스에서 사용 중인 테스트 대상 장치의 ID.

IDT에서 테스트 실행을 완료했을 때 콘솔에 테스트 요약을 인쇄하려면 테스트 오케스트레이터에 [Report 상태](#)를 포함해야 합니다. 테스트 요약에는 테스트 제품군, 실행된 각 그룹의 테스트 결과, 생성된 로그 및 보고서 파일의 위치에 대한 정보가 포함됩니다. 다음 예제에서는 테스트 요약 메시지를 보여줍니다.

```
===== Test Summary =====
Execution Time:      5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
-----
Test Groups:
  GroupA:           PASSED
  GroupB:           FAILED
-----
Failed Tests:
  Group Name: GroupB
    Test Name: TestB1
      Reason: Something bad happened
-----
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester 보고서 스키마

awsiotdevicetester_report.xml은 다음 정보가 포함된 서명된 보고서입니다.

- IDT 버전
- 테스트 제품군 버전입니다.
- 보고서에 서명하는 데 사용된 보고서 서명 및 키.
- device.json 파일에 지정된 SKU 및 장치 풀 이름.
- 테스트된 제품 버전 및 장치 특성.
- 테스트 결과의 집계 요약 이 정보는 *suite-name_report.xml* 파일에 포함된 정보와 동일합니다.

```
<apnreport>
  <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
  <testsuiteversion>test-suite-version</testsuiteversion>
  <signature>signature</signature>
  <keyname>keyname</keyname>
  <session>
    <testsession>execution-id</testsession>
    <starttime>start-time</starttime>
    <endtime>end-time</endtime>
  </session>
  <awsproduct>
    <name>product-name</name>
    <version>product-version</version>
    <features>
      <feature name="<feature-name>" value="supported | not-supported | <feature-value>" type="optional | required" />
    </features>
  </awsproduct>
  <device>
    <sku>device-sku</sku>
    <name>device-name</name>
    <features>
      <feature name="<feature-name>" value="<feature-value>" />
    </features>
    <executionMethod>ssh | uart | docker</executionMethod>
  </device>
  <devenvironment>
    <os name="<os-name>" />
  </devenvironment>
  <report>
    <suite-name-report-contents>
  </report>
</apnreport>
```


awsiotdevicetester_report.xml 파일에는 테스트하는 제품에 대한 정보와 테스트 제품군을 실행한 후 확인된 제품 기능에 대한 정보를 포함하는 <awsproduct> 태그가 포함되어 있습니다.

<awsproduct> 태그에 사용되는 속성

name

테스트하는 제품의 이름입니다.

version

테스트하는 제품의 버전입니다.

features

확인된 기능입니다. 필수로 required 표시된 특성은 테스트 제품군에서 장치를 검증하는 데 필요합니다. 다음 코드 조각은 awsiotdevicetester_report.xml 파일에 이 정보가 나타나는 방식을 보여 줍니다.

```
<feature name="ssh" value="supported" type="required"></feature>
```

optional 표시된 특성은 검증에 필요하지 않습니다. 다음 코드 조각은 선택적 기능을 보여 줍니다.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

테스트 제품군 보고서 스키마

suite-name_Result.xml 보고서는 [JUnit XML 형식](#)입니다. [Jenkins](#), [Bamboo](#) 등과 같은 지속적 통합 및 배포 플랫폼에 이 보고서를 통합할 수 있습니다. 보고서는 테스트 결과의 집계 요약에 포함합니다.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
disabled="0">
  <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
disabled="0">
    <!--success-->
```

```

<testcase classname="<classname>" name="<name>" time="<run-duration>" />
<!--failure-->
<testcase classname="<classname>" name="<name>" time="<run-duration>">
  <failure type="<failure-type>">
    reason
  </failure>
</testcase>
<!--skipped-->
<testcase classname="<classname>" name="<name>" time="<run-duration>">
  <skipped>
    reason
  </skipped>
</testcase>
<!--error-->
<testcase classname="<classname>" name="<name>" time="<run-duration>">
  <error>
    reason
  </error>
</testcase>
</testsuite>
</testsuites>

```

awsiotdevicetester_report.xml 또는 *suite-name*_report.xml의 보고서 섹션에는 실행된 테스트 및 결과가 나열됩니다.

첫 번째 XML 태그 <testsuites>에는 테스트 실행의 요약이 포함됩니다. 예:

```

<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
  disabled="0">

```

<testsuites> 태그에 사용되는 속성

name

테스트 제품군의 이름입니다.

time

테스트 제품군을 실행하는 데 걸린 시간(초).

tests

실행된 테스트의 수입니다.

failures

실행되었지만 통과하지 못한 테스트의 수입니다.

errors

IDT에서 실행하지 못한 테스트의 수입니다.

disabled

이 속성은 사용되지 않으므로 무시해도 좋습니다.

테스트 실패 또는 오류의 경우 <testsuites> XML 태그를 검토하여 실패한 테스트를 식별할 수 있습니다. <testsuite> 태그 내부의 <testsuites> XML 태그는 테스트 그룹에 대한 테스트 결과 요약 을 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> 태그 가 있습니다. 예:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

<testcase> 태그에 사용되는 속성

name

테스트의 이름입니다.

attempts

IDT에서 테스트 사례를 실행한 횟수입니다.

테스트가 실패하거나 오류가 발생하는 경우 문제 해결에 대한 정보와 함께 <failure> 또는 <error> 태그가 <testcase> 태그에 추가됩니다. 예:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
<failure type="Failure">Reason for the test failure</failure>
```

```
<error>Reason for the test execution error</error>
</testcase>
```

IDT 사용량 지표

필요한 권한이 있는 AWS 자격 증명을 제공하는 경우는 사용량 지표를 AWS IoT Device Tester 수집하여 제출합니다 AWS. 이는 옵트인 기능이며 IDT 기능을 개선하는 데 사용됩니다. IDT는 다음과 같은 정보를 수집합니다.

- AWS 계정 IDT를 실행하는 데 사용되는 ID
- 테스트를 실행하는 데 사용되는 IDT AWS CLI 명령
- 실행되는 테스트 제품군
- *<device-tester-extract-location>* 폴더의 테스트 제품군
- 디바이스 풀에 구성된 디바이스 수
- 테스트 케이스 이름 및 런타임
- 테스트 결과 정보(예: 테스트 통과, 실패, 오류 발생 또는 건너뛰었는지 여부)
- 제품 기능 테스트
- IDT 종료 행동(예: 예상치 못한 종료 또는 조기 종료)

IDT가 전송하는 모든 정보는 *<device-tester-extract-location>/results/<execution-id>/* 폴더의 *metrics.log* 파일에도 기록됩니다. 로그 파일을 보면 테스트 실행 중에 수집된 정보를 볼 수 있습니다. 이 파일은 사용량 지표를 수집하도록 선택한 경우에만 생성됩니다.

지표 수집을 비활성화하기 위해 추가 조치를 취할 필요가 없습니다. 자격 AWS 증명을 저장하지 말고 자격 증명에 저장된 경우 해당 AWS 자격 증명에 액세스하도록 *config.json* 파일을 구성하지 마십시오.

자격 AWS 증명 구성

가 아직 없는 경우 [생성](#) AWS 계정해야 합니다. 이미가 있는 경우 IDT가 사용자를 대신하여에 사용량 지표를 [전송할 수 있도록 계정에 필요한 권한을 구성](#)하기만 AWS 계정하면 AWS 됩니다.

1단계: 생성 AWS 계정

이 단계에는 AWS 계정을 생성하고 구성합니다. 이미 AWS 계정이 있으면 [the section called “2단계: IDT에 대한 권한 구성”](#) 섹션으로 건너뛩니다.

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. <https://portal.aws.amazon.com/billing/signup>을 엽니다.
2. 온라인 지시 사항을 따르세요.

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

에 가입하면 AWS 계정AWS 계정 루트 사용자의 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 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 사례 를 참조하세요.	AWS IAM Identity Center 사용 설명서의 시작하기 지침을 따릅니다.	AWS Command Line Interface 사용 설명서에서 사용하도록 AWS CLI를 구성 AWS IAM Identity Center 하여 프로그래밍 방식 액세스를 구성합니다.
IAM에서 (권장되지 않음)	장기 보안 인증 정보를 사용하여 AWS에 액세스합니다.	IAM 사용 설명서의 비상 액세스를 위한 IAM 사용자 생성 에 나와 있는 지침을 따르세요.	IAM 사용 설명서에 나온 IAM 사용자의 액세스 키 관리 를 수행하여 프로그래밍 방식의 액세스를 구성합니다.

2단계: IDT에 대한 권한 구성

이 단계에서는 IDT가 테스트를 실행하고 IDT 사용 데이터를 수집하는 데 사용하는 권한을 구성합니다. AWS Management Console 또는 AWS Command Line Interface (AWS CLI)를 사용하여 IDT에 대한 IAM 정책과 사용자를 생성한 다음 사용자에게 정책을 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 수행하세요.

1. [IAM 콘솔](#)에 로그인합니다.
2. 특정 권한으로 역할을 생성하는 권한을 부여하는 고객 관리형 정책을 만듭니다.
 - a. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
 - b. JSON 탭에서 자리 표시자 콘텐츠를 다음 정책으로 바꿉니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. 정책 검토를 선택합니다.
 - d. 이름에서 **IDTUsageMetricsIAMPermissions**을 입력합니다. Summary(요약) 아래에서 정책에 의해 부여된 권한을 검토합니다.
 - e. 정책 생성을 선택합니다.
3. IAM 사용자를 생성하고 사용자에게 권한을 연결합니다.

- a. IAM 사용자를 생성합니다. [IAM 사용 설명서](#)에서 IAM 사용자 생성(콘솔)의 1~5단계를 따르세요. 이미 IAM 사용자를 생성한 경우 다음 단계로 건너뛰세요.
- 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 IDUsageMetricsIAMPermissions 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. 나중에이 정보를 사용하여 AWS 자격 증명 파일을 구성합니다.

IDT에 대한 권한을 구성하려면(AWS CLI)

다음 단계에 따라를 사용하여 AWS CLI 에 대한 IDT 권한을 구성합니다 AWS IoT Greengrass.

1. 아직 설치되지 않은 AWS CLI 경우 컴퓨터에를 설치하고 구성합니다. AWS Command Line Interface 사용 설명서 [AWS CLI설치](#) 단계를 따르세요.

Note

AWS CLI 는 명령줄 셸의 AWS 서비스와 상호 작용하는 데 사용할 수 있는 오픈 소스 도구입니다.

2. IDT 및 AWS IoT Greengrass 역할을 관리할 수 있는 권한을 부여하는 다음 고객 관리형 정책을 생성합니다.

Linux or Unix

```
aws iam create-policy --policy-name IDUsageMetricsIAMPermissions --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. IAM 사용자를 만들고 AWS IoT Greengrass용 IDT에 필요한 권한을 연결합니다.

a. IAM 사용자를 생성합니다.

```
aws iam create-user --user-name user-name
```

b. 생성한 IDTUsageMetricsIAMPermissions 정책을 새 IAM 사용자에게 연결합니다. *user-name*을 IAM 사용자 이름으로 바꾸고 명령에서 *<account-id>*를 자신이 사용하는 AWS 계정계정의 ID로 바꾸세요.

```
aws iam attach-user-policy --user-name user-name --policy-arn
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. 사용자에게 대한 비밀 액세스 키를 만듭니다.

```
aws iam create-access-key --user-name user-name
```

출력을 안전한 위치에 저장합니다. 나중에이 정보를 사용하여 AWS 자격 증명 파일을 구성합니다.

IDT에 AWS 자격 증명 제공

IDT가 자격 AWS 증명에 액세스하고 지표를 제출하도록 허용하려면 다음을 AWS수행합니다.

1. IAM 사용자의 AWS 자격 증명을 환경 변수로 저장하거나 자격 증명 파일에 저장합니다.

a. 다음 명령을 실행하여 환경 변수를 설정합니다.

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 다음 정보를 ~/.aws/credentials 파일에 추가합니다.

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. config.json 파일의 auth 섹션을 구성합니다. 자세한 내용은 [\(선택 사항\) config.json 구성](#) 단원을 참조하십시오.

IDT for AWS IoT Greengrass V2 문제 해결

IDT for AWS IoT Greengrass V2는 오류 유형에 따라 다양한 위치에 오류를 기록합니다. IDT는 콘솔, 로그 파일 및 테스트 보고서에 오류를 작성합니다.

오류를 찾을 수 있는 곳

상위 수준의 오류는 테스트를 실행하는 동안 콘솔에 표시되고, 실패한 테스트의 요약은 모든 테스트가 완료될 때 표시됩니다. awsiotdevicetester_report.xml에 테스트 실패의 원인이 되는 모든 오류에 대한 요약이 들어 있습니다. IDT는 각 테스트 실행에 대한 로그 파일을 테스트 실행에 대한 UUID가 포함된 디렉터리에 저장하고, 테스트 실행 도중 콘솔에 표시됩니다.

IDT 테스트 로그 디렉터리는 *<device-tester-extract-location>/results/<execution-id>/logs/*입니다. 이 디렉터리에는 테이블에 표시된 다음 파일이 포함되어 있습니다. 이는 디버깅에 유용합니다.

파일	설명
test_manager.log	테스트가 실행되는 동안 콘솔에 기록된 로그. 이 파일의 끝에 있는 결과 요약에는 실패한 테스트 목록이 포함되어 있습니다.
<i>test-group-id</i> / <i>test-case-id</i> / <i>test-name</i> .log	이 파일의 경고 및 오류 로그에서 실패에 대한 일부 정보를 확인할 수 있습니다.
	테스트 그룹의 특정 테스트에 대한 상세 로그. Greengrass 구성 요소를 배포하는 테스트의 경

파일	설명
<code>test-group-id /test-case-id /greengrass.log</code>	우 테스트 사례 로그 파일을 greengrass-test-run.log 라고 합니다.
<code>test-group-id /test-case-id /greengrass.log</code>	AWS IoT Greengrass 코어 소프트웨어에 대한 세부 로그입니다. IDT는 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하는 테스트를 실행할 때 테스트 중인 디바이스에서 파일을 복사합니다. 이 로그 파일의 메시지에 대한 자세한 내용은 문제 해결 AWS IoT Greengrass V2 의 내용을 참조하세요.
<code>test-group-id /test-case-id/component-name .log</code>	테스트 실행 중에 배포된 Greengrass 구성 요소에 대한 상세 로그. IDT는 특정 구성 요소를 배포하는 테스트를 실행할 때 테스트 중인 디바이스에서 구성 요소 로그 파일을 복사합니다. 각 구성 요소 로그 파일의 이름은 배포된 구성 요소의 이름에 해당합니다. 이 로그 파일의 메시지에 대한 자세한 내용은 문제 해결 AWS IoT Greengrass V2 의 내용을 참조하세요.

IDT for AWS IoT Greengrass V2 오류 해결

용 IDT를 실행하기 전에 AWS IoT Greengrass올바른 구성 파일을 준비하십시오. 구문 분석 및 구성 오류가 발생할 경우 첫 번째 단계는 환경에 적합한 구성 템플릿을 찾아서 사용하는 것입니다.

그래도 문제가 발생할 경우 다음 디버깅 프로세스를 참조하십시오.

주제

- [별칭 확인 오류](#)
- [충돌 오류](#)
- [테스트를 시작할 수 없음 오류](#)
- [Docker 검증 이미지에 오류가 있음](#)
- [자격 증명 읽기 실패](#)
- [사전 설치된 Greengrass의 Guice 오류](#)

- [잘못된 서명 예외](#)
- [기계 학습 검증 오류](#)
- [OTF\(Open Test Framework\) 배포 실패](#)
- [구문 분석 오류](#)
- [권한 거부 오류](#)
- [검증 보고서 생성 오류](#)
- [필수 파라미터 누락 오류](#)
- [macOS에서의 보안 예외](#)
- [SSH 연결 오류](#)
- [스트림 관리자 검증 오류](#)
- [제한 시간 오류](#)
- [버전 확인 오류](#)

별칭 확인 오류

사용자 지정 테스트 제품군을 실행하면 콘솔 및 `test_manager.log`에서 다음 오류가 표시될 수 있습니다.

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

이 오류는 IDT 테스트 오케스트레이터에 구성된 별칭이 올바르게 확인되지 않거나 확인된 값이 구성 파일에 없는 경우 발생할 수 있습니다. 이 오류를 해결하려면 `device.json` 및 `userdata.json`에 테스트 제품군에 필요한 올바른 정보가 포함되어 있어야 합니다. AWS IoT Greengrass 검증에 필요한 구성에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 검증 제품군을 실행하도록 IDT 설정 구성](#).

충돌 오류

두 개 이상의 디바이스에서 AWS IoT Greengrass 검증 제품군을 동시에 실행하면 다음 오류가 표시될 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]" }
```

AWS IoT Greengrass 검증 제품군에 대한 동시 테스트 실행은 아직 지원되지 않습니다. 각 디바이스에 대해 순차적으로 검증 제품군을 실행하세요.

테스트를 시작할 수 없음 오류

테스트 시작을 시도할 때 발생한 실패를 가리키는 오류가 발생할 수 있습니다. 몇 가지 원인이 있을 수 있으므로 다음을 수행하십시오.

- 실행 명령의 풀 이름이 실제로 존재하는지 확인합니다. IDT는 `device.json` 파일에서 풀 이름을 직접 참조합니다.
- 풀에 있는 장치에 올바른 구성 파라미터가 있는지 확인합니다.

Docker 검증 이미지에 오류가 있음

Docker 애플리케이션 관리자 검증 테스트는 Amazon ECR의 `amazon/amazon-ec2-metadata-mock` 컨테이너 이미지를 사용하여 테스트 중인 디바이스를 검증합니다.

이미지가 테스트 중인 디바이스의 Docker 컨테이너에 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 디바이스에서 `amazon/amazon-ec2-metadata-mock` 컨테이너를 실행한 경우 검증 테스트를 실행하기 전에 테스트 중인 디바이스에서 이 이미지를 제거해야 합니다.

자격 증명 읽기 실패

Windows 디바이스를 테스트할 때 테스트 중인 디바이스에 연결하는 데 사용하는 사용자가 해당 디바이스의 자격 증명 관리자에 설정되지 않은 경우 `greengrass.log` 파일에 `Failed to read credential` 오류가 발생할 수 있습니다.

이 오류를 해결하려면 테스트 중인 디바이스의 자격 증명 관리자에서 IDT 사용자의 사용자와 암호를 구성합니다.

자세한 내용은 [Windows 디바이스에 대한 사용자 자격 증명 구성](#) 단원을 참조하십시오.

사전 설치된 Greengrass의 Guice 오류

사전 설치된 Greengrass로 IDT를 실행하는 동안 Guice 또는 `ErrorInCustomProvider` 오류가 발생하면 `userdata.json` 파일에서 `InstalledDirRootOnDevice`가 Greengrass 설치

폴더로 설정되어 있는지 확인합니다. IDT는 <InstallationDirRootOnDevice>/config/effectiveConfig.yaml에서 effectiveConfig.yaml 파일을 확인합니다.

자세한 내용은 [Windows 디바이스에 대한 사용자 자격 증명 구성](#) 단원을 참조하십시오.

잘못된 서명 예외

Lambda 검증 테스트를 실행할 때 IDT 호스트 시스템에 네트워크 액세스 문제가 발생하는 경우 invalidsignatureexception 오류가 발생할 수 있습니다. 라우터를 재설정하고 테스트를 다시 실행하세요.

기계 학습 검증 오류

기계 학습(ML) 검증 테스트를 실행할 때 디바이스가 AWS제공 ML 구성 요소를 배포하기 위한 [요구](#) 사항을 충족하지 않는 경우 검증 실패가 발생할 수 있습니다. 다음을 수행하여 ML 검증 오류 해결:

- 테스트 실행 중에 배포된 구성 요소의 구성 요소 로그에서 오류 세부 정보를 찾습니다. 구성 요소 로그는 `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` 디렉터리에 있습니다.
- 실패한 테스트 사례의 test.json 파일에 -Dgg.persist=installed.software 인수를 추가합니다. test.json 파일은 `<device-tester-extract-location>/tests/GGV2Q_version` directory. 에 있습니다.

OTF(Open Test Framework) 배포 실패

OTF 테스트에서 배포를 완료하지 못하는 경우 TempResourcesDirOnDevice 및 InstallationDirRootOnDevice의 상위 폴더에 설정된 권한이 원인일 수 있습니다. 이 폴더의 권한을 올바르게 설정하려면 다음 명령을 실행합니다. `folder-name`을 상위 폴더의 이름으로 바꿉니다.

```
sudo chmod 755 folder-name
```

구문 분석 오류

JSON 구성의 오타로 인해 구문 분석 오류가 발생할 수 있습니다. 대부분의 경우 문제는 JSON 파일에서 대괄호, 쉼표 또는 따옴표를 생략한 결과입니다. IDT는 JSON 확인을 수행하고 디버깅 정보를 인쇄합니다. IDT는 오류가 발생한 줄, 줄 번호, 구문 오류의 열 번호를 인쇄합니다. 이 정보만 있으면 오류를 수정할 수 있지만, 여전히 오류를 찾을 수 없는 경우 IDE나 텍스트 편집기(예: Atom 또는 Sublime)에서 또는 온라인 도구(예: JSONLint)를 통해 수동으로 확인을 수행할 수 있습니다.

권한 거부 오류

IDT는 테스트 대상 장치에서 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 username은 IDT가 테스트 대상 장치에 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 visudo`를 사용하는 것이 좋습니다.

검증 보고서 생성 오류

IDT는 AWS IoT Greengrass V2 검증 제품군(GGV2Q)의 네 가지 최신 *major.minor* 버전을 지원하여 AWS Partner 디바이스 카탈로그에 디바이스를 포함 AWS Partner Network 하기 위해 제출할 수 있는 검증 보고서를 생성합니다. 이전 버전의 검증 제품군은 검증 보고서를 생성하지 않습니다.

지원 정책에 대해 궁금한 점이 있으면 [AWS Support](#)에 문의하세요.

필수 파라미터 누락 오류

IDT에서 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 스키마를 검사하여 지원되는 최신 버전이 사용되었는지 확인합니다.

macOS에서의 보안 예외

macOS 호스트 컴퓨터에서 IDT를 실행하면 IDT 실행이 차단됩니다. IDT를 실행하려면 IDT 런타임 기능의 일부인 실행 파일에 보안 예외를 부여합니다. 호스트 컴퓨터에 경고 메시지가 표시되면 해당하는 각 실행 파일에 대해 다음을 수행합니다.

IDT 실행 파일에 보안 예외 부여

1. macOS 컴퓨터의 Apple 메뉴에서 시스템 기본 설정을 엽니다.
2. 보안 및 개인 정보 보호를 선택한 다음 일반 탭에서 잠금 아이콘을 선택하여 보안 설정을 변경합니다.
3. `devicetester_mac_x86-64`가 차단된 경우 "`devicetester_mac_x86-64`" was blocked from use because it is not from an identified developer. 메시지를 찾아 모두 허용을 선택합니다.
4. 관련된 모든 실행 파일을 테스트할 때까지 IDT 테스트를 재개합니다.

SSH 연결 오류

IDT가 테스트 대상 디바이스에 연결할 수 없으면 연결 실패가 `/results/<execution-id>/logs/<test-case-id>.log`에 기록됩니다. 테스트 대상 디바이스에 대한 연결은 IDT가 수행하는 첫 번째 작업 중 하나이므로 SSH 메시지는 이 로그 파일의 맨 위에 표시됩니다.

대부분의 Windows 구성은 PuTTY 터미널 애플리케이션을 사용하여 Linux 호스트에 연결합니다. 이 애플리케이션에서는 표준 PEM 프라이빗 키 파일을 PPK라는 전용 Windows 형식으로 변환해야 합니다. `device.json` 파일에서 SSH를 구성하는 경우 PEM 파일을 사용하세요. PPK 파일을 사용하는 경우 IDT는 AWS IoT Greengrass 디바이스와의 SSH 연결을 생성할 수 없으며 테스트를 실행할 수 없습니다.

IDT v4.4.0부터 테스트 중인 디바이스에서 SFTP를 활성화하지 않은 경우 로그 파일에 다음 오류가 표시될 수 있습니다.

```
SSH connection failed with EOF
```

이 오류를 해결하려면 디바이스에서 SFTP를 활성화합니다.

스트림 관리자 검증 오류

스트림 관리자 검증 테스트를 실행할 때 `com.aws.StreamManagerExport.log` 파일에 다음 오류가 표시될 수 있습니다.

Failed to upload data to S3

이 오류는 스트림 관리자가 IDT가 테스트 중인 디바이스로 내보내는 환경 AWS 자격 증명을 사용하는 대신 디바이스의 `~/root/.aws/credentials` 파일에 있는 자격 증명을 사용할 때 발생할 수 있습니다. 이 문제를 방지하려면 디바이스에서 `credentials` 파일을 삭제하고, 검증 테스트를 다시 실행합니다.

제한 시간 오류

각 테스트의 제한 시간 기본값에 적용되는 제한 시간 승수를 지정하여 각 테스트에 대한 제한 시간을 늘릴 수 있습니다. 이 플래그에 대해 구성된 값은 1.0보다 크거나 같아야 합니다.

제한 시간 승수를 사용하려면 테스트를 실행할 때 `--timeout-multiplier` 플래그를 사용합니다. 예제:

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

자세한 내용을 보려면 `run-suite --help`을(를) 실행하세요.

일부 제한 시간 오류는 구성 문제로 인해 IDT 테스트 사례를 완료할 수 없을 때 발생합니다. 제한 시간 승수를 늘려서 이러한 오류를 해결할 수는 없습니다. 테스트 실행의 로그를 사용하여 기본 구성 문제를 해결합니다.

- MQTT 또는 Lambda 구성 요소 로그에 Access denied 오류가 있는 경우 Greengrass 설치 폴더에 올바른 파일 권한이 없을 수 있습니다. `userdata.json` 파일에 정의한 설치 경로의 각 폴더에 대해 다음 명령을 실행합니다.

```
sudo chmod 755 folder-name
```

- Greengrass 로그에 Greengrass CLI 배포가 완료되지 않은 것으로 표시되는 경우 다음을 수행합니다.
 - bash가 테스트 중인 디바이스에 설치되어 있는지 확인합니다.
 - `userdata.json` 파일에 `GreengrassCliVersion` 구성 파라미터가 포함된 경우 제거합니다. 이 파라미터는 IDT v4.1.0 이상 버전에서는 더 이상 사용되지 않습니다. 자세한 내용은 [userdata.json 구성](#) 단원을 참조하십시오.
- “Lambda 게시 검사: 제한 시간” 오류 메시지와 함께 Lambda 배포 테스트가 실패하고 테스트 로그 파일(`idt-gg2-lambda-function-idt-<resource-id>.log`)에 `Error: Could not find`

or load main class com.amazonaws.greengrass.runtime.LambdaRuntime. 오류가 표시되면 다음을 수행합니다.

- userdata.json 파일에서 InstallationDirRootOnDevice에 사용된 폴더를 확인합니다.
- 디바이스에 올바른 사용자 권한이 설정되어 있는지 확인합니다. 자세한 내용은 [디바이스에서 사용자 권한 구성](#)을 참조하세요.

버전 확인 오류

IDT 사용자의 AWS 사용자 자격 증명에 필요한 IAM 권한이 없는 경우 IDT에서 다음 오류가 발생합니다.

Failed to check version compatibility

필요한 IAM 권한이 없는 AWS 사용자입니다.

AWS IoT Device Tester용 AWS IoT Greengrass에 대한 지원 정책

AWS IoT Greengrass용 AWS IoT Device Tester는 AWS IoT Greengrass 디바이스를 [AWS Partner 디바이스 카탈로그](#)에 포함하기 위해 검증하고 [자격을 부여](#)하는 테스트 자동화 도구입니다. 최신 버전의 AWS IoT Greengrass 및 AWS IoT Device Tester를 사용하여 디바이스를 테스트하고 자격을 부여하는 것이 좋습니다.

AWS IoT Greengrass의 지원되는 각 버전에 대해 하나 이상의 AWS IoT Device Tester 버전을 사용할 수 있습니다. AWS IoT Greengrass의 지원되는 버전은 [Greengrass nucleus 버전](#)을 참조하세요. AWS IoT Device Tester의 지원되는 버전은 [AWS IoT Device Tester for AWS IoT Greengrass V2 지원 버전](#) 섹션을 참조하세요.

AWS IoT Greengrass 및 AWS IoT Device Tester의 지원되는 버전 중 하나를 사용하여 디바이스를 테스트하고 자격을 부여할 수 있습니다. AWS IoT Device Tester의 지원되지 않는 버전을 계속 사용할 수 있지만, 이러한 버전에는 버그 수정 또는 업데이트가 제공되지 않습니다. 지원 정책에 대해 궁금한 점이 있으면 [AWS Support](#)에 문의하세요.

Greengrass 기반 IoT 솔루션

Eurotech의 Everyware GreenEdge는 AWS IoT Greengrass 미리 보기 릴리스 단계로 변경될 수 있습니다. 이 솔루션은 AWS에서 지원되지 않습니다. 이 디바이스에 문제가 있는 경우 Eurotech에 문의해야 합니다.

AWS IoT Greengrass는 Greengrass 설치 경험을 최적화하는 파트너 솔루션을 제공합니다. 다음은 AWS와 Eurotech의 파트너십으로 제공하는 솔루션입니다. 이 솔루션은 AWS IoT Greengrass 코어 엣지 런타임 및 추가 기능이 사전 설치된 상태로 제공됩니다.

Eurotech

AWS는 Eurotech와의 파트너십을 통해 AWS IoT Greengrass 코어 소프트웨어가 사전 설치된 디바이스를 찾는 고객에게 IoT 솔루션을 제공합니다. Eurotech의 Everyware GreenEdge는 AWS에서 사전 구성 및 검증한 IoT 엣지 소프트웨어입니다. 이 솔루션은 Greengrass와 Eurotech ESF(Everyware Software Framework)의 기능을 결합하여 Modbus, OPC-UA Client/Server, S7, TwinCAT, J1939, DNP3 Master/Outstation 등과 같은 프로토콜 어댑터를 통해 고객에게 광범위한 사우스바운드 연결을 제공합니다. 이 솔루션을 사용하면 AWS 클라우드로 데이터를 보내고 모든 노스바운드 AWS 서비스(예: AWS IoT Core, AWS IoT SiteWise, AWS IoT Analytics, Amazon S3 및 Amazon Kinesis Video Streams)에 연결할 수도 있습니다. 이 솔루션은 Eurotech의 디바이스 관리 솔루션인 Everyware Cloud와 결합하여 디바이스 온보딩 및 대량 배포를 간소화하는 새로운 제로 터치 프로비저닝 서비스를 도입합니다.

Eurotech에 대한 자세한 내용은 [Eurotech](#)를 참조하세요.

문제 해결 AWS IoT Greengrass V2

이 섹션의 문제 해결 정보 및 솔루션을 사용하여 문제를 해결할 수 있습니다 AWS IoT Greengrass Version 2.

주제

- [AWS IoT Greengrass 코어 소프트웨어 및 구성 요소 로그 보기](#)
- [AWS IoT Greengrass 핵심 소프트웨어 문제](#)
- [AWS IoT Greengrass 클라우드 문제](#)
- [코어 디바이스 배포 문제](#)
- [코어 디바이스 구성 요소 문제](#)
- [코어 디바이스 Lambda 함수 구성 요소 문제](#)
- [구성 요소 버전 중단](#)
- [Greengrass 명령줄 인터페이스 문제](#)
- [AWS Command Line Interface 문제](#)
- [세부 배포 오류 코드](#)
- [세부 구성 요소 상태 코드](#)

AWS IoT Greengrass 코어 소프트웨어 및 구성 요소 로그 보기

AWS IoT Greengrass 코어 소프트웨어는 코어 디바이스에 대한 실시간 정보를 보는 데 사용할 수 있는 로그를 로컬 파일 시스템에 기록합니다. 또한 코어 디바이스가 CloudWatch Logs에 로그를 쓰도록 구성하여 원격으로 코어 디바이스의 문제를 해결할 수도 있습니다. 이러한 로그는 구성 요소, 배포, 코어 디바이스 관련 문제를 식별하는 데 도움이 될 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 단원을 참조하십시오.

AWS IoT Greengrass 핵심 소프트웨어 문제

AWS IoT Greengrass 코어 소프트웨어 문제를 해결합니다.

주제

- [ListDeployments API에서 ThrottlingException 발생](#)
- [코어 디바이스를 설정할 수 없음](#)

- [AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 시작할 수 없음](#)
- [nucleus를 시스템 서비스로 설정할 수 없음](#)
- [에 연결할 수 없음 AWS IoT Core](#)
- [메모리 부족 오류](#)
- [Greengrass CLI를 설치할 수 없음](#)
- [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 못함](#)
- [com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager](#)
- [com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot 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](#)
- [Greengrass core device stuck on nucleus v2.12.3](#)

- [Greengrass nucleus v2.14.0 systemd template issue](#)

ListDeployments API에서 ThrottlingException 발생

ListDeployments API에서 ThrottlingException 발생: 많은 수의 배포가 계정에 있는 경우 이 오류가 발생할 수 있습니다.

이 문제를 해결하려면 다음 중 한 가지 방법을 사용합니다.

- SDK를 사용하는 경우 MaxResult 파라미터를 지정합니다. 예: [Java SDK](#)에서 작은 값(예: 5)으로 설정.
- [AWS Service Quotas](#)를 사용하여 DescribeJob API의 속도 제한을 늘리도록 요청합니다. 서비스 할당량 콘솔로 이동하여 할당량을 선택하고 AWS IoT 제한 이름은 DescribeJob 제한입니다. 기본값 10을 50으로 증가시킬 수 있습니다.

코어 디바이스를 설정할 수 없음

AWS IoT Greengrass 코어 소프트웨어 설치 관리자가 실패하고 코어 디바이스를 설정할 수 없는 경우 소프트웨어를 제거하고 다시 시도해야 할 수 있습니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 제거](#) 단원을 참조하십시오.

AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 시작할 수 없음

AWS IoT Greengrass 코어 소프트웨어가 시작되지 않는 경우 [시스템 서비스 로그를 확인하여](#) 문제를 식별합니다. 일반적인 문제 중 하나는 PATH 환경 변수(Linux) 또는 PATH 시스템 변수(Windows)에서 Java를 사용할 수 없는 경우입니다.

nucleus를 시스템 서비스로 설정할 수 없음

AWS IoT Greengrass 코어 소프트웨어 설치 관리자가 시스템 서비스로 설정되지 않은 경우 이 오류가 표시될 AWS IoT Greengrass 수 있습니다. Linux 디바이스에서 이 오류는 일반적으로 코어 디바이스에 [systemd](#) init 시스템이 없는 경우에 발생합니다. 설치 관리자는 시스템 서비스를 설정하지 못하더라도 AWS IoT Greengrass 코어 소프트웨어를 성공적으로 설정할 수 있습니다.

다음 중 하나를 수행하세요.

- AWS IoT Greengrass 코어 소프트웨어를 시스템 서비스로 구성하고 실행합니다. AWS IoT Greengrass의 모든 기능을 사용하려면 소프트웨어를 시스템 서비스로 구성해야 합니다. [systemd](#)를

설치하거나 다른 init 시스템을 사용할 수 있습니다. 자세한 내용은 [Greengrass nucleus를 시스템 서비스로 구성](#) 단원을 참조하십시오.

- 시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어를 실행합니다. 설치 관리자가 Greengrass 루트 폴더에 설정한 로더 스크립트를 사용하여 소프트웨어를 실행할 수 있습니다. 자세한 내용은 [시스템 서비스 없이 AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

에 연결할 수 없음 AWS IoT Core

예를 들어 AWS IoT Greengrass 코어 소프트웨어가 배포 작업을 검색하기 AWS IoT Core 위해에 연결할 수 없는 경우 오류가 표시될 수 있습니다. 해결 방법:

- 코어 디바이스가 인터넷 및에 연결할 수 있는지 확인합니다 AWS IoT Core. 디바이스가 연결되는 AWS IoT Core 엔드포인트에 대한 자세한 내용은 섹션을 참조하세요 [AWS IoT Greengrass 코어 소프트웨어 구성](#).
- 코어 디바이스의 AWS IoT 사물이 `iot:Connect`, `iot:Publish`, `iot:Receive`, 및 `iot:Subscribe` 권한을 허용하는 인증서를 사용하는지 확인합니다.
- 코어 디바이스가 [네트워크 프록시](#)를 사용하는 경우 코어 디바이스에 [디바이스 역할](#)이 있고 해당 역할이 `iot:Connect`, `iot:Publish`, `iot:Receive`, `iot:Subscribe` 권한을 허용하는지 확인하세요.

메모리 부족 오류

이 오류는 일반적으로 디바이스에 Java 힙에 객체를 할당할 메모리가 충분하지 않은 경우에 발생합니다. 메모리가 제한된 디바이스에서는 메모리 할당을 제어하기 위해 최대 힙 크기를 지정해야 할 수도 있습니다. 자세한 내용은 [JVM 옵션으로 메모리 할당 제어](#) 단원을 참조하십시오.

Greengrass CLI를 설치할 수 없음

AWS IoT Greengrass Core용 설치 명령에서 `--deploy-dev-tools` 인수를 사용하면 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 내용은 [Greengrass CLI 설치](#) 단원을 참조하십시오.

User root is not allowed to execute

AWS IoT Greengrass 코어 소프트웨어를 실행하는 사용자(일반적으로 root)에게 sudo 사용자 및 그룹에서 실행할 권한이 없는 경우가 오류가 표시될 수 있습니다. 기본 ggc_user 시스템 사용자의 경우 이 오류는 다음과 같이 표시됩니다.

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

/etc/sudoers 파일이 사용자에게 sudo를 다른 그룹으로 실행할 수 있는 권한을 부여하는지 확인합니다. /etc/sudoers의 사용자에 대한 권한은 다음 예제와 유사해야 합니다.

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

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

코어 디바이스가 구성 요소를 실행하려고 하는데 Greengrass nucleus가 구성 요소를 실행하는 데 사용할 기본 시스템 사용자를 지정하지 않으면 이 오류가 발생할 수 있습니다.

이 문제를 해결하려면 구성 요소를 실행하는 기본 시스템 사용자를 지정하도록 Greengrass nucleus를 구성하세요. 자세한 내용은 [구성 요소를 실행하는 사용자 구성](#) 및 [기본 구성 요소 사용자 구성](#) 섹션을 참조하세요.

Failed to map segment from shared object: operation not permitted

/tmp 폴더가 noexec 권한으로 마운트되어 코어 AWS IoT Greengrass 소프트웨어가 시작되지 않으면 이 오류가 표시될 수 있습니다. [AWS 공통 런타임\(CRT\) 라이브러리](#)는 기본적으로 /tmp 폴더를 사용합니다.

다음 중 하나를 수행하세요.

- 다음 명령을 실행하여 exec 권한이 있는 /tmp 폴더를 다시 탑재하고 다시 시도하세요.

```
sudo mount -o remount,exec /tmp
```


- Greengrass nucleus v2.5.0 이상을 실행하는 경우 JVM 옵션을 설정하여 AWS CRT 라이브러리가 사용하는 폴더를 변경할 수 있습니다. 배포의 Greengrass nucleus 구성 요소 구성에서 또는 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 `jvmOptions` 파라미터를 지정할 수 있습니다. `/path/to/use`를 AWS CRT 라이브러리가 사용할 수 있는 폴더의 경로로 바꿉니다.

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

Windows 서비스를 설정하지 못함

Microsoft Windows 2016 디바이스에 AWS IoT Greengrass 코어 소프트웨어를 설치하는 경우 오류가 발생할 수 있습니다. AWS IoT Greengrass 코어 소프트웨어는 Windows 2016에서 지원되지 않습니다. 지원되는 운영 체제 목록은 [섹션을 참조하세요](#) [지원하는 플랫폼](#).

Windows 2016을 사용하려면 먼저 다음과 같이 해야 합니다.

1. 다운로드한 AWS IoT Greengrass 코어 설치 아카이브의 압축을 풉니다.
2. Greengrass 디렉터리에서 `bin/greengrass.xml.template` 파일을 엽니다.
3. `<autoRefresh>` 태그 바로 앞 파일의 끝에 `</service>` 태그를 추가합니다.

```
</log>
<autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

루트 인증 기관(CA) 파일 없이 AWS IoT Greengrass 코어 소프트웨어를 설치할 때 오류가 발생할 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 관리자가 제공하는 구성 파일에 rootCaPath 파라미터를 사용하여 유효한 루트 CA 파일을 지정했는지 확인하세요. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 설치](#) 단원을 참조하십시오.

com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime

코어 디바이스가 배포 작업 알림을 구독 AWS IoT Core 하기 위해에 연결할 수 없는 경우이 경고 메시지가 표시될 수 있습니다. 해결 방법:

- 코어 디바이스가 인터넷에 연결되어 있고 구성한 AWS IoT 데이터 엔드포인트에 도달할 수 있는지 확인합니다. 코어 디바이스가 사용하는 엔드포인트에 대한 자세한 내용은 [프록시 또는 방화벽을 통해 디바이스 트래픽 허용](#) 섹션을 참조하세요.
- Greengrass 로그에서 기타 근본 원인을 알려주는 다른 오류가 있는지 확인하세요.

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

[자동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치하고](#) 설치 관리자가 유효하지 않은 AWS 세션 토큰을 사용하는 경우이 오류가 발생할 수 있습니다. 해결 방법:

- 임시 보안 인증을 사용하는 경우 세션 토큰이 올바르고 전체 세션 토큰을 복사하고 붙여넣고 있는지 확인하세요.
- 장기 보안 자격 증명을 사용하는 경우 이전에 임시 자격 증명을 사용한 시점의 세션 토큰이 디바이스에 없는지 확인하세요. 해결 방법:

1. 세션 토큰 환경 변수의 설정을 해제하려면 다음 명령을 실행하세요.

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 파일에 세션 토큰이 `~/.aws/credentials`에 포함되어 있는지 확인합니다. `aws_session_token`. 그렇다면 파일에서 해당 줄을 제거합니다.

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwmqKwRcOIfrRh3c/LTo6UDdyJw00vEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

자격 AWS 증명을 제공하지 않고도 AWS IoT Greengrass 코어 소프트웨어를 설치할 수도 있습니다. 자세한 내용은 [수동 리소스 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#) 또는 [AWS IoT 플릿 프로비저닝을 사용하여 AWS IoT Greengrass 코어 소프트웨어 설치](#)를 참조하세요.

`software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy`

[자동 프로비저닝으로 AWS IoT Greengrass 코어 소프트웨어를 설치하고](#) 설치 관리자가 필요한 권한이 없는 AWS 자격 증명을 사용하는 경우 오류가 발생할 수 있습니다. 필요한 권한에 대한 자세한 내용은 [설치 관리자가 리소스를 프로비저닝하기 위한 최소 IAM 정책](#) 섹션을 참조하세요.

자격 증명의 IAM ID에 대한 권한을 확인하고 필요한 권한이 누락된 경우 해당 IAM ID에 부여합니다.

Error:

`com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request`

[새도 관리자 구성 요소를](#) 사용하여 [디바이스 새도와 동기화하면](#)이 오류가 표시될 수 있습니다 [AWS IoT Core](#). HTTP 403 상태 코드는 코어 디바이스의 AWS IoT 정책을 호출할 수 있는 권한을 부여하지 않기 때문에이 오류가 발생했음을 나타냅니다 `GetThingShadow`.

```
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 새도 AWS IoT Core우를와 동기화하려면 코어 디바이스의 AWS IoT 정책에서 다음 권한을 부여해야 합니다.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

코어 디바이스의 AWS IoT 정책을 확인하고 누락된 필수 권한을 추가합니다. 자세한 내용은 다음을 참조하세요.

- AWS IoT 개발자 안내서의 [AWS IoT Core 정책 작업](#)
- [코어 디바이스의 AWS IoT 정책 업데이트](#)

Operation `aws.greengrass#<operation>` is not supported by Greengrass

사용자 지정 Greengrass 구성 요소에서 [프로세스 간 통신\(IPC\) 작업](#)을 사용하고 필수 AWS제공 구성 요소가 코어 디바이스에 설치되지 않은 경우 이 오류가 발생할 수 있습니다.

이 문제를 해결하려면 구성 요소 [레시피에 필수 구성 요소를 종속성](#)으로 추가하여 구성 요소를 배포할 때 AWS IoT Greengrass 코어 소프트웨어가 필수 구성 요소를 설치하도록 합니다.

- [암호 값 검색](#) – `aws.greengrass.SecretManager`
- [로컬 새도와 상호 작용](#) – `aws.greengrass.ShadowManager`
- [로컬 배포 및 구성 요소 관리](#) – `aws.greengrass.Cli v2.6.0` 이상
- [클라이언트 디바이스 인증 및 권한 부여](#) – `aws.greengrass.clientdevices.Auth v2.2.0` 이상

`java.io.FileNotFoundException: <stream-manager-store-root-dir>/
stream_manager_metadata_store (Permission denied)`

존재하지 않거나 올바른 권한이 없는 루트 폴더를 사용하도록 [스트림 관리자](#)를 구성하는 경우 스트림 관리자 로그 파일(`aws.greengrass.StreamManager.log`)에 이 오류가 발생할 수 있습니다. 이 폴더를 구성하는 방법에 대한 자세한 내용은 [스트림 관리자 구성](#)을 참조하세요.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

이 오류는 [PKCS#11 공급자 구성 요소가 하드웨어 보안 모듈\(HSM\)](#)을 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 때 지정한 프라이빗 키 또는 인증서를 찾거나 로드할 수 없을 때 발생합니다. 해결 방법:

- 사용할 AWS IoT Greengrass 코어 소프트웨어를 구성하는 슬롯, 사용자 PIN 및 객체 레이블을 사용하여 프라이빗 키와 인증서가 HSM에 저장되어 있는지 확인합니다.
- 프라이빗 키와 인증서가 HSM에서 동일한 객체 레이블을 사용하는지 확인하세요.
- HSM에서 객체 ID를 지원하는 경우 프라이빗 키와 인증서가 HSM에서 동일한 객체 ID를 사용하는지 확인하세요.

HSM의 보안 토큰에 대한 세부 정보를 쿼리하는 방법을 알아보려면 해당 HSM의 설명서를 확인하세요. 보안 토큰의 슬롯, 객체 레이블 또는 객체 ID를 변경해야 하는 경우 HSM의 설명서를 확인하여 변경 방법을 알아보세요.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

이 오류는 [보안 암호 관리자 구성 요소를](#) 사용하여 AWS Secrets Manager 보안 암호를 배포할 때 발생할 수 있습니다. 코어 디바이스의 [토큰 교환 IAM 역할](#)이 암호를 가져올 수 있는 권한을 부여하지 않으면 배포가 실패하고 Greengrass 로그에 이 오류가 포함됩니다.

코어 디바이스가 암호를 다운로드하도록 승인하려면

1. 코어 디바이스의 토큰 교환 역할에 `secretsmanager:GetSecretValue` 권한을 추가합니다. 다음 예제 정책 설명문에서는 암호의 값을 가져올 수 있는 권한을 부여합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "secretsmanager:GetSecretValue"
  ],
  "Resource": [
    "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-abcdef"
  ]
}
```

```
]
}
```

자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 단원을 참조하십시오.

2. 코어 디바이스에 배포를 다시 적용합니다. 다음 중 하나를 수행하세요.

- 변경 없이 배포를 수정합니다. 코어 디바이스는 수정된 배포를 수신하는 경우 암호를 다시 다운로드하려고 합니다. 자세한 내용은 [배포 수정](#) 단원을 참조하십시오.
- AWS IoT Greengrass 코어 소프트웨어를 다시 시작하여 배포를 다시 시도합니다. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 섹션을 참조하세요.

보안 암호 관리자가 보안 암호를 성공적으로 다운로드하면 배포가 성공합니다.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed

이 오류는 [보안 암호 관리자 구성 요소를](#) 사용하여 키로 암호화된 AWS Secrets Manager 보안 암호를 배포할 때 발생할 수 있습니다 AWS Key Management Service . 코어 디바이스의 [토큰 교환 IAM 역할](#)이 암호를 해독할 수 있는 권한을 부여하지 않으면 배포가 실패하고 Greengrass 로그에 이 오류가 포함됩니다.

이 문제를 해결하려면 코어 디바이스의 토큰 교환 역할에 kms:Decrypt 권한을 추가합니다. 자세한 내용은 다음을 참조하세요.

- AWS Secrets Manager 사용 설명서의 [암호 암호화 및 해독](#)
- [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#)

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

[하드웨어 보안](#)으로 AWS IoT Greengrass 코어 소프트웨어를 설치하려고 하고 하드웨어 보안 통합을 지원하지 않는 이전 Greengrass nucleus 버전을 사용하는 경우 이 오류가 발생할 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

AWS IoT Greengrass Core를 시스템 서비스로 실행할 때 TPM2 라이브러리를 사용하면 이 오류가 발생할 수 있습니다.

이 오류는 AWS IoT Greengrass 코어 시스템 서비스 파일에 PKCS#11 저장소의 위치를 제공하는 환경 변수를 추가해야 함을 나타냅니다.

자세한 내용은 [PKCS#11 공급자](#) 구성 요소 설명서의 요구 사항 섹션을 참조하세요.

Greengrass core device stuck on nucleus v2.12.3

Greengrass 코어 디바이스가 nucleus 버전 2.12.3에서 배포를 수정하지 않는 경우 다운로드하여 Greengrass.jar 파일을 Greengrass nucleus 버전 2.12.2로 바꾸어야 할 수 있습니다. 해결 방법:

1. Greengrass 코어 디바이스에서 다음 명령을 실행하여 Greengrass 코어 소프트웨어를 중지합니다.

Linux or Unix

```
sudo systemctl stop greengrass
```

Windows Command Prompt (CMD)

```
sc stop "greengrass"
```

PowerShell

```
Stop-Service -Name "greengrass"
```

2. 코어 디바이스에서 AWS IoT Greengrass 소프트웨어를 라는 파일에 다운로드합니다greengrass-2.12.2.zip.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip > greengrass-2.12.2.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip > greengrass-2.12.2.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -OutFile greengrass-2.12.2.zip
```

- 디바이스의 폴더에 AWS IoT Greengrass 코어 소프트웨어의 압축을 풉니다. *GreengrassInstaller*를 사용하려는 폴더로 바꿉니다.

Linux or Unix

```
unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -C GreengrassInstaller && del greengrass-2.12.2.zip
```

PowerShell

```
Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\  
\GreengrassInstaller  
rm greengrass-2.12.2.zip
```

- 다음 명령을 실행하여 nucleus 버전 2.12.3 Greengrass JAR 파일을 nucleus 버전 2.12.2 Greengrass JAR 파일로 재정의합니다.

Linux or Unix

```
sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/  
artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
```


Windows Command Prompt (CMD)

```
robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
```

PowerShell

```
cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
```

5. 다음 명령을 실행하여 Greengrass 코어 소프트웨어를 시작합니다.

Linux or Unix

```
sudo systemctl start greengrass
```

Windows Command Prompt (CMD)

```
sc start "greengrass"
```

PowerShell

```
Start-Service -Name "greengrass"
```

Greengrass nucleus v2.14.0 systemd template issue

이 문제는 Linux 디바이스에 Greengrass nucleus v2.14.0을 기본 systemd 서비스 템플릿으로 설치한 경우 발생할 수 있습니다. 해결 방법:

1. Greengrass 코어 디바이스에서 다음 명령을 실행하여 systemd 서비스 파일을 nucleus v2.13.0 이전 버전과 동일한 상태로 되돌립니다.

Linux or Unix

```
sudo sed -i 's|ExecStart=/bin/sh -c "\(.*\)" >> ./logs/loader.log 2>&1|ExecStart=/bin/sh \1|' /etc/systemd/system/greengrass.service
```

2. 변경 사항을 적용합니다.

Linux or Unix

```
sudo systemctl daemon-reload
sudo systemctl restart greengrass
```

AWS IoT Greengrass 클라우드 문제

다음 정보를 사용하여 AWS IoT Greengrass 콘솔 및 API 관련 문제를 해결합니다. 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 [CreateComponentVersion](#) 작업을 사용하여 구성 요소 버전을 생성할 때 오류가 발생할 수 있습니다.

이 오류는 레시피가 유효한 JSON 또는 YAML이 아님을 나타냅니다. 레시피의 구문을 확인하고 구문 문제를 해결한 다음 다시 시도하세요. 온라인 JSON 또는 YAML 구문 검사기를 사용하여 레시피에서 구문 문제를 식별할 수 있습니다.

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

AWS IoT Greengrass 콘솔에서 또는 [CreateComponentVersion](#) 작업을 사용하여 구성 요소 버전을 생성할 때 오류가 발생할 수 있습니다. 이 오류는 구성 요소 레시피의 S3 아티팩트가 유효하지 않음을 나타냅니다.

해결 방법:

- S3 버킷이 구성 요소를 생성하는 AWS 리전 곳과 동일한에 있는지 확인합니다. AWS IoT Greengrass 는 구성 요소 아티팩트에 대한 리전 간 요청을 지원하지 않습니다.
- 아티팩트 URI가 유효한 S3 객체 URL인지 확인하고 해당 S3 객체 URL에 아티팩트가 있는지 확인하세요.

- AWS 계정 에 S3 객체 URL에서 아티팩트에 액세스할 수 있는 권한이 있는지 확인합니다.

INACTIVE deployment status

필요한 종속 AWS IoT 정책 없이 [ListDeployments](#) API를 호출하면 INACTIVE 배포 상태가 될 수 있습니다. 정확한 배포 상태를 가져오려면 필요한 권한이 있어야 합니다. [AWS IoT Greengrass V2에서 정의한 작업](#)을 살펴보고 ListDeployments에 필요한 권한을 따르면 종속 작업을 찾을 수 있습니다. 필요한 종속 AWS IoT 권한이 없으면 여전히 배포 상태가 표시되지만 부정확한 배포 상태가 표시될 수 있습니다. INACTIVE.

코어 디바이스 배포 문제

Greengrass 코어 디바이스의 배포 문제를 해결하세요. 각 항목은 코어 디바이스에 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.PackageDownloadException`
Failed to download artifact

AWS IoT Greengrass 코어 디바이스가 배포를 적용할 때 코어 소프트웨어가 구성 요소 아티팩트를 다운로드하지 못하면이 오류가 발생할 수 있습니다. 이 오류로 인해 배포가 실패합니다.

이러한 오류가 발생하는 경우 로그에 특정 문제를 식별하는 데 사용할 수 있는 스택 추적도 포함됩니다. 다음 각 항목은 Failed to download artifact 오류 메시지의 스택 추적이 표시될 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, ...)`

다음 경우에는 [PackageDownloadException 오류](#)에 이 스택 추적이 포함될 수 있습니다.

- 구성 요소의 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 사용할 수 없습니다. 아티팩트를 S3 버킷에 업로드했고 아티팩트 URI가 버킷에 있는 아티팩트의 S3 객체 URL과 일치하는지 확인하세요.

- 코어 디바이스의 [토큰 교환 역할](#)은 AWS IoT Greengrass 코어 소프트웨어가 구성 요소의 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 다운로드하도록 허용하지 않습니다. 토큰 교환 역할이 아티팩트를 사용할 수 있는 S3 객체 URL에 대해 `s3:GetObject`를 허용하는지 확인하세요.

`software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>`

코어 디바이스에 `s3:GetBucketLocation`을 직접적으로 호출할 권한이 없는 경우

[PackageDownloadException 오류](#)에 이 스택 추적이 포함될 수 있습니다. 오류 메시지에는 다음 메시지도 포함되어 있습니다.

```
reason: Failed to determine S3 bucket location
```

코어 디바이스의 [토큰 교환 역할](#)이 아티팩트를 사용할 수 있는 S3 버킷에 대해 `s3:GetBucketLocation`을 허용하는지 확인하세요.

Error:

`com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.`

AWS IoT Greengrass 코어 디바이스가 배포를 적용할 때 코어 소프트웨어가 구성 요소 아티팩트를 다운로드하지 못하면 오류가 발생할 수 있습니다. 다운로드한 아티팩트 파일의 체크섬이 구성 요소를 생성할 때 AWS IoT Greengrass 계산된 체크섬과 일치하지 않기 때문에 배포가 실패합니다.

해결 방법:

- 아티팩트 파일을 호스팅하는 S3 버킷에서 아티팩트 파일이 변경되었는지 확인하세요. 구성 요소를 생성한 후 파일이 변경된 경우 코어 디바이스에서 예상하는 이전 버전으로 복원하세요. 파일을 이전 버전으로 복원할 수 없거나 파일의 새 버전을 사용하려면 아티팩트 파일로 구성 요소의 새 버전을 생성합니다.
- 코어 디바이스의 인터넷 연결을 확인하세요. 아티팩트 파일이 다운로드하는 동안 손상된 경우 이 오류가 발생할 수 있습니다. 새 배포를 생성하고 다시 시도하세요.

Error:

`com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException`

Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

코어 디바이스에서 해당 코어 디바이스에 대한 배포 요구 사항을 충족하는 구성 요소 버전을 찾을 수 없는 경우 이 오류가 발생할 수 있습니다. 코어 디바이스는 AWS IoT Greengrass 서비스 및 로컬 디바이스에서 구성 요소를 확인합니다. 오류 메시지는 각 배포의 대상과 해당 배포의 구성 요소에 대한 버전 요구 사항을 포함합니다. 배포 대상은 코어 디바이스의 로컬 배포를 나타내는 사물, 사물 그룹 또는 `LOCAL_DEPLOYMENT`일 수 있습니다.

이 문제는 다음 경우에 발생할 수 있습니다.

- 코어 디바이스는 구성 요소 버전 요구 사항이 충돌하는 여러 배포의 대상입니다. 예를 들어 코어 디바이스는 `com.example.HelloWorld` 구성 요소가 포함된 여러 배포의 대상으로 한 배포에는 버전 1.0.0이 필요하고 다른 배포에는 버전 1.0.1이 필요할 수 있습니다. 두 가지 요구 사항을 모두 충족하는 구성 요소를 보유하는 것은 불가능하므로 배포가 실패합니다.
- 구성 요소 버전이 AWS IoT Greengrass 서비스 또는 로컬 디바이스에 존재하지 않습니다. 예를 들어 구성 요소가 삭제되었을 수 있습니다.
- 버전 요구 사항을 충족하는 구성 요소 버전이 있지만 어느 것도 코어 디바이스의 플랫폼과 호환되지 않습니다.
- 코어 디바이스의 AWS IoT 정책은 `greengrass:ResolveComponentCandidates` 권한을 부여하지 않습니다. 오류 로그에서 `Status Code: 403`을 찾아 이 문제를 식별합니다. 이 문제를 해결하려면 코어 디바이스의 AWS IoT 정책에 `greengrass:ResolveComponentCandidates` 권한을 추가합니다. 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 단원을 참조하십시오.

이 문제를 해결하려면 호환되는 구성 요소 버전을 포함하도록 배포를 수정하거나 호환되지 않는 구성 요소 버전을 제거합니다. 클라우드 배포를 수정하는 방법에 대한 자세한 내용은 [배포 수정](#) 섹션을 참조하세요. 로컬 배포를 수정하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass CLI 배포 생성](#) 명령을 참조하세요.

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

코어 디바이스에 구성 요소를 배포할 때 구성 요소가 코어 디바이스의 플랫폼과 호환되는 플랫폼을 나열하지 않는 경우 이 오류가 발생할 수 있습니다. 다음 중 하나를 수행하세요.

- 구성 요소가 사용자 지정 Greengrass 구성 요소인 경우 코어 디바이스와 호환되도록 구성 요소를 업데이트할 수 있습니다. 코어 디바이스의 플랫폼과 일치하는 새로운 매니페스트를 추가하거나 코어 디바이스의 플랫폼과 일치하도록 기존 매니페스트를 업데이트하세요. 자세한 내용은 [AWS IoT Greengrass 구성 요소 레시피 참조](#) 단원을 참조하십시오.
- 에서 구성 요소를 제공하는 경우 구성 요소의 다른 버전이 코어 디바이스와 호환되는지 AWS 확인합니다. 호환되는 버전이 없는 경우 [AWS IoT Greengrass 태그](#)를 사용하여 [AWS re:Post](#)에 문의하거나 [지원](#)에 문의하세요.

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 nucleus](#)에 따라 달라지는 구성 요소를 배포하고 코어 디바이스가 사용 가능한 최신 마이너 버전보다 이전의 Greengrass nucleus 버전을 실행하는 경우 이 오류가 발생할 수 있습니다. 이 오류는 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 호환되는 최신 버전으로 자동 업데이트하려고 하기 때문에 발생합니다. 그러나 AWS IoT Greengrass 코어 소프트웨어는 Greengrass nucleus의 특정 마이너 버전에 따라 여러 AWS제공 구성 요소가 달라지기 때문에 Greengrass nucleus가 새 마이너 버전으로 업데이트되는 것을 방지합니다. 자세한 내용은 [Greengrass nucleus 업데이트 동작](#) 단원을 참조하십시오.

사용하려는 Greengrass nucleus 버전을 지정하려면 [배포를 수정](#)해야 합니다. 다음 중 하나를 수행하세요.

- 코어 디바이스가 현재 실행하는 Greengrass nucleus 버전을 지정하도록 배포를 수정하세요.
- Greengrass nucleus의 이후 마이너 버전을 지정하도록 배포를 수정하세요. 이 옵션을 선택하는 경우 Greengrass nucleus의 특정 마이너 버전에 따라 AWS제공되는 구성 요소의 버전도 업데이트해야 합니다. 자세한 내용은 [AWS 제공 구성 요소](#) 단원을 참조하십시오.

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 그룹에서 다른 그룹으로 이동한 다음 Greengrass를 다시 시작해야 하는 배포를 사용하여 원래 그룹으로 다시 돌아오는 경우 이 오류가 발생할 수 있습니다.

이 문제를 해결하려면 디바이스의 시작 디렉터리를 다시 생성합니다. 또한 Greengrass nucleus 버전 2.9.6 이상으로 업그레이드하는 것이 좋습니다.

다음은 시작 디렉터리를 다시 생성하기 위한 Linux 스크립트입니다. `fix_directory.sh`라는 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
```


이 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.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration

코어 디바이스가 7KB(사물을 대상으로 하는 배포의 경우) 또는 31KB(사물 그룹을 대상으로 하는 배포의 경우)보다 큰 용량의 배포 문서인 큰 배포 문서를 수신하는 경우 이 오류가 발생할 수 있습니다. 대규모 배포 문서를 검색하려면 코어 디바이스의 AWS IoT 정책에서 greengrass:GetDeploymentConfiguration 권한을 허용해야 합니다. 코어 디바이스에 해당 권한이 없는 경우 이 오류가 발생할 수 있습니다. 이 오류가 발생하면 배포는 무기한으로 다시 시도되며 그 상태는 진행 중(IN_PROGRESS)입니다.

이 문제를 해결하려면 코어 디바이스의 AWS IoT 정책에 greengrass:GetDeploymentConfiguration 권한을 추가합니다. 자세한 내용은 [코어 디바이스의 AWS IoT 정책 업데이트](#) 단원을 참조하십시오.

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

코어 디바이스가 배포를 수신하고 코어 디바이스의 AWS IoT 정책에서 greengrass:ListThingGroupsForCoreDevice 권한을 허용하지 않는 경우 이 경고가 표시될 수 있습니다. 배포를 생성하는 경우 코어 디바이스는 이 권한을 사용하여 사물 그룹을 식별하고 코어 디바이스를 제거한 모든 사물 그룹의 구성 요소를 제거합니다. 코어 디바이스가 [Greengrass nucleus](#) v2.5.0을 실행하는 경우 배포가 실패합니다. 코어 디바이스가 Greengrass nucleus v2.5.1 이상을 실행하는 경우에는 배포가 진행되지만 구성 요소는 제거되지 않습니다. 사물 그룹 제거 동작에 대한 자세한 내용은 [디바이스에 AWS IoT Greengrass 구성 요소 배포](#) 섹션을 참조하세요.

코어 디바이스를 제거하는 사물 그룹의 구성 요소를 제거하도록 코어 디바이스의 동작을 업데이트하려면 코어 디바이스의 AWS IoT 정책에 greengrass:ListThingGroupsForCoreDevice 권한을 추가합니다. 자세한 내용은 [코어 디바이스의 AWS IoT 정책 업데이트](#) 단원을 참조하십시오.

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

코어 디바이스가 DEBUG 로그 수준에서 오류를 기록하므로 이 정보 메시지가 오류 없이 여러 번 인쇄될 수 있습니다. 이 문제는 코어 디바이스가 큰 배포 문서를 수신하는 경우 발생할 수 있습니다. 이 문제가 발생하면 배포는 무기한으로 다시 시도되며 그 상태는 진행 중(IN_PROGRESS)입니다. 이 문제를 해결하는 방법에 대한 자세한 내용은 [이 문제 해결 항목](#)을 참조하세요.

Caused by:

software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

데이터 영역 API에 `iot:Connect` 권한이 없는 경우 이 오류가 발생할 수 있습니다. 올바른 정책이 없는 경우 `GreengrassV2DataException: 403`을 수신하게 됩니다. 권한 정책을 생성하려면 [AWS IoT 정책 생성](#) 지침을 따르세요.

코어 디바이스 구성 요소 문제

코어 디바이스에서 Greengrass 구성 요소 문제를 해결하세요.

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 않음](#)
- [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 코어 소프트웨어가 구성 요소의 수명 주기 스크립트에서 명령을 실행하지 못하면 Greengrass 구성 요소의 로그에 이 오류가 표시될 수 있습니다. 이 오류로 인해 구성 요소의 상태가 BROKEN이 됩니다. ggc_user와 같은 구성 요소를 실행하는 시스템 사용자가 [경로](#)의 폴더에서 명령의 실행 파일을 찾을 수 없는 경우 이러한 오류가 발생할 수 있습니다.

Windows 디바이스의 경우 실행 파일이 포함된 폴더가 구성 요소를 실행하는 시스템 사용자의 PATH에 있는지 확인하세요. PATH에서 누락된 경우 다음 중 하나를 수행합니다.

- 실행 파일의 폴더를 모든 사용자가 사용할 수 있는 PATH 시스템 변수에 추가합니다. 그런 다음 구성 요소를 다시 시작합니다.

Greengrass nucleus 2.5.0을 실행하는 경우 PATH 시스템 변수를 업데이트한 후 AWS IoT Greengrass 코어 소프트웨어를 다시 시작하여 업데이트된 로 구성 요소를 실행해야 합니다. PATH. 소프트웨어를 다시 시작한 PATH 후 AWS IoT Greengrass 코어 소프트웨어가 업데이트된를 사용하지 않는 경우 디바이스를 다시 시작하고 다시 시도하세요. 자세한 내용은 [AWS IoT Greengrass 코어 소프트웨어 실행](#) 단원을 참조하십시오.

- 구성 요소를 실행하는 시스템 사용자의 PATH 사용자 변수에 실행 파일의 폴더를 추가합니다.

Python 스크립트는 메시지를 기록하지 않습니다.

Greengrass 코어 디바이스는 구성 요소의 문제를 식별하는 데 사용할 수 있는 로그를 수집합니다. Python 스크립트의 stdout 및 stderr 메시지가 구성 요소 로그에 표시되지 않는 경우 Python에서 버퍼를 플러시하거나 이러한 표준 출력 스트림에 대한 버퍼링을 비활성화해야 할 수 있습니다. 해결 방법:

- [-u](#) 인수로 Python을 실행하여 stdout 및 stderr에서 버퍼링을 비활성화합니다.

Linux or Unix

```
python3 -u hello_world.py
```

Windows

```
py -3 -u hello_world.py
```

- 구성 요소의 레시피에서 [Setenv](#)를 사용하여 [PYTHONUNBUFFERED](#) 환경 변수를 비어 있지 않은 문자열로 설정합니다. 이 환경 변수는 stdout 및 stderr에서 버퍼링을 비활성화합니다.
- stdout 또는 stderr 스트림의 버퍼를 플러시합니다. 다음 중 하나를 수행하세요.
 - 인쇄할 때 메시지를 플러시합니다.

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

- 인쇄 후에 메시지를 플러시합니다. 스트림을 플러시하기 전에 여러 메시지를 전송할 수 있습니다.

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

Python 스크립트가 로그 메시지를 출력하는지 확인하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass 로그 모니터링](#) 섹션을 참조하세요.

기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않음

구성 요소의 레시피에서 DefaultConfiguration을 변경하면 배포 중에 새 기본 구성이 구성 요소의 기존 구성을 대체하지 않습니다. 새 기본 구성을 적용하려면 구성 요소의 구성을 기본 설정으로 재설정해야 합니다. 구성 요소를 배포할 때 단일 빈 문자열을 [재설정 업데이트](#)로 지정합니다.

Console

재설정 경로

```
[\""]
```

AWS CLI

다음 명령은 코어 디바이스에 배포를 생성합니다.

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-deployment.json
```

reset-configuration-deployment.json 파일에는 다음 JSON 문서가 포함되어 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
```

reset-configuration-deployment.json 파일에는 다음 JSON 문서가 포함되어 있습니다.

```
{
```

```

    "com.example.HelloWorld": {
      "RESET": [""]
    }
  }
}

```

awsiot.greengrasscoreipc.model.UnauthorizedError

구성 요소에 리소스에서 IPC 작업을 수행할 권한이 없는 경우 Greengrass 구성 요소의 로그에 이 오류가 발생할 수 있습니다. 구성 요소에 IPC 작업을 직접 호출할 수 있는 권한을 부여하려면 구성 요소의 구성에서 IPC 권한 부여 정책을 정의합니다. 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 단원을 참조하십시오.

Tip

구성 요소의 레시피에서 DefaultConfiguration을 변경하는 경우 구성 요소의 구성을 새로운 기본 구성으로 재설정해야 합니다. 구성 요소를 배포할 때 단일 빈 문자열을 [재설정 업데이트](#)로 지정합니다. 자세한 내용은 [기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않음](#) 단원을 참조하십시오.

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

코어 디바이스의 모든 구성 요소를 포함하여 여러 IPC 권한 부여 정책에서 동일한 정책 ID를 사용하는 경우 이 오류가 발생할 수 있습니다.

구성 요소의 IPC 권한 부여 정책을 확인하고 중복 항목을 수정한 다음 다시 시도하세요. 고유한 정책 ID를 생성하려면 구성 요소 이름, IPC 서비스 이름, 카운터를 결합하는 것이 좋습니다. 자세한 내용은 [구성 요소에 IPC 작업을 수행할 수 있는 권한 부여](#) 단원을 참조하십시오.

Tip

구성 요소의 레시피에서 DefaultConfiguration을 변경하는 경우 구성 요소의 구성을 새로운 기본 구성으로 재설정해야 합니다. 구성 요소를 배포할 때 단일 빈 문자열을 [재설정 업데이트](#)로 지정합니다. 자세한 내용은 [기본 구성을 변경할 때 구성 요소 구성이 업데이트되지 않음](#) 단원을 참조하십시오.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

코어 디바이스가 [토큰 교환 서비스](#)에서 AWS 자격 증명을 가져올 수 없는 경우 오류가 표시될 수 있습니다. HTTP 400 상태 코드는 코어 디바이스의 [토큰 교환 IAM 역할](#)이 존재하지 않거나 자격 AWS IoT 증명 공급자가 이를 수입할 수 있는 신뢰 관계가 없기 때문에 오류가 발생했음을 나타냅니다.

해결 방법:

1. 코어 디바이스가 사용하는 토큰 교환 역할을 식별합니다. 오류 메시지에는 토큰 교환 AWS IoT 역할을 가리키는 코어 디바이스의 역할 별칭이 포함됩니다. 개발 컴퓨터에서 다음 명령을 실행하고 *MyGreengrassCoreTokenExchangeRoleAlias*를 오류 메시지의 AWS IoT 역할 별칭의 이름으로 바꿉니다.

```
aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
```

응답에는 토큰 교환 IAM 역할의 Amazon 리소스 이름(ARN)이 포함되어 있습니다.

```
{
  "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 있는지 확인하세요. 다음 명령을 실행하고 *MyGreengrassV2TokenExchangeRole*을 토큰 교환 역할의 이름으로 바꿉니다.

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

명령이 NoSuchEntity 오류를 반환하면 역할이 존재하지 않으므로 이를 생성해야 합니다. 역할을 생성하고 구성하는 방법에 대한 자세한 내용은 [코어 디바이스가 AWS 서비스와 상호 작용할 수 있도록 권한 부여](#) 섹션을 참조하세요.

3. 역할에 자격 AWS IoT 증명 공급자가 역할을 수임할 수 있는 신뢰 관계가 있는지 확인합니다. 이전 단계의 응답에는 역할의 신뢰 관계를 정의하는 AssumeRolePolicyDocument가 포함되어 있습니다. 역할은 `credentials.iot.amazonaws.com`이 수임할 수 있도록 허용하는 신뢰 관계를 정의해야 합니다. 이 문서는 다음 예제와 유사할 것입니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

역할의 신뢰 관계에서 `credentials.iot.amazonaws.com`이 이를 수임하도록 허용하지 않는 경우 이 신뢰 관계를 역할에 추가해야 합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [역할 수정](#)을 참조하세요.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

코어 디바이스가 [토큰 교환 서비스](#)에서 AWS 자격 증명을 가져올 수 없는 경우 오류가 표시될 수 있습니다. HTTP 403 상태 코드는 코어 디바이스의 AWS IoT 정책이 코어 디바이스의 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 별칭에 대한 `iot:AssumeRoleWithCertificate` 권한을 추가합니다. 오류 메시지에는 코어 디바이스의 현재 AWS IoT 역할 별칭이 포함됩니다. 이 권한과 코어 디바이스의 AWS IoT 정책을 업데이트하는 방법에 대한 자세한 내용은 [AWS IoT Greengrass V2 코어 디바이스에 대한 최소 AWS IoT 정책](#) 및 섹션을 참조하세요 [코어 디바이스의 AWS IoT 정책 업데이트](#).

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

구성 요소가 AWS 자격 증명을 요청하려고 하고 [토큰 교환 서비스에](#) 연결할 수 없는 경우 이 오류가 표시될 수 있습니다.

해결 방법:

- 구성 요소가 토큰 교환 서비스 구성 요소인 `aws.greengrass.TokenExchangeService`에 대한 종속성을 선언하는지 확인하세요. 그렇지 않은 경우 종속성을 추가하고 구성 요소를 다시 배포합니다.
- 구성 요소가 Docker에서 실행되는 경우 [Docker 컨테이너 구성 요소에서 AWS 자격 증명 사용 \(Linux\)](#)에 따라 올바른 네트워크 설정과 환경 변수를 적용해야 합니다.
- 구성 요소가 NodeJS에서 작성된 경우 [dns.setDefaultResultOrder](#)를 **ipv4first**로 설정합니다.
- :::1로 시작하고 localhost를 포함하는 항목이 있는지 `/etc/hosts`를 검사합니다. 해당 항목을 제거하여 구성 요소가 잘못된 주소의 토큰 교환 서비스에 연결되어 있는지 확인합니다.

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

구성 요소가 [토큰 교환 서비스](#)를 실행하지 않고 구성 요소가 AWS 자격 증명을 요청하려고 하면 이 오류가 표시될 수 있습니다.

해결 방법:

- 구성 요소가 토큰 교환 서비스 구성 요소인 `aws.greengrass.TokenExchangeService`에 대한 종속성을 선언하는지 확인하세요. 그렇지 않은 경우 종속성을 추가하고 구성 요소를 다시 배포합니다.
- 구성 요소가 `install` 수명 주기에서 AWS 자격 증명을 사용하는지 확인합니다. AWS IoT Greengrass 는 `install` 수명 주기 동안 토큰 교환 서비스의 가용성을 보장하지 않습니다. AWS 자격 증명을 사용하는 코드를 `startup` 또는 `run` 수명 주기로 이동하도록 구성 요소를 업데이트한 다음 해당 구성 요소를 재배포합니다.

copyFrom: <configurationPath> is already a container, not a leaf

구성 값을 컨테이너 유형(목록 또는 객체)에서 비컨테이너 유형(문자열, 숫자 또는 부울)으로 변경하는 경우 이 오류가 발생할 수 있습니다. 해결 방법:

1. 구성 요소의 레시피를 확인하여 기본 구성이 해당 구성 값을 목록 또는 객체로 설정하는지 확인하세요. 해당하는 경우 구성 값을 제거하거나 변경합니다.
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.DockerLoginError logging into the registry using credentials - 'The stub received bad data.'

[Docker 애플리케이션 관리자 구성 요소](#)가 Amazon Elastic Container Registry(Amazon ECR)의 프라이빗 리포지토리에서 Docker 이미지를 다운로드하려고 하는 경우 Greengrass nucleus 로그에서 이 오류가 발생할 수 있습니다. 이 오류는 wincred [Docker 자격 증명 도우미](#)(docker-credential-wincred)를 사용하는 경우 발생합니다. 따라서 Amazon ECR은 로그인 자격 증명을 저장할 수 없습니다.

다음 작업 중 하나를 수행합니다.

- wincred Docker 자격 증명 도우미를 사용하지 않는 경우 코어 디바이스에서 docker-credential-wincred 프로그램을 제거합니다.
- wincred Docker 자격 증명 도우미를 사용하는 경우 다음을 수행합니다.
 1. 코어 디바이스에서 docker-credential-wincred 프로그램의 이름을 바꿉니다. wincred를 Windows Docker 자격 증명 도우미의 새로운 이름으로 바꿉니다. 예를 들어 이름을 docker-credential-wincredreal로 바꿀 수 있습니다.
 2. Windows Docker 자격 증명 도우미의 새로운 이름을 사용하도록 Docker 구성 파일(.docker/config.json)의 credsStore 옵션을 업데이트합니다. 예를 들어 프로그램 이름을 docker-credential-wincredreal로 바꾼 경우 credsStore 옵션을 wincredreal로 업데이트합니다.

```
{
```

```
"credsStore": "wincredreal"
}
```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

ggc_user와 같은 구성 요소의 프로세스를 실행하는 시스템 사용자의 암호가 만료된 경우 Windows 코어 디바이스에서 이 오류가 발생할 수 있습니다. 따라서 AWS IoT Greengrass 코어 소프트웨어가 해당 시스템 사용자로 구성 요소 프로세스를 실행할 수 없습니다.

Greengrass 시스템 사용자의 암호를 업데이트하려면

1. 관리자로 다음 명령을 실행하여 사용자의 암호를 설정합니다. *ggc_user*를 시스템 사용자로 바꾸고 *##*를 설정할 암호로 바꿉니다.

```
net user ggc_user password
```

2. [PsExec 유틸리티](#)를 사용하여 사용자의 새로운 암호를 LocalSystem 계정의 자격 증명 관리자 인스턴스에 저장합니다. *##*를 설정한 사용자의 암호로 바꿉니다.

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

Tip

Windows 구성에 따라 사용자의 암호가 미래의 날짜에 만료되도록 설정할 수 있습니다. Greengrass 애플리케이션이 계속 작동하도록 하려면 암호가 만료되는 시기를 추적하고 만료되기 전에 이를 업데이트합니다. 사용자의 암호가 만료되지 않도록 설정할 수도 있습니다.

- 사용자와 암호가 만료되는 시기를 확인하려면 다음 명령을 실행합니다.

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

- 사용자의 암호가 만료되지 않도록 설정하려면 다음 명령을 실행합니다.

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

- [wmic 명령이 더 이상 사용되지 않는](#) Windows 10 이상을 사용하는 경우 다음 PowerShell 명령을 실행합니다.

```
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 않는 경우 스트림 관리자 구성 요소의 로그에 다음 오류가 발생할 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["lastExportTime"]
{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 배포하고 이후 버전으로 업그레이드하려는 경우 스트림 관리자 v2.0.12로 직접 업그레이드해야 합니다. 스트림 관리자 구성 요소에 대한 자세한 내용은 [스트림 관리자](#) 섹션을 참조하세요.

코어 디바이스 Lambda 함수 구성 요소 문제

코어 디바이스의 Lambda 함수 구성 요소 문제를 해결하세요.

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

다음 사례에서 컨테이너화된 Lambda 함수를 실행하는 경우 이 오류가 발생할 수 있습니다.

- 코어 디바이스에 메모리 또는 디바이스 cgroup에 대한 cgroup v1이 활성화되어 있지 않습니다.
- 코어 디바이스에 cgroup v2가 활성화되어 있습니다. Greengrass Lambda 함수에는 cgroup v1이 필요하며 cgroup v1과 v2는 상호 배타적입니다.

cgroup v1을 활성화하려면 디바이스를 다음 Linux 커널 파라미터로 부팅합니다.

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

Tip

Raspberry Pi에서 /boot/cmdline.txt 파일을 편집하여 디바이스의 커널 파라미터를 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>

[레거시 구독 라우터 구성](#) 요소에 구독을 지정하지 않고 V2 코어 디바이스에서 AWS IoT Greengrass 코어 SDK를 사용하는 V1 Lambda 함수를 실행할 때 오류가 발생할 수 있습니다. V2 이 문제를 해결하려면 레거시 구독 라우터를 배포하고 구성하여 필요한 구독을 지정하세요. 자세한 내용은 [V1 Lambda 함수 가져오기](#) 단원을 참조하십시오.

구성 요소 버전 중단

코어 디바이스의 구성 요소 버전이 중단되면 Personal Health Dashboard(PHD)에 알림이 표시될 수 있습니다. 구성 요소 버전은 중단 후 60분 이내에 이 알림을 PHD로 전송합니다.

수정해야 하는 배포를 확인하려면 AWS Command Line Interface를 사용하여 다음을 수행합니다.

1. 다음 명령을 실행하여 코어 디바이스 목록을 가져옵니다.

```
aws greengrassv2 list-core-devices
```

2. 다음 명령을 실행하여 1단계에서 각 코어 디바이스의 구성 요소 상태를 검색합니다.
*coreDeviceName*을 쿼리할 각 코어 디바이스의 이름으로 바꿉니다.

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

3. 이전 단계에서 중단된 구성 요소 버전이 설치된 코어 디바이스를 수집합니다.
4. 다음 명령을 실행하여 3단계에서 각 코어 디바이스에 대한 모든 배포 작업의 상태를 검색합니다. `coreDeviceName`을 쿼리할 코어 디바이스의 이름으로 바꿉니다.

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

응답에 코어 디바이스에 대한 배포 작업의 목록이 포함되어 있습니다. 배포를 수정하여 다른 구성 요소 버전을 선택할 수 있습니다. 배포를 수정하는 방법에 대한 자세한 내용은 [Revise deployments](#)를 참조하세요.

Greengrass 명령줄 인터페이스 문제

[Greengrass CLI](#) 관련 문제를 해결하세요.

주제

- [java.lang.RuntimeException: Unable to create ipc client](#)

java.lang.RuntimeException: Unable to create ipc client

Greengrass CLI 명령을 실행하고 AWS IoT Greengrass 코어 소프트웨어가 설치된 위치와 다른 루트 폴더를 지정하면 오류가 표시될 수 있습니다.

다음 중 하나를 수행하여 루트 경로를 설정하고 AWS IoT Greengrass 코어 소프트웨어 설치 경로 `/greengrass/v2`로 바꿉니다.

- GGC_ROOT_PATH 환경 변수를 `/greengrass/v2`로 설정합니다.
- 다음 예제에 표시된 대로 명령에 `--ggcRootPath /greengrass/v2` 인수를 추가합니다.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface 문제

에 대한 AWS CLI 문제를 해결합니다 AWS IoT Greengrass V2.

주제

- [Error: Invalid choice: 'greengrassv2'](#)

Error: Invalid choice: 'greengrassv2'

AWS CLI (예:)를 사용하여 AWS IoT Greengrass V2 명령을 실행하면이 오류가 표시될 수 있습니다
다aws greengrassv2 list-core-devices.

이 오류는가 지원하지 AWS CLI 않는 버전이 있음을 나타냅니다 AWS IoT Greengrass V2. 와 AWS IoT Greengrass V2 함께를 사용하려면 다음 버전 중 하나 이상이 있어야 AWS CLI합니다.

- Minimum AWS CLI V1 버전: v1.18.197
- Minimum AWS CLI V2 버전: v2.1.11

Tip

다음 명령을 실행하여 보유한의 버전을 확인할 수 AWS CLI 있습니다.

```
aws --version
```

이 문제를 해결하려면 AWS CLI 를에서 지원하는 이후 버전으로 업데이트합니다 AWS IoT Greengrass V2. 자세한 내용은 AWS Command Line Interface 사용 설명서의 [AWS CLI의 설치, 업데이트, 제거](#)를 참조하세요.

세부 배포 오류 코드

이 섹션의 오류 코드와 해결 방법을 사용하여 Greengrass nucleus 버전 2.8.0 이상을 사용하는 경우 구성 요소 배포 관련 문제를 해결합니다.

Greengrass nucleus는 가장 구체적이지 않은 코드부터 가장 구체적인 코드까지 계층 구조로 배포 오류를 보고합니다. 이 계층 구조를 사용하면 배포 오류의 원인을 정확히 파악할 수 있습니다. 예를 들어 다음 오류 계층 구조가 있을 수 있습니다.

- DEPLOYMENT_FAILURE
 - ARTIFACT_DOWNLOAD_ERROR
 - IO_ERROR
 - DISK_SPACE_CRITICAL

오류 코드는 유형별로 정리되어 있습니다. 각 유형은 발생할 수 있는 오류 클래스를 나타냅니다. 콘솔, API 및에서 이러한 오류 유형을 AWS IoT Greengrass 보고합니다 AWS CLI. 오류 계층 구조에 보고된 오류에 따라 두 가지 이상의 오류 유형이 있을 수 있습니다. 이전 예제에서 반환된 오류 유형은 `DEVICE_ERROR`입니다.

다음과 같은 유형이 있습니다.

- `PERMISSION_ERROR` – 권한이 필요한 작업에 대한 액세스가 거부되었습니다.
- `REQUEST_ERROR` – 배포 문서의 문제로 인해 오류가 발생했습니다.
- `COMPONENT_RECIPE_ERROR` – 구성 요소 레시피의 문제로 인해 오류가 발생했습니다.
- `AWS_COMPONENT_ERROR` - AWS 제공된 구성 요소를 시작하거나 제거할 때 오류가 발생했습니다.
- `USER_COMPONENT_ERROR` – 사용자 구성 요소를 시작하거나 제거할 때 오류가 발생했습니다.
- `COMPONENT_ERROR` - 구성 요소를 시작하거나 제거할 때 오류가 발생했지만 Greengrass nucleus에서 해당 구성 요소가 AWS 에서 제공한 구성 요소인지 사용자 구성 요소인지 확인할 수 없습니다.
- `DEVICE_ERROR` – 로컬 I/O에 오류가 발생하거나 다른 디바이스 오류가 발생했습니다.
- `DEPENDENCY_ERROR` – 배포가 Amazon S3에서 아티팩트를 다운로드하지 못했거나 ECR 레지스트리에서 이미지를 가져오지 못했습니다.
- `HTTP_ERROR` – HTTP 요청에서 오류가 발생했습니다.
- `NETWORK_ERROR` – 디바이스 네트워크에 오류가 발생했습니다.
- `NUCLEUS_ERROR` – Greengrass nucleus가 구성 요소 또는 활성 nucleus 버전을 찾을 수 없습니다.
- `SERVER_ERROR` – 서버가 요청에 대한 응답으로 500 오류를 반환했습니다.
- `CLOUD_SERVICE_ERROR` – AWS IoT Greengrass 클라우드 서비스에서 오류가 발생했습니다.
- `UNKNOWN_ERROR` - 구성 요소에 의해 확인되지 않은 예외가 발생했습니다.

이 섹션의 많은 오류는 AWS IoT Greengrass 코어 로그에 추가 정보를 보고합니다. 이러한 로그는 코어 디바이스의 로컬 파일 시스템에 저장됩니다. AWS IoT Greengrass 코어 코어 소프트웨어 및 각 개별 구성 요소에 대한 로그가 있습니다. 로그 액세스에 대한 자세한 내용은 [파일 시스템 로그 액세스](#) 섹션을 참조하세요.

권한 오류

ACCESS_DENIED

권한이 올바르게 설정되지 않았기 때문에 AWS 서비스 작업에서 403 오류를 반환할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 더 구체적인 오류 코드에서 확인하세요.

GET_DEPLOYMENT_CONFIGURATION_ACCESS_DENIED

AWS IoT 정책에서 `GetDeploymentConfiguration` 작업 호출 권한을 허용하지 않는 경우 이 오류가 발생할 수 있습니다. 코어 디바이스의 정책에 `greengrass::GetDeploymentConfiguration` 권한을 추가합니다.

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

코어 디바이스 AWS IoT 정책에서 `greengrass:GetComponentVersionArtifact` 권한을 허용하지 않는 경우 이 오류가 발생할 수 있습니다. 코어 디바이스의 정책에 권한을 추가합니다.

RESOLVE_COMPONENT_CANDIDATES_ACCESS_DENIED

코어 디바이스 AWS IoT 정책에서 `greengrass:ResolveComponentCandidates` 권한을 허용하지 않는 경우 이 오류가 발생할 수 있습니다. 코어 디바이스의 정책에 권한을 추가합니다.

GET_ECR_CREDENTIAL_ERROR

배포가 ECR의 프라이빗 레지스트리로 인증할 수 없는 경우 이 오류가 발생할 수 있습니다. 로그에 특정 오류가 있는지 확인한 다음 배포를 다시 시도하세요.

USER_NOT_AUTHORIZED_FOR_DOCKER

Greengrass 사용자에게 Docker를 사용할 권한이 없는 경우 이 오류가 발생할 수 있습니다. Greengrass를 루트로 실행 중이거나 사용자가 `docker` 그룹에 추가되어 있는지 확인합니다. 그런 다음 배포를 다시 시도하세요.

S3_ACCESS_DENIED

Amazon S3 작업에서 403 오류를 반환하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

S3_HEAD_OBJECT_ACCESS_DENIED

디바이스의 토큰 교환 역할에서 구성 요소의 레시피에 지정한 S3 객체 URL에서 AWS IoT Greengrass 코어 소프트웨어가 구성 요소 아티팩트를 다운로드하도록 허용하지 않거나 구성 요소 아티팩트를 사용할 수 없는 경우 이 오류가 발생할 수 있습니다. 토큰 교환 역할이 아티팩트를 사용할 수 있는 S3 객체 URL에 대해 `s3:GetObject`를 허용하고 아티팩트가 있는지 확인하세요.

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

디바이스의 토큰 교환 역할이 아티팩트를 사용할 수 있는 Amazon S3 버킷에 대한 `s3:GetBucketLocation` 권한을 허용하지 않는 경우 이 오류가 발생할 수 있습니다. 디바이스에서 권한을 허용하는지 확인한 다음 배포를 다시 시도하세요.

S3_GET_OBJECT_ACCESS_DENIED

디바이스의 토큰 교환 역할에서 구성 요소의 레시피에 지정한 S3 객체 URL에서 AWS IoT Greengrass 코어 소프트웨어가 구성 요소 아티팩트를 다운로드하도록 허용하지 않거나 구성 요소 아티팩트를 사용할 수 없는 경우 이 오류가 발생할 수 있습니다. 토큰 교환 역할이 아티팩트를 사용할 수 있는 S3 객체 URL에 대해 `s3:GetObject`를 허용하고 아티팩트가 있는지 확인하세요.

요청 오류

NUCLEUS_MISSING_REQUIRED_CAPABILITIES

배포의 `nucleus` 버전이 대규모 구성을 다운로드하거나 Linux 리소스 제한을 설정하는 등 요청된 작업을 수행할 수 없는 경우 이 오류가 발생할 수 있습니다. 작업을 지원하는 `nucleus` 버전으로 배포를 다시 시도하세요.

MULTIPLE_NUCLEUS_RESOLVED_ERROR

배포에서 여러 `nucleus` 구성 요소를 배포하려고 하는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 `nucleus` 소프트웨어 업데이트 페이지에서 이후 버전의 `nucleus`에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

COMPONENT_CIRCULAR_DEPENDENCY_ERROR

배포의 두 구성 요소가 서로 종속되어 있는 경우 이 오류가 발생할 수 있습니다. 배포의 구성 요소가 서로 의존하지 않도록 구성 요소 설정을 수정하세요.

UNAUTHORIZED_NUCLEUS_MINOR_VERSION_UPDATE

배포의 구성 요소에 `nucleus` 마이너 버전 업데이트가 필요하지만 해당 버전이 배포에서 지정되지 않은 경우 이 오류가 발생할 수 있습니다. 이는 다른 버전에 의존하는 구성 요소에 대한 마이너 버전을 실수로 업데이트하는 것을 줄이는 데 도움이 됩니다. 배포에 새 마이너 `nucleus` 버전을 포함하세요.

MISSING_DOCKER_APPLICATION_MANAGER

Docker 애플리케이션 관리자를 배포하지 않고 Docker 구성 요소를 배포하는 경우 이 오류가 발생할 수 있습니다. 배포에 Docker 애플리케이션 관리자가 포함되어 있는지 확인하세요.

MISSING_TOKEN_EXCHANGE_SERVICE

배포에서 토큰 교환 서비스를 배포하지 않고 프라이빗 ECR 레지스트리에서 Docker 이미지 아티팩트를 다운로드하려고 하는 경우 이 오류가 발생할 수 있습니다. 배포에 토큰 교환 서비스가 포함되어 있는지 확인하세요.

COMPONENT_VERSION_REQUIREMENTS_NOT_MET

버전 제약 조건이 충돌하거나 구성 요소 버전이 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 [Error:](#)

[com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>](#) 단원을 참조하십시오.

THROTTLING_ERROR

AWS 서비스 작업이 속도 할당량을 초과하면 이 오류가 발생할 수 있습니다. 배포를 다시 시도하십시오.

CONFLICTED_REQUEST

배포가 한 번에 두 개 이상의 작업을 수행하려고 하기 때문에 AWS 서비스 작업에서 409 오류를 반환할 때 이 오류가 발생할 수 있습니다. 배포를 다시 시도하십시오.

RESOURCE_NOT_FOUND

리소스를 찾을 수 없어 AWS 서비스 작업에서 404 오류를 반환할 때 이 오류가 발생할 수 있습니다. 로그에서 누락된 리소스가 있는지 확인하세요.

RUN_WITH_CONFIG_NOT_VALID

구성 요소를 실행하기 위해 지정한 `posixUser`, `posixGroup` 또는 `windowsUser` 정보가 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 사용자가 유효한지 확인한 다음 배포를 다시 시도하십시오.

UNSUPPORTED_REGION

배포를 위해 지정된 리전을 AWS IoT Greengrass에서 지원하지 않는 경우 이 오류가 발생할 수 있습니다. 리전을 확인하고 배포를 다시 시도하십시오.

IOT_CRED_ENDPOINT_NOT_VALID

구성에 지정된 AWS IoT 자격 증명 엔드포인트가 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 엔드포인트를 확인하고 요청을 다시 시도하십시오.

IOT_DATA_ENDPOINT_NOT_VALID

구성에 지정된 AWS IoT 데이터 엔드포인트가 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 엔드포인트를 확인하고 요청을 다시 시도하세요.

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

구성 요소의 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 사용할 수 없는 경우 이 오류가 발생할 수 있습니다. 아티팩트를 S3 버킷에 업로드했고 아티팩트 URI가 버킷에 있는 아티팩트의 S3 객체 URL과 일치하는지 확인하세요.

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

Amazon S3 버킷을 찾을 수 없는 경우 이 오류가 발생할 수 있습니다. 버킷이 있는지 확인하고 배포를 다시 시도하세요.

S3_GET_OBJECT_RESOURCE_NOT_FOUND

구성 요소의 레시피에 지정한 S3 객체 URL에서 구성 요소 아티팩트를 사용할 수 없는 경우 이 오류가 발생할 수 있습니다. 아티팩트를 S3 버킷에 업로드했고 아티팩트 URI가 버킷에 있는 아티팩트의 S3 객체 URL과 일치하는지 확인하세요.

IO_MAPPING_ERROR

배포 문서 또는 레시피의 구문을 분석할 때 I/O 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

구성 요소 레시피 오류

RECIPE_PARSE_ERROR

레시피 구조에 오류가 있어 배포 레시피의 구문을 분석할 수 없는 경우 이 오류가 발생할 수 있습니다. 레시피의 형식이 올바른지 확인하고 배포를 다시 시도하세요.

RECIPE_METADATA_PARSE_ERROR

클라우드에서 다운로드한 배포 레시피 메타데이터의 구문을 분석할 수 없는 경우 이 오류가 발생할 수 있습니다. 에 문의하세요 지원.

ARTIFACT_URI_NOT_VALID

레시피의 아티팩트 URI 형식이 올바르지 않으면 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

S3_ARTIFACT_URI_NOT_VALID

레시피에 있는 아티팩트의 Amazon S3 URI가 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

DOCKER_ARTIFACT_URI_NOT_VALID

레시피에 있는 아티팩트의 Docker URI가 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

EMPTY_ARTIFACT_URI

아티팩트의 URI가 레시피에서 지정되지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 URI가 누락된 아티팩트를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

EMPTY_ARTIFACT_SCHEME

아티팩트에 대한 URI 체계가 정의되지 않은 경우 이 오류가 발생할 수 있습니다. 로그에서 유효하지 않은 URI를 확인하고 레시피에서 URI를 업데이트한 다음 배포를 다시 시도하세요.

UNSUPPORTED_ARTIFACT_SCHEME

실행 중인 nucleus 버전에서 URI 체계를 지원하지 않는 경우 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.

RECIPE_MISSING_MANIFEST

매니페스트 섹션이 레시피에 포함되어 있지 않은 경우 이 오류가 발생할 수 있습니다. 레시피에 매니페스트를 추가하고 배포를 다시 시도하세요.

RECIPE_MISSING_ARTIFACT_HASH_ALGORITHM

해시 알고리즘이 없는 레시피 내부에 로컬이 아닌 아티팩트가 지정되어 있는 경우 이 오류가 발생할 수 있습니다. 아티팩트에 알고리즘을 추가한 다음 요청을 다시 시도하세요.

ARTIFACT_CHECKSUM_MISMATCH

다운로드한 아티팩트의 다이제스트가 레시피에 지정되어 있는 것과 다른 경우 이 오류가 발생할 수 있습니다. 레시피에 올바른 다이제스트가 포함되어 있는지 확인한 다음 배포를 다시 시도하세요. 자세한 내용은 [Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.](https://docs.aws.amazon.com/greengrass/componentmanager/exceptions/ArtifactChecksumMismatchException.html)을 참조하세요.

COMPONENT_DEPENDENCY_NOT_VALID

배포 레시피에 지정되어 있는 종속성 유형이 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 레시피를 확인한 다음 요청을 다시 시도하세요.

CONFIG_INTERPOLATE_ERROR

레시피 변수를 보간하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

IO_MAPPING_ERROR

배포 문서 또는 레시피의 구문을 분석할 때 I/O 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

AWS 구성 요소 오류, 사용자 구성 요소 오류, 구성 요소 오류

구성 요소에 문제가 있는 경우 다음 오류 코드가 반환됩니다. 보고되는 실제 오류 유형은 오류가 발생하게 한 특정 구성 요소에 따라 다릅니다. Greengrass nucleus가 구성 요소에서 제공한 구성 요소로 식별하면 AWS IoT Greengrass를 반환합니다. `AWS_COMPONENT_ERROR`. 구성 요소가 사용자 구성 요소로 식별되면 Greengrass nucleus는 `USER_COMPONENT_ERROR`를 반환합니다. Greengrass nucleus에서 알지 못하는 경우 `COMPONENT_ERROR`를 반환합니다.

COMPONENT_UPDATE_ERROR

배포 중에 구성 요소가 업데이트되지 않는 경우 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하세요.

COMPONENT_BROKEN

배포 중에 구성 요소가 손상된 경우 이 오류가 발생할 수 있습니다. 구성 요소 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도하세요.

REMOVE_COMPONENT_ERROR

배포 중에 nucleus가 구성 요소를 제거할 수 없는 경우 이 오류가 발생할 수 있습니다. 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도하세요.

COMPONENT_BOOTSTRAP_TIMEOUT

구성 요소의 부트스트랩 태스크가 구성된 제한 시간보다 오래 걸리는 경우 이 오류가 발생할 수 있습니다. 제한 시간을 늘리거나 부트스트랩 태스크의 실행 시간을 줄인 다음 배포를 다시 시도하세요.

COMPONENT_BOOTSTRAP_ERROR

구성 요소의 부트스트랩 태스크에 오류가 있는 경우 이 오류가 발생할 수 있습니다. 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도하세요.

COMPONENT_CONFIGURATION_NOT_VALID

Nucleus가 구성 요소에 대해 배포된 구성을 검증할 수 없는 경우 이 오류가 발생할 수 있습니다. 로그에서 오류 세부 정보를 확인한 다음 배포를 다시 시도하세요.

디바이스 오류

IO_WRITE_ERROR

파일에 쓸 때 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

IO_READ_ERROR

파일에서 읽을 때 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

DISK_SPACE_CRITICAL

배포 요청을 완료할 디스크 공간이 충분하지 않은 경우 이 오류가 발생할 수 있습니다. 사용 가능한 공간이 20MB 이상이거나 더 큰 아티팩트를 보관하기에 충분해야 합니다. 디스크 공간을 확보한 다음 배포를 다시 시도하세요.

IO_FILE_ATTRIBUTE_ERROR

파일 시스템에서 기존 파일 크기를 검색할 수 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

SET_PERMISSION_ERROR

다운로드한 아티팩트 또는 아티팩트 디렉터리에 권한을 설정할 수 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

IO_UNZIP_ERROR

아티팩트의 압축을 해제할 수 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 로그를 확인하세요.

LOCAL_RECIPE_NOT_FOUND

레시피 파일의 로컬 복사본을 찾을 수 없는 경우 이 오류가 발생할 수 있습니다. 배포를 다시 시도하세요.

LOCAL_RECIPE_CORRUPTED

레시피 다운로드 이후 로컬 복사본이 변경된 경우 이 오류가 발생할 수 있습니다. 레시피의 기존 복사본을 삭제하고 배포를 다시 시도하세요.

LOCAL_RECIPE_METADATA_NOT_FOUND

레시피 메타데이터 파일의 로컬 복사본을 찾을 수 없는 경우 이 오류가 발생할 수 있습니다. 배포를 다시 시도하세요.

LAUNCH_DIRECTORY_CORRUPTED

Greengrass nucleus(/greengrass/v2/alts/current)를 시작하는 데 사용된 디렉터리가 마지막으로 nucleus가 시작된 이후 수정되는 경우 이 오류가 발생할 수 있습니다. Nucleus를 다시 시작한 다음 배포를 다시 시도하세요.

HASHING_ALGORITHM_UNAVAILABLE

디바이스의 Java 배포에서 필수 해싱 알고리즘을 지원하지 않거나 구성 요소 레시피에 지정된 해싱 알고리즘이 유효하지 않은 경우 이 오류가 발생할 수 있습니다.

DEVICE_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNLOAD

디바이스 구성에 오류가 있어 배포가 Amazon S3 또는 Greengrass 클라우드에서 아티팩트를 다운로드할 수 없는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 구성 오류를 확인한 다음 배포를 다시 시도하세요.

종속성 오류

DOCKER_ERROR

Docker 이미지를 풀링할 때 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

DOCKER_SERVICE_UNAVAILABLE

Greengrass가 Docker 레지스트리에 로그인할 수 없는 경우 이 오류가 발생할 수 있습니다. 로그에 특정 오류가 있는지 확인한 다음 배포를 다시 시도하세요.

DOCKER_LOGIN_ERROR

Docker에 로그인할 때 예기치 않은 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에 특정 오류가 있는지 확인한 다음 배포를 다시 시도하세요.

DOCKER_PULL_ERROR

레지스트리에서 Docker 이미지를 풀링할 때 예기치 않은 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에 특정 오류가 있는지 확인한 다음 배포를 다시 시도하세요.

DOCKER_IMAGE_NOT_VALID

요청된 Docker 이미지가 없는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인하고 배포를 다시 시도하세요.

DOCKER_IMAGE_QUERY_ERROR

Docker에서 사용 가능한 이미지를 쿼리하는 동안 예기치 않은 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인하고 배포를 다시 시도하세요.

S3_ERROR

Amazon S3 아티팩트를 다운로드하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

S3_RESOURCE_NOT_FOUND

Amazon S3 작업에서 404 오류를 반환하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

S3_BAD_REQUEST

Amazon S3 작업에서 400 오류를 반환하는 경우 이 오류가 발생할 수 있습니다. 특정 오류에 대한 로그를 확인하고 요청을 다시 시도하세요.

HTTP 오류

HTTP_REQUEST_ERROR

HTTP 요청을 하는 동안 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 오류를 확인하세요.

DOWNLOAD_DEPLOYMENT_DOCUMENT_ERROR

배포 문서를 다운로드할 때 HTTP 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 HTTP 오류를 확인하세요.

GET_GREENGRASS_ARTIFACT_SIZE_ERROR

퍼블릭 구성 요소 아티팩트의 크기를 가져올 때 HTTP 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 HTTP 오류를 확인하세요.

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

퍼블릭 구성 요소 아티팩트를 다운로드할 때 HTTP 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 로그에서 특정 HTTP 오류를 확인하세요.

네트워크 오류

NETWORK_ERROR

배포 중에 연결 문제가 있는 경우 이 오류가 발생할 수 있습니다. 디바이스가 인터넷에 연결되어 있는지 확인하고 배포를 다시 시도하세요.

Nucleus 오류

BAD_REQUEST

AWS 클라우드 작업이 400 오류를 반환하면 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류를 일으킨 API를 확인한 다음 nucleus 소프트웨어 업데이트 페이지를 확인하여 이후 버전의 nucleus에서 문제가 수정되었는지 확인하거나 문의하세요 지원.

NUCLEUS_VERSION_NOT_FOUND

코어 디바이스가 활성 nucleus의 버전을 찾을 수 없는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

NUCLEUS_RESTART_FAILURE

Nucleus를 다시 시작해야 하는 배포 중에 nucleus가 다시 시작되지 않는 경우 이 오류가 발생할 수 있습니다. 로더 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

INSTALLED_COMPONENT_NOT_FOUND

Nucleus가 설치된 구성 요소를 찾을 수 없는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

DEPLOYMENT_DOCUMENT_NOT_VALID

디바이스가 유효하지 않은 배포 문서를 수신하는 경우 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하세요.

EMPTY_DEPLOYMENT_REQUEST

디바이스에서 빈 배포 요청을 수신하는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

DEPLOYMENT_DOCUMENT_PARSE_ERROR

배포 요청 형식이 예상 형식과 일치하지 않는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

배포 요청에 유효하지 않은 구성 요소 메타데이터가 포함되어 있는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

LAUNCH_DIRECTORY_CORRUPTED

Greengrass 디바이스를 한 그룹에서 다른 그룹으로 이동한 다음 Greengrass를 다시 시작해야 하는 배포를 사용하여 원래 그룹으로 다시 돌아오는 경우 이 오류가 발생할 수 있습니다. 오류를 해결하려면 디바이스에서 Greengrass의 시작 디렉터리를 다시 생성하세요.

자세한 내용은 [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](#) 단원을 참조하십시오.

서버 오류

SERVER_ERROR

서비스가 현재 요청을 처리할 수 없기 때문에 AWS 서비스 작업에서 500 오류를 반환할 때 이 오류가 발생할 수 있습니다. 나중에 배포를 다시 시도하세요.

S3_SERVER_ERROR

Amazon S3 작업에서 500 오류를 반환하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

클라우드 서비스 오류

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

Greengrass 클라우드 서비스가 ResolveComponentCandidates 작업에 호환되지 않는 응답을 보내는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

DEPLOYMENT_DOCUMENT_SIZE_EXCEEDED

요청된 배포 문서가 최대 크기 할당량을 초과한 경우 이 오류가 발생할 수 있습니다. 배포 문서의 크기를 줄이고 배포를 다시 시도하세요.

GREENGRASS_ARTIFACT_SIZE_NOT_FOUND

Greengrass가 퍼블릭 구성 요소 아티팩트의 크기를 가져올 수 없는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

DEPLOYMENT_DOCUMENT_NOT_VALID

디바이스가 유효하지 않은 배포 문서를 수신하는 경우 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하세요.

EMPTY_DEPLOYMENT_REQUEST

디바이스에서 빈 배포 요청을 수신하는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

DEPLOYMENT_DOCUMENT_PARSE_ERROR

배포 요청 형식이 예상 형식과 일치하지 않는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

배포 요청에 유효하지 않은 구성 요소 메타데이터가 포함되어 있는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

일반 오류

이러한 일반 오류에는 연결된 오류 유형이 없습니다.

DEPLOYMENT_INTERRUPTED

nucleus 종료 또는 기타 외부 이벤트로 인해 배포를 완료할 수 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

ARTIFACT_DOWNLOAD_ERROR

아티팩트를 다운로드하는 데 문제가 있는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

NO_AVAILABLE_COMPONENT_VERSION

구성 요소 버전이 클라우드 또는 로컬에 없거나 종속성 해결 충돌이 있는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

COMPONENT_PACKAGE_LOADING_ERROR

다운로드한 아티팩트를 처리하는 동안 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

CLOUD_API_ERROR

AWS 서비스 API를 호출하는 동안 오류가 발생하면 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

IO_ERROR

배포 중에 I/O 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 추가 오류 코드 또는 로그를 확인하세요.

COMPONENT_UPDATE_ERROR

배포 중에 구성 요소가 업데이트되지 않는 경우 이 오류가 발생할 수 있습니다. 추가 오류 코드를 확인하거나 로그를 확인하여 오류의 원인을 확인하세요.

알 수 없는 오류

DEPLOYMENT_FAILURE

확인되지 않은 예외가 발생하여 배포에 실패하는 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

DEPLOYMENT_TYPE_NOT_VALID

배포 유형이 유효하지 않은 경우 이 오류가 발생할 수 있습니다. 로그를 확인하여 오류의 원인을 확인한 다음 nucleus 소프트웨어 업데이트 페이지에서 이후 버전의 nucleus에서 문제가 해결되었는지 확인하거나 지원에 문의하세요.

세부 구성 요소 상태 코드

이 섹션의 상태 코드와 해결 방법을 사용하여 Greengrass nucleus 버전 2.8.0 이상을 사용하는 경우 구성 요소 관련 문제를 해결하세요.

이 주제의 많은 상태는 AWS IoT Greengrass 코어 로그에서 추가 정보를 보고합니다. 이러한 로그는 코어 디바이스의 로컬 파일 시스템에 저장됩니다. 각 개별 구성 요소에 대한 로그가 있습니다. 로그 액세스에 대한 자세한 내용은 [파일 시스템 로그 액세스](#) 섹션을 참조하세요.

INSTALL_ERROR

설치 스크립트를 실행하는 동안 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 오류 코드는 구성 요소 로그에서 보고됩니다. 설치 스크립트에서 오류를 확인하고 구성 요소를 다시 배포하세요.

INSTALL_CONFIG_NOT_VALID

레시피의 `install` 섹션이 유효하지 않아 구성 요소 설치를 완료할 수 없는 경우 이 오류가 발생할 수 있습니다. 레시피의 설치 섹션에서 오류를 확인하고 배포를 다시 시도하세요.

INSTALL_IO_ERROR

구성 요소를 설치하는 동안 I/O 오류가 발생하면 이 오류가 발생할 수 있습니다. 오류에 대한 자세한 내용은 구성 요소 오류 로그에서 확인하세요.

INSTALL_MISSING_DEFAULT_RUNWITH

AWS IoT Greengrass에서 구성 요소를 설치할 때 사용할 사용자 또는 그룹을 결정할 수 없는 경우 이 오류가 발생할 수 있습니다. 설치 레시피의 `runWith` 섹션에 유효한 사용자 또는 그룹이 포함되어 있는지 확인합니다.

INSTALL_TIMEOUT

설치 스크립트가 구성된 제한 시간 내에 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피의 `install` 섹션에 지정된 Timeout 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 설치 스크립트를 수정하세요.

STARTUP_ERROR

시작 스크립트를 실행하는 동안 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 오류 코드는 구성 요소 로그에서 보고됩니다. 설치 스크립트에서 오류를 확인하고 구성 요소를 다시 배포하세요.

STARTUP_CONFIG_NOT_VALID

레시피의 `startup` 섹션이 유효하지 않아 구성 요소 설치를 완료할 수 없는 경우 이 오류가 발생할 수 있습니다. 레시피의 시작 섹션에서 오류를 확인하고 배포를 다시 시도하세요.

STARTUP_IO_ERROR

구성 요소를 시작하는 동안 I/O 오류가 발생하면 이 오류가 발생할 수 있습니다. 오류에 대한 자세한 내용은 구성 요소 오류 로그에서 확인하세요.

START_MISSING_DEFAULT_RUNWITH

AWS IoT Greengrass에서 구성 요소를 실행할 때 사용할 사용자 또는 그룹을 결정할 수 없는 경우 이 오류가 발생할 수 있습니다. 시작 레시피의 `runWith` 섹션에 유효한 사용자 또는 그룹이 포함되어 있는지 확인합니다.

STARTUP_TIMEOUT

시작 스크립트가 구성된 제한 시간 내에 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피의 `startup` 섹션에 지정된 Timeout 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 시작 스크립트를 수정하세요.

RUN_ERROR

구성 요소 스크립트를 실행하는 동안 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 오류 코드는 구성 요소 로그에서 보고됩니다. 실행 스크립트에서 오류를 확인하고 구성 요소를 다시 배포하세요.

RUN_MISSING_DEFAULT_RUNWITH

AWS IoT Greengrass에서 구성 요소를 실행할 때 사용할 사용자 또는 그룹을 결정할 수 없는 경우 이 오류가 발생할 수 있습니다. 실행 레시피의 `runWith` 섹션에 유효한 사용자 또는 그룹이 포함되어 있는지 확인합니다.

RUN_CONFIG_NOT_VALID

레시피의 `run` 섹션이 유효하지 않아 구성 요소를 실행할 수 없는 경우 이 오류가 발생할 수 있습니다. 레시피의 실행 섹션에서 오류를 확인하고 배포를 다시 시도하세요.

RUN_IO_ERROR

구성 요소가 실행되는 동안 I/O 오류가 발생하면 이 오류가 발생할 수 있습니다. 오류에 대한 자세한 내용은 구성 요소 오류 로그에서 확인하세요.

RUN_TIMEOUT

실행 스크립트가 구성된 제한 시간 내에 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피의 `run` 섹션에 지정된 Timeout 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 실행 스크립트를 수정하세요.

SHUTDOWN_ERROR

구성 요소 스크립트를 종료하는 동안 오류가 발생하는 경우 이 오류가 발생할 수 있습니다. 오류 코드는 구성 요소 로그에서 보고됩니다. 종료 스크립트에서 오류를 확인하고 구성 요소를 다시 배포하세요.

SHUTDOWN_TIMEOUT

종료 스크립트가 구성된 제한 시간 내에 완료되지 않은 경우 이 오류가 발생할 수 있습니다. 레시피의 `shutdown` 섹션에 지정된 Timeout 기간을 늘리거나 구성된 제한 시간 내에 완료되도록 실행 스크립트를 수정하세요.

AWS IoT Greengrass Version 2 리소스에 태그 지정

태그를 사용하여 AWS IoT Greengrass의 리소스를 구성하고 관리할 수 있습니다. 태그를 사용하여 리소스에 메타데이터를 할당하고, IAM 정책에 태그를 사용하여 리소스에 대한 조건부 액세스를 정의할 수 있습니다.

Note

현재 Greengrass 리소스 태그는 AWS IoT 결제 그룹 또는 비용 할당 보고서에서 지원되지 않습니다.

에서 태그 사용 AWS IoT Greengrass V2

태그를 사용하여 용도, 소유자, 환경 또는 사용 사례에 맞는 기타 분류별로 AWS IoT Greengrass 리소스를 분류할 수 있습니다. 동일한 유형의 리소스가 여러 개인 경우 태그를 사용하면 특정 리소스를 더 빠르게 식별할 수 있습니다.

각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 예를 들어 코어 디바이스에 대한 태그 세트를 정의하여 디바이스를 소유한 고객이 디바이스를 추적하는 데 도움을 줄 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.

로 태그 지정 AWS Management Console

의 태그 편집기는 모든 AWS 서비스의 리소스에 대한 태그를 생성하고 관리할 수 있는 중앙 통합 방법을 AWS Management Console 제공합니다. 자세한 내용을 알아보려면 AWS Resource Groups 사용 설명서의 [태그 편집기](#)를 참조하세요.

AWS IoT Greengrass V2 API로 태그 지정

AWS IoT Greengrass V2 API를 사용하여 태그를 사용할 수도 있습니다. 태그를 생성하기 전에 태그 지정에 대한 제한 사항을 유의하세요. 자세한 내용은 AWS 일반 참조의 [태그 이름 지정 및 사용 규칙](#)을 참조하세요.

- 리소스를 만들 때 태그를 추가하려면 리소스의 tags 속성에서 태그를 정의합니다.
- 기존 리소스에 태그를 추가하거나 태그 값을 업데이트하려면 [TagResource](#) 작업을 사용합니다.
- 리소스에서 태그를 제거하려면 [UntagResource](#) 작업을 사용합니다.

- 리소스와 연결된 태그를 검색하려면 [ListTagsForResource](#) 작업을 사용하거나, 해당 리소스를 설명하고 tags 속성을 검사합니다.

다음 표에는 AWS IoT Greengrass V2 API를 사용하여 태그를 지정할 수 있는 리소스와 해당 Create 및 Describe 또는 Get 작업이 나열되어 있습니다.

태그 지정 가능한 AWS IoT Greengrass V2 리소스

Resource	작업 생성	작업 설명 또는 가져오기
코어 디바이스	없음. 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 실행하여 코어 디바이스를 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. 그러면 이전 값이 새 값으로 바뀝니다. 값을 빈 문자열로 설정할 수 있지만, 값을 Null로 설정할 수는 없습니다.

리소스를 삭제하면 해당 리소스와 연결된 태그도 삭제됩니다.

IAM 정책에 태그 사용

IAM 정책에서는 리소스 태그를 사용하여 사용자 액세스 및 권한을 제어할 수 있습니다. 예를 들어, 정책은 사용자가 특정 태그가 있는 리소스만 생성하도록 허용할 수 있습니다. 또한 정책은 사용자가 특정 태그가 있는 리소스를 생성하거나 수정하는 것을 제한할 수 있습니다.

 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 설명의 Condition 요소(Condition 블록이라고도 함)에 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 지정이 가능한 리소스를 생성 또는 수정하는 경우 특정 태그를 사용하거나 사용하지 않도록 요구합니다.

`aws:TagKeys: [tag-key, ...]`

태그 지정이 가능한 리소스를 생성 또는 수정하는 경우 특정 태그 키 세트를 사용하거나 사용하지 않도록 요구합니다.

 Note

IAM 정책의 조건 컨텍스트 키 및 값은 태그 지정 가능한 리소스가 필수 파라미터인 작업에만 적용됩니다. 예를 들어 [GetComponent](#)에 대한 태그 기반 조건부 액세스를 설정할 수 있습니다.

자세한 내용은 IAM 사용 설명서의 [AWS 리소스 태그를 사용하여 리소스에 대한 액세스 제어](#) 및 IAM JSON 정책 참조를 참조하세요. https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies.html

를 사용하여 AWS IoT Greengrass 리소스 생성 AWS CloudFormation

AWS IoT Greengrass 는 AWS 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 리소스를 모델링하고 설정하는 데 도움이 되는 AWS CloudFormation 서비스와 통합됩니다. 원하는 모든 AWS 리소스(예: 구성 요소 버전 및 배포)를 설명하는 템플릿을 생성하고 해당 리소스를 CloudFormation 프로비저닝하고 구성합니다.

를 사용하면 템플릿을 재사용하여 AWS IoT Greengrass 리소스를 일관되고 반복적으로 설정할 CloudFormation 수 있습니다. 리소스를 한 번 설명한 다음 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝합니다.

AWS IoT Greengrass 및 CloudFormation 템플릿

AWS IoT Greengrass 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 [CloudFormation 템플릿](#)을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이러한 템플릿은 CloudFormation 스택에서 프로비저닝하려는 리소스를 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 CloudFormation Designer를 사용하여 CloudFormation 템플릿을 시작할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용자 안내서에서 [CloudFormation Designer란?](#)을 참조하세요.

AWS IoT Greengrass 는에서 구성 요소 버전 및 배포 생성을 지원합니다 CloudFormation. 구성 요소 버전과 배포에 대한 JSON 및 YAML 템플릿의 예제를 포함한 자세한 내용은 AWS CloudFormation 사용 설명서의 [AWS IoT Greengrass 리소스 유형 참조](#)를 참조하세요.

ComponentVersion 템플릿 예제

다음은 간단한 구성 요소의 버전에 대한 YAML 템플릿입니다. JSON 레시피는 읽기 쉽도록 줄이 나누어져 있습니다.

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
  TestSimpleComponentVersion:
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
      InlineRecipe: !Sub
        - "{\n
```

```

    "RecipeFormatVersion": "\"2020-01-25\"",
    "ComponentName": "\"component1\"",
    "ComponentVersion": "\"${ComponentVersion}\"",
    "ComponentType": "\"aws.greengrass.generic\"",
    "ComponentDescription": "\"This\"",
    "ComponentPublisher": "\"You\"",
    "Manifests": [
      {
        "Platform": {
          "os": "\"darwin\""
        },
        "Lifecycle": {},
        "Artifacts": []
      },
      {
        "Lifecycle": {},
        "Artifacts": []
      }
    ],
    "Lifecycle": {
      "install": {
        "script": "\"yuminstallpython\""
      }
    }
  }
- { ComponentVersion: !Ref ComponentVersion }

```

배포 템플릿 예제

다음은 배포에 대한 간단한 템플릿이 정의되는 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
```

에 대해 자세히 알아보기 CloudFormation

에 대해 자세히 알아보려면 다음 리소스를 CloudFormation참조하세요.

- [AWS CloudFormation](#)
- [AWS CloudFormation 사용 설명서](#)
- [CloudFormation API Reference](#)
- [AWS CloudFormation 명령줄 인터페이스 사용자 안내서](#)

오픈 소스 AWS IoT Greengrass 코어 소프트웨어

AWS IoT Greengrass 코어 소프트웨어의 AWS IoT Greengrass Version 2 엣지 런타임(nucleus) 및 기타 구성 요소는 오픈 소스입니다. 즉, 코드를 검토하여 애플리케이션과의 연동 문제를 해결할 수 있습니다. 특정 소프트웨어 및 하드웨어 요구 사항에 맞게 AWS IoT Greengrass 코어 소프트웨어를 사용자 지정하고 확장할 수도 있습니다.

AWS IoT Greengrass 코어 소프트웨어의 오픈 소스 리포지토리에 대한 자세한 내용은 GitHub의 [aws-greengrass](#) 조직을 참조하세요. 오픈 소스 소프트웨어 사용에는 [해당 GitHub 리포지토리](#)의 오픈 소스 라이선스가 적용됩니다.

오픈 소스 라이선스가 적용되지 않는 AWS IoT Greengrass 코어 소프트웨어 및 구성 요소의 사용에는 [AWS Greengrass 코어 소프트웨어 라이선스](#)가 적용됩니다.

AWS IoT Greengrass V2 개발자 안내서의 문서 기록

다음 표에서는 이 릴리스에 대한 설명서를 설명합니다 AWS IoT Greengrass Version 2.

- API 버전: 2020-11-30

변경 사항	설명	날짜
새도 관리자 v2.3.13 릴리스	새도 관리자 v2.3.13를 사용할 수 있습니다. 이 릴리스에서는 동기화 방향 구성에서 허용하는 경우에도 로컬 새도우 업데이트가 클라우드에 간헐적으로 동기화되지 않는 문제를 해결합니다.	2026년 2월 18일
Greengrass nucleus lite 구성 요소 v2.3.2 릴리스	Greengrass nucleus lite 구성 요소의 버전 2.3.2를 사용할 수 있습니다. 이 버전은 올바른 nucleus 버전 보고를 위해 버전 파일을 업데이트합니다.	2026년 2월 12일
Greengrass nucleus lite 구성 요소 v2.3.1 릴리스	Greengrass nucleus lite 구성 요소의 버전 2.3.1을 사용할 수 있습니다. 이 버전에는 여러 가지 버그 수정 및 개선 사항이 포함되어 있습니다.	2026년 2월 11일
Greengrass CLI v2.16.1 릴리스	Greengrass CLI 구성 요소 v2.16.1을 사용할 수 있습니다.	2025년 12월 23일
AWS IoT Greengrass 코어 v2.16.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.16.1과 업데이트 제공 구성 요소를 AWS제공합니다.	2025년 12월 23일

시스템 로그 전달자 v2.1.0 릴리스	시스템 로그 전달자 v2.1.0을 사용할 수 있습니다.	2025년 11월 6일
새도 관리자 v2.3.12 릴리스	새도 관리자 v2.3.12를 사용할 수 있습니다.	2025년 11월 6일
로그 관리자 v2.3.11 릴리스	로그 관리자 구성 요소 v2.3.11를 사용할 수 있습니다.	2025년 11월 6일
Greengrass nucleus lite 구성 요소 v2.3.0 릴리스	Greengrass nucleus lite 구성 요소의 버전 2.3.0을 사용할 수 있습니다. 이 버전에는 TPM 2.0, 광범위한 운영 체제 및 RestartComponent IPC에 대한 지원이 추가되었습니다. 기타 여러 버그 수정 및 개선 사항도 포함합니다.	2025년 11월 6일
Greengrass CLI v2.16.0 릴리스	Greengrass CLI 구성 요소 v2.16.0을 사용할 수 있습니다.	2025년 11월 6일
AWS IoT Greengrass 코어 v2.16.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.16.0과 업데이트 제공 구성 요소를 AWS제공합니다.	2025년 11월 6일
AWS IoT Greengrass 코어 v2.15.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.15.1과 업데이트 제공 구성 요소를 AWS제공합니다.	2025년 11월 6일
시스템 로그 전달자 v2.0.1 릴리스	시스템 로그 전달자 v2.0.1을 사용할 수 있습니다. 이 버전에서는 aarch64(arm64) 시스템을 올바르게 지원하도록 구성 요소 레시피를 업데이트합니다.	2025년 10월 2일

Greengrass nucleus lite 구성 요소 v2.2.2 릴리스	Greengrass nucleus lite 구성 요소 버전 2.2.2를 사용할 수 있습니다. 이 버전에는 여러 가지 버그 수정 및 개선 사항이 포함되어 있습니다.	2025년 8월 18일
시스템 로그 전달자 v2.0.0 릴리스	시스템 로그 전달자 구성 요소 v2.0.0을 사용할 수 있습니다. 초기 버전입니다.	2025년 7월 25일
Greengrass nucleus lite 구성 요소 v2.2.1 릴리스	Greengrass nucleus lite 구성 요소 버전 2.2.1을 사용할 수 있습니다. 이 버전에서는 nucleus가 TES 자격 증명을 가져오지 못하는 문제를 수정합니다.	2025년 7월 25일
Greengrass nucleus lite 구성 요소 v2.2.0 릴리스	Greengrass nucleus lite 구성 요소 버전 2.2.0을 사용할 수 있습니다. 이 버전에서는 컨테이너 이미지 아티팩트 URI에 대한 지원을 추가합니다.	2025년 7월 3일
AWS IoT Greengrass 코어 v2.15.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.15.0과 업데이트 제공 구성 요소를 AWS제공합니다.	2025년 7월 3일
보안 터널링 v1.1.3 릴리스	보안 터널링 v1.1.3을 사용할 수 있습니다. 이 버전은 구성 요소가 간접적으로 호출하는 기본 AWS IoT Device Client를 버전 1.10.0에서 1.10.1로 업그레이드합니다. 이 버전은 또한 보안 터널링 구성 요소 v1.1.2에서 발생하던 GNU C 라이브러리(glibc) 호환성 문제를 수정합니다.	2025년 5월 16일

[Greengrass nucleus lite 구성
요소 v2.1.0 릴리스](#)

Greengrass nucleus lite 구성
요소 버전 2.1.0을 사용할 수 있
습니다. 이 버전은 HTTP 프록
시 지원을 추가하며, 기타 여러
버그 수정 및 개선 사항도 포함
합니다.

2025년 5월 5일

[보안 암호 관리자 v2.2.5 릴리
스](#)

보안 암호 관리자 v2.2.5를 사
용할 수 있습니다. 이 버전은 로
컬 캐시에 없는 AWS 클라우드
경우에서 보안 암호를 가져오
지 않는 문제를 해결합니다.

2025년 4월 16일

[AWS IoT Greengrass 코어
v2.14.3 소프트웨어 업데이트](#)

이 릴리스는 Greengrass
nucleus 구성 요소의 버전
2.14.3과 업데이트 제공 구성
요소를 AWS제공합니다.

2025년 4월 11일

[보안 암호 관리자 v2.2.4 릴리
스](#)

보안 암호 관리자 v2.2.4를 사
용할 수 있습니다. 이 버전에서
는 로컬 암호 저장소에 쓰는 빈
도를 줄입니다. 이제 보안 암호
관리자는 보안 암호가 업데이
트될 때만 로컬 저장소에 기록
합니다.

2025년 4월 8일

[Greengrass CLI v2.14.3 릴리
스](#)

Greengrass CLI 구성 요소
v2.14.3을 사용할 수 있습니다.

2025년 4월 8일

[보안 터널링 v1.1.2 릴리스](#)

보안 터널링 v1.1.2를 사용할 수 있습니다. 이 버전은 구성 요소가 간접적으로 호출하는 기본 AWS IoT Device Client를 버전 1.9.0에서 버전 1.10.0으로 업그레이드합니다. 또한 보안 터널링 v1.1.2에서는 보안 터널을 통해 Greengrass V2 코어 디바이스에서 소스 디바이스로 대용량 파일을 전달할 수 없던 페이로드 전송 문제를 수정합니다.

2025년 3월 27일

[Greengrass CLI v2.14.2 릴리스](#)

Greengrass CLI 구성 요소 v2.14.2를 사용할 수 있습니다.

2025년 3월 27일

[AWS IoT Greengrass 코어 v2.14.2 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.14.2와 업데이트 제공 구성 요소를 AWS제공합니다.

2025년 3월 27일

[보안 암호 관리자 v2.2.3 릴리스](#)

보안 암호 관리자 v2.2.3을 사용할 수 있습니다. 이 버전에서는 코어 디바이스가 오프라인 상태이고 디바이스의 보안 서비스(HSM 등)를 사용할 수 없을 때 보안 암호 관리자가 로컬에 저장된 암호를 삭제하는 문제를 수정합니다.

2025년 3월 18일

[스트림 관리자 v2.2.1 릴리스](#)

이제 스트림 관리자 v2.2.1을 사용할 수 있습니다. 이 릴리스는 스트림 관리자가 Kinesis 대상으로 메시지를 내보내지 못하는 문제를 수정하며, Kinesis 대상으로의 내보내기 성능도 향상합니다.

2025년 3월 12일

[클라이언트 디바이스 인증 구성 요소 v2.5.3 릴리스](#)

클라이언트 디바이스 인증 구성 요소 v2.5.3을 사용할 수 있습니다. 이 릴리스는 클라이언트 인증서가 최신 상태가 아니어서 클라이언트 디바이스가 코어 디바이스에 연결하지 못하는 문제를 수정합니다.

2025년 3월 12일

[AWS IoT 플릿 프로비저닝 플러그인 v1.2.2 릴리스](#)

AWS IoT 플릿 프로비저닝 플러그인 v1.2.2를 사용할 수 있습니다. 이 릴리스에서는 사용자 지정 코어 디바이스 인증서(certificatePath) 및 프라이빗 키(privateKeyPath) 경로에 대한 지원을 추가합니다.

2025년 3월 12일

[보안 터널링 v1.1.1 릴리스](#)

보안 터널링 v1.1.1을 사용할 수 있습니다. 이 버전에서는 Greengrass nucleus lite를 지원하기 위한 구성을 추가합니다.

2025년 2월 7일

[Greengrass CLI v2.14.1 릴리스](#)

Greengrass CLI 구성 요소 v2.14.1을 사용할 수 있습니다.

2025년 2월 7일

[AWS IoT Greengrass 코어 v2.14.1 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.14.1과 업데이트 제공 구성 요소를 AWS제공합니다.

2025년 2월 7일

[Greengrass nucleus lite 구성 요소 v2.0.2 릴리스](#)

Greengrass nucleus lite 구성 요소 버전 2.0.2를 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.

2025년 2월 6일

Greengrass nucleus lite 구성 요소 v2.0.1 릴리스	Greengrass nucleus lite 구성 요소 버전 2.0.1을 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2025년 1월 28일
Greengrass CLI v2.14.0 릴리스	Greengrass CLI 구성 요소 v2.14.0을 사용할 수 있습니다.	2024년 12월 24일
스트림 관리자 v2.2.0 릴리스	이제 스트림 관리자 v2.2.0을 사용할 수 있습니다.	2024년 12월 16일
새도 관리자 v2.3.10 릴리스	새도 관리자 v2.3.10를 사용할 수 있습니다.	2024년 12월 16일
보안 터널링 v1.1.0 릴리스	보안 터널링 v1.1.0을 사용할 수 있습니다. 이 버전에서는 Greengrass nucleus lite에 대한 레시피 지원을 추가합니다.	2024년 12월 16일
새로운 nucleus 원격 측정 이미터 구성 요소	nucleus 원격 측정 이미터 구성 요소의 버전 1.0.10을 사용할 수 있습니다.	2024년 12월 16일
Modbus-RTU 프로토콜 어댑터 v2.1.10 릴리스	Modbus-RTU 프로토콜 어댑터 구성 요소 v2.1.10를 사용할 수 있습니다.	2024년 12월 16일
로그 관리자 v2.3.9 릴리스	로그 관리자 구성 요소 v2.3.9를 사용할 수 있습니다.	2024년 12월 16일
로컬 디버그 콘솔 v2.4.4 릴리스	로컬 디버그 콘솔 구성 요소 v2.4.4를 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2024년 12월 16일
Lambda 관리자 v2.3.5 릴리스	Lambda 관리자 구성 요소 v2.3.5를 사용할 수 있습니다.	2024년 12월 16일

IP 감지기 v2.2.1 릴리스	IP 감지기 구성 요소 v2.2.1을 사용할 수 있습니다.	2024년 12월 16일
Greengrass nucleus lite 구성 요소 v2.0.0 릴리스	Greengrass nucleus lite 구성 요소 버전 2.0.0을 사용할 수 있습니다. 이 버전이 초기 버전입니다.	2024년 12월 16일
디스크 스펠러 v1.0.5 릴리스	디스크 스펠러 구성 요소 v1.0.5를 사용할 수 있습니다.	2024년 12월 16일
클라이언트 디바이스 인증 구성 요소 v2.5.2 릴리스	클라이언트 디바이스 인증 구성 요소 v2.5.2를 사용할 수 있습니다.	2024년 12월 16일

[AWS IoT Greengrass 코어 v2.14.0 소프트웨어 업데이트](#)

2024년 12월 16일

이 릴리스에서는 Greengrass nucleus 구성 요소 버전 2.14.0과 새로운 AWS IoT Greengrass nucleus lite 업데이트를 제공합니다. AWS IoT Greengrass nucleus lite는 버전 2에서 AWS IoT Greengrass 사용할 수 있는 새로운 런타임입니다. 이는 메모리 사용량을 줄인 경량 대안으로, 리소스가 제한된 디바이스에 적합합니다. 이는 nucleus 기능의 일부만 구현하며, 향후 릴리스에서 기능 호환성을 점차 확대할 예정입니다. 소스 코드는 현재 [Github](#)에서 확인할 수 있습니다. nucleus lite 런타임을 사용하면 다음을 수행할 수 있습니다.

- Greengrass 코어 디바이스에 구성 요소를 배포할 수 있습니다. 동일한 레시피 형식을 사용하지만 일부 고급 기능은 아직 지원되지 않을 수 있습니다.
- Greengrass 구성 요소로 배포된 애플리케이션은 디바이스 SDKs를 사용하여 AWS IoT Core MQTT 액세스, 로컬 pub/sub 및 Greengrass 구성 액세스와 같은 지원되는 Greengrass IPC APIs에 액세스할 수 있습니다. [지원되는 IPC API](#) 목록은 호환성 차트를 참조하세요.

- nucleus lite 지원을 위해 일부 AWS 관리형 구성 요소가 업데이트되었습니다. 기존 호환 구성 요소 목록은 [AWS 제공 구성 요소](#)를 참조하세요.

새로운 기능:

- 메모리 및 디스크 공간 사용량이 더 적습니다(RAM 5MB 미만, 저장소 5MB 미만).
- 구성 요소가 호스트 시스템의 서비스 관리자(systemd, 현재 지원되는 Linux 플랫폼 기준)와 통합됩니다.

주의 사항:

- AWS IoT Greengrass nucleus lite 레시피는 대/소문자를 구분합니다. 정확한 (키) 대소문자 구분이 <https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html> 레시피 참조에 있는 것처럼 사용되었는지 확인합니다.
- nucleus lite 런타임은 사물 그룹 배포를 지원하며, (단일) 코어 디바이스 배포 대상 유형은 아직 지원하지 않습니다. 단일 Greengrass 디바이스에 배포하려면 해당 디

바이스만 포함된 사물 그룹을 사용하세요.

- nucleus lite 런타임은 제한된 메모리 리소스를 사용합니다. classic 런타임에서 사용량에 따라 확장되는 기능이 lite에서는 리소스 부족으로 실패할 수 있습니다. 여기에는 한 번에 최대 50개의 MQTT 구독만 허용되는 현행 제한과 레시피 파일 크기 및 배포에 대한 상한이 포함됩니다. 이러한 제한의 일부는 lite 런타임을 직접 컴파일할 경우 컴파일 시점에 구성할 수 있습니다.
- nucleus lite 런타임은 Java를 포함하지 않습니다. Java가 필요한 구성 요소를 사용하려면 시스템에 Java가 설치되어 있어야 하거나, 구성 요소를 통해 Java를 설치해야 합니다.
- nucleus lite 런타임을 소스에서 컴파일하여 시스템에 맞게 자체 빌드를 사용하는 것을 권장합니다. Yocto 시스템의 경우, 시스템 이미지에 nucleus lite 런타임을 통합할 수 있는 계층이 제공됩니다.
- 현재 nucleus lite는 systemd를 사용하는 Linux 시스템 또는 systemd를 사용하는 컨테이너 이미지를 가집니다.

- 레시피 스크립트로 Docker 컨테이너를 관리할 수 있지만, Greengrass 관리형 컨테이너 아티팩트는 아직 제공되지 않습니다.
- nucleus lite 런타임은 PKCS11 모듈에 저장된 키를 아직 지원하지 않습니다. 사용 사례에 보안 요소에 저장된 키가 필요한 경우, 현재 classic 런타임이 이 사용 사례를 지원할 수 있습니다. 디바이스 자격 증명의 유출을 방지하려면 프로덕션 디바이스가 전체 디스크 암호화를 사용하고 있는지 확인합니다.

nucleus lite 도입과 함께, nucleus v2.14.0도 출시됩니다. 이 업데이트에는 기존 Greengrass nucleus에 대한 주요 개선 사항이 포함되어 있습니다.

주요 기능 및 개선 사항

- 새로운 듀얼 스택 엔드포인트 지원을 통해 IPv6 네트워크 통신이 가능합니다.
- nucleus 재시작 실패 및 디렉터리 손상에 대한 복원력이 강화되었습니다.
- IPC PubSub 구독 종료 시 발생하던 메모리 누수 문제를 수정합니다.

스트림 관리자 v2.1.13 릴리스	이제 스트림 관리자 v2.1.13을 사용할 수 있습니다. 이 릴리스에는 대한 FIPS 엔드포인트에 대한 지원을 추가합니다. AWS IoT SiteWise	2024년 8월 26일
새도 관리자 v2.3.9 릴리스	새도 관리자 v2.3.9를 사용할 수 있습니다.	2024년 8월 26일
Modbus-RTU 프로토콜 어댑터 v2.1.9 릴리스	Modbus-RTU 프로토콜 어댑터 구성 요소 v2.1.9를 사용할 수 있습니다.	2024년 8월 26일
로그 관리자 v2.3.8 릴리스	로그 관리자 구성 요소 v2.3.8을 사용할 수 있습니다.	2024년 8월 26일
로컬 디버그 콘솔 v2.4.3 릴리스	로컬 디버그 콘솔 구성 요소 v2.4.3을 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2024년 8월 26일
디스크 스폰서 v1.0.4 릴리스	디스크 스폰서 구성 요소 v1.0.4를 사용할 수 있습니다.	2024년 8월 26일
AWS IoT Greengrass 코어 v2.13.0 소프트웨어 업데이트	이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.13.0이 제공됩니다.	2024년 8월 26일
새로운 nucleus 원격 측정 이미터 구성 요소	nucleus 원격 측정 이미터 구성 요소의 버전 1.0.9를 사용할 수 있습니다.	2024년 8월 23일
Lambda 관리자 v2.3.4 릴리스	Lambda 관리자 구성 요소 v2.3.4를 사용할 수 있습니다.	2024년 8월 23일
Greengrass CLI v2.13.0 릴리스	Greengrass CLI 구성 요소 v2.13.0을 사용할 수 있습니다.	2024년 8월 23일

[클라이언트 디바이스 인증 구성 요소 v2.5.1 릴리스](#)

클라이언트 디바이스 인증 구성 요소 v2.5.0을 사용할 수 있습니다. 이 릴리스에서는 FIPS 엔드포인트 지원을 추가합니다.

2024년 8월 23일

[레시피 검증](#)

구성 요소 버전을 생성할 때 구성 요소 레시피를 검증하는 레시피 검증 기능이 추가되었습니다.

2024년 8월 15일

[IP 감지기 v2.2.0 릴리스](#)

IP 감지기 구성 요소 v2.2.0을 사용할 수 있습니다. 이 릴리스에서는 IPv6 지원을 추가합니다. 이제 로컬 메시징에 IPv6를 사용할 수 있습니다.

2024년 7월 29일

[샤도 관리자 v2.3.8 릴리스](#)

샤도 관리자 v2.3.8을 사용할 수 있습니다. 이 릴리스에서는 MQTT 클라이언트 연결 중에 샤도 관리자가 교착 상태를 생성하는 문제를 해결합니다.

2024년 6월 5일

[Greengrass CLI v2.12.6 릴리스](#)

Greengrass CLI 구성 요소 v2.12.6을 사용할 수 있습니다.

2024년 5월 24일

[AWS IoT Greengrass 코어 v2.12.6 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.6과 업데이트 제공 구성 요소를 AWS제공합니다.

2024년 5월 24일

[AWS IoT Device Tester
GGV2Q v2.5.4가 포함된
v4.9.4 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.9.4를 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.5.4가 포함되어 있으며 Greengrass nucleus 버전 2.12.0, 2.11.0, 2.10.0, 2.9.5를 지원합니다.

2024년 5월 3일

[보안 터널링 v1.0.19 릴리스](#)

보안 터널링 v1.0.19를 사용할 수 있습니다. 이 버전은 구성 요소가 간접적으로 호출하는 기본 AWS IoT Device Client를 버전 1.8.0에서 버전 1.9.0으로 업그레이드합니다. 보안 터널링 v1.0.19에서는 구성 요소 수준에서 동시 터널 제한이 20개로 증가합니다. 또한 이 새 버전은 AWS IoT Greengrass 코어 IPC 제한 시간을 3초에서 10초로 늘립니다.

2024년 5월 1일

[Kinesis Video Streams 구성 요소용 엣지 커넥터 v1.0.5 릴리스](#)

Kinesis Video Streams 구성 요소용 엣지 커넥터 버전 1.0.5를 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.

2024년 4월 29일

[Greengrass CLI v2.12.5 릴리스](#)

Greengrass CLI 구성 요소 v2.12.5를 사용할 수 있습니다.

2024년 4월 25일

[클라이언트 디바이스 인증 구성 요소 v2.5.0 릴리스](#)

클라이언트 디바이스 인증 구성 요소 v2.5.0을 사용할 수 있습니다. 이 릴리스에서는 사물 이름에 대한 정책 변수 지원이 추가됩니다. 또한 와일드카드가 포함된 정책 리소스가 허용됩니다.

2024년 4월 25일

[AWS IoT Greengrass 코어 v2.12.5 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.5와 업데이트 제공 구성 요소를 AWS제공합니다.

2024년 4월 25일

[AWS IoT Device Tester GGV2Q v2.5.3이 포함된 v4.9.3 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.9.3을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.5.3이 포함되어 있으며 Greengrass nucleus 버전 2.12.0, 2.11.0, 2.10.0, 2.9.5를 지원합니다.

2024년 4월 5일

[Greengrass CLI v2.12.4 릴리스](#)

Greengrass CLI 구성 요소 v2.12.4를 사용할 수 있습니다.

2024년 4월 2일

[AWS IoT Greengrass 코어 v2.12.4 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.4와 업데이트 제공 구성 요소를 AWS제공합니다.

2024년 4월 2일

[새도 관리자 v2.3.7 릴리스](#)

새도 관리자 v2.3.7을 사용할 수 있습니다. 이 릴리스에서는 새도 관리자가 새도 관리자 동기화 중에 주기적으로 NullPointerException 오류를 기록하는 문제를 해결합니다.

2024년 3월 27일

Moquette MQTT 3.1.1 브로커 v2.3.6 릴리스	Moquette MQTT 3.1.1 브로커 구성 요소 v2.3.6을 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2024년 3월 27일
로컬 디버그 콘솔 v2.4.2 릴리스	로컬 디버그 콘솔 구성 요소 v2.4.2를 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2024년 3월 27일
Lambda 관리자 v2.3.3 릴리스	Lambda 관리자 구성 요소 v2.3.3을 사용할 수 있습니다. 이 버전에는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2024년 3월 27일
IP 감지기 v2.1.9 릴리스	IP 감지기 구성 요소 v2.1.9를 사용할 수 있습니다. 이 릴리스에서는 디버그 로그 수준에서만 로그를 전송하도록 IP 획득 단계가 조정됩니다.	2024년 3월 27일
AWS IoT 플릿 프로비저닝 플러그인 v1.2.1 릴리스	AWS IoT 플릿 프로비저닝 플러그인 v1.2.1을 사용할 수 있습니다. 이 릴리스에서는 Greengrass nucleus 시작 중에 플릿 프로비저닝 플러그인이 오프라인 상태가 되는 문제를 해결합니다. 플릿 프로비저닝 플러그인이 이제 MQTT 연결 직접 호출을 무기한 재시도합니다.	2024년 3월 27일
AWS IoT Greengrass 코어 v2.12.3 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.3과 업데이트 제공 구성 요소를 AWS제공합니다.	2024년 3월 27일

Greengrass CLI v2.12.3 릴리스	Greengrass CLI 구성 요소 v2.12.3을 사용할 수 있습니다.	2024년 3월 25일
AWS IoT Device Tester GGV2Q v2.5.2가 포함된 v4.9.2 릴리스	IDT AWS IoT Greengrass V2를 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.5.2가 포함되어 있으며 Greengrass nucleus 버전 2.12.0, 2.11.0, 2.10.0, 2.9.5를 지원합니다.	2024년 3월 18일
AWS IoT Greengrass 코어 v2.12.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.2와 업데이트 제공 구성 요소를 AWS제공합니다.	2024년 2월 15일
새도 관리자 v2.3.6 릴리스	새도 관리자 v2.3.6을 사용할 수 있습니다. 이 릴리스에서는 디바이스가 오프라인 상태인 동안 AWS 클라우드 업데이트를 통해 삭제된 새도 속성이 다시 연결한 후에도 로컬 새도에 계속 존재하는 문제를 해결합니다.	2024년 2월 14일
Lambda 런처 v2.0.13 릴리스	Lambda 런처 구성 요소의 버전 2.0.13을 사용할 수 있습니다. 이 릴리스에서는 일반적인 버그 수정 및 개선 사항이 포함됩니다.	2024년 2월 14일
디스크 스폰러 v1.0.3 릴리스	디스크 스폰러 구성 요소 v1.0.3을 사용할 수 있습니다. 이 릴리스에서는 데이터베이스 연결을 재사용하여 성능을 개선합니다.	2024년 2월 14일

[Greengrass 개발 키트 CLI v1.6.2](#)

Greengrass 개발 키트 CLI 버전 1.6.2를 사용할 수 있습니다. 이 버전에서는 상대 경로로 인해 Windows gradlew.bat이 작동하지 않는 문제를 해결합니다. 추가 개선 사항도 포함되어 있습니다.

2024년 1월 16일

[새 CloudTrail 데이터 이벤트](#)

이제 AWS CloudTrail 데이터 이벤트를 로깅하여 구성 요소 가져오기 또는 배포 구성과 같은 리소스 작업에 대한 정보를 가져올 수 있습니다. 이러한 이벤트를 사용하여 Greengrass 디바이스의 작업에 관한 인사이트를 얻을 수 있습니다.

2023년 12월 20일

[스트림 관리자 v2.1.12 릴리스](#)

이제 스트림 관리자 v2.1.12를 사용할 수 있습니다. 이 릴리스는 Greengrass가 AWS 서비스 호출을 위한 자격 증명 세트를 선택하는 데 사용하는 순서를 변경합니다.

2023년 12월 8일

[MQTT 브리지 v2.3.1 릴리스](#)

MQTT 브리지 v2.3.1을 사용할 수 있습니다. 이 릴리스에서는 로컬 MQTT 클라이언트에서 드물게 연결 해제 루프가 발생하는 문제가 해결됩니다.

2023년 12월 8일

[디스크 스펠러 v1.0.2 릴리스](#)

디스크 스펠러 구성 요소 v1.0.2를 사용할 수 있습니다. 이 릴리스에서는 특정 사례에서 MQTT 메시지 형식이 지속되지 않은 문제를 해결합니다.

2023년 12월 8일

[클라이언트 디바이스 인증 구성 요소 v2.4.5 릴리스](#)

클라이언트 디바이스 인증 구성 요소 v2.4.5를 사용할 수 있습니다. 이 릴리스에서는 선택 규칙의 사물 이름 끝의 와일드 카드에 대한 지원이 추가되어 있으며, 경우에 따라 인증서가 새 연결 정보로 업데이트되지 않는 문제를 해결합니다.

2023년 12월 8일

[AWS IoT Greengrass 코어 v2.12.1 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.1과 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 12월 8일

[Greengrass 개발 키트 CLI v1.6.1](#)

Greengrass 개발 키트 CLI 버전 1.6.1을 사용할 수 있습니다. 이 버전에는 버그 수정과 개선 사항이 포함되어 있습니다.

2023년 12월 6일

[레시피 검증](#)

구성 요소 버전을 생성할 때 구성 요소 레시피를 검증하는 레시피 검증 기능이 추가되었습니다.

2023년 11월 16일

[게시자 지원 구성 요소](#)

AWS IoT Greengrass 는 이제 게시자 지원 구성 요소를 제공합니다. 이러한 구성 요소는 타사 공급업체에서 개발, 제공 및 서비스합니다.

2023년 11월 16일

[Greengrass Testing Framework v1.2.0 릴리스](#)

Greengrass Testing Framework v1.2.0을 사용할 수 있습니다.

2023년 11월 15일

[Greengrass 개발 키트 CLI v1.6.0](#)

Greengrass 개발 키트 CLI 버전 1.6.0을 사용할 수 있습니다. 이 버전에서는 component build 및 component publish 명령도 중 Greengrass 레시피 스키마에 대한 레시피 검증 검사가 추가됩니다. 이 업데이트를 통해 개발자는 구성 요소 생성 프로세스 초기에 구성 요소 레시피 내에서 조치를 취할 수 있는 문제를 식별할 수 있습니다. 또한 이 버전에는 test-e2e init 명령으로 가져올 수 있는 템플릿에 신뢰도 테스트 제품군이 추가됩니다. 이 신뢰도 테스트 제품군에는 기본 구성 요소 테스트 요구 사항에 맞춰 사용하고 확장할 수 있는 8가지 일반 테스트가 포함되어 있습니다.

2023년 11월 15일

[AWS IoT Device Tester v4.9.1 에서 Greengrass nucleus 버전 2.12.0 지원](#)

이제 IDT for AWS IoT Greengrass V2 버전 4.9.1에서 Greengrass nucleus 버전 2.12.0을 지원합니다.

2023년 11월 7일

[AWS IoT Greengrass 코어 v2.12.0 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.12.0과 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 11월 7일

[VPC에서 Greengrass 코어 디바이스 작동](#)

VPC에서 Greengrass 코어 디바이스를 작동할 수 있습니다. 이 기능을 사용하면 퍼블릭 인터넷에 액세스하지 않고 VPC에서 배포를 수행할 수 있습니다.

2023년 11월 3일

[Greengrass CLI v2.12.0 릴리스](#)

Greengrass CLI 구성 요소 v2.12.0을 사용할 수 있습니다.

2023년 10월 30일

[스트림 관리자 v2.1.10 릴리스](#)

이제 스트림 관리자 v2.1.10을 사용할 수 있습니다. 이 릴리스에서는 HTTPS 프록시 구성이 Greengrass CA 인증서 체인을 신뢰하지 않는 문제를 해결합니다.

2023년 10월 26일

[Lambda 런처 v2.0.12 릴리스](#)

Lambda 런처 구성 요소의 버전 2.0.12를 사용할 수 있습니다. 이 릴리스에서는 이전 프로세스가 제대로 중지되지 않은 경우 Lambda 런처에서 오류가 발생할 수 있는 문제를 해결합니다.

2023년 10월 26일

[Greengrass 개발 키트 CLI v1.5.0](#)

Greengrass 개발 키트 CLI 버전 1.5.0을 사용할 수 있습니다. 이 버전에서는 build_system 가 zip인 경우 excludes 빌드 옵션에서 인식하는 패턴이 업데이트됩니다. 이제 이 버전에서 와일드카드 문자를 기반으로 경로 이름과 일치하는 glob 패턴을 인식합니다. 이를 통해 제외할 디렉터리를 사용자 지정할 수 있습니다.

2023년 10월 26일

새도 관리자 v2.3.4 릴리스	새도 관리자 v2.3.4를 사용할 수 있습니다. 이 릴리스에서는 null 및 빈 새도 상태 문서에 대한 지원을 추가합니다.	2023년 10월 18일
로그 관리자 v2.3.6 릴리스	로그 관리자 구성 요소 v2.3.6을 사용할 수 있습니다.	2023년 10월 18일
로컬 디버그 콘솔 v2.4.0 릴리스	로컬 디버그 콘솔 구성 요소 v2.4.0을 사용할 수 있습니다.	2023년 10월 18일
Lambda 관리자 v2.3.1 릴리스	Lambda 관리자 구성 요소 v2.3.1을 사용할 수 있습니다.	2023년 10월 18일
Greengrass CLI v2.11.3 릴리스	Greengrass 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 2.11.3과 업데이트 제공 구성 요소를 AWS제공합니다.	2023년 10월 18일
보안 터널링 v1.0.17 릴리스	보안 터널링 v1.0.17을 사용할 수 있습니다.	2023년 10월 4일
Greengrass 개발 키트 CLI v1.4.0	Greengrass 개발 키트 CLI 버전 1.4.0을 사용할 수 있습니다. 이 버전에서는 기존 GDK 구성 파일 내의 필드를 수정하는 대화형 프롬프트를 시작하는 config 명령이 새로 추가됩니다. 또한 계속하기 전에 레시피 크기가 Greengrass 요구 사항(16,000바이트 이하) 내에 있는지 확인하기 위해 gdk component build 및 gdk component publish 명령이 수정됩니다.	2023년 10월 2일

Moquette MQTT 3.1.1 브로커 v2.3.5 릴리스	Moquette MQTT 3.1.1 브로커 구성 요소 v2.3.5를 사용할 수 있습니다. 이 버전에서는 Moquette이 버전 0.17로 업데이트됩니다.	2023년 9월 28일
MQTT 브리지 v2.3.0 릴리스	MQTT 브리지 v2.3.0을 사용할 수 있습니다. 이 릴리스에서는 AWS IoT Core 및 로컬 MQTT 소스 간의 브리징에 대한 MQTT 5 지원이 추가됩니다.	2023년 9월 28일
Lambda 관리자 v2.3.0 릴리스	Lambda 관리자 구성 요소 v2.3.0을 사용할 수 있습니다.	2023년 9월 15일
Lambda 런처 v2.0.11 릴리스	Lambda 런처 구성 요소의 버전 2.0.11을 사용할 수 있습니다. 이 버전은 Lambda 관리자 2.3.0을 지원합니다.	2023년 9월 15일
Moquette MQTT 3.1.1 브로커 v2.3.4 릴리스	Moquette MQTT 3.1.1 브로커 구성 요소 v2.3.4를 사용할 수 있습니다.	2023년 9월 1일
Greengrass Testing Framework	GTF는 종단 간 자동화를 지원하는 빌딩 블록의 컬렉션입니다. 이를 통해 AWS IoT Greengrass Version 2 내부 고객은 서비스 팀이 적격 소프트웨어 변경, 자동 수락 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 2.11.2와 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 8월 9일

[Greengrass 개발 키트 CLI
v1.3.0](#)

Greengrass 개발 키트 CLI 버전 1.3.0을 사용할 수 있습니다. 이 버전에서는 Open Test Framework를 사용하여 구성 요소의 종단 간 테스트를 지원하는 test-e2e 명령이 새로 추가됩니다.

2023년 7월 21일

[AWS IoT Greengrass 코어
v2.11.1 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.11.1과 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 7월 21일

[디스크 스폴러 v1.0.0 릴리스](#)

디스크 스폴러 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 2.11.0과 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 6월 28일

[AWS IoT Greengrass 코어
v2.10.3 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.3과 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 6월 21일

[AWS IoT Greengrass 코어
v2.10.2 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.2와 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 6월 5일

AWS IoT Greengrass 코어 v2.10.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.1과 업데이트 제공 구성 요소를 AWS제공합니다.	2023년 5월 11일
AWS IoT Greengrass 코어 v2.10.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.10.0과 업데이트 제공 구성 요소를 AWS제공합니다.	2023년 5월 9일
SageMaker AI Edge Manager 사용 중단	Amazon SageMaker AI Edge Manager 구성 요소는 2024년 4월 26일에 사용 중단됩니다.	2023년 4월 28일
AWS IoT Greengrass 코어 v2.9.6 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.6과 업데이트 제공 구성 요소를 AWS제공합니다.	2023년 4월 20일
로그 관리자 v2.3.2 릴리스	로그 관리자 구성 요소 v2.3.2 를 사용할 수 있습니다.	2023년 4월 19일
스트림 관리자 v2.1.4 릴리스	이제 스트림 관리자 v2.1.4를 사용할 수 있습니다. 이 릴리스에서는 단일 배치 내에서 동일한 타임스탬프가 있는 동일한 속성 자산에 대한 항목이 SiteWise API에서 ConflictingOperationException 을 반환하여 스트림 관리자가 계속 재시도하는 문제를 해결합니다. 또한 이 릴리스에서는 기본 연결 제한 시간을 3초에서 1분으로 업데이트합니다.	2023년 4월 13일

Greengrass 개발 키트 CLI v1.2.3	Greengrass 개발 키트 CLI 버전 1.2.3을 사용할 수 있습니다. 이 버전에는 버그 수정이 포함되어 있습니다.	2023년 4월 13일
클라이언트 디바이스 인증 구성 요소 v2.4.0 릴리스	클라이언트 디바이스 인증 구성 요소 v2.4.0을 사용할 수 있습니다. 이 릴리스에서는 클라이언트 디바이스 인증에서 Greengrass Client Device 대신 보드에 표시할 수 있는 운영 지표 내보내기에 대한 지원을 추가합니다.	2023년 4월 10일
Greengrass 개발 키트 CLI v1.2.2	Greengrass 개발 키트 CLI 버전 1.2.2를 사용할 수 있습니다. 이 버전에는 개선 사항과 버그 수정이 포함되어 있습니다.	2023년 4월 7일
AWS IoT Greengrass 코어 v2.9.5 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.5와 업데이트 제공 구성 요소를 AWS제공합니다.	2023년 3월 30일
스트림 관리자 v2.1.3 릴리스	이제 스트림 관리자 v2.1.3을 사용할 수 있습니다. 이 릴리스에서는 SYSTEM 사용자로 실행 시 Windows OS의 시작 문제가 해결됩니다.	2023년 3월 7일
Modbus-RTU 프로토콜 어댑터 v2.1.5 릴리스	Modbus-RTU 프로토콜 어댑터 구성 요소 v2.1.5를 사용할 수 있습니다. 이 릴리스에서는 ReadDiscreteInput 작업 문제를 해결합니다.	2023년 3월 7일

[클라이언트 디바이스 인증 구성 요소 v2.3.2 릴리스](#)

클라이언트 디바이스 인증 구성 요소 v2.3.2를 사용할 수 있습니다. 이 릴리스에서는 오프라인 상태에서 다시 시작할 때 구성 요소에서 인증서 주체가 올바르게 생성되도록 호스트 이름 정보 캐싱에 대한 지원을 추가합니다.

2023년 3월 7일

[AWS IoT Device Tester v4.7.0에서 Greengrass nucleus 버전 2.9.4 지원](#)

이제 IDT for AWS IoT Greengrass V2 버전 4.7.0에서 Greengrass nucleus 버전 2.9.4를 지원합니다.

2023년 3월 2일

[Greengrass 명령줄 인터페이스 v1.2.0 릴리스](#)

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 2.9.4와 업데이트 제공 구성 요소를 AWS제공합니다.

2023년 2월 24일

[새도 관리자 v2.3.1 릴리스](#)

새도 관리자 v2.3.1을 사용할 수 있습니다. 이 릴리스에서는 클라우드 새도 업데이트가 동기화되지 않을 수 있는 조건이 수정됩니다. 또한 이 릴리스에서는 이름이 지정된 새도 동기화 구성에 대한 변경 사항이 이름이 지정된 새도 하나에만 적용되는 문제를 해결합니다.

2023년 2월 21일

[AWS IoT Device Tester v4.7.0에서 Greengrass nucleus 버전 2.9.3 지원](#)

이제 IDT for AWS IoT Greengrass V2 버전 4.7.0에서 Greengrass nucleus 버전 2.9.3을 지원합니다.

2023년 2월 9일

IAM 모범 사례 업데이트	IAM 모범 사례에 따라 가이드가 업데이트되었습니다. 자세한 내용은 IAM의 보안 모범 사례 를 참조하세요.	2023년 2월 3일
AWS IoT Greengrass 코어 v2.9.3 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.3과 업데이트 제공 구성 요소를 AWS제공합니다.	2023년 2월 1일
로그 관리자 v2.3.1 릴리스	로그 관리자 v2.3.1을 사용할 수 있습니다.	2023년 1월 27일
AWS IoT Device Tester v4.7.0에서 Greengrass nucleus 버전 2.9.2 지원	이제 IDT for AWS IoT Greengrass V2 버전 4.7.0에서 Greengrass nucleus 버전 2.9.2를 지원합니다.	2023년 1월 3일
새도 관리자 v2.3.0 릴리스	새도 관리자 v2.3.0을 사용할 수 있습니다. 이 릴리스에서는 디바이스가 Greengrass 디바이스 프라이빗 키를 하드웨어 보안 모듈에 저장하는 경우 새도가 동기화되지 않을 수 있는 문제를 해결합니다.	2022년 12월 29일
AWS IoT 플릿 프로비저닝 플러그인 v1.2.0 릴리스	AWS IoT 플릿 프로비저닝 플러그인 v1.2.0을 사용할 수 있습니다. 이 릴리스에서는 구성 가능한 프라이빗 키 경로를 사용하는 인증서 서명 요청을 통한 디바이스 프로비저닝이 지원됩니다.	2022년 12월 22일
AWS IoT Greengrass 코어 v2.9.2 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.2와 업데이트 제공 구성 요소를 AWS제공합니다.	2022년 12월 22일

AWS IoT Device Tester GGV2Q v2.5.0이 포함된 v4.7.0 릴리스	IDT for AWS IoT Greengrass V2 버전 4.7.0을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.5.0이 포함되어 있으며 Greengrass nucleus 버전 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 및 2.6.0을 지원합니다.	2022년 12월 13일
새도 관리자 v2.2.4 릴리스	로컬 새도 문서를 업데이트할 때 새도 크기 검증이 클라우드와 일치하지 않는 문제를 해결합니다. 또한 배포가 구성 노드에서 RESET을 수행하는 경우 새도 관리자가 구성 업데이트 수신을 중지하는 문제를 해결합니다.	2022년 12월 8일
로그 관리자 v2.3.0 릴리스	로그 관리자 구성 요소 v2.3.0을 사용할 수 있습니다.	2022년 11월 18일
AWS IoT Device Tester v4.5.11에서 Greengrass nucleus 버전 2.9.1 지원	이제 IDT for AWS IoT Greengrass V2 버전 4.5.11에서 Greengrass nucleus 버전 2.9.1을 지원합니다.	2022년 11월 18일
AWS IoT Greengrass 코어 v2.9.1 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.1과 업데이트 제공 구성 요소를 AWS제공합니다.	2022년 11월 18일
AWS IoT Device Tester v4.5.11에서 Greengrass nucleus 버전 2.9.0 지원	이제 IDT for AWS IoT Greengrass V2 버전 4.5.11에서 Greengrass nucleus 버전 2.9.0을 지원합니다.	2022년 11월 17일

<u>스트림 관리자 v2.1.2 릴리스</u>	이제 스트림 관리자 v2.1.2를 사용할 수 있습니다. 이 릴리스에서는 영어 이외의 언어를 사용하는 Windows OS 문제를 해결합니다.	2022년 11월 15일
<u>AWS IoT Greengrass 코어 v2.9.0 소프트웨어 업데이트</u>	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.9.0과 업데이트 제공 구성 요소를 AWS제공합니다.	2022년 11월 15일
<u>AWS IoT Device Tester v4.5.11에서 Greengrass nucleus 버전 2.8.1 지원</u>	이제 IDT for AWS IoT Greengrass V2 버전 4.5.11에서 Greengrass nucleus 버전 2.8.1을 지원합니다.	2022년 10월 19일
<u>AWS IoT Device Tester GGV2Q v2.4.1이 포함된 v4.5.11 릴리스</u>	버전 4.5.11의 IDT for AWS IoT Greengrass V2를 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.4.1이 포함되어 있으며 Greengrass nucleus 버전 2.8.0, 2.7.0 및 2.6.0을 지원합니다.	2022년 10월 13일
<u>AWS IoT Greengrass 코어 v2.8.1 소프트웨어 업데이트</u>	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.8.1과 업데이트 제공 구성 요소를 AWS제공합니다.	2022년 10월 13일
<u>AWS IoT Greengrass 코어 v2.8.0 소프트웨어 업데이트</u>	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.8.0과 업데이트 제공 구성 요소를 AWS제공합니다.	2022년 10월 7일
<u>배포에 대한 CloudFormation 지원 추가</u>	CloudFormation 는 이제 AWS IoT Greengrass 배포를 리소스로 지원합니다.	2022년 10월 6일

[SageMaker AI Edge Manager
v1.3.0 릴리스](#)

Amazon SageMaker AI Edge Manager 구성 요소 v1.3.0을 사용할 수 있습니다. 이 릴리스에서는 해당 구성 요소에 대한 지원을 추가하여 TensorRT 모델 캐시의 디스크 크기를 설정하고 예측 동시성을 개선하여 GPU와 같은 디바이스 액셀러레이터 엔진을 더 잘 활용할 수 있도록 했습니다.

2022년 9월 1일

[IPC\(프로세스 간 통신\) 클라이언트 V2 사용](#)

IPC 클라이언트 V2에 대한 정보가 추가되어 IPC 작업을 사용하기 위해 작성해야 하는 코드의 양이 감소하고 IPC 클라이언트 V1에서 발생할 수 있는 일반적인 오류를 방지하는 데 도움이 됩니다.

2022년 8월 12일

[AWS IoT Device Tester
GGV2Q v2.4.0이 포함된
v4.5.8 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.5.8을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.4.0이 포함되어 있으며 Greengrass nucleus 버전 2.7.0, 2.6.0 및 2.5.6을 지원합니다.

2022년 8월 12일

[SageMaker AI Edge Manager v1.2.0 릴리스](#)

Amazon SageMaker AI Edge Manager 구성 요소 v1.2.0을 사용할 수 있습니다. 이 릴리스에서는 Amazon S3에 업로드한 SageMaker AI Neo 컴파일 모델을 자동으로 검색하도록 구성 요소에 대한 지원을 추가하므로 배포를 생성할 필요 없이 새 모델을 AWS IoT Greengrass 배포할 수 있습니다.

2022년 8월 3일

[AWS IoT Device Tester v4.5.3에서 Greengrass nucleus 버전 2.7.0 지원](#)

이제 IDT for AWS IoT Greengrass V2 버전 4.5.3에서 Greengrass nucleus 버전 2.7.0을 지원합니다.

2022년 8월 1일

[스트림 관리자 v2.1.0 릴리스](#)

이제 스트림 관리자 v2.1.0을 사용할 수 있습니다. 이 릴리스에서는 Amazon EventBridge로 원격 측정 지표를 보낼 수 있는 지원이 포함됩니다.

2022년 7월 28일

[AWS IoT Greengrass 코어 v2.7.0 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.7.0과 업데이트 제공 구성 요소를 AWS제공합니다. Amazon EventBridge로 원격 측정 지표를 보낼 수 있는 지원이 포함됩니다.

2022년 7월 28일

[IoT SiteWise 게시자 v2.2.0 릴리스](#)

IoT SiteWise 게시자 구성 요소 v2.2.0을 사용할 수 있습니다. 이 릴리스에서는 AWS IoT SiteWise 서비스로 전송하기 전에 데이터를 압축하도록 구성 요소를 업데이트하므로 대역폭 사용량이 최대 75% 감소합니다.

2022년 7월 19일

[자습서: 클라이언트 디바이스 새도와 상호 작용하는 구성 요소 개발](#)

[자습서: MQTT를 통해 로컬 IoT 디바이스와 상호 작용](#)에 새 모듈이 추가되어 이를 따라 클라이언트 디바이스 새도와 상호 작용하는 구성 요소를 개발하는 방법을 학습할 수 있습니다.

2022년 7월 18일

[로컬 MQTT 브로커 선택](#)

클라이언트 디바이스가 코어 디바이스에 연결되는 로컬 MQTT 브로커 선택 방법에 관한 정보가 추가되었습니다.

2022년 7월 18일

[AWS IoT Device Tester v4.5.3에서 Greengrass nucleus 버전 2.6.0 지원](#)

이제 IDT for AWS IoT Greengrass V2 버전 4.5.3에서 Greengrass nucleus 버전 2.6.0을 지원합니다.

2022년 6월 29일

[AWS IoT Greengrass 코어 v2.6.0 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.6.0과 업데이트 제공 구성 요소를 AWS제공합니다. 클라이언트 디바이스 새도 및 클라이언트 디바이스용 로컬 MQTT 5 브로커에 대한 지원이 포함됩니다. 또한 로컬 게시/구독 주제의 와일드카드, 구성 요소 구성의 레시피 변수, IPC 권한 부여 정책의 와일드카드에 대한 지원도 포함됩니다. 이러한 기능을 사용하면 코어 디바이스 플릿에 배포하는 구성 요소를 보다 쉽게 개발 및 구성할 수 있습니다. 이 릴리스에서는 코어 디바이스에서 로컬 배포 및 구성 요소를 관리하는 IPC 작업을 사용하는 구성 요소에 대한 지원도 포함되어 있습니다.

2022년 6월 27일

[클라이언트 디바이스 구성 요소 업데이트](#)

[클라이언트 디바이스 인증 v2.1.0](#), [MQTT 브로커\(Moquette\) v2.1.0](#), [MQTT 브리지 v2.1.1](#) 및 [IP 감지기 v2.1.2](#)를 사용할 수 있습니다. 이 릴리스는 인증서 교체를 개선하고, MQTT 브로커 성능을 높이고, 이러한 구성 요소가 구성 재설정 업데이트를 처리하는 방식에 관한 문제를 해결합니다.

2022년 6월 14일

[AWS IoT Device Tester v4.5.3에서 Greengrass nucleus 버전 2.5.6 지원](#)

이제 IDT for AWS IoT Greengrass V2 버전 4.5.3에서 Greengrass nucleus 버전 2.5.6을 지원합니다.

2022년 6월 1일

[AWS IoT Greengrass 코어 v2.5.6 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.6과 업데이트 제공 구성 요소를 AWS제공합니다. ECC 키가 있는 하드웨어 보안 모듈에 대한 지원이 포함됩니다. 버그 수정 및 개선 사항도 포함되어 있습니다.

2022년 5월 31일

[AWS IoT 플릿 프로비저닝 플러그인 v1.1.0 릴리스](#)

AWS IoT 플릿 프로비저닝 플러그인 v1.1.0을 사용할 수 있습니다. 이 릴리스에서는 Windows 디바이스에서 플러그인을 구성할 때 추가 파일 경로 형식에 대한 지원을 추가합니다.

2022년 5월 12일

[새 Lambda 런타임 릴리스](#)

Python 3.9, Java 11 및 NodeJS 14와 같은 새로운 Lambda 런타임 지원이 추가되었습니다.

2022년 5월 10일

[구성 요소 업데이트를 지연시키는 Greengrass 구성 요소 개발](#)

배포에서 구성 요소 업데이트를 지연시키는 Greengrass 구성 요소를 개발하는 방법을 학습할 수 있는 자습서가 추가되었습니다. 예를 들어 디바이스의 배터리 잔량이 낮거나 중단될 수 없는 프로세스를 실행하는 동안 업데이트를 지연시킬 수 있습니다.

2022년 5월 4일

[CloudWatch 지표 v3.1.0 및 AWS IoT Device Defender v3.1.0 릴리스](#)

CloudWatch 지표 구성 요소 v3.1.0 및 AWS IoT Device Defender 구성 요소 v3.1.0을 사용할 수 있습니다. 이 릴리스에서는 HTTPS 네트워크 프록시 구성에 대한 지원이 추가됩니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스에서 HTTPS 프록시 신뢰 활성화](#)를 참조하세요.

2022년 4월 27일

[에서 마이그레이션 AWS IoT Greengrass Version 1](#)

에서 로 마이그레이션하기 위해 따를 수 있는 가이드 AWS IoT Greengrass V1 가 추가되었습니다 AWS IoT Greengrass V2.

2022년 4월 26일

[AWS IoT Device Tester GGV2Q v2.3.1이 업데이트된 v4.5.3 및 GGV2Q v2.3.0이 지원되는 버전에 추가된 IDT v4.5.1](#)

Greengrass nucleus 버전 AWS IoT Greengrass V2.5.5, 2.5.4 및 2.5.3에 대한 지원을 포함하도록 V2 검증 제품군 (GGV2Q) v2.3.1이 포함된 IDT AWS IoT Greengrass forV2 버전 4.5.3이 업데이트되었습니다. 이 업데이트에는 지원되는 버전으로 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.3.0이 포함된 IDT 4.5.1도 포함되어 있습니다. AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.3.0이 포함된 IDT 4.5.1은 Greengrass nucleus 버전 2.5.3을 지원합니다.

2022년 4월 25일

[Modbus-RTU 프로토콜 어댑터 v2.1.0 릴리스](#)

Modbus-RTU 프로토콜 어댑터 구성 요소 v2.1.0을 사용할 수 있습니다. 이 릴리스에서는 Modbus RTU 디바이스와의 직렬 통신을 구성하도록 지정할 수 있는 새 파라미터가 추가됩니다.

2022년 4월 20일

[CloudWatch 지표 v2.1.0, Firehose v2.1.0 및 Amazon SNS v2.1.0 릴리스](#)

CloudWatch 지표 구성 요소 v2.1.0, Firehose 구성 요소 v2.1.0 및 Amazon SNS 구성 요소 v2.1.0을 사용할 수 있습니다. 이 릴리스에서는 HTTPS 네트워크 프록시 구성에 대한 지원이 추가됩니다. 자세한 내용은 [포트 443에서 또는 네트워크 프록시를 통해 연결 및 코어 디바이스에서 HTTPS 프록시 신뢰 활성화](#)를 참조하세요.

2022년 4월 19일

[AWS IoT Device Tester GGV2Q v2.3.1이 포함된 v4.5.3 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.5.3을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.3.1이 포함되어 있으며 Greengrass nucleus 버전 2.5.5를 지원합니다.

2022년 4월 15일

[AWS IoT Greengrass 코어
v2.5.5 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.5와 업데이트 제공 구성 요소를 AWS제공합니다. 영어 이외의 표시 언어를 사용하는 Windows 디바이스에 대한 지원을 추가합니다. 또한 특정 시나리오에서 프로비저닝한 후 코어 디바이스가 AWS IoT Greengrass 클라우드 서비스에 상태를 보고하지 않는 문제를 해결합니다.

2022년 4월 6일

[AWS IoT Greengrass 코어
v2.5.4 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.4와 업데이트 제공 구성 요소를 AWS제공합니다. 버그 수정 및 개선 사항이 포함되어 있습니다.

2022년 3월 23일

[AWS IoT Device Tester 프로그
래밍 방식으로 다운로드](#)

AWS IoT Greengrass V2 프로그래밍 방식으로 용 IDT를 다운로드하는 방법에 대한 정보가 추가되었습니다.

2022년 3월 15일

[Greengrass 개발 키트 CLI
v1.1.0](#)

Greengrass 개발 키트 CLI 버전 1.1.0을 사용할 수 있습니다. 이 버전에서는 component init 및 component publish 명령에 새 인수가 추가됩니다. 또한 빌드되지 않은 경우 구성 요소를 빌드하도록 component publish 명령이 업데이트됩니다.

2022년 2월 24일

새도 관리자 v2.1.0 릴리스	새도 관리자 구성 요소 v2.1.0을 사용할 수 있습니다. 이 릴리스에서는 구성 요소가 새도우를 동기화하는 간격을 구성하는 옵션을 추가합니다 AWS IoT Core. 예를 들어 대역폭 사용량과 요금을 줄이기 위해 더 긴 간격을 지정할 수 있습니다.	2022년 2월 3일
AWS IoT Greengrass 코어 소프트웨어 v2.5.3용 Dockerfile 및 Docker 이미지	이제 AWS IoT Greengrass 코어 소프트웨어 v2.5.3용 Dockerfile 및 Docker 이미지를 사용할 수 있습니다.	2022년 1월 12일
AWS IoT Device Tester GGV2Q v2.3.0이 포함된 v4.5.1 릴리스	IDT for AWS IoT Greengrass V2 버전 4.5.1을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.3.0이 포함되어 있으며, 하드웨어 보안 모듈(HSM)을 사용하여 AWS IoT Greengrass 코어 소프트웨어에서 사용하는 프라이빗 키와 인증서를 저장하는 Linux 기반 디바이스의 검증 및 검증을 지원합니다.	2022년 1월 11일
AWS IoT Greengrass 코어 v2.5.3 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.3과 업데이트 제공 구성 요소를 AWS제공합니다. 여기에는 하드웨어 보안 모듈(HSM)에 안전하게 저장하는 프라이빗 키와 인증서를 사용하도록 AWS IoT Greengrass 코어 소프트웨어를 구성할 수 있는 지원이 포함됩니다.	2022년 1월 6일

[AWS IoT Greengrass 코어 소프트웨어 v2.5.2용 Dockerfile 및 Docker 이미지](#)

이제 AWS IoT Greengrass 코어 소프트웨어 v2.5.2용 Dockerfile 및 Docker 이미지를 사용할 수 있습니다.

2021년 12월 20일

[AWS IoT Device Tester GGV2Q v2.2.1이 포함된 v4.4.1 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.4.1을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.2.1이 포함되어 있으며 디바이스 검증을 위해 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 관한 정보가 추가되었습니다. Lookout for Vision은 컴퓨터 비전을 사용하여 산업용 제품의 시각적 결함을 찾아냅니다.

2021년 12월 8일

[AWS IoT Device Tester GGV2Q v2.2.0이 포함된 v4.4.1 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.4.1을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.2.0이 포함되어 있으며 디바이스 검증을 위해 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 2.5.2와 업데이트 제공 구성 요소를 AWS제공합니다. Greengrass nucleus 업데이트 후 발생하는 Windows 서비스의 문제를 해결합니다. 또한 Windows 디바이스의 AWS IoT Device Defender 구성 요소에 대한 지원도 포함됩니다.

2021년 12월 3일

[Kinesis Video Streams 구성 요 소용 새 엣지 커넥터](#)

Kinesis Video Streams 구성 요소용 엣지 커넥터 버전 1.0.0을 사용할 수 있습니다. 이 AWS 제공 커넥터는 로컬 카메라의 비디오 피드를 읽고 스트림을 Kinesis Video Streams에 게시합니다. 이 구성 요소는 AWS IoT TwinMaker와 통합되어 Grafana 대시보드에서 비디오 스트림 및 기타 데이터를 보고 관리할 수 있습니다.

2021년 11월 30일

[를 사용하여 Greengrass 코어 디바이스 관리 AWS Systems Manager](#)

를 사용하여 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

[Greengrass 개발 키트 CLI](#)

로컬 AWS IoT Greengrass 개발 컴퓨터에 다운로드하여 사용자 지정 Greengrass 구성 요소를 개발하는 데 도움이 되는 도구인 개발 키트 명령줄 인터페이스(GDK CLI)에 대한 정보가 추가되었습니다. GDK CLI를 사용하여 사용자 지정 구성 요소를 생성, 빌드 및 게시할 수 있습니다.

2021년 11월 29일

[커뮤니티 제공 Greengrass 구성 요소](#)

Greengrass 커뮤니티에서 개발한 Greengrass 구성 요소의 인덱스인 Greengrass Software Catalog에 관한 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 2.5.1과 업데이트 제공 구성 요소를 AWS제공합니다. Windows 디바이스에서 32비트 Java에 대한 지원이 포함됩니다. 또한 Windows 디바이스에서 새로운 사물 그룹 제거 동작 및 시스템 환경 변수 로드 관련 문제가 해결됩니다.

2021년 11월 23일

[AWS IoT Device Tester
GGV2Q v2.1.0이 포함된
v4.4.0 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.4.0을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.1.0이 포함되어 있으며 Greengrass nucleus 버전 2.5.0을 실행하는 Windows 기반 Greengrass 디바이스의 검증을 지원합니다.

2021년 11월 19일

[AWS IoT Greengrass 코어
v2.5.0 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.5.0과 업데이트 제공 구성 요소를 AWS제공합니다. 여기에는 Windows 디바이스에서 AWS IoT Greengrass 코어 소프트웨어를 실행할 수 있는 지원이 포함됩니다. 또한 사물 그룹 제거 동작이 변경되고 HTTPS 프록시에 대한 지원을 추가합니다.

2021년 11월 12일

[SageMaker AI Edge Manager
v1.1.0 릴리스](#)

Amazon SageMaker AI Edge Manager 구성 요소 v1.1.0을 사용할 수 있습니다. 이 릴리스에서는 Amazon Linux 2를 실행하는 Greengrass 코어 디바이스에 대한 지원을 추가하고 새 구성 파라미터를 추가하여 디바이스의 캡처 데이터 폴더 위치를 지정합니다.

2021년 11월 3일

[교차 서비스 혼동된 대리자 예방 업데이트](#)

AWS IoT Greengrass V2는 IAM 리소스 정책에서 [aws:SourceArn](#) 및 [aws:SourceAccount](#) 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지할 수 있도록 지원합니다.

2021년 11월 1일

[클라이언트 디바이스 구성 요소 업데이트](#)

[클라이언트 디바이스 인증](#) v2.0.3, [IP 감지기](#) v2.1.0, [MQTT 브리지](#) v2.1.0 및 [MQTT 브로커\(Moquette\)](#) v2.0.2를 사용할 수 있습니다. 이 릴리스에서는 기본이 아닌 MQTT 브로커 포트에 대한 전체 지원이 추가되고 기타 버그 수정 및 개선 사항이 포함됩니다.

2021년 10월 28일

[새도 관리자 v2.0.4 릴리스](#)

새도 관리자 구성 요소 v2.0.4를 사용할 수 있습니다. 이 릴리스에서는 새도 관리자가 이전에 삭제된 새도의 새로 생성된 버전을 삭제하는 문제를 해결합니다. 이 릴리스부터 DeleteThingShadow IPC 작업 시 새도 버전이 증가합니다.

2021년 10월 20일

[로그 관리자 v2.2.0 릴리스](#)

로그 관리자 구성 요소 v2.2.0을 사용할 수 있습니다. 이제 로그 관리자에서 구성 맵을 사용하여 구성 요소 로그 구성을 제공할 수 있습니다.

2021년 10월 20일

Lambda 관리자 v2.1.4 릴리스	Lambda 관리자 구성 요소 v2.1.4를 사용할 수 있습니다. 이 릴리스에서는 NodeJS 런타임을 사용하는 Lambda 함수가 하나의 메시지만 처리하게 되는 문제를 해결합니다.	2021년 10월 20일
Docker 컨테이너 구성 요소에서 프로세스 간 통신, AWS 자격 증명 및 스트림 관리자 사용	사용자 지정 Docker 컨테이너 구성 요소에서 IPC(프로세스 간 통신), AWS 자격 증명 및 스트림 관리자를 사용하는 방법에 관한 정보가 추가되었습니다.	2021년 10월 19일
새로운 nucleus 원격 측정 이미터 구성 요소	nucleus 원격 측정 이미터 구성 요소의 버전 1.0.0을 사용할 수 있습니다. 이 AWS제공 구성 요소는 시스템 상태 원격 측정 데이터를 수집하여 로컬 주제 및 AWS IoT Core MQTT 주제에 지속적으로 게시합니다.	2021년 9월 30일
프록시 또는 방화벽을 통해 디바이스 트래픽 허용	Greengrass 코어 디바이스가 사용하는 엔드포인트 및 포트에 관한 정보가 추가되어 보안 조치로써 트래픽을 제한할 수 있습니다.	2021년 9월 16일
AWS IoT Device Tester GGV2Q v2.0.1이 포함된 v4.2.0 릴리스	IDT AWS IoT Greengrass forV2 버전 4.2.0이 AWS IoT Greengrass V2 검증 제품군 (GGV2Q) v2.0.1로 업데이트되었습니다. 이 릴리스에서는 Greengrass nucleus 버전 2.4.0의 디바이스 검증이 지원됩니다.	2021년 8월 31일

[기계 학습 설치 관리자 구성 요소 업데이트](#)

DLR 설치 관리자 구성 요소 v1.6.5 및 TensorFlow Lite 설치 관리자 구성 요소 v2.5.4를 사용할 수 있습니다. 이러한 구성 요소 버전에는 기본 설치 스크립트를 비활성화할 수 있는 새로운 UseInstaller 구성 파라미터가 포함되어 있습니다.

2021년 8월 30일

[에 대한 임베디드 Linux 지원 AWS IoT Greengrass](#)

에 대한 BitBake 레시피 AWS IoT Greengrass V2 는 GitHub 의 meta-aws 프로젝트에서 사용할 수 있습니다. 이 레시피를 사용하여 Yocto 프로젝트를 사용하는 사용자 지정 Linux 기반 운영 체제를 빌드할 수 있습니다.

2021년 8월 20일

[코드 무결성](#)

가 Greengrass 코어 디바이스에서 다운로드하는 소프트웨어의 무결성을 AWS IoT Greengrass V2 확인하는 방법에 대한 정보가 추가되었습니다 AWS 클라우드.

2021년 8월 19일

[VPC 엔드포인트\(AWS PrivateLink\)](#)

AWS IoT Greengrass 는 이제 AWS IoT Greengrass 컨트롤 플레인에 대한 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 지원합니다. VPC와 AWS IoT Greengrass 컨트롤 플레인 간에 프라이빗 연결을 설정할 수 있습니다.

2021년 8월 16일

스트림 관리자 v2.0.12 릴리스	이제 스트림 관리자 v2.0.12를 사용할 수 있습니다. 이 릴리스에서는 스트림 관리자 구성 요소가 v2.0.7에서 v2.0.8과 v2.0.11 사이의 버전으로 업그레이드되지 않는 문제를 해결합니다.	2021년 8월 10일
AWS IoT Greengrass 코어 소프트웨어 v2.4.0용 Dockerfile 및 Docker 이미지	이제 AWS IoT Greengrass 코어 소프트웨어 v2.4.0용 Dockerfile 및 Docker 이미지를 사용할 수 있습니다.	2021년 8월 9일
AWS IoT Greengrass 코어 v2.4.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.4.0과 업데이트 제공 구성 요소를 AWS제공합니다. 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 SiteWise OPC-UA 수집기, AWS IoT SiteWise IoT SiteWise 게시자 및 IoT SiteWise IoT SiteWise 프로세서에 대해 다음 AWS제공 구성 요소가 추가되었습니다. IoT SiteWise	2021년 7월 29일

[AWS IoT Device Tester
GGV2Q v2.0.0이 포함된
v4.2.0 릴리스](#)

IDT AWS IoT Greengrass V2.0을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v2.0.0이 포함되어 있으며 Docker 구성 요소, 기계 학습 및 스트림 관리자에 대한 선택적 검증 테스트 지원이 포함되어 있습니다.

[AWS IoT Greengrass C++ v2
AWS IoT Device SDK 용에서
사용 가능한 코어 IPC 라이브러리](#)

C++ v2 AWS IoT Device SDK 용의 버전 1.13.0은 AWS IoT Greengrass 코어 IPC를 지원하므로 AWS IoT Greengrass 코어 소프트웨어와 상호 작용하는 C++ 구성 요소를 개발할 수 있습니다.

[SageMaker AI Edge Manager
구성 요소 v1.0.2 릴리스](#)

Amazon SageMaker AI Edge Manager 구성 요소 v1.0.2를 사용할 수 있습니다. 이 릴리스에서는 구성 요소 수명 주기의 설치 스크립트가 업데이트됩니다. 이 구성 요소를 배포하기 전에 코어 디바이스에 Python 버전에 대한 pip를 포함하여 Python 3.6 이상이 설치되어 있어야 합니다.

[AWS IoT Device Tester for
AWS IoT Greengrass V2에 대한
업데이트 지원](#)

IDT for AWS IoT Greengrass V2 버전 4.1.0은 이제 디바이스 검증을 위해 Greengrass nucleus 버전 2.3.0 사용을 지원합니다.

AWS IoT Greengrass 코어 소프트웨어 v2.3.0용 Dockerfile 및 Docker 이미지	이제 AWS IoT Greengrass 코어 소프트웨어 v2.3.0용 Dockerfile 및 Docker 이미지를 사용할 수 있습니다.	2021년 7월 7일
AWS 관리형 정책	에 대한 AWS 관리형 정책에 대한 정보가 추가되었습니다 AWS IoT Greengrass.	2021년 7월 2일
새로운 권장 JVM 옵션	AWS IoT Greengrass 코어 소프트웨어의 메모리 할당을 제어하는 권장 JVM 옵션에 대한 정보가 추가되었습니다.	2021년 6월 30일
AWS IoT Greengrass 코어 v2.3.0 소프트웨어 업데이트	이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.3.0과 업데이트 제공 구성 요소를 AWS제공합니다. 배포의 대규모 구성 요소 구성 문서에 대한 지원이 포함됩니다.	2021년 6월 29일
AWS IoT Greengrass 코어 소프트웨어 v2.2.0용 Dockerfile 및 Docker 이미지	이제 AWS IoT Greengrass 코어 소프트웨어 v2.2.0용 Dockerfile 및 Docker 이미지를 사용할 수 있습니다.	2021년 6월 28일
AWS IoT Device Tester GGV2Q v1.1.1이 포함된 v4.1.0 릴리스	IDT for AWS IoT Greengrass V2 버전 4.1.0을 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v1.1.1이 포함되어 있으며 디바이스 검증을 위해 Greengrass nucleus v2.2.0, v2.1.0 및 v2.0.5 사용을 지원합니다.	2021년 6월 18일

[AWS IoT Greengrass 코어
v2.2.0 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.2.0과 업데이트 제공 구성 요소를 AWS제공합니다. 여기에는 클라이언트 디바이스에 대한 지원을 추가하고 로컬 새도 서비스를 추가하기 위해 배포할 수 있는 구성 요소가 포함되어 있습니다.

2021년 6월 18일

[Lambda 런처 v2.0.6 릴리스](#)

Lambda 런처 구성 요소의 버전 2.0.6을 사용할 수 있습니다. 이 버전에는 개선 사항과 버그 수정이 포함됩니다.

2021년 6월 13일

[새 SageMaker AI Edge
Manager 구성 요소 릴리스](#)

Amazon SageMaker AI Edge Manager 구성 요소의 버전 1.0.0을 AWS IoT Greengrass에서 사용할 수 있습니다. 이 구성 요소는 Greengrass 코어 디바이스에 SageMaker AI Edge Manager 에이전트 바이너리를 설치합니다.

2021년 6월 10일

[구성 요소 유형](#)

AWS IoT Greengrass의 구성 요소 유형에 관한 정보가 추가되었습니다. 구성 요소 유형으로 AWS IoT Greengrass 코어 소프트웨어가 구성 요소를 실행하는 방법이 지정됩니다.

2021년 6월 3일

[AWS IoT Device Tester
GGV2Q v1.1.0이 포함된
v4.0.2 릴리스](#)

IDT AWS IoT Greengrass V2를 사용할 수 있습니다. 이 릴리스에는 AWS IoT Greengrass V2 검증 제품군(GGV2Q) v1.1.0이 포함되어 있으며 디바이스 검증을 위해 Greengrass CLI v2.1.0과 함께 Greengrass nucleus v2.1.0 사용을 지원합니다. 여기에는 MQTT 및 Lambda에 필요한 새로운 테스트 그룹과 기타 사소한 버그 수정 및 개선 사항도 포함됩니다.

2021년 5월 5일

[AWS IoT Greengrass 코어 소프트웨어 v2.1.0용 Dockerfile
및 Docker 이미지](#)

이제 AWS IoT Greengrass 코어 소프트웨어 v2.1.0용 Dockerfile 및 Docker 이미지를 사용할 수 있습니다. Docker 이미지를 사용하면 Amazon Linux 2를 기본 운영 체제로 사용하는 Docker 컨테이너에서 AWS IoT Greengrass 코어 소프트웨어를 실행할 수 있습니다.

2021년 4월 27일

[AWS IoT Greengrass 코어 v2.1.0 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.1.0과 업데이트 제공 구성 요소를 AWS제공합니다. 여기에는 프라이빗 Amazon ECR 리포지토리에서 Docker 이미지를 다운로드하는 데 사용할 수 있는 새 구성 요소와 TensorFlow Lite를 사용하여 기계 학습 추론을 수행하는 새 샘플 구성 요소가 포함되어 있습니다.

2021년 4월 26일

[Secrets Manager를 사용하는 예제 구성 요소](#)

코어 디바이스에 배포하는 AWS Secrets Manager 보안 암호의 값을 인쇄하는 예제 구성 요소가 추가되었습니다.

2021년 4월 8일

[Greengrass 코어 디바이스에 대한 최소 AWS IoT 정책](#)

코어 디바이스에서 기본 Greengrass 기능을 지원하는 데 필요한 최소 권한 세트에 관한 정보가 추가되었습니다.

2021년 4월 2일

[IPC 이벤트 스트림 구독](#)

IPC(프로세스 간 통신) 작업을 사용하여 Greengrass 코어 디바이스에서 이벤트 스트림을 구독하는 방법에 관한 정보가 추가되었습니다.

2021년 4월 1일

[용 AWS IoT Device Tester 에 대한 업데이트 지원 AWS IoT Greengrass](#)

IDT for AWS IoT Greengrass V2 버전 4.0.1은 이제 디바이스 검증을 위해 Greengrass CLI 버전 2.0.5와 함께 Greengrass nucleus 버전 2.0.5 사용을 지원합니다.

2021년 3월 17일

[스트림 관리자를 사용하는 사용자 지정 구성 요소 생성](#)

데이터 스트림을 관리하는 애플리케이션을 개발하도록 구성 요소 레시피 및 아티팩트를 구성하는 방법에 관한 정보가 추가되었습니다.

2021년 3월 9일

[AWS IoT Greengrass 코어 v2.0.5 소프트웨어 업데이트](#)

이 릴리스는 Greengrass nucleus 구성 요소의 버전 2.0.5와 업데이트 제공 구성 요소를 AWS에서 제공합니다. AWS 중국 리전에서 네트워크 프로시저 지원 관련 문제와 Greengrass 데이터 영역 엔드포인트 관련 문제를 해결합니다.

2021년 3월 9일

[구성 요소 환경 변수 참조](#)

AWS IoT Greengrass 코어 소프트웨어가 구성 요소에 대해 설정하는 환경 변수에 대한 정보가 추가되었습니다. 이러한 환경 변수를 사용하여 사물 이 름 AWS 리전 및 Greengrass nucleus 버전을 가져올 수 있습니다.

2021년 2월 23일

[수동 설치](#)

필요한 AWS 리소스를 수동으 로 생성하거나 방화벽 또는 네트워크 프록시 뒤에 설치하는 방법에 대한 정보가 추가되었습니다. 수동 설치를 사용하면 필요한 AWS IoT 및 IAM 리소스를 생성 AWS 계정으로 설 치 프로그램에서 리소스를 생성할 수 있는 권한을 부여할 필요가 없습니다. 포트 443 또는 네트워크 프록시를 통해 연결하도록 디바이스를 구성할 수도 있습니다.

2021년 2월 17일

[AWS IoT Greengrass Python v2 AWS IoT Device SDK 용의 코어 IPC 라이브러리 업데이트](#)

Python v2 AWS IoT Device SDK 용 버전 1.5.4는 AWS IoT Greengrass 코어 IPC 서비스에 연결하는 데 필요한 단계를 간소화합니다.

2021년 2월 11일

[용 AWS IoT Device Tester 에 대한 업데이트 지원 AWS IoT Greengrass](#)

IDT for AWS IoT Greengrass V2 버전 4.0.1은 이제 디바이스 검증을 위해 Greengrass CLI 버전 2.0.4와 함께 Greengrass nucleus 버전 2.0.4 사용을 지원합니다.

2021년 2월 5일

[Lambda 함수 가져오기 새 자습서](#)

Lambda 함수를 Greengrass 코어 디바이스에서 실행되는 구성 요소로 가져오는 새 콘솔 기반 자습서를 추가했습니다.

2021년 2월 5일

[AWS IoT Greengrass 코어 v2.0.4 소프트웨어 업데이트](#)

이 릴리스에서는 Greengrass nucleus 구성 요소의 버전 2.0.4가 제공됩니다. 포트 443을 통해 HTTPS 통신을 구성하는 새로운 greengrassDataPlanePort 파라미터와 버그 수정이 포함되어 있습니다. 이제 최소 IAM 정책에는 AWS IoT Greengrass 코어 소프트웨어 설치 관리자가에서 실행될 sts:GetCallerIdentity 때 iam:GetPolicy 및 true가 필요합니다--provision

2021년 2월 4일

[새로운 보안 터널링 구성 요소 릴리스](#)

보안 터널링 구성 요소의 버전 1.0.0을 사용할 수 있습니다 AWS IoT Greengrass. 이 AWS 제공 구성 요소는 AWS IoT 보안 터널링을 사용하여 제한된 방화벽 뒤에 있는 Greengrass 코어 디바이스와 안전한 양방향 통신을 설정합니다.

2021년 1월 21일

[AWS IoT Device TesterAWS IoT Greengrass v4.0.1용 릴리스](#)

IDT for AWS IoT Greengrass V2 버전 4.0.1을 사용할 수 있습니다. 이 버전을 사용하면 IDT를 사용하여 장치 검증을 위한 사용자 지정 테스트 제품군을 개발하고 실행할 수 있습니다. 여기에는 macOS 및 Windows용 코드 서명 IDT 애플리케이션도 포함됩니다.

2020년 12월 22일

[의 최초 릴리스 AWS IoT Greengrass Version 2](#)

AWS IoT Greengrass V2 는의 새로운 메이저 버전 릴리스입니다 AWS IoT Greengrass. 이 버전에는 모듈식 소프트웨어 구성 요소 및 지속적 배포와 같은 여러 기능이 추가되어 있습니다. 이러한 기능을 사용하면 엣지 애플리케이션을 더욱 쉽게 개발하고 관리할 수 있습니다.

2020년 12월 15일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 [AWS 용어집](#)을 참조하세요.

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