aws

개발자 가이드

AWS IoT Core



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

AWS IoT Core: 개발자 가이드

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

Amazon 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용되어서는 안되며, 고객에게 혼동을 일으키거나 Amazon 브랜드 이미지를 떨어뜨리고 폄하하는 방식으로 이용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

AWS IoT란 무엇인가요? 1
디바이스 및 앱의 액세스 방식 AWS IoT 2
에서 수행할 AWS IoT 수 있는 작업 2
산업용 IoT 3
홈 자동화의 IoT 3
AWS IoT 작동 방식 4
IoT 세계 4
AWS IoT 서비스 개요 7
AWS IoT Core 서비스 11
에 대해 자세히 알아보기 AWS IoT 14
에 대한 훈련 리소스 AWS IoT 14
AWS IoT 리소스 및 가이드 15
AWS IoT 소셜 미디어의 16
AWS AWS IoT Core 규칙 엔진에서 사용하는 서비스 16
에서 지원하는 통신 프로토콜 AWS IoT Core 17
AWS IoT 콘솔의 새로운 기능 18
범례 21
AWS SDKs 작업 21
시작하기 자습서 23
첫 번째 디바이스를에 연결 AWS IoT Core 23
설정 AWS 계정 25
에 가입 AWS 계정 25
관리자 액세스 권한이 있는 사용자 생성 26
AWS IoT 콘솔 열기 27
대화형 자습서 27
IoT 디바이스 연결 28
오프라인 디바이스 상태 저장 29
디바이스 데이터를 서비스로 라우팅 30
빠른 연결 자습서 31
1단계. 자습서 시작 32
2단계. 사물 객체 만들기 32
3단계. 디바이스에 파일 다운로드 35
4단계. 샘플 실행 38
5단계. 더 살펴보기 41

연결 테스트 42
고급 연결 자습서 47
어떤 디바이스 옵션이 가장 적합한가요? 48
AWS IoT 리소스 생성 49
디바이스 구성 54
MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기 91
MQTT 클라이언트에서 MQTT 메시지 보기 91
MQTT 클라이언트에서 MQTT 메시지 게시 94
MQTT 클라이언트에서 공유 구독 테스트 96
AWS IoT 자습서 98
AWS IoT Device Client를 사용하여 데모 빌드 98
AWS IoT Device Client로 데모를 빌드하기 위한 사전 조건 99
IoT Device Client 사용 준비 101
IoT Device Client 설치 및 구성 115
MQTT를 사용하여 디바이스 클라이언트와 통신 127
Device Client를 사용하여 IoT 작업 실행 146
정리 160
AWS IoT 디바이스 SDKs를 사용하여 솔루션 구축 170
AWS IoT 디바이스 SDKs로 솔루션 구축 시작 170
디바이스 SDK를 AWS IoT Core 사용하여에 AWS IoT 디바이스 연결 170
디바이스 데이터를 다른 서비스로 라우팅하는 AWS IoT 규칙 생성 193
디바이스 섀도우로 디바이스가 오프라인 상태일 때 디바이스 상태 유지 234
에 대한 사용자 지정 권한 부여자 생성 AWS IoT Core 263
AWS IoT 및 Raspberry Pi를 사용하여 토양 수분 모니터링 280
에 연결 AWS IoT Core 293
AWS IoT Core - 컨트롤 플레인 엔드포인트 293
AWS IoT 디바이스 엔드포인트 295
AWS IoT Core for LoRaWAN 게이트웨이 및 디바이스 296
AWS IoT Core 서비스 엔드포인트에 연결 297
AWS CLI 용 AWS IoT Core 298
AWS SDKs 298
AWS 모바일 SDKs 304
AWS IoT Core 서비스의 REST APIs 305
디바이스를에 연결 AWS IoT 305
AWS IoT 디바이스 데이터 및 서비스 엔드포인트 306
AWS IoT 디바이스 SDKs 308

디바이스 통신 프로토콜 311
MQTT 주제 354
도메인 구성 378
AWS IoT FIPS 엔드포인트에 연결 404
AWS IoT Core - 제어 영역 엔드포인트 404
AWS IoT Core - 데이터 영역 엔드포인트 405
AWS IoT Core- 자격 증명 공급자 엔드포인트 405
AWS IoT Device Management - 작업 데이터 엔드포인트 406
AWS IoT Device Management- Fleet Hub 엔드포인트 406
AWS IoT Device Management- 보안 터널링 엔드포인트 406
AWS IoT Device Management - 관리형 통합 엔드포인트 407
디바이스 관리 408
레지스트리 408
사물 생성 409
사물 목록 표시 410
사물 설명 412
사물 업데이트 413
사물 삭제 413
사물에 위탁자 연결 413
위탁자와 연결된 사물 나열 414
사물에 연결된 위탁자 나열 415
위탁자와 연결된 사물 나열 V2 416
사물에 연결된 위탁자 나열 V2 416
사물에서 보안 주체 분리 417
사물 유형 417
사물 유형 생성 418
사물 유형 목록 표시 418
사물 유형 설명 419
사물에 사물 유형 연결 420
사물 유형 업데이트 420
사물 유형 사용 중지 421
사물 유형 삭제 422
정적 사물 그룹 422
정적 물체 그룹 생성 424
사물 그룹 설명 425
정적 사물 그룹에 사물 추가 426

정적 사물 그룹에서 사물 제거 427
사물 그룹에 속한 사물 나열 427
사물 그룹 목록 표시 428
사물이 속하는 그룹의 목록 표시 430
정적 사물 그룹 업데이트 430
사물 그룹 삭제 431
정적 사물 그룹에 정책 연결 432
정적 사물 그룹에서 정책 분리 432
정적 사물 그룹에 연결되어 있는 정책 나열 433
정책이 연결되어 있는 그룹의 목록 표시 433
사물에 적용되는 정책 가져오기 434
MQTT 작업에 대한 권한 부여 테스트 434
동적 사물 그룹 436
동적 사물 그룹의 사용 사례 437
동적 사물 그룹 생성 438
동적 사물 그룹 설명 439
동적 사물 그룹 업데이트 440
동적 사물 그룹 삭제 441
동적 및 정적 사물 그룹 제한 441
동적 사물 그룹 제한 사항 441
연결에 사물을 연결 444
사용 사례 444
연결에 사물을 연결하는 방법 445
전파 속성 추가 447
AWS Management Console 448
AWS CLI 449
리소스 태깅 451
태그 기본 사항 451
태그 규제 및 제한 452
태그가 있는 IAM 정책 453
결제 그룹 455
비용 할당 및 사용 데이터 보기 456
보안 458
의 보안 AWS IoT 459
Authentication 460
X.509 인증서 개요 460

서버 인증 460
클라이언트 인증 464
사용자 지정 인증 및 권한 부여 501
권한 부여 528
AWS 교육 및 인증 531
AWS IoT Core 정책 531
AWS IoT Core 자격 증명 공급자를 사용하여 AWS 서비스에 대한 직접 호출 권한 부여 609
IAM을 통한 교차 계정 액세스 615
데이터 보호 617
의 데이터 암호화 AWS IoT 619
의 전송 보안 AWS IoT Core 619
데이터 암호화 624
ID 및 액세스 관리 639
대상 639
IAM 자격 증명을 통한 인증 640
정책을 사용하여 액세스 관리 641
AWS IoT 에서 IAM을 사용하는 방법 642
자격 증명 기반 정책 예 673
AWS 관리형 정책 677
문제 해결 692
로깅 및 모니터링 694
모니터링 도구 694
규정 준수 확인 696
복원력 696
VPC 엔드포인트 AWS IoT Core 와 함께 사용 697
AWS IoT Core 컨트롤 플레인용 VPC 엔드포인트 생성 698
AWS IoT Core 데이터 영역에 대한 VPC 엔드포인트 생성 698
AWS IoT Core 자격 증명 공급자를 위한 VPC 엔드포인트 생성 699
Amazon VPC 엔드포인트 생성 700
프라이빗 호스팅 영역 구성 701
VPC 엔드포인트를 AWS IoT Core 통해에 대한 액세스 제어 704
제한 사항 705
를 사용하여 VPC 엔드포인트 확장 AWS IoT Core 706
VPC 엔드포인트에 사용자 지정 도메인 사용 706
에 대한 VPC 엔드포인트의 가용성 AWS IoT Core 706
VPC 엔드포인트 AWS IoT Device Management 와 함께 사용 707

인프라 보안 712
보안 모니터링 712
보안 모범 사례 713
에서 MQTT 연결 보호 AWS IoT 713
디바이스의 시계를 동기화 상태로 유지 716
서버 인증서 검증 716
디바이스별로 단일 자격 증명 사용 717
백업 AWS 리전 으로 두 번째 사용 717
정시 프로비저닝 사용 717
AWS IoT Device Advisor 테스트를 실행할 수 있는 권한 717
Device Advisor에 대한 교차 서비스 혼동된 대리자 예방 719
AWS 교육 및 인증 720
모니터링 AWS IoT 721
AWS IoT 로깅 구성 722
에서 V2 로깅 구성 AWS IoT 722
로깅 역할 및 정책 구성 723
에서 로깅 구성 AWS IoT (콘솔) 725
에서 계정 및 이벤트 수준 로깅 구성 AWS IoT (CLI) 727
AWS IoT (CLI)에서 리소스별 재정의 구성 729
로그 수준 732
Amazon CloudWatch를 사용하여 AWS IoT 경보 및 지표 모니터링 732
AWS IoT 지표 사용 733
CloudWatch 경보 생성 733
지표 및 차원 737
CloudWatch Logs AWS IoT 를 사용하여 모니터링 758
CloudWatch 콘솔에서 AWS IoT 로그 보기 759
CloudWatch Logs AWS IoT 로그 항목 760
디바이스 측 로그를 Amazon CloudWatch에 업로드합니다. 798
작동 방식 799
AWS IoT 규칙을 사용하여 디바이스 측 로그 업로드 800
AWS IoT API 호출 로깅 809
AWS IoT CloudTrail의 정보 809
AWS IoT 로그 파일 항목 이해 810
규칙 813
액세스 권한 부여 814
규칙 엔진 액세스 취소 816

역할 전달 권한 817
규칙 생성 818
규칙 생성(콘솔) 819
규칙 생성(CLI) 820
규칙 관리 825
규칙 태그 지정 825
규칙 보기 826
규칙 삭제 826
AWS IoT 규칙 작업 827
Apache Kafka 830
CloudWatch 경보 842
CloudWatch Logs 844
CloudWatch 지표 846
DynamoDB 848
DynamoDBv2 851
Elasticsearch 854
HTTP 856
AWS IoT Events 870
AWS IoT SiteWise 872
Firehose 878
Kinesis Data Streams 880
Lambda 882
Location 885
OpenSearch 889
재게시 892
S3 895
Salesforce IoT 897
SNS 898
SQS 900
단계 함수 903
Timestream 904
규칙 문제 해결 911
크로스 계정 리소스에 액세스 912
사전 조건 912
Amazon SQS에 대한 교차 계정 설정 912
Amazon SNS를 위한 교차 계정 설정 914

Amazon S3를 위한 교차 계정 설정 916
에 대한 교차 계정 설정 AWS Lambda 918
오류 처리(오류 작업) 920
오류 작업 메시지 형식 921
오류 작업 예제 922
Basic Ingest 923
Basic Ingest 사용 924
AWS IoT SQL 참조 925
SELECT 절 926
FROM 절 928
SET 절 929
WHERE 절 933
데이터 타입 933
연산자 938
함수 948
리터럴 1017
Case 문 1018
JSON 확장 1019
대체 템플릿 1021
중첩된 객체 쿼리 1023
이진 페이로드 1025
SQL 버전 1031
Shadows 1033
섀도우 사용 1033
명명된 섀도우 또는 명명되지 않은 섀도우 사용 선택 1034
섀도우 액세스 1034
디바이스, 앱 및 기타 클라우드 서비스에서 섀도우 사용 1035
메시지 순서 1036
섀도우 메시지 트리밍 1038
디바이스에서 섀도우 사용 1038
에 처음 연결할 때 디바이스 초기화 AWS IoT 1039
디바이스가 연결되어 있는 동안 메시지 처리 AWS IoT 1042
디바이스가에 다시 연결될 때 메시지 처리 AWS IoT 1042
앱 및 서비스에서 섀도우 사용 1043
에 연결할 때 앱 또는 서비스 초기화 AWS IoT 1044
앱 또는 서비스가에 연결되어 있는 동안 처리 상태 변경 AWS IoT 1044

디바이스 연결 상태 감지 1044
디바이스 섀도우 서비스 통신 시뮬레이션 1046
시뮬레이션 설정 1047
디바이스 초기화 1047
앱에서 업데이트 전송하기 1051
디바이스의 업데이트에 응답 1053
앱에서 업데이트 관찰 1058
시뮬레이션 이후의 작업 1060
섀도우와의 상호 작용 1060
프로토콜 지원 1061
상태 요청 및 보고 1061
섀도우 업데이트 1061
섀도우 문서 검색 1066
섀도우 데이터 삭제 1066
디바이스 섀도우 REST API 1069
GetThingShadow 1070
UpdateThingShadow 1071
DeleteThingShadow 1073
ListNamedShadowsForThing 1074
디바이스 섀도우 MQTT 주제 1075
/get 1076
/get/accepted 1077
/get/rejected 1078
/update 1079
/update/delta 1080
/update/accepted 1082
/update/documents 1083
/update/rejected 1084
/delete 1084
/delete/accepted 1085
/delete/rejected 1086
디바이스 섀도우 서비스 문서 1087
섀도우 문서 예제 1087
문서 속성 1094
델타 상태 1095
섀도우 문서 버전 관리 1097

섀도우 문서의 클라이언트 토큰 1097
빈 섀도우 문서 속성 1098
섀도우 문서의 배열 값 1098
디바이스 섀도우 오류 메시지 1099
소프트웨어 패키지 카탈로그 1101
소프트웨어 패키지 카탈로그 사용 준비 1101
패키지 버전 수명 주기 1101
패키지 버전 명명 규칙 1103
기본 버전 1104
버전 속성 1104
소프트웨어 재료표 1105
AWS IoT 플릿 인덱싱 활성화 1108
명명된 예약 섀도우 1108
소프트웨어 패키지 삭제 1110
보안 준비 1110
리소스 기반 인증 1110
AWS IoT 패키지 버전을 배포할 수 있는 작업 권한 1112
AWS IoT 예약된 명명된 섀도우를 업데이트하는 작업 권한 1113
AWS IoT Amazon S3에서 다운로드할 수 있는 작업 권한 1115
패키지 버전의 소프트웨어 재료 명세서를 업데이트할 수 있는 권한 1115
플릿 인덱싱 준비 1118
$package 섀도우를 데이터 소스로 설정 1119
콘솔에 표시된 지표 1119
쿼리 패턴 1120
getBucketsAggregation을 통한 패키지 버전 배포 수집 1122
AWS IoT 작업 준비 1123
AWS IoT 작업의 대체 파라미터 1123
배포를 위한 작업 문서 및 패키지 버전 준비 1127
배포 시 패키지 및 버전 이름 지정 1130
AWS IoT 동적 사물 그룹을 통한 작업 대상 지정 1131
명명된 예약 섀도우 및 패키지 버전 1131
소프트웨어 패키지 제거 1132
시작하기 1132
패키지 및 버전 생성 1133
패키지 버전 배포 1135
패키지 버전 연결 1137

작업 1139
AWS IoT 작업 액세스 1139
AWS IoT 작업 리전 및 엔드포인트 1139
원격 작업이란 무엇인가요? 1139
원격 작업에 AWS IoT Device Management 작업 사용의 이점 1140
AWS IoT 작업이란 무엇입니까? 1142
작업 관련 주요 개념 1143
작업 및 작업 실행 상태 1146
작업 관리 1151
작업에 대한 코드 서명 1151
작업 문서 1151
미리 서명된 URL 1152
파일 업로드를 위한 미리 서명된 URL 1154
Amazon S3 버전 관리를 사용하여 미리 서명된 URL 1155
콘솔을 사용하여 작업 생성 및 관리 1157
CLI를 사용하여 작업 생성 및 관리 1159
작업 템플릿 1171
사용자 지정 및 AWS 관리형 템플릿 1171
AWS 관리형 템플릿 사용 1172
사용자 정의 작업 템플릿 생성 1191
작업 구성 1199
작업 구성 작동 방식 1199
추가 구성 지정 1212
디바이스와 작업 1221
작업 서비스와 작업할 수 있도록 디바이스를 프로그래밍 1224
디바이스 워크플로우 1224
작업 워크플로 1226
작업 알림 1230
AWS IoT 작업 API 작업 1238
작업 관리 및 제어 API 및 데이터 형식 1241
작업 디바이스 MQTT API 및 HTTPS API 작업 및 데이터 형식 1260
작업의 사용자 및 디바이스 보호 1275
AWS IoT 작업에 필요한 정책 유형 1275
작업 사용자 및 클라우드 서비스 권한 부여 1276
디바이스에서 작업을 사용하도록 권한 부여 1288
AWS IoT 작업 제한 1292

작업 실행 한도 1292
활성 및 동시 작업 제한 1293
명령 1297
빠른 시작 1298
명령 개념 및 상태 1298
명령 주요 개념 1299
명령 상태 1303
명령 실행 상태 1303
명령 워크플로 1307
명령 생성 및 관리 1308
대상 선택 및 주제 구독 1308
명령 실행 시작 및 모니터링 1310
(선택 사항) 명령 이벤트에 대한 알림 활성화 1311
명령 생성 및 관리 1312
명령 리소스 생성 1312
명령에 대한 정보 검색 1318
에서 명령 나열 AWS 계정 1320
명령 리소스 업데이트 1321
명령 리소스의 사용 중지 또는 복원 1323
명령 리소스 삭제 1324
명령 실행 시작 및 모니터링 1325
명령 실행 시작 1326
명령 실행 결과 업데이트 1332
명령 실행 검색 1338
MQTT 테스트 클라이언트를 사용하여 명령 업데이트 보기 1341
에서 명령 실행 나열 AWS 계정 1343
명령 실행 삭제 1345
명령 리소스 사용 중지 1346
주요 고려 사항 1347
명령 리소스 사용 중지(콘솔) 1347
명령 리소스 사용 중지(CLI) 1347
사용 중지 시간 및 상태 확인 1348
명령 리소스 복원 1348
보안 터널링 1350
보안 터널링이란 무엇입니까? 1350
보안 터널링 개념 1350

보안 터널링 작동 방식 1352
보안 터널 수명 주기 1353
보안 터널링 자습서 1353
이 섹션의 자습서 1354
터널을 열고 원격 디바이스에 대한 SSH 세션 시작 1355
원격 디바이스에 대한 터널을 열고 브라우저 기반 SSH 사용 1372
로컬 프록시 1376
로컬 프록시 사용 방법 1376
웹 프록시를 사용하는 디바이스에 대한 로컬 프록시 구성 1382
멀티플렉싱 및 동시 TCP 연결 1390
다중 데이터 스트림 멀티플렉싱 1390
동시 TCP 연결 사용 1394
원격 디바이스 구성 및 IoT 에이전트 사용 1396
IoT 에이전트 코드 조각 1397
터널에 대한 액세스 제어 1399
터널 액세스 사전 조건 1399
터널 액세스 정책 1399
보안 터널링 연결 문제 해결 1406
잘못된 클라이언트 액세스 토큰 오류 1407
클라이언트 토큰 불일치 오류 1407
원격 디바이스 연결 문제 1408
디바이스 프로비저닝 1411
에서 디바이스 프로비저닝 AWS IoT 1412
플릿 프로비저닝 API 1413
플릿 프로비저닝을 사용하여 디바이스 인증서가 없는 디바이스 프로비저닝 1414
클레임에 의한 프로비저닝 1414
신뢰할 수 있는 사용자에 의한 프로비저닝 1417
AWS CLI에서 사전 프로비저닝 후크 사용 1419
디바이스 인증서가 있는 디바이스 프로비저닝 1422
단일 사물 프로비저닝 1423
JIT 프로비저닝 1424
대량 등록 1430
프로비저닝 템플릿 1431
파라미터 섹션 1431
리소스 섹션 1432
대량 등록에 대한 템플릿 예 1438

JIT 프로비저닝(JITP)의 템플릿 예 1439
플릿 프로비저닝 1441
사전 프로비저닝 후크 1444
사전 프로비저닝 후크 입력 1445
사전 프로비저닝 후크 반환 값 1445
사전 프로비저닝 후크 Lambda 예제 1446
인증서 공급자를 사용한 자체 관리형 AWS IoT Core 인증서 서명 1449
플릿 프로비저닝에서 자체 관리형 인증서 서명의 작동 방식 1450
인증서 공급자 Lambda 함수 입력 1451
인증서 공급자 Lambda 함수 반환 값 1452
Lambda 함수 예제 1452
플릿 프로비저닝을 위한 자체 관리형 인증서 서명 1454
AWS CLI 인증서 공급자에 대한 명령 1455
디바이스를 설치하는 사용자를 위한 IAM 정책 및 역할 생성 1458
디바이스를 설치할 사용자에 대한 IAM 정책 생성 1458
디바이스를 설치할 사용자에 대한 IAM 역할 생성 1459
기존 정책을 업데이트하여 새 템플릿 승인 1460
디바이스 프로비저닝 MQTT API 1461
CreateCertificateFromCsr 1462
CreateKeysAndCertificate 1464
RegisterThing 1466
플릿 인덱싱 1470
인덱스 업데이트 관리 1470
특정 디바이스의 연결 상태 쿼리 1470
데이터 원본 전체에서 검색 1470
집계 데이터 쿼리 1471
플릿 지표를 사용하여 집계 데이터 모니터링 및 경보 생성 1471
플릿 인덱싱 관리 1471
사물 인덱싱 1471
사물 그룹 인덱싱 1472
관리형 필드 1473
사용자 정의 필드 1475
사물 인덱싱 관리 1475
사물 그룹 인덱싱 관리 1491
디바이스 연결 상태 1493
작동 방식 1493

특성 1493
이점 1494
사전 조건 1494
예제 1495
집계 데이터 쿼리 1496
GetStatistics 1497
GetCardinality 1499
GetPercentiles 1500
GetBucketsAggregation 1503
권한 부여 1504
쿼리 구문 1504
지원되는 기능 1504
지원되지 않는 기능 1505
참고 1505
사물 쿼리 예 1506
사물 그룹 쿼리 예 1510
위치 데이터 인덱싱 1512
지원되는 데이터 형식 1512
위치 데이터를 인덱싱하는 방법 1513
사물 인덱싱 구성 업데이트 1514
지오쿼리 예시 1516
시작하기 자습서 1517
플릿 지표 1522
시작하기 자습서 1522
플릿 지표 관리 1529
MQTT 기반 파일 전송 1536
스트림이란 무엇입니까? 1536
스트림 관리 1537
디바이스에 권한 부여 1538
디바이스를에 연결 AWS IoT 1539
TagResource 사용 1539
디바이스에서 AWS IoT MQTT 기반 파일 전송 사용 1540
DescripbeStream을 사용하여 스트림 데이터 가져오기 1541
스트림 파일에서 데이터 블록 가져오기 1543
AWS IoT MQTT 기반 파일 전송으로 인한 오류 처리 1548
FreeRTOS OTA의 예제 사용 사례 1550

디바이스 어드바이저 1551
설정 1552
IoT 사물 생성 1553
디바이스 역할로 사용할 IAM 역할 생성 1553
IAM 사용자가 Device Advisor를 사용할 수 있도록 사용자 정의 관리형 정책 생성 1556
Device Advisor를 사용할 IAM 사용자 생성 1556
디바이스 구성 1559
콘솔에서 Device Advisor 시작하기 1560
Device Advisor 워크플로 1570
사전 조건 1570
테스트 스위트 정의 만들기 1570
테스트 스위트 정의 가져오기 1572
테스트 엔드포인트 가져오기 1573
테스트 스위트 실행 시작하기 1573
테스트 스위트 실행 가져오기 1574
테스트 스위트 실행 중지 1574
성공적인 자격 테스트 스위트 실행을 위한 자격 보고서 받기 1575
Device Advisor 세부 콘솔 워크플로 1575
사전 조건 1576
테스트 스위트 정의 만들기 1576
테스트 스위트 실행 시작하기 1583
테스트 스위트 실행 중지(선택 사항) 1584
테스트 스위트 실행 세부 정보 및 로그 보기 1586
AWS IoT 검증 보고서 다운로드 1588
장기간 테스트 콘솔 워크플로 1588
Device Advisor VPC 엔드포인트(AWS PrivateLink) 1597
AWS IoT Core Device Advisor VPC 엔드포인트에 대한 고려 사항 1597
에 대한 인터페이스 VPC 엔드포인트 생성 AWS IoT Core Device Advisor 1598
VPC 엔드포인트를 AWS IoT Core Device Advisor 통해에 대한 액세스 제어 1599
Device Advisor 테스트 케이스 1600
Device Advisor 테스트 사례는 AWS Device Qualification Program에 적합합니다. 1600
TLS 1601
MQTT 1608
섀도우 1621
작업 실행 1624
권한 및 정책 1626

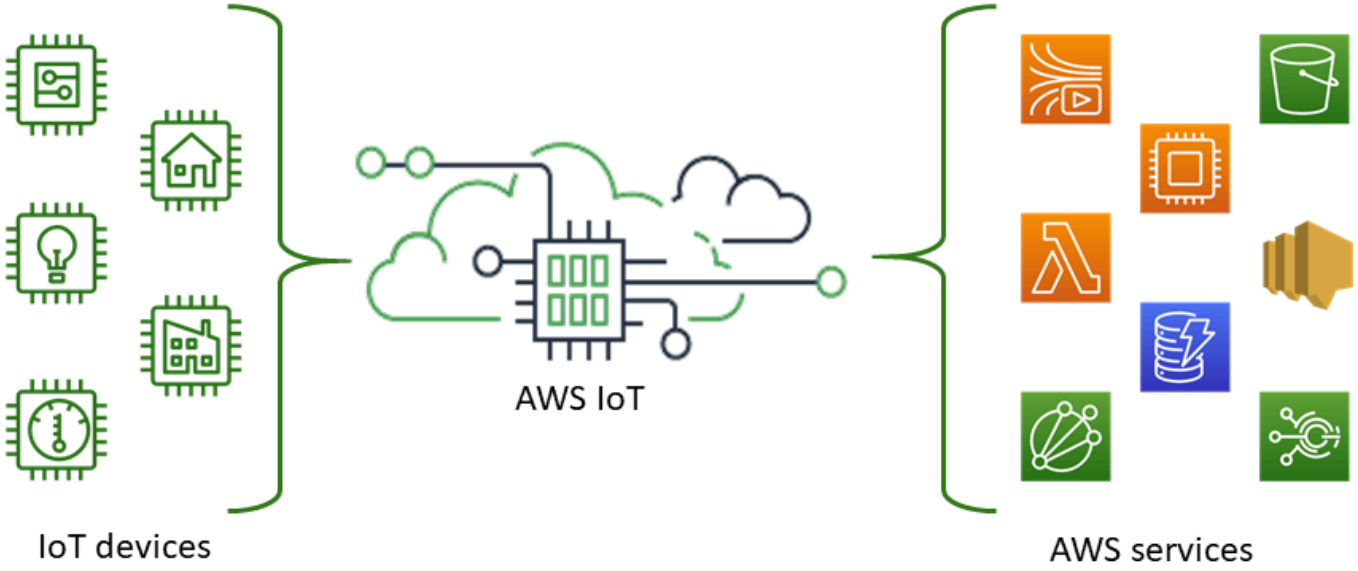
장기간 테스트 1627
Device Location 1645
측정 유형 및 솔버 1645
AWS IoT Core 디바이스 위치 작동 방식 1646
AWS IoT Core 디바이스 위치를 사용하는 방법 1648
IoT 디바이스의 위치 확인 1649
디바이스 위치 확인(콘솔) 1649
디바이스 위치 확인(API) 1652
위치를 확인할 때 발생하는 오류 해결 1654
MQTT 주제를 사용하여 디바이스 위치 해석 1655
디바이스 위치 MQTT 주제의 형식 1655
디바이스 위치 MQTT 주제 정책 1656
디바이스 위치 주제 및 페이로드 1657
위치 솔버 및 디바이스 페이로드 1662
Wi-Fi 기반 솔버 1663
셀룰러 기반 솔버 1664
IP 역방향 조회 솔버 1668
GNSS 솔버 1669
Amazon Sidewalk 지원 디바이스용 BLE 솔버 1670
이벤트 메시지 1672
이벤트 메시지 생성 방법 1672
이벤트 메시지 수신 정책 1672
에 대한 이벤트 활성화 AWS IoT 1673
레지스트리 이벤트 1678
사물 이벤트 1678
사물 유형 이벤트 1680
사물 그룹 이벤트 1683
작업 이벤트 1689
명령 실행 이벤트 1694
수명 주기 이벤트 1695
연결/연결 해제 이벤트 1696
연결 시도 실패 이벤트 1700
구독/구독 취소 이벤트 1702
문제 해결 1704
AWS IoT Core 문제 해결 가이드 1704
연결 문제 진단 1705

진단 규칙 문제 1708
Shadows 문제 진단 1710
Salesforce 작업 문제 진단 1711
스트림 제한 진단 1713
디바이스 플릿 연결 해제 문제 해결 1713
AWS IoT Device Management 문제 해결 가이드 1714
AWS IoT 작업 문제 해결 1714
플릿 인덱싱 문제 해결 1719
AWS IoT 디바이스 관리 소프트웨어 패키지 카탈로그 문제 해결 1721
AWS IoT Device Advisor 문제 해결 가이드 1728
AWS IoT 오류 1731
AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트 1733
AWS IoT 디바이스 SDK 1733
AWS IoT Device SDK for Embedded C 1735
AWS Mobile SDK 1736
AWS IoT 디바이스 클라이언트 1737
이전 AWS IoT 디바이스 SDK 버전 1738
코드 예제 1739
기본 사항 1740
안녕하세요 AWS IoT 1740
기본 사항 알아보기 1749
작업 1860
AWS IoT 할당량 1954
AWS IoT Core 요금 1955
..... mcmlvi

AWS IoT란 무엇인가요?

AWS IoT 는 IoT 디바이스를 다른 디바이스 및 AWS 클라우드 서비스에 연결하는 클라우드 서비스를 제공합니다.는 IoT 디바이스를 AWS IoT기반 솔루션에 통합하는 데 도움이 되는 디바이스 소프트웨어를 AWS IoT 제공합니다. 디바이스가에 연결할 수 있는 경우 AWS IoT는 AWS 가 제공하는 클라우드 서비스에 디바이스를 AWS IoT 연결할 수 있습니다.

에 대한 실습 소개는 단원을 AWS IoT참조하십시오시작하기 자습서.



를 사용하면 솔루션에 가장 적합한 up-to-date 기술을 선택할 AWS IoT수 있습니다. 필드에서 IoT 디바이스를 관리하고 지원하는 데 도움이 되도록는 다음 프로토콜을 AWS IoT Core 지원합니다.

- MQTT(메시지 큐 및 원격 분석 전송)
- MQTT over WSS(웹 소켓 보안)
- HTTPS(하이퍼텍스트 전송 프로토콜 - 보안)
- LoRaWAN(장거리 광역 네트워크)

AWS IoT Core 메시지 브로커는 MQTT 및 MQTT over WSS 프로토콜을 사용하여 메시지를 게시하고 구독하는 디바이스 및 클라이언트를 지원합니다. HTTPS 프로토콜을 사용하여 메시지를 게시하는 디바이스와 클라이언트도 지원합니다.

AWS IoT Core for LoRaWAN은 무선 LoRaWAN(저전력 장거리 광역 네트워크) 디바이스를 연결하고 관리하는 데 도움이 됩니다. AWS IoT Core for LoRaWAN은 LoRaWAN 네트워크 서버(LNS)를 개발하고 운영할 필요성을 대체합니다.

디바이스 통신, 규칙 또는 작업과 같은 AWS IoT 기능이 필요하지 않은 경우 AWS 요구 사항에 더 적합할 수 있는 다른 AWS IoT 메시징 서비스에 대한 자세한 내용은 메시징을 참조하세요.

디바이스 및 앱의 액세스 방식 AWS IoT

AWS IoT 는 AWS IoT 자습서에 대해 다음 인터페이스를 제공합니다.

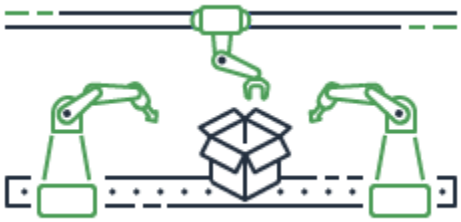
- AWS IoT 디바이스 SDKs- 디바이스에서 메시지를 보내고 받는 애플리케이션을 빌드합니다 AWS IoT. 자세한 내용은 AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트 단원을 참조하십시오.
- AWS IoT Core for LoRaWAN - for LoRaWANAWS IoT Core LoRaWAN) 디바이스 및 게이트웨이를 연결하고 관리합니다.
- AWS Command Line Interface (AWS CLI) - Windows, macOS 및 Linux AWS IoT 에서에 대한 명령을 실행합니다. 이러한 명령을 사용하여 사물 객체, 인증서, 규칙, 작업 및 정책을 생성하고 관리할 수 있습니다. 시작하려면 AWS Command Line Interface 사용 설명서를 참조하세요. 명령에 대한 자세한 내용은 명령 AWS IoT참조의 iot를 AWS CLI 참조하세요.
- AWS IoT API - HTTP 또는 HTTPS 요청을 사용하여 IoT 애플리케이션을 빌드합니다. 이러한 API 작업을 사용하여 사물 객체, 인증서, 규칙 및 정책을 프로그래밍 방식으로 생성하고 관리할 수 있습니다. 의 API 작업에 대한 자세한 내용은 API AWS IoT참조의 작업을 AWS IoT 참조하세요.
- AWS SDKs- 언어별 API를 사용하여 IoT 애플리케이션을 빌드합니다. APIs 이러한 SDK는 HTTP/HTTPS API를 래핑하고 지원되는 언어로 프로그램을 작성할 수 있게 해줍니다. 자세한 정보는 AWS SDK 및 도구 단원을 참조하세요.

IoT 솔루션의 사물 객체, 인증서, 규칙, 작업, 정책 및 기타 요소를 구성하고 관리할 수 있는 그래픽 사용자 인터페이스(GUI)를 제공하는 AWS IoT 콘솔을 AWS IoT 통해에 액세스할 수도 있습니다.

에서 수행할 AWS IoT 수 있는 작업

이 주제에서는 AWS IoT 가 지원하는 데 필요할 수 있는 몇 가지 솔루션에 대해 설명합니다.

산업용 IoT



다음은 IoT 기술을 적용하여 산업 프로세스의 성능과 생산성을 개선하는 산업 사용 사례를 위한 AWS IoT 솔루션의 몇 가지 예입니다.

산업용 사용 사례의 솔루션

- AWS IoT 를 사용하여 산업 운영에서 예측 품질 모델 구축

  AWS IoT 가 산업 운영에서 데이터를 수집하고 분석하여 예측 품질 모델을 구축하는 방법을 알아봅니다. 자세히 알아보기
- AWS IoT 를 사용하여 산업 운영에서 예측 유지 관리 지원

  AWS IoT 가 예방 유지 관리를 계획하여 예상치 못한 가동 중지 시간을 줄이는 데 도움이 되는 방법을 알아봅니다. 자세히 알아보기

홈 자동화의 IoT



다음은 IoT 기술을 적용하여 연결된 홈 디바이스를 사용하여 가정 활동을 자동화하는 확장 가능한 IoT 애플리케이션을 구축하는 홈 자동화 사용 사례를 위한 AWS IoT 솔루션의 몇 가지 예입니다. IoT

홈 자동화를 위한 솔루션

- 연결된 홈 AWS IoT 에서 사용

  AWS IoT 가 통합 홈 자동화 솔루션을 제공하는 방법을 알아봅니다.

- AWS IoT 를 사용하여 홈 보안 및 모니터링 제공

  AWS IoT 가 홈 자동화 솔루션에 기계 학습 및 엣지 컴퓨팅을 적용하는 방법을 알아봅니다.

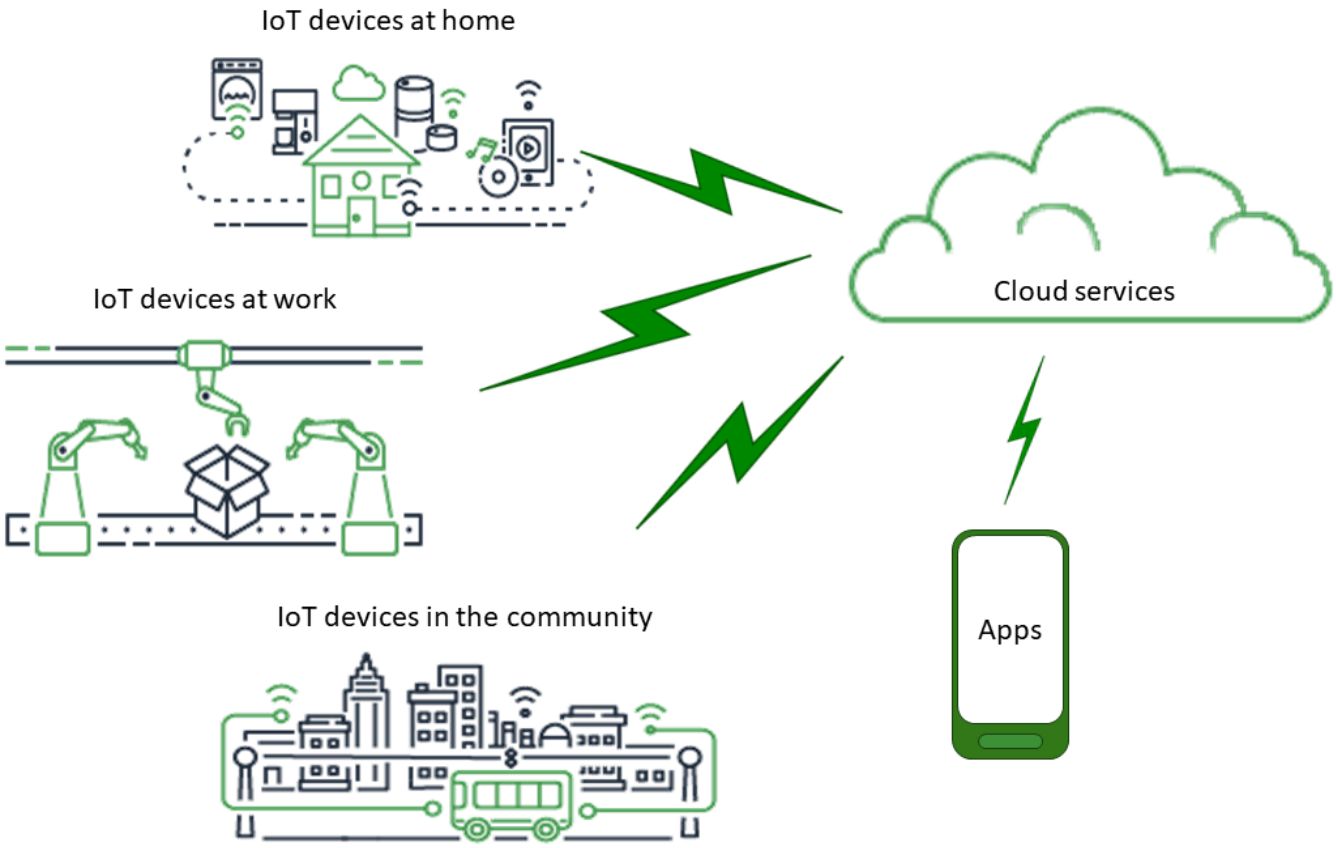
산업, 소비자 및 상업용 사용 사례에 대한 솔루션 목록은 AWS IoT 솔루션 리포지토리를 참조하세요.

AWS IoT 작동 방식

AWS IoT 는 IoT 솔루션을 구현하는 데 사용할 수 있는 클라우드 서비스 및 디바이스 지원을 제공합니다.는 IoT 기반 애플리케이션을 지원하는 많은 클라우드 서비스를 AWS 제공합니다. 따라서 어디서부터 시작해야 할지 이해하는 데 도움이 되도록 이 섹션에서는 IoT 세계를 소개하는 데 필수적인 개념의 다이어그램과 정의를 다룹니다.

IoT 세계

일반적으로 사물 인터넷(IoT)은 이 다이어그램에 표시된 주요 구성 요소로 이루어집니다.



앱

앱을 사용하면 최종 사용자가 IoT 디바이스와 해당 디바이스가 연결된 클라우드 서비스에서 제공하는 기능에 액세스할 수 있습니다.

클라우드 서비스

클라우드 서비스는 인터넷에 연결된 분산, 대규모 데이터 저장 및 처리 서비스입니다. 그러한 예는 다음과 같습니다.

- IoT 연결 및 관리 서비스

  AWS IoT 는 IoT 연결 및 관리 서비스의 예입니다.
- Amazon Elastic Compute Cloud 및와 같은 컴퓨팅 서비스 AWS Lambda
- 데이터베이스 서비스(예: Amazon DynamoDB)

통신

디바이스는 다양한 기술 및 프로토콜을 사용하여 클라우드 서비스와 통신합니다. 그러한 예는 다음과 같습니다.

- Wi-Fi/광대역 인터넷
- 광대역 셀룰러 데이터
- 협대역 셀룰러 데이터
- 장거리 광역 네트워크(LoRaWAN)
- 독점 RF 통신

Devices

디바이스는 인터페이스와 통신을 관리하는 하드웨어 유형입니다. 디바이스는 일반적으로 모니터하고 제어하는 실제 인터페이스와 가까운 곳에 있습니다. 디바이스에는 마이크로 컨트롤러, CPU, 메모리와 같은 컴퓨팅 및 스토리지 리소스가 포함될 수 있습니다. 그러한 예는 다음과 같습니다.

- Raspberry Pi
- Arduino
- 음성 인터페이스 어시스턴트

- LoRaWAN 및 디바이스
- Amazon Sidewalk 디바이스
- 맞춤형 IoT 디바이스

인터페이스

인터페이스는 디바이스를 물리적 세계에 연결하는 구성 요소입니다.

- 사용자 인터페이스

  디바이스와 사용자가 서로 통신할 수 있게 해주는 구성 요소입니다.

  - 입력 인터페이스

    사용자가 디바이스와 통신할 수 있도록 함

    예: 키패드, 버튼
  - 출력 인터페이스

    디바이스가 사용자와 통신할 수 있도록 함

    예: 영숫자 디스플레이, 그래픽 디스플레이, 표시등, 알람 벨
- 센서

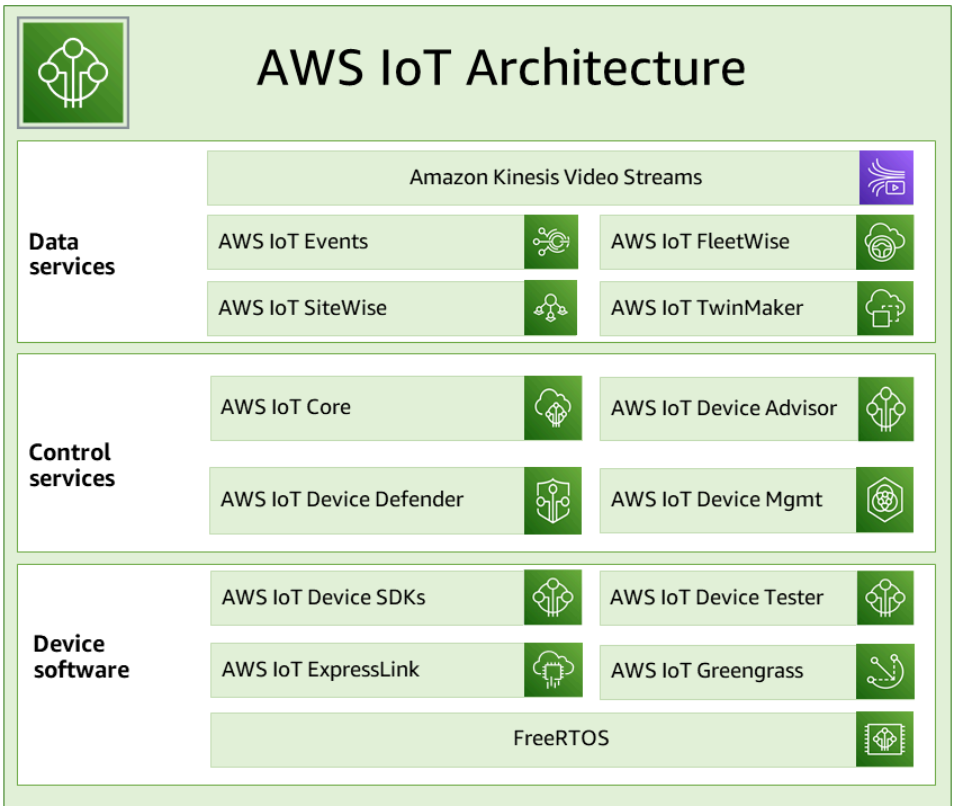
  디바이스가 이해하는 방식으로 외부 세계의 무언가를 측정하거나 감지하는 입력 구성 요소. 그러한 예는 다음과 같습니다.
  - 온도 센서(온도를 아날로그 또는 디지털 신호로 변환)
  - 습도 센서(상대 습도를 아날로그 또는 디지털 신호로 변환)
  - 아날로그-디지털 변환기(아날로그 전압을 숫자 값으로 변환)
  - 초음파 거리 측정 장치(거리를 숫자 값으로 변환)
  - 광학 센서(라이트 레벨을 숫자 값으로 변환)
  - 카메라(이미지 데이터를 디지털 데이터로 변환)
- 액추에이터

  디바이스가 외부 세계에서 무언가를 제어하는 데 사용할 수 있는 출력 구성 요소. 그러한 예는 다음과 같습니다.
  - 스테퍼 모터(전기 신호를 움직임으로 변환)

- 릴레이(높은 전기 전압 및 전류 제어)

AWS IoT 서비스 개요

IoT 유니버스에서는 세계와 상호 작용하는 디바이스를 지원하는 서비스와와 사이를 통과하는 데이터를 AWS IoT 제공합니다 AWS IoT. AWS IoT 는 IoT 솔루션을 지원하기 위해이 그림에 표시된 서비스로 구성됩니다.



AWS IoT 디바이스 소프트웨어

AWS IoT 는 IoT 디바이스를 지원하기 위해이 소프트웨어를 제공합니다.

AWS IoT 디바이스 SDKs

AWS IoT 디바이스 및 모바일 SDKs 사용하면 디바이스를에 효율적으로 연결할 수 있습니다 AWS IoT. AWS IoT 디바이스 및 모바일 SDKs에는 오픈 소스 라이브러리, 샘플이 포함된 개발자 가이드, 이식 가이드가 포함되어 있으므로 선택한 하드웨어 플랫폼에 따라 혁신적인 IoT 제품 또는 솔루션을 구축할 수 있습니다.

AWS IoT Device Tester

AWS IoT Device Tester FreeRTOS용 및 AWS IoT Greengrass 는 마이크로컨트롤러용 테스트 자동화 도구입니다.는 디바이스를 AWS IoT Device Tester 테스트하여 FreeRTOS를 실행할지 아니면 AWS IoT 서비스와 AWS IoT Greengrass 상호 운용할지 결정합니다.

AWS IoT ExpressLink

AWS IoT ExpressLink는 AWS 파트너가 개발하고 제공하는 다양한 하드웨어 모듈을 지원합니다. 연결 모듈에는 AWS검증된 소프트웨어가 포함되어 있으므로 디바이스를 클라우드에 안전하게 연결하고 다양한 AWS 서비스와 원활하게 통합할 수 있습니다. 자세한 내용은 AWS IoT ExpressLink 개요 페이지 또는 AWS IoT ExpressLink Programmer's Guide(ExpressLink 프로그래머 가이드)를 참조하세요.

AWS IoT Greengrass

AWS IoT Greengrass는 엣지 디바이스 AWS IoT 로 확장되므로 생성한 데이터에 대해 로컬로 작동하고, 기계 학습 모델을 기반으로 예측을 실행하고, 디바이스 데이터를 필터링 및 집계할 수 있습니다. AWS IoT Greengrass 는 디바이스가 해당 데이터가 생성된 위치와 더 가까운 곳에서 데이터를 수집 및 분석하고, 로컬 이벤트에 자율적으로 대응하고, 로컬 네트워크의 다른 디바이스와 안전하게 통신할 수 있습니다. AWS IoT Greengrass 를 사용하여 엣지 디바이스를 AWS 서비스 또는 타사 서비스에 연결할 수 있는 구성 요소라는 사전 구축된 소프트웨어 모듈을 사용하여 엣지 애플리케이션을 구축할 수 있습니다.

FreeRTOS

FreeRTOS는 IoT 솔루션에 소형의 저전력 엣지 디바이스를 포함시킬 수 있는 마이크로 컨트롤러용 오픈 소스 실시간 운영 체제입니다. FreeRTOS에는 커널과 많은 애플리케이션을 지원하는 성장하는 소프트웨어 라이브러리 세트가 포함되어 있습니다. FreeRTOS 시스템은 소형 저전력 디바이스를 AWS IoT에 안전하게 연결하고 AWS IoT Greengrass를 실행하는 보다 강력한 엣지 디바이스를 지원할 수 있습니다.

AWS IoT 제어 서비스

다음 AWS IoT 서비스에 연결하여 IoT 솔루션의 디바이스를 관리합니다.

AWS IoT Core

AWS IoT Core는 연결된 디바이스가 클라우드 애플리케이션 및 기타 디바이스와 안전하게 상호 작용할 수 있는 관리형 클라우드 서비스입니다. AWS IoT Core 는 많은 디바이스 및 메시지를 지원할

수 있으며 이러한 메시지를 처리하고 AWS IoT 엔드포인트 및 기타 디바이스로 라우팅할 수 있습니다. AWS IoT Core를 사용하면 애플리케이션이 연결되지 않은 경우에도 모든 디바이스와 상호 작용할 수 있습니다.

AWS IoT Core Device Advisor

AWS IoT Core Device Advisor는 디바이스 소프트웨어 개발 중에 IoT 디바이스의 유효성을 검사하기 위한 클라우드 기반의 완전 관리형 테스트 기능입니다. Device Advisor는 프로덕션에 디바이스를 배포하기 AWS IoT Core전에 IoT 디바이스의 안정적이고 안전한 연결을 검증하는 데 사용할 수 있는 사전 구축된 테스트를 제공합니다.

AWS IoT Device Defender

AWS IoT Device Defender는 IoT 디바이스 플릿을 보호하는 데 도움이 됩니다. AWS IoT Device Defender는 IoT 구성을 지속적으로 감사하여 보안 모범 사례를 벗어나지 않는지 확인합니다. AWS IoT Device Defender는 여러 디바이스에서 공유되는 자격 증명 인증서 또는에 연결하려는 취소된 자격 증명 인증서가 있는 디바이스와 같이 보안 위험을 초래할 수 있는 IoT 구성의 격차를 감지하면 알림을 보냅니다AWS IoT Core.

AWS IoT 디바이스 관리

AWS IoT 디바이스 관리 서비스는 디바이스 플릿을 구성하는 수많은 연결된 디바이스를 추적, 모니터링 및 관리하는 데 도움이 됩니다. AWS IoT 디바이스 관리 서비스는 IoT 디바이스가 배포된 후 적절하고 안전하게 작동하는지 확인하는 데 도움이 됩니다. 또한 디바이스 액세스, 디바이스 상태 모니터링, 문제 감지 및 원격 문제 해결을 위한 보안 터널링과, 디바이스 소프트웨어 및 펌웨어 업데이트 관리 서비스를 제공합니다.

AWS IoT 데이터 서비스

IoT 솔루션의 디바이스에서 데이터를 분석하고 다음 AWS IoT 서비스를 사용하여 적절한 조치를 취합니다.

Amazon Kinesis Video Streams

Amazon Kinesis Video Streams를 사용하면 디바이스에서 AWS 클라우드로 라이브 비디오를 스트리밍하여 내구성 있게 저장, 암호화 및 인덱싱할 수 있으므로 easy-to-use APIs를 통해 데이터에 액세스할 수 있습니다. Amazon Kinesis Video Streams를 사용하여 스마트폰, 보안 카메라, 웹캠, 차량 내장 카메라, 드론 등을 포함한 수많은 소스로부터 방대한 양의 라이브 비디오 데이터를 캡처할 수 있습니다. Amazon Kinesis Video Streams를 사용하면 라이브 및 온디맨드 시청용으로 비디오를 재생하고, Amazon Rekognition Video 및 ML 프레임워크용 라이브러리와 통합하여 컴퓨터 비전 및

비디오 분석을 활용하는 애플리케이션을 신속하게 구축할 수 있습니다. 오디오 데이터, 열 화상, 깊이 데이터, RADAR 데이터 등과 같은 비영상 시계열 데이터도 전송할 수 있습니다.

WebRTC 지원 Amazon Kinesis Video Streams

WebRTC 지원 Amazon Kinesis Video Streams는 표준을 준수하는 WebRTC 구현을 완전관리형 기능으로 제공합니다. WebRTC 지원 Amazon Kinesis Video Streams를 사용하면 미디어를 안전하게 라이브 스트리밍하거나 카메라 IoT 디바이스 및 WebRTC 호환 모바일 또는 웹 플레이어 간에 양방향 오디오 또는 비디오 상호 작용을 수행할 수 있습니다. 완전관리형 기능이므로 애플리케이션과 디바이스 전반에서 미디어를 안전하게 스트리밍하기 위해 신호 전송 또는 미디어 릴레이 서버와 같은 WebRTC 관련 클라우드 인프라를 구축, 운영 또는 확장할 필요가 없습니다. WebRTC 지원 Amazon Kinesis Video Streams를 사용하면 다양한 사용 사례에 필요한 카메라 IoT 디바이스, 웹 브라우저 및 모바일 디바이스 간의 실시간 오디오 또는 비디오 상호 작용과 라이브 피어 투 피어 미디어 스트리밍을 위한 애플리케이션을 손쉽게 구축할 수 있습니다.

AWS IoT 이벤트

AWS IoT 이벤트는 IoT 센서 및 애플리케이션의 이벤트를 감지하고 이에 응답합니다. 이벤트는 동작 신호를 사용하여 조명 및 보안 카메라를 활성화하는 모션 감지기와 같이 예상보다 복잡한 상황을 식별하는 데이터 패턴입니다. AWS IoT 이벤트는 여러 IoT 센서 및 애플리케이션의 데이터를 지속적으로 모니터링하고 AWS IoT Core IoT SiteWise, DynamoDB 등과 같은 다른 서비스와 통합하여 조기 감지 및 고유한 인사이트를 제공합니다.

AWS IoT FleetWise

AWS IoT FleetWise는 차량 데이터를 수집하고 거의 실시간으로 클라우드로 전송하는 데 사용할 수 있는 관리형 서비스입니다. AWS IoT FleetWise를 사용하면 다양한 프로토콜과 데이터 형식을 사용하는 차량에서 데이터를 쉽게 수집하고 구성할 수 있습니다. AWS IoT FleetWise는 저수준 메시지를 사람이 읽을 수 있는 값으로 변환하고 데이터 분석을 위해 클라우드에서 데이터 형식을 표준화하는 데 도움을 줍니다. 또한 데이터 수집 체계를 정의하여 차량에서 수집할 데이터와 클라우드로 전송할 시기를 제어할 수 있습니다.

AWS IoT SiteWise

AWS IoT SiteWise는 사업장의 게이트웨이에서 실행되는 소프트웨어를 제공하여 MQTT 메시지 또는 API에 의해 산업 장비에서 전달된 데이터를 대규모로 수집, 저장, 구성 및 모니터링합니다. 게이트웨이는 온프레미스 데이터 서버에 안전하게 연결하고 데이터를 수집 및 구성하고 AWS 클라우드로 전송하는 프로세스를 자동화합니다.

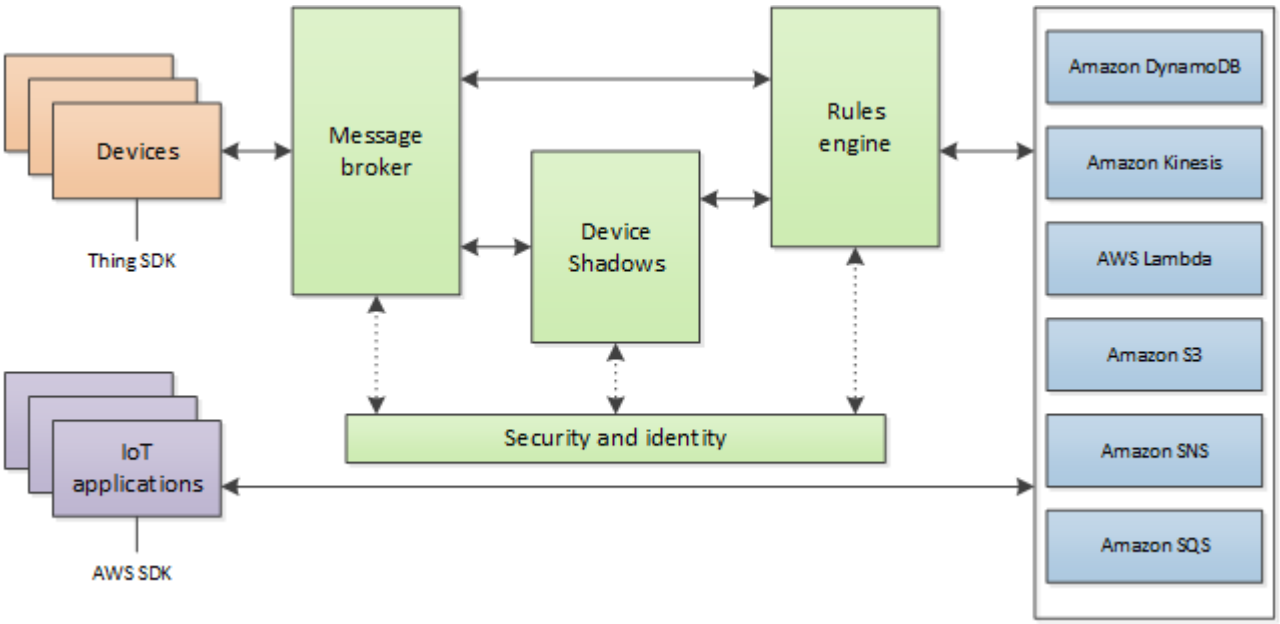
AWS IoT TwinMaker

AWS IoT TwinMaker는 물리적 및 디지털 시스템의 운영 디지털 트윈을 구축합니다. AWS IoT TwinMaker는 다양한 실제 센서, 카메라 및 엔터프라이즈 애플리케이션의 측정 및 분석을 사용하여

디지털 시각화를 생성하여 물리적 공장, 건물 또는 산업 공장을 추적할 수 있도록 지원합니다. 실제 데이터를 사용하여 작업을 모니터링하고 오류를 진단 및 해결하며 작업을 최적화할 수 있습니다.

AWS IoT Core 서비스

AWS IoT Core 는 다른 AWS 클라우드 서비스 및 애플리케이션이 인터넷에 연결된 디바이스와 상호 작용할 수 있도록 IoT 디바이스를 클라우드에 연결하는 서비스를 제공합니다.



다음 섹션에서는 그림에 표시된 각 AWS IoT Core 서비스에 대해 설명합니다.

AWS IoT Core 메시징 서비스

AWS IoT Core 연결 서비스는 IoT 디바이스와의 보안 통신을 제공하고와 간에 전달되는 메시지를 관리합니다 AWS IoT.

디바이스 게이트웨이

디바이스가 안전하고 효율적으로 AWS IoT와(과) 통신할 수 있게 해줍니다. 디바이스 통신은 X.509 인증서를 사용하는 보안 프로토콜에 의해 보호됩니다.

메시지 브로커

디바이스와 AWS IoT 애플리케이션이 서로 메시지를 게시하고 수신할 수 있는 안전한 메커니즘을 제공합니다. MQTT 프로토콜을 사용하여 직접 또는 MQTT over WebSocket을 사용하여 게시 및 구독할 수 있습니다. AWS IoT 가 지원하는 프로토콜 및 포트에 대한 자세한 내용은 the section called

"디바이스 통신 프로토콜" 단원을 참조하세요. 디바이스와 클라이언트는 HTTP REST 인터페이스를 사용하여 메시지 브로커에 데이터를 게시할 수도 있습니다.

메시지 브로커는 디바이스 데이터를 구독한 디바이스와 디바이스 섀도우 AWS IoT Core 서비스 및 규칙 엔진과 같은 다른 서비스에 디바이스 데이터를 배포합니다.

AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN을 사용하면 LoRaWAN 네트워크 서버(LNS)를 개발하고 운영할 필요 없이 LoRaWAN 디바이스 및 게이트웨이를 AWS 에 연결하여 프라이빗 LoRaWAN 네트워크를 설정할 수 있습니다. LoRaWAN 디바이스에서 받은 메시지는 규칙 엔진으로 전송되고, 여기에서 형식이 지정되어 다른 AWS IoT 서비스로 전송할 수 있습니다.

규칙 엔진

규칙 엔진은 저장 및 추가 처리를 위해 메시지 브로커의 데이터를 다른 AWS IoT 서비스에 연결합니다. 예를 들어 DynamoDB 테이블을 삽입, 업데이트 또는 쿼리하거나 규칙 엔진에서 정의한 표현식을 기반으로 Lambda 함수를 호출할 수 있습니다. SQL 기반 언어를 사용하여 메시지 페이로드에서 데이터를 선택하여 처리하고, Amazon Simple Storage Service(Amazon S3), Amazon DynamoDB, AWS Lambda등의 다른 서비스로 데이터를 전송할 수 있습니다. 메시지 브로커 및 다른 구독자에게 메시지를 다시 게시하는 규칙을 만들 수도 있습니다. 자세한 내용은 에 대한 규칙 AWS IoT 단원을 참조하십시오.

AWS IoT Core 제어 서비스

AWS IoT Core 제어 서비스는 디바이스 보안, 관리 및 등록 기능을 제공합니다.

사용자 지정 인증 서비스

사용자 지정 인증 서비스와 Lambda 함수를 사용하여 자신의 인증 및 권한 부여 전략을 관리할 수 있도록 사용자 지정 권한 부여자를 정의할 수 있습니다. 사용자 지정 권한 부여자를 사용하면 AWS IoT 가 보유자 토큰 인증 및 권한 부여 전략을 사용하여 디바이스를 인증하고 작업을 승인할 수 있습니다.

사용자 지정 권한 부여자는 다양한 인증 전략(예: JSON 웹 토큰 확인 또는 OAuth 공급자 콜아웃)을 구현할 수 있습니다. 디바이스 게이트웨이에서 MQTT 작업에 권한을 부여하는 데 사용하는 정책 문서를 반환해야 합니다. 자세한 내용은 사용자 지정 인증 및 권한 부여 단원을 참조하세요.

디바이스 프로비저닝 서비스

디바이스에 필요한 리소스, 즉 사물 객체, 인증서 및 한 가지 이상의 정책을 설명하는 템플릿을 사용하여 디바이스를 프로비저닝할 수 있습니다. 여기에서 사물 객체란 레지스트리에서 디바이스를 설

명하는 속성이 포함된 항목을 말합니다. 디바이스는 인증서를 사용하여 인증합니다 AWS IoT. 정책은 AWS IoT에서 실행할 수 있는 디바이스의 작업을 결정합니다.

템플릿에는 딕셔너리(맵)의 값으로 치환되는 변수가 포함됩니다. 딕셔너리에서 템플릿 변수 값만 다르게 전달하면 동일한 템플릿을 사용하여 다수의 디바이스를 프로비저닝하는 것도 가능합니다. 자세한 내용은 디바이스 프로비저닝 단원을 참조하세요.

그룹 레지스트리

그룹을 사용하면 디바이스를 그룹별로 범주화하여 여러 디바이스를 한 번에 관리할 수 있습니다. 그룹은 다른 그룹을 포함할 수 있으므로 그룹 계층 구조를 만들 수 있습니다. 상위 그룹에서 수행한 모든 작업은 하위 그룹에 적용됩니다. 상위 그룹의 모든 디바이스와 하위 그룹의 모든 디바이스에도 동일한 작업이 적용됩니다. 그룹에 부여된 권한은 해당 그룹과 해당 그룹의 모든 하위 그룹에 속하는 디바이스 전체에 적용됩니다. 자세한 내용은 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하세요.

작업 서비스

AWS IoT에 연결된 하나 이상의 디바이스에 전송되어 실행되는 일련의 원격 작업을 정의할 수 있습니다. 예를 들어 애플리케이션 또는 펌웨어 업데이트를 다운로드하여 설치하거나, 재부팅하거나, 인증서를 교체하거나, 원격 문제 해결 작업을 수행하도록 일련의 디바이스에 지시하는 작업을 정의할 수 있습니다.

작업을 생성하려면 먼저 실행할 원격 작업에 대한 설명과 작업을 실행할 대상 목록을 지정합니다. 대상은 개별 디바이스, 그룹 또는 둘 다가 될 수 있습니다. 자세한 내용은 AWS IoT 작업 단원을 참조하세요.

레지스트리

AWS 클라우드에서 각 디바이스에 연결된 리소스를 체계화합니다. 디바이스를 등록하고 각 디바이스에 최대 3개의 사용자 지정 속성을 연결할 수 있습니다. 또한 각 디바이스에 연결하여 인증서 및 MQTT 클라이언트 ID를 디바이스를 관리하고 문제 해결하는 능력을 개선할 수도 있습니다. 자세한 내용은 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하세요.

보안 및 자격 증명 서비스

AWS 클라우드의 보안에 대한 공동 책임을 제공합니다. 데이터를 안전하게 메시지 브로커로 전송하기 위해서는 디바이스가 자격 증명을 안전하게 보관해야 합니다. 메시지 브로커 및 규칙 엔진은 AWS 보안 기능을 사용하여 데이터를 디바이스 또는 다른 AWS 서비스로 안전하게 전송합니다. 자세한 내용은 Authentication 단원을 참조하십시오.

AWS IoT Core 데이터 서비스

AWS IoT Core 데이터 서비스는 IoT 솔루션이 항상 연결되어 있지 않은 디바이스에서도 안정적인 애플리케이션 환경을 제공하는 데 도움이 됩니다.

디바이스 섀도우

디바이스의 현재 상태 정보를 저장 및 검색하는 데 사용되는 JSON 문서입니다.

디바이스 섀도우 서비스

디바이스 섀도우 서비스는 디바이스가 온라인 상태인지 여부에 관계없이 애플리케이션이 디바이스와 통신할 수 있도록 디바이스의 상태를 유지합니다. 디바이스가 오프라인 상태이면 디바이스 섀도우 서비스는 연결된 애플리케이션의 데이터를 관리합니다. 디바이스가 다시 연결되면 디바이스 섀도 서비스의 섀도우 상태와 동기화합니다. 또한 디바이스는 항상 연결되어 있지 않을 수 있는 애플리케이션이나 기타 디바이스에서 사용할 수 있도록 현재 상태를 섀도우에 게시할 수 있습니다. 자세한 내용은 AWS IoT 디바이스 섀도우 서비스 단원을 참조하십시오.

AWS IoT Core 지원 서비스

용 Amazon Sidewalk 통합 AWS IoT Core

Amazon Sidewalk는 연결 옵션을 개선하는 공유 네트워크로, 디바이스가 더 잘 작동할 수 있도록 도와줍니다. Amazon Sidework는 반려동물 또는 귀중품을 찾는 디바이스, 스마트 홈 보안 및 조명 제어를 제공하는 디바이스, 기기 및 도구에 대한 원격 진단을 제공하는 디바이스 등 다양한 고객 디바이스를 지원합니다. 용 Amazon Sidewalk 통합을 AWS IoT Core 통해 디바이스 제조업체는 Sidewalk 디바이스 플릿을 AWS IoT 클라우드에 추가할 수 있습니다.

자세한 내용은 AWS IoT Core for Amazon Sidewalk 단원을 참조하십시오.

에 대해 자세히 알아보기 AWS IoT

이 주제는의 세계에 익숙해지는 데 도움이 됩니다 AWS IoT. 다양한 사용 사례에서 IoT 솔루션을 적용하는 방법에 대한 일반적인 정보, 교육 리소스, AWS IoT 및 기타 모든 AWS 서비스에 대한 소셜 미디어 링크,에서 사용하는 서비스 및 통신 프로토콜 목록을 얻을 수 AWS IoT 있습니다.

에 대한 훈련 리소스 AWS IoT

이러한 교육 과정을 통해 AWS IoT 에 대해 알아보고 솔루션 설계에 적용하는 방법을 배울 수 있습니다.

- 소개 AWS IoT

  AWS IoT 및 핵심 서비스에 대한 동영상 개요입니다.

- AWS IoT 인증 및 권한 부여에 대한 심층 분석

  AWS IoT 인증 및 권한 부여의 개념을 살펴보는 고급 과정입니다. 클라이언트가 AWS IoT 컨트롤 플레인 및 데이터 플레인 APIs에 액세스할 수 있도록 인증하고 권한을 부여하는 방법을 알아봅니다.

- 사물 인터넷 기초 시리즈

  다양한 IoT 기술 및 기능에 대한 IoT eLearning 모듈의 학습 경로입니다.

AWS IoT 리소스 및 가이드

이는의 특정 측면에 대한 심층적인 기술 리소스입니다 AWS IoT.

- IoT Lens - AWS IoT Well-Architected Framework

  IoT 애플리케이션을 설계하기 위한 모범 사례를 설명하는 문서입니다 AWS.

- 에 대한 MQTT 주제 설계 AWS IoT Core

  에서 MQTT 주제를 설계 AWS IoT Core 하고 MQTT로 AWS IoT Core 기능을 활용하기 위한 모범 사례를 설명하는 백서입니다.

- 요약 및 소개

  가 대규모 디바이스 플릿을 프로비저닝하기 위해 AWS IoT 제공하는 다양한 방법을 설명하는 PDF 문서입니다.

- AWS IoT Core Device Advisor

  AWS IoT Core Device Advisor는 프로덕션에 디바이스를 배포하기 AWS IoT Core전에 IoT 디바이스의 안정적이고 안전한 연결 모범 사례를 검증하는 데 사용할 수 있는 사전 구축된 테스트를 제공합니다.

- AWS IoT 리소스

  기술 가이드, 참조 아키텍처, eBook 및 엄선된 블로그 게시물과 같은 IoT 관련 리소스이며, 검색 가능한 색인으로 표시됩니다.

- IoT Atlas

일반적인 IoT 설계 문제를 해결하는 방법에 대한 개요입니다. IoT Atlas에서는 IoT 솔루션을 개발하는 동안 발생할 수 있는 설계 문제에 대해 심층적으로 살펴봅니다.

- AWS 백서 및 가이드

  AWS IoT 및 기타 AWS 기술에 대한 최신 백서 및 가이드 모음입니다.

AWS IoT 소셜 미디어의

이러한 소셜 미디어 채널은 AWS IoT 및 AWS관련 주제에 대한 정보를 제공합니다.

- 사물 인터넷 AWS IoT - 공식 블로그
- AWS IoT YouTube의 Amazon Web Services 채널에 있는 동영상

이러한 소셜 미디어 계정은 다음을 포함한 모든 AWS 서비스를 다룹니다. AWS IoT

- YouTube의 Amazon Web Services 채널
- Twitter의 Amazon Web Services
- Facebook의 Amazon Web Services
- Instagram의 Amazon Web Services
- LinkedIn의 Amazon Web Services

AWS AWS IoT Core 규칙 엔진에서 사용하는 서비스

AWS IoT Core 규칙 엔진은 이러한 AWS 서비스에 연결할 수 있습니다.

- Amazon DynamoDB

  Amazon DynamoDB는 빠르고 예측 가능한 데이터베이스 성능을 제공하는 확장 가능한 NoSQL 데이터베이스 서비스입니다.

- Amazon Kinesis

  Amazon Kinesis를 사용하면 실시간 스트리밍 데이터를 손쉽게 수집, 처리 및 분석할 수 있으므로 적시에 인사이트를 확보하고 새로운 정보에 신속하게 대응할 수 있습니다. Amazon Kinesis는 기계 학습, 분석 및 기타 애플리케이션을 위한 비디오, 오디오, 애플리케이션 로그, 웹 사이트 클릭스트림 및 IoT 원격 분석 데이터와 같은 실시간 데이터를 수집할 수 있습니다.

- AWS Lambda

  AWS Lambda 를 사용하면 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있습니다. AWS IoT 데이터 및 이벤트에서 자동으로 트리거하거나 웹 또는 모바일 앱에서 직접 호출하도록 코드를 설정할 수 있습니다.

- Amazon Simple Storage Service

  Amazon Simple Storage Service(Amazon S3)는 웹의 어디서나 언제든지 원하는 양의 데이터를 저장하고 검색할 수 있습니다. AWS IoT 규칙은 스토리지를 위해 Amazon S3로 데이터를 전송할 수 있습니다.

- Amazon Simple Notification Service

  Amazon Simple Notification Service(Amazon SNS)는 애플리케이션, 최종 사용자 및 디바이스가 클라우드에서 알림을 전송하고 수신할 수 있게 해 주는 웹 서비스입니다.

- Amazon Simple Queue Service

  Amazon Simple Queue Service(Amazon SQS)는 마이크로서비스, 분산 시스템 및 서버리스 애플리케이션을 분리하고 확장하는 메시지 대기열 서비스입니다.

- Amazon OpenSearch Service

  Amazon OpenSearch Service(OpenSearch Service)는 인기 있는 오픈 소스 검색 및 분석 엔진인 OpenSearch를 손쉽게 배포, 운영, 확장할 수 있는 관리형 서비스입니다.

- Amazon SageMaker AI

  Amazon SageMaker AI는 IoT 데이터에서 패턴을 검색하여 기계 학습(ML) 모델을 생성할 수 있습니다. 이 서비스는 이러한 모델을 사용하여 새 데이터를 처리하고 애플리케이션에 대한 예측을 생성합니다.

- Amazon CloudWatch

  Amazon CloudWatch는 안정적이고 확장 가능하며 유연한 모니터링 솔루션을 제공하여 자체 모니터링 시스템 및 인프라를 설치, 관리 및 확장할 수 있습니다.

에서 지원하는 통신 프로토콜 AWS IoT Core

이 주제에서는 AWS IoT에서 사용하는 통신 프로토콜에 대한 자세한 내용을 제공합니다. 에서 사용하는 프로토콜 AWS IoT 과 디바이스 및 서비스를에 연결하는 방법에 대한 자세한 내용은 섹션을 AWS IoT참조하세요에 연결 AWS IoT Core.

- MQTT(메시지 큐 원격 분석 전송)

  MQTT 프로토콜 사양을 찾을 수 있는 MQTT.org 사이트의 홈 페이지입니다. 가 MQTT를 AWS IoT 지원하는 방법에 대한 자세한 내용은 섹션을 참조하세요MQTT.

- HTTPS(하이퍼텍스트 전송 프로토콜 - 보안)

  디바이스와 앱은 HTTPS를 사용하여 AWS IoT 서비스에 액세스할 수 있습니다.

- LoRaWAN(장거리 광역 네트워크)

  LoRaWAN 디바이스 및 게이트웨이는 AWS IoT Core for LoRaWAN을 사용하여 AWS IoT Core 에 연결할 수 있습니다.

- TLS(Transport Layer Security) v1.3

  TLS v1.3(RFC 5246)의 사양입니다.는 TLS v1.3을 AWS IoT 사용하여 디바이스와 간에 보안 연결을 설정합니다 AWS IoT.

AWS IoT 콘솔의 새로운 기능

AWS IoT 콘솔의 사용자 인터페이스를 새로운 환경으로 업데이트하는 중입니다. 사용자 인터페이스를 단계별로 업데이트하기 때문에 콘솔의 일부 페이지에는 새로운 환경이 제공되고, 일부는 원래 환경과 새 환경을 모두 가질 수 있으며, 일부는 원래 환경만 가질 수 있습니다.

2022년 1월 27일부터 이 테이블에는 AWS IoT 콘솔 사용자 인터페이스의 개별 영역의 상태가 표시됩니다.

AWS IoT 콘솔 사용자 인터페이스 상태

콘솔 페이지	원래 환경	새로운 환경	설명
모니터링	사용할 수 없음	사용 가능	
활동	사용할 수 없음	사용 가능	
온보딩 - 시작하기	사용할 수 없음	사용 가능	CN 지역에서는 사용할 수 없음
온보딩 - 플릿 프로비저닝 템플릿	사용 가능	Available	

콘솔 페이지	원래 환경	새로운 환경	설명
관리 - 사물	사용 가능	Available	
관리 - 종류	사용 가능	Available	
관리 - 사물 그룹	사용 가능	Available	
관리 - 결제 그룹	사용 가능	Available	
관리 - 작업	사용 가능	Available	
관리 - 작업 템플릿	사용할 수 없음	사용 가능	
관리 - 터널	사용할 수 없음	사용 가능	
플릿 허브 - 시작하기	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
플릿 허브 - 애플리케이션	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
Greengrass - 시작하기	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
Greengrass - 코어 디바이스	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
Greengrass - 구성 요소	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
Greengrass - 배포	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
Greengrass - 클래식 (V1)	사용 가능	Available	
무선 연결 - 소개	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음

콘솔 페이지	원래 환경	새로운 환경	설명
무선 연결 - 게이트웨이	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
무선 연결 - 디바이스	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
무선 연결 - 프로파일	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
무선 연결 - 대상	사용할 수 없음	사용 가능	일부 AWS 리전에서는 사용할 수 없음
보안 - 인증서	사용 가능	Available	
보안 - 정책	사용 가능	Available	
보안 - CA	사용 가능	Available	
보안 - 역할 별칭	사용 가능	Available	
보안 - 권한 부여자	사용 가능	Available	
방어 - 소개	사용할 수 없음	사용 가능	
방어 - 감사	사용할 수 없음	사용 가능	
방어 - 감지	사용할 수 없음	사용 가능	
방어 - 완화 작업	사용할 수 없음	사용 가능	
방어 - 설정	사용할 수 없음	사용 가능	
활동(Act) - 규칙	사용 가능	Available	
활동(Act) - 대상	사용 가능	Available	
테스트 - Device Advisor	사용 가능	Available	일부 AWS 리전에서는 사용할 수 없음

콘솔 페이지	원래 환경	새로운 환경	설명
테스트 - MQTT 테스트 클라이언트	사용 가능	Available	
소프트웨어.	사용 가능	Available	
설정:	사용할 수 없음	사용 가능	
자세히 알아보기	Available	아직 사용할 수 없음	

범례

상태 값

- Available

  이 사용자 인터페이스 환경을 사용할 수 있습니다.
- 사용할 수 없음

  이 사용자 인터페이스 환경을 사용할 수 없습니다.
- 아직 사용할 수 없음

  새로운 사용자 인터페이스 환경이 작업 중이지만 아직 준비되지 않았습니다.
- 진행 중

  새로운 사용자 인터페이스 환경이 업데이트되는 중입니다. 그러나 일부 페이지에는 여전히 원래 사용자 환경이 있을 수 있습니다.

AWS SDK AWS IoT 에서 사용

AWS 소프트웨어 개발 키트(SDKs)는 널리 사용되는 여러 프로그래밍 언어에 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예제 및 설명서를 제공합니다.

SDK 설명서	코드 예제
AWS SDK for C++	AWS SDK for C++ 코드 예제
AWS CLI	AWS CLI 코드 예제
AWS SDK for Go	AWS SDK for Go 코드 예제
AWS SDK for Java	AWS SDK for Java 코드 예제
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예제
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예제
AWS SDK for .NET	AWS SDK for .NET 코드 예제
AWS SDK for PHP	AWS SDK for PHP 코드 예제
AWS Tools for PowerShell	AWS Tools for PowerShell 코드 예제
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예제
AWS SDK for Ruby	AWS SDK for Ruby 코드 예제
AWS SDK for Rust	AWS SDK for Rust 코드 예제
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예제
AWS SDK for Swift	AWS SDK for Swift 코드 예제

예제 가용성

필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

AWS IoT Core 자습서 시작하기

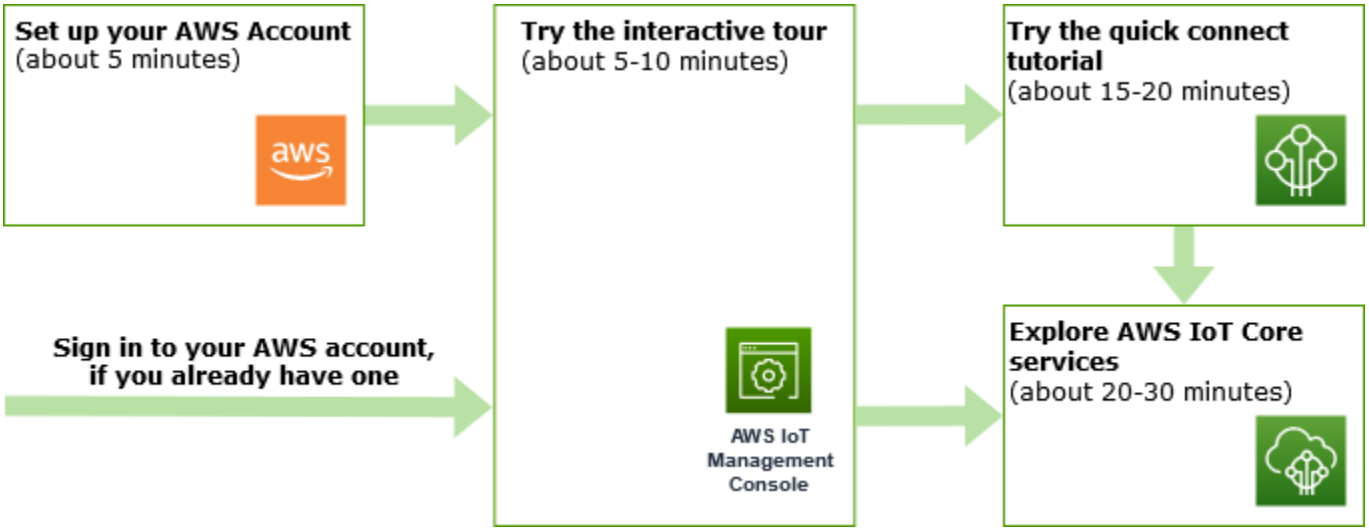
IoT를 처음 사용하든 다년간의 경험이 있든 이러한 리소스는 사용을 시작하는 데 도움이 되는 AWS IoT 개념과 용어를 제공합니다 AWS IoT.

- 의 내부 AWS IoT 및 해당 구성 요소를 살펴봅니다AWS IoT 작동 방식.
- 교육 자료 및 비디오 컬렉션에서 AWS IoT에 대해 자세히 알아보기 이 항목에는 AWS IoT 이(가) 연결할 수 있는 서비스 목록, 소셜 미디어 링크 및 통신 프로토콜 사양에 대한 링크도 포함됩니다.
- the section called “첫 번째 디바이스를에 연결 AWS IoT Core”.
- 에 연결 AWS IoT Core 및 AWS IoT 자습서 탐색을 통해 IoT 솔루션을 개발합니다.
- 디바이스 어드바이저을(를) 사용하여 안전하고 안정적인 통신을 위해 IoT 디바이스를 테스트 및 검증합니다.
- 플릿 인덱싱, AWS IoT 작업 및 AWS IoT Device Defender와 같은 AWS IoT Core 관리 서비스를 사용하여 솔루션을 관리합니다.
- AWS IoT 데이터 서비스을(를) 사용하여 디바이스에서 데이터를 분석합니다.

첫 번째 디바이스를에 연결 AWS IoT Core

AWS IoT Core 서비스는 IoT 디바이스를 AWS IoT 서비스 및 기타 AWS 서비스에 연결합니다. AWS IoT Core 에는 IoT 디바이스와 클라우드 간에 메시지를 연결하고 처리하는 디바이스 게이트웨이와 메시지 브로커가 포함되어 있습니다.

다음은 AWS IoT Core 및를 시작하는 방법입니다 AWS IoT.



이 섹션에서는 주요 서비스를 소개 AWS IoT Core 하기 위한 둘러보기를 제공하고 디바이스를 AWS IoT Core 에 연결하고 디바이스 간에 메시지를 전달하는 방법에 대한 몇 가지 예를 제공합니다. 디바이스와 클라우드 간에 메시지를 전달하는 것은 모든 IoT 솔루션의 기본이며 디바이스가 다른 AWS 서비스와 상호 작용하는 방법입니다.

- 설정 AWS 계정

  AWS IoT 서비스를 사용하려면 먼저를 설정해야 합니다 AWS 계정. AWS 계정 및 IAM 사용자가 이미 있는 경우 해당 사용자를 사용하고이 단계를 건너뛸 수 있습니다.

- 빠른 연결 자습서 사용해 보기

  이 자습서는를 빠르게 시작하고 제한된 시나리오에서 작동 방식을 AWS IoT 확인하려는 경우에 가장 적합합니다. 이 자습서에서는 디바이스가 필요하며 디바이스에 일부 AWS IoT 소프트웨어를 설치합니다. IoT 디바이스가 없는 경우 Windows, Linux 또는 macOS 개인용 컴퓨터를 이 자습서의 디바이스로 사용할 수 있습니다. 시도하고 싶지 AWS IoT만 디바이스가 없는 경우 다음 옵션을 시도하세요.

- 대화형 자습서 사용하기

  이 데모는 디바이스를 연결하거나 소프트웨어를 다운로드하지 않고도 기본 AWS IoT 솔루션이 수행할 수 있는 작업을 확인하려는 경우에 가장 적합합니다. 대화형 자습서에서는 상호 작용하는 방식을 보여주는 AWS IoT Core 서비스에 구축된 시뮬레이션된 솔루션을 제공합니다.

- 실습 자습서를 사용하여 AWS IoT Core 서비스 살펴보기

  이 자습서는 규칙 엔진 및 섀도우와 같은 다른 AWS IoT Core 기능을 계속 탐색할 수 AWS IoT 있도록를 시작하려는 개발자에게 가장 적합합니다. 이 자습서에서는 빠른 연결 자습서와 유사한 프로세스를 따르지만 고급 자습서로 더 원활하게 전환할 수 있도록 각 단계에 대한 자세한 정보를 제공합니다.

- MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기

  MQTT 테스트 클라이언트를 사용하여 첫 번째 디바이스가 AWS IoT에 MQTT 메시지를 게시하는 것을 보는 방법을 알아봅니다. MQTT 테스트 클라이언트는 디바이스 연결을 모니터링하고 문제를 해결하는 데 유용한 도구입니다.

> **Note**
>
> 이러한 시작 자습서 중 하나 이상을 시도하거나 동일한 자습서를 반복하려면 다른 자습서를 시작하기 전에 이전 자습서에서 만든 사물 객체를 삭제해야 합니다. 이전 자습서에서 사물 객체

를 삭제하지 않은 경우 후속 자습서에서는 다른 이름을 사용해야 합니다. 이는 사물 이름이 계정과 AWS 리전에 고유해야 하기 때문입니다.

에 대한 자세한 내용은 란 무엇입니까 AWS IoT Core?를 AWS IoT Core참조하십시오.

설정 AWS 계정

AWS IoT Core 를 처음 사용하기 전에 다음 작업을 완료합니다.

주제

- 에 가입 AWS 계정
- 관리자 액세스 권한이 있는 사용자 생성
- AWS IoT 콘솔 열기

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

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

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

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

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

관리자 액세스 권한이 있는 사용자 생성

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

보안 AWS 계정 루트 사용자

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

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

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

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

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

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

관리 액세스 권한이 있는 사용자로 로그인

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

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

2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 그룹 추가를 참조하세요.

- AWS IoT 콘솔 열기

AWS 계정 및 사용자가 이미 있는 경우 이를 사용하고 로 건너뛸 수 있습니다the section called “AWS IoT 콘솔 열기”.

AWS IoT 콘솔 열기

이 섹션의 콘솔 지향 주제는 대부분 AWS IoT 콘솔에서 시작됩니다. 아직에 로그인하지 않은 경우 AWS 계정로그인한 후 AWS IoT 콘솔을 열고 다음 섹션으로 계속 진행합니다 AWS IoT.

대화형 자습서

이 대화형 자습서에서는 AWS IoT에 빌드된 간단한 IoT 솔루션의 구성 요소를 보여줍니다. 이 자습서에서는 IoT 디바이스가 AWS IoT Core 서비스와 상호 작용하는 방식을 보여줍니다. 이 주제에서는 AWS IoT Core 대화형 자습서의 미리 보기를 제공합니다.

> **Note**
>
> 콘솔의 이미지에는 이 자습서의 이미지에 나타나지 않는 애니메이션이 포함되어 있습니다.

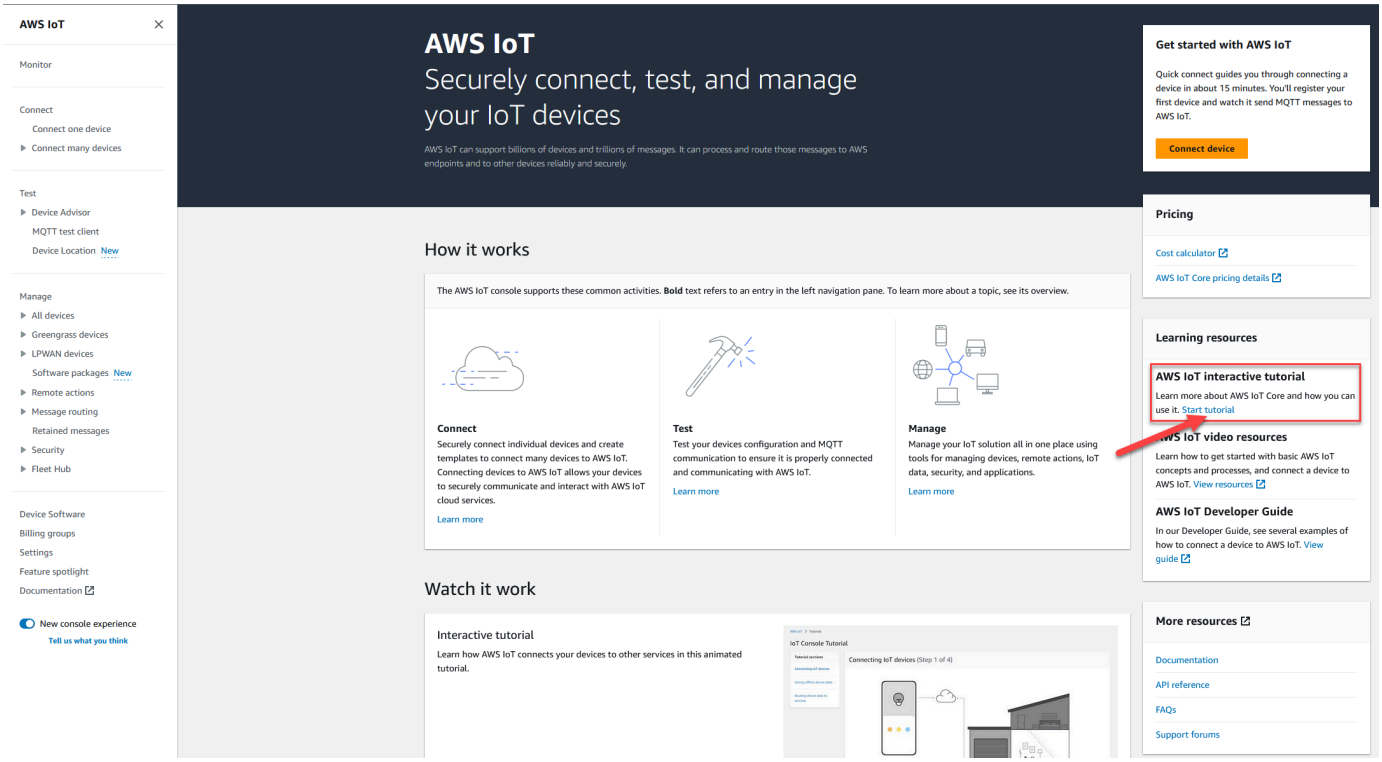
데모를 실행하려면 먼저 the section called “설정 AWS 계정”. 그러나이 자습서에는 AWS IoT 리소스, 추가 소프트웨어 또는 코딩이 필요하지 않습니다.

이 데모의 예상 소요 시간은 약 5~10분입니다. 10분을 할애하면 각 단계를 차근차근 이해할 수 있습니다.

AWS IoT Core 대화형 자습서를 실행하려면

1. AWS IoT 콘솔에서 AWS IoT 홈 페이지를 엽니다.

   AWS IoT 홈 페이지의 학습 리소스 창에서 자습서 시작을 선택합니다.



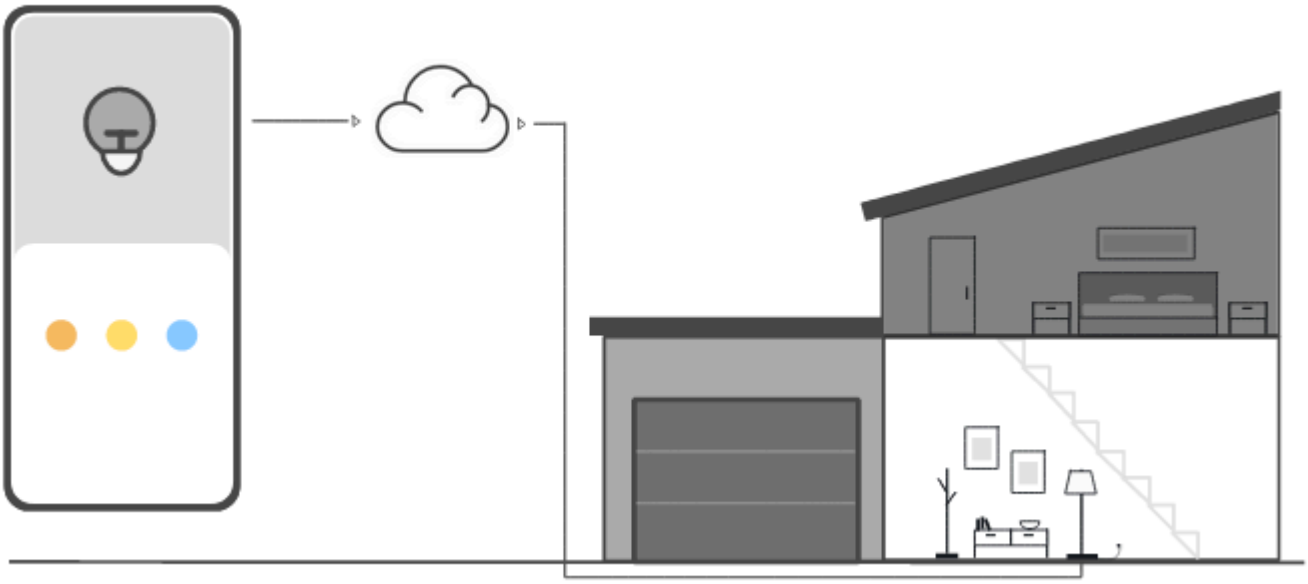
2. AWS IoT 콘솔 자습서 페이지에서 자습서 섹션을 검토하고 계속할 준비가 되면 섹션 시작을 선택합니다.

다음 섹션에서는 AWS IoT 콘솔 자습서에서 이러한 AWS IoT Core 기능을 제공하는 방법을 설명합니다.

- IoT 디바이스 연결
- 오프라인 디바이스 상태 저장
- 디바이스 데이터를 서비스로 라우팅

IoT 디바이스 연결

IoT 디바이스가와 통신하는 방법을 알아봅니다 AWS IoT Core.

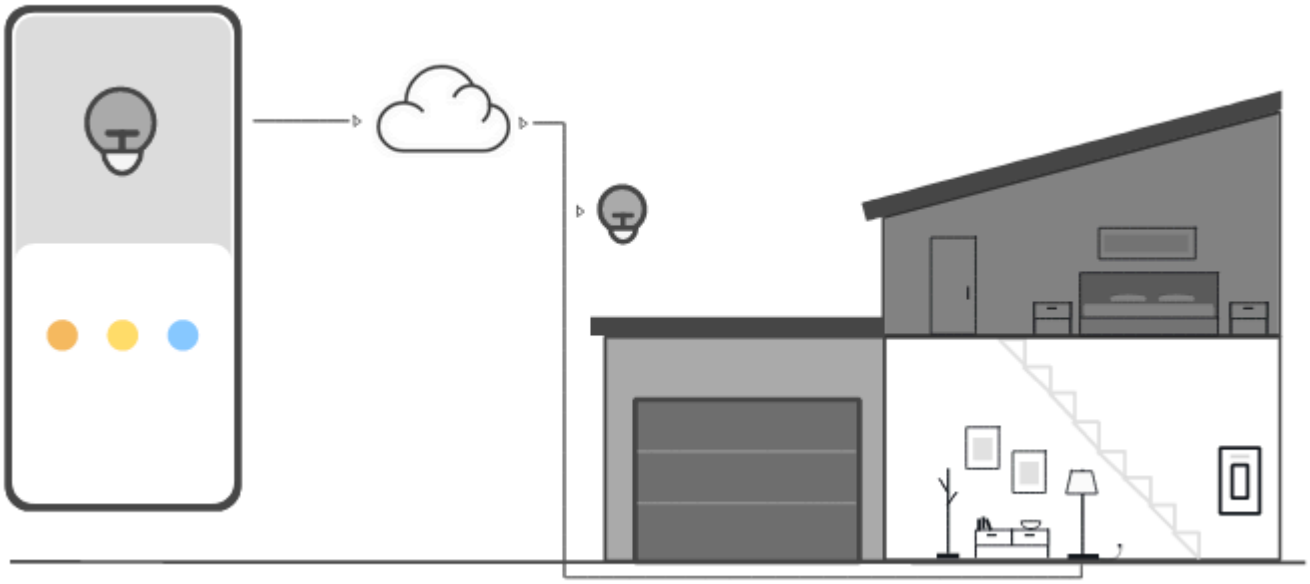


이 단계의 애니메이션은 왼쪽의 제어 디바이스와 오른쪽에 있는 집안의 스마트 램프가 어떻게 연결되고 클라우드에서 AWS IoT Core 와 통신하는지 보여줍니다. 애니메이션은 디바이스가 수신하는 메시지와 통신 AWS IoT Core 하고 대응하는 것을 보여줍니다.

디바이스를에 연결하는 방법에 대한 자세한 내용은 섹션을 AWS IoT Core참조하세요에 연결 AWS IoT Core.

오프라인 디바이스 상태 저장

디바이스 또는 앱이 오프라인 상태일 때가에 대한 디바이스 상태를 AWS IoT Core 저장하는 방법을 알아봅니다.



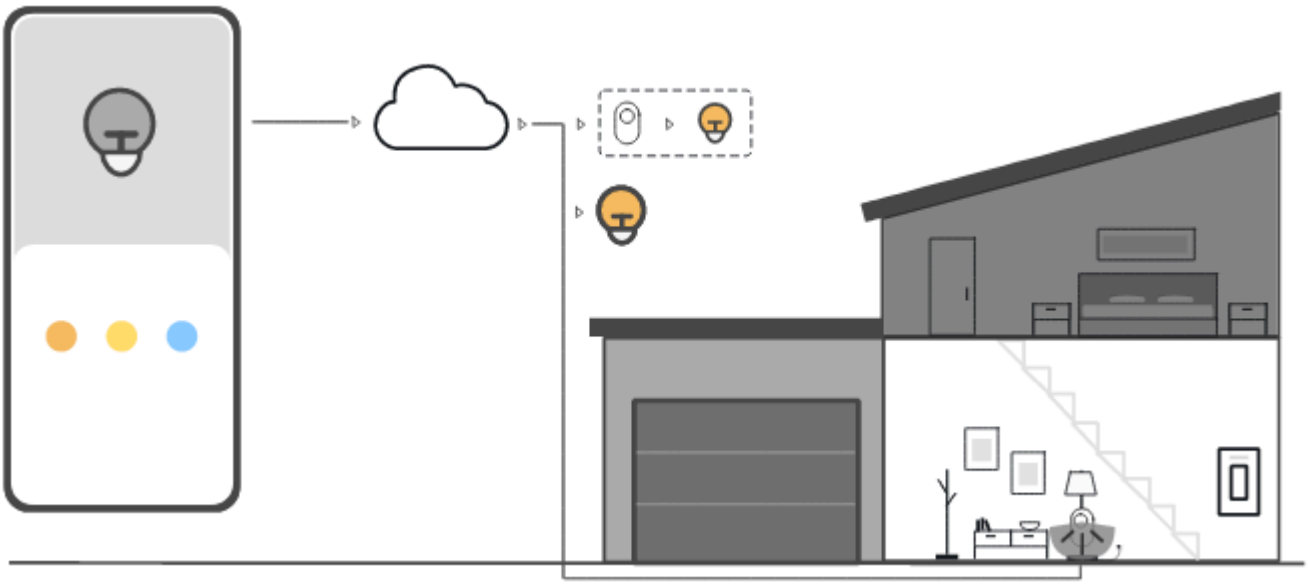
이 단계의 애니메이션은의 디바이스 섀도우 서비스가 제어 디바이스 및 스마트 램프에 대한 디바이스 상태 정보를 AWS IoT Core 저장하는 방법을 보여줍니다. 스마트 램프가 오프라인 상태이면 디바이스 섀도우는 제어 디바이스의 명령을 저장합니다.

스마트 램프가 다시 연결되면 해당 명령을 AWS IoT Core검색합니다. 제어 디바이스가 오프라인 상태이면 디바이스 섀도우는 스마트 램프의 명령을 저장합니다. 제어 디바이스가 다시 연결되면 스마트 램프의 현재 상태를 검색하여 디스플레이를 업데이트합니다.

디바이스 섀도우에 관한 자세한 내용은 AWS IoT 디바이스 섀도우 서비스 섹션을 참조하세요.

디바이스 데이터를 서비스로 라우팅

가 디바이스 상태를 다른 AWS 서비스로 AWS IoT Core 전송하는 방법을 알아봅니다.



이 단계의 애니메이션은가 AWS IoT rules. AWS IoT rules를 사용하여 디바이스의 데이터를 다른 AWS 서비스로 AWS IoT Core 보내는 방법을 보여줍니다. 규칙은 디바이스의 특정 메시지를 구독하고, 해당 메시지의 데이터를 해석하고, 해석된 데이터를 다른 서비스로 라우팅합니다. 이 예제에서 AWS IoT 규칙은 모션 센서의 데이터를 해석하고 디바이스 섀도우로 명령을 전송한 다음 스마트 전구로 전송합니다. 앞의 예와 마찬가지로 디바이스 섀도우는 제어 디바이스에 대한 디바이스 상태 정보를 저장합니다.

AWS IoT 규칙에 대한 자세한 내용은 섹션을 참조하세요에 대한 규칙 AWS IoT.

AWS IoT Core 빠른 연결 자습서 사용

이 자습서에서는 첫 번째 사물 객체를 만들고 해당 객체에 디바이스를 연결한 다음 MQTT 메시지를 전송하는 것을 살펴봅니다.

이 자습서의 예상 소요 시간은 15~20분입니다.

이 자습서는 제한된 시나리오에서 작동 방식을 AWS IoT 빠르게 확인하려는 사용자에게 가장 적합합니다. 더 많은 기능과 서비스를 탐색할 수 있도록 시작하는 데 도움이 되는 예제를 찾고 있다면 실습 자습서 AWS IoT Core 에서 살펴보기을(를) 사용해 보세요.

이 자습서에서는 매우 작은 IoT 솔루션의 AWS IoT Core 일부로의 사물 리소스에 연결하는 디바이스에서 소프트웨어를 다운로드하고 실행합니다. 이 디바이스는 Raspberry Pi와 같은 IoT 디바이스일 수도 있고 Linux, OS 및 OSX 또는 Windows를 실행하는 컴퓨터일 수도 있습니다. 장거리 WAN(LoRaWAN) 디바이스를 연결하려는 경우 자습서 > for LoRaWAN에 디바이스 및 게이트웨이 연결을 AWS IoT참조하세요. AWS IoT Core LoRaWAN

디바이스가 AWS IoT 콘솔을 실행할 수 있는 브라우저를 지원하는 경우 해당 디바이스에서 이 자습서를 완료하는 것이 좋습니다.

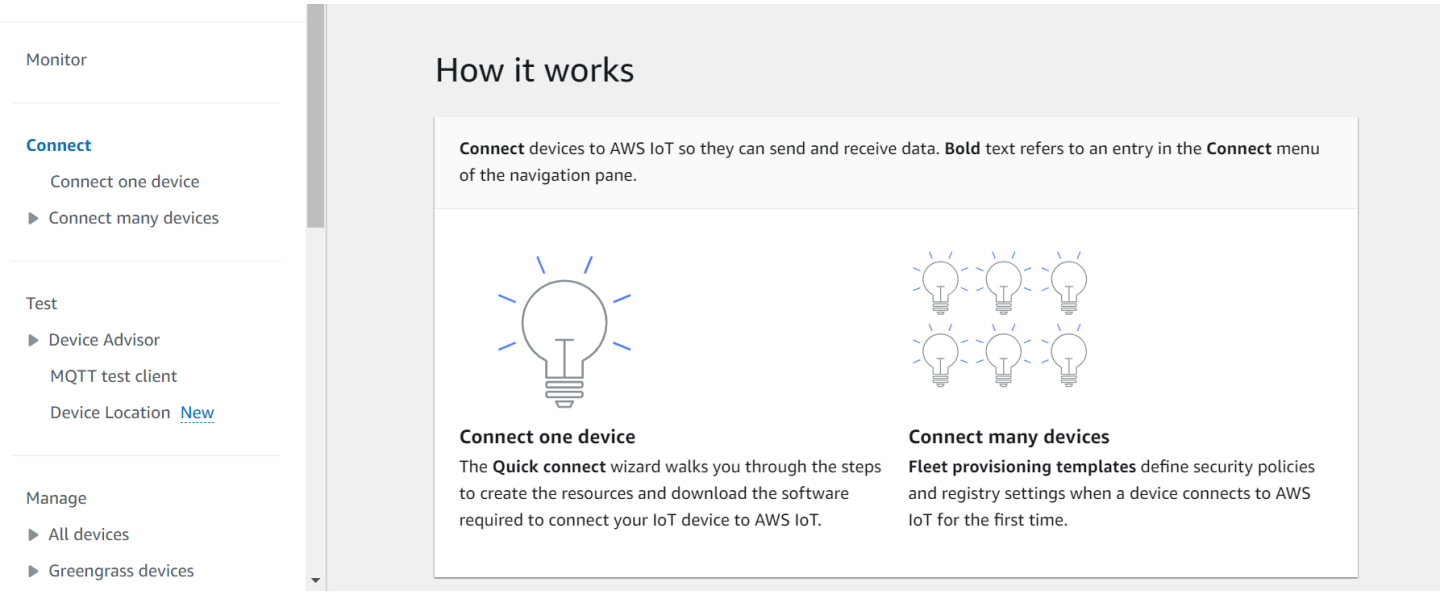
> **Note**
>
> 디바이스에 호환되는 브라우저가 없는 경우 컴퓨터에서 이 자습서를 따르세요. 절차에서 파일을 다운로드하라는 메시지가 나타나면 컴퓨터에 다운로드한 다음 SCP(Secure Copy) 또는 유사한 프로세스를 사용하여 다운로드한 파일을 디바이스로 전송합니다.

이 자습서에서는 IoT 디바이스가 사용자의 AWS 계정디바이스 엔드포인트의 포트 8443과 통신해야 합니다. 해당 포트에 액세스할 수 있는지 테스트하려면 디바이스 데이터 엔드포인트와의 연결 테스트의 절차를 따르세요.

1단계. 자습서 시작

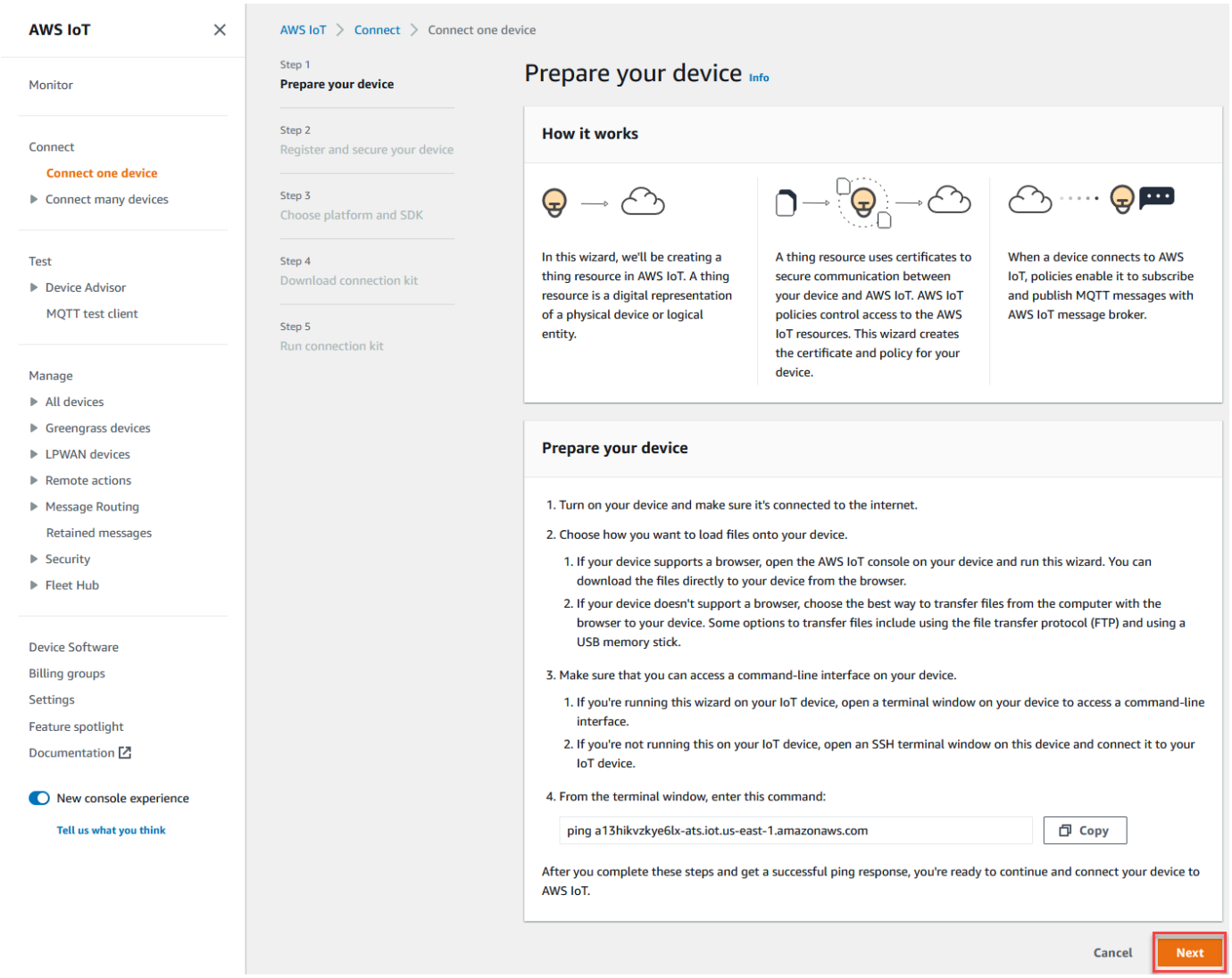
가능하면 디바이스에서 이 절차를 완료하세요. 그렇지 않은 경우 이 절차의 뒷부분에서 디바이스로 파일을 전송할 준비를 하세요.

자습서를 시작하려면 AWS IoT 콘솔에 로그인합니다. AWS IoT 콘솔 홈 페이지의 왼쪽에서 연결을 선택한 다음 디바이스 하나를 연결을 선택합니다.



2단계. 사물 객체 만들기

1. 디바이스 준비 섹션에서는 화면의 지침에 따라 AWS IoT에 연결할 디바이스를 준비합니다.

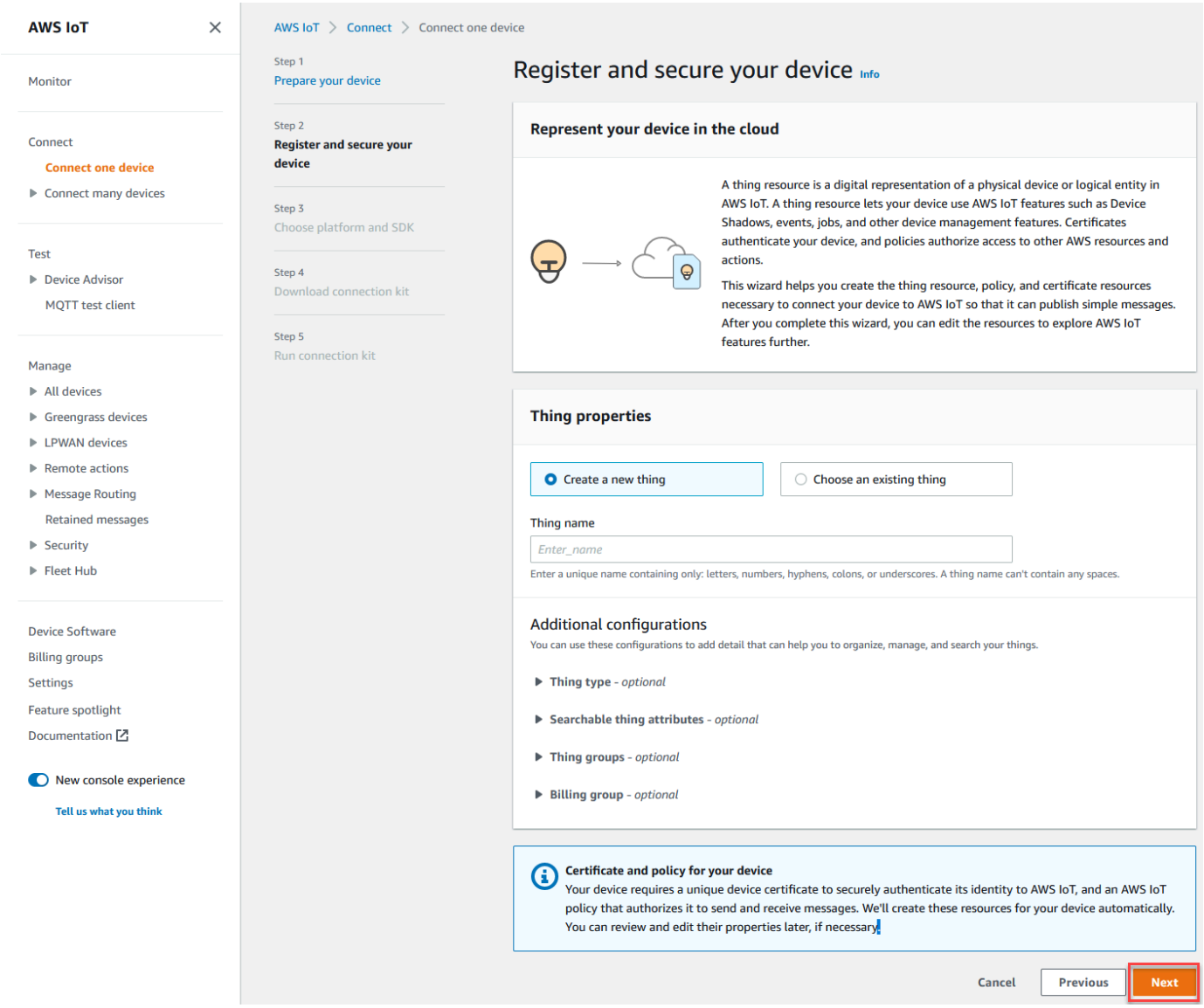


2. 디바이스 등록 및 보안 섹션에서는 새로운 사물 생성 또는 기존 사물 선택을 선택합니다. 사물 이름 필드에서는 사물 객체의 이름을 입력합니다. 이 예제에서 사용된 사물 이름은 **`TutorialTestThing`**입니다.

> **Important**
>
> 계속하기 전에 사물 이름을 다시 확인하세요.
> 사물 객체가 생성된 후에는 사물 이름을 변경할 수 없습니다. 사물 이름을 변경하려면 올바른 사물 이름을 가진 새 사물 객체를 생성한 다음 잘못된 이름의 사물 객체를 삭제해야 합니다.

추가 구성 섹션에서 나열된 선택적 구성을 사용하여 사물 리소스를 추가로 사용자 지정합니다.

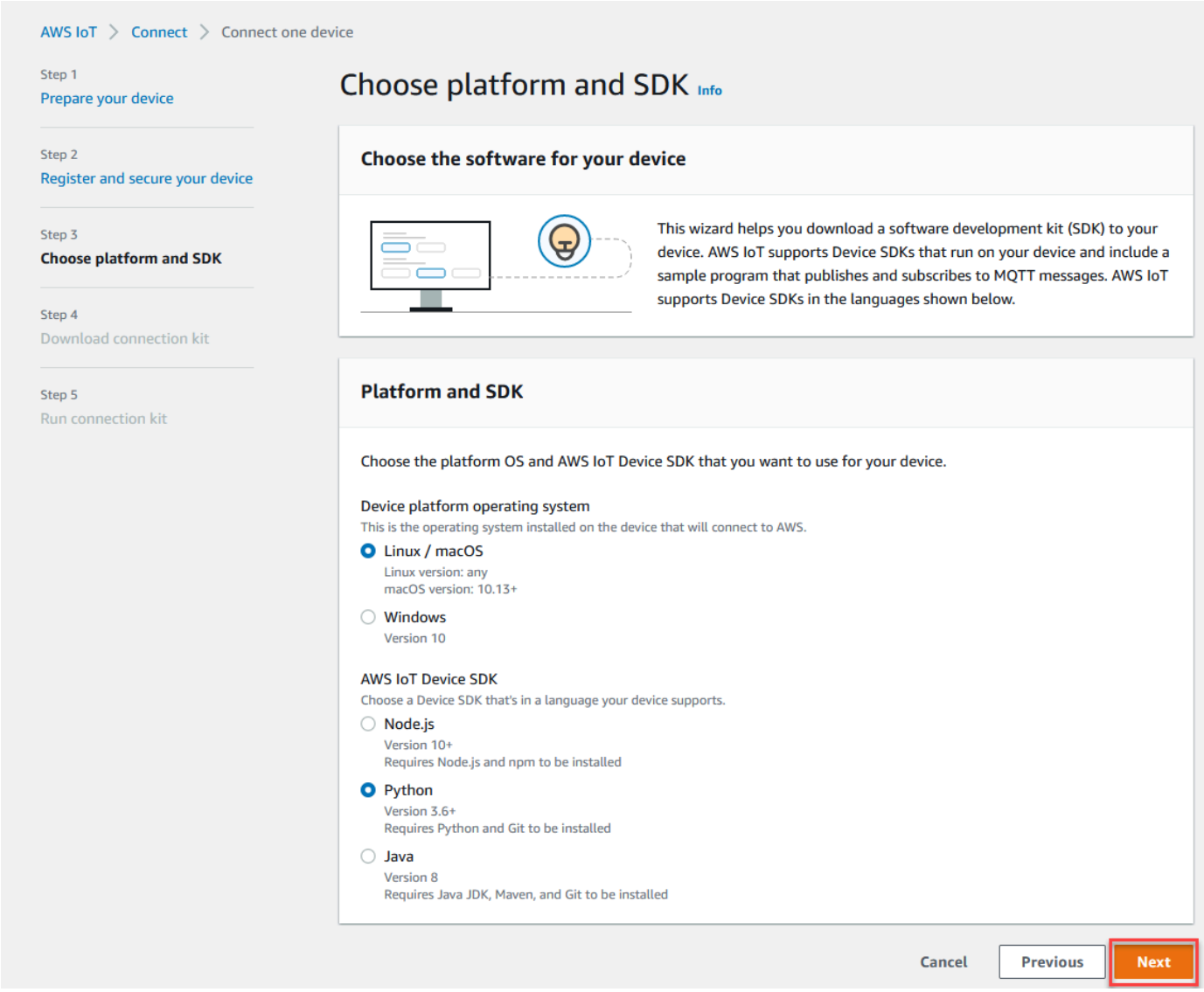
사물 객체 이름을 입력하고 추가 구성을 선택한 후 다음을 선택합니다.



3. 플랫폼 및 SDK 선택 섹션에서 사용할 AWS IoT 디바이스 SDK의 플랫폼과 언어를 선택합니다. 이 예제에서는 Linux/OSX 플랫폼과 Python SDK를 사용합니다. 다음 단계를 계속하기 전에 대상 디바이스에 python3 및 pip3이 설치되어 있는지 확인합니다.

> **Note**
>
> 콘솔 페이지 하단에서 선택한 SDK에 필요한 필수 소프트웨어 목록을 확인하세요.
> 다음 단계를 진행하기 전에 대상 컴퓨터에 필수 소프트웨어가 설치되어 있어야 합니다.

플랫폼 및 디바이스 SDK 언어를 선택한 후 다음을 선택합니다.

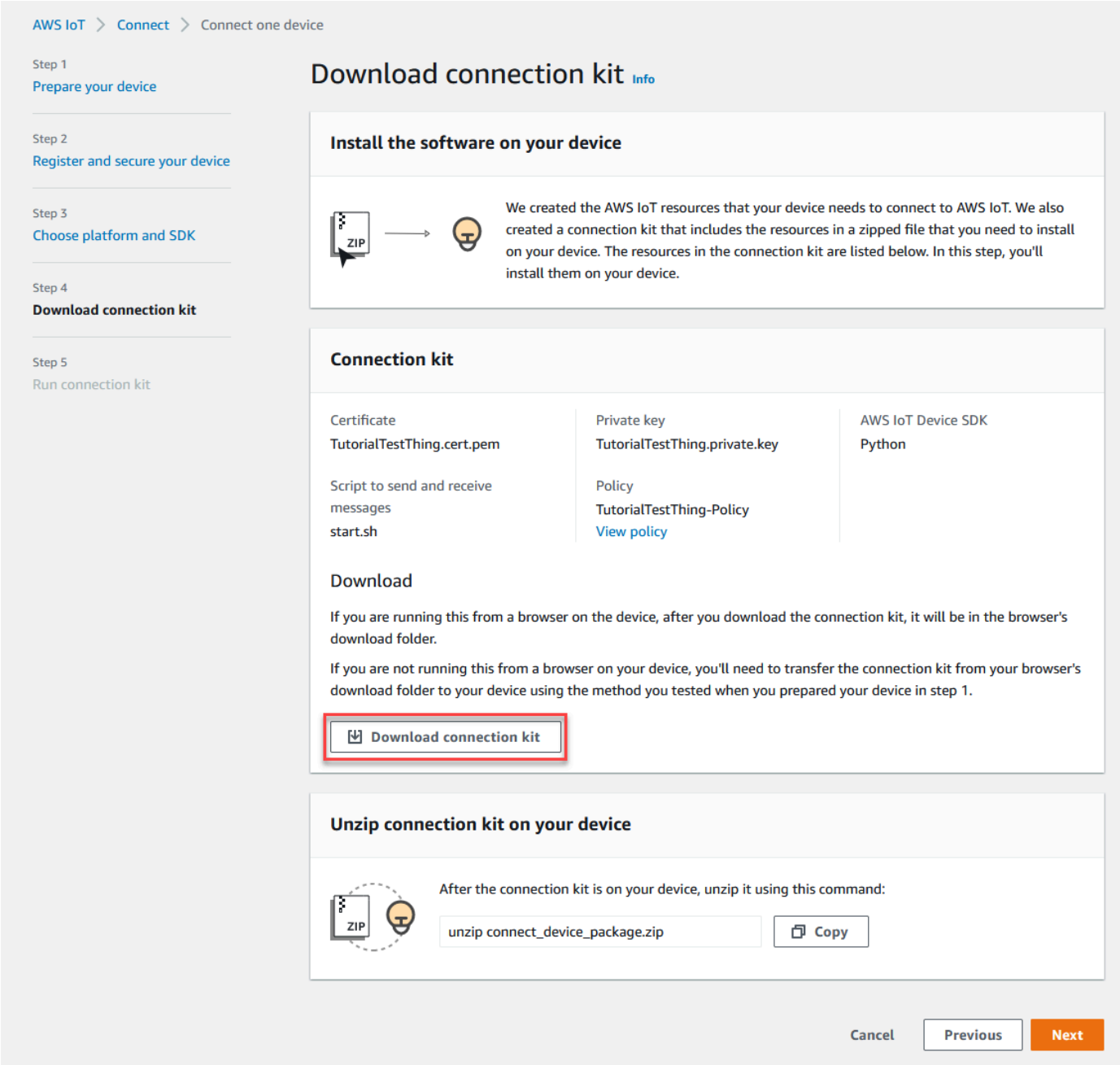


3단계. 디바이스에 파일 다운로드

이 페이지 AWS IoT 는가 디바이스에 필요한 다음 파일과 리소스를 포함하는 연결 키트를 생성한 후에 나타납니다.

- 디바이스를 인증하는 데 사용되는 사물의 인증서 파일
- 사물 객체가 AWS IoT와(과) 상호 작용할 수 있도록 권한을 부여하는 정책 리소스
- AWS 디바이스 SDK를 다운로드하고 디바이스에서 샘플 프로그램을 실행하는 스크립트

1. 계속 진행할 준비가 되면 연결 키트 다운로드 버튼을 선택하여 이전에 선택한 플랫폼에 대한 연결 키트를 다운로드합니다.



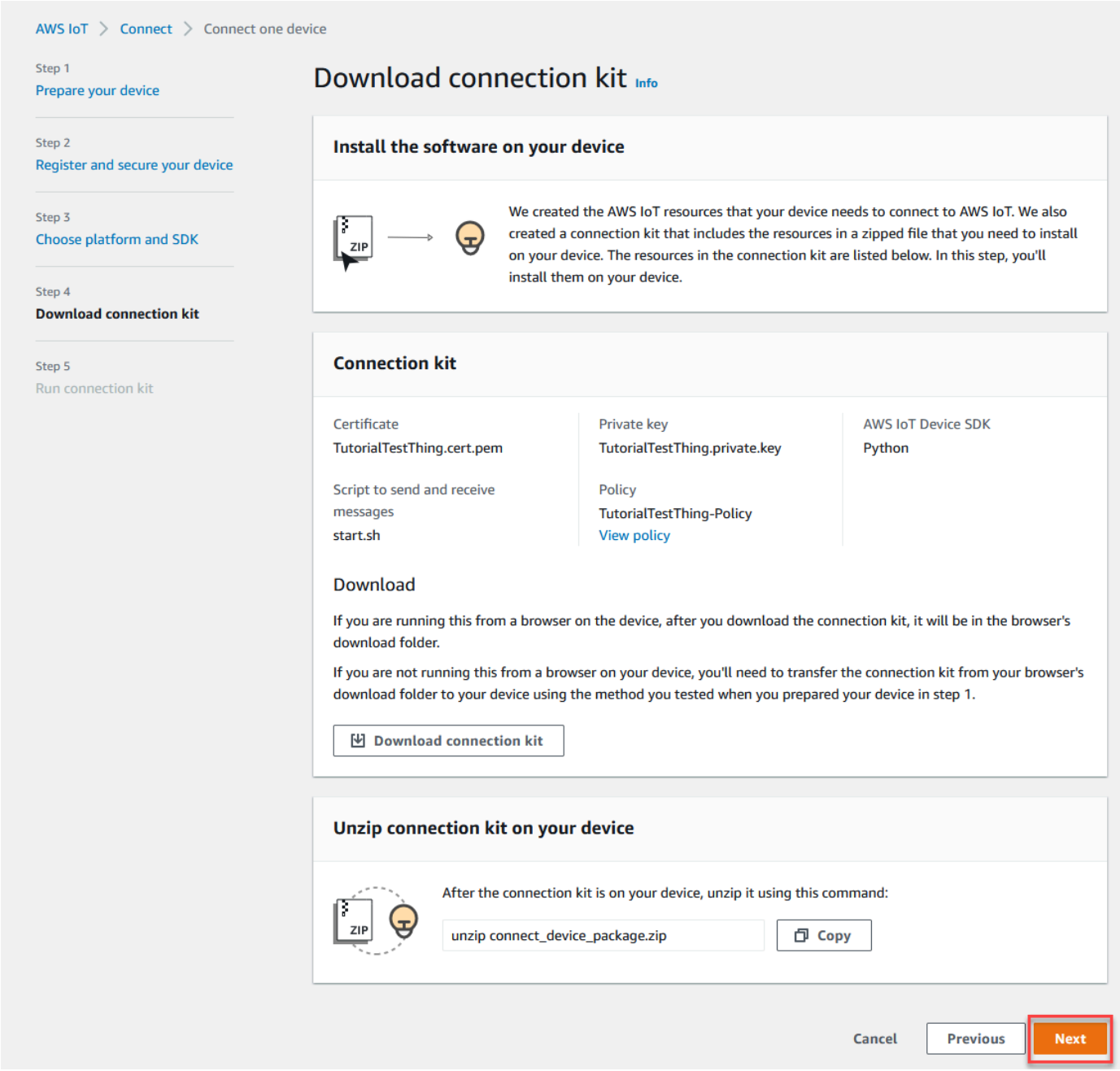
2. 디바이스에서 이 절차를 실행하는 경우 명령줄 명령을 실행할 수 있는 디렉터리에 연결 키트 파일을 저장합니다.

   디바이스에서 이 절차를 실행하지 않는 경우 연결 키트 파일을 로컬 디렉터리에 저장한 다음 해당 파일을 디바이스로 전송합니다.

3. 디바이스의 연결 키트 압축 풀기 섹션에서 연결 키트 파일이 있는 디렉터리에 unzip connect_device_package.zip을 입력합니다.

Windows PowerShell 명령 창을 사용하고 있고 unzip 명령이 작동하지 않으면 unzip을 expand-archive로 바꾸고 명령줄을 다시 시도하세요.

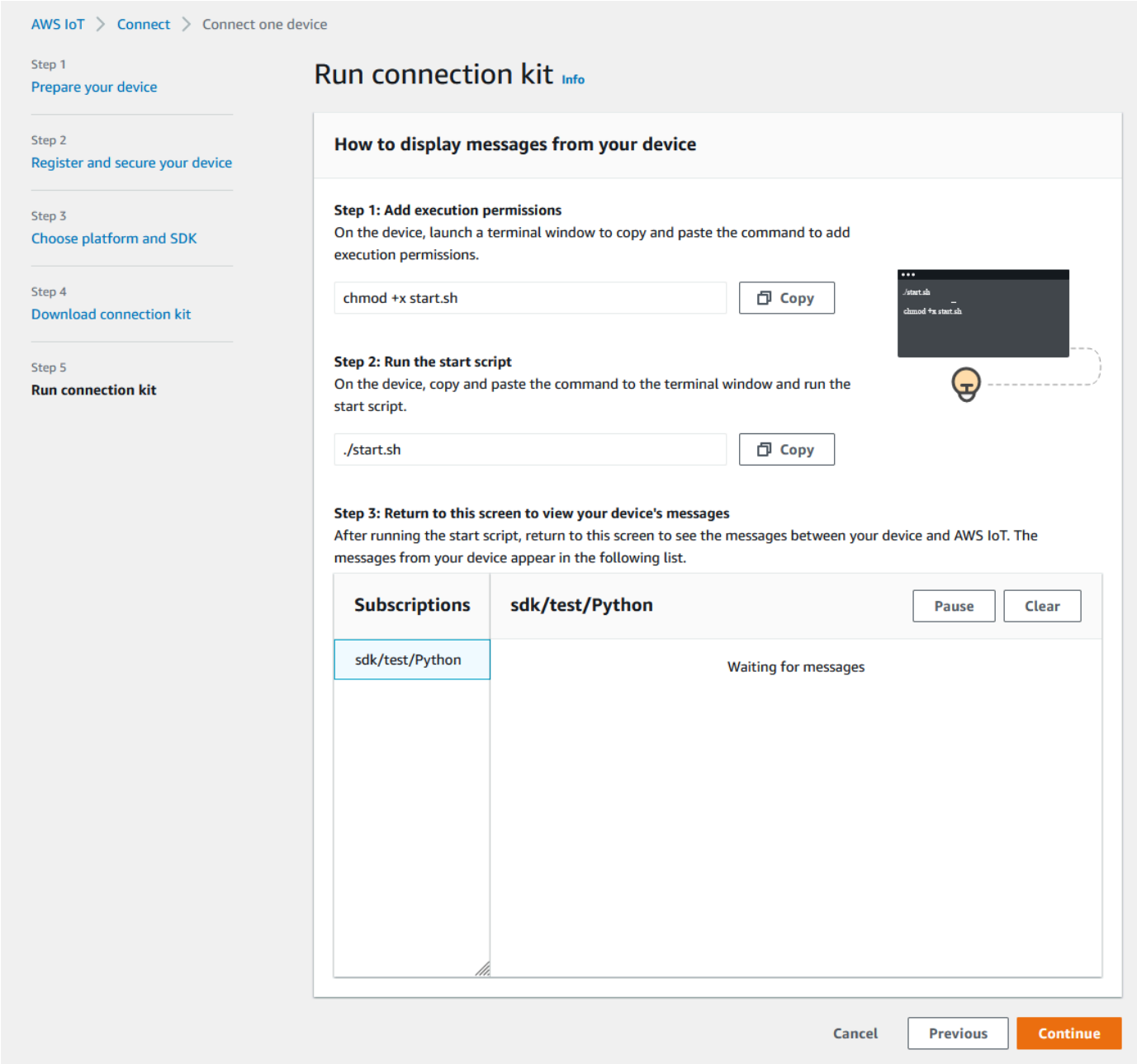
4. 디바이스에 연결 키트 파일이 있으면 다음을 선택하여 자습서를 계속합니다.



4단계. 샘플 실행

이 절차는 콘솔에 표시된 지침에 따르면서 디바이스의 터미널 또는 명령 창에서 수행합니다. 콘솔에 표시된 명령은 the section called “2단계. 사물 객체 만들기”에서 선택한 운영 체제에 대한 것입니다. 여기에 표시된 내용은 Linux/OSX 운영 체제용입니다.

1. 디바이스의 터미널 또는 명령 창의 연결 키트 파일이 있는 디렉터리에서 AWS IoT 콘솔에 표시된 단계를 수행합니다.



2. 콘솔에서 2단계의 명령을 입력하면 디바이스의 터미널 또는 명령 창에 다음과 유사한 출력이 표시되어야 합니다. 이 출력은 프로그램이 AWS IoT Core에 전송한 후 그로부터 다시 수신하는 메시지에서 나온 것입니다.

```
Running pub/sub sample application...
Connecting to a13hikvzkye6lx-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT_LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'"Hello World! [1]"'
Publishing message to topic 'sdk/test/Python': Hello World! [2]
Received message from topic 'sdk/test/Python': b'"Hello World! [2]"'
Publishing message to topic 'sdk/test/Python': Hello World! [3]
Received message from topic 'sdk/test/Python': b'"Hello World! [3]"'
```

   샘플 프로그램이 실행되는 동안 테스트 메시지 `Hello World!`도 함께 나타납니다. 테스트 메시지가 디바이스의 터미널 또는 명령 창에 나타납니다.

   > **Note**
   >
   > 주제 구독 및 게시에 대한 자세한 내용은 선택한 SDK의 예제 코드를 참조하세요.

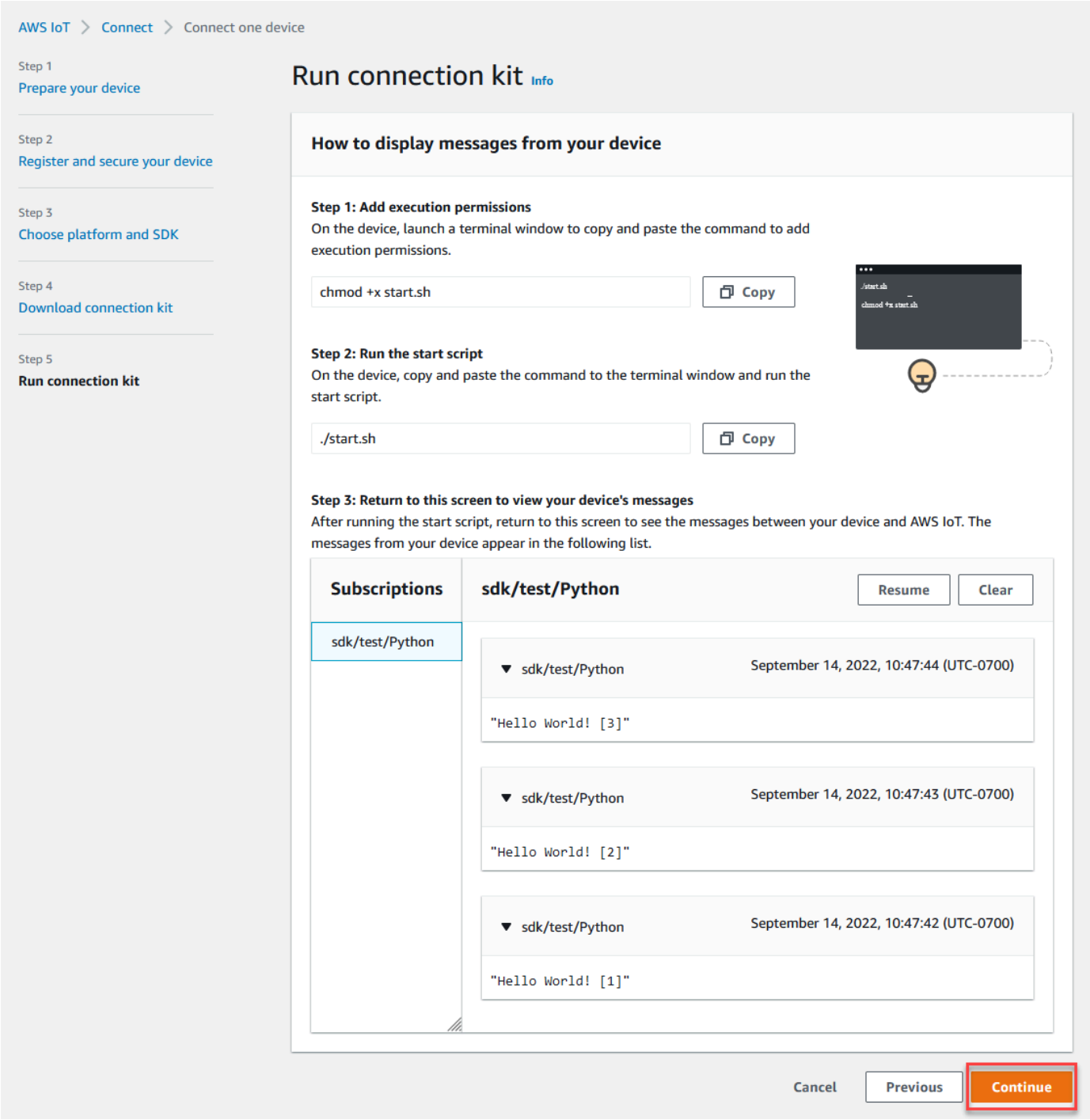
3. 샘플 프로그램을 다시 실행하려면 이 절차의 콘솔에서 2단계의 명령을 반복할 수 있습니다.
4. (선택 사항) AWS IoT 콘솔에서 IoT 클라이언트의 메시지를 보려면 AWS IoT 콘솔의 테스트 페이지에서 MQTT 테스트 클라이언트를 엽니다. Python SDK를 선택한 경우 MQTT 테스트 클라이언트(MQTT test client)의 주제 필터(Topic filter)에 **`sdk/test/python`**를 입력하여 디바이스의 메시지를 구독합니다. 주제 필터는 대/소문자를 구분하며 1단계에서 선택한 SDK의 프로그래밍 언어에 따라 달라집니다. 주제 구독 및 게시에 대한 자세한 내용은 선택한 SDK의 코드 예제를 참조하세요.
5. 테스트 주제를 구독한 후 디바이스에서 ./start.sh를 실행합니다. 자세한 내용은 the section called "MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기" 단원을 참조하십시오.

   ./start.sh 실행 후 MQTT 클라이언트에 다음과 유사한 메시지가 나타납니다.

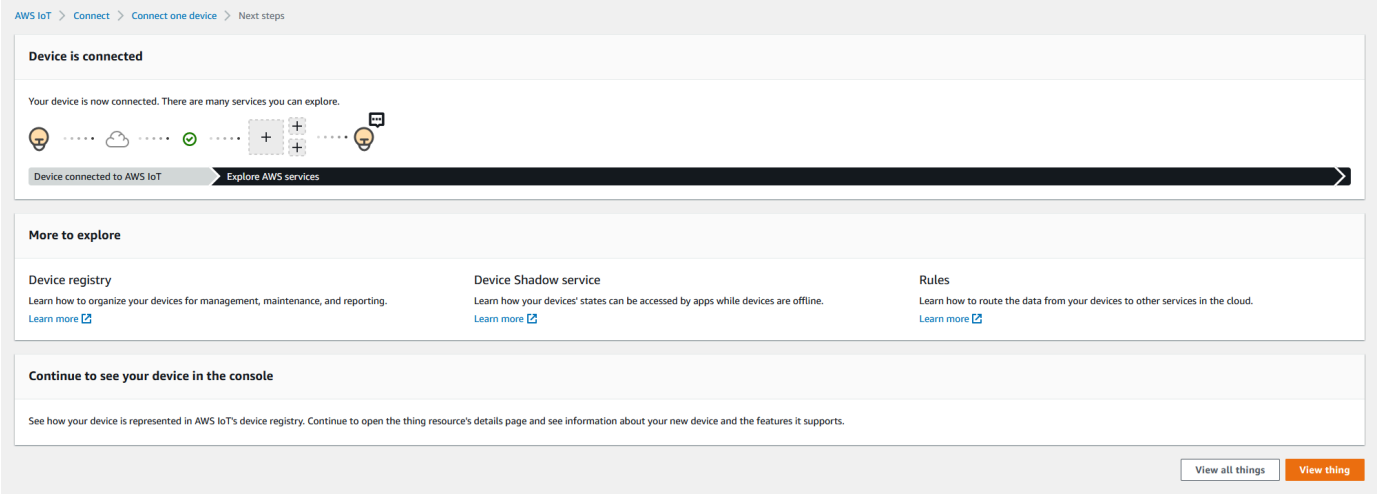
```
{
  "message": "Hello World!" [1]
}
```

   `[]` 안의 `sequence` 숫자는 새 `Hello World!` 메시지가 수신될 때마다 1씩 증가하고 프로그램을 종료하면 중지됩니다.

6. 자습서를 완료하고 요약을 보려면 AWS IoT 콘솔에서 계속을 선택합니다.



7. 이제 AWS IoT 빠른 연결 자습서의 요약이 표시됩니다.



5단계. 더 살펴보기

다음은 빠른 시작을 완료한 후 AWS IoT 더 자세히 살펴볼 몇 가지 아이디어입니다.

- MQTT 테스트 클라이언트에서 MQTT 메시지 보기

  AWS IoT 콘솔을 사용하는 경우 AWS IoT 콘솔의 테스트 페이지에서 MQTT 클라이언트를 열 수 있습니다. MQTT test client(MQTT 테스트 클라이언트)에서 #를 구독한 다음 디바이스를 사용하여 이전 단계에서 설명한 대로 프로그램 ./start.sh를 실행합니다. 자세한 내용은 the section called “MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기” 단원을 참조하십시오.

- Device Advisor를 사용하여 디바이스에서 테스트 실행

  Device Advisor를 사용하여 디바이스가에 안전하고 안정적으로 연결하고 상호 작용할 수 있는지 테스트합니다 AWS IoT.

- the section called “대화형 자습서”

  대화형 자습서를 시작하려면 AWS IoT 콘솔의 학습 페이지의 작동 방식 AWS IoT 보기 타일에서 자습서 시작을 선택합니다.

- 더 많은 자습서를 살펴볼 준비하기

  이 빠른 시작은 샘플만 제공합니다 AWS IoT. AWS IoT 더 자세히 살펴보고 이를 강력한 IoT 솔루션 플랫폼으로 만드는 기능에 대해 알아보려면에서 개발 플랫폼 준비를 시작하세요실습 자습서 AWS IoT Core 에서 살펴보기.

디바이스 데이터 엔드포인트와의 연결 테스트

이 주제에서는 사용자의 IoT 디바이스가 AWS IoT와의 연결에 사용하는 계정의 디바이스 데이터 엔드포인트와 디바이스 연결을 테스트하는 방법에 대해 설명합니다.

테스트하려는 디바이스에서 또는 테스트하려는 디바이스에 연결된 SSH 터미널 세션을 사용하여 이러한 절차를 수행합니다.

디바이스 데이터 엔드포인트와의 연결 테스트.

- 디바이스 데이터 엔드포인트 찾기
- 신속하게 연결 테스트
- 앱을 통해 디바이스 데이터 엔드포인트 및 포트에 대한 연결을 테스트합니다.
- 디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트

디바이스 데이터 엔드포인트 찾기

이 절차에서는 IoT 디바이스에 대한 연결을 테스트하기 위해 AWS IoT 콘솔에서 디바이스 데이터 엔드포인트를 찾는 방법을 설명합니다.

디바이스 데이터 엔드포인트 찾기

1. AWS IoT 콘솔의 연결 섹션에서 도메인 구성으로 이동합니다.
2. 도메인 구성 페이지에서 도메인 구성 컨테이너로 이동하여 도메인 이름을 복사합니다. 엔드포인트 값은에 고유 AWS 계정 하며 다음 예와 유사합니다`a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com`.
3. 다음 절차에 따라 디바이스 데이터 엔드포인트를 저장합니다.

신속하게 연결 테스트

이 절차는 디바이스 데이터 엔드포인트와의 일반적인 연결을 테스트하지만, 디바이스에서 사용할 특정 포트를 테스트하지는 않습니다. 이 테스트는 일반적인 프로그램을 사용하며 디바이스가 AWS IoT에 연결할 수 있는지 여부를 확인하는 데 대부분 충분합니다.

디바이스에서 사용할 특정 포트와의 연결을 테스트하려면 이 절차를 건너뛰고 앱을 통해 디바이스 데이터 엔드포인트 및 포트에 대한 연결을 테스트합니다. 섹션부터 계속 진행합니다.

디바이스 데이터 엔드포인트를 빠르게 테스트하기

1. 디바이스의 터미널 또는 명령줄 창에서 샘플 디바이스 데이터 엔드포인트(*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*)를 계정의 디바이스 데이터 엔드포인트로 교체하고 다음 명령을 입력합니다.

   Linux

   ```
   ping -c 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

   Windows

   ```
   ping -n 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

2. ping에서 다음과 유사한 출력이 표시되면 디바이스 데이터 엔드포인트에 연결된 것입니다. 와 AWS IoT 직접 통신하지는 않았지만 서버를 찾았으며이 엔드포인트를 통해 AWS IoT 를 사용할 수 있을 것입니다.

   ```
   PING a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (xx.xx.xxx.xxx) 56(84) bytes of
    data.
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=1 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=2 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=3 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=4 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=5 ttl=231 time=127 ms
   ```

   이 결과에 만족하는 경우 여기에서 테스트를 중단할 수 있습니다.

   AWS IoT에서 사용되는 특정 포트와의 연결을 테스트하려는 경우 앱을 통해 디바이스 데이터 엔드포인트 및 포트에 대한 연결을 테스트합니다. 섹션으로 계속 진행합니다.

3. ping이 성공적인 출력을 반환하지 않은 경우 엔드포인트 값을 확인하여 올바른 엔드포인트가 있는지 확인하고 디바이스의 인터넷 연결을 확인합니다.

앱을 통해 디바이스 데이터 엔드포인트 및 포트에 대한 연결을 테스트합니다.

nmap을 사용하여 보다 철저한 연결 테스트를 수행할 수 있습니다. 이 절차는 디바이스에 nmap이 설치되어 있는지 테스트합니다.

디바이스 상의 **nmap** 확인

1. 테스트할 디바이스의 터미널 또는 명령줄 창에서 이 명령을 입력하여 nmap이 설치되어 있는지 확인합니다.

   ```
   nmap --version
   ```

2. 다음과 유사한 출력이 표시되면 nmap이 설치되어 있다는 뜻이며 the section called “디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트” 섹션으로 계속 진행할 수 있습니다.

   ```
   Nmap version 6.40 ( http://nmap.org )
   Platform: x86_64-koji-linux-gnu
   Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
   libdnet-1.12 ipv6
   Compiled without:
   Available nsock engines: epoll poll select
   ```

3. 이전 단계에서 설명한 것과 비슷한 응답이 표시되지 않으면 디바이스에 nmap을 설치해야 합니다. 디바이스의 운영 체제에 맞는 절차를 선택합니다.

Linux

이 절차를 수행하려면 컴퓨터에 소프트웨어를 설치할 권한이 있어야 합니다.

Linux 컴퓨터에 nmap 설치

1. 디바이스의 터미널 또는 명령줄 창에서 실행 중인 Linux 버전에 해당하는 명령을 입력합니다.

   a. Debian 또는 Ubuntu의 경우:

      ```
      sudo apt install nmap
      ```

   b. CentOS 또는 RHEL의 경우:

      ```
      sudo yum install nmap
      ```

2. 다음 명령을 사용하여 설치를 테스트합니다.

```
nmap --version
```

3. 다음과 유사한 출력이 표시되면 nmap이 설치되어 있다는 뜻이며 the section called “디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트” 섹션으로 계속 진행할 수 있습니다.

```
Nmap version 6.40 ( http://nmap.org )
Platform: x86_64-koji-linux-gnu
Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

macOS

이 절차를 수행하려면 컴퓨터에 소프트웨어를 설치할 권한이 있어야 합니다.

macOS 컴퓨터에 nmap 설치

1. 브라우저에서 https://nmap.org/download#macosx를 열고 안정적인 최신 설치 프로그램을 다운로드합니다.

   메시지가 표시되면 DiskImageInstaller으로 열기(Open with DiskImageInstaller)를 선택합니다.
2. 설치 창에서 패키지를 애플리케이션(Applications) 폴더로 이동합니다.
3. 파인더에서 애플리케이션 폴더 내의 nmap-xxxx-mpkg 패키지를 찾습니다. 패키지를 Ctrl-click한 다음 열기를 선택하여 패키지를 엽니다.
4. 보안 대화 상자를 검토합니다. nmap을 설치할 준비가 되었으면 열기(Open)를 선택하여 nmap을 설치합니다.
5. Terminal에서 다음 명령을 사용하여 설치를 테스트합니다.

```
nmap --version
```

6. 다음과 유사한 출력이 표시되면 nmap이 설치되어 있다는 뜻이며 the section called “디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트” 섹션으로 계속 진행할 수 있습니다.

```
Nmap version 7.92 ( https://nmap.org )
Platform: x86_64-apple-darwin17.7.0
```

```
Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 libz-1.2.11
 nmap-libpcre-7.6 nmap-libpcap-1.9.1 nmap-libdnet-1.12 ipv6 Compiled without:
Available nsock engines: kqueue poll select
```

Windows

이 절차를 수행하려면 컴퓨터에 소프트웨어를 설치할 권한이 있어야 합니다.

Windows 컴퓨터에 nmap 설치

1. 브라우저에서 https://nmap.org/download#windows를 열고 설정 프로그램의 안정적인 최신 릴리스를 다운로드합니다.

   메시지가 표시되면 파일 저장(Save file)을 선택합니다. 파일을 다운로드한 후 다운로드 폴더에서 엽니다.
2. 설정 파일이 다운로드를 완료하면 다운로드된 nmap-xxxx-setup.exe를 열고 앱을 설치합니다.
3. 프로그램이 설치될 때 기본 설정을 적용합니다.

   이 테스트에는 Npcap 앱이 필요하지 않습니다. 설치하지 않으려면 이 옵션을 선택 해제하면 됩니다.
4. Command에서 다음 명령을 사용하여 설치를 테스트합니다.

   ```
   nmap --version
   ```
5. 다음과 유사한 출력이 표시되면 nmap이 설치되어 있다는 뜻이며 the section called “디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트” 섹션으로 계속 진행할 수 있습니다.

   ```
   Nmap version 7.92 ( https://nmap.org )
   Platform: i686-pc-windows-windows
   Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 nmap-
   libz-1.2.11 nmap-libpcre-7.6 Npcap-1.50 nmap-libdnet-1.12 ipv6
   Compiled without:
   Available nsock engines: iocp poll select
   ```

디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트

이 절차에서는 선택한 포트를 사용하여 디바이스 데이터 엔드포인트에 대한 IoT 디바이스의 연결을 테스트합니다.

디바이스 데이터 엔드포인트 및 포트에 대한 연결 테스트

1. 디바이스의 터미널 또는 명령줄 창에서 샘플 디바이스 데이터 엔드포인트(*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*)를 계정의 디바이스 데이터 엔드포인트로 교체하고 다음 명령을 입력합니다.

   ```
   nmap -p 8443 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

2. nmap에서 다음과 유사한 출력이 표시되면 nmap이 선택된 포트의 디바이스 데이터 엔드포인트에 연결된 것입니다.

   ```
   Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-18 16:23 Pacific Standard Time
   Nmap scan report for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
    (xx.xxx.147.160)
   Host is up (0.036s latency).
   Other addresses for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (not scanned):
    xx.xxx.134.144 xx.xxx.55.139 xx.xxx.110.235 xx.xxx.174.233 xx.xxx.74.65
    xx.xxx.122.179 xx.xxx.127.126
   rDNS record for xx.xxx.147.160: ec2-EXAMPLE-160.eu-west-1.compute.amazonaws.com

   PORT     STATE SERVICE
   8443/tcp open  https-alt
   MAC Address: 00:11:22:33:44:55 (Cimsys)

   Nmap done: 1 IP address (1 host up) scanned in 0.91 seconds
   ```

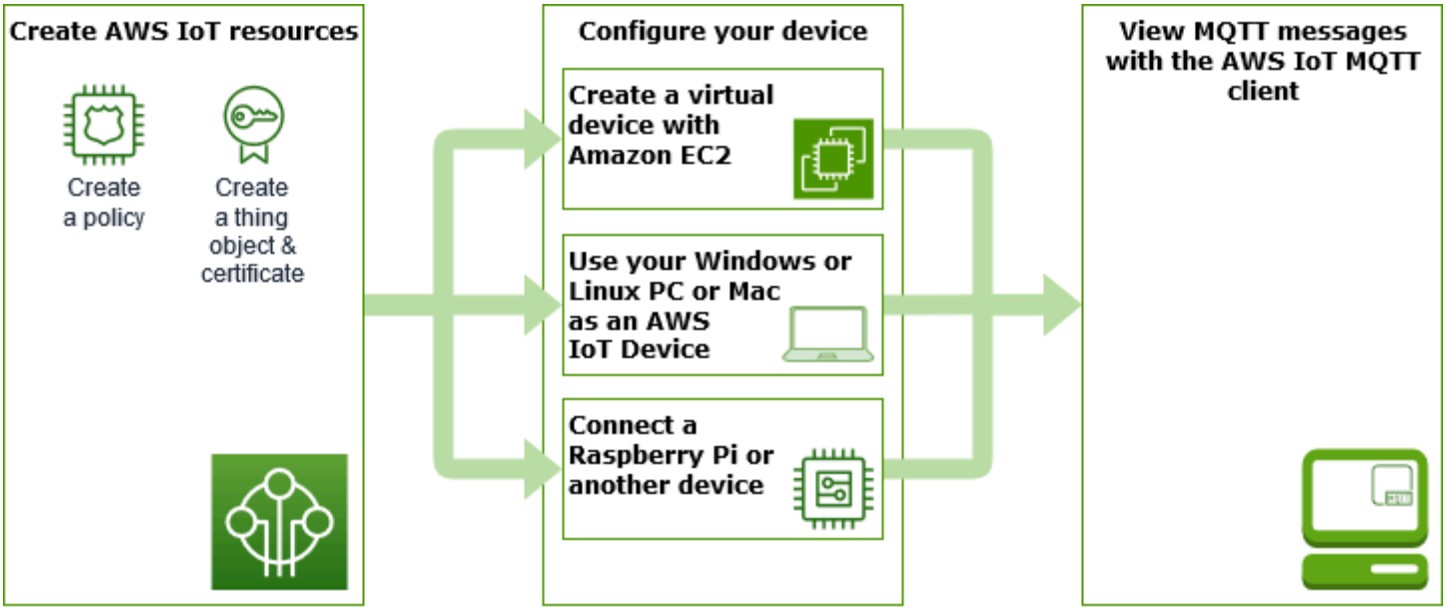
3. nmap이 성공적인 출력을 반환하지 않은 경우 엔드포인트 값을 확인하여 올바른 엔드포인트가 있는지 확인하고 디바이스의 인터넷 연결을 확인합니다.

1단계에서 사용된 포트 **8443**을 테스트하려는 포트로 교체하여 기본 HTTPS 포트인 포트 443과 같은 디바이스 데이터 엔드포인트의 다른 포트를 테스트할 수 있습니다.

실습 자습서 AWS IoT Core 에서 살펴보기

이 자습서에서는를 사용하여 MQTT 메시지를 보내고 받을 수 AWS IoT Core 있도록 소프트웨어를 설치하고 디바이스를에 연결하는 데 필요한 AWS IoT 리소스를 생성합니다 AWS IoT Core. AWS IoT 콘솔의 MQTT 클라이언트에 메시지가 표시됩니다.

이 자습서의 예상 소요 시간은 20~30분입니다. IoT 디바이스 또는 Raspberry Pi를 사용하는 경우 예를 들어 운영 체제를 설치하고 디바이스를 구성해야 한다면 이 자습서가 더 오래 걸릴 수 있습니다.



이 자습서는 규칙 엔진 및 섀도우와 같은 고급 기능을 계속 탐색할 수 AWS IoT Core 있도록를 시작하려는 개발자에게 가장 적합합니다. 이 자습서에서는 빠른 시작 자습서보다 단계를 더 자세히 설명하여에 대해 AWS IoT Core 계속 알아보고 다른 AWS 서비스와 상호 작용하는 방법을 배울 수 있도록 준비합니다. ??? 빠른 Hello World 경험을 찾고 있다면 AWS IoT Core 빠른 연결 자습서 사용을(를) 사용해 보세요.

AWS 계정 및 AWS IoT 콘솔을 설정한 후 다음 단계에 따라 디바이스를 연결하고 메시지를 보내도록 하는 방법을 확인합니다 AWS IoT Core.

다음 단계

- 가장 적합한 디바이스 옵션 선택
- Amazon EC2로 가상 디바이스를 생성하지 않으려는 경우, the section called “AWS IoT 리소스 생성”
- the section called “디바이스 구성”
- the section called “MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기”

에 대한 자세한 내용은 란 무엇입니까 AWS IoT Core?를 AWS IoT Core참조하십시오.

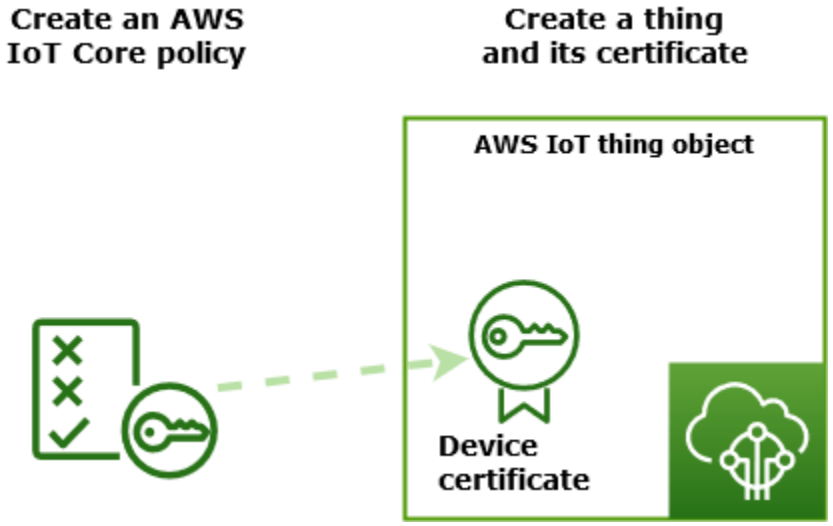
어떤 디바이스 옵션이 가장 적합한가요?

어떤 옵션을 선택할지 잘 모르는 경우 다음 각 옵션의 장점과 단점 목록을 사용하여 자신에게 가장 적합한 옵션을 결정할 수 있습니다.

옵션	다음과 같은 경우에 좋은 옵션일 수 있습니다.	다음과 같은 경우에는 좋은 옵션이 아닐 수 있습니다.
the section called “Amazon EC2를 사용하여 가상 디바이스 생성”	• 테스트할 자체 디바이스가 없습니다. • 자체 시스템에 소프트웨어를 설치하고 싶지 않습니다. • Linux OS에서 테스트하고 싶습니다.	• 명령줄 명령 사용이 익숙하지 않습니다. • 추가 AWS 요금을 발생시키고 싶지 않습니다. • Linux OS에서 테스트하고 싶지 않습니다.
the section called “Windows 또는 Linux PC 또는 Mac을 AWS IoT 디바이스로 사용”	• 추가 AWS 요금을 발생시키고 싶지 않습니다. • 추가 디바이스를 구성하고 싶지 않습니다.	• 개인용 컴퓨터에 소프트웨어를 설치하고 싶지 않습니다. • 좀 더 대표적인 테스트 플랫폼을 원합니다.
the section called “Raspberry Pi 또는 다른 디바이스 연결”	• 실제 디바이스 AWS IoT 로 테스트하려고 합니다. • 테스트할 디바이스가 이미 있습니다. • 하드웨어를 시스템에 통합한 경험이 있습니다.	• 디바이스를 단지 시험해보기 위해 구입하거나 구성하고 싶지 않습니다. • 지금은 가능한 한 AWS IoT 간단하게 테스트하고 싶습니다.

AWS IoT 리소스 생성

이 자습서에서는 디바이스가 메시지에 연결하고 메시지를 AWS IoT Core 교환하는 데 필요한 AWS IoT 리소스를 생성합니다.



1. 디바이스가 AWS IoT 서비스와 상호 작용할 수 있는 권한을 부여하는 AWS IoT 정책 문서를 생성합니다.
2. AWS IoT 및 해당 X.509 디바이스 인증서에서 사물 객체를 생성한 다음 정책 문서를 연결합니다. 사물 객체는 AWS IoT 레지스트리에 있는 디바이스의 가상 표현입니다. 인증서는에 디바이스를 인증 AWS IoT Core하고 정책 문서는 디바이스가 상호 작용할 수 있는 권한을 부여합니다 AWS IoT.

> **Note**
>
> the section called “Amazon EC2를 사용하여 가상 디바이스 생성”을(를) 계획 중인 경우 이 페이지를 건너뛰고 the section called “디바이스 구성”(으)로 이동할 수 있습니다. 가상 사물을 만들 때 이러한 리소스를 만듭니다.

이 자습서에서는 AWS IoT 콘솔을 사용하여 AWS IoT 리소스를 생성합니다. 디바이스에서 웹 브라우저를 지원하는 경우 인증서 파일을 디바이스에 직접 다운로드할 수 있으므로 디바이스의 웹 브라우저에서 이 절차를 실행하는 것이 더 쉬울 수 있습니다. 다른 컴퓨터에서 이 절차를 실행하는 경우 인증서 파일을 디바이스에 복사해야 샘플 앱에서 사용할 수 있습니다.

AWS IoT 정책 생성

디바이스는 X.509 인증서를 사용하여 인증합니다 AWS IoT Core. 인증서에는 AWS IoT 정책이 연결되어 있습니다. 이러한 정책은 MQTT 주제 구독 또는 게시와 같은 어떤 AWS IoT 작업을 디바이스가 수행할 수 있는지를 결정합니다. 디바이스가 연결하고 메시지를 보낼 때 인증서를 제공합니다 AWS IoT Core.

단계를 따라서 디바이스가 예제 프로그램을 실행하는 데 필요한 AWS IoT 작업을 수행하도록 허용하는 정책을 생성합니다. 나중에 생성할 디바이스 인증서에 연결할 수 있도록 먼저 AWS IoT 정책을 생성해야 합니다.

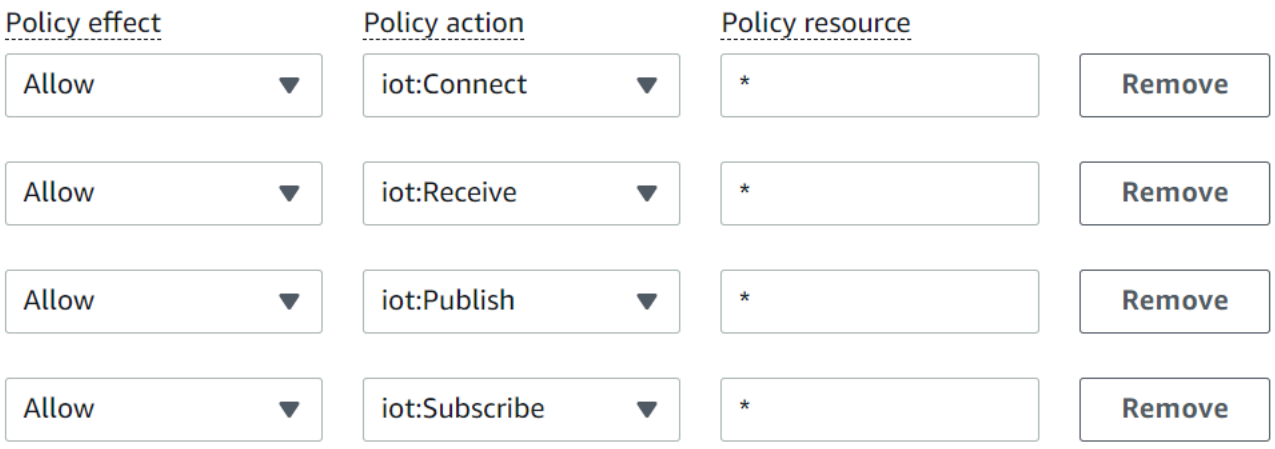
AWS IoT 정책을 생성하려면

1. AWS IoT 콘솔의 왼쪽 메뉴에서 보안을 선택한 다음 정책을 선택합니다.
2. 아직 정책이 없습니다 페이지에서 정책 생성(Create a policy)을 선택합니다.

   계정에 기존 정책이 있는 경우 정책 생성을 선택합니다.
3. 정책 생성 페이지에서 다음을 수행합니다.

   1. 정책 속성(Policy properties) 섹션의 정책 이름(Policy name) 필드에 정책 이름(예:**`My_Iot_Policy`**)을 입력합니다. 정책 이름에 개인 식별 정보를 사용하면 안 됩니다.
   2. 정책 문서(Policy document) 섹션에서 AWS IoT Core 작업에 대한 리소스 액세스 권한을 부여하거나 거부하는 정책 문을 생성합니다. 모든 클라이언트가 **`iot:Connect`**를 수행할 수 있도록 허용하는 정책 문을 생성하려면 다음 단계를 따르세요.
      - 정책 효과(Policy effect) 필드에서 허용(Allow)을 선택합니다. 이렇게 하면 인증서에 이 정책이 연결된 모든 클라이언트가 정책 작업(Policy action) 필드에 나열된 작업을 수행할 수 있습니다.
      - 정책 작업(Policy action) 필드에서 **`iot:Connect`** 등의 정책 작업을 선택합니다. 정책 작업이란 디바이스가 Device SDK에서 예제 프로그램을 실행할 때 수행 권한이 필요한 작업입니다.
      - 정책 리소스(Policy resource) 필드에 Amazon 리소스 이름(ARN) 또는 **`*`**를 입력합니다. **`*`**는 모든 클라이언트(디바이스)를 선택합니다.

      `iot:Receive`, **`iot:Publish`** 및 **`iot:Subscribe`**에 대한 정책 문을 생성하려면 새 문 추가를 선택한 다음 단계를 반복합니다.



> **Note**
>
> 편의를 위해 이 빠른 시작에서는 와일드카드(*) 문자가 사용됩니다. 보안을 강화하려면 와일드카드 문자 대신 클라이언트 ARN을 리소스로 지정하여 메시지를 연결하고 게시할 수 있는 클라이언트(디바이스)를 제한해야 합니다. 클라이언트 ARN은 다음 형식을 따릅니다. `arn:aws:iot:`*`your-region`*`:`*`your-aws-account`*`:client/`*`my-client-id`*
> 그러나 리소스(클라이언트 디바이스, 사물 섀도우 등)를 먼저 생성해야 해당 ARN을 정책에 할당할 수 있습니다. 자세한 내용은 AWS IoT Core 작업 리소스를 참조하세요.

4. 정책 정보를 입력한 다음 생성(Create)를 선택합니다.

자세한 내용은 AWS IoT 에서 IAM을 사용하는 방법 단원을 참조하십시오.

사물 객체 만들기

에 연결된 디바이스는 AWS IoT 레지스트리의 사물 객체로 AWS IoT Core 표시됩니다. 사물 객체는 특정 디바이스 또는 논리적 엔터티를 나타냅니다. 물리적 디바이스 또는 센서(예: 전구 또는 벽면 스위치)일 수 있습니다. 또한 연결되지는 AWS IoT않지만 연결되는 다른 디바이스(예: 엔진 센서 또는 제어판이 있는 자동차)와 관련된 애플리케이션 인스턴스 또는 물리적 개체와 같은 논리적 개체일 수도 있습니다.

AWS IoT 콘솔에서 사물을 생성하려면

1. AWS IoT 콘솔의 왼쪽 메뉴에서 모든 디바이스를 선택한 다음 사물을 선택합니다.
2. 사물 페이지에서 사물 생성을 선택합니다.
3. 사물 생성(Create things) 페이지에서 단일 사물 생성(Create a single thing)을 선택한 후 다음(Next)을 선택합니다.
4. 사물 속성 지정(Specify thing properties) 페이지에서 사물 이름(Thing name)에 사물의 이름(예: **`MyIotThing`**)을 입력합니다.

   사물 이름은 나중에 바꿀 수 없으므로 신중하게 선택합니다.

   사물 이름을 변경하려면 새 사물을 생성하고 새 이름을 지정한 다음 이전 사물을 삭제해야 합니다.

> **Note**
>
> 사물 이름에 개인 식별 정보를 사용하면 안 됩니다. 사물 이름은 암호화되지 않은 통신 및 보고서에 나타날 수 있습니다.

5. 이 페이지의 나머지 필드는 비워 둡니다. 다음(Next)을 선택합니다.
6. 디바이스 인증서 구성 - 선택 사항 페이지에서 새 인증서 자동 생성(권장)을 선택합니다. 다음을 선택합니다.
7. 인증서에 정책 연결 - 선택 사항 페이지에서 이전 단원에서 생성한 정책을 선택합니다. 이 섹션에서 정책의 이름은 **`My_Iot_Policy`**입니다. 사물 생성(Create thing)을 선택합니다.
8. 인증서 및 키 다운로드 페이지:

   1. 각 인증서 및 키 파일을 다운로드하여 나중에 사용할 수 있도록 저장해 둡니다. 디바이스에 이러한 파일을 설치해야 합니다.

      인증서 파일을 저장할 때 다음 표에 이름을 지정합니다. 다음은 이후 예제에서 사용되는 파일 이름입니다.

      인증서 파일 이름

파일	파일 경로
프라이빗 키	`private.pem.key`
퍼블릭 키	(이 예에서는 사용되지 않음)
디바이스 인증서	`device.pem.crt`
루트 CA 인증서	`Amazon-root-CA-1.pem`

   2. 사용 중인 데이터 엔드포인트 및 암호 스위트 유형에 해당하는 루트 CA 인증서 파일의 다운로드 링크를 선택하여 이러한 파일에 대한 루트 CA 파일을 다운로드합니다. 본 자습서에서는 RSA 2048 비트 키: Amazon Root CA 1 오른쪽에 있는 다운로드를 선택하고 RSA 2048 비트 키: Amazon Root CA 1 인증서 파일을 다운로드합니다.

      > **Important**
      >
      > 이 페이지를 나가기 전에 인증서 파일을 저장해야 합니다. 콘솔에서 이 페이지를 나가면 인증서 파일에 더 이상 액세스할 수 없습니다.

이 단계에서 만든 인증서 파일을 다운로드하는 것을 잊은 경우 이 콘솔 화면을 종료하고 콘솔의 사물 목록으로 이동하여 만든 사물 개체를 삭제한 다음 처음부터 이 절차를 다시 시작해야 합니다.

3. 완료를 선택합니다.

이 절차를 완료한 후에는 사물 목록에 새 사물 개체가 표시됩니다.

디바이스 구성

이 단원에서는 디바이스를 AWS IoT Core에 연결하는 방법에 대해 설명합니다. 를 시작하려면 Amazon EC2를 사용하여 가상 디바이스 AWS IoT Core 를 생성하거나 Windows PC 또는 Mac을 IoT 디바이스로 사용할 수 있습니다.

시도할 가장 적합한 디바이스 옵션을 선택합니다 AWS IoT Core. 모든 것을 시도할 수는 있지만 한 번에 하나만 시도하세요. 어떤 디바이스 옵션이 가장 적합한지 모르는 경우 최상의 디바이스 옵션(which device option is the best)을 선택하는 방법을 읽은 다음 이 페이지로 돌아옵니다.

디바이스 옵션

- Amazon EC2를 사용하여 가상 디바이스 생성
- Windows 또는 Linux PC 또는 Mac을 AWS IoT 디바이스로 사용
- Raspberry Pi 또는 다른 디바이스 연결

Amazon EC2를 사용하여 가상 디바이스 생성

이 자습서에서는 클라우드에서 가상 디바이스로 사용할 Amazon EC2 인스턴스를 생성합니다.

이 자습서를 완료하려면이 필요합니다 AWS 계정. 없는 경우 계속하기 전에 설정 AWS 계정에서 설명하는 단계를 완료하세요.

이 자습서에서 배울 내용은 다음과 같습니다.

- Amazon EC2 인스턴스 설정
- Git, Node.js 설치 및 구성 AWS CLI
- 가상 디바이스에 대한 AWS IoT 리소스 생성
- AWS IoT Device SDK for JavaScript 설치

- 샘플 애플리케이션 실행
- AWS IoT 콘솔에서 샘플 앱의 메시지 보기

Amazon EC2 인스턴스 설정

다음 단계에서는 물리적 디바이스 대신 가상 디바이스 역할을 할 Amazon EC2 인스턴스를 생성하는 방법을 보여 줍니다.

Amazon EC2 인스턴스를 처음 생성하는 경우 Amazon EC2Linux 인스턴스 시작하기가 더 도움이 될 것입니다.

인스턴스 시작

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽의 콘솔 메뉴에서 Instances(인스턴스) 섹션을 펼치고 Instances(인스턴스)를 선택합니다. Instances(인스턴스) 대시보드에서 오른쪽의 Launch Instance(인스턴스 시작)를 선택하면 기본 구성 목록이 표시됩니다.
3. Name and tags(이름 및 태그) 섹션에서 인스턴스의 이름을 입력하고 선택적으로 태그를 추가합니다.
4. Application and OS Images(Amazon Machine Image)(애플리케이션 및 OS 이미지(Amazon Machine Image)) 섹션에서 Amazon Linux 2 AMI(HVM)와 같이 인스턴스에 대한 AMI 템플릿을 선택합니다. 해당되는 AMI는 "프리 티어 사용 가능"으로 표시됩니다.
5. Instance type(인스턴스 유형) 섹션에서 인스턴스의 하드웨어 구성을 선택할 수 있습니다. 기본으로 선택된 `t2.micro` 유형을 선택합니다. 프리 티어에 적격인 인스턴스 유형입니다.
6. Key pair (login)(키 페어(로그인)) 섹션의 드롭다운 목록에서 키 페어 이름을 선택하거나 Create a new key pair(새 키 페어 생성)를 선택하여 새 키 페어를 생성합니다. 새 키 페어를 생성할 때는 프라이빗 키 파일을 다운로드하여 안전한 곳에 저장해야 합니다. 이때만 다운로드하고 저장할 수 있기 때문입니다. 인스턴스를 시작할 때 키 페어의 이름을 제공하고, 인스턴스에 연결할 때마다 해당 프라이빗 키를 제공해야 합니다.

   > **Warning**
   >
   > Proceed without a key pair(키 페어 없이 계속) 옵션을 선택하지 마세요. 키 페어 없이 인스턴스를 시작하면 인스턴스에 연결할 수 없습니다.

7. Network settings(네트워크 설정) 섹션 및 Configure storage(스토리지 구성) 섹션에서 기본 설정을 유지할 수 있습니다. 준비가 완료되면 Launch instances(인스턴스 시작)를 선택합니다.

8. 확인 페이지에서 인스턴스가 실행 중인지 확인할 수 있습니다. 인스턴스 보기를 선택하여 확인 페이지를 닫고 콘솔로 돌아갑니다.
9. 인스턴스 화면에서 시작 상태를 볼 수 있습니다. 인스턴스를 시작하는 데 약간 시간이 걸립니다. 인스턴스를 시작할 때 초기 상태는 `pending`입니다. 인스턴스가 시작된 후에는 상태가 `running`으로 바뀌고 퍼블릭 DNS 이름을 받습니다. (퍼블릭 DNS(IPv4) 열이 숨겨져 있는 경우 페이지 오른쪽 상단 모서리에 있는 열 표시/숨기기(기어 모양 아이콘)를 선택한 다음 퍼블릭 DNS(IPv4)를 선택합니다.)
10. 연결할 수 있도록 인스턴스가 준비될 때까지 몇 분 정도 걸릴 수 있습니다. 인스턴스가 상태 확인을 통과했는지 확인하세요. 상태 검사 열에서 이 정보를 볼 수 있습니다.

    새 인스턴스가 상태 확인을 통과한 후 다음 절차를 계속하여 인스턴스에 연결합니다.

인스턴스에 연결하려면

Amazon EC2 콘솔에서 인스턴스를 선택하고 Amazon EC2 Instance Connect를 사용하여 연결하도록 선택하면 브라우저 기반 클라이언트를 사용하여 인스턴스에 연결할 수 있습니다. Instance Connect는 권한을 처리하여 성공적인 연결을 제공합니다.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽 메뉴에서 Instances(인스턴스)를 선택합니다.
3. 인스턴스를 선택한 다음 연결을 선택합니다.
4. Amazon EC2 Instance Connect, Connect(연결)를 선택합니다.

이제 새 Amazon EC2 인스턴스에 로그인된 Amazon EC2 Instance Connect 창이 있어야 합니다.

Git, Node.js 설치 및 구성 AWS CLI

이 섹션에서는 Linux 인스턴스에 Git과 Node.js를 설치합니다.

Git를 설치하려면

1. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 인스턴스를 업데이트합니다.

   ```
   sudo yum update -y
   ```

2. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 Git를 설치합니다.

   ```
   sudo yum install git -y
   ```

3. Git이 설치되어 있고 현재 버전의 Git이 설치되어 있는지 확인하려면 다음 명령을 실행합니다.

```
git --version
```

Node.js를 설치하려면

1. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 nvm(노드 버전 관리자)를 설치합니다.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
```

nvm을 사용하면 여러 버전의 Node.js를 설치할 수 있고 여러 버전 간을 전환할 수 있기 때문에 여기서는 nvm을 사용하여 Node.js를 설치합니다.

2. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 nvm을 활성화합니다.

```
. ~/.nvm/nvm.sh
```

3. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 최신 버전의 Node.js를 설치하기 위해 nvm을 사용합니다.

```
nvm install 16
```

> **Note**
>
> 이 명령어를 실행하면 Node.js의 최신 LTS 릴리스를 설치합니다.

Node.js를 설치하면 npm(노드 패키지 관리자)도 설치되므로 필요에 따라 추가 모듈을 설치할 수 있습니다.

4. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 Node.js가 설치되었고 올바르게 실행되는지 테스트합니다.

```
node -e "console.log('Running Node.js ' + process.version)"
```

이 자습서에서는 Node v10.0 이상이 필요합니다. 자세한 정보는 자습서: Amazon EC2 인스턴스에서 Node.js 설정(Tutorial: Setting Up Node.js on an Amazon EC2 Instance)을 참조하세요.

를 구성하려면 AWS CLI

Amazon EC2 인스턴스에는 AWS CLI이(가) 미리 로드됩니다. 그러나 AWS CLI 프로필을 작성해야 합니다. CLI를 구성하는 방법에 대한 자세한 정보는 AWS CLI구성을 참조하세요.

1. 다음 예제는 샘플 값을 보여줍니다. 이들을 사용자의 고유한 값으로 교체합니다. 이 값은 AWS 콘솔의 계정 정보에 있는 보안 자격 증명(Security credentials)에서 찾을 수 있습니다.

   Amazon EC2 Instance Connect 창에서 다음 명령을 입력합니다.

   ```
   aws configure
   ```

   그런 다음 표시된 프롬프트에 계정의 값을 입력합니다.

   ```
   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]: json
   ```

2. 다음 명령을 사용하여 AWS CLI 구성을 테스트할 수 있습니다.

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

   AWS CLI 이 올바르게 구성된 경우 명령은에서 엔드포인트 주소를 반환해야 합니다 AWS 계정.

가상 디바이스에 대한 AWS IoT 리소스 생성

이 섹션에서는 AWS CLI 를 사용하여 가상 디바이스에서 사물 객체와 해당 인증서 파일을 직접 생성하는 방법을 설명합니다. 이 작업은 다른 컴퓨터에서 디바이스로 복사할 때 발생할 수 있는 잠재적인 복합 문제를 피하기 위해 디바이스에서 직접 수행됩니다. 이 섹션에서는 가상 디바이스를 위한 다음과 같은 리소스를 생성합니다.

- 가상 디바이스를 나타낼 사물 객체입니다 AWS IoT.
- 가상 디바이스를 인증하기 위한 인증서.
- 가상 디바이스가 AWS IoT에 연결하고 메시지를 게시, 수신 및 구독할 수 있는 권한을 부여하는 정책 문서.

Linux 인스턴스에서 AWS IoT 사물 객체를 생성하려면

에 연결된 디바이스는 AWS IoT 레지스트리의 사물 객체로 AWS IoT 표시됩니다. 사물 객체는 특정 디바이스 또는 논리적 엔터티를 나타냅니다. 이 경우 사물 객체는 가상 디바이스인 이 Amazon EC2 인스턴스를 나타냅니다.

1. Amazon EC2 Instance Connect 창에서 다음 명령을 실행하여 사물 객체를 만듭니다.

```
aws iot create-thing --thing-name "MyIotThing"
```

2. JSON 응답은 다음과 같아야 합니다.

```
{
    "thingArn": "arn:aws:iot:your-region:your-aws-account:thing/MyIotThing",
    "thingName": "MyIotThing",
    "thingId": "6cf922a8-d8ea-4136-f3401EXAMPLE"
}
```

Linux 인스턴스에서 AWS IoT 키와 인증서를 생성하고 연결하려면

create-keys-and-certificate 명령은 Amazon Root 인증 기관에서 서명한 클라이언트 인증서를 생성합니다. 이 인증서는 가상 디바이스의 자격 증명을 인증하는 데 사용됩니다.

1. Amazon EC2 Instance Connect 창에서 인증서 및 키 파일을 저장할 디렉터리를 생성합니다.

```
mkdir ~/certs
```

2. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 Amazon CA(인증 기관) 인증서의 사본을 다운로드합니다.

```
curl -o ~/certs/Amazon-root-CA-1.pem \
    https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

3. Amazon EC2 Instance Connect 창에서 다음 명령을 실행하여 프라이빗 키, 퍼블릭 키 및 X.509 인증서 파일을 생성합니다. 또한이 명령은 인증서를 등록하고 활성화합니다 AWS IoT.

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile "~/certs/device.pem.crt" \
    --public-key-outfile "~/certs/public.pem.key" \
```

```
    --private-key-outfile "~/certs/private.pem.key"
```

그 응답은 다음과 같습니다. 후속 명령에서 사용할 수 있도록 certificateArn을 저장해 둡니다. 이것은 이후 단계에서 인증서를 사물에 연결하고 정책을 인증서에 연결하는 데 필요합니다.

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC
 KEY-----\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h
\nMMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/
gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB\n-----
END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omitted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

4. Amazon EC2 Instance Connect 창에서 다음 명령과, 앞의 명령의 응답으로 얻은 *certificateArn*을 사용하여 방금 만든 인증서에 사물 객체를 연결합니다.

```
aws iot attach-thing-principal \
    --thing-name "MyIotThing" \
    --principal "certificateArn"
```

이 명령이 제대로 실행되면 어떤 출력도 표시하지 않습니다.

정책을 생성하여 연결하려면

1. Amazon EC2 Instance Connect 창에서 이 정책 문서를 복사하여 이름이 **~/policy.json**인 파일에 붙여 넣어서 정책 파일을 생성합니다.

   자주 사용하는 Linux 편집기가 없는 경우 다음 명령을 사용하여 nano를 열 수 있습니다.

   ```
   nano ~/policy.json
   ```

   policy.json에 대한 정책 문서를 붙여넣습니다. nano 편집기를 종료(Ctrl-X)하고 파일을 저장합니다.

   policy.json용 정책 문서의 내용.

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "iot:Publish",
                   "iot:Subscribe",
                   "iot:Receive",
                   "iot:Connect"
               ],
               "Resource": [
                   "*"
               ]
           }
       ]
   }
   ```

2. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 정책을 생성합니다.

```
aws iot create-policy \
    --policy-name "MyIotThingPolicy" \
    --policy-document "file://~/policy.json"
```

출력:

```
{
    "policyName": "MyIotThingPolicy",
    "policyArn": "arn:aws:iot:us-east-1:your-aws-account:policy/
MyIotThingPolicy",
    "policyDocument": "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [
            {
                \"Effect\": \"Allow\",
                \"Action\": [
                    \"iot:Publish\",
                    \"iot:Receive\",
                    \"iot:Subscribe\",
                    \"iot:Connect\"
                ],
                \"Resource\": [
                    \"*\"
                ]
        }
        ]
    }",
    "policyVersionId": "1"
}
```

3. Amazon EC2 Instance Connect 창에서 다음 명령을 사용하여 정책을 가상 디바이스의 인증서에 연결합니다.

```
aws iot attach-policy \
    --policy-name "MyIotThingPolicy" \
    --target "certificateArn"
```

이 명령이 제대로 실행되면 어떤 출력도 표시하지 않습니다.

AWS IoT Device SDK for JavaScript 설치

이 단원에서는 애플리케이션이 AWS IoT 및 샘플 프로그램과 통신하는 데 사용할 수 있는 코드가 포함된 AWS IoT Device SDK for JavaScript를 설치합니다. 자세한 내용은 JavaScript GitHub 리포지토리용AWS IoT 디바이스 SDK를 참조하세요.

Linux 인스턴스에 AWS IoT Device SDK for JavaScript를 설치하려면

1. Amazon EC2 Instance Connect 창에서이 명령을 사용하여 AWS IoT Device SDK for JavaScript 리포지토리를 홈 `aws-iot-device-sdk-js-v2` 디렉터리의 디렉터리에 복제합니다.

   ```
   cd ~
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

2. 이전 단계에서 생성한 `aws-iot-device-sdk-js-v2` 디렉터리로 이동합니다.

   ```
   cd aws-iot-device-sdk-js-v2
   ```

3. npm을 사용하여 SKD를 설치합니다.

   ```
   npm install
   ```

샘플 애플리케이션 실행

다음 섹션의 명령은 이 표에 표시된 대로 키 및 인증서 파일이 가상 디바이스에 저장되어 있다고 가정합니다.

인증서 파일 이름

파일	파일 경로
프라이빗 키	`~/certs/private.pem.key`
디바이스 인증서	`~/certs/device.pem.crt`
루트 CA 인증서	`~/certs/Amazon-root-CA-1.pem`

이 섹션에서는 AWS IoT Device SDK for JavaScript의 `aws-iot-device-sdk-js-v2/samples/node` 디렉터리에 있는 `pub-sub.js` 샘플 앱을 설치하고 실행합니다. 이 앱은 디바이스인 Amazon

EC2 인스턴스가 MQTT 라이브러리를 사용하여 MQTT 메시지를 게시하고 구독하는 방법을 보여줍니다. `pub-sub.js` 샘플 앱은 `topic_1` 주제를 구독하고 해당 주제에 10개의 메시지를 게시하고 메시지 브로커로부터 받은 메시지를 표시합니다.

샘플 앱을 설치하고 실행하려면

1. Amazon EC2 Instance Connect에서 다음 명령을 사용하여 SDK가 생성한 `aws-iot-device-sdk-js-v2/samples/node/pub_sub` 디렉터리로 이동하고 샘플 앱을 설치합니다.

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. Amazon EC2 Instance Connect 창에서이 명령을 사용하여에서 *`your-iot-endpoint`* AWS IoT 를 가져옵니다.

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

3. Amazon EC2 Instance Connect 창에서 *`your-iot-endpoint`*를 표시된 대로 삽입하고 다음 명령을 실행합니다.

   ```
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

샘플 앱은 다음과 같은 작업을 수행합니다.

1. 계정의 AWS IoT Core 에 연결합니다.
2. 메시지 주제 topic_1을 구독하고 해당 주제에 대해 수신하는 메시지 표시.
3. topic_1 주제에 10개의 메시지 게시.
4. 다음과 유사한 출력 표시.

```
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":1}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":2}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":3}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":4}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
```

```
{"message":"Hello world!","sequence":5}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":6}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":7}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":8}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":9}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":10}
```

샘플 앱을 실행하는 데 문제가 있는 경우 the section called “샘플 애플리케이션 문제 해결”를 검토합니다.

샘플 앱이 수행 중인 작업에 대한 자세한 메시지를 표시하도록 명령줄에 `--verbosity debug` 파라미터를 추가할 수도 있습니다. 이 정보는 문제를 해결하는 데 필요한 도움을 제공할 수 있습니다.

AWS IoT 콘솔에서 샘플 앱의 메시지 보기

AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하여, 샘플 앱의 메시지가 메시지 브로커를 통과할 때 해당 메시지를 볼 수 있습니다.

샘플 앱에서 게시한 MQTT 메시지를 보려면

1. MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기을(를) 검토합니다. 이는 MQTT 메시지가 메시지 브로커를 통과할 때 해당 메시지를 보기 위해 AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하는 방법을 파악하는 데 도움이 됩니다.
2. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.
3. Subscribe to a topic(주제 구독)에서 topic_1 주제를 구독합니다.
4. Amazon EC2 Instance Connect 창에서 샘플 앱을 다시 실행하고 AWS IoT 콘솔의 MQTT 테스트 클라이언트에서 메시지를 확인합니다.

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

MQTT 및가 프로토콜을 AWS IoT Core 지원하는 방법에 대한 자세한 내용은 MQTT를 참조하세요.

Windows 또는 Linux PC 또는 Mac을 AWS IoT 디바이스로 사용

이 자습서에서는와 함께 사용할 개인용 컴퓨터를 구성합니다 AWS IoT. 이 지침은 Windows 및 Linux PC와 Mac을 지원합니다. 이를 위해서는 컴퓨터에 소프트웨어를 설치해야 합니다. 컴퓨터에 소프트웨어를 설치하지 않으려는 경우 Amazon EC2를 사용하여 가상 디바이스 생성을(를) 시도하여 가상 머신에 모든 소프트웨어를 설치할 수 있습니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 개인용 컴퓨터 설정
- Git, Python 및 AWS IoT Device SDK for Python 설치
- 정책을 설정하고 샘플 애플리케이션을 실행합니다.
- AWS IoT 콘솔에서 샘플 앱의 메시지 보기
- Python에서 공유 구독 예시 실행하기

개인용 컴퓨터 설정

이 자습서를 완료하려면 인터넷에 연결된 Windows 또는 Linux PC 또는 Mac이 필요합니다.

다음 단계를 진행하기 전에 컴퓨터에서 명령줄 창을 열 수 있는지 확인합니다. Windows PC에서 cmd.exe를 사용합니다. Linux PC 또는 Mac에서는 Terminal을 사용합니다.

Git, Python 및 AWS IoT Device SDK for Python 설치

이 단원에서는 Python과 Python용 AWS IoT 디바이스 SDK를 컴퓨터에 설치합니다.

최신 버전의 Git와 Python 설치

이 절차에서는 PC에 최신 버전의 Git 및 Python을 설치하는 방법을 설명합니다.

Git와 Python을 컴퓨터에 다운로드하고 설치하려면

1. 컴퓨터에 Git가 설치되어 있는지 확인합니다. 명령줄에 다음 명령을 입력합니다.

   ```
   git --version
   ```

   명령이 Git 버전을 표시하면 Git가 설치된 것이므로, 다음 단계를 진행할 수 있습니다.

   명령에 오류가 표시되면 https://git-scm.com/download를 열고 컴퓨터에 Git를 설치하세요.

2. Python이 이미 설치되어 있는지 확인합니다. 명령줄에 다음 명령을 입력합니다.

```
python -V
```

> **Note**
>
> 이 명령으로 `Python was not found` 오류가 발생하는 경우 운영 체제가 Python v3.x 실행 파일을 `Python3`(으)로 호출하기 때문일 수 있습니다. 이 경우 `python`의 모든 인스턴스를 `python3`(으)로 바꾸고 이 자습서의 나머지 단원을 진행합니다.

명령이 Python 버전을 표시하면 Python이 이미 설치되어 있는 것입니다. 이 자습서에서는 Python v3.7 이상이 필요합니다.

3. Python이 설치되어 있으면 이 단원의 나머지 단계를 건너뛸 수 있습니다. 그렇지 않은 경우 계속 진행합니다.
4. https://www.python.org/downloads/를 열고 컴퓨터에 설치 프로그램을 다운로드하세요.
5. 다운로드가 자동으로 설치되지 않으면 다운로드한 프로그램을 실행하여 Python을 설치합니다.
6. Python의 설치를 확인합니다.

```
python -V
```

명령이 Python 버전을 표시하는지 확인합니다. Python 버전이 표시되지 않으면 Python을 다시 다운로드하여 설치해 보세요.

AWS IoT Device SDK for Python 설치

컴퓨터에 Python용 AWS IoT Device SDK를 설치하려면

1. AWS IoT Device SDK for Python의 v2를 설치합니다.

```
python3 -m pip install awsiotsdk
```

2. AWS IoT Device SDK for Python 리포지토리를 홈 디렉터리의 aws-iot-device-sdk-python-v2 디렉터리에 복제합니다. 이 절차에서는 설치하는 파일의 기본 디렉터리를 *#(home)*으로 지칭합니다.

   # 디렉터리의 실제 위치는 운영 체제에 따라 다릅니다.

Linux/macOS

macOS와 Linux에서 #디렉터리는 ~입니다.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

Windows

Windows에서는 cmd 창에서 다음 명령을 실행하여 # 디렉터리 경로를 찾을 수 있습니다.

```
echo %USERPROFILE%
cd %USERPROFILE%
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

> **Note**
>
> cmd.exe 대신 Windows PowerShell을 사용하는 경우 다음 명령을 사용합니다.
>
> ```
> echo $home
> ```

자세한 내용은 Python GitHub 리포지토리용AWS IoT 디바이스 SDK를 참조하세요.

샘플 애플리케이션 실행 준비

샘플 애플리케이션을 실행하도록 시스템을 준비하려면

- certs 디렉터리를 만듭니다. the section called “AWS IoT 리소스 생성”에서 사물 객체를 생성하고 등록할 때 저장한 프라이빗 키, 디바이스 인증서 및 루트 CA 인증서 파일을 certs 디렉터리에 복사합니다. 대상 디렉터리에 있는 각 파일의 이름은 테이블의 파일 이름과 일치해야 합니다.

  다음 섹션의 명령은 이 표에 표시된 대로 키 및 인증서 파일이 디바이스에 저장되어 있다고 가정합니다.

Linux/macOS

다음 명령을 실행하여 샘플 애플리케이션을 실행할 때 사용할 certs 하위 디렉터리를 생성합니다.

```
mkdir ~/certs
```

새 하위 디렉터리에, 다음 표에 표시된 대상 파일 경로로 파일을 복사합니다.

인증서 파일 이름

파일	파일 경로
프라이빗 키	~/certs/private.pem.key
디바이스 인증서	~/certs/device.pem.crt
루트 CA 인증서	~/certs/Amazon-root-CA-1.pem

다음 명령을 실행하여 certs 디렉터리의 파일을 나열하고 표에 나열된 파일과 비교합니다.

```
ls -l ~/certs
```

Windows

다음 명령을 실행하여 샘플 애플리케이션을 실행할 때 사용할 certs 하위 디렉터리를 생성합니다.

```
mkdir %USERPROFILE%\certs
```

새 하위 디렉터리에, 다음 표에 표시된 대상 파일 경로로 파일을 복사합니다.

인증서 파일 이름

파일	파일 경로
프라이빗 키	%USERPROFILE%\certs\private.pem.key

파일	파일 경로
디바이스 인증서	`%USERPROFILE%\certs\device.pem.crt`
루트 CA 인증서	`%USERPROFILE%\certs\Amazon-root-CA-1.pem`

다음 명령을 실행하여 `certs` 디렉터리의 파일을 나열하고 표에 나열된 파일과 비교합니다.

```
dir %USERPROFILE%\certs
```

정책을 설정하고 샘플 애플리케이션을 실행합니다.

이 단원에서는 정책을 설정하고 AWS IoT Device SDK for Python의 `aws-iot-device-sdk-python-v2/samples`디렉터리에 있는 `pubsub.py` 샘플 스크립트를 실행합니다. 이 스크립트는 디바이스가 MQTT 라이브러리를 사용하여 MQTT 메시지를 게시 및 구독하는 방법을 보여 줍니다.

`pubsub.py` 샘플 앱은 `test/topic` 주제를 구독하고 해당 주제에 10개의 메시지를 게시하고 메시지 브로커로부터 받은 메시지를 표시합니다.

`pubsub.py` 샘플 스크립트를 실행하려면 다음 정보가 필요합니다.

애플리케이션 파라미터 값

파라미터	값을 찾을 수 있는 위치
your-iot-endpoint	1. AWS IoT 콘솔의 왼쪽 메뉴에서 설정(Settings)을 선택합니다. 2. 설정 페이지에서 엔드포인트가 디바이스 데이터 엔드포인트 섹션에 표시됩니다.

your-iot-endpoint 값의 형식은 *endpoint_id*`-ats.iot.`*region*`.amazonaws.com`(예: `a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com`)입니다.

스크립트를 실행하기 전에 사물 정책이 샘플 스크립트에 연결, 구독, 게시 및 수신할 수 있는 권한을 제공하는지 확인합니다.

사물 리소스에 대한 정책 문서를 찾고 검토하려면

1. AWS IoT 콘솔의 사물(Things) 목록에서 디바이스를 나타내는 사물 리소스를 찾습니다.
2. 디바이스를 나타내는 사물 리소스의 이름(Name) 링크를 선택하여 사물 세부 정보(Thing details) 페이지를 엽니다.
3. 사물 세부 정보(Thing details) 페이지의 인증서(Certificates) 탭에서 사물 리소스에 연결된 인증서를 선택합니다. 목록에는 인증서가 하나만 있어야 합니다. 여러 개 있는 경우 디바이스에 파일이 설치되어 있고 AWS IoT Core에 연결하는 데 사용할 인증서를 선택합니다.

   인증서(Certificate) 세부 정보 페이지의 정책(Policies) 탭에서 인증서에 연결된 정책을 선택합니다. 단 하나만 있어야 합니다. 여러 개 있는 경우 각각에 대해 다음 단계를 반복하여 최소한 하나의 정책이 필요한 액세스 권한을 부여하는지 확인합니다.
4. 정책(Policy) 개요 페이지에서 JSON 편집기를 찾고 정책 문서 편집(Edit policy document)을 선택하여 필요에 따라 정책 문서를 검토하고 편집합니다.
5. 다음 예제에 정책 JSON이 표시됩니다. `"Resource"` 요소에서를 각 `Resource` 값 AWS 계정 에서 AWS 리전 및 *`region:account`*로 바꿉니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/test/topic"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/test-*"
            ]
        }
    ]
}
```

Linux/macOS

Linux/macOS에서 샘플 스크립트를 실행하려면

1. 명령줄 창에서 다음 명령을 사용하여 SDK가 만든 ~/aws-iot-device-sdk-python-v2/samples/node/pub_sub 디렉터리로 이동합니다.

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 명령줄 창에서 *your-iot-endpoint*를 표시된 대로 바꾸고 다음 명령을 실행합니다.

```
python3 pubsub.py --endpoint your-iot-endpoint --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/private.pem.key
```

Windows

Windows PC에서 샘플 앱을 실행하려면

1. 명령줄 창에서 다음 명령을 사용하여 SDK가 만든 %USERPROFILE%\aws-iot-device-sdk-python-v2\samples 디렉터리로 이동하고 샘플 앱을 설치합니다.

```
cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
```

2. 명령줄 창에서 *your-iot-endpoint*를 표시된 대로 바꾸고 다음 명령을 실행합니다.

```
python3 pubsub.py --endpoint your-iot-endpoint --ca_file %USERPROFILE%
\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key
 %USERPROFILE%\certs\private.pem.key
```

샘플 스크립트는 다음과 같은 작업을 수행합니다.

1. 계정의 AWS IoT Core 에 연결합니다.
2. 메시지 주제 test/topic를 구독하고 해당 주제에 대해 수신하는 메시지 표시.
3. test/topic 주제에 10개의 메시지 게시.
4. 다음과 유사한 출력 표시.

```
Connected!
Subscribing to topic 'test/topic'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'test/topic': Hello World! [1]
Received message from topic 'test/topic': b'"Hello World! [1]"'
Publishing message to topic 'test/topic': Hello World! [2]
Received message from topic 'test/topic': b'"Hello World! [2]"'
Publishing message to topic 'test/topic': Hello World! [3]
Received message from topic 'test/topic': b'"Hello World! [3]"'
Publishing message to topic 'test/topic': Hello World! [4]
Received message from topic 'test/topic': b'"Hello World! [4]"'
Publishing message to topic 'test/topic': Hello World! [5]
Received message from topic 'test/topic': b'"Hello World! [5]"'
Publishing message to topic 'test/topic': Hello World! [6]
Received message from topic 'test/topic': b'"Hello World! [6]"'
Publishing message to topic 'test/topic': Hello World! [7]
Received message from topic 'test/topic': b'"Hello World! [7]"'
Publishing message to topic 'test/topic': Hello World! [8]
Received message from topic 'test/topic': b'"Hello World! [8]"'
Publishing message to topic 'test/topic': Hello World! [9]
Received message from topic 'test/topic': b'"Hello World! [9]"'
Publishing message to topic 'test/topic': Hello World! [10]
Received message from topic 'test/topic': b'"Hello World! [10]"'
10 message(s) received.
Disconnecting...
Disconnected!
```

샘플 앱을 실행하는 데 문제가 있는 경우 the section called “샘플 애플리케이션 문제 해결”를 검토합니다.

샘플 앱이 수행 중인 작업에 대한 자세한 메시지를 표시하도록 명령줄에 `--verbosity Debug` 파라미터를 추가할 수도 있습니다. 이 정보는 문제를 해결하는 데 도움이 될 수 있습니다.

AWS IoT 콘솔에서 샘플 앱의 메시지 보기

AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하여, 샘플 앱의 메시지가 메시지 브로커를 통과할 때 해당 메시지를 볼 수 있습니다.

샘플 앱에서 게시한 MQTT 메시지를 보려면

1. MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기을(를) 검토합니다. 이는 MQTT 메시지가 메시지 브로커를 통과할 때 해당 메시지를 보기 위해 AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하는 방법을 파악하는 데 도움이 됩니다.
2. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.
3. Subscribe to a topic(주제 구독)에서 test/topic 주제를 구독합니다.
4. 명령줄 창에서 샘플 앱을 다시 실행하고 AWS IoT 콘솔의 MQTT 클라이언트에서 메시지를 확인합니다.

   Linux/macOS

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic test/topic --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

   Windows

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
   python3 pubsub.py --topic test/topic --ca_file %USERPROFILE%\certs\Amazon-root-
   CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs
   \private.pem.key --endpoint your-iot-endpoint
   ```

MQTT 및가 프로토콜을 AWS IoT Core 지원하는 방법에 대한 자세한 내용은 MQTT를 참조하세요.

Python에서 공유 구독 예시 실행하기

AWS IoT Core 는 MQTT 3 및 MQTT 5 모두에 대해 공유 구독을 지원합니다. 공유 구독을 사용하면 여러 클라이언트가 한 주제에 대한 구독을 공유할 수 있으며 무작위 배포를 통해 해당 주제에 게시된 메시지를 한 클라이언트만 수신할 수 있습니다. 공유 구독을 사용하려면 클라이언트가 공유 구독의 주제 필터를 구독합니다. `$share/{ShareName}/{TopicFilter}`

정책을 설정하고 공유 구독 예시를 실행하는 방법

1. 공유 구독 예시를 실행하려면 MQTT 5 공유 구독에 설명된 대로 사물 정책을 설정해야 합니다.
2. 공유 구독 예시를 실행하려면 다음 명령을 실행합니다.

   Linux/macOS

   Linux/macOS에서 샘플 스크립트를 실행하려면

   1. 명령줄 창에서 다음 명령을 사용하여 SDK가 만든 `~/aws-iot-device-sdk-python-v2/samples` 디렉터리로 이동합니다.

      ```
      cd ~/aws-iot-device-sdk-python-v2/samples
      ```

   2. 명령줄 창에서 *your-iot-endpoint*를 표시된 대로 바꾸고 다음 명령을 실행합니다.

      ```
      python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
       ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
      private.pem.key --group_identifier consumer
      ```

   Windows

   Windows PC에서 샘플 앱을 실행하려면

   1. 명령줄 창에서 다음 명령을 사용하여 SDK가 만든 `%USERPROFILE%\aws-iot-device-sdk-python-v2\samples` 디렉터리로 이동하고 샘플 앱을 설치합니다.

      ```
      cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
      ```

   2. 명령줄 창에서 *your-iot-endpoint*를 표시된 대로 바꾸고 다음 명령을 실행합니다.

      ```
      python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
       %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs
      ```

```
\device.pem.crt --key %USERPROFILE%\certs\private.pem.key --group_identifier
 consumer
```

> **Note**
>
> 샘플을 실행할 때 필요에 따라 그룹 식별자를 선택적으로 지정할 수 있습니다(예: `--group_identifier consumer`). 지정하지 않는 경우 `python-sample`이 기본 그룹 식별자입니다.

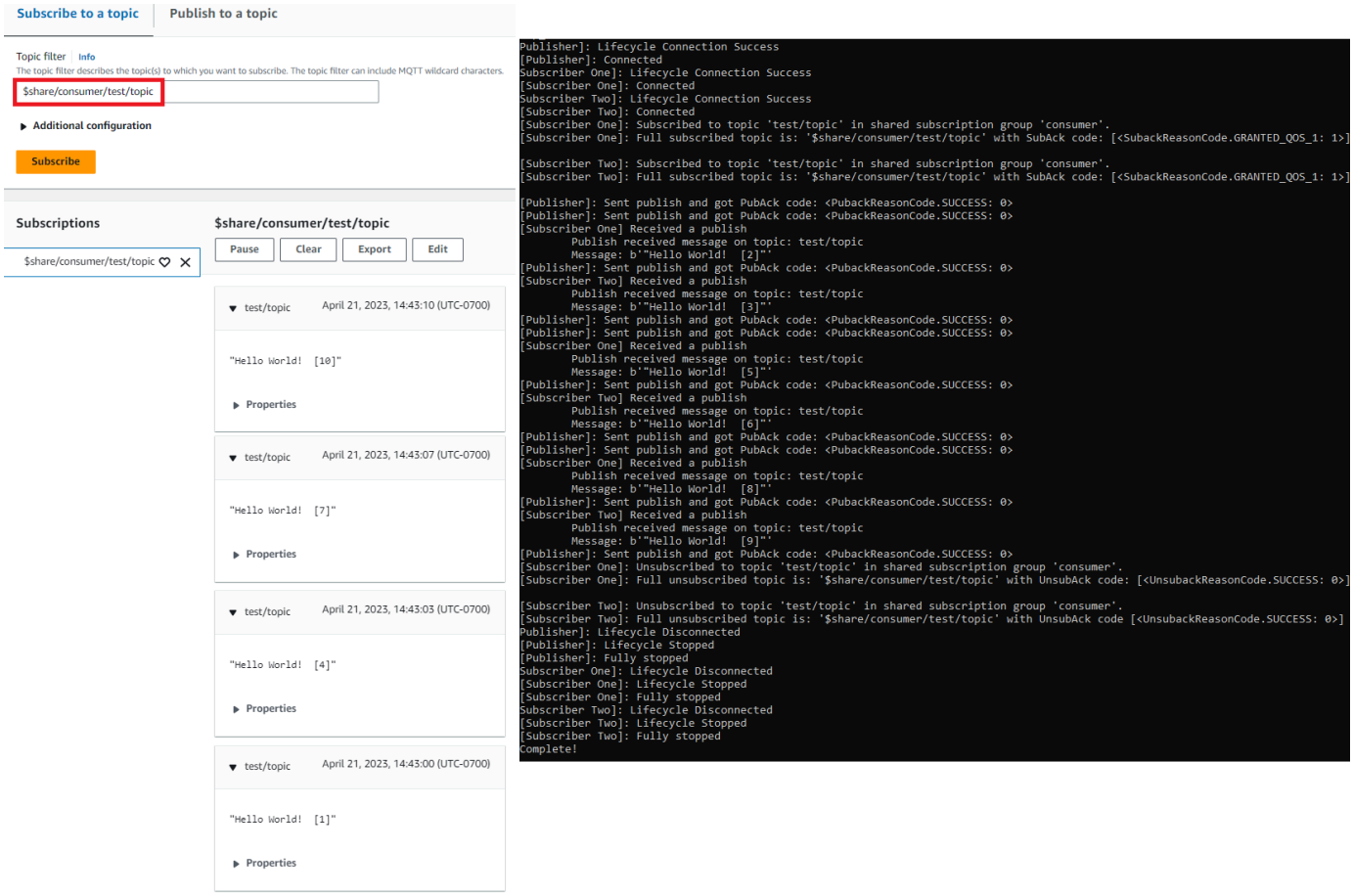
3. 명령줄의 출력은 다음과 같을 수 있습니다.

```
Publisher]: Lifecycle Connection Success
[Publisher]: Connected
Subscriber One]: Lifecycle Connection Success
[Subscriber One]: Connected
Subscriber Two]: Lifecycle Connection Success
[Subscriber Two]: Connected
[Subscriber One]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Subscriber Two]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [1]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [2]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [3]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [4]"'
```

```
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [5]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [6]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [7]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [8]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [9]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [10]"'
[Subscriber One]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code: [<UnsubackReasonCode.SUCCESS: 0>]
[Subscriber Two]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code [<UnsubackReasonCode.SUCCESS: 0>]
Publisher]: Lifecycle Disconnected
[Publisher]: Lifecycle Stopped
[Publisher]: Fully stopped
Subscriber One]: Lifecycle Disconnected
[Subscriber One]: Lifecycle Stopped
[Subscriber One]: Fully stopped
Subscriber Two]: Lifecycle Disconnected
[Subscriber Two]: Lifecycle Stopped
[Subscriber Two]: Fully stopped
Complete!
```

4. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다. 주제 구독에서 `$share/consumer/test/topic`과 같은 공유 구독 주제를 구독하세요. 샘플을 실행할 때 필요에 따라 그룹 식별자를 지정할 수 있습니다(예: `--group_identifier consumer`). 그룹 식별자를 지정하지 않는 경우 기본값은 `python-sample`입니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 MQTT 5 공유 구독 Python 예시 및 공유 구독을 참조하세요.

   명령줄 창에서 샘플 앱을 다시 실행하고 AWS IoT 콘솔 및 명령줄의 MQTT 테스트 클라이언트에서 메시지 배포를 확인합니다.



Raspberry Pi 또는 다른 디바이스 연결

이 섹션에서는와 함께 사용할 Raspberry Pi를 구성합니다 AWS IoT. 연결하려는 다른 디바이스가 있는 경우 Raspberry Pi에 대한 지침에는 이러한 지침을 디바이스에 적용하는 데 도움이 되는 참조가 포함되어 있습니다.

이 작업은 일반적으로 20분 정도 소요되지만 시스템 소프트웨어 업그레이드가 많은 경우 더 오래 걸릴 수 있습니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 디바이스 설정
- AWS IoT Device SDK에 필요한 도구 및 라이브러리 설치
- AWS IoT 디바이스 SDK 설치
- 샘플 앱 설치 및 실행
- AWS IoT 콘솔에서 샘플 앱의 메시지 보기

> **Important**
>
> 이러한 지침을 다른 디바이스 및 운영 체제에 적용하는 것은 어려울 수 있습니다. 이러한 지침을 해석하고 디바이스에 적용할 수 있을 만큼 디바이스를 잘 이해해야 합니다.
> 디바이스를 구성하는 동안 문제가 발생하는 경우 Amazon EC2를 사용하여 가상 디바이스 생성 또는와 같은 다른 디바이스 옵션 중 하나를 대안으로 시도 AWS IoT할 수 있습니다Windows 또는 Linux PC 또는 Mac을 AWS IoT 디바이스로 사용.

디바이스 설정

이 단계의 목표는 운영 체제(OS)를 시작하고 인터넷에 연결하며 명령줄 인터페이스에서 디바이스와 상호 작용할 수 있도록 디바이스를 구성하는 데 필요한 항목을 수집하는 것입니다.

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정. 없는 경우 계속하기 전에 설정 AWS 계정에서 설명하는 단계를 완료하세요.
- Raspberry Pi 3 Model B 또는 최신 모델. 이 기능은 이전 버전의 Raspberry Pi에서 작동할 수 있지만 테스트되지 않았습니다.
- Raspberry Pi OS(32비트) 이상. 최신 버전의 Raspberry OS를 사용하는 것이 좋습니다. 이전 버전의 OS는 작동하지만 테스트되지 않았습니다.

  이 예제를 실행하가 위해 GUI(그래픽 사용자 인터페이스)가 있는 데스크톱을 설치할 필요는 없지만, Raspberry Pi에 익숙하지 않고 Raspberry Pi 하드웨어가 GUI가 있는 데스크톱을 지원한다면 GUI가 있는 데스크탑을 사용하는 것이 더 쉬울 수 있습니다.
- 이더넷 또는 Wi-Fi 연결.
- 키보드, 마우스, 모니터, 케이블, 전원 공급 장치, 및 디바이스에 필요한 기타 하드웨어.

> **Important**
>
> 다음 단계를 진행하기 전에 디바이스에 운영 체제가 설치, 구성 및 실행되어야 합니다. 디바이스가 인터넷에 연결되어 있어야 하며 명령줄 인터페이스를 사용하여 디바이스에 액세스할 수 있어야 합니다. 명령줄 액세스는 직접 연결된 키보드, 마우스 및 모니터를 통해 또는 SSH 터미널 원격 인터페이스를 사용하여 수행할 수 있습니다.

그래픽 사용자 인터페이스(GUI)가 있는 Raspberry Pi에서 운영 체제를 실행 중인 경우 디바이스에서 터미널 창을 열고 해당 창에서 다음 지침을 수행합니다. 그렇지 않고 PuTTY와 같은 원격 터미널을 사용하여 디바이스에 연결하는 경우 디바이스에 대한 원격 터미널을 열고 사용하세요.

AWS IoT Device SDK에 필요한 도구 및 라이브러리 설치

AWS IoT 디바이스 SDK 및 샘플 코드를 설치하기 전에 시스템이 최신 상태이고 SDKs를 설치하는 데 필요한 도구와 라이브러리가 있는지 확인합니다.

1. 운영 체제 업데이트 및 필수 라이브러리 설치

   AWS IoT 디바이스 SDK를 설치하기 전에 디바이스의 터미널 창에서이 명령을 실행하여 운영 체제를 업데이트하고 필요한 라이브러리를 설치합니다.

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```

   ```
   sudo apt-get install cmake
   ```

   ```
   sudo apt-get install libssl-dev
   ```

2. Git 설치

   디바이스의 운영 체제에 Git이 설치되어 있지 않은 경우 이를 설치하여 AWS IoT Device SDK for JavaScript를 설치해야 합니다.

   a. 다음 명령을 실행하여 Git가 이미 설치되었는지 테스트합니다.

      ```
      git --version
      ```

b. 앞의 명령이 Git 버전을 반환하면 Git가 이미 설치되어 있는 것이므로, 3단계로 건너뛸 수 있습니다.

c. git 명령을 실행할 때 오류가 표시되는 경우 다음 명령을 실행하여 Git를 설치하세요.

```
sudo apt-get install git
```

d. 다음 명령을 실행하여 Git가 이미 설치되었는지 다시 테스트합니다.

```
git --version
```

e. Git가 설치된 경우 다음 섹션을 진행합니다. 그렇지 않은 경우 계속하기 전에 문제를 해결하고 오류를 수정하세요. AWS IoT Device SDK for JavaScript를 설치하려면 Git이 필요합니다.

AWS IoT 디바이스 SDK 설치

AWS IoT 디바이스 SDK를 설치합니다.

Python

이 단원에서는 Python, 개발 도구 및 AWS IoT Device SDK for Python을 디바이스에 설치합니다. 이 지침은 최신 Raspberry Pi OS를 실행하는 Raspberry Pi에 대한 것입니다. 다른 디바이스가 있거나 다른 운영 체제를 사용하는 경우 디바이스에 다음 지침을 적용해야 합니다.

1. Python과 개발 도구 설치

   Python용 AWS IoT 디바이스 SDK를 사용하려면 Raspberry Pi에 Python v3.5 이상을 설치해야 합니다.

   디바이스의 터미널 창에서 다음 명령을 실행합니다.

   1. 다음 명령을 실행하여 디바이스에 설치된 Python의 버전을 확인합니다.

      ```
      python3 --version
      ```

      Python이 설치되어 있는 경우, 해당 버전이 표시됩니다.

   2. 표시된 버전이 `Python 3.5` 이상인 경우 2단계로 건너뛸 수 있습니다.

   3. 표시된 버전이 `Python 3.5`보다 작은 경우 다음 명령을 실행하여 올바른 버전을 설치할 수 있습니다.

```
sudo apt install python3
```

4. 다음 명령을 실행하여 올바른 버전의 Python이 설치되었는지 확인합니다.

```
python3 --version
```

2. pip3에 대한 테스트

   디바이스의 터미널 창에서 다음 명령을 실행합니다.

   1. pip3가 설치되어 있으면 다음 명령을 실행합니다.

```
pip3 --version
```

   2. 명령이 버전 번호를 반환하면 pip3이(가) 설치되어 있는 것이므로, 3단계로 건너뛸 수 있습니다.
   3. 앞의 명령이 오류를 반환하면 다음 명령을 실행하여 pip3을(를) 설치합니다.

```
sudo apt install python3-pip
```

   4. pip3가 설치되어 있으면 다음 명령을 실행합니다.

```
pip3 --version
```

3. 현재 AWS IoT Device SDK for Python 설치

   AWS IoT Device SDK for Python을 설치하고 샘플 앱을 디바이스에 다운로드합니다.

   디바이스에서 다음 명령을 실행합니다.

```
cd ~
python3 -m pip install awsiotsdk
```

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

JavaScript

이 섹션에서는 디바이스에 Node.js, npm 패키지 관리자 및 JavaScript용 AWS IoT 디바이스 SDK를 설치합니다. 이 지침은 Raspberry Pi OS를 실행하는 Raspberry Pi에 대한 것입니다. 다른 디바이스가 있거나 다른 운영 체제를 사용하는 경우 디바이스에 다음 지침을 적용해야 합니다.

1. 최신 버전의 Node.js 설치

   JavaScript용 AWS IoT 디바이스 SDK를 사용하려면 Raspberry Pi에 Node.js와 npm 패키지 관리자를 설치해야 합니다.

   a. 다음 명령을 입력하여 노드 리포지토리의 최신 버전을 다운로드합니다.

   ```
   cd ~
   curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
   ```

   b. 노드와 npm을 설치합니다.

   ```
   sudo apt-get install -y nodejs
   ```

   c. 노드 설치를 확인합니다.

   ```
   node -v
   ```

   명령에 노드 버전이 표시되는지 확인합니다. 이 자습서에서는 Node v10.0 이상이 필요합니다. 노드 버전이 표시되지 않으면 노드 리포지토리를 다시 다운로드해 보세요.

   d. npm 설치를 확인합니다.

   ```
   npm -v
   ```

   명령이 npm 버전을 표시하는지 확인합니다. npm 버전이 표시되지 않으면 노드와 npm을 다시 설치하세요.

   e. 디바이스를 다시 시작합니다.

   ```
   sudo shutdown -r 0
   ```

   디바이스가 다시 시작된 후 계속하세요.

2. AWS IoT Device SDK for JavaScript 설치

Raspberry Pi에 AWS IoT Device SDK for JavaScript를 설치합니다.

a. AWS IoT Device SDK for JavaScript 리포지토리를 # `aws-iot-device-sdk-js-v2` 디렉터리의 디렉터리에 복제합니다. Raspberry Pi에서 # 디렉터리는 `~/`이고, 이는 다음 명령의 # 디렉터리로 사용됩니다. 디바이스가 # 디렉터리의 다른 경로를 사용하는 경우 다음 명령에서 `~/`을(를) 디바이스의 올바른 경로로 바꾸세요.

   이러한 명령은 `~/aws-iot-device-sdk-js-v2` 디렉터리를 만들고 여기에 SDK 코드를 복사해 넣습니다.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

b. 이전 단계에서 생성한 `aws-iot-device-sdk-js-v2` 디렉터리로 변경하고 `npm install` 명령을 실행하여 SDK를 설치합니다. `npm install` 명령은 `aws-crt` 라이브러리 빌드를 호출합니다(완료하는 데 몇 분 정도 걸릴 수 있음).

```
cd ~/aws-iot-device-sdk-js-v2
npm install
```

샘플 앱 설치 및 실행

이 섹션에서는 AWS IoT 디바이스 SDK에 있는 `pubsub` 샘플 앱을 설치하고 실행합니다. 이 앱은 디바이스가 MQTT 라이브러리를 사용하여 MQTT 메시지를 게시 및 구독하는 방법을 보여줍니다. 샘플 앱은 `topic_1` 주제를 구독하고, 해당 주제에 10개의 메시지를 게시하고, 메시지 브로커로부터 받은 메시지를 표시합니다.

인증서 파일 설치

샘플 앱에서는 디바이스를 인증하는 인증서 파일을 디바이스에 설치해야 합니다.

샘플 앱에 대한 디바이스 인증서 파일을 설치하려면

1. 다음 명령을 사용하여 # 디렉터리에 `certs` 하위 디렉터리를 만듭니다.

```
cd ~
mkdir certs
```

2. the section called “AWS IoT 리소스 생성”에서 생성한 프라이빗 키, 디바이스 인증서 및 루트 CA 인증서를 ~/certs 디렉터리에 복사합니다.

   인증서 파일을 디바이스에 복사하는 방법은 디바이스 및 운영 체제에 따라 다르며 여기서는 설명하지 않습니다. 그러나 디바이스가 그래픽 사용자 인터페이스(GUI)를 지원하고 웹 브라우저를 사용하는 경우 디바이스의 웹 브라우저에서 the section called “AWS IoT 리소스 생성”에 설명된 절차를 수행하여 결과 파일을 디바이스에 직접 다운로드할 수 있습니다.

   다음 섹션의 명령은 이 표에 표시된 대로 키 및 인증서 파일이 디바이스에 저장되어 있다고 가정합니다.

   인증서 파일 이름

파일	파일 경로
루트 CA 인증서	~/certs/Amazon-root-CA-1.pem
디바이스 인증서	~/certs/device.pem.crt
프라이빗 키	~/certs/private.pem.key

샘플 앱을 실행하려면 다음 정보가 필요합니다.

애플리케이션 파라미터 값

파라미터	값을 찾을 수 있는 위치
your-iot-endpoint	AWS IoT 콘솔에서 All devices(모든 디바이스)를 선택한 다음 Things(사물)를 선택합니다. AWS IoT 메뉴의 설정 페이지입니다. 엔드포인트는 디바이스 데이터 엔드포인트(Device data endpoint) 섹션에 표시됩니다.

your-iot-endpoint 값의 형식은 *endpoint_id*-ats.iot.*region*.amazonaws.com(예: a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com)입니다.

Python

샘플 앱을 설치하고 실행하려면

1. 샘플 앱 디렉터리로 이동합니다.

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 명령줄 창에서 *your-iot-endpoint*를 표시된 대로 바꾸고 다음 명령을 실행합니다.

```
python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

3. 샘플 앱에서 다음 사항을 확인합니다.

   1. 계정의 AWS IoT 서비스에 연결합니다.
   2. 메시지 주제 topic_1을 구독하고 해당 주제에 대해 수신하는 메시지 표시.
   3. topic_1 주제에 10개의 메시지 게시.
   4. 다음과 유사한 출력 표시.

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to topic 'topic_1'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'topic_1': Hello World! [1]
Received message from topic 'topic_1': b'Hello World! [1]'
Publishing message to topic 'topic_1': Hello World! [2]
Received message from topic 'topic_1': b'Hello World! [2]'
Publishing message to topic 'topic_1': Hello World! [3]
Received message from topic 'topic_1': b'Hello World! [3]'
Publishing message to topic 'topic_1': Hello World! [4]
Received message from topic 'topic_1': b'Hello World! [4]'
Publishing message to topic 'topic_1': Hello World! [5]
Received message from topic 'topic_1': b'Hello World! [5]'
Publishing message to topic 'topic_1': Hello World! [6]
Received message from topic 'topic_1': b'Hello World! [6]'
Publishing message to topic 'topic_1': Hello World! [7]
Received message from topic 'topic_1': b'Hello World! [7]'
```

```
Publishing message to topic 'topic_1': Hello World! [8]
Received message from topic 'topic_1': b'Hello World! [8]'
Publishing message to topic 'topic_1': Hello World! [9]
Received message from topic 'topic_1': b'Hello World! [9]'
Publishing message to topic 'topic_1': Hello World! [10]
Received message from topic 'topic_1': b'Hello World! [10]'
10 message(s) received.
Disconnecting...
Disconnected!
```

샘플 앱을 실행하는 데 문제가 있는 경우 the section called “샘플 애플리케이션 문제 해결”를 검토합니다.

샘플 앱이 수행 중인 작업에 대한 자세한 메시지를 표시하도록 명령줄에 `--verbosity Debug` 파라미터를 추가할 수도 있습니다. 이 정보는 문제를 해결하는 데 필요한 도움을 제공할 수 있습니다.

JavaScript

샘플 앱을 설치하고 실행하려면

1. 명령줄 창에서 다음 명령을 사용하여 SDK가 만든 `~/aws-iot-device-sdk-js-v2/samples/node/pub_sub` 디렉터리로 이동하고 샘플 앱을 설치합니다. `npm install` 명령은 `aws-crt` 라이브러리 빌드를 호출합니다(완료하는 데 몇 분 정도 걸릴 수 있음).

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. 명령줄 창에서 *`your-iot-endpoint`*를 표시된 대로 바꾸고 다음 명령을 실행합니다.

   ```
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

3. 샘플 앱에서 다음 사항을 확인합니다.

   1. 계정의 AWS IoT 서비스에 연결합니다.
   2. 메시지 주제 topic_1을 구독하고 해당 주제에 대해 수신하는 메시지 표시.
   3. topic_1 주제에 10개의 메시지 게시.
   4. 다음과 유사한 출력 표시.

```
Publish received on topic topic_1
{"message":"Hello world!","sequence":1}
Publish received on topic topic_1
{"message":"Hello world!","sequence":2}
Publish received on topic topic_1
{"message":"Hello world!","sequence":3}
Publish received on topic topic_1
{"message":"Hello world!","sequence":4}
Publish received on topic topic_1
{"message":"Hello world!","sequence":5}
Publish received on topic topic_1
{"message":"Hello world!","sequence":6}
Publish received on topic topic_1
{"message":"Hello world!","sequence":7}
Publish received on topic topic_1
{"message":"Hello world!","sequence":8}
Publish received on topic topic_1
{"message":"Hello world!","sequence":9}
Publish received on topic topic_1
{"message":"Hello world!","sequence":10}
```

샘플 앱을 실행하는 데 문제가 있는 경우 the section called “샘플 애플리케이션 문제 해결”를 검토합니다.

샘플 앱이 수행 중인 작업에 대한 자세한 메시지를 표시하도록 명령줄에 `--verbosity Debug` 파라미터를 추가할 수도 있습니다. 이 정보는 문제를 해결하는 데 필요한 도움을 제공할 수 있습니다.

AWS IoT 콘솔에서 샘플 앱의 메시지 보기

AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하여, 샘플 앱의 메시지가 메시지 브로커를 통과할 때 해당 메시지를 볼 수 있습니다.

샘플 앱에서 게시한 MQTT 메시지를 보려면

1. MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기을(를) 검토합니다. 이는 MQTT 메시지가 메시지 브로커를 통과할 때 해당 메시지를 보기 위해 AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하는 방법을 파악하는 데 도움이 됩니다.
2. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.

3. topic_1 주제를 구독합니다.
4. 명령줄 창에서 샘플 앱을 다시 실행하고 AWS IoT 콘솔의 MQTT 클라이언트에서 메시지를 확인합니다.

   Python

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

   JavaScript

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

샘플 애플리케이션 문제 해결

샘플 앱을 실행하려고 할 때 오류가 발생하는 경우 다음과 같이 확인할 수 있습니다.

인증서 확인

인증서가 활성화되지 않은 경우 AWS IoT 는 인증에 사용하는 연결 시도를 수락하지 않습니다. 인증서를 만들 때 활성화(Activate) 버튼을 간과하기가 쉽습니다. 다행히 AWS IoT 콘솔에서 인증서를 활성화할 수 있습니다.

인증서의 활성화를 확인하려면

1. AWS IoT 콘솔의 왼쪽 메뉴에서 보안(Secure)을 선택한 다음 인증서를 선택합니다.
2. 인증서 목록에서 실습을 위해 생성한 인증서를 찾고 상태 열에서 상태를 확인합니다.

   인증서의 이름이 기억나지 않는 경우 비활성 상태인 항목이 있는지 확인하여 사용 중인 항목인지 확인하세요.

   목록에서 인증서를 선택하여 세부 정보 페이지를 엽니다. 세부 정보 페이지에서 인증서를 식별하는 데 도움이 되는 생성 날짜를 볼 수 있습니다.

3. 비활성 인증서를 활성화하려면 인증서의 세부 정보 페이지에서 작업을 선택한 다음 활성화를 선택합니다.

활성 상태인 올바른 인증서를 찾았지만 샘플 앱을 실행하는 데 여전히 문제가 있는 경우 다음 단계에서 설명하는 대로 해당 정책을 확인하세요.

the section called “사물 객체 만들기”의 단계에 따라 새 사물과 새 인증서를 만들 수도 있습니다. 새로운 사물을 만드는 경우 새 사물 이름을 지정하고 새 인증서 파일을 디바이스에 다운로드해야 합니다.

인증서에 연결된 정책 확인

정책은의 작업을 승인합니다 AWS IoT. AWS IoT 에 연결하는 데 사용된 인증서에 정책이 없거나 연결을 허용하는 정책이 없으면 인증서가 활성 상태인 경우에도 연결이 거부됩니다.

인증서에 연결된 정책을 확인하려면

1. 이전 항목에 설명된 대로 인증서를 찾아 세부 정보 페이지를 엽니다.
2. 인증서 세부 정보 페이지의 왼쪽 메뉴에서 정책을 선택하고 인증서에 연결된 정책을 봅니다.
3. 인증서에 연결된 정책이 없는 경우 작업 메뉴를 선택한 다음 정책 연결을 선택하여 정책을 추가합니다.

   the section called “AWS IoT 리소스 생성”에서 이전에 생성한 정책을 선택합니다.
4. 연결된 정책이 있는 경우 정책 타일을 선택하여 해당 세부 정보 페이지를 엽니다.

   세부 정보 페이지에서 정책 문서를 검토하여 the section called “AWS IoT 정책 생성”에서 생성한 것과 동일한 정보가 포함되어 있는지 확인합니다.

명령줄 확인

시스템에 맞는 명령줄을 사용했는지 확인하세요. Linux 및 macOS 시스템에서 사용되는 명령은 Windows 시스템에서 사용되는 명령과 다른 경우가 많습니다.

엔드포인트 주소 확인

입력한 명령을 검토하고 명령의 엔드포인트 주소가 AWS IoT 콘솔의 엔드포인트 주소와 일치하는지 다시 확인하세요.

인증서 파일의 파일 이름 확인

입력한 명령의 파일 이름을 `certs` 디렉터리에 있는 인증서 파일의 파일 이름과 비교합니다.

일부 시스템에서는 제대로 작동하려면 파일 이름을 따옴표로 묶어야 할 수 있습니다.

SDK 설치 확인

SDK 설치가 완전하고 올바른지 확인하세요.

의심스러운 경우 디바이스에 SDK를 다시 설치합니다. 대부분의 경우 ***SDK ##***용 AWS IoT 디바이스 SDK 설치라는 자습서의 섹션을 찾고 절차를 다시 따르는 것이 중요합니다.

JavaScript용AWS IoT 디바이스 SDK를 사용하는 경우 샘플 앱을 실행하기 전에 샘플 앱을 먼저 설치해야 합니다. SDK를 설치해도 샘플 앱은 자동으로 설치되지 않습니다. SDK를 설치한 후 샘플 앱을 수동으로 설치해야 합니다.

MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기

이 섹션에서는 AWS IoT 콘솔에서 AWS IoT MQTT 테스트 클라이언트를 사용하여에서 보내고 받은 MQTT 메시지를 보는 방법을 설명합니다 AWS IoT. 이 섹션에서 사용되는 예제는 AWS IoT Core 자습서 시작하기에서 사용한 예제와 관련이 있지만, 예제에 사용된 *topicName*을 IoT 솔루션에서 사용한 주제 이름 또는 주제 필터로 대체할 수 있습니다.

디바이스는 상태를 전달할 주제로 식별되는 MQTT 메시지를 게시하고 AWS IoT, MQTT 메시지를 AWS IoT 게시하여 디바이스 및 앱에 변경 사항 및 이벤트를 알립니다. MQTT 클라이언트를 사용하여 이러한 주제를 구독하고 메시지 발생 시 메시지를 확인할 수 있습니다. 또한 MQTT 테스트 클라이언트를 사용하여에서 구독한 디바이스 및 서비스에 MQTT 메시지를 게시할 수 있습니다 AWS 계정.

내용

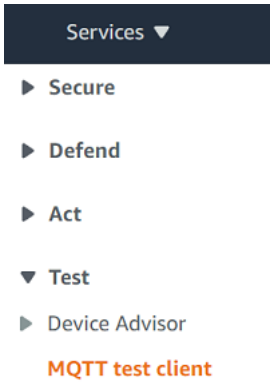
- MQTT 클라이언트에서 MQTT 메시지 보기
- MQTT 클라이언트에서 MQTT 메시지 게시
- MQTT 클라이언트에서 공유 구독 테스트

MQTT 클라이언트에서 MQTT 메시지 보기

다음 절차에서는 디바이스가 메시지를 게시하고 AWS IoT 콘솔에서 메시지를 보는 특정 MQTT 주제를 구독하는 방법을 설명합니다.

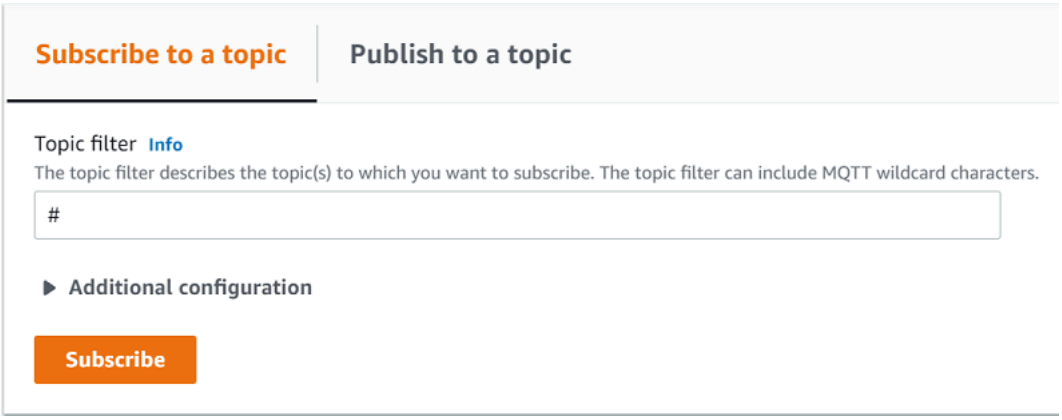
MQTT 테스트 클라이언트에서 MQTT 메시지 보기

1. AWS IoT 콘솔을 열고 왼쪽 메뉴에서 테스트(Test)를 선택하고 MQTT 클라이언트(MQTT test client)를 선택합니다.

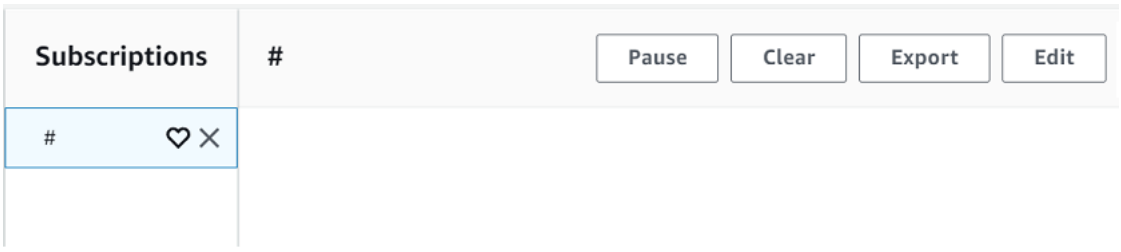


2. 주제 구독(Subscribe to a topic) 탭에서 *topicName*을 입력하여 디바이스가 게시하는 주제를 구독합니다. 시작하기 샘플 앱의 경우, 모든 메시지 주제를 구독하는 #을 구독합니다.

   시작하기 예제를 계속 진행하여 주제 구독 탭의 주제 필터 필드에 #을(를) 입력한 다음 구독을 선택합니다.



   주제 메시지 로그 페이지 #이 열려 구독 목록에 #이(가) 나타납니다. 에서 구성한 디바이스the section called “디바이스 구성”가 예제 프로그램을 실행하는 경우 # 메시지 로그 AWS IoT 에 해당 디바이스가 보내는 메시지가 표시됩니다. 구독 주제가 포함된 메시지를 수신하면 게시 섹션 아래에 메시지 로그 항목이 나타납니다 AWS IoT.



3. # 메시지 로그 페이지에서 메시지를 주제에 게시할 수도 있지만 주제 이름을 지정해야 합니다. # 주제에 게시할 수 없습니다.

구독한 주제에 게시된 메시지는 수신 시 메시지 로그에 표시되며 가장 최근 메시지가 먼저 표시됩니다.

MQTT 메시지 문제 해결

와일드카드 주제 필터 사용

메시지가 예상대로 메시지 로그에 표시되지 않으면 주제 이름 필터에 설명된 와일드카드 주제 필터를 구독해 보세요. MQTT 다중 레벨 와일드카드 주제 필터는 해시 또는 파운드 기호(#)이며 구독 주제 필드에서 주제 필터로 사용할 수 있습니다.

주제 필터를 구독하는 것은 메시지 브로커가 수신한 모든 주제를 구독하는 것입니다. 주제 필터 경로의 요소를 # 다중 레벨 와일드카드 문자 또는 '+' 단일 레벨 와일드카드 문자로 대체하여 필터 범위를 좁힐 수 있습니다.

주제 필터에서 와일드카드를 사용하는 경우

- 다중 레벨 와일드카드 문자는 주제 필터의 마지막 문자여야 합니다.
- 주제 필터 경로에는 주제 레벨당 단일 레벨 와일드카드 문자가 하나만 있을 수 있습니다.

다음 예를 참조하세요.

주제 필터	다음이 포함된 메시지 표시
#	모든 주제 이름
topic_1/#	topic_1/(으)로 시작하는 주제 이름
topic_1/level_2/#	topic_1/level_2/ (으)로 시작하는 주제 이름
topic_1/+/level_3	topic_1/(으)로 시작하고 /level_3(으)로 끝나며 그 사이에 임의 값의 한 요소가 있는 주제 이름.

주제 필터에 대한 자세한 내용은 주제 이름 필터 단원을 참조하세요.

주제 이름 오류 확인

MQTT 주제 이름과 주제 필터는 대소문자를 구분합니다. 예를 들어, 구독한 주제인 `topic_1` 대신 디바이스가 `Topic_1`(대문자 T 포함)에 메시지를 게시하는 경우 해당 메시지는 MQTT 테스트 클라이언트에 나타나지 않습니다. 그러나 와일드카드 주제 필터를 구독하면 디바이스가 메시지를 게시하고 있음을 알 수 있으며 예상한 주제가 아닌 주제 이름을 사용하고 있음을 알 수 있습니다.

MQTT 클라이언트에서 MQTT 메시지 게시

MQTT 주제에 메시지를 게시하려면

1. MQTT 클라이언트 페이지에서 Publish to a topic(주제 게시) 탭의 Topic name(주제 이름) 필드에 메시지의 *topicName*을 입력합니다. 이 예에서는 **my/topic**을 사용합니다.

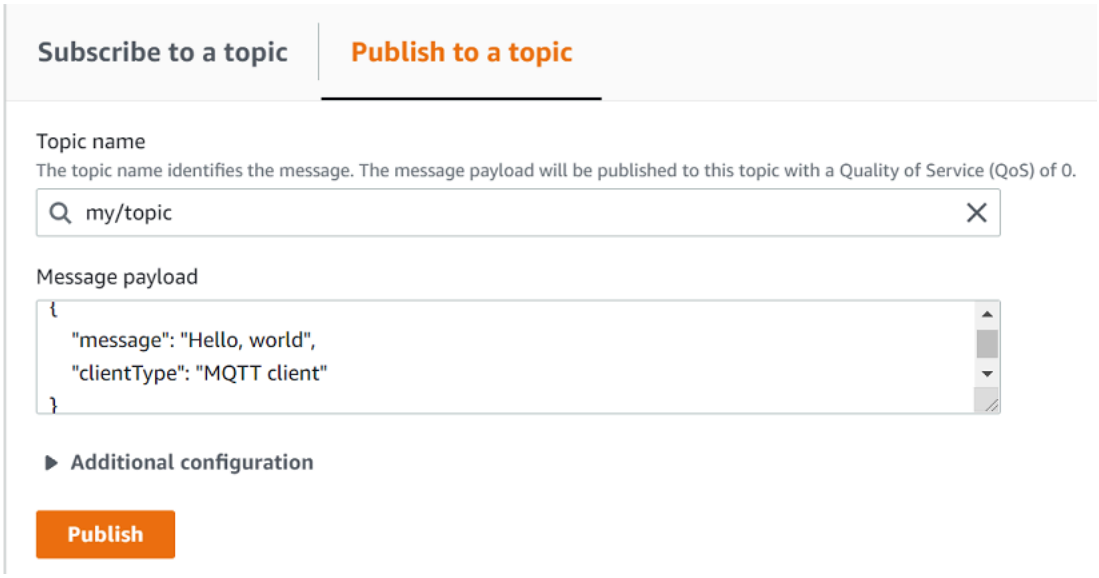
   > **Note**
   >
   > MQTT 테스트 클라이언트 또는 시스템 구현 시 주제 이름에 개인 식별이 가능한 정보를 사용하지 마세요. 주제 이름은 암호화되지 않은 통신 및 보고서에 나타날 수 있습니다.

2. 메시지 페이로드 창에 다음 JSON을 입력합니다.

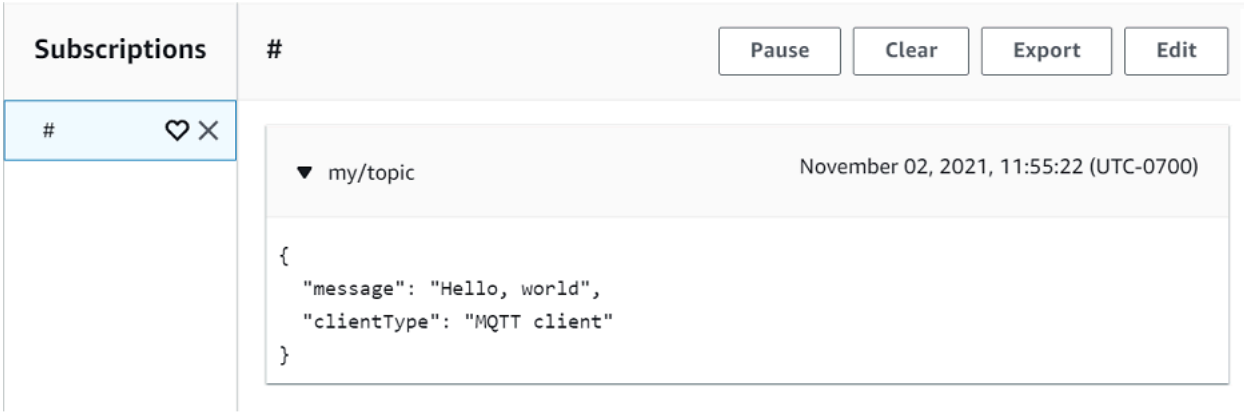
   ```
   {
       "message": "Hello, world",
       "clientType": "MQTT test client"
   }
   ```

3. 메시지를 AWS IoT에 게시하려면 게시를 선택합니다.

   > **Note**
   >
   > 메시지를 게시하기 전에 my/topic 주제를 구독하고 있어야 합니다.



4. 메시지를 보려면 구독(Subscription) 목록에서 my/topic을 선택합니다. 메시지 페이로드 게시 창 아래의 MQTT 테스트 클라이언트에서 메시지가 보여야 합니다.



주제 이름 필드의 *topicName*을 변경하고 게시 버튼을 선택하면 MQTT 메시지를 다른 주제에 게시할 수 있습니다.

> **Important**
>
> 중복 주제(예: probe1/ temperature 및 probe1/#)로 여러 구독을 생성할 경우 두 구독과 일치하는 주제에 게시된 단일 메시지가 중복 구독마다 한 번씩 여러 번 전달될 가능성이 있습니다.

MQTT 클라이언트에서 공유 구독 테스트

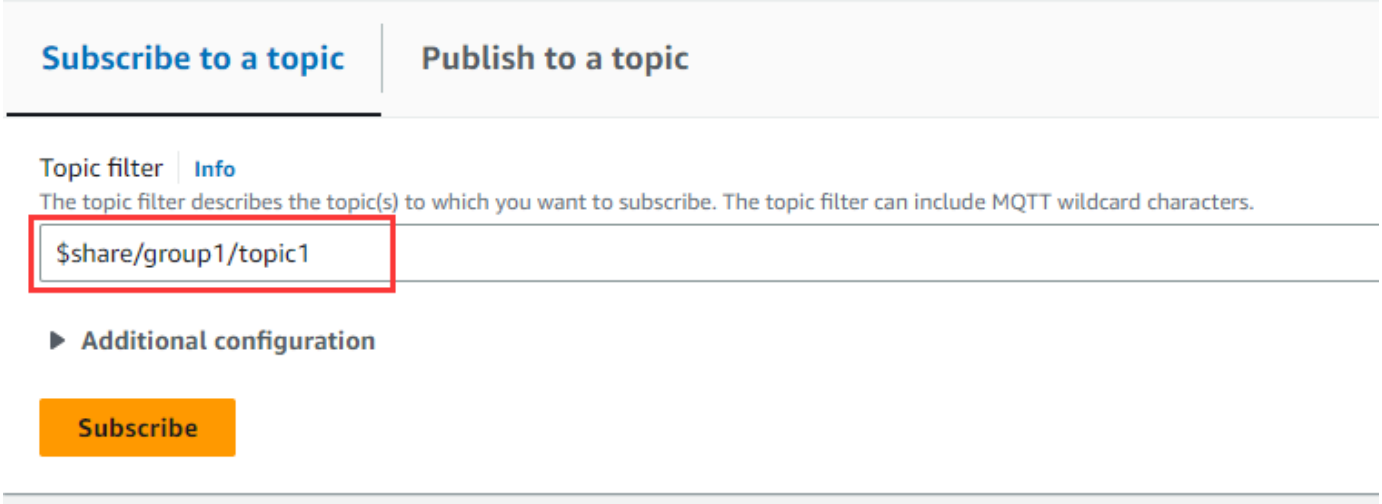
이 섹션에서는 AWS IoT 콘솔에서 AWS IoT MQTT 클라이언트를 사용하여 공유 구독을 AWS IoT 사용하여 송수신되는 MQTT 메시지를 보는 방법을 설명합니다.는 여러 클라이언트가 무작위 배포를 사용하여 주제에 게시된 메시지를 수신하는 클라이언트 한 명과 주제에 대한 구독을 공유할 수 있도록 ??? 허용합니다. 동일한 구독을 공유하는 여러 MQTT 클라이언트(이 예에서는 두 MQTT 클라이언트)를 시뮬레이션하려면 여러 웹 브라우저에서 AWS IoT 콘솔에서 AWS IoT MQTT 클라이언트를 엽니다. 이 섹션에 사용된 예시는 AWS IoT Core 자습서 시작하기에서 사용된 예시와 관련이 없습니다. 자세한 내용은 공유 구독을 참조하세요.

MQTT 주제에 구독을 공유하는 방법

1. AWS IoT 콘솔을 열고 탐색 창에서 테스트를 선택하고 MQTT 테스트 클라이언트를 선택합니다.
2. 주제 구독(Subscribe to a topic) 탭에서 *topicName*을 입력하여 디바이스가 게시하는 주제를 구독합니다. 공유 구독을 사용하려면 공유 구독의 주제 필터를 다음과 같이 구독합니다.

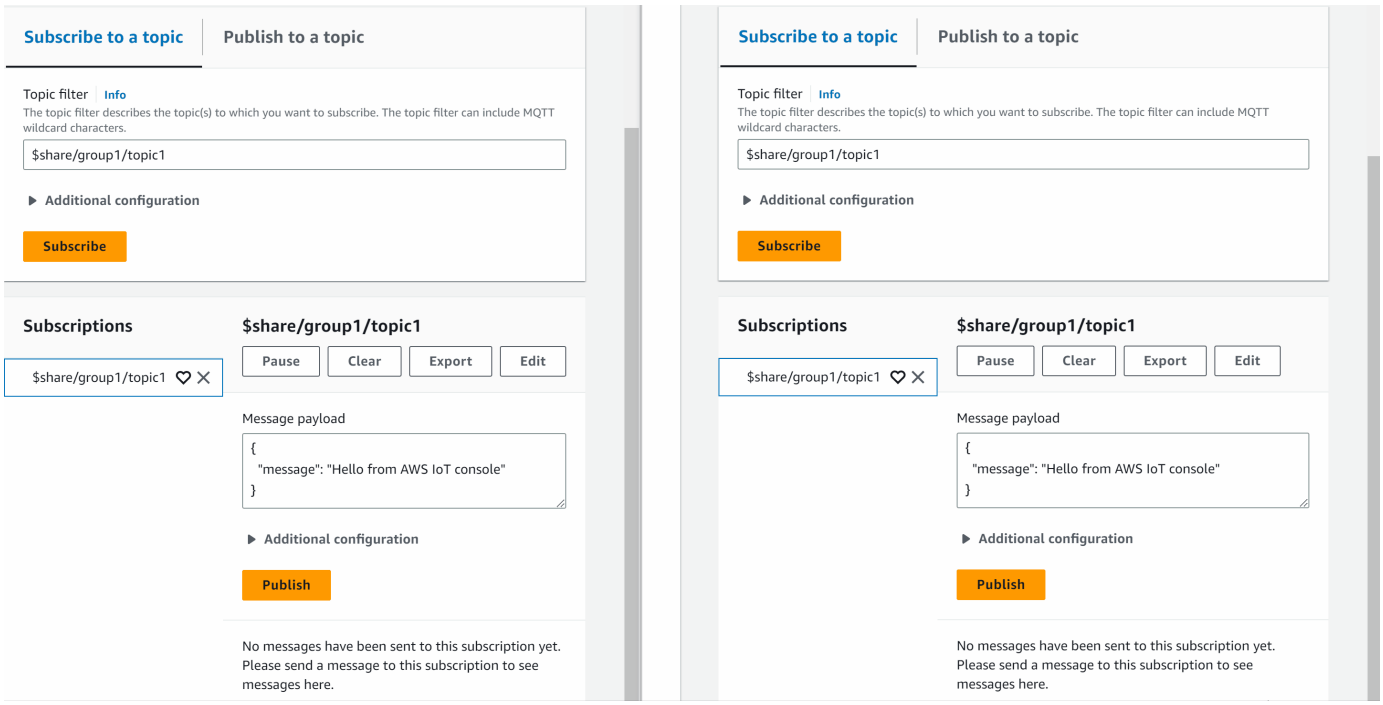
   ```
   $share/{ShareName}/{TopicFilter}
   ```

   주제 필터의 예로는 메시지 주제 **topic1**을 구독하는 **$share/group1/topic1**을 들 수 있습니다.

3. 다른 웹 브라우저를 열고 1단계와 2단계를 반복합니다. 이렇게 하면 동일한 **$share/group1/topic1** 구독을 공유하는 서로 다른 두 MQTT 클라이언트를 시뮬레이션할 수 있습니다.
4. 한 MQTT 클라이언트를 선택하고 주제에 게시 탭의 주제 이름 필드에 메시지의 *topicName*을 입력합니다. 이 예에서는 **topic1**을 사용합니다. 메시지를 몇 번 게시해 보세요. 두 MQTT 클라이언트의 구독 목록에서 클라이언트가 무작위 배포를 사용하여 메시지를 수신하는 것을 확인할 수 있을 것입니다. 이 예시에서는 'Hello from AWS IoT console'이라는 동일한 메시지를 세 번 게시합니

다. 왼쪽의 MQTT 클라이언트는 메시지를 두 번 받았고 오른쪽의 MQTT 클라이언트는 메시지를 한 번 받았습니다.



AWS IoT 자습서

자 AWS IoT 습서는 두 가지 목표를 지원하기 위해 두 가지 학습 경로로 나뉩니다. 목표에 가장 적합한 학습 경로를 선택하세요.

- AWS IoT 솔루션 아이디어를 테스트하거나 보여주기 위해 proof-of-concept 구축하려고 합니다.

  디바이스에서 AWS IoT Device Client를 사용하는 일반적인 IoT 작업 및 애플리케이션을 시연하려면 the section called “AWS IoT Device Client를 사용하여 데모 빌드” 학습 경로를 따르세요. AWS IoT Device Client는 자체 클라우드 리소스를 적용하여 최소한의 개발로 end-to-end 솔루션을 시연할 수 있는 디바이스 소프트웨어를 제공합니다.

  AWS IoT Device Client에 대한 자세한 내용은 AWS IoT Device Client를 참조하세요.
- 솔루션을 배포하기 위해 프로덕션 소프트웨어를 구축하는 방법에 대해 알아보려는 경우

  AWS IoT 디바이스 SDK를 사용하여 특정 요구 사항을 충족하는 자체 솔루션 소프트웨어를 생성하려면 the section called “AWS IoT 디바이스 SDKs를 사용하여 솔루션 구축” 학습 경로를 따르세요.

  사용 가능한 AWS IoT 디바이스 SDKs???. AWS SDKs. AWS

AWS IoT 자습서 학습 경로 옵션

- AWS IoT Device Client를 사용하여 데모 빌드
- AWS IoT 디바이스 SDKs를 사용하여 솔루션 구축

AWS IoT Device Client를 사용하여 데모 빌드

이 학습 경로의 자습서에서는 AWS IoT Device Client를 사용하여 데모 소프트웨어를 개발하는 단계를 안내합니다. AWS IoT Device Client는 IoT 디바이스에서 실행되는 소프트웨어를 제공하여 내장된 IoT 솔루션의 측면을 테스트하고 보여줍니다 AWS IoT.

이 자습서의 목표는 디바이스 소프트웨어를 개발하기 전에가 솔루션을 AWS IoT 지원한다고 확신할 수 있도록 탐색 및 실험을 용이하게 하는 것입니다.

이 자습서에서 배울 내용:

- 에서 IoT 디바이스로 사용할 Raspberry Pi를 준비하는 방법 AWS IoT
- 디바이스에서 AWS IoT Device Client를 사용하여 AWS IoT 기능을 시연하는 방법

이 학습 경로에서는 자체 Raspberry Pi에 AWS IoT Device Client를 설치하고 클라우드에서 AWS IoT 리소스를 생성하여 IoT 솔루션 아이디어를 보여줍니다. 이 학습 경로의 튜토리얼은 Raspberry Pi를 사용하여 기능을 보여주면서 다른 디바이스에 이들을 적용하는 데 도움이 되는 목표와 절차를 설명합니다.

AWS IoT Device Client로 데모를 빌드하기 위한 사전 조건

이 섹션에서는 이 학습 경로에서 튜토리얼을 시작하기 전에 갖추어야 할 사항에 대해 설명합니다.

이 학습 경로의 튜토리얼을 완료하려면 다음이 필요합니다.

- 의 경우 AWS 계정

  기존가 있는 AWS 계정경우를 사용할 수 있지만 이러한 자습서에서 AWS IoT 사용하는 기능을 사용하려면 추가 역할 또는 권한을 추가해야 할 수 있습니다.

  새를 생성해야 하는 경우 섹션을 AWS 계정참조하세요the section called “설정 AWS 계정”.

- Raspberry Pi 또는 호환되는 IoT 디바이스

  튜토리얼에서는 다양한 폼 팩터로 제공되는 Raspberry Pi를 사용합니다. Raspberry Pi는 일반적으로 많이 사용하며 비교적 저렴한 데모 디바이스입니다. 튜토리얼은 Raspberry Pi 3 모델 B+, Raspberry Pi 4 모델 B 및 Ubuntu Server 20.04 LTS(HVM)를 실행하는 Amazon EC2 인스턴스에서 테스트되었습니다. 를 사용하고 명령을 AWS CLI 실행하려면 최신 버전의 Raspberry Pi OS(Raspberry Pi OS(64비트) 또는 OS Lite)를 사용하는 것이 좋습니다. 이전 버전의 OS는 작동하지만 테스트되지 않았습니다.

  > **Note**
  >
  > 튜토리얼은 각 단계의 목표를 설명하여 당사에서 시도하지 않은 IoT 하드웨어에 적응하는 데 도움이 됩니다. 그러나 다른 디바이스에 적용하는 방법을 구체적으로 설명하지는 않습니다.

- IoT 디바이스의 운영 체제에 대한 이해

  이 튜토리얼의 단계에서는 Raspberry Pi에서 지원하는 명령줄 인터페이스에서 기본 Linux 명령 및 작업을 사용하는 데 익숙하다고 가정합니다. 이러한 작업에 익숙하지 않은 경우 튜토리얼을 완료하는 데 더 많은 시간을 할애해야 할 수 있습니다.

  이 튜토리얼을 완료하려면 다음 방법에 대한 이해가 필요합니다.

  - 부품 조립 및 연결, 필요한 전원에 디바이스 연결, 메모리 카드 설치 및 제거와 같은 기본 디바이스 작업을 안전하게 수행합니다.

  - 시스템 소프트웨어 및 파일을 디바이스에 업로드하고 다운로드합니다. 디바이스에서 microSD 카드와 같은 이동식 저장 디바이스를 사용하지 않는 경우 디바이스에 연결하고 시스템 소프트웨어와 파일을 디바이스에 업로드 및 다운로드하는 방법을 알아야 합니다.
  - 디바이스를 사용하려는 네트워크에 연결합니다.
  - SSH 터미널 또는 유사한 프로그램을 사용하여 다른 컴퓨터에서 디바이스에 연결합니다.
  - 명령줄 인터페이스를 사용하여 디바이스에 있는 파일 및 디렉터리의 생성, 복사, 이동, 이름 변경 및 권한 설정을 수행합니다.
  - 디바이스에 새 프로그램을 설치합니다.
  - FTP 또는 SCP와 같은 도구를 사용하여 디바이스와 파일을 주고받습니다.
- IoT 솔루션을 위한 개발 및 테스트 환경

  튜토리얼에서는 필요한 소프트웨어와 하드웨어를 설명합니다. 그러나 튜토리얼에서는 명시적으로 설명되지 않은 작업을 수행할 수 있다고 가정합니다. 이러한 하드웨어 및 작업의 예는 다음과 같습니다.

  - 파일을 다운로드하고 저장할 로컬 호스트 컴퓨터

    Raspberry Pi의 경우 일반적으로 microSD 메모리 카드를 읽고 쓸 수 있는 개인용 컴퓨터 또는 랩톱입니다. 로컬 호스트 컴퓨터는
    - 인터넷에 연결되어 있어야 합니다.
    - AWS CLI를 설치하고 구성합니다.
    - AWS 콘솔을 지원하는 웹 브라우저가 있어야 합니다.
  - 로컬 호스트 컴퓨터를 디바이스에 연결하여 통신하고, 명령을 입력하고, 파일을 전송하는 방법

    Raspberry Pi에서 이는 종종 로컬 호스트 컴퓨터의 SSH 및 SCP를 사용하여 수행됩니다.
  - IoT 디바이스에 연결할 모니터 및 키보드

    이들은 도움이 될 수 있지만 튜토리얼을 완료하는 데 필수 사항은 아닙니다.
  - 로컬 호스트 컴퓨터와 IoT 디바이스가 인터넷에 연결하는 방법

    인터넷에 연결된 라우터 또는 게이트웨이에 대한 유선 또는 무선 네트워크 연결일 수 있습니다. 로컬 호스트도 Raspberry Pi에 연결할 수 있어야 합니다. 이를 위해서는 동일한 LAN에 있어야 합니다. 튜토리얼에서는 특정 디바이스 또는 디바이스 구성에 대해 이를 설정하는 방법을 보여줄 수 없지만 이 연결을 테스트하는 방법은 보여줍니다.
  - LAN 라우터에 액세스하여 연결된 디바이스 보기

이 학습 경로의 튜토리얼을 완료하려면 IoT 디바이스의 IP 주소를 찾을 수 있어야 합니다.

LAN에서는 디바이스가 연결된 네트워크 라우터의 관리자 인터페이스에 액세스하여 이를 수행할 수 있습니다. 라우터에서 디바이스에 고정 IP 주소를 할당할 수 있으면 디바이스가 다시 시작될 때마다 재연결을 간소화할 수 있습니다.

디바이스에 키보드와 모니터가 연결되어 있는 경우 ifconfig는 디바이스의 IP 주소를 표시할 수 있습니다.

이러한 옵션이 없는 경우 재시작할 때마다 디바이스의 IP 주소를 식별하는 방법을 찾아야 합니다.

모든 자료를 준비한 후 the section called “IoT Device Client 사용 준비” 섹션으로 진행합니다.

이 학습 경로의 튜토리얼

- 자습서: Device Client용 AWS IoT 디바이스 준비
- 자습서: AWS IoT Device Client 설치 및 구성
- 자습서: AWS IoT Device Client와의 MQTT 메시지 통신 시연
- 자습서: AWS IoT Device Client를 사용하여 원격 작업(작업) 시연
- 자습서: AWS IoT Device Client 자습서를 실행한 후 정리

자습서: Device Client용 AWS IoT 디바이스 준비

이 튜토리얼에서는 이 학습 경로의 후속 튜토리얼을 위한 준비를 위해 Raspberry Pi를 초기화하는 과정을 안내합니다.

이 튜토리얼의 목표는 현재 버전의 디바이스 운영 체제를 설치하고 개발 환경의 컨텍스트에서 디바이스와 통신할 수 있는지 확인하는 것입니다.

사전 조건

이 자습서를 시작하기 전에 the section called “AWS IoT Device Client로 데모를 빌드하기 위한 사전 조건”에 나열된 항목을 사용할 수 있고 사용할 준비가 되었는지 확인하세요.

이 자습서는 완료하는 데 약 90분이 소요됩니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 디바이스의 운영 체제 설치 및 업데이트입니다.

- 자습서를 실행하는 데 필요한 추가 소프트웨어를 설치하고 확인합니다.
- 디바이스의 연결을 테스트하고 필요한 인증서를 설치합니다.

이 자습서를 완료한 후 다음 자습서에서는 Device Client를 사용하는 데모를 위해 AWS IoT 디바이스를 준비합니다.

이 튜토리얼의 절차

- 디바이스의 운영 체제 설치 및 업데이트
- 디바이스에 필요한 소프트웨어 설치 및 확인
- 디바이스 테스트 및 Amazon CA 인증서 저장

디바이스의 운영 체제 설치 및 업데이트

이 섹션의 절차에서는 Raspberry Pi가 시스템 드라이브에 사용하는 microSD 카드를 초기화하는 방법을 설명합니다. Raspberry Pi의 microSD 카드에는 운영 체제(OS) 소프트웨어와 애플리케이션 파일 스토리지 공간이 있습니다. Raspberry Pi를 사용하지 않는 경우 디바이스의 지침에 따라 디바이스의 운영 체제 소프트웨어를 설치하고 업데이트합니다.

이 섹션을 완료한 후에는 IoT 디바이스를 시작하고 로컬 호스트 컴퓨터의 터미널 프로그램에서 연결할 수 있어야 합니다.

필수 장비:

- 로컬 개발 및 테스트 환경
- 인터넷에 연결할 수 있는 Raspberry Pi 또는 IoT 디바이스
- 최소 8GB 용량의 microSD 메모리 카드 또는 OS 및 필요한 소프트웨어를 위한 충분한 스토리지.

> **Note**
>
> 이러한 연습을 위해 microSD 카드를 선택할 때 필요한 만큼 크지만 가능한 한 작은 카드를 선택합니다.
> 작은 SD 카드는 백업 및 업데이트가 더 빠릅니다. Raspberry Pi에서는 이 튜토리얼을 위해 8GB 이상의 microSD 카드가 필요하지 않습니다. 특정 애플리케이션에 더 많은 공간이 필요한 경우 이 튜토리얼에서 저장하는 더 작은 이미지 파일로 더 큰 카드의 파일 시스템 크기를 조정하여 선택한 카드의 지원되는 공간을 모두 사용할 수 있습니다.

옵션 장비:

- Raspberry Pi에 연결된 USB 키보드
- HDMI 모니터 및 Raspberry Pi에 모니터를 연결하는 케이블

이 섹션의 절차:

- microSD 카드에 디바이스의 운영 체제 로드
- 새로운 운영 체제로 IoT 디바이스 시작
- 디바이스에 로컬 호스트 컴퓨터 연결

microSD 카드에 디바이스의 운영 체제 로드

이 절차는 로컬 호스트 컴퓨터를 사용하여 디바이스의 운영 체제를 microSD 카드에 로드합니다.

> **Note**
>
> 디바이스가 운영 체제에 이동식 저장 매체를 사용하지 않는 경우 해당 디바이스에 대한 절차를 사용하여 운영 체제를 설치하고 the section called “IoT 디바이스 시작” 섹션으로 진행합니다.

Raspberry Pi에 운영 체제를 설치하려면

1. 로컬 호스트 컴퓨터에서 사용하려는 Raspberry Pi 운영 체제 이미지를 다운로드하고 압축을 풉니다. 최신 버전은 https://www.raspberrypi.com/software/operating-systems/에서 구할 수 있습니다.

   Raspberry Pi OS 버전 선택

   이 학습 경로의 이들 튜토리얼을 지원하는 가장 작은 버전이기 때문에 이 튜토리얼에서는 Raspberry Pi OS Lite 버전을 사용합니다. 이 버전의 Raspberry Pi OS에는 명령줄 인터페이스만 있고 그래픽 사용자 인터페이스는 없습니다. 그래픽 사용자 인터페이스가 있는 최신 Raspberry Pi OS 버전도 이 튜토리얼에서도 사용할 수 있습니다. 그러나 이 학습 경로에 설명된 절차는 명령줄 인터페이스만 사용하여 Raspberry Pi에서 작업을 수행합니다.
2. microSD 카드를 로컬 호스트 컴퓨터에 삽입합니다.
3. SD 카드 이미징 도구를 사용하여 압축을 푼 OS 이미지 파일을 microSD 카드에 씁니다.
4. microSD 카드에 Raspberry Pi OS 이미지를 쓴 후

   a. 명령줄 창 또는 파일 탐색기 창에서 microSD 카드의 BOOT 파티션을 엽니다.

b. microSD 카드의 BOOT 파티션 루트 디렉터리에 파일 확장자와 내용이 없는 ssh라는 빈 파일을 생성합니다. 이 파일은 Raspberry Pi가 처음 시작할 때 SSH 통신을 사용 설정하도록 지시합니다.

5. microSD 카드를 꺼내고 로컬 호스트 컴퓨터에서 안전하게 제거합니다.

the section called “IoT 디바이스 시작”에 사용할 microSD 카드가 준비되었습니다.

새로운 운영 체제로 IoT 디바이스 시작

이 절차는 microSD 카드를 설치하고 다운로드한 운영 체제를 사용하여 처음으로 Raspberry Pi를 시작합니다.

새로운 운영 체제로 IoT 디바이스를 시작하려면

1. 디바이스의 전원을 분리한 상태에서 이전 단계인 the section called “OS 로드”의 microSD 카드를 Raspberry Pi에 삽입합니다.
2. 네트워크에 디바이스를 연결합니다.
3. 이 튜토리얼은 SSH 터미널을 사용하여 로컬 호스트 컴퓨터에서 Raspberry Pi와 상호 작용합니다.

   기기와 직접 상호 작용하려면 다음을 수행합니다.

   a. 로컬 호스트 컴퓨터의 터미널 창을 Raspberry Pi에 연결하기 전에 HDMI 모니터를 연결하여 Raspberry Pi의 콘솔 메시지를 확인합니다.

   b. Raspberry Pi와 직접 상호 작용하려면 Raspberry Pi에 USB 키보드를 연결합니다.

4. Raspberry Pi에 전원을 연결하고 초기화될 때까지 1분 정도 기다립니다.

   Raspberry Pi에 모니터가 연결되어 있으면 모니터에서 시작 프로세스를 볼 수 있습니다.

5. 디바이스의 IP 주소 찾기:

   - Raspberry Pi에 HDMI 모니터를 연결한 경우 모니터에 표시된 메시지에 IP 주소가 나타납니다.
   - Raspberry Pi가 연결된 라우터에 액세스할 수 있는 경우 라우터의 관리 인터페이스에서 주소를 확인할 수 있습니다.

Raspberry Pi의 IP 주소가 있으면 the section called “호스트 컴퓨터 연결” 준비가 된 것입니다.

디바이스에 로컬 호스트 컴퓨터 연결

이 절차에서는 로컬 호스트 컴퓨터의 터미널 프로그램을 사용하여 Raspberry Pi에 연결하고 기본 암호를 변경합니다.

디바이스에 로컬 호스트 컴퓨터를 연결하려면

1. 로컬 호스트 컴퓨터에서 SSH 터미널 프로그램을 엽니다.

   - Windows: PuTTY
   - Linux/macOS: `Terminal`

   > **Note**
   >
   > PuTTY는 Windows에 자동으로 설치되지 않습니다. 컴퓨터에 없으면 다운로드하여 설치해야 할 수 있습니다.

2. 터미널 프로그램을 Raspberry Pi의 IP 주소에 연결하고 기본 자격 증명을 사용하여 로그인합니다.

   ```
   username: pi
   password: raspberry
   ```

3. Raspberry Pi에 로그인한 후 `pi` 사용자의 암호를 변경합니다.

   ```
   passwd
   ```

   프롬프트에 따라 암호를 변경합니다.

   ```
   Changing password for pi.
   Current password: raspberry
   New password: YourNewPassword
   Retype new password: YourNewPassword
   passwd: password updated successfully
   ```

터미널 창에 Raspberry Pi의 명령줄 프롬프트가 표시되고 암호가 변경되면 the section called “필수 소프트웨어 설치 및 확인”으로 진행할 준비가 된 것입니다.

디바이스에 필요한 소프트웨어 설치 및 확인

이 섹션의 절차는 이전 섹션에서 계속되어 Raspberry Pi의 운영 체제를 최신 상태로 유지하고 다음 섹션에서 AWS IoT Device Client를 빌드하고 설치하는 데 사용할 소프트웨어를 Raspberry Pi에 설치합니다.

이 섹션을 완료하면 Raspberry Pi에 최신 운영 체제와 이 학습 경로의 튜토리얼에 필요한 소프트웨어가 설치되고 해당 위치에 맞게 구성됩니다.

필수 장비:

- 이전 섹션의 로컬 개발 및 테스트 환경
- 이전 섹션에서 사용한 Raspberry Pi
- 이전 섹션의 microSD 메모리 카드

> **Note**
>
> Raspberry Pi 모델 3+ 및 Raspberry Pi 모델 4는 이 학습 경로에 설명된 모든 명령을 수행할 수 있습니다. IoT 디바이스가 소프트웨어를 컴파일하거나를 실행할 수 없는 경우 로컬 호스트 컴퓨터에 필요한 컴파일러를 설치하여 소프트웨어를 빌드한 다음 IoT 디바이스로 전송해야 할 AWS Command Line Interface수 있습니다. 디바이스용 소프트웨어를 설치하고 빌드하는 방법에 대한 자세한 내용은 디바이스 소프트웨어의 설명서를 참조하세요.

이 섹션의 절차:

- 운영 체제 소프트웨어 업데이트
- 필요한 애플리케이션 및 라이브러리 설치
- (선택 사항) microSD 카드 이미지 저장

운영 체제 소프트웨어 업데이트

이 절차는 운영 체제 소프트웨어를 업데이트합니다.

Raspberry Pi에서 운영 체제 소프트웨어를 업데이트하려면

로컬 호스트 컴퓨터의 터미널 창에서 다음 단계를 수행합니다.

1. 다음 명령을 입력하여 Raspberry Pi의 시스템 소프트웨어를 업데이트합니다.

```
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y autoremove
```

2. Raspberry Pi의 로캘 및 시간대 설정을 업데이트합니다(선택 사항).

   이 명령을 입력하여 디바이스의 로캘 및 시간대 설정을 업데이트합니다.

   ```
   sudo raspi-config
   ```

   a. 디바이스의 로캘을 설정하려면

      i. Raspberry Pi 소프트웨어 구성 도구(raspi-config)(Raspberry Pi Software Configuration Tool (raspi-config)) 화면에서 옵션 5를 선택합니다.

         5 Localisation Options Configure language and regional settings

         Tab 키를 사용하여 <선택>(<Select>)으로 이동하고 space bar를 누릅니다.

      ii. 현지화 옵션 메뉴에서 L1 옵션을 선택합니다.

         L1 Locale Configure language and regional settings

         Tab 키를 사용하여 <선택>(<Select>)으로 이동하고 space bar를 누릅니다.

      iii. 로캘 옵션 목록에서 화살표 키를 사용하여 스크롤하고 space bar를 사용하여 원하는 로캘을 표시하여 Raspberry Pi에 설치하려는 로캘을 선택합니다.

         미국에서는 **en_US.UTF-8**을 선택하는 것이 좋습니다.

      iv. 디바이스의 로케일을 선택한 후 Tab 키를 사용하여 <확인>(<OK>)을 선택한 다음 space bar를 눌러 로캘 구성(Configuring locales) 확인 페이지를 표시합니다.

   b. 디바이스의 시간대를 설정하려면

      i. raspi-config 화면에서 옵션 5를 선택합니다.

         5 Localisation Options Configure language and regional settings

         Tab 키를 사용하여 <선택>(<Select>)으로 이동하고 space bar를 누릅니다.

      ii. 현지화 옵션 메뉴에서 화살표 키를 사용하여 옵션 L2를 선택합니다.

         L2 time zone Configure time zone

Tab 키를 사용하여 <선택>(<Select>)으로 이동하고 space bar를 누릅니다.

iii. tzdata 구성(Configuring tzdata) 메뉴의 목록에서 해당하는 지리적 영역을 선택합니다.

Tab 키를 사용하여 <확인>(<OK>)으로 이동하고 space bar를 누릅니다.

iv. 도시 목록에서 화살표 키를 사용하여 해당 시간대의 도시를 선택합니다.

시간대를 설정하려면 Tab 키를 사용하여 <확인>(<OK>)으로 이동하고 space bar를 누릅니다.

c. 설정 업데이트가 완료되면 Tab 키를 사용하여 <마침>(<Finish>)으로 이동한 후 space bar를 눌러 raspi-config 앱을 닫습니다.

3. 이 명령을 입력하여 Raspberry Pi를 다시 시작합니다.

```
sudo shutdown -r 0
```

4. Raspberry Pi가 다시 시작될 때까지 기다립니다.
5. Raspberry Pi를 다시 시작한 후 로컬 호스트 컴퓨터의 터미널 창을 Raspberry Pi에 다시 연결합니다.

이제 Raspberry Pi 시스템 소프트웨어가 구성되었으며 the section called “애플리케이션 및 라이브러리 설치”로 진행할 준비가 된 것입니다.

필요한 애플리케이션 및 라이브러리 설치

이 절차는 후속 튜토리얼에서 사용하는 애플리케이션 소프트웨어 및 라이브러리를 설치합니다.

Raspberry Pi를 사용 중이거나 IoT 디바이스에서 필요한 소프트웨어를 컴파일할 수 있는 경우 로컬 호스트 컴퓨터의 터미널 창에서 다음 단계를 수행합니다. 로컬 호스트 컴퓨터에서 IoT 디바이스용 소프트웨어를 컴파일해야 하는 경우 디바이스에서 이러한 단계를 수행하는 방법은 IoT 디바이스용 소프트웨어 설명서를 참조하세요.

Raspberry Pi에 어플리케이션 소프트웨어 및 라이브러리를 설치하려면

1. 이 명령을 입력하여 애플리케이션 소프트웨어 및 라이브러리를 설치합니다.

```
sudo apt-get -y install build-essential libssl-dev cmake unzip git python3-pip
```

2. 이 명령을 입력하여 올바른 버전의 소프트웨어가 설치되었는지 확인합니다.

```
gcc --version
cmake --version
openssl version
git --version
```

3. 다음 버전의 애플리케이션 소프트웨어가 설치되어 있는지 확인합니다.

   - gcc: 9.3.0 이상
   - cmake: 3.10.x 이상
   - OpenSSL: 1.1.1 이상
   - git: 2.20.1 이상

Raspberry Pi에 필요한 어플리케이션 소프트웨어의 허용 가능한 버전이 있는 경우 the section called “(선택 사항) microSD 이미지 저장”으로 진행할 준비가 된 것입니다.

(선택 사항) microSD 카드 이미지 저장

이 학습 경로의 튜토리얼 전체에 Raspberry Pi의 microSD 카드 이미지 사본을 로컬 호스트 컴퓨터의 파일에 저장하는 절차가 있습니다. 권장되지만 필수 태스크는 아닙니다. 제안된 위치에 microSD 카드 이미지를 저장하면 이 학습 경로에서 저장 시점 이전의 절차를 건너뛸 수 있으므로 다시 시도해야 할 경우 시간을 절약할 수 있습니다. microSD 카드 이미지를 주기적으로 저장하지 않으면 microSD 카드가 손상되었거나 실수로 앱 또는 해당 설정을 잘못 구성한 경우 처음부터 학습 경로의 튜토리얼을 다시 시작해야 할 수 있습니다.

이 시점에서 Raspberry Pi의 microSD 카드에는 업데이트된 OS와 기본 어플리케이션 소프트웨어가 로드되어 있습니다. 지금 microSD 카드의 내용을 파일에 저장하면 이전 단계를 완료하는 데 걸리는 시간을 절약할 수 있습니다. 디바이스의 microSD 카드 이미지의 현재 이미지가 있으면 소프트웨어를 처음부터 설치하고 업데이트할 필요 없이 이 시점부터 시작하여 튜토리얼 또는 절차를 계속하거나 다시 시도할 수 있습니다.

파일에 microSD 카드 이미지를 저장하려면

1. 이 명령을 입력하여 Raspberry Pi를 종료합니다.

   ```
   sudo shutdown -h 0
   ```

2. Raspberry Pi가 완전히 종료되면 전원을 분리합니다.
3. Raspberry Pi에서 microSD 카드를 제거합니다.

4. 로컬 호스트 컴퓨터에서
    a. microSD 카드를 삽입합니다.
    b. SD 카드 이미징 도구를 사용하여 microSD 카드의 이미지를 파일에 저장합니다.
    c. microSD 카드의 이미지를 저장한 후 로컬 호스트 컴퓨터에서 카드를 꺼냅니다.
5. Raspberry Pi의 전원을 분리한 상태에서 microSD 카드를 Raspberry Pi에 삽입합니다.
6. Raspberry Pi에 전원을 공급합니다.
7. 1분 정도 기다린 후 로컬 호스트 컴퓨터에서 Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창을 다시 연결한 다음 Raspberry Pi에 로그인합니다.

디바이스 테스트 및 Amazon CA 인증서 저장

이 섹션의 절차는 이전 섹션에서 계속되어 AWS Command Line Interface 및 연결을 인증하는 데 사용되는 인증 기관 인증서를 설치합니다 AWS IoT Core.

이 섹션을 완료하면 Raspberry Pi에 AWS IoT Device Client를 설치하는 데 필요한 시스템 소프트웨어가 있고 인터넷에 연결되어 있음을 알 수 있습니다.

필수 장비:

- 이전 섹션의 로컬 개발 및 테스트 환경
- 이전 섹션에서 사용한 Raspberry Pi
- 이전 섹션의 microSD 메모리 카드

이 섹션의 절차:

- 설치 AWS Command Line Interface
- 자격 AWS 계정 증명 구성
- Amazon 루트 CA 인증서 다운로드
- (선택 사항) microSD 카드 이미지 저장

설치 AWS Command Line Interface

이 절차에서는 Raspberry Pi AWS CLI 에를 설치합니다.

Raspberry Pi를 사용 중이거나 IoT 장치에서 소프트웨어를 컴파일할 수 있는 경우 로컬 호스트 컴퓨터의 터미널 창에서 다음 단계를 수행합니다. 로컬 호스트 컴퓨터에서 IoT 디바이스용 소프트웨어를 컴

파일해야 하는 경우 IoT 디바이스에 대한 소프트웨어 설명서에서 필요한 라이브러리에 대한 정보를 검토합니다.

Raspberry Pi AWS CLI 에를 설치하려면

1. 다음 명령을 실행하여 AWS CLI를 다운로드하고 설치합니다.

```
export PATH=$PATH:~/.local/bin # configures the path to include the directory with
 the AWS CLI
git clone https://github.com/aws/aws-cli.git # download the AWS CLI code from
 GitHub
cd aws-cli && git checkout v2 # go to the directory with the repo and checkout
 version 2
pip3 install -r requirements.txt # install the prerequisite software
```

2. 이 명령을 실행하여를 설치합니다 AWS CLI. 이 명령을 완료하는 데 최대 15분이 소요될 수 있습니다.

```
pip3 install . # install the AWS CLI
```

3. 이 명령을 실행하여 올바른 버전의가 설치 AWS CLI 되었는지 확인합니다.

```
aws --version
```

   의 버전은 2.2 이상이어야 AWS CLI 합니다.

에 현재 버전이 AWS CLI 표시되면 로 계속 진행할 준비가 된 것입니다the section called “계정 자격 증명 구성”.

자격 AWS 계정 증명 구성

이 절차에서는 자격 AWS 계정 증명을 얻고 Raspberry Pi에서 사용할 자격 증명을 추가합니다.

디바이스에 AWS 계정 자격 증명을 추가하려면

1. 에서 액세스 키 ID 및 보안 액세스 키를 가져 AWS 계정 와 디바이스 AWS CLI 에서를 인증합니다.

   AWS IAM을 처음 사용하는 경우 https://aws.amazon.com/premiumsupport/knowledge-center/create-access-key/ 콘솔에서 실행하여 디바이스에서 사용할 IAM 자격 증명을 AWS 생성하는 AWS 프로세스를 설명합니다.

2. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 디바이스의 액세스 키 ID(Access Key ID) 및 비밀 액세스 키(Secret Access Key) 자격 증명을 사용하여 다음을 수행합니다.

   a. 다음 명령을 사용하여 AWS 구성 앱을 실행합니다.

   ```
   aws configure
   ```

   b. 메시지가 표시되면 자격 증명 및 구성 정보를 입력합니다.

   ```
   AWS Access Key ID: your Access Key ID
   AWS Secret Access Key: your Secret Access Key
   Default region name: your AWS ## code
   Default output format: json
   ```

3. 이 명령을 실행하여 AWS 계정 및 AWS IoT Core 엔드포인트에 대한 디바이스의 액세스를 테스트합니다.

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

   이 예제와 같이 AWS 계정특정 AWS IoT 데이터 엔드포인트를 반환해야 합니다.

   ```
   {
       "endpointAddress": "a3EXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

AWS 계정특정 AWS IoT 데이터 엔드포인트가 표시되면 Raspberry Pi에를 계속할 수 있는 연결 및 권한이 있습니다the section called “Amazon 루트 CA 인증서 다운로드”.

> **Important**
>
> 이제 자격 AWS 계정 증명이 Raspberry Pi의 microSD 카드에 저장됩니다. 이렇게 하면 향후 사용자 및이 자습서에서 생성할 소프트웨어와의 상호 작용이 AWS 쉬워지지만 기본적으로이 단계 이후에 만든 microSD 카드 이미지에도 저장되고 복제됩니다.
> 자격 AWS 계정 증명의 보안을 보호하려면 microSD 카드 이미지를 더 저장하기 전에를 `aws configure` 다시 실행하고 액세스 키 ID 및 보안 액세스 키에 임의의 문자를 입력하여 자격 증명을 삭제하여 자격 AWS 계정 증명이 손상되지 않도록 하는 것이 좋습니다.
> 실수로 자격 증명을 저장한 경우 AWS IAM 콘솔에서 자격 AWS 계정 증명을 비활성화할 수 있습니다.

Amazon 루트 CA 인증서 다운로드

이 절차는 Amazon 루트 인증 기관(CA)의 인증서 사본을 다운로드하고 저장합니다. 이 인증서를 다운로드하면 다음 튜토리얼에서 사용할 수 있도록 저장되며 AWS 서비스와의 디바이스 연결도 테스트합니다.

Amazon 루트 CA 인증서를 다운로드하고 저장하려면

1. 이 명령을 실행하여 인증서 디렉터리를 생성합니다.

   ```
   mkdir ~/certs
   ```

2. 이 명령을 실행하여 Amazon Root CA 인증서를 다운로드합니다.

   ```
   curl -o ~/certs/AmazonRootCA1.pem https://www.amazontrust.com/repository/
   AmazonRootCA1.pem
   ```

3. 이 명령을 실행하여 인증서 디렉터리와 해당 파일에 대한 액세스를 설정합니다.

   ```
   chmod 745 ~
   chmod 700 ~/certs
   chmod 644 ~/certs/AmazonRootCA1.pem
   ```

4. 이 명령을 실행하여 새 디렉터리에서 CA 인증서 파일을 확인합니다.

   ```
   ls -l ~/certs
   ```

   다음과 같은 항목이 표시되어야 합니다. 날짜와 시간은 다릅니다. 그러나 파일 크기와 기타 모든 정보는 여기에 표시된 것과 동일해야 합니다.

   ```
   -rw-r--r-- 1 pi pi 1188 Oct 28 13:02 AmazonRootCA1.pem
   ```

   파일 크기가 1188이 아닌 경우 curl 명령 파라미터를 확인합니다. 잘못된 파일을 다운로드했을 수 있습니다.

(선택 사항) microSD 카드 이미지 저장

이 시점에서 Raspberry Pi의 microSD 카드에는 업데이트된 OS와 기본 어플리케이션 소프트웨어가 로드되어 있습니다.

파일에 microSD 카드 이미지를 저장하려면

1. 로컬 호스트 컴퓨터의 터미널 창에서 AWS 자격 증명을 지웁니다.

   a. 다음 명령을 사용하여 AWS 구성 앱을 실행합니다.

   ```
   aws configure
   ```

   b. 메시지가 나타나면 자격 증명을 바꿉니다. Enter 키를 눌러 기본 영역 이름(Default region name)과 기본 출력 형식(Default output format)을 그대로 둘 수 있습니다.

   ```
   AWS Access Key ID [****************YT2H]: XYXYXYXYX
   AWS Secret Access Key [****************9plH]: XYXYXYXYX
   Default region name [us-west-2]:
   Default output format [json]:
   ```

2. 이 명령을 입력하여 Raspberry Pi를 종료합니다.

   ```
   sudo shutdown -h 0
   ```

3. Raspberry Pi가 완전히 종료되면 전원 커넥터를 분리합니다.
4. 디바이스에서 microSD 카드를 분리합니다.
5. 로컬 호스트 컴퓨터에서

   a. microSD 카드를 삽입합니다.

   b. SD 카드 이미징 도구를 사용하여 microSD 카드의 이미지를 파일에 저장합니다.

   c. microSD 카드의 이미지를 저장한 후 로컬 호스트 컴퓨터에서 카드를 꺼냅니다.
6. Raspberry Pi의 전원을 분리한 상태에서 microSD 카드를 Raspberry Pi에 삽입합니다.
7. 디바이스에 전원을 공급합니다.
8. 약 1분 후 로컬 호스트 컴퓨터에서 터미널 창 세션을 다시 시작하고 디바이스에 로그인합니다.

   자격 AWS 계정 증명을 아직 다시 입력하지 마세요.

다시 시작하고 Raspberry Pi에 로그인하면 the section called “IoT Device Client 설치 및 구성”으로 진행할 준비가 된 것입니다.

자습서: AWS IoT Device Client 설치 및 구성

이 자습서에서는 AWS IoT Device Client의 설치 및 구성과이 데모 및 기타 데모에서 사용할 AWS IoT 리소스 생성을 안내합니다.

이 자습서를 시작하려면:

- 이전 튜토리얼의 로컬 호스트 컴퓨터와 Raspberry Pi를 준비합니다.

이 자습서는 완료하는 데 약 90분이 소요됩니다.

이 주제를 마치면

- IoT 디바이스는 다른 AWS IoT Device Client 데모에서 바로 사용할 수 있습니다.
- IoT 디바이스를 프로비저닝했습니다 AWS IoT Core.
- 디바이스에 AWS IoT Device Client를 다운로드하고 설치했습니다.
- 이후 튜토리얼에서 사용할 수 있는 디바이스의 microSD 카드 이미지를 저장했습니다.

필수 장비:

- 이전 섹션의 로컬 개발 및 테스트 환경
- 이전 섹션에서 사용한 Raspberry Pi
- 이전 섹션에서 사용한 Raspberry Pi의 microSD 메모리 카드

이 튜토리얼의 절차

- AWS IoT Device Client 다운로드 및 저장
- 에서 Raspberry Pi 프로비저닝 AWS IoT
- 연결을 테스트하도록 AWS IoT Device Client 구성

AWS IoT Device Client 다운로드 및 저장

이 섹션의 절차에서는 AWS IoT Device Client를 다운로드하고 컴파일한 다음 Raspberry Pi에 설치합니다. 설치를 테스트한 후 나중에 튜토리얼을 다시 시도할 때 사용할 수 있도록 Raspberry Pi의 microSD 카드 이미지를 저장할 수 있습니다.

이 섹션의 절차:

- AWS IoT Device Client 다운로드 및 빌드
- 튜토리얼에서 사용하는 디렉터리 생성
- (선택 사항) microSD 카드 이미지 저장

AWS IoT Device Client 다운로드 및 빌드

이 절차에서는 Raspberry Pi에 AWS IoT Device Client를 설치합니다.

Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 다음 명령을 수행합니다.

Raspberry Pi에 AWS IoT Device Client를 설치하려면

1. 다음 명령을 입력하여 Raspberry Pi에서 AWS IoT Device Client를 다운로드하고 빌드합니다.

   ```
   cd ~
   git clone https://github.com/awslabs/aws-iot-device-client aws-iot-device-client
   mkdir ~/aws-iot-device-client/build && cd ~/aws-iot-device-client/build
   cmake ../
   ```

2. 이 명령을 실행하여 AWS IoT Device Client를 빌드합니다. 이 명령을 완료하는 데 최대 15분이 소요될 수 있습니다.

   ```
   cmake --build . --target aws-iot-device-client
   ```

   AWS IoT Device Client 컴파일로 표시되는 경고 메시지는 무시할 수 있습니다.

   이 자습서는 gcc2021년 10월 30일 Raspberry Pi OS(bullseye)의 , 버전(Raspbian 10.2.1-6+rpi1) 10.2.1 20210110, 2021년 5월 7일 Raspberry Pi OS(buster)의 gcc버전(Raspbian 8.3.0-6+rpi1) 8.3.0에 구축된 AWS IoT Device Client로 테스트되었습니다.

3. AWS IoT Device Client 빌드가 완료되면이 명령을 실행하여 테스트합니다.

   ```
   ./aws-iot-device-client --help
   ```

AWS IoT Device Client에 대한 명령줄 도움말이 표시되면 AWS IoT Device Client가 성공적으로 빌드되었으며 사용할 준비가 된 것입니다.

튜토리얼에서 사용하는 디렉터리 생성

이 절차에서는 이 학습 경로의 튜토리얼에서 사용하는 파일을 저장하는 데 사용할 디렉터리를 Raspberry Pi에 생성합니다.

이 학습 경로의 튜토리얼에서 사용하는 디렉터리를 생성하려면

1. 이 명령을 실행하여 필요한 디렉터리를 생성합니다.

```
mkdir ~/dc-configs
mkdir ~/policies
mkdir ~/messages
mkdir ~/certs/testconn
mkdir ~/certs/pubsub
mkdir ~/certs/jobs
```

2. 이 명령을 실행하여 새 디렉터리에 대한 권한을 설정합니다.

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 700 ~/certs/pubsub
chmod 700 ~/certs/jobs
```

이러한 디렉터리를 생성하고 해당 권한을 설정한 후 the section called “(선택 사항) microSD 카드 이미지 저장”으로 진행합니다.

(선택 사항) microSD 카드 이미지 저장

이때 Raspberry Pi의 microSD 카드에는 업데이트된 OS, 기본 애플리케이션 소프트웨어 및 AWS IoT Device Client가 있습니다.

이 연습과 튜토리얼을 다시 시도하기 위해 돌아와서 이 절차에서 저장한 microSD 카드 이미지를 새 microSD 카드에 기록하여 이전 절차를 건너뛰고 the section called “Raspberry Pi 프로비저닝”부터 튜토리얼을 계속할 수 있습니다.

파일에 microSD 카드 이미지를 저장하려면

Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서

1. 자격 AWS 계정 증명이 저장되지 않았는지 확인합니다.

   a. 다음 명령을 사용하여 AWS 구성 앱을 실행합니다.

```
aws configure
```

b. 자격 증명이 저장된 경우(프롬프트에 표시되는 경우) 여기에 표시된 대로 메시지가 나타나면 **XYXYXYXYX** 문자열을 입력합니다. 기본 리전 이름(Default region name)과 기본 출력 형식(Default output format)을 비워 둡니다.

```
AWS Access Key ID [****************YXYX]: XYXYXYXYX
AWS Secret Access Key [****************YXYX]: XYXYXYXYX
Default region name:
Default output format:
```

2. 이 명령을 입력하여 Raspberry Pi를 종료합니다.

```
sudo shutdown -h 0
```

3. Raspberry Pi가 완전히 종료되면 전원 커넥터를 분리합니다.
4. 디바이스에서 microSD 카드를 분리합니다.
5. 로컬 호스트 컴퓨터에서

a. microSD 카드를 삽입합니다.

b. SD 카드 이미징 도구를 사용하여 microSD 카드의 이미지를 파일에 저장합니다.

c. microSD 카드의 이미지를 저장한 후 로컬 호스트 컴퓨터에서 카드를 꺼냅니다.

the section called “Raspberry Pi 프로비저닝”에서 이 microSD 카드를 계속 사용할 수 있습니다.

에서 Raspberry Pi 프로비저닝 AWS IoT

이 섹션의 절차는 및 AWS IoT Device Client가 AWS CLI 설치된 저장된 microSD 이미지로 시작하고 Raspberry Pi를 프로비저닝하는 AWS IoT 리소스 및 디바이스 인증서를 생성합니다 AWS IoT.

Raspberry Pi에 microSD 카드를 설치합니다.

이 절차에서는 필요한 소프트웨어가 로드되고 Raspberry Pi에 구성된 microSD 카드를 설치하고이 학습 경로의 자습서를 계속할 수 AWS 계정 있도록를 구성합니다.

이 학습 경로의 연습 및 튜토리얼에 필요한 소프트웨어가 있는 the section called “(선택 사항) microSD 카드 이미지 저장”의 microSD 카드를 사용합니다.

Raspberry Pi에 microSD 카드를 설치하려면

1. Raspberry Pi의 전원을 분리한 상태에서 microSD 카드를 Raspberry Pi에 삽입합니다.
2. Raspberry Pi에 전원을 공급합니다.
3. 약 1분 후 로컬 호스트 컴퓨터에서 터미널 창 세션을 다시 시작하고 Raspberry Pi에 로그인합니다.
4. 로컬 호스트 컴퓨터의 터미널 창에서 Raspberry Pi의 액세스 키 ID(Access Key ID) 및 비밀 액세스 키(Secret Access Key) 자격 증명을 사용하여

   a. 다음 명령을 사용하여 AWS 구성 앱을 실행합니다.

   ```
   aws configure
   ```

   b. 메시지가 표시되면 자격 AWS 계정 증명 및 구성 정보를 입력합니다.

   ```
   AWS Access Key ID [****************YXYX]: your Access Key ID
   AWS Secret Access Key [****************YXYX]: your Secret Access Key
   Default region name [us-west-2]: your AWS ## code
   Default output format [json]: json
   ```

자격 AWS 계정 증명을 복원한 후에는 로 계속 진행할 준비가 된 것입니다the section called “에서 디바이스 프로비저닝 AWS IoT Core”.

에서 디바이스 프로비저닝 AWS IoT Core

이 섹션의 절차에서는 Raspberry Pi를 프로비저닝하는 AWS IoT 리소스를 생성합니다 AWS IoT. 이러한 리소스를 생성할 때 다양한 정보를 기록하라는 메시지가 나타납니다. 이 정보는 다음 절차에서 AWS IoT Device Client 구성에 사용됩니다.

Raspberry Pi가 함께 작동하려면 프로비저닝해야 AWS IoT합니다. 프로비저닝은 Raspberry Pi를 IoT 디바이스로 지원하는 데 필요한 AWS IoT 리소스를 생성하고 구성하는 프로세스입니다.

Raspberry Pi의 전원을 켜고 다시 시작한 상태에서 로컬 호스트 컴퓨터의 터미널 창을 Raspberry Pi에 연결하고 이 절차를 완료합니다.

이 섹션의 절차:

- 디바이스 인증서 파일 생성 및 다운로드
- AWS IoT 리소스 생성

디바이스 인증서 파일 생성 및 다운로드

이 절차는 이 데모에 대한 디바이스 인증서 파일을 생성합니다.

Raspberry Pi에 대한 디바이스 인증서 파일을 생성하고 다운로드하려면

1. 로컬 호스트 컴퓨터의 터미널 창에서 이 명령을 입력하여 디바이스에 대한 디바이스 인증서 파일을 생성합니다.

```
mkdir ~/certs/testconn
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/testconn/device.pem.crt" \
--public-key-outfile "~/certs/testconn/public.pem.key" \
--private-key-outfile "~/certs/testconn/private.pem.key"
```

   이 명령은 다음과 같은 응답을 반환합니다. 나중에 사용하기 위해 *certificateArn* 값을 적어 둡니다.

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
    "certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
    "certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

2. 이 명령을 입력하여 인증서 디렉터리와 해당 파일에 대한 권한을 설정합니다.

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 644 ~/certs/testconn/*
chmod 600 ~/certs/testconn/private.pem.key
```

3. 이 명령을 실행하여 인증서 디렉터리 및 파일에 대한 권한을 검토합니다.

```
ls -l ~/certs/testconn
```

명령의 출력은 파일 날짜와 시간이 다르다는 점을 제외하고 여기에 표시되는 것과 같아야 합니다.

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

이 시점에서 Raspberry Pi에 디바이스 인증서 파일이 설치되어 있고 the section called “AWS IoT 리소스 생성”을 계속할 수 있습니다.

AWS IoT 리소스 생성

이 절차에서는 디바이스가 AWS IoT 기능 및 서비스에 액세스하는 데 필요한 리소스를 생성 AWS IoT 하여에 디바이스를 프로비저닝합니다.

에서 디바이스를 프로비저닝하려면 AWS IoT

1. 로컬 호스트 컴퓨터의 터미널 창에서 이 명령을 입력하여 AWS 계정의 디바이스 데이터 엔드포인트 주소를 가져옵니다.

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

이전 단계의 명령에서 다음과 같은 응답이 반환됩니다. 나중에 사용하기 위해 *endpointAddress* 값을 적어 둡니다.

```
{
    "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

2. Raspberry Pi에 대한 AWS IoT 사물 리소스를 생성하려면이 명령을 입력합니다.

```
aws iot create-thing --thing-name "DevCliTestThing"
```

AWS IoT 사물 리소스가 생성된 경우 명령은 이와 같은 응답을 반환합니다.

```
{
```

```
    "thingName": "DevCliTestThing",
    "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/DevCliTestThing",
    "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. 터미널 창에서

   a. nano 등의 텍스트 편집기를 엽니다.

   b. 이 JSON 정책 문서를 복사하여 열려 있는 텍스트 편집기에 붙여 넣습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

> **Note**
>
> 이 정책 문서는 모든 리소스에 연결, 수신, 게시 및 구독을 위한 권한을 충분히 부여합니다. 일반적으로 정책은 특정 리소스에만 특정 작업을 수행할 수 있는 권한만 부여합니다. 그러나 초기 디바이스 연결 테스트의 경우 이 테스트 중 액세스 문제 발생 가능성을 최소화하기 위해 이 지나치게 일반적이고 허용적인 정책이 사용됩니다. 후속 튜토리얼에서는 정책 설계의 더 나은 사례를 보여주기 위해 보다 좁은 범위의 정책 문서를 사용할 것입니다.

   c. 텍스트 편집기에서 파일을 **~/policies/dev_cli_test_thing_policy.json**으로 저장합니다.

4. 이전 단계의 정책 문서를 사용하여 AWS IoT 정책을 생성하려면이 명령을 실행합니다.

```
aws iot create-policy \
--policy-name "DevCliTestThingPolicy" \
--policy-document "file://~/policies/dev_cli_test_thing_policy.json"
```

정책이 생성되면 명령에서 다음과 같은 응답을 반환합니다.

```
{
    "policyName": "DevCliTestThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/DevCliTestThingPolicy",
    "policyDocument": "{\n    \"Version\": \"2012-10-17\",       \n    \"Statement
\": [\n        {\n            \"Effect\": \"Allow\",\n            \"Action\": [\n
                \"iot:Publish\",\n                \"iot:Subscribe\",\n
    \"iot:Receive\",\n                \"iot:Connect\"\n            ],\n
 \"Resource\": [\n                \"*\"\n            ]\n        }\n    ]\n}\n",
    "policyVersionId": "1"
}
```

5. 이 명령을 실행하여 디바이스 인증서에 정책을 연결합니다. *certificateArn*을 이전에 저장한 certificateArn 값으로 바꿉니다.

```
aws iot attach-policy \
--policy-name "DevCliTestThingPolicy" \
--target "certificateArn"
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

6. 이 명령을 실행하여 디바이스 인증서를 AWS IoT 사물 리소스에 연결합니다. *certificateArn*을 이전에 저장한 certificateArn 값으로 바꿉니다.

```
aws iot attach-thing-principal \
--thing-name "DevCliTestThing" \
--principal "certificateArn"
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

에서 디바이스를 성공적으로 프로비저닝 AWS IoT했으면 로 계속 진행할 준비가 된 것입니다the section called “연결을 테스트하도록 Device Client 구성”.

연결을 테스트하도록 AWS IoT Device Client 구성

이 섹션의 절차는 Raspberry Pi에서 MQTT 메시지를 게시하도록 AWS IoT Device Client를 구성합니다.

이 섹션의 절차:

- 구성 파일 생성
- MQTT 테스트 클라이언트 열기
- AWS IoT 디바이스 클라이언트 실행

구성 파일 생성

이 절차에서는 AWS IoT Device Client를 테스트하는 구성 파일을 생성합니다.

AWS IoT Device Client를 테스트하기 위한 구성 파일을 생성하려면

- Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서

  a. 이 명령을 입력하여 구성 파일에 대한 디렉터리를 생성하고 디렉터리에 대한 권한을 설정합니다.

  ```
  mkdir ~/dc-configs
  chmod 745 ~/dc-configs
  ```

  b. nano 등의 텍스트 편집기를 엽니다.

  c. 이 JSON 문서를 복사하여 열려 있는 텍스트 편집기에 붙여 넣습니다.

  ```
  {
    "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
    "cert": "~/certs/testconn/device.pem.crt",
    "key": "~/certs/testconn/private.pem.key",
    "root-ca": "~/certs/AmazonRootCA1.pem",
    "thing-name": "DevCliTestThing",
    "logging": {
      "enable-sdk-logging": true,
      "level": "DEBUG",
      "type": "STDOUT",
      "file": ""
    },
    "jobs": {
  ```

```
    "enabled": false,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

d. ##### 값을에서 찾 AWS 계정 은의 디바이스 데이터 엔드포인트로 바꿉니다the section called “에서 디바이스 프로비저닝 AWS IoT Core”.

e. 텍스트 편집기에서 파일을 **~/dc-configs/dc-testconn-config.json**으로 저장합니다.

f. 이 명령을 실행하여 새로운 구성 파일에 대한 권한을 설정합니다.

```
chmod 644 ~/dc-configs/dc-testconn-config.json
```

파일을 저장한 후에는 the section called “MQTT 테스트 클라이언트 열기”로 진행할 준비가 된 것입니다.

MQTT 테스트 클라이언트 열기

이 절차는 AWS IoT 콘솔에서 MQTT 테스트 클라이언트가 실행될 때 AWS IoT Device Client가 게시하는 MQTT 메시지를 구독하도록 준비합니다.

모든 MQTT 메시지를 구독하도록 MQTT 테스트 클라이언트(MQTT test client)를 준비하려면

1. 로컬 호스트 컴퓨터의 AWS IoT 콘솔에서 MQTT 테스트 클라이언트(MQTT test client)를 선택합니다.
2. 주제 구독 탭의 주제 필터에 #(파운드 기호 1개)를 입력하고 구독을 선택하여 모든 MQTT 주제를 구독합니다.
3. 구독(Subscriptions) 레이블 아래에 #(파운드 기호 1개)가 표시되는지 확인합니다.

the section called “AWS IoT 디바이스 클라이언트 실행”으로 진행하면서 MQTT 테스트 클라이언트(MQTT test client)가 열려 있는 창을 그대로 둡니다.

AWS IoT 디바이스 클라이언트 실행

이 절차는 AWS IoT Device Client를 실행하여 MQTT 테스트 클라이언트가 수신하고 표시하는 단일 MQTT 메시지를 게시합니다.

AWS IoT Device Client에서 MQTT 메시지를 보내려면

1. 이 절차를 수행하는 동안 Raspberry Pi에 연결된 터미널 창과 MQTT 테스트 클라이언트(MQTT test client)가 있는 창이 모두 표시되는지 확인합니다.
2. 터미널 창에서 다음 명령을 입력하여에서 생성된 구성 파일을 사용하여 AWS IoT Device Client를 실행합니다the section called “구성 파일 생성”.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-testconn-config.json
   ```

   터미널 창에서 AWS IoT Device Client는 정보 메시지와 실행 시 발생하는 오류를 표시합니다.

터미널 창에 오류가 표시되지 않으면 MQTT 테스트 클라이언트(MQTT test client)를 검토합니다.

3. MQTT 테스트 클라이언트의 구독 창에서 `test/dc/pubtopic` 메시지 주제로 Hello World! 메시지가 전송되었음을 확인할 수 있습니다.
4. AWS IoT Device Client에 오류가 표시되지 않고 Hello World!가 MQTT 테스트 클라이언트의 `test/dc/pubtopic` 메시지로 전송된 것을 볼 수 있는 경우 성공적인 연결을 입증한 것입니다.
5. 터미널 창에서 **^C** (Ctrl-C)를 입력하여 AWS IoT Device Client를 중지합니다.

AWS IoT 디바이스 클라이언트가 Raspberry Pi에서 올바르게 실행되고 있고와 통신할 수 있음을 입증한 후에는 로 계속 AWS IoT진행할 수 있습니다the section called “MQTT를 사용하여 디바이스 클라이언트와 통신”.

자습서: AWS IoT Device Client와의 MQTT 메시지 통신 시연

이 자습서에서는 AWS IoT Device Client가 IoT 솔루션에서 일반적으로 사용되는 MQTT 메시지를 구독하고 게시하는 방법을 보여줍니다.

이 자습서를 시작하려면:

- 이전 섹션에서 사용한 대로 로컬 호스트 컴퓨터와 Raspberry Pi를 구성합니다.

  AWS IoT Device Client를 설치한 후 microSD 카드 이미지를 저장한 경우 해당 이미지와 함께 microSD 카드를 Raspberry Pi와 함께 사용할 수 있습니다.
- 이전에이 데모를 실행한 적이 있는 경우 ???를 검토하여 이전 실행에서 생성한 모든 AWS IoT 리소스를 삭제하여 중복 리소스 오류를 방지합니다.

이 자습서는 완료하는 데 약 45분이 소요됩니다.

이 주제를 마치면

- IoT 디바이스가의 MQTT 메시지를 구독 AWS IoT 하고 MQTT 메시지를 게시할 수 있는 다양한 방법을 보여드렸습니다 AWS IoT.

필수 장비:

- 이전 섹션의 로컬 개발 및 테스트 환경
- 이전 섹션에서 사용한 Raspberry Pi

- 이전 섹션에서 사용한 Raspberry Pi의 microSD 메모리 카드

이 튜토리얼의 절차

- MQTT 메시지 통신 시연을 위해 Raspberry Pi 준비
- AWS IoT Device Client를 사용하여 메시지 게시 시연
- AWS IoT Device Client를 사용하여 메시지 구독 시연

MQTT 메시지 통신 시연을 위해 Raspberry Pi 준비

이 절차에서는 AWS IoT 및 Raspberry Pi에 리소스를 생성하여 AWS IoT Device Client를 사용한 MQTT 메시지 통신을 보여줍니다.

이 섹션의 절차:

- 인증서 파일을 생성하여 MQTT 통신을 시연합니다.
- MQTT 통신을 시연하기 위해 디바이스 프로비저닝
- MQTT 통신을 보여주도록 AWS IoT Device Client 구성 파일 및 MQTT 테스트 클라이언트 구성

인증서 파일을 생성하여 MQTT 통신을 시연합니다.

이 절차는 이 데모에 대한 디바이스 인증서 파일을 생성합니다.

Raspberry Pi에 대한 디바이스 인증서 파일을 생성하고 다운로드하려면

1. 로컬 호스트 컴퓨터의 터미널 창에서 다음 명령을 입력하여 디바이스에 대한 디바이스 인증서 파일을 만듭니다.

   ```
   mkdir ~/certs/pubsub
   aws iot create-keys-and-certificate \
   --set-as-active \
   --certificate-pem-outfile "~/certs/pubsub/device.pem.crt" \
   --public-key-outfile "~/certs/pubsub/public.pem.key" \
   --private-key-outfile "~/certs/pubsub/private.pem.key"
   ```

   이 명령은 다음과 같은 응답을 반환합니다. 나중에 사용할 수 있도록 *certificateArn* 값을 저장합니다.

   ```
   {
   ```

```
"certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. 이 명령을 입력하여 인증서 디렉터리와 해당 파일에 대한 권한을 설정합니다.

```
chmod 700 ~/certs/pubsub
chmod 644 ~/certs/pubsub/*
chmod 600 ~/certs/pubsub/private.pem.key
```

3. 이 명령을 실행하여 인증서 디렉터리 및 파일에 대한 권한을 검토합니다.

```
ls -l ~/certs/pubsub
```

명령의 출력은 파일 날짜와 시간이 다르다는 점을 제외하고 여기에 표시되는 것과 같아야 합니다.

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

4. 이 명령을 입력하여 로그 파일의 디렉터리를 생성합니다.

```
mkdir ~/.aws-iot-device-client
mkdir ~/.aws-iot-device-client/log
chmod 745 ~/.aws-iot-device-client/log
echo " " > ~/.aws-iot-device-client/log/aws-iot-device-client.log
echo " " > ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
chmod 600 ~/.aws-iot-device-client/log/*
```

MQTT 통신을 시연하기 위해 디바이스 프로비저닝

이 섹션에서는에서 Raspberry Pi를 프로비저닝하는 AWS IoT 리소스를 생성합니다 AWS IoT.

AWS IoT에서 디바이스를 프로비저닝하려면

1. 로컬 호스트 컴퓨터의 터미널 창에서 이 명령을 입력하여 AWS 계정의 디바이스 데이터 엔드포인트 주소를 가져옵니다.

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

   엔드포인트 값은 이전 튜토리얼에서 이 명령을 실행한 이후로 변경되지 않았습니다. 여기에서 명령을 다시 실행하면 데이터 엔드포인트 값을 쉽게 찾아 이 튜토리얼에서 사용하는 구성 파일에 붙여 넣을 수 있습니다.

   이전 단계의 명령에서 다음과 같은 응답이 반환됩니다. 나중에 사용하기 위해 *endpointAddress* 값을 적어 둡니다.

   ```
   {
   "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Raspberry Pi에 대한 새 AWS IoT 사물 리소스를 생성하려면이 명령을 입력합니다.

   ```
   aws iot create-thing --thing-name "PubSubTestThing"
   ```

   AWS IoT 사물 리소스는 클라우드에서 디바이스를 가상으로 표현하기 때문에에서 여러 용도로 AWS IoT 사용할 여러 사물 리소스를 생성할 수 있습니다. 모두 동일한 물리적 IoT 디바이스에서 사용하여 디바이스의 다양한 속성을 나타낼 수 있습니다.

   이 튜토리얼에서는 Raspberry Pi를 표현하기 위해 한 번에 하나의 사물 리소스만 사용합니다. 이렇게 하면이 자습서에서는 다양한 데모를 나타내므로 데모용 AWS IoT 리소스를 생성한 후 돌아가서 각각에 대해 특별히 생성한 리소스를 사용하여 데모를 반복할 수 있습니다.

   AWS IoT 사물 리소스가 생성된 경우 명령은 이와 같은 응답을 반환합니다.

   ```
   {
   "thingName": "PubSubTestThing",
   "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/PubSubTestThing",
   "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
   ```

```
}
```

3. 터미널 창에서

   a. nano 등의 텍스트 편집기를 엽니다.

   b. 이 JSON 문서를 복사하여 열려 있는 텍스트 편집기에 붙여 넣습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/PubSubTestThing"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/
subtopic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic"
            ]
        }
    ]
}
```

c. 편집기의 정책 문서의 각 Resource 섹션에서 *us-west-2:57EXAMPLE833*을 , 콜론 문자(:) 및 12자리 AWS 계정 숫자 AWS 리전로 바꿉니다.

d. 텍스트 편집기에서 파일을 **~/policies/pubsub_test_thing_policy.json**으로 저장합니다.

4. 이전 단계의 정책 문서를 사용하여 AWS IoT 정책을 생성하려면이 명령을 실행합니다.

```
aws iot create-policy \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_test_thing_policy.json"
```

정책이 생성되면 명령에서 다음과 같은 응답을 반환합니다.

```
{
                                    "policyName": "PubSubTestThingPolicy",
                                    "policyArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
                                    "policyDocument": "{\n\"Version\":
 \"2012-10-17\",       \n\"Statement\": [\n{\n\"Effect\": \"Allow\",\n
\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
                                    "policyVersionId": "1"
                                    }
```

5. 이 명령을 실행하여 디바이스 인증서에 정책을 연결합니다. *certificateArn*을 이 섹션에서 이전에 저장한 certificateArn 값으로 바꿉니다.

```
aws iot attach-policy \
```

```
--policy-name "PubSubTestThingPolicy" \
--target "certificateArn"
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

6. 이 명령을 실행하여 AWS IoT 사물 리소스에 디바이스 인증서를 연결합니다. *certificateArn*을 이 섹션에서 이전에 저장한 certificateArn 값으로 바꿉니다.

```
aws iot attach-thing-principal \
--thing-name "PubSubTestThing" \
--principal "certificateArn"
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

에서 디바이스를 성공적으로 프로비저닝하면 로 계속 진행할 준비가 된 AWS IoT것입니다the section called “Device Client 구성 파일 및 MQTT 클라이언트 구성”.

MQTT 통신을 보여주도록 AWS IoT Device Client 구성 파일 및 MQTT 테스트 클라이언트 구성

이 절차에서는 AWS IoT Device Client를 테스트하는 구성 파일을 생성합니다.

AWS IoT Device Client를 테스트하기 위한 구성 파일을 생성하려면

1. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서

   a. nano 등의 텍스트 편집기를 엽니다.

   b. 이 JSON 문서를 복사하여 열려 있는 텍스트 편집기에 붙여 넣습니다.

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/pubsub/device.pem.crt",
  "key": "~/certs/pubsub/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "PubSubTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
```

```
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. ##### 값을에서 찾 AWS 계정 은의 디바이스 데이터 엔드포인트로 바꿉니다the section called “에서 디바이스 프로비저닝 AWS IoT Core”.

d. 텍스트 편집기에서 파일을 **~/dc-configs/dc-pubsub-config.json**으로 저장합니다.

e. 이 명령을 실행하여 새로운 구성 파일에 대한 권한을 설정합니다.

```
chmod 644 ~/dc-configs/dc-pubsub-config.json
```

2. 모든 MQTT 메시지를 구독하도록 MQTT 테스트 클라이언트(MQTT test client)를 준비하려면

   a. 로컬 호스트 컴퓨터의 AWS IoT 콘솔에서 MQTT 테스트 클라이언트(MQTT test client)를 선택합니다.

   b. 주제 구독(Subscribe to a topic) 탭의 주제 필터(Topic filter)에 #(파운드 기호 1개)를 입력하고 구독(Subscribe)을 선택합니다.

   c. 구독(Subscriptions) 레이블 아래에 #(파운드 기호 1개)가 표시되는지 확인합니다.

   이 자습서를 진행하면서 MQTT 테스트 클라이언트(MQTT test client)가 열려 있는 창을 그대로 둡니다.

파일을 저장하고 MQTT 테스트 클라이언트(MQTT test client)를 구성한 후 the section called “IoT Device Client로 메시지 게시”으로 진행할 준비가 된 것입니다.

AWS IoT Device Client를 사용하여 메시지 게시 시연

이 섹션의 절차는 AWS IoT Device Client가 기본 및 사용자 지정 MQTT 메시지를 보내는 방법을 보여줍니다.

이러한 연습을 위해 이전 단계에서 생성한 정책의 이러한 정책 설명은 Raspberry Pi에 다음 작업을 수행할 수 있는 권한을 부여합니다.

- **`iot:Connect`**

  AWS IoT Device Client를 실행하는 Raspberry Pi`PubSubTestThing`인 라는 클라이언트에 연결할 권한을 부여합니다.

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Connect"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing"
    ]
  }
  ```

- **`iot:Publish`**

Raspberry Pi에 `test/dc/pubtopic`의 MQTT 주제가 있는 메시지를 게시할 수 있는 권한을 부여합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Publish"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic"
  ]
}
```

`iot:Publish` 작업은 리소스 배열에 나열된 MQTT 주제에 게시할 수 있는 권한을 부여합니다. 이러한 메시지의 내용은 정책 설명에 의해 제어되지 않습니다.

AWS IoT Device Client를 사용하여 기본 메시지 게시

이 절차는 AWS IoT Device Client를 실행하여 MQTT 테스트 클라이언트가 수신하고 표시하는 단일 기본 MQTT 메시지를 게시합니다.

AWS IoT Device Client에서 기본 MQTT 메시지를 보내려면

1. 이 절차를 수행하는 동안 Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창과 MQTT 테스트 클라이언트(MQTT test client)가 있는 창이 모두 표시되는지 확인합니다.
2. 터미널 창에서 다음 명령을 입력하여에서 생성된 구성 파일을 사용하여 AWS IoT Device Client를 실행합니다the section called “구성 파일 생성”.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-config.json
   ```

   터미널 창에서 AWS IoT Device Client는 정보 메시지와 실행 시 발생하는 오류를 표시합니다.

   터미널 창에 오류가 표시되지 않으면 MQTT 테스트 클라이언트(MQTT test client)를 검토합니다.
3. MQTT 테스트 클라이언트의 구독 창에서 `test/dc/pubtopic` 메시지 주제로 Hello World! 메시지가 전송되었음을 확인할 수 있습니다.
4. AWS IoT Device Client에 오류가 표시되지 않고 Hello World!가 MQTT 테스트 클라이언트의 `test/dc/pubtopic` 메시지로 전송된 것을 볼 수 있는 경우 성공적인 연결을 입증한 것입니다.

5. 터미널 창에서 **^C** (Ctrl-C)를 입력하여 AWS IoT Device Client를 중지합니다.

AWS IoT Device Client가 기본 MQTT 메시지를 게시했음을 입증한 후 로 계속 진행할 수 있습니다the section called “사용자 지정 MQTT 메시지 게시”.

AWS IoT Device Client를 사용하여 사용자 지정 메시지 게시

이 섹션의 절차에서는 사용자 정의 MQTT 메시지를 생성한 다음 AWS IoT Device Client를 실행하여 MQTT 테스트 클라이언트(MQTT test client)가 수신하고 표시할 수 있도록 사용자 정의 MQTT 메시지를 한 번 게시합니다.

AWS IoT Device Client에 대한 사용자 지정 MQTT 메시지 생성

Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 다음 단계를 수행합니다.

AWS IoT Device Client가 게시할 사용자 지정 메시지를 생성하려면

1. 터미널 창에서 nano 등의 텍스트 편집기를 엽니다.
2. 텍스트 편집기에 다음 JSON 문서를 복사하여 붙여 넣습니다. 이는 AWS IoT Device Client가 게시하는 MQTT 메시지 페이로드입니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

3. 텍스트 편집기의 내용을 **~/messages/sample-ws-message.json**으로 저장합니다.
4. 다음 명령을 입력하여 방금 생성한 메시지 파일의 권한을 설정합니다.

```
chmod 600 ~/messages/*
```

AWS IoT Device Client가 사용자 지정 메시지를 전송하는 데 사용할 구성 파일을 생성하려면

1. 터미널 창의와 같은 텍스트 편집기에서 기존 AWS IoT Device Client 구성 파일을 nano엽니다**~/dc-configs/dc-pubsub-config.json**.

2. 다음과 같이 `samples` 객체를 편집합니다. 이 파일의 다른 부분은 변경할 필요가 없습니다.

```
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "~/messages/sample-ws-message.json",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

3. 텍스트 편집기의 내용을 **~/dc-configs/dc-pubsub-custom-config.json**으로 저장합니다.
4. 이 명령을 실행하여 새로운 구성 파일에 대한 권한을 설정합니다.

```
chmod 644 ~/dc-configs/dc-pubsub-custom-config.json
```

AWS IoT Device Client를 사용하여 사용자 지정 MQTT 메시지 게시

이 변경 사항은 MQTT 메시지 페이로드의 내용에만 영향을 주므로 현재 정책이 계속 작동합니다. 그러나 MQTT 주제(`~/dc-configs/dc-pubsub-custom-config.json`의 `publish-topic` 값으로 정의됨)가 변경된 경우 Raspberry Pi가 새 MQTT 주제에 게시할 수 있도록 `iot::Publish` 정책 설명도 수정해야 합니다.

AWS IoT Device Client에서 MQTT 메시지를 보내려면

1. 이 절차를 수행하는 동안 터미널 창과 MQTT 테스트 클라이언트(MQTT test client)가 있는 창이 모두 표시되는지 확인합니다. 또한 MQTT 테스트 클라이언트(MQTT test client)가 여전히 # 주제 필터를 구독하고 있는지 확인합니다. 그렇지 않은 경우 # 주제 필터를 다시 구독합니다.
2. 터미널 창에서 이 명령을 입력하여 the section called “구성 파일 생성”에서 생성한 config 파일을 사용하여 AWS IoT Device Client를 실행합니다.

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
```

   터미널 창에서 AWS IoT Device Client는 정보 메시지와 실행 시 발생하는 오류를 표시합니다.

   터미널 창에 오류가 표시되지 않으면 MQTT 테스트 클라이언트(MQTT test client)를 검토합니다.
3. MQTT 테스트 클라이언트의 구독 창에서 `test/dc/pubtopic` 메시지 주제로 사용자 정의 메시지 페이로드가 전송되었음을 확인할 수 있습니다.

4. AWS IoT Device Client에 오류가 표시되지 않고 MQTT 테스트 클라이언트의 메시지에 게시한 사용자 지정 `test/dc/pubtopic` 메시지 페이로드가 표시되면 사용자 지정 메시지를 성공적으로 게시한 것입니다.
5. 터미널 창에서 **^C** (Ctrl-C)를 입력하여 AWS IoT Device Client를 중지합니다.

AWS IoT Device Client가 사용자 지정 메시지 페이로드를 게시했음을 입증한 후 로 계속 진행할 수 있습니다the section called “IoT Device Client를 사용하여 메시지 구독”.

AWS IoT Device Client를 사용하여 메시지 구독 시연

이 섹션에서는 두 가지 유형의 메시지 구독을 보여줍니다.

- 단일 주제 구독
- 와일드카드 주제 구독

이러한 연습을 위해 생성한 정책의 이러한 정책 설명은 Raspberry Pi에 다음 작업을 수행할 수 있는 권한을 부여합니다.

- **`iot:Receive`**

  AWS IoT Device Client에 `Resource` 객체에 이름이 지정된 주제와 일치하는 MQTT 주제를 수신할 수 있는 권한을 부여합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Receive"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic"
  ]
}
```

- **`iot:Subscribe`**

  AWS IoT Device Client에 `Resource` 객체에 이름이 지정된 것과 일치하는 MQTT 주제 필터를 구독할 수 있는 권한을 부여합니다.

```
{
```

```
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic"
      ]
    }
```

단일 MQTT 메시지 주제 구독

이 절차에서는 AWS IoT Device Client가 MQTT 메시지를 구독하고 로깅하는 방법을 보여줍니다.

Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 **~/dc-configs/dc-pubsub-custom-config.json**의 내용을 나열하거나 텍스트 편집기에서 파일을 열어 내용을 검토합니다. 다음과 같은 `samples` 객체를 찾습니다.

```
"samples": {
  "pub-sub": {
    "enabled": true,
    "publish-topic": "test/dc/pubtopic",
    "publish-file": "~/messages/sample-ws-message.json",
    "subscribe-topic": "test/dc/subtopic",
    "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

`subscribe-topic` 값은 AWS IoT Device Client가 실행 시 구독할 MQTT 주제입니다. AWS IoT Device Client는이 구독에서 수신한 메시지 페이로드를 `subscribe-file` 값에 라는 파일에 씁니다.

AWS IoT Device Client에서 MQTT 메시지 주제를 구독하려면

1. 이 절차를 수행하는 동안 터미널 창과 MQTT 테스트 클라이언트(MQTT test client)가 있는 창이 모두 표시되는지 확인합니다. 또한 MQTT 테스트 클라이언트(MQTT test client)가 여전히 # 주제 필터를 구독하고 있는지 확인합니다. 그렇지 않은 경우 # 주제 필터를 다시 구독합니다.
2. 터미널 창에서 다음 명령을 입력하여에서 생성된 구성 파일을 사용하여 AWS IoT Device Client를 실행합니다the section called “구성 파일 생성”.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
   ```

   터미널 창에서 AWS IoT Device Client는 정보 메시지와 실행 시 발생하는 오류를 표시합니다.

터미널 창에 오류가 표시되지 않으면 AWS IoT 콘솔에서 계속합니다.

3. AWS IoT 콘솔의 MQTT 테스트 클라이언트에서 주제에 게시 탭을 선택합니다.
4. 주제 이름(Topic name)에 **test/dc/subtopic**을 입력합니다.
5. 메시지 페이로드(Message payload)에서 메시지 내용을 검토합니다.
6. 게시(Publish)를 선택하여 MQTT 메시지를 게시합니다.
7. 터미널 창에서 다음과 같은 AWS IoT Device Client에서 받은 메시지 항목을 확인합니다.

```
2021-11-10T16:02:20.890Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 45 bytes
```

8. 메시지가 수신되었음을 보여주는 메시지 수신 항목이 표시되면 **^C** (Ctrl-C)를 입력하여 AWS IoT Device Client를 중지합니다.
9. 이 명령을 입력하여 메시지 로그 파일의 끝을 보고 MQTT 테스트 클라이언트(MQTT test client)에서 게시한 메시지를 확인합니다.

```
tail ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

로그 파일에서 메시지를 보고 AWS IoT Device Client가 MQTT 테스트 클라이언트에서 게시한 메시지를 수신했음을 보여줍니다.

와일드카드 문자를 사용하여 여러 MQTT 메시지 주제 구독

이 절차는 AWS IoT Device Client가 와일드카드 문자를 사용하여 MQTT 메시지를 구독하고 로깅하는 방법을 보여줍니다. 이를 위해 다음을 수행합니다.

1. AWS IoT Device Client가 MQTT 주제를 구독하는 데 사용하는 주제 필터를 업데이트합니다.
2. 새 구독을 허용하도록 디바이스에서 사용하는 정책을 업데이트합니다.
3. AWS IoT Device Client를 실행하고 MQTT 테스트 콘솔에서 메시지를 게시합니다.

와일드카드 MQTT 주제 필터를 사용하여 여러 MQTT 메시지 주제를 구독하는 구성 파일을 생성하려면

1. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 편집을 위해 **~/dc-configs/dc-pubsub-custom-config.json**을 열고 `samples` 객체를 찾습니다.
2. 텍스트 편집기에서 `samples` 객체를 찾고 `subscribe-topic` 값을 다음과 같이 업데이트합니다.

```
"samples": {
  "pub-sub": {
    "enabled": true,
    "publish-topic": "test/dc/pubtopic",
    "publish-file": "~/messages/sample-ws-message.json",
    "subscribe-topic": "test/dc/#",
    "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

새 `subscribe-topic` 값은 끝에 MQTT 와일드카드 문자가 있는 MQTT 주제 필터입니다. 이는 `test/dc/`로 시작하는 모든 MQTT 주제에 대한 구독을 설명합니다. AWS IoT Device Client는 이 구독에서 수신한 메시지 페이로드를의 파일에 씁니다`subscribe-file`.

3. 수정된 구성 파일을 **`~/dc-configs/dc-pubsub-wild-config.json`**으로 저장하고 편집기를 종료합니다.

여러 MQTT 메시지 주제 구독 및 수신을 허용하도록 Raspberry Pi에서 사용하는 정책을 수정하려면

1. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 즐겨 사용하는 텍스트 편집기에서 편집을 위해 **`~/policies/pubsub_test_thing_policy.json`**을 연 다음 파일에서 `iot::Subscribe` 및 `iot::Receive` 정책 설명을 찾습니다.
2. `iot::Subscribe` 정책 설명에서 Resource 개체의 문자열을 업데이트하여 `subtopic`을 `*`로 바꾸면 다음과 같이 됩니다.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Subscribe"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*"
  ]
}
```

> **Note**
>
> MQTT 주제 필터 와일드 카드 문자는 +(더하기 기호)와 #(파운드 기호)입니다. 끝에 #이 있는 구독 요청은 # 문자 앞에 오는 문자열(예: 이 경우 `test/dc/`)로 시작하는 모든 주제를 구독합니다.

그러나 이 구독 권한을 부여하는 정책 설명의 리소스 값은 주제 필터 ARN에서 #(파운드 기호) 대신 *(별표)를 사용해야 합니다. 이는 정책 프로세서가 MQTT에서 사용하는 것과 다른 와일드카드 문자를 사용하기 때문입니다.
정책의 주제 및 주제 필터에 와일드카드 문자 사용에 대한 자세한 내용은 MQTT 및 AWS IoT Core 정책에서 와일드카드 문자 사용 섹션을 참조하세요.

3. `iot::Receive` 정책 설명에서 Resource 개체의 문자열을 업데이트하여 `subtopic`을 *로 바꾸면 다음과 같이 됩니다.

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*"
      ]
    }
```

4. 업데이트된 정책 문서를 **~/policies/pubsub_wild_test_thing_policy.json**으로 저장하고 편집기를 종료합니다.

5. 이 명령을 입력하여 새 리소스 정의를 사용하도록 이 튜토리얼의 정책을 업데이트합니다.

```
aws iot create-policy-version \
--set-as-default \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_wild_test_thing_policy.json"
```

명령이 성공하면 다음과 같은 응답이 반환됩니다. `policyVersionId`는 이제 2로, 이 정책의 두 번째 버전임을 나타냅니다.

정책을 성공적으로 업데이트했으면 다음 절차로 진행할 수 있습니다.

```
{
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
    "policyDocument": "{\n  \"Version\": \"2012-10-17\",       \n  \"Statement
\": [\n    {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n
 \"iot:Connect\"\n      ],\n      \"Resource\": [\n        \"arn:aws:iot:us-
west-2:57EXAMPLE833:client/PubSubTestThing\"\n      ]\n    },\n    {\n
 \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Publish\"\n      ],
```

```
\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/
test/dc/pubtopic\"\n      ]\n    },\n    {\n      \"Effect\": \"Allow\",\n
 \"Action\": [\n        \"iot:Subscribe\"\n      ],\n      \"Resource\": [\n
  \"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*\"\n      ]\n    },\n
   {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Receive\"\n
     ],\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/
test/dc/*\"\n      ]\n    }\n  ]\n}\n",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

정책 버전이 너무 많아 새 버전을 저장할 수 없다는 오류가 발생하면 이 명령을 입력하여 정책의 현재 버전을 나열합니다. 이 명령이 반환하는 목록을 검토하여 삭제할 수 있는 정책 버전을 찾습니다.

```
aws iot list-policy-versions --policy-name "PubSubTestThingPolicy"
```

이 명령을 입력하여 더 이상 필요 없는 버전을 삭제합니다. 기본 정책 버전은 삭제할 수 없습니다. 기본 정책 버전은 `isDefaultVersion` 값이 `true`인 버전입니다.

```
aws iot delete-policy-version \
--policy-name "PubSubTestThingPolicy" \
--policy-version-id policyId
```

정책 버전을 삭제한 후 이 단계를 다시 시도합니다.

업데이트된 구성 파일 및 정책을 사용하면 AWS IoT Device Client를 사용하여 와일드카드 구독을 시연할 준비가 된 것입니다.

AWS IoT Device Client가 여러 MQTT 메시지 주제를 구독하고 수신하는 방법을 보여주는 방법

1. MQTT 테스트 클라이언트(MQTT test client)에서 구독을 확인합니다. MQTT 테스트 클라이언트가 **#** 주제 필터를 구독하는 경우 다음 단계로 진행합니다. 그렇지 않은 경우 MQTT 테스트 클라이언트의 주제 구독 탭에 있는 주제 필터에 **#**(파운드 기호)를 입력한 다음 구독을 선택하여 이를 구독합니다.
2. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 이 명령을 입력하여 AWS IoT Device Client를 시작합니다.

```
cd ~/aws-iot-device-client/build
```

```
./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-wild-config.json
```

3. 로컬 호스트 컴퓨터의 터미널 창에서 AWS IoT Device Client 출력을 보면서 MQTT 테스트 클라이언트로 돌아갑니다. 주제에 게시(Publish to a topic) 탭의 주제 이름(Topic name)에 **test/dc/subtopic**을 입력한 다음 게시(Publish)를 선택합니다.
4. 터미널 창에서 다음과 같은 메시지를 찾아 메시지가 수신되었는지 확인합니다.

```
2021-11-10T16:34:20.101Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 76 bytes
```

5. 로컬 호스트 컴퓨터의 터미널 창에서 AWS IoT Device Client 출력을 보면서 MQTT 테스트 클라이언트로 돌아갑니다. 주제에 게시(Publish to a topic) 탭의 주제 이름(Topic name)에 **test/dc/subtopic2**을 입력한 다음 게시(Publish)를 선택합니다.
6. 터미널 창에서 다음과 같은 메시지를 찾아 메시지가 수신되었는지 확인합니다.

```
2021-11-10T16:34:32.078Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 77 bytes
```

7. 두 메시지가 모두 수신되었음을 확인하는 메시지가 표시되면 **^C** (Ctrl-C)를 입력하여 AWS IoT Device Client를 중지합니다.
8. 이 명령을 입력하여 메시지 로그 파일의 끝을 보고 MQTT 테스트 클라이언트(MQTT test client)에서 게시한 메시지를 확인합니다.

```
tail -n 20 ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

> **Note**
>
> 로그 파일에는 메시지 페이로드만 포함됩니다. 수신한 메시지 로그 파일에는 메시지 주제가 기록되지 않습니다.
> 수신된 로그에 AWS IoT Device Client가 게시한 메시지가 표시될 수도 있습니다. 이는 와일드카드 주제 필터에 해당 메시지 주제가 포함되어 있으며, 게시된 메시지가 구독자에게 전송되기 전에 메시지 브로커가 구독 요청을 처리할 수도 있기 때문입니다.

로그 파일의 항목은 메시지가 수신되었음을 보여줍니다. 다른 주제 이름을 사용하여 이 절차를 반복할 수 있습니다. 주제 이름이 `test/dc/`로 시작하는 모든 메시지는 수신되고 로그되어야 합니다. 주제 이름이 다른 텍스트로 시작하는 메시지는 무시됩니다.

AWS IoT Device Client가 MQTT 메시지를 게시하고 구독하는 방법을 보여준 후 로 계속 진행합니다자습서: AWS IoT Device Client를 사용하여 원격 작업(작업) 시연.

자습서: AWS IoT Device Client를 사용하여 원격 작업(작업) 시연

이 튜토리얼에서는 IoT 디바이스에 원격 작업을 전송하는 방법을 보여주기 위해 작업을 구성하고 Raspberry Pi에 배포합니다.

이 자습서를 시작하려면:

- 이전 섹션에서 사용한 대로 로컬 호스트 컴퓨터와 Raspberry Pi를 구성합니다.
- 이전 섹션의 자습서를 완료하지 않은 경우에 AWS IoT Device Client를 설치한 후 저장한 이미지가 있는 microSD 카드와 함께 Raspberry Pi를 사용하여이 자습서를 시도할 수 있습니다(선택 사항) microSD 카드 이미지 저장.
- 이전에이 데모를 실행한 적이 있는 경우 ???를 검토하여 이전 실행에서 생성한 모든 AWS IoT 리소스를 삭제하여 중복 리소스 오류를 방지합니다.

이 자습서는 완료하는 데 약 45분이 소요됩니다.

이 주제를 마치면

- IoT 디바이스가를 AWS IoT Core 사용하여에서 관리하는 원격 작업을 실행할 수 있는 다양한 방법을 보여 주었을 것입니다 AWS IoT .

필수 장비:

- 이전 섹션에서 테스트한 로컬 개발 및 테스트 환경
- 이전 섹션에서 테스트한 Raspberry Pi
- 이전 섹션에서 테스트한 Raspberry Pi의 microSD 메모리 카드

이 튜토리얼의 절차

- 작업을 실행할 Raspberry Pi 준비
- AWS IoT Device Client를 AWS IoT 사용하여에서 작업 생성 및 실행

작업을 실행할 Raspberry Pi 준비

이 섹션의 절차에서는 AWS IoT Device Client를 사용하여 작업을 실행하도록 Raspberry Pi를 준비하는 방법을 설명합니다.

> **Note**
>
> 이러한 절차는 디바이스에 따라 다릅니다. 동시에 여러 디바이스로 이 섹션의 절차를 수행하려면 각 디바이스에 고유한 정책과 고유한 디바이스별 인증서 및 사물 이름이 필요합니다. 각 디바이스에 고유한 리소스를 제공하려면 절차에 설명된 대로 디바이스별 요소를 변경하면서 각 디바이스에 대해 이 절차를 한 번 수행합니다.

이 튜토리얼의 절차

- Raspberry Pi를 프로비저닝하여 직업 시연
- 작업 에이전트를 실행하도록 AWS IoT Device Client 구성

Raspberry Pi를 프로비저닝하여 직업 시연

이 섹션의 절차는 Raspberry Pi에 대한 AWS IoT 리소스 및 디바이스 인증서를 생성 AWS IoT 하여에서 Raspberry Pi를 프로비저닝합니다.

주제

- 디바이스 인증서 파일을 생성하고 다운로드하여 AWS IoT 작업 시연
- AWS IoT 작업을 시연하기 위한 AWS IoT 리소스 생성

디바이스 인증서 파일을 생성하고 다운로드하여 AWS IoT 작업 시연

이 절차는 이 데모에 대한 디바이스 인증서 파일을 생성합니다.

여러 디바이스를 준비하는 경우 각 디바이스에서 이 절차를 수행해야 합니다.

Raspberry Pi에 대한 디바이스 인증서 파일을 생성하고 다운로드하려면

Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서 이 명령을 입력합니다.

1. 다음 명령을 입력하여 디바이스에 대한 디바이스 인증서 파일을 만듭니다.

```
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/jobs/device.pem.crt" \
--public-key-outfile "~/certs/jobs/public.pem.key" \
--private-key-outfile "~/certs/jobs/private.pem.key"
```

이 명령은 다음과 같은 응답을 반환합니다. 나중에 사용할 수 있도록 *certificateArn* 값을 저장합니다.

```
{
"certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. 이 명령을 입력하여 인증서 디렉터리와 해당 파일에 대한 권한을 설정합니다.

```
chmod 700 ~/certs/jobs
chmod 644 ~/certs/jobs/*
chmod 600 ~/certs/jobs/private.pem.key
```

3. 이 명령을 실행하여 인증서 디렉터리 및 파일에 대한 권한을 검토합니다.

```
ls -l ~/certs/jobs
```

명령의 출력은 파일 날짜와 시간이 다르다는 점을 제외하고 여기에 표시되는 것과 같아야 합니다.

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
```

```
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

디바이스 인증서 파일을 Raspberry Pi에 다운로드했으면 the section called “작업에 대한 Raspberry Pi 프로비저닝”으로 진행할 준비가 된 것입니다.

AWS IoT 작업을 시연하기 위한 AWS IoT 리소스 생성

이 디바이스에 대한 AWS IoT 리소스를 생성합니다.

여러 디바이스를 준비하는 경우 각 디바이스에 대해 이 절차를 수행해야 합니다.

AWS IoT에서 디바이스를 프로비저닝하려면

Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서

1. 다음 명령을 입력하여 AWS 계정에 대한 디바이스 데이터 엔드포인트의 주소를 가져옵니다.

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

   엔드포인트 값은 이 명령을 마지막으로 실행한 이후로 변경되지 않았습니다. 여기에서 명령을 다시 실행하면 데이터 엔드포인트 값을 쉽게 찾아 이 자습서에서 사용하는 구성 파일에 붙여 넣을 수 있습니다.

   describe-endpoint 명령은 다음과 같은 응답을 반환합니다. 나중에 사용하기 위해 *endpointAddress* 값을 적어 둡니다.

   ```
   {
   "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. *uniqueThingName*을 디바이스의 고유 이름으로 바꿉니다. 여러 디바이스에서 이 튜토리얼을 수행하려면 각 디바이스에 고유한 이름을 지정합니다. 예: **TestDevice01**, **TestDevice02** 등.

   Raspberry Pi에 대한 새 AWS IoT 사물 리소스를 생성하려면이 명령을 입력합니다.

   ```
   aws iot create-thing --thing-name "uniqueThingName"
   ```

AWS IoT 사물 리소스는 클라우드에서 디바이스를 가상으로 표현하기 때문에에서 여러 용도로 AWS IoT 사용할 여러 사물 리소스를 생성할 수 있습니다. 모두 동일한 물리적 IoT 디바이스에서 사용하여 디바이스의 다양한 속성을 나타낼 수 있습니다.

> **Note**
>
> 여러 장치에 대한 정책을 보호하려는 경우 고정 사물 이름인 *uniqueThingName* 대신 `${iot:Thing.ThingName}`을 사용할 수 있습니다.

이 튜토리얼에서는 디바이스당 한 번에 하나의 사물 리소스만 사용합니다. 이렇게 하면이 자습서에서는 다양한 데모를 나타내므로 데모용 AWS IoT 리소스를 생성한 후 돌아가서 각각에 대해 특별히 생성한 리소스를 사용하여 데모를 반복할 수 있습니다.

AWS IoT 사물 리소스가 생성된 경우 명령은 이와 같은 응답을 반환합니다. 나중에 이 디바이스에서 실행할 작업을 생성할 때 사용할 *thingArn* 값을 기록합니다.

```
{
"thingName": "uniqueThingName",
"thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/uniqueThingName",
"thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. 터미널 창에서

   a. nano 등의 텍스트 편집기를 엽니다.

   b. 이 JSON 문서를 복사하여 열려 있는 텍스트 편집기에 붙여 넣습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/uniqueThingName"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/job/
*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/
subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iot:DescribeJobExecution",
                "iot:GetPendingJobExecutions",
                "iot:StartNextPendingJobExecution",
                "iot:UpdateJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName"
            ]
        }
    ]
}
```

c. 편집기에서 모든 정책 설명 Resource 섹션의 *us-west-2:57EXAMPLE833*을 AWS 리전, 콜론 문자(:) 및 12자리 AWS 계정 번호로 바꿉니다.

d. 편집기의 모든 정책 설명에서 *uniqueThingName*을 이 사물 리소스에 지정한 사물 이름으로 바꿉니다.

e. 텍스트 편집기에서 파일을 **~/policies/jobs_test_thing_policy.json**으로 저장합니다.

   여러 디바이스에 대해 이 절차를 실행하는 경우 파일을 각 디바이스에 이 파일 이름으로 저장합니다.

4. *uniqueThingName*을 디바이스의 사물 이름으로 바꾼 다음이 명령을 실행하여 해당 디바이스에 맞는 AWS IoT 정책을 생성합니다.

```
aws iot create-policy \
--policy-name "JobTestPolicyForuniqueThingName" \
--policy-document "file://~/policies/jobs_test_thing_policy.json"
```

정책이 생성되면 명령에서 다음과 같은 응답을 반환합니다.

```
{
    "policyName": "JobTestPolicyForuniqueThingName",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
JobTestPolicyForuniqueThingName",
    "policyDocument": "{\n\"Version\": \"2012-10-17\",\n\"Statement\": [\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n
```

```
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
    "policyVersionId": "1"
}
```

5. *uniqueThingName*을 디바이스의 사물 이름으로 바꾸고 *certificateArn*을 이 디바이스에 대해 이 섹션 앞부분에서 저장한 certificateArn 값으로 바꾼 다음 이 명령을 실행하여 정책을 디바이스 인증서에 연결합니다.

   ```
   aws iot attach-policy \
   --policy-name "JobTestPolicyFor<uniqueThingName>" \
   --target "<certificateArn>"
   ```

   성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

6. *uniqueThingName*을 디바이스의 사물 이름으로 바꾸고이 섹션*certificateArn*의 앞부분에서 저장한 certificateArn 값으로 바꾼 다음이 명령을 실행하여 디바이스 인증서를 AWS IoT 사물 리소스에 연결합니다.

   ```
   aws iot attach-thing-principal \
   --thing-name "<uniqueThingName>" \
   --principal "<certificateArn>"
   ```

   성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

Raspberry Pi를 성공적으로 프로비저닝한 후에는 테스트에서 다른 Raspberry Pi에 대해 이 섹션을 반복할 준비가 된 것입니다. 모든 디바이스가 프로비저닝된 경우 the section called “작업 에이전트를 실행하도록 Device Client 구성”으로 진행합니다.

작업 에이전트를 실행하도록 AWS IoT Device Client 구성

이 절차에서는 AWS IoT Device Client가 작업 에이전트를 실행할 수 있는 구성 파일을 생성합니다.

참고: 여러 디바이스를 준비하는 경우 각 디바이스에서 이 절차를 수행해야 합니다.

AWS IoT Device Client를 테스트하는 구성 파일을 생성하려면:

1. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서

a. nano 등의 텍스트 편집기를 엽니다.

b. 이 JSON 문서를 복사하여 열려 있는 텍스트 편집기에 붙여 넣습니다.

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/jobs/device.pem.crt",
  "key": "~/certs/jobs/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "uniqueThingName",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": true,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": false,
      "publish-topic": "",
      "publish-file": "",
      "subscribe-topic": "",
      "subscribe-file": ""
    }
  },
```

```
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. *#####* 값을에서 찾 AWS 계정 은의 디바이스 데이터 엔드포인트 값으로 바꿉니다the section called “에서 디바이스 프로비저닝 AWS IoT Core”.

d. *uniqueThingName*을 이 디바이스에 사용한 사물 이름으로 바꿉니다.

e. 텍스트 편집기에서 파일을 **~/dc-configs/dc-jobs-config.json**으로 저장합니다.

2. 이 명령을 실행하여 새로운 구성 파일의 파일 권한을 설정합니다.

```
chmod 644 ~/dc-configs/dc-jobs-config.json
```

이 테스트에는 MQTT 테스트 클라이언트(MQTT test client)를 사용하지 않습니다. 디바이스가 작업 관련 MQTT 메시지를와 교환하는 동안 AWS IoT작업 진행률 메시지는 작업을 실행하는 디바이스와만 교환됩니다. 작업 진행률 메시지는 작업을 실행하는 디바이스와만 교환되므로 AWS IoT 콘솔과 같은 다른 디바이스에서 메시지를 구독할 수 없습니다.

구성 파일을 저장한 후에는 the section called “IoT 작업 생성 및 실행” 단계를 진행할 준비가 된 것입니다.

AWS IoT Device Client를 AWS IoT 사용하여에서 작업 생성 및 실행

이 섹션의 절차에서는 작업 문서와 AWS IoT 작업 리소스를 생성합니다. 작업 리소스를 생성한 후는 작업 에이전트가 작업 문서를 디바이스 또는 클라이언트에 적용하는 지정된 작업 대상으로 작업 문서를 AWS IoT 보냅니다.

이 섹션의 절차

- IoT 작업의 작업 문서 생성 및 저장
- 하나의 IoT 디바이스에 AWS IoT 대해에서 작업 실행

IoT 작업의 작업 문서 생성 및 저장

이 절차에서는 작업 리소스에 포함할 간단한 AWS IoT 작업 문서를 생성합니다. 이 작업 문서는 작업 대상에서 “Hello world!”를 표시합니다.

작업 문서를 생성하고 저장하려면

1. 작업 문서를 저장할 Amazon S3 버킷을 선택합니다. 이를 위해 사용할 기존 Amazon S3 버킷이 없는 경우에는 버킷을 생성해야 합니다. Amazon S3 버킷을 생성하는 방법에 대한 자세한 내용은 Amazon S3 시작하기의 주제를 참조하세요.
2. 이 작업에 대한 작업 문서 생성 및 저장

   a. 로컬 호스트 컴퓨터에서 텍스트 편집기를 엽니다.

   b. 이 텍스트를 복사하여 편집기에 붙여 넣습니다.

   ```
   {
       "operation": "echo",
       "args": ["Hello world!"]
   }
   ```

   c. 로컬 호스트 컴퓨터에서 편집기의 내용을 **hello-world-job.json**이라는 파일에 저장합니다.

   d. 파일이 올바르게 저장되었는지 확인합니다. 일부 텍스트 편집기는 텍스트 파일을 저장할 때 파일 이름에 자동으로 `.txt`를 추가합니다. 편집기에서 파일 이름에 `.txt`를 추가한 경우 계속하기 전에 파일 이름을 수정합니다.

3. *path_to_file*을 **hello-world-job.json**의 경로로 바꾸고(현재 디렉터리에 없는 경우) *s3_bucket_name*을 선택한 버킷에 대한 Amazon S3 버킷 경로로 바꾼 다음 이 명령을 실행하여 작업 문서를 Amazon S3 버킷에 넣습니다.

   ```
   aws s3api put-object \
   --key hello-world-job.json \
   --body path_to_file/hello-world-job.json --bucket s3_bucket_name
   ```

   Amazon S3에 저장한 작업 문서를 식별하는 작업 문서 URL은 다음 URL에서 *s3_bucket_name* 및 *AWS_region*을 바꿔서 결정됩니다. 나중에 *job_document_path*로 사용할 결과 URL을 기록합니다.

   ```
   https://s3_bucket_name.s3.AWS_Region.amazonaws.com/hello-world-job.json
   ```

> **Note**
>
> AWS 보안은 AWS 계정예를 들어 브라우저를 사용하여 외부에서이 URL을 열 수 없도록 합니다. URL은 기본적으로 파일에 액세스할 수 있는 AWS IoT 작업 엔진에서 사용됩니다. 프로덕션 환경에서는 AWS IoT 서비스에 Amazon S3에 저장된 작업 문서에 대한 액세스 권한이 있는지 확인해야 합니다.

작업 문서의 URL을 저장한 후 the section called “단일 디바이스에 대한 작업 실행”으로 진행합니다.

하나의 IoT 디바이스에 AWS IoT 대해에서 작업 실행

이 섹션의 절차는 Raspberry Pi에서 AWS IoT Device Client를 시작하여 디바이스에서 작업 에이전트를 실행하고 작업이 실행될 때까지 기다립니다. 또한에서 작업 리소스를 생성 AWS IoT하여 IoT 디바이스로 작업을 전송하고 실행합니다.

> **Note**
>
> 이 절차는 단일 디바이스에서만 작업을 실행합니다.

Raspberry Pi에서 작업 에이전트를 시작하려면

1. Raspberry Pi에 연결된 로컬 호스트 컴퓨터의 터미널 창에서이 명령을 실행하여 AWS IoT Device Client를 시작합니다.

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-jobs-config.json
```

2. 터미널 창에서 AWS IoT Device Client 및가 이러한 메시지를 표시하는지 확인합니다.

```
2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Jobs is enabled
                        .
                        .
                        .
2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Client base has been notified that
 Jobs has started
2021-11-15T18:45:56.708Z [INFO]  {JobsFeature.cpp}: Running Jobs!
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 startNextPendingJobExecution accepted and rejected
```

```
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 nextJobChanged events
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusAccepted for jobId +
2021-11-15T18:45:56.738Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionAccepted with code {0}
2021-11-15T18:45:56.739Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusRejected for jobId +
2021-11-15T18:45:56.753Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToNextJobChanged with code {0}
2021-11-15T18:45:56.760Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobRejected with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobAccepted with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionRejected with code {0}
2021-11-15T18:45:56.777Z [DEBUG] {JobsFeature.cpp}: Publishing
 startNextPendingJobExecutionRequest
2021-11-15T18:45:56.785Z [DEBUG] {JobsFeature.cpp}: Ack received for
 StartNextPendingJobPub with code {0}
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. 터미널 창에서 이 메시지가 표시되면 다음 절차로 진행하고 작업 리소스를 생성합니다. 목록의 마지막 항목이 아닐 수도 있습니다.

```
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

AWS IoT 작업 리소스를 생성하려면

1. 로컬 호스트 컴퓨터에서

   a. *job_document_url*을 the section called “작업 문서 생성 및 저장”의 작업 문서 URL로 바꿉니다.

   b. *thing_arn*을 디바이스에 대해 생성한 사물 리소스의 ARN으로 바꾼 다음 이 명령을 실행합니다.

```
aws iot create-job \
--job-id hello-world-job-1 \
--document-source "job_document_url" \
```

```
--targets "thing_arn" \
--target-selection SNAPSHOT
```

성공하면 명령이 다음과 같은 결과를 반환합니다.

```
{
  "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-job-1",
  "jobId": "hello-world-job-1"
}
```

2. 터미널 창에 다음과 같이 AWS IoT Device Client의 출력이 표시됩니다.

```
2021-11-15T18:02:26.688Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Job ids differ
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: Executing job: hello-world-
job-1
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Assuming executable is in PATH
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: About to execute: echo Hello
 world!
2021-11-15T18:10:24.890Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken 3TEWba9Xj6 in the updateJobExecution promises map
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process now running
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process about to call
 execvp
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Parent process now running, child
 PID is 16737
2021-11-15T18:10:24.891Z [DEBUG] {16737}: Hello world!
2021-11-15T18:10:24.891Z [DEBUG] {JobEngine.cpp}: JobEngine finished waiting for
 child process, returning 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job exited with status: 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job executed successfully!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
```

```
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.892Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.892Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken GmQ0HTzWGg in the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken 3TEWba9Xj6
 from the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:24.917Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken GmQ0HTzWGg
 from the updateJobExecution promises map
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:25.861Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. AWS IoT Device Client가 실행 중이고 작업을 기다리는 동안 `job-id` 값을 변경하고 1단계create-job에서를 다시 실행하여 다른 작업을 제출할 수 있습니다.

작업 실행이 완료되면 터미널 창에서 ^C (control-C)를 입력하여 AWS IoT Device Client를 중지합니다.

자습서: AWS IoT Device Client 자습서를 실행한 후 정리

이 튜토리얼의 절차는 이 학습 경로의 튜토리얼을 완료하는 동안 생성한 파일과 리소스를 제거하는 과정을 안내합니다.

이 튜토리얼의 절차

- 1단계: Device Client로 데모를 빌드한 후 AWS IoT 디바이스 정리
- 2단계: AWS IoT Device Client로 데모를 빌드한 AWS 계정 후 정리

1단계: Device Client로 데모를 빌드한 후 AWS IoT 디바이스 정리

이 튜토리얼에서는 이 학습 경로에서 데모를 빌드한 후 microSD 카드를 정리하는 방법에 대한 두 가지 옵션을 설명합니다. 필요한 보안 수준을 제공하는 옵션을 선택합니다.

디바이스의 microSD 카드를 정리해도 생성한 AWS IoT 리소스는 제거되지 않습니다. 디바이스의 microSD 카드를 정리한 후 AWS IoT 리소스를 정리하려면의 자습서를 검토해야 합니다the section called “ AWS IoT Device Client로 데모를 빌드한 후 정리”.

옵션 1: microSD 카드를 다시 써서 정리

이 학습 경로의 튜토리얼을 완료한 후 microSD 카드를 지우는 가장 쉽고 철저한 방법은 디바이스를 처음 준비할 때 생성한 저장된 이미지 파일로 microSD 카드를 덮어쓰는 것입니다.

이 절차에서는 로컬 호스트 컴퓨터를 사용하여 저장된 microSD 카드 이미지를 microSD 카드에 씁니다.

> **Note**
>
> 디바이스에서 운영 체제에 이동식 저장 매체를 사용하지 않는 경우 해당 디바이스에 대한 절차를 참조하세요.

microSD 카드에 새 이미지를 쓰려면

1. 로컬 호스트 컴퓨터에서 microSD 카드에 쓸 저장된 microSD 카드 이미지를 찾습니다.
2. microSD 카드를 로컬 호스트 컴퓨터에 삽입합니다.
3. SD 카드 이미징 도구를 사용하여 선택한 이미지 파일을 microSD 카드에 씁니다.
4. Raspberry Pi OS 이미지를 microSD 카드에 쓴 후 microSD 카드를 꺼내 로컬 호스트 컴퓨터에서 안전하게 제거합니다.

microSD 카드를 사용할 준비가 되었습니다.

옵션 2: 사용자 디렉터리를 삭제하여 정리

튜토리얼을 완료한 후 microSD 카드 이미지를 다시 쓰지 않고 microSD 카드를 지우려면 사용자 디렉터리를 개별적으로 삭제합니다. 이 방법은 설치되었을 수 있는 시스템 파일을 제거하지 않기 때문에 저장된 이미지에서 microSD 카드를 다시 쓰는 것만큼 철저하지 않습니다.

사용자 디렉터리 제거만으로도 충분하다면 이 절차를 따를 수 있습니다.

디바이스에서 이 학습 경로의 사용자 디렉터리를 삭제하려면

1. 이 명령을 실행하여 디바이스에 연결된 터미널 창에서 이 학습 경로에 생성된 사용자 디렉터리, 하위 디렉터리 및 모든 파일을 삭제합니다.

> **Note**
>
> 이러한 디렉터리와 파일을 삭제하면 튜토리얼을 다시 완료하지 않고 데모를 실행할 수 없게 됩니다.

```
rm -Rf ~/dc-configs
rm -Rf ~/policies
rm -Rf ~/messages
rm -Rf ~/certs
rm -Rf ~/.aws-iot-device-client
```

2. 이 명령을 실행하여 디바이스에 연결된 터미널 창에서 애플리케이션 소스 디렉터리 및 파일을 삭제합니다.

> **Note**
>
> 이 명령은 프로그램을 제거하지 않습니다. 프로그램 빌드 및 설치에 사용된 소스 파일만 제거합니다. 이러한 파일을 삭제한 후에는 AWS CLI 및 AWS IoT Device Client가 작동하지 않을 수 있습니다.

```
rm -Rf ~/aws-cli
rm -Rf ~/aws
rm -Rf ~/aws-iot-device-client
```

2단계: AWS IoT Device Client로 데모를 빌드한 AWS 계정 후 정리

이 절차는이 학습 경로의 자습서를 완료하는 동안 생성한 AWS 리소스를 식별하고 제거하는 데 도움이 됩니다.

AWS IoT 리소스 정리

이 절차는이 학습 경로의 자습서를 완료하는 동안 생성한 AWS IoT 리소스를 식별하고 제거하는 데 도움이 됩니다.

AWS IoT 이 학습 경로에 생성된 리소스

자습서	사물 리소스	정책 리소스
the section called “IoT Device Client 설치 및 구성”	DevCliTestThing	DevCliTestThingPolicy
the section called “MQTT를 사용하여 디바이스 클라이언트와 통신”	PubSubTestThing	PubSubTestThingPolicy
the section called “Device Client를 사용하여 IoT 작업 실행”	사용자 정의(여러 개 있을 수 있음)	사용자 정의(여러 개 있을 수 있음)

AWS IoT 리소스를 삭제하려면 생성한 각 사물 리소스에 대해이 절차를 따릅니다.

1. *thing_name*을 삭제하려는 사물 리소스의 이름으로 바꾼 다음 이 명령을 실행하여 로컬 호스트 컴퓨터에서 사물 리소스에 연결된 인증서를 나열합니다.

   ```
   aws iot list-thing-principals --thing-name thing_name
   ```

   이 명령은 연결된 인증서를 나열하는 이와 같은 응답을 반환합니다.*thing_name* 대부분의 경우 목록에 인증서가 하나만 있습니다.

   ```
   {
       "principals": [
           "arn:aws:iot:us-
   west-2:57EXAMPLE833:cert/23853eea3cf0edc7f8a69c74abeafa27b2b52823cab5b3e156295e94b26ae8ac"
       ]
   }
   ```

2. 이전 명령으로 나열된 각 인증서에 대해

   a. *certificate_ID*를 이전 명령의 인증서 ID로 바꿉니다. 인증서 ID는 이전 명령에서 반환된 ARN에서 `cert/` 뒤에 오는 영숫자 문자입니다. 그런 다음 이 명령을 실행하여 인증서를 비활성화합니다.

```
aws iot update-certificate --new-status INACTIVE --certificate-
id certificate_ID
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

b. *certificate_ARN*을 이전에 반환된 인증서 목록의 인증서 ARN으로 바꾼 다음 이 명령을 실행하여 이 인증서에 연결된 정책을 나열합니다.

```
aws iot list-attached-policies --target certificate_ARN
```

이 명령은 인증서에 연결된 정책을 나열하는 이와 같은 응답을 반환합니다. 대부분의 경우 목록에 정책이 하나만 있습니다.

```
{
    "policies": [
        {
            "policyName": "DevCliTestThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
DevCliTestThingPolicy"
        }
    ]
}
```

c. 인증서에 연결된 각 정책에 대해:

i. *policy_name*을 이전 명령의 policyName 값으로 바꾸고 *certificate_ARN*을 인증서 ARN으로 바꾼 다음 이 명령을 실행하여 인증서에서 정책을 분리합니다.

```
aws iot detach-policy --policy-name policy_name --target certificate_ARN
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

ii. *policy_name*을 policyName 값으로 바꾼 다음 이 명령을 실행하여 정책이 추가 인증서에 연결되어 있는지 확인합니다.

```
aws iot list-targets-for-policy --policy-name policy_name
```

명령이 이와 같은 빈 목록을 반환하는 경우 정책은 인증서에 연결되지 않은 것입니다. 계속해서 정책 버전을 나열합니다. 정책에 연결된 인증서가 여전히 있는 경우 detach-thing-principal 단계를 계속합니다.

```
{
    "targets": []
}
```

iii. *policy_name*을 policyName 값으로 바꾼 다음 이 명령을 실행하여 정책 버전을 확인합니다. 정책을 삭제하려면 버전이 하나만 있어야 합니다.

```
aws iot list-policy-versions --policy-name policy_name
```

이 예와 같이 정책에 버전이 하나만 있는 경우 지금 delete-policy 단계로 건너뛰고 정책을 삭제할 수 있습니다.

```
{
    "policyVersions": [
        {
            "versionId": "1",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:02:46.778000+00:00"
        }
    ]
}
```

이 예와 같이 정책에 여러 버전이 있는 경우 isDefaultVersion 값이 false인 정책 버전을 삭제해야 정책을 삭제할 수 있습니다.

```
{
    "policyVersions": [
        {
            "versionId": "2",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:52:04.423000+00:00"
        },
        {
            "versionId": "1",
            "isDefaultVersion": false,
            "createDate": "2021-11-18T01:30:18.083000+00:00"
```

```
        }
    ]
}
```

정책 버전을 삭제해야 하는 경우 *policy_name*을 policyName 값으로 바꾸고 *version_ID*를 이전 명령의 versionId 값으로 바꾼 다음 이 명령을 실행하여 정책 버전을 삭제합니다.

```
aws iot delete-policy-version --policy-name policy_name --policy-version-
id version_ID
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

정책 버전을 삭제한 후 정책에 정책 버전이 하나만 있을 때까지 이 단계를 반복합니다.

iv. *policy_name*을 policyName 값으로 바꾼 다음 이 명령을 실행하여 정책을 삭제합니다.

```
aws iot delete-policy --policy-name policy_name
```

d. *thing_name*을 사물의 이름으로 바꾸고 *certificate_ARN*을 인증서의 ARN으로 바꾼 다음, 이 명령을 실행하여 사물 리소스에서 인증서를 분리합니다.

```
aws iot detach-thing-principal --thing-name thing_name --
principal certificate_ARN
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

e. *certificate_ID*를 이전 명령의 인증서 ID로 바꿉니다. 인증서 ID는 이전 명령에서 반환된 ARN에서 cert/ 뒤에 오는 영숫자 문자입니다. 그런 다음 이 명령을 실행하여 인증서 리소스를 삭제합니다.

```
aws iot delete-certificate --certificate-id certificate_ID
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

3. *thing_name*을 사물의 이름으로 바꾼 다음 이 명령을 실행하여 사물을 삭제합니다.

```
aws iot delete-thing --thing-name thing_name
```

성공한 경우 이 명령은 아무 것도 반환하지 않습니다.

AWS 리소스 정리

이 절차는이 학습 경로의 자습서를 완료하는 동안 생성한 다른 AWS 리소스를 식별하고 제거하는 데 도움이 됩니다.

이 학습 경로에 생성된 기타 AWS 리소스

자습서	리소스 유형	리소스 이름 또는 ID
the section called “Device Client를 사용하여 IoT 작업 실행”	Amazon S3 객체	hello-world-job.json
the section called “Device Client를 사용하여 IoT 작업 실행”	AWS IoT 작업 리소스	사용자 정의

이 학습 경로에서 생성된 AWS 리소스를 삭제하려면

1. 이 학습 경로에서 생성된 작업을 삭제하려면

   a. 이 명령을 실행하여의 작업을 나열합니다 AWS 계정.

   ```
   aws iot list-jobs
   ```

   명령은 다음과 AWS 리전 같은 및의 AWS IoT 작업 목록을 반환합니다 AWS 계정 .

   ```
   {
       "jobs": [
           {
               "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
   job-2",
               "jobId": "hello-world-job-2",
               "targetSelection": "SNAPSHOT",
               "status": "COMPLETED",
               "createdAt": "2021-11-16T23:40:36.825000+00:00",
               "lastUpdatedAt": "2021-11-16T23:40:41.375000+00:00",
   ```

```
            "completedAt": "2021-11-16T23:40:41.375000+00:00"
        },
        {
            "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
job-1",
            "jobId": "hello-world-job-1",
            "targetSelection": "SNAPSHOT",
            "status": "COMPLETED",
            "createdAt": "2021-11-16T23:35:26.381000+00:00",
            "lastUpdatedAt": "2021-11-16T23:35:29.239000+00:00",
            "completedAt": "2021-11-16T23:35:29.239000+00:00"
        }
    ]
}
```

b. 목록에서이 학습 경로에서 생성한 작업으로 인식하는 각 작업에 대해를 삭제할 작업의 `jobId` 값으로 *`jobId`* 바꾼 다음이 명령을 실행하여 AWS IoT 작업을 삭제합니다.

```
aws iot delete-job --job-id jobId
```

명령이 성공하면 아무 것도 반환하지 않습니다.

2. 이 학습 경로의 Amazon S3 버킷에 저장한 작업 문서를 삭제하려면

a. *`bucket`*을 사용한 버킷의 이름으로 바꾼 다음 이 명령을 실행하여 사용한 Amazon S3 버킷의 객체를 나열합니다.

```
aws s3api list-objects --bucket bucket
```

이 명령은 다음과 같은 버킷의 Amazon S3 객체 목록을 반환합니다.

```
{
    "Contents": [
        {
            "Key": "hello-world-job.json",
            "LastModified": "2021-11-18T03:02:12+00:00",
            "ETag": "\"868c8bc3f56b5787964764d4b18ed5ef\"",
            "Size": 54,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
```

```
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
            "Key": "iot_job_firmware_update.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"7c68c591949391791ecf625253658c61\"",
            "Size": 66,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
            "Key": "order66.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"bca60d5380b88e1a70cc27d321caba72\"",
            "Size": 29,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        }
    ]
}
```

b. 목록에서 이 학습 경로에서 생성한 객체로 인식한 각 객체에 대해 *bucket*을 버킷 이름으로 바꾸고, *key*를 삭제할 객체의 키 값으로 바꾼 다음 이 명령을 실행하여 Amazon S3 객체를 삭제합니다.

```
aws s3api delete-object --bucket bucket --key key
```

명령이 성공하면 아무 것도 반환하지 않습니다.

이 학습 경로를 완료하는 동안 생성한 모든 AWS 리소스와 객체를 삭제한 후 자습서를 다시 시작하고 반복할 수 있습니다.

AWS IoT 디바이스 SDKs를 사용하여 솔루션 구축

이 섹션의 자습서에서는를 사용하여 프로덕션 환경에 배포할 수 있는 IoT 솔루션을 개발하는 단계를 안내합니다 AWS IoT.

이 자습서는 AWS IoT 디바이스 SDKs를 사용하고 안전하고 신뢰할 수 있는 솔루션을 만드는 데 도움이 되도록 적용되는 개념을 자세히 설명the section called “AWS IoT Device Client를 사용하여 데모 빌드”하기 때문에 섹션의 자습서보다 완료하는 데 더 많은 시간이 걸릴 수 있습니다.

AWS IoT 디바이스 SDKs로 솔루션 구축 시작

이 자습서에서는 다양한 AWS IoT 시나리오를 안내합니다. 해당하는 경우 자습서에서는 AWS IoT 디바이스 SDKs를 사용합니다.

주제

- 자습서: Device SDK를 AWS IoT Core 사용하여에 AWS IoT 디바이스 연결
- 디바이스 데이터를 다른 서비스로 라우팅하는 AWS IoT 규칙 생성
- 디바이스 섀도우로 디바이스가 오프라인 상태일 때 디바이스 상태 유지
- 자습서:에 대한 사용자 지정 권한 부여자 생성 AWS IoT Core
- 자습서: AWS IoT 및 Raspberry Pi를 사용하여 토양 수분 모니터링

자습서: Device SDK를 AWS IoT Core 사용하여에 AWS IoT 디바이스 연결

이 자습서에서는 데이터를 주고 받을 수 AWS IoT Core 있도록 디바이스를에 연결하는 방법을 보여줍니다 AWS IoT. 이 자습서를 완료하면 디바이스가에 연결되도록 구성 AWS IoT Core 되고 디바이스가 통신하는 방식을 이해할 수 있습니다 AWS IoT.

주제

- 필수 조건
- 용 디바이스 준비 AWS IoT
- MQTT 프로토콜 검토
- pubsub.py Device SDK 샘플 앱 검토
- 디바이스 연결 및와 통신 AWS IoT Core
- 결과 검토
- 자습서: AWS IoT Device SDK for Embedded C 사용

필수 조건

이 자습서를 시작하기 전에 다음 사항을 준비해야 합니다.

- AWS IoT Core 자습서 시작하기 완료

  the section called “디바이스 구성”해야 하는 해당 자습서의 섹션에서 디바이스에 대한 the section called “Raspberry Pi 또는 다른 디바이스 연결” 옵션을 선택하고 Python 언어 옵션을 사용하여 디바이스를 구성합니다.

  > **Note**
  >
  > 이 자습서에서도 사용하므로 해당 자습서에서 사용하는 터미널 창을 계속 열어 둡니다.

- AWS IoT Device SDK v2 for Python을 실행할 수 있는 디바이스입니다.

  이 자습서에서는 비교적 강력한 디바이스가 필요한 Python 코드 예제를 사용하여 디바이스를 AWS IoT Core 에 연결하는 방법을 보여줍니다. 리소스가 제한된 디바이스로 작업하는 경우 이러한 코드 예제는 해당 디바이스에서 작동하지 않을 수 있습니다. 이 경우 the section called “AWS IoT Device SDK for Embedded C 사용” 자습서를 통해 더욱 성공적으로 수행할 수 있습니다.

- 디바이스에 연결하는 데 필요한 정보를 얻었습니다.

  디바이스를 연결하려면 사물 이름 AWS IoT, 호스트 이름 및 포트 번호에 대한 정보가 있어야 합니다.

  > **Note**
  >
  > 사용자 지정 인증을 사용하여 디바이스를 AWS IoT Core에 연결할 수도 있습니다. 권한 부여자 Lambda 함수에 전달하는 연결 데이터는 사용하는 프로토콜에 따라 다릅니다.

  - 사물 이름: 연결하려는 AWS IoT 사물의 이름입니다. 를 디바이스로 AWS IoT 사물로 등록해야 합니다. 자세한 내용은 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하십시오.
  - 호스트 이름: 계정별 IoT 엔드포인트의 호스트 이름입니다.
  - 포트 번호: 연결하려는 포트 번호입니다.

  AWS IoT Python SDK의 `configureEndpoint` 메서드를 사용하여 호스트 이름과 포트 번호를 구성할 수 있습니다.

```
myAWSIoTMQTTClient.configureEndpoint("random.iot.region.amazonaws.com", 8883)
```

용 디바이스 준비 AWS IoT

AWS IoT Core 자습서 시작하기에서는 디바이스와 AWS 계정이 통신할 수 있도록 준비했습니다. 이 섹션에서는 모든 디바이스 연결에 적용되는 준비의 측면을 검토합니다 AWS IoT Core.

디바이스가 AWS IoT Core에 연결하려면 다음 사항이 필요합니다.

1. AWS 계정이 있어야 합니다.

   의 절차에서는 아직 생성하지 않은 AWS 계정 경우를 생성하는 방법을 설정 AWS 계정 설명합니다.
2. 해당 계정에서는 AWS 계정 및 리전의 디바이스에 대해 다음과 같은 AWS IoT 리소스가 정의되어 있어야 합니다.

   AWS IoT 리소스 생성의 절차에서는 AWS 계정 및 리전의 디바이스에 대해 이러한 리소스를 생성하는 방법을 설명합니다.

   - AWS IoT 에 등록되고 디바이스 인증을 위해 활성화된 디바이스 인증서.

     인증서는 종종 AWS IoT 사물 객체와 함께 만들어지고 연결됩니다. 디바이스를 연결하는 데 사물 객체는 필요하지 않지만 디바이스 AWS IoT에서 추가 AWS IoT 기능을 사용할 수 있습니다.
   - 디바이스 인증서에 연결된 정책으로,에 연결하고 원하는 모든 작업을 AWS IoT Core 수행할 수 있는 권한을 부여합니다.
3. AWS 계정의 디바이스 엔드포인트에 액세스할 수 있는 인터넷 연결.

   디바이스 엔드포인트는에 설명되어 AWS IoT 디바이스 데이터 및 서비스 엔드포인트 있으며 AWS IoT 콘솔의 설정 페이지에서 확인할 수 있습니다.
4. AWS IoT Device SDK와 같은 통신 소프트웨어. 이 자습서에서는 AWS IoT 디바이스 SDK v2 for Python을 사용합니다.

MQTT 프로토콜 검토

샘플 앱에 대해 이야기하기 전에 MQTT 프로토콜을 이해하는 것이 도움이 됩니다. MQTT 프로토콜은 HTTP 등의 다른 네트워크 통신 프로토콜에 비해 몇 가지 이점을 제공하므로 IoT 디바이스에서 널리 사용됩니다. 이 섹션에서는 이 자습서에 적용되는 MQTT의 주요 측면을 검토합니다. MQTTT를 HTTP와

비교하는 방법에 대한 자세한 내용은 디바이스 통신을 위한 애플리케이션 프로토콜 선택 단원을 참조하세요.

MQTT는 게시/구독 통신 모델을 사용합니다.

MQTT 프로토콜은 호스트와 게시/구독 통신 모델을 사용합니다. 이 모델은 HTTP가 사용하는 요청/응답 모델과 다릅니다. MQTT를 사용하면 디바이스가 고유한 클라이언트 ID로 식별되는 호스트와의 세션을 설정합니다. 데이터를 전송하기 위해 디바이스는 주제별로 식별되는 메시지를 호스트의 메시지 브로커에 게시합니다. 메시지 브로커로부터 메시지를 수신하기 위해 디바이스는 구독 요청의 주제 필터를 메시지 브로커에 전송하여 주제를 구독합니다.

영구 세션을 지원하는 MQTT

메시지 브로커는 디바이스에서 메시지를 수신하고 해당 디바이스를 구독한 디바이스에 메시지를 게시합니다. 영구 세션(초기 디바이스의 연결이 끊어진 경우에도 활성 상태로 유지되는 세션)을 사용하면 디바이스는 연결이 끊긴 동안 게시된 메시지를 검색할 수 있습니다. 디바이스 측에서 MQTT는 서비스 품질 수준(QoS)를 사용하여 호스트가 디바이스에서 전송한 메시지를 수신하도록 합니다.

pubsub.py Device SDK 샘플 앱 검토

이 섹션에서는 이 자습서에 사용된 AWS IoT Device SDK v2 for Python의 `pubsub.py` 샘플 앱을 검토합니다. 여기서는 MQTT 메시지를 게시하고 구독 AWS IoT Core 하기 위해에 연결하는 방법을 살펴보겠습니다. 다음 섹션에서는 디바이스가 어떻게 연결되고 통신하는지 탐색하는 데 도움이 되는 몇 가지 연습을 제공합니다 AWS IoT Core.

pubsub.py 샘플 앱은 AWS IoT Core MQTT 연결의 다음 측면을 보여줍니다.

- 통신 프로토콜
- 영구 세션
- 서비스 품질
- 메시지 게시
- 메시지 구독
- 디바이스 연결 해제 및 재연결

통신 프로토콜

`pubsub.py` 샘플에서는 MQTT 및 MQTT over WSS 프로토콜을 사용하는 MQTT 연결을 보여 줍니다. AWS 공통 런타임(AWS CRT) 라이브러리는 하위 수준의 통신 프로토콜을 지원하며 Python용 AWS IoT Device SDK v2에 포함되어 있습니다.

MQTT

pubsub.py 샘플은 MQTT 프로토콜을 AWS IoT Core 사용하여 와의 연결을 설정[mqtt_connection_builder](#)하기 위해에서 mtls_from_path (여기에 표시됨)를 호출합니다.는 X.509 인증서 및 TLS v1.2를 mtls_from_path 사용하여 디바이스를 인증합니다. AWS CRT 라이브러리는 해당 연결의 하위 수준 세부 정보를 처리합니다.

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
    endpoint=args.endpoint,
    cert_filepath=args.cert,
    pri_key_filepath=args.key,
    ca_filepath=args.ca_file,
    client_bootstrap=client_bootstrap,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

endpoint

AWS 계정사용자의 IoT 디바이스 엔드포인트

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

cert_filepath

디바이스의 인증서 파일 경로

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

pri_key_filepath

인증서 파일을 사용하여 만든 디바이스의 개인 키 파일에 대한 경로

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

ca_filepath

루트 CA 파일의 경로. MQTT 서버가 trust store에 없는 인증서를 사용하는 경우에만 필요합니다.

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

`client_bootstrap`

소켓 통신 활동을 처리하는 공통 런타임 객체

샘플 앱에서 이 객체는 `mqtt_connection_builder.mtls_from_path`를 호출하기 전에 인스턴스화됩니다.

`on_connection_interrupted`, `on_connection_resumed`

디바이스의 연결이 중단되고 재개될 때 호출할 콜백 함수

`client_id`

AWS 리전에서 이 디바이스를 고유하게 식별하는 ID

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

`clean_session`

새 영구 세션을 시작할지, 아니면 존재하는 경우 기존 세션에 다시 연결할지 여부

`keep_alive_secs`

CONNECT 요청을 전송할 연결 유지 값(초)입니다. 이 간격으로 ping이 자동으로 전송됩니다. 이 값의 1.5배 후에도 서버가 ping을 받지 못하면 연결이 끊어진 것으로 간주합니다.

MQTT over WSS

`pubsub.py` 샘플은 `mqtt_connection_builder`에서 `websockets_with_default_aws_signing`(여기에 표시됨)을 직접 호출하여 WSS를 통한 MQTT 프로토콜을 사용하여 AWS IoT Core 와의 연결을 설정합니다. `websockets_with_default_aws_signing`은 디바이스를 인증하기 위해 Signature V4를 사용하여 WSS를 통한 MQTT 연결을 만듭니다.

```
mqtt_connection = mqtt_connection_builder.websockets_with_default_aws_signing(
    endpoint=args.endpoint,
    client_bootstrap=client_bootstrap,
    region=args.signing_region,
    credentials_provider=credentials_provider,
    websocket_proxy_options=proxy_options,
    ca_filepath=args.ca_file,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
```

```
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

AWS 계정사용자의 IoT 디바이스 엔드포인트

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

`client_bootstrap`

소켓 통신 활동을 처리하는 공통 런타임 객체

샘플 앱에서 이 객체는 `mqtt_connection_builder.websockets_with_default_aws_signing`를 호출하기 전에 인스턴스화됩니다.

`region`

AWS Signature V4 인증에 사용되는 서명 리전입니다. `pubsub.py`에서 명령줄에 입력된 파라미터를 전달합니다.

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

`credentials_provider`

인증에 사용하기 위해 제공된 AWS 자격 증명

샘플 앱에서 이 객체는 `mqtt_connection_builder.websockets_with_default_aws_signing`을 호출하기 전에 인스턴스화됩니다.

`websocket_proxy_options`

HTTP 프록시 옵션(프록시 호스트를 사용하는 경우)

샘플 앱에서 이 값은 `mqtt_connection_builder.websockets_with_default_aws_signing`을 호출하기 전에 초기화됩니다.

`ca_filepath`

루트 CA 파일의 경로. MQTT 서버가 trust store에 없는 인증서를 사용하는 경우에만 필요합니다.

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

`on_connection_interrupted`, `on_connection_resumed`

디바이스의 연결이 중단되고 재개될 때 호출할 콜백 함수

`client_id`

AWS 리전에서 이 디바이스를 고유하게 식별하는 ID.

샘플 앱에서 이 값은 명령줄에서 전달됩니다.

`clean_session`

새 영구 세션을 시작할지, 아니면 존재하는 경우 기존 세션에 다시 연결할지 여부

`keep_alive_secs`

CONNECT 요청을 전송할 연결 유지 값(초)입니다. 이 간격으로 ping이 자동으로 전송됩니다. 이 값의 1.5배 후에도 서버가 ping을 받지 못하면 연결이 끊어진 것으로 간주합니다.

HTTPS

HTTPS는 어떤가요?는 HTTPS 요청을 게시하는 디바이스를 AWS IoT Core 지원합니다. 프로그래밍 관점에서 디바이스는 다른 애플리케이션과 마찬가지로 AWS IoT Core 에 HTTPS 요청을 전송합니다. 디바이스에서 HTTP 메시지를 전송하는 Python 프로그램의 예는 Python의 `requests` 라이브러리를 사용하는 HTTPS 코드 예를 참조하세요. 이 예제에서는가 메시지를 MQTT 메시지로 해석하도록 AWS IoT Core HTTPS를 AWS IoT Core 사용하여에 메시지를 보냅니다.

는 디바이스의 HTTPS 요청을 AWS IoT Core 지원하지만 디바이스 통신에 사용할 프로토콜을 정보에 입각하여 결정할 수 디바이스 통신을 위한 애플리케이션 프로토콜 선택 있도록에 대한 정보를 검토해야 합니다.

영구 세션

샘플 앱에서 `clean_session` 파라미터를 `False`로 설정하면 연결이 지속되어야 함을 나타냅니다. 실제로 이것은 이 호출에 의해 열린 연결이 존재하는 경우 기존 영구 세션에 다시 연결된다는 것을 의미합니다. 그렇지 않으면 새 영구 세션을 만들고 연결합니다.

영구 세션에서는 디바이스가 연결되어 있지 않은 동안 디바이스로 전송되는 메시지가 메시지 브로커에 의해 저장됩니다. 디바이스가 영구 세션에 다시 연결되면 메시지 브로커는 디바이스가 구독한 저장된 메시지를 디바이스에 전송합니다.

영구 세션이 없으면 디바이스가 연결되어 있지 않은 동안 전송된 메시지를 수신하지 않습니다. 사용할 옵션은 애플리케이션에 따라 다르고 디바이스가 연결되지 않은 동안 발생하는 메시지를 전달해야 하는지 여부에 따라 다릅니다. 자세한 내용은 MQTT 지속적 세션 단원을 참조하세요.

서비스 품질

디바이스가 메시지를 게시하고 구독하면 선호하는 서비스 품질(QoS)을 설정할 수 있습니다.는 게시 및 구독 작업에 대해 QoS 수준 0 및 1을 AWS IoT 지원합니다. 의 QoS 수준에 대한 자세한 내용은 섹션을 AWS IoT참조하세요MQTT 서비스 품질(QoS) 옵션.

Python용 AWS CRT 런타임은 지원하는 QoS 수준에 대해 다음 상수를 정의합니다.

Python 서비스 품질 수준

MQTT QoS 수준	SDK에서 사용하는 Python 기호 값	설명
QoS 수준 0	`mqtt.QoS.AT_MOST_ONCE`	수신 여부에 관계없이 메시지 전송을 한 번만 시도합니다. 예를 들어 디바이스가 연결되어 있지 않거나 네트워크 오류가 있는 경우 메시지가 전혀 전송되지 않을 수 있습니다.
QoS 수준 1	`mqtt.QoS.AT_LEAST_ONCE`	메시지는 PUBACK 승인이 수신될 때까지 반복적으로 전송됩니다.

샘플 앱에서 게시 및 구독 요청은 QoS 수준 1(`mqtt.QoS.AT_LEAST_ONCE`)로 수행됩니다.

- 게시 중 QoS

  디바이스가 QoS 수준 1의 메시지를 게시하면 메시지 브로커로부터 PUBACK 응답을 받을 때까지 메시지를 반복해서 전송합니다. 디바이스가 연결되어 있지 않으면 메시지가 다시 연결한 후 전송되도록 대기열에 저장됩니다.

- 구독 중 QoS

디바이스가 QoS 수준 1의 메시지를 구독하면 메시지 브로커는 디바이스로 전송할 수 있을 때까지 디바이스가 구독한 메시지를 저장합니다. 메시지 브로커는 디바이스로부터 PUBACK 응답을 받을 때까지 메시지를 다시 전송합니다.

메시지 게시

연결을 성공적으로 설정하면 AWS IoT Core디바이스가 메시지를 게시할 수 있습니다. pubsub.py 샘플은 mqtt_connection 객체의 publish 작업을 호출하여 이를 수행합니다.

```
mqtt_connection.publish(
    topic=args.topic,
    payload=message,
    qos=mqtt.QoS.AT_LEAST_ONCE
)
```

`topic`

메시지를 식별하는 메시지의 주제 이름

샘플 앱에서 이 이름은 명령줄에서 전달됩니다.

`payload`

문자열로 형식이 지정된 메시지 페이로드(예: JSON 문서)

샘플 앱에서 이것은 명령줄에서 전달됩니다.

JSON 문서는 일반적인 페이로드 형식이며 다른 AWS IoT 서비스에서 인식하는 형식이지만 메시지 페이로드의 데이터 형식은 게시자와 구독자가 동의하는 모든 형식일 수 있습니다. 그러나 다른 AWS IoT 서비스는 경우에 따라 대부분의 작업에 대해 JSON 및 CBOR만 인식합니다.

`qos`

이 메시지의 QoS 수준

메시지 구독

AWS IoT 및 기타 서비스 및 디바이스에서 메시지를 수신하기 위해 디바이스는 주제 이름으로 해당 메시지를 구독합니다. 디바이스는 주제 이름을 지정하여 개별 메시지를 구독하고 와일드카드 문자를 포함할 수 있는 주제 필터를 지정하여 메시지 그룹을 구독할 수 있습니다. pubsub.py 샘플은 여기에 표시된 코드를 사용하여 메시지를 구독하고 콜백 함수를 등록하여 메시지를 받은 후 처리합니다.

```
subscribe_future, packet_id = mqtt_connection.subscribe(
    topic=args.topic,
    qos=mqtt.QoS.AT_LEAST_ONCE,
    callback=on_message_received
)
subscribe_result = subscribe_future.result()
```

`topic`

구독할 주제. 주제 이름 또는 주제 필터일 수 있습니다.

샘플 앱에서 이것은 명령줄에서 전달됩니다.

`qos`

디바이스 연결이 끊어지는 동안 메시지 브로커가 이러한 메시지를 저장할지 여부입니다.

값이 `mqtt.QoS.AT_LEAST_ONCE`(QoS 수준 1)이면 연결이 생성될 때 영구 세션을 지정해야 합니다(`clean_session=False`).

`callback`

구독된 메시지를 처리하기 위해 호출할 함수.

`mqtt_connection.subscribe` 함수는 미래의 패킷과 패킷 ID를 반환합니다. 구독 요청이 성공적으로 시작된 경우 반환된 패킷 ID는 0보다 큽니다. 구독이 메시지 브로커에 의해 수신 및 등록되었는지 확인하려면 코드 예제와 같이 비동기 작업의 결과가 반환될 때까지 기다려야 합니다.

콜백 함수

`pubsub.py` 샘플의 콜백은 디바이스가 수신한 구독 메시지를 처리합니다.

```
def on_message_received(topic, payload, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == args.count:
        received_all_event.set()
```

`topic`

메시지의 주제

주제 필터를 구독한 경우에도 수신된 메시지의 특정 주제 이름입니다.

`payload`

메시지 페이로드

이 형식은 애플리케이션에 따라 다릅니다.

`kwargs`

`mqtt.Connection.subscribe`에 설명된 대로 가능한 추가 인수입니다.

pubsub.py 샘플에서 on_message_received는 주제와 해당 페이로드만 표시합니다. 또한 한도에 도달한 후 프로그램을 종료하기 위해 수신된 메시지 수를 계산합니다.

앱은 주제 및 페이로드를 평가하여 수행할 작업을 결정합니다.

디바이스 연결 해제 및 재연결

pubsub.py 샘플에는 디바이스의 연결이 끊어지고 연결이 다시 설정될 때 호출되는 콜백 함수가 포함되어 있습니다. 디바이스에서 이러한 이벤트에 대해 수행하는 작업은 애플리케이션에 따라 다릅니다.

디바이스가 처음으로 연결되면 수신할 주제를 구독해야 합니다. 디바이스가 다시 연결될 때 디바이스의 세션이 있으면 구독이 복원되고 해당 구독에서 저장된 메시지가 다시 연결되면 디바이스로 전송됩니다.

디바이스가 다시 연결될 때 디바이스의 세션이 더 이상 존재하지 않으면 구독을 다시 구독해야 합니다. 영구 세션은 수명이 제한되어 있으며 디바이스의 연결이 너무 오래 끊어지면 만료될 수 있습니다.

디바이스 연결 및와 통신 AWS IoT Core

이 섹션에서는 디바이스를 AWS IoT Core에 연결하는 다양한 측면을 탐색하는 데 도움이 되는 몇 가지 연습을 제공합니다. 이 연습에서는 AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하여 디바이스가 게시하는 내용을 확인하고 디바이스에 메시지를 게시합니다. 이 연습에서는 AWS IoT Device SDK v2 for Python의 `pubsub.py` 샘플을 사용하고 시작하기 자습서 자습서의 경험을 바탕으로 빌드합니다.

이 단원에서는 다음을 수행합니다.

- 와일드카드 주제 필터 구독
- 주제 필터 구독 처리
- 디바이스에서 메시지 게시

이러한 연습의 경우 먼저 pubsub.py 샘플 프로그램에서 시작합니다.

> **Note**
>
> 이 연습에서는 시작하기 자습서 자습서를 완료하고 해당 자습서에서 디바이스의 터미널 창을 사용한다고 가정합니다.

와일드카드 주제 필터 구독

이 연습에서는 와일드카드 주제 필터를 구독하고 메시지 주제에 따라 수신된 메시지를 처리하기 위해 pubsub.py를 호출하는 데 사용되는 명령줄을 수정합니다.

연습 절차

이 연습에서는 디바이스에 온도 컨트롤과 조명 컨트롤이 있다고 가정합니다. 이러한 주제 이름을 사용하여 해당 주제에 대한 메시지를 식별합니다.

1. 연습을 시작하기 전에 디바이스의 시작하기 자습서 자습서에서 이 명령을 실행하여 연습에 필요한 모든 것이 준비되었는지 확인하세요.

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

   시작하기 자습서에서 본 것과 동일한 출력이 표시되어야 합니다.

2. 이 연습에서는 다음 명령행 파라미터를 변경합니다.

작업	명령줄 파라미터	Effect
추가	--message ""	수신 대기만 하도록 pubsub.py 구성
추가	--count 2	두 개의 메시지를 받은 후 프로그램 종료
변경	--topic device/+/details	구독할 주제 필터 정의

초기 명령줄을 이렇게 변경하면 이 명령줄이 생성됩니다. 디바이스의 터미널 창에 이 명령을 입력합니다.

```
python3 pubsub.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

프로그램은 다음과 같이 표시되어야 합니다.

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-24d7cdcc-cc01-458c-8488-2d05849691e1'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
```

터미널에 이와 같은 항목이 표시되면 디바이스가 준비되고 주제 이름이 `device`(으)로 시작하고 `/detail`(으)로 끝나는 메시지를 수신 대기 중인 것입니다. 그럼 테스트해 보겠습니다.

3. 다음은 디바이스에서 수신할 수 있는 몇 가지 메시지입니다.

주제 이름	메시지 페이로드
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

4. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하여 이전 단계에서 설명한 메시지를 디바이스로 전송합니다.

   a. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.

   b. 주제 구독(Subscribe to a topic)의 Subscription topic field(구독 주제 필드)에 주제 필터 **`device/+/details`**를 입력한 다음 주제 구독(Subscribe to topic)을 선택합니다.

   c. MQTT 테스트 클라이언트의 구독(Subscriptions) 열에서 device/+/details를 선택합니다.

   d. 앞의 표에 있는 각 주제에 대해 MQTT 테스트 클라이언트에서 다음을 수행합니다.

1. 게시(Publish)에서 테이블의 주제 이름(Topic name) 열에 있는 값을 입력합니다.
2. 주제 이름 아래의 메시지 페이로드 필드에 테이블의 메시지 페이로드(Message payload) 열에 있는 값을 입력합니다.
3. pubsub.py가 실행 중인 터미널 창을 살펴보고 MQTT 테스트 클라이언트에서 주제에 게시(Publish to topic)를 선택합니다.

터미널 창에서 pubsub.py가 메시지를 수신했음을 확인해야 합니다.

연습 결과

pubsub.py를 통해 와일드카드 주제 필터를 사용하여 메시지를 구독하고 수신하여 터미널 창에 표시했습니다. 단일 주제 필터를 구독하고 두 개의 개별 주제가 있는 메시지를 처리하기 위해 콜백 함수가 호출된 방법에 주목하세요.

주제 필터 구독 처리

이전 연습을 기반으로 pubsub.py 샘플 앱을 수정하여 메시지 주제를 평가하고 주제를 기반으로 구독된 메시지를 처리합니다.

연습 절차

메시지 주제를 평가하려면

1. pubsub.py를 pubsub2.py에 복사합니다.
2. 즐겨 사용하는 텍스트 편집기나 IDE에서 pubsub2.py를 엽니다.
3. pubsub2.py에서 on_message_received 함수를 찾습니다.
4. on_message_received에서 print("Received message로 시작하는 줄 뒤와 global received_count로 시작하는 줄 앞에 다음 코드를 삽입합니다.

```
    topic_parsed = False
    if "/" in topic:
        parsed_topic = topic.split("/")
        if len(parsed_topic) == 3:
            # this topic has the correct format
            if (parsed_topic[0] == 'device') and (parsed_topic[2] == 'details'):
                # this is a topic we care about, so check the 2nd element
                if (parsed_topic[1] == 'temp'):
```

```
                    print("Received temperature request: {}".format(payload))
                    topic_parsed = True
                if (parsed_topic[1] == 'light'):
                    print("Received light request: {}".format(payload))
                    topic_parsed = True
    if not topic_parsed:
        print("Unrecognized message topic.")
```

5. 변경 사항을 저장하고 이 명령줄을 사용하여 수정된 프로그램을 실행합니다.

```
python3 pubsub2.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

6. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.
7. 주제 구독(Subscribe to a topic)의 Subscription topic field(구독 주제 필드)에 주제 필터 **`device/+/details`**를 입력한 다음 주제 구독(Subscribe to topic)을 선택합니다.
8. MQTT 테스트 클라이언트의 구독(Subscriptions) 열에서 device/+/details를 선택합니다.
9. 이 표에 있는 각 주제에 대해 MQTT 테스트 클라이언트에서 다음을 수행합니다.

주제 이름	메시지 페이로드
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

   1. 게시(Publish)에서 테이블의 주제 이름(Topic name) 열에 있는 값을 입력합니다.
   2. 주제 이름 아래의 메시지 페이로드 필드에 테이블의 메시지 페이로드(Message payload) 열에 있는 값을 입력합니다.
   3. `pubsub.py`가 실행 중인 터미널 창을 살펴보고 MQTT 테스트 클라이언트에서 주제에 게시(Publish to topic)를 선택합니다.

   터미널 창에서 `pubsub.py`가 메시지를 수신했음을 확인해야 합니다.

터미널 창에서 이와 유사한 결과가 나타날 것입니다.

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID 'test-
af794be0-7542-45a0-b0af-0b0ea7474517'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
Received message from topic 'device/light/details': b'{ "desiredLight": 100,
 "currentLight": 50 }'
Received light request: b'{ "desiredLight": 100, "currentLight": 50 }'
Received message from topic 'device/temp/details': b'{ "desiredTemp": 20,
 "currentTemp": 15 }'
Received temperature request: b'{ "desiredTemp": 20, "currentTemp": 15 }'
2 message(s) received.
Disconnecting...
Disconnected!
```

연습 결과

이 연습에서는 샘플 앱이 콜백 함수에서 여러 메시지를 인식하고 처리할 수 있도록 코드를 추가했습니다. 이를 통해 디바이스가 메시지를 수신하고 이에 대한 조치를 취할 수 있습니다.

디바이스에서 여러 메시지를 수신하고 처리하는 또 다른 방법은 서로 다른 메시지를 별도로 구독하고 각 구독을 자체 콜백 함수에 할당하는 것입니다.

디바이스에서 메시지 게시

pubsub.py 샘플 앱을 사용하여 디바이스에서 메시지를 게시할 수 있습니다. 메시지를 그대로 게시하지만 메시지를 JSON 문서로 읽을 수 없습니다. 이 연습에서는 읽을 수 있는 메시지 페이로드에 JSON 문서를 게시할 수 있도록 샘플 앱을 수정합니다 AWS IoT Core.

연습 절차

이 연습에서는 `device/data` 주제와 함께 다음 메시지가 전송됩니다.

```
{
    "timestamp": 1601048303,
    "sensorId": 28,
    "sensorData": [
        {
        "sensorName": "Wind speed",
        "sensorValue": 34.2211224
        }
```

```
    ]
}
```

이 연습에서 발생한 메시지를 모니터링하기 위해 MQTT 테스트 클라이언트를 준비하려면

1. 주제 구독(Subscribe to a topic)의 Subscription topic field(구독 주제 필드)에 주제 필터 **device/data**를 입력한 다음 주제 구독(Subscribe to topic)을 선택합니다.
2. MQTT 테스트 클라이언트의 구독(Subscriptions) 열에서 device/data를 선택합니다.
3. MQTT 테스트 클라이언트 창을 열어 두고 디바이스의 메시지를 기다립니다.

pubsub.py 샘플 앱을 사용하여 JSON 문서를 전송하려면

1. 디바이스에서 pubsub.py를 pubsub3.py로 복사합니다.
2. 게시하는 메시지의 형식을 변경하도록 pubsub3.py를 편집합니다.
   a. 텍스트 편집기에서 pubsub3.py을(를) 엽니다.
   b. 다음 코드 행을 찾습니다.

      ```
      message = "{} [{}]".format(message_string, publish_count)
      ```

   c. 다음과 같이 변경하세요.

      ```
      message = "{}".format(message_string)
      ```

   d. 다음 코드 행을 찾습니다.

      ```
      message_json = json.dumps(message)
      ```

   e. 다음과 같이 변경하세요.

      ```
      message = "{}".json.dumps(json.loads(message))
      ```

   f. 변경 내용을 저장합니다.
3. 디바이스에서 이 명령을 실행하여 메시지를 두 번 전송합니다.

   ```
   python3 pubsub3.py  --ca_file ~/certs/Amazon-root-CA-1.pem  --cert ~/certs/
   device.pem.crt  --key ~/certs/private.pem.key  --topic device/data  --count 2 --
   message '{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
    speed","sensorValue":34.2211224}]}'  --endpoint your-iot-endpoint
   ```

4. MQTT 테스트 클라이언트에서 다음과 같이 메시지 페이로드에서 JSON 문서를 해석하고 형식을 지정했는지 확인합니다.



기본적으로 pubsub3.py는 전송하는 메시지도 구독합니다. 앱의 출력에서 메시지를 수신한 것을 볼 수 있습니다. 터미널 창은 다음과 유사해야 합니다.

```
Connecting to a3qEXAMPLEsffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-5cff18ae-1e92-4c38-a9d4-7b9771afc52f'...
Connected!
Subscribing to topic 'device/data'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 2 message(s)
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
2 message(s) received.
Disconnecting...
Disconnected!
```

연습 결과

이를 통해 디바이스는에 보낼 메시지를 생성 AWS IoT Core 하여 기본 연결을 테스트하고가 처리할 디바이스 메시지를 제공할 수 AWS IoT Core 있습니다. 예를 들어이 앱을 사용하여 디바이스에서 테스트 데이터를 전송하여 AWS IoT 규칙 작업을 테스트할 수 있습니다.

결과 검토

이 자습서의 예제는 디바이스가 AWS IoT 솔루션의 기본 부분 AWS IoT Core인와 통신하는 방법의 기본 사항에 대한 실습 경험을 제공합니다. 디바이스와 통신할 수 있는 경우 서비스 및 디바이스가 AWS 작동할 AWS IoT Core수 있는 다른 디바이스에 메시지를 전달할 수 있습니다. 마찬가지로 AWS 서비스 및 기타 디바이스는 디바이스로 다시 전송되는 메시지를 생성하는 정보를 처리할 수 있습니다.

AWS IoT Core 더 자세히 살펴볼 준비가 되면 다음 자습서를 시도해 보세요.

- the section called “Amazon SNS 알림 전송”
- the section called “DynamoDB 테이블에 디바이스 데이터 저장”
- the section called “ AWS Lambda 함수를 사용하여 알림 형식 지정”

자습서: AWS IoT Device SDK for Embedded C 사용

이 섹션에서는 AWS IoT Device SDK for Embedded C를 실행하는 방법에 대해 설명합니다.

이 섹션의 절차

- 1단계: AWS IoT Device SDK for Embedded C 설치
- 2단계: 샘플 앱 구성
- 3단계: 샘플 애플리케이션 빌드 및 실행

1단계: AWS IoT Device SDK for Embedded C 설치

AWS IoT Device SDK for Embedded C는 일반적으로 최적화된 C 언어 런타임을 필요로 하는 리소스 제약 디바이스를 대상으로 합니다. 모든 운영 체제에서 SDK를 사용하고, 모든 프로세서 유형(예: MCU 및 MPU)에서 호스팅할 수 있습니다. 더 많은 메모리 및 처리 리소스를 사용할 수 있는 경우, 상위 순위의 AWS IoT 디바이스 및 모바일 SDK(예: C++, Java, JavaScript 및 Python) 중 하나를 사용하도록 하세요.

일반적으로 AWS IoT Device SDK for Embedded C는 임베디드 운영 체제를 실행하는 MCU 또는 저사양의 MPU를 사용하는 시스템용입니다. 이 섹션의 프로그래밍 예제에서는 디바이스가 Linux를 사용한다고 가정합니다.

Example

1. GitHub에서 AWS IoT Device SDK for Embedded C를 디바이스로 다운로드하세요.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-embedded-c.git --recurse-
   submodules
   ```

   그러면 현재 디렉토리에 `aws-iot-device-sdk-embedded-c`라는 디렉터리가 생성됩니다.

2. 해당 디렉터리로 이동하여 최신 릴리스를 체크아웃합니다. 최신 릴리스 태그는 github.com/aws/aws-iot-device-sdk-embedded-C/tags를 참조하세요.

   ```
   cd aws-iot-device-sdk-embedded-c
   git checkout latest-release-tag
   ```

3. OpenSSL 버전 1.1.0 이상을 설치합니다. OpenSSL 개발 라이브러리는 패키지 관리자를 통해 설치될 때 일반적으로 'libssl-dev' 또는 'openssl-devel'이라고 합니다.

   ```
   sudo apt-get install libssl-dev
   ```

2단계: 샘플 앱 구성

AWS IoT Device SDK for Embedded C에는 사용자가 시도할 수 있는 샘플 애플리케이션이 포함되어 있습니다. 간단한 설명을 위해 이 자습서에서는 `mqtt_demo_mutual_auth` 애플리케이션을 사용합니다. 이 애플리케이션은 AWS IoT Core 메시지 브로커에 연결하여 MQTT 주제를 구독하고 게시하는 방법을 보여줍니다.

1. AWS IoT Core 자습서 시작하기에서 만든 인증서와 개인 키를 `build/bin/certificates` 디렉터리에 복사합니다.

   > **Note**
   >
   > 디바이스 및 루트 CA 인증서는 만료 또는 취소될 수 있습니다. 인증서가 만료되거나 취소되면 새 CA 인증서 또는 프라이빗 키와 디바이스 인증서를 디바이스에 복사해야 합니다.

2. 개인 AWS IoT Core 엔드포인트, 프라이빗 키, 인증서 및 루트 CA 인증서를 사용하여 샘플을 구성해야 합니다. `aws-iot-device-sdk-embedded-c/demos/mqtt/mqtt_demo_mutual_auth` 디렉터리로 이동합니다.

   AWS CLI를 설치한 경우 이 명령을 사용하여 계정의 엔드포인트 URL을 찾을 수 있습니다.

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

   AWS CLI를 설치하지 않은 경우 AWS IoT 콘솔을 엽니다. 탐색 창에서 Manage(관리)를 선택한 다음 Things(사물)를 선택합니다. 디바이스에서 IoT 사물을 선택한 다음 상호 작용(Interact)을 선택합니다. 사물 세부 정보 페이지의 HTTPS 섹션에 엔드포인트가 표시됩니다.

3. `demo_config.h` 파일을 열어 다음의 값을 업데이트합니다.

   AWS_IOT_ENDPOINT

   개인 엔드포인트입니다.

   CLIENT_CERT_PATH

   인증서 파일 경로(예: `certificates/device.pem.crt"`)입니다.

   CLIENT_PRIVATE_KEY_PATH

   프라이빗 키 파일 이름(예: `certificates/private.pem.key`)입니다.

   다음 예를 참조하세요.

   ```
   // Get from demo_config.h
   // ================================================
   #define AWS_IOT_ENDPOINT                "my-endpoint-ats.iot.us-
   east-1.amazonaws.com"
   #define AWS_MQTT_PORT                   8883
   #define CLIENT_IDENTIFIER               "testclient"
   #define ROOT_CA_CERT_PATH               "certificates/AmazonRootCA1.crt"
   #define CLIENT_CERT_PATH                "certificates/my-device-cert.pem.crt"
   #define CLIENT_PRIVATE_KEY_PATH         "certificates/my-device-private-key.pem.key"
   // ================================================
   ```

4. 이 명령을 사용하여 디바이스에 CMake가 설치되어 있는지 확인하세요.

   ```
   cmake --version
   ```

컴파일러에 대한 버전 정보가 표시되면 다음 섹션을 계속 진행할 수 있습니다.

오류가 발생하거나 정보가 표시되지 않는 경우 이 명령을 사용하여 cmake 패키지를 설치해야 합니다.

```
sudo apt-get install cmake
```

cmake --version 명령을 다시 실행하고 CMake가 설치되었으며 계속할 준비가 되었는지 확인합니다.

5. 이 명령을 사용하여 디바이스에 개발 도구가 설치되어 있는지 확인하세요.

```
gcc --version
```

컴파일러에 대한 버전 정보가 표시되면 다음 섹션을 계속 진행할 수 있습니다.

오류가 발생하거나 컴파일러 정보가 표시되지 않는 경우 이 명령을 사용하여 `build-essential` 패키지를 설치해야 합니다.

```
sudo apt-get install build-essential
```

gcc --version 명령을 다시 실행하고 빌드 도구가 설치되었으며 계속할 준비가 되었는지 확인합니다.

3단계: 샘플 애플리케이션 빌드 및 실행

이 절차에서는 디바이스에서 `mqtt_demo_mutual_auth` 애플리케이션을 생성하고 AWS IoT Device SDK for Embedded C를 사용하여 AWS IoT 콘솔에 연결하는 방법을 설명합니다.

AWS IoT Device SDK for Embedded C 샘플 애플리케이션을 실행하려면

1. `aws-iot-device-sdk-embedded-c`로 이동하여 빌드 디렉터리를 생성합니다.

```
mkdir build && cd build
```

2. 다음 CMake 명령을 입력하여 빌드에 필요한 Makefile을 생성합니다.

```
cmake ..
```

3. 다음 명령어를 입력하여 실행 가능한 앱 파일을 빌드합니다.

```
make
```

4. 이 명령으로 mqtt_demo_mutual_auth 앱을 실행합니다.

```
cd bin
./mqtt_demo_mutual_auth
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:584] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8883.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1264] Creating an MQTT connection to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com.
[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt_serializer.c:970] CONNACK session present bit not set.
[INFO] [MQTT] [core_mqtt_serializer.c:912] Connection accepted.
[INFO] [MQTT] [core_mqtt.c:1526] Received MQTT CONNACK successfully from broker.
[INFO] [MQTT] [core_mqtt.c:1792] MQTT connection established with the broker.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1033] MQTT connection successfully established with broker.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1296] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1314] Subscribing to the MQTT topic testclient/example/topic.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1097] SUBSCRIBE sent for topic testclient/example/topic to broker.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:921] Subscribed to the topic testclient/example/topic. with maximum QoS 1.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1358] Sending Publish to the MQTT topic testclient/example/topic.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1195] PUBLISH sent for topic testclient/example/topic to broker with packet ID 2.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt.c:1126] Ack packet deserialized with result: MQTTSuccess.
[INFO] [MQTT] [core_mqtt.c:1139] State record updated. New state=MQTTPublishDone.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:946] PUBACK received for packet id 2.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:672] Cleaned up outgoing publish packet with packet id 2.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=40.
[INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
```

이제 디바이스가 AWS IoT Device SDK for Embedded C를 사용하여 AWS IoT에 연결되었습니다.

AWS IoT 콘솔을 사용하여 샘플 앱이 게시하는 MQTT 메시지를 확인할 수도 있습니다. AWS IoT 콘솔에서 MQTT 클라이언트를 사용하는 방법에 대한 내용은 the section called “MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기” 단원을 참조하세요.

디바이스 데이터를 다른 서비스로 라우팅하는 AWS IoT 규칙 생성

이 자습서에서는 보다 일반적인 AWS IoT 규칙 작업 중 일부를 사용하여 규칙을 생성하고 테스트하는 방법을 보여줍니다.

AWS IoT 규칙은 디바이스에서 다른 AWS 서비스로 데이터를 전송합니다. 이들은 특정 MQTT 메시지를 수신하고, 메시지 페이로드의 데이터를 형식화한 다음 결과를 다른 AWS 서비스로 전송합니다.

Lambda 함수 또는 더 복잡한 규칙을 사용하는 규칙을 만드는 것이 목표인 경우에도 여기에 표시된 순서대로 시도해 보는 것이 좋습니다. 자습서는 기본적인 것부터 복잡한 것의 순서로 제공됩니다. 새로운 개념을 점진적으로 제시하여 특정 자습서가 없는 규칙 작업을 만드는 데 사용할 수 있는 개념을 익힐 수 있습니다.

> **Note**
>
> AWS IoT 규칙은 IoT 디바이스에서 다른 AWS 서비스로 데이터를 전송하는 데 도움이 됩니다. 그러나 이를 성공적으로 수행하려면 데이터를 전송하려는 다른 서비스에 대한 실무 지식이 필요합니다. 이 자습서에서는 작업을 완료하는 데 필요한 정보를 제공하지만 솔루션에서 데이터를 사용하기 전에 데이터를 전송할 서비스에 대해 자세히 알아보는 것이 도움이 될 수 있습니다. 다른 AWS 서비스에 대한 자세한 설명은이 자습서의 범위를 벗어납니다.

자습서 시나리오 개요

이 자습서의 시나리오는 주기적으로 데이터를 게시하는 기상 센서 디바이스의 시나리오입니다. 이 가상 시스템에는 많은 센서 디바이스가 있습니다. 그러나 이 섹션의 자습서에서는 단일 디바이스에 초점을 맞추면서 여러 센서를 수용할 수 있는 방법을 보여 줍니다.

이 섹션의 자습서에서는 AWS IoT 규칙을 사용하여이 가상 기상 센서 디바이스 시스템으로 다음 작업을 수행하는 방법을 보여줍니다.

- 자습서: MQTT 메시지 재게시

  이 자습서에서는 기상 센서에서 받은 MQTT 메시지를 센서 ID와 온도 값만 포함하는 메시지로 다시 게시하는 방법을 보여 줍니다. AWS IoT Core 서비스만 사용하고 간단한 SQL 쿼리와 MQTT 클라이언트를 사용하여 규칙을 테스트하는 방법을 보여줍니다.
- 자습서: Amazon SNS 알림 전송

  이 자습서에서는 기상 센서 디바이스의 값이 특정 값을 초과할 때 SNS 메시지를 전송하는 방법을 보여 줍니다. 이전 자습서에 제시된 개념을 기반으로 하며 다른 AWS 서비스인 Amazon Simple Notification Service(Amazon SNS)로 작업하는 방법을 추가합니다.

  Amazon SNS를 처음 사용한다면 이 자습서를 시작하기 전에 시작하기 연습을 검토합니다.
- 자습서: DynamoDB 테이블에 디바이스 데이터 저장

이 자습서에서는 기상 센서 디바이스의 데이터를 데이터베이스 테이블에 저장하는 방법을 보여 줍니다. 규칙 쿼리 문과 대체 템플릿을 사용하여 대상 서비스에 대한 Amazon DynamoDB 메시지 데이터 형식을 지정합니다.

DynamoDB를 처음 사용한다면 이 자습서를 시작하기 전에 시작하기 연습을 검토합니다.

- 자습서: AWS Lambda 함수를 사용하여 알림 형식 지정

  이 자습서에서는 Lambda 함수를 호출하여 디바이스 데이터를 다시 형식 지정한 다음 문자 메시지로 전송하는 방법을 보여 줍니다. AWS Lambda 함수에 Python 스크립트 및 AWS SDK 함수를 추가하여 날씨 센서 디바이스의 메시지 페이로드 데이터로 형식을 지정하고 문자 메시지를 전송합니다.

  Lambda를 처음 사용한다면 이 자습서를 시작하기 전에 시작하기 연습을 검토합니다.

AWS IoT 규칙 개요

이러한 모든 자습서가 AWS IoT 규칙을 생성합니다.

디바이스에서 다른 AWS 서비스로 데이터를 전송하는 AWS IoT 규칙의 경우 다음을 사용합니다.

- 다음으로 구성된 규칙 쿼리 문입니다.
  - 메시지 페이로드에서 데이터를 선택하고 형식을 지정하는 SQL SELECT 절
  - 사용할 메시지를 식별하는 주제 필터(규칙 쿼리 문의 FROM 객체)
  - 작동할 조건을 지정하는 선택적 조건문(SQL WHERE 절)
- 하나 이상의 규칙 작업

디바이스는 MQTT 주제에 메시지를 게시합니다. SQL SELECT 문의 주제 필터는 규칙을 적용할 MQTT 주제를 식별합니다. SQL SELECT 문에 지정된 필드는 규칙 작업에 사용할 수신 MQTT 메시지 페이로드의 데이터 형식을 지정합니다. 규칙 작업의 전체 목록은 AWS IoT 규칙 작업 단원을 참조하세요.

이 섹션의 자습서

- 자습서: MQTT 메시지 재게시
- 자습서: Amazon SNS 알림 전송
- 자습서: DynamoDB 테이블에 디바이스 데이터 저장
- 자습서: AWS Lambda 함수를 사용하여 알림 형식 지정

자습서: MQTT 메시지 재게시

이 자습서에서는 지정된 MQTT 메시지가 수신될 때 MQTT 메시지를 게시하는 AWS IoT 규칙을 생성하는 방법을 보여줍니다. 수신 메시지 페이로드는 게시되기 전에 규칙에 의해 수정될 수 있습니다. 이를 통해 디바이스나 펌웨어를 변경할 필요 없이 특정 애플리케이션에 맞는 메시지를 생성할 수 있습니다. 규칙의 필터링 측면을 사용하여 특정 조건이 충족되는 경우에만 메시지를 게시할 수도 있습니다.

규칙에 의해 다시 게시된 메시지는 다른 AWS IoT 디바이스 또는 클라이언트에서 보낸 메시지처럼 작동합니다. 디바이스는 다른 MQTT 메시지 주제를 구독할 수 있는 것과 동일한 방식으로 다시 게시된 메시지를 구독할 수 있습니다.

이 자습서에서 배울 내용:

- 규칙 쿼리 문에서 간단한 SQL 쿼리 및 함수를 사용하는 방법
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트하는 방법

이 자습서는 완료하는 데 약 30분 걸립니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- MQTT 주제 및 AWS IoT 규칙 검토
- 1단계: MQTT 메시지를 다시 게시하는 AWS IoT 규칙 생성
- 2단계: 새 규칙 테스트
- 3단계: 결과 및 다음 단계 검토

이 자습서를 시작하기 전에 다음 사항을 확인해야 합니다.

- 설정 AWS 계정

  이 자습서를 완료하려면 AWS 계정 및 AWS IoT 콘솔이 필요합니다.
- 검토된 MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기

  MQTT 클라이언트를 사용하여 주제를 구독하고 게시할 수 있는지 확인하세요. 이 절차에서는 MQTT 클라이언트를 사용하여 새 규칙을 테스트합니다.

MQTT 주제 및 AWS IoT 규칙 검토

AWS IoT 규칙에 대해 설명하기 전에 MQTT 프로토콜을 이해하는 데 도움이 됩니다. IoT 솔루션에서 MQTT 프로토콜은 HTTP와 같은 다른 네트워크 통신 프로토콜에 비해 몇 가지 이점을 제공하므로 IoT

디바이스에서 널리 사용됩니다. 이 섹션에서는 이 자습서에 적용되는 MQTT의 주요 측면을 검토합니다. MQTTT를 HTTP와 비교하는 방법에 대한 자세한 내용은 디바이스 통신을 위한 애플리케이션 프로토콜 선택 단원을 참조하세요.

MQTT 프로토콜

MQTT 프로토콜은 호스트와 게시/구독 통신 모델을 사용합니다. 데이터를 전송하기 위해 디바이스는 주제로 식별되는 메시지를 AWS IoT 메시지 브로커에 게시합니다. 메시지 브로커로부터 메시지를 수신하기 위해 디바이스는 구독 요청의 주제 필터를 메시지 브로커에 전송하여 수신할 주제를 구독합니다. AWS IoT 규칙 엔진은 메시지 브로커에서 MQTT 메시지를 수신합니다.

AWS IoT 규칙

AWS IoT 규칙은 규칙 쿼리 문과 하나 이상의 규칙 작업으로 구성됩니다. AWS IoT 규칙 엔진이 MQTT 메시지를 수신하면 이러한 요소는 다음과 같이 메시지에 작동합니다.

- 규칙 쿼리 문

  규칙의 쿼리 문은 사용할 MQTT 주제를 설명하고, 메시지 페이로드의 데이터를 해석하며, 널리 사용되는 SQL 데이터베이스에서 사용하는 문과 유사한 SQL 문에 설명된 대로 데이터의 형식을 지정합니다. 쿼리 문의 결과는 규칙의 작업으로 전송되는 데이터입니다.
- 규칙 작업

  규칙의 각 규칙 작업은 규칙의 쿼리 문에서 발생하는 데이터에 적용됩니다.는 많은 규칙 작업을 AWS IoT 지원합니다. 그러나 이 자습서에서는 쿼리 문의 결과를 특정 주제가 포함된 MQTT 메시지로 게시하는 재게시 규칙 작업에 집중할 것입니다.

1단계: MQTT 메시지를 다시 게시하는 AWS IoT 규칙 생성

이 자습서에서 생성할 AWS IoT 규칙은 *device_id*가 메시지를 전송한 디바이스의 ID인 device/*device_id*/data MQTT 주제를 구독합니다. 이러한 주제는 주제 필터에 의해 device/+/data로 설명됩니다. 여기서 +는 두 개의 슬래시 문자 사이의 모든 문자열과 일치하는 와일드카드 문자입니다.

규칙은 일치하는 주제에서 메시지를 수신하면 device_id 및 temperature 값을 device/data/temp 주제가 있는 새 MQTT 메시지로 다시 게시합니다.

예를 들어, device/22/data 주제가 있는 MQTT 메시지의 페이로드는 다음과 같습니다.

```
{
  "temperature": 28,
```

```
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

규칙은 메시지 페이로드에서 temperature 값을 가져오고 주제에서 device_id를 가져와 다음과 같은 device/data/temp 주제 및 메시지 페이로드가 있는 MQTT 메시지로 다시 게시합니다.

```
{
  "device_id": "22",
  "temperature": 28
}
```

이 규칙을 사용할 경우 디바이스의 ID와 온도 데이터만 필요한 디바이스는 해당 정보만 수신하기 위해 device/data/temp 주제를 구독합니다.

MQTT 메시지를 다시 게시하는 규칙을 생성하려면

1. AWS IoT 콘솔의 규칙 허브를 엽니다.
2. 규칙에서 생성을 선택하고 새 규칙 생성을 시작합니다.
3. 규칙 생성의 상단 부분에서:

   a. 이름에 규칙 이름을 입력합니다. 본 자습서에서는 이름을 **republish_temp**로 지정합니다.

      규칙 이름은 계정 및 지역 내에서 고유해야 하며 공백을 포함할 수 없습니다. 이 이름에 밑줄 문자를 사용하여 규칙 이름의 두 단어를 구분했습니다.

   b. 설명에서 규칙을 설명합니다.

      의미 있는 설명은 이 규칙이 수행하는 작업과 규칙을 생성한 이유를 기억하는 데 도움이 됩니다. 설명은 필요한 만큼 길어질 수 있으므로 가능한 한 자세하게 설명하세요.
4. 규칙 생성의 규칙 쿼리 문에서:

   a. SQL 버전 사용에서 **2016-03-23**을(를) 선택합니다.

   b. 규칙 쿼리 문 편집 상자에 쿼리 문을 입력합니다.

```
SELECT topic(2) as device_id, temperature FROM 'device/+/data'
```

이 문은 다음을 수행합니다.

- `device/+/data` 주제 필터와 일치하는 주제가 있는 MQTT 메시지를 수신합니다.
- 주제 문자열에서 두 번째 요소를 선택하여 `device_id` 필드에 할당합니다.
- 메시지 페이로드에서 `temperature` 값 필드를 선택하여 `temperature` 필드에 할당합니다.

5. 하나 이상의 작업 설정에서:

   a. 이 규칙에 대한 규칙 작업 목록을 열려면 작업 추가를 선택합니다.

   b. 작업 선택에서 AWS IoT 주제에 메시지 다시 게시를 선택합니다.

   c. 작업 목록 하단에서 구성 작업을 선택하여 선택한 작업의 구성 페이지를 엽니다.

6. 구성 작업에서:

   a. 주제에 **device/data/temp**를 입력합니다. 이 규칙이 게시할 메시지의 MQTT 주제입니다.

   b. 서비스 품질(QoS)에서 0 - 메시지가 0번 이상 전송됨을 선택합니다.

   c. 이 작업을 수행할 수 있는 AWS IoT 액세스 권한을 부여할 역할 선택 또는 생성에서:

      i. 역할 생성을 선택합니다. 새 역할 생성 대화 상자가 열립니다.

      ii. 새 역할을 설명하는 이름을 입력합니다. 본 자습서에서는 **republish_role**을 사용합니다.

      새 역할을 만들면 규칙 작업을 수행할 올바른 정책이 만들어지고 새 역할에 연결됩니다. 이 규칙 작업의 주제를 변경하거나 다른 규칙 작업에서 이 역할을 사용하는 경우 새 주제 또는 작업에 권한을 부여하도록 해당 역할에 대한 정책을 업데이트해야 합니다. 기존 역할을 업데이트하려면 이 섹션의 역할 업데이트를 선택합니다.

      iii. 역할 생성을 선택하여 역할을 생성하고 대화 상자를 닫습니다.

   d. 작업 추가를 선택하여 규칙에 작업을 추가하고 규칙 생성 페이지로 이동합니다.

7. 이제 AWS IoT 주제에 메시지 다시 게시 작업이 하나 이상의 작업 설정에 나열됩니다.

   새 작업 타일의 AWS IoT 주제에 메시지 재게시 아래에서 재게시 작업으로 게시할 주제를 볼 수 있습니다.

   이 규칙 작업은 이 규칙에 추가할 유일한 규칙 작업입니다.

8. 규칙 생성에서 하단으로 스크롤하고 규칙 생성을 선택하여 규칙을 생성하고 이 단계를 완료합니다.

2단계: 새 규칙 테스트

새 규칙을 테스트하려면 MQTT 클라이언트를 사용하여 이 규칙에서 사용하는 MQTT 메시지를 게시하고 구독합니다.

새 창에서 AWS IoT 콘솔의 MQTT 클라이언트를 엽니다. 이렇게 하면 MQTT 클라이언트의 구성을 그대로 유지하면서 규칙을 편집할 수 있습니다. MQTT 클라이언트는 콘솔의 다른 페이지로 이동하기 위해 나가면 구독 또는 메시지 로그를 유지하지 않습니다.

MQTT 클라이언트를 사용하여 규칙을 테스트하려면

1. AWS IoT 콘솔의 MQTT 클라이언트에서 입력 주제(이 경우 `device/+/data`)를 구독합니다.

   a. MQTT 클라이언트에서 구독 아래에서 주제 구독을 선택합니다.
   b. 구독 주제에 입력 주제 필터 **`device/+/data`**의 주제를 입력합니다.
   c. 나머지 필드는 기본 설정을 유지합니다.
   d. 주제 구독을 선택합니다.

   구독 열의 주제 게시 아래에 **`device/+/data`**가 나타납니다.
2. 규칙에서 게시할 주제 `device/data/temp`를 구독합니다.

   a. 구독 아래에서 주제 구독을 다시 선택하고, 구독 주제에 다시 게시된 메시지의 주제 **`device/data/temp`**를 입력합니다.
   b. 나머지 필드는 기본 설정을 유지합니다.
   c. 주제 구독을 선택합니다.

   구독 열의 device/+/data에서 **`device/data/temp`**가 나타납니다.
3. 특정 디바이스 ID **`device/22/data`**로 입력 주제에 메시지를 게시합니다. 와일드카드 문자가 포함된 MQTT 항목에는 게시할 수 없습니다.

   a. MQTT 클라이언트의 구독 아래에서 주제 게시를 선택합니다.
   b. 게시 필드에 입력 주제 이름 **`device/22/data`**를 입력합니다.
   c. 여기에 표시된 샘플 데이터를 복사하고 주제 이름 아래의 편집 상자에 샘플 데이터를 붙여 넣습니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
```

```
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. MQTT 메시지를 전송하려면 주제 게시를 선택합니다.

4. 전송한 메시지를 검토합니다.

a. MQTT 클라이언트의 구독 아래에서 이전에 구독한 두 주제 옆에 녹색 점이 있습니다.

녹색 점은 마지막으로 메시지를 본 이후 하나 이상의 새 메시지가 수신되었음을 나타냅니다.

b. 구독 아래에서 device/+/data를 선택하여 메시지 페이로드가 방금 게시한 것과 일치하고 다음과 같이 표시되는지 확인하세요.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. 구독 아래에서 device/data/temp를 선택하여 게시된 메시지 페이로드가 다음과 같이 표시되는지 확인하세요.

```
{
  "device_id": "22",
  "temperature": 28
}
```

`device_id` 값은 따옴표로 묶인 문자열이고 `temperature` 값은 숫자여야 합니다. 이는 [topic()] 함수가 입력 메시지의 주제 이름에서 문자열을 추출하는 반면 `temperature` 값은 입력 메시지의 페이로드에서 숫자 값을 사용하기 때문입니다.

`device_id` 값을 숫자 값으로 만들려면 규칙 쿼리 문에서 `topic(2)`를 다음으로 바꿉니다.

```
cast(topic(2) AS DECIMAL)
```

`topic(2)` 값을 숫자 값으로 캐스팅하는 것은 주제의 해당 부분에 숫자만 포함된 경우에만 작동합니다.

5. device/data/temp 주제에 올바른 메시지가 게시되었으면 규칙이 작동한 것입니다. 다음 섹션에서 재게시 규칙 작업에 대해 자세히 알아볼 수 있는 내용을 확인하세요.

   device/+/data 또는 device/data/temp 주제에 올바른 메시지가 게시되었음을 확인할 수 없는 경우 문제 해결 팁을 확인하세요.

재게시 메시지 규칙 문제 해결

다음은 예상한 결과가 표시되지 않는 경우에 대비하여 확인해야 할 몇 가지 사항입니다.

- 오류 배너가 있음

  입력 메시지를 게시할 때 오류가 나타나면 먼저 해당 오류를 수정하세요. 다음 단계는 해당 오류를 수정하는 데 도움이 될 수 있습니다.

- MQTT 클라이언트에서 입력 메시지가 표시되지 않음

  `device/+/data` 주제 필터를 구독한 경우 입력 메시지를 `device/22/data` 주제에 게시할 때마다 절차에 설명된 대로 해당 메시지가 MQTT 클라이언트에 나타나야 합니다.

  확인해야 할 사항

  - 구독한 주제 필터 확인

    절차에 설명된 대로 입력 메시지 주제를 구독한 경우 게시할 때마다 입력 메시지의 복사본이 표시되어야 합니다.

    메시지가 표시되지 않으면 구독한 주제 이름을 확인하고 게시한 주제와 비교합니다. 주제 이름은 대소문자를 구분하며 구독한 주제는 메시지 페이로드를 게시한 주제와 동일해야 합니다.

  - 메시지 게시 함수 확인

    MQTT 클라이언트의 구독에서 device/+/data를 선택하고 게시 메시지의 주제를 확인한 다음 주제에 게시를 선택합니다. 주제 아래의 편집 상자에서 메시지 페이로드가 메시지 목록에 나타납니다.

- MQTT 클라이언트에서 다시 게시된 메시지가 표시되지 않음

  규칙이 작동하려면 메시지를 수신하고 다시 게시하도록 권한을 부여하는 올바른 정책이 있어야 하며 메시지를 수신해야 합니다.

확인해야 할 사항

- MQTT 클라이언트 AWS 리전 의 및 생성한 규칙을 확인합니다.

  MQTT 클라이언트를 실행 중인 콘솔은 생성한 규칙과 동일한 AWS 리전에 있어야 합니다.

- 규칙 쿼리 문의 입력 메시지 주제 확인

  규칙이 작동하려면 규칙 쿼리 문의 FROM 절에 있는 주제 필터와 일치하는 주제 이름이 포함된 메시지를 받아야 합니다.

  MQTT 클라이언트에 있는 주제의 철자와 함께 규칙 쿼리 문에서 주제 필터의 철자를 확인하세요. 주제 이름은 대/소문자를 구분하며 메시지의 주제는 규칙 쿼리 문의 주제 필터와 일치해야 합니다.

- 입력 메시지 페이로드의 내용 확인

  규칙이 작동하려면 SELECT 문에서 선언된 메시지 페이로드에서 데이터 필드를 찾아야 합니다.

  MQTT 클라이언트에 있는 메시지 페이로드의 철자와 함께 규칙 쿼리 명령문의 `temperature` 필드 철자를 확인하세요. 필드 이름은 대소문자를 구분하며 규칙 쿼리 문의 `temperature` 필드는 메시지 페이로드의 `temperature` 필드와 동일해야 합니다.

  메시지 페이로드의 JSON 문서의 형식이 올바른지 확인하세요. JSON에 쉼표 누락과 같은 오류가 있으면 규칙에서 읽을 수 없습니다.

- 규칙 동작에서 다시 게시된 메시지 주제 확인

  재게시 규칙 작업에서 새 메시지를 게시하는 주제는 MQTT 클라이언트에서 구독한 주제와 일치해야 합니다.

  콘솔에서 만든 규칙을 열고 규칙 작업이 메시지를 다시 게시할 주제를 확인합니다.

- 규칙에서 사용 중인 역할 확인

  규칙 작업에는 원래 주제를 받고 새 주제를 게시할 수 있는 권한이 있어야 합니다.

  메시지 데이터를 수신하고 다시 게시하도록 규칙을 인증하는 정책은 사용된 주제에 따라 다릅니다. 메시지 데이터를 다시 게시하는 데 사용되는 주제를 변경하는 경우 규칙 작업의 역할을 업데이트하여 현재 주제와 일치하도록 정책을 업데이트해야 합니다.

  문제가 의심되는 경우 규칙 재게시 작업을 편집하고 새 역할을 만듭니다. 규칙 작업에 의해 생성된 새 역할에는 이러한 작업을 수행하는 데 필요한 권한이 부여됩니다.

3단계: 결과 및 다음 단계 검토

이 자습서에서는

- 간단한 SQL 쿼리와 규칙 쿼리 문에서 몇 가지 함수를 사용하여 새 MQTT 메시지를 생성했습니다.
- 새 메시지를 다시 게시 하는 규칙을 만들었습니다.
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트했습니다.

다음 단계

이 규칙을 사용하여 몇 가지 메시지를 다시 게시한 후 이 규칙을 실험하여 자습서의 일부 측면을 변경하면 다시 게시된 메시지에 어떤 영향을 주는지 확인해 보세요. 다음은 시작하는 데 도움이 될 몇 가지 아이디어입니다.

- 입력 메시지의 주제에서 *device_id*를 변경하고 다시 게시된 메시지 페이로드의 영향을 관찰합니다.
- 규칙 쿼리 문에서 선택한 필드를 변경하고 다시 게시된 메시지 페이로드의 영향을 관찰합니다.
- 이 시리즈의 다음 자습서를 사용해 보고 자습서: Amazon SNS 알림 전송의 방법을 알아봅니다.

이 자습서에서 사용되는 규칙 다시 게시 작업은 규칙 쿼리 문을 디버깅하는 데 도움이 될 수도 있습니다. 예를 들어 규칙에 이 작업을 추가하여 규칙 쿼리 문이 규칙 작업에 사용되는 데이터의 서식을 지정하는 방법을 확인할 수 있습니다.

자습서: Amazon SNS 알림 전송

이 자습서에서는 MQTT 메시지 데이터를 SMS 문자 메시지로 전송할 수 있도록 Amazon SNS 주제로 전송하는 AWS IoT 규칙을 생성하는 방법을 보여줍니다.

이 자습서에서는 온도가 규칙에 설정된 값을 초과할 때마다 기후 센서에서 Amazon SNS 주제의 모든 구독자에게 메시지 데이터를 전송하는 규칙을 생성합니다. 이 규칙은 보고된 온도가 규칙에 의해 설정된 값을 초과할 때를 감지하고 디바이스 ID, 보고된 온도 및 초과된 온도 제한만 포함하는 새 메시지 페이로드를 생성합니다. 이 규칙은 새 메시지 페이로드를 JSON 문서로 SNS 주제에 전송하여 모든 구독자에게 SNS 주제를 알립니다.

이 자습서에서 배울 내용:

- Amazon SNS 알림을 생성하고 테스트하는 방법
- AWS IoT 규칙에서 Amazon SNS 알림을 호출하는 방법

- 규칙 쿼리 문에서 간단한 SQL 쿼리 및 함수를 사용하는 방법
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트하는 방법

이 자습서는 완료하는 데 약 30분 걸립니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 1단계: SMS 문자 메시지를 전송하는 Amazon SNS 주제 생성
- 2단계: 텍스트 메시지를 전송하는 AWS IoT 규칙 생성
- 3단계: AWS IoT 규칙 및 Amazon SNS 알림 테스트
- 4단계: 결과 및 다음 단계 검토

이 자습서를 시작하기 전에 다음 사항을 확인해야 합니다.

- 설정 AWS 계정

  이 자습서를 완료하려면 AWS 계정 및 AWS IoT 콘솔이 필요합니다.
- 검토된 MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기

  MQTT 클라이언트를 사용하여 주제를 구독하고 게시할 수 있는지 확인하세요. 이 절차에서는 MQTT 클라이언트를 사용하여 새 규칙을 테스트합니다.
- Amazon Simple Notification Service를 검토했습니다.

  이전에 Amazon SNS 사용한 적이 없는 경우 Amazon SNS 액세스 설정을 검토합니다. 다른 AWS IoT 자습서를 이미 완료한 경우가 AWS 계정 이미 올바르게 구성되어 있어야 합니다.

1단계: SMS 문자 메시지를 전송하는 Amazon SNS 주제 생성

이 절차에서는 날씨 센서가 메시지 데이터를 보낼 수 있는 Amazon SNS 주제를 생성하는 방법을 설명합니다. 그러면 Amazon SNS 주제가 SMS 텍스트 메시지를 통해 모든 구독자에게 초과된 온도 제한을 알립니다.

SMS 문자 메시지를 전송하는 Amazon SNS 주제를 생성하려면

1. Amazon SNS 주제를 생성합니다.

   a. Amazon SNS 콘솔에 로그인합니다.

   b. 왼쪽 탐색 창에서 주제를 선택합니다.

c. 주제(Topics) 페이지에서 새 주제 생성(Create new topic)을 선택합니다.

d. 세부 정보에서 표준 유형을 선택합니다. 기본적으로 콘솔은 FIFO 주제를 만듭니다.

e. 이름에 SNS 주제 이름을 입력합니다. 이 자습서에서는 **high_temp_notice**을 입력합니다.

f. 페이지 끝으로 스크롤하고 주제 생성을 선택합니다.

콘솔에 새 주제의 세부 정보 페이지가 열립니다.

2. Amazon SNS 구독을 생성합니다.

> **Note**
>
> 이 구독에서 사용하는 전화번호는 이 자습서에서 전송할 메시지에서 문자 메시지 요금을 부과할 수 있습니다.

a. high_temp_notice 주제 세부 정보 페이지에서 구독 생성(Create subscription)을 선택합니다.

b. 구독 생성에 있는 세부 정보 섹션의 프로토콜 목록에서 SMS를 선택합니다.

c. 엔드포인트에 문자 메시지를 받을 수 있는 전화 번호를 입력합니다. +로 시작하고 국가 및 지역 코드를 포함하고 다른 구두점은 포함하지 않도록 입력해야 합니다.

d. 구독 생성을 선택합니다.

3. Amazon SNS 알림을 테스트합니다.

a. Amazon SNS 콘솔의 왼쪽 탐색 창에서 주제를 선택합니다.

b. 주제의 세부 정보 페이지를 열려면 주제의 주제 목록에서 high_temp_notice를 선택합니다.

c. 주제에 메시지 게시(Publish message to topic) 페이지를 열려면 high_temp_notice 세부 정보 페이지에서 메시지 게시(Publish message)를 선택합니다.

d. 주제에 메시지 게시(Publish message to topic)에 있는 메시지 본문 섹션에서 엔드포인트로 전송할 메시지 본문(Message body to send to the endpoint)에 짧은 메시지를 입력합니다.

e. 페이지의 하단으로 스크롤하고 메시지 게시(Publish message)를 선택합니다.

f. 구독을 만들 때 이전에 사용한 번호가 있는 전화에서 메시지가 수신되었는지 확인하세요.

테스트 메시지를 받지 못한 경우 전화 번호와 휴대 전화 설정을 다시 확인하세요.

자습서를 계속하기 전에 Amazon SNS 콘솔에서 테스트 메시지를 게시할 수 있는지 확인하세요.

2단계: 텍스트 메시지를 전송하는 AWS IoT 규칙 생성

이 자습서에서 생성할 AWS IoT 규칙은 메시지를 전송한 디바이스의 ID*device_id*가 인 device/*device_id*/data MQTT 주제를 구독합니다. 이러한 주제는 주제 필터에서 device/+/data로 설명됩니다. 여기서 +는 두 개의 슬래시 문자 사이의 모든 문자열과 일치하는 와일드카드 문자입니다. 이 규칙은 메시지 페이로드의 temperature 필드 값도 테스트합니다.

규칙이 일치하는 주제로부터 메시지를 수신하면 주제 이름에서 *device_id*, 메시지 페이로드에서 temperature 값을 가져와 테스트 중인 제한에 대한 상수 값을 추가하고 이러한 값을 JSON 문서로 Amazon SNS 알림 주제에 전송합니다.

예를 들어, 기상 센서 디바이스 번호 32의 MQTT 메시지는 device/32/data 주제를 사용하고 다음과 같은 메시지 페이로드가 있습니다.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

규칙의 규칙 쿼리 문은 메시지 페이로드에서 temperature 값을 가져오고 주제 이름에서 *device_id*를 가져오고 상수 max_temperature 값을 추가하여 다음과 같은 메시지 페이로드를 Amazon SNS 주제로 전송합니다.

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30
}
```

제한 초과 온도 값을 감지하고 Amazon SNS 주제로 전송할 데이터를 생성하는 AWS IoT 규칙을 생성하려면

1. AWS IoT 콘솔의 규칙 허브를 엽니다.
2. 이것이 첫 번째 규칙인 경우 생성 또는 규칙 생성을 선택합니다.
3. 규칙 생성(Create a rule)에서:

a. 이름(Name)에 **temp_limit_notify**를 입력합니다.

   규칙 이름은 AWS 계정 및 리전 내에서 고유해야 하며 공백을 가질 수 없습니다. 이 이름에 밑줄 문자를 사용하여 규칙 이름의 단어를 구분했습니다.

b. 설명에서 규칙을 설명합니다.

   의미 있는 설명을 사용하면 이 규칙이 수행하는 작업과 규칙을 만든 이유를 쉽게 기억할 수 있습니다. 설명은 필요한 만큼 길어질 수 있으므로 가능한 한 자세하게 설명하세요.

4. 규칙 생성의 규칙 쿼리 문에서:

   a. SQL 버전 사용에서 2016-03-23을 선택합니다.

   b. 규칙 쿼리 문 편집 상자에 쿼리 문을 입력합니다.

```
SELECT topic(2) as device_id,
    temperature as reported_temperature,
    30 as max_temperature
  FROM 'device/+/data'
  WHERE temperature > 30
```

   이 문은 다음을 수행합니다.

   - `device/+/data` 주제 필터와 일치하고 `temperature` 값이 30보다 큰 주제가 있는 MQTT 메시지를 수신 대기합니다.
   - 주제 문자열에서 두 번째 요소를 선택하여 `device_id` 필드에 할당합니다.
   - 메시지 페이로드에서 `temperature` 값 필드를 선택하여 `reported_temperature` 필드에 할당합니다.
   - 한계 값을 나타내는 상수 값 `30`을 생성하고 `max_temperature` 필드에 할당합니다.

5. 이 규칙에 대한 규칙 작업 목록을 열려면 하나 이상의 작업 설정(Set one or more actions)에서 작업 추가(Add action)를 선택합니다.

6. Select an action(작업 선택)에서 SNS 푸시 알림으로 메시지 전송(Send a message as an SNS push notification)을 선택합니다.

7. 작업 목록 하단에서 선택한 작업의 구성 페이지를 열려면 구성 작업을 선택합니다.

8. 구성 작업에서:

   a. SNS 대상에서 선택(Select)을 클릭하고 high_temp_notice라는 이름의 SNS 주제를 찾은 다음 선택(Select)을 클릭합니다.

b. 메시지 형식(Message format)에서 RAW를 선택합니다.

c. 이 작업을 수행할 수 있는 AWS IoT 액세스 권한을 부여할 역할 선택 또는 생성에서 역할 생성을 선택합니다.

d. 새 역할 생성에서 이름에 새 역할의 고유 이름을 입력합니다. 본 자습서에서는 **sns_rule_role**를 사용합니다.

e. 역할 생성을 선택합니다.

이 자습서를 반복하거나 기존 역할을 재사용하는 경우 계속 진행하기 전에 역할 업데이트를 선택합니다. 그러면 SNS 대상에서 작동하도록 역할의 정책 문서가 업데이트됩니다.

9. 작업 추가를 선택하여 규칙 생성 페이지로 이동합니다.

새 작업의 타일에서 SNS 푸시 알림으로 메시지 전송하기(Send a message as an SNS push notification) 아래에서 규칙이 호출할 SNS 주제를 볼 수 있습니다.

이 규칙 작업은 이 규칙에 추가할 유일한 규칙 작업입니다.

10. 규칙을 생성하고 이 단계를 완료하려면 규칙 생성(Create a rule)에서 하단으로 스크롤하여 규칙 생성(Create rule)을 선택합니다.

3단계: AWS IoT 규칙 및 Amazon SNS 알림 테스트

새 규칙을 테스트하려면 MQTT 클라이언트를 사용하여 이 규칙에서 사용하는 MQTT 메시지를 게시하고 구독합니다.

새 창에서 AWS IoT 콘솔의 MQTT 클라이언트를 엽니다. 이렇게 하면 MQTT 클라이언트의 구성을 그대로 유지하면서 규칙을 편집할 수 있습니다. 콘솔의 다른 페이지로 이동하기 위해 MQTT 클라이언트를 나가면 구독이나 메시지 로그가 유지되지 않습니다.

MQTT 클라이언트를 사용하여 규칙을 테스트하려면

1. AWS IoT 콘솔의 MQTT 클라이언트에서 입력 주제(이 경우 `device/+/data`)를 구독합니다.

a. MQTT 클라이언트에서 구독 아래에서 주제 구독을 선택합니다.

b. 구독 주제에 입력 주제 필터 **`device/+/data`**의 주제를 입력합니다.

c. 나머지 필드는 기본 설정을 유지합니다.

d. 주제 구독을 선택합니다.

구독 열의 주제 게시 아래에 **`device/+/data`**가 나타납니다.

2. 특정 디바이스 ID **device/32/data**로 입력 주제에 메시지를 게시합니다. 와일드카드 문자가 포함된 MQTT 항목에는 게시할 수 없습니다.

   a. MQTT 클라이언트의 구독 아래에서 주제 게시를 선택합니다.

   b. 게시 필드에 입력 주제 이름 **device/32/data**를 입력합니다.

   c. 여기에 표시된 샘플 데이터를 복사하고 주제 이름 아래의 편집 상자에 샘플 데이터를 붙여 넣습니다.

   ```
   {
     "temperature": 38,
     "humidity": 80,
     "barometer": 1013,
     "wind": {
       "velocity": 22,
       "bearing": 255
     }
   }
   ```

   d. 주제에 게시를 선택하여 MQTT 메시지를 게시합니다.

3. 문자 메시지가 전송되었는지 확인합니다.

   a. MQTT 클라이언트의 구독 아래에서 이전에 구독한 주제 옆에 녹색 점이 있습니다.

   녹색 점은 마지막으로 메시지를 보았을 때 하나 이상의 새 메시지가 수신되었음을 나타냅니다.

   b. 구독 아래에서 device/+/data를 선택하여 메시지 페이로드가 방금 게시한 것과 일치하고 다음과 같이 표시되는지 확인하세요.

   ```
   {
     "temperature": 38,
     "humidity": 80,
     "barometer": 1013,
     "wind": {
       "velocity": 22,
       "bearing": 255
     }
   }
   ```

   c. SNS 주제를 구독하는 데 사용한 전화를 확인하고 메시지 페이로드의 내용이 다음과 같이 표시되는지 확인합니다.

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

`device_id` 값은 따옴표로 묶인 문자열이고 `temperature` 값은 숫자여야 합니다. 이는 `topic()` 함수가 입력 메시지의 주제 이름에서 문자열을 추출하는 반면 `temperature` 값은 입력 메시지의 페이로드에서 숫자 값을 사용하기 때문입니다.

`device_id` 값을 숫자 값으로 만들려면 규칙 쿼리 문에서 `topic(2)`를 다음으로 바꿉니다.

```
cast(topic(2) AS DECIMAL)
```

주제의 해당 부분에 숫자만 포함된 경우에만 `topic(2)` 값을 숫자 `DECIMAL` 값으로 캐스팅할 수 있다는 점을 유의하세요.

4. 온도가 한계를 초과하지 않는 MQTT 메시지를 전송해보세요.

   a. MQTT 클라이언트의 구독 아래에서 주제 게시를 선택합니다.

   b. 게시 필드에 입력 주제 이름 **device/33/data**를 입력합니다.

   c. 여기에 표시된 샘플 데이터를 복사하고 주제 이름 아래의 편집 상자에 샘플 데이터를 붙여 넣습니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. MQTT 메시지를 전송하려면 주제 게시를 선택합니다.

device/+/data 구독에서 전송한 메시지가 표시되어야 합니다. 그러나 온도 값이 규칙 쿼리 문에서 최대 온도보다 낮기 때문에 문자 메시지를 받지 않아야 합니다.

올바른 동작이 표시되지 않으면 문제 해결 팁을 확인합니다.

SNS 메시지 규칙 문제 해결

예상한 결과가 표시되지 않는 경우를 대비하여 확인해야 할 몇 가지 사항이 있습니다.

- 오류 배너가 있음

  입력 메시지를 게시할 때 오류가 나타나면 먼저 해당 오류를 수정하세요. 다음 단계는 해당 오류를 수정하는 데 도움이 될 수 있습니다.
- MQTT 클라이언트에서 입력 메시지가 표시되지 않음

  `device/+/data` 주제 필터를 구독한 경우 입력 메시지를 `device/22/data` 주제에 게시할 때마다 절차에 설명된 대로 해당 메시지가 MQTT 클라이언트에 나타나야 합니다.

  확인해야 할 사항

  - 구독한 주제 필터 확인

    절차에 설명된 대로 입력 메시지 주제를 구독한 경우 게시할 때마다 입력 메시지의 복사본이 표시되어야 합니다.

    메시지가 표시되지 않으면 구독한 주제 이름을 확인하고 게시한 주제와 비교합니다. 주제 이름은 대소문자를 구분하며 구독한 주제는 메시지 페이로드를 게시한 주제와 동일해야 합니다.
  - 메시지 게시 함수 확인

    MQTT 클라이언트의 구독에서 device/+/data를 선택하고 게시 메시지의 주제를 확인한 다음 주제에 게시를 선택합니다. 주제 아래의 편집 상자에서 메시지 페이로드가 메시지 목록에 나타납니다.
- SMS 메시지가 수신되지 않음

  규칙이 작동하려면 메시지를 수신하고 SNS 알림을 전송할 수 있도록 권한을 부여하는 올바른 정책이 있어야 하며 메시지를 수신해야 합니다.

  확인해야 할 사항

  - MQTT 클라이언트 AWS 리전 의 및 생성한 규칙을 확인합니다.

    MQTT 클라이언트를 실행 중인 콘솔은 생성한 규칙과 동일한 AWS 리전에 있어야 합니다.
  - 메시지 페이로드의 온도 값이 테스트 임계값을 초과하는지 확인

    규칙 쿼리 문에 정의된 대로 온도 값이 30보다 작거나 같으면 규칙은 해당 작업을 수행하지 않습니다.
  - 규칙 쿼리 문의 입력 메시지 주제 확인

규칙이 작동하려면 규칙 쿼리 문의 FROM 절에 있는 주제 필터와 일치하는 주제 이름이 포함된 메시지를 받아야 합니다.

MQTT 클라이언트에 있는 주제의 철자와 함께 규칙 쿼리 문에서 주제 필터의 철자를 확인하세요. 주제 이름은 대/소문자를 구분하며 메시지의 주제는 규칙 쿼리 문의 주제 필터와 일치해야 합니다.

- 입력 메시지 페이로드의 내용 확인

  규칙이 작동하려면 SELECT 문에서 선언된 메시지 페이로드에서 데이터 필드를 찾아야 합니다.

  MQTT 클라이언트에 있는 메시지 페이로드의 철자와 함께 규칙 쿼리 명령문의 `temperature` 필드 철자를 확인하세요. 필드 이름은 대소문자를 구분하며 규칙 쿼리 문의 `temperature` 필드는 메시지 페이로드의 `temperature` 필드와 동일해야 합니다.

  메시지 페이로드의 JSON 문서의 형식이 올바른지 확인하세요. JSON에 쉼표 누락과 같은 오류가 있으면 규칙에서 읽을 수 없습니다.

- 규칙 동작에서 다시 게시된 메시지 주제 확인

  재게시 규칙 작업에서 새 메시지를 게시하는 주제는 MQTT 클라이언트에서 구독한 주제와 일치해야 합니다.

  콘솔에서 만든 규칙을 열고 규칙 작업이 메시지를 다시 게시할 주제를 확인합니다.

- 규칙에서 사용 중인 역할 확인

  규칙 작업에는 원래 주제를 받고 새 주제를 게시할 수 있는 권한이 있어야 합니다.

  메시지 데이터를 수신하고 다시 게시하도록 규칙을 인증하는 정책은 사용된 주제에 따라 다릅니다. 메시지 데이터를 다시 게시하는 데 사용되는 주제를 변경하는 경우 규칙 작업의 역할을 업데이트하여 현재 주제와 일치하도록 정책을 업데이트해야 합니다.

  문제가 의심되는 경우 규칙 재게시 작업을 편집하고 새 역할을 만듭니다. 규칙 작업에 의해 생성된 새 역할에는 이러한 작업을 수행하는 데 필요한 권한이 부여됩니다.

4단계: 결과 및 다음 단계 검토

이 자습서에서

- Amazon SNS 알림 주제 및 구독을 생성하고 테스트했습니다.
- 규칙 쿼리 문에서 간단한 SQL 쿼리 및 함수를 사용하여 알림에 대한 새 메시지를 만들었습니다.

- 사용자 지정 메시지 페이로드를 사용한 Amazon SNS 알림을 보내는 AWS IoT 규칙을 생성했습니다.
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트했습니다.

다음 단계

이 규칙을 사용하여 몇 가지 문자 메시지를 전송한 후 이 규칙을 실험하여 자습서의 일부 측면을 변경하면 메시지가 전송되는 시기에 영향을 받는지 확인하세요. 다음은 시작하는 데 도움이 될 몇 가지 아이디어입니다.

- 입력 메시지의 주제에서 *device_id*를 변경하고 문자 메시지 내용의 영향을 관찰합니다.
- 규칙 쿼리 문에서 선택한 필드를 변경하고 텍스트 메시지 내용의 영향을 관찰합니다.
- 최대 온도 대신 최소 온도에 대한 테스트 규칙 쿼리 문에서 테스트를 변경합니다. `max_temperature` 이름을 변경하는 것을 잊지 마세요!
- SNS 알림이 전송될 때 MQTT 메시지를 전송하기 위한 재게시 규칙 작업을 추가합니다.
- 이 시리즈의 다음 자습서를 사용해 보고 자습서: DynamoDB 테이블에 디바이스 데이터 저장의 방법을 알아봅니다.

자습서: DynamoDB 테이블에 디바이스 데이터 저장

이 자습서에서는 DynamoDB 테이블로 메시지 데이터를 보내는 AWS IoT 규칙을 생성하는 방법을 보여줍니다.

이 자습서에서는 가상 기상 센서 디바이스에서 DynamoDB 테이블로 메시지 데이터를 전송하는 규칙을 생성합니다. 이 규칙은 여러 기상 센서의 데이터를 단일 데이터베이스 테이블에 추가할 수 있도록 포맷합니다.

이 자습서에서 배울 내용

- DynamoDB 테이블 생성 방법
- AWS IoT 규칙에서 DynamoDB 테이블로 메시지 데이터를 보내는 방법
- AWS IoT 규칙에서 대체 템플릿을 사용하는 방법
- 규칙 쿼리 문에서 간단한 SQL 쿼리 및 함수를 사용하는 방법
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트하는 방법

이 자습서는 완료하는 데 약 30분 걸립니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 1단계: 이 자습서를 위한 DynamoDB 테이블 생성
- 2단계: DynamoDB 테이블로 데이터를 전송하는 AWS IoT 규칙 생성
- 3단계: AWS IoT 규칙 및 DynamoDB 테이블 테스트
- 4단계: 결과 및 다음 단계 검토

이 자습서를 시작하기 전에 다음 사항을 확인해야 합니다.

- 설정 AWS 계정

  이 자습서를 완료하려면 AWS 계정 및 AWS IoT 콘솔이 필요합니다.
- 검토된 MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기

  MQTT 클라이언트를 사용하여 주제를 구독하고 게시할 수 있는지 확인하세요. 이 절차에서는 MQTT 클라이언트를 사용하여 새 규칙을 테스트합니다.
- Amazon DynamoDB 개요를 검토했습니다.

  이전에 DynamoDB 를 사용하지 않았다면 DynamoDB의 기본 개념과 운영에 익숙해지도록 DynamoDB 시작하기를 검토하세요.

1단계: 이 자습서를 위한 DynamoDB 테이블 생성

이 자습서에서는 다음과 같은 속성이 포함된 DynamoDB 테이블을 만들어 가상 기상 센서 디바이스의 데이터를 기록합니다.

- `sample_time`은 기본 키이며 샘플이 기록된 시간을 설명합니다.
- `device_id`는 정렬 키이며 샘플을 제공한 디바이스를 설명합니다.
- `device_data`는 디바이스에서 수신되고 규칙 쿼리 문에 의해 형식이 지정된 데이터입니다.

이 자습서용 DynamoDB 테이블을 생성하려면

1. DynamoDB 콘솔을 열고 테이블 생성을 선택합니다.
2. 테이블 생성(Create table)에서

   a. 테이블 이름에 테이블 이름 **`wx_data`**를 입력합니다.

b. 파티션 키(Partition key)에 **sample_time**을 입력하고 필드 옆에 있는 옵션 목록에서 **Number**를 선택합니다.

c. 정렬 키(Sort key)에 **device_id**를 입력하고 필드 옆에 있는 옵션 목록에서 **Number**를 선택합니다.

d. 페이지 하단에서 생성(Create)을 선택합니다.

나중에 DynamoDB 규칙 작업을 구성할 때 `device_data`를 정의합니다.

2단계: DynamoDB 테이블로 데이터를 전송하는 AWS IoT 규칙 생성

이 단계에서는 규칙 쿼리 문을 사용하여 가상 기상 센서 디바이스의 데이터 형식을 데이터베이스 테이블에 기록하도록 지정합니다.

기상 센서 디바이스에서 받은 샘플 메시지 페이로드는 다음과 같습니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

데이터베이스 항목의 경우 규칙 쿼리 문을 사용하여 메시지 페이로드의 구조를 다음과 같이 병합합니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind_velocity": 22,
  "wind_bearing": 255
}
```

이 규칙에서는 몇 가지 대체 템플릿을(를) 사용합니다. 대체 템플릿은 함수 및 메시지 데이터에서 동적 값을 삽입할 수 있는 표현식입니다.

DynamoDB 테이블로 데이터를 전송하는 AWS IoT 규칙을 생성하려면

1. AWS IoT 콘솔의 규칙 허브를 엽니다. 또는 내에서 AWS IoT AWS Management Console 홈페이지를 열고 메시지 라우팅>규칙으로 이동할 수 있습니다.
2. 규칙(Rules)에서 새 규칙 생성을 시작하려면 규칙 생성(Create rule)을 선택합니다.
3. 규칙 속성(Rule properties)에서

   a. 규칙 이름(Rule name)에 **wx_data_ddb**를 입력합니다.

      규칙 이름은 AWS 계정 및 리전 내에서 고유해야 하며 공백을 가질 수 없습니다. 이 이름에 밑줄 문자를 사용하여 규칙 이름의 두 단어를 구분했습니다.

   b. 규칙 설명(Rule description)에서 규칙을 설명합니다.

      의미 있는 설명을 사용하면 이 규칙이 수행하는 작업과 규칙을 만든 이유를 쉽게 기억할 수 있습니다. 설명은 필요한 만큼 길어질 수 있으므로 가능한 한 자세하게 설명하세요.
4. 다음을 선택하여 계속 진행합니다.
5. SQL 문(SQL statement)에서

   a. SQL 버전(SQL version)에서 **2016-03-23**을 선택합니다.

   b. SQL 문(SQL statement) 편집 상자에 문을 입력합니다.

   ```
   SELECT temperature, humidity, barometer,
     wind.velocity as wind_velocity,
     wind.bearing as wind_bearing,
   FROM 'device/+/data'
   ```

   이 문은 다음을 수행합니다.

   - `device/+/data` 주제 필터와 일치하는 주제가 있는 MQTT 메시지를 수신합니다.
   - `wind` 속성의 요소를 개별 속성으로 형식화합니다.
   - `temperature`, `humidity` 및 `barometer` 속성은 변경되지 않은 상태로 전달합니다.
6. 다음을 선택하여 계속 진행합니다.
7. 규칙 작업(Rule actions)에서

   a. 이 규칙에 대한 규칙 작업 목록을 열려면 작업 1(Action 1)에서 **DynamoDB**를 선택합니다.

> **Note**
>
> 규칙 작업으로 DynamoDBv2가 아니라 DynamoDB를 선택해야 합니다.

b. 테이블 이름에서 이전 단계에서 생성한 DynamoDB 테이블의 이름 **wx_data**를 선택합니다.

   파티션 키 유형(Partition key type) 및 정렬 키 유형(Sort key type) 필드는 DynamoDB 테이블의 값으로 채워집니다.

c. 파티션 키에서 **sample_time**를 입력합니다.

d. 파티션 키 값에 **${timestamp()}**를 입력합니다.

   이것은 이 규칙에서 사용할 첫 번째 대체 템플릿입니다. 메시지 페이로드의 값을 사용하는 대신 timestamp 함수에서 반환된 값을 사용합니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 timestamp를 참조하세요.

e. 정렬 키(Sort key)에 **device_id**를 입력합니다.

f. 정렬 키 값에 **${cast(topic(2) AS DECIMAL)}**를 입력합니다.

   이것은 이 규칙에서 사용할 두 번째 대체 템플릿입니다. 키의 숫자 형식과 일치하도록 DECIMAL 값으로 캐스트한 후 디바이스 ID인 주제 이름에 두 번째 요소의 값을 삽입합니다. 주제에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 topic을 참조하세요. 캐스팅에 대한 자세한 내용은 AWS IoT Core 개발자 안내서의 cast를 참조하세요.

g. 이 열에 메시지 데이터 쓰기에 **device_data**를 입력합니다.

   그러면 DynamoDB 테이블에 `device_data` 열이 생성됩니다.

h. 작업은 비워 둡니다.

i. IAM role(IAM 역할)에서 Create new role(새 역할 생성)을 선택합니다.

j. Create role(역할 생성) 대화 상자에서 Role name(역할 이름)에 wx_ddb_role을 입력합니다. 이 새 역할에는 'aws-iot-rule'이라는 접두사가 붙은 정책이 자동으로 포함되며, 이 정책을 통해 **wx_data_ddb** 규칙이 사용자가 생성한 **wx_data** DynamoDB 테이블로 데이터를 전송할 수 있습니다.

k. IAM 역할(IAM role)에서 **wx_ddb_role**을 선택합니다.

l. 페이지 하단에서 다음(Next)을 선택합니다.

8. 검토 및 생성(Review and create) 페이지 하단에서 생성(Create)을 선택하여 규칙을 생성합니다.

3단계: AWS IoT 규칙 및 DynamoDB 테이블 테스트

새 규칙을 테스트하려면 MQTT 클라이언트를 사용하여 이 테스트에 사용된 MQTT 메시지를 게시하고 구독합니다.

새 창에서 AWS IoT 콘솔의 MQTT 클라이언트를 엽니다. 이렇게 하면 MQTT 클라이언트의 구성을 그대로 유지하면서 규칙을 편집할 수 있습니다. MQTT 클라이언트는 콘솔의 다른 페이지로 이동하기 위해 나가면 구독 또는 메시지 로그를 유지하지 않습니다. 또한 규칙이 보내는 새 항목을 보려면 콘솔의 DynamoDB Tables 허브에 별도의 AWS IoT 콘솔 창을 열어야 합니다.

MQTT 클라이언트를 사용하여 규칙을 테스트하려면

1. AWS IoT 콘솔의 MQTT 클라이언트에서 입력 주제(`device/+/data`)를 구독합니다.
   a. MQTT 클라이언트에서 주제 구독(Subscribe to a topic)을 선택합니다.
   b. 주제 필터에 입력 주제 필터의 주제(**`device/+/data`**)를 입력합니다.
   c. 구독을 선택합니다.
2. 이제 특정 디바이스 ID **`device/22/data`**로 입력 주제에 메시지를 게시합니다. 와일드카드 문자가 포함된 MQTT 항목에는 게시할 수 없습니다.
   a. MQTT 클라이언트에서 주제에 게시(Publish to a topic)를 선택합니다.
   b. 주제 이름(Topic name)에 주제 이름(**`device/22/data`**)을 입력합니다.
   c. 메시지 페이로드에 다음 샘플 데이터를 입력합니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. MQTT 메시지를 게시하려면 게시를 선택합니다.
   e. 이제 MQTT 클라이언트에서 주제 구독(Subscribe to a topic)을 선택합니다. 구독 열에서 **`device/+/data`** 구독을 선택합니다. 이전 단계의 샘플 데이터가 나타나는지 확인합니다.
3. 규칙에서 생성한 DynamoDB 테이블의 행을 보려면 선택합니다.

a. AWS IoT 콘솔의 DynamoDB 테이블 허브에서 wx_data를 선택한 다음 항목 탭을 선택합니다.

   항목(Items) 탭이 이미 사용 중인 경우 테이블 헤더의 오른쪽 상단 모서리에서 새로 고침 아이콘을 선택하여 디스플레이를 새로 고쳐야 할 수 있습니다.

b. 테이블의 sample_time 값은 링크이며 하나를 엽니다. 방금 첫 번째 메시지를 전송한 경우 이는 목록에 있는 유일한 메시지입니다.

   이 링크는 테이블의 해당 행에 있는 모든 데이터를 표시합니다.

c. device_data 항목을 확장하여 규칙 쿼리 문에서 생성된 데이터를 확인합니다.

d. 이 디스플레이에서 사용할 수 있는 데이터의 다양한 표현을 살펴보세요. 이 디스플레이에서 데이터를 편집할 수도 있습니다.

e. 이 데이터 행 검토를 마친 후 변경 사항을 저장하려면 저장을 선택하고 변경 사항을 저장하지 않고 종료하려면 취소를 선택합니다.

올바른 동작이 표시되지 않으면 문제 해결 팁을 확인합니다.

DynamoDB 규칙 문제 해결

다음은 예상한 결과가 표시되지 않는 경우에 대비하여 확인해야 할 몇 가지 사항입니다.

- 오류 배너가 있음

  입력 메시지를 게시할 때 오류가 나타나면 먼저 해당 오류를 수정하세요. 다음 단계는 해당 오류를 수정하는 데 도움이 될 수 있습니다.

- MQTT 클라이언트에서 입력 메시지가 표시되지 않음

  `device/+/data` 주제 필터를 구독한 경우 입력 메시지를 `device/22/data` 주제에 게시할 때마다 절차에 설명된 대로 해당 메시지가 MQTT 클라이언트에 나타나야 합니다.

  확인해야 할 사항

  - 구독한 주제 필터 확인

    절차에 설명된 대로 입력 메시지 주제를 구독한 경우 게시할 때마다 입력 메시지의 복사본이 표시되어야 합니다.

    메시지가 표시되지 않으면 구독한 주제 이름을 확인하고 게시한 주제와 비교합니다. 주제 이름은 대소문자를 구분하며 구독한 주제는 메시지 페이로드를 게시한 주제와 동일해야 합니다.

- 메시지 게시 함수 확인

  MQTT 클라이언트의 구독에서 device/+/data를 선택하고 게시 메시지의 주제를 확인한 다음 주제에 게시를 선택합니다. 주제 아래의 편집 상자에서 메시지 페이로드가 메시지 목록에 나타납니다.

- DynamoDB 테이블에 데이터가 표시되지 않음

  가장 먼저 할 일은 테이블 헤더의 오른쪽 상단 모서리에 있는 새로 고침 아이콘을 선택하여 디스플레이를 새로 고치는 것입니다. 찾고있는 데이터가 표시되지 않으면 다음을 확인하세요.

  확인해야 할 사항

  - MQTT 클라이언트 AWS 리전 의 및 생성한 규칙을 확인합니다.

    MQTT 클라이언트를 실행 중인 콘솔은 생성한 규칙과 동일한 AWS 리전에 있어야 합니다.

  - 규칙 쿼리 문의 입력 메시지 주제 확인

    규칙이 작동하려면 규칙 쿼리 문의 FROM 절에 있는 주제 필터와 일치하는 주제 이름이 포함된 메시지를 받아야 합니다.

    MQTT 클라이언트에 있는 주제의 철자와 함께 규칙 쿼리 문에서 주제 필터의 철자를 확인하세요. 주제 이름은 대/소문자를 구분하며 메시지의 주제는 규칙 쿼리 문의 주제 필터와 일치해야 합니다.

  - 입력 메시지 페이로드의 내용 확인

    규칙이 작동하려면 SELECT 문에서 선언된 메시지 페이로드에서 데이터 필드를 찾아야 합니다.

    MQTT 클라이언트에 있는 메시지 페이로드의 철자와 함께 규칙 쿼리 명령문의 `temperature` 필드 철자를 확인하세요. 필드 이름은 대소문자를 구분하며 규칙 쿼리 문의 `temperature` 필드는 메시지 페이로드의 `temperature` 필드와 동일해야 합니다.

    메시지 페이로드의 JSON 문서의 형식이 올바른지 확인하세요. JSON에 쉼표 누락과 같은 오류가 있으면 규칙에서 읽을 수 없습니다.

  - 규칙 작업에 사용된 키 및 필드 이름 확인

    주제 규칙에 사용되는 필드 이름은 게시된 메시지의 JSON 메시지 페이로드에 있는 필드 이름과 일치해야 합니다.

    콘솔에서 생성한 규칙을 열고 MQTT 클라이언트에서 사용된 규칙 작업 구성의 필드 이름을 확인합니다.

  - 규칙에서 사용 중인 역할 확인

규칙 작업에는 원래 주제를 받고 새 주제를 게시할 수 있는 권한이 있어야 합니다.

메시지 데이터를 수신하고 DynamoDB 테이블을 업데이트하도록 규칙을 인증하는 정책은 사용된 주제에 따라 다릅니다. 규칙에 사용되는 주제 또는 DynamoDB 테이블 이름을 변경하는 경우 규칙 작업의 역할을 업데이트하여 해당 정책을 일치하도록 업데이트해야 합니다.

문제가 의심되는 경우 규칙 작업을 편집하고 새 역할을 만듭니다. 규칙 작업에 의해 생성된 새 역할에는 이러한 작업을 수행하는 데 필요한 권한이 부여됩니다.

4단계: 결과 및 다음 단계 검토

이 규칙을 사용하여 DynamoDB 테이블로 몇 개의 메시지를 전송한 후 이를 실험하여 자습서의 일부 측면을 변경하면 테이블에 기록된 데이터에 어떤 영향을 주는지 확인하세요. 다음은 시작하는 데 도움이 될 몇 가지 아이디어입니다.

- 입력 메시지의 주제에서 *device_id*를 변경하고 데이터의 영향을 관찰합니다. 이 기능을 사용하여 여러 기상 센서의 수신 데이터를 시뮬레이션할 수 있습니다.
- 규칙 쿼리 문에서 선택한 필드를 변경하고 데이터의 영향을 관찰합니다. 이 옵션을 사용하여 테이블에 저장된 데이터를 필터링할 수 있습니다.
- 테이블에 추가된 각 행에 대해 MQTT 메시지를 전송하려면 재게시 규칙 작업을 추가합니다. 디버깅에 사용할 수 있습니다.

이 자습서를 완료했으면 the section called “ AWS Lambda 함수를 사용하여 알림 형식 지정” 섹션을 확인합니다.

자습서: AWS Lambda 함수를 사용하여 알림 형식 지정

이 자습서에서는 형식 지정 및 다른 AWS 서비스로 전송을 위해 AWS Lambda 작업에 MQTT 메시지 데이터를 전송하는 방법을 보여줍니다. 이 자습서에서 AWS Lambda 작업은 AWS SDK를 사용하여 에 대한 자습서에서 생성한 Amazon SNS 주제로 형식이 지정된 메시지를 보냅니다the section called “Amazon SNS 알림 전송”.

the section called “Amazon SNS 알림 전송” 수행 방법에 대한 자습서에서는 규칙의 쿼리 문에서 발생한 JSON 문서가 문자 메시지의 본문으로 전송되었습니다. 결과는 다음 예제와 같은 문자 메시지입니다.

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

이 자습서에서는 AWS Lambda 규칙 작업을 사용하여 규칙 쿼리 문의 데이터를 다음과 같은 친숙한 형식으로 포맷하는 AWS Lambda 함수를 호출합니다.

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

이 자습서에서 생성할 AWS Lambda 함수는 규칙 쿼리 문의 데이터를 사용하여 메시지 문자열의 형식을 지정하고 AWS SDK의 SNS 게시 함수를 호출하여 알림을 생성합니다.

이 자습서에서 배울 내용

- AWS Lambda 함수를 생성하고 테스트하는 방법
- AWS Lambda 함수에서 AWS SDK를 사용하여 Amazon SNS 알림을 게시하는 방법
- 규칙 쿼리 문에서 간단한 SQL 쿼리 및 함수를 사용하는 방법
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트하는 방법

이 자습서는 완료하는 데 약 45분이 소요됩니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 1단계: 텍스트 메시지를 보내는 AWS Lambda 함수 생성
- 2단계: AWS IoT 규칙 작업을 사용하여 AWS Lambda 규칙 생성
- 3단계: AWS IoT 규칙 및 AWS Lambda 규칙 작업 테스트
- 4단계: 결과 및 다음 단계 검토

이 자습서를 시작하기 전에 다음 사항을 확인해야 합니다.

- 설정 AWS 계정

  이 자습서를 완료하려면 AWS 계정 및 AWS IoT 콘솔이 필요합니다.
- 검토된 MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기

  MQTT 클라이언트를 사용하여 주제를 구독하고 게시할 수 있는지 확인하세요. 이 절차에서는 MQTT 클라이언트를 사용하여 새 규칙을 테스트합니다.
- 이 섹션의 다른 규칙 자습서를 완료했습니다.

  이 자습서에는 the section called “Amazon SNS 알림 전송” 수행 방법에 대한 자습서에서 생성한 SNS 알림 주제가 필요합니다. 또한 이 섹션의 다른 규칙 관련 자습서를 완료했다고 가정합니다.

- AWS Lambda 개요를 검토했습니다.

  AWS Lambda 이전에를 사용한 적이 없는 경우 AWS Lambda 및 Lambda 시작하기를 검토하여 해당 용어와 개념을 알아봅니다.

1단계: 텍스트 메시지를 보내는 AWS Lambda 함수 생성

이 자습서의 AWS Lambda 함수는 규칙 쿼리 문의 결과를 수신하고 요소를 텍스트 문자열에 삽입한 다음 결과 문자열을 Amazon SNS에 알림의 메시지로 보냅니다.

규칙 작업을 사용하여 AWS IoT 알림을 보내는 방법에 대한 자습서the section called “Amazon SNS 알림 전송”와 달리이 자습서에서는 AWS SDK의 함수를 사용하여 Lambda 함수에서 알림을 보냅니다. 그러나 이 자습서에서 사용된 실제 Amazon SNS 알림 주제는 the section called “Amazon SNS 알림 전송” 수행 방법에 대한 자습서에서 사용한 것과 동일합니다.

텍스트 메시지를 보내는 AWS Lambda 함수를 생성하려면

1. 새 AWS Lambda 함수를 생성합니다.

   a. AWS Lambda 콘솔에서 함수 생성을 선택합니다.
   b. 함수 생성(Create function)에서 블루프린트 사용(Use a blueprint)을 선택합니다.

      `hello-world-python` 블루프린트를 검색하여 선택한 다음 구성을 선택합니다.
   c. 기본 정보(Basic information)에서:

      i. Function name(함수 이름)에 이 함수 이름(**`format-high-temp-notification`**)을 입력합니다.
      ii. 실행 역할에서 AWS 정책 템플릿에서 새 역할 생성을 선택합니다.
      iii. 역할 이름에 새 역할의 이름(**`format-high-temp-notification-role`**)을 입력합니다.
      iv. 정책 템플릿 - 선택 사항에서 Amazon SNS 게시 정책을 검색하고 선택합니다.
      v. 함수 생성을 선택합니다.
2. 블루프린트 코드를 수정하여 Amazon SNS 알림의 형식을 지정하고 전송합니다.

   a. 함수를 생성한 후에는 format-high-temp-notification 세부 정보 페이지를 참조하세요. 그렇지 않은 경우 Lambda 함수 페이지에서 엽니다.
   b. format-high-temp-notification 세부 정보 페이지에서 구성(Configuration) 탭을 선택하고 함수 코드(Function code) 패널로 스크롤합니다.

c. 함수 코드(Function code) 창의 환경(Environment) 창에서 Python 파일 (`lambda_function.py`)을 선택합니다.

d. 함수 코드(Function code) 창에서 블루프린트의 원본 프로그램 코드를 모두 삭제하고 이 코드로 교체합니다.다.

```
import boto3
#
#   expects event parameter to contain:
#   {
#       "device_id": "32",
#       "reported_temperature": 38,
#       "max_temperature": 30,
#       "notify_topic_arn": "arn:aws:sns:us-
east-1:57EXAMPLE833:high_temp_notice"
#   }
#
#   sends a plain text string to be used in a text message
#
#      "Device {0} reports a temperature of {1}, which exceeds the limit of
 {2}."
#
#   where:
#       {0} is the device_id value
#       {1} is the reported_temperature value
#       {2} is the max_temperature value
#
def lambda_handler(event, context):

    # Create an SNS client to send notification
    sns = boto3.client('sns')

    # Format text message from data
    message_text = "Device {0} reports a temperature of {1}, which exceeds the
 limit of {2}.".format(
            str(event['device_id']),
            str(event['reported_temperature']),
            str(event['max_temperature'])
        )

    # Publish the formatted message
    response = sns.publish(
            TopicArn = event['notify_topic_arn'],
```

```
            Message = message_text
        )

    return response
```

e. 배포(Deploy)를 선택합니다.

3. 새 창에서 the section called “Amazon SNS 알림 전송” 수행 방법에 대한 자습서에서 Amazon SNS 주제의 Amazon 리소스 이름(ARN)을 조회합니다.

   a. 새 창에서 Amazon SNS 콘솔의 주제 페이지를 엽니다.

   b. 주제 페이지에서 Amazon SNS 주제 목록에서 high_temp_notice 알림 주제를 찾습니다.

   c. 다음 단계에서 사용할 high_temp_notice 알림 주제의 ARN을 찾습니다.

4. Lambda 함수에 대한 테스트 케이스 생성

   a. 콘솔의 Lambda 함수 페이지에서 format-high-temp-notification 세부 정보 페이지 오른쪽 상단 모서리에 있는 테스트 이벤트 선택(Select a test event)(비활성화된 것처럼 보이더라도)을 선택한 다음 테스트 이벤트 구성(Configure test events)을 선택합니다.

   b. 테스트 이벤트 구성(Configure test event)에서 새 테스트 이벤트 구성(Create new test event)을 선택합니다.

   c. 이벤트 이름에 **SampleRuleOutput**을 입력합니다.

   d. JSON 편집기에서 이벤트 이름 아래에 이 샘플 JSON 문서를 붙여 넣습니다. 다음은 AWS IoT 규칙이 Lambda 함수로 전송할 항목의 예입니다.

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

   e. high_temp_notice 알림 주제의 ARN이 있는 창을 참조하여 ARN 값을 복사합니다.

   f. JSON 편집기의 `notify_topic_arn` 값을 알림 주제의 ARN으로 바꿉니다.

      AWS IoT 규칙을 생성할 때 이 ARN 값을 다시 사용할 수 있도록 이 창을 열어 두세요.

   g. 생성(Create)을 선택합니다.

5. 샘플 데이터로 함수를 테스트합니다.

a. format-high-temp-notification 세부 정보 페이지의 오른쪽 상단 모서리에서 SampleRuleOutput이 테스트(Test) 버튼 옆에 나타나는지 확인합니다. 그렇지 않은 경우 사용 가능한 테스트 이벤트 목록에서 선택합니다.

b. 함수에 샘플 규칙 출력 메시지를 전송하려면 테스트(Test)를 선택합니다.

함수와 알림이 모두 작동하면 알림을 구독한 휴대 전화에서 문자 메시지를 받습니다.

휴대 전화에서 문자 메시지를 받지 못한 경우 작업 결과를 확인하세요. 함수 코드 패널의 실행 결과 탭에서 응답을 검토하여 발생한 오류를 찾습니다. 함수가 휴대폰으로 알림을 전송할 때까지 다음 단계로 이행하지 마세요.

2단계: AWS IoT 규칙 작업을 사용하여 AWS Lambda 규칙 생성

이 단계에서는 규칙 쿼리 문을 사용하여 가상 기상 센서 디바이스의 데이터 형식을 Lambda 함수로 전송하여 문자 메시지를 형식 지정하고 전송합니다.

기상 디바이스에서 받은 샘플 메시지 페이로드는 다음과 같습니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

이 규칙에서는 규칙 쿼리 문을 사용하여 다음과 같은 Lambda 함수에 대한 메시지 페이로드를 만듭니다.

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

여기에는 Lambda 함수가 올바른 문자 메시지로 형식을 지정하고 전송하는 데 필요한 모든 정보가 들어 있습니다.

Lambda 함수를 호출하는 AWS IoT 규칙을 생성하려면

1. AWS IoT 콘솔의 규칙 허브를 엽니다.
2. 규칙에서 새 규칙 생성을 시작하려면 생성을 선택합니다.
3. 규칙 생성의 상단 부분에서:

   a. 이름에 규칙 이름 **wx_friendly_text**를 입력합니다.

      규칙 이름은 AWS 계정 및 리전 내에서 고유해야 하며 공백을 가질 수 없습니다. 이 이름에 밑줄 문자를 사용하여 규칙 이름의 두 단어를 구분했습니다.

   b. 설명에서 규칙을 설명합니다.

      의미 있는 설명을 사용하면 이 규칙이 수행하는 작업과 규칙을 만든 이유를 쉽게 기억할 수 있습니다. 설명은 필요한 만큼 길어질 수 있으므로 가능한 한 자세하게 설명하세요.

4. 규칙 생성의 규칙 쿼리 문에서:

   a. SQL 버전 사용에서 **2016-03-23**을(를) 선택합니다.

   b. 규칙 쿼리 문 편집 상자에 쿼리 문을 입력합니다.

```
SELECT
  cast(topic(2) AS DECIMAL) as device_id,
  temperature as reported_temperature,
  30 as max_temperature,
  'arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice' as notify_topic_arn
FROM 'device/+/data' WHERE temperature > 30
```

      이 문은 다음을 수행합니다.

      - device/+/data 주제 필터와 일치하고 temperature 값이 30보다 큰 주제가 있는 MQTT 메시지를 수신 대기합니다.
      - 주제 문자열에서 두 번째 요소를 선택하고 10진수로 변환한 다음 device_id 필드에 할당합니다.
      - 메시지 페이로드에서 temperature 값 필드를 선택하여 reported_temperature 필드에 할당합니다.
      - 한계 값을 나타내는 상수 값 30을 생성하고 max_temperature 필드에 할당합니다.
      - notify_topic_arn 필드에 대한 상수 값을 만듭니다.

   c. high_temp_notice 알림 주제의 ARN이 있는 창을 참조하여 ARN 값을 복사합니다.

d. 규칙 쿼리 문 편집기의 ARN 값(*arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice*)을 알림 주제의 ARN으로 바꿉니다.

5. 하나 이상의 작업 설정에서:

a. 이 규칙에 대한 규칙 작업 목록을 열려면 작업 추가를 선택합니다.

b. 작업 선택(Select an action)에서 Lambda 함수로 메시지 전송하기(Send a message to a Lambda function)를 선택합니다.

c. 작업 목록 하단에서 선택한 작업의 구성 페이지를 열려면 구성 작업을 선택합니다.

6. 구성 작업에서:

a. 함수 이름에서 선택(Select)을 클릭합니다.

b. format-high-temp-notification을 선택합니다.

c. 작업 구성(Configure action) 하단에서 작업 추가(Add action)를 선택합니다.

d. 규칙을 생성하려면 규칙 생성(Create a rule) 하단에서 규칙 생성(Create rule)을 선택합니다.

3단계: AWS IoT 규칙 및 AWS Lambda 규칙 작업 테스트

새 규칙을 테스트하려면 MQTT 클라이언트를 사용하여 이 규칙에서 사용하는 MQTT 메시지를 게시하고 구독합니다.

새 창에서 AWS IoT 콘솔의 MQTT 클라이언트를 엽니다. 이제 MQTT 클라이언트의 구성을 그대로 유지하면서 규칙을 편집할 수 있습니다. MQTT 클라이언트를 떠나 콘솔의 다른 페이지로 이동하면 구독 또는 메시지 로그가 손실됩니다.

MQTT 클라이언트를 사용하여 규칙을 테스트하려면

1. AWS IoT 콘솔의 MQTT 클라이언트에서 입력 주제(이 경우 `device/+/data`)를 구독합니다.

a. MQTT 클라이언트에서 구독 아래에서 주제 구독을 선택합니다.

b. 구독 주제에 입력 주제 필터 **`device/+/data`**의 주제를 입력합니다.

c. 나머지 필드는 기본 설정을 유지합니다.

d. 주제 구독을 선택합니다.

구독 열의 주제 게시 아래에 **`device/+/data`**가 나타납니다.

2. 특정 디바이스 ID **`device/32/data`**로 입력 주제에 메시지를 게시합니다. 와일드카드 문자가 포함된 MQTT 항목에는 게시할 수 없습니다.

a. MQTT 클라이언트의 구독 아래에서 주제 게시를 선택합니다.

b. 게시 필드에 입력 주제 이름 **device/32/data**를 입력합니다.

c. 여기에 표시된 샘플 데이터를 복사하고 주제 이름 아래의 편집 상자에 샘플 데이터를 붙여 넣습니다.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. MQTT 메시지를 게시하려면 주제에 게시(Publish to topic)를 선택합니다.

3. 문자 메시지가 전송되었는지 확인합니다.

a. MQTT 클라이언트의 구독 아래에서 이전에 구독한 주제 옆에 녹색 점이 있습니다.

녹색 점은 마지막으로 메시지를 보았을 때 하나 이상의 새 메시지가 수신되었음을 나타냅니다.

b. 구독 아래에서 device/+/data를 선택하여 메시지 페이로드가 방금 게시한 것과 일치하고 다음과 같이 표시되는지 확인하세요.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. SNS 주제를 구독하는 데 사용한 전화를 확인하고 메시지 페이로드의 내용이 다음과 같이 표시되는지 확인합니다.

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

메시지 주제에서 주제 ID 요소를 변경하는 경우 메시지 주제의 해당 요소에 숫자만 포함된 경우에만 `topic(2)` 값을 숫자 값으로 캐스팅할 수 있다는 점을 기억하세요.

4. 온도가 한계를 초과하지 않는 MQTT 메시지를 전송해보세요.

   a. MQTT 클라이언트의 구독 아래에서 주제 게시를 선택합니다.

   b. 게시 필드에 입력 주제 이름 **device/33/data**를 입력합니다.

   c. 여기에 표시된 샘플 데이터를 복사하고 주제 이름 아래의 편집 상자에 샘플 데이터를 붙여 넣습니다.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. MQTT 메시지를 전송하려면 주제 게시를 선택합니다.

   device/+/data 구독에서 전송한 메시지가 표시되어야 하지만 온도 값이 규칙 쿼리 문에서 최대 온도보다 낮기 때문에 문자 메시지를 받지 않아야 합니다.

   올바른 동작이 표시되지 않으면 문제 해결 팁을 확인합니다.

AWS Lambda 규칙 및 알림 문제 해결

예상한 결과가 표시되지 않는 경우를 대비하여 확인해야 할 몇 가지 사항이 있습니다.

- 오류 배너가 있음

  입력 메시지를 게시할 때 오류가 나타나면 먼저 해당 오류를 수정하세요. 다음 단계는 해당 오류를 수정하는 데 도움이 될 수 있습니다.

- MQTT 클라이언트에서 입력 메시지가 표시되지 않음

  `device/+/data` 주제 필터를 구독한 경우 입력 메시지를 `device/32/data` 주제에 게시할 때마다 절차에 설명된 대로 해당 메시지가 MQTT 클라이언트에 나타나야 합니다.

확인해야 할 사항

- 구독한 주제 필터 확인

  절차에 설명된 대로 입력 메시지 주제를 구독한 경우 게시할 때마다 입력 메시지의 복사본이 표시되어야 합니다.

  메시지가 표시되지 않으면 구독한 주제 이름을 확인하고 게시한 주제와 비교합니다. 주제 이름은 대소문자를 구분하며 구독한 주제는 메시지 페이로드를 게시한 주제와 동일해야 합니다.

- 메시지 게시 함수 확인

  MQTT 클라이언트의 구독에서 device/+/data를 선택하고 게시 메시지의 주제를 확인한 다음 주제에 게시를 선택합니다. 주제 아래의 편집 상자에서 메시지 페이로드가 메시지 목록에 나타납니다.

- SMS 메시지가 수신되지 않음

  규칙이 작동하려면 메시지를 수신하고 SNS 알림을 전송할 수 있도록 권한을 부여하는 올바른 정책이 있어야 하며 메시지를 수신해야 합니다.

  확인해야 할 사항

  - MQTT 클라이언트 AWS 리전 의 및 생성한 규칙을 확인합니다.

    MQTT 클라이언트를 실행 중인 콘솔은 생성한 규칙과 동일한 AWS 리전에 있어야 합니다.

  - 메시지 페이로드의 온도 값이 테스트 임계값을 초과하는지 확인

    규칙 쿼리 문에 정의된 대로 온도 값이 30보다 작거나 같으면 규칙은 해당 작업을 수행하지 않습니다.

  - 규칙 쿼리 문의 입력 메시지 주제 확인

    규칙이 작동하려면 규칙 쿼리 문의 FROM 절에 있는 주제 필터와 일치하는 주제 이름이 포함된 메시지를 받아야 합니다.

    MQTT 클라이언트에 있는 주제의 철자와 함께 규칙 쿼리 문에서 주제 필터의 철자를 확인하세요. 주제 이름은 대/소문자를 구분하며 메시지의 주제는 규칙 쿼리 문의 주제 필터와 일치해야 합니다.

  - 입력 메시지 페이로드의 내용 확인

    규칙이 작동하려면 SELECT 문에서 선언된 메시지 페이로드에서 데이터 필드를 찾아야 합니다.

MQTT 클라이언트에 있는 메시지 페이로드의 철자와 함께 규칙 쿼리 명령문의 `temperature` 필드 철자를 확인하세요. 필드 이름은 대소문자를 구분하며 규칙 쿼리 문의 `temperature` 필드는 메시지 페이로드의 `temperature` 필드와 동일해야 합니다.

메시지 페이로드의 JSON 문서의 형식이 올바른지 확인하세요. JSON에 쉼표 누락과 같은 오류가 있으면 규칙에서 읽을 수 없습니다.

- Amazon SNS 알림 확인

  1단계: SMS 문자 메시지를 전송하는 Amazon SNS 주제 생성에서 Amazon SNS 알림을 테스트하고 알림을 테스트하여 알림이 작동하는지 확인하는 방법을 설명하는 3단계를 참조하세요.

- Lambda 함수 확인

  1단계: 텍스트 메시지를 보내는 AWS Lambda 함수 생성에서 테스트 데이터를 사용하여 Lambda 함수를 테스트하고 Lambda 함수를 테스트하는 방법을 설명하는 5단계를 참조하세요.

- 규칙에서 사용 중인 역할 확인

  규칙 작업에는 원래 주제를 받고 새 주제를 게시할 수 있는 권한이 있어야 합니다.

  메시지 데이터를 수신하고 다시 게시하도록 규칙을 인증하는 정책은 사용된 주제에 따라 다릅니다. 메시지 데이터를 다시 게시하는 데 사용되는 주제를 변경하는 경우 규칙 작업의 역할을 업데이트하여 현재 주제와 일치하도록 정책을 업데이트해야 합니다.

  문제가 의심되는 경우 규칙 재게시 작업을 편집하고 새 역할을 만듭니다. 규칙 작업에 의해 생성된 새 역할에는 이러한 작업을 수행하는 데 필요한 권한이 부여됩니다.

4단계: 결과 및 다음 단계 검토

이 자습서에서는:

- 사용자 지정 메시지 페이로드를 사용한 Amazon SNS 알림을 보낸 Lambda 함수를 호출하는 AWS IoT 규칙을 생성했습니다.
- 규칙 쿼리 문에서 간단한 SQL 쿼리와 함수를 사용하여 Lambda 함수에 대한 새 메시지 페이로드를 만들었습니다.
- MQTT 클라이언트를 사용하여 AWS IoT 규칙을 테스트했습니다.

다음 단계

이 규칙을 사용하여 몇 가지 문자 메시지를 전송한 후 이 규칙을 실험하여 자습서의 일부 측면을 변경하면 메시지가 전송되는 시기에 영향을 받는지 확인하세요. 다음은 시작하는 데 도움이 될 몇 가지 아이디어입니다.

- 입력 메시지의 주제에서 *device_id*를 변경하고 문자 메시지 내용의 영향을 관찰합니다.
- 규칙 쿼리 문에서 선택한 필드를 변경하고, 새 메시지에서 사용하도록 Lambda 함수를 업데이트하고, 문자 메시지 내용의 영향을 관찰합니다.
- 최대 온도 대신 최소 온도에 대한 테스트 규칙 쿼리 문에서 테스트를 변경합니다. Lambda 함수를 업데이트하여 새 메시지 형식을 지정하고 `max_temperature` 이름을 변경하는 것을 잊지 마세요.
- AWS IoT 규칙을 개발하고 사용하는 동안 발생할 수 있는 오류를 찾는 방법에 대한 자세한 내용은 섹션을 참조하세요모니터링 AWS IoT.

디바이스 섀도우로 디바이스가 오프라인 상태일 때 디바이스 상태 유지

이 자습서에서는 AWS IoT 디바이스 섀도우 서비스를 사용하여 디바이스의 상태 정보를 저장하고 업데이트하는 방법을 보여줍니다. JSON 문서인 섀도우 문서는 디바이스, 로컬 앱 또는 서비스에서 게시한 메시지를 기반으로 디바이스 상태의 변화를 보여줍니다. 이 자습서에서 섀도우 문서는 전구의 색상 변화를 보여줍니다. 이 자습서에서는 디바이스가 인터넷에서 연결이 끊어진 경우에도 섀도우가 이 정보를 저장하는 방법과 디바이스가 다시 온라인 상태가 되어 이 정보를 요청할 때 디바이스에 최신 상태 정보를 다시 전달하는 방법도 보여줍니다.

생성하는 데 필요한 AWS IoT 리소스 및 필요한 하드웨어 설정부터 시작하여 여기에 표시된 순서대로 이 자습서를 시도하는 것이 좋습니다. 그러면 개념을 점진적으로 배우는 데도 도움이 됩니다. 이 자습서에서는와 함께 사용할 Raspberry Pi 디바이스를 구성하고 연결하는 방법을 보여줍니다 AWS IoT. 필요한 하드웨어가 없는 경우 자습서를 사용자의 디바이스에 맞게 적용하거나 Amazon EC2로 가상 디바이스를 생성하여 자습서를 따라할 수 있습니다.

자습서 시나리오 개요

이 자습서의 시나리오는 전구의 색상을 변경하고 해당 데이터를 예약된 섀도우 주제에 게시하는 로컬 앱 또는 서비스입니다. 이 자습서는 대화형 시작하기 자습서에 설명된 디바이스 섀도우 기능과 유사하며 Raspberry Pi 디바이스에서 구현됩니다. 이 섹션의 자습서에서는 단일 클래식 섀도우에 초점을 맞추면서 명명된 섀도우 또는 여러 디바이스를 수용할 수 있는 방법을 보여줍니다.

다음 자습서는 AWS IoT 디바이스 섀도우 서비스를 사용하는 방법을 배우는 데 도움이 됩니다.

- 튜토리얼: 섀도우 애플리케이션 실행을 위해 Raspberry Pi 준비

이 자습서에서는 연결을 위해 Raspberry Pi 디바이스를 설정하는 방법을 보여줍니다 AWS IoT. 또한 AWS IoT 정책 문서와 사물 리소스를 생성하고 인증서를 다운로드한 다음 해당 사물 리소스에 정책을 연결합니다. 이 자습서는 완료하는 데 약 30분 소요됩니다.

- 자습서: 디바이스 SDK 설치 및 디바이스 섀도우용 샘플 애플리케이션 실행

  이 자습서에서는 필요한 도구, 소프트웨어 및 AWS IoT Device SDK for Python을 설치한 다음 샘플 섀도우 애플리케이션을 실행하는 방법을 보여줍니다. 이 자습서는 Raspberry Pi 또는 다른 디바이스 연결에 제시된 개념을 기반으로 하며 완료하는 데 20분이 소요됩니다.
- 자습서: 샘플 앱 및 MQTT 테스트 클라이언트를 사용하여 디바이스 섀도우와 상호 작용

  이 자습서에서는 `shadow.py` 샘플 앱과 AWS IoT 콘솔을 사용하여 AWS IoT 디바이스 섀도우와 전구의 상태 변경 간의 상호 작용을 관찰하는 방법을 보여줍니다. 이 자습서에서는 디바이스 섀도우의 예약된 주제에 MQTT 메시지를 전송하는 방법도 보여줍니다. 이 자습서는 완료하는 데 약 45분이 소요됩니다.

AWS IoT 디바이스 섀도우 개요

디바이스 섀도우는 AWS IoT 레지스트리에서 생성한 사물 리소스로 관리되는 디바이스의 지속적인 가상 표현입니다. 섀도우 문서는 디바이스의 현재 상태 정보를 저장하고 검색하는 데 사용되는 JSON 또는 JavaScript 표기법 문서입니다. 디바이스가 인터넷에 연결되어 있는지 여부에 관계없이 섀도우를 사용하여 MQTT 주제 또는 HTTP REST API를 통해 디바이스의 상태를 가져오고 설정할 수 있습니다.

섀도우 문서에는 디바이스 상태의 이러한 측면을 설명하는 `state` 속성이 포함되어 있습니다.

- `desired`: 앱은 `desired` 객체를 업데이트하여 디바이스 속성에 대해 원하는 상태를 지정합니다.
- `reported`: 디바이스는 `reported` 객체에 현재 상태를 보고합니다.
- `delta`: `delta` 객체에서 원하는 상태와 보고된 상태 간의 차이를 AWS IoT 보고합니다.

다음은 섀도우 문서의 예입니다.

```
{
  "state": {
    "desired": {
      "color": "green"
      },
    "reported": {
      "color": "blue"
      },
```

```
    "delta": {
      "color": "green"
      }
   }
}
```

디바이스의 섀도우 문서를 업데이트하려면 예약된 MQTT 주제, HTTP로 GET, UPDATE 및 DELETE 작업을 지원하는 디바이스 섀도우 REST API 및 AWS IoT CLI를 사용합니다.

이전 예에서 desired 색상을 yellow로 변경하려 한다고 가정합니다. 이렇게 하려면 UpdateThingShadow API에 요청을 전송하거나 업데이트 주제 $aws/things/THING_NAME/shadow/update에 메시지를 게시하세요.

```
{
  "state": {
    "desired": {
      "color": yellow
    }
  }
}
```

업데이트는 요청에 지정된 필드에만 영향을 미칩니다. 디바이스 섀도우를 성공적으로 업데이트한 후는 새 desired 상태를 delta 주제에 AWS IoT 게시합니다$aws/things/THING_NAME/shadow/delta. 이 경우 섀도우 문서는 다음과 같습니다.

```
{
  "state": {
    "desired": {
      "color": yellow
    },
    "reported": {
      "color": green
    },
    "delta": {
      "color": yellow
      }
  }
}
```

그러면 새 상태가 다음 JSON 메시지$aws/things/THING_NAME/shadow/update와 함께 Update 주제를 사용하여 AWS IoT 디바이스 섀도우에 보고됩니다.

```
{
  "state": {
    "reported": {
      "color": yellow
    }
  }
}
```

현재 상태 정보를 얻으려면 GetThingShadow API에 요청을 전송하거나 Get 주제 `$aws/things/THING_NAME/shadow/get`에 MQTT 메시지를 게시하세요.

디바이스 섀도우 서비스 사용에 대한 자세한 내용은 AWS IoT 디바이스 섀도우 서비스 단원을 참조하세요.

디바이스, 앱 및 서비스에서 디바이스 섀도우 사용에 대한 자세한 내용은 디바이스에서 섀도우 사용 및 앱 및 서비스에서 섀도우 사용 단원을 참조하세요.

AWS IoT 섀도우와의 상호 작용에 대한 자세한 내용은 섹션을 참조하세요섀도우와의 상호 작용.

MQTT 예약 주제 및 HTTP REST API에 대한 자세한 내용은 디바이스 섀도우 MQTT 주제 및 디바이스 섀도우 REST API 단원을 참조하세요.

튜토리얼: 섀도우 애플리케이션 실행을 위해 Raspberry Pi 준비

이 자습서에서는 Raspberry Pi 디바이스를 설정 및 구성하고 디바이스가 MQTT 메시지를 연결하고 교환하는 데 필요한 AWS IoT 리소스를 생성하는 방법을 보여줍니다.

> **Note**
>
> the section called “Amazon EC2를 사용하여 가상 디바이스 생성”을(를) 계획 중인 경우 이 페이지를 건너뛰고 the section called “디바이스 구성”을(를) 계속할 수 있습니다. 가상 사물을 만들 때 이러한 리소스를 만듭니다. Raspberry Pi가 아닌 다른 디바이스를 사용하려는 경우 원하는 디바이스에 맞게 이러한 자습서를 따라해볼 수 있습니다.

이 자습서에서는 다음을 수행하는 방법을 알아봅니다.

- Raspberry Pi 디바이스를 설정하고와 함께 사용하도록 구성합니다 AWS IoT.
- 디바이스가 서비스와 상호 작용 AWS IoT 할 수 있는 권한을 부여하는 AWS IoT 정책 문서를 생성합니다.

- X.509 디바이스 인증서 AWS IoT 에서 사물 리소스를 생성한 다음 정책 문서를 연결합니다.

  문제는 AWS IoT 레지스트리에 있는 디바이스의 가상 표현입니다. 인증서는 코어에 디바이스를 인증 AWS IoT 하고 정책 문서는 디바이스가 상호 작용할 수 있는 권한을 부여합니다 AWS IoT.

이 자습서를 실행하는 방법

디바이스 섀도우용 `shadow.py` 샘플 애플리케이션을 실행하려면 AWS IoT에 연결하는 Raspberry Pi 디바이스가 필요합니다. 여기에 나와 있는 순서대로 이 자습서를 따라하여 Raspberry Pi 및 해당 액세서리 설정부터 시작해서 정책을 생성하고, 생성한 사물 리소스에 이 정책을 연결하는 것이 좋습니다. 그런 다음 Raspberry Pi에서 지원하는 그래픽 사용자 인터페이스(GUI)를 사용하여 디바이스의 웹 브라우저에서 AWS IoT 콘솔을 열어이 자습서를 따를 수 있습니다. 그러면 연결을 위해 인증서를 Raspberry Pi에 직접 다운로드할 수 있습니다 AWS IoT.

이 자습서를 시작하기 전에 다음 사항을 확인해야 합니다.

- AWS 계정. 없는 경우 계속하기 전에 설정 AWS 계정에서 설명하는 단계를 완료하세요. 이 자습서를 완료하려면 AWS 계정 및 AWS IoT 콘솔이 필요합니다.
- Raspberry Pi 및 필요한 해당 액세서리. 다음 사항이 필요합니다.
  - Raspberry Pi 3 Model B 또는 최신 모델. 이 자습서는 이전 버전의 Raspberry Pi에서 작동할 수 있지만 테스트되지 않았습니다.
  - Raspberry Pi OS(32비트) 이상. 최신 버전의 Raspberry Pi OS를 사용하는 것이 좋습니다. 이전 버전의 OS는 작동하지만 테스트되지 않았습니다.
  - 이더넷 또는 Wi-Fi 연결.
  - 키보드, 마우스, 모니터, 케이블 및 전원 공급장치.

이 자습서는 완료하는 데 약 30분 소요됩니다.

1단계: Raspberry Pi 디바이스 설정 및 구성

이 섹션에서는와 함께 사용할 Raspberry Pi 디바이스를 구성합니다 AWS IoT.

> **Important**
>
> 이러한 지침을 다른 디바이스 및 운영 체제에 적용하는 것은 어려울 수 있습니다. 이러한 지침을 해석하고 디바이스에 적용할 수 있을 만큼 디바이스를 잘 이해해야 합니다. 문제가 발생하

면 다른 디바이스 옵션(예: Amazon EC2를 사용하여 가상 디바이스 생성 또는 Windows 또는 Linux PC 또는 Mac을 AWS IoT 디바이스로 사용) 중 하나를 대안으로 사용해 볼 수 있습니다.

운영 체제(OS)를 시작하고, 인터넷에 연결하고, 명령줄 인터페이스에서 상호 작용할 수 있도록 Raspberry Pi를 구성해야 합니다. Raspberry Pi와 함께 지원되는 그래픽 사용자 인터페이스(GUI)를 사용하여 AWS IoT 콘솔을 열고이 자습서의 나머지 부분을 실행할 수도 있습니다.

Raspberry Pi를 설정하려면 다음을 수행합니다.

1. SD 카드를 Raspberry Pi의 MicroSD 카드 슬롯에 삽입합니다. 일부 SD 카드에는 보드를 부팅한 후 OS 설치 메뉴를 표시하는 설치 관리자가 미리 로드되어 있습니다. Raspberry Pi 이미저를 사용하여 카드에 OS를 설치할 수도 있습니다.
2. Raspberry Pi의 HDMI 포트에 연결하는 HDMI 케이블에 HDMI TV 또는 모니터를 연결합니다.
3. 키보드와 마우스를 Raspberry Pi의 USB 포트에 연결한 다음 전원 어댑터를 연결하여 보드를 부팅합니다.

Raspberry Pi가 부팅된 후, SD 카드에 설치 관리자가 미리 로드된 경우 운영 체제를 설치하는 메뉴가 나타납니다. OS를 설치하는 데 문제가 있는 경우 다음 단계를 시도할 수 있습니다. Raspberry Pi 설정에 대한 자세한 내용은 Raspberry Pi 설정을 참조하세요.

Raspberry Pi 설정에 문제가 있는 경우:

- 보드를 부팅하기 전에 SD 카드를 삽입했는지 확인하세요. 보드를 부팅한 후 SD 카드를 연결하면 설치 메뉴가 나타나지 않을 수 있습니다.
- TV 또는 모니터가 켜져 있고 올바른 입력이 선택되어 있는지 확인합니다.
- Raspberry Pi 호환 소프트웨어를 사용하고 있는지 확인합니다.

Raspberry Pi OS를 설치하고 구성한 후 Raspberry Pi의 웹 브라우저를 열고 AWS IoT Core 콘솔로 이동하여이 자습서의 나머지 단계를 계속합니다.

AWS IoT Core 콘솔을 열 수 있으면 Raspberry Pi가 준비되고 로 계속 진행할 수 있습니다the section called “에서 디바이스 프로비저닝 AWS IoT ”.

문제가 계속되거나 추가 도움이 필요한 경우 Raspberry Pi에 대한 도움말을 참조하세요

자습서:에서 디바이스 프로비저닝 AWS IoT

이 섹션에서는 자습서에서 사용할 AWS IoT Core 리소스를 생성합니다.

에서 디바이스를 프로비저닝하는 단계 AWS IoT

- 1단계: 디바이스 섀도우에 대한 AWS IoT 정책 생성
- 2단계: 사물 리소스를 만들고 정책을 사물에 연결
- 3단계: 결과 및 다음 단계 검토

1단계: 디바이스 섀도우에 대한 AWS IoT 정책 생성

X.509 인증서는 디바이스를 인증합니다 AWS IoT Core. AWS IoT 정책은 디바이스가 디바이스 섀도우 서비스에서 사용하는 MQTT 예약 주제를 구독하거나 게시하는 등의 AWS IoT 작업을 수행하도록 허용하는 인증서에 연결됩니다. 디바이스가 연결하고 메시지를 보낼 때 인증서를 제공합니다 AWS IoT Core.

이 절차에서는 디바이스가 예제 프로그램을 실행하는 데 필요한 AWS IoT 작업을 수행하도록 허용하는 정책을 만듭니다. 작업을 수행하는 데 필요한 권한만 부여하는 정책을 생성하는 것이 좋습니다. 먼저 AWS IoT 정책을 생성한 다음 나중에 생성할 디바이스 인증서에 연결합니다.

AWS IoT 정책을 생성하려면

1. 왼쪽 메뉴에서 보안(Secure)을 선택한 다음 정책(Policies)을 선택합니다. 계정에 기존 정책이 있는 경우 생성(Create)을 선택하고, 그렇지 않은 경우 아직 정책이 없습니다(You don't have a policy yet) 페이지에서 정책 생성(Create a policy)을 선택합니다.
2. 정책 생성 페이지에서 다음을 수행합니다.

   a. 이름(Name) 필드에 정책 이름(예: **`My_Device_Shadow_policy`**)을 입력합니다. 정책 이름에 개인 식별 정보를 사용하면 안 됩니다.

   b. 정책 문서에서, MQTT 예약 주제를 게시하고 구독할 수 있는 권한을 디바이스에 부여하는 연결, 구독, 수신 및 게시 작업을 설명합니다.

      다음 샘플 정책을 복사하여 정책 문서에 붙여넣습니다. `thingname`를 생성할 사물의 이름(예: `My_light_bulb`), `region`를 서비스를 사용 중인 AWS IoT 리전, `account`를 사용자 AWS 계정 번호로 바꿉니다. AWS IoT 정책에 대한 자세한 내용은 섹션을 참조하세요AWS IoT Core 정책.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/get/accepted",
```

```
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/get/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/test-*"
        }
    ]
}
```

2단계: 사물 리소스를 만들고 정책을 사물에 연결

에 연결된 디바이스는 AWS IoT 레지스트리의 사물 리소스로 표현할 AWS IoT 수 있습니다. 사물 리소스는 이 자습서의 전구와 같은 특정 디바이스 또는 논리적 엔터티를 나타냅니다.

에서 사물을 생성하는 방법을 알아보려면에 설명된 단계를 AWS IoT따르세요사물 객체 만들기. 이 자습서의 단계를 수행할 때 주의해야 할 주요 사항은 다음과 같습니다.

1. 단일 사물 생성(Create a single thing)을 선택하고 이름(Name) 필드에 이전에 정책을 생성할 때 지정한 `thingname`(예: `My_light_bulb`)과 동일한 사물의 이름을 입력합니다.

   사물 이름이 생성된 이후에는 그 이름을 바꿀 수 없습니다. `thingname`이 아닌 다른 이름을 부여했다면 `thingname`이라는 이름으로 새 것을 만들고 이전 것을 삭제하세요.

   > **Note**
   >
   > 사물 이름에 개인 식별 정보를 사용하면 안 됩니다. 사물 이름은 암호화되지 않은 통신 및 보고서에 나타날 수 있습니다.

2. 인증서 생성됨(Certificate created!) 페이지의 각 인증서 파일을, 쉽게 찾을 수 있는 위치에 다운로드하는 것이 좋습니다. 샘플 애플리케이션을 실행하려면 이러한 파일을 설치해야 합니다.

Raspberry Pi의 home 디렉터리에 있는 `certs` 하위 디렉터리에 파일을 다운로드하고 다음 표에서 제안하는 대로 각각의 이름을 더 간단한 이름으로 지정하는 것이 좋습니다.

인증서 파일 이름

파일	파일 경로
루트 CA 인증서	`~/certs/Amazon-root-CA-1.pem`
디바이스 인증서	`~/certs/device.pem.crt`
프라이빗 키	`~/certs/private.pem.key`

3. 인증서를 활성화하여 연결을 활성화한 후 정책 연결을 AWS IoT선택하고 이전에 생성한 정책(예: **`My_Device_Shadow_policy`**)을 사물에 연결했는지 확인합니다.

   사물을 생성한 후에는 AWS IoT 콘솔의 사물 목록에 사물 리소스가 표시됩니다.

3단계: 결과 및 다음 단계 검토

이 자습서에서는 다음을 수행하는 방법을 알아보았습니다.

- Raspberry Pi 디바이스 설정 및 구성.
- 디바이스가 서비스와 상호 작용 AWS IoT 할 수 있도록 권한을 부여하는 AWS IoT 정책 문서를 생성합니다.
- 사물 리소스 및 연결된 X.509 디바이스 인증서를 만들고 여기에 정책 문서 연결.

다음 단계

이제 Python용 AWS IoT 디바이스 SDK를 설치하고, `shadow.py` 샘플 애플리케이션을 실행하고, 디바이스 섀도우를 사용하여 상태를 제어할 수 있습니다. 이 자습서를 실행하는 방법에 대한 자세한 내용은 자습서: 디바이스 SDK 설치 및 디바이스 섀도우용 샘플 애플리케이션 실행 단원을 참조하세요.

자습서: 디바이스 SDK 설치 및 디바이스 섀도우용 샘플 애플리케이션 실행

이 섹션에서는 필요한 소프트웨어와 Python용 AWS IoT Device SDK를 설치하고 `shadow.py` 샘플 애플리케이션을 실행하여 섀도우 문서를 편집하고 섀도우 상태를 제어하는 방법을 보여줍니다.

이 자습서에서는 다음을 수행하는 방법을 알아봅니다.

- 설치된 소프트웨어와 AWS IoT Device SDK for Python을 사용하여 샘플 앱을 실행합니다.
- 샘플 앱을 사용하여 값을 입력하면 AWS IoT 콘솔에 원하는 값이 게시되도록 하는 방법을 알아봅니다.
- `shadow.py` 샘플 앱과 MQTT 프로토콜을 사용하여 섀도우 상태를 업데이트하는 방법을 검토합니다.

이 자습서를 실행하기 전에 다음을 수행합니다.

를 설정하고 AWS 계정, Raspberry Pi 디바이스를 구성하고, 디바이스에 디바이스 섀도우 서비스의 MQTT 예약 주제를 게시하고 구독할 수 있는 권한을 부여하는 AWS IoT 사물 및 정책을 생성해야 합니다. 자세한 내용은 튜토리얼: 섀도우 애플리케이션 실행을 위해 Raspberry Pi 준비 단원을 참조하십시오.

Git, Python 및 AWS IoT Device SDK for Python도 설치해야 합니다. 이 자습서는 자습서 Raspberry Pi 또는 다른 디바이스 연결에 제공된 개념을 기반으로 합니다. 해당 자습서를 시도하지 않은 경우 해당 자습서에 설명된 단계에 따라 인증서 파일 및 Device SDK를 설치한 다음 이 자습서로 돌아와서 `shadow.py` 샘플 앱을 실행하는 것이 좋습니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 1단계: shadow.py 샘플 앱 실행
- 2단계: shadow.py 디바이스 SDK 샘플 앱 검토
- 3단계: shadow.py 샘플 앱 문제 해결
- 4단계: 결과 및 다음 단계 검토

이 자습서는 완료하는 데 약 20분이 소요됩니다.

1단계: shadow.py 샘플 앱 실행

`shadow.py` 샘플 앱을 실행하기 전에 설치한 인증서 파일의 이름과 위치 외에 다음 정보가 필요합니다.

애플리케이션 파라미터 값

파라미터	값을 찾을 수 있는 위치
your-iot-thing-name	에서 이전에 생성한 AWS IoT 사물의 이름입니다the section called “2단계: 사물 리소스를 만들고 정책을 사물에 연결”. 이 값을 찾으려면 AWS IoT 콘솔에서 관리(Manage)를 선택한 다음 사물(Things)을 선택합니다.
your-iot-endpoint	*your-iot-endpoint* 값의 형식은 *endpoint_id* -ats.iot. *region*.amazonaws.com (예: a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com)입니다. 이 값을 찾으려면: 1. AWS IoT 콘솔에서 관리(Manage)를 선택한 다음 사물(Things)을 선택합니다. 2. 디바이스용으로 생성한 IoT 사물(이전에 사용한 이름인 My_light_bulb)을 선택한 다음 상호작용(Interact)을 선택합니다. 사물 세부 정보 페이지의 HTTPS 섹션에 엔드포인트가 표시됩니다.

샘플 앱 설치 및 실행

1. 샘플 앱 디렉터리로 이동합니다.

```
cd ~/aws-iot-device-sdk-python-v2/samples/service-clients
```

2. 명령줄 창에서 표시된 대로 *your-iot-endpoint* 및 *your-iot-thing-name*을 바꾸고 이 명령을 실행합니다.

```
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

3. 샘플 앱에서 다음 사항을 확인합니다.

   1. 계정의 AWS IoT 서비스에 연결합니다.
   2. `Delta` 이벤트와 `Update` 및 `Get` 응답을 구독합니다.
   3. 터미널에 원하는 값을 입력하라는 메시지가 표시됩니다.
   4. 다음과 유사한 출력이 표시됩니다.

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow contains reported value 'off'.
Enter desired value:
```

> **Note**
>
> `shadow.py` 샘플 앱을 실행하는 데 문제가 있는 경우 the section called “3단계: `shadow.py` 샘플 앱 문제 해결”을(를) 검토합니다. 문제를 해결하는 데 도움이 될 수 있는 추가 정보를 얻으려면 명령줄에 `--verbosity debug` 파라미터를 추가하여 샘플 앱이 수행 중인 작업에 대한 자세한 메시지를 표시하도록 합니다.

섀도우 문서에서 값을 입력하고 업데이트를 관찰합니다.

터미널에 값을 입력하여 `desired` 값을 업데이트합니다. 이는 `reported` 값도 업데이트합니다. 터미널에 색상 `yellow`를 입력한다고 가정합니다. `reported` 값도 `yellow` 색상으로 업데이트됩니다. 다음은 터미널에 표시되는 메시지를 보여줍니다.

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
```

```
Finished updating reported shadow value to 'yellow'.
```

이 업데이트 요청을 게시하면 사물 리소스에 대한 기본 클래식 섀도우가 AWS IoT 생성됩니다. 생성한 사물 리소스(예:)에 대한 섀도우 문서를 확인하여 AWS IoT 콘솔의 `reported` 및 `desired` 값에 게시한 업데이트 요청을 관찰할 수 있습니다`My_light_bulb`. 섀도우 문서에서 업데이트를 보려면:

1. AWS IoT 콘솔에서 관리를 선택한 다음 사물을 선택합니다.
2. 표시된 사물 목록에서 생성한 사물을 선택하고 섀도우(Shadows)를 선택한 다음 클래식 섀도우(Classic Shadow)를 선택합니다.

섀도우 문서는 다음과 유사해야 하며, `yellow` 색상으로 설정된 `reported` 및 `desired` 값을 보여줍니다. 이 값들은 문서의 섀도우 상태 섹션에 나와 있습니다.

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "yellow"
},
"reported": {
  "welcome": "aws-iot",
  "color": "yellow"
}
}
```

타임스탬프 정보와 요청 버전 번호가 포함된 메타데이터 섹션도 표시됩니다.

상태 문서 버전을 사용하면 업데이트하는 디바이스 섀도우 문서가 최신 버전인지 확인할 수 있습니다. 다른 업데이트 요청을 전송하면 버전 번호가 1씩 증가합니다. 업데이트 요청에 버전을 입력하면 상태 문서의 현재 버전이 입력된 버전과 일치하지 않을 경우 서비스가 HTTP 409 충돌 응답 코드와 함께 요청을 거부합니다.

```
{
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
```

```
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  }
},
"version": 10
}
```

섀도우 문서에 대해 자세히 알아보고 상태 정보의 변경 사항을 관찰하려면 이 자습서의 4단계: 결과 및 다음 단계 검토 섹션에 설명된 대로 다음 자습서 자습서: 샘플 앱 및 MQTT 테스트 클라이언트를 사용하여 디바이스 섀도우와 상호 작용(으)로 진행하세요. 선택적으로 다음 섹션에서 shadow.py 샘플 코드 및 이 코드가 MQTT 프로토콜을 사용하는 방법에 대해 알아볼 수도 있습니다.

2단계: shadow.py 디바이스 SDK 샘플 앱 검토

이 섹션에서는 이 자습서에 사용된 AWS IoT Device SDK v2 for Python의 shadow.py 샘플 앱을 검토합니다. 여기서는 MQTT 및 MQTT over WSS 프로토콜을 AWS IoT Core 사용하여에 연결하는 방법을 살펴보겠습니다. AWS 공통 런타임(AWS-CRT) 라이브러리는 하위 수준 통신 프로토콜을 지원하며 Python용 AWS IoT Device SDK v2에 포함되어 있습니다.

이 자습서에서는 WSS를 통해 MQTT 및 MQTT를 사용하지만는 HTTPS 요청을 게시하는 디바이스를 AWS IoT 지원합니다. 디바이스에서 HTTP 메시지를 전송하는 Python 프로그램의 예는 Python의 requests 라이브러리를 사용하는 HTTPS 코드 예를 참조하세요.

디바이스 통신에 사용할 프로토콜에 대한 정보를 바탕으로 결정을 내릴 수 있는 방법에 대한 자세한 내용은 디바이스 통신을 위한 애플리케이션 프로토콜 선택 단원을 검토하세요.

MQTT

shadow.py 샘플은 mqtt_connection_builder에서 mtls_from_path(여기에 표시됨)를 직접 호출해서 MQTT 프로토콜을 사용하여 AWS IoT Core 와의 연결을 설정합니다. mtls_from_path는 X.509 인증서와 TLS v1.2를 사용하여 디바이스를 인증합니다. AWS CRT 라이브러리는 해당 연결의 하위 수준 세부 정보를 처리합니다.

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
```

```
    endpoint=args.endpoint,
    cert_filepath=args.cert,
    pri_key_filepath=args.key,
    ca_filepath=args.ca_file,
    client_bootstrap=client_bootstrap,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

- endpoint는 명령줄에서 전달한 AWS IoT 엔드포인트이며 client_id는에서이 디바이스를 고유하게 식별하는 ID입니다 AWS 리전.
- cert_filepath,pri_key_filepath 및 ca_filepath는 디바이스의 인증서 및 프라이빗 키 파일과 루트 CA 파일에 대한 경로입니다.
- client_bootstrap은 소켓 통신 활동을 처리하는 공통 런타임 객체이며 mqtt_connection_builder.mtls_from_path를 호출하기 전에 인스턴스화됩니다.
- on_connection_interrupted 및 on_connection_resumed는 디바이스의 연결이 중단되었다가 재개될 때 호출하는 콜백 함수입니다.
- clean_session은 새로운 영구 세션을 시작할지 또는 존재하는 경우 기존 세션에 다시 연결할지 여부입니다. keep_alive_secs는 CONNECT 요청에서 전송할 연결 유지 값(초)입니다. 이 간격으로 ping이 자동으로 전송됩니다. 서버는 이 값의 1.5배 후에 ping을 수신하지 않으면 연결이 끊어지는 것으로 가정합니다.

shadow.py 샘플은 또한 mqtt_connection_builder에서 websockets_with_default_aws_signing을 호출하여 WSS를 통해 MQTT 프로토콜을 사용하여 AWS IoT Core 와의 연결을 설정합니다. MQTT over WSS는 MQTT와 동일한 파라미터를 사용하며 다음과 같은 추가 파라미터를 사용합니다.

- region는 Signature V4 인증에 사용되는 AWS 서명 리전이며 credentials_provider는 인증에 사용하기 위해 제공되는 AWS 자격 증명입니다. 리전은 명령줄에서 전달되고 credentials_provider 객체는 mqtt_connection_builder.websockets_with_default_aws_signing을 호출하기 직전에 인스턴스화됩니다.
- websocket_proxy_options는 프록시 호스트를 사용하는 경우 HTTP 프록시 옵션입니다. shadow.py 샘플 앱에서 이 값은

mqtt_connection_builder.websockets_with_default_aws_signing을 호출하기 직전에 인스턴스화됩니다.

섀도우 주제 및 이벤트 구독

shadow.py 샘플은 연결을 설정하려고 시도하고 완전히 연결될 때까지 기다립니다. 연결되어 있지 않으면 명령이 대기열에 저장됩니다. 연결되면 샘플은 델타 이벤트를 구독하고 메시지를 업데이트하고 가져오고 서비스 품질(QoS) 수준이 1(mqtt.QoS.AT_LEAST_ONCE)인 메시지를 게시합니다.

디바이스가 QoS 수준 1의 메시지를 구독하면 메시지 브로커는 디바이스로 전송할 수 있을 때까지 디바이스가 구독한 메시지를 저장합니다. 메시지 브로커는 디바이스로부터 PUBACK 응답을 받을 때까지 메시지를 다시 전송합니다.

MQTT 프로토콜에 대한 자세한 내용은 MQTT 프로토콜 검토 및 MQTT 단원을 참조하세요.

이 자습서에서 사용되는 MQTT, MQTT over WSS, 영구 세션 및 QoS 수준에 대한 자세한 내용은 pubsub.py Device SDK 샘플 앱 검토 단원을 참조하세요.

3단계: **shadow.py** 샘플 앱 문제 해결

shadow.py 샘플 앱을 실행하면 터미널에 일부 메시지가 표시되고 desired 값을 입력하라는 메시지가 표시되어야 합니다. 프로그램에서 오류가 발생한 경우 오류를 디버깅하려면 먼저 시스템에 올바른 명령을 실행했는지 여부를 확인할 수 있습니다.

경우에 따라 오류 메시지는 연결 문제를 나타내며 Host name was invalid for dns resolution 또는 Connection was closed unexpectedly와 유사하게 표시될 수 있습니다. 이 경우 확인할 수 있는 사항은 다음과 같습니다.

- 명령에서 엔드포인트 주소 확인

  샘플 앱을 실행하기 위해 입력한 명령의 endpoint 인수(예: a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com)를 검토하고 AWS IoT 콘솔에서 이 값을 확인합니다.

  올바른 값을 사용했는지 확인하려면 다음과 같이 하세요.

  1. AWS IoT 콘솔에서 관리(Manage)를 선택한 다음 사물(Things)을 선택합니다.
  2. 샘플 앱에 대해 만든 사물(예: My_light_bulb)을 선택한 다음 상호 작용(Interact)을 선택합니다.

  사물 세부 정보 페이지의 HTTPS 섹션에 엔드포인트가 표시됩니다. 또한 This thing already appears to be connected.와 같은 메시지가 표시됩니다.

- 인증서 활성화 확인

  인증서는 디바이스를 인증합니다 AWS IoT Core.

  인증서가 활성 상태인지 확인하려면 다음과 같이 하세요.

  1. AWS IoT 콘솔에서 관리(Manage)를 선택한 다음 사물(Things)을 선택합니다.
  2. 샘플 앱에 대해 만든 사물(예: My_light_bulb)를 선택한 다음 보안(Security)을 선택합니다.
  3. 인증서를 선택하고 인증서의 세부 정보 페이지에서 인증서 선택을 선택한 후 인증서의 세부 정보 페이지에서 작업(Actions)을 선택합니다.

  드롭다운 목록에서 활성화(Activate)를 사용할 수 없고 비활성화(Deactivate)만 선택할 수 있는 경우 인증서가 활성화된 것입니다. 그렇지 않은 경우 활성화(Activate)를 선택하고 샘플 프로그램을 다시 실행하세요.

  그래도 프로그램이 실행되지 않으면 `certs` 폴더의 인증서 파일 이름을 확인합니다.

- 사물 리소스에 연결된 정책 확인

  인증서가 디바이스를 인증하는 동안 AWS IoT 정책은 디바이스가 MQTT 예약 주제 구독 또는 게시와 같은 AWS IoT 작업을 수행하도록 허용합니다.

  올바른 정책이 연결되어 있는지 확인하려면 다음과 같이 하세요.

  1. 앞에서 설명한 대로 인증서를 찾은 다음 정책(Policies)을 선택합니다.
  2. 표시된 정책을 선택하고, MQTT 예약 주제를 게시하고 구독할 수 있는 권한을 디바이스에 부여하는 `connect`, `subscribe`, `receive` 및 `publish` 작업을 설명하는지 확인하세요.

     샘플 정책은 1단계: 디바이스 섀도우에 대한 AWS IoT 정책 생성 단원을 참조하세요.

  에 연결하는 AWS IoT데 문제가 있음을 나타내는 오류 메시지가 표시되면 정책에 사용 중인 권한 때문일 수 있습니다. 이 경우 AWS IoT 리소스에 대한 전체 액세스를 제공하는 정책으로 시작한 다음 샘플 프로그램을 다시 실행하는 것이 좋습니다. 현재 정책을 편집하거나 현재 정책을 선택하고 분리(Detach)를 선택한 다음 전체 액세스 권한을 제공하는 다른 정책을 생성하여 사물 리소스에 연결할 수 있습니다. 나중에 프로그램을 실행하는 데 필요한 작업 및 정책으로만 정책을 제한할 수 있습니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
```

```
            "Action": [
                "iot:*"
            ],
            "Resource": "*"
        }
    ]
}
```

- Device SDK 설치 확인

  그래도 프로그램이 실행되지 않으면 Device SDK를 다시 설치하여 SDK 설치가 올바르게 완료되었는지 확인할 수 있습니다.

4단계: 결과 및 다음 단계 검토

이 자습서에서는 다음을 수행하는 방법을 알아보았습니다.

- 필요한 소프트웨어, 도구 및 AWS IoT Device SDK for Python을 설치합니다.
- 샘플 앱 `shadow.py`가 MQTT 프로토콜을 사용하여 섀도우의 현재 상태를 검색하고 업데이트하는 방식 이해.
- 디바이스 섀도우용 샘플 앱을 실행하고 AWS IoT 콘솔에서 섀도우 문서에 대한 업데이트를 관찰합니다. 또한 프로그램을 실행할 때 문제를 해결하고 오류를 수정하는 방법을 배웠습니다.

다음 단계

이제 `shadow.py` 샘플 애플리케이션을 실행하고 디바이스 섀도우를 사용하여 상태를 제어할 수 있습니다. AWS IoT 콘솔에서 섀도우 문서에 대한 업데이트를 관찰하고 샘플 앱이 응답하는 델타 이벤트를 관찰할 수 있습니다. MQTT 테스트 클라이언트를 사용하여 예약된 섀도 주제를 구독하고 샘플 프로그램을 실행할 때 주제에서 수신한 메시지를 관찰할 수 있습니다. 이 자습서를 실행하는 방법에 대한 자세한 내용은 자습서: 샘플 앱 및 MQTT 테스트 클라이언트를 사용하여 디바이스 섀도우와 상호 작용 단원을 참조하세요.

자습서: 샘플 앱 및 MQTT 테스트 클라이언트를 사용하여 디바이스 섀도우와 상호 작용

`shadow.py` 샘플 앱과 상호 작용하려면 터미널에 `desired` 값에 대한 값을 입력합니다. 예를 들어 신호등과 비슷하고 요청에 AWS IoT 응답하며 보고된 값을 업데이트하는 색상을 지정할 수 있습니다.

이 자습서에서는 다음을 수행하는 방법을 알아봅니다.

- `shadow.py` 샘플 앱을 사용하여 원하는 상태를 지정하고 섀도우의 현재 상태를 업데이트합니다.

- 섀도우 문서를 편집하여 델타 이벤트를 관찰하고 shadow.py 샘플 앱이 델타 이벤트에 응답하는 방식 관찰.
- MQTT 테스트 클라이언트를 사용하여 섀도우 주제를 구독하고 샘플 프로그램 실행 시 업데이트를 관찰합니다.

이 자습서를 실행하기 전에 수행해야 할 사항:

를 설정하고 AWS 계정, Raspberry Pi 디바이스를 구성하고, 사물과 정책을 생성 AWS IoT 했습니다. 또한 필수 소프트웨어, Device SDK, 인증서 파일을 설치하고 터미널에서 샘플 프로그램을 실행했어야 합니다. 자세한 내용은 이전 자습서 튜토리얼: 섀도우 애플리케이션 실행을 위해 Raspberry Pi 준비 및 1단계: shadow.py 샘플 앱 실행을(를) 참조하세요. 이들 자습서를 아직 완료하지 않았다면 완료해야 합니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 1단계: shadow.py 샘플 앱을 사용하여 원하는 값 및 보고된 값 업데이트
- 2단계: MQTT 테스트 클라이언트에서 shadow.py 샘플 앱의 메시지 보기
- 3단계: 디바이스 섀도우 상호 작용 오류 문제 해결
- 4단계: 결과 및 다음 단계 검토

이 자습서는 완료하는 데 약 45분이 소요됩니다.

1단계: **shadow.py** 샘플 앱을 사용하여 원하는 값 및 보고된 값 업데이트

이전 자습서 에서는 단원에 설명된 대로 원하는 값을 입력할 때 AWS IoT 콘솔의 섀도우 문서에 게시된 메시지를 관찰하는 방법을 배1단계: shadow.py 샘플 앱 실행웠습니다자습서: 디바이스 SDK 설치 및 디바이스 섀도우용 샘플 애플리케이션 실행.

이전 예에서 원하는 색상을 yellow로 설정했습니다. 각 값을 입력하면 터미널에서 다른 desired 값을 입력하라는 메시지가 표시됩니다. 같은 값(yellow)을 다시 입력하면 앱이 이를 인식하고 새 desired 값을 입력하라는 메시지를 표시합니다.

```
Enter desired value:
yellow
Local value is already 'yellow'.
Enter desired value:
```

이제 색상을 입력한다고 가정합니다green.는 요청에 AWS IoT 응답하고 reported 값을 로 업데이트합니다green. 이런 방식으로 desired 상태가 reported 상태와 다를 때 업데이트가 발생하여 델타가 발생하게 됩니다.

shadow.py 샘플 앱이 디바이스 섀도우 상호 작용을 시뮬레이션하는 방식:

1. 터미널에 desired 값(예: yellow)을 입력하여 원하는 상태를 게시합니다.
2. desired 상태가 reported 상태(예: green 색상)와 다르기 때문에 델타가 발생하고 델타를 구독하는 앱이 이 메시지를 수신합니다.
3. 앱은 메시지에 응답하고 상태를 desired 값인 yellow로 업데이트합니다.
4. 그런 다음 앱은 디바이스 상태 yellow의 새로 보고된 값으로 업데이트 메시지를 게시합니다.

다음은 업데이트 요청이 게시되는 방식을 보여주는, 터미널에 표시되는 메시지입니다.

```
Enter desired value:
green
Changed local shadow value to 'green'.
Updating reported shadow value to 'green'...
Update request published.
Finished updating reported shadow value to 'green'.
```

AWS IoT 콘솔에서 섀도우 문서는 reported 및 desired 필드 모두에 green 대해 업데이트된 값을로 반영하고 버전 번호는 1씩 증가합니다. 예를 들어 이전 버전 번호가 10으로 표시된 경우 현재 버전 번호는 11로 표시됩니다.

> **Note**
>
> 섀도우를 삭제해도 버전 번호가 0으로 재설정되지는 않습니다. 업데이트 요청을 게시하거나 동일한 이름의 다른 섀도우를 만들 때 섀도우 버전이 1씩 증가하는 것을 볼 수 있습니다.

델타 이벤트를 관찰하기 위해 섀도우 문서 편집

shadow.py 샘플 앱도 delta 이벤트를 구독하고 desired 값이 변경되면 응답합니다. 예를 들어 desired 값을 red 색상으로 변경할 수 있습니다. 이렇게 하려면 AWS IoT 콘솔에서 편집을 클릭하여 섀도우 문서를 편집한 다음 desired 값을 로 유지하면서 JSONred에서 reported 값을 로 설정합니다green. 변경 사항을 저장하기 전에 Raspberry Pi에서 터미널을 열어 두면 변경 사항이 발생할 때 터미널에 메시지가 표시됩니다.

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

새 값을 저장한 후 shadow.py 샘플 앱은 이 변경에 응답하고 델타를 나타내는 메시지를 터미널에 표시합니다. 그러면 desired 값을 입력하라는 프롬프트 아래에 다음 메시지가 표시됩니다.

```
Enter desired value:
Received shadow delta event.
Delta reports that desired value is 'red'. Changing local value...
Changed local shadow value to 'red'.
Updating reported shadow value to 'red'...
Finished updating reported shadow value to 'red'.
Enter desired value:
Update request published.
Finished updating reported shadow value to 'red'.
```

2단계: MQTT 테스트 클라이언트에서 **shadow.py** 샘플 앱의 메시지 보기

AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하여 AWS 계정에서 전달되는 MQTT 메시지를 모니터링할 수 있습니다. 디바이스 섀도우 서비스에서 사용하는 예약된 MQTT 주제를 구독하면 샘플 앱을 실행할 때 주제에서 수신한 메시지를 관찰할 수 있습니다.

MQTT 테스트 클라이언트를 아직 사용하지 않았다면 MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기를 검토할 수 있습니다. 이는 MQTT 메시지가 메시지 브로커를 통과할 때 해당 메시지를 보기 위해 AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 사용하는 방법을 파악하는 데 도움이 됩니다.

1. MQTT 테스트 클라이언트 열기

   MQTT 테스트 클라이언트의 구성을 잃지 않고 MQTT 주제에서 수신한 메시지를 관찰할 수 있도록 새 창에서 AWS IoT 콘솔의 MQTT 테스트 클라이언트를 엽니다. MQTT 테스트 클라이언트는 콘솔의 다른 페이지로 이동하기 위해 나가면 구독 또는 메시지 로그를 유지하지 않습니다. 자습서의이 섹션에서는 AWS IoT 사물의 섀도우 문서와 MQTT 테스트 클라이언트를 별도의 창에서 열어 디바이스 섀도우와의 상호 작용을 보다 쉽게 관찰할 수 있습니다.

2. MQTT 예약 섀도우 주제 구독

   MQTT 테스트 클라이언트를 사용하여 디바이스 섀도우의 MQTT 예약 주제 이름을 입력하고 `shadow.py` 샘플 앱 실행 시 업데이트를 수신하도록 구독할 수 있습니다. 주제를 구독하려면:

   a. AWS IoT 콘솔의 MQTT 테스트 클라이언트에서 주제 구독(Subscribe to a topic)을 선택합니다.

   b. 주제 필터(Topic filter) 섹션에 $aws/things/***thingname***/shadow/update/#을 입력합니다. 여기서 `thingname`은 이전에 생성한 사물 리소스의 이름(예: `My_light_bulb`)입니다

   c. 추가 구성 설정에 대한 기본값을 유지한 후 구독(Subscribe)을 선택합니다.

   주제 구독에서 # 와일드카드를 사용하여 여러 MQTT 주제를 동시에 구독하고 디바이스와 섀도우 간에 교환되는 모든 메시지를 단일 창에서 관찰할 수 있습니다. 와일드카드 문자 및 해당 사용에 대한 자세한 내용은 MQTT 주제 단원을 참조하세요.

3. **`shadow.py` 샘플 프로그램 실행 및 메시지 관찰**

   Raspberry Pi의 명령줄 창에서 프로그램 연결을 끊었다면 샘플 앱을 다시 실행하고 AWS IoT 콘솔의 MQTT 테스트 클라이언트에서 메시지를 확인합니다.

   a. 다음 명령을 실행하여 샘플 프로그램을 다시 시작하세요. *`your-iot-thing-name`* 및 *`your-iot-endpoint`*를 이전에 생성한 AWS IoT 사물의 이름(예: `My_light_bulb`)과 디바이스와 상호 작용할 엔드포인트로 바꿉니다.

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples/service-clients
   python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
   device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
   thing_name your-iot-thing-name
   ```

   `shadow.py` 샘플 앱이 실행되고 현재 섀도우 상태가 검색됩니다. 섀도우를 삭제했거나 현재 상태를 지운 경우 프로그램은 현재 값을 `off`로 설정한 다음 `desired` 값을 입력하라는 메시지를 표시합니다.

   ```
   Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
    'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
   Connected!
   Subscribing to Delta events...
   Subscribing to Update responses...
   Subscribing to Get responses...
   ```

```
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow document lacks 'color' property. Setting defaults...
Changed local shadow value to 'off'.
Updating reported shadow value to 'off'...
Update request published.
Finished updating reported shadow value to 'off'...
Enter desired value:
```

반면에 프로그램이 실행 중이었고 다시 시작하면 터미널에 최신 색상 값이 보고됩니다. MQTT 테스트 클라이언트에서 $aws/things/***thingname***/shadow/get 및 $aws/things/***thingname***/shadow/get/accepted 주제에 대한 업데이트를 볼 수 있습니다.

보고된 최신 색상이 `green`이라고 가정합니다. 다음은 $aws/things/***thingname***/shadow/get/accepted JSON 파일의 내용을 보여줍니다.

```
{
"state": {
  "desired": {
    "welcome": "aws-iot",
    "color": "green"
  },
  "reported": {
    "welcome": "aws-iot",
    "color": "green"
  }
},
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
```

```
      "timestamp": 1620161643
    }
  }
},
"version": 10,
"timestamp": 1620173908
}
```

b. 터미널에 desired 값(예: yellow)을 입력합니다. shadow.py 샘플 앱은 reported 값이 yellow로 변경되었음을 보여주는 다음 메시지를 터미널에 응답으로 표시합니다.

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

AWS IoT 콘솔의 MQTT 테스트 클라이언트에서 구독 아래에 다음 주제가 메시지를 수신한 것이 표시됩니다.

- $aws/things/***thingname***/shadow/update: desired 및 updated 값이 모두 yellow 색상으로 변경되었음을 보여줍니다.
- $aws/things/***thingname***/shadow/update/accepted: desired 및 reported 상태의 현재 값과 해당 메타데이터 및 버전 정보를 보여줍니다.
- $aws/things/***thingname***/shadow/update/documents: desired 및 reported 상태의 이전 및 현재 값과 해당 메타데이터 및 버전 정보를 보여줍니다.

$aws/things/***thingname***/shadow/update/documents 문서에도 다른 두 주제에 포함된 정보가 있으므로 이를 검토하여 상태 정보를 볼 수 있습니다. 이전 상태는 green으로 설정된 보고 값, 해당 메타데이터 및 버전 정보, 보고 값이 yellow로 업데이트된 현재 상태를 보여줍니다.

```
{
"previous": {
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "green"
    },
```

```
    "reported": {
      "welcome": "aws-iot",
      "color": "green"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    }
  },
  "version": 10
},
"current": {
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "yellow"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "yellow"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
```

```
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
    }
  },
  "version": 11
},
"timestamp": 1617297904
}
```

c. 이제 다른 desired 값을 입력하면 이러한 주제에서 수신한 reported 값 및 메시지 업데이트에 대한 추가 변경 사항을 볼 수 있습니다. 버전 번호도 1씩 증가합니다. 예를 들어 값 green을 입력하면 이전 상태는 값 yellow를 보고하고 현재 상태는 값 green을 보고합니다.

4. 델타 이벤트를 관찰하기 위해 섀도우 문서 편집

델타 주제의 변경 내용을 관찰하려면 AWS IoT 콘솔의 섀도우 문서를 편집합니다. 예를 들어 desired 값을 red 색상으로 변경할 수 있습니다. 이렇게 하려면 AWS IoT 콘솔에서 편집을 선택한 다음 desired 값을 로 설정하면서 JSON에서 reported 값을 빨간색으로 설정합니다green. 변경 사항을 저장하기 전에 터미널에 보고된 델타 메시지가 표시되므로 터미널을 열어 두세요.

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

shadow.py 샘플 앱은 이 변경에 응답하고 델타를 나타내는 메시지를 터미널에 표시합니다. MQTT 테스트 클라이언트에서 update 주제는 desired 및 reported 값의 변경 사항을 보여주는 메시지를 수신합니다.

또한 $aws/things/***thingname***/shadow/update/delta 주제에서 메시지를 수신한 것을 볼 수 있습니다. 메시지를 보려면 구독(Subscriptions) 아래에 나열된 이 주제를 선택하세요.

```
{
"version": 13,
"timestamp": 1617318480,
"state": {
  "color": "red"
},
"metadata": {
  "color": {
    "timestamp": 1617318480
  }
}
}
```

3단계: 디바이스 섀도우 상호 작용 오류 문제 해결

섀도우 샘플 앱을 실행할 때 디바이스 섀도우 서비스와의 상호 작용을 관찰하는 데 문제가 발생할 수 있습니다.

프로그램이 성공적으로 실행되어 `desired` 값을 입력하라는 메시지가 표시되는 경우 앞에서 설명한 대로 섀도우 문서와 MQTT 테스트 클라이언트를 사용하여 디바이스 섀도우 상호 작용을 관찰할 수 있어야 합니다. 그러나 상호 작용이 표시되지 않는 경우 확인할 수 있는 몇 가지 사항은 다음과 같습니다.

- AWS IoT 콘솔에서 사물 이름과 해당 섀도우 확인

  섀도우 문서에 메시지가 표시되지 않으면 명령을 검토하고 AWS IoT 콘솔의 사물 이름과 일치하는지 확인하세요. 사물 리소스를 선택한 다음 섀도우(Shadows)를 선택하여 클래식 섀도우가 있는지 여부를 확인할 수도 있습니다. 이 자습서에서는 주로 클래식 섀도우와의 상호 작용에 중점을 둡니다.

  또한 사용한 디바이스가 인터넷에 연결되어 있는지 확인할 수도 있습니다. AWS IoT 콘솔에서 이전에 생성한 사물을 선택한 다음 상호 작용(Interact)을 선택합니다. 사물 세부 정보 페이지에 `This thing already appears to be connected.`와 같은 메시지가 표시됩니다.
- 구독한 MQTT 예약 주제 확인

  MQTT 테스트 클라이언트에 메시지가 표시되지 않으면 구독한 주제의 형식이 올바른지 확인합니다. MQTT 디바이스 섀도우 주제의 형식은 $aws/things/***thingname***/shadow/이며 섀도우에서 수행하려는 작업에 따라 `update`, `get` 또는 `delete`가 뒤따를 수 있습니다. 이 자습서에서는 $aws/

things/***thingname***/shadow/# 주제를 사용하므로 테스트 클라이언트의 주제 필터(Topic filter) 섹션에서 주제를 구독할 때 올바르게 입력했는지 확인하세요.

주제 이름을 입력할 때 ## ### 이전에 생성한 AWS IoT 사물의 이름과 동일한지 확인합니다. 또한 추가 MQTT 주제를 구독하여 업데이트가 성공적으로 수행되었는지 확인할 수 있습니다. 예를 들어 $aws/things/***thingname***/shadow/update/rejected 주제를 구독하면 업데이트 요청이 실패할 때마다 메시지를 수신하여 연결 문제를 디버깅할 수 있습니다. 예약된 주제에 대한 자세한 내용은 디바이스 섀도우 MQTT 주제 및 the section called “섀도우 주제” 단원을 참조하세요.

4단계: 결과 및 다음 단계 검토

이 자습서에서는 다음을 수행하는 방법을 알아보았습니다.

- shadow.py 샘플 앱을 사용하여 원하는 상태를 지정하고 섀도우의 현재 상태 업데이트.
- 섀도우 문서를 편집하여 델타 이벤트를 관찰하고 shadow.py 샘플 앱이 델타 이벤트에 응답하는 방식 관찰.
- MQTT 테스트 클라이언트를 사용하여 섀도우 주제를 구독하고 샘플 프로그램을 실행할 때 업데이트 관찰.

다음 단계

추가 MQTT 예약 주제를 구독하여 섀도우 애플리케이션에 대한 업데이트를 관찰할 수 있습니다. 예를 들어 $aws/things/***thingname***/shadow/update/accepted 주제만 구독하는 경우 업데이트가 성공적으로 수행되었을 때 현재 상태 정보만 표시됩니다.

또한 추가 섀도우 주제를 구독하여 문제를 디버그하거나 디바이스 섀도우 상호 작용에 대해 자세히 알아보고 디바이스 섀도우 상호 작용과 관련된 문제를 디버그할 수도 있습니다. 자세한 내용은 the section called “섀도우 주제” 및 디바이스 섀도우 MQTT 주제 단원을 참조하세요.

명명된 셰도우를 사용하거나 LED용 Raspberry Pi와 연결된 추가 하드웨어를 사용하여 애플리케이션을 확장하고 터미널에서 전송한 메시지를 사용하여 상태 변경을 관찰하도록 선택할 수도 있습니다.

디바이스 섀도우 서비스에 대한 자세한 내용과 디바이스, 앱 및 서비스에서 서비스를 사용하는 방법에 대한 자세한 내용은 AWS IoT 디바이스 섀도우 서비스, 디바이스에서 섀도우 사용 및 앱 및 서비스에서 섀도우 사용 단원을 참조하세요.

자습서:에 대한 사용자 지정 권한 부여자 생성 AWS IoT Core

이 자습서에서는 AWS CLI을(를) 사용하여 사용자 지정 인증을 만들고 유효성을 검사하고 사용하는 단계를 보여 줍니다. 선택적으로 이 자습서를 사용하여 HTTP 게시 API를 통해 AWS IoT Core 에 데이터를 전송할 수 있도록 Postman을 사용할 수 있습니다.

이 자습서에서는 권한 부여 및 인증 논리를 구현하는 샘플 Lambda 함수와 토큰 서명이 활성화된 상태에서 create-authorizer 호출을 사용하는 사용자 지정 권한 부여자를 생성하는 방법을 보여줍니다. 그런 다음 권한 부여자는를 사용하여 검증되며test-invoke-authorizer, 마지막으로 HTTP 게시 API를 AWS IoT Core 사용하여 테스트 MQTT 주제에 데이터를 전송할 수 있습니다. 샘플 요청은 `x-amz-customauthorizer-name` 헤더를 사용하여 호출할 권한 부여자를 지정하고, 요청 헤더에 토큰 키 이름과 `x-amz-customauthorizer-signature`을(를) 전달합니다.

이 자습서에서 배울 내용:

- Lambda 함수를 사용자 지정 권한 부여 처리기로 만드는 방법
- 토큰 서명이 활성화된 AWS CLI 를 사용하여 사용자 지정 권한 부여자를 생성하는 방법
- test-invoke-authorizer 명령을 사용하여 사용자 지정 권한 부여자를 테스트 하는 방법
- Postman을 사용하여 MQTT 항목을 게시하는 방법 및 사용자 정의 권한 부여자로 요청을 검증하는 방법

이 자습서는 완료하는 데 약 60분이 소요됩니다.

이 자습서에서 배울 내용은 다음과 같습니다.

- 1단계: 사용자 지정 권한 부여자를 위한 Lambda 함수 생성
- 2단계: 사용자 지정 권한 부여자를 위한 퍼블릭 및 프라이빗 키 페어 생성
- 3단계: 사용자 지정 권한 부여자 리소스 및 권한 부여 생성
- 4단계: test-invoke-authorizer를 호출하여 권한 부여자 테스트
- 5단계: Postman을 사용하여 MQTT 메시지 게시 테스트
- 6단계: MQTT 테스트 클라이언트에서 메시지 보기
- 7단계: 결과 및 다음 단계 검토
- 8단계: 정리

이 자습서를 시작하기 전에 다음 사항을 확인해야 합니다.

- 설정 AWS 계정

이 자습서를 완료하려면 AWS 계정 및 AWS IoT 콘솔이 필요합니다.

이 자습서에 사용하는 계정은 적어도 이러한 AWS 관리형 정책을 포함 할 때 가장 잘 작동합니다.

- IAMFullAccess
- AWSIoTFullAccess
- AWSLambda_FullAccess

> **Important**
>
> 이 자습서에서 사용된 IAM 정책은 프로덕션 구현에서 따라야 하는 것보다 권한이 많습니다. 프로덕션 환경에서는 계정 및 리소스 정책이 필요한 권한만 부여하도록 해야 합니다. 프로덕션용 IAM 정책을 생성할 때는 사용자 및 역할이 필요로 하는 액세스 수준을 결정한 다음 사용자들이 이러한 작업만 수행할 수 있도록 허용하는 정책을 설계합니다. 자세한 내용은 IAM의 보안 모범 사례를 참조하세요.

- 를 설치했습니다. AWS CLI

  설치 방법에 대한 자세한 내용은 CLI 설치를 AWS CLI참조하세요. AWS 이 자습서에는 AWS CLI 버전 `aws-cli/2.1.3 Python/3.7.4 Darwin/18.7.0 exe/x86_64` 이상이 필요합니다.

- OpenSSL 도구

  이 자습서의 예제에서는 LibreSSL 2.6.5를 사용합니다. 이 자습서를 위해서는 OpenSSL v1.1.1i 도구도 사용할 수 있습니다.

- AWS Lambda 개요 검토

  AWS Lambda 이전에를 사용한 적이 없는 경우 AWS Lambda 및 Lambda 시작하기를 검토하여 해당 용어와 개념을 알아봅니다.

- Postman에서 요청을 빌드하는 방법 검토

  자세한 내용은 요청 빌드 단원을 참조하세요.

- 이전 자습서의 사용자 지정 권한 부여자 제거

  에는 한 번에 제한된 수의 사용자 지정 권한 부여자만 구성할 AWS 계정 수 있습니다. 사용자 지정 권한 부여자를 제거하는 방법에 대한 자세한 내용은 the section called “8단계: 정리” 단원을 참조하세요.

1단계: 사용자 지정 권한 부여자를 위한 Lambda 함수 생성

의 사용자 지정 인증 AWS IoT Core 은 사용자가 생성한 권한 부여자 리소스를 사용하여 클라이언트를 인증하고 권한을 부여합니다. 이 섹션에서 생성할 함수는 클라이언트가 AWS IoT 리소스에 연결하고 액세스할 때 클라이언트를 인증 AWS IoT Core 하고 승인합니다.

Lambda 함수는 다음 작업을 수행합니다.

- test-invoke-authorizer에서 요청이 오는 경우 `Deny` 작업과 함께 IAM 정책을 반환합니다.
- 요청이 HTTP를 사용하여 Postma에서 오고 `actionToken` 파라미터 값이 `allow`이면 `Allow` 작업과 함께 IAM 정책을 반환합니다. 그렇지 않은 경우`Deny` 작업과 함께 IAM 정책을 반환합니다.

사용자 지정 권한 부여자에 대한 Lambda 함수를 생성하려면

1. Lambda 콘솔에서 함수를 엽니다.
2. 함수 생성을 선택합니다.
3. 처음부터 작성(Author from scratch)이 선택되어 있는지 확인합니다.
4. 기본 정보에서

   a. 함수 이름에 **`custom-auth-function`**을(를) 입력합니다.

   b. 런타임에서 Node.js 18.x를 확인합니다.
5. 함수 생성을 선택합니다.

   Lambda는 Node.js 함수와 로그 업로드 권한을 함수에 부여하는 실행 역할을 생성합니다. Lambda 함수는 함수를 호출할 때 실행 역할을 수임하고 실행 역할을 사용하여 AWS SDK에 대한 자격 증명을 생성하고 이벤트 소스에서 데이터를 읽습니다.
6. AWS Cloud9 편집기에서 함수의 코드 및 구성을 보려면 디자이너 창에서 custom-auth-function을 선택한 후 편집기 탐색 창에서 index.js를 선택합니다.

   Node.js 같은 스크립트 언어의 경우 Lambda는 성공 응답을 반환하는 기본 함수를 포함합니다. 소스 코드가 3MB를 초과하지 않는 한 AWS Cloud9 편집기를 사용하여 함수를 편집할 수 있습니다.
7. 다음 코드를 사용하여 편집기에서 index.js 코드를 대체합니다.

```
// A simple Lambda function for an authorizer. It demonstrates
// How to parse a CLI and Http password to generate a response.

export const handler = async (event, context, callback) => {
```

```
    //Http parameter to initiate allow/deny request
    const HTTP_PARAM_NAME='actionToken';
    const ALLOW_ACTION = 'Allow';
    const DENY_ACTION = 'Deny';

    //Event data passed to Lambda function
    var event_str = JSON.stringify(event);
    console.log('Complete event :'+ event_str);

    //Read protocolData from the event json passed to Lambda function
    var protocolData = event.protocolData;
    console.log('protocolData value---> ' + protocolData);

    //Get the dynamic account ID from function's ARN to be used
    // as full resource for IAM policy
    var ACCOUNT_ID = context.invokedFunctionArn.split(":")[4];
    console.log("ACCOUNT_ID---"+ACCOUNT_ID);

    //Get the dynamic region from function's ARN to be used
    // as full resource for IAM policy
    var REGION = context.invokedFunctionArn.split(":")[3];
    console.log("REGION---"+REGION);

    //protocolData data will be undefined if testing is done via CLI.
    // This will help to test the set up.
    if (protocolData === undefined) {

        //If CLI testing, pass deny action as this is for testing purpose only.
        console.log('Using the test-invoke-authorizer cli for testing only');
        callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

    } else{

        //Http Testing from Postman
        //Get the query string from the request
        var queryString = event.protocolData.http.queryString;
        console.log('queryString values -- ' + queryString);
        /*         global URLSearchParams       */
        const params = new URLSearchParams(queryString);
        var action = params.get(HTTP_PARAM_NAME);

        if(action!=null && action.toLowerCase() === 'allow'){

            callback(null, generateAuthResponse(ALLOW_ACTION,ACCOUNT_ID,REGION));
```

```
        }else{

            callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

        }

    }

};

// Helper function to generate the authorization IAM response.
var generateAuthResponse = function(effect,ACCOUNT_ID,REGION) {

    var full_resource = "arn:aws:iot:"+ REGION + ":" + ACCOUNT_ID + ":*";
    console.log("full_resource---"+full_resource);

    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'principalId';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var statement = {};
    statement.Action = 'iot:*';
    statement.Effect = effect;
    statement.Resource = full_resource;
    policyDocument.Statement[0] = statement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 600;

    console.log('custom auth policy function called from http');
    console.log('authResponse --> ' + JSON.stringify(authResponse));
    console.log(authResponse.policyDocuments[0]);

    return authResponse;
}
```

8. 배포를 선택합니다.
9. 편집기 위에 변경 사항 배포됨(Changes deployed)이 나타나면

a. 편집기 위의 함수 개요(Function overview) 섹션으로 스크롤합니다.

b. 이 자습서의 후반부에서 사용할 수 있도록 함수 ARN을 복사하여 저장해 둡니다.

10. 함수를 테스트합니다.

a. 테스트 탭을 선택합니다.

b. 기본 테스트 설정을 사용하여 호출(Invoke)을 선택합니다.

c. 테스트가 성공한 경우 실행 결과(Execution results)에서 세부 정보(Details) 보기를 엽니다. 함수가 반환한 정책 문서가 표시되어야 합니다.

테스트에 실패했거나 정책 문서가 표시되지 않으면 코드를 검토하여 오류를 찾아 수정합니다.

2단계: 사용자 지정 권한 부여자를 위한 퍼블릭 및 프라이빗 키 페어 생성

사용자 지정 권한 부여자를 인증하려면 퍼블릭 및 프라이빗 키가 필요합니다. 이 섹션의 명령은 OpenSSL 도구를 사용하여 이 키 페어를 생성합니다.

사용자 지정 권한 부여자를 위한 퍼블릭 및 프라이빗 키 페어를 생성하려면

1. 프라이빗 키 파일을 생성합니다.

```
openssl genrsa -out private-key.pem 4096
```

2. 방금 생성한 프라이빗 키 파일을 확인합니다.

```
openssl rsa -check -in private-key.pem -noout
```

명령에 오류가 표시되지 않으면 프라이빗 키 파일이 유효합니다.

3. 퍼블릭 키 파일을 생성합니다.

```
openssl rsa -in private-key.pem -pubout -out public-key.pem
```

4. 퍼블릭 키 파일을 확인합니다.

```
openssl pkey -inform PEM -pubin -in public-key.pem -noout
```

명령에 오류가 표시되지 않으면 퍼블릭 키 파일이 유효합니다.

3단계: 사용자 지정 권한 부여자 리소스 및 권한 부여 생성

AWS IoT 사용자 지정 권한 부여자는 이전 단계에서 생성된 모든 요소를 연결하는 리소스입니다. 이 단원에서는 사용자 지정 권한 부여자 리소스를 만들고 이전에 만든 Lambda 함수를 실행할 수 있는 권한을 부여합니다. AWS IoT 콘솔, AWS CLI또는 API를 사용하여 사용자 지정 권한 부여자 리소스를 생성할 수 있습니다 AWS .

이 자습서에서는 사용자 지정 권한 부여자를 하나만 생성하면 됩니다. 이 섹션에서는 AWS IoT 콘솔과를 사용하여를 생성하는 방법을 설명 AWS CLI하므로 가장 편리한 방법을 사용할 수 있습니다. 두 가지 방법으로 생성된 사용자 지정 권한 부여자 리소스 간에는 차이가 없습니다.

사용자 지정 권한 부여자 리소스 생성

다음 옵션 중 하나를 선택하여 사용자 지정 권한 부여자 리소스를 생성합니다.

- AWS IoT 콘솔을 사용하여 사용자 지정 권한 부여자 생성
- AWS CLI를 사용하여 사용자 지정 권한 부여자 생성

사용자 지정 권한 부여자를 생성하려면(콘솔)

1. AWS IoT 콘솔의 사용자 지정 권한 부여자 페이지를 열고 권한 부여자 생성을 선택합니다.
2. 권한 부여자에서 다음 작업을 수행합니다.
   a. 권한 부여자 이름에 **my-new-authorizer**를 입력합니다.
   b. 권한 부여자 상태에서 활성을 선택합니다.
   c. 권한 부여자 함수에서 앞서 생성한 Lambda 함수를 선택합니다.
   d. 토큰 검증 - 선택 사항에서
      i. 토큰 검증을 활성화합니다.
      ii. 토큰 키 이름에 **tokenKeyName**을 입력합니다.
      iii. 키 추가를 선택합니다.
      iv. 키 이름에 **FirstKey**를 입력합니다.
      v. 퍼블릭 키에 `public-key.pem` 파일의 내용을 입력합니다. 파일로부터 `-----BEGIN PUBLIC KEY-----` 및 `-----END PUBLIC KEY-----`이(가) 있는 행을 포함해야 하며, 파일 내용에서 줄 바꿈, 캐리지 리턴 또는 기타 문자를 추가하거나 제거하지 마세요. 입력한 문자열은 이 예제와 유사해야 합니다.

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----
```

3. 권한 부여자 생성을 선택합니다.
4. 사용자 지정 권한 부여자 리소스가 생성된 경우 사용자 지정 권한 부여자 목록이 표시되고 새 사용자 지정 권한 부여자가 목록에 나타나며 다음 섹션으로 이동하여 테스트할 수 있습니다.

   오류가 표시되면 오류를 검토하고 사용자 지정 권한 부여자를 다시 만들고 항목을 다시 확인합니다. 각 사용자 지정 권한 부여자 리소스에는 고유한 이름이 있어야 합니다.

사용자 지정 권한 부여자를 생성하려면(AWS CLI)

1. 값을 `authorizer-function-arn` 및 `token-signing-public-keys`(으)로 대체한 후 다음 명령을 실행합니다.

```
aws iot create-authorizer \
--authorizer-name "my-new-authorizer" \
--token-key-name "tokenKeyName" \
--status ACTIVE \
--no-signing-disabled \
--authorizer-function-arn "arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function" \
--token-signing-public-keys FirstKey="-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
```

```
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----"
```

위치:

- `authorizer-function-arn` 값은 사용자 지정 권한 부여자에 대해 생성한 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.
- `token-signing-public-keys` 값은 키 이름, **`FirstKey`** 및 `public-key.pem` 파일의 내용을 포함합니다. 파일로부터 `-----BEGIN PUBLIC KEY-----` 및 `-----END PUBLIC KEY-----`이(가) 있는 행을 포함해야 하며, 파일 내용에서 줄 바꿈, 캐리지 리턴 또는 기타 문자를 추가하거나 제거하지 마세요.

  참고: 퍼블릭 키 값을 변경하면 퍼블릭 키를 사용할 수 없게 되므로 퍼블릭 키를 입력하는 것은 주의해야 합니다.

2. 사용자 지정 권한 부여자가 생성되면 명령은 다음과 같은 새 리소스의 이름 및 ARN을 반환합니다.

```
{
    "authorizerName": "my-new-authorizer",
    "authorizerArn": "arn:aws:iot:Region:57EXAMPLE833:authorizer/my-new-authorizer"
}
```

다음 단계에서 사용하도록 `authorizerArn` 값을 저장합니다.

각 사용자 지정 권한 부여자 리소스에는 고유한 이름이 있어야 합니다.

사용자 지정 권한 부여자 리소스 권한 부여하기

이 섹션에서는 방금 생성한 사용자 지정 권한 부여자 리소스에 Lambda 함수를 실행할 수 있는 권한을 부여합니다. 권한을 부여하려면 add-permission CLI 명령을 사용할 수 있습니다.

를 사용하여 Lambda 함수에 권한 부여 AWS CLI

1. 값을 삽입한 후 다음 명령을 입력합니다. 참고: `statement-id` 값은 고유해야 합니다. 이전에 이 자습서를 실행한 적이 있거나 `ResourceConflictException` 오류가 발생하면, *Id-1234*를 다른 값으로 대체합니다.

```
aws lambda add-permission  \
--function-name "custom-auth-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn authorizerArn
```

2. 명령이 성공하면 이 예제와 같은 권한 문이 반환됩니다. 다음 단원을 계속하여 사용자 지정 권한 부여자를 테스트할 수 있습니다.

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
}
```

명령이 성공하지 못하면 이 예제와 같은 오류가 반환됩니다. 계속하기 전에 오류를 검토하고 수정해야 합니다.

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

4단계: test-invoke-authorizer를 호출하여 권한 부여자 테스트

이 단원에서 정의된 모든 리소스를 사용해 명령줄에서 test-invoke-authorizer를 호출하여 권한 부여 성공 여부를 테스트합니다.

명령줄에서 권한 부여자를 호출할 때 protocolData이(가) 정의되지 않았으므로 권한 부여자는 항상 DENY 문서를 반환합니다. 그러나 이 테스트는 Lambda 함수를 완전히 테스트하지 않더라도 사용자 지정 권한 부여자와 Lambda 함수가 올바르게 구성되었는지 확인합니다.

를 사용하여 사용자 지정 권한 부여자와 Lambda 함수를 테스트하려면 AWS CLI

1. 이전 단계에서 생성한 private-key.pem 파일이 있는 디렉터리에서 다음 명령을 실행합니다.

```
echo -n "tokenKeyValue" | openssl dgst -sha256 -sign private-key.pem | openssl
 base64 -A
```

이 명령은 다음 단계에서 사용할 서명 문자열을 만듭니다. 서명 문자열은 다음과 같은 형태입니다.

```
dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V
+MMWu09YSA86+64Y3Gt4tOykpZqn9mn
VB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeeh
bQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjj
szEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY
+d1vTvdthKtYHBq8MjhzJ0kggbt29V
QJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuX
f3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+K
EWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFH
xRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

이 서명 문자열을 복사하여 다음 단계에서 사용합니다. 여분의 문자를 포함하거나 남겨 두지 않도록 합니다.

2. 이 명령에서 token-signature 값을 이전 단계의 서명 문자열로 바꾸고 이 명령을 실행하여 권한 부여자를 테스트합니다.

```
aws iot test-invoke-authorizer \
--authorizer-name my-new-authorizer \
--token tokenKeyValue \
--token-signature dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr
+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp
+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsH
+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/
```

```
P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuXf3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o
+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

명령이 성공적으로 실행되는 경우 이 예와 같이 사용자 지정 권한 부여자 함수가 생성한 정보가 반환됩니다.

```
{
    "isAuthenticated": true,
    "principalId": "principalId",
    "policyDocuments": [
        "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Action\":\"iot:*\",
\"Effect\":\"Deny\",\"Resource\":\"arn:aws:iot:Region:57EXAMPLE833:*\"}]}"
    ],
    "refreshAfterInSeconds": 600,
    "disconnectAfterInSeconds": 3600
}
```

명령이 오류를 반환하면 오류를 검토하고 이 단원에서 사용한 명령을 다시 확인합니다.

5단계: Postman을 사용하여 MQTT 메시지 게시 테스트

1. 명령줄에서 디바이스 데이터 엔드포인트를 가져오려면 여기에 표시된 대로 describe-endpoint를 호출합니다.

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

   이후 단계에서 *device_data_endpoint_address*로 사용할 수 있도록 이 주소를 저장합니다.

2. 새 Postman 창을 열고 새 HTTP POST 요청을 만듭니다.

   a. 컴퓨터에서 Postman 앱을 엽니다.

   b. Postman의 파일 메뉴에서 새로 만들기...를 선택합니다.

   c. 새로 만들기(New) 대화 상자에서 요청(Request)을 선택합니다.

   d. 저장 요청에서

      i. 요청 이름에서 **Custom authorizer test request**을(를) 입력합니다.

ii. 저장할 컬렉션 또는 폴더 선택:에서 이 요청을 저장할 컬렉션을 선택하거나 만듭니다.

iii. ***collection_name***에 저장을 선택합니다.

3. POST 요청을 생성하여 사용자 지정 권한 부여자를 테스트합니다.

a. URL 필드 옆의 요청 메소드 선택기에서 POST를 선택합니다.

b. URL 필드에서 이전 단계의 describe-endpoint 명령의 *device_data_endpoint_address*와 함께 다음 URL을 사용하여 요청에 대한 URL을 생성합니다.

```
https://device_data_endpoint_address:443/topics/test/cust-auth/topic?
qos=0&actionToken=allow
```

이 URL에는 AWS IoT에 대한 액세스를 허용하는 정책 문서를 반환하도록 Lambda 함수에 지시하는 `actionToken=allow` 쿼리 파라미터가 포함됩니다. URL을 입력한 후 쿼리 파라미터가 Postman의 Params 탭에도 나타납니다.

c. Auth 탭의 유형(Type) 필드에서 No Auth를 선택합니다.

d. 헤더(Headers) 탭에서:

i. 호스트 키가 선택되어 있으면 이 키를 선택 취소하세요.

ii. 헤더 목록의 맨 아래에 이러한 새 헤더를 추가하고 새 헤더가 선택되었는지 확인합니다. **Host** 값을 *device_data_endpoint_address*로 바꾸고, **x-amz-customauthorizer-signature** 값을 이전 단원에서 test-invoke-authorize 명령과 함께 사용한 서명 문자열로 바꿉니다.

Key(키)	값
x-amz-customauthorizer-name	**my-new-authorizer**
Host	*device_data_endpoint_address*
tokenKeyName	**tokenKeyValue**
x-amz-customauthorizer-signature	*dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp+0bDZh8hmqUAU*

Key(키)	값
	H3fwi3fPjBvCa4cwNuLQNqBZzbC vsluv7i2IMjEg+CPY0zrWt1jr9B ikgGPDxWkjaeehbQHHTo357TegK s9pP30Uf4TrxypNmFswA5k7QIc0 1n4bIyRTm90OyZ94R4bdJsHNig1 JePgnuOBvMGCEFE09jGjjszEHfg AUAQIWXiVGQj16BU1xKpTGSiTAw heLKUjITOEXAMPLECK3aHKYKY+d 1vTvdthKtYHBq8MjhzJ0kggbt29 VQJCb8RilN/P5+vcVniSXWPplyB 5jkYs9UvG08REoy64AtizfUhvSu l/r/F3VV8ITtQp3aXiUtcspACi6 ca+tsDuXf3LzCwQQF/YSUy02u5X kWn+sto6KCkpNlkD0wU8gl3+k0z xrthnQ8gEajd5Iylx230iqcXo3o sjPha7JDyWM5o+KEWckTe91I1mo kDr5sJ4JXixvnJTVSx1li49IalW 4en1DAkc1a0s2U2UNm236EXAMPL ELotyh7h+flFeloZlAWQFHxRlXs PqiVKS1ZIUClaZWprh/orDJplpi WfBgBIOgokJIDGP9gwhXIIk7zWr GmWpMK9o=

e. 본문(Body) 탭에서:

i. 데이터 형식 옵션 상자에서 원시(Raw)를 선택합니다.

ii. 데이터 유형 목록에서 JavaScript를 선택합니다.

iii. 텍스트 필드에서 테스트 메시지에 대해 다음 JSON 메시지 페이로드를 입력합니다.

```
{
    "data_mode": "test",
    "vibration": 200,
    "temperature": 40
}
```

4. 전송(Send)을 선택하여 요청을 전송합니다.

요청이 성공적이면 다음을 반환합니다.

```
{
    "message": "OK",
    "traceId": "ff35c33f-409a-ea90-b06f-fbEXAMPLE25c"
}
```

응답이 성공하면 사용자 지정 권한 부여자가에 대한 연결을 허용 AWS IoT 했고 테스트 메시지가의 브로커에 전송되었음을 나타냅니다 AWS IoT Core.

오류가 반환되면 오류 메시지 *device_data_endpoint_address*, 서명 문자열 및 기타 헤더 값이 포함됩니다.

다음 단원에서 사용하도록 이 요청을 Postman에 유지합니다.

6단계: MQTT 테스트 클라이언트에서 메시지 보기

이전 단계에서는 Postman을 사용하여 시뮬레이션된 디바이스 메시지를 AWS IoT 에 보냈습니다. 성공 응답은 사용자 지정 권한 부여자가 AWS IoT 에 대한 연결을 허용했으며 서 테스트 메시지가 AWS IoT Core의 브로커에게 전달되었음을 나타냅니다. 이 섹션에서는 AWS IoT 콘솔의 MQTT 테스트 클라이언트를 사용하여 다른 디바이스 및 서비스와 마찬가지로 해당 메시지의 메시지 내용을 확인합니다.

사용자 지정 권한 부여자가 승인한 테스트 메시지를 보려면

1. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.
2. 주제 구독(Subscribe to topic) 탭의 주제 필터(Topic filter)에 **test/cust-auth/topic**을(를) 입력합니다. 이는 이전 단원의 Postman 예제에서 사용된 메시지 주제입니다.
3. 구독을 선택합니다.

   이 창은 다음 단계를 위해 계속 표시되도록 합니다.
4. Postman에서 이전 섹션에 대해 만든 요청에서 전송(Send)을 선택합니다.

   응답을 검토하여 성공했는지 확인합니다. 그렇지 않은 경우 이전 단원에서 설명한 대로 오류 문제를 해결하세요.
5. MQTT 테스트 클라이언트에서 메시지 주제를 표시하는 새 항목이 표시되고, 확장되면 Postman에서 전송한 요청의 메시지 페이로드가 표시됩니다.

   MQTT 테스트 클라이언트에서 메시지가 표시되지 않으면 다음 몇 가지를 확인해야 합니다.

- Postman 요청이 성공적으로 반환되었는지 확인하세요. 가 연결을 AWS IoT 거부하고 오류를 반환하면 요청의 메시지가 메시지 브로커로 전달되지 않습니다.
- AWS IoT 콘솔을 여는 데 AWS 리전 사용되는 AWS 계정 및가 Postman URL에서 사용 중인와 동일한지 확인합니다.
- 사용자 지정 권한 부여자에 적절한 엔드포인트를 사용하고 있는지 확인합니다. 기본 IoT 엔드포인트는 Lambda 함수와 함께 사용자 지정 권한 부여자를 사용하도록 지원하지 않을 수 있습니다. 그 대신, 도메인 구성을 사용하여 새 엔드포인트를 정의한 다음, 사용자 지정 권한 부여자에 대해 해당 엔드포인트를 지정할 수 있습니다.
- MQTT 테스트 클라이언트에서 주제를 올바르게 입력했는지 확인하세요. 주제 필터는 대소문자를 구분하지 않습니다. 의심스러운 경우 주제를 구독할 수도 있습니다.이 **#** 주제는 메시지 브로커를 통과 AWS 계정 하고 콘솔을 여는 데 AWS 리전 사용되는 모든 MQTT 메시지를 구독합니다 AWS IoT .

7단계: 결과 및 다음 단계 검토

이 자습서에서는:

- Lambda 함수를 사용자 지정 권한 부여자 처리기로 만들었습니다.
- 토큰 서명이 활성화된 상태에서 사용자 지정 권한 부여자를 만들었습니다.
- test-invoke-authorizer 명령을 사용하여 사용자 지정 권한 부여자를 테스트했습니다.
- Postman을 사용하여 MQTT 주제를 게시했고 사용자 지정 권한 부여자로 요청을 검증합니다.
- MQTT 테스트 클라이언트를 사용하여 Postman 테스트에서 전송한 메시지를 보았습니다.

다음 단계

Postman에서 일부 메시지를 전송해 사용자 지정 권한 부여자가 작동하는지 확인한 후 이 자습서의 다양한 측면을 변경하면 결과에 어떤 영향을 미치는지 실험해 보세요. 다음은 시작하는 데 도움이 될 몇 가지 예제입니다.

- 더 이상 유효하지 않게 서명 문자열을 변경하여 무단 연결 시도가 어떻게 처리되는지 확인합니다. 이 오류 응답과 같은 오류 응답이 나타나야 하며 MQTT 테스트 클라이언트에 메시지가 나타나지 않아야 합니다.

```
{
    "message": "Forbidden",
```

```
    "traceId": "15969756-a4a4-917c-b47a-5433e25b1356"
}
```

- AWS IoT 규칙을 개발하고 사용하는 동안 발생할 수 있는 오류를 찾는 방법에 대한 자세한 내용은 섹션을 참조하세요모니터링 AWS IoT.

8단계: 정리

이 자습서를 반복하려면 일부 사용자 지정 권한 부여자를 제거해야 할 수도 있습니다. 는 한 번에 제한된 수의 사용자 지정 권한 부여자만 구성할 AWS 계정 수 있으며 기존 사용자 지정 권한 부여자를 제거하지 않고 새 권한 부여자를 추가하려고 할 `LimitExceededException` 때를 가져올 수 있습니다.

사용자 지정 권한 부여자를 제거하려면(콘솔)

1. AWS IoT 콘솔의 사용자 지정 권한 부여자 페이지를 열고 사용자 지정 권한 부여자 목록에서 제거할 사용자 지정 권한 부여자를 찾습니다.
2. 사용자 지정 권한 부여자 세부 정보 페이지를 열고 작업 메뉴에서 편집을 선택합니다.
3. 권한 부여자 활성화의 선택을 취소한 다음 업데이트를 선택합니다.

   활성 상태인 동안에는 사용자 지정 권한 부여자를 삭제할 수 없습니다.
4. 사용자 지정 권한 부여자 세부 정보 페이지에서 작업 메뉴를 열고 삭제를 선택합니다.

사용자 지정 권한 부여자를 제거하려면(AWS CLI)

1. 설치한 사용자 지정 권한 부여자를 나열하고 삭제할 사용자 지정 권한 부여자의 이름을 찾습니다.

   ```
   aws iot list-authorizers
   ```
2. *Custom_Auth_Name*을 삭제할 사용자 지정 권한 부여자의 `authorizerName`으로 바꾼 후 이 명령을 실행하여, 사용자 지정 권한 부여자를 `inactive`로 설정합니다.

   ```
   aws iot update-authorizer --status INACTIVE --authorizer-name Custom_Auth_Name
   ```
3. *Custom_Auth_Name*을 삭제할 사용자 지정 권한 부여자의 `authorizerName`으로 바꾼 후 이 명령을 실행하여 사용자 지정 권한 부여자를 삭제합니다.

   ```
   aws iot delete-authorizer --authorizer-name Custom_Auth_Name
   ```

자습서: AWS IoT 및 Raspberry Pi를 사용하여 토양 수분 모니터링

이 자습서에서는 Raspberry Pi, 습도 센서를 사용하고 주택 식물 또는 정원의 토양 습도 수준을 AWS IoT 모니터링하는 방법을 보여줍니다. Raspberry Pi는 센서에서 습도 수준과 온도를 읽은 다음 데이터를 보내는 코드를 실행합니다 AWS IoT. 습도 수준이 임계값 아래로 떨어질 때 Amazon SNS 주제를 구독하는 주소로 이메일을 AWS IoT 보내는 규칙을에서 생성합니다.

> **Note**
>
> 이 튜토리얼은 최신이 아닐 수 있습니다. 이 주제가 처음 게시된 이후로 일부 참조가 대체되었을 수 있습니다.

목차

- 사전 조건
- 설 AWS IoT정
  - 1단계: AWS IoT 정책 생성
  - 2단계: AWS IoT 사물, 인증서 및 프라이빗 키 생성
  - 3단계: Amazon SNS 주제 생성 및 구독
  - 4단계: 이메일 전송 AWS IoT 규칙 생성
- Raspberry Pi와 습도 센서 설정

사전 조건

이 자습서를 완료하려면 다음이 필요합니다.

- AWS 계정.
- 관리자 권한이 있는 IAM 사용자.
- Windows, macOS, Linux 또는 Unix를 실행하며 AWS IoT 콘솔에 액세스할 수 있는 개발 컴퓨터.
- 최신 Raspberry Pi OS를 실행하는 Raspberry Pi 3B 또는 4B. 설치 방법은 Rasberry Pi 웹사이트의 운영 체제 설치 섹션을 참조하세요.
- Raspberry Pi용 모니터, 키보드, 마우스 및 Wi-Fi 네트워크 또는 이더넷 연결.
- Raspberry Pi 호환 습도 센서.. 이 자습서에서 사용되는 센서는 JST 4핀 및 암 소켓 케이블 헤더가 있는 Adafruit STEMMA I2C 정전식 습도 센서입니다.

설 AWS IoT정

이 자습서를 완료하려면 다음 리소스를 생성해야 합니다. 디바이스를 연결하려면 IoT 사물 AWS IoT, 디바이스 인증서 및 AWS IoT 정책을 생성합니다.

- AWS IoT 사물입니다.

  사물이란 물리적 디바이스(이 경우는 Rasberry Pi)를 의미하며, 그 안에는 디바이스에 관한 정적 메타데이터가 들어 있습니다.
- 디바이스 인증서입니다.

  모든 디바이스에는 AWS IoT와 연결하여 인증하는 디바이스 인증서가 있어야 합니다.
- AWS IoT 정책.

  각 디바이스 인증서에는 하나 이상의 AWS IoT 정책이 연결되어 있습니다. 이러한 정책은 디바이스가 액세스할 수 있는 AWS IoT 리소스를 결정합니다.
- AWS IoT 루트 CA 인증서입니다.

  디바이스 및 기타 클라이언트는 AWS IoT 루트 CA 인증서를 사용하여 통신 중인 AWS IoT 서버를 인증합니다. 자세한 내용은 서버 인증 단원을 참조하십시오.
- AWS IoT 규칙입니다.

  규칙에는 쿼리와 하나 이상의 규칙 작업이 있습니다. 쿼리는 디바이스 메시지에서 데이터를 추출하여 메시지 데이터를 처리해야 할지 여부를 결정합니다. 규칙 작업은 데이터가 쿼리와 일치할 경우에 해야 할 일을 지정합니다.
- Amazon SNS 주제 및 주제 구독

  규칙은 Raspberry Pi의 습도 데이터를 모니터링합니다. 값이 임계값 미만일 경우에는 Amazon SNS 주제로 메시지를 전송합니다. Amazon SNS에서 주제를 구독하는 모든 이메일 주소로 메시지를 전송합니다.

1단계: AWS IoT 정책 생성

Raspberry Pi가 연결하고 메시지를 보낼 수 있도록 허용하는 AWS IoT 정책을 생성합니다 AWS IoT.

1. AWS IoT 콘솔에서 시작하기 버튼이 표시되면 선택합니다. 또는 탐색 창에서 Security(보안)를 확장한 후 Policies(정책)를 선택합니다.

2. You don't have any policies yet(아직 정책이 없습니다) 대화 상자가 나타나면 Create a policy(정책 생성)를 선택합니다. 그렇지 않은 경우, 생성을 선택합니다.
3. AWS IoT 정책의 이름을 입력합니다(예: **MoistureSensorPolicy**).
4. 설명문 추가 부분에서 기존 정책을 다음 JSON으로 바꿉니다. ## 및 ### AWS 리전 및 AWS 계정 번호로 바꿉니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/RaspberryPi"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete/rejected"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/delete/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/RaspberryPi"
        }
    ]
}
```

5. 생성(Create)을 선택합니다.

2단계: AWS IoT 사물, 인증서 및 프라이빗 키 생성

AWS IoT 레지스트리에 Raspberry Pi를 나타내는 사물을 생성합니다.

1. AWS IoT 콘솔 탐색 창에서 관리를 선택한 후 사물을 선택합니다.
2. You don't have any things yet(아직 사물이 없습니다) 대화 상자가 표시되면 Register a thing(사물 등록)을 선택합니다. 그렇지 않은 경우, 생성을 선택합니다.
3. AWS IoT 사물 생성 페이지에서 단일 사물 생성을 선택합니다.

4. 디바이스를 디바이스 레지스트리에 추가 페이지에서 IoT 사물 이름(예: **RaspberryPi**)을 입력하고 다음을 선택합니다. 사물을 생성한 후에는 사물 이름을 변경할 수 없습니다. 사물 이름을 변경하려면 새 사물을 생성하고 새 이름을 지정한 다음 이전 사물을 삭제해야 합니다.
5. 사물에 인증서 추가 페이지에서 Create certificate(인증서 생성)를 선택하세요.
6. 다운로드 링크를 선택하여 인증서, 프라이빗 키 및 루트 CA 인증서를 다운로드합니다.

> **Important**
>
> 이 때에만 인증서와 프라이빗 키를 다운로드할 수 있습니다.

7. 인증서를 활성화하려면 활성화를 선택합니다. 디바이스를 AWS IoT에 연결하려면 인증서가 활성 상태여야 합니다.
8. 정책 연결을 선택합니다.
9. Add a policy for your thing(사물에 정책 추가)은 MoistureSensorPolicy를 선택하고 사물 등록을 선택합니다.

3단계: Amazon SNS 주제 생성 및 구독

Amazon SNS 주제 생성 및 구독

1. AWS SNS 콘솔의 탐색 창에서 주제를 선택한 다음 주제 생성을 선택합니다.
2. 유형을 표준으로 선택하고 주제의 이름(예: **MoistureSensorTopic**)을 입력합니다.
3. 주제의 표시 이름을 입력합니다(예: **Moisture Sensor Topic**). 이것이 Amazon SNS 콘솔에서 표시되는 주제의 이름입니다.
4. 주제 생성을 선택합니다.
5. Amazon SNS 주제 세부 정보 페이지에서 구독 생성을 선택합니다.
6. 프로토콜에서 이메일을 선택합니다.
7. 엔드포인트에 이메일 주소를 입력합니다.
8. 구독 생성을 선택합니다.
9. 이메일 클라이언트를 열고 제목이 **MoistureSensorTopic**인 메시지를 찾습니다. 이메일을 열고 구독 확인 링크를 클릭합니다.

> **Important**
>
> 구독을 확인할 때까지 이 Amazon SNS 주제에서 이메일 알림을 수신할 수 없습니다.

입력한 텍스트가 포함된 이메일 메시지를 수신해야 합니다.

4단계: 이메일 전송 AWS IoT 규칙 생성

AWS IoT 규칙은 디바이스에서 메시지를 수신할 때 수행할 쿼리와 하나 이상의 작업을 정의합니다. AWS IoT 규칙 엔진은 디바이스에서 보낸 메시지를 수신하고 메시지의 데이터를 사용하여 일부 조치를 취해야 하는지 여부를 결정합니다. 자세한 내용은 에 대한 규칙 AWS IoT 단원을 참조하십시오.

이 자습서에서는 Raspberry Pi가 `aws/things/RaspberryPi/shadow/update`에 메시지를 게시합니다. 이것은 디바이스와 Thing Shadow 서비스에서 사용하는 내부 MQTT 주제입니다. Raspberry Pi는 다음 형식의 메시지를 게시합니다.

```
{
    "reported": {
        "moisture" : moisture-reading,
        "temp" : temperature-reading
    }
}
```

수신되는 메시지에서 습도와 온도 데이터를 추출하는 쿼리를 생성합니다. 그리고 데이터를 가져와 습도 값이 임계값 미만일 경우 Amazon SNS 주제 구독자에게 전송하는 Amazon SNS 작업도 만듭니다.

Amazon SNS 규칙 생성

1. AWS IoT 콘솔에서 메시지 라우팅을 선택한 다음 규칙을 선택합니다. 아직 규칙이 없습니다 대화상자가 나타나면 규칙 생성을 선택합니다. 또는 규칙 생성을 선택합니다.
2. 규칙 속성 페이지에서 규칙 이름(예: **`MoistureSensorRule`**)을 입력하고 간단한 규칙 설명(예: **`Sends an alert when soil moisture level readings are too low`**)을 제공합니다.
3. 다음을 선택하고 SQL 문을 구성합니다. SQL 버전을 2016-03-23로 선택하고 다음 AWS IoT SQL 쿼리 문을 입력합니다.

   ```
   SELECT * FROM '$aws/things/RaspberryPi/shadow/update/accepted' WHERE
    state.reported.moisture < 400
   ```

   이 설명문은 `moisture` 값이 `400` 미만일 때 규칙 작업을 트리거합니다.

> **Note**
>
> 다른 값을 사용해야 할 수 있습니다. Raspberry Pi에서 코드를 실행한 후에 센서를 터치하거나, 물 속에 넣거나, 화분에 두면 센서에서 가져온 값을 볼 수 있습니다.

4. 다음을 선택하고 규칙 작업을 첨부합니다. 작업 1에서 단순 알림 서비스를 선택합니다. 이 규칙 작업에 대한 설명은 메시지를 SNS 푸시 알림으로 보내기입니다.
5. SNS 주제에서 3단계: Amazon SNS 주제 생성 및 구독, MoistureSensorTopic에서 생성한 주제를 선택하고 메시지 형식을 RAW로 둡니다. IAM 역할에서 새 역할 생성을 선택합니다. 역할 이름(예: **`LowMoistureTopicRole`**)을 입력한 후 역할 생성을 선택합니다.
6. 다음을 선택하고 검토한 후 생성을 선택하여 규칙을 생성합니다.

Raspberry Pi와 습도 센서 설정

Raspberry Pi에 microSD 카드를 삽입하고, 모니터, 키보드, 마우스, 이더넷 케이블(Wi-Fi를 사용하지 않는 경우)을 연결합니다. 전원 케이블은 아직 연결하지 마세요.

JST 점퍼 케이블을 습도 센서에 연결합니다. 점퍼의 반대쪽에 다음 4개 와이어가 있습니다.

- 녹색: I2C SCL
- 흰색: I2C SDA
- 적색: 전원(3.5V)
- 검은색: 접지

Raspberry Pi를 이더넷 잭으로 오른쪽에 고정합니다. 이 방향에서 상단에 GPIO 핀 열이 2개 있습니다. 습도 센서의 와이어를 핀 하단 열에 다음 순서로 연결합니다. 제일 왼쪽 핀부터 시작하여 적색(전원), 흰색(SDA), 녹색(SCL)을 연결합니다. 핀 하나는 건너뛰고 검은색(접지) 와이어를 연결합니다. 자세한 내용은 Python 컴퓨터 배선 단원을 참조하세요.

전원 케이블을 Raspberry Pi에 연결하고 반대쪽 끝을 벽면 콘센트에 꽂아 전원을 켭니다.

Raspberry Pi 구성

1. Welcome to Raspberry Pi(Raspberry Pi를 소개합니다)에서 다음을 선택합니다.
2. 국가, 언어, 시간대 및 키보드 배열을 선택합니다. 다음을 선택합니다.
3. Raspberry Pi 암호를 입력하고 다음을 선택합니다.

4. Wi-Fi 네트워크를 선택하고 다음을 선택합니다. Wi-Fi 네트워크를 사용하지 않을 때는 건너뛰기를 선택합니다.
5. 다음을 선택하고 소프트웨어 업데이트를 확인합니다. 업데이트가 완료되면 다시 시작을 클릭하여 Raspberry Pi를 다시 시작합니다.

Raspberry Pi가 시작되면 I2C 인터페이스를 활성화합니다.

1. Raspbian 데스크톱 좌측 상단 모서리에서 Raspberry 아이콘을 클릭하고, 기본 설정을 선택한 다음 Raspberry Pi Configuration(Raspberry Pi 구성)을 선택합니다.
2. 인터페이스 탭에서 I2C의활성화를 선택합니다.
3. 확인을 선택합니다.

CircuitPython의 Adafruit STEMMA 습도 센서용 라이브러리가 작성됩니다. Raspberry Pi에서 이를 실행하려면 Python 3 최신 버전을 설치해야 합니다.

1. 명령 프롬프트에서 다음 명령을 실행하여 Raspberry Pi 소프트웨어를 업데이트합니다.

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```
2. 다음 명령을 실행하여 Phthon 3 설치를 업데이트합니다.

   ```
   sudo pip3 install --upgrade setuptools
   ```
3. 다음 명령을 실행하여 Raspberry Pi GPIO 라이브러리를 설치합니다.

   ```
   pip3 install RPI.GPIO
   ```
4. 다음 명령을 실행하여 Adafruit Blinka 라이브러리를 설치합니다.

   ```
   pip3 install adafruit-blinka
   ```

   자세한 내용은 Raspberry Pi에서 CircuitPython 라이브러리 설치를 참조하세요.
5. 다음 명령을 실행하여 Adafruit Seesaw 라이브러리를 설치합니다.

   ```
   sudo pip3 install adafruit-circuitpython-seesaw
   ```
6. 다음 명령을 실행하여 AWS IoT Device SDK for Python을 설치합니다.

   ```
   pip3 install AWSIoTPythonSDK
   ```

이제 Raspberry Pi에 필요한 라이브러리가 모두 설치되었습니다. **moistureSensor.py**라는 파일을 만들고 다음 Python 코드를 파일에 복사합니다.

```
from adafruit_seesaw.seesaw import Seesaw
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
from board import SCL, SDA

import logging
import time
import json
import argparse
import busio

# Shadow JSON schema:
#
# {
#   "state": {
#       "desired":{
#           "moisture":<INT VALUE>,
#           "temp":<INT VALUE>
#       }
#   }
# }

# Function called when a shadow is updated
def customShadowCallback_Update(payload, responseStatus, token):

    # Display status and data from update request
    if responseStatus == "timeout":
        print("Update request " + token + " time out!")

    if responseStatus == "accepted":
        payloadDict = json.loads(payload)
        print("~~~~~~~~~~~~~~~~~~~~~~")
        print("Update request with token: " + token + " accepted!")
        print("moisture: " + str(payloadDict["state"]["reported"]["moisture"]))
        print("temperature: " + str(payloadDict["state"]["reported"]["temp"]))
        print("~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Update request " + token + " rejected!")
```

```
# Function called when a shadow is deleted
def customShadowCallback_Delete(payload, responseStatus, token):

     # Display status and data from delete request
    if responseStatus == "timeout":
        print("Delete request " + token + " time out!")

    if responseStatus == "accepted":
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Delete request with token: " + token + " accepted!")
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Delete request " + token + " rejected!")


# Read in command-line parameters
def parseArgs():

    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host",
 help="Your device data endpoint")
    parser.add_argument("-r", "--rootCA", action="store", required=True,
 dest="rootCAPath", help="Root CA file path")
    parser.add_argument("-c", "--cert", action="store", dest="certificatePath",
 help="Certificate file path")
    parser.add_argument("-k", "--key", action="store", dest="privateKeyPath",
 help="Private key file path")
    parser.add_argument("-p", "--port", action="store", dest="port", type=int,
 help="Port number override")
    parser.add_argument("-n", "--thingName", action="store", dest="thingName",
 default="Bot", help="Targeted thing name")
    parser.add_argument("-id", "--clientId", action="store", dest="clientId",
 default="basicShadowUpdater", help="Targeted client id")

    args = parser.parse_args()
    return args


# Configure logging
# AWSIoTMQTTShadowClient writes data to the log
def configureLogging():

    logger = logging.getLogger("AWSIoTPythonSDK.core")
```

```
    logger.setLevel(logging.DEBUG)
    streamHandler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
 %(message)s')
    streamHandler.setFormatter(formatter)
    logger.addHandler(streamHandler)


# Parse command line arguments
args = parseArgs()

if not args.certificatePath or not args.privateKeyPath:
    parser.error("Missing credentials for authentication.")
    exit(2)

# If no --port argument is passed, default to 8883
if not args.port:
    args.port = 8883


# Init AWSIoTMQTTShadowClient
myAWSIoTMQTTShadowClient = None
myAWSIoTMQTTShadowClient = AWSIoTMQTTShadowClient(args.clientId)
myAWSIoTMQTTShadowClient.configureEndpoint(args.host, args.port)
myAWSIoTMQTTShadowClient.configureCredentials(args.rootCAPath, args.privateKeyPath,
 args.certificatePath)

# AWSIoTMQTTShadowClient connection configuration
myAWSIoTMQTTShadowClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTShadowClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTShadowClient.configureMQTTOperationTimeout(5) # 5 sec

# Initialize Raspberry Pi's I2C interface
i2c_bus = busio.I2C(SCL, SDA)

# Intialize SeeSaw, Adafruit's Circuit Python library
ss = Seesaw(i2c_bus, addr=0x36)

# Connect to AWS IoT
myAWSIoTMQTTShadowClient.connect()

# Create a device shadow handler, use this to update and delete shadow document
deviceShadowHandler =
 myAWSIoTMQTTShadowClient.createShadowHandlerWithName(args.thingName, True)
```

```
# Delete current shadow JSON doc
deviceShadowHandler.shadowDelete(customShadowCallback_Delete, 5)

# Read data from moisture sensor and update shadow
while True:

    # read moisture level through capacitive touch pad
    moistureLevel = ss.moisture_read()

    # read temperature from the temperature sensor
    temp = ss.get_temp()

    # Display moisture and temp readings
    print("Moisture Level: {}".format(moistureLevel))
    print("Temperature: {}".format(temp))

    # Create message payload
    payload = {"state":{"reported":{"moisture":str(moistureLevel),"temp":str(temp)}}}

    # Update shadow
    deviceShadowHandler.shadowUpdate(json.dumps(payload), customShadowCallback_Update,
 5)
    time.sleep(1)
```

파일을 찾을 수 있는 위치에 파일을 저장합니다. 다음 파라미터로 `moistureSensor.py` 명령줄을 실행할 수 있습니다.

엔드포인트

사용자 지정 AWS IoT 엔드포인트입니다. 자세한 내용은 디바이스 섀도우 REST API 단원을 참조하십시오.

rootCA

AWS IoT 루트 CA 인증서의 전체 경로입니다.

cert

AWS IoT 디바이스 인증서의 전체 경로입니다.

키

AWS IoT 디바이스 인증서 프라이빗 키의 전체 경로입니다.

thingName

사물 이름(이 경우는 `RaspberryPi`)입니다.

clientId

MQTT 클라이언트 ID입니다. `RaspberryPi`를 사용합니다.

명령줄의 모양은 다음과 같아야 합니다.

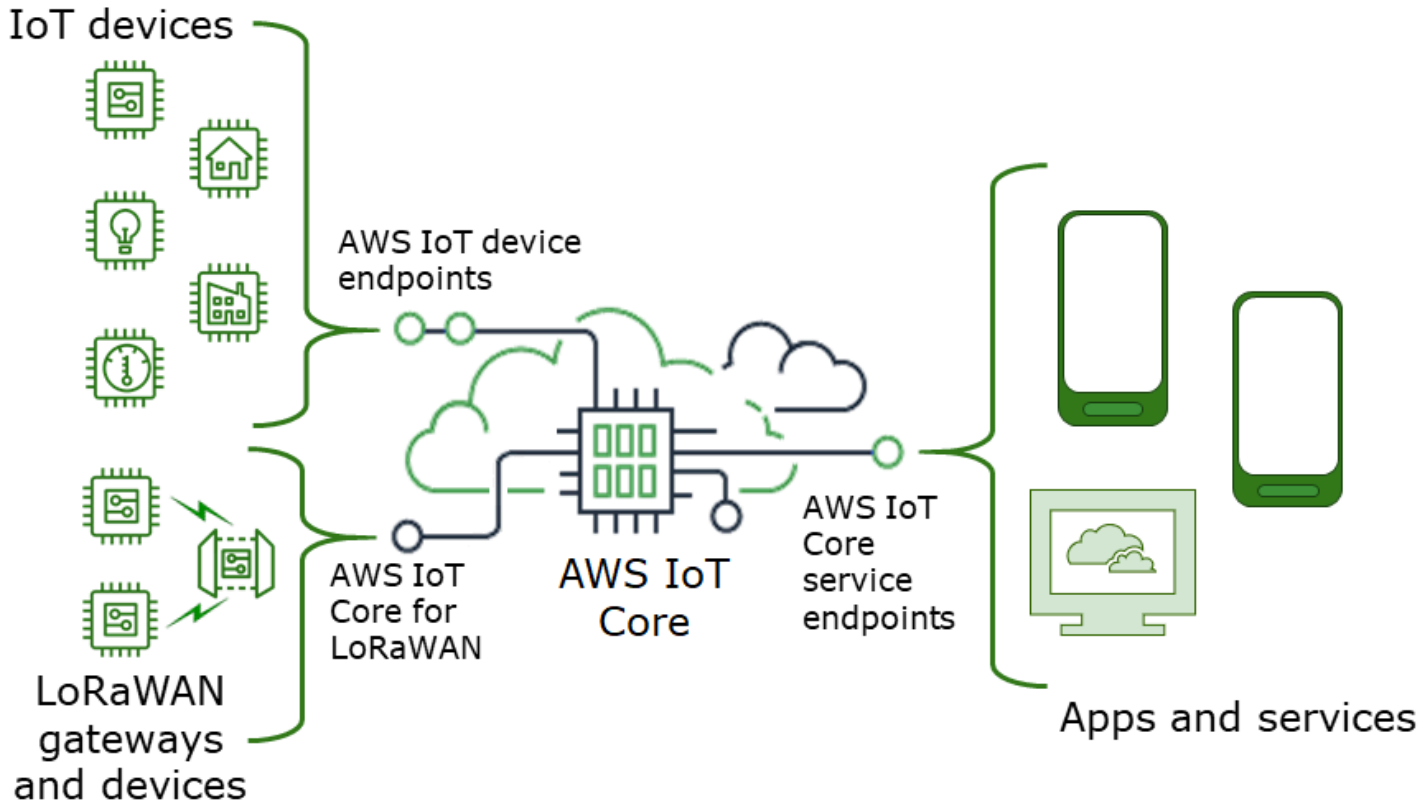
```
python3 moistureSensor.py --endpoint your-endpoint --rootCA ~/certs/
AmazonRootCA1.pem --cert ~/certs/raspberrypi-certificate.pem.crt --key
~/certs/raspberrypi-private.pem.key --thingName RaspberryPi --clientId
RaspberryPi
```

센서를 만지거나, 화분에 넣거나, 물 컵에 넣어 센서가 다양한 습도에 반응하는지 확인해봅니다. 필요할 경우 `MoistureSensorRule`에서 임계값을 변경할 수 있습니다. 습도 센서 판독값이 규칙의 SQL 쿼리 문에 지정된 값 아래로 떨어지면는 Amazon SNS 주제에 메시지를 AWS IoT 게시합니다. 습도와 온도 데이터가 있는 이메일 메시지가 수신되어야 합니다.

Amazon SNS에서 이메일 메시지 수신을 확인한 후에 CTRL+C를 눌러 Python 프로그램을 저장합니다. Python 프로그램으로 전송되는 메시지로 요금이 발생할 가능성은 적지만 사용 후에는 프로그램을 중지시키는 것이 좋습니다.

에 연결 AWS IoT Core

AWS IoT Core 는 IoT 디바이스, 무선 게이트웨이, 서비스 및 앱과의 연결을 지원합니다. 디바이스는에 연결 AWS IoT Core 되므로 AWS IoT 서비스 및 기타 디바이스로 데이터를 전송하고 수신할 수 있습니다. 앱 및 기타 서비스도 AWS IoT Core 에 연결하여 IoT 디바이스를 제어 및 관리하고 IoT 솔루션의 데이터를 처리합니다. 이 섹션에서는 IoT 솔루션의 각 측면에 AWS IoT Core 대해를 연결하고 통신하는 가장 좋은 방법을 선택하는 방법을 설명합니다.



와 상호 작용하는 방법에는 여러 가지가 있습니다 AWS IoT. 앱 및 서비스는를 사용할 수 AWS IoT Core - 컨트롤 플레인 엔드포인트 있으며 디바이스는 AWS IoT 디바이스 엔드포인트 또는 for LoRaWAN 리전 및 엔드포인트 AWS IoT Core 를 사용하여에 연결할 수 있습니다. AWS IoT Core LoRaWAN

AWS IoT Core - 컨트롤 플레인 엔드포인트

AWS IoT Core - 컨트롤 플레인 엔드포인트는 AWS IoT 솔루션을 제어하고 관리하는 함수에 대한 액세스를 제공합니다.

- 엔드포인트

AWS IoT Core- 제어 영역 및 AWS IoT Core Device Advisor 제어 영역 엔드포인트는 리전별로 다르며 AWS IoT Core 엔드포인트 및 할당량에 나열되어 있습니다. 엔드포인트의 형식은 다음과 같습니다.

엔드포인트 용도	엔드포인트 형식	제공
AWS IoT Core - 제어 영역	AWS IoT 컨트롤 플레인 엔드포인트	AWS IoT 컨트롤 플레인 API
AWS IoT Core Device Advisor - 컨트롤 플레인	`api.iotdeviceadvisor.` *`aws-region`* `.amazonaws.com`	AWS IoT Core Device Advisor 컨트롤 플레인 API

- IPv4 엔드포인트 - IPv4 엔드포인트는 IPv4 트래픽만 지원하며 모든 리전에서 사용할 수 있습니다.

  IPv4 엔드포인트에는 다음과 같은 명명 규칙이 사용됩니다.

  ```
  iot.aws-region.amazonaws.com
  ```

  예를 들어 us-east-1 리전의 IPv4 엔드포인트 이름은 `iot.us-east-1.amazonaws.com`입니다.
- 듀얼 스택(IPv4 및 IPv6) 엔드포인트 - 듀얼 스택 엔드포인트는 IPv4 및 IPv6 트래픽을 모두 지원합니다. 이중 스택 엔드포인트에 요청하는 경우, 엔드포인트 URL이 네트워크 및 클라이언트에서 사용하는 프로토콜에 따라 IPv6 또는 IPv4 주소로 확인됩니다.

  이중 스택 엔드포인트에는 다음과 같은 명명 규칙이 사용됩니다.

  ```
  iot.aws-region.api.aws
  ```

  예를 들어 us-east-1 리전의 이중 스택 엔드포인트 이름은 `iot.us-east-1.api.aws`입니다.
- SDK 및 도구

AWS SDKs는 AWS IoT Core APIs 및 다른 AWS 서비스의 APIs에 대한 언어별 지원을 제공합니다. AWS Mobile SDKs는 앱 개발자에게 AWS IoT Core API 및 모바일 디바이스의 기타 AWS 서비스에 대한 플랫폼별 지원을 제공합니다.

는 AWS IoT 서비스 엔드포인트에서 제공하는 함수에 대한 명령줄 액세스를 AWS CLI 제공합니다. AWS Tools for PowerShell은 PowerShell 스크립팅 환경에서 AWS 서비스와 리소스를 관리하는 도구를 제공합니다.

- Authentication

  서비스 엔드포인트는 IAM 사용자 및 AWS 자격 증명을 사용하여 사용자를 인증합니다.

- 자세히 알아보기

  SDK 참조에 대한 자세한 내용과 링크는 the section called “AWS IoT Core 서비스 엔드포인트에 연결” 섹션을 참조하세요.

AWS IoT 디바이스 엔드포인트

AWS IoT 디바이스 엔드포인트는 IoT 디바이스와 간의 통신을 지원합니다 AWS IoT.

- 엔드포인트

  디바이스 엔드포인트는 AWS IoT Core 및 AWS IoT Device Management 함수를 지원합니다. 이들은에 고유 AWS 계정 하며 describe-endpoint 명령을 사용하여가 무엇인지 확인할 수 있습니다.

엔드포인트 용도	엔드포인트 형식	제공
AWS IoT Core - 데이터 영역	???을(를) 참조하세요.	AWS IoT 데이터 영역 API
AWS IoT Device Management - 작업 데이터	???을(를) 참조하세요.	AWS IoT 작업 데이터 영역 API
AWS IoT Device Advisor - 데이터 영역	???을(를) 참조하세요.	해당 사항 없음
AWS IoT Device Management - 보안 터널링	`api.tunneling.iot.` *`aws-region`* `.amazonaws.com`	AWS IoT 보안 터널링 API

데이터 플레인 작업에 듀얼 스택 엔드포인트(IPv4 및 IPv6)를 사용하는 경우 `iot:Data-ATS` 엔드포인트 유형을 사용합니다. `iot:Jobs`는 IPv4에만 사용할 수 있습니다. 이러한 엔드포인트 및 해당

엔드포인트가 지원하는 함수에 대한 자세한 내용은 the section called “AWS IoT 디바이스 데이터 및 서비스 엔드포인트” 섹션을 참조하세요.

- SDK

  AWS IoT 디바이스 SDKs는 디바이스가 통신에 사용하는 Message Queueing Telemetry Transport(MQTT) 및 WebSocket Secure(WSS) 프로토콜에 대한 언어별 지원을 제공합니다 AWS IoT. AWS 모바일 SDKs 또한는 MQTT 디바이스 통신, AWS IoT APIs 및 모바일 디바이스에 있는 다른 AWS 서비스의 APIs에 대한 지원을 제공합니다.

- Authentication

  디바이스 엔드포인트는 X.509 인증서 또는 자격 증명이 있는 AWS IAM 사용자를 사용하여 사용자를 인증합니다.

- 자세히 알아보기

  SDK 참조에 대한 자세한 내용과 링크는 the section called “AWS IoT 디바이스 SDKs” 섹션을 참조하세요.

AWS IoT Core for LoRaWAN 게이트웨이 및 디바이스

AWS IoT Core for LoRaWAN은 무선 게이트웨이와 디바이스를 연결합니다 AWS IoT Core.

- 엔드포인트

  AWS IoT Core for LoRaWAN은 계정 및 리전별 AWS IoT Core 엔드포인트에 대한 게이트웨이 연결을 관리합니다. 게이트웨이는 for LoRaWAN에서 제공하는 계정의 구성 및 업데이트 서버(CUPS) 엔드포인트 AWS IoT Core 에 연결할 수 있습니다.

엔드포인트 용도	엔드포인트 형식	제공
구성 및 업데이트 서버(CUPS)	*account-specific-prefix* .cups.lorawan. *aws-region* .amazonaws.com:443	AWS IoT Core for LoRaWAN에서 제공하는 구성 및 업데이트 서버와의 게이트웨이 통신
LoRaWAN 네트워크 서버(LNS)	*account-specific-prefix* .gateway.	AWS IoT Core for LoRaWAN에서 제공하는 LoRaWAN 네

엔드포인트 용도	엔드포인트 형식	제공
	`lorawan.` *`aws-region`* `.amazonaws.com:443`	트워크 서버와의 게이트웨이 통신

- SDK

  AWS IoT Core for LoRaWAN이 구축된 AWS IoT Wireless API는 AWS SDK에서 지원됩니다. 자세한 내용은 AWS SDK 및 도구 단원을 참조하세요.

- Authentication

  AWS IoT Core for LoRaWAN 디바이스 통신은 X.509 인증서를 사용하여 통신을 보호합니다 AWS IoT.

- 자세히 알아보기

  무선 디바이스 구성 및 연결에 대한 자세한 내용은 AWS IoT Core for LoRaWAN 리전 및 엔드포인트 섹션을 참조하세요.

AWS IoT Core 서비스 엔드포인트에 연결

선호하는 언어에 맞는 AWS SDK를 사용하거나 REST API AWS CLI를 직접 호출하여AWS IoT Core - 컨트롤 플레인의 기능에 액세스할 수 있습니다. AWS CLI 또는 AWS SDK를 사용하여와 상호 작용하는 것이 좋습니다. AWS 서비스 호출 모범 사례를 통합하기 AWS IoT Core 때문입니다. REST API를 직접 호출하는 것은 선택 사항이지만 API에 대한 액세스를 가능하게 하는 필요한 보안 자격 증명을 제공해야 합니다.

> **Note**
>
> IoT 디바이스는 AWS IoT 디바이스 SDKs를 사용해야 합니다. 디바이스 SDKs는 디바이스에서 사용하도록 최적화되어 있고, 디바이스에서 가장 많이 사용하는 AWS IoT APIs와 MQTT 통신을 지원하며 AWS IoT, API를 지원합니다. Device SDK에 대한 자세한 내용과 이 SDK가 제공하는 기능에 대해서는 AWS IoT 디바이스 SDKs 섹션을 참조하세요.
> 모바일 디바이스는 AWS 모바일 SDKs을(를) 사용해야 합니다. Mobile SDKs는 AWS IoT APIs, MQTT 디바이스 통신 및 모바일 디바이스에 있는 다른 AWS 서비스의 APIs에 대한 지원을 제공합니다. Mobile SDK에 대한 자세한 내용과 이 SDK가 제공하는 기능에 대해서는 AWS 모바일 SDKs 단원을 참조하세요.

웹 및 모바일 애플리케이션의 AWS Amplify 도구 및 리소스를 사용하여 보다 쉽게 연결할 수 있습니다 AWS IoT Core. Amplify를 사용하여 AWS IoT Core 에 연결하는 방법에 대한 자세한 내용은 Amplify 설명서의 PubSub를 참조하세요.

다음 섹션에서는 및 AWS IoT 기타 AWS 서비스를 개발하고 상호 작용하는 데 사용할 수 있는 도구 및 SDKs에 대해 설명합니다. 앱을 빌드하고 관리하는 데 사용할 수 있는 AWS 도구 및 개발 키트의 전체 목록은 빌드할 도구를 AWS참조하세요. AWS

AWS CLI 용 AWS IoT Core

는 AWS APIs에 대한 명령줄 액세스를 AWS CLI 제공합니다.

- 설치

  설치 방법에 대한 자세한 내용은 설치를 AWS CLI참조하세요. AWS CLI
- Authentication

  는의 자격 증명을 AWS CLI 사용합니다 AWS 계정.
- 레퍼런스

  이러한 AWS IoT Core 서비스의 AWS CLI 명령에 대한 자세한 내용은 다음을 참조하세요.
  - AWS CLI IoT에 대한 명령 참조
  - AWS CLI IoT 데이터에 대한 명령 참조
  - AWS CLI IoT 작업 데이터에 대한 명령 참조
  - AWS CLI IoT 보안 터널링에 대한 명령 참조

PowerShell 스크립팅 환경에서 AWS 서비스 및 리소스를 관리하는 도구는 AWS Tools for PowerShell을 참조하세요.

AWS SDKs

AWS SDKs 사용하면 앱과 호환 디바이스가 AWS IoT APIs와 다른 AWS 서비스의 APIs를 호출할 수 있습니다. 이 섹션에서는 AWS SDKs 및 AWS IoT Core 서비스의 API에 대한 APIs 참조 설명서에 대한 링크를 제공합니다.

AWS SDKs는 이러한 AWS IoT Core APIs 지원합니다.

- AWS IoT
- AWS IoT 데이터 영역

- AWS IoT 작업 데이터 영역
- AWS IoT 보안 터널링
- AWS IoT 무선

C++

AWS SDK for C++를 설치하고 사용하여 AWS IoT에 연결하기

1. AWS SDK for C++ 사용 시작하기의 지침을 따릅니다.

   다음 지침은 실행 방법을 설명합니다.

   - 소스 파일에서 SDK 설치 및 빌드
   - SDK를 사용하기 위한 자격 증명을 AWS 계정에 제공
   - 앱 또는 서비스에서 SDK 초기화 및 종료
   - 앱이나 서비스를 빌드하는 CMake 프로젝트 만들기
2. 샘플 앱을 만들고 실행합니다. AWS SDK for C++을 사용하는 샘플 앱의 경우 AWS SDK for C++ 코드 예제를 참조하세요.

에서 AWS SDK for C++ 지원하는 AWS IoT Core 서비스에 대한 설명서

- AWS::IoTClient" 참조 문서
- Aws::IoTDataPlane::IoTDataPlaneClient 참조 문서
- Aws::IoTJobsDataPlane::IoTJobsDataPlaneClient 참조 문서
- Aws::IoTSecureTunneling::IoTSecureTunnelingClient 참조 문서

Go

AWS SDK for Go을 설치하고 사용하여 AWS IoT에 연결하기

1. 시작하기의 지침을 따릅니다. AWS SDK for Go

   다음 지침은 실행 방법을 설명합니다.

   - 설치 AWS SDK for Go
   - SDK로 AWS 계정에 액세스하기 위한 액세스 키 가져오기
   - 앱 또는 서비스의 소스 코드로 패키지 가져오기

2. 샘플 앱을 만들고 실행합니다. AWS SDK for Go를 사용하는 샘플 앱의 경우 AWS SDK for Go 코드 예제를 참조하세요.

에서 AWS SDK for Go 지원하는 AWS IoT Core 서비스에 대한 설명서

- IoT 참조 문서
- IoTDataPlane 참조 문서
- IoTJobsDataPlane 참조 문서
- IoTSecureTunneling 참조 문서

Java

AWS SDK for Java을 설치하고 사용하여 AWS IoT에 연결하기

1. 시작하기의 지침을 따릅니다. AWS SDK for Java 2.x

   다음 지침은 실행 방법을 설명합니다.

   - IAM 사용자 가입 AWS 및 생성
   - SDK 다운로드
   - 자격 AWS 증명 및 리전 설정
   - Apache Maven으로 SDK 사용하기
   - Gradle로 SDK 사용하기
2. AWS SDK for Java 2.x 코드 예제 중 하나를 사용하여 샘플 앱을 만들고 실행합니다.
3. SDK API 참조 문서 검토

에서 AWS SDK for Java 지원하는 AWS IoT Core 서비스에 대한 설명서

- IotClient 참조 문서
- IotDataPlaneClient 참조 문서
- IotJobsDataPlaneClient 참조 문서
- IoTSecureTunnelingClient 참조 문서

JavaScript

를 설치하고 이를 AWS SDK for JavaScript 사용하여에 연결하려면 AWS IoT:

1. AWS SDK for JavaScript설정의 지침을 따릅니다. 이 지침은 브라우저 AWS SDK for JavaScript 에서 및 Node.JS와 함께를 사용하는 데 적용됩니다. 설치에 적용되는 지침을 따라야 합니다.

   다음 지침은 실행 방법을 설명합니다.

   - 사전 조건 확인
   - SDK for JavaScript 설치
   - SDK for JavaScript 로드
2. 사용자 환경의 시작 옵션에서 설명하는 것처럼 샘플 앱을 만들고 실행하여 SDK를 시작합니다.
   - 브라우저에서AWS SDK for JavaScript 시작하기
   - Node.js의AWS SDK for JavaScript 시작하기

에서 AWS SDK for JavaScript 지원하는 AWS IoT Core 서비스에 대한 설명서

- `AWS.Iot reference documentation`
- `AWS.IotData reference documentation`
- `AWS.IotJobsDataPlane reference documentation`
- `AWS.IotSecureTunneling reference documentation`

.NET

AWS SDK for .NET를 설치하고 사용하여 AWS IoT에 연결하기

1. AWS SDK for .NET 환경 설정의 지침을 따릅니다.
2. AWS SDK for .NET 프로젝트 설정의 지침을 따릅니다.

   다음 지침은 실행 방법을 설명합니다.

   - 새 프로젝트 시작
   - AWS 자격 증명 획득 및 구성
   - AWS SDK 패키지 설치
3. .NET용 AWS SDK의 AWS 서비스 작업에서 샘플 프로그램 중 하나를 생성하고 실행합니다.
4. SDK API 참조 문서 검토

에서 AWS SDK for .NET 지원하는 AWS IoT Core 서비스에 대한 설명서

- Amazon.IoT.Model 참조 문서
- Amazon.IotData.Model 참조 문서
- Amazon.IoTJobsDataPlane.Model 참조 문서
- Amazon.IoTSecureTunneling.Model 참조 문서

PHP

AWS SDK for PHP을 설치하고 사용하여 AWS IoT에 연결하기

1. AWS SDK for PHP 버전 3 시작하기의 지침을 따릅니다.

   다음 지침은 실행 방법을 설명합니다.
   - 사전 조건 확인
   - SDK 설치
   - PHP 스크립트에 SDK 적용
2. AWS SDK for PHP 버전 3 코드 예제 중 하나를 사용하여 샘플 앱을 만들고 실행합니다.

에서 AWS SDK for PHP 지원하는 AWS IoT Core 서비스에 대한 설명서

- IoTClient 참조 문서
- IoTDataPlaneClient 참조 문서
- IoTJobsDataPlaneClient 참조 문서
- IoTSecureTunnelingClient 참조 문서

Python

AWS SDK for Python (Boto3)을 설치하고 사용하여 AWS IoT에 연결하기

1. AWS SDK for Python (Boto3) Quickstart의 지침을 따르세요.

   다음 지침은 실행 방법을 설명합니다.
   - SDK 설치
   - SDK 구성
   - 코드에서 SDK 사용

2. AWS SDK for Python (Boto3)을 사용하는 샘플 프로그램 생성 및 실행

   이 프로그램은 계정의 현재 구성된 로깅 옵션을 표시합니다. SDK를 설치하고 계정에 대해 구성한 후 이 프로그램을 실행할 수 있습니다.

```
import boto3
import json

# initialize client
iot = boto3.client('iot')

# get current logging levels, format them as JSON, and write them to stdout
response = iot.get_v2_logging_options()
print(json.dumps(response, indent=4))
```

   이 예제에 사용되는 함수에 대한 자세한 내용은 the section called “AWS IoT 로깅 구성” 단원을 참조하세요.

에서 AWS SDK for Python (Boto3) 지원하는 AWS IoT Core 서비스에 대한 설명서

- IoT 참조 문서
- IoTDataPlane 참조 문서
- IoTJobsDataPlane 참조 문서
- IoTSecureTunneling 참조 문서

Ruby

AWS SDK for Ruby을 설치하고 사용하여 AWS IoT에 연결하기

- 시작하기의 지침을 따릅니다. AWS SDK for Ruby

  다음 지침은 실행 방법을 설명합니다.

  - SDK 설치
  - SDK 구성
- Hello World 자습서를 생성하고 실행하세요.

AWS SDK for Ruby가 지원하는 AWS IoT Core 서비스에 대한 설명서

- Aws::IoT::Client 참조 문서
- Aws::IoTDataPlane::Client 참조 문서
- Aws::IoTJobsDataPlane::Client 참조 문서
- Aws::IoTSecureTunneling::Client 참조 문서

AWS 모바일 SDKs

AWS Mobile SDKs는 모바일 앱 개발자에게 AWS IoT Core 서비스의 APIs, MQTT를 사용한 IoT 디바이스 통신 및 기타 AWS 서비스의 APIs에 대한 플랫폼별 지원을 제공합니다.

Android

AWS Mobile SDK for Android

에는 개발자가를 사용하여 연결된 모바일 애플리케이션을 빌드할 수 있는 라이브러리, 샘플 및 설명서가 AWS Mobile SDK for Android 포함되어 있습니다 AWS. 이 SDK에는 MQTT 디바이스 통신 및 AWS IoT Core 서비스의 APIs 호출에 대한 지원도 포함되어 있습니다. 자세한 내용은 다음을 참조하세요.

- GitHub의 Android용AWS Mobile SDK
- AWS Android Readme용 Mobile SDK
- AWS Android용 Mobile SDK 샘플
- AWS SDK for Android API 참조
- AWSIoTClient Class 참조 문서

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS 는 Apache 오픈 소스 라이선스에 따라 배포되는 오픈 소스 소프트웨어 개발 키트입니다. iOS용 SDK는 개발자가를 사용하여 연결된 모바일 애플리케이션을 구축하는 데 도움이 되는 라이브러리, 코드 샘플 및 설명서를 제공합니다 AWS. 이 SDK에는 MQTT 디바이스 통신 및 AWS IoT Core 서비스의 APIs 호출에 대한 지원도 포함되어 있습니다. 자세한 내용은 다음을 참조하세요.

- AWS Mobile SDK for iOS GitHub의
- AWS SDK for iOS Readme
- AWS SDK for iOS 샘플
- AWS IoTAWS SDK for iOS의 클래스 참조 문서

AWS IoT Core 서비스의 REST APIs

AWS IoT Core 서비스의 REST APIs는 HTTP 요청을 사용하여 직접 호출할 수 있습니다.

- 엔드포인트 URL

  AWS IoT Core 서비스의 REST API를 노출하는 서비스 엔드포인트는 리전에 따라 다르며 AWS IoT Core 엔드포인트 및 할당량에 나열되어 있습니다. AWS IoT 리소스는 리전별로 다르므로 액세스하려는 AWS IoT 리소스가 있는 리전에 엔드포인트를 사용해야 합니다.

- Authentication

  AWS IoT Core 서비스의 REST APIs는 인증에 AWS IAM 자격 증명을 사용합니다. 자세한 내용은 AWS 일반 참조의 API 요청 서명을 AWS 참조하세요.

- API 참조

  AWS IoT Core 서비스의 REST APIs

  - IoT용 API 참조.
  - IoT 데이터용 API 참조.
  - IoT 작업 데이터용 API 참조.
  - IoT 보안 터널링을 위한 API 참조.

디바이스를에 연결 AWS IoT

디바이스는를 통해 AWS IoT 및 기타 서비스에 연결됩니다 AWS IoT Core. AWS IoT Core를 통해 디바이스는 계정과 관련된 디바이스 엔드포인트를 사용하여 메시지를 보내고 받습니다. the section called “AWS IoT 디바이스 SDKs”는 MQTT 및 WSS 프로토콜을 사용하여 디바이스 통신을 지원합니다. 디바이스가 사용할 수 있는 프로토콜에 대한 자세한 내용은 the section called “디바이스 통신 프로토콜” 단원을 참조하세요.

메시지 브로커

AWS IoT 는 메시지 브로커를 통해 디바이스 통신을 관리합니다. 디바이스 및 클라이언트는 메시지 브로커에 메시지를 게시하고 메시지 브로커가 게시하는 메시지를 구독합니다. 메시지는 애플리케이션이 정의한 주제로 식별됩니다. 메시지 브로커는 디바이스 또는 클라이언트에서 게시한 메시지를 수신하면 해당 메시지를 메시지 주제를 구독한 디바이스 및 클라이언트에 다시 게시합니다. 메시지 브로커는 메시지 내용을 처리할 수 있는 AWS IoT 규칙 엔진에도 메시지를 전달합니다.

AWS IoT 메시지 보안

인증을 위해 the section called “X.509 클라이언트 인증서” 및 AWS 서명 V4를 AWS IoT 사용할 디바이스 연결입니다. 디바이스 통신은 TLS 버전 1.3으로 보호되며 디바이스가 연결할 때 SNI(Server Name Indication) 확장을 전송 AWS IoT 해야 합니다. 자세한 내용은 전송 보안을 참조하세요 AWS IoT.

AWS IoT 디바이스 데이터 및 서비스 엔드포인트

> **Important**
>
> 엔드포인트를 디바이스에 캐시하거나 저장할 수 있습니다. 즉, 새 디바이스가 연결될 때마다 `DescribeEndpoint` API를 쿼리할 필요가 없습니다. 가 계정에 엔드포인트를 AWS IoT Core 생성한 후에는 엔드포인트가 변경되지 않습니다.

각 계정에는 계정에 고유한 여러 디바이스 엔드포인트가 있으며 특정 IoT 기능을 지원합니다. AWS IoT 디바이스 데이터 엔드포인트는 IoT 디바이스의 통신 요구 사항에 맞게 설계된 게시/구독 프로토콜을 지원하지만 앱 및 서비스와 같은 다른 클라이언트도 애플리케이션에 이러한 엔드포인트가 제공하는 특수 기능이 필요한 경우이 인터페이스를 사용할 수 있습니다. AWS IoT 디바이스 서비스 엔드포인트는 보안 및 관리 서비스에 대한 디바이스 중심 액세스를 지원합니다.

계정의 디바이스 데이터 엔드포인트를 알아보려면 AWS IoT Core 콘솔의 설정 페이지에서 찾을 수 있습니다.

디바이스 데이터 엔드포인트를 포함하여 특정 용도에 대한 계정의 디바이스 엔드포인트를 알아보려면, 여기에 나타난 describe-endpoint CLI 명령 또는 `DescribeEndpoint` REST API를 사용하고 다음 표의 *`endpointType`* 파라미터 값을 제공합니다.

```
aws iot describe-endpoint --endpoint-type endpointType
```

이 명령은 *`account-specific-prefix`*`.iot.`*`aws-region`*`.amazonaws.com` 형식으로 *`iot-endpoint`*를 반환합니다.

모든 고객에게는 `iot:Data-ATS` 및 `iot:Data` 엔드포인트가 있습니다. 각 엔드포인트는 X.509 인증서를 사용하여 클라이언트를 인증합니다. Symantec 인증 기관에 대해 만연해 있는 불신과 관련된 문제를 피하려면 `iot:Data-ATS` 엔드포인트 유형을 사용하는 것이 좋습니다. 이전 버전과의 호환성을 위해 VeriSign 인증서를 사용하는 이전 엔드포인트에서 데이터를 검색하는 디바이스용 `iot:Data` 엔드포인트를 제공합니다. 자세한 내용은 서버 인증을 참조하세요.

AWS IoT 디바이스의 엔드포인트

엔드포인트 용도	*endpointType* 값	설명
AWS IoT Core - 데이터 영역 작업	`iot:Data-ATS`	메시지 브로커, 디바이스 섀도우 및 AWS IoT의 규칙 엔진 구성 요소와 데이터를 주고받는 데 사용됩니다. `iot:Data-ATS` 은(는) ATS 서명 데이터 엔드포인트를 반환합니다.
AWS IoT Core - 데이터 영역 작업(레거시)	`iot:Data`	`iot:Data`는 이전 버전과의 호환성을 위해 제공된 VeriSign 서명 데이터 엔드포인트를 반환합니다. MQTT 5는 Symantec(`iot:Data`) 엔드포인트에서 지원되지 않습니다.
AWS IoT Core 자격 증명 액세스	`iot:CredentialProvider`	다른 AWS 서비스와 직접 연결하기 위해 디바이스의 기본 제공 X.509 인증서를 임시 자격 증명과 교환하는 데 사용됩니다. 다른 AWS 서비스에 연결하는 방법에 대한 자세한 내용은 AWS 서비스에 대한 직접 호출 권한 부여를 참조하세요.
AWS IoT Device Management - 작업 데이터 작업	`iot:Jobs`	AWS IoT 작업 디바이스 HTTPS APIs를 사용하여 디바이스가 작업 서비스와 상호 작

엔드포인트 용도	*endpointType* 값	설명
		용할 수 있도록 하는 데 사용됩니다. `iot:Jobs`는 IPv4에만 사용할 수 있습니다. 듀얼 스택 엔드포인트(IPv4 및 IPv6)를 사용하는 경우 `iot:Data-ATS` 엔드포인트 유형을 사용합니다.
AWS IoT Device Advisor 작업	`iot:DeviceAdvisor`	Device Advisor를 사용하여 디바이스를 테스트하는 데 사용되는 테스트 엔드포인트 유형입니다. 자세한 내용은 ??? 단원을 참조하십시오.
AWS IoT Core 데이터 베타(미리 보기)	`iot:Data-Beta`	베타 릴리스용으로 예약된 엔드포인트 유형입니다. 현재 사용에 대한 자세한 내용은 ??? 섹션을 참조하세요.

example.com 같은 자체 FQDN(정규화된 도메인 이름)과 관련 서버 인증서를 사용하여를 사용하여 디바이스를 AWS IoT 에 연결할 수도 있습니다the section called “도메인 구성”.

AWS IoT 디바이스 SDKs

AWS IoT 디바이스 SDKs는 IoT 디바이스를에 연결하는 데 도움이 AWS IoT Core 되며 MQTT 및 MQTT over WSS 프로토콜을 지원합니다.

AWS IoT 디바이스 SDKs는 AWS IoT 디바이스 SDKs가 IoT 디바이스의 특수 통신 요구 사항을 지원하지만 AWS SDKs에서 지원하는 모든 서비스를 지원하지는 않는다는 점에서 AWS SDKs 다릅니다. AWS IoT 디바이스 SDKs는 모든 AWS 서비스를 지원하는 AWS SDKs와 호환되지만, 서로 다른 인증 방법을 사용하고 서로 다른 엔드포인트에 연결되므로 IoT 디바이스에서 AWS SDKs를 사용하는 것이 실용적이지 않을 수 있습니다.

모바일 디바이스

는 MQTT 디바이스 통신, 일부 AWS IoT 서비스 APIs 및 다른 AWS 서비스의 APIs를 모두 the section called “AWS 모바일 SDKs” 지원합니다. 지원되는 모바일 디바이스에서 개발하는 경우 IoT 솔루션 개발에 가장 적합한 옵션인지 확인하기 위해 SDK를 검토하세요.

C++

AWS IoT C++ 디바이스 SDK

AWS IoT C++ 디바이스 SDK를 사용하면 개발자가 AWS 및 AWS IoT Core 서비스의 APIs를 사용하여 연결된 애플리케이션을 구축할 수 있습니다. 특히 이 SDK는 리소스의 제한을 받지 않으면서 메시지 대기열, 멀티-스레딩 지원, 최신 언어 같은 고급 기능이 필요한 디바이스를 위해 설계되었습니다. 자세한 내용은 다음을 참조하세요.

- AWS IoT GitHub의 Device SDK C++ v2
- AWS IoT Device SDK C++ v2 Readme
- AWS IoT Device SDK C++ v2 샘플
- AWS IoT Device SDK C++ v2 API 설명서

Python

AWS IoT Python용 디바이스 SDK

AWS IoT Device SDK for Python을 사용하면 개발자가 Python 스크립트를 작성하여 디바이스를 사용하여 MQTT 또는 MQTT over WebSocket Secure(WSS) 프로토콜을 통해 AWS IoT 플랫폼에 액세스할 수 있습니다. 디바이스를 AWS IoT Core 서비스의 APIs에 연결하면 사용자는 , AWS Lambda Amazon Kinesis, Amazon S3 등과 같은 다른 서비스와 함께 및를 AWS IoT Core 제공하는 메시지 브로커, 규칙 및 디바이스 섀도우 AWS 서비스와 안전하게 작업할 수 있습니다.

- AWS IoT GitHub의 Python v2용 디바이스 SDK
- AWS IoT Device SDK for Python v2 Readme
- AWS IoT Device SDK for Python v2 샘플
- AWS IoT Device SDK for Python v2 API 설명서

JavaScript

AWS IoT JavaScript용 디바이스 SDK

AWS IoT Device SDK for JavaScript를 사용하면 개발자가 WebSocket 프로토콜을 통해 MQTT 또는 MQTT를 AWS IoT Core 사용하여의 APIs에 액세스하는 JavaScript 애플리케이션을 작성할 수 있습니다. 이 패키지는 Node.js 환경 및 브라우저 애플리케이션에서 사용할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- AWS IoT GitHub의 JavaScript v2용 디바이스 SDK
- AWS IoT Device SDK for JavaScript v2 Readme
- AWS IoT Device SDK for JavaScript v2 샘플
- AWS IoT Device SDK for JavaScript v2 API 설명서

Java

AWS IoT Java용 디바이스 SDK

Java용 AWS IoT 디바이스 SDK를 사용하면 Java 개발자가 WebSocket 프로토콜을 통해 MQTT 또는 MQTT를 AWS IoT Core 통해의 APIs에 액세스할 수 있습니다. SDK는 디바이스 섀도우 서비스를 지원합니다. 섀도우는 GET, UPDATE, DELETE 등의 HTTP 메서드를 사용하여 액세스할 수 있습니다. 또한 이 SDK는 간소화된 섀도우 액세스 모델을 지원합니다. 이 모델에서는 개발자가 JSON 문서를 직렬화 또는 역직렬화할 필요 없이 getter 및 setter 메서드를 사용하여 데이터를 섀도우와 교환할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- AWS IoT GitHub의 Java v2용 디바이스 SDK
- AWS IoT Device SDK for Java v2 Readme
- AWS IoT Device SDK for Java v2 샘플
- AWS IoT Device SDK for Java v2 API 설명서

Embedded C

AWS IoT 임베디드 C용 디바이스 SDK

> **Important**
>
> 이 SDK는 숙련된 임베디드 소프트웨어 개발자가 사용하기 위한 것입니다.

AWS IoT Device SDK for Embedded C (C-SDK)는 임베디드 애플리케이션에서 IoT 디바이스를 AWS IoT Core에 안전하게 연결하는 데 사용할 수 있는 MIT 오픈 소스 라이선스에 따른 C 소스 파

일 모음입니다. 여기에는 MQTT, JSON Parser, AWS IoT Device Shadow 라이브러리 등이 포함됩니다. 소스 형식으로 배포되며 애플리케이션 코드, 기타 라이브러리 및 선택적으로 RTOS(실시간 운영 체제)와 함께 고객 펌웨어에 구축됩니다.

AWS IoT Device SDK for Embedded C 는 일반적으로 최적화된 C 언어 런타임이 필요한 리소스 제한 디바이스를 대상으로 합니다. 모든 운영 체제에서 SDK를 사용하고, 모든 프로세서 유형(예: MCU 및 MPU)에서 호스팅할 수 있습니다. 디바이스에 사용 가능한 메모리 및 처리 리소스가 충분한 경우 C++, Java, JavaScript 또는 Python용 AWS IoT 디바이스 SDKs와 같은 다른 AWS IoT 디바이스 및 모바일 SDK 중 하나를 사용하는 것이 좋습니다.

자세한 내용은 다음을 참조하세요.

- AWS IoT GitHub의 임베디드 C용 디바이스 SDK
- AWS IoT 임베디드 C Readme용 디바이스 SDK
- AWS IoT 임베디드 C 샘플용 디바이스 SDK

디바이스 통신 프로토콜

AWS IoT Core 는 MQTT 및 MQTT over WebSocket Secure(WSS) 프로토콜을 사용하여 메시지를 게시하고 구독하는 디바이스 및 클라이언트와 HTTPS 프로토콜을 사용하여 메시지를 게시하는 디바이스 및 클라이언트를 지원합니다. 모든 프로토콜은 IPv4 및 IPv6을 지원합니다. 이 섹션에서는 디바이스 및 클라이언트에 대한 다양한 연결 옵션에 대해 설명합니다.

TLS 프로토콜 버전

AWS IoT Core 는 TLS 버전 1.2 및 TLS 버전 1.3을 사용하여 모든 통신을 암호화합니다. 도메인 구성에서 TLS 설정을 구성하여 엔드포인트에 대한 추가 TLS 정책 버전을 구성할 수 있습니다. 디바이스를 AWS IoT Core에 연결할 때 클라이언트는 다중 계정 등록, 구성 가능한 엔드포인트, 사용자 지정 도메인 및 VPC 엔드포인트와 같은 기능에 필요한 서버 이름 표시(SNI) 확장을 전송할 수 있습니다. 자세한 내용은 AWS IoT의 전송 보안을 참조하세요.

AWS IoT 디바이스 SDKs는 MQTT 및 MQTT over WSS를 지원하고 클라이언트 연결의 보안 요구 사항을 지원합니다. 클라이언트를 AWS IoT에 연결하기 위해 AWS IoT 디바이스 SDKs를 사용하는 것이 좋습니다.

프로토콜, 포트 매핑 및 인증

디바이스 또는 클라이언트를 메시지 브로커에 연결하는 방법은 인증 유형을 사용하여 구성할 수 있습니다. 기본적으로 또는 SNI 확장이 전송되지 않는 경우 인증 방법은 디바이스가 사용하는 애플리케이

션 프로토콜, 포트 및 애플리케이션 계층 프로토콜 협상(ALPN) TLS 확장을 기반으로 합니다. 다음 표에는 포트, 포트 및 ALPN을 기반으로 예상되는 인증이 나열되어 있습니다.

프로토콜, 인증 및 포트 매핑

프로토콜	지원되는 작업	Authentication	포트	ALPN 프로토콜 이름
MQTT over WebSocket	게시, 구독	서명 버전 4	443	해당 사항 없음
MQTT over WebSocket	게시, 구독	사용자 지정 인증	443	해당 사항 없음
MQTT	게시, 구독	X.509 클라이언트 인증서	443†	`x-amzn-mqtt-ca`
MQTT	게시, 구독	X.509 클라이언트 인증서	8883	해당 사항 없음
MQTT	게시, 구독	사용자 지정 인증	443†	`mqtt`
HTTPS	게시 전용	서명 버전 4	443	해당 사항 없음
HTTPS	게시 전용	X.509 클라이언트 인증서	443†	`x-amzn-http-ca`
HTTPS	게시 전용	X.509 클라이언트 인증서	8443	해당 사항 없음
HTTPS	게시 전용	사용자 지정 인증	443	해당 사항 없음

애플리케이션 계층 프로토콜 협상(ALPN)

†기본 엔드포인트 구성을 사용할 때 X.509 클라이언트 인증서 인증을 사용하여 포트 443에 연결하는 클라이언트는 애플리케이션 계층 프로토콜 협상(ALPN) TLS 확장을 구현하고 클라이언트가 `ClientHello` 메시지의 일부로 전송한 ALPN ProtocolNameList에 나열된 ALPN 프로토콜 이름을 사용해야 합니다.

포트 443에서 IoT:Data-ATS 엔드포인트는 ALPN x-amzn-http-ca HTTP를 지원하지만 IoT:Jobs 엔드포인트는 그렇지 않습니다.
ALPN x-amzn-mqtt-ca를 지원하는 포트 8443 HTTPS 및 포트 443 MQTT에서 사용자 지정 인증은 사용할 수 없습니다.

클라이언트는 AWS 계정디바이스 엔드포인트에 연결합니다. 계정의 디바이스 엔드포인트를 찾는 방법에 대한 자세한 내용은 the section called “AWS IoT 디바이스 데이터 및 서비스 엔드포인트” 섹션을 참조하세요.

> **Note**
>
> AWS SDKs 전체 URL이 필요하지 않습니다. pubsub.py GitHub의 AWS IoT Device SDK for Python 샘플과 같은 엔드포인트 호스트 이름만 필요합니다. 다음 표에 제공된 대로 전체 URL을 전달하면 잘못된 호스트 이름과 같은 오류가 발생할 수 있습니다.

에 연결 AWS IoT Core

프로토콜	엔드포인트 또는 URL
MQTT	*iot-endpoint*
MQTT over WSS	wss://*iot-endpoint* /mqtt
HTTPS	https://*iot-endpoint* /topics

디바이스 통신을 위한 애플리케이션 프로토콜 선택

디바이스 엔드포인트를 통한 대부분의 IoT 디바이스 통신에 대해 대부분의 경우 Secure MQTT 또는 WebSocket Secure(WSS)를 통한 MQTT 프로토콜을 사용하지만 디바이스 엔드포인트에서는 HTTPS도 지원합니다.

다음 표에서는가 디바이스 통신을 위해 두 개의 상위 수준 프로토콜(MQTT 및 HTTPS)을 AWS IoT Core 사용하는 방법을 비교합니다.

AWS IoT 디바이스 프로토콜(MQTT 및 HTTPS) side-by-side

기능	MQTT	HTTPS
게시/구독 지원	게시 및 구독	게시 전용
SDK 지원	AWS 디바이스 SDKs MQTT 및 WSS 프로토콜 지원	SDK를 지원하지 않지만 언어별 메서드를 사용하여 HTTPS 요청을 할 수 있습니다.
서비스 품질 지원	MQTT QoS 레벨 0과 1	QoS는 값이 0 또는 1일 수 있는 쿼리 문자열 파라미터 ? qos=qos를 전달하여 지원됩니다. 이 쿼리 문자열을 추가하여 원하는 QoS 값으로 메시지를 게시할 수 있습니다.
디바이스가 오프라인 상태일 때 누락된 메시지를 받을 수 있음	예	아니요
clientId 현장 지원	예	아니요
디바이스 연결 해제 감지	예	아니요
통신 보안	예. ??? 섹션을 참조하세요.	예. ??? 섹션을 참조하세요.
주제 정의	정의된 애플리케이션	정의된 애플리케이션
메시지 데이터 형식	정의된 애플리케이션	정의된 애플리케이션
프로토콜 오버헤드	낮음	높음
전력 소비	낮음	높음

디바이스 통신을 위한 인증 유형 선택

구성 가능한 엔드포인트를 사용하여 IoT 엔드포인트의 인증 유형을 구성할 수 있습니다. 또는 기본 구성을 사용하여 디바이스가 애플리케이션 프로토콜, 포트 및 ALPN TLS 확장 조합으로 인증하는 방법

을 결정합니다. 선택한 인증 유형에 따라 디바이스가 연결할 때 인증하는 방법이 결정됩니다 AWS IoT Core. 5가지 인증 유형이 있습니다.

X.509 인증서

가 디바이스를 인증하도록 AWS IoT Core 검증하는 X.509 클라이언트 인증서를 사용하여 디바이스를 인증합니다. 이 인증 유형은 보안 MQTT(TLS를 통한 MQTT) 및 HTTPS 프로토콜에서 작동합니다.

사용자 지정 권한 부여자가 있는 X.509 인증서

X.509 클라이언트 인증서를 사용하여 디바이스를 인증하고 X.509 클라이언트 인증서 정보를 수신하는 사용자 지정 권한 부여자를 사용하여 추가 인증 작업을 수행합니다. 이 인증 유형은 보안 MQTT(TLS를 통한 MQTT) 및 HTTPS 프로토콜에서 작동합니다. 이 인증 유형은 X.509 사용자 지정 인증을 사용하여 구성 가능한 엔드포인트를 사용해야만 가능합니다. ALPN 옵션은 존재하지 않습니다.

AWS 서명 버전 4(SigV4)

Cognito 또는 백엔드 서비스를 사용하여 디바이스를 인증하고 소셜 및 엔터프라이즈 페더레이션을 지원합니다. 이 인증 유형은 WebSocket Secure(WSS)를 통한 MQTT 및 HTTPS 프로토콜에서 작동합니다.

사용자 지정 권한 부여자

AWS IoT Core로 전송된 사용자 지정 인증 정보를 처리하도록 Lambda 함수를 구성하여 디바이스를 인증합니다. 이 인증 유형은 보안 MQTT(TLS를 통한 MQTT), HTTPS 및 WebSocket Secure(WSS)를 통한 MQTT와 함께 작동합니다.

기본값

디바이스가 사용하는 포트 및/또는 애플리케이션 계층 프로토콜 협상(ALPN) 확장을 기반으로 디바이스를 인증합니다. 일부 추가 인증 옵션은 지원되지 않습니다. 자세한 내용은 ??? 단원을 참조하십시오.

아래 표에는 인증 유형과 애플리케이션 프로토콜의 지원되는 모든 조합이 나와 있습니다.

인증 유형 및 애플리케이션 프로토콜의 지원되는 조합

인증 유형	Secure MQTT(TLS를 통한 MQTT)	WebSocket Secure(WSS)를 통한 MQTT	HTTPS	기본값
X.509 인증서	✓		✓	

인증 유형	Secure MQTT(TLS를 통한 MQTT)	WebSocket Secure(WSS)를 통한 MQTT	HTTPS	기본값
사용자 지정 권한 부여자가 있는 X.509 인증서	✓		✓	
AWS 서명 버전 4(SigV4)		✓	✓	
사용자 지정 권한 부여자	✓	✓	✓	
기본값				✓

연결 지속 시간 제한

HTTPS 연결이 요청을 수신하고 응답하는 데 걸리는 시간보다 오래 지속되지 않을 수 있습니다.

MQTT 연결 지속 시간은 사용하는 인증 기능에 따라 다릅니다. 다음 표에는 각 기능에 이상적인 조건에서 최대 연결 지속 시간이 나열되어 있습니다.

인증 기능별 MQTT 연결 지속 시간

기능	최대 지속 시간*
X.509 클라이언트 인증서	1~2주
사용자 지정 인증	1~2주
서명 버전 4	최대 24시간

* 보장되지 않음

X.509 인증서 및 사용자 정의 인증을 사용하면 연결 지속 시간에 엄격한 제한이 없지만 몇 분 정도로 짧을 수 있습니다. 연결 중단은 다양한 이유로 발생할 수 있습니다. 다음 목록에는 가장 일반적인 이유 중 일부가 나와 있습니다.

- Wi-Fi 가용성 중단
- 인터넷 서비스 제공업체(ISP) 연결 중단
- 서비스 패치
- 서비스 배포
- 서비스 Auto Scaling
- 사용할 수 없는 서비스 호스트
- 로드 밸런서 문제 및 업데이트
- 클라이언트 측 오류

디바이스는 연결 해제를 감지하고 다시 연결하기 위한 전략을 구현해야 합니다. 연결 해제 이벤트 및 처리 방법에 대한 자세한 내용은 ???에서 ??? 섹션을 참조하세요.

MQTT

MQTT(Message Queuing Telemetry Transport)는 제약된 디바이스용으로 설계된 경량의 메시징 프로토콜로서 널리 사용되고 있습니다. MQTT에 대한 AWS IoT Core 지원은 MQTT v3.1.1 사양 및 MQTT v5.0 사양을 기반으로 하며, the section called “AWS IoT MQTT 사양과의 차이점”에 설명된 대로 몇 가지 차이가 있습니다. 최신 버전의 표준 MQTT 5에는 새로운 확장성 개선 기능, 사유 코드 응답을 통한 향상된 오류 보고, 메시지 및 세션 만료 타이머, 사용자 지정 사용자 메시지 헤더를 비롯하여 MQTT 기반 시스템을 더욱 강력하게 만드는 몇 가지 주요 기능이 도입되었습니다. 에서 AWS IoT Core 지원하는 MQTT 5 기능에 대한 자세한 내용은 MQTT 5 지원 기능을 참조하세요.는 교차 MQTT 버전(MQTT 3 및 MQTT 5) 통신 AWS IoT Core 도 지원합니다. MQTT 3 게시자는 MQTT 5 게시 메시지를 수신할 MQTT 5 구독자에게 MQTT 3 메시지를 보낼 수 있으며, 그 반대의 경우도 마찬가지입니다.

AWS IoT Core 는 MQTT 프로토콜 및 MQTT over WSS 프로토콜을 사용하고 클라이언트 ID로 식별되는 디바이스 연결을 지원합니다. AWS IoT 디바이스 SDKs는 두 프로토콜을 모두 지원하며 디바이스를 AWS IoT Core에 연결할 때 권장되는 방법입니다. AWS IoT 디바이스 SDKs는 디바이스 및 클라이언트가 서비스에 연결하고 액세스하는 AWS IoT 데 필요한 기능을 지원합니다. 디바이스 SDKs는 AWS IoT 서비스에 필요한 인증 프로토콜과 MQTT 프로토콜 및 MQTT over WSS 프로토콜에 필요한 연결 ID 요구 사항을 지원합니다. AWS 디바이스 SDKs를 AWS IoT 사용하여에 연결하는 방법과 지원되는 언어로 된의 예제에 대한 링크 AWS IoT 에 대한 자세한 내용은 섹션을 참조하세요the section called “AWS IoT 디바이스 SDKs를 사용하여 MQTT에 연결”. MQTT 메시지의 인증 방법과 포트 매핑에 대한 자세한 내용은 ??? 단원을 참조하세요.

AWS IoT 디바이스 SDKs를 사용하여에 연결하는 AWS IoT것이 좋지만 필수는 아닙니다. 그러나 AWS IoT 디바이스 SDKs를 사용하지 않는 경우 필요한 연결 및 통신 보안을 제공해야 합니다. 연결 요청 시

클라이언트는 SNI(서버 이름 표시) TLS 확장을 전송해야 합니다. SNI를 포함하지 않는 연결 시도는 거부됩니다. 자세한 내용은 전송 보안을 참조하세요 AWS IoT. IAM 사용자 및 AWS 자격 증명을 사용하여 클라이언트를 인증하는 클라이언트는 올바른 서명 버전 4 인증을 제공해야 합니다.

클라이언트가 연결되면 API를 사용해 해당하는 MQTT 클라이언트 연결을 모니터링하고 관리할 수 있습니다. 자세한 내용은 the section called “MQTT 연결 관리” 단원을 참조하십시오.

이 주제에서 수행할 작업

- AWS IoT 디바이스 SDKs를 사용하여 MQTT에 연결
- MQTT 서비스 품질(QoS) 옵션
- MQTT 지속적 세션
- MQTT 보존 메시지
- MQTT 마지막 유언 및 증거(LWT) 메시지
- connectAttributes 사용
- MQTT 5에 지원되는 기능
- MQTT 5 속성
- MQTT 사유 코드
- AWS IoT MQTT 사양과의 차이점
- MQTT 연결 관리

AWS IoT 디바이스 SDKs를 사용하여 MQTT에 연결

이 섹션에는 AWS IoT 디바이스 SDKs 및 디바이스를 연결하는 방법을 보여주는 샘플 프로그램의 소스 코드에 대한 링크가 포함되어 있습니다 AWS IoT. 여기에 연결된 샘플 앱은 MQTT 프로토콜 및 MQTT over WSS를 AWS IoT 사용하여에 연결하는 방법을 보여줍니다.

> **Note**
>
> AWS IoT 디바이스 SDKs가 MQTT 5 클라이언트를 릴리스했습니다.

C++

AWS IoT C++ 디바이스 SDK를 사용하여 디바이스 연결

- C++로 MQTT 연결 예제를 보여주는 샘플 앱의 소스 코드

- AWS IoT GitHub의 C++ v2용 디바이스 SDK

Python

AWS IoT Device SDK for Python을 사용하여 디바이스 연결

- Python으로 MQTT 연결 예제를 보여주는 샘플 앱의 소스 코드
- AWS IoT GitHub의 Python용 Device SDK v2

JavaScript

AWS IoT Device SDK for JavaScript를 사용하여 디바이스 연결

- JavaScript로 MQTT 연결 예제를 보여주는 샘플 앱의 소스 코드
- AWS IoT Device SDK for JavaScript v2 on GitHub

Java

Java용 AWS IoT Device SDK를 사용하여 디바이스 연결

> **Note**
>
> Java v2용 AWS IoT 디바이스 SDK는 이제 Android 개발을 지원합니다. 자세한 내용은 Android용AWS IoT Device SDK를 참조하세요.

- Java로 MQTT 연결 예제를 보여주는 샘플 앱의 소스 코드
- AWS IoT GitHub의 Java v2용 디바이스 SDK

Embedded C

임베디드 C용 AWS IoT 디바이스 SDK를 사용하여 디바이스 연결

> **Important**
>
> 이 SDK는 숙련된 임베디드 소프트웨어 개발자가 사용하기 위한 것입니다.

- Embedded C로 MQTT 연결 예제를 보여주는 샘플 앱의 소스 코드
- AWS IoT GitHub의 임베디드 C용 디바이스 SDK

MQTT 서비스 품질(QoS) 옵션

AWS IoT 및 AWS IoT 디바이스 SDKs MQTT 서비스 품질(QoS) 수준 0 및 1를 지원합니다. MQTT 프로토콜은 세 번째 수준의 QoS, 수준를 정의2하지만 이를 지원하지 AWS IoT 않습니다. MQTT 프로토콜만 QoS 기능을 지원합니다. HTTPS는 값이 0 또는 1일 수 있는 쿼리 문자열 파라미터 ?qos=qos를 전달하여 QoS를 지원합니다.

이 표에서는 각 QoS 수준이 메시지 브로커에 게시된 메시지에 미치는 영향을 설명합니다.

QoS 수준	메시지	설명
QoS 수준 0	0회 이상 전송됨	이 수준은 신뢰할 수 있는 통신 링크를 통해 전송되거나 누락되어도 문제 없는 메시지에 사용해야 합니다.
QoS 수준 1	한 번 이상 전송한 다음 PUBACK 응답이 수신될 때까지 반복적으로 전송합니다.	전송한 사람이 PUBACK 응답을 수신하여 성공적인 전달을 나타낼 때까지 메시지는 완료되지 않은 것으로 간주됩니다.

MQTT 지속적 세션

영구 세션에는 클라이언트가 승인하지 않은 서비스 품질(QoS)이 1인 클라이언트의 구독 및 메시지가 저장됩니다. 디바이스가 영구 세션에 다시 연결하면 세션이 재개되고 구독이 복원되며 재연결 전에 수신되어 확인되지 않은 구독된 메시지가 클라이언트로 전송됩니다.

저장된 메시지의 처리는 CloudWatch 지표 및 CloudWatch Logs에 기록됩니다. CloudWatch 및 CloudWatch Logs에 작성된 항목에 대한 자세한 내용은 메시지 브로커 지표 및 대기 중 로그 항목 섹션을 참조하세요.

영구 세션 생성

MQTT 3에서 CONNECT 메시지를 전송하고 cleanSession 플래그를 0으로 설정하여 MQTT 영구 세션을 생성할 수 있습니다. CONNECT 메시지를 전송하는 클라이언트의 세션이 없으면 영구 세션을 새로

만듭니다. 클라이언트에 대한 세션이 이미 있으면 클라이언트는 기존 세션을 재개합니다. 클린 세션을 생성하려면 CONNECT 메시지를 전송하고 cleanSession 플래그를 1로 설정합니다. 이렇게 하면 클라이언트의 연결이 끊어질 때 브로커가 세션 상태를 저장하지 않습니다.

MQTT 5에서는 Clean Start 플래그와 Session Expiry Interval을 설정하여 영구 세션을 처리합니다. 클린 스타트는 연결 세션의 시작과 이전 세션의 종료를 제어합니다. Clean Start = 1로 설정하면 새 세션이 생성되고 이전 세션(있는 경우)이 종료됩니다. Clean Start = 0으로 설정하면 연결 세션이 이전 세션(있는 경우)을 재개합니다. 세션 만료 간격은 연결 세션의 종료를 제어합니다. 세션 만료 간격은 연결이 끊어진 후에도 세션이 지속되는 시간을 초 단위(4바이트 정수)로 지정합니다. Session Expiry interval = 0으로 설정하면 연결이 끊어지는 즉시 세션이 종료됩니다. CONNECT 메시지에 세션 만료 간격이 지정되지 않은 경우 기본값은 0입니다.

MQTT 5 클린 스타트 및 세션 만료

속성 값	설명
Clean Start= 1	새 세션을 생성하고 이전 세션(있는 경우)을 종료합니다.
Clean Start= 0	이전 세션이 있는 경우 세션을 재개합니다.
Session Expiry Interval> 0	세션을 지속합니다.
Session Expiry interval= 0	세션을 지속하지 않습니다.

MQTT 5에서 Clean Start = 1 및 Session Expiry Interval = 0으로 설정하면 이는 MQTT 3 클린 세션과 동일합니다. Clean Start = 0 및 Session Expiry Interval > 0으로 설정하면 이는 MQTT 3 영구 세션과 동일합니다.

> **Note**
>
> 교차 MQTT 버전(MQTT 3 및 MQTT 5) 영구 세션은 지원되지 않습니다. MQTT 3 영구 세션은 MQTT 5 세션으로 재개될 수 없으며, 그 반대의 경우도 마찬가지입니다.

영구 세션 중 작업

클라이언트는 연결 확인됨(CONNACK) 메시지의 `sessionPresent` 속성을 사용해 영구 세션의 존재 여부를 판별해야 합니다. `sessionPresent`가 `1`인 경우 영구 세션이 존재하며 클라이언트에 대해 저장된 메시지는 영구 세션에 다시 연결한 후 메시지 트래픽에 설명된 대로 해당 클라이언트가 CONNACK을 수신한 후에 클라이언트에 전달됩니다. `sessionPresent`가 `1`인 경우, 클라이언트가 다시 구독할 필요가 없습니다. `sessionPresent`가 `0`인 경우 영구 세션이 없으며 클라이언트는 주제 필터를 다시 구독해야 합니다.

클라이언트가 영구 세션에 참가한 후에는 각 작업에 대한 추가 플래그 없이 메시지를 게시하고 주제 필터를 구독할 수 있습니다.

영구 세션에 다시 연결한 후 메시지 트래픽

영구 세션이란 클라이언트와 MQTT 메시지 브로커 간의 지속적 연결을 나타냅니다. 클라이언트가 영구 세션을 사용하여 메시지 브로커에 연결하면 메시지 브로커는 해당 클라이언트가 연결 중 설정한 구독을 모두 저장합니다. 클라이언트의 연결이 해제되면 메시지 브로커는 클라이언트가 구독 중인 주제에 대해 게시된 미확인 QoS 1 메시지와 새로운 QoS 1 메시지를 저장합니다. 메시지는 계정 한도에 따라 저장됩니다. 한도를 초과하는 메시지는 삭제됩니다. 영구 메시지 제한에 대한 자세한 내용은 AWS IoT Core 엔드포인트 및 할당량 섹션을 참조하세요. 클라이언트가 영구 세션에 다시 연결하면 모든 구독이 복구되며 저장된 메시지는 모두 1초당 최대 메시지 10개의 속도로 클라이언트에 전송됩니다. MQTT 5에서 클라이언트가 오프라인일 때 메시지 만료 간격이 있는 아웃바운드 QoS 1이 만료되면, 연결이 재개된 후 클라이언트가 만료된 메시지를 받지 못합니다.

다시 연결한 후 저장된 메시지는 초당 저장 메시지 10개로 제한된 속도로 `Publish requests per second per connection` 제한에 도달할 때까지 현재 메시지 트래픽과 함께 클라이언트에 전송됩니다. 저장된 메시지의 전달 속도가 제한되기 때문에 세션에 다시 연결 후 전달할 저장된 메시지가 10개 이상 있는 경우 저장된 모든 메시지를 전달하는 데 몇 초가 걸립니다.

공유 구독자의 경우, 그룹의 구독자 한 명 이상이 영구 세션을 사용하고 QoS 1 메시지를 수신할 온라인 상태의 구독자가 없는 경우 메시지가 대기열에 추가됩니다. 메시지 대기열 해제는 그룹의 활성 구독자마다 초당 최대 메시지 20개의 속도로 수행됩니다. 자세한 내용은 공유 구독 메시지 대기열을 참조하세요.

영구 세션 종료

영구 세션은 다음과 같은 방식으로 종료될 수 있습니다.

- 영구 세션 만료 시간이 경과합니다. 클라이언트 연결 해제를 통해 또는 연결 시간 초과로 인해 메시지 브로커가 클라이언트 연결 해제를 감지하면 영구 세션 만료 타이머가 시작됩니다.

- 클라이언트가 `cleanSession` 플래그를 1로 설정한 `CONNECT` 메시지를 전송합니다.
- 클라이언트를 수동으로 연결 해제하고 `DeleteConnection` API를 사용하여 세션을 삭제합니다. 자세한 내용은 the section called “MQTT 연결 관리” 단원을 참조하십시오.

MQTT 3에서 영구 세션 만료 시간의 기본값은 1시간이며, 이는 계정의 모든 세션에 적용됩니다.

MQTT 5에서는 CONNECT 및 DISCONNECT 패킷의 각 세션에 대해 세션 만료 간격을 설정할 수 있습니다.

DISCONNECT 패킷의 세션 만료 간격의 경우:

- 현재 세션의 세션 만료 간격이 0이면 DISCONNECT 패킷에서 세션 만료 간격을 0보다 크게 설정할 수 없습니다.
- 현재 세션의 세션 만료 간격이 0보다 크고 DISCONNECT 패킷에서 세션 만료 간격을 0으로 설정한 경우 세션이 DISCONNECT 시 종료됩니다.
- 그렇지 않으면 DISCONNECT 패킷의 세션 만료 간격이 현재 세션의 세션 만료 간격을 업데이트합니다.

> **Note**
>
> 세션이 끝날 때 클라이언트로 전송되기를 기다리는 저장된 메시지는 삭제되지만 전송할 수 없더라도 표준 메시징 요금으로 요금이 청구됩니다. 메시지 요금에 대한 자세한 내용은 AWS IoT Core 요금을 참조하세요. 만료 시간 간격을 구성할 수 있습니다.

영구 세션이 만료된 후 다시 연결

클라이언트가 만료되기 전에 영구 세션에 다시 연결하지 않으면 세션이 종료되고 저장된 메시지가 삭제됩니다. `cleanSession` 플래그를 `0`(으)로 설정한 세션이 만료된 후 클라이언트가 다시 연결하면 서비스가 새 영구 세션을 생성합니다. 이전 세션의 구독이나 메시지는 이전 세션이 만료되었을 때 삭제되었기 때문에 이 세션에서 사용할 수 없습니다.

영구 세션 메시지 요금

메시지 브로커가 클라이언트 또는 오프라인 영구 세션에 메시지를 보낼 AWS 계정 때에 메시지가 청구됩니다. 영구 세션이 있는 오프라인 디바이스가 다시 연결되고 세션을 재개하면 저장된 메시지가 디바이스로 전송되고 계정에 다시 요금이 청구됩니다. 메시지 요금에 대한 자세한 내용은 AWS IoT Core 요금 - 메시징을 참조하세요.

표준 한도 증가 프로세스를 사용하여 기본 영구 세션 만료 시간을 1시간으로 늘릴 수 있습니다. 세션 만료 시간을 늘리면 오프라인 디바이스에 더 많은 메시지를 저장할 수 있고 이러한 추가 메시지는 표준 메시징 요금으로 계정에 청구되기 때문에 메시지 요금이 증가할 수 있습니다. 세션 만료 시간은 대략적인 것이며 세션은 계정 제한보다 최대 30분 더 오래 지속될 수 있습니다. 그러나 세션은 계정 제한보다 짧지 않습니다. 세션 제한에 대한 자세한 내용은 AWS Service Quotas를 참조하세요.

MQTT 보존 메시지

AWS IoT Core 는 MQTT 프로토콜에 설명된 `RETAIN` 플래그를 지원합니다. 클라이언트가 게시하는 MQTT 메시지에 `RETAIN` 플래그를 설정하면가 메시지를 AWS IoT Core 저장합니다. 그런 다음 새 구독자에게 전송하고 `GetRetainedMessage` 작업을 호출하여 검색한 다음 AWS IoT 콘솔에서 볼 수 있습니다.

MQTT 보존 메시지 사용 예

- 초기 구성 메시지로

  MQTT 보관된 메시지는 클라이언트가 주제를 구독한 후 클라이언트로 전송됩니다. 주제를 구독하는 모든 클라이언트가 구독 직후 MQTT 보관된 메시지를 수신하도록 하려면 `RETAIN` 플래그가 설정된 구성 메시지를 게시합니다. 구독 클라이언트는 새 구성 메시지가 게시될 때마다 해당 구성에 대한 업데이트도 받습니다.
- 마지막으로 알려진 상태 메시지로

  디바이스는 현재 상태 메시지에 `RETAIN` 플래그를 설정하여 AWS IoT Core 가 메시지를 저장할 수 있도록 할 수 있습니다. 애플리케이션이 연결되거나 다시 연결되면 이 주제를 구독하고 보관된 메시지 주제를 구독한 직후에 마지막으로 보고된 상태를 가져올 수 있습니다. 이렇게 하면 디바이스에서 다음 메시지가 표시될 때까지 기다리지 않아도 현재 상태를 확인할 수 있습니다.

이 섹션의 내용:

- 에서 MQTT 보존 메시지를 사용하는 일반적인 작업 AWS IoT Core
- 청구 및 보관된 메시지
- MQTT 보관된 메시지와 MQTT 영구 세션 비교
- MQTT 보존 메시지 및 AWS IoT 디바이스 섀도우

에서 MQTT 보존 메시지를 사용하는 일반적인 작업 AWS IoT Core

AWS IoT Core 는 RETAIN 플래그가 설정된 MQTT 메시지를 저장합니다. 이렇게 보관된 메시지는 해당 주제를 구독한 모든 클라이언트에게 일반 MQTT 메시지로 전송되며, 해당 주제의 신규 구독자에게도 전송되도록 저장됩니다.

MQTT 보관된 메시지에는 클라이언트가 액세스할 수 있도록 권한을 부여하기 위한 특정 정책 작업이 필요합니다. 보관된 메시지 정책 사용의 예는 보관된 메시지 정책 예 섹션을 참조하세요.

이 섹션에서는 보관된 메시지와 관련된 일반적인 작업에 대해 설명합니다.

- 보관된 메시지 생성

  클라이언트는 MQTT 메시지를 게시할 때 메시지가 보관되는지 여부를 결정합니다. 클라이언트는 Device SDK를 사용하여 메시지를 게시할 때 RETAIN 플래그를 설정할 수 있습니다. 애플리케이션 및 서비스는 Publish 작업을 사용하여 MQTT 메시지를 게시할 때 RETAIN 플래그를 설정할 수 있습니다.

  주제 이름당 하나의 메시지만 보관됩니다. 주제에 게시된 RETAIN 플래그가 설정된 새 메시지는 이전에 주제로 전송된 기존 보관된 메시지를 대체합니다.

  > **Note**
  >
  > RETAIN 플래그가 설정된 예약된 주제에는 게시할 수 없습니다.

- 보관된 메시지 주제 구독

  클라이언트는 다른 MQTT 메시지 주제와 마찬가지로 보관된 메시지 주제를 구독합니다. 보관된 메시지 주제를 구독하여 수신한 보관된 메시지에는 RETAIN 플래그가 설정되어 있습니다.

  보관된 메시지는 클라이언트가 보관된 메시지 주제에 0바이트 메시지 페이로드를 사용하여 보관된 메시지를 게시할 AWS IoT Core 때에서 삭제됩니다. 보관된 메시지 주제를 구독한 클라이언트도 0바이트 메시지를 수신합니다.

  보관된 메시지 주제를 포함하는 와일드카드 주제 필터를 구독하면 클라이언트가 보관된 메시지의 주제에 게시된 후속 메시지를 수신할 수 있지만 구독 시 보관된 메시지는 전송하지 않습니다.

> **Note**
>
> 구독 시 보관된 메시지를 수신하려면 구독 요청의 주제 필터가 보관된 메시지 주제와 정확히 일치해야 합니다.

보관된 메시지 주제를 구독하여 수신한 보관된 메시지에는 RETAIN 플래그가 설정되어 있습니다. 구독 후 구독 클라이언트가 수신한 보관된 메시지는 그렇지 않습니다.

- 보관된 메시지 검색

  보관된 메시지는 보관된 메시지와 함께 주제를 구독하면 클라이언트에 자동으로 전송됩니다. 클라이언트가 구독 시 보관된 메시지를 수신하려면 보관된 메시지의 정확한 주제 이름을 구독해야 합니다. 보관된 메시지 주제를 포함하는 와일드카드 주제 필터를 구독하면 클라이언트가 보관된 메시지의 주제에 게시된 후속 메시지를 수신할 수 있지만 구독 시 보관된 메시지는 배달하지 않습니다.

  서비스와 앱은 `ListRetainedMessages` 및 `GetRetainedMessage`를 호출하여 보관된 메시지를 나열하고 검색할 수 있습니다.

  클라이언트가 RETAIN 플래그를 설정하지 않고 보관된 메시지 주제에 메시지를 게시하는 것은 금지되지 않습니다. 이로 인해 보관된 메시지가 주제를 구독하여 받은 메시지와 일치하지 않는 등 예기치 않은 결과가 발생할 수 있습니다.

  MQTT 5에서는 보관된 메시지에 메시지 만료 간격이 설정되어 있고 보관된 메시지가 만료되는 경우 해당 주제를 구독하는 새 구독자는 구독에 성공해도 보관된 메시지를 받지 못합니다.

- 보관된 메시지 주제 나열

  `ListRetainedMessages`를 호출하여 보관된 메시지를 나열할 수 있으며 AWS IoT 콘솔에서 보관된 메시지를 볼 수 있습니다.

- 보관된 메시지 세부 정보 가져오기

  `GetRetainedMessage`를 호출하여 보관된 메시지 세부 정보를 가져오고 AWS IoT 콘솔에서 보관된 메시지를 볼 수 있습니다.

- Will 메시지 보관

  디바이스 연결 시 생성되는 MQTT Will 메시지는 `Will Retain` 필드에 `Connect Flag bits` 플래그를 설정하여 보관할 수 있습니다.

- 보관된 메시지 삭제

디바이스, 애플리케이션 및 서비스는 RETAIN 플래그가 설정된 메시지를 게시하고 삭제할 보관된 메시지의 주제 이름에 빈(0바이트) 메시지 페이로드를 게시하여 보관된 메시지를 삭제할 수 있습니다. 이러한 메시지는 보관된 메시지를에서 삭제 AWS IoT Core하고 주제에 대한 구독이 있는 클라이언트로 전송되지만 에서는 보관되지 않습니다 AWS IoT Core.

보관된 메시지는 AWS IoT 콘솔에서 보관된 메시지에 액세스하여 대화식으로 삭제할 수도 있습니다. AWS IoT 콘솔을 사용하여 삭제된 보관된 메시지는 또한 보관된 메시지의 주제를 구독한 클라이언트에 0바이트 메시지를 전송합니다.

보관된 메시지는 삭제된 후에는 복원할 수 없습니다. 클라이언트는 삭제된 메시지를 대신하기 위해 보존된 새 메시지를 게시해야 합니다.

- 보관된 메시지 디버깅 및 문제 해결

  AWS IoT 콘솔은 보관된 메시지 문제를 해결하는 데 도움이 되는 여러 도구를 제공합니다.

  - 보관된 메시지(Retained messages) 페이지

    AWS IoT 콘솔의 보관된 메시지(Retained messages) 페이지는 현재 리전의 계정에서 저장한 보관된 메시지의 목록(페이지가 매겨짐)을 제공합니다. 이 페이지에서 다음을 수행할 수 있습니다.

    - 메시지 페이로드, QoS, 수신 시간 등 보관된 각 메시지의 세부 정보를 봅니다.
    - 보관된 메시지의 내용을 업데이트합니다.
    - 보관된 메시지를 삭제합니다.

  - MQTT 테스트 클라이언트(MQTT test client)

    AWS IoT 콘솔의 MQTT 테스트 클라이언트(MQTT test client) 페이지는 MQTT 주제를 구독하고 게시할 수 있습니다. 게시 옵션을 사용하면 게시하는 메시지에 RETAIN 플래그를 설정하여 디바이스의 작동 방식을 시뮬레이션할 수 있습니다. MQTT 테스트 클라이언트를 사용하여, 클라이언트 연결 인터페이스를 통해 관리하는 연결된 클라이언트의 메시지를 모니터링할 수도 있습니다. 클라이언트 연결 관리에 대한 자세한 내용은 the section called “MQTT 연결 관리” 섹션을 참조하세요.

일부 예상치 못한 결과는 보관된 메시지가 구현되는 방식에 대한 이러한 측면의 결과일 수 있습니다 AWS IoT Core.

- 보관된 메시지 제한

  계정이 보존된 메시지의 최대 수를 저장한 경우, 일부 보존된 메시지가 삭제되고 보존된 메시지 수가 한도 아래로 떨어질 때까지 RETAIN 세트 및 페이로드로 게시된 메시지에 대해 제한된 응답을 AWS IoT Core 반환합니다.

- 보관된 메시지 전송 순서

  보관된 메시지와 구독된 메시지의 전송 순서는 보장되지 않습니다.

청구 및 보관된 메시지

클라이언트에서 RETAIN 플래그가 설정된 메시지를 AWS IoT 콘솔을 사용하거나 Publish를 직접 호출하여 게시하면 AWS IoT Core 요금 - 메시징에 설명된 추가 메시징 요금이 발생합니다.

클라이언트, AWS IoT 콘솔 또는를 사용하여 보관된 메시지를 검색하면 일반 API 사용 요금 외에 메시징 요금이 GetRetainedMessage 발생합니다. 추가 요금은 AWS IoT Core 요금 - 메시징에 설명되어 있습니다.

디바이스가 예기치 않게 연결 해제될 때 게시되는 MQTT Will 메시지에는 AWS IoT Core 요금 - 메시징에 설명된 메시징 요금이 발생합니다.

메시징 비용에 대한 자세한 내용은 AWS IoT Core 요금 - 메시징을 참조하세요.

MQTT 보관된 메시지와 MQTT 영구 세션 비교

보관된 메시지 및 영구 세션은 디바이스가 오프라인 상태일 때 게시된 메시지를 수신할 수 있도록 해주는 MQTT의 표준 기능입니다. 보관된 메시지는 영구 세션에서 게시할 수 있습니다. 이 섹션에서는 이러한 기능의 주요 측면과 이들이 함께 작동하는 방법을 설명합니다.

	보관된 메시지	영구 세션
주요 기능	보관된 메시지는 연결 후 대규모 디바이스 그룹을 구성하거나 알리는 데 사용할 수 있습니다. 보관된 메시지는 디바이스가 재연결 후 주제에 게시된 마지막 메시지만 수신하도록 하려는 경우에도 사용할 수 있습니다.	영구 세션은 연결이 간헐적이며 몇 가지 중요한 메시지를 놓칠 수 있는 디바이스에 유용합니다. 디바이스는 오프라인 상태에서 전송된 메시지를 수신하기 위해 영구 세션에 연결할 수 있습니다.
예시	보관된 메시지는 디바이스가 온라인 상태가 될 때 환경에 대	간헐적 연결로 셀룰러 네트워크를 통해 연결하는 디바이스

	보관된 메시지	영구 세션
	한 구성 정보를 제공할 수 있습니다. 초기 구성에는 구독해야 하는 다른 메시지 주제 목록이나 현지 시간대를 구성하는 방법에 대한 정보가 포함될 수 있습니다.	는 디바이스가 네트워크 범위를 벗어났거나 셀룰러 라디오를 꺼야 하는 동안 전송된 중요한 메시지를 놓치는 것을 방지하기 위해 영구 세션을 사용할 수 있습니다.
주제에 대한 초기 구독 시 수신된 메시지	보관된 메시지가 있는 주제를 구독하면 가장 최근에 보관된 메시지가 수신됩니다.	보관된 메시지가 없는 주제를 구독하면 주제에 게시될 때까지 메시지가 수신되지 않습니다.
재연결 후 구독된 주제	영구 세션이 없으면 클라이언트는 재연결한 후 주제를 구독해야 합니다.	구독한 주제는 재연결 후 복원됩니다.
재연결 후 수신된 메시지	보관된 메시지가 있는 주제를 구독하면 가장 최근에 보관된 메시지가 수신됩니다.	디바이스가 연결 해제된 동안 QOS = 1로 게시되고 QOS = 1로 구독된 모든 메시지는 디바이스가 다시 연결된 후에 전송됩니다.
데이터/세션 만료	MQTT 3에서는 보관된 메시지가 만료되지 않습니다. 대체 또는 삭제될 때까지 저장됩니다. MQTT 5에서 보관된 메시지는 설정된 메시지 만료 간격이 지나면 만료됩니다. 자세한 내용은 메시지 만료를 참조하세요.	클라이언트가 시간 제한 기간 내에 다시 연결하지 않으면 영구 세션이 만료됩니다. 영구 세션이 만료된 후 디바이스가 연결 해제된 동안 QOS = 1로 게시되고 QOS = 1로 구독된 클라이언트의 구독 및 저장된 메시지가 삭제됩니다. 만료된 메시지는 전송되지 않습니다. 영구 세션을 사용한 세션 만료에 대한 자세한 내용은 the section called “MQTT 지속적 세션” 섹션을 참조하세요.

영구 세션에 대한 자세한 내용은 the section called “MQTT 지속적 세션” 섹션을 참조하세요.

보관된 메시지의 경우 게시 클라이언트는 이전 세션이 있었는지 여부에 관계없이 메시지가 연결된 후 디바이스에 보관되고 전달되어야 하는지 여부를 결정합니다. 메시지를 저장할지에 대한 선택은 게시자가 하며 저장된 메시지는 QoS 0 또는 QoS 1 구독을 구독하는 현재 및 미래의 모든 클라이언트에게 전송됩니다. 보관된 메시지는 지정된 주제에 대해 한 번에 하나의 메시지만 보관합니다.

계정에 보관된 메시지의 최대 수를 저장한 경우 AWS IoT Core 는 일부 보관된 메시지가 삭제되고 보관된 메시지 수가 제한 아래로 떨어질 때까지 RETAIN이 설정되고 페이로드가 0바이트보다 큰 게시된 메시지에 대해 조절된 응답을 반환합니다.

MQTT 보존 메시지 및 AWS IoT 디바이스 섀도우

보관된 메시지와 디바이스 섀도우는 모두 디바이스의 데이터를 유지하지만 다르게 작동하고 다른 용도로 사용됩니다. 이 섹션에서는 이들의 유사점과 차이점을 설명합니다.

	보관된 메시지	디바이스 섀도우
메시지 페이로드에는 미리 정의된 구조 또는 스키마가 있음	구현에 의해 정의된 대로 MQTT는 해당 메시지 페이로드에 대한 구조 또는 스키마를 지정하지 않습니다.	AWS IoT 는 특정 데이터 구조를 지원합니다.
메시지 페이로드를 업데이트하면 이벤트 메시지가 생성됨	보관된 메시지를 게시하면 구독된 클라이언트에 메시지가 전송되지만 추가 업데이트 메시지는 생성되지 않습니다.	디바이스 섀도우를 업데이트하면 변경 사항을 설명하는 업데이트 메시지가 생성됩니다.
메시지 업데이트에 번호가 매겨짐	보관된 메시지는 자동으로 번호가 매겨지지 않습니다.	디바이스 섀도우 문서에는 자동 버전 번호와 타임스탬프가 있습니다.
메시지 페이로드가 사물 리소스에 연결됨	보관된 메시지는 사물 리소스에 연결되지 않습니다.	디바이스 섀도우는 사물 리소스에 연결됩니다.
메시지 페이로드의 개별 요소 업데이트	전체 메시지 페이로드를 업데이트하지 않으면 메시지의 개별 요소를 변경할 수 없습니다.	디바이스 섀도우 문서의 개별 요소는 전체 디바이스 섀도우 섀도 문서를 업데이트할 필요 없이 업데이트할 수 있습니다.

	보관된 메시지	디바이스 섀도우
구독 시 클라이언트가 메시지 데이터 수신	클라이언트는 보관된 메시지가 있는 주제를 구독한 후 자동으로 보관된 메시지를 받습니다.	클라이언트는 디바이스 섀도우 업데이트를 구독할 수 있지만 현재 상태를 의도적으로 요청해야 합니다.
인덱싱 및 검색 가능성	보관된 메시지는 검색을 위해 인덱싱되지 않습니다.	플릿 인덱싱은 검색 및 집계를 위해 디바이스 섀도우 데이터를 인덱싱합니다.

MQTT 마지막 유언 및 증거(LWT) 메시지

마지막 유언 및 증거(LWT)는 MQTT의 기능입니다. LWT를 사용하면 클라이언트가 메시지를 지정하여 브로커가 클라이언트가 정의한 주제에 메시지를 게시하고 시작되지 않은 연결 해제가 발생할 경우 해당 주제를 구독한 모든 클라이언트에게 메시지를 보냅니다. 클라이언트가 지정하는 메시지를 LWT 메시지 또는 유언 메시지라고 하며, 클라이언트가 정의하는 주제를 유언 주제라고 합니다. 디바이스가 브로커에 연결할 때 LWT 메시지를 지정할 수 있습니다. 연결 중에 `Connect Flag bits` 필드에 `Will Retain` 플래그를 설정하여 이러한 메시지를 보존할 수 있습니다. 예를 들어 `Will Retain` 플래그가 1로 설정되면 브로커의 관련 유언 주제에 유언 메시지가 저장됩니다. 자세한 내용은 유언 메시지를 참조하세요.

클라이언트 연결을 관리하는 경우에 클라이언트 연결을 해제할 때 LWT 메시지의 게시 여부를 제어할 수 있습니다. 자세한 내용은 the section called “MQTT 연결 관리” 단원을 참조하십시오.

브로커는 시작되지 않은 연결 해제가 발생할 때까지 유언 메시지를 저장합니다. 이 경우 브로커는 유언 주제를 구독한 모든 클라이언트에게 메시지를 게시하여 연결 해제를 알립니다. 클라이언트가 MQTT DISCONNECT 메시지를 사용하여 클라이언트가 시작한 연결 해제로 브로커와의 연결을 끊으면 브로커는 저장된 LWT 메시지를 게시하지 않습니다. 다른 모든 경우에는 LWT 메시지가 발송됩니다. 연결 해제 처리의 비동기적 특성으로 인해 LWT 메시지는 재연결 중에 순서대로 디스패치되지 않습니다. 수명 주기 이벤트를 사용하여 연결 상태 감지의 정확도를 높이는 것이 좋습니다. 이러한 이벤트는 타임스탬프 및 버전 번호와 같은 속성을 제공하여 out-of-order 이벤트를 관리하기 때문입니다. 브로커가 LWT 메시지를 전송하는 연결 해제 시나리오의 전체 목록은 연결/연결 해제 이벤트를 참조하세요.

connectAttributes 사용

`ConnectAttributes`을(를) 사용하면 IAM 정책의 연결 메시지에 사용할 속성(예: `PersistentConnect` 또는 `LastWill`)을 지정할 수 있습니다. `ConnectAttributes`를 사용하면

디바이스에 기본적으로 새 기능에 대한 액세스 권한을 부여하지 않는 정책을 만들 수 있습니다. 이 정책은 디바이스가 손상된 경우 도움이 될 수 있습니다.

`connectAttributes`는 다음 기능을 지원합니다.

`PersistentConnect`

클라이언트와 브로커 간의 연결이 중단될 때 `PersistentConnect` 기능을 사용하여 클라이언트가 연결 중 설정한 구독을 모두 저장할 수 있습니다.

`LastWill`

클라이언트가 예기치 않게 연결을 끊을 때 `LastWill` 기능을 사용하여 `LastWillTopic`에 메시지를 게시할 수 있습니다.

기본적인 정책은 비영구적 연결이며, 이 연결에 대해 전달된 속성이 없습니다. IAM 정책에 영구 연결을 하기 원하면 영구 연결을 지정해야 합니다.

클라이언트 연결을 관리할 때 연결된 클라이언트에 대한 연결 속성 및 세션 구성을 볼 수 있습니다. 자세한 내용은 the section called “MQTT 연결 관리” 단원을 참조하십시오.

`ConnectAttributes` 예제는 연결 정책 예제를 참조하세요.

MQTT 5에 지원되는 기능

AWS IoT Core MQTT 5에 대한 지원은에 설명된 대로 몇 가지 차이점이 있는 MQTT v5.0 사양을 기반으로 합니다the section called “AWS IoT MQTT 사양과의 차이점”.

AWS IoT Core 는 다음과 같은 MQTT 5 기능을 지원합니다.

- 공유 구독
- 클린 스타트 및 세션 만료
- 모든 ACK의 사유 코드
- 주제 별칭
- 메시지 만료
- 기타 MQTT 5 기능

공유 구독

AWS IoT Core 는 MQTT 3.1.1 및 MQTT 5 모두에 대한 공유 구독을 지원합니다. 공유 구독을 사용하면 여러 클라이언트가 한 주제에 대한 구독을 공유할 수 있으며 무작위 배포를 통해 해당 주제에 게시

된 메시지를 한 클라이언트만 수신할 수 있습니다. 공유 구독은 여러 구독자 간에 MQTT 메시지를 효과적으로 로드 밸런싱할 수 있습니다. 예를 들어 동일한 주제에 대해 게시하는 디바이스가 1,000개이고 이러한 메시지를 처리하는 백엔드 애플리케이션이 10개라고 가정해 보겠습니다. 이 경우에 백엔드 애플리케이션은 동일한 주제를 구독할 수 있으며 각각은 디바이스에서 공유 주제에 게시한 메시지를 무작위로 수신하게 됩니다. 이렇게 하면 해당 메시지의 로드를 효과적으로 '공유'할 수 있습니다. 또한 공유 구독을 사용하면 복원력이 향상됩니다. 백엔드 애플리케이션의 연결이 끊어지면 브로커가 그룹 내 나머지 구독자에게 로드를 분산합니다. 모든 구독자의 연결이 해제되면 메시지가 대기열에 추가됩니다.

메시지 대기열 기능은 메시지 전송 신뢰성을 높이기 위해 MQTT 3.1.1 및 MQTT 5 연결의 공유 구독에서 사용할 수 있습니다.

공유 구독을 사용하려면 클라이언트가 공유 구독의 주제 필터를 다음과 같이 구독합니다.

```
$share/{ShareName}/{TopicFilter}
```

- `$share`는 공유 구독의 주제 필터를 나타내는 리터럴 문자열이며, `$share`로 시작해야 합니다.
- `{ShareName}`은 구독자 그룹이 사용하는 공유 이름을 지정하는 문자열입니다. 공유 구독의 주제 필터는 ShareName을 포함하고 그 뒤에 / 문자가 와야 합니다. `{ShareName}`은 /, +, # 문자를 포함할 수 없습니다. `{ShareName}`의 최대 길이는 UTF-8 문자로 128자입니다.
- `{TopicFilter}`는 비공유 구독과 동일한 주제 필터 구문을 따릅니다. `{TopicFilter}`의 최대 길이는 UTF-8 문자로 256자입니다.
- `$share/{ShareName}/{TopicFilter}`에 대한 두 개의 필수 슬래시(/)는 주제 및 주제 필터의 최대 슬래시 수 제한에 포함되지 않습니다.

`{ShareName}/{TopicFilter}`가 동일한 구독은 동일한 공유 구독 그룹에 속합니다. 공유 구독 그룹을 여러 개 만들 수 있으며 그룹당 공유 구독 제한을 초과하지 않아야 합니다. 자세한 내용은 AWS 일반 참조의 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

다음 테이블은 비공유 구독과 공유 구독을 비교한 것입니다.

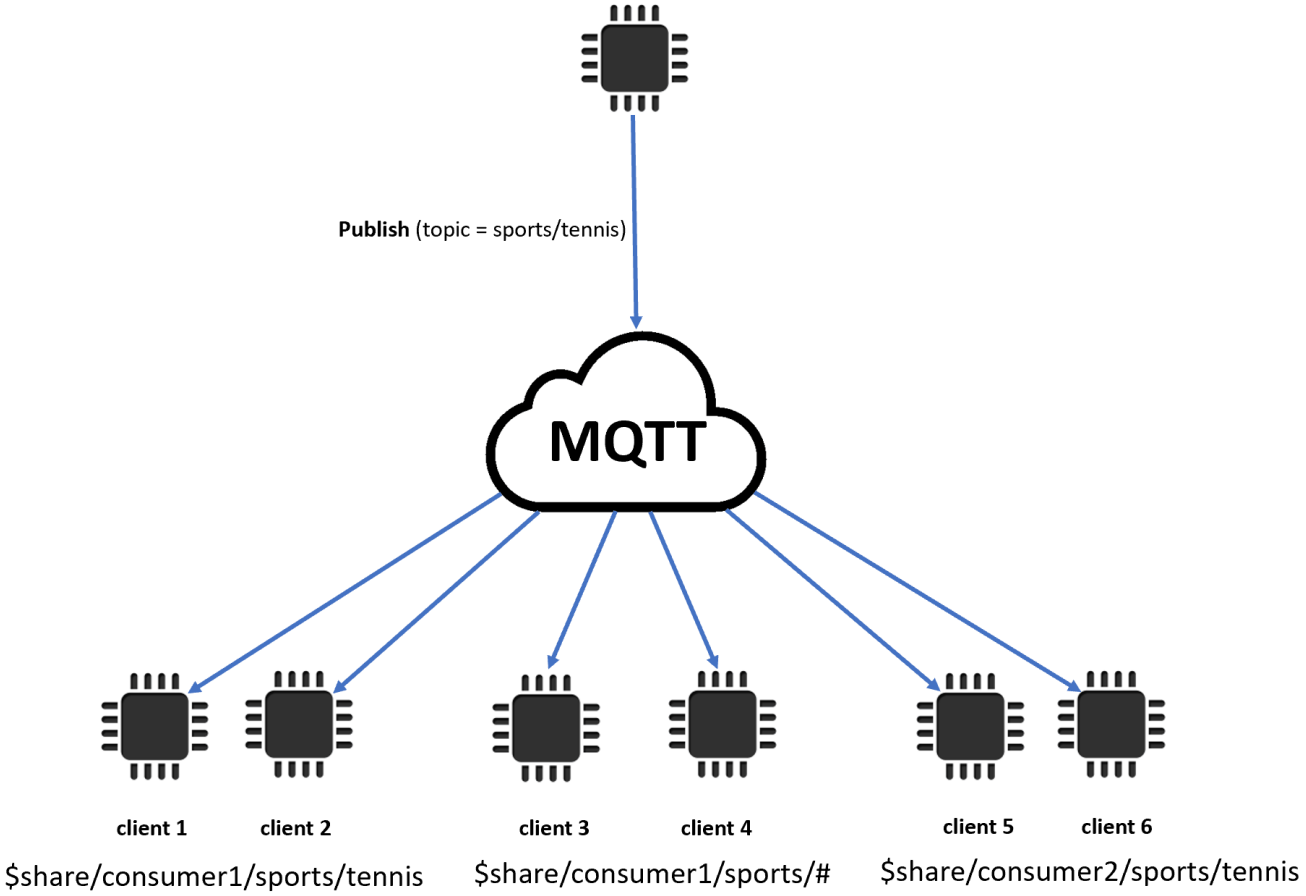
구독	설명	주제 필터 예시
비공유 구독	각 클라이언트는 게시된 메시지를 수신하기 위해 별도의 구독을 생성합니다. 메시지가 주제에 게시되면 해당 주제의 모든 구독자는 메시지 사본을 받게 됩니다.	`sports/tennis` `sports/#`

구독	설명	주제 필터 예시
공유 구독	여러 클라이언트가 한 주제에 대한 구독을 공유할 수 있으며 무작위 배포를 통해 해당 주제에 게시된 메시지를 한 클라이언트만 수신합니다.	`$share/consumer/sports/tennis` `$share/consumer/sports/#`

비공유 구독 흐름	공유 구독 흐름		
			

공유 구독 사용에 대한 중요 참고 사항

- 공유 구독자 그룹이 영구 세션 구독자로 구성된 경우, 공유 그룹의 모든 구독자가 연결 해제된 경우 또는 구독자가 연결별 초당 게시 요청 한도를 위반하는 경우, 승인되지 않은 QoS 1 메시지와 공유 구독 그룹에 게시되었지만 전송되지 않은 QoS 1 메시지가 대기열에 추가됩니다. 자세한 내용은 공유 구독 메시지 대기열을 참조하세요.
- 공유 구독 그룹에 게시된 QoS 0 메시지는 실패 시 삭제됩니다.
- 공유 구독자 그룹의 일부로 주제 패턴을 구독할 때 공유 구독은 보관된 메시지를 수신하지 않습니다. 공유 구독자가 있고 `RETAIN` 플래그가 설정된 주제에 게시된 메시지는 다른 게시 메시지와 마찬가지로 공유 구독자에게 전달됩니다.
- 공유 구독에 와일드카드 문자(# 또는 +)가 포함된 경우, 주제와 매칭되는 공유 구독이 여러 개 있을 수 있습니다. 이 경우 메시지 브로커는 게시 메시지를 복사하여 매칭되는 각 공유 구독의 클라이언트에 무작위로 전송합니다. 공유 구독의 와일드카드 동작은 다음 다이어그램에 설명되어 있습니다.



이 예시에는 게시 MQTT 주제 `sports/tennis`와 매칭되는 공유 구독이 세 개 있습니다. 메시지 브로커는 게시된 메시지를 복사하여 매칭되는 각 그룹의 클라이언트에 메시지를 무작위로 보냅니다.

클라이언트 1과 클라이언트 2는 `$share/consumer1/sports/tennis` 구독을 공유합니다.

클라이언트 3과 클라이언트 4는 `$share/consumer1/sports/#` 구독을 공유합니다.

클라이언트 5와 클라이언트 6은 `$share/consumer2/sports/tennis` 구독을 공유합니다.

공유 구독 제한에 대한 자세한 내용은 AWS 일반 참조의 AWS IoT Core 엔드포인트 및 할당량을 참조하세요. AWS IoT 콘솔에서 AWS IoT MQTT 클라이언트를 사용하여 공유 구독을 테스트하려면 섹션을 참조하세요???. 클라이언트 연결 관리 기능을 사용하여, 공유 구독을 포함해 연결된 클라이언트가 구독하는 주제를 볼 수도 있습니다. 자세한 내용은 the section called “MQTT 연결 관리” 단원을 참조하십시오. 공유 구독에 대한 자세한 내용은 MQTTv5.0 사양의 공유 구독을 참조하세요.

공유 구독 메시지 대기열

메시지 전송 신뢰성을 높이기 위해 공유 구독에는 온라인 구독자를 사용할 수 없을 때 메시지를 저장하는 메시지 대기열 기능이 포함됩니다. 공유 구독 그룹에 영구 세션이 있는 멤버가 하나 이상 포함된 경우 그룹에 대해 대기열 기능이 활성화됩니다. 메시지를 배포할 때 온라인 멤버가 수신자로 선택됩니다. QoS 1 메시지는 온라인에서 멤버를 찾을 수 없거나 구독자가 Publish requests per second per connection 한도를 초과할 때 대기열에 저장됩니다. 대기 중인 메시지는 기존 멤버가 영구 세션을 재개하거나 새 멤버가 그룹에 가입할 때 전달됩니다. 대기 중인 메시지는 구독에 따라 구독자에게 전달되는 다른 메시지와 함께 활성 그룹 구독자별로 초당 최대 20개의 대기 중인 메시지로 전달됩니다.

기본적으로 대기 중인 메시지 보존은 Persistent Session expiry period 할당량을 따릅니다. 그러나 인바운드 게시 메시지에 메시지 만료 간격(MEI)이 설정된 경우 MEI가 우선합니다. MEI가 있는 경우 MEI가 영구 세션 만료 기간에 관계없이 메시지 보존 기간을 결정합니다.

메시지 대기열 속도는 Queued messages per second per account 할당량에 따라 제한되며 메시지 수는 Maximum number of queued messages per shared subscription group 할당량에 따라 제한됩니다. 할당량을 보고 관리하려면 Service Quotas 콘솔에 액세스합니다.

AWS/Usage 네임스페이스ApproximateQueueDepth에서를 검색하여 대기열 CloudWatch를 모니터링하거나 다음 CLI 명령을 사용하여 각 공유 구독 그룹의 대기열 깊이와 연결된 지표를 나열할 수 있습니다.

```
aws cloudwatch list-metrics --namespace "AWS/Usage" --dimensions
 Name=Resource,Value='ApproximateQueueDepth'
```

이러한 한도를 초과하면, 한도에 도달하기 전에 이미 대기열에 있던 메시지만 유지됩니다. 한도를 초과하는 새 수신 메시지는 삭제됩니다. 시스템은 이전 대기 메시지를 최신 메시지로 대체하지 않습니다.

메시지 대기열은 CloudWatch 지표 및 CloudWatch Logs에 기록됩니다. CloudWatch 및 CloudWatch Logs에 작성된 항목에 대한 자세한 내용은 메시지 브로커 지표 및 대기 중 로그 항목 섹션을 참조하세요. 대기 중인 메시지는 여전히 표준 메시징 요금으로 청구됩니다. 메시지 요금에 대한 자세한 내용은 AWS IoT Core 요금을 참조하세요.

공유 구독 그룹의 세션 수명 주기

클린 세션이 그룹을 구독하면 그룹의 온라인 멤버가 됩니다. 구독을 취소하거나 연결을 해제하면 클린 세션이 그룹을 떠납니다.

영구 세션이 그룹을 구독하면 그룹의 온라인 멤버가 됩니다. 연결이 끊어지면, 여전히 그룹에 남아 있지만 그룹의 오프라인 멤버가 됩니다. 다시 연결되면, 다시 온라인 멤버가 됩니다. 영구 세션은 명시적으로 구독을 취소하거나 연결 해제 후 만료될 때 그룹을 떠납니다.

클린 스타트 및 세션 만료

클린 스타트 및 세션 만료를 사용하여 영구 세션을 보다 유연하게 처리할 수 있습니다. Clean Start 플래그는 기존 세션을 사용하지 않고 세션을 시작해야 하는지 여부를 나타냅니다. 세션 만료 간격은 연결이 끊어지는 후 세션을 유지하는 기간을 나타냅니다. 연결 해제 시 세션 만료 간격을 수정할 수 있습니다. 자세한 내용은 the section called “MQTT 지속적 세션” 단원을 참조하십시오.

모든 ACK의 사유 코드

사유 코드를 사용하여 오류 메시지를 더 쉽게 디버깅하거나 처리할 수 있습니다. 사유 코드는 브로커와의 상호 작용 유형(구독, 게시, 확인)에 따라 메시지 브로커에 의해 반환됩니다. 자세한 내용은 MQTT 사유 코드를 참조하세요. MQTT 사유 코드의 전체 목록은 MQTT v5 사양을 참조하세요.

주제 별칭

주제 이름을 2바이트 정수인 주제 별칭으로 대체할 수 있습니다. 주제 별칭을 사용하면 주제 이름 전송을 최적화하여 측정된 데이터 서비스의 데이터 비용을 잠재적으로 줄일 수 있습니다. AWS IoT Core의 기본 한도는 주제 별칭 8개입니다. 자세한 내용은 AWS 일반 참조의 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

메시지 만료

게시된 메시지에 메시지 만료 값을 추가할 수 있습니다. 이 값은 메시지 만료 간격(MEI)을 초 단위로 나타냅니다. 메시지가 해당 간격 내에 구독자에게 전송되지 않은 경우 메시지가 만료되고 제거됩니다. 메시지 만료 값을 설정하지 않으면 메시지가 만료되지 않습니다.

아웃바운드에서 구독자는 만료 간격에 시간이 남아 있는 메시지를 받게 됩니다. 예를 들어, 인바운드 게시 메시지의 메시지 만료 시간이 30초이고 20초 후에 해당 메시지가 구독자에게 라우팅되는 경우 메시지 만료 필드는 10으로 업데이트됩니다. 구독자가 수신한 메시지의 업데이트된 MEI는 0이 될 수 있습니다. 남은 시간이 999ms 이하가 되면 바로 0으로 업데이트되기 때문입니다.

에서 AWS IoT Core최소 MEI는 1입니다. 클라이언트 측에서 간격을 0으로 설정하면 1로 조정됩니다. 최대 메시지 만료 간격은 604,800(7일)입니다. 이보다 높은 값은 최대값으로 조정됩니다.

버전 간 통신에서 메시지 만료 동작은 인바운드 게시 메시지의 MQTT 버전에 따라 결정됩니다. 예를 들어, MQTT5를 통해 연결된 세션에서 전송된 메시지 만료가 있는 메시지는 MQTT3 세션을 구독한 디바이스의 경우 만료될 수 있습니다. 아래 표에는 메시지 만료 시 다음과 같은 유형의 게시 메시지가 어떻게 지원되는지 나와 있습니다.

게시 메시지 유형	메시지 만료 간격
정기 게시	서버가 지정된 시간 내에 메시지를 전송하지 못하면 만료된 메시지가 제거되고 구독자는 메시지를 받지 못합니다. 여기에는 디바이스가 QoS 1 메시지를 게시하지 않는 경우와 같은 상황이 포함됩니다.
보관	보관된 메시지가 만료되고 새 클라이언트가 주제를 구독하는 경우 해당 클라이언트는 구독 시 메시지를 받지 못합니다.
유언	유언 메시지 간격은 클라이언트 연결이 끊어지고 서버가 유언 메시지를 구독자에게 전달하려고 시도한 후에 시작됩니다.
대기열 메시지	클라이언트가 오프라인일 때 메시지 만료 간격이 있는 아웃바운드 QoS 1이 만료되면, 영구 세션이 재개된 후 클라이언트가 만료된 메시지를 받지 못합니다.

기타 MQTT 5 기능

서버 연결 해제

연결이 끊어지면 서버는 사전에 클라이언트에 DISCONNECT를 전송하여 연결 해제 사유 코드와 함께 연결 종료를 알릴 수 있습니다.

요청 및 응답

게시자는 수신 시 게시자가 지정한 주제로 수신자가 응답을 보내도록 요청할 수 있습니다.

최대 패킷 크기

클라이언트와 서버는 지원하는 최대 패킷 크기를 독립적으로 지정할 수 있습니다.

페이로드 형식 및 콘텐츠 유형

메시지를 게시할 때 페이로드 형식(이진, 텍스트)과 콘텐츠 유형을 지정할 수 있습니다. 이러한 정보는 메시지 수신자에게 전달됩니다.

MQTT 5 속성

MQTT 5 속성은 세션 만료 및 요청 및 응답 패턴과 같은 새로운 MQTT 5 기능을 지원하기 위해 MQTT 표준에 추가된 중요한 사항입니다. 에서 아웃바운드 메시지의 속성을 전달할 수 있는 규칙을 생성하거나 HTTP 게시를 사용하여 일부 새 속성으로 MQTT 메시지를 게시할 AWS IoT Core수 있습니다.

다음 표에는에서 AWS IoT Core 지원하는 모든 MQTT 5 속성이 나열되어 있습니다.

속성	설명	입력 유형	패킷
Payload Format Indicator	페이로드가 UTF-8 형식으로 지정되었는지 여부를 나타내는 부울 값입니다.	바이트	PUBLISH, CONNECT
콘텐츠 유형	페이로드의 내용을 설명하는 UTF-8 문자열입니다.	UTF-8 문자열	PUBLISH, CONNECT
Response Topic	수신자가 요청-응답 흐름의 일부로 게시해야 하는 주제를 설명하는 UTF-8 문자열입니다. 주제에는 와일드카드 문자가 포함되어서는 안 됩니다.	UTF-8 문자열	PUBLISH, CONNECT
Correlation Data	어떤 요청에 대한 응답 메시지인지 식별하기 위해 요청 메시지의 발신자가 사용하는 이진 데이터입니다.	바이너리	PUBLISH, CONNECT
User Property	UTF-8 문자열 페어입니다. 이 속성은 한 패킷에 여러 번 나타날 수 있습니다. 수신자는 전송된 순서대로 키-값 페어를 수신합니다.	UTF-8 문자열 페어	CONNECT, PUBLISH, Will Properties, SUBSCRIBE, DISCONNECT, UNSUBSCRIBE
Message Expiry Interval	메시지 만료 간격을 초 단위로 나타내는 4바이트 정수입니다. 이 값을 설정하지 않으면 메시지가 만료되지 않습니다.	4바이트 정수	PUBLISH, CONNECT
Session Expiry Interval	세션 만료 간격을 초 단위로 나타내는 4바이트 정수입니다.는 최대 7일을 AWS IoT Core 지원하고 기본 최대 1시간을 지원합니다. 설정한 값이 계정의 최대값을 초과하면 AWS IoT Core 는 CONNACK에 조정된 값을 반환합니다.	4바이트 정수	CONNECT, CONNACK, DISCONNECT

속성	설명	입력 유형	패킷
Assigned Client Identifier	디바이스에서 클라이언트 ID를 지정하지 않을 AWS IoT Core 때에서 생성되는 임의 클라이언트 ID입니다. 이 임의 클라이언트 ID는 현재 브로커가 관리하는 다른 세션에서 사용하지 않는 새 클라이언트 식별자여야 합니다.	UTF-8 문자열	CONNACK
Server Keep Alive	서버에서 지정한 연결 유지 시간을 나타내는 2바이트 정수입니다. 클라이언트가 연결 유지 시간 이상 비활성 상태일 경우 서버는 클라이언트의 연결을 끊습니다.	2바이트 정수	CONNACK
Request Problem Information	실패 시 사유 문자열을 보낼지 아니면 사용자 속성을 보낼지를 나타내는 부울 값입니다.	바이트	CONNECT
Receive Maximum	PUBACK을 수신하지 않고 전송할 수 있는 PUBLISH QOS > 0 패킷의 최대 수를 나타내는 2바이트 정수입니다.	2바이트 정수	CONNECT, CONNACK
Topic Alias Maximum	이 값은 주제 별칭으로 허용되는 가장 높은 값을 나타냅니다. 기본값은 0.	2바이트 정수	CONNECT, CONNACK
Maximum QoS	가 AWS IoT Core 지원하는 QoS의 최대값입니다. 기본값은 1입니다. AWS IoT Core 에서는 QoS 2를 지원하지 않습니다.	바이트	CONNACK
Retain Available	AWS IoT Core 메시지 브로커가 보관된 메시지를 지원하는지 여부를 나타내는 부울 값입니다. 기본값은 1입니다.	바이트	CONNACK
Maximum Packet Size	가 AWS IoT Core 수락하고 전송하는 최대 패킷 크기입니다. 128KB를 초과할 수 없습니다.	4바이트 정수	CONNECT, CONNACK

속성	설명	입력 유형	패킷
Wildcard Subscription Available	AWS IoT Core 메시지 브로커가 사용 가능한 와일드카드 구독을 지원하는지 여부를 나타내는 부울 값입니다. 기본값은 1입니다.	바이트	CONNACK
Subscription Identifier Available	AWS IoT Core 메시지 브로커가 구독 식별자 사용 가능을 지원하는지 여부를 나타내는 부울 값입니다. 기본값은 0입니다.	바이트	CONNACK

MQTT 사유 코드

MQTT 5는 사유 코드 응답과 함께 향상된 오류 보고를 도입합니다. AWS IoT Core 는 패킷별로 그룹화된 다음을 포함하되 이에 국한되지 않는 사유 코드를 반환할 수 있습니다. MQTT 5에 지원되는 사유 코드의 전체 목록은 MQTT 5 사양을 참조하세요.

CONNACK 사유 코드

값	16진수	사유 코드 이름	설명
0	0x00	Success	연결이 수락되었습니다.
128	0x80	Unspecified error	서버에서 실패의 사유를 밝히려 하지 않거나 다른 사유 코드가 적용되지 않습니다.
133	0x85	Client Identifier not valid	클라이언트 식별자가 유효한 문자열이지만 서버에서 허용되지 않습니다.
134	0x86	Bad User Name or Password	서버가 클라이언트에서 지정한 사용자 이름 또는 암호를 수락하지 않습니다.
135	0x87	Not authorized	클라이언트에 연결할 수 있는 권한이 없습니다.
144	0x90	Topic Name invalid	Will Topic Name이 올바른 형식이지만 서버에서 허용되지 않습니다.
151	0x97	할당량 초과	구현 또는 관리상 부과된 한도를 초과했습니다.

값	16진수	사유 코드 이름	설명
155	0x9B	QoS not supported	서버가 Will QoS에 설정된 QoS를 지원하지 않습니다.

PUBACK 사유 코드

값	16진수	사유 코드 이름	설명
0	0x00	Success	메시지가 수락되었습니다. QoS 1 메시지의 게시가 진행됩니다.
128	0x80	Unspecified error	수신자가 게시를 승인하지 않지만 사유를 밝히려고 하지 않거나 사유가 다른 값 중 하나와 일치하지 않습니다.
135	0x87	Not authorized	PUBLISH 권한이 없습니다.
144	0x90	Topic Name invalid	주제 이름이 올바른 형식이지만 클라이언트 또는 서버에서 허용되지 않습니다.
145	0x91	Packet identifier in use	패킷 식별자가 이미 사용 중입니다. 이는 클라이언트와 서버 간에 세션 상태가 일치하지 않음을 나타낼 수 있습니다.
151	0x97	할당량 초과	구현 또는 관리상 부과된 한도를 초과했습니다.

DISCONNECT 사유 코드

값	16진수	사유 코드 이름	설명
129	0x81	Malformed Packet	수신된 패킷이 이 사양을 준수하지 않습니다.
130	0x82	Protocol Error	예기치 않은 패킷이나 순서가 잘못된 패킷이 수신되었습니다.
135	0x87	Not authorized	요청이 승인되지 않았습니다.

값	16진수	사유 코드 이름	설명
139	0x8B	Server shutting down	서버가 종료되고 있습니다.
141	0x8D	Keep Alive timeout	연결 유지 시간의 1.5배 시간 동안 패킷이 수신되지 않아 연결이 종료되었습니다.
142	0x8E	Session taken over	동일한 ClientID를 사용하는 다른 연결이 연결되어 이 연결이 종료되었습니다.
143	0x8F	Topic Filter invalid	주제 필터가 올바른 형식이지만 서버에서 허용되지 않습니다.
144	0x90	Topic Name invalid	주제 이름이 올바른 형식이지만 이 클라이언트 또는 서버에서 허용되지 않습니다.
147	0x93	Receive Maximum exceeded	클라이언트 또는 서버가 PUBACK 또는 PUBCOMP를 전송하지 않고 수신할 수 있는 최대 수의 게시 항목보다 많은 항목을 수신했습니다.
148	0x94	Topic Alias invalid	클라이언트 또는 서버가 CONNECT 또는 CONNACK 패킷에서 보낸 최대 주제 별칭보다 큰 주제 별칭이 포함된 PUBLISH 패킷을 받았습니다.
151	0x97	할당량 초과	구현 또는 관리상 부과된 한도를 초과했습니다.
152	0x98	Administrative action	관리 조치로 인해 연결이 종료되었습니다.
155	0x9B	QoS not supported	클라이언트가 CONNACK의 최대 QoS에 지정된 QoS보다 큰 QoS를 지정했습니다.
161	0xA1	Subscription Identifiers not supported	서버가 구독 식별자를 지원하지 않으므로 구독이 수락되지 않았습니다.

SUBACK 사유 코드

값	16진수	사유 코드 이름	설명
0	0x00	Granted QoS 0	구독이 수락되었으며 전송되는 최대 QoS는 QoS 0입니다. 요청한 것보다 QoS가 낮을 수 있습니다.
1	0x01	Granted QoS 1	구독이 수락되었으며 전송되는 최대 QoS는 QoS 1입니다. 요청한 것보다 QoS가 낮을 수 있습니다.
128	0x80	Unspecified error	구독이 수락되지 않았으며, 서버에서 사유를 밝히려고 하지 않거나 다른 사유 코드가 적용되지 않습니다.
135	0x87	Not authorized	클라이언트는 이 구독을 할 수 있는 권한이 없습니다.
143	0x8F	Topic Filter invalid	주제 필터가 올바른 형식이지만 이 클라이언트에서 허용되지 않습니다.
145	0x91	Packet Identifier in use	지정된 패킷 식별자가 이미 사용 중입니다.
151	0x97	할당량 초과	구현 또는 관리상 부과된 한도를 초과했습니다.

UNSUBACK 사유 코드

값	16진수	사유 코드 이름	설명
0	0x00	Success	구독이 삭제됩니다.
128	0x80	Unspecified error	구독 취소를 완료할 수 없으며, 서버에서 사유를 밝히려고 하지 않거나 다른 사유 코드가 적용되지 않습니다.
143	0x8F	Topic Filter invalid	주제 필터가 올바른 형식이지만 이 클라이언트에서 허용되지 않습니다.
145	0x91	Packet Identifier in use	지정된 패킷 식별자가 이미 사용 중입니다.

AWS IoT MQTT 사양과의 차이점

메시지 브로커 구현은 MQTT v3.1.1 사양 및 MQTT v5.0 사양을 기반으로 하지만 다음과 같은 점에서 이러한 사양과 차이가 있습니다.

- AWS IoT 는 MQTT 3에 대해 PUBREC, PUBREL 및 PUBCOMP 패킷을 지원하지 않습니다.
- AWS IoT 는 MQTT 5에 대해 PUBREC, PUBREL, PUBCOMP 및 AUTH 패킷을 지원하지 않습니다.
- AWS IoT 는 MQTT 5 서버 리디렉션을 지원하지 않습니다.
- AWS IoT 는 MQTT 서비스 품질(QoS) 수준 0 및 1만 지원합니다. AWS IoT 는 QoS 수준 2의 게시 또는 구독을 지원하지 않습니다. 메시지 브로커는 QoS 수준 2가 요청될 경우 PUBACK 또는 SUBACK을 전송하지 않습니다.
- 에서 QoS 수준이 0인 주제를 AWS IoT구독하면 메시지가 0회 이상 전송됩니다. 메시지가 두 번 이상 전달될 수 있습니다. 여러 번 전달되는 메시지는 다른 패킷 ID를 사용하여 전송될 수 있습니다. 이러한 경우 DUP 플래그가 설정되지 않습니다.
- 연결 요청에 응답할 때 메시지 브로커는 CONNACK 메시지를 전송합니다. 이 메시지에는 연결이 이전 세션을 재개하는 것인지 여부를 나타내는 플래그가 포함됩니다.
- 추가 제어 패킷 또는 연결 끊기 요청을 전송하기 전에 클라이언트는 CONNACK 메시지가 AWS IoT 메시지 브로커로부터 디바이스에 수신될 때까지 기다려야 합니다.
- 클라이언트가 주제를 구독할 때 메시지 브로커가 SUBACK을 전송하는 시간과 클라이언트가 일치하는 새 메시지를 수신하기 시작하는 시간 사이에 지연이 있을 수 있습니다.
- 클라이언트가 주제 필터에서 와일드카드 문자 #을 사용하여 주제를 구독하면 주제 계층 구조에서 해당 수준 이하의 모든 문자열이 일치됩니다. 그러나 상위 주제는 일치하지 않습니다. 예를 들어 주제 `sensor/#`을 구독할 경우 주제 `sensor/`, `sensor/temperature`, `sensor/temperature/room1`에 게시된 메시지가 수신되지만 `sensor`에 게시된 메시지는 수신되지 않습니다. 와일드카드에 대한 자세한 내용은 주제 이름 필터 섹션을 참조하세요.
- 메시지 브로커는 클라이언트 ID를 사용하여 각 클라이언트를 식별합니다. 클라이언트 ID는 MQTT 페이로드의 일부로 클라이언트가 메시지 브로커로 전달합니다. 클라이언트 ID가 동일한 2개의 클라이언트를 동시에 메시지 브로커에 연결할 수 없습니다. 한 클라이언트가 다른 클라이언트에서 사용 중인 클라이언트 ID를 사용하여 메시지 브로커에 연결할 경우 새 클라이언트 연결이 수락되고 이전에 연결된 클라이언트가 연결 해제됩니다. API를 사용하여 수동으로 클라이언트를 연결 해체할 수도 있습니다. 자세한 내용은 the section called “MQTT 연결 관리” 단원을 참조하십시오.
- 드물지만 메시지 브로커가 다른 패킷 ID로 동일한 논리적 PUBLISH 메시지를 재전송할 수 있습니다.
- 와일드카드 문자가 포함된 주제 필터를 구독하면 보관된 메시지를 수신할 수 없습니다. 보관된 메시지를 수신하려면 구독 요청에 보관된 메시지 주제와 정확히 일치하는 주제 필터가 포함되어야 합니다.

- 메시지 브로커는 메시지 및 ACK가 수신되는 순서를 보장하지 않습니다.
- AWS IoT 에는 사양과 다른 제한이 있을 수 있습니다. 자세한 내용은 AWS IoT 참조 가이드의 AWS IoT Core 메시지 브로커 및 프로토콜 제한 및 할당량을 참조하세요.
- MQTT DUP 플래그는 지원되지 않습니다.

MQTT 연결 관리

AWS IoT Core 는 클라이언트 연결 해제 및 세션 관리 기능을 포함하여 MQTT 연결을 관리하는 데 도움이 되는 APIs를 제공합니다. 이러한 기능을 사용하면 AWS IoT 클라이언트 플릿을 더 잘 제어할 수 있으며 연결 문제를 해결하는 데 도움이 됩니다.

DeleteConnection API

`DeleteConnection` API를 사용하여 클라이언트 ID를 지정 AWS IoT Core 하여에서 MQTT 디바이스의 연결을 해제합니다. IDs 클라이언트를 연결 해제하면는 AWS IoT Core 메시지 브로커에서 클라이언트를 연결 AWS IoT Core 해제하고 선택적으로 세션 상태를 지우고 마지막 유언 및 증명(LWT) 메시지를 억제할 수 있습니다.

`DeleteConnection` API를 호출하면는 몇 가지 작업을 AWS IoT Core 수행하여 연결이 끊기지 않도록 합니다. AWS IoT Core 먼저는 MQTT 세션을 종료하기 위해 MQTT 연결 해제 메시지를 클라이언트에 전송합니다. 그러면 서비스가 기본 TCP/TLS 소켓을 닫습니다.

메시지 브로커는 `DISCONNECT` 패킷을 디바이스로 전송하고, 연결 해제 이유 `API_INITIATED_DISCONNECT`와 함께 수명 주기 이벤트를 게시합니다. 이렇게 하면 클라이언트가 아닌 API를 통해 또는 네트워크 문제로 인해 연결 해제가 시작된 시기를 식별할 수 있습니다. 가시성, 문제 해결 및 감사 목적으로 이러한 이벤트를 모니터링할 수 있습니다. 예를 들어 AWS IoT 규칙을 사용하여 이러한 이벤트를 처리하여 클라이언트가 연결 해제된 시기와 이유를 추적할 수 있습니다.

`cleanSession` 파라미터를 로 설정하면 모든 구독 및 대기 중인 메시지를 포함하여 클라이언트의 세션 상태를 `true` AWS IoT Core 제거합니다. 세션을 정리하면 영구 세션이 종료됩니다. 클라이언트가 영구 세션이고 `preventWillMessage` 파라미터가 `false`로 설정된 경우, 서비스는 가능하다면 LWT 메시지를 디스패치합니다. 이는 계획된 유지 관리 작업 중에 유용합니다.

`DeleteConnection` API를 직접 호출하면 연결 해제 프로세스가 즉시 시작되지만, 클라이언트가 연결 해제를 인식하는 정확한 타이밍은 네트워크 조건 및 클라이언트 구현에 따라 달라질 수 있습니다. 대부분의 클라이언트는 몇 초 이내에 연결 해제를 감지하지만, 네트워크 연결이 불량일 경우 클라이언트가 연결 해제된 것을 인식하는 데 더 오랜 시간이 걸릴 수 있습니다.

> **Note**
>
> 연결 해제를 강제로 실행하면 클라이언트가 새 세션 상태로 다시 인증하고 권한을 부여해야 합니다. API 직접 호출 자체는 클라이언트의 재연결을 방지하지 않습니다. 원하는 경우 `DeleteConnection` API 직접 호출을 실행하기 전에 클라이언트의 자격 증명 또는 정책을 추가로 수정해야 합니다.

요금에 대한 자세한 정보는 AWS IoT Core 요금을 참조하세요.

사용 사례

`DeleteConnection` API는 문제가 있는 동작을 보이거나 과도한 리소스를 소비하는 오작동하는 클라이언트를 관리하는 데 유용합니다. 연결을 강제로 끊으면 클라이언트가 적절한 인증 및 권한 부여로 연결을 다시 설정하는지 확인할 수 있으며, 이는 리소스 소비 문제를 해결하는 데 도움이 될 수 있습니다.

클라이언트 리디렉션 시나리오도 이 API의 이점을 누릴 수 있습니다. 클라이언트를 다른 엔드포인트로 리디렉션해야 하거나 프로그래밍 방식으로 연결을 해제하고 DNS 설정을 변경하여 다른 AWS IoT Core 엔드포인트에 다시 연결하도록 할 AWS 리전수 있습니다. 이 API는 멈춘 연결을 해결하거나 정상적인 작동을 방해할 수 있는 문제가 있는 세션 상태를 지우는 데 도움이 될 수 있습니다.

API 파라미터

`DeleteConnection` API는 다음 파라미터를 허용합니다.

clientId(필수)

연결을 해제할 MQTT 클라이언트의 고유 식별자입니다. 이 식별자는 URL 경로에 지정됩니다. 클라이언트 ID는 달러 기호($)로 시작할 수 없습니다.

> **Note**
>
> MQTT 클라이언트 ID는 HTTP 요청에서 유효하지 않은 문자가 포함될 수 있습니다. `DeleteConnection` API를 사용할 때는 MQTT에서는 유효하지만 HTTP에서는 유효하지 않은 클라이언트 ID의 모든 문자를 URL 인코딩(백분율 인코딩)해야 합니다. 여기에는 공백, 슬래시(/) 및 UTF-8 문자와 같은 특수 문자가 포함됩니다. 예를 들어 공백은 %20이 되고, 슬래시는 %2F가 되며, UTF-8 문자 ü는 %C3%BC가 됩니다. 올바른 인코딩은 MQTT 클라이언트 ID가 HTTP 기반 API 직접 호출에서 올바르게 전송되도록 합니다.

cleanSession(선택 사항)

연결 해제 시 클라이언트의 세션 상태를 제거할지 여부를 지정합니다. 구독 및 대기 중인 메시지를 포함한 모든 세션 정보를 삭제하려면 `true`로 설정합니다. 세션 상태를 보존하려면 `false`로 설정합니다. 기본적으로 이 값은 `false`(세션 상태 보존)로 설정됩니다. 클린 세션의 경우, 이 파라미터는 무시됩니다.

preventWillMessage(선택 사항)

연결 해제 시 사용 가능한 경우가 LWT(Last Will and Testament) 메시지를 AWS IoT Core 디스패치할지 여부를 제어합니다. LWT 메시지가 디스패치되지 않도록 하려면 `true`로 설정합니다. 디스패치를 허용하려면 `false`로 설정합니다. 기본적으로 이 값은 `false`로 설정됩니다. (사용 가능한 경우 LWT를 디스패치)

API 구문

`DeleteConnection` API는 다음 HTTP 요청 형식을 사용합니다.

```
DELETE /connections/<clientId>?
cleanSession=<cleanSession>&preventWillMessage=<preventWillMessage> HTTP/1.1
```

요청 예제:

```
// Basic disconnect (preserves session, allows LWT message)
DELETE /connections/myDevice123 HTTP/1.1

// Disconnect and clear session
DELETE /connections/myDevice123?cleanSession=TRUE HTTP/1.1

// Disconnect, clear session, and prevent LWT message
DELETE /connections/myDevice123?cleanSession=TRUE&preventWillMessage=TRUE HTTP/1.1
```

요청이 성공하면 응답 본문 없이 HTTP 200 OK가 반환됩니다.

> **Note**
>
> AWS Signature Version 4에서 요청에 서명하는 데 사용하는 서비스 이름은 iotdevicegateway입니다. `aws iot describe-endpoint --endpoint-type iot:Data-ATS` 명령을 사용하여 엔드포인트를 찾을 수도 있습니다.

필수 권한

DeleteConnection API를 사용하려면 다음과 같은 IAM 권한이 필요합니다.

```
iot:DeleteConnection
```

리소스 기반 정책을 사용하여 이 권한의 범위를 특정 클라이언트 ID로 지정할 수 있습니다. 예제:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteConnection",
            "Resource": "arn:aws:iot:region:account:client/myDevice*"
        }
    ]
}
```

중요 고려 사항

연결이 끊긴 클라이언트는 재연결을 방지하기 위해 추가 로직을 구현하지 않는 한 즉시 재연결을 시도할 수 있습니다. 연결 해제 작업은 현재 연결만 종료하고 연결 재연결을 방지하지 않습니다. 재연결을 방지해야 하는 경우, 클라이언트 측 로직을 구현하거나 디바이스의 자격 증명을 비활성화하는 것이 좋습니다.

속도 제한은 표준 API 속도 제한의 일부로 AWS IoT Core API에 적용됩니다. 대량 연결 해제 작업을 계획할 때는 이러한 제한을 고려하고, 스로틀링을 방지하기 위해 적절한 재시도 로직 및 일괄 처리 전략을 구현해야 합니다. 자세한 내용은 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

오류 응답

DeleteConnection API는 다음과 같은 오류 응답을 반환할 수 있습니다.

InvalidRequestException

요청이 유효하지 않습니다. 이는 클라이언트 ID 형식이 유효하지 않거나 달러 기호($) 접두사를 포함하거나 필수 파라미터가 누락된 경우에 발생할 수 있습니다.

ResourceNotFoundException

지정된 클라이언트 ID가 존재하지 않거나 현재 연결되어 있지 않으며, 영구 세션이 없습니다.

UnauthorizedException

이 작업을 수행할 수 있는 권한이 없습니다. 필수 `iot:DeleteConnection` 권한이 있는지 확인합니다.

ForbiddenException

호출자는 요청을 만들 권한이 없습니다. 이는 IAM 권한 부족 또는 리소스 기반 정책 제한으로 인해 발생할 수 있습니다.

ThrottlingException

속도가 제한을 초과했습니다. API 직접 호출 빈도를 줄이고 지수 백오프를 사용하여 적절한 재시도 로직을 구현합니다.

InternalFailureException

예상치 못한 오류가 발생했습니다. 잠시 후 요청을 다시 시도합니다.

ServiceUnavailableException

서비스를 일시적으로 사용할 수 없습니다. 잠시 후 요청을 다시 시도합니다.

HTTPS 게시

클라이언트는 HTTP 1.0 또는 1.1 프로토콜을 사용하여 REST API에 요청하여 메시지를 게시할 수 있습니다. HTTP 요청에 사용되는 인증 및 포트 매핑에 대해서는 ??? 단원을 참조하세요.

> **Note**
>
> HTTPS는 MQTT와 같이 `clientId` 값을 지원하지 않습니다. `clientId`는 MQTT를 사용할 때 이용할 수 있지만 HTTPS를 사용할 때는 이용할 수 없습니다.

HTTPS 메시지 URL

디바이스와 클라이언트는 클라이언트별 엔드포인트 및 주제별 URL에 대한 POST 요청을 함으로서 메시지를 게시합니다.

```
https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1
```

- *`IoT_data_endpoint`*는 AWS IoT 디바이스 데이터 엔드포인트입니다. 콘솔의 AWS IoT 사물 세부 정보 페이지 또는 클라이언트에서 명령을 사용하여 엔드포인트를 찾을 수 있습니다 AWS CLI .

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

엔드포인트는 `a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`과 같아야 합니다.

- *`url_encoded_topic_name`*은 전송되는 메시지의 전체 topic name입니다.

HTTPS 메시지 코드 예제

다음은 HTTPS 메시지를 AWS IoT에 전송하는 방법에 대한 몇 가지 예입니다.

Python (port 8443)

```
import requests
import argparse

# define command-line parameters
parser = argparse.ArgumentParser(description="Send messages through an HTTPS
 connection.")
parser.add_argument('--endpoint', required=True, help="Your AWS IoT data custom
 endpoint, not including a port. " +
                                                       "Ex: \"abcdEXAMPLExyz-
ats.iot.us-east-1.amazonaws.com\"")
parser.add_argument('--cert', required=True, help="File path to your client
 certificate, in PEM format.")
parser.add_argument('--key', required=True, help="File path to your private key, in
 PEM format.")
parser.add_argument('--topic', required=True, default="test/topic", help="Topic to
 publish messages to.")
parser.add_argument('--message', default="Hello World!", help="Message to publish. "
 +
                                                       "Specify empty string to
 publish nothing.")

# parse and load command-line parameter values
args = parser.parse_args()

# create and format values for HTTPS request
publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'
publish_msg = args.message.encode('utf-8')

# make request
publish = requests.request('POST',
```

```
            publish_url,
            data=publish_msg,
            cert=[args.cert, args.key])

# print results
print("Response status: ", str(publish.status_code))
if publish.status_code == 200:
        print("Response body:", publish.text)
```

Python (port 443)

```
import requests
import http.client
import json
import ssl

ssl_context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_CLIENT)
ssl_context.minimum_version = ssl.TLSVersion.TLSv1_2

# note the use of ALPN
ssl_context.set_alpn_protocols(["x-amzn-http-ca"])
ssl_context.load_verify_locations(cafile="./<root_certificate>")

# update the certificate and the AWS endpoint
ssl_context.load_cert_chain("./<certificate_in_PEM_Format>",
 "<private_key_in_PEM_format>")
connection = http.client.HTTPSConnection('<the ats IoT endpoint>', 443,
 context=ssl_context)
message = {'data': 'Hello, I'm using TLS Client authentication!'}
json_data = json.dumps(message)
connection.request('POST', '/topics/device%2Fmessage?qos=1', json_data)

# make request
response = connection.getresponse()

# print results
print(response.read().decode())
```

CURL

클라이언트 또는 디바이스의 curl을 사용하여 AWS IoT에 메시지를 전송할 수 있습니다.

curl을 사용하여 AWS IoT 클라이언트 디바이스에서 메시지를 보내려면

1. curl 버전을 확인합니다.

   a. 클라이언트의 명령 프롬프트에서 이 명령을 실행합니다.

      curl --help

      도움말 텍스트에서 TLS 옵션을 찾습니다. `--tlsv1.2` 옵션이 표시됩니다.

   b. `--tlsv1.2` 옵션이 표시되면 계속합니다.

   c. `--tlsv1.2` 옵션이 표시되지 않거나 `command not found` 오류가 발생하면 계속하기 전에 클라이언트에서 curl을 업데이트 또는 설치하거나 `openssl`을(를) 설치합니다.

2. 클라이언트에 인증서를 설치합니다.

   AWS IoT 콘솔에 클라이언트(사물)를 등록할 때 생성한 인증서 파일을 복사합니다. 계속하기 전에 클라이언트에 이 세 가지 인증서 파일이 있는지 확인합니다.

   - CA 인증서 파일(이 예에서는 *Amazon-root-CA-1.pem*).
   - 클라이언트의 인증서 파일(이 예에서는 *device.pem.crt*).
   - 클라이언트의 프라이빗 키 파일(이 예에서는 *private.pem.key*).

3. curl 명령줄을 생성하여 계정 및 시스템의 대체 가능한 값을 바꿉니다.

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
    --request POST \
    --data "{ \"message\": \"Hello, world\" }" \
    "https://IoT_data_endpoint:8443/topics/topic?qos=1"
```

--tlsv1.2

TLS 1.2(SSL)를 사용합니다.

--cacert *Amazon-root-CA-1.pem*

피어를 확인하는 데 필요한 경우 CA 인증서의 파일 이름과 경로입니다.

--cert *device.pem.crt*

필요한 경우 클라이언트의 인증서 파일 이름과 경로입니다.

--key *private.pem.key*

필요한 경우 클라이언트의 프라이빗 키 파일 이름과 경로입니다.

--request POST

HTTP 요청의 유형입니다(이 경우 POST).

--data "*{ \"message\": \"Hello, world\" }*"

게시할 HTTP POST 데이터입니다. 이 경우 JSON 문자열이며 내부 인용 부호는 백슬래시 문자(\)로 이스케이프됩니다.

"https://*IoT_data_endpoint*:8443/topics/*topic*?qos=1"

클라이언트 디바이스 AWS IoT 데이터 엔드포인트의 URL, HTTPS 포트 , 키워드 `:8443`, `topic`이 경우 `/topics/` 주제 이름 입니다. 쿼리 파라미터 `?qos=1`로 서비스 품질을 지정합니다.

4. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.

   MQTT 클라이언트를 사용하여 AWS IoT MQTT 메시지 보기의 지침에 따라 curl 명령에 사용된 *##(topic)*의 주제 이름으로 메시지를 구독하도록 콘솔을 구성하거나 #의 와일드카드 주제 필터를 사용합니다.

5. 명령을 테스트합니다.

   AWS IoT 콘솔의 테스트 클라이언트에서 주제를 모니터링하는 동안 클라이언트로 이동하여 3단계에서 만든 curl 명령줄을 실행합니다. 콘솔에 클라이언트의 메시지가 표시되어야 합니다.

MQTT 주제

MQTT 주제는 AWS IoT 메시지를 식별합니다. AWS IoT 클라이언트는 메시지 주제 이름을 지정하여 게시하는 메시지를 식별합니다. 클라이언트는 주제 필터를 AWS IoT Core에 등록하여 구독(수신)하려는 메시지를 식별합니다. 메시지 브로커는 주제 이름 및 주제 필터를 사용하여 메시지를 게시 클라이언트에서 구독 클라이언트로 라우팅합니다.

메시지 브로커는 주제를 사용하여 HTTPS 메시지 URL에 MQTT를 통해 전송된 메시지인지, HTTP를 통해 전송된 메시지인지 식별합니다.

는 일부 예약된 시스템 주제를 AWS IoT 지원하지만 대부분의 MQTT 주제는 사용자가 생성하고 관리합니다. 시스템 디자이너는 다음 섹션에 설명된 대로 주제를 AWS IoT 사용하여 게시 클라이언트에서 수신한 메시지를 식별하고 구독 클라이언트에 보낼 메시지를 선택합니다. 시스템에 대한 주제 네임스

페이스를 생성하기 전에 MQTT 주제의 특성을 검토하여 IoT 시스템에 가장 적합한 주제 이름 계층 구조를 생성합니다.

주제 이름

주제 이름과 주제 필터는 UTF-8로 인코딩된 문자열입니다. 계층 구조의 수준을 구분하기 위해 슬래시(/) 문자를 사용하여 정보의 계층 구조를 나타낼 수 있습니다. 예를 들어 이 주제 이름은 룸 1의 온도 센서를 참조할 수 있습니다.

- `sensor/temperature/room1`

이 예제에서는 다른 룸에 다음과 같은 주제 이름을 가진 다른 유형의 센서가 있을 수 있습니다.

- `sensor/temperature/room2`
- `sensor/humidity/room1`
- `sensor/humidity/room2`

> **Note**
>
> 시스템의 메시지에 대한 주제 이름을 고려할 때 다음 사항에 유의하세요.
>
> - 주제 이름과 주제 필터는 대소문자를 구분합니다.
> - 주제 이름에는 개인 식별 정보가 포함되어서는 안 됩니다.
> - $로 시작하는 주제 이름은 AWS IoT Core에서만 사용되는 예약된 주제입니다.
> - AWS IoT Core 는 또는 리전 간에 메시지를 보내거나 받을 AWS 계정수 없습니다.

주제 이름 및 네임스페이스를 디자인하는 방법에 대한 자세한 내용은 백서 AWS IoT Core에 대한 MQTT 주제 디자인을 참조하세요.

앱에서 메시지를 게시하고 구독하는 방법에 대한 예는 AWS IoT Core 자습서 시작하기 및 AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트에서 시작하세요.

> **Important**
>
> 주제 네임스페이스는 AWS 계정 및 리전으로 제한됩니다. 예를 들어 한 리전의 AWS 계정 에서 사용하는 `sensor/temp/room1` 주제는 다른 리전의 동일한 AWS 계정에서 사용하거나 다른 리전의 다른 계정 AWS 계정 에서 사용하는 `sensor/temp/room1` 주제와 다릅니다.

주제 ARN

모든 주제 Amazon 리소스 이름(ARN)의 형식은 다음과 같습니다.

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

예를 들어, `arn:aws:iot:us-west-2:123EXAMPLE456:topic/application/topic/device/sensor`는 `application/topic/device/sensor` 주제에 대한 ARN입니다.

주제 이름 필터

구독 클라이언트는 메시지 브로커에 주제 이름 필터를 등록하여 메시지 브로커가 클라이언트에 전송해야 하는 메시지 주제를 지정합니다. 주제 이름 필터는 단일 주제 이름을 구독하기 위한 단일 주제 이름일 수도 있고, 여러 주제 이름을 동시에 구독하기 위한 와일드카드 문자를 포함할 수도 있습니다.

게시 클라이언트는 게시하는 주제 이름에 와일드카드 문자를 사용할 수 없습니다.

다음 표에는 주제 필터에 사용할 수 있는 와일드카드 문자가 나열되어 있습니다.

주제 와일드카드

와일드카드 문자	일치 항목	참고
#	주제 계층 구조에서 해당 수준 이하의 모든 문자열입니다.	주제 필터의 마지막 문자여야 합니다. 주제 계층 구조에서 해당 수준의 유일한 문자여야 합니다. + 와일드카드 문자가 포함된 항목 필터에서 사용할 수 있습니다.

와일드카드 문자	일치 항목	참고
+	해당 문자를 포함하는 수준의 모든 문자열입니다.	주제 계층 구조에서 해당 수준의 유일한 문자여야 합니다. 주제 필터의 여러 수준에서 사용할 수 있습니다.

이전 센서 주제 이름 예제와 함께 와일드카드를 사용합니다.

- `sensor/#`를 구독할 경우 `sensor/`, `sensor/temperature`, `sensor/temperature/room1`에 게시된 메시지가 수신되지만 `sensor`에 게시된 메시지는 수신되지 않습니다.
- `sensor/+/room1`을 구독할 경우 `sensor/temperature/room1` 및 `sensor/humidity/room1`에 게시된 메시지가 수신되지만 `sensor/temperature/room2` 또는 `sensor/humidity/room2`에 게시된 메시지는 수신되지 않습니다.

주제 필터 ARN

모든 주제 필터 Amazon 리소스 이름(ARN)의 형식은 다음과 같습니다.

```
arn:aws:iot:aws-region:AWS-account-ID:topicfilter/TopicFilter
```

예, `arn:aws:iot:us-west-2:123EXAMPLE456:topicfilter/application/topic/+/sensor`은(는) `application/topic/+/sensor` 주제 필터에 대한 ARN입니다.

MQTT 메시지 페이로드

MQTT 메시지로 전송되는 메시지 페이로드는 중 하나에 대한 AWS IoT것이 아니라면에서 지정하지 않습니다the section called “예약된 주제”. 애플리케이션의 요구 사항을 수용하려면 프로토콜에 대한 AWS IoT Core Service Quotas의 제약 내에서 주제에 대한 메시지 페이로드를 정의하는 것이 좋습니다.

메시지 페이로드에 JSON 형식을 사용하면 AWS IoT 규칙 엔진이 메시지를 구문 분석하고 SQL 쿼리를 적용할 수 있습니다. 애플리케이션에서 메시지 페이로드에 SQL 쿼리를 적용할 규칙 엔진이 필요하지 않은 경우 애플리케이션에 필요한 모든 데이터 형식을 사용할 수 있습니다. SQL 쿼리에 사용되는 JSON 문서의 제한 사항 및 예약 문자에 대한 자세한 내용은 JSON 확장 섹션을 참조하세요.

MQTT 주제 및 해당 메시지 페이로드를 디자인하는 방법에 대한 자세한 내용은 AWS IoT Core에 대한 MQTT 주제 디자인을 참조하세요.

메시지 크기 제한이 서비스 할당량을 초과하면 이유가 PAYLOAD_LIMIT_EXCEEDED인 CLIENT_ERROR와 "메시지 페이로드가 메시지 유형의 크기 제한을 초과함(Message payload exceeds size limit for message type)"이 발생합니다. 메시지 크기 제한에 대한 자세한 내용은 AWS IoT Core 메시지 브로커 제한 및 할당량을 참조하세요.

예약된 주제

달러 기호($)로 시작하는 주제는에서 사용하도록 예약되어 있습니다 AWS IoT. 예약된 주제는 허용된 대로 구독하고 게시할 수 있지만 달러 기호로 시작하는 새 주제는 생성할 수 없습니다. 예약된 주제에 대해 지원되지 않는 게시 또는 구독 작업으로 인해 연결이 종료될 수 있습니다.

자산 모델 주제

Topic	허용된 클라이언트 작업	설명
$aws/sitewise/asset-models/*assetModelId* / assets/*assetId*/properties/*propertyId*	Subscribe	AWS IoT SiteWise 는이 주제에 자산 속성 알림을 게시합니다. 자세한 내용은 AWS IoT SiteWise 사용 설명서의 다른 AWS 서비스와 상호 작용을 참조하세요.

AWS IoT Device Defender 주제

이러한 메시지는 주제의 *#### ##*에 따라 간결한 이진 객체 표현(CBOR) 형식 및 JavaScript 객체 표기법(JSON) 형식의 응답 버퍼를 지원합니다. AWS IoT Device Defender 주제는 MQTT 게시만 지원합니다.

payload-format	응답 형식 데이터 유형
cbor	Concise Binary Object Representation(CBOR)
json	JavaScript Object Notation(JSON)

자세한 내용은 디바이스에서 지표 전송을 참조하세요.

Topic	허용된 작업	설명
$aws/things/*thingName* / defender/metrics/*payload-format*	게시	AWS IoT Device Defender 에이전트는이 주제에 지표를 게시합니다. 자세한 내용은 디바이스에서 지표 전송을 참조하세요.
$aws/things/*thingName* / defender/metrics/*payload-format* /accepted	Subscribe	AWS IoT 는 AWS IoT Device Defender 에이전트가 $aws/things/*thingName* / defender/metrics/*payload-format* 에 성공적인 메시지를 게시한 후이 주제에 게시합니다. 자세한 내용은 디바이스에서 지표 전송을 참조하세요.
$aws/things/*thingName* / defender/metrics/*payload-format* /rejected	Subscribe	AWS IoT 는 AWS IoT Device Defender 에이전트가 실패한 메시지를 $aws/things/*thingName* /defender/metrics/*payload-format* 에 게시한 후이 주제에 게시합니다. 자세한 내용은 디바이스에서 지표 전송을 참조하세요.

AWS IoT Core 디바이스 위치 주제

AWS IoT Core 디바이스 위치는 디바이스의 측정 데이터를 확인하고 IoT 디바이스의 예상 위치를 제공할 수 있습니다. 디바이스의 측정 데이터에는 GNSS, Wi-Fi, 셀룰러 및 IP 주소가 포함될 수 있습니다. 그런 다음 AWS IoT Core 디바이스 위치는 최상의 정확도를 제공하고 디바이스 위치 정보를 해결하는 측정 유형을 선택합니다. 자세한 내용은 AWS IoT Core 디바이스 위치 및 디바이스 위치 MQTT 주제를 사용하여 AWS IoT Core 디바이스 위치 확인 섹션을 참조하세요.

Topic	허용된 작업	설명
$aws/device_location/*customer_device_id* /get_position_estimate	게시	디바이스는이 주제에 게시하여 AWS IoT Core 디바이스 위치에서 확인할 스캔된 원시 측정 데이터를 가져옵니다.

Topic	허용된 작업	설명
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	Subscribe	AWS IoT Core 디바이스 위치가 성공적으로 확인되면 디바이스 위치가이 주제에 게시됩니다.
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	Subscribe	AWS IoT Core Device Location은 4xx 오류로 인해 디바이스 위치를 성공적으로 확인할 수 없는 경우이 주제에 게시합니다.

이벤트 주제

이벤트 메시지는 특정 이벤트가 발생하면 게시됩니다. 예를 들어 이벤트는 사물이 추가되거나, 업데이트되거나, 삭제될 때 레지스트리에 의해 생성됩니다. 이 표에는 다양한 AWS IoT 이벤트와 예약된 주제가 나와 있습니다.

Topic	허용된 클라이언트 작업	설명
$aws/events/certificates/registered/*caCertificateId*	Subscribe	AWS IoT 는가 인증서를 AWS IoT 자동으로 등록하고 클라이언트가 `PENDING_ACTIVATION` 상태의 인증서를 제공할 때이 메시지를 게시합니다. 자세한 내용은 the section called “자동 등록을 위해 클라이언트에 의한 첫 번째 연결 구성” 단원을 참조하십시오.
$aws/events/job/*jobID*/canceled	Subscribe	AWS IoT 는 작업이 취소될 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/job/*jobID*/cancellation_in_progress	Subscribe	AWS IoT 는 작업 취소가 진행 중일 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/events/job/*jobID*/completed	Subscribe	AWS IoT 는 작업이 완료되면이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/job/*jobID*/deleted	Subscribe	AWS IoT 는 작업이 삭제될 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/job/*jobID*/deletion_in_progress	Subscribe	AWS IoT 는 작업 삭제가 진행 중일 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/jobExecution/*jobID*/canceled	Subscribe	AWS IoT 는 작업 실행이 취소될 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/jobExecution/*jobID*/deleted	Subscribe	AWS IoT 는 작업 실행이 삭제될 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/jobExecution/*jobID*/failed	Subscribe	AWS IoT 는 작업 실행이 실패할 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/jobExecution/*jobID*/rejected	Subscribe	AWS IoT 는 작업 실행이 거부될 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/jobExecution/*jobID*/removed	Subscribe	AWS IoT 는 작업 실행이 제거될 때이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/jobExecution/*jobID*/succeeded	Subscribe	AWS IoT 는 작업 실행이 성공하면이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/events/jobExecution/*jobID*/timed_out	Subscribe	AWS IoT 는 작업 실행 시간이 초과되면이 메시지를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.
$aws/events/presence/connected/*clientId*	Subscribe	AWS IoT 는 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 연결될 때이 주제에 게시합니다 AWS IoT. 자세한 내용은 연결/연결 해제 이벤트 단원을 참조하십시오.
$aws/events/presence/disconnected/*clientId*	Subscribe	AWS IoT 는 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 연결 해제될 때이 주제에 게시합니다 AWS IoT. 자세한 내용은 연결/연결 해제 이벤트 단원을 참조하십시오.
$aws/events/subscriptions/subscribed/*clientId*	Subscribe	AWS IoT 는 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 MQTT 주제를 구독할 때이 주제에 게시합니다. 자세한 내용은 구독/구독 취소 이벤트 단원을 참조하십시오.
$aws/events/subscriptions/unsubscribed/*clientId*	Subscribe	AWS IoT 는 지정된 클라이언트 ID를 가진 MQTT 클라이언트가 MQTT 주제 구독을 취소할 때이 주제에 게시합니다. 자세한 내용은 구독/구독 취소 이벤트 단원을 참조하십시오.
$aws/events/thing/*thingName* /created	Subscribe	AWS IoT 는 *thingName* 사물이 생성될 때이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/events/thing/*thingName*/updated	Subscribe	AWS IoT 는 *thingName* 사물이 업데이트될 때이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thing/*thingName*/deleted	Subscribe	AWS IoT 는 *thingName* 사물이 삭제될 때이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingGroup/*thingGroupName*/created	Subscribe	AWS IoT 사물 그룹 *thingGroupName* 이 생성되면가이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingGroup/*thingGroupName*/updated	Subscribe	AWS IoT 사물 그룹 *thingGroupName* 이 업데이트되면가이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingGroup/*thingGroupName*/deleted	Subscribe	AWS IoT 사물 그룹 *thingGroupName* 이 삭제되면가이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingType/*thingTypeName*/created	Subscribe	AWS IoT 는 *thingTypeName* 사물 유형이 생성될 때이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingType/*thingTypeName*/updated	Subscribe	AWS IoT 는 *thingTypeName* 사물 유형이 업데이트될 때이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/events/thingType/*thingTypeName*/deleted	Subscribe	AWS IoT 는 *thingTypeName* 사물 유형이 삭제될 때이 주제에 게시합니다. 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingTypeAssociation/thing/*thingName*/*thingTypeName*	Subscribe	AWS IoT 는 사물 이름이 사물 유형*thingName* *thingTypeName* . 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/added	Subscribe	AWS IoT 는 사물 그룹 *thingName* 에 사물 thingName이 추가될 때이 주제에 게시합니다. *thingGroupName* 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingGroupMembership/thingGroup/*thingGroupName*/thing/*thingName*/removed	Subscribe	AWS IoT 사물 그룹 *thingName* 에서 사물 thingName이 제거되면가이 주제에 게시합니다. *thingGroupName* 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/added	Subscribe	AWS IoT 사물 그룹 parentThingGroupName에 사물 그룹 *childThingGroupName parentThingGroupName* . 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName* /childThingGroup/*childThingGroupName* /removed	Subscribe	AWS IoT 사물 그룹 *childThingGroupName* 에서 사물 그룹 *parentThingGroupName* . 자세한 내용은 the section called “레지스트리 이벤트” 단원을 참조하십시오.

플릿 프로비저닝 주제

> **Note**
>
> 이 표에서 수신으로 표시된 클라이언트 작업은 클라이언트가 주제를 구독했는지 여부에 관계없이가 요청한 클라이언트에 직접 AWS IoT 게시하는 주제를 나타냅니다. 클라이언트는 이러한 응답 메시지를 구독하지 않은 경우에도 이들을 수신해야 합니다. 이러한 응답 메시지는 메시지 브로커를 통과하지 않으며 다른 클라이언트나 규칙에 의해 구독될 수 없습니다.

이 메시지는 주제의 *payload-format*에 따라 Concise Binary Object Representation(CBOR) 형식 및 JavaScript Object Notation(JSON) 형식의 응답 버퍼를 지원합니다.

payload-format	응답 형식 데이터 유형
cbor	Concise Binary Object Representation(CBOR)
json	JavaScript Object Notation(JSON)

자세한 내용은 디바이스 프로비저닝 MQTT API 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/certificates/create/*payload-format*	게시	인증서 서명 요청(CSR)에서 인증서를 생성하려면 이 주제에 게시합니다.

Topic	허용된 클라이언트 작업	설명
$aws/certificates/create/*payload-format* /accepted	구독, 수신	AWS IoT 는 $aws/certificates/create/*payload-format* 을 성공적으로 호출한 후이 주제에 게시합니다.
$aws/certificates/create/*payload-format* /rejected	구독, 수신	AWS IoT 는 $aws/certificates/create/*payload-format* 에 대한 호출이 실패한 후이 주제에 게시합니다.
$aws/certificates/create-from-csr/*payload-format*	게시	CSR에서 인증서를 생성하려면 이 주제에 게시합니다.
$aws/certificates/create-from-csr/*payload-format* /accepted	구독, 수신	AWS IoT 는 $aws/certificates/create-from-csr/*payload-format* 에 대한 성공적인 호출을이 주제에 게시합니다.
$aws/certificates/create-from-csr/*payload-format* /rejected	구독, 수신	AWS IoT 는 $aws/certificates/create-from-csr/*payload-format* 에 대한 실패한 호출을이 주제에 게시합니다.
$aws/provisioning-templates/*templateName* /provision/*payload-format*	게시	사물을 등록하려면 이 주제에 게시합니다.
$aws/provisioning-templates/*templateName* /provision/*payload-format* /accepted	구독, 수신	AWS IoT 는 $aws/provisioning-templates/*templateName* /provision/*payload-format* 을 성공적으로 호출한 후이 주제에 게시합니다.
$aws/provisioning-templates/*templateName* /provision/*payload-format* /rejected	구독, 수신	AWS IoT 는 $aws/provisioning-templates/*templateName* /provision/*payload-format* 에 대한 호출이 실패한 후이 주제에 게시합니다.

작업 주제

> **Note**
>
> 이 표에서 수신으로 표시된 클라이언트 작업은 클라이언트가 주제를 구독했는지 여부에 관계없이가 요청한 클라이언트에 직접 AWS IoT 게시하는 주제를 나타냅니다. 클라이언트는 이러한 응답 메시지를 구독하지 않은 경우에도 이들을 수신해야 합니다.
> 이러한 응답 메시지는 메시지 브로커를 통과하지 않으며 다른 클라이언트나 규칙에 의해 구독될 수 없습니다. 작업 활동 관련 메시지를 구독하려면 `notify` 및 `notify-next` 주제를 사용합니다.
> 플릿 모니터링 솔루션에 대한 작업 및 `jobExecution` 이벤트 주제를 구독할 때 먼저 작업 및 작업 실행 이벤트를 사용하여 클라우드 측에서 이벤트를 수신해야 합니다.
> 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/things/*thingName* /jobs/get	게시	디바이스에서 이 주제에 메시지를 게시해 `GetPendingJobExecutions` 요청을 수행합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* /jobs/get/accepted	구독, 수신	디바이스에서 이 주제를 구독해 성공적인 `GetPendingJobExecutions` 요청 응답을 받습니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* /jobs/get/rejected	구독, 수신	`GetPendingJobExecutions` 요청이 거부되면 디바이스에서 응답을 받기 위해 이 주제를 구독합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* /jobs/start-next	게시	디바이스에서 이 주제에 메시지를 게시해 `StartNextPendingJobExecutio`

Topic	허용된 클라이언트 작업	설명
		n 요청을 수행합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* / jobs/start-next/accepted	구독, 수신	디바이스에서 이 주제를 구독해 성공적인 StartNextPendingJobExecutio n 요청 응답을 받습니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* / jobs/start-next/rejected	구독, 수신	StartNextPendingJobExecutio n 요청이 거부되면 디바이스에서 응답을 받기 위해 이 주제를 구독합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* / jobs/*jobId*/get	게시	디바이스에서 이 주제에 메시지를 게시해 DescribeJobExecution 요청을 수행합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* / jobs/*jobId*/get/accepted	구독, 수신	디바이스에서 이 주제를 구독해 성공적인 DescribeJobExecution 요청 응답을 받습니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* / jobs/*jobId*/get/rejected	구독, 수신	DescribeJobExecution 요청이 거부되면 디바이스에서 응답을 받기 위해 이 주제를 구독합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.

Topic	허용된 클라이언트 작업	설명
$aws/things/*thingName* / jobs/*jobId*/update	게시	디바이스에서 이 주제에 메시지를 게시해 UpdateJobExecution 요청을 수행합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오.
$aws/things/*thingName* /jobs/*jobId*/update/accepted	구독, 수신	디바이스에서 이 주제를 구독해 성공적인 UpdateJobExecution 요청 응답을 받습니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오. **Note** $aws/things/*thingName* / jobs/*jobId*/update에 게시하는 디바이스만 이 주제에 대한 메시지를 수신합니다.
$aws/things/*thingName* / jobs/*jobId*/update/rejected	구독, 수신	UpdateJobExecution 요청이 거부되면 디바이스에서 응답을 받기 위해 이 주제를 구독합니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하십시오. **Note** $aws/things/*thingName* / jobs/*jobId*/update에 게시하는 디바이스만 이 주제에 대한 메시지를 수신합니다.

Topic	허용된 클라이언트 작업	설명
$aws/things/*thingName* /jobs/notify	구독, 수신	디바이스에서 이 주제를 구독해 작업 실행이 사물에 대해 대기 중인 실행의 목록에서 추가되거나 제거될 때 알림을 받습니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하세요.
$aws/things/*thingName* /jobs/notify-next	구독, 수신	디바이스에서 이 주제를 구독해 사물에 대해 대기 중인 다음 작업 실행이 변경될 때 알림을 받습니다. 자세한 내용은 작업 디바이스 MQTT API 작업 단원을 참조하세요.
$aws/events/job/*jobId*/completed	Subscribe	작업 완료 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/job/*jobId*/canceled	Subscribe	작업 취소 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/job/*jobId*/deleted	Subscribe	작업 삭제 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/job/*jobId*/cancellation_in_progress	Subscribe	작업 취소 시작 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/job/*jobId*/deletion_in_progress	Subscribe	작업 삭제 시작 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/jobExecution/*jobId*/succeeded	Subscribe	작업 실행 성공 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.

Topic	허용된 클라이언트 작업	설명
$aws/events/jobExecution/*jobId*/failed	Subscribe	작업 실행 실패 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/jobExecution/*jobId*/rejected	Subscribe	작업 실행 거부 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/jobExecution/*jobId*/canceled	Subscribe	작업 실행 취소 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/jobExecution/*jobId*/timed_out	Subscribe	작업 실행 시간 초과 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/jobExecution/*jobId*/removed	Subscribe	작업 실행 제거 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하세요.
$aws/events/jobExecution/*jobId*/deleted	Subscribe	작업 실행 삭제 시 작업 서비스가 이 주제에 대해 이벤트를 게시합니다. 자세한 내용은 작업 이벤트 단원을 참조하십시오.

명령 주제

> **Note**
>
> 이 표에서 수신으로 표시된 클라이언트 작업은 클라이언트가 주제를 구독했는지 여부에 관계없이가 요청한 클라이언트에 직접 AWS IoT 게시하는 주제를 나타냅니다. 클라이언트는 이러한 응답 메시지를 구독하지 않은 경우에도 이들을 수신해야 합니다.
> 이러한 응답 메시지는 메시지 브로커를 통과하지 않으며 다른 클라이언트나 규칙에 의해 구독될 수 없습니다.

Topic	허용된 클라이언트 작업	설명
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/request/*<PayloadFormat>* $aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/request	구독, 수신	콘솔에서 또는 `StartCommandExecution` API를 사용하여 명령 실행을 시작하도록 요청하면 디바이스가 이 주제에 대한 메시지를 수신합니다. 이 경우에 *<devices>* 는 IoT 사물 또는 MQTT 클라이언트일 수 있고, *<DeviceID>* 는 IoT 사물 이름 또는 MQTT 클라이언트 ID일 수 있습니다.
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/response/*<PayloadFormat>*	게시	디바이스는 `UpdateCommandExecution` MQTT API를 사용하여 명령 실행에 대한 메시지를 이 주제에 게시합니다. 메시지는 콘솔에서 또는 `StartCommandExecution` API를 사용하여 명령 실행을 시작하라는 요청에 대한 응답으로 게시됩니다. 게시된 메시지는 JSON 또는 CBOR을 *<PayloadFormat>* 으로 사용합니다.
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/response/accepted/*<PayloadFormat>* $aws/commands/*<devices>*/*<DeviceID>*/executio	구독, 수신	클라우드 서비스가 명령 실행 결과를 성공적으로 처리한 경우 AWS IoT Device Management 는 /accepted 주제에 대한 응답을 게시합니다.

Topic	허용된 클라이언트 작업	설명
ns/*<ExecutionId>* / response/accepted		
$aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* / response/rejected /*<PayloadFormat>* $aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* / response/rejected	게시	클라우드 서비스가 명령 실행 결과를 처리하지 못한 경우 AWS IoT Device Management 는 /rejected 주제에 대한 응답을 게시합니다.

규칙 주제

Topic	허용된 클라이언트 작업	설명
$aws/rules/*ruleName*	게시	규칙을 직접 트리거하기 위해 디바이스 또는 애플리케이션이 이 주제에 게시합니다. 자세한 내용은 Basic Ingest를 통한 메시징 비용 절감 단원을 참조하세요.

보안 터널링 주제

Topic	허용된 클라이언트 작업	설명
$aws/things/*thing-name* / tunnels/notify	Subscribe	AWS IoT 는 IoT 에이전트가 원격 디바이스에서 로컬 프록시를 시작할 수 있도록

Topic	허용된 클라이언트 작업	설명
		이 메시지를 게시합니다. 자세한 내용은 the section called “IoT 에이전트 코드 조각” 단원을 참조하십시오.

섀도우 주제

이 섹션의 주제는 명명된 섀도우와 명명되지 않은 섀도우에서 사용됩니다. 각 섀도우에 사용되는 주제는 주제 접두사만 다릅니다. 이 표에서는 각 섀도우 유형에서 사용하는 주제 접두사를 보여줍니다.

ShadowTopicPrefix 값	섀도우 유형
$aws/things/*thingName* /shadow	명명되지 않은(클래식) 섀도우
$aws/things/*thingName* /shadow/name/*shadowName*	명명된 섀도우

전체 주제를 생성하려면 참조할 섀도우 유형에 *ShadowTopicPrefix*를 선택하고 *thingName* 및 *ShadowName*(해당하는 경우)을 해당 값으로 바꾸고 다음 표에 표시된 대로 주제 스텁을 추가합니다. 주제는 대/소문자를 구분합니다.

Topic	허용된 클라이언트 작업	설명
ShadowTopicPrefix /delete	게시/구독	섀도우를 삭제하기 위해 디바이스 또는 애플리케이션이 이 주제에 게시합니다. 자세한 내용은 /delete 단원을 참조하세요.
ShadowTopicPrefix /delete/accepted	Subscribe	섀도우가 삭제되면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /delete/accepted/ 단원을 참조하세요.

Topic	허용된 클라이언트 작업	설명
ShadowTopicPrefix /delete/rejected	Subscribe	섀도우 삭제 요청이 거부되면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /delete/rejected 단원을 참조하세요.
ShadowTopicPrefix /get	게시/구독	섀도우를 가져오기 위해 애플리케이션 또는 사물이 이 주제에 빈 메시지를 게시합니다. 자세한 내용은 디바이스 섀도우 MQTT 주제 단원을 참조하세요.
ShadowTopicPrefix /get/accepted	Subscribe	섀도우에 대한 요청이 성공적으로 이루어지면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /get/accepted 단원을 참조하세요.
ShadowTopicPrefix /get/rejected	Subscribe	섀도우 요청이 거부되면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /get/rejected 단원을 참조하세요.
ShadowTopicPrefix /update	게시/구독	섀도우를 업데이트하기 위해 사물 또는 애플리케이션이 이 주제에 게시합니다. 자세한 내용은 /update 단원을 참조하세요.
ShadowTopicPrefix /update/accepted	Subscribe	섀도우가 성공적으로 업데이트되면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /update/accepted 단원을 참조하세요.
ShadowTopicPrefix /update/rejected	Subscribe	섀도우 업데이트가 거부되면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /update/rejected 단원을 참조하세요.

Topic	허용된 클라이언트 작업	설명
ShadowTopicPrefix / update/delta	Subscribe	섀도우의 reported 섹션과 desired 섹션 간의 차이가 감지되면 디바이스 섀도우 서비스가 이 주제로 메시지를 전송합니다. 자세한 내용은 /update/delta 단원을 참조하세요.
ShadowTopicPrefix / update/documents	Subscribe	AWS IoT 는 섀도우에 대한 업데이트가 성공적으로 수행될 때마다이 주제에 상태 문서를 게시합니다. 자세한 내용은 / update/documents 단원을 참조하세요.

MQTT 기반 파일 전송 주제

> **Note**
>
> 이 표에서 수신으로 표시된 클라이언트 작업은 클라이언트가 주제를 구독했는지 여부에 관계없이가 요청한 클라이언트에 직접 AWS IoT 게시하는 주제를 나타냅니다. 클라이언트는 이러한 응답 메시지를 구독하지 않은 경우에도 이들을 수신해야 합니다. 이러한 응답 메시지는 메시지 브로커를 통과하지 않으며 다른 클라이언트나 규칙에 의해 구독될 수 없습니다.

이 메시지는 주제의 *payload-format*에 따라 Concise Binary Object Representation(CBOR) 형식 및 JavaScript Object Notation(JSON) 형식의 응답 버퍼를 지원합니다.

payload-format	응답 형식 데이터 유형
cbor	Concise Binary Object Representation(CBOR)
json	JavaScript Object Notation(JSON)

Topic	허용된 클라이언트 작업	설명
$aws/things/*ThingName* /streams/*StreamId*/data/*payload-format*	구독, 수신	AWS 디바이스의 "GetStream" 요청이 수락되면 MQTT 기반 파일 전송이이 주제에 게시됩니다. 페이로드는 스트림 데이터를 포함합니다. 자세한 내용은 디바이스에서 AWS IoT MQTT 기반 파일 전송 사용 단원을 참조하세요.
$aws/things/*ThingName* /streams/*StreamId*/get/*payload-format*	게시	디바이스는 “GetStream” 요청을 수행하기 위해 이 주제에 게시합니다. 자세한 내용은 디바이스에서 AWS IoT MQTT 기반 파일 전송 사용 단원을 참조하세요.
$aws/things/*ThingName* /streams/*StreamId*/description/*payload-format*	구독, 수신	AWS 디바이스의 "DescribeStream" 요청이 수락되면 MQTT 기반 파일 전송이이 주제에 게시됩니다. 페이로드는 스트림 설명을 포함합니다. 자세한 내용은 디바이스에서 AWS IoT MQTT 기반 파일 전송 사용 단원을 참조하세요.
$aws/things/*ThingName* /streams/*StreamId*/describe/*payload-format*	게시	디바이스는 “DescribeStream” 요청을 수행하기 위해 이 주제에 게시합니다. 자세한 내용은 디바이스에서 AWS IoT MQTT 기반 파일 전송 사용 단원을 참조하세요.
$aws/things/*ThingName* /streams/*StreamId*/rejected/*payload-format*	구독, 수신	AWS 디바이스의 "DescribeStream" 또는 "GetStream" 요청이 거부되면 MQTT 기반 파일 전송이이 주제에 게시됩니다. 자세한 내용은 디바이스에서 AWS IoT MQTT 기반 파일 전송 사용 단원을 참조하십시오.

예약된 주제 ARN

모든 예약된 주제 Amazon 리소스 이름(ARN)의 형식은 다음과 같습니다.

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

예, `arn:aws:iot:us-west-2:123EXAMPLE456:topic/$aws/things/thingName/jobs/get/accepted`은(는) 예약된 주제 `$aws/things/thingName/jobs/get/accepted`에 대한 ARN입니다.

도메인 구성

에서 도메인 구성을 사용하여 데이터 엔드포인트의 동작을 구성하고 관리할 AWS IoT Core수 있습니다. 도메인 구성을 사용하면 여러 AWS IoT Core 데이터 엔드포인트를 생성하고, 자체 FQDN(정규화된 도메인 이름) 및 관련 서버 인증서로 사용자 지정하고, 사용자 지정 권한 부여자를 연결할 수 있습니다. 자세한 내용은 사용자 지정 인증 및 권한 부여 단원을 참조하십시오.

> **Note**
>
> 이 기능은에서 사용할 수 없습니다 AWS GovCloud (US) AWS 리전.

이 장의 주제:

- 도메인 구성이란 무엇인가요?
- AWS 관리형 도메인 생성 및 구성
- 고객 관리형 도메인 생성 및 구성
- 도메인 구성 관리
- 도메인 구성에서 TLS 설정 구성
- OCSP 스테이플링을 위한 서버 인증서 구성

도메인 구성이란 무엇인가요?

에서 AWS IoT Core도메인 구성은 AWS IoT Core 데이터 엔드포인트에 대한 도메인(AWS 관리형 도메인 또는 고객 관리형 도메인)의 설정 및 구성을 나타냅니다. AWS IoT Core 또한는 디바이스가 통신할 수 있도록 AWS 계정(`iot:Data-ATS`)에 대한 기본 엔드포인트를 제공합니다 AWS IoT Core.

이 주제에서 수행할 작업

- 사용 사례
- 주요 개념
- 중요 정보

사용 사례

도메인 구성을 사용하여 다음과 같은 작업을 단순화할 수 있습니다.

- 디바이스를 로 마이그레이션합니다 AWS IoT Core.
- 별도의 디바이스 유형에 대해 별도의 도메인 구성을 유지하여 이기종 디바이스 플릿을 지원합니다.
- 애플리케이션 인프라를 로 마이그레이션하는 동안 브랜드 자격 증명(예: 도메인 이름을 통해)을 유지합니다 AWS IoT Core.

주요 개념

다음 개념은 도메인 구성 및 관련 개념에 대한 세부 정보를 제공합니다.

- 도메인 구성

  AWS IoT Core 엔드포인트에 대한 도메인의 설정 및 구성입니다.

- 기본 엔드포인트 도메인

  와 같은 기본 엔드포인트와 함께가 AWS IoT 제공하는 도메인입니다`iot:Data-ATS`. 기본 엔드포인트를 찾으려면 describe-endpoint 또는 describe-domain-configuration CLI 명령을 실행합니다. 또는 AWS IoT Core 콘솔로 이동하여 왼쪽 탐색의 연결에서 도메인 구성을 선택합니다. 기본 엔드포인트는 `iot:Data-ATS` 이름으로 나열됩니다.

- AWS 관리형 도메인

  에서 관리할 도메인 AWS 입니다. AWS 관리형 도메인을 선택하면 디바이스가에서 제공하는 데이터 엔드포인트를 사용하여 연결됩니다 AWS. AWS 는 도메인과 인증서를 관리합니다.

- 고객 관리형 도메인

  관리할 도메인입니다. 사용자 지정 도메인이라고도 합니다. 고객 관리형 도메인을 선택하면 디바이스가 사용자 지정 도메인 데이터 엔드포인트를 사용하여 연결됩니다. 도메인과 인증서를 관리합니

다. 고객 관리형 도메인을 사용하면 필요에 맞게 엔드포인트 URL로 조정할 수 있습니다. 예를 들어 사용자 지정 도메인 이름(`your-domain-name.com`)을 사용하거나 특정 액세스 정책을 적용할 수 있습니다.

- 인증 유형

  AWS IoT Core에 연결할 때 디바이스를 인증하기 위해 선택하는 인증 유형입니다. 도메인 구성을 생성할 때 인증 유형을 지정해야 합니다. 자세한 내용은 ??? 단원을 참조하십시오.

- 애플리케이션 프로토콜

  AWS IoT Core에 연결할 때 디바이스가 사용하는 애플리케이션 계층 프로토콜입니다. 도메인 구성을 생성할 때 애플리케이션 프로토콜을 지정해야 합니다. 자세한 내용은 ??? 단원을 참조하십시오.

중요 정보

AWS IoT Core 는 서버 이름 표시(SNI) TLS 확장을 사용하여 도메인 구성을 적용합니다. 디바이스를에 연결할 때 클라이언트는 다중 계정 등록 AWS IoT Core, 구성 가능한 엔드포인트, 사용자 지정 도메인 및 VPC 엔드포인트와 같은 기능에 필요한 서버 이름 표시(SNI) 확장을 전송할 수 있습니다. https://docs.aws.amazon.com//iot/latest/developerguide/IoTCore-VPC.html 또한 도메인 구성에 지정한 도메인 이름과 동일한 서버 이름을 전달해야 합니다. 이 서비스를 테스트하려면 GitHub의 AWS IoT Device SDK v2 버전을 사용합니다.

에서 여러 데이터 엔드포인트를 생성하는 경우 MQTT 주제 AWS 계정, 디바이스 섀도우 및 규칙과 같은 리소스를 공유합니다 AWS IoT Core .

AWS IoT Core 사용자 지정 도메인 구성을 위한 서버 인증서를 제공하면 인증서의 도메인 이름은 최대 4개입니다. 자세한 내용은 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

AWS 관리형 도메인 생성 및 구성

CreateDomainConfiguration API를 사용하여 AWS 관리형 도메인에서 구성 가능한 엔드포인트를 생성합니다. AWS 관리형 도메인의 도메인 구성은 다음으로 구성됩니다.

- `domainConfigurationName`

  도메인 구성과 값을 식별하는 사용자 정의 이름은에 고유해야 합니다 AWS 리전. `IoT:`로 시작하는 도메인 구성 이름은 기본 엔드포인트용으로 예약되어 있으므로 사용할 수 없습니다.
- `defaultAuthorizerName` (선택 사항)

엔드포인트에서 사용할 사용자 지정 권한 부여자의 이름입니다.

- `allowAuthorizerOverride` (선택 사항)

  요청의 HTTP 헤더에 다른 권한 부여자를 지정하여 디바이스가 기본 권한 부여자를 재정의할 수 있는지 여부를 지정하는 부울 값입니다. `defaultAuthorizerName`에 값이 지정된 경우 이 값이 필요합니다.

- `serviceType` (선택 사항)

  엔드포인트가 제공하는 서비스 유형입니다.는 `DATA` 서비스 유형 AWS IoT Core 만 지원합니다. `DATA`을(를) 지정할 때 AWS IoT Core 은(는) 엔드포인트 유형 `iot:Data-ATS`의 엔드포인트를 반환합니다. 구성 가능한 `iot:Data`(VeriSign) 엔드포인트는 생성할 수 없습니다.

- `TlsConfig` (선택 사항)

  도메인의 TLS 구성을 지정하는 객체입니다. 자세한 내용은 ??? 단원을 참조하십시오.

다음 예제 AWS CLI 명령은 `Data` 엔드포인트에 대한 도메인 구성을 생성합니다.

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
```

명령의 출력은 다음과 같을 수 있습니다.

```
{
    "domainConfigurationName": "myDomainConfigurationName",
    "domainConfigurationArn": "arn:aws:iot:us-east-1:123456789012:domainconfiguration/
myDomainConfigurationName/itihw"
}
```

고객 관리형 도메인 생성 및 구성

도메인 구성을 사용하면 AWS IoT Core에 연결할 사용자 지정 FQDN(정규화된 도메인 이름)을 지정할 수 있습니다. 고객 관리형 도메인(사용자 지정 도메인이라고도 함)을 사용하면 다음과 같은 많은 이점이 있습니다. 브랜딩 목적으로 고객에게 자체 도메인 또는 회사 자체 도메인을 공개할 수 있습니다. 새 브로커를 가리키도록 도메인을 쉽게 변경할 수 있습니다. 다중 테넌시를 지원하여 동일한 내에 다른 도메인이 있는 고객에게 서비스를 제공할 수 있습니다 AWS 계정. 자체 서버 인증서 세부 정보를 관리할 수 있습니다. 예: 인증서에 서명하는 데 사용되는 루트 인증 기관(CA) 서명 알고리즘, 인증서 체인 깊이 및 인증서의 수명 주기입니다.

사용자 지정 도메인으로 도메인 구성을 설정하는 워크플로우는 다음 세 단계로 구성됩니다.

1. 에 서버 인증서 등록 AWS Certificate Manager
2. 도메인 구성 생성
3. DNS 레코드 생성

인증서 관리자에 서버 AWS 인증서 등록

사용자 지정 도메인으로 도메인 구성을 생성하기 전에 AWS Certificate Manager (ACM)에 서버 인증서 체인을 등록해야 합니다. 다음 세 가지 유형의 서버 인증서를 사용할 수 있습니다.

- ACM에서 생성된 공인 인증서
- 퍼블릭 CA에서 서명한 외부 인증서
- 프라이빗 CA에서 서명한 외부 인증서

> **Note**
>
> AWS IoT Core 는 Mozilla의 신뢰할 수 있는 ca-bundle에 포함된 인증서를 퍼블릭 CA에서 서명한 것으로 간주합니다.

인증서 요구 사항

인증서를 가져오기 위한 사전 조건에서 인증서를 ACM으로 가져오기 위한 요구 사항을 확인하세요. 이러한 요구 사항 외에도 AWS IoT Core 는 다음과 같은 요구 사항을 추가합니다.

- 리프 인증서에는 값이 serverAuth(TLS 웹 서버 인증)인 확장 키 사용 x509 v3 확장을 반드시 포함해야 합니다. ACM에서 인증서를 요청하면 이 확장이 자동으로 추가됩니다.
- 최대 인증서 체인 깊이는 5개의 인증서입니다.
- 최대 인증서 체인 크기는 16KB입니다.
- 지원되는 암호화 알고리즘 및 키 크기에는 RSA 2,048비트(RSA_2048) 및 ECDSA 256비트(EC_prime256v1)가 포함됩니다.

여러 도메인에 인증서 한 개 사용

하나의 인증서를 사용하여 여러 하위 도메인을 처리하려는 경우 CN(일반 이름) 또는 SAN(주체 대체 이름) 필드에 와일드카드 도메인을 사용합니다. 예를 들어, **`*.iot.example.com`**을 사용하여

dev.iot.example.com, qa.iot.example.com 및 prod.iot.example.com을 처리합니다. 각 FQDN에는 자체 도메인 구성이 필요하지만 둘 이상의 도메인 구성에서 동일한 와일드카드 값을 사용할 수 있습니다. CN 또는 SAN은 사용자 지정 도메인으로 사용할 FQDN을 처리해야 합니다. SAN이 있는 경우 CN은 무시되고 SAN은 사용자 정의 도메인으로 사용하려는 FQDN을 포함해야 합니다. 이러한 처리는 정확히 일치 또는 와일드카드 일치를 통해 가능할 수 있습니다. 와일드카드 인증서가 검증되고 계정에 등록되면 해당 지역의 다른 계정은 인증서와 겹치는 사용자 지정 도메인을 만들 수 없습니다.

다음 단원에서는 각 유형의 인증서를 가져오는 방법에 대해 설명합니다. 모든 인증서 리소스에는 도메인 구성을 생성할 때 사용하는 ACM에 등록된 Amazon 리소스 이름(ARN)이 필요합니다.

ACM에서 생성된 공인 인증서

RequestCertificate API를 사용하여 사용자 지정 도메인의 공인 인증서를 생성할 수 있습니다. 이 방법으로 인증서를 생성하면 ACM에서 사용자 지정 도메인의 소유권을 확인합니다. 자세한 내용은 AWS Certificate Manager 사용 설명서의 공인 인증서 요청을 참조하세요.

퍼블릭 CA에서 서명한 외부 인증서

퍼블릭 CA(Mozilla의 신뢰할 수 있는 CA-번들에 포함된 CA)로 서명한 서버 인증서가 이미 있는 경우 ImportCertificate API를 사용하여 인증서를 ACM으로 직접 가져올 수 있습니다. 이 작업 및 필수 구성 요소 및 인증서 형식 요구 사항에 대한 자세한 내용은 인증서 가져오기를 참조하세요.

프라이빗 CA에서 서명한 외부 인증서

프라이빗 CA에서 서명하거나 자체 서명된 서버 인증서가 이미 있는 경우 해당 인증서를 사용하여 도메인 구성을 생성할 수 있지만 도메인 소유권을 확인하기 위해서 ACM에서 추가 공인 인증서를 생성하기도 해야 합니다. 이렇게 하려면 ImportCertificate API를 사용하여 ACM에 서버 인증서 체인을 등록하세요. 이 작업 및 필수 구성 요소 및 인증서 형식 요구 사항에 대한 자세한 내용은 인증서 가져오기를 참조하세요.

검증 인증서 생성

인증서를 ACM으로 가져온 후 RequestCertificate API를 사용하여 사용자 지정 도메인의 공인 인증서를 생성합니다. 이 방법으로 인증서를 생성하면 ACM에서 사용자 지정 도메인의 소유권을 확인합니다. 자세한 내용은 공인 인증서 요청을 참조하세요. 도메인 구성을 생성할 때 이 공인 인증서를 검증 인증서로 사용합니다.

도메인 구성 생성

CreateDomainConfiguration API를 사용하여 사용자 지정 도메인의 구성 가능한 엔드포인트를 생성합니다. 사용자 지정 도메인의 도메인 구성은 다음으로 구성됩니다.

- `domainConfigurationName`

  도메인 구성을 식별하는 사용자 정의 이름입니다. `IoT:`로 시작하는 도메인 구성 이름은 기본 엔드포인트용으로 예약되어 있으므로 사용할 수 없습니다. 또한이 값은에 고유해야 합니다 AWS 리전.

- `domainName`

  디바이스가 연결하는 데 사용하는 FQDN입니다 AWS IoT Core. AWS IoT Core 는 서버 이름 표시(SNI) TLS 확장을 활용하여 도메인 구성을 적용합니다. 디바이스는 연결할 때 이 확장을 사용하고 도메인 구성에 지정된 도메인 이름과 동일한 서버 이름을 전달해야 합니다.

- `serverCertificateArns`

  ACM에 등록한 서버 인증서 체인의 ARN은 AWS IoT Core 현재 하나의 서버 인증서만 지원합니다.

- `validationCertificateArn`

  사용자 지정 도메인의 소유권을 확인하기 위해 ACM에서 생성한 퍼블릭 인증서의 ARN입니다. 공개적으로 서명되거나 ACM에서 생성된 서버 인증서를 사용하는 경우에는 이 인수가 필요하지 않습니다.

- `defaultAuthorizerName (optional)`

  엔드포인트에서 사용할 사용자 지정 권한 부여자의 이름입니다.

- `allowAuthorizerOverride`

  요청의 HTTP 헤더에 다른 권한 부여자를 지정하여 디바이스가 기본 권한 부여자를 재정의할 수 있는지 여부를 지정하는 부울 값입니다. `defaultAuthorizerName`에 값이 지정된 경우 이 값이 필요합니다.

- `serviceType`

  AWS IoT Core 는 현재 `DATA` 서비스 유형만 지원합니다. 를 지정하면는 엔드포인트 유형이 인 엔드포인트를 `DATA` AWS IoT 반환합니다`iot:Data-ATS`.

- `TlsConfig` (선택 사항)

  도메인의 TLS 구성을 지정하는 객체입니다. 자세한 내용은 ??? 단원을 참조하십시오.

- `serverCertificateConfig` (선택 사항)

  도메인의 서버 인증서 구성을 지정하는 객체입니다. 자세한 내용은 ??? 단원을 참조하십시오.

다음 AWS CLI 명령은 iot.example.com 도메인 구성을 생성합니다.

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
--domain-name "iot.example.com" --server-certificate-arns serverCertARN --validation-
certificate-arn validationCertArn
```

> **Note**
>
> 도메인 구성을 생성한 후가 사용자 지정 서버 인증서를 AWS IoT Core 제공할 때까지 최대 60분이 걸릴 수 있습니다.

자세한 내용은 ??? 단원을 참조하십시오.

DNS 레코드 생성

서버 인증서 체인을 등록하고 도메인 구성을 생성한 후 사용자 지정 도메인이 AWS IoT 도메인을 가리키도록 DNS 레코드를 생성합니다. 이 레코드는 유형의 AWS IoT 엔드포인트를 가리켜야 합니다`iot:Data-ATS`. DescribeEndpoint API를 사용하여 엔드포인트를 얻을 수 있습니다.

다음 AWS CLI 명령은 엔드포인트를 가져오는 방법을 보여줍니다.

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

`iot:Data-ATS` 엔드포인트를 가져온 후 사용자 지정 도메인에서이 AWS IoT 엔드포인트로 `CNAME` 레코드를 생성합니다. 동일한에 여러 사용자 지정 도메인을 생성하는 경우 AWS 계정는이 도메인을 동일한 `iot:Data-ATS` 엔드포인트에 별칭으로 지정합니다.

문제 해결

디바이스를 사용자 지정 도메인에 연결하는 데 문제가 있는 경우 AWS IoT Core 가 서버 인증서를 수락하고 적용했는지 확인합니다. AWS IoT Core 콘솔 또는를 사용하여가 인증서를 수락 AWS IoT Core 했는지 확인할 수 있습니다 AWS CLI.

AWS IoT Core 콘솔을 사용하려면 도메인 구성 페이지로 이동하여 도메인 구성 이름을 선택합니다. 서버 인증서 세부 정보 섹션에서 상태 및 상태 세부 정보를 확인합니다. 인증서가 유효하지 않은 경우 ACM에서 이전 섹션에 나열된 인증서 요구 사항에 부합하는 인증서로 바꿉니다. 인증서의 ARN이 동일한 경우 AWS IoT Core 가 인증서를 선택하여 자동으로 적용합니다.

를 사용하여 인증서 상태를 확인하려면 DescribeDomainConfiguration API를 AWS CLI호출하고 도메인 구성 이름을 지정합니다.

> **Note**
>
> 인증서가 유효하지 않은 경우 AWS IoT Core 는 마지막 유효한 인증서를 계속 제공합니다.

다음 openssl 명령을 사용하여 엔드포인트에서 제공되는 인증서를 확인할 수 있습니다.

```
openssl s_client -connect custom-domain-name:8883 -showcerts -servername
custom-domain-name
```

도메인 구성 관리

이 주제에서는 도메인 구성 리소스를 관리하는 데 필요한 주요 작업을 다룹니다. 또한 ListDomainConfigurations, DescribeDomainConfiguration UpdateDomainConfiguration및 DeleteDomainConfiguration API를 사용하여 기존 구성의 수명 주기를 관리할 수 있습니다.

이 주제에서 수행할 작업

- 도메인 구성 보기
- 도메인 구성 업데이트
- 도메인 구성 삭제
- 사용자 지정 도메인에서 인증서 교체

도메인 구성 보기

에서 모든 도메인 구성의 페이지 매김 목록을 반환하려면 ListDomainConfigurations API를 AWS 계정 사용합니다. DescribeDomainConfiguration API를 사용하여 특정 도메인 구성의 세부 정보를 볼 수 있습니다. 이 API는 단일 `domainConfigurationName` 파라미터를 사용하고 지정된 구성의 세부 정보를 반환합니다.

예제

도메인 구성 업데이트

도메인 구성의 상태 또는 사용자 지정 권한 부여자를 업데이트하려면 UpdateDomainConfiguration API를 사용합니다. 상태를 `ENABLED` 또는 `DISABLED`로 설정할 수 있습니다. 도메인 구성을 비활성화하면 해당 도메인에 연결된 디바이스에서 인증 오류가 발생합니다. 현재 도메인 구성에서 서버 인증서를 업데이트할 수 없습니다. 도메인 구성의 인증서를 변경하려면 해당 인증서를 삭제하고 다시 생성해야 합니다.

예제

도메인 구성 삭제

도메인 구성을 삭제하기 전에 UpdateDomainConfiguration API를 사용하여 상태를 DISABLED로 설정합니다. 이렇게 하면 엔드포인트가 실수로 삭제되는 것을 방지할 수 있습니다. 도메인 구성을 비활성화한 후 DeleteDomainConfiguration API를 사용하여 삭제합니다. 도메인을 삭제하려면 먼저 7일 동안 관리 AWS형 도메인을 DISABLED 상태로 두어야 합니다. 사용자 지정 도메인을 DISABLED 상태로 지정한 다음 즉시 삭제할 수 있습니다.

예제

도메인 구성을 삭제하면는 더 이상 해당 사용자 지정 도메인과 연결된 서버 인증서를 제공하지 AWS IoT Core 않습니다.

사용자 지정 도메인에서 인증서 교체

정기적으로 서버 인증서를 업데이트된 인증서로 교체해야 할 수 있습니다. 이 작업을 수행하는 비율은 인증서의 유효 기간에 따라 다릅니다. AWS Certificate Manager (ACM)를 사용하여 서버 인증서를 생성한 경우 인증서를 자동으로 갱신하도록 설정할 수 있습니다. ACM이 인증서를 갱신하면는 새 인증서를 AWS IoT Core 자동으로 선택합니다. 추가 작업을 수행할 필요가 없습니다. 다른 소스에서 서버 인증서를 가져온 경우 ACM으로 다시 가져와 교체할 수 있습니다. 인증서 다시 가져오기에 대한 자세한 내용은 인증서 다시 가져오기를 참조하세요.

> **Note**
>
> AWS IoT Core 는 다음 조건에서만 인증서 업데이트를 선택합니다.
>
> - 새 인증서에 이전 인증서와 동일한 ARN이 있는 경우.
> - 새 인증서에 이전 인증서와 동일한 서명 알고리즘, 일반 이름 또는 주체 대체 이름이 있는 경우.

도메인 구성에서 TLS 설정 구성

AWS IoT Core 는 도메인 구성에서 TLS 1.2 및 TLS 1.3에 대한 TLS(전송 계층 보안) 설정을 사용자 지정할 수 있는 사전 정의된 보안 정책을 제공합니다. 보안 정책은 클라이언트와 서버 간의 TLS 협상 중에 지원되는 프로토콜과 암호를 결정하는 TLS 프로토콜과 해당 암호의 조합입니다. 지원되는 보안 정책을 사용하면 디바이스의 TLS 설정을 보다 유연하게 관리하고, 새 디바이스를 연결할 때 최신 보안 조치를 적용하고, 기존 디바이스에 대해 일관된 TLS 구성을 유지할 수 있습니다.

다음 테이블에는 보안 정책, TLS 버전 및 지원 리전이 설명되어 있습니다.

보안 정책 이름	지원됨 AWS 리전
IoTSecurityPolicy_ TLS13_1_3_2022_10	모두 AWS 리전
IoTSecurityPolicy_ TLS13_1_2_2022_10	모두 AWS 리전
IoTSecurityPolicy_ TLS12_1_2_2022_10	모두 AWS 리전
IoTSecurityPolicy_ TLS12_1_0_2016_01	ap-east-1, ap-northeast-2, ap-south-1, ap-southeast-2, ca-central-1, cn-north-1, cn-northwest-1, eu-north-1, eu-west-2, eu-west-3, me-south-1, sa-east-1, us-east-2, us-west-1
IoTSecurityPolicy_ TLS12_1_0_2015_01	ap-northeast-1, ap-southeast-1, eu-central-1, eu-west-1, us-east-1, us-west-2

의 보안 정책 이름에는 릴리스된 연도 및 월을 기반으로 한 버전 정보가 AWS IoT Core 포함됩니다. 새 도메인 구성을 생성하는 경우 보안 정책은 기본적으로 `IoTSecurityPolicy_TLS13_1_2_2022_10`으로 설정됩니다. 프로토콜, TCP 포트 및 암호에 대한 세부 정보가 포함된 보안 정책의 전체 표는 Security policys. AWS IoT Core doesn not support custom security policies를 참조하세요. 자세한 내용은 ??? 단원을 참조하십시오.

도메인 구성에서 TLS 설정을 구성하려면 AWS IoT 콘솔 또는를 사용할 수 있습니다 AWS CLI.

내용

- 도메인 구성에서 TLS 설정 구성 (콘솔)
- 도메인 구성에서 TLS 설정 구성 (CLI)

도메인 구성에서 TLS 설정 구성 (콘솔)

AWS IoT 콘솔을 사용하여 TLS 설정을 구성하려면

1. 에 로그인 AWS Management Console 하고 AWS IoT 콘솔을 엽니다.
2. 새 도메인 구성을 만들 때 TLS 설정을 구성하려면 다음 단계를 따르세요.

1. 왼쪽 탐색 창에서 도메인 구성을 선택한 다음 도메인 구성 생성을 선택합니다.
2. 도메인 구성 생성 페이지에서 사용자 지정 도메인 설정 - 선택 사항 섹션의 보안 정책 선택에서 보안 정책을 선택합니다.
3. 위젯을 따라 나머지 단계를 완료하세요. 도메인 구성 생성을 선택합니다.

3. 기존 도메인 구성에서 TLS 설정을 업데이트하려면 다음 단계를 따르세요.

1. 왼쪽 탐색 창에서 도메인 구성을 선택한 다음 도메인 구성을 선택합니다.
2. 도메인 구성 세부 정보 페이지에서 편집을 선택합니다. 그런 다음 사용자 지정 도메인 설정 - 선택 사항 섹션의 보안 정책 선택에서 보안 정책을 선택합니다.
3. 도메인 구성 업데이트를 선택합니다.

자세한 내용은 도메인 구성 생성 및 도메인 구성 관리를 참조하세요.

도메인 구성에서 TLS 설정 구성 (CLI)

create-domain-configuration 및 update-domain-configuration CLI 명령을 사용하여 도메인 구성에서 TLS 설정을 구성할 수 있습니다.

1. create-domain-configuration CLI 명령을 사용하여 TLS 설정을 지정하는 방법:

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

이 명령의 출력은 다음과 같을 수 있습니다.

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

보안 정책을 지정하지 않고 새 도메인 구성을 생성하는 경우 값은 기본값인 `IoTSecurityPolicy_TLS13_1_2_2022_10`으로 설정됩니다.

2. describe-domain-configuration CLI 명령을 사용하여 TLS 설정을 설명하는 방법:

```
aws iot describe-domain-configuration \
```

```
    --domain-configuration-name domainConfigurationName
```

이 명령은 다음과 같은 TLS 설정이 포함된 도메인 구성 세부 정보를 반환할 수 있습니다.

```
{
 "tlsConfig": {
 "securityPolicy": "IoTSecurityPolicy_TLS13_1_2_2022_10"
 },
 "domainConfigurationStatus": "ENABLED",
 "serviceType": "DATA",
 "domainType": "AWS_MANAGED",
 "domainName": "d1234567890abcdefghij-ats.iot.us-west-2.amazonaws.com",
 "serverCertificates": [],
 "lastStatusChangeDate": 1678750928.997,
 "domainConfigurationName": "test",
 "domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

3. update-domain-configuration CLI 명령을 사용하여 TLS 설정을 업데이트하는 방법:

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

이 명령의 출력은 다음과 같을 수 있습니다.

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

4. ATS 엔드포인트의 TLS 설정을 업데이트하려면 update-domain-configuration CLI 명령을 실행합니다. ATS 엔드포인트의 도메인 구성 이름은 `iot:Data-ATS`입니다.

```
aws iot update-domain-configuration \
    --domain-configuration-name "iot:Data-ATS" \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

명령의 출력은 다음과 같을 수 있습니다.

```
{
"domainConfigurationName": "iot:Data-ATS",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
iot:Data-ATS"
}
```

자세한 내용은 AWS API 참조의 CreateDomainConfiguration 및 UpdateDomainConfiguration을 참조하세요.

OCSP 스테이플링을 위한 서버 인증서 구성

AWS IoT Core 는 서버 인증서 OCSP 스테이플링 또는 OCSP 스테이플링이라고도 하는 서버 인증서에 대한 온라인 인증서 상태 프로토콜(OCSP) 스테이플링을 지원합니다. Transport Layer Security(TLS) 핸드셰이크에서 서버 인증서의 해지 상태를 확인하는 데 사용되는 보안 메커니즘입니다. 의 OCSP 스테이플링 AWS IoT Core 을 사용하면 사용자 지정 도메인의 서버 인증서 유효성에 추가 확인 계층을 추가할 수 있습니다.

에서 서버 인증서 OCSP 스테이플링을 활성화 AWS IoT Core 하여 OCSP 응답기를 주기적으로 쿼리하여 인증서의 유효성을 확인할 수 있습니다. OCSP 스테이플링 설정은 사용자 지정 도메인으로 도메인 구성을 생성하거나 업데이트하는 프로세스의 일부입니다. OCSP 스테이플링은 서버 인증서의 취소 상태를 지속적으로 확인합니다. 이렇게 하면 CA에서 취소한 모든 인증서가 사용자 지정 도메인에 연결하는 클라이언트에서 더 이상 신뢰할 수 없는지 확인할 수 있습니다. 자세한 내용은 ??? 단원을 참조하십시오.

서버 인증서 OCSP 스테이플링은 실시간 취소 상태 확인을 제공하고, 취소 상태 확인과 관련된 지연 시간을 줄이고, 보안 연결의 프라이버시와 신뢰성을 개선합니다. OCSP 스테이플링 사용의 이점에 대한 자세한 내용은 ??? 섹션을 참조하세요.

> **Note**
>
> 이 기능은에서 사용할 수 없습니다 AWS GovCloud (US) Regions.

이 주제에서 수행할 작업

- OCSP란 무엇인가요?
- OCSP 스테이플링 작동 방식
- 에서 서버 인증서 OCSP 활성화 AWS IoT Core

- 에서 프라이빗 엔드포인트에 대한 서버 인증서 OCSP 구성 AWS IoT Core
- 에서 서버 인증서 OCSP 스테이플링을 사용하기 위한 중요 참고 사항 AWS IoT Core
- 에서 서버 인증서 OCSP 스테이플링 문제 해결 AWS IoT Core

OCSP란 무엇인가요?

온라인 인증서 상태 프로토콜(OCSP)은 Transport Layer Security(TLS) 핸드셰이크에 대한 서버 인증서의 취소 상태를 제공하는 데 도움이 됩니다.

주요 개념

다음 주요 개념은 온라인 인증서 상태 프로토콜(OCSP)에 대한 세부 정보를 제공합니다.

OCSP

OCSP는 Transport Layer Security(TLS) 핸드셰이크 중에 인증서 취소 상태를 확인하는 데 사용됩니다. OCSP는 인증서의 실시간 검증을 허용합니다. 이렇게 하면 인증서가 발급된 이후 취소되거나 만료되지 않았음을 확인할 수 있습니다. 또한 OCSP는 기존 인증서 해지 목록(CRL)에 비해 확장성이 뛰어납니다. OCSP 응답은 크기가 작아 효율적으로 생성할 수 있으므로 대규모 프라이빗 키 인프라(PKI)에 더 적합합니다.

OCSP 응답기

OCSP 응답기(OCSP 서버라고도 함)는 인증서의 해지 상태를 확인하려는 클라이언트로부터 OCSP 요청을 수신하고 응답합니다.

클라이언트 측 OCSP

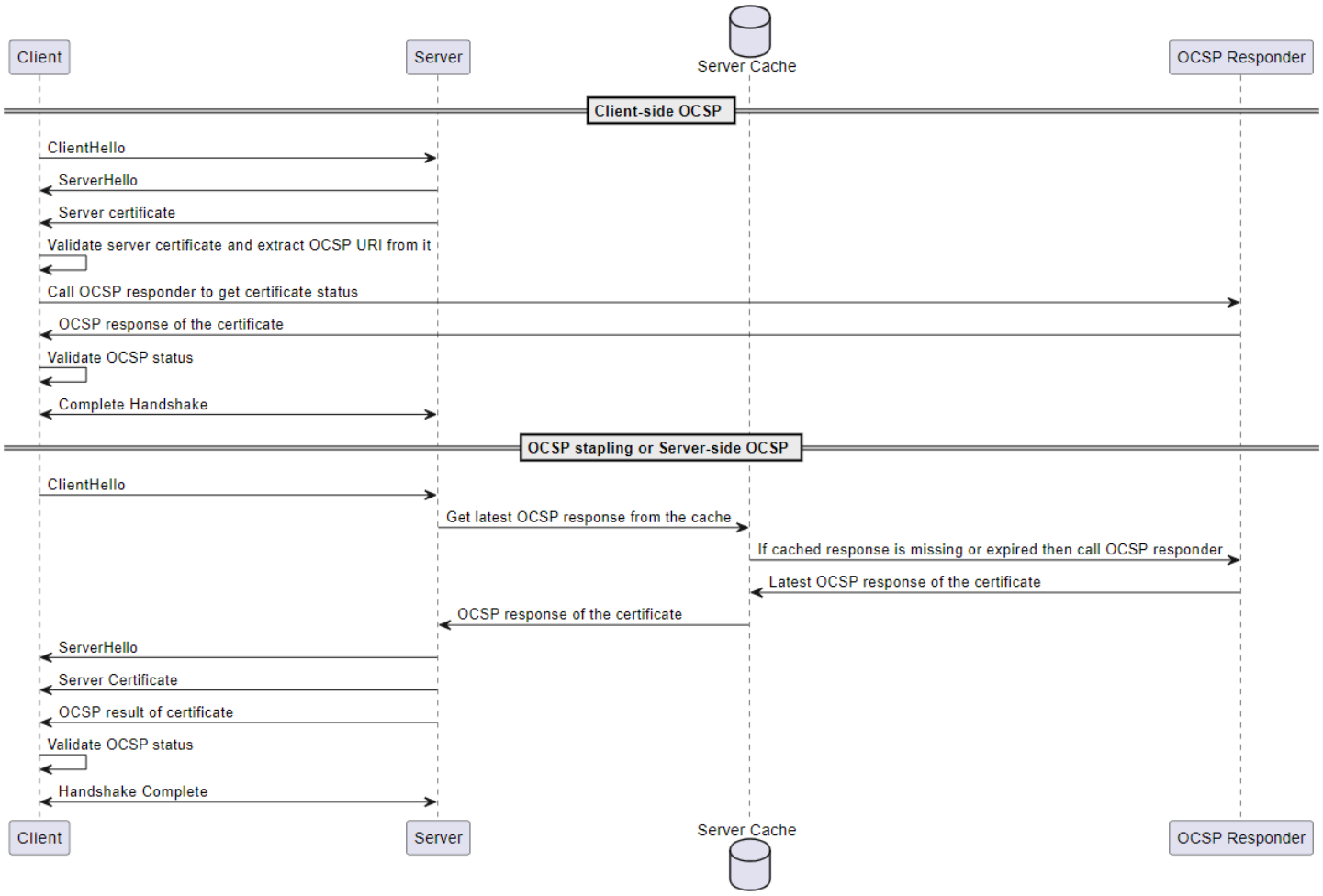
클라이언트 측 OCSP에서 클라이언트는 OCSP로 OCSP 응답자에게 연락하여 TLS 핸드셰이크 중에 인증서의 해지 상태를 확인합니다.

서버 측 OCSP

서버 측 OCSP(OCSP 스테이플링이라고도 함)에서 서버는 (클라이언트가 아닌) OCSP 응답기에 대한 요청을 할 수 있도록 활성화됩니다. 서버는 인증서에 대한 OCSP 응답을 스테이플링하고 TLS 핸드셰이크 중에 클라이언트에 반환합니다.

OCSP 다이어그램

다음 다이어그램은 클라이언트 측 OCSP 및 서버 측 OCSP의 작동 방식을 보여줍니다.



클라이언트 측 OCSP

1. 클라이언트는 서버와 함께 TLS 핸드셰이크를 시작하는 `ClientHello` 메시지를 보냅니다.
2. 서버는 메시지를 수신하고 `ServerHello` 메시지로 응답합니다. 서버는 또한 서버 인증서를 클라이언트로 전송합니다.
3. 클라이언트는 서버 인증서를 검증하고 OCSP URI를 추출합니다.
4. 클라이언트는 OCSP 응답자에게 인증서 취소 확인 요청을 보냅니다.
5. OCSP 응답기가 OCSP 응답을 전송합니다.
6. 클라이언트는 OCSP 응답에서 인증서 상태를 검증합니다.
7. TLS 핸드셰이크가 완료되었습니다.

서버 측 OCSP

1. 클라이언트는 서버와 함께 TLS 핸드셰이크를 시작하는 `ClientHello` 메시지를 보냅니다.

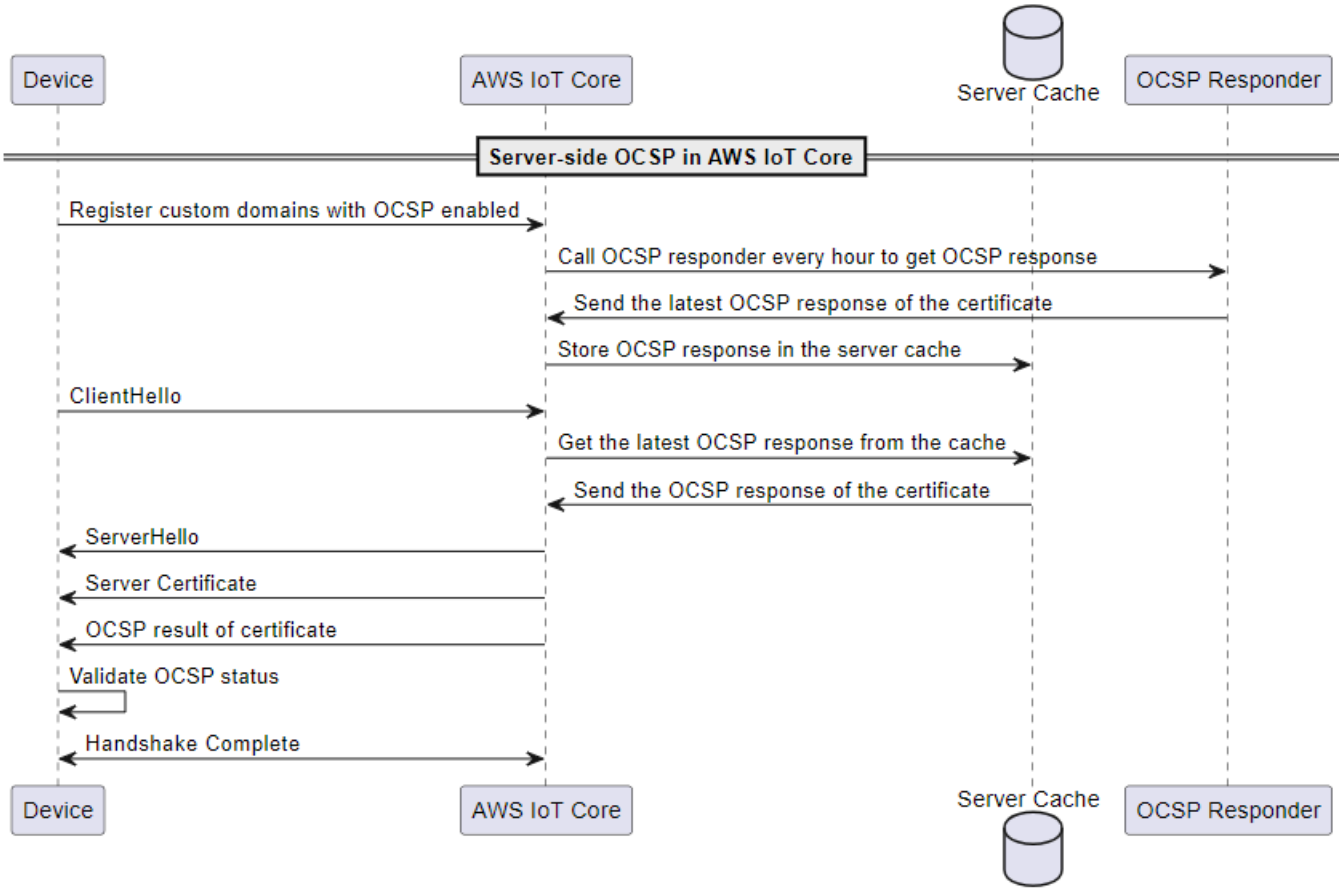
2. 서버는 메시지를 수신하고 캐시된 최신 OCSP 응답을 가져옵니다. 캐시된 응답이 누락되거나 만료된 경우 서버는 인증서 상태를 위해 OCSP 응답기를 직접적으로 호출합니다.
3. OCSP 응답기는 OCSP 응답을 서버에 전송합니다.
4. 서버가 `ServerHello` 메시지를 보냅니다. 또한 서버는 서버 인증서와 인증서 상태를 클라이언트로 전송합니다.
5. 클라이언트가 OCSP 인증서 상태를 검증합니다.
6. TLS 핸드셰이크가 완료되었습니다.

OCSP 스테이플링 작동 방식

OCSP 스테이플링은 클라이언트와 서버 간의 TLS 핸드셰이크 중에 서버 인증서 해지 상태를 확인하는 데 사용됩니다. 서버는 OCSP 응답기에 OCSP 요청을 하고 클라이언트에 반환된 인증서에 대한 OCSP 응답을 스테이플링합니다. 서버가 OCSP 응답기에 요청을 하도록 하면 응답을 캐시한 다음 여러 클라이언트에 여러 번 사용할 수 있습니다.

에서 OCSP 스테이플링 작동 방식 AWS IoT Core

다음 다이어그램은 AWS IoT Core에서 서버 측 OCSP 스테이플링이 작동하는 방식을 보여줍니다.



1. OCSP 스테이플링이 활성화된 사용자 지정 도메인에 디바이스를 등록해야 합니다.
2. AWS IoT Core 는 1시간마다 OCSP 응답기를 호출하여 인증서 상태를 가져옵니다.
3. OCSP 응답기는 요청을 수신하고, 최신 OCSP 응답을 전송하고, 캐시된 OCSP 응답을 저장합니다.
4. 디바이스는 TLS 핸드셰이크를 시작하는 `ClientHello` 메시지를 보냅니다 AWS IoT Core.
5. AWS IoT Core 는 인증서의 OCSP 응답으로 응답하는 서버 캐시에서 최신 OCSP 응답을 가져옵니다.
6. 서버는 디바이스에 `ServerHello` 메시지를 보냅니다. 또한 서버는 서버 인증서와 인증서 상태를 클라이언트로 전송합니다.
7. 디바이스가 OCSP 인증서 상태를 검증합니다.
8. TLS 핸드셰이크가 완료되었습니다.

클라이언트 측 OCSP 확인과 비교하여 OCSP 스테이플링 사용의 이점

서버 인증서 OCSP 스테이플링 사용의 이점에는 다음이 포함됩니다.

개인 정보 보호 개선

OCSP 스테이플링이 없으면 클라이언트의 디바이스가 서드파티 OCSP 응답자에게 정보를 노출하여 사용자 프라이버시를 손상시킬 수 있습니다. OCSP 스테이플링은 서버가 OCSP 응답을 받아 클라이언트에 직접 전달하도록 하여 이 문제를 완화합니다.

신뢰성 향상

OCSP 스테이플링은 OCSP 서버 중단 위험을 줄이기 때문에 보안 연결의 신뢰성을 개선할 수 있습니다. OCSP 응답이 스테이플링되면 서버에 인증서에 가장 최근의 응답이 포함됩니다. 이는 OCSP 응답기를 일시적으로 사용할 수 없는 경우에도 클라이언트가 취소 상태에 액세스할 수 있도록 하기 위한 것입니다. OCSP 스테이플링은 서버가 OCSP 응답을 주기적으로 가져오고 TLS 핸드셰이크에 캐시된 응답을 포함하므로 이러한 문제를 완화하는 데 도움이 됩니다. 이렇게 하면 OCSP 응답자의 실시간 가용성에 대한 의존도가 줄어듭니다.

서버 부하 감소

OCSP 스테이플링은 OCSP 응답자의 OCSP 요청에 응답하는 부담을 서버로 덜어 줍니다. 이렇게 하면 부하를 더 균등하게 분산하여 인증서 검증 프로세스를 더 효율적이고 확장할 수 있습니다.

지연 시간 단축

OCSP 스테이플링은 TLS 핸드셰이크 중에 인증서의 취소 상태를 확인하는 것과 관련된 지연 시간을 줄입니다. 클라이언트가 OCSP 서버를 별도로 쿼리해야 하는 대신 서버는 요청을 보내고 핸드셰이크 중에 서버 인증서와 OCSP 응답을 연결합니다.

에서 서버 인증서 OCSP 활성화 AWS IoT Core

서버 인증서 OCSP 스테이플링을 활성화하려면 사용자 지정 도메인에 대한 도메인 구성을 AWS IoT Core생성하거나 기존 사용자 지정 도메인 구성을 업데이트합니다. 사용자 지정 도메인을 사용하여 도메인 구성을 생성하는 방법에 대한 일반적인 내용은 ??? 섹션을 참조하세요.

다음 지침에 따라 AWS Management Console 또는를 사용하여 OCSP 서버 스테이플링을 활성화합니다 AWS CLI.

콘솔

AWS IoT 콘솔을 사용하여 서버 인증서 OCSP 스테이플링을 활성화하려면:

1. 탐색 메뉴에서 설정을 선택한 다음 도메인 구성 생성 또는 사용자 지정 도메인의 기존 도메인 구성을 선택합니다.
2. 이전 단계에서 새 도메인 구성을 생성하도록 선택하면 도메인 구성 생성 페이지가 표시됩니다. 도메인 구성 속성 섹션에서 사용자 지정 도메인을 선택합니다. 정보를 입력하여 도메인 구성을 생성합니다.

   사용자 지정 도메인의 기존 도메인 구성을 업데이트하도록 선택하면 도메인 구성 세부 정보 페이지가 표시됩니다. 편집을 선택합니다.
3. OCSP 서버 스테이플링을 활성화하려면 서버 인증서 구성 하위 섹션에서 서버 인증서 OCSP 스테이플링 활성화를 선택합니다.
4. 도메인 구성 생성 또는 도메인 구성 업데이트를 선택합니다.

AWS CLI

AWS CLI를 사용하여 서버 인증서 OCSP 스테이플링을 활성화하려면:

1. 사용자 지정 도메인에 대한 새 도메인 구성을 생성하는 경우 OCSP 서버 스테이플링을 활성화하는 명령은 다음과 같을 수 있습니다.

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
```

```
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

2. 사용자 지정 도메인의 기존 도메인 구성을 업데이트하는 경우 OCSP 서버 스테이플링을 활성화하는 명령은 다음과 같을 수 있습니다.

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

자세한 내용은 AWS IoT API 참조의 CreateDomainConfiguration 및 UpdateDomainConfiguration을 참조하세요.

에서 프라이빗 엔드포인트에 대한 서버 인증서 OCSP 구성 AWS IoT Core

프라이빗 엔드포인트용 OCSP를 사용하면 Amazon Virtual Private Cloud(Amazon VPC) 내의 프라이빗 OCSP 리소스를 AWS IoT Core 작업에 사용할 수 있습니다. 이 프로세스에는 OCSP 응답자 역할을 하는 Lambda 함수를 설정하는 작업이 포함됩니다. Lambda 함수는 프라이빗 OCSP 리소스를 사용하여에서 사용할 OCSP 응답을 생성할 AWS IoT Core 수 있습니다.

Lambda 함수

프라이빗 엔드포인트에 대해 서버 OCSP를 구성하기 전에 의견 요청(RFC) 6960 준수 온라인 인증서 상태 프로토콜(OCSP) 응답자 역할을 하며 기본 OCSP 응답을 지원하는 Lambda 함수를 생성합니다. Lambda 함수는 고유 인코딩 규칙(DER) 형식으로 OCSP 요청의 base64 인코딩을 수락합니다. Lambda 함수의 응답은 DER 형식의 base64로 인코딩된 OCSP 응답이기도 합니다. 응답 크기는 4KB(킬로바이트)를 초과해서는 안 됩니다. Lambda 함수는 도메인 구성과 동일한 AWS 계정 및 AWS 리전 에 있어야 합니다. 다음은 Lambda 함수 예제입니다.

예 Lambda 함수

JavaScript

```
import * as pkijs from 'pkijs';
console.log('Loading function');
```

```
export const handler = async (event, context) => {
    const requestBytes = decodeBase64(event);
    const ocspRequest = pkijs.OCSPRequest.fromBER(requestBytes);

    console.log("Here is a better look at the OCSP request");
    console.log(ocspRequest.toJSON());

    const ocspResponse = getOcspResponse();

    console.log("Here is a better look at the OCSP response");
    console.log(ocspResponse.toJSON());

   const responseBytes = ocspResponse.toSchema().toBER();
   return encodeBase64(responseBytes);
};

function getOcspResponse() {
    const responseString = "MIIC/
woBAKCCAvgwggL0BgkrBgEFBQcwAQEEggLlMIIC4TCByqFkMGIxJzAlBgNVBAoMHlJpY2hhcmQncyBEaXNjb3VudCBMY
p5w7WOtPjp3otNtVgIBAYAAGA8yMDI0MDQyMzE4NTMyNVowDQYJKoZIhvcNAQELBQADggIBAJFRyjDAHfazNejo704Ra
+s82R1spDarr3k7Pzkod9jJhwsZ2YgushlS4Npfe4lHCdwFyZR75WXrW55aXFddy03KLz01ZLNYyxkleW3f5dgrUcRU3
DEBiyS7ZsyhKo6igWU/SY7YMSKgwBvFsqSDcOa/hRYQkxWKWJ19gcz8CIkWN7NvfIxCs6VrAdzEJwmE7y3v
+jdfhxW9JmI4xStE4K0tAR9vVOOfKs7NvxXj7oc9pCSG60xl96kaEE6PaY1YsfNTsKQ7pyCJ0s7/2q
+ieZ4AtNyzw1XBadPzPJNv6E0LvI24yQZqN5wACvtut5prMMRxAHbOy
+abLZR58wloFSELtGJ7UD96LFv1GgtC5s
+2QlzPc4bEEof7Lo1EISt3j2ibNch8LxhqTQ4ufrbhsMkpSOTFYEJVMJF6aKj/OGXBUUqgc0Jx6jjJXNQd
+l5KCY9pQFeb/wVUYC6mYqZOkNNMMJxPbHHbFnqb68yO+g5BE9011N44YXoPVJYoXxBLFX+OpRu9cqPkT9/
vlkKd+SYXQknwZ81agKzhf1HsBKabtJwNVMlBKaI8g5UGa7Bxi6ewH3ezdWiERRUK7F56OM53wto/";
    const responseBytes = decodeBase64(responseString);
    return pkijs.OCSPResponse.fromBER(responseBytes);
}

function decodeBase64(input) {
    const binaryString = atob(input);

    const byteArray = new Uint8Array(binaryString.length);
    for (var i = 0; i < binaryString.length; i++) {
        byteArray[i] = binaryString.charCodeAt(i);
    }

    return byteArray.buffer;
}
```

```
function encodeBase64(buffer) {
    var binary = '';
    const bytes = new Uint8Array( buffer );
    const len = bytes.byteLength;

    for (var i = 0; i < len; i++) {
        binary += String.fromCharCode( bytes[ i ] );
    }

    return btoa(binary);
}
```

Java

```
package com.example.ocsp.responder;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import org.bouncycastle.cert.ocsp.OCSPReq;
import org.bouncycastle.cert.ocsp.OCSPResp;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Base64;

public class LambdaResponderApplication implements RequestHandler<String, String> {
    @Override
    public String handleRequest(final String input, final Context context) {
        LambdaLogger logger = context.getLogger();

        byte[] decodedInput = Base64.getDecoder().decode(input);

        OCSPReq req;
        try {
            req = new OCSPReq(decodedInput);
        } catch (IOException e) {
            logger.log("Got an IOException creating the OCSP request: " +
 e.getMessage());
            throw new RuntimeException(e);
        }

        try {
            OCSPResp response = businessLogic.getMyResponse();
```

```
            String toReturn =
 Base64.getEncoder().encodeToString(response.getEncoded());
            return toReturn;
        } catch (Exception e) {
            logger.log("Got an exception creating the response: " + e.getMessage());
            return "";
        }
    }
}
```

Lambda 함수 AWS IoT 호출 권한 부여

Lambda OCSP 응답기로 도메인 구성을 생성하는 과정에서 함수가 생성된 후 Lambda 함수를 호출할 수 있는 AWS IoT 권한을 부여해야 합니다. 권한을 부여하려면 add-permission CLI 명령을 사용할 수 있습니다.

를 사용하여 Lambda 함수에 권한 부여 AWS CLI

1. 값을 삽입한 후 다음 명령을 입력합니다. 참고: `statement-id` 값은 고유해야 합니다. *Id-1234*을(를) 정확한 값으로 바꿉니다. 그렇지 않으면 `ResourceConflictException` 오류가 발생할 수 있습니다.

   ```
   aws lambda add-permission  \
   --function-name "ocsp-function" \
   --principal "iot.amazonaws.com" \
   --action "lambda:InvokeFunction" \
   --statement-id "Id-1234" \
   --source-arn arn:aws:iot:us-east-1:123456789012:domainconfiguration/<domain-config-name>/*
   --source-account 123456789012
   ```

   IoT 도메인 구성 ARN은 다음 패턴을 따릅니다. 생성 시간 전에는 서비스 생성 접미사를 알 수 없으므로 접미사를 *로 바꿔야 합니다. 도메인 구성이 생성되고 정확한 ARN을 알면 권한을 업데이트할 수 있습니다.

   arn:*aws*:iot:*use-east-1:123456789012*:domainconfiguration/*domain-config-name/service-generated-suffix*

2. 명령이 성공하면 이 예제와 같은 권한 문이 반환됩니다. 다음 섹션으로 계속 진행하여 프라이빗 엔드포인트에 대한 OCSP 스테이플링을 구성할 수 있습니다.

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-east-1:123456789012:function:ocsp-
function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:iot:us-
east-1:123456789012:domainconfiguration/domain-config-name/*\"}}}"
}
```

명령이 성공하지 못하면 이 예제와 같은 오류가 반환됩니다. 계속하기 전에 오류를 검토하고 수정해야 합니다.

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:us-east-1:123456789012:function:ocsp-function
```

프라이빗 엔드포인트에 대한 서버 OCSP 스테이플링 구성

콘솔

AWS IoT 콘솔을 사용하여 서버 인증서 OCSP 스테이플링을 구성하려면:

1. 탐색 메뉴에서 설정을 선택한 다음 도메인 구성 생성 또는 사용자 지정 도메인의 기존 도메인 구성을 선택합니다.
2. 이전 단계에서 새 도메인 구성을 생성하도록 선택하면 도메인 구성 생성 페이지가 표시됩니다. 도메인 구성 속성 섹션에서 사용자 지정 도메인을 선택합니다. 정보를 입력하여 도메인 구성을 생성합니다.

   사용자 지정 도메인의 기존 도메인 구성을 업데이트하도록 선택하면 도메인 구성 세부 정보 페이지가 표시됩니다. 편집을 선택합니다.
3. OCSP 서버 스테이플링을 활성화하려면 서버 인증서 구성 하위 섹션에서 서버 인증서 OCSP 스테이플링 활성화를 선택합니다.
4. 도메인 구성 생성 또는 도메인 구성 업데이트를 선택합니다.

AWS CLI

AWS CLI을 사용하여 서버 인증서 OCSP 스테이플링을 구성하려면 다음을 수행하세요.

1. 사용자 지정 도메인에 대한 새 도메인 구성을 생성하는 경우 프라이빗 엔드포인트에 대한 서버 인증서 OCSP를 구성하는 명령은 다음과 같을 수 있습니다.

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true,
 ocspAuthorizedResponderArn=arn:aws:acm:us-
east-1:123456789012:certificate/certificate_ID, ocspLambdaArn=arn:aws:lambda:us-
east-1:123456789012:function:my-function"
```

2. 사용자 지정 도메인에 대한 기존 도메인 구성을 업데이트하는 경우 프라이빗 엔드포인트에 대한 서버 인증서 OCSP를 구성하는 명령은 다음과 같을 수 있습니다.

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true,
 ocspAuthorizedResponderArn=arn:aws:acm:us-
east-1:123456789012:certificate/certificate_ID, ocspLambdaArn=arn:aws:lambda:us-
east-1:123456789012:function:my-function"
```

enableOCSPCheck

서버 OCSP 스테이플링 검사가 활성화되었는지 여부를 나타내는 부울 값입니다. 서버 인증서 OCSP 스테이플링을 활성화하려면 이 값이 true여야 합니다.

ocspAuthorizedResponderArn

(ACM)에 저장된 X.509 인증서에 대한 Amazon 리소스 이름 AWS Certificate Manager (ARN)의 문자열 값입니다. 제공된 경우 AWS IoT Core 는이 인증서를 사용하여 수신된 OCSP 응답의 서명을 검증합니다. 제공되지 않으면 발급 인증서를 AWS IoT Core 사용하여 응답을 검증합니다. 인증서는

도메인 구성과 동일한 AWS 계정 및 AWS 리전 에 있어야 합니다. 승인된 응답자 인증서를 등록하는 방법에 대한 자세한 내용은 AWS Certificate Manager로 인증서 가져오기를 참조하세요.

ocspLambdaArn

기본 OCSP 응답을 지원하며, 의견 요청(RFC) 6960 준수(OCSP) 응답자 역할을 하는 Lambda 함수에 대한 Amazon 리소스 이름(ARN)의 문자열 값입니다. Lambda 함수는 DER 형식을 사용하여 인코딩되는 OCSP 요청의 base64 인코딩을 수락합니다. Lambda 함수의 응답은 DER 형식의 base64로 인코딩된 OCSP 응답이기도 합니다. 응답 크기는 4KB(킬로바이트)를 초과해서는 안 됩니다. Lambda 함수는 도메인 구성과 동일한 AWS 계정 및 AWS 리전 에 있어야 합니다.

자세한 내용은 AWS IoT API 참조의 CreateDomainConfiguration 및 UpdateDomainConfiguration을 참조하세요.

에서 서버 인증서 OCSP 스테이플링을 사용하기 위한 중요 참고 사항 AWS IoT Core

에서 서버 인증서 OCSP를 사용하는 경우 다음 사항에 유의 AWS IoT Core하세요.

1. AWS IoT Core 는 퍼블릭 IPv4 주소를 통해 연결할 수 있는 OCSP 대응 담당자만 지원합니다.
2. 의 OCSP 스테이플링 기능은 승인된 응답기를 지원하지 AWS IoT Core 않습니다. 모든 OCSP 응답은 인증서에 서명한 CA가 서명해야 하며 CA는 사용자 지정 도메인의 인증서 체인에 속해야 합니다.
3. 의 OCSP 스테이플링 기능은 자체 서명된 인증서를 사용하여 생성된 사용자 지정 도메인을 지원하지 AWS IoT Core 않습니다.
4. AWS IoT Core 는 매시간 OCSP 응답기를 호출하고 응답을 캐싱합니다. 응답기에 대한 호출이 실패하면 AWS IoT Core 는 가장 최근의 유효한 응답을 스테이플링합니다.
5. `nextUpdateTime`가 더 이상 유효하지 않으면 AWS IoT Core 는 캐시에서 응답을 제거하고 TLS 핸드셰이크는 OCSP 응답기에 대한 다음 번 성공적인 호출까지 OCSP 응답 데이터를 포함하지 않습니다. 이는 서버가 OCSP 응답기로부터 유효한 응답을 받기 전에 캐시된 응답이 만료되었을 때 발생할 수 있습니다. `nextUpdateTime`의 값은 OCSP 응답이 이 시점까지 유효할 것임을 제안합니다. `nextUpdateTime`에 대한 자세한 정보는 ??? 섹션을 참조하세요.
6. 경우에 따라가 OCSP 응답을 수신 AWS IoT Core 하지 못하거나 기존 OCSP 응답이 만료되어 제거합니다. 이러한 상황이 발생하면 AWS IoT Core 는 OCSP 응답 없이 사용자 지정 도메인에서 제공하는 서버 인증서를 계속 사용합니다.
7. OCSP 응답의 크기는 4KiB를 초과할 수 없습니다.

에서 서버 인증서 OCSP 스테이플링 문제 해결 AWS IoT Core

AWS IoT Core 는 `RetrieveOCSPStapleData.Success` 지표와 `RetrieveOCSPStapleData` 로그 항목을 CloudWatch로 내보냅니다. 지표와 로그 항목은 OCSP 응답 검색과 관련된 문제를 감지하는 데 도움이 될 수 있습니다. 자세한 내용은 ??? 및 ??? 섹션을 참조하세요.

AWS IoT FIPS 엔드포인트에 연결

AWS IoT 는 FIPS(Federal Information Processing Standard) 140-2를 지원하는 엔드포인트를 제공합니다. FIPS 준수 엔드포인트는 표준 AWS 엔드포인트와 다릅니다. FIPS 준수 방식으로 AWS IoT 와 상호 작용하려면 FIPS 준수 클라이언트와 함께 아래에 설명된 엔드포인트를 사용해야 합니다. AWS IoT 콘솔이 FIPS를 준수하지 않습니다.

다음 섹션에서는 REST API, SDK 또는를 사용하여 FIPS 준수 AWS IoT 엔드포인트에 액세스하는 방법을 설명합니다 AWS CLI.

주제

- AWS IoT Core - 제어 영역 엔드포인트
- AWS IoT Core - 데이터 영역 엔드포인트
- AWS IoT Core- 자격 증명 공급자 엔드포인트
- AWS IoT Device Management - 작업 데이터 엔드포인트
- AWS IoT Device Management- Fleet Hub 엔드포인트
- AWS IoT Device Management- 보안 터널링 엔드포인트
- AWS IoT Device Management - 관리형 통합 엔드포인트

AWS IoT Core - 제어 영역 엔드포인트

AWS IoT 작업 및 관련 CLI 명령을 지원하는 FIPS 준수 AWS IoT Core - 제어 영역 엔드포인트는 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Core - 제어 영역 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

AWS IoT 작업에 액세스할 때 FIPS 준수 엔드포인트를 사용하려면 AWS SDK 또는 REST API를에 적합한 엔드포인트와 함께 사용합니다 AWS 리전.

aws iot CLI 명령을 실행할 때 FIPS 준수 엔드포인트를 사용하려면 AWS 리전 에 대한 적절한 엔드포인트가 있는 --endpoint 파라미터를 명령에 추가합니다.

AWS IoT Core - 데이터 영역 엔드포인트

FIPS 준수 AWS IoT Core - 데이터 영역 엔드포인트는 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Core - 데이터 영역 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

AWS IoT 디바이스 SDK를 사용하고 계정의 기본AWS IoT Core - 데이터 영역 엔드포인트 대신 SDK의 연결 함수에 엔드포인트를 제공하여 FIPS 준수 클라이언트 AWS 리전 에서에 대한 FIPS 준수 엔드포인트를 사용할 수 있습니다. 연결 함수는 AWS IoT 디바이스 SDK에만 해당됩니다. 연결 함수의 예는 AWS IoT Device SDK for Python의 연결 함수를 참조하세요.

> **Note**
>
> AWS IoT 는 FIPS를 준수하는 AWS 계정특정AWS IoT Core - 데이터 영역 엔드포인트를 지원하지 않습니다. 서버 이름 표시(SNI)에 AWS 계정특정 엔드포인트가 필요한 서비스 기능은 사용할 수 없습니다. FIPS 준수 AWS IoT Core- 데이터 영역 엔드포인트는 다중 계정 등록 인증서, 사용자 지정 도메인, 사용자 지정 권한 부여자 및 구성 가능한 엔드포인트(지원되는 TLS 정책 포함)를 지원할 수 없습니다.

AWS IoT Core- 자격 증명 공급자 엔드포인트

FIPS 준수 AWS IoT Core- 자격 증명 공급자 엔드포인트는 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Core - 자격 증명 공급자 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

> **Note**
>
> AWS IoT 는 AWS 계정 FIPS를 준수하는 특정 자격 AWS IoT Core 증명 공급자 엔드포인트를 지원하지 않습니다. 서버 이름 표시(SNI)에 AWS 계정특정 엔드포인트가 필요한 서비스 기능은 사용할 수 없습니다. FIPS 준수 AWS IoT Core- 자격 증명 공급자 엔드포인트는 다중 계정 등록 인증서, 사용자 지정 도메인, 사용자 지정 권한 부여자 및 구성 가능한 엔드포인트(지원되는 TLS 정책 포함)를 지원할 수 없습니다.

AWS IoT Device Management - 작업 데이터 엔드포인트

FIPS 준수 AWS IoT Device Management - 작업 데이터 엔드포인트는 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Device Management - 작업 데이터 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

aws iot-jobs-data CLI 명령을 실행할 때 FIPS 준수 AWS IoT Device Management- 작업 데이터 엔드포인트를 사용하려면 AWS 리전 에 대한 적절한 엔드포인트가 있는 --endpoint 파라미터를 명령에 추가합니다. 이 엔드포인트와 함께 REST API를 사용할 수도 있습니다.

`Data-ATS` 대신 `iot:Jobs`를 사용하는 것이 좋습니다. `iot:Data-ATS`는 듀얼 스택 엔드포인트(IPv4 및 IPv6)를 지원하는 반면 `iot:Jobs`는 IPv4만 지원합니다.

AWS IoT 디바이스 SDK를 사용하고 계정의 기본AWS IoT Device Management - 작업 데이터 엔드포인트 대신 SDK의 연결 함수에 엔드포인트를 제공하여 FIPS 준수 클라이언트와 AWS 리전 함께에 대한 FIPS 준수 엔드포인트를 사용할 수 있습니다. 연결 함수는 AWS IoT 디바이스 SDK에 한정됩니다. 연결 함수의 예는 AWS IoT Device SDK for Python의 연결 함수를 참조하세요.

AWS IoT Device Management- Fleet Hub 엔드포인트

AWS IoT Device Management Fleet Hub for Device Management CLI 명령과 함께 사용할 FIPS 준수 - Fleet Hub AWS IoT 엔드포인트는 서비스별 FIPS 엔드포인트에 나열되어 있습니다. https://docs.aws.amazon.com//cli/latest/reference/iotfleethub/index.html 서비스별 FIPS 엔드포인트에서 AWS IoT Device Management - Fleet Hub 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

aws iotfleethub CLI 명령을 실행할 때 FIPS 준수AWS IoT Device Management - Fleet Hub 엔드포인트를 사용하려면에 적합한 엔드포인트가 있는 --endpoint 파라미터를 명령 AWS 리전 에 추가합니다. 이 엔드포인트와 함께 REST API를 사용할 수도 있습니다.

AWS IoT Device Management- 보안 터널링 엔드포인트

AWS IoT 보안 터널링 API용 FIPS 준수 AWS IoT Device Management - 보안 터널링 엔드포인트 및 해당 CLI 명령은 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Device Management - 보안 터널링 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

aws iotsecuretunneling CLI 명령을 실행할 때 FIPS 준수 AWS IoT Device Management- 보안 터널링 엔드포인트를 사용하려면 AWS 리전 에 대한 적절한 엔드포인트가 있는 --endpoint 파라미터를 명령에 추가합니다. 이 엔드포인트와 함께 REST API를 사용할 수도 있습니다.

AWS IoT Device Management - 관리형 통합 엔드포인트

관리형 통합 작업 및 관련 AWS CLI 명령을 지원하는 FIPS 준수 컨트롤 플레인 엔드포인트는 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Device Management - 관리형 통합 서비스를 찾아 AWS 리전에 대한 엔드포인트를 조회합니다.

관리형 통합 작업에 액세스할 때 FIPS 준수 엔드포인트를 사용하려면 AWS SDK 또는 REST API를에 적합한 엔드포인트와 함께 사용합니다 AWS 리전.

관리형 통합 CLI 명령을 실행할 때 FIPS 준수 엔드포인트를 사용하려면 AWS 리전 에 대한 적절한 엔드포인트가 있는 --endpoint 파라미터를 명령에 추가합니다.

를 사용하여 디바이스 관리 AWS IoT

AWS IoT 는 사물을 관리하는 데 도움이 되는 레지스트리를 제공합니다. 사물이란 특정 장치 또는 논리적 엔터티를 의미합니다. 사물은 물리적 장치 또는 센서일 수 있습니다(예: 전구 또는 벽면 스위치). 또한 애플리케이션의 인스턴스와 같은 논리적 개체이거나에 연결되지 AWS IoT 는 않지만에 연결된 다른 디바이스(예: 엔진 센서 또는 제어판이 있는 자동차)와 관련된 물리적 개체일 수도 있습니다.

사물에 대한 정보는 레지스트리에 JSON 데이터로 저장됩니다. 다음은 사물의 예입니다.

```
{
     "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

사물은 이름으로 식별됩니다. 사물은 일련번호, 제조업체 등 사물에 대한 정보를 저장하는 데 사용할 수 있는 이름-값 페어인 속성을 가질 수도 있습니다.

일반적인 디바이스 사용 사례에서는 사물 이름을 기본 MQTT 클라이언트 ID로 사용합니다. 사물의 레지스트리 이름과 사물의 MQTT 클라이언트 ID, 인증서 또는 섀도우 상태 사용 사이의 매핑을 강제 적용하고 있지는 않지만 사물 이름을 하나 선택하여 레지스트리 및 디바이스 섀도우 서비스 모두에서 MQTT 클라이언트 ID로 사용할 것을 권장합니다. 그러면 기본 디바이스 인증서 모델 또는 섀도우의 유연성을 해치지 않으면서 IoT 집합에 체계성 및 편의성을 제공할 수 있습니다.

디바이스를 AWS IoT에 연결하기 위해 레지스트리에서 사물을 생성할 필요는 없습니다. 사물을 레지스트리에 추가할 경우 디바이스를 보다 쉽게 관리하고 검색할 수 있습니다.

규칙 엔진에서 사물 레지스트리 데이터(속성, 사물 유형 및 그룹 멤버십 포함)를 동적으로 검색하여 메시지 처리 및 라우팅에 사용할 수도 있습니다. 자세한 내용은 get_registry_data 단원을 참조하십시오.

레지스트리로 사물 관리

AWS IoT 콘솔, AWS IoT API 또는를 사용하여 레지스트리와 AWS CLI 상호 작용합니다. 다음 섹션에서는 CLI를 사용하여 레지스트리 작업을 수행하는 방법을 설명합니다.

사물의 이름을 지정할 때:

- 사물 이름에 개인 식별 정보를 사용하면 안 됩니다. 사물 이름은 암호화되지 않은 통신 및 보고서에 나타날 수 있습니다.

주제

- 사물 생성
- 사물 목록 표시
- 사물 설명
- 사물 업데이트
- 사물 삭제
- 사물에 위탁자 연결
- 위탁자와 연결된 사물 나열
- 사물에 연결된 위탁자 나열
- 위탁자와 연결된 사물 나열 V2
- 사물에 연결된 위탁자 나열 V2
- 사물에서 보안 주체 분리

사물 생성

다음 명령은 CLI의 AWS IoT CreateThing 명령을 사용하여 사물을 생성하는 방법을 보여줍니다. 사물을 생성한 후에는 사물 이름을 변경할 수 없습니다. 사물 이름을 변경하려면 새 사물을 생성하고 새 이름을 지정한 다음 이전 사물을 삭제해야 합니다.

```
$ aws iot create-thing \
    --thing-type-name "MyLightBulb" \
    --attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

CreateThing 명령은 새 사물의 이름과 Amazon 리소스 이름(ARN)을 표시합니다.

```
{
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678"
}
```

> **Note**
>
> 사물 이름에 개인 식별 정보를 사용하지 않는 것이 좋습니다.

자세한 내용은 AWS CLI 명령 참조의 create-thing을 참조하세요.

사물 목록 표시

다음과 같이 ListThings 명령을 사용하여 계정에 속한 모든 사물을 나열할 수 있습니다.

```
$ aws iot list-things
```

```
{
    "things": [
       {
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyLightBulb"
        },
        {
            "attributes": {
                "numOfStates":"3"
             },
            "version": 11,
            "thingName": "MyWallSwitch"
        }
    ]
}
```

ListThings 명령을 사용하여 특정 사물 유형의 모든 사물을 검색할 수 있습니다.

```
$  aws iot list-things --thing-type-name "LightBulb"
```

```
{
    "things": [
        {
```

```
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

다음과 같이 ListThings 명령을 사용하여 특정 값의 속성을 갖는 모든 사물을 검색할 수 있습니다. 이 명령은 최대 세 개의 속성을 검색합니다.

```
$  aws iot list-things --attribute-name "wattage" --attribute-value "75"
```

```
{
    "things": [
        {
            "thingTypeName": "StopLight",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3,
            "thingName": "MyLightBulb"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
```

```
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

자세한 내용은 AWS CLI 명령 참조의 list-things를 참조하세요.

사물 설명

DescribeThing 명령을 사용하여 사물에 대한 자세한 정보를 표시할 수 있습니다.

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "version": 3,
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "StopLight",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

인라인 함수를 사용하여 규칙 엔진 내에서이 API에 액세스할 수도 있습니다`get_registry_data()`. 이 함수를 사용하면 AWS IoT 규칙 내에서 `DescribeThing` 직접 및 `ListThingGroupsForThing` APIs를 호출하여 사물 레지스트리 정보(속성, 사물 유형 및 그룹 멤버십 포함)에 동적으로 액세스하고 활용하여 디바이스 레지스트리 데이터를 기반으로 실시간 메시지 처리 및 라우팅을 활성화할 수 있습니다. 자세한 내용은 `get_registry_data` 단원을 참조하십시오.

자세한 내용은 AWS CLI 명령 참조의 describe-thing을 참조하세요.

사물 업데이트

UpdateThing 명령을 사용하여 사물을 업데이트할 수 있습니다. 이 명령은 사물의 속성만 업데이트합니다. 사물의 이름은 변경할 수 없습니다. 사물 이름을 변경하려면 새 사물을 생성하고 새 이름을 지정한 다음 이전 사물을 삭제해야 합니다.

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"150\", \"model\":\"456\"}}"
```

UpdateThing 명령은 출력을 생성하지 않습니다. DescribeThing 명령을 사용하여 결과를 확인할 수 있습니다.

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

자세한 내용은 AWS CLI 명령 참조의 update-thing을 참조하세요.

사물 삭제

다음과 같이 DeleteThing 명령을 사용하여 사물을 삭제할 수 있습니다.

```
$ aws iot delete-thing --thing-name "MyThing"
```

삭제에 성공하거나 존재하지 않는 사물을 지정하는 경우 이 명령은 오류가 발생하지 않고 성공적으로 반환합니다.

자세한 내용은 AWS CLI 명령 참조의 delete-thing을 참조하세요.

사물에 위탁자 연결

물리적 디바이스는 보안 주체를 사용하여 통신할 수 있습니다 AWS IoT. 위탁자는 X.509 인증서 또는 Amazon Cognito ID일 수 있습니다. attach-thing-principal 명령을 실행하여, 디바이스를 나타내는 레지스트리의 사물과 인증서 또는 Amazon Cognito ID를 연결할 수 있습니다.

인증서 또는 Amazon Cognito ID를 사물에 연결하려면 attach-thing-principal 명령을 사용합니다.

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

연결 유형(독점 연결 또는 비독점 연결)을 사용하여 사물에 인증서를 연결하려면 attach-thing-principal 명령을 사용하고 `--thing-principal-type` 필드에 유형을 지정합니다. 독점 연결은 사용자의 IoT 사물만 유일하게 인증서에 연결되어 있으며, 이 인증서를 다른 사물과는 연결할 수 없다는 것을 의미합니다. 비독점 연결은 사용자의 IoT 사물이 인증서에 연결되어 있고 이 인증서를 다른 사물과도 연결할 수 있다는 것을 의미합니다. 자세한 내용은 ??? 단원을 참조하십시오.

> **Note**
>
> 이 ??? 기능의 경우, X.509 인증서만 위탁자로 사용할 수 있습니다.

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb2" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --thing-principal-type "EXCLUSIVE_THING"
```

연결에 성공하면 AttachThingPrincipal 명령이 출력을 생성하지 않습니다. 연결을 설명하려면 list-thing-principals-v2 CLI 명령을 사용합니다.

자세한 내용은 AWS IoT Core API 참조의 AttachThingPrincipal을 참조하세요.

위탁자와 연결된 사물 나열

지정한 위탁자와 연결된 사물을 나열하려면, `list-principal-things` 명령을 실행합니다. 단, 이 명령은 사물과 인증서 간의 연결 유형을 나열하지 않습니다. 연결 유형을 나열하려면 `list-principal-things-v2` 명령을 사용합니다. 자세한 내용은 ??? 단원을 참조하십시오.

```
$ aws iot list-principal-things \
```

```
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

출력은 다음과 같습니다.

```
{
    "things": [
        "MyLightBulb1",
        "MyLightBulb2"
    ]
}
```

자세한 내용은 AWS IoT Core API 참조의 ListPrincipalThings를 참조하세요.

사물에 연결된 위탁자 나열

지정한 사물에 연결된 위탁자를 나열하려면, `list-thing-principals` 명령을 실행합니다. 단, 이 명령은 사물과 인증서 간의 연결 유형을 나열하지 않습니다. 연결 유형을 나열하려면 `list-thing-principals-v2` 명령을 사용합니다. 자세한 내용은 ??? 단원을 참조하십시오.

```
$ aws iot list-thing-principals \
    --thing-name "MyLightBulb1"
```

출력은 다음과 같습니다.

```
{
    "principals": [
        "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8",
        "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
    ]
}
```

자세한 내용은 AWS IoT Core API 참조의 ListThingPrincipals를 참조하세요.

위탁자와 연결된 사물 나열 V2

지정된 인증서와 연결된 사물을 연결 유형과 함께 나열하려면 list-principal-things-v2 명령을 실행합니다. 연결 유형은 인증서가 사물에 연결되는 방식을 나타냅니다.

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

출력은 다음과 같습니다.

```
{
    "PrincipalThingObjects": [
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        },
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_2"
        }

    ]
}
```

자세한 내용은 AWS IoT Core API 참조의 ListPrincipalThingsV2를 참조하세요.

사물에 연결된 위탁자 나열 V2

지정된 사물에 연결된 인증서를 연결 유형과 함께 나열하려면 list-thing-principals-V2 명령을 실행합니다. 연결 유형은 인증서가 사물에 연결되는 방식을 나타냅니다.

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

출력은 다음과 같습니다.

```
{
    "ThingPrincipalObjects": [
        {
```

```
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        },
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
        }
    ]
}
```

자세한 내용은 AWS IoT Core API 참조의 ListThingsPrincipalV2를 참조하세요.

사물에서 보안 주체 분리

다음과 같이 DetachThingPrincipal 명령을 사용하여 사물에서 인증서를 분리할 수 있습니다.

```
$ aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

DetachThingPrincipal 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT Core API 참조의 detach-thing-principal을 참조하세요.

사물 유형

사물 유형을 사용하면 동일한 사물 유형과 연결된 모든 사물에 공통된 설명 및 구성 정보를 저장할 수 있습니다. 그러면 사물 레지스트리에서 관리가 간소화됩니다. 예를 들어 LightBulb 사물 유형을 정의할 수 있습니다. LightBulb 사물 유형과 연결된 모든 사물은 일단의 속성, 일련번호, 제조업체, 와트 수를 공유합니다. LightBulb 사물 유형을 생성(또는 기존 사물의 유형을 LightBulb로 변경)할 때 LightBulb 사물 유형에 정의된 각 속성에 값을 지정할 수 있습니다.

사물 유형은 선택 사항이지만 사물 유형을 사용하면 사물을 보다 쉽게 검색할 수 있습니다.

- 사물 유형이 있는 사물은 최대 50개의 속성을 가질 수 있습니다.

- 사물 유형이 없는 사물은 최대 3개의 속성을 가질 수 있습니다.
- 사물은 한 사물 유형에만 연결될 수 있습니다.
- 계정에서 생성할 수 있는 사물 유형의 수에는 제한이 없습니다.

사물 유형이 생성된 이후에는 그 이름을 바꿀 수 없습니다. 언제라도 사물 유형을 사용 중지하여 새 사물이 연결되는 것을 방지할 수 있습니다. 또한 연결된 사물이 없는 사물 유형을 삭제할 수 있습니다.

주제:

- 사물 유형 생성
- 사물 유형 목록 표시
- 사물 유형 설명
- 사물에 사물 유형 연결
- 사물 유형 업데이트
- 사물 유형 사용 중지
- 사물 유형 삭제

사물 유형 생성

다음과 같이 CreateThingType 명령을 사용하여 사물 유형을 생성할 수 있습니다.

```
$ aws iot create-thing-type

                --thing-type-name "LightBulb" --thing-type-properties
 "thingTypeDescription=light bulb type, searchableAttributes=wattage,model"
```

CreateThingType 명령은 사물 유형 및 ARN을 포함하는 응답을 반환합니다.

```
{
    "thingTypeName": "LightBulb",
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb"
}
```

사물 유형 목록 표시

다음과 같이 ListThingTypes 명령을 사용하여 사물 유형을 나열할 수 있습니다.

```
$ aws iot list-thing-types
```

ListThingTypes 명령은 AWS 계정에 정의된 사물 유형의 목록을 반환합니다.

```
{
    "thingTypes": [
        {
            "thingTypeName": "LightBulb",
            "thingTypeProperties": {
                "searchableAttributes": [
                    "wattage",
                    "model"
                ],
                "thingTypeDescription": "light bulb type"
            },
            "thingTypeMetadata": {
                "deprecated": false,
                "creationDate": 1468423800950
            }
        }
    ]
}
```

사물 유형 설명

다음과 같이 DescribeThingType 명령을 사용하여 사물 유형에 대한 정보를 가져올 수 있습니다.

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

DescribeThingType 명령은 다음과 같이 지정한 유형에 대한 정보를 반환합니다.

```
{
    "thingTypeProperties": {
        "searchableAttributes": [
            "model",
            "wattage"
        ],
        "thingTypeDescription": "light bulb type"
    },
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
```

```
    "thingTypeName": "LightBulb",
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1544466338.399
    }
}
```

사물에 사물 유형 연결

사물을 생성할 때는 다음과 같이 CreateThing 명령을 사용하여 사물 유형을 지정할 수 있습니다.

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

언제라도 UpdateThing 명령을 사용하여 사물과 연결된 사물 유형을 변경할 수 있습니다.

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

또한 UpdateThing 명령을 사용하여 사물을 사물 유형에서 분리할 수도 있습니다.

사물 유형 업데이트

사물을 생성할 때는 다음과 같이 UpdateThingType 명령을 사용하여 사물 유형을 업데이트할 수 있습니다.

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

언제라도 UpdateThing 명령을 사용하여 사물과 연결된 사물 유형을 변경할 수 있습니다.

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

또한 UpdateThing 명령을 사용하여 사물을 사물 유형에서 분리할 수도 있습니다.

사물 유형 사용 중지

사물 유형은 변경이 불가능합니다. 사물 유형은 정의 후에 변경할 수 없습니다. 하지만 언제라도 사물 유형을 사용 중지하여 사용자가 새 사물을 연결하는 것을 방지할 수 있습니다. 사물 유형과 연결된 기존 사물은 변경되지 않습니다.

사물 유형을 사용 중지하려면 DeprecateThingType 명령을 사용합니다.

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

DescribeThingType 명령을 사용하여 결과를 확인할 수 있습니다.

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type",
    },
    "thingTypeMetadata": {
        "deprecated": true,
        "creationDate": 1468425854308,
        "deprecationDate": 1468446026349
    }
}
```

사물 유형 사용 중지는 취소 가능한 작업입니다. `--undo-deprecate` CLI 명령에서 DeprecateThingType 플래그를 사용하여 사용 중지를 실행 취소할 수 있습니다.

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

DescribeThingType CLI 명령을 사용하여 결과를 확인할 수 있습니다.

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/StopLight",
    "thingTypeId": "12345678abcdefgh12345678ijklmnop12345678"
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type"
    },
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1468425854308,
    }
}
```

사물 유형 삭제

사물 유형은 사용 중지한 후에만 삭제할 수 있습니다. 사물 유형을 삭제하려면 DeleteThingType 명령을 사용합니다.

```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

> **Note**
>
> 사물 유형을 삭제하려면 먼저 이를 사용 중지한 후 5분간 기다려야 합니다.

정적 사물 그룹

정적 사물 그룹을 사용하면 사물을 그룹별로 범주화하여 여러 사물을 한번에 관리할 수 있습니다. 정적 항목 그룹에는 콘솔, CLI 또는 API를 사용하여 관리되는 사물 그룹이 포함됩니다. 반면에 동적 사물 그룹에는 지정된 쿼리와 일치하는 사물이 포함됩니다. 정적 사물 그룹은 다른 정적 사물 그룹을 포함할 수 있으므로 그룹 계층 구조를 만들 수 있습니다. 정책은 상위 그룹에 연결할 수 있으며, 이렇게 연결된 정책은 하위 그룹을 비롯해 해당 그룹과 해당 그룹의 하위 그룹에 속한 모든 사물에게 상속됩니다. 따라서 사물의 수가 많더라도 권한을 쉽게 제어할 수 있습니다.

> **Note**
>
> 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. 자세한 내용은 AWS IoT Greengrass 개발자 안내서의 디바이스 인증 및 권한 부여를 참조하세요.

다음은 정적 사물 그룹으로 수행할 수 있는 사물입니다.

- 그룹을 생성, 설명 또는 삭제합니다.
- 사물을 단일 그룹 또는 다수의 그룹에 추가합니다.
- 그룹에서 사물을 제거합니다.
- 생성한 그룹을 나열합니다.
- 그룹의 하위 그룹(직/간접적 최하위 그룹)을 모두 나열합니다.
- 하위 그룹에 속한 모든 사물을 포함하여 그룹의 사물을 나열합니다.
- 그룹의 최상위 그룹(직/간접적 상위 그룹)을 모두 나열합니다.
- 그룹 속성을 추가하거나, 삭제하거나, 업데이트합니다. (속성이란 그룹에 대한 정보를 저장할 때 사용하는 이름-값 페어를 말합니다)
- 정책을 그룹에 연결하거나, 혹은 그룹에서 분리합니다.
- 그룹에 연결되어 있는 정책을 나열합니다.
- 사물에게 상속되는 정책을 나열합니다(그룹 또는 상위 그룹 중 하나에 연결된 정책에 따라).
- 그룹에 속한 사물에 로깅 옵션을 구성합니다. AWS IoT 로깅 구성을(를) 참조하세요.
- 그룹과 그룹의 하위 그룹에 속한 모든 사물로 전송된 후 실행되는 작업을 생성합니다. AWS IoT 작업을(를) 참조하세요.

> **Note**
>
> AWS IoT Core 정책이 연결된 정적 사물 그룹에 사물이 연결되면 사물 이름이 클라이언트 ID와 일치해야 합니다.

다음은 정적 사물 그룹에 적용되는 몇 가지 제한입니다.

- 그룹은 직접 상위 그룹을 1개까지만 가질 수 있습니다.

- 그룹이 다른 그룹의 하위 그룹인 경우 그룹 생성 시 이를 지정해야 합니다.
- 그룹의 상위는 나중에 변경할 수 없으므로 그룹에 포함되는 하위 그룹을 만들기 전에 그룹 계층을 계획하고 상위 그룹을 만들어야 합니다.
- 사물이 속할 수 있는 그룹의 수는 제한되어 있습니다.
- 사물 1개를 동일 계층에 속한 그룹 2개 이상에 추가할 수 없습니다. (다시 말해서 사물 1개를 상위 그룹을 공유하는 그룹 2개에 추가할 수 없습니다.)
- 그룹 이름은 변경할 수 없습니다.
- 사물 그룹 이름은 û, é, ñ 같은 국제 문자를 포함할 수 없습니다.
- 그룹 이름에 개인 식별 정보를 사용하면 안 됩니다. 사물 그룹 이름은 암호화되지 않은 통신 및 보고서에 나타날 수 있습니다.

정책을 그룹에 연결한 후 분리하면 여러 가지 유의적인 방법으로 AWS IoT 작업 보안을 강화할 수 있습니다. 디바이스 단위로 정책을 인증서에 연결하고 나서 인증서를 사물에 연결하려면 시간이 많이 걸릴 뿐만 아니라 다수의 디바이스에서 정책을 빠르게 업데이트하거나 변경하는 것이 어렵습니다. 하지만 정책을 사물 그룹에 연결하면 사물에서 인증서를 교체해야 할 때 여러 단계를 생략할 수 있습니다. 또한 그룹 멤버십을 교체할 때는 정책이 동적으로 사물에 적용되기 때문에 디바이스가 그룹의 멤버십을 교체할 때마다 복잡한 권한들을 다시 생성할 필요가 없습니다.

정적 물체 그룹 생성

CreateThingGroup 명령을 사용하여 정적 사물 그룹을 생성합니다.

```
$ aws iot create-thing-group --thing-group-name LightBulbs
```

CreateThingGroup 명령은 정적 사물 그룹의 이름, ID 및 ARN을 포함하는 응답을 반환합니다.

```
{
    "thingGroupName": "LightBulbs",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
}
```

> **Note**
>
> 사물 그룹 이름에 개인 식별 정보를 사용하지 않는 것이 좋습니다.

다음은 정적 사물 그룹을 생성할 때 정적 사물 그룹의 상위 그룹을 지정하는 예제입니다.

```
$ aws iot create-thing-group --thing-group-name RedLights --parent-group-name
 LightBulbs
```

앞에서 설명한 바와 같이 CreateThingGroup 명령은 정적 사물 그룹의 이름, ID 및 ARN을 포함하는 응답을 반환합니다.

```
{
    "thingGroupName": "RedLights",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
}
```

Important

사물 그룹 계층 구조를 생성할 때는 다음과 같은 제한 사항을 유의하세요.

- 사물 그룹은 직접 상위 그룹을 1개까지만 가질 수 있습니다.
- 사물 그룹이 가질 수 있는 직접 하위 그룹의 수는 제한되어 있습니다.
- 그룹 계층 구조의 최대 깊이는 제한되어 있습니다.
- 사물 그룹이 가질 수 있는 속성의 수는 제한되어 있습니다. (속성이란 그룹에 대한 정보를 저장할 때 사용하는 이름-값 페어를 말합니다) 각 속성 이름과 각 값의 길이도 제한되어 있습니다.

사물 그룹 설명

다음과 같이 DescribeThingGroup 명령을 사용하여 사물 그룹에 대한 정보를 가져올 수 있습니다.

```
$ aws iot describe-thing-group --thing-group-name RedLights
```

DescribeThingGroup 명령은 다음과 같이 지정한 그룹에 대한 정보를 반환합니다.

```
{
    "thingGroupName": "RedLights",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
    "thingGroupId": "12345678abcdefgh12345678ijklmnop12345678",
    "version": 1,
```

```
    "thingGroupMetadata": {
        "creationDate": 1478299948.882
        "parentGroupName": "Lights",
        "rootToParentThingGroups": [
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ShinyObjects",
                "groupName": "ShinyObjects"
            },
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs",
                "groupName": "LightBulbs"
            }
        ]
    },
    "thingGroupProperties": {
        "attributePayload": {
            "attributes": {
                "brightness": "3400_lumens"
            },
        },
        "thingGroupDescription": "string"
    },
}
```

정적 사물 그룹에 사물 추가

AddThingToThingGroup 명령을 사용하여 정적 사물 그룹에 사물을 추가할 수 있습니다.

```
$ aws iot add-thing-to-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

AddThingToThingGroup 명령은 출력을 생성하지 않습니다.

> **Important**
>
> 사물 1개를 추가할 수 있는 최대 그룹 수는 10개입니다. 하지만 사물 1개를 동일 계층에 속한 그룹 2개 이상에 추가할 수는 없습니다. (다시 말해서 사물 1개를 공통 상위 그룹을 공유하는 그룹 2개에 추가할 수 없습니다)
> 사물이 가능한 많은 사물 그룹에 속하고 이러한 그룹 중 하나 이상이 동적 사물 그룹인 경우 overrideDynamicGroups 플래그를 사용하면 정적 그룹이 동적 그룹보다 우선합니다.

정적 사물 그룹에서 사물 제거

다음과 같이 RemoveThingFromThingGroup 명령을 사용하여 사물을 그룹에서 제거할 수 있습니다.

```
$ aws iot remove-thing-from-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

RemoveThingFromThingGroup 명령은 출력을 생성하지 않습니다.

사물 그룹에 속한 사물 나열

다음과 같이 ListThingsInThingGroup 명령을 사용하여 그룹에 속한 사물을 나열할 수 있습니다.

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs
```

ListThingsInThingGroup 명령은 지정하는 그룹에 속한 사물 목록을 반환합니다.

```
{
    "things":[
        "TestThingA"
    ]
}
```

다음과 같이 `--recursive` 파라미터를 추가하면 그룹에 속한 사물과 해당 그룹의 모든 하위 그룹에 속한 사물을 나열할 수 있습니다.

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs --recursive
```

```
{
    "things":[
        "TestThingA",
        "MyLightBulb"
    ]
}
```

> **Note**
>
> 이 작업은 결과적 일관성을 갖습니다. 다시 말해서 사물 그룹을 변경하더라도 즉시 반영되지 않을 수도 있습니다.

사물 그룹 목록 표시

ListThingGroups 명령을 사용하여 계정의 사물 그룹을 나열할 수 있습니다.

```
$ aws iot list-thing-groups
```

ListThingGroups 명령은 AWS 계정다음의 사물 그룹 목록을 반환합니다.

```
{
    "thingGroups": [
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

다음과 같이 옵션으로 제공되는 필터를 사용하여 상위 그룹으로 지정된 그룹(`--parent-group`) 또는 이름이 지정된 접두사로 시작하는 그룹(`--name-prefix-filter`)을 나열할 수도 있습니다. 여기에 `--recursive` 파라미터를 추가하면 사물 그룹의 직접 하위 그룹 뿐만 아니라 모든 하위 그룹까지 나열할 수 있습니다.

```
$ aws iot list-thing-groups --parent-group LightBulbs
```

이 경우 ListThingGroups 명령은 AWS 계정다음에 정의된 사물 그룹의 직접 하위 그룹 목록을 반환합니다.

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        }
    ]
}
```

ListThingGroups 명령에 `--recursive` 파라미터를 사용하면 직접 하위 그룹들 외에도 사물 그룹의 모든 하위 그룹까지 나열합니다.

```
$ aws iot list-thing-groups --parent-group LightBulbs --recursive
```

ListThingGroups 명령은 사물 그룹의 모든 하위 그룹 목록을 반환합니다.

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
    ]
}
```

> **Note**
>
> 이 작업은 결과적 일관성을 갖습니다. 다시 말해서 사물 그룹을 변경하더라도 즉시 반영되지 않을 수도 있습니다.

사물이 속하는 그룹의 목록 표시

ListThingGroupsForThing 명령을 사용하여 사물이 속한 직접 그룹을 나열할 수 있습니다.

```
$ aws iot list-thing-groups-for-thing --thing-name MyLightBulb
```

ListThingGroupsForThing 명령은 해당 사물이 속하는 직접 사물 그룹 목록을 반환합니다.

```
{
    "thingGroups":[
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

인라인 함수를 사용하여 규칙 엔진 내에서이 API에 액세스할 수도 있습니다`get_registry_data()`. 이 함수를 사용하면 AWS IoT 규칙 내에서 `DescribeThing` 직접 및 `ListThingGroupsForThing` APIs를 호출하여 사물 레지스트리 정보(속성, 사물 유형 및 그룹 멤버십 포함)에 동적으로 액세스하고 활용하여 디바이스 레지스트리 데이터를 기반으로 실시간 메시지 처리 및 라우팅을 활성화할 수 있습니다. 자세한 내용은 `get_registry_data` 단원을 참조하십시오.

정적 사물 그룹 업데이트

UpdateThingGroup 명령을 사용하여 정적 사물 그룹의 속성을 업데이트할 수 있습니다.

```
$ aws iot update-thing-group --thing-group-name "LightBulbs" --thing-group-properties
 "thingGroupDescription=\"this is a test group\", attributePayload=\"{\"attributes
\"={\"Owner\"=\"150\",\"modelNames\"=\"456\"}}"
```

UpdateThingGroup 명령은 업데이트 이후 그룹의 버전 번호가 포함된 응답을 반환합니다.

```
{
    "version": 4
}
```

> **Note**
>
> 사물이 가질 수 있는 속성의 수는 제한되어 있습니다.

사물 그룹 삭제

사물 그룹을 삭제하려면 다음과 같이 DeleteThingGroup 명령을 사용합니다.

```
$ aws iot delete-thing-group --thing-group-name "RedLights"
```

DeleteThingGroup 명령은 출력을 생성하지 않습니다.

> **Important**
>
> 하위 사물 그룹이 있는 사물 그룹을 삭제하려고 하면 아래와 같이 오류 메시지가 발생합니다.
>
> ```
> A client error (InvalidRequestException) occurred when calling the
>  DeleteThingGroup
> operation: Cannot delete thing group : RedLights when there are still child
>  groups attached to it.
> ```
>
> 그룹을 삭제하려면 먼저 하위 그룹을 모두 삭제해야 합니다.

하위 사물이 포함된 그룹도 삭제할 수는 있지만 이 경우 그룹 멤버십에 따라 사물에 부여된 모든 권한이 더 이상 적용되지 않습니다. 따라서 정책이 연결되어 있는 그룹을 삭제하려면 먼저 부여된 권한을 제거할 경우 그룹에 속한 사물이 정상적인 기능을 하지 못하는지 주의 깊게 확인해야 합니다. 또한 사

물이 속한 그룹을 나열하는 명령(예: ListGroupsForThing)이 클라우드 레코드가 업데이트되는 동안에도 계속해서 그룹을 표시할 수 있는지 살펴봐야 합니다.

정적 사물 그룹에 정책 연결

AttachPolicy 명령을 사용하여 정적 사물 그룹에 정책을 연결하고, 그리고 여기에서 확장하여 해당 그룹에 속한 모든 사물과 해당 그룹의 하위 그룹에 속한 사물에 정책을 연결하는 것도 가능합니다.

```
$ aws iot attach-policy \
  --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs" \
  --policy-name "myLightBulbPolicy"
```

AttachPolicy 명령은 출력을 생성하지 않습니다.

Important

그룹에는 정책을 최대 2개까지 연결할 수 있습니다.

Note

정책 이름에 개인 식별 정보를 사용하지 않는 것이 좋습니다.

`--target` 파라미터는 사물 그룹 ARN(위와 같음), 인증서 ARN 또는 Amazon Cognito 자격 증명일 수 있습니다. 정책, 인증서 및 인증에 대한 자세한 내용은 단원을 참조하세요Authentication

자세한 내용은 AWS IoT Core 정책을 참조하세요.

정적 사물 그룹에서 정책 분리

다음과 같이 DetachPolicy 명령을 사용하여 사물 그룹에서 정책을 분리하고, 그리고 여기에서 확장하여 해당 그룹에 속한 모든 사물과 해당 그룹의 하위 그룹에 속한 사물에서 정책을 분리하는 것도 가능합니다.

```
$ aws iot detach-policy --target "arn:aws:iot:us-west-2:123456789012:thinggroup/
LightBulbs" --policy-name "myLightBulbPolicy"
```

DetachPolicy 명령은 출력을 생성하지 않습니다.

정적 사물 그룹에 연결되어 있는 정책 나열

ListAttachedPolicies 명령을 사용하여 정적 사물 그룹에 연결되어 있는 정책을 나열할 수 있습니다.

```
$ aws iot list-attached-policies --target "arn:aws:iot:us-
west-2:123456789012:thinggroup/RedLights"
```

`--target` 파라미터는 사물 그룹 ARN(위와 같음), 인증서 ARN 또는 Amazon Cognito 자격 증명일 수 있습니다.

`--recursive` 파라미터(선택 사항)를 추가하면 그룹의 상위 그룹에 연결되어 있는 모든 정책이 포함됩니다.

ListAttachedPolicies 명령은 정책 목록을 반환합니다.

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

정책이 연결되어 있는 그룹의 목록 표시

다음과 같이 ListTargetsForPolicy 명령을 사용하여 그룹을 포함하여 정책이 연결되어 있는 대상을 나열할 수 있습니다.

```
$ aws iot list-targets-for-policy --policy-name "MyLightBulbPolicy"
```

옵션으로 제공되는 `--page-size` *`number`* 파라미터를 추가하면 각 쿼리마다 반환되는 최대 결과 수를 지정하고, 이어지는 호출에서 `--marker` *`string`* 파라미터를 추가하면 다음 결과 집합(있는 경우)을 가져옵니다.

ListTargetsForPolicy 명령은 대상 목록과 더 많은 결과를 가져올 때 사용할 토큰을 반환합니다.

```
{
    "nextMarker": "string",
    "targets": [ "string" ... ]
```

```
}
```

사물에 적용되는 정책 가져오기

다음과 같이 GetEffectivePolicies 명령을 사용하여 사물이 속한 그룹(직접 상위 그룹 또는 간접 최상위 그룹)에 연결되어 있는 정책을 포함하여 사물에 적용되고 있는 정책을 나열할 수 있습니다.

```
$ aws iot get-effective-policies \
  --thing-name "MyLightBulb" \
  --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

`--principal` 파라미터를 사용하여 사물에 연결되어 있는 인증서 ARN을 지정합니다. Amazon Cognito 자격 증명 인증을 사용할 때는 `--cognito-identity-pool-id` 파라미터를 사용하고, 옵션으로 `--principal` 파라미터를 추가하여 Amazon Cognito 자격 증명을 지정합니다. `--cognito-identity-pool-id` 파라미터만 지정할 경우에는 인증을 받지 못한 사용자들을 위해 해당 자격 증명 풀의 역할에 연결된 정책이 반환됩니다. 둘 다 사용할 경우에는 인증 받은 사용자들을 위해 해당 자격 증명 풀의 역할에 연결된 정책이 반환됩니다.

`--thing-name` 파라미터는 선택 사항이기 때문에 `--principal` 파라미터 대신 사용할 수 있습니다. 이 파라미터를 사용하면 사물이 속한 모든 그룹에 연결된 정책과 해당 그룹의 모든 상위 그룹(계층 구조에서 루트 그룹까지)에 연결된 정책이 반환됩니다.

GetEffectivePolicies 명령은 정책 목록을 반환합니다.

```
{
    "effectivePolicies": [
        {
            "policyArn": "string",
            "policyDocument": "string",
            "policyName": "string"
        }
        ...
    ]
}
```

MQTT 작업에 대한 권한 부여 테스트

다음과 같이 TestAuthorization 명령을 사용하여 사물에서 MQTT 작업(`Publish`, `Subscribe`)의 허용 여부를 테스트할 수 있습니다.

```
aws iot test-authorization \
    --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --auth-infos "{\"actionType\": \"PUBLISH\", \"resources\": [ \"arn:aws:iot:us-
east-1:123456789012:topic/my/topic\"]}"
```

`--principal` 파라미터를 사용하여 사물에 연결되어 있는 인증서 ARN을 지정합니다. Amazon Cognito 자격 증명 인증을 사용할 때는 Cognito 자격 증명을 `--principal`로 지정하거나 `--cognito-identity-pool-id` 파라미터를 사용하거나, 혹은 둘 다를 사용합니다. `--cognito-identity-pool-id` 파라미터만 지정할 경우에는 인증을 받지 못한 사용자들을 위해 해당 자격 증명 풀의 역할에 연결된 정책을 고려합니다. 둘 다 사용할 경우에는 인증 받은 사용자들을 위해 해당 자격 증명 풀의 역할에 연결된 정책을 고려합니다.

`--auth-infos` 파라미터 이후에 리소스 집합과 작업 유형을 입력하여 테스트할 MQTT 작업을 1가지 이상 지정합니다. `actionType` 필드에는 "PUBLISH", "SUBSCRIBE", "RECEIVE" 또는 "CONNECT"를 입력해야 합니다. 그리고 `resources` 필드에는 리소스 ARN 목록을 입력해야 합니다. 자세한 내용은 AWS IoT Core 정책를 참조하세요.

`--policy-names-to-add` 파라미터에서 정책을 지정하여 정책 추가 효과를 테스트할 수 있습니다. 혹은 `--policy-names-to-skip` 파라미터에서 정책을 지정하여 정책 제거 효과를 테스트할 수도 있습니다.

또한 옵션으로 `--client-id` 파라미터를 사용하여 결과를 더욱 구체화하는 것도 가능합니다.

TestAuthorization 명령은 지정한 `--auth-infos` 쿼리 집합마다 허용되거나 거부된 작업에 대한 세부 정보를 반환합니다.

```
{
    "authResults": [
        {
            "allowed": {
                "policies": [
                    {
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            },
            "authDecision": "string",
            "authInfo": {
```

```
                "actionType": "string",
                "resources": [ "string" ]
            },
            "denied": {
                "explicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                },
                "implicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                }
            },
            "missingContextValues": [ "string" ]
        }
    ]
}
```

동적 사물 그룹

동적 사물 그룹은 레지스트리의 특정 검색 쿼리에서 생성됩니다. 디바이스 연결, 디바이스 섀도우 생성 및 AWS IoT Device Defender 위반 데이터와 같은 검색 쿼리 파라미터는 이를 지원합니다. 동적 사물 그룹은 디바이스의 데이터를 인덱싱, 검색 및 집계하기 위해 플릿 인덱싱을 활성화해야 합니다. 이를 생성하기 전에 플릿 인덱싱 검색 쿼리를 사용해 동적 사물 그룹의 사물을 미리 볼 수 있습니다. 자세한 내용은 플릿 인덱싱 및 쿼리 구문 섹션을 참조하세요.

> **Note**
>
> 동적 사물 그룹 실행은 레지스트리 실행에서 측정됩니다. 자세한 내용은 AWS IoT Core 측정 추가 정보를 참조하세요.

동적 사물 그룹은 정적 사물 그룹과 다음과 같은 차이가 있습니다.

- 사물 멤버십이 명시적으로 정의되지 않았습니다. 동적 사물 그룹을 생성하려면 그룹 멤버십을 정의하기 위해 검색 쿼리 문자열을 정의해야 합니다.
- 동적 사물 그룹은 계층 구조의 일부일 수 없습니다.
- 동적 사물 그룹에는 정책을 적용할 수 없습니다.
- 동적 사물 그룹은 일련의 다른 명령을 사용해 생성, 업데이트 및 삭제합니다. 다른 모든 작업의 경우 두 유형의 사물 그룹에 동일한 명령을 사용합니다.
- 당 동적 그룹 수는 AWS 계정 제한됩니다.

- 그룹 이름에 개인 식별 정보를 사용하면 안 됩니다. 사물 그룹 이름은 암호화되지 않은 통신 및 보고서에 나타날 수 있습니다.

정적 사물 그룹에 대한 자세한 내용은 정적 사물 그룹 단원을 참조하세요.

동적 사물 그룹의 사용 사례

다음 사용 사례에 동적 사물 그룹을 사용할 수 있습니다.

동적 사물 그룹을 작업의 대상으로 지정

동적 사물 그룹을 대상으로 하는 연속 작업을 생성하면 원하는 기준을 충족할 때 디바이스를 자동으로 대상으로 지정할 수 있습니다. 기준은 연결 상태이거나 소프트웨어 버전 또는 모델과 같은 레지스트리 또는 섀도우에 저장된 모든 기준일 수 있습니다. 동적 사물 그룹에 사물이 나타나지 않으면 작업에서 작업 문서를 받지 못합니다.

예를 들어 업데이트 프로세스 중 중단 위험을 최소화하기 위해 디바이스 플릿에 펌웨어 업데이트가 필요하고 배터리 수명이 80% 이상인 디바이스의 펌웨어만 업데이트하려는 경우입니다. 배터리 수명이 80%를 초과하는 디바이스만 포함하는 80PercentBatteryLife라는 동적 사물 그룹을 생성하고 이를 작업의 대상으로 사용할 수 있습니다. 배터리 수명 기준을 충족하는 디바이스만 펌웨어 업데이트를 받게 됩니다. 디바이스의 배터리 수명이 80%라는 기준에 도달하면 해당 디바이스는 동적 사물 그룹에 자동으로 추가되어 펌웨어 업데이트를 받습니다.

펌웨어 또는 운영 체제가 다른 여러 디바이스 모델이 있을 수 있으므로 새 소프트웨어 업데이트 버전이 다를 수 있습니다. 이는 연속 작업이 있는 동적 그룹의 가장 일반적인 사용 사례로, 각 디바이스 모델, 펌웨어 및 OS 조합에 대해 동적 그룹을 생성할 수 있습니다. 그런 다음 정의된 기준에 따라 디바이스가 자동으로 이러한 그룹의 멤버가 되면 소프트웨어 업데이트를 푸시하도록 이러한 각 동적 그룹에 연속 작업을 설정할 수 있습니다.

사물 그룹을 작업 대상으로 지정하는 방법에 대한 자세한 내용은 CreateJob을 참조하세요.

동적 그룹 멤버십 변경 사항을 사용하여 원하는 작업 수행

디바이스가 동적 사물 그룹에 추가되거나 동적 사물 그룹에서 제거될 때마다 레지스트리 이벤트 업데이트의 일부로 MQTT 주제로 알림이 전송됩니다. 동적 그룹 멤버십 업데이트를 기반으로 AWS 서비스와 상호 작용하도록 AWS IoT Core 규칙을 구성하고 원하는 작업을 수행할 수 있습니다. 예시 작업에는 Amazon DynamoDB에 쓰거나, Lambda 함수를 간접적으로 호출하거나 Amazon SNS 주제에 알림을 보내는 경우를 포함합니다.

자동 위반 탐지를 위해 동적 사물 그룹에 디바이스 추가

AWS IoT Device Defender 고객 감지는 동적 사물 그룹에 보안 프로필을 정의할 수 있습니다. 동적 사물 그룹의 디바이스는 그룹에 정의된 보안 프로필에 의해 위반이 있는지 자동으로 감지됩니다.

세분화된 로깅이 있는 디바이스를 관찰하기 위해 동적 사물 그룹에 대한 로그 수준 설정

동적 사물 그룹에 로그 수준을 지정할 수 있습니다. 이는 특정 기준을 충족하는 디바이스에 대해서만 로깅 수준 및 세부 정보를 사용자 지정하려는 경우에 유용합니다. 예를 들어 특정 펌웨어 버전의 디바이스가 특정 규칙의 게시된 주제에 오류를 유발한다고 의심되는 경우 이러한 문제를 디버깅하도록 세부 로깅을 설정할 수 있습니다. 이 경우 이 펌웨어 버전이 있는 모든 디바이스에 대해 동적 그룹을 생성할 수 있습니다. 이 그룹은 가 레지스트리 속성 또는 디바이스 섀도우로 저장된다고 가정합니다. 그런 다음 로깅 대상을 이 동적 사물 그룹으로 정의하여 디버그 수준을 설정할 수 있습니다. 세분화된 로깅에 대한 자세한 내용은 CloudWatch Logs AWS IoT 를 사용하여 모니터링을 참조하세요. 특정 사물 그룹에 대한 로깅 수준을 지정하는 방법에 대한 자세한 내용은 AWS IoT의 리소스별 로깅 구성을 참조하세요.

동적 사물 그룹 생성

CreateDynamicThingGroup 명령을 사용하여 동적 사물 그룹을 생성합니다. 80PercentBatteryLife 시나리오에 대한 동적 사물 그룹을 생성하려면 create-dynamic-thing-group CLI 명령을 사용합니다.

```
$ aws iot create-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --query-string "attributes.batterylife80"
```

> **Note**
>
> 그룹 이름에 개인 식별 정보를 사용하면 안 됩니다.

CreateDynamicThingGroup 명령은 응답을 반환합니다. 응답은 인덱스 이름, 쿼리 문자열, 쿼리 버전, 사물 그룹 이름, 사물 그룹 ID 및 사물 그룹의 Amazon 리소스 이름(ARN)을 포함합니다.

```
{
    "indexName": "AWS_Things",
    "queryVersion": "2017-09-30",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx"
}
```

동적 사물 그룹의 생성은 한 번에 이루어지지 않습니다. 동적 사물 그룹 채우기를 완료하는 데에는 시간이 걸립니다. 동적 사물 그룹이 생성되면 그룹의 상태가 BUILDING으로 설정됩니다. 채우기가 완료되면 상태가 ACTIVE로 바뀝니다. 동적 사물 그룹의 상태를 확인하려면 DescribeThingGroup 명령을 사용하세요.

동적 사물 그룹 설명

다음과 같이 DescribeThingGroup 명령을 사용하여 동적 사물 그룹에 대한 정보를 가져올 수 있습니다.

```
$ aws iot describe-thing-group --thing-group-name "80PercentBatteryLife"
```

DescribeThingGroup 명령은 다음과 같이 지정한 그룹에 대한 정보를 반환합니다.

```
{
    "status": "ACTIVE",
    "indexName": "AWS_Things",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1548716921.289
    },
    "thingGroupProperties": {},
    "queryVersion": "2017-09-30",
    "thingGroupId": "84dd9b5b-2b98-4c65-84e4-be0e1ecf4fd8"
}
```

동적 사물 그룹에 대해 DescribeThingGroup을 설정하면 동적 사물 그룹에 고유한 속성이 반환됩니다. 반환 속성의 예는 queryString과 상태입니다.

동적 사물 그룹의 상태 값은 다음과 같습니다.

`ACTIVE`

동적 사물 그룹을 사용할 준비가 되었습니다.

`BUILDING`

동적 사물 그룹이 생성 중이고 사물 멤버십이 처리 중입니다.

`REBUILDING`

그룹의 검색 쿼리 조정 후 동적 사물 그룹의 멤버십을 업데이트하는 중입니다.

> **Note**
>
> 동적 사물 그룹을 생성하면 상태에 상관 없이 해당 그룹을 사용합니다. `ACTIVE` 상태인 동적 사물 그룹에만 동적 사물 그룹에 대한 검색 쿼리와 일치하는 사물이 모두 포함됩니다. `BUILDING` 및 `REBUILDING` 상태인 동적 사물 그룹에는 검색 쿼리와 일치하는 사물 중 일부가 포함되지 않을 수 있습니다.

동적 사물 그룹 업데이트

그룹의 검색 쿼리를 포함한 동적 사물 그룹의 속성은 UpdateDynamicThingGroup 명령을 사용해 업데이트합니다. 다음 명령은 두 속성을 업데이트합니다. 하나는 사물 그룹 설명이고 다른 하나는 멤버십 기준을 배터리 수명을 85 이상으로 변경하는 쿼리 문자열입니다.

```
$ aws iot update-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --thing-
group-properties "thingGroupDescription=\"This thing group contains devices with a
 battery life greater than 85 percent.\"" --query-string "attributes.batterylife85"
```

UpdateDynamicThingGroup 명령은 업데이트 이후 그룹의 버전 번호가 포함된 응답을 반환합니다.

```
{
```

```
    "version": 2
}
```

동적 사물 그룹의 업데이트는 한 번에 수행되지 않습니다. 동적 사물 그룹 채우기를 완료하는 데에는 시간이 걸립니다. 동적 사물 그룹이 업데이트할 때 해당 그룹이 멤버십을 업데이트하는 동안 그룹의 상태가 REBUILDING으로 변경됩니다. 채우기가 완료되면 상태가 ACTIVE로 바뀝니다. 동적 사물 그룹의 상태를 확인하려면 DescribeThingGroup 명령을 사용하세요.

동적 사물 그룹 삭제

동적 사물 그룹은 DeleteDynamicThingGroup 명령을 사용하여 삭제합니다.

```
$ aws iot delete-dynamic-thing-group --thing-group-name "80PercentBatteryLife"
```

DeleteDynamicThingGroup 명령은 출력을 생성하지 않습니다.

사물이 속한 그룹을 나열하는 명령(예: ListGroupsForThing)이 클라우드 레코드가 업데이트되는 동안에도 계속해서 그룹을 표시할 수 있는지 살펴봐야 합니다.

동적 및 정적 사물 그룹 제한

동적 사물 그룹은 정적 사물 그룹과 다음 제한을 공유합니다.

- 사물 그룹이 가질 수 있는 속성의 수는 제한되어 있습니다.
- 사물이 속할 수 있는 그룹의 수는 제한되어 있습니다.
- 그룹 이름은 변경할 수 없습니다.
- 사물 그룹 이름은 û, é, ñ 같은 국제 문자를 포함할 수 없습니다.

동적 사물 그룹 제한 사항

동적 사물 그룹에는 다음과 같은 제한이 있습니다.

플릿 인덱싱

플릿 인덱싱 서비스를 활성화하면 디바이스 플릿에서 검색 쿼리를 수행할 수 있습니다. 플릿 인덱싱 채우기가 완료된 후 동적 사물 그룹을 생성하고 관리할 수 있습니다. 채우기 프로세스의 완료 시간은 AWS 클라우드에 등록된 디바이스 플릿의 크기에 따라 직접 영향을 받습니다. 동적 사물 그룹에 대해

플릿 인덱싱 서비스를 활성화하면 동적 사물 그룹을 모두 삭제해야 이 서비스를 비활성화할 수 있습니다.

> **Note**
>
> 플릿 인덱스를 쿼리할 권한이 있는 경우 전체 플릿에서 사물 데이터에 액세스할 수 있습니다.

동적 사물 그룹의 수는 제한되어 있습니다.

동적 사물 그룹의 수는 제한되어 있습니다.

성공한 명령은 오류를 로깅할 수 있습니다.

동적 사물 그룹을 생성하거나 업데이트할 때 일부 사물은 동적 사물 그룹에 추가할 수 있지만 아직 추가되지 않았을 수 있습니다. 이 시나리오는 오류를 로깅하고 `AddThingToDynamicThingGroupsFailed` 지표를 생성하는 동안 성공적인 생성 또는 업데이트 명령을 유발합니다. 단일 지표는 여러 로그 항목을 나타낼 수 있습니다.

CloudWatch 로그의 오류 로그 항목은 다음과 같은 경우에 생성됩니다.

- 적격 사물은 동적 사물 그룹에 추가할 수 없습니다.
- 동적 사물 그룹에서 사물을 제거하여 다른 그룹에 추가합니다.

사물을 동적 사물 그룹에 추가할 수 있게 되면 다음 사항을 고려합니다.

- 사물이 이미 가능한 많은 그룹에 있습니까? (제한 참조)
  - 아니오: 사물이 동적 사물 그룹에 추가됩니다.
  - 예: 사물이 동적 사물 그룹의 멤버입니까?
    - 아니오: 사물을 동적 사물 그룹에 추가할 수 없고, 오류가 로깅되며 `AddThingToDynamicThingGroupsFailed` 지표가 생성됩니다.
    - 예: 참가할 동적 사물 그룹이 사물이 이미 멤버인 동적 사물 그룹보다 오래되었습니까?
      - 아니오: 사물을 동적 사물 그룹에 추가할 수 없고, 오류가 로깅되며 `AddThingToDynamicThingGroupsFailed` 지표가 생성됩니다.
      - 예: 최근 동적 사물 그룹에서 사물을 제거하고 오류를 로깅하고 사물을 동적 사물 그룹에 추가합니다. 이렇게 하면 사물이 제거된 동적 사물 그룹에 대한 오류 및 `AddThingToDynamicThingGroupsFailed` 지표가 생성됩니다.

동적 사물 그룹의 사물이 더 이상 검색 쿼리를 충족하지 않으면 동적 사물 그룹에서 제거됩니다. 마찬가지로 동적 사물 그룹의 검색 쿼리를 충족하도록 사물이 업데이트되면 이전에 설명한 대로 그룹에 추가됩니다. 이러한 추가 및 제거는 정상이며 오류 로그 항목을 생성하지 않습니다.

`overrideDynamicGroups`를 활성화하면 정적 그룹이 동적 그룹보다 우선합니다.

사물이 속할 수 있는 그룹의 수는 제한되어 있습니다. AddThingToThingGroup 또는 UpdateThingGroupsForThing 명령을 사용하여 사물 멤버십을 업데이트할 때 `--overrideDynamicGroups` 파라미터를 추가하면 정적 사물 그룹이 동적 사물 그룹보다 우선합니다.

정적 사물 그룹에 사물을 추가할 때 다음 사항을 고려합니다.

- 사물이 이미 최대 그룹 수에 속해 있습니까?
  - 아니오: 사물이 정적 사물 그룹에 추가됩니다.
  - 예: 사물이 동적 그룹에 있습니까?
    - 아니요(NO): 사물을 사물 그룹에 추가할 수 없습니다. 이 명령은 예외를 발생시킵니다.
    - 예: --overrideDynamicGroups가 활성화되었습니까?
      - 아니요(NO): 사물을 사물 그룹에 추가할 수 없습니다. 이 명령은 예외를 발생시킵니다.
      - 예: 가장 최근에 생성된 동적 사물 그룹에서 사물이 제거되고 오류가 로깅되며 사물이 제거된 동적 사물 그룹에 대한 `AddThingToDynamicThingGroupsFailed` 지표가 생성됩니다. 그런 다음 사물이 정적 사물 그룹에 추가됩니다.

이전 동적 사물 그룹이 최신 사물 그룹보다 우선합니다.

사물이 속할 수 있는 그룹의 수는 제한되어 있습니다. 생성 또는 업데이트 작업이 사물에 대한 추가 그룹 자격을 생성하고 해당 사물이 그룹 한도에 도달하면 다른 동적 사물 그룹에서 제거되어 이 추가를 활성화할 수 있습니다. 발생 방법에 대한 자세한 내용은 성공한 명령은 오류를 로깅할 수 있습니다. 및 `overrideDynamicGroups`를 활성화하면 정적 그룹이 동적 그룹보다 우선합니다.에서 예제를 참조하세요.

사물이 동적 사물 그룹에서 제거되면 오류가 로깅되고 이벤트가 발생합니다.

정책을 동적 항목 그룹에 적용할 수 없습니다.

동적 사물 그룹에 정책을 적용하려고 하면 예외가 생성됩니다.

동적 사물 그룹 멤버십은 일관성을 가짐

레지스트리에 대해서는 사물의 최종 상태만 평가됩니다. 상태가 빠르게 업데이트되는 경우에는 중간 상태를 건너뛸 수 있습니다. 중간 상태에 따라 멤버십이 달라지는 동적 사물 그룹에는 규칙 또는 작업을 연결하지 마세요.

AWS IoT 사물을 MQTT 클라이언트 연결에 연결

독점 사물 연결은 X.509 인증서를 단일 AWS IoT 사물에 연결하는 경우를 말합니다. 이 경우에 해당 인증서를 다른 사물과 함께 사용할 수 없습니다. 인증서가 단일 IoT 사물에서만 사용되도록 보장하면, 보안 취약성을 방지하는 데 도움이 됩니다.

에서 AWS IoT클라이언트 ID는 AWS IoT Core MQTT 브로커에 연결할 때 사물 또는 디바이스의 고유 식별자입니다. 비독점 연결을 사용하는 경우에 여러 사물을 동일한 인증서에 연결할 수 있습니다. 비독점 사물 연결이 이루어졌을 경우, 명확한 연결 관계를 유지하면서 잠재적 충돌을 방지하려면 클라이언트 ID가 사물 이름과 일치해야 합니다.

이 주제의 내용

- 사용 사례
- 연결에 사물을 연결하는 방법

사용 사례

사물을 연결에 연결하면 다음과 같은 기능이 제공됩니다.

> **Note**
>
> IoT 사물과 클라이언트 연결 사이에 비독점 연결이 이루어진 경우, 수명 주기 이벤트 기능을 제외한 다음 기능을 모두 사용할 수 있습니다. 수명 주기 이벤트 메시지에 사물 이름을 포함하려면 IoT 사물과 클라이언트 연결 사이에 독점 연결이 이루어져야 합니다.

사물 정책 변수 - 사물 정책 변수를 사용하여 AWS IoT API 작업에 대한 디바이스 액세스를 승인할 수 있습니다. 이러한 변수를 사용하면 이름, 유형 및 속성 값과 같은 사물 속성을 기반으로 권한을 부여하거나 거부하는 AWS IoT Core 정책을 작성할 수 있습니다. 사물 정책 변수를 사용하여 동일한 정책을 적용하여 여러 AWS IoT Core 디바이스를 제어할 수 있습니다. 이를 통해 정책 관리를 간소화하고 리소스 중복을 줄일 수 있습니다. 자세한 내용은 사물 정책 변수를 참조하세요.

수명 주기 이벤트 - 수명 주기 이벤트(예: 연결, 연결 해제, 구독, 구독 취소)에서 사물 이름을 수신할 수 있습니다. 따라서 메시지(예: 규칙)에 포함된 사물 이름을 처리할 수 있습니다. 자세한 내용은 수명 주기 이벤트를 참조하세요.

리소스별 로깅 - 사물 그룹에 대한 리소스별 로깅을 구성하고, 정의된 사물 그룹 내의 모든 사물에 원하는 로깅 구성을 쉽게 적용할 수 있습니다. 자세한 내용은 ??? 단원을 참조하십시오.

비용 할당 - 비용 할당을 위한 사용자 지정 태그가 있는 결제 그룹을 생성하고, 이러한 결제 그룹에 사물을 추가할 수 있습니다. 자세한 내용은 결제 그룹을 참조하세요.

연결에 사물을 연결하는 방법

클라이언트 ID가 레지스트리의 사물 이름과 일치하는 경우 X.509 인증서를 해당 IoT 사물에 연결하면 AWS IoT Core 가 클라이언트 연결을 사물과 연결합니다. 클라이언트 ID가 레지스트리의 사물 이름과 일치하지 않는 경우, 사물에 X.509 인증서를 독점적으로 연결하여 이 연결을 설정할 수 있습니다. 독점 연결이 된 사물을 독점 사물이라고 합니다. 그렇지 않으면 비독점 사물이라고 합니다. 인증서가 독점 사물과 연결된 경우, 이 인증서는 독점 사물과 분리된 경우에만 다른 사물과 연결할 수 있습니다. 이 섹션에서 AWS Management Console 또는 AWS CLI 를 선택하여 사물을 연결에 연결합니다.

AWS Management Console

AWS Management Console을 사용하여 사물에 인증서를 독점적으로 연결하려면

1. AWS IoT 콘솔에서 AWS IoT 홈 페이지를 엽니다. 왼쪽 탐색 창의 보안에서 인증서를 선택합니다.
2. 인증서 페이지에서 사물을 연결할 인증서를 선택합니다. 그런 다음 페이지 오른쪽 상단의 작업에서 사물에 연결을 선택합니다.

   또는 인증서를 선택하고 인증서 세부 정보 페이지로 이동합니다. 사물 탭을 선택한 다음 사물에 연결을 선택합니다.
3. 사물에 인증서 연결 페이지에서 연결에 사물 연결 확인란을 선택합니다. 그런 다음 사물 드롭다운 목록에서 이 인증서와 연결할 사물을 선택합니다.
4. 사물 연결을 선택합니다. 작업이 성공하면 “인증서에 사물을 성공적으로 연결했습니다”라는 배너가 표시되고 연결된 사물이 사물 탭에 추가됩니다.

를 사용하여 독점 사물에서 인증서를 분리하려면 AWS Management Console

1. AWS IoT 콘솔에서 AWS IoT 홈 페이지를 엽니다. 왼쪽 탐색 창의 보안에서 인증서를 선택합니다.
2. 인증서 페이지에서 인증서를 선택하고 인증서 세부 정보 페이지로 이동합니다.

3. 인증서 세부 정보 페이지에서 사물을 선택합니다. 그런 다음, 인증서와 분리할 사물을 선택합니다. 사물 분리를 선택합니다.
4. 사물 분리 창에서 작업을 확인합니다. 분리를 선택합니다. 작업이 성공하면 "인증서에서 사물을 성공적으로 분리했습니다"라는 배너가 표시되고 사물 탭에 분리된 해당 사물이 더 이상 표시되지 않습니다.

AWS CLI

1. 를 사용하여 사물에 인증서를 연결하려면 attach-thing-principal 명령을 AWS CLI실행합니다. 독점 certificate-to-thing 연결을 지정하려면 `--thing-principal-type` 필드에 `EXCLUSIVE_THING`를 지정해야 합니다. 예제 명령은 다음과 같을 수 있습니다.

```
aws iot attach-thing-principal \
    --thing-name "thing_1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-principal-type "EXCLUSIVE_THING"
```

   이 명령은 출력을 생성하지 않습니다. 자세한 내용은 ??? 단원을 참조하십시오.

2. 지정된 인증서와 연결된 사물을 연결 유형과 함께 나열하려면 `list-principal-things-v2` 명령을 실행합니다. 연결 유형은 인증서가 사물에 연결되는 방식을 나타냅니다. 예제 명령은 다음과 같을 수 있습니다.

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

   출력은 다음과 같습니다.

```
{
    "PrincipalThingObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        }
    ]
```

```
}
```

자세한 내용은 ??? 단원을 참조하십시오.

3. 지정된 사물과 연결된 위탁자를 연결 유형과 함께 나열하려면 `list-thing-principals-v2` 명령을 실행합니다. 연결 유형은 인증서가 사물에 연결되는 방식을 나타냅니다. 예제 명령은 다음과 같을 수 있습니다.

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

출력은 다음과 같습니다.

```
{
    "ThingPrincipalObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        }
    ]
}
```

자세한 내용은 ??? 단원을 참조하십시오.

4. 사물에서 인증서를 분리하려면 detach-thing-principal 명령을 실행합니다.

```
aws iot detach-thing-principal \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-name "thing_1"
```

이 명령은 출력을 생성하지 않습니다. 자세한 내용은 ??? 단원을 참조하십시오.

메시지 보강을 위한 전파 속성 추가

에서는 사물 속성 또는 연결 세부 정보의 컨텍스트 메타데이터인 전파 속성을 추가하여 디바이스의 MQTT 메시지를 보강 AWS IoT Core할 수 있습니다. 메시지 보강이라고 하는 이 프로세스는 다양한 시

나리오에서 유용할 수 있습니다. 예를 들어, 디바이스 측을 변경하거나 규칙을 사용하지 않고도 모든 인바운드 게시 작업의 메시지를 보강할 수 있습니다. 전파 속성을 활용하면 복잡하게 규칙을 구성하거나 재게시 구성을 관리하지 않고도 더 효율적이고 비용 효율적인 방법으로 IoT 데이터를 보강할 수 있습니다.

메시지 보강 기능은 기본 수집 및 메시지 브로커를 사용하는 AWS IoT Core 고객이 사용할 수 있습니다. 디바이스를 게시할 때 모든 MQTT 버전을 사용할 수 있지만, 구독자(메시지를 사용하는 애플리케이션 또는 서비스)는 전파 속성으로 보강된 메시지를 수신하려면 MQTT 5를 지원해야 합니다. 보강된 메시지는 디바이스에서 게시된 모든 메시지에 MQTT 5 사용자 속성으로 추가됩니다. 규칙을 사용하는 경우, get_user_properties 함수를 활용하여, 데이터를 기반으로 메시지 라우팅 또는 처리를 위해 보강된 데이터를 검색할 수 있습니다.

에서 AWS Management Console 또는를 사용하여 사물 유형을 생성하거나 업데이트할 때 전파 속성을 추가할 AWS IoT Core수 있습니다 AWS CLI.

> **Important**
>
> 전파 속성을 추가할 때, 메시지를 게시하는 클라이언트가 인증서로 인증되었는지 확인해야 합니다. 자세한 내용은 클라이언트 인증 단원을 참조하십시오.

> **Note**
>
> 콘솔 내에서 MQTT 테스트 클라이언트를 사용하여 이 기능을 테스트하려고 할 경우, 이 기능을 사용하기 위해서는 연결된 인증서로 인증된 MQTT 클라이언트가 필요하므로 해당 기능이 작동하지 않을 수 있습니다.

AWS Management Console

를 사용하여 메시지 보강에 대한 전파 속성을 추가하려면 AWS Management Console

1. AWS IoT 콘솔에서 AWS IoT 홈 페이지를 엽니다. 왼쪽 탐색 창의 관리에서 모든 디바이스를 선택합니다. 그 다음에 사물 유형을 선택합니다.
2. 사물 유형 페이지에서 사물 유형 생성을 선택합니다.

   사물 유형을 업데이트하여 메시지 보강을 구성하려면, 사물 유형을 선택합니다. 그 다음에 사물 유형 세부 정보 페이지에서 업데이트를 선택합니다.

3. 사물 유형 생성 페이지의 사물 유형 속성에서 사물 유형 정보를 선택하거나 입력합니다.

   사물 유형을 업데이트하도록 선택하면, 이전 단계에서 업데이트를 선택한 후에 사물 유형 속성이 표시됩니다.

4. 추가 구성에서 전파 속성을 펼칩니다. 그 다음에 사물 속성을 선택하고, 게시된 MQTT5 메시지에 채우고자 하는 사물 속성을 입력합니다. 콘솔을 사용하여, 최대 3개의 사물 속성을 추가할 수 있습니다.

   전파 속성 섹션에서 연결 속성을 선택하고, 속성 유형을 입력합니다. 속성 이름은 선택 입력 사항입니다.

5. 원한다면, 태그를 추가합니다. 그 다음에 사물 유형 생성을 선택합니다.

   사물 유형을 업데이트하도록 선택한 경우, 사물 유형 업데이트를 선택합니다.

AWS CLI

1. 를 사용하여 새 사물 유형을 생성하여 메시지 보강에 대한 전파 속성을 추가하려면 create-thing-type 명령을 AWS CLI실행합니다. 예제 명령은 다음과 같을 수 있습니다.

```
aws iot create-thing-type \
    --thing-type-name "LightBulb" \
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

   명령의 출력은 다음과 같을 수 있습니다.

```
{
 "thingTypeName": "LightBulb",
 "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
 "thingTypeId": "ce3573b0-0a3c-45a7-ac93-4e0ce14cd190"
}
```

2. 를 사용하여 사물 유형을 업데이트하여 메시지 보강을 구성하려면 update-thing-type 명령을 AWS CLI실행합니다. 단, 이 명령을 실행할 때 `mqtt5Configuration`만 업데이트할 수 있습니다. 예제 명령은 다음과 같을 수 있습니다.

```
aws iot update-thing-type \
    --thing-type-name "MyThingType" \
```

```
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

이 명령은 출력을 생성하지 않습니다.

3. 사물 유형을 설명하려면 `describe-thing-type` 명령을 실행합니다. 이 명령은 `thing-type-properties` 필드에 메시지 보강 구성 정보가 포함된 출력을 생성합니다. 예제 명령은 다음과 같을 수 있습니다.

```
aws iot describe-thing-type \
    --thing-type-name "LightBulb"
```

출력은 다음과 같습니다.

```
{
 "thingTypeName": "LightBulb",
 "thingTypeId": "bdf72512-0116-4392-8d79-bf39b17ef73d",
 "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/LightBulb",
 "thingTypeProperties": {
  "mqtt5Configuration": {
   "propagatingAttributes": [
    {
     "userPropertyKey": "iot:ClientId",
     "connectionAttribute": "iot:ClientId"
    },
    {
     "userPropertyKey": "test",
     "thingAttribute": "attribute"
    }
   ]
  }
 },
 "thingTypeMetadata": {
  "deprecated": false,
  "creationDate": "2024-10-18T17:37:46.656000+00:00"
 }
}
```

자세한 내용은 ??? 단원을 참조하십시오.

AWS IoT 리소스에 태그 지정

사물 그룹, 사물 유형, 주제 규칙, 작업, 예정된 감사 및 보안 프로필을 쉽게 관리 및 구성하기 위해 이러한 각 리소스에 고유한 메타데이터를 태그의 형태로 할당할 수 있습니다. 이 단원에서는 태그를 설명하고 태그를 생성하는 방법을 보여 줍니다.

사물과 관련된 비용을 쉽게 관리할 수 있도록 사물이 포함된 결제 그룹을 생성할 수 있습니다. 그런 다음 각 결제 그룹에 메타데이터가 포함된 태그를 할당할 수 있습니다. 이 단원에서는 결제 그룹과 태그를 생성 및 관리하는 데 사용할 수 있는 명령에 대해서도 알아봅니다.

태그 기본 사항

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

추가하거나 적용한 태그를 기준으로 리소스를 검색하고 필터링할 수 있습니다. 또한 결제 그룹을 사용하여 비용을 분류 및 추적할 수 있습니다. IAM 정책에 태그 사용의 설명처럼 리소스에 대한 액세스 제어에 태그를 사용할 수도 있습니다.

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

AWS CLI 및 AWS IoT API를 사용하여 태그로 작업할 수도 있습니다. 다음 명령에서 Tags 필드를 사용하여 태그를 생성할 때 사물 그룹, 사물 유형, 주제 규칙, 작업, 보안 프로필, 정책, 결제 그룹 및 사물과 연결된 패키지 및 버전과 태그를 연결할 수 있습니다.

- CreateBillingGroup
- CreateDestination
- CreateDeviceProfile
- CreateDynamicThingGroup
- CreateJob
- CreateOTAUpdate

- CreatePolicy
- CreateScheduledAudit
- CreateSecurityProfile
- CreateServiceProfile
- CreateStream
- CreateThingGroup
- CreateThingType
- CreateTopicRule
- CreateWirelessGateway
- CreateWirelessDevice

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

- TagResource
- ListTagsForResource
- UntagResource

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

태그 규제 및 제한

태그에 적용되는 기본 제한 사항은 다음과 같습니다.

- 리소스 당 최대 태그 수 - 50개
- 최대 키 길이 - UTF-8의 유니코드 문자 127자
- 최대 값 길이 – UTF8의 유니코드 문자 255자
- 태그 키와 값은 대소문자를 구분합니다.
- 태그 이름이나 값에서 접두사 `aws:`는 사용하지 마세요. 이는 AWS 사용을 위해 예약되어 있습니다. 이 접두사가 지정된 태그 이름이나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.

- 태깅 스키마를 여러 서비스와 리소스에서 사용하는 경우 다른 서비스에서는 허용되는 문자에 제한이 있을 수 있다는 점에 주의하세요. 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 공백 및 숫자와 + - = . _ : / @ 등의 특수 문자입니다.

IAM 정책에 태그 사용

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

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

> **Note**
>
> IAM 정책의 조건 컨텍스트 키 및 값은 태그를 지정할 수 있는 리소스의 식별자가 필수 파라미터인 AWS IoT 작업에만 적용됩니다. 예를 들어 이 요청에서는 태그 지정이 가능한 리소스(사물 그룹, 사물 유형, 주제 규칙, 작업 또는 보안 프로필)를 참조하지 않기 때문에 조건 컨텍스트 키 및 값을 기반으로 DescribeEndpoint를 사용할 수 없거나 거부됩니다. 태그 지정이 가능한 AWS IoT 리소스 및 지원하는 조건 키에 대한 자세한 내용은 작업, 리소스 및 조건 키를 AWS IoT 참조하세요.

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

다음은 `ThingGroup` 작업에 대해 2개의 태그 기반 제한을 적용하는 정책 예제입니다. 이 정책으로 제한되는 IAM 사용자는 다음과 같습니다.

- 사물 그룹을 "env=prod" 태그로 생성할 수 없습니다(예에서 `"aws:RequestTag/env" : "prod"` 줄 참조).

- 기존 태그 "env=prod"가 지정된 사물 그룹을 수정 또는 액세스할 수 없습니다. 이 예에서는 `"aws:ResourceTag/env" : "prod"` 행을 참조하세요.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iot:CreateThingGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

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

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

> **Note**
>
> 태그를 기준으로 리소스에 대한 사용자 액세스를 허용 또는 거부하는 경우 동일한 리소스에서 태그를 추가 또는 제거할 수 있도록 사용자를 명시적으로 거부할 것을 고려해야 합니다. 그렇지 않으면 사용자가 제한을 피해 태그를 수정하여 리소스에 대한 액세스 권한을 얻을 수 있습니다.

결제 그룹

AWS IoT 에서는 개별 사물에 직접 태그를 적용할 수 없지만 결제 그룹에 사물을 배치하고 여기에 태그를 적용할 수 있습니다. 의 경우 태그를 기반으로 비용 및 사용량 데이터를 AWS IoT할당하는 것은 결제 그룹으로 제한됩니다.

AWS IoT Core 무선 디바이스 및 게이트웨이와 같은 for LoRaWAN 리소스는 결제 그룹에 추가할 수 없습니다. 하지만 결제 그룹에 추가할 수 있는 AWS IoT 사물과 연결할 수 있습니다.

다음과 같은 명령을 사용할 수 있습니다.

- AddThingToBillingGroup은 결제 그룹에 사물을 추가합니다.
- CreateBillingGroup은 결제 그룹을 생성합니다.
- DeleteBillingGroup은 결제 그룹을 삭제합니다.
- DescribeBillingGroup은 결제 그룹에 대한 정보를 반환합니다.
- ListBillingGroups은 사용자가 생성한 결제 그룹을 나열합니다.
- ListThingsInBillingGroup은 지정된 결제 그룹에 추가한 사물을 나열합니다.
- RemoveThingFromBillingGroup은 결제 그룹에서 지정된 사물을 제거합니다.

- UpdateBillingGroup은 결제 그룹에 대한 정보를 업데이트합니다.
- CreateThing은 사물을 생성할 때 사물에 대한 결제 그룹을 지정하도록 허용합니다.
- DescribeThing은 사물이 속한 결제 그룹(있는 경우)을 포함해 사물에 대한 설명을 반환합니다.

AWS IoT Wireless API는 무선 디바이스 및 게이트웨이를 AWS IoT 사물과 연결하는 이러한 작업을 제공합니다.

- AssociateWirelessDeviceWithThing
- AssociateWirelessGatewayWithThing

비용 할당 및 사용 데이터 보기

결제 그룹을 사용하여 비용을 분류 및 추적할 수 있습니다. 결제 그룹(및 포함된 항목에)에 태그를 적용하면는 사용 내역과 비용을 태그별로 집계한 CSV(쉼표로 구분된 값) 파일로 비용 할당 보고서를 AWS 생성합니다. 비즈니스 카테고리를 나타내는 태그(예: 비용 센터, 애플리케이션 이름 또는 소유자)를 적용하여 여러 서비스에 대한 비용을 정리할 수 있습니다. 비용 할당 태그 사용에 대한 자세한 내용은 AWS Billing and Cost Management 사용 설명서의 비용 할당 태그 사용을 참조하세요.

> **Note**
>
> 결제 그룹에 배치한 사물과 사용 및 비용 데이터를 정확하게 연결하려면 각 디바이스 또는 애플리케이션이 다음과 같아야 합니다.
>
> - 사물로 등록해야 합니다 AWS IoT. 자세한 내용은 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하십시오.
> - 사물의 이름만 클라이언트 ID로 사용하여 MQTT를 통해 AWS IoT 메시지 브로커에 연결합니다. 자세한 내용은 the section called “디바이스 통신 프로토콜” 단원을 참조하십시오. 클라이언트 ID가 사물 이름과 일치하지 않는 경우, 독점 사물 연결을 활성화하여 연결을 설정할 수 있습니다. 자세한 내용은 ??? 단원을 참조하십시오.
> - 사물과 연결된 클라이언트 인증서를 사용해 인증해야 합니다.

(결제 그룹과 연관된 작업에 따라) 결제 그룹에는 다음 요금 차원을 사용할 수 있습니다.

- 연결(연결에 클라이언트 ID로 사용된 사물 이름 기준).
- 메시징(사물에서 인바운드되는 메시지 및 사물로 아웃바운드되는 메시지 기준, MQTT만 해당).

- 섀도우 작업(메시지가 섀도우 업데이트를 트리거한 사물 기준).
- 트리거된 규칙(인바운드 메시지로 규칙을 트리거한 사물이 기준이며, MQTT 수명 주기 이벤트에 의해 트리거된 규칙에는 적용되지 않음).
- 사물 인덱스 업데이트(인덱스에 추가된 사물 기준).
- 원격 작업(업데이트된 사물 기준).
- AWS IoT Device Defender detect 보고서(작업이 보고된 사물 기준).

(결제 그룹에 대해 보고되고) 태그를 기반으로 하는 비용 및 사용 데이터에는 다음 작업은 반영되지 않습니다.

- 디바이스 레지스트리 작업(사물, 사물 그룹 및 사물 유형에 대한 업데이트 포함). 자세한 정보는 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하세요.
- 사물 그룹 인덱스 업데이트(사물 그룹을 추가하는 경우).
- 인덱스 검색 쿼리.
- 디바이스 프로비저닝.
- AWS IoT Device Defender audit 보고서입니다.

의 보안 AWS IoT

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이를 클라우드의 보안과 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사원은 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 AWS 규정 준수 프로그램 제공 범위 내 서비스를 AWS IoT참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 AWS IoT. 다음 주제에서는 보안 및 규정 준수 목표를 충족 AWS IoT 하도록를 구성하는 방법을 보여줍니다. 또한 AWS IoT 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

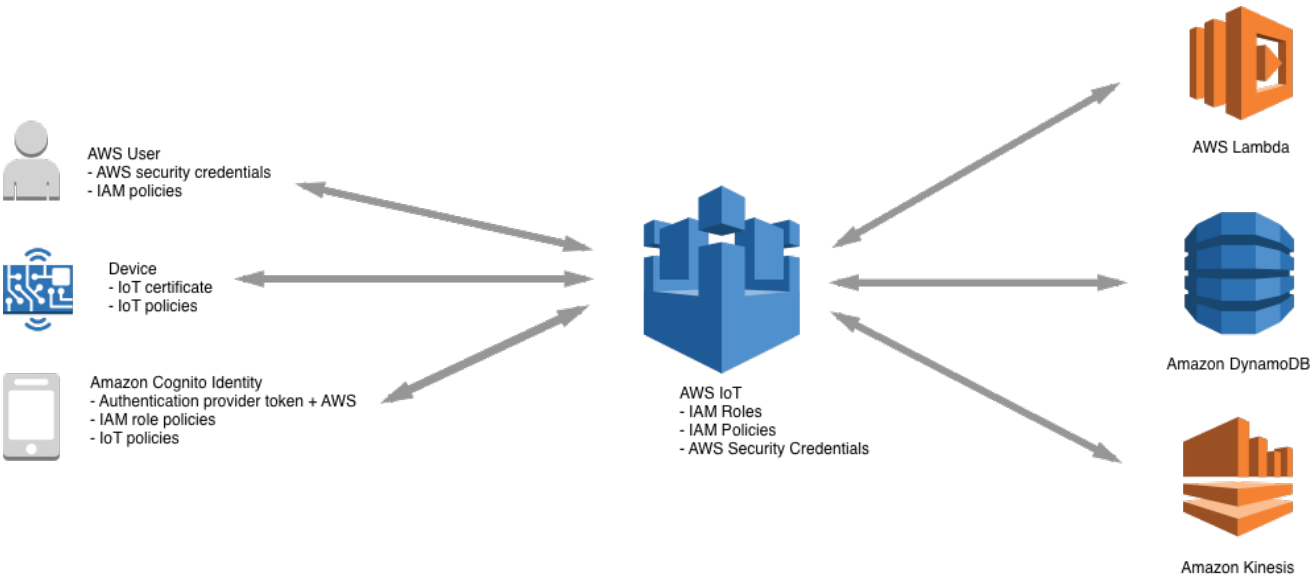
주제

- AWS IoT 보안
- Authentication
- 권한 부여
- 의 데이터 보호 AWS IoT Core
- 에 대한 자격 증명 및 액세스 관리 AWS IoT
- 로깅 및 모니터링
- AWS IoT 코어에 대한 규정 준수 검증
- AWS IoT Core의 복원력
- 인터페이스 VPC 엔드포인트 AWS IoT Core 와 함께 사용
- 의 인프라 보안 AWS IoT
- AWS IoT Core를 사용한 프로덕션 플릿 또는 디바이스의 보안 모니터링
- 의 보안 모범 사례 AWS IoT Core

- AWS 교육 및 인증

AWS IoT 보안

연결된 각 디바이스 또는 클라이언트에는 AWS IoT와 상호 작용할 자격 증명이 있어야 합니다. 송수신되는 모든 트래픽 AWS IoT 은 TLS(전송 계층 보안)를 통해 안전하게 전송됩니다. AWS 클라우드 보안 메커니즘은 AWS IoT 와 다른 AWS 서비스 간에 이동할 때 데이터를 보호합니다.



- AWS IoT에서 디바이스 자격 증명(X.509 인증서, AWS 자격 증명, Amazon Cognito 자격 증명, 연동 자격 증명 또는 사용자 지정 인증 토큰) 및 정책을 관리하는 것은 사용자의 책임입니다. 또한, 고유한 ID를 각 디바이스에 할당하고 디바이스 또는 디바이스 그룹에 대한 권한을 관리하는 것은 사용자의 책임입니다.
- 디바이스는 보안 TLS 연결을 통해 X.509 인증서 또는 Amazon Cognito 자격 증명을 AWS IoT 사용하여에 연결됩니다. 연구 및 개발 중에, API 호출을 하거나 WebSockets를 사용하는 일부 애플리케이션의 경우, IAM 사용자 및 그룹 또는 사용자 지정 인증 토큰을 사용하여 인증할 수도 있습니다. 자세한 내용은 IAM 사용자, 그룹 및 역할 단원을 참조하십시오.
- AWS IoT 인증을 사용할 때 메시지 브로커는 디바이스를 인증하고, 디바이스 데이터를 안전하게 수집하고, AWS IoT 정책을 사용하여 디바이스에 지정한 액세스 권한을 부여하거나 거부할 책임이 있습니다.
- 사용자 지정 인증을 사용하는 경우 사용자 지정 권한 부여자는 디바이스를 인증하고 AWS IoT 또는 IAM 정책을 사용하여 디바이스에 지정한 액세스 권한을 부여하거나 거부할 책임이 있습니다.
- AWS IoT 규칙 엔진은 사용자가 정의한 규칙에 따라 디바이스 데이터를 다른 디바이스 또는 다른 AWS 서비스로 전달합니다. AWS Identity and Access Management 를 사용하여 데이터를 최종 대

상으로 안전하게 전송합니다. 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT 단원을 참조하십시오.

Authentication

인증은 클라이언트 또는 서버의 자격 증명을 확인하는 메커니즘입니다. 서버 인증은 디바이스 또는 다른 클라이언트가 실제 AWS IoT 엔드포인트와 통신하는지 확인하는 프로세스입니다. 클라이언트 인증은 디바이스 또는 다른 클라이언트가 자신을 인증하는 프로세스입니다 AWS IoT.

X.509 인증서 개요

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 하는 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. X.509 인증서 체인은 클라이언트의 서버 인증과 서버의 클라이언트 인증에 모두 사용됩니다.

서버 인증

디바이스 또는 다른 클라이언트 AWS IoT Core가 AWS IoT Core 연결을 시도하면 서버는 디바이스가 서버를 인증하는 데 사용하는 X.509 인증서를 전송합니다. 인증은 X.509 인증서 체인 검증을 통해 TLS 계층에서 이루어집니다. 이는 HTTPS URL을 방문할 때 브라우저에서 사용하는 것과 동일한 방법입니다. 자체 인증 기관의 인증서를 사용하려면 단원을 참조하세요CA 인증서 관리

디바이스 또는 다른 클라이언트가 엔드포인트에 AWS IoT Core TLS 연결을 설정하면는 디바이스가 다른 서버가 가장하는 것이 AWS IoT Core 아니라와 통신하고 있는지 확인하는 데 사용하는 인증서 체인을 AWS IoT Core 나타냅니다 AWS IoT Core. 표시되는 체인은 디바이스가 연결하는 엔드포인트 유형과 클라이언트가 TLS 핸드셰이크 중에 AWS IoT Core 협상한 암호 제품군의 조합에 따라 달라집니다.

엔드포인트 유형

AWS IoT Core 는를 지원합니다`iot:Data-ATS`. `iot:Data-ATS` 엔드포인트는 Amazon Trust Services CA에서 서명한 서버 인증서를 제공합니다.

ATS 엔드포인트에서 제공하는 인증서는 Starfield에서 교차 서명합니다. 일부 TLS 클라이언트 구현에는 신뢰 루트의 검증이 필요하며 클라이언트의 신뢰할 수 있는 스토어에 Starfield CA 인증서가 설치되어 있어야 합니다.

> **Warning**
>
> 전체 인증서(발급자 이름 등 포함)를 해시하는 인증서 고정 방법을 사용하는 것은 권장되지 않습니다. 이렇게 하면 AWS에서 제공하는 ATS 인증서가 Starfield에 의해 교차 서명되고 발급자 이름이 다르기 때문에 인증서 확인에 실패할 수 있습니다.

> **Important**
>
> `iot:Data-ATS` 엔드포인트를 사용합니다. Symantec 및 Verisign 인증서는 더 이상 사용되지 않으며에서 더 이상 지원되지 않습니다 AWS IoT Core.

`describe-endpoint` 명령을 사용하여 ATS 엔드포인트를 생성할 수 있습니다.

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

`describe-endpoint` 명령은 다음 형식으로 엔드포인트를 반환합니다.

```
account-specific-prefix.iot.your-region.amazonaws.com
```

> **Note**
>
> `describe-endpoint`를 처음 호출하면 엔드포인트가 생성됩니다. 이후 `describe-endpoint`에 대한 모든 호출은 동일한 엔드포인트를 반환합니다.

> **Note**
>
> AWS IoT Core 콘솔에서 `iot:Data-ATS` 엔드포인트를 보려면 설정을 선택합니다. 콘솔에는 `iot:Data-ATS` 엔드포인트만 표시됩니다.

AWS SDK for Java**`IotDataPlaneClient`**를 사용하여 생성

`iot:Data-ATS` 엔드포인트를 사용하는 `IotDataPlaneClient`를 생성하려면 다음을 수행해야 합니다.

- DescribeEndpoint API를 사용하여 `iot:Data-ATS` 엔드포인트를 생성합니다.
- `IotDataPlaneClient`를 생성할 때 해당 엔드포인트를 지정합니다.

다음 예제에서는 이 두 작업을 모두 수행합니다.

```
public void setup() throws Exception {
        IotClient client =
 IotClient.builder().credentialsProvider(CREDENTIALS_PROVIDER_CHAIN).region(Region.US_EAST_1).b
        String endpoint = client.describeEndpoint(r -> r.endpointType("iot:Data-
ATS")).endpointAddress();
        iot = IotDataPlaneClient.builder()
                                .credentialsProvider(CREDENTIALS_PROVIDER_CHAIN)
                                .endpointOverride(URI.create("https://" + endpoint))
                                .region(Region.US_EAST_1)
                                .build();
}
```

서버 인증을 위한 CA 인증서

사용 중인 데이터 엔드포인트 유형과 협상한 암호 제품군에 따라 AWS IoT Core 서버 인증 인증서는 다음 루트 CA 인증서 중 하나로 서명됩니다.

Amazon Trust Services 엔드포인트(기본 설정됨)

> **Note**
>
> 이 링크를 마우스 오른쪽 버튼으로 클릭하고 다른 이름으로 링크 저장을 선택하여 이러한 인증서를 파일로 저장해야 할 수 있습니다.

- RSA 2048비트 키: Amazon Root CA 1.
- RSA 4096비트 키: Amazon Root CA 2. 추후 사용 예약.
- ECC 256비트 키: Amazon Root CA 3.
- ECC 384비트 키: Amazon Root CA 4. 추후 사용 예약.

이러한 인증서는 모두 Starfield 루트 CA 인증서에서 교차 서명됩니다. 아시아 태평양(뭄바이) AWS IoT Core 리전 AWS IoT Core 에서 2018년 5월 9일 출시부터 모든 신규 리전은 ATS 인증서만 제공합니다.

VeriSign 엔드포인트(레거시)

- RSA 2048비트 키: VeriSign Class 3 Public Primary G5 루트 CA 인증서

서버 인증 지침

디바이스에서 AWS IoT Core 서버 인증 인증서를 검증하는 기능에 영향을 미칠 수 있는 많은 변수가 있습니다. 예를 들어 디바이스의 메모리가 너무 한정적이어서 가능한 모든 루트 CA 인증서를 보유할 수 없거나, 디바이스가 비표준 인증서 검증 방법을 구현할 수 있습니다. 이러한 이유로 다음 지침을 따르는 것이 좋습니다.

- ATS 엔드포인트를 사용하고 지원되는 모든 Amazon Root CA 인증서를 설치하는 것이 좋습니다.
- 디바이스에 이러한 인증서를 모두 저장할 수 없고 디바이스에서 ECC 기반 검증을 사용하지 않는 경우 Amazon Root CA 3 및 Amazon Root CA 4 ECC 인증서를 생략할 수 있습니다. 디바이스가 RSA 기반 인증서 검증을 구현하지 않는 경우 Amazon Root CA 1 및 Amazon Root CA 2 RSA 인증서를 생략할 수 있습니다. 이 링크를 마우스 오른쪽 버튼으로 클릭하고 다른 이름으로 링크 저장을 선택하여 이러한 인증서를 파일로 저장해야 할 수 있습니다.
- ATS 엔드포인트에 연결할 때 서버 인증서 검증 문제가 발생하는 경우 관련 교차 서명된 Amazon Root CA 인증서를 신뢰할 수 있는 스토어에 추가해 보세요. 이 링크를 마우스 오른쪽 버튼으로 클릭하고 다른 이름으로 링크 저장을 선택하여 이러한 인증서를 파일로 저장해야 할 수 있습니다.
  - 교차 서명 Amazon Root CA 1
  - 교차 서명된 Amazon Root CA 2 - 향후 사용을 위해 예약됩니다.
  - 교차 서명 Amazon Root CA 3
  - 교차 서명된 Amazon Root CA 4 - 향후 사용을 위해 예약됩니다.
- 서버 인증서 검증 문제가 발생하는 경우 디바이스가 루트 CA를 명시적으로 신뢰해야 할 수 있습니다. Starfield Root CA Certificate를 트러스트 스토어에 추가하세요.
- 위의 단계를 실행한 후에도 문제가 계속 발생하면 AWS 개발자 지원 센터에 문의하세요.

> **Note**
>
> CA 인증서에는 서버 인증서를 검증하는 데 사용할 수 없는 만료 날짜가 있습니다. 따라서 만료 날짜가 도래하기 전에 CA 인증서를 교체해야 하는 경우도 있습니다. 지속적 연결을 보장하고 최신 보안 모범 사례를 유지하려면 모든 디바이스 또는 클라이언트에서 루트 CA 인증서를 업데이트해야 합니다.

> **Note**
>
> 디바이스 코드 AWS IoT Core 에서에 연결할 때 연결에 사용하는 API에 인증서를 전달합니다. 사용하는 API는 SDK에 따라 다릅니다. 자세한 내용은 AWS IoT Core 디바이스 SDK를 참조하세요.

클라이언트 인증

AWS IoT 는 디바이스 또는 클라이언트 인증을 위해 세 가지 유형의 자격 증명 보안 주체를 지원합니다.

- X.509 클라이언트 인증서
- IAM 사용자, 그룹 및 역할
- Amazon Cognito 자격 증명

이러한 자격 증명은 디바이스, 모바일, 웹 또는 데스크톱 애플리케이션에서 사용할 수 있습니다. 명령줄 인터페이스(CLI) AWS IoT 명령을 입력하는 사용자도 사용할 수 있습니다. 일반적으로 AWS IoT 디바이스는 X.509 인증서를 사용하는 반면, 모바일 애플리케이션은 Amazon Cognito 자격 증명을 사용합니다. 웹 및 데스크톱 애플리케이션은 IAM 또는 연동 자격 증명을 사용합니다. AWS CLI 명령은 IAM을 사용합니다. IAM 자격 증명에 대한 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT 단원을 참조하세요.

X.509 클라이언트 인증서

X.509 인증서 AWS IoT 는 클라이언트 및 디바이스 연결을 인증하는 기능을 제공합니다. 클라이언트가 통신하려면 AWS IoT 먼저 클라이언트 인증서를에 등록해야 합니다 AWS IoT. 클라이언트 인증서는 동일한 리전의 간에 디바이스를 쉽게 이동할 수 AWS 리전 있도록 동일한 AWS 계정의 여러 AWS 계정에 등록할 수 있습니다. 자세한 내용은 다중 계정 등록을 통해 여러 AWS 계정에서 X.509 클라이언트 인증서 사용를 참조하세요.

인증서 취소를 포함하여 세부적인 클라이언트 관리 작업이 가능하도록 각 디바이스 또는 클라이언트에 고유한 인증서를 부여하는 것이 좋습니다. 인증서 만료 시 원활한 작동을 보장하기 위해 디바이스 및 클라이언트가 인증서 교체 및 대체를 지원해야 합니다.

X.509 인증서를 사용하여 몇 개 이상의 디바이스를 지원하는 방법에 대한 자세한 내용은 디바이스 프로비저닝 단원을 참조하여 AWS IoT 이(가) 지원하는 다양한 인증서 관리 및 프로비저닝 옵션을 검토하세요.

AWS IoT 는 다음과 같은 유형의 X.509 클라이언트 인증서를 지원합니다.

- 에서 생성된 X.509 인증서 AWS IoT
- 에 등록된 CA가 서명한 X.509 인증서 AWS IoT.
- AWS IoT에 등록되지 않은 CA에서 서명한 X.509 인증서입니다.

이 섹션에서는 AWS IoT에서 X.509 인증서를 관리하는 방법에 대해 설명합니다. AWS IoT 콘솔 또는 AWS CLI 를 사용하여 다음 인증서 작업을 수행할 수 있습니다.

- AWS IoT 클라이언트 인증서 생성
- 자체 클라이언트 인증서 생성
- 클라이언트 인증서 등록
- 클라이언트 인증서 활성화 또는 비활성화
- 클라이언트 인증서 취소

이러한 작업을 수행하는 AWS CLI 명령에 대한 자세한 내용은 AWS IoT CLI 참조를 참조하세요.

X.509 클라이언트 인증서 사용

X.509 인증서는 클라이언트 및 디바이스 연결을 인증합니다 AWS IoT. X.509 인증서는 식별 및 인증 메커니즘보다 유용한 여러 가지 장점을 제공합니다. X.509 인증서가 있으면 디바이스에서 비대칭 키를 사용할 수 있습니다. 예를 들어 중요한 암호화 구성 요소가 디바이스를 벗어나지 않도록 프라이빗 키를 디바이스의 안전한 스토리지에 저장할 수 있습니다. 프라이빗 키가 디바이스를 전혀 벗어나지 않으므로 X.509 인증서는 사용자 이름 및 암호 또는 보유자 토큰과 같은 다른 체계보다 강력한 클라이언트 인증을 제공합니다.

AWS IoT 는 TLS 프로토콜의 클라이언트 인증 모드를 사용하여 클라이언트 인증서를 인증합니다. TLS 지원은 다수의 프로그래밍 언어 및 운영 체제에서 제공되며 데이터 암호화에 흔히 사용됩니다. TLS 클라이언트 인증에서는 X.509 클라이언트 인증서를 AWS IoT 요청하고 인증서의 레지스트리에 AWS 계정 대해 인증서의 상태를 검증합니다. 그런 다음 인증서에 포함된 퍼블릭 키에 해당하는 프라이빗 키의 소유권 증명을 위해 클라이언트에 문제를 제기합니다. AWS IoT 는 클라이언트가 전송 계층 보안(TLS) 프로토콜에 SNI(Server Name Indication) 확장을 전송하도록 요구합니다. SNI 확장 구성에 대한 자세한 내용은 단원을 참조하세요의 전송 보안 AWS IoT Core

AWS IoT 코어에 대한 안전하고 일관된 클라이언트 연결을 용이하게 하려면 X.509 클라이언트 인증서에 다음이 포함되어야 합니다.

- AWS IoT 코어에 등록됨. 자세한 내용은 클라이언트 인증서 등록 단원을 참조하십시오.
- ACTIVE 상태여야 합니다. 자세한 내용은 클라이언트 인증서 활성화 또는 비활성화 단원을 참조하십시오.
- 아직 인증서 만료 날짜가 되지 않았어야 합니다.

Amazon Root CA를 사용하는 클라이언트 인증서를 생성하고 다른 인증 기관(CA)에서 서명한 자체 클라이언트 인증서를 사용할 수 있습니다. AWS IoT 콘솔을 사용하여 Amazon Root CA를 사용하는 인증서를 생성하는 방법에 대한 자세한 내용은 섹션을 참조하세요AWS IoT 클라이언트 인증서 생성. 자체 X.509 인증서 사용에 대한 자세한 내용은 자체 클라이언트 인증서 생성 단원을 참조하세요.

CA 인증서에서 서명한 인증서의 만료 날짜 및 시간은 인증서 생성 시 설정됩니다. 에서 생성된 X.509 인증서는 2049년 12월 31일 자정 UTC(2049-12-31T23:59:59Z)에 AWS IoT 만료됩니다.

AWS IoT Device Defender 는 일반적인 IoT 보안 모범 사례를 지원하는 AWS 계정 및 디바이스에 대한 감사를 수행할 수 있습니다. 여기에는 CA 또는 Amazon Root CA에서 서명한 X.509 인증서의 만료 날짜 관리가 포함됩니다. 인증서의 만료 날짜 관리에 대한 자세한 내용은 장치 인증서 만료 및 CA 인증서 만료를 참조하세요.

공식 AWS IoT 블로그에서는를 사용하여 IoT 디바이스 인증서 교체를 관리하는 방법에서 디바이스 인증서 교체 AWS IoT 및 보안 모범 사례 관리에 대해 자세히 살펴봅니다.

다중 계정 등록을 통해 여러 AWS 계정에서 X.509 클라이언트 인증서 사용

다중 계정 등록을 통해 동일한 리전 및 다른 리전의 AWS 계정사이에 디바이스를 이동할 수 있습니다. 프리 프로덕션 계정에 디바이스를 등록, 테스트 및 구성한 다음 프로덕션 계정에 동일한 디바이스 및 디바이스 인증서를 등록하고 사용할 수 있습니다. 디바이스에 클라이언트 인증서를 등록하거나 등록된 CA 없이 디바이스 인증서를 등록할 수도 있습니다 AWS IoT. 자세한 정보는 등록되지 않은 CA로 서명된 클라이언트 인증서 등록(CLI)을 참조하세요.

> **Note**
>
> 다중 계정 등록에 사용되는 인증서는 `iot:Data-ATS`, `iot:Data`(레거시), `iot:Jobs` 및 `iot:CredentialProvider` 엔드포인트 유형으로 지원됩니다. AWS IoT 디바이스 엔드포인트에 대한 자세한 내용은 섹션을 참조하세요AWS IoT 디바이스 데이터 및 서비스 엔드포인트.

다중 계정 등록을 사용하는 디바이스는 전송 계층 보안(TLS) 프로토콜로 SNI(Server Name Indication) 확장을 전송하고에 연결할 때 `host_name` 필드에 전체 엔드포인트 주소를 제공해야 합니다 AWS IoT. 는의 엔드포인트 주소를 AWS IoT 사용하여 연결을 올바른 AWS IoT 계정으로 `host_name` 라우팅합

니다. host_name에서 유효한 엔드포인트 주소를 전송하지 않는 기존 디바이스도 계속 작동하기는 하지만 이 정보가 필요한 기능은 사용할 수 없습니다. SNI 확장에 대한 자세한 내용과 host_name 필드의 엔드포인트 주소를 식별하는 방법은 의 전송 보안 AWS IoT Core 단원을 참조하세요.

다중 계정 등록을 사용하려면

1. CA로 디바이스 인증서를 등록할 수 있습니다. SNI_ONLY 모드에서 여러 계정에 서명 CA를 등록하고 해당 CA를 사용하여 여러 계정에 동일한 클라이언트 인증서를 등록할 수 있습니다. 자세한 내용은 SNI_ONLY 모드에서 CA 인증서 등록(CLI) - 권장됨 단원을 참조하십시오.
2. CA 없이 디바이스 인증서를 등록할 수 있습니다. 등록되지 않은 CA로 서명된 클라이언트 인증서 등록(CLI)을(를) 참조하세요. CA 등록은 선택 사항입니다. 디바이스 인증서에 서명한 CA를 등록할 필요는 없습니다 AWS IoT.

에서 지원하는 인증서 서명 알고리즘 AWS IoT

AWS IoT 는 다음과 같은 인증서 서명 알고리즘을 지원합니다.

- SHA256WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- SHA256WITHRSAANDMGF1(RSASSA-PSS)
- SHA384WITHRSAANDMGF1(RSASSA-PSS)
- SHA512WITHRSAANDMGF1(RSASSA-PSS)
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256
- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

인증서 인증 및 보안에 대한 자세한 내용은 디바이스 인증서 키 품질 섹션을 참조하세요.

> **Note**
>
> 인증서 서명 요청(CSR)에는 퍼블릭 키가 포함되어야 합니다. 키는 길이가 2,048비트 이상인 RSA 키이거나 NIST P-256, NIST P-384, NIST P-521 곡선의 ECC 키일 수 있습니다. 자세한 내용은AWS IoT API 참조 가이드의 CreateCertificateFromCsr을 참조하세요.

에서 지원하는 키 알고리즘 AWS IoT

아래 표는 키 알고리즘이 지원되는 방법을 보여줍니다.

주요 알고리즘	인증서 서명 알고리즘	TLS 버전	지원 여부 [Yes] 또는 [No]
키 크기가 2,048비트 이상인 RSA	모두	TLS 1.2 TLS 1.3	예
ECC NIST P-256/P-384/P-521	모두	TLS 1.2 TLS 1.3	예
키 크기가 2,048비트 이상인 RSA-PSS	모두	TLS 1.2	아니요
키 크기가 2,048비트 이상인 RSA-PSS	모두	TLS 1.3	예

CreateCertificateFromCSR을 사용하여 인증서를 생성하려면 지원되는 키 알고리즘을 사용하여 CSR에 대한 퍼블릭 키를 생성할 수 있습니다. RegisterCertificate 또는 RegisterCertificateWithoutCA를 사용하여 자체 인증서를 등록하려면 지원되는 키 알고리즘을 사용하여 인증서에 대한 퍼블릭 키를 생성할 수 있습니다.

자세한 내용은 보안 정책을 참조하세요.

AWS IoT 클라이언트 인증서 생성

AWS IoT 는 Amazon Root 인증 기관(CA)에서 서명한 클라이언트 인증서를 제공합니다.

이 주제에서는 Amazon Root 인증 기관에서 서명한 클라이언트 인증서를 생성하고 인증서 파일을 다운로드하는 방법에 대해 설명합니다. 클라이언트 인증서 파일을 생성한 후에는 클라이언트에 설치해야 합니다.

> **Note**
>
> 에서 제공하는 각 X.509 클라이언트 인증서 AWS IoT 에는 인증서 생성 시 설정한 발급자 및 주체 속성이 있습니다. 인증서 속성은 인증서를 만든 후에만 변경할 수 없습니다.

AWS IoT 콘솔 또는를 사용하여 Amazon Root AWS IoT 인증 기관에서 서명한 인증서를 AWS CLI 생성할 수 있습니다.

AWS IoT 인증서 생성(콘솔)

AWS IoT 콘솔을 사용하여 AWS IoT 인증서를 생성하려면

1. 에 로그인 AWS Management Console 하고 AWS IoT 콘솔을 엽니다.
2. 탐색 창에서 보안, 인증서, 생성을 차례대로 선택합니다.
3. 원클릭 인증서 생성(권장)과 인증서 생성을 선택합니다.
4. 인증서 생성 완료 페이지에서 사물, 퍼블릭 키 및 프라이빗 키에 대한 클라이언트 인증서 파일을 안전한 위치로 다운로드합니다. 에서 생성된 이러한 인증서 AWS IoT 는 AWS IoT 서비스에서만 사용할 수 있습니다.

   Amazon Root CA 인증서 파일도 필요한 경우 이 페이지에는 다운로드할 수 있는 페이지로 연결되는 링크도 있습니다.
5. 이제 클라이언트 인증서가 생성되고 AWS IoT에 등록되었습니다. 클라이언트에서 인증서를 사용하기 전에 인증서를 활성화해야 합니다.

   지금 클라이언트 인증서를 활성화하려면 활성화를 선택합니다. 지금 인증서를 지금 활성화하지 않으려면 클라이언트 인증서 활성화(콘솔)에서 나중에 인증서를 활성화하는 방법을 알아보세요.
6. 인증서에 정책을 연결하려면 정책 연결을 선택합니다.

   지금 정책을 연결하지 않으려면 완료를 선택하여 완료합니다. 나중에 정책을 연결할 수 있습니다.

절차를 완료 한 후 클라이언트에 인증서 파일을 설치합니다.

AWS IoT 인증서 생성(CLI)

는 Amazon Root 인증 기관에서 서명한 클라이언트 인증서를 생성하는 create-keys-and-certificate 명령을 AWS CLI 제공합니다. 그러나 이 명령은 Amazon Root CA 인증서 파일을 다운로드하지 않습니다. 서버 인증을 위한 CA 인증서에서 Amazon Root CA 인증서 파일을 다운로드할 수 있습니다.

이 명령은 프라이빗 키, 퍼블릭 키 및 X.509 인증서 파일을 생성하고 인증서를 등록하고 활성화합니다 AWS IoT.

```
aws iot create-keys-and-certificate \
    --set-as-active \
```

```
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

인증서를 생성하고 등록할 때 인증서를 활성화하지 않으려면 이 명령은 프라이빗 키, 퍼블릭 키 및 X.509 인증서 파일을 생성하고 인증서를 등록하지만 인증서를 활성화하지는 않습니다. 클라이언트 인증서 활성화(CLI)에서는 나중에 인증서를 활성화하는 방법에 대해 설명합니다.

```
aws iot create-keys-and-certificate \
    --no-set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

클라이언트에 인증서 파일을 설치합니다.

자체 클라이언트 인증서 생성

AWS IoT 는 루트 또는 중간 인증 기관(CA)에서 서명한 클라이언트 인증서를 지원합니다.는 CA 인증서를 AWS IoT 사용하여 인증서의 소유권을 확인합니다. Amazon의 CA가 아닌 CA에서 서명한 디바이스 인증서를 사용하려면 디바이스 인증서의 소유권을 확인할 수 AWS IoT 있도록 CA의 인증서를에 등록해야 합니다.

AWS IoT 는 자체 인증서를 가져오는 여러 가지 방법(BYOC)을 지원합니다.

- 먼저 클라이언트 인증서 서명에 사용되는 CA를 등록한 다음 개별 클라이언트 인증서를 등록합니다. 디바이스 또는 클라이언트가 처음 연결될 때 클라이언트 인증서에 등록하려면 AWS IoT (Just-in-Time Provisioning이라고도 함) 서명 CA를에 AWS IoT 등록하고 자동 등록을 활성화해야 합니다.
- 서명 CA를 등록할 수 없는 경우 CA 없이 클라이언트 인증서를 등록하도록 선택할 수 있습니다. CA 없이 등록된 디바이스의 경우 AWS IoT에 연결할 때 서버 이름 표시(SNI)를 제시해야 합니다.

> **Note**
>
> CA를 사용하여 클라이언트 인증서를 등록하려면 계층 구조의 다른 CA가 AWS IoT아닌에 서명 CAs를 등록해야 합니다.

> **Note**
>
> CA 인증서는 한 리전의 한 계정에서만 DEFAULT 모드로 등록할 수 있습니다. CA 인증서는 한 리전의 여러 계정에서 SNI_ONLY 모드로 등록할 수 있습니다.

X.509 인증서를 사용하여 몇 개 이상의 디바이스를 지원하는 방법에 대한 자세한 내용은 디바이스 프로비저닝 단원을 참조하여 AWS IoT 이(가) 지원하는 다양한 인증서 관리 및 프로비저닝 옵션을 검토하세요.

주제

- CA 인증서 관리
- CA 인증서를 사용하여 클라이언트 인증서 생성

CA 인증서 관리

이 섹션에서는 자체 인증 기관(CA) 인증서를 관리하기 위한 일반적인 작업에 대해 설명합니다.

인식하지 못하는 CA에서 서명한 클라이언트 인증서를 사용하는 AWS IoT 경우 인증 기관(CA)을에 등록할 수 AWS IoT 있습니다.

클라이언트가 처음 연결할 AWS IoT 때 클라이언트 인증서를에 자동으로 등록하도록 하려면 클라이언트 인증서에 서명한 CA를에 등록해야 합니다 AWS IoT. 그렇지 않으면 클라이언트 인증서를 서명한 CA 인증서를 등록할 필요가 없습니다.

> **Note**
>
> CA 인증서는 한 리전의 한 계정에서만 DEFAULT 모드로 등록할 수 있습니다. CA 인증서는 한 리전의 여러 계정에서 SNI_ONLY 모드로 등록할 수 있습니다.

주제:

- CA 인증서 생성
- CA 인증서 등록
- CA 인증서 비활성화

CA 인증서 생성

CA 인증서가 없을 경우 OpenSSL v1.1.1i 도구를 사용하여 인증서를 생성할 수 있습니다.

> **Note**
>
> AWS IoT 콘솔에서는이 절차를 수행할 수 없습니다.

OpenSSL v1.1.1i 도구를 사용하여 CA 인증서를 생성하려면

1. 키 페어를 생성합니다.

```
openssl genrsa -out root_CA_key_filename.key 2048
```

2. 키 페어의 프라이빗 키를 사용하여 CA 인증서를 생성합니다.

```
openssl req -x509 -new -nodes \
    -key root_CA_key_filename.key \
    -sha256 -days 1024 \
    -out root_CA_cert_filename.pem
```

CA 인증서 등록

이 절차에서는 Amazon의 CA가 아닌 인증 기관(CA)의 인증서를 등록하는 방법을 설명합니다. CA 인증서를 AWS IoT Core 사용하여 인증서 소유권을 확인합니다. Amazon의 CA가 아닌 CA에서 서명한 디바이스 인증서를 사용하려면 디바이스 인증서의 소유권을 확인할 수 AWS IoT Core 있도록 CA 인증서를에 등록해야 합니다.

CA 인증서 등록(콘솔)

> **Note**
>
> 콘솔에서 CA 인증서를 등록하려면 콘솔에서 CA 인증서 등록(Register CA certificate)을 시작합니다. 확인 인증서를 제공하거나 프라이빗 키에 액세스할 필요 없이 다중 계정 모드에서 CA를 등록할 수 있습니다. 다중 계정 모드에서 같은 AWS 리전의 여러 AWS 계정 으로 CA를 등록할 수 있습니다. CA 프라이빗 키의 확인 인증서와 소유권 증명을 제공하여 단일 계정 모드에서 CA를 등록할 수 있습니다.

CA 인증서 등록(CLI)

CA 인증서를 DEFAULT 모드 또는 SNI_ONLY 모드로 등록할 수 있습니다. CA는 한 AWS 계정 에 하나씩 DEFAULT 모드에 등록할 수 있습니다 AWS 리전. CA는 동일한에서 여러에 의해 SNI_ONLY 모드 AWS 계정 에 등록될 수 있습니다 AWS 리전. CA 인증서에 대한 자세한 정보는 certificateMode를 참조하세요.

> **Note**
>
> SNI_ONLY 모드에서 CA를 등록하는 것이 좋습니다. 확인 인증서나 프라이빗 키에 대한 액세스 권한을 제공할 필요가 없으며 동일한 AWS 계정 에서 여러 로 CA를 등록할 수 있습니다 AWS 리전.

SNI_ONLY 모드에서 CA 인증서 등록(CLI) - 권장됨

사전 조건

계속하기 전에 컴퓨터에서 다음 항목을 사용할 수 있는지 확인합니다.

- 루트 CA의 인증서 파일(다음 예시에서 *root_CA_cert_filename.pem*으로 언급됨)
- OpenSSL v1.1.1i 이상

를 사용하여 **SNI_ONLY** 모드에서 CA 인증서를 등록하려면 AWS CLI

1. CA 인증서를에 등록합니다 AWS IoT. register-ca-certificate 명령을 사용하여 CA 인증서 파일 이름을 입력합니다. 자세한 정보는 AWS CLI 명령 참조의 egister-ca-certificate를 참조하세요.

   ```
   aws iot register-ca-certificate \
       --ca-certificate file://root_CA_cert_filename.pem \
       --certificate-mode SNI_ONLY
   ```

   성공한 경우 이 명령은 *certificateId*를 반환합니다.

2. 이 시점에서 CA 인증서는에 등록 AWS IoT 되었지만 비활성 상태입니다. CA 인증서에서 서명한 클라이언트 인증서를 등록하려면 먼저 CA 인증서가 활성 상태여야 합니다.

   이 단계에서는 CA 인증서를 활성화합니다.

다음과 같은 update-certificate CLI 명령을 사용하여 CA 인증서를 활성화합니다. 자세한 정보는 AWS CLI 명령 참조의 update-certificate를 참조하세요.

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

describe-ca-certificate 명령을 사용하여 CA 인증서의 상태를 확인합니다. 자세한 정보는 AWS CLI 명령 참조의 describe-ca-certificate를 참조하세요.

DEFAULT 모드에서 CA 인증서 등록(CLI)

사전 조건

계속하기 전에 컴퓨터에서 다음 항목을 사용할 수 있는지 확인합니다.

- 루트 CA의 인증서 파일(다음 예시에서 *root_CA_cert_filename.pem*으로 언급됨)
- 루트 CA 인증서의 프라이빗 키 파일(다음 예시에서 *root_CA_key_filename.key*로 언급됨)
- OpenSSL v1.1.1i 이상

를 사용하여 **DEFAULT** 모드에서 CA 인증서를 등록하려면 AWS CLI

1. 에서 등록 코드를 가져오려면를 AWS IoT사용합니다get-registration-code. 반환된 `registrationCode`를 저장하여 프라이빗 키 확인 인증서의 `Common Name`으로 사용합니다. 자세한 내용은 AWS CLI 명령 참조 안내서의 get-registration-code를 참조하세요.

   ```
   aws iot get-registration-code
   ```

2. 프라이빗 키 확인 인증서의 키 페어를 생성합니다.

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

3. 프라이빗 키 확인 인증서에 대한 인증서 서명 요청(CSR)을 생성합니다. 인증서의 `Common Name` 필드를 get-registration-code에서 반환한 `registrationCode`로 설정합니다.

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

인증서의 Common Name을 비롯해 일부 정보를 입력하라는 메시지가 표시됩니다.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

4. CSR을 사용하여 프라이빗 키 확인 인증서를 생성합니다.

```
openssl x509 -req \
    -in verification_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out verification_cert_filename.pem \
    -days 500 -sha256
```

5. CA 인증서를에 등록합니다 AWS IoT. 다음과 같이 CA 인증서 파일 이름과 프라이빗 키 확인 인증서 파일 이름을 register-ca-certificate 명령에 전달합니다. 자세한 정보는 AWS CLI 명령 참조의 egister-ca-certificate를 참조하세요.

```
aws iot register-ca-certificate \
    --ca-certificate file://root_CA_cert_filename.pem \
    --verification-cert file://verification_cert_filename.pem
```

성공한 경우 이 명령은 *### ID*를 반환합니다.

6. 현재 CA 인증서는에 등록 AWS IoT 되었지만 활성 상태가 아닙니다. 서명한 클라이언트 인증서를 등록하려면 CA 인증서가 활성 상태여야 합니다.

   이 단계에서는 CA 인증서를 활성화합니다.

   다음과 같은 update-certificate CLI 명령을 사용하여 CA 인증서를 활성화합니다. 자세한 정보는 AWS CLI 명령 참조의 update-certificate를 참조하세요.

   ```
   aws iot update-ca-certificate \
       --certificate-id certificateId \
       --new-status ACTIVE
   ```

describe-ca-certificate 명령을 사용하여 CA 인증서의 상태를 확인합니다. 자세한 정보는 AWS CLI 명령 참조의 describe-ca-certificate를 참조하세요.

CA 검증 인증서를 생성하여 콘솔에서 CA 인증서를 등록합니다.

> **Note**
>
> 이 절차는 AWS IoT 콘솔에서 CA 인증서를 등록하는 경우에만 사용할 수 있습니다.
> AWS IoT 콘솔에서이 절차를 수행하지 않은 경우 콘솔의 CA 인증서 등록에서 CA 인증서 등록 프로세스를 시작합니다.

계속하기 전에 동일한 컴퓨터에서 다음 항목을 사용할 수 있는지 확인합니다.

- 루트 CA의 인증서 파일(다음 예시에서 *root_CA_cert_filename.pem*으로 언급됨)
- 루트 CA 인증서의 프라이빗 키 파일(다음 예시에서 *root_CA_key_filename.key*로 언급됨)
- OpenSSL v1.1.1i 이상

명령줄 인터페이스를 사용하여 CA 확인 인증서를 생성하여 콘솔에서 CA 인증서를 등록하는 방법

1. 생성하려는 확인 인증서 키 파일의 이름으로 *verification_cert_key_filename.key*를 바꿉니다(예: **verification_cert.key**). 그런 다음, 이 명령을 실행하여 프라이빗 키 확인 인증서의 키 페어를 생성합니다.

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

2. *verification_cert_key_filename.key*를 1단계에서 생성한 키 파일의 이름으로 바꿉니다.

   *verification_cert_csr_filename.csr*을 생성하려는 인증서 서명 요청(CSR) 파일의 이름으로 바꿉니다. 예: **verification_cert.csr**.

   이 명령을 실행하여 CSR 파일을 생성합니다.

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   이 명령은 나중에 설명할 추가 정보를 묻는 메시지를 표시합니다.

3. AWS IoT 콘솔의 확인 인증서 컨테이너에서 등록 코드를 복사합니다.

4. openssl 명령에서 묻는 정보가 다음 예제에 나와 있습니다. Common Name 필드를 제외하고는 원하는 값을 입력하거나 비워 둘 수 있습니다.

   Common Name 필드에 이전 단계에서 복사한 등록 코드를 붙여 넣습니다.

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:
       State or Province Name (full name) []:
       Locality Name (for example, city) []:
       Organization Name (for example, company) []:
       Organizational Unit Name (for example, section) []:
       Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
       Email Address []:

       Please enter the following 'extra' attributes
       to be sent with your certificate request
       A challenge password []:
       An optional company name []:
   ```

   작업을 마치면 명령이 CSR 파일을 생성합니다.

5. *verification_cert_csr_filename.csr*을 이전 단계에서 사용한 *verification_cert_csr_filename.csr*로 바꿉니다.

   *root_CA_cert_filename.pem*을 등록하려는 CA 인증서의 파일 이름으로 바꿉니다.

   *root_CA_key_filename.key*를 CA 인증서 프라이빗 키 파일의 파일 이름으로 바꿉니다.

   *verification_cert_filename.pem*을 생성하려는 확인 인증서의 파일 이름으로 바꿉니다. 예를 들어 **verification_cert.pem**입니다.

   ```
   openssl x509 -req \
       -in verification_cert_csr_filename.csr \
       -CA root_CA_cert_filename.pem \
       -CAkey root_CA_key_filename.key \
       -CAcreateserial \
       -out verification_cert_filename.pem \
       -days 500 -sha256
   ```

6. OpenSSL 명령이 완료되면 콘솔로 돌아갈 때 이러한 파일을 사용할 준비가 되어 있을 것입니다.

   - CA 인증서 파일(이전 명령에서 사용된 *root_CA_cert_filename.pem*)
   - 이전 단계에서 생성한 확인 인증서(이전 명령에서 사용된 *verification_cert_filename.pem*)

CA 인증서 비활성화

자동 클라이언트 인증서 등록을 위해 CA(인증 기관) 인증서가 활성화되면는 CA 인증서를 AWS IoT 확인하여 CA가 인지 확인합니다ACTIVE. CA 인증서가 인 경우 클라이언트 인증서 등록을 허용하지 INACTIVE AWS IoT 않습니다.

CA 인증서를 INACTIVE로 설정하면 CA에서 발급한 새 클라이언트 인증서가 자동으로 등록되지 않습니다.

> **Note**
>
> 훼손된 CA 인증서에서 서명한 모든 등록된 클라이언트 인증서는 사용자가 하나씩 명시적으로 취소할 때까지 계속 작동합니다.

CA 인증서 비활성화(콘솔)

AWS IoT 콘솔을 사용하여 CA 인증서를 비활성화하려면

1. 에 로그인 AWS Management Console 하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안을 선택한 후 CA를 선택합니다.
3. 인증 기관 목록에서 비활성화할 인증 기관을 찾은 다음 줄임표 아이콘을 사용하여 옵션 메뉴를 엽니다.
4. 옵션 메뉴에서 비활성화를 선택합니다.

인증 기관은 목록에 비활성으로 표시되어야 합니다.

> **Note**
>
> AWS IoT 콘솔은 비활성화한 CA에서 서명한 인증서를 나열하는 방법을 제공하지 않습니다. 이러한 인증서를 나열하는 AWS CLI 옵션은 CA 인증서 비활성화(CLI) 단원을 참조하세요.

CA 인증서 비활성화(CLI)

는 CA 인증서를 비활성화하는 update-ca-certificate 명령을 AWS CLI 제공합니다.

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

list-certificates-by-ca 명령을 사용하여 지정된 CA에서 서명한 모든 등록된 디바이스 인증서의 목록을 가져옵니다. 지정된 CA 인증서에서 서명한 각 디바이스 인증서에 대해 클라이언트 인증서가 사용되지 않도록 update-certificate 명령을 사용하여 클라이언트 인증서를 취소합니다.

describe-ca-certificate 명령을 사용하여 CA 인증서의 상태를 확인합니다.

CA 인증서를 사용하여 클라이언트 인증서 생성

자체 인증 기관(CA)을 사용하여 클라이언트 인증서를 생성할 수 있습니다. 클라이언트 인증서는 사용하기 AWS IoT 전에에 등록해야 합니다. 클라이언트 인증서의 등록 옵션에 대한 자세한 내용은 단원을 참조하세요클라이언트 인증서 등록

클라이언트 인증서 생성(CLI)

> **Note**
>
> AWS IoT 콘솔에서는이 절차를 수행할 수 없습니다.

를 사용하여 클라이언트 인증서를 생성하려면 AWS CLI

1. 키 페어를 생성합니다.

```
openssl genrsa -out device_cert_key_filename.key 2048
```

2. 클라이언트 인증서에 대한 CSR을 생성합니다.

```
openssl req -new \
    -key device_cert_key_filename.key \
    -out device_cert_csr_filename.csr
```

아래와 같이 몇 가지 정보를 입력하라는 메시지가 표시됩니다.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

3. CSR에서 클라이언트 인증서를 생성합니다.

```
openssl x509 -req \
    -in device_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out device_cert_filename.pem \
    -days 500 -sha256
```

현재 클라이언트 인증서가 생성되었지만 아직 등록되지 않았습니다 AWS IoT. 클라이언트 인증서를 등록하는 방법 및 시기에 대한 자세한 내용은 단원을 참조하세요클라이언트 인증서 등록

클라이언트 인증서 등록

클라이언트와 간의 통신을 활성화 AWS IoT 하려면 클라이언트 인증서를에 등록해야 합니다 AWS IoT. 각 클라이언트 인증서를 수동으로 등록하거나 클라이언트가 AWS IoT 에 처음 연결할 때 자동으로 등록하도록 클라이언트 인증서를 구성할 수 있습니다.

처음 연결할 때 클라이언트 및 디바이스가 클라이언트 인증서를 등록하도록 하려면 CA 인증서 등록을(를) 사용하여 해당 클라이언트 인증서를 사용하려는 리전에서 AWS IoT (으)로 클라이언트 인증서에 서명해야 합니다. Amazon Root CA는에 자동으로 등록됩니다 AWS IoT.

클라이언트 인증서는 AWS 계정 및 리전에서 공유할 수 있습니다. 이 주제의 절차는 클라이언트 인증서를 사용할 각 계정 및 리전에서 수행해야 합니다. 한 계정 또는 리전에서 등록된 클라이언트 인증서는 다른 계정 또는 리전에서 자동으로 인식되지 않습니다.

> **Note**
>
> 전송 계층 보안(TLS) 프로토콜을 사용하여 AWS IoT 에 연결하는 클라이언트는 TLS에 대한 SNI(서버 이름 표시) 확장을 지원해야 합니다. 자세한 내용은 의 전송 보안 AWS IoT Core 단원을 참조하세요.

주제

- 수동으로 클라이언트 인증서 등록
- 클라이언트가 JITR(AWS IoT just-in-time Registration)에 연결할 때 클라이언트 인증서 등록

수동으로 클라이언트 인증서 등록

AWS IoT 콘솔 및를 사용하여 클라이언트 인증서를 수동으로 등록할 수 있습니다 AWS CLI.

사용할 등록 절차는 인증서가 AWS 계정및 리전에서 공유되는지 여부에 따라 달라집니다. 한 계정 또는 리전에서 등록된 클라이언트 인증서는 다른 계정 또는 리전에서 자동으로 인식되지 않습니다.

이 주제의 절차는 클라이언트 인증서를 사용할 각 계정 및 리전에서 수행해야 합니다. 클라이언트 인증서는 AWS 계정및 리전에서 공유할 수 있습니다.

등록된 CA로 서명된 클라이언트 인증서 등록(콘솔)

> **Note**
>
> 이 절차를 수행하기 전에 클라이언트 인증서의 .pem 파일이 있고 클라이언트 인증서가 등록한 AWS IoT CA에 의해 서명되었는지 확인합니다.

콘솔을 AWS IoT 사용하여에 기존 인증서를 등록하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 탐색 창의 관리(Manage) 섹션에서 보안(Security)을 선택한 다음 인증서(Certificates)를 선택합니다.
3. 인증서(Certificates) 대화 상자의 인증서(Certificates) 페이지에서 인증서 추가(Add certificate)를 선택한 다음 인증서 등록(Register certificates)을 선택합니다.
4. 업로드할 인증서(Certificates to upload) 대화 상자의 인증서 등록(Register certificate) 페이지에서 다음을 수행합니다.
   - CA가 AWS IoT에 등록됨을 선택합니다.
   - CA 인증서 선택(Choose a CA certificate)에서 인증 기관(Certification authority)을 선택합니다.
     - AWS IoT에 등록되지 않은 새 인증 기관(Certification authority)을 등록하려면 새 CA 등록(Register a new CA)을 선택합니다.
     - Amazon 루트 인증 기관(Amazon Root certificate authority)이 해당 인증 기관인 경우 CA 인증서 선택(Choose a CA certificate)을 비워 둡니다.
   - 업로드하고 등록할 인증서를 최대 10개까지 선택합니다 AWS IoT.
     - AWS IoT 클라이언트 인증서 생성 및 CA 인증서를 사용하여 클라이언트 인증서 생성에서 생성한 인증서 파일을 사용합니다.

- 활성화(Activate) 또는 비활성화(Deactivate)를 선택합니다. 비활성화(Deactivate)를 선택한 경우 인증서 등록 후 인증서를 활성화하는 방법이 클라이언트 인증서 활성화 또는 비활성화에 설명되어 있습니다.
- 등록을 선택합니다.

이제 인증서(Certificates) 대화 상자의 인증서(Certificates) 페이지에 등록된 인증서가 나타납니다.

등록되지 않은 CA로 서명된 클라이언트 인증서 등록(콘솔)

> **Note**
>
> 이 절차를 수행하기 전에 클라이언트 인증서의 .pem 파일이 있는지 확인해야 합니다.

콘솔을 AWS IoT 사용하여에 기존 인증서를 등록하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안, 인증서, 생성을 차례대로 선택합니다.
3. 인증서 생성에서 내 인증서 사용 항목을 찾아 시작하기를 선택합니다.
4. CA 선택에서 다음을 선택합니다.
5. 기존 디바이스 인증서 등록에서 인증서 선택을 선택하고 등록할 인증서 파일을 최대 10개까지 선택합니다.
6. 파일 대화 상자를 닫은 후 클라이언트 인증서를 등록할 때 클라이언트 인증서를 활성화할지 또는 취소할지 선택합니다.

   인증서를 등록할 때 인증서를 활성화하지 않으면 클라이언트 인증서 활성화(콘솔)에서 나중에 인증서를 활성화하는 방법에 대해 설명합니다.

   인증서를 등록할 때 인증서가 취소되면 나중에 활성화할 수 없습니다.

   등록할 인증서 파일을 선택하고 등록 후 수행할 작업을 선택한 후 인증서 등록을 선택합니다.

성공적으로 등록된 클라이언트 인증서가 인증서 목록에 나타납니다.

등록된 CA로 서명된 클라이언트 인증서 등록(CLI)

> **Note**
>
> 이 절차를 수행하기 전에 인증 기관(CA) .pem 및 클라이언트 인증서의 .pem 파일이 있는지 확인해야 합니다. 클라이언트 인증서는 등록한 AWS IoT 인증 기관(CA)에서 서명해야 합니다.

register-certificate 명령을 사용하여 클라이언트 인증서를 활성화하지 않고 등록합니다.

```
aws iot register-certificate \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

클라이언트 인증서가에 등록 AWS IoT되었지만 아직 활성화되지 않았습니다. 나중에 활성화하는 방법에 대한 자세한 내용은 클라이언트 인증서 활성화(CLI) 단원을 참조하세요.

이 명령을 사용하여 클라이언트 인증서를 등록할 때 클라이언트 인증서를 활성화할 수도 있습니다.

```
aws iot register-certificate \
    --set-as-active \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

연결에 사용할 수 있도록 인증서를 활성화하는 방법에 대한 자세한 내용은 섹션을 AWS IoT참조하세요. 클라이언트 인증서 활성화 또는 비활성화

등록되지 않은 CA로 서명된 클라이언트 인증서 등록(CLI)

> **Note**
>
> 이 절차를 수행하기 전에 인증서의 .pem 파일이 있는지 확인해야 합니다.

register-certificate-without-ca 명령을 사용하여 클라이언트 인증서를 활성화하지 않고 등록합니다.

```
aws iot register-certificate-without-ca \
    --certificate-pem file://device_cert_filename.pem
```

클라이언트 인증서가에 등록 AWS IoT되었지만 아직 활성화되지 않았습니다. 나중에 활성화하는 방법에 대한 자세한 내용은 클라이언트 인증서 활성화(CLI) 단원을 참조하세요.

이 명령을 사용하여 클라이언트 인증서를 등록할 때 클라이언트 인증서를 활성화할 수도 있습니다.

```
aws iot register-certificate-without-ca \
    --status ACTIVE \
    --certificate-pem file://device_cert_filename.pem
```

연결에 사용할 수 있도록 인증서를 활성화하는 방법에 대한 자세한 내용은 섹션을 AWS IoT참조하세요클라이언트 인증서 활성화 또는 비활성화.

클라이언트가 JITR(AWS IoT just-in-time Registration)에 연결할 때 클라이언트 인증서 등록

클라이언트가 처음 연결할 때에 AWS IoT 자동으로 등록하도록 서명된 클라이언트 인증서를 활성화하도록 CA 인증서를 구성할 수 있습니다 AWS IoT.

클라이언트가 AWS IoT 에 처음 연결할 때 클라이언트 인증서를 등록하려면 자동 등록을 위해 CA 인증서를 활성화하고 필요한 인증서를 제공하도록 클라이언트의 첫 번째 연결을 구성해야 합니다.

자동 등록을 지원하도록 CA 인증서 구성(콘솔)

AWS IoT 콘솔을 사용하여 자동 클라이언트 인증서 등록을 지원하도록 CA 인증서를 구성하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안을 선택한 후 CA를 선택합니다.
3. 인증 기관 목록에서 자동 등록을 활성화할 인증 기관을 찾은 다음 줄임표 아이콘을 사용하여 옵션 메뉴를 엽니다.
4. 옵션 메뉴에서 자동 등록 활성화를 선택합니다.

> **Note**
>
> 인증 기관 목록에 자동 등록 상태가 표시되지 않습니다. 인증 기관의 자동 등록 상태를 보려면 인증 기관의 세부 정보 페이지를 열어야 합니다.

자동 등록을 지원하도록 CA 인증서 구성(CLI)

CA 인증서를에 이미 등록한 경우 update-ca-certificate 명령을 AWS IoT사용하여 `autoRegistrationStatus` CA 인증서를 로 설정합니다`ENABLE`.

```
aws iot update-ca-certificate \
--certificate-id caCertificateId \
--new-auto-registration-status ENABLE
```

CA 인증서를 등록할 때 autoRegistrationStatus를 활성화하려면 register-ca-certificate 명령을 사용합니다.

```
aws iot register-ca-certificate \
--allow-auto-registration  \
--ca-certificate file://root_CA_cert_filename.pem \
--verification-cert file://verification_cert_filename.pem
```

describe-ca-certificate 명령을 사용하여 CA 인증서의 상태를 확인합니다.

자동 등록을 위해 클라이언트에 의한 첫 번째 연결 구성

클라이언트가 처음으로 AWS IoT 에 연결하려고 하면 전송 계층 보안(TLS) 핸드셰이크 중에 CA 인증서로 서명된 클라이언트 인증서가 클라이언트에 있어야 합니다.

클라이언트가 연결되면 AWS IoT 클라이언트 인증서 생성 또는 자체 클라이언트 인증서 생성에서 생성한 클라이언트 인증서를 AWS IoT사용합니다.는 CA 인증서를 등록된 CA 인증서로 AWS IoT 인식하고, 클라이언트 인증서를 등록하고, 상태를 로 설정합니다PENDING_ACTIVATION. 이는 클라이언트 인증서가 자동으로 등록되었고 활성화를 기다리는 중임을 의미합니다. 클라이언트 인증서를 사용하여 AWS IoT에 연결하려면 먼저 클라이언트 인증서가 ACTIVE 상태여야 합니다. 클라이언트 인증서 활성화에 대한 자세한 내용은 클라이언트 인증서 활성화 또는 비활성화 섹션을 참조하세요.

> **Note**
>
> 디바이스의 첫 번째 연결에서 전체 신뢰 체인을 보낼 필요 없이 JITR(AWS IoT Core just-in-time Registration) 기능을 사용하여 디바이스를 프로비저닝할 수 있습니다 AWS IoT Core. CA 인증서를 제시하는 것은 선택 사항이지만 디바이스가 연결할 때 서버 이름 표시(SNI) 확장을 전송해야 합니다.

가 인증서를 AWS IoT 자동으로 등록하거나 클라이언트가 PENDING_ACTIVATION 상태의 인증서를 제시하면는 다음 MQTT 주제에 메시지를 AWS IoT 게시합니다.

$aws/events/certificates/registered/*caCertificateId*

여기서 *caCertificateId*는 디바이스 인증서를 발행한 CA 인증서의 ID입니다.

이 주제에 게시된 메시지는 구조가 다음과 같습니다.

```
{
        "certificateId": "certificateId",
        "caCertificateId": "caCertificateId",
        "timestamp": timestamp,
        "certificateStatus": "PENDING_ACTIVATION",
        "awsAccountId": "awsAccountId",
        "certificateRegistrationTimestamp": "certificateRegistrationTimestamp"
}
```

이 주제를 수신 대기하고 일부 작업을 수행하는 규칙을 생성할 수 있습니다. 클라이언트 인증서가 인증서 취소 목록(CRL)에 포함되지 않음을 확인하고, 인증서를 활성화하고, 정책을 생성하여 인증서에 연결하는 Lambda 규칙을 생성하는 것이 좋습니다. 이 정책은 클라이언트가 어떤 리소스에 액세스할 수 있는지를 결정합니다. 생성 중인 정책에 연결되어 있는 디바이스의 클라이언트 ID가 필요한 경우, 규칙의 clientid() 함수를 사용하여 클라이언트 ID를 검색할 수 있습니다. 규칙 정의 예제는 다음과 같습니다.

```
SELECT *,
   clientid() as clientid
from $aws/events/certificates/registered/caCertificateId
```

이 예제에서 규칙은 JITR 주제 `$aws/events/certificates/registered/caCertificateID`를 구독하고 clientid() 함수를 사용하여 클라이언트 ID를 검색합니다. 그런 다음 규칙은 클라이언트 ID를 JITR 페이로드에 추가합니다. 규칙의 clientid() 함수에 대한 자세한 내용은 clientid()를 참조하세요.

`$aws/events/certificates/registered/caCertificateID` 주제를 수신 대기하고 이러한 작업을 수행하는 Lambda 규칙을 생성하는 방법에 대한 자세한 내용은Just-in-Time Registration of Device Certificates on AWS IoT을 참조하세요.

클라이언트 인증서의 자동 등록 중에 오류 또는 예외가 발생하면는 CloudWatch Logs의 로그에 이벤트 또는 메시지를 AWS IoT 전송합니다. 계정에 대한 로그 설정에 대한 자세한 내용은 Amazon CloudWatch 설명서를 참조하세요.

클라이언트 인증서 관리

AWS IoT 는 클라이언트 인증서를 관리할 수 있는 기능을 제공합니다.

이 주제에서 수행할 작업

- 클라이언트 인증서 활성화 또는 비활성화

- 클라이언트 인증서에 사물 또는 정책 연결
- 클라이언트 인증서 취소
- 인증서를 다른 계정으로 이전합니다.

클라이언트 인증서 활성화 또는 비활성화

AWS IoT 는 연결을 인증할 때 클라이언트 인증서가 활성 상태인지 확인합니다.

클라이언트 인증서를 활성화하지 않은 상태에서 생성하고 등록할 수 있으므로 사용할 때까지 클라이언트 인증서를 사용할 수 없습니다. 활성 클라이언트 인증서를 비활성화하여 일시적으로 비활성화할 수도 있습니다. 마지막으로 클라이언트 인증서를 취소하여 나중에 사용할 수 없도록 할 수 있습니다.

클라이언트 인증서 활성화(콘솔)

AWS IoT 콘솔을 사용하여 클라이언트 인증서를 활성화하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.
3. 인증서 목록에서 활성화할 인증서를 찾은 다음 줄임표 아이콘을 사용하여 옵션 메뉴를 엽니다.
4. 옵션 메뉴에서 활성화를 선택합니다.

인증서는 인증서 목록에 활성으로 표시되어야 합니다.

클라이언트 인증서 비활성화(콘솔)

AWS IoT 콘솔을 사용하여 클라이언트 인증서를 비활성화하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.
3. 인증서 목록에서 비활성화할 인증서를 찾은 다음 줄임표 아이콘을 사용하여 옵션 메뉴를 엽니다.
4. 옵션 메뉴에서 비활성화를 선택합니다.

인증서는 인증서 목록에 비활성으로 표시되어야 합니다.

클라이언트 인증서 활성화(CLI)

는 인증서를 활성화하는 update-certificate 명령을 AWS CLI 제공합니다.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

명령이 성공하면 인증서가 ACTIVE 상태가 됩니다. describe-certificate를 실행하여 인증서의 상태를 확인합니다.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

클라이언트 인증서 비활성화(CLI)

는 인증서를 비활성화하는 update-certificate 명령을 AWS CLI 제공합니다.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

명령이 성공하면 인증서가 INACTIVE 상태가 됩니다. describe-certificate를 실행하여 인증서의 상태를 확인합니다.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

클라이언트 인증서에 사물 또는 정책 연결

AWS IoT 사물과 별도로 인증서를 생성하고 등록하면 AWS IoT 작업을 승인하는 정책이 없으며 AWS IoT 사물 객체와도 연결되지 않습니다. 이 섹션에서는 이러한 관계를 등록된 인증서에 추가하는 방법에 대해 설명합니다.

> **Important**
>
> 이러한 절차를 완료하려면 인증서에 연결할 사물이나 정책을 이미 생성했어야 합니다.

인증서는 연결할 수 AWS IoT 있도록를 사용하여 디바이스를 인증합니다. 인증서를 사물 리소스에 연결하면 디바이스와 (인증서를 통해) 리소스 간의 관계가 설정됩니다. 디바이스가 메시지를 연결하고 게시할 수 있도록 허용하는 등의 AWS IoT 작업을 수행하도록 디바이스에 권한을 부여하려면 디바이스의 인증서에 적절한 정책을 연결해야 합니다.

클라이언트 인증서에 사물 연결(콘솔)

이 절차를 완료하려면 사물 객체의 이름이 필요합니다.

등록된 인증서에 사물 객체를 연결하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.
3. 인증서 목록에서 정책을 연결할 인증서를 찾고 줄임표 아이콘을 선택하여 인증서의 옵션 메뉴를 연 다음 사물 연결을 선택합니다.
4. 팝업에서 인증서에 연결할 사물의 이름을 찾아 해당 확인란을 선택한 다음 연결을 선택합니다.

이제 사물 객체가 인증서의 세부 정보 페이지에 있는 사물 목록에 나타나야 합니다.

클라이언트 인증서에 정책 연결(콘솔)

이 절차를 완료하려면 정책 객체의 이름이 필요합니다.

등록된 인증서에 정책 객체를 연결하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.
3. 인증서 목록에서 정책을 연결할 인증서를 찾고 줄임표 아이콘을 선택하여 인증서의 옵션 메뉴를 연 다음 정책 연결을 선택합니다.
4. 팝업에서 인증서에 연결할 정책의 이름을 찾아 해당 확인란을 선택한 다음 연결을 선택합니다.

이제 정책 객체가 인증서의 세부 정보 페이지에 있는 정책 목록에 나타납니다.

클라이언트 인증서에 사물 연결(CLI)

는 사물 객체를 인증서에 연결하는 attach-thing-principal 명령을 AWS CLI 제공합니다.

```
aws iot attach-thing-principal \
    --principal certificateArn \
    --thing-name thingName
```

클라이언트 인증서에 정책 연결(CLI)

는 인증서에 정책 객체를 연결하는 attach-policy 명령을 AWS CLI 제공합니다.

```
aws iot attach-policy \
    --target certificateArn \
    --policy-name policyName
```

클라이언트 인증서 취소

등록된 클라이언트 인증서에서 의심스러운 활동이 감지되는 경우 다시 사용할 수 없도록 클라이언트 인증서를 취소할 수 있습니다.

> **Note**
>
> 인증서가 취소되면 상태를 변경할 수 없습니다. 즉, 인증서 상태를 `Active` 또는 다른 상태로 변경할 수 없습니다.

클라이언트 인증서 취소(콘솔)

AWS IoT 콘솔을 사용하여 클라이언트 인증서를 취소하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.
3. 인증서 목록에서 취소할 인증서를 찾은 다음 줄임표 아이콘을 사용하여 옵션 메뉴를 엽니다.
4. 옵션 메뉴에서 취소를 선택합니다.

인증서가 성공적으로 취소되면 인증서 목록에 취소됨으로 표시됩니다.

클라이언트 인증서(CLI) 취소

는 인증서를 취소하는 update-certificate 명령을 AWS CLI 제공합니다.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status REVOKED
```

명령이 성공하면 인증서가 `REVOKED` 상태가 됩니다. describe-certificate를 실행하여 인증서의 상태를 확인합니다.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

인증서를 다른 계정으로 이전합니다.

하나에 속하는 X.509 인증서는 다른 인증서로 전송할 AWS 계정 수 있습니다 AWS 계정.

X.509 인증서를 한 인증서에서 AWS 계정 다른 인증서로 전송하려면

1. the section called “인증서 전송 시작”

   전송을 시작하기 전에 인증서를 비활성화하고 모든 정책과 사물에서 분리해야 합니다.
2. the section called “인증서 전송 수락 또는 거부”

   수신 계정은 전송된 인증서를 명시적으로 수락하거나 거부해야 합니다. 수신 계정에서 인증서를 수락한 후에는 사용하기 전에 인증서를 활성화해야 합니다.
3. the section called “인증서 전송 취소”

   인증서가 수락되지 않은 경우 원래 계정에서 전송을 취소할 수 있습니다.

인증서 전송 시작

AWS IoT 콘솔 또는를 사용하여 인증서를 다른 AWS 계정 로 전송하기 시작할 수 있습니다 AWS CLI.

인증서 전송 시작(콘솔)

이 절차를 완료하려면 전송할 인증서의 ID가 필요합니다.

전송할 인증서가 있는 계정에서 이 절차를 수행하세요.

인증서를 다른 로 전송하기 시작하려면 AWS 계정

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.

   전송하려는 활성 상태 또는 비활성 상태의 인증서를 선택하고 세부 정보 페이지를 엽니다.
3. 인증서의 세부 정보 페이지의 작업 메뉴에서 비활성화 옵션을 사용할 수 있는 경우 비활성화 옵션을 사용하여 인증서를 비활성화할 수 있습니다.
4. 인증서의 세부 정보 페이지의 왼쪽 메뉴에서 정책을 선택합니다.
5. 인증서의 정책 페이지에서 인증서에 연결된 정책이 있는 경우 정책의 옵션 메뉴를 열고 분리를 선택하여 각 정책을 분리합니다.

   계속하기 전에 인증서에는 정책이 연결되어 있으면 안 됩니다.

6. 인증서의 정책 페이지의 왼쪽 메뉴에서 사물을 선택합니다.
7. 인증서의 사물 페이지에서 인증서에 연결된 사물이 있는 경우 사물의 옵션 메뉴를 열고 분리를 선택하여 각 사물을 분리합니다.

   계속하기 전에 인증서에는 사물이 연결되어 있으면 안 됩니다.
8. 인증서의 사물 페이지의 왼쪽 메뉴에서 세부 정보를 선택합니다.
9. 인증서의 세부 정보 페이지의 작업 메뉴에서 전송 시작을 선택하고 전송 시작 대화 상자를 엽니다.
10. 전송 시작 대화 상자에서 인증서를 수신할 계정의 AWS 계정 번호와 선택적 단축 메시지를 입력합니다.
11. 전송 시작을 선택하여 인증서를 전송합니다.

콘솔에 전송의 성공 또는 실패를 나타내는 메시지가 표시됩니다. 전송이 시작된 경우 인증서의 상태가 전송됨으로 업데이트됩니다.

인증서 전송 시작(CLI)

이 절차를 완료하려면 전송하려는 인증서의 *certificateId* 및 *certificateArn*이 필요합니다.

전송할 인증서가 있는 계정에서 이 절차를 수행하세요.

인증서를 다른 AWS 계정으로 이전하기 시작하려면

1. update-certificate 명령을 사용하여 인증서를 비활성화합니다.

   ```
   aws iot update-certificate --certificate-id certificateId --new-status INACTIVE
   ```
2. 모든 정책을 분리합니다.

   1. list-attached-policies 명령을 사용하여 인증서에 연결되어 있는 정책을 나열합니다.

      ```
      aws iot list-attached-policies --target certificateArn
      ```
   2. 연결된 각 정책에 대해 detach-policy 명령을 사용하여 정책을 분리합니다.

      ```
      aws iot detach-policy --target certificateArn --policy-name policy-name
      ```
3. 모든 사물을 분리합니다.

   1. list-principal-things 명령을 사용하여 인증서에 연결되어 있는 사물을 나열합니다.

```
aws iot list-principal-things --principal certificateArn
```

2. 연결된 각 사물에 대해 detach-thing-principal 명령을 사용하여 사물을 분리합니다.

```
aws iot detach-thing-principal --principal certificateArn --thing-name thing-name
```

4. transfer-certificate 명령을 사용하여 인증서 전송을 시작합니다.

```
aws iot transfer-certificate --certificate-id certificateId --target-aws-
account account-id
```

인증서 전송 수락 또는 거부

AWS IoT 콘솔 또는를 사용하여 다른에서 전송 AWS 계정 AWS 계정 된 인증서를 수락하거나 거부할 수 있습니다 AWS CLI.

인증서 전송 수락 또는 거부(콘솔)

이 절차를 완료하려면 계정으로 전송한 인증서의 ID가 필요합니다.

전송된 인증서를 수신하는 계정에서 이 절차를 수행하세요.

로 전송된 인증서를 수락하거나 거부하려면 AWS 계정

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.

   수락 또는 거부하려는 전송 대기 중 상태의 인증서를 선택하고 세부 정보 페이지를 엽니다.
3. 인증서의 세부 정보 페이지의 작업 메뉴에서

   - 인증서를 수락하려면 전송 수락을 선택합니다.
   - 인증서를 수락하지 않으려면 전송 거부를 선택합니다.

인증서 전송 수락 또는 거부(CLI)

이 절차를 완료하려면 수락 또는 거부하려는 인증서 전송의 *certificateId*가 필요합니다.

전송된 인증서를 수신하는 계정에서 이 절차를 수행하세요.

로 전송된 인증서를 수락하거나 거부하려면 AWS 계정

1. accept-certificate-transfer 명령을 사용하여 인증서를 수락합니다.

```
aws iot accept-certificate-transfer --certificate-id certificateId
```

2. reject-certificate-transfer 명령을 사용하여 인증서를 거부합니다.

```
aws iot reject-certificate-transfer --certificate-id certificateId
```

인증서 전송 취소

AWS IoT 콘솔 또는 AWS CLI을(를) 사용하여 수락하기 전에 인증서 전송을 취소할 수 있습니다.

인증서 전송 취소(콘솔)

이 절차를 완료하려면 취소하려는 인증서 전송의 ID가 필요합니다.

인증서 전송을 시작한 계정에서 이 절차를 수행하세요.

인증서 전송을 취소하려면

1. AWS Management Console에 로그인하고 AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 보안(Secure)을 선택한 후 인증서(Certificates)를 선택합니다.

   취소하려는 전송됨 상태의 인증서를 선택하고 옵션 메뉴를 엽니다.
3. 인증서의 옵션 메뉴에서 전송 취소 옵션을 선택하여 인증서 전송을 취소합니다.

> **Important**
>
> 전송 취소 옵션을 취소 옵션과 혼동하지 않도록 주의하세요.
> 전송 취소 옵션은 인증서 전송을 취소하고 취소 옵션은 인증서를 불가역적으로 사용 불가능하게 만듭니다. AWS IoT.

인증서 전송 취소(CLI)

이 절차를 완료하려면 취소하려는 인증서 전송의 *certificateId*가 필요합니다.

인증서 전송을 시작한 계정에서 이 절차를 수행하세요.

cancel-certificate-transfer 명령을 사용하여 인증서 전송을 취소합니다.

```
aws iot cancel-certificate-transfer --certificate-id certificateId
```

사용자 지정 클라이언트 인증서 검증

AWS IoT Core 는 X.509 클라이언트 인증서에 대한 사용자 지정 클라이언트 인증서 검증을 지원하여 클라이언트 인증 관리를 개선합니다. 이 인증서 검증 방법은 사전 인증 인증서 검사라고도 하며, 이 검사에서는 자체 기준(Lambda 함수에 정의)을 기반으로 클라이언트 인증서를 평가하고 클라이언트 인증서 또는 인증서의 서명 인증 기관(CA) 인증서를 취소하여 클라이언트가 AWS IoT Core에 연결되지 않도록 합니다. 예를 들어 온라인 인증서 상태 프로토콜(OCSP) 또는 인증서 해지 목록(CRL) 엔드포인트를 지원하는 검증 기관과 비교하여 인증서의 상태를 검증하고 취소된 인증서가 있는 클라이언트에 대한 연결을 방지하는 자체 인증서 해지 검사를 생성할 수 있습니다. 클라이언트 인증서를 평가하는 데 사용되는 기준은 Lambda 함수(사전 인증 Lambda라고도 함)에 정의됩니다. 도메인 구성에서 엔드포인트 세트를 사용해야 하며 인증 유형은 X.509 인증서여야 합니다. 또한 클라이언트는 연결할 때 서버 이름 표시(SNI) 확장을 제공해야 합니다 AWS IoT Core.

> **Note**
>
> 이 기능은 AWS GovCloud (US) 리전에서 지원되지 않습니다.

사용자 지정 클라이언트 인증서 검증을 수행하는 프로세스에는 다음 단계가 포함됩니다.

- 1단계:에 X.509 클라이언트 인증서 등록 AWS IoT Core
- 2단계: Lambda 함수 생성
- 3단계: Lambda 함수 AWS IoT 호출 권한 부여
- 4단계: 도메인에 대한 인증 구성 설정

1단계:에 X.509 클라이언트 인증서 등록 AWS IoT Core

아직 등록하지 않은 경우에 X.509 클라이언트 인증서를 등록하고 활성화합니다 AWS IoT Core. 그렇지 않은 경우 다음 단계로 건너뜁니다.

에 클라이언트 인증서를 등록하고 활성화하려면 다음 단계를 AWS IoT Core따릅니다.

1. 를 사용하여 직접 클라이언트 인증서를 생성하는 AWS IoT 경우. 이러한 클라이언트 인증서는에 자동으로 등록됩니다 AWS IoT Core.

2. 자체 클라이언트 인증서를 생성하는 경우 다음 지침에 따라 인증서를 등록합니다 AWS IoT Core.
3. 클라이언트 인증서를 활성화하려면 다음 지침을 따르세요.

2단계: Lambda 함수 생성

인증서 확인을 수행하고 구성된 엔드포인트에 대한 모든 클라이언트 연결 시도에 대해 호출되는 Lambda 함수를 생성해야 합니다. 이 Lambda 함수를 생성할 때는 첫 번째 Lambda 함수 생성의 일반 지침을 따릅니다. 또한 Lambda 함수가 다음과 같이 예상 요청 및 응답 형식을 준수하는지 확인합니다.

Lambda 함수 이벤트 예제

```
{
 "connectionMetadata": {
  "id": "string"
 },
 "principalId": "string",
 "serverName": "string",
 "clientCertificateChain": [
  "string",
  "string"
 ]
}
```

`connectionMetadata`

클라이언트의 AWS IoT Core에 대한 연결과 관련된 메타데이터 또는 추가 정보입니다.

`principalId`

TLS 연결에서 클라이언트와 연결된 위탁자 식별자입니다.

`serverName`

서버 이름 표시(SNI) 호스트 이름 스트링입니다. AWS IoT Core 는 디바이스가 SNI 확장을 Transport Layer Security (TLS) 프로토콜로 전송하고 `host_name` 필드에 전체 엔드포인트 주소를 제공하도록 요구합니다.

`clientCertificateChain`

클라이언트의 X.509 인증서 체인을 나타내는 문자열 배열입니다.

Lambda 함수 응답 예제

```
{
 "isAuthenticated": "boolean"
}
```

`isAuthenticated`

요청 인증 여부를 나타내는 부울 값입니다.

> **Note**
>
> Lambda 응답에서 추가 인증 및 권한 부여를 진행하려면 `isAuthenticated`가 반드시 `true`여야 합니다. 그렇지 않으면 IoT 클라이언트 인증서를 비활성화하고 추가 인증 및 권한 부여를 위해 X.509 클라이언트 인증서를 사용한 사용자 지정 인증을 차단할 수 있습니다.

3단계: Lambda 함수 AWS IoT 호출 권한 부여

Lambda 함수를 생성한 후 add-permission CLI 명령을 사용하여 AWS IoT 에 호출 권한을 부여해야 합니다. 이 Lambda 함수는 구성된 엔드포인트에 연결할 때마다 간접적으로 호출됩니다. 자세한 내용은 Lambda 함수 호출 권한 부여 AWS IoT 를 참조하세요.

4단계: 도메인에 대한 인증 구성 설정

다음 섹션에서는 AWS CLI를 사용하여 사용자 지정 도메인에 대한 인증 구성을 설정하는 방법을 설명합니다.

도메인에 대한 클라이언트 인증서 구성 설정(CLI)

도메인 구성이 없는 경우 create-domain-configuration CLI 명령을 사용하여 생성합니다. 도메인 구성이 이미 있는 경우 update-domain-configuration CLI 명령을 사용하여 도메인에 대한 클라이언트 인증서 구성을 업데이트합니다. 이전 단계에서 생성한 Lambda 함수의 ARN을 추가해야 합니다.

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type AWS_X509|CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT|HTTPS \
    --client-certificate-config 'clientCertificateCallbackArn":"arn:aws:lambda:us-
east-2:123456789012:function:my-function:1"}'
```

```
aws iot update-domain-configuration \
```

```
    --domain-configuration-name domainConfigurationName \
    --authentication-type AWS_X509|CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT|HTTPS \
    --client-certificate-config '{"clientCertificateCallbackArn":"arn:aws:lambda:us-
east-2:123456789012:function:my-function:1"}'
```

`domain-configuration-name`

도메인 구성의 이름입니다.

`authentication-type`

도메인 구성의 인증 유형입니다. 자세한 내용은 인증 유형 선택을 참조하세요.

`application-protocol`

AWS IoT Core와 통신하는 데 사용하는 디바이스의 애플리케이션 프로토콜입니다. 자세한 내용은 애플리케이션 프로토콜 선택을 참조하세요.

`client-certificate-config`

도메인의 클라이언트 인증 구성을 지정하는 객체입니다.

`clientCertificateCallbackArn`

새 연결이 설정될 때 TLS 계층에서 AWS IoT 호출하는 Lambda 함수의 Amazon 리소스 이름(ARN)입니다. 사용자 지정 클라이언트 인증서 검증을 수행하도록 클라이언트 인증을 사용자 지정하려면 이전 단계에서 생성한 Lambda 함수의 ARN을 추가해야 합니다.

자세한 내용은 AWS IoT API 참조의 CreateDomainConfiguration 및 UpdateDomainConfiguration을 참조하세요. 도메인 구성에 대한 자세한 내용은 Domain configurations를 참조하세요.

IAM 사용자, 그룹 및 역할

IAM 사용자, 그룹 및 역할은 AWS에서 자격 증명 및 인증을 관리하기 위한 표준 메커니즘입니다. SDK AWS 및를 사용하여 AWS IoT HTTP 인터페이스에 연결할 수 있습니다 AWS CLI.

또한 IAM 역할을 사용하면 AWS IoT 가 사용자를 대신하여 계정의 다른 AWS 리소스에 액세스할 수 있습니다. 예를 들어 디바이스가 DynamoDB 테이블에 상태를 게시하도록 하려면 IAM 역할을 통해가 Amazon DynamoDB와 상호 작용 AWS IoT 할 수 있습니다. 자세한 내용은 IAM 역할을 참조하세요.

HTTP를 통한 메시지 브로커 연결의 경우 서명 버전 4 서명 프로세스를 사용하여 사용자, 그룹 및 역할을 AWS IoT 인증합니다. 자세한 내용은 AWS API 요청 서명을 참조하세요.

에서 AWS 서명 버전 4를 사용하는 경우 AWS IoT클라이언트는 TLS 구현에서 다음을 지원해야 합니다.

- TLS 1.2
- SHA-256 RSA 인증서 서명 검증
- TLS 암호 그룹 지원 섹션에서 암호 그룹 중 하나

자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT 단원을 참조하세요.

Amazon Cognito 자격 증명

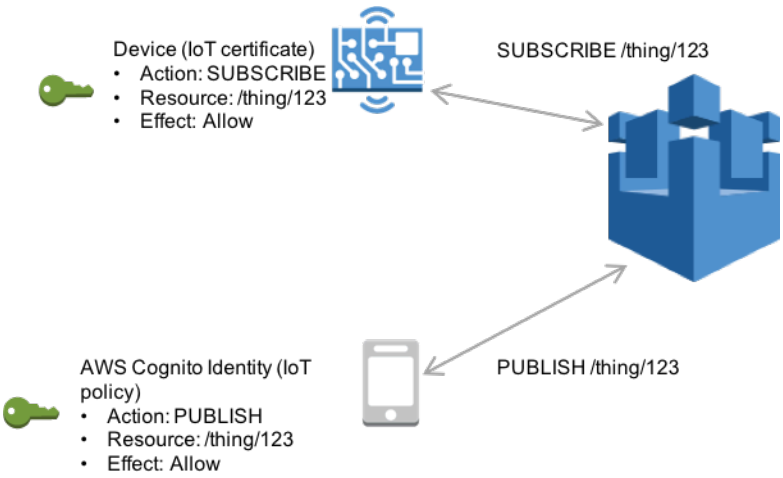
Amazon Cognito 자격 증명을 사용하면 모바일 및 웹 애플리케이션에서 사용할 수 있는 제한된 임시 권한 AWS 자격 증명을 생성할 수 있습니다. Amazon Cognito 자격 증명을 사용하는 경우, 사용자의 고유 자격 증명을 생성하는 ID 풀을 생성하여 Login with Amazon, Facebook 및 Google과 같은 자격 증명 공급자를 사용하여 인증하는 ID 풀을 생성합니다. 자체 개발자 인증 자격 증명과 함께 Amazon Cognito 자격 증명을 사용할 수도 있습니다. 자세한 내용은 Amazon Cognito 자격 증명을 참조하세요.

Amazon Cognito 자격 증명을 사용하려면 IAM 역할과 연결된 Amazon Cognito ID 풀을 정의합니다. IAM 역할은 ID 풀의 ID에 서비스 호출 AWS 과 같은 AWS 리소스에 액세스할 수 있는 권한을 부여하는 IAM 정책과 연결됩니다.

Amazon Cognito 자격 증명은 인증되지 않은 자격 증명 및 인증된 자격 증명을 생성합니다. 인증되지 않은 자격 증명은 로그인하지 않고 앱을 사용하려는 모바일 또는 웹 애플리케이션의 게스트 사용자에게 사용됩니다. 인증되지 않은 사용자에게는 자격 증명 풀과 연결된 IAM 정책에 지정된 권한만 부여됩니다.

인증된 자격 증명을 사용하는 경우 자격 증명 풀에 연결된 IAM 정책 외에도 Amazon Cognito 자격 증명에 AWS IoT 정책을 연결해야 합니다. AWS IoT 정책을 연결하려면 AttachPolicy API를 사용하고 AWS IoT 애플리케이션의 개별 사용자에게 권한을 부여합니다. AWS IoT 정책을 사용하여 특정 고객과 디바이스에 세분화된 권한을 할당할 수 있습니다.

인증된 사용자와 인증되지 않은 사용자의 자격 증명 유형은 서로 다릅니다. Amazon Cognito 자격 증명에 AWS IoT 정책을 연결하지 않으면 인증된 사용자가에서 권한 부여에 실패 AWS IoT 하고 AWS IoT 리소스 및 작업에 액세스할 수 없습니다. Amazon Cognito 자격 증명에 대한 정책을 생성하는 방법에 대한 자세한 내용은 게시/구독 정책 예제 및 Amazon Cognito 자격 증명으로 권한 부여 섹션을 참조하세요.



사용자 지정 인증 및 권한 부여

AWS IoT Core 를 사용하면 사용자 지정 권한 부여자를 정의하여 자체 클라이언트 인증 및 권한 부여를 관리할 수 있습니다. 이는에서 AWS IoT Core 기본적으로 지원하는 인증 메커니즘 이외의 인증 메커니즘을 사용해야 하는 경우에 유용합니다. (기본적으로 지원되는 메커니즘에 대한 자세한 내용은 the section called “클라이언트 인증” 단원을 참조하세요).

예를 들어 필드의 기존 디바이스를 로 마이그레이션 AWS IoT Core 하고 이러한 디바이스가 사용자 지정 보유자 토큰 또는 MQTT 사용자 이름과 암호를 사용하여 인증하는 경우 새 자격 증명을 프로비저닝 AWS IoT Core 하지 않고도 로 마이그레이션할 수 있습니다. 가 AWS IoT Core 지원하는 모든 통신 프로토콜에서 사용자 지정 인증을 사용할 수 있습니다. AWS IoT Core 이(가) 지원하는 프로토콜 및 포트에 대한 자세한 내용은 the section called “디바이스 통신 프로토콜” 단원을 참조하세요.

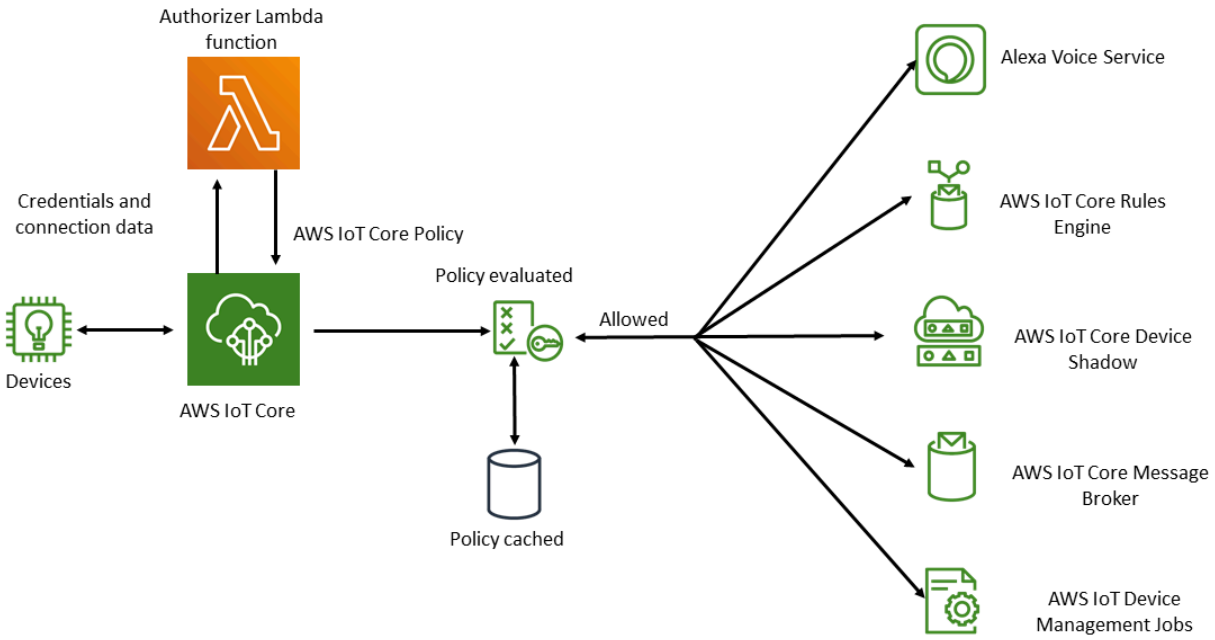
주제

- 사용자 지정 인증 워크플로 이해
- 사용자 지정 권한 부여자 생성 및 관리(CLI)
- X.509 클라이언트 인증서를 사용한 사용자 지정 인증
- 사용자 지정 인증을 사용하여 AWS IoT Core 에 연결
- Lambda 권한 부여자 문제 해결

사용자 지정 인증 워크플로 이해

사용자 지정 인증은 권한 부여자 리소스를 사용하여 클라이언트를 인증하고 권한을 부여하는 방법을 정의할 수 있도록 합니다. 각 권한 부여자에는 고객 관리형 Lambda 함수에 대한 참조, 디바이스 자격

증명을 검증하기 위한 공개 키 옵션 및 추가 구성 정보가 포함되어 있습니다. 다음 다이어그램은 AWS IoT Core에서 사용자 지정 인증에 대한 권한 부여 워크플로를 보여 줍니다.



AWS IoT Core 사용자 지정 인증 및 권한 부여 워크플로

다음 목록에서는 사용자 지정 인증 및 권한 부여 워크플로의 각 단계를 설명합니다.

1. 디바이스는 지원되는 중 하나를 사용하여 고객의 AWS IoT Core 데이터 엔드포인트에 연결합니다the section called “디바이스 통신 프로토콜”. 디바이스는 요청의 헤더 필드 또는 쿼리 파라미터(HTTP Publish 또는 WebSockets 프로토콜을 통한 MQTT의 경우) 중 하나에서, 또는 MQTT CONNECT 메시지의 사용자 이름 및 암호 필드(MQTT 및 WebSockets 프로토콜을 통한 MQTT의 경우)에서 자격 증명을 전달합니다.
2. AWS IoT Core 는 다음 두 가지 조건 중 하나를 확인합니다.
   - 수신되는 요청이 권한 부여자를 지정합니다.
   - 요청을 수신하는 AWS IoT Core 데이터 엔드포인트에는 기본 권한 부여자가 구성되어 있습니다.

   가 이러한 방법 AWS IoT Core 중 하나로 권한 부여자를 AWS IoT Core 찾으면는 권한 부여자와 연결된 Lambda 함수를 트리거합니다.

3. (선택 사항) 토큰 서명을 활성화한 경우는 Lambda 함수를 트리거하기 전에 권한 부여자에 저장된 퍼블릭 키를 사용하여 요청 서명을 AWS IoT Core 확인합니다. 유효성 검사에 실패하면 AWS IoT Core 이(가) Lambda 함수를 호출하지 않고 요청을 중지합니다.
4. Lambda 함수는 요청에서 자격 증명 및 연결 메타데이터를 수신하고 인증 결정을 내립니다.
5. Lambda 함수는 인증 결정 결과와 연결에서 허용되는 작업을 지정하는 AWS IoT Core 정책 문서를 반환합니다. 또한 Lambda 함수는가 Lambda 함수를 호출하여 요청의 자격 증명을 AWS IoT Core 재검증하는 빈도를 지정하는 정보도 반환합니다.
6. AWS IoT Core 는 Lambda 함수에서 수신한 정책을 기준으로 연결에 대한 활동을 평가합니다.
7. 연결이 설정되고 사용자 지정 권한 부여자 Lambda가 처음 호출되면 MQTT 작업 없이 유휴 연결에서 다음 간접 호출을 최대 5분 동안 지연할 수 있습니다. 그런 다음 후속 간접 호출은 사용자 지정 권한 부여자 Lambda의 새로 고침 간격을 따릅니다. 이 접근 방식은의 Lambda 동시성 제한을 초과할 수 있는 과도한 호출을 방지할 수 있습니다 AWS 계정.

스케일 아웃 고려 사항

Lambda 함수는 권한 부여자에 대한 인증 및 권한 부여를 처리하므로 동시 실행 속도와 같은 Lambda 요금 및 서비스 제한이 적용됩니다. Lambda 요금에 대한 자세한 내용은 Lambda 요금 단원을 참조하세요. Lambda 함수 응답에서 `refreshAfterInSeconds` 및 `disconnectAfterInSeconds` 파라미터를 조정하여 Lambda 함수의 로드를 관리할 수 있습니다. Lambda 함수 응답의 내용에 대한 자세한 내용은 the section called “Lambda 함수 정의” 단원을 참조하세요.

> **Note**
>
> 서명을 활성화한 상태로 두면 인식할 수 없는 클라이언트가 Lambda를 과도하게 트리거하는 것을 방지할 수 있습니다. 권한 부여자의 로그인을 비활성화하기 전에 이 점을 고려하세요.

> **Note**
>
> 사용자 지정 권한 부여자에 대한 Lambda 함수 제한 시간은 5초입니다.

사용자 지정 권한 부여자 생성 및 관리(CLI)

AWS IoT Core 는 사용자 지정 권한 부여자를 사용하여 사용자 지정 인증 및 권한 부여 체계를 구현합니다. 사용자 지정 권한 부여자는 특정 요구 사항에 따라 규칙과 정책을 정의하고 구현할 수 있는 유연

성을 제공하는 AWS IoT Core 리소스입니다. 단계별 지침에 따라 사용자 지정 권한 부여자를 생성하려면 Tutorial: Creating a custom authorizer for AWS IoT Core를 참조하세요.

각 권한 부여자는 다음과 같은 구성 요소로 이루어집니다.

- 이름: 권한 부여자를 식별하는 고유한 사용자 정의 문자열입니다.
- ARN Lambda 함수: 권한 부여 및 인증 논리를 구현하는 Lambda 함수의 Amazon 리소스 이름(ARN)입니다.
- 토큰 키 이름: 서명 유효성 검사를 수행하기 위해 HTTP 헤더, 쿼리 파라미터 또는 MQTT CONNECT 사용자 이름에서 토큰을 추출하는 데 사용되는 키 이름입니다. 권한 부여자에서 서명이 활성화된 경우 이 값이 필요합니다.
- 서명 비활성화 플래그(선택 사항): 자격 증명에서 서명 요구 사항을 사용하지 않도록 설정할지 여부를 지정하는 부울 값입니다. 이 기능은 MQTT 사용자 이름 및 암호를 사용하는 인증 스키마와 같이 자격 증명에 서명하는 것이 의미가 없는 시나리오에 유용합니다. 기본값은 `false`(으)로 설정되어 있으므로 기본적으로 서명이 활성화됩니다.
- 토큰 서명 공개 키: AWS IoT Core 이(가) 토큰 서명의 유효성을 검사하는 데 사용하는 퍼블릭 키입니다. 최소 길이는 2,048비트입니다. 권한 부여자에서 서명이 활성화된 경우 이 값이 필요합니다.

Lambda는 Lambda 함수가 실행되는 횟수와 함수의 코드가 실행되는 데 걸리는 시간에 대해 요금을 청구합니다. Lambda 요금에 대한 자세한 내용은 Lambda 요금을 참조하세요. Lambda 함수에 대한 자세한 내용은 Lambda 개발자 가이드를 참조하세요.

> **Note**
>
> 서명을 활성화한 상태로 두면 인식할 수 없는 클라이언트가 Lambda를 과도하게 트리거하는 것을 방지할 수 있습니다. 권한 부여자의 로그인을 비활성화하기 전에 이 점을 고려하세요.

> **Note**
>
> 사용자 지정 권한 부여자에 대한 Lambda 함수 제한 시간은 5초입니다.

이 장의 주제:

- Lambda 함수 정의
- 권한 부여자 생성

- Lambda 함수 AWS IoT 호출 권한 부여
- 권한 부여자 테스트
- 사용자 지정 권한 부여자 관리

Lambda 함수 정의

가 권한 부여자를 AWS IoT Core 호출하면 다음 JSON 객체가 포함된 이벤트와 함께 권한 부여자와 연결된 Lambda가 트리거됩니다. 예제 JSON 객체에는 가능한 모든 필드가 포함되어 있습니다. 연결 요청과 관련이 없는 필드는 포함되지 않습니다.

```
{
    "token" :"aToken",
    "signatureVerified": Boolean, // Indicates whether the device gateway has validated
 the signature.
    "protocols": ["tls", "http", "mqtt"], // Indicates which protocols to expect for
 the request.
    "protocolData": {
        "tls" : {
            "serverName": "serverName" // The server name indication (SNI) host_name
 string.
        },
        "http": {
            "headers": {
                "#{name}": "#{value}"
            },
            "queryString": "?#{name}=#{value}"
        },
        "mqtt": {
            "username": "myUserName",
            "password": "myPassword", // A base64-encoded string.
            "clientId": "myClientId" // Included in the event only when the device
 sends the value.
        }
    },
    "connectionMetadata": {
        "id": UUID // The connection ID. You can use this for logging.
    },
}
```

Lambda 함수는 이 정보를 사용하여 들어오는 연결을 인증하고 연결에 허용되는 작업을 결정해야 합니다. 함수는 다음 값을 포함하는 응답을 전송해야 합니다.

- `isAuthenticated`: 요청 인증 여부를 나타내는 부울 값입니다.
- `principalId`: 사용자 지정 권한 부여 요청에서 전송되는 토큰의 식별자 역할을 하는 영숫자 문자열입니다. 값은 1~128자의 영숫자 문자열이어야 하며 정규식(regex) 패턴 (`[a-zA-Z0-9]`) `{1,128}`과 일치해야 합니다. 영숫자가 아닌 특수 문자는 `principalId`에서와 함께 사용할 수 없습니다 AWS IoT Core. 에 영숫자가 아닌 특수 문자가 허용되는 경우 다른 AWS 서비스에 대한 설명서를 참조하세요`principalId`.
- `policyDocuments`: JSON 형식의 AWS IoT Core 정책 문서 목록 AWS IoT Core 정책 생성에 대한 자세한 내용은 섹션을 참조하세요the section called “AWS IoT Core 정책”. 정책 문서의 최대 수는 10개의 정책 문서입니다. 각 정책 문서는 최대 2,048자를 포함할 수 있습니다.
- `disconnectAfterInSeconds`: AWS IoT Core 게이트웨이에 대한 연결의 최대 지속 시간(초)을 지정하는 정수입니다. 최소값은 300초이고 최대값은 86,400초입니다. 기본값은 86,400입니다.

> **Note**
>
> `disconnectAfterInSeconds`(Lambda 함수에서 반환됨) 값은 연결이 설정될 때 설정됩니다. 후속 정책 새로 고침 Lambda 간접 호출 중에는 이 값을 수정할 수 없습니다.

- `refreshAfterInSeconds`: 정책 새로 고침 사이의 간격을 지정하는 정수입니다. 이 간격이 경과하면 AWS IoT Core 은(는) Lambda 함수를 호출하여 정책 새로 고침을 허용합니다. 최소값은 300초이고 최대값은 86,400초입니다.

다음 JSON 객체에는 Lambda 함수가 전송할 수 있는 응답의 예가 들어 있습니다.

```
{ "isAuthenticated":true, //A Boolean that determines whether
client can connect. "principalId": "xxxxxxxx",  //A string that
identifies the connection in logs. "disconnectAfterInSeconds": 86400,
  "refreshAfterInSeconds": 300,   "policyDocuments": [       {
 "Version": "2012-10-17",         "Statement": [           {
    "Action": "iot:Publish",                "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:<your_aws_account_id>:topic/
customauthtesting"            }         ]       }     ] }
```

`policyDocument` 값에는 유효한 AWS IoT Core 정책 문서가 포함되어야 합니다. AWS IoT Core 정책에 대한 자세한 내용은 섹션을 참조하세요the section called “AWS IoT Core 정책”. MQTT over TLS 및 MQTT over WebSockets 연결에서 `refreshAfterInSeconds` 필드 값에 지정된 간격 동안이 정책을 AWS IoT Core 캐시합니다. HTTP 연결의 경우 디바이스가 HTTP 지속 연결(HTTP 연결 유지 또는 HTTP 연결 재사용이라고도 함)을 사용하지 않는 한 모든 권한 부여 요청에 대해 Lambda 함수가 호

출됩니다. 권한 부여자를 구성할 때 캐싱을 사용하도록 선택할 수 있습니다. 이 간격 동안는 Lambda 함수를 다시 트리거하지 않고이 캐시된 정책에 대해 설정된 연결의 작업을 AWS IoT Core 승인합니다. 사용자 지정 인증 중에 오류가 발생하면는 연결을 AWS IoT Core 종료합니다. AWS IoT Core 또한 `disconnectAfterInSeconds`파라미터에 지정된 값보다 오래 열려 있으면 연결을 종료합니다.

다음 JavaScript에는 값이 인 MQTT Connect 메시지에서 암호를 찾고 이름이 인 클라이언트 AWS IoT Core 와 연결하고 동일한 클라이언트 이름이 포함된 주제에 `myClientName` 게시할 수 있는 권한을 부여하는 정책을 `test` 반환하는 샘플 Node.js Lambda 함수가 포함되어 있습니다. 예상된 암호를 찾지 못하면 이 두 작업을 거부하는 정책을 반환합니다.

```
// A simple Lambda function for an authorizer. It demonstrates
// how to parse an MQTT password and generate a response.

exports.handler = function(event, context, callback) {
    var uname = event.protocolData.mqtt.username;
    var pwd = event.protocolData.mqtt.password;
    var buff = new Buffer(pwd, 'base64');
    var passwd = buff.toString('ascii');
    switch (passwd) {
        case 'test':
            callback(null, generateAuthResponse(passwd, 'Allow'));
            break;
        default:
            callback(null, generateAuthResponse(passwd, 'Deny'));
    }
};

// Helper function to generate the authorization response.
var generateAuthResponse = function(token, effect) {
    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'TEST123';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var publishStatement = {};
    var connectStatement = {};
    connectStatement.Action = ["iot:Connect"];
    connectStatement.Effect = effect;
    connectStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:client/
myClientName"];
    publishStatement.Action = ["iot:Publish"];
```

```
    publishStatement.Effect = effect;
    publishStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:topic/telemetry/
myClientName"];
    policyDocument.Statement[0] = connectStatement;
    policyDocument.Statement[1] = publishStatement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 300;

    return authResponse;
}
```

앞의 Lambda 함수는 MQTT Connect 메시지에서 `test`의 예상 암호를 수신하면 다음 JSON 값을 반환합니다. `password` 및 `principalId` 속성의 값은 MQTT Connect 메시지의 값이 됩니다.

```
{
  "password": "password",
  "isAuthenticated": true,
  "principalId": "principalId",
  "policyDocuments": [
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "iot:Connect",
          "Effect": "Allow",
          "Resource": "*"
        },
        {
          "Action": "iot:Publish",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        },
        {
          "Action": "iot:Subscribe",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topicfilter/telemetry/
${iot:ClientId}"
        },
        {
          "Action": "iot:Receive",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
```

```
        }
      ]
    }
  ],
  "disconnectAfterInSeconds": 3600,
  "refreshAfterInSeconds": 300
}
```

권한 부여자 생성

CreateAuthorizer API를 사용하여 권한 부여자를 생성할 수 있습니다. 다음 예에서는 이 명령에 대해 설명합니다.

```
aws iot create-authorizer
--authorizer-name MyAuthorizer
--authorizer-function-arn arn:aws:lambda:us-
west-2:<account_id>:function:MyAuthorizerFunction  //The ARN of the Lambda function.
[--token-key-name MyAuthorizerToken //The key used to extract the token from headers.
[--token-signing-public-keys FirstKey=
 "-----BEGIN PUBLIC KEY-----
  [...insert your public key here...]
  -----END PUBLIC KEY-----"
[--status ACTIVE]
[--tags <value>]
[--signing-disabled | --no-signing-disabled]
```

`signing-disabled` 파라미터를 사용하여 권한 부여자를 호출할 때마다 서명 유효성 검사를 옵트아웃합니다. 필요한 경우가 아니면 서명을 비활성화하지 않는 것이 좋습니다. 서명 유효성 검사는 알 수 없는 디바이스에서 Lambda 함수를 과도하게 호출하지 않도록 보호합니다. 생성한 후에는 권한 부여자의 `signing-disabled` 상태를 업데이트 할 수 없습니다. 이 동작을 변경하려면 `signing-disabled` 파라미터에 대해 또 다른 값의 사용자 지정 권한 부여자를 생성해야 합니다.

`tokenKeyName` 및 `tokenSigningPublicKeys` 파라미터의 값은 서명을 사용하지 않도록 설정한 경우 선택 사항입니다. 서명이 활성화된 경우 필수 값입니다.

Lambda 함수와 사용자 지정 권한 부여자를 생성한 후에는 사용자를 대신하여 함수를 호출할 수 있는 AWS IoT Core 서비스 권한을 명시적으로 부여해야 합니다. 다음 명령을 사용하여이 작업을 수행할 수 있습니다.

> **Note**
>
> 기본 IoT 엔드포인트는 Lambda 함수와 함께 사용자 지정 권한 부여자를 사용하도록 지원하지 않을 수 있습니다. 그 대신, 도메인 구성을 사용하여 새 엔드포인트를 정의한 다음, 사용자 지정 권한 부여자에 대해 해당 엔드포인트를 지정할 수 있습니다.

```
aws lambda add-permission --function-name <lambda_function_name>
--principal iot.amazonaws.com --source-arn <authorizer_arn>
--statement-id Id-123 --action "lambda:InvokeFunction"
```

Lambda 함수 AWS IoT 호출 권한 부여

이 섹션에서는 방금 생성한 사용자 지정 권한 부여자 리소스에 Lambda 함수를 실행할 수 있는 권한을 부여합니다. 권한을 부여하려면 add-permission CLI 명령을 사용할 수 있습니다.

를 사용하여 Lambda 함수에 권한 부여 AWS CLI

1. 값을 삽입한 후 다음 명령을 입력합니다. 참고: `statement-id` 값은 고유해야 합니다. *Id-1234*을(를) 정확한 값으로 바꿉니다. 그렇지 않으면 `ResourceConflictException` 오류가 발생할 수 있습니다.

   ```
   aws lambda add-permission  \
   --function-name "custom-auth-function" \
   --principal "iot.amazonaws.com" \
   --action "lambda:InvokeFunction" \
   --statement-id "Id-1234" \
   --source-arn authorizerArn
   ```

2. 명령이 성공하면 이 예제와 같은 권한 문이 반환됩니다. 다음 단원을 계속하여 사용자 지정 권한 부여자를 테스트할 수 있습니다.

   ```
   {
       "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
   \":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
   \",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
   auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
   \"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
   }
   ```

명령이 성공하지 못하면 이 예제와 같은 오류가 반환됩니다. 계속하기 전에 오류를 검토하고 수정해야 합니다.

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

권한 부여자 테스트

TestInvokeAuthorizer API를 사용하여 권한 부여자의 호출 및 반환 값을 테스트합니다. 이 API를 사용하면 권한 부여자에서 프로토콜 메타데이터를 지정하고 서명 유효성 검사를 테스트할 수 있습니다.

다음 탭은를 사용하여 권한 부여자를 테스트 AWS CLI 하는 방법을 보여줍니다.

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER \
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER ^
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER `
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

`token-signature` 파라미터의 값은 서명된 토큰입니다. 이 값을 얻는 방법을 알아보려면 the section called “토큰에 서명하기” 단원을 참조하세요.

권한 부여자가 사용자 이름과 암호를 얻은 경우 `--mqtt-context` 파라미터를 사용하여 이 정보를 전달할 수 있습니다. 다음 탭은 `TestInvokeAuthorizer` API를 사용하여 사용자 이름, 암호 및 클라이언트 이름이 포함된 JSON 객체를 사용자 지정 권한 부여자에게 전송하는 방법을 보여줍니다.

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  \
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  ^
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  `
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

암호는 base64 인코딩해야 합니다. 다음 예제에서는 Unix 계열 환경에서 암호를 인코딩하는 방법을 보여 줍니다.

```
echo -n PASSWORD | base64
```

사용자 지정 권한 부여자 관리

다음 API를 사용하여 권한 부여자를 관리할 수 있습니다.

- ListAuthorizers: 계정에 속한 모든 권한 부여자를 표시합니다.
- DescribeAuthorizer: 지정한 권한 부여자의 속성을 표시합니다. 이러한 값에는 생성 날짜, 마지막 수정 날짜 및 기타 속성이 포함됩니다.
- SetDefaultAuthorizer: AWS IoT Core 데이터 엔드포인트의 기본 권한 부여자를 지정합니다. 디바이스가 AWS IoT Core 자격 증명을 전달하지 않고 권한 부여자를 지정하지 않은 경우이 권한 부여자를 AWS IoT Core 사용합니다. 자격 AWS IoT Core 증명 사용에 대한 자세한 내용은 섹션을 참조하세요the section called “클라이언트 인증”.
- UpdateAuthorizer: 지정된 권한 부여자의 상태, 토큰 키 이름 또는 퍼블릭 키를 변경합니다.
- DeleteAuthorizer: 지정된 권한 부여자를 삭제합니다.

> **Note**
>
> 권한 부여자의 서명 요구 사항은 업데이트할 수 없습니다. 즉, 필요한 기존 권한 부여자에서 로그인을 비활성화할 수 없습니다. 또한 필요하지 않은 기존 권한 부여자에는 로그인을 요구할 수 없습니다.

X.509 클라이언트 인증서를 사용한 사용자 지정 인증

디바이스를 연결할 때 AWS IoT Core여러 인증 유형을 사용할 수 있습니다. 클라이언트 및 디바이스 연결을 인증하는 데 사용할 수 있는 X.509 클라이언트 인증서를 사용하거나 사용자 지정 권한 부여자를 정의하여 자체 클라이언트 인증 및 권한 부여 로직을 관리할 수 있습니다. 이 주제에서는 X.509 클라이언트 인증서와 함께 사용자 지정 인증을 사용하는 방법을 다룹니다.

X.509 인증서를 사용하여 디바이스를 이미 인증하고 추가 검증 및 사용자 지정 인증을 수행하려는 경우 X.509 인증서를 사용하여 사용자 지정 인증을 사용하는 것이 도움이 될 수 있습니다. 예를 들어 X.509 클라이언트 인증서에 일련 번호와 같은 디바이스의 데이터를 저장하는 경우 X.509 클라이언트 인증서를 AWS IoT Core 인증한 후 사용자 지정 권한 부여자를 사용하여 인증서의 CommonName 필드에 저장된 정보를 기반으로 특정 디바이스를 식별할 수 있습니다. X.509 인증서와 함께 사용자 지정 인증을 사용하면 디바이스를에 연결할 때 디바이스 보안 관리를 개선하고 인증 및 권한 부여 로직을 보다 유연하게 관리할 수 있습니다. AWS IoT Core 는 X.509 인증서 및 MQTT 프로토콜과 HTTPS 프로토콜 모두에서 작동하는 사용자 지정 권한 부여자 인증 유형을 사용하여 X.509 인증서를 사용한 사용자 지정 인증을 AWS IoT Core 지원합니다. AWS IoT Core 디바이스 엔드포인트가 지원하는 인증 유형 및 애플리케이션 프로토콜에 대한 자세한 내용은 Device communication protocols 섹션을 참조하세요.

> **Note**
>
> X.509 클라이언트 인증서를 사용한 사용자 지정 인증은 AWS GovCloud (US) 리전에서 지원되지 않습니다.

> **Important**
>
> 도메인 구성 을 사용하여 생성된 엔드포인트를 사용해야 합니다. 또한 클라이언트는 연결할 때 서버 이름 표시(SNI) 확장을 제공해야 합니다 AWS IoT Core.

X.509 클라이언트 인증서로 사용자 지정 인증을 사용하여 디바이스를 인증하는 프로세스는 다음 단계로 구성됩니다.

- 1단계:에 X.509 클라이언트 인증서 등록 AWS IoT Core
- 2단계: Lambda 함수 생성
- 3단계: 사용자 지정 권한 부여자 생성
- 4단계: 도메인 구성에서 인증 유형 및 애플리케이션 프로토콜 설정

1단계:에 X.509 클라이언트 인증서 등록 AWS IoT Core

아직 등록하지 않은 경우에 X.509 클라이언트 인증서를 등록하고 활성화합니다 AWS IoT Core. 그렇지 않은 경우 다음 단계로 건너뜁니다.

에 클라이언트 인증서를 등록하고 활성화하려면 다음 단계를 AWS IoT Core따릅니다.

1. 를 사용하여 직접 클라이언트 인증서를 생성하는 AWS IoT 경우. 이러한 클라이언트 인증서는에 자동으로 등록됩니다 AWS IoT Core.
2. 자체 클라이언트 인증서를 생성하는 경우 다음 지침에 따라 인증서를 등록합니다 AWS IoT Core.
3. 클라이언트 인증서를 활성화하려면 다음 지침을 따르세요.

2단계: Lambda 함수 생성

AWS IoT Core 는 사용자 지정 권한 부여자를 사용하여 사용자 지정 인증 및 권한 부여 체계를 구현합니다. 사용자 지정 권한 부여자는 디바이스가 인증되었는지 여부와 디바이스가 수행할 수 있는 작업을 결정하는 Lambda 함수와 연결됩니다. 디바이스가에 연결되면 권한 부여자 이름 및 연결된 Lambda 함수를 포함한 권한 부여자 세부 정보를 AWS IoT Core AWS IoT Core 검색하고 Lambda 함수를 호출합니다. Lambda 함수는 디바이스의 X.509 클라이언트 인증서 데이터가 포함된 JSON 객체가 포함된 이벤트를 수신합니다. Lambda 함수는 이 이벤트 JSON 객체를 사용하여 인증 요청을 평가하고 수행할 작업을 결정한 다음 응답을 다시 보냅니다.

Lambda 함수 이벤트 예제

다음 예제 JSON 객체에는 포함할 수 있는 모든 필드가 포함되어 있습니다. 실제 JSON 객체에는 특정 연결 요청과 관련된 필드만 포함됩니다.

```
{
 "token": "aToken",
 "signatureVerified": true,
 "protocols": [
```

```
   "tls",
   "mqtt"
  ],
  "protocolData": {
   "tls": {
    "serverName": "serverName",
    "x509CertificatePem": "x509CertificatePem",
    "principalId": "principalId"
   },
   "mqtt": {
    "clientId": "myClientId",
                      "username": "myUserName",
                      "password": "myPassword"
   }
  },
  "connectionMetadata": {
   "id": "UUID"
  }
}
```

`signatureVerified`

권한 부여자의 Lambda 함수를 호출하기 전에 권한 부여자에 구성된 토큰 서명이 확인되는지 여부를 나타내는 부울 값입니다. 권한 부여자가 토큰 서명을 비활성화하도록 구성된 경우 이 필드는 거짓입니다.

`protocols`

요청에 대해 예상할 프로토콜이 포함된 배열입니다.

`protocolData`

연결에 사용되는 프로토콜의 정보가 포함된 객체입니다. 인증, 권한 부여 등에 유용할 수 있는 프로토콜별 세부 정보를 제공합니다.

`tls` - 이 객체는 TLS(Transport Layer Security) 프로토콜과 관련된 정보를 보유합니다.

- `serverName` - 서버 이름 표시(SNI) 호스트 이름 스트링입니다. AWS IoT Core 는 디바이스가 SNI 확장을 Transport Layer Security(TLS) 프로토콜로 전송하고 `host_name` 필드에 전체 엔드포인트 주소를 제공하도록 요구합니다.
- `x509CertificatePem` - TLS 연결에서 클라이언트 인증에 사용되는 PEM 형식의 X.509 인증서입니다.
- `principalId` - TLS 연결에서 클라이언트와 연결된 위탁자 식별자입니다.

`mqtt` - 이 객체에는 MQTT 프로토콜과 관련된 정보가 들어 있습니다.

- `clientId` - 디바이스가 이 값을 보내는 경우에만 문자열을 포함해야 합니다.
- `username` - MQTT Connect 패킷에 제공된 사용자 이름입니다.
- `password` - MQTT Connect 패킷에 제공된 암호입니다.

`connectionMetadata`

연결의 메타데이터입니다.

`id` - 로깅 및 문제 해결에 사용할 수 있는 연결 ID입니다.

> **Note**
>
> 이 경우 JSON 객체 `x509CertificatePem` 및 `principalId`는 요청의 두 가지 새 필드입니다. `principalId`의 값은 `certificateId`의 값과 동일합니다. 자세한 내용은 인증서를 참조하세요.

Lambda 함수 응답 예제

Lambda 함수는 이벤트 JSON 객체의 정보를 사용하여 수신 연결을 인증하고 연결에서 허용되는 작업을 결정해야 합니다.

다음 JSON 객체에는 Lambda 함수가 전송할 수 있는 응답의 예가 들어 있습니다.

```
{
 "isAuthenticated": true,
 "principalId": "xxxxxxxx",
 "disconnectAfterInSeconds": 86400,
 "refreshAfterInSeconds": 300,
 "policyDocuments": [
  {
   "Version": "2012-10-17",
   "Statement": [
    {
     "Effect": "Allow",
     "Action": "iot:Publish",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topic/customauthtesting"
    }
   ]
  }
```

```
 ]
}
```

이 예제에서는 이 함수가 다음 값을 포함하는 응답을 보내야 합니다.

`isAuthenticated`

요청 인증 여부를 나타내는 부울 값입니다.

`principalId`

사용자 지정 권한 부여 요청에서 전송되는 토큰의 식별자 역할을 하는 영숫자 문자열입니다. 값은 반드시 1~128개 글자의 영숫자 문자열이어야 합니다. 로그에서 연결을 식별합니다. `principalId` 값은 이벤트 JSON 객체(즉, X.509 인증서의 certificateId)의 `principalId` 값과 동일해야 합니다.

`policyDocuments`

JSON 형식의 AWS IoT Core 정책 문서 목록입니다. 값은 선택 사항이며 사물 정책 변수 및 인증서 정책 변수를 지원합니다. 정책 문서의 최대 수는 10개입니다. 각 정책 문서는 최대 2,048자를 포함할 수 있습니다. 클라이언트 인증서와 Lambda 함수에 여러 정책이 연결되어 있는 경우 권한은 모든 정책의 모음입니다. AWS IoT Core 정책 생성에 대한 자세한 내용은 정책을 참조하세요.

`disconnectAfterInSeconds`

AWS IoT Core 게이트웨이에 대한 연결의 최대 지속 시간(초)을 지정하는 정수입니다. 최솟값은 300초이고 최댓값은 86,400초입니다. `disconnectAfterInSeconds`는 연결 수명 동안 사용되며 연속 정책 새로 고침 시 새로 고쳐지지 않습니다.

`refreshAfterInSeconds`

정책 새로 고침 사이의 간격을 지정하는 정수입니다. 이 간격이 경과하면가 Lambda 함수를 AWS IoT Core 호출하여 정책 새로 고침을 허용합니다. 최소값은 300초이고 최대값은 86,400초입니다.

Lambda 함수 예제

다음은 샘플 Node.js Lambda 함수입니다. 이 함수는 클라이언트의 X.509 인증서를 검사하고 일련 번호, 지문 및 주제 이름과 같은 관련 정보를 추출합니다. 추출된 정보가 예상 값과 일치하면 클라이언트에 연결할 수 있는 액세스 권한이 부여됩니다. 이 메커니즘은 유효한 인증서가 있는 승인된 클라이언트만 연결을 설정할 수 있도록 합니다.

```
const crypto = require('crypto');

exports.handler = async (event) => {
```

```
    // Extract the certificate PEM from the event
    const certPem = event.protocolData.tls.x509CertificatePem;

    // Parse the certificate using Node's crypto module
    const cert = new crypto.X509Certificate(certPem);

    var effect = "Deny";
    // Allow permissions only for a particular certificate serial, fingerprint, and
 subject
    if (cert.serialNumber === "7F8D2E4B9C1A5036DE8F7C4B2A91E5D80463BC9A1257" // This is
 a random serial
        && cert.fingerprint ===
 "F2:9A:C4:1D:B5:E7:08:3F:6B:D0:4E:92:A7:C1:5B:8D:16:0F:E3:7A" // This is a random
 fingerprint
        && cert.subject === "allow.example.com") {
      effect = "Allow";
    }

    return generateAuthResponse(event.protocolData.tls.principalId, effect);
};


// Helper function to generate the authorization response.
function generateAuthResponse(principalId, effect) {
    const authResponse = {
        isAuthenticated: true,
        principalId,
        disconnectAfterInSeconds: 3600,
        refreshAfterInSeconds: 300,
        policyDocuments: [
          {
            Version: "2012-10-17",
            Statement: [
              {
                Action: ["iot:Connect"],
                Effect: effect,
                Resource: [
                  "arn:aws:iot:us-east-1:123456789012:client/myClientName"
                ]
              },
              {
                Action: ["iot:Publish"],
                Effect: effect,
                Resource: [
```

```
                "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
              ]
            },
            {
              Action: ["iot:Subscribe"],
              Effect: effect,
              Resource: [
                 "arn:aws:iot:us-east-1:123456789012:topicfilter/telemetry/
myClientName"
              ]
            },
            {
              Action: ["iot:Receive"],
              Effect: effect,
              Resource: [
                 "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
              ]
            }
          ]
        }
      ]
    };

  return authResponse;
}
```

이전 Lambda 함수는 예상 직렬, 지문 및 제목이 포함된 인증서를 수신하면 다음 JSON을 반환합니다. `x509CertificatePem`의 값은 TLS 핸드셰이크에 제공된 클라이언트 인증서입니다. 자세한 내용은 Defining your Lambda function을 참조하세요.

```
{
 "isAuthenticated": true,
 "principalId": "principalId in the event JSON object",
 "policyDocuments": [
  {
   "Version": "2012-10-17",
   "Statement": [
    {
     "Action": "iot:Connect",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:client/myClientName"
    },
    {
```

```
      "Action": "iot:Publish",
      "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
     },
     {
      "Action": "iot:Subscribe",
      "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/telemetry/
myClientName"
     },
     {
      "Action": "iot:Receive",
      "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
     }
    ]
   }
  ],
  "disconnectAfterInSeconds": 3600,
  "refreshAfterInSeconds": 300
}
```

3단계: 사용자 지정 권한 부여자 생성

Lambda 함수 를 정의한 후 사용자 지정 권한 부여자를 생성하여 자체 클라이언트 인증 및 권한 부여 로직을 관리합니다. 3단계: 사용자 지정 권한 부여자 리소스 생성 및 권한 부여의 세부 지침을 따를 수 있습니다. 자세한 내용은 권한 부여자 생성을 참조하세요.

사용자 지정 권한 부여자를 생성하는 과정에서 Lambda 함수를 생성한 후 이를 호출할 수 있는 AWS IoT 권한을 부여해야 합니다. 자세한 지침은 Lambda 함수 호출 권한 부여 AWS IoT 를 참조하세요.

4단계: 도메인 구성에서 인증 유형 및 애플리케이션 프로토콜 설정

X.509 클라이언트 인증서로 사용자 지정 인증을 사용하여 디바이스를 인증하려면 도메인 구성에서 인증 유형 및 애플리케이션 프로토콜을 설정하고 SNI 확장을 전송해야 합니다. `authenticationType`의 값은 `CUSTOM_AUTH_X509`이어야 하며, `applicationProtocol`의 값은 `SECURE_MQTT` 또는 `HTTPS`일 수 있습니다.

도메인 구성에서 인증 유형 및 애플리케이션 프로토콜 설정(CLI)

도메인 구성이 없는 경우 create-domain-configuration 명령을 사용하여 생성합니다. `authenticationType`의 값은 `CUSTOM_AUTH_X509`이어야 하며, `applicationProtocol`의 값은 `SECURE_MQTT` 또는 `HTTPS`일 수 있습니다.

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
    }'
```

도메인 구성이 이미 있는 경우 필요한 경우 update-domain-configuration 명령 업데이트 authenticationType 및 applicationProtocol를 사용합니다. 기본 엔드포인트iot:Data-ATS()에서는 인증 유형 또는 프로토콜을 변경할 수 없습니다.

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
    }'
```

`domain-configuration-name`

도메인 구성의 이름입니다.

`authentication-type`

도메인 구성의 인증 유형입니다. 자세한 내용은 인증 유형 선택을 참조하세요.

`application-protocol`

AWS IoT Core와 통신하는 데 사용하는 디바이스의 애플리케이션 프로토콜입니다. 자세한 내용은 애플리케이션 프로토콜 선택을 참조하세요.

`--authorizer-config`

도메인 구성에서 권한 부여자 구성을 지정하는 객체입니다.

`defaultAuthorizerName`

도메인 구성에 대한 권한 부여자의 이름입니다.

자세한 내용은 AWS IoT API 참조의 CreateDomainConfiguration 및 UpdateDomainConfiguration을 참조하세요. 도메인 구성에 대한 자세한 내용은 Domain configurations를 참조하세요.

사용자 지정 인증을 사용하여 AWS IoT Core 에 연결

디바이스는 AWS IoT Core 에서 디바이스 메시징을 AWS IoT Core 지원하는 모든 프로토콜에서 사용자 지정 인증을 사용하여에 연결할 수 있습니다. 지원되는 통신 프로토콜에 대한 자세한 내용은 the section called “디바이스 통신 프로토콜” 단원을 참조하세요. 권한 부여자 Lambda 함수에 전달하는 연결 데이터는 사용하는 프로토콜에 따라 다릅니다. 권한 부여자 Lambda 함수를 만드는 방법에 대한 자세한 내용은 the section called “Lambda 함수 정의” 단원을 참조하세요. 다음 단원에서는 지원되는 각 프로토콜을 사용하여 인증에 연결하는 방법을 설명합니다.

HTTPS

HTTP 게시 API를 사용하여 AWS IoT Core 로 데이터를 전송하는 디바이스는 HTTP POST 요청의 요청 헤더 또는 쿼리 파라미터를 통해 자격 증명을 전달할 수 있습니다. 디바이스는 `x-amz-customauthorizer-name` 헤더 또는 쿼리 파라미터를 사용하여 호출할 권한 부여자를 지정할 수 있습니다. 권한 부여자에서 토큰 서명을 활성화한 경우 요청 헤더 또는 쿼리 파라미터 중 하나에서 *`token-key-name`* 및 `x-amz-customauthorizer-signature`을(를) 전달해야 합니다. 참고: 브라우저에서 JavaScript를 사용할 경우 *`token-signature`* 값은 URL로 인코딩해야 합니다.

> **Note**
>
> HTTPS 프로토콜의 고객 권한 부여자는 게시 작업만 지원합니다. HTTPS 프로토콜에 대한 자세한 내용은 the section called “디바이스 통신 프로토콜” 섹션을 참조하세요.

다음 예제 요청은 요청 헤더와 쿼리 파라미터 모두에서 이러한 파라미터를 전달하는 방법을 보여 줍니다.

```
//Passing credentials via headers
POST /topics/topic?qos=qos HTTP/1.1
Host: your-endpoint
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
x-amz-customauthorizer-name: authorizer-name

//Passing credentials via query parameters
POST /topics/topic?qos=qos&x-amz-customauthorizer-signature=token-signature&token-key-name=token-value HTTP/1.1
```

MQTT

MQTT 연결을 사용하여 AWS IoT Core 에 연결하는 디바이스는 MQTT 메시지의 `username` 및 `password` 필드를 통해 자격 증명을 전달할 수 있습니다. `username` 값에는 추가 값(토큰, 서명 및 권한 부여자 이름 포함)을 권한 부여자에게 전달하는 쿼리 문자열이 선택적으로 포함될 수도 있습니다. `username` 및 `password` 값 대신 토큰 기반 인증 체계를 사용하려는 경우 이 쿼리 문자열을 사용할 수 있습니다.

> **Note**
>
> 암호 필드의 데이터는 base64로 인코딩됩니다 AWS IoT Core. 디코딩은 Lambda 함수로 해야 합니다.

다음 예제는 토큰 및 서명을 지정하는 추가 파라미터가 포함된 `username` 문자열을 포함합니다.

```
username?x-amz-customauthorizer-name=authorizer-name&x-amz-customauthorizer-
signature=token-signature&token-key-name=token-value
```

권한 부여자를 호출하려면 MQTT 및 사용자 지정 인증을 사용하여 AWS IoT Core 에 연결하는 디바이스가 포트 443에서 연결되어야 합니다. 또한 값이 인 Application Layer Protocol Negotiation(ALPN) TLS 확장`mqtt`과 AWS IoT Core 데이터 엔드포인트의 호스트 이름이 있는 Server Name Indication(SNI) 확장을 전달해야 합니다. 잠재적 오류를 방지하기 위해 `x-amz-customauthorizer-signature` 값은 URL 인코딩을 해야 합니다. 또한 `x-amz-customauthorizer-name` 및 `token-key-name` 값을 URL 인코딩하는 것을 적극 권장합니다. 이러한 값에 대한 자세한 정보는 the section called “디바이스 통신 프로토콜” 섹션을 참조하세요. V2 AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트는 이 두 확장을 모두 구성할 수 있습니다.

WebSocket을 통한 MQTT

MQTT over WebSockets AWS IoT Core 을 사용하여에 연결하는 디바이스는 다음 두 가지 방법 중 하나로 자격 증명을 전달할 수 있습니다.

- WebSockets 연결 설정을 위해 HTTP 업그레이드 요청의 요청 헤더 또는 쿼리 파라미터를 통해.
- MQTT 연결 메시지의 `username` 및 `password` 필드를 통해.

MQTT 연결 메시지를 통해 자격 증명을 전달하면 ALPN 및 SNI TLS 확장이 필요합니다. 이러한 확장에 대한 자세한 내용은 the section called “MQTT” 단원을 참조하세요. 다음 예에서는 HTTP 업그레이드 요청을 통해 자격 증명을 전달하는 방법을 보여 줍니다.

```
GET /mqtt HTTP/1.1
Host: your-endpoint
Upgrade: WebSocket
Connection: Upgrade
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
sec-WebSocket-Key: any random base64 value
sec-websocket-protocol: mqtt
sec-WebSocket-Version: websocket version
```

토큰에 서명하기

`create-authorizer` 호출에서 사용한 퍼블릭-프라이빗 키 페어의 프라이빗 키를 사용하여 토큰에 서명해야 합니다. 다음 예제에서는 UNIX 계열 명령과 JavaScript를 사용하여 토큰 서명을 생성하는 방법을 보여줍니다. 이들은 SHA-256 해시 알고리즘을 사용하여 서명을 인코딩합니다.

Command line

```
echo -n TOKEN_VALUE | openssl dgst -sha256 -sign PEM encoded RSA private key |
 openssl base64
```

JavaScript

```
const crypto = require('crypto')

const key = "PEM encoded RSA private key"

const k = crypto.createPrivateKey(key)
let sign = crypto.createSign('SHA256')
sign.write(t)
sign.end()
const s = sign.sign(k, 'base64')
```

Lambda 권한 부여자 문제 해결

이 주제에서는 사용자 지정 인증 워크플로에서 문제를 일으킬 수 있는 일반적인 문제와 이를 해결하기 위한 단계를 안내합니다. 문제를 가장 효과적으로 해결하려면에 대해 CloudWatch 로그

를 활성화 AWS IoT Core 하고 로그 수준을 DEBUG로 설정합니다. AWS IoT Core 콘솔(https://console.aws.amazon.com/iot/)에서 CloudWatch 로그를 활성화할 수 있습니다. AWS IoT Core에 대해 로그를 활성화 및 구성하는 자세한 내용은 the section called “AWS IoT 로깅 구성” 단원을 참조하세요.

> **Note**
>
> 로그 수준을 장기간 디버그 상태로 두면 CloudWatch에서 많은 양의 로깅 데이터를 저장할 것이고, 이렇게 하면 CloudWatch 요금이 증가할 수 있습니다. 특정 사물 그룹의 디바이스에 대해서만 상세도를 높이도록 리소스 기반 로깅을 사용하는 것이 좋습니다. 리소스 기반 로깅에 대한 자세한 내용은 the section called “AWS IoT 로깅 구성” 단원을 참조하세요. 또한 문제 해결이 완료되면 로그 수준의 상세도를 낮추세요.

문제 해결을 시작하기 전에 사용자 지정 인증 프로세스를 자세히 볼 수 있도록 the section called “사용자 지정 인증 워크플로 이해”을(를) 검토합니다. 이렇게 하면 문제의 원인을 찾을 위치를 파악하는 데 도움이 됩니다.

이 주제에서는 다음과 같은 두 가지 조사 영역에 대해 설명합니다.

- 권한 부여자의 Lambda 함수와 관련된 문제.
- 디바이스와 관련된 문제.

권한 부여자의 Lambda 함수에서 문제 확인

장치의 연결 시도가 Lambda 함수를 호출하는지 확인하려면 다음 단계를 수행하세요.

1. 권한 부여자와 연결된 Lambda 함수를 확인합니다.

   DescribeAuthorizer API를 호출하거나 AWS IoT Core 콘솔의 보안 단원에서 원하는 권한 부여자를 클릭하여 수행합니다.
2. Lambda 함수의 호출 지표를 확인합니다. 이렇게 하려면 다음 단계를 수행하세요.

   a. AWS Lambda 콘솔(https://console.aws.amazon.com/lambda/)을 열고 권한 부여자와 연결된 함수를 선택합니다.

   b. 모니터링 탭을 선택하고 문제와 관련된 시간 프레임에 대한 지표를 봅니다.
3. 호출이 표시되지 않으면 AWS IoT Core 에 Lambda 함수를 호출할 권한이 있는지 확인합니다. 호출이 표시되면 다음 단계로 건너뜁니다. Lambda 함수에 필요한 권한이 있는지 확인하려면 다음 단계를 수행하세요.

a. AWS Lambda 콘솔에서 함수에 대한 권한 탭을 선택합니다.

b. 페이지 하단에서 리소스 기반 정책 단원을 찾습니다. Lambda 함수에 필요한 권한이 있는 경우 정책은 다음 예와 같습니다.

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Id123",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
east-1:111111111111:function:FunctionName",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:iot:us-east-1:111111111111:authorizer/
AuthorizerName"
        },
        "StringEquals": {
          "AWS:SourceAccount": "111111111111"
        }
      }
    }
  ]
}
```

c. 이 정책은 AWS IoT Core 보안 주체에게 함수에 대한 `InvokeFunction` 권한을 부여합니다. 표시되지 않으면 AddPermission API를 사용하여 추가해야 합니다. 다음 예에서는 AWS CLI을(를) 사용하여 이 작업을 수행하는 방법을 보여줍니다.

```
aws lambda add-permission --function-name FunctionName --principal
 iot.amazonaws.com --source-arn AuthorizerARn --statement-id Id-123 --action
 "lambda:InvokeFunction"
```

4. 호출이 표시되면 오류가 없는지 확인합니다. 오류는 Lambda 함수가에서 AWS IoT Core 전송한 연결 이벤트를 제대로 처리하지 못하고 있음을 나타낼 수 있습니다.

Lambda 함수에서 이벤트를 처리하는 방법에 대한 자세한 내용은 the section called “Lambda 함수 정의” 단원을 참조하세요. AWS Lambda 콘솔(https://console.aws.amazon.com/lambda/)의 테스트 기능을 사용하여 함수의 테스트 값을 하드 코딩하여 함수가 이벤트를 올바르게 처리하고 있는지 확인할 수 있습니다.

5. 오류 없이 호출이 표시되지만 디바이스가 연결할 수 없거나 메시지를 게시, 구독 및 수신할 수 없는 경우 Lambda 함수가 반환하는 정책이 디바이스가 수행하려는 작업에 대한 권한을 부여하지 않는 문제일 수 있습니다. 함수가 반환하는 정책에 문제가 있는지 확인하려면 다음 단계를 수행하세요.

   a. Amazon CloudWatch Logs Insights 쿼리를 사용하여 짧은 기간 동안 로그를 스캔하여 실패를 확인합니다. 다음 예제 쿼리는 타임스탬프별로 이벤트를 정렬하고 실패를 찾습니다.

   ```
   display clientId, eventType, status, @timestamp | sort @timestamp desc | filter
    status = "Failure"
   ```

   b. Lambda 함수를 업데이트하여 반환되는 데이터와 함수 AWS IoT Core 를 트리거하는 이벤트를 로깅합니다. 이러한 로그를 사용하여 함수가 만드는 정책을 검사할 수 있습니다.

6. 오류 없이 호출이 표시되지만 디바이스가 연결할 수 없거나 메시지를 게시, 구독 및 수신할 수 없는 경우 Lambda 함수가 제한 시간을 초과했기 때문일 수 있습니다. 사용자 지정 권한 부여자에 대한 Lambda 함수 제한 시간은 5초입니다. CloudWatch 로그 또는 지표에서 함수 기간을 확인할 수 있습니다.

디바이스 문제 조사

Lambda 함수를 호출하거나 함수가 반환하는 정책에 문제가 없으면 디바이스의 연결 시도에 문제가 있는지 확인합니다. 잘못된 연결 요청으로 인해가 권한 부여자를 트리거 AWS IoT Core 하지 않을 수 있습니다. TLS 및 애플리케이션 계층 모두에서 연결 문제가 발생할 수 있습니다.

가능한 TLS 계층 문제:

- 고객은 모든 사용자 지정 인증 요청에서 호스트 이름 헤더(HTTP, WebSockets을 통한 MQTT) 또는 서버 이름 표시 TLS 확장(HTTP, WebSockets을 통한 MQTT, MQTT)을 전달해야 합니다. 두 경우 모두 전달된 값은 계정의 AWS IoT Core 데이터 엔드포인트 중 하나와 일치해야 합니다. 이러한 엔드포인트는 다음 CLI 명령을 수행할 때 반환되는 엔드포인트입니다.
  - `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

  - `aws iot describe-endpoint --endpoint-type iot:Data`(레거시 VeriSign 엔드포인트용)
- MQTT 연결에서 사용자 지정 인증을 사용하는 디바이스가 `mqtt` 값의 ALPN(Application Layer Protocol Negotiation) TLS 확장을 전달해야 합니다.
- 사용자 지정 인증은 현재 포트 443에서만 사용할 수 있습니다.

가능한 애플리케이션 계층 문제:

- 서명이 활성화된 경우(`signingDisabled` 필드가 false인 경우) 다음 서명 문제를 찾습니다.
  - 토큰 서명을 `x-amz-customauthorizer-signature` 헤더 또는 쿼리 문자열 파라미터로 전달하고 있는지 확인합니다.
  - 서비스가 토큰이 아닌 다른 값에 서명하고 있지 않은지 확인합니다.
  - 권한 지정자의 `token-key-name` 필드에 지정한 헤더 또는 쿼리 파라미터로 토큰을 전달하는지 확인합니다.
- `x-amz-customauthorizer-name` 헤더 또는 쿼리 문자열 파라미터로 전달한 권한 부여자 이름이 유효하거나 계정에 대해 기본 권한 부여자가 정의되어 있어야 합니다.

권한 부여

권한 부여는 인증된 자격 증명에 권한을 부여하는 프로세스입니다. AWS IoT Core 및 IAM 정책을 AWS IoT Core 사용하여에 권한을 부여합니다. 이 주제에서는 AWS IoT Core 정책을 다룹니다. IAM 정책에 대한 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT 및 AWS IoT 에서 IAM을 사용하는 방법 단원을 참조하세요.

AWS IoT Core 정책은 인증된 자격 증명이 수행할 수 있는 작업을 결정합니다. 인증된 자격 증명은 디바이스, 모바일 애플리케이션, 웹 애플리케이션 및 데스크톱 애플리케이션이 사용합니다. 인증된 자격 증명은 AWS IoT Core CLI 명령을 입력하는 사용자일 수도 있습니다. 자격 증명은 해당 AWS IoT Core 작업에 대한 권한을 부여하는 정책이 있는 경우에만 작업을 실행할 수 있습니다.

AWS IoT Core 정책 및 IAM 정책은 모두 자격 증명(보안 주체라고도 함)이 수행할 수 있는 작업을 제어하는 AWS IoT Core 데와 함께 사용됩니다. 사용하는 정책 유형은 인증에 사용하는 자격 증명 유형에 따라 다릅니다 AWS IoT Core.

AWS IoT Core 작업은 두 그룹으로 나뉩니다.

- 컨트롤 플레인 API는 인증서, 사물, 규칙 등을 생성 또는 업데이트하는 것과 같은 관리 작업을 수행할 수 있습니다.
- 데이터 영역 API를 사용하면 데이터를 로 보내고 받을 수 있습니다 AWS IoT Core.

사용하는 정책 유형은 제어 플레인 API 또는 데이터 플레인 API를 사용하는지에 따라 달라집니다.

다음 표에는 자격 증명, 사용되는 프로토콜, 인증에 사용될 수 있는 정책 유형이 나와 있습니다.

AWS IoT Core 데이터 영역 API 및 정책 유형

프로토콜 및 인증 메커니즘	SDK	자격 증명 유형	정책 유형		
MQTT over TLS/TCP, TLS 상호 인증(포트 8883 또는 443)[†])	AWS IoT 디바이스 SDK	X.509 인증서	AWS IoT Core 정책		
HTTPS/WebSocket을 통한 MQTT, AWS SigV4 인증(포트 443)	AWS 모바일 SDK	인증된 Amazon Cognito 자격 증명	IAM 및 AWS IoT Core 정책		
		인증되지 않은 Amazon Cognito 자격 증명	IAM 정책		
		IAM 또는 연동 자격 증명	IAM 정책		
HTTPS, AWS 서명 버전 4 인증(포트 443)	AWS CLI	Amazon Cognito, IAM 또는 연동 자격 증명	IAM 정책		

프로토콜 및 인증 메커니즘	SDK	자격 증명 유형	정책 유형		
HTTPS, TLS 상호 인증(포트 8443)	SDK 지원하지 않음	X.509 인증서	AWS IoT Core 정책		
사용자 지정 인증을 통한 HTTPS(포트 443)	AWS IoT 디바이스 SDK	사용자 지정 권한 부여자	사용자 지정 권한 부여자 정책		

AWS IoT Core 컨트롤 플레인 API 및 정책 유형

프로토콜 및 인증 메커니즘	SDK	자격 증명 유형	정책 유형		
HTTPS AWS 서명 버전 4 인증(포트 443)	AWS CLI	Amazon Cognito 자격 증명	IAM 정책		
		IAM 또는 연동 자격 증명	IAM 정책		

AWS IoT Core 정책은 X.509 인증서, Amazon Cognito 자격 증명 또는 사물 그룹에 연결됩니다. IAM 정책은 IAM 사용자, 그룹 또는 역할에 연결됩니다. AWS IoT 콘솔 또는 AWS IoT Core CLI를 사용하여 정책을 연결하는 경우(인증서, Amazon Cognito 자격 증명 또는 사물 그룹에 연결) AWS IoT Core 정책을 사용합니다. 그렇지 않으면 사물 그룹에 연결된 IAM policy. AWS IoT Core policies가 해당 사물 그룹 내의 모든 사물에 적용됩니다. AWS IoT Core 정책을 적용하려면 `clientId` 및 사물 이름이 일치해야 합니다.

정책 기반 인증은 강력한 도구입니다. 이를 통해 AWS IoT Core에서 디바이스, 사용자 또는 애플리케이션이 수행할 수 있는 작업을 완벽하게 제어할 수 있습니다. 예를 들어 인증서를 사용하여 AWS IoT Core 에 연결하는 디바이스를 가정해 보겠습니다. 디바이스가 모든 MQTT 주제에 액세스하도록 허용할 수도 있고, 단일 주제에만 액세스하도록 제한할 수도 있습니다. 또 하나의 예로 명령줄에 CLI 명령을

입력하는 사용자를 생각해 봅시다. 정책을 사용하면 사용자의 명령 또는 AWS IoT Core 리소스에 대한 액세스를 허용하거나 거부할 수 있습니다. 또한 AWS IoT Core 리소스에 대한 애플리케이션의 액세스를 제어할 수도 있습니다.

AWS IoT 이(가) 정책 문서를 캐시하는 방법 때문에 정책에 대한 변경 사항이 적용되는 데 몇 분이 소요될 수 있습니다. 즉, 최근에 액세스 권한이 부여된 리소스에 액세스하는 데 몇 분 정도 걸릴 수 있으며 액세스가 취소된 후 몇 분 동안 리소스에 액세스할 수 있습니다.

AWS 교육 및 인증

의 권한 부여에 대한 자세한 내용은 AWS 훈련 및 인증 웹 사이트의 인증 및 권한 부여에 대한 AWS IoT Core심층 분석 과정을 참조하십시오. AWS IoT Core

AWS IoT Core 정책

AWS IoT Core 정책은 JSON 문서입니다. IAM 정책과 동일한 규칙을 따릅니다.는 명명된 정책을 AWS IoT Core 지원하므로 많은 자격 증명이 동일한 정책 문서를 참조할 수 있습니다. 명명된 정책은 버전 관리되므로 용이하게 롤백이 가능합니다.

AWS IoT Core 정책을 사용하면 AWS IoT Core 데이터 영역에 대한 액세스를 제어할 수 있습니다. AWS IoT Core 데이터 영역은 AWS IoT Core 메시지 브로커에 연결하고, MQTT 메시지를 송수신하고, 사물의 디바이스 섀도우를 가져오거나 업데이트할 수 있는 작업으로 구성됩니다.

AWS IoT Core 정책은 하나 이상의 정책 문이 포함된 JSON 문서입니다. 각 문에는 다음이 포함됩니다.

- `Effect` - 작업이 허용되는지 또는 거부되는지를 지정합니다.
- `Action` - 정책이 허용하거나 거부하는 작업을 지정합니다.
- `Resource` - 작업이 허용되거나 거부되는 리소스를 지정합니다.

가 정책 문서를 AWS IoT 캐싱하는 방식 때문에 정책에 대한 변경 사항이 적용되려면 6~8분이 걸릴 수 있습니다. 즉, 최근에 액세스 권한이 부여된 리소스에 액세스하는 데 몇 분 정도 걸릴 수 있으며 액세스가 취소된 후 몇 분 동안 리소스에 액세스할 수 있습니다.

AWS IoT Core 정책은 X.509 인증서, Amazon Cognito 자격 증명 및 사물 그룹에 연결할 수 있습니다. 사물 그룹에 연결된 정책은 해당 그룹 내의 모든 사물에 적용됩니다. 정책이 발효되려면 `clientId`와 사물 이름이 일치해야 합니다. AWS IoT Core 정책은 IAM 정책과 동일한 정책 평가 논리를 따릅니다. 기본적으로 모든 정책이 묵시적으로 거부됩니다. 자격 증명 기반 또는 리소스 기반 정책에 포함된 명시적 허용은 이 기본 동작을 재정의합니다. 어떠한 정책의 명시적 거부도 허용을 무시합니다. 자세한 내용은 AWS Identity and Access Management 사용 설명서의 정책 평가 로직을 참조하세요.

주제

- AWS IoT Core 정책 작업
- AWS IoT Core 작업 리소스
- AWS IoT Core 정책 변수
- 교차 서비스 혼동된 대리인 방지
- AWS IoT Core 정책 예제
- Amazon Cognito 자격 증명으로 권한 부여

AWS IoT Core 정책 작업

AWS IoT Core가 정의하는 정책 작업은 다음과 같습니다.

MQTT 정책 작업

`iot:Connect`

AWS IoT Core 메시지 브로커에 연결할 수 있는 권한을 나타냅니다. `CONNECT` 요청이 브로커로 전송될 때마다 `iot:Connect` 권한이 확인됩니다. 메시지 브로커는 클라이언트 ID가 동일한 2개의 클라이언트가 동시에 연결하도록 허용하지 않습니다. 두 번째 클라이언트가 연결된 후 브로커는 기존 연결을 닫습니다. `iot:Connect` 권한을 사용하여 특정 클라이언트 ID를 사용하는 권한 부여된 클라이언트만 연결할 수 있도록 합니다.

`iot:DeleteConnection`

연결된 MQTT 클라이언트를 AWS IoT Core에서 연결 해제할 수 있는 권한을 나타냅니다. 클라이언트를 강제로 연결 해제하도록 요청할 때마다 `iot:DeleteConnection` 권한이 확인됩니다. 클라이언트 연결을 해제하면가 클라이언트의 네트워크 연결을 AWS IoT Core 닫고 선택적으로 세션 상태를 정리합니다.

`iot:GetRetainedMessage`

유지된 단일 메시지의 내용을 가져올 수 있는 권한을 나타냅니다. 보관된 메시지는 RETAIN 플래그가 설정되고 저장된 상태로 게시된 메시지입니다 AWS IoT Core. 계정의 모든 보관된 메시지 목록을 가져올 수 있는 권한은 `iot:ListRetainedMessages` 섹션을 참조하세요.

`iot:ListRetainedMessages`

계정의 보관된 메시지에 대한 요약 정보를 검색할 수 있는 권한을 나타내지만 메시지 내용은 검색할 수 없습니다. 보관된 메시지는 RETAIN 플래그가 설정되고 저장된 상태로 게시된 메시지입니다

AWS IoT Core. 이 작업에 대해 지정된 리소스 ARN은 *여야 합니다. 유지된 단일 메시지의 내용을 가져올 수 있는 권한은 iot:GetRetainedMessage 섹션을 참조하세요.

`iot:Publish`

MQTT 주제를 게시할 수 있는 권한을 나타냅니다. PUBLISH 요청이 브로커로 전송될 때마다 이 권한이 확인됩니다. 이 권한을 사용하여 클라이언트가 특정 주제 패턴에 게시하도록 허용할 수 있습니다.

> **Note**
>
> 또한 `iot:Publish` 권한을 부여하기 위해 `iot:Connect` 권한을 부여할 수도 있습니다.

`iot:Receive`

메시지를 수신할 수 있는 권한을 나타냅니다 AWS IoT Core. 메시지가 클라이언트로 전달될 때마다 `iot:Receive` 권한이 확인됩니다. 이 권한은 전달 시마다 확인되므로 현재 주제를 구독 중인 클라이언트에 대한 권한을 취소하는 데 이 권한을 사용할 수 있습니다.

`iot:RetainPublish`

RETAIN 플래그가 설정된 MQTT 메시지를 게시할 수 있는 권한을 나타냅니다.

> **Note**
>
> 또한 `iot:RetainPublish` 권한을 부여하기 위해 `iot:Publish` 권한을 부여할 수도 있습니다.

`iot:Subscribe`

주제 필터를 구독할 수 있는 권한을 나타냅니다. SUBSCRIBE 요청이 브로커로 전송될 때마다 이 권한이 확인됩니다. 이 권한을 사용하여 클라이언트가 특정 주제 패턴과 일치하는 주제를 구독하도록 허용합니다.

> **Note**
>
> 또한 `iot:Subscribe` 권한을 부여하기 위해 `iot:Connect` 권한을 부여할 수도 있습니다.

디바이스 섀도우 정책 작업

`iot:DeleteThingShadow`

사물의 디바이스 섀도우를 삭제할 수 있는 권한을 나타냅니다. 사물의 디바이스 섀도우의 내용을 삭제하는 요청이 생성될 때마다 `iot:DeleteThingShadow` 권한이 확인됩니다.

`iot:GetThingShadow`

사물의 디바이스 섀도우를 검색할 수 있는 권한을 나타냅니다. 사물의 디바이스 섀도우의 내용을 검색하는 요청이 생성될 때마다 `iot:GetThingShadow` 권한이 확인됩니다.

`iot:ListNamedShadowsForThing`

사물의 이름 지정된 섀도우를 나열할 수 있는 권한을 나타냅니다. 사물의 이름 지정된 섀도우를 나열하는 요청이 생성될 때마다 `iot:ListNamedShadowsForThing` 권한이 확인됩니다.

`iot:UpdateThingShadow`

디바이스 섀도우를 업데이트할 수 있는 권한을 나타냅니다. 사물의 디바이스 섀도우의 내용을 업데이트하는 요청이 생성될 때마다 `iot:UpdateThingShadow` 권한이 확인됩니다.

> **Note**
>
> 작업 실행 정책 작업은 HTTP TLS 엔드포인트에만 적용됩니다. MQTT 엔드포인트를 사용할 경우 이 주제에서 정의하는 MQTT 정책 작업을 사용해야 합니다.
> 이를 보여주는 작업 실행 정책의 예는 MQTT 프로토콜과 함께 작동하는 the section called “기본 작업 정책 예제”을 참조하세요.

작업 실행 AWS IoT Core 정책 작업

`iotjobsdata:DescribeJobExecution`

해당 사물에 대해 작업 실행을 검색할 수 있는 권한을 나타냅니다. 작업 실행 가져오기를 요청할 때마다 `iotjobsdata:DescribeJobExecution` 권한이 확인됩니다.

`iotjobsdata:GetPendingJobExecutions`

사물에 대해 단말 상태가 아닌 작업의 목록을 검색할 수 있는 권한을 나타냅니다. 목록 검색이 요청될 때마다 `iotjobsdata:GetPendingJobExecutions` 권한이 확인됩니다.

`iotjobsdata:UpdateJobExecution`

작업 실행을 업데이트할 수 있는 권한을 나타냅니다. 작업 실행 상태 업데이트가 요청될 때마다 `iotjobsdata:UpdateJobExecution` 권한이 확인됩니다.

`iotjobsdata:StartNextPendingJobExecution`

사물에 대해 대기 중인 다음 작업 실행을 가져오고 시작할 수 있는 권한을 나타냅니다. (즉, 상태가 QUEUED인 작업 실행을 IN_PROGRESS로 업데이트할 수 있는 권한입니다.) 대기 중인 다음 작업의 실행 시작이 요청될 때마다 `iotjobsdata:StartNextPendingJobExecution` 권한이 확인됩니다.

AWS IoT Core 자격 증명 공급자 정책 작업

`iot:AssumeRoleWithCertificate`

자격 AWS IoT Core 증명 공급자를 호출하여 인증서 기반 인증으로 IAM 역할을 수임할 수 있는 권한을 나타냅니다. 자격 AWS IoT Core 증명 공급자가 역할을 수임하도록 요청할 때마다 `iot:AssumeRoleWithCertificate` 권한이 확인됩니다.

AWS IoT Core 작업 리소스

AWS IoT Core 정책 작업에 대한 리소스를 지정하려면 리소스의 Amazon 리소스 이름(ARN)을 사용합니다. 모든 리소스 ARN은 다음 형식으로 되어 있습니다.

```
arn:partition:iot:region:AWS-account-ID:Resource-type/Resource-name
```

다음 표는 각 작업 유형에 지정되는 리소스를 보여 줍니다. ARN 예제는 파티션 `aws`의 계정 ID `123456789012`에 대한 것으로 리전 `us-east-1`에 특정합니다. ARN 형식에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 Amazon 리소스 이름(ARNs)을 참조하세요.

작업	리소스 유형	리소스 이름	ARN 예제
`iot:Connect`	`client`	클라이언트의 클라이언트 ID	`arn:aws:iot:us-east-1:123456789012:client/myClientId`

작업	리소스 유형	리소스 이름	ARN 예제
iot:DeleteConnection	client	클라이언트의 클라이언트 ID	arn:aws:iot:us-east-1:123456789012:client/myClientId
iot:DeleteThingShadow	thing	사물 이름과 섀도우 이름(해당되는 경우)	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iotjobsdata:DescribeJobExecution	thing	사물 이름	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iotjobsdata:GetPendingJobExecutions	thing	사물 이름	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:GetRetainedMessage	topic	보관된 메시지 주제	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:GetThingShadow	thing	사물 이름과 섀도우 이름(해당되는 경우)	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:ListNamedShadowsForThing	모두	모두	*

작업	리소스 유형	리소스 이름	ARN 예제
iot:ListRetainedMessages	모두	모두	*
iot:Publish	topic	주제 문자열	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:Receive	topic	주제 문자열	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:RetainPublish	topic	RETAIN 플래그를 설정하여 게시할 주제	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iotjobsdata:StartNextPendingJobExecution	thing	사물 이름	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:Subscribe	topicfilter	주제 필터 문자열	arn:aws:iot:us-east-1:123456789012:topicfilter/myTopicFilter
iotjobsdata:UpdateJobExecution	thing	사물 이름	arn:aws:iot:us-east-1:123456789012:thing/thingOne

작업	리소스 유형	리소스 이름	ARN 예제
iot:UpdateThingShadow	thing	사물 이름과 섀도우 이름(해당되는 경우)	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:AssumeRoleWithCertificate	rolea]	역할 ARN을 가리키는 역할 별칭	arn:aws:iot:us-east-1:123456789012:rolealias/CredentialProviderRole_alias

AWS IoT Core 정책 변수

AWS IoT Core 는 Resource 또는 Condition 블록의 정책에 사용할 수 있는 AWS IoT Core 정책 변수를 정의합니다. 정책이 평가될 때 정책 변수가 실제 값으로 대체됩니다. 예를 들어 디바이스가 클라이언트 ID가 100-234-3456인 AWS IoT Core 메시지 브로커에 연결된 경우 iot:ClientId 정책 문서에서 정책 변수는 100-234-3456로 대체됩니다.

AWS IoT Core 정책은 와일드카드 문자를 사용하고 IAM 정책과 유사한 규칙을 따를 수 있습니다. 문자열에 *(별표)를 삽입하면 임의의 문자와 일치하는 와일드카드로 취급될 수 있습니다. 예를 들어, 정책의 * 속성에서 여러 MQTT 이름을 설명하기 위해 Resource를 사용할 수 있습니다. + 및 # 문자는 정책에서 리터럴 문자열로 취급됩니다. 와일드카드를 사용하는 방법을 보여 주는 정책 예시는 MQTT 및 AWS IoT Core 정책에서 와일드카드 문자 사용 섹션을 참조하세요.

사전 정의된 정책 변수와 고정된 값을 사용하여 그렇지 않으면 특별한 의미를 갖는 문자를 나타낼 수도 있습니다. 이러한 특수 문자는 ${*}, ${?}, ${$}입니다. 정책 변수 및 특수 문자에 대한 자세한 내용은 IAM 정책 요소: 변수 및 태그와 다수의 키 또는 값을 사용하는 조건 생성을 참조하세요.

주제

- 기본 AWS IoT Core 정책 변수
- 사물 정책 변수
- X.509 인증서 AWS IoT Core 정책 변수

기본 AWS IoT Core 정책 변수

AWS IoT Core 는 다음과 같은 기본 정책 변수를 정의합니다.

- `aws:SourceIp`: AWS IoT Core 메시지 브로커에 연결된 클라이언트의 IP 주소입니다.
- `iot:ClientId`: AWS IoT Core 메시지 브로커에 연결하는 데 사용되는 클라이언트 ID입니다.
- `iot:DomainName`: 연결된 클라이언트의 도메인 이름입니다 AWS IoT Core.

예제

- ClientId 및 SourceIp 정책 변수의 예
- iot:DomainName 정책 변수의 예

`ClientId` 및 **`SourceIp`** 정책 변수의 예

다음 AWS IoT Core 정책은 정책 변수를 사용하는 정책을 보여줍니다.는 정책의 조건 요소에서 보안 주체가 특정 주소 범위 내에서만 API를 요청하도록 허용하는 데 사용할 `aws:SourceIp` 수 있습니다. 예시는 사용자 및 클라우드 서비스에 AWS IoT 작업 사용 권한 부여 섹션을 참조하세요.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
   ],
   "Condition": {
```

```
    "IpAddress": {
     "aws:SourceIp": "123.45.167.89"
    }
   }
  }
 ]
}
```

이 예제에서는 정책이 평가될 때 ${iot:ClientId}가 AWS IoT Core 메시지 브로커에 연결된 클라이언트의 ID로 대체됩니다. ${iot:ClientId} 같은 정책 변수를 사용할 경우 잘못하여 의도치 않은 주제에 대한 액세스를 개방할 수 있습니다. 예를 들어 ${iot:ClientId}를 사용하여 주제 필터를 지정하는 정책을 사용할 경우

```
{
 "Effect": "Allow",
 "Action": [
  "iot:Subscribe"
 ],
 "Resource": [
  "arn:aws:iot:us-east-1:123456789012:topicfilter/my/${iot:ClientId}/topic"
 ]
}
```

클라이언트가 +를 클라이언트 ID로 사용하여 연결할 수 있습니다. 이는 사용자가 주제 필터 my/+/topic와 일치하는 모든 주제를 구독하도록 허용하는 것입니다. 그러한 보안 간극을 방지하려면 iot:Connect 정책 작업을 사용하여 어느 클라이언트 ID가 연결할 수 있는지 제어합니다. 예를 들어 이 정책은 클라이언트 ID가 clientid1인 클라이언트만 연결하도록 허용합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid"
   ]
  }
```

```
 ]
}
```

> **Note**
>
> 정책 변수 `${iot:ClientId}`를 Connect와 함께 사용하는 것은 권장되지 않습니다. `ClientId`의 값은 확인하지 않으므로 다른 클라이언트의 ID를 가진 어태처가 검증을 통과할 수 있지만 연결이 끊길 수 있습니다. `ClientId`는 무엇이든 허용되므로 무작위로 클라이언트 ID를 설정하면 사물 그룹 정책을 우회할 수 있습니다.

iot:DomainName 정책 변수의 예

`iot:DomainName` 정책 변수를 추가하여 사용할 수 있는 도메인을 제한할 수 있습니다. `iot:DomainName` 정책 변수를 추가하면 디바이스가 구성된 특정 엔드포인트에만 연결할 수 있습니다.

다음 정책은 디바이스가 지정된 도메인에 연결할 수 있도록 허용합니다.

```
{
 "Version":"2012-10-17",
 "Statement": {
  "Sid": "AllowConnectionsToSpecifiedDomain",
  "Effect": "Allow",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "arn:aws:iot:us-east-1:123456789012:client/clientid",
  "Condition": {
   "StringEquals": {
    "iot:DomainName": "d1234567890abcdefghij-ats.iot.us-east-1.amazonaws.com"
   }
  }
 }
}
```

다음 정책은 지정된 도메인에 연결하는 디바이스를 거부합니다.

```
{
 "Version":"2012-10-17",
 "Statement": {
  "Sid": "DenyConnectionsToSpecifiedDomain",
  "Effect": "Deny",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "arn:aws:iot:us-east-1:123456789012:client/clientid",
  "Condition": {
   "StringEquals": {
    "iot:DomainName": "d1234567890abcdefghij-ats.iot.us-east-1.amazonaws.com"
   }
  }
 }
}
```

IAM 조건 연산자에 대한 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건 연산자를 참조하세요. 도메인 구성에 대한 자세한 내용은 What is a domain configuration?을 참조하세요.

사물 정책 변수

사물 정책 변수를 사용하면 사물 이름, 사물 유형 및 사물 속성 값과 같은 사물 속성을 기반으로 권한을 부여하거나 거부하는 AWS IoT Core 정책을 작성할 수 있습니다. 사물 정책 변수를 사용하여 동일한 정책을 적용하여 많은 AWS IoT Core 디바이스를 제어할 수 있습니다. 디바이스 프로비저닝에 대한 자세한 내용은 디바이스 프로비저닝을 참조하세요.

비독점 사물 연결을 사용하는 경우, 동일한 인증서를 여러 사물에 연결할 수 있습니다. 명확한 연결 관계를 유지하고 잠재적 충돌을 방지하려면, 클라이언트 ID가 사물 이름과 일치해야 합니다. 이 경우 사물이 연결될 때 전송되는 MQTT `Connect` 메시지의 클라이언트 ID에서 사물 이름을 가져옵니다 AWS IoT Core.

AWS IoT Core 정책에서 사물 정책 변수를 사용할 때는 다음 사항에 유의하세요.

- AttachThingPrincipal API를 사용하여 인증서 또는 보안 주체(인증된 Amazon Cognito 자격 증명)를 사물에 연결할 수 있습니다.
- 비독점 사물 연결이 이루어졌을 경우, 사물 이름을 사물 정책 변수로 바꿀 때 MQTT 연결 메시지 또는 TLS 연결의 `clientId` 값이 사물 이름과 정확히 일치해야 합니다.

다음의 사물 정책 변수를 사용할 수 있습니다.

- `iot:Connection.Thing.ThingName`

  이렇게 하면 정책이 평가되는 AWS IoT Core 레지스트리의 사물 이름으로 확인됩니다.는 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.

- `iot:Connection.Thing.ThingTypeName`

  이 변수는 정책이 평가되는 사물과 연결된 사물 유형으로 변환됩니다. MQTT/WebSocket 연결의 클라이언트 ID는 사물 이름과 동일해야 합니다. 이 정책 변수는 MQTT 또는 WebSocket 프로토콜을 통한 MQTT를 통해 연결하는 경우에만 사용할 수 있습니다.

- `iot:Connection.Thing.Attributes[`*`attributeName`*`]`

  이 변수는 정책이 평가되는 사물과 연결된 지정된 속성의 값으로 변환됩니다. 사물은 최대 50개의 속성을 가질 수 있습니다. 각 속성은 정책 변수 `iot:Connection.Thing.Attributes[`*`attributeName`*`]`으로 사용할 수 있습니다. 여기서 *`attributeName`*은 속성의 이름입니다. MQTT/WebSocket 연결의 클라이언트 ID는 사물 이름과 동일해야 합니다. 이 정책 변수는 MQTT 또는 MQTT over WebSocket 프로토콜을 통해 연결하는 경우에만 사용할 수 있습니다.

- `iot:Connection.Thing.IsAttached`

  `iot:Connection.Thing.IsAttached: ["true"]`는에 등록 AWS IoT 되고 보안 주체에 연결된 디바이스만 정책 내의 권한에 액세스할 수 있도록 합니다. 레지스트리의 IoT 사물에 AWS IoT Core 연결되지 않은 인증서를 제공하는 AWS IoT Core 경우이 변수를 사용하여 디바이스가에 연결되지 않도록 할 수 있습니다.이 변수에는 연결 사물이 AttachThingPrincipal API를 사용하여 레지스트리의 인증서 또는 Amazon Cognito 자격 증명에 연결되었음을 `false` 나타내는 `true` 또는 값이 있습니다. 사물 이름은 클라이언트 ID로 간주됩니다.

클라이언트 ID가 사물 이름과 일치하거나 인증서를 하나의 사물에만 독점적으로 연결하는 경우, 정책 정의에서 정책 변수를 사용하면 정책 관리를 간소화할 수 있습니다. 각 IoT 사물에 대해 개별 정책을 생성하는 대신, 사물 정책 변수를 사용하여 단일 정책을 정의할 수 있습니다. 이 정책은 모든 디바이스에 동적으로 적용할 수 있습니다. 다음은 작동 방식을 보여주는 예제 정책입니다. 자세한 내용은 ??? 단원을 참조하십시오.

```
{
```

```
 "Version":"2012-10-17",
 "Statement": [
  {
   "Condition": {
    "StringLike": {
     "iot:ClientId": "*${iot:Connection.Thing.Attributes[envType]}"
    }
   },
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/*"
  }
 ]
}
```

이 정책 예제에서는 클라이언트 ID가 envType 속성 값으로 끝나는 AWS IoT Core 경우 사물을에 연결할 수 있습니다. 클라이언트 ID 패턴이 일치하는 사물만 연결할 수 있습니다.

X.509 인증서 AWS IoT Core 정책 변수

X.509 인증서 정책 변수는 AWS IoT Core 정책 작성을 지원합니다. 이러한 정책은 X.509 인증서 속성을 기반으로 권한을 부여합니다. 다음 섹션에서는 이러한 인증서 정책 변수를 사용하는 방법을 설명합니다.

> **Important**
>
> X.509 인증서에 특정 인증서 속성이 포함되어 있지 않지만 정책 문서에서 해당 인증서 정책 변수를 사용하는 경우 정책 평가가 예기치 않은 동작으로 이어질 수 있습니다.

CertificateId

RegisterCertificate API에서 `certificateId`가 응답 본문에 나타납니다. 인증서에 대한 정보를 가져오려면 DescribeCertificate의 `certificateId`를 사용할 수 있습니다.

발행자 속성

다음 AWS IoT Core 정책 변수는 인증서 발급자가 설정한 인증서 속성을 기반으로 권한 허용 또는 거부를 지원합니다.

- `iot:Certificate.Issuer.DistinguishedNameQualifier`

- iot:Certificate.Issuer.Country
- iot:Certificate.Issuer.Organization
- iot:Certificate.Issuer.OrganizationalUnit
- iot:Certificate.Issuer.State
- iot:Certificate.Issuer.CommonName
- iot:Certificate.Issuer.SerialNumber
- iot:Certificate.Issuer.Title
- iot:Certificate.Issuer.Surname
- iot:Certificate.Issuer.GivenName
- iot:Certificate.Issuer.Initials
- iot:Certificate.Issuer.Pseudonym
- iot:Certificate.Issuer.GenerationQualifier

제목 속성

다음 AWS IoT Core 정책 변수는 인증서 발급자가 설정한 인증서 주체 속성을 기반으로 권한 부여 또는 거부를 지원합니다.

- iot:Certificate.Subject.DistinguishedNameQualifier
- iot:Certificate.Subject.Country
- iot:Certificate.Subject.Organization
- iot:Certificate.Subject.OrganizationalUnit
- iot:Certificate.Subject.State
- iot:Certificate.Subject.CommonName
- iot:Certificate.Subject.SerialNumber
- iot:Certificate.Subject.Title
- iot:Certificate.Subject.Surname
- iot:Certificate.Subject.GivenName
- iot:Certificate.Subject.Initials
- iot:Certificate.Subject.Pseudonym
- iot:Certificate.Subject.GenerationQualifier

X.509 인증서는 이러한 속성이 하나 이상의 값을 포함한 옵션을 가지도록 허용합니다. 기본적으로 각 다중 값 속성의 정책 변수는 첫 번째 값을 반환합니다. 예를 들어 `Certificate.Subject.Country` 속성에는 국가 이름 목록이 포함되었을 수 있지만, 정책에서 평가될 때 `iot:Certificate.Subject.Country`는 첫 번째 국가 이름으로 대체됩니다.

1 기반 인덱스를 사용하여 첫 번째 값 이외의 특정 속성 값을 요청할 수 있습니다. 예를 들어 `iot:Certificate.Subject.Country.1`은 `Certificate.Subject.Country` 속성의 두 번째 국가로 대체됩니다. 존재하지 않는 인덱스 값을 지정할 경우(예: 속성에 값이 2개만 할당되었지만 세 번째 값을 요청하는 경우) 값이 대체되지 않고 권한 부여가 실패합니다. 정책 변수 이름에 `.List` 접미사를 사용하여 속성의 모든 값을 지정할 수 있습니다.

발행자 대체 이름 속성

다음 AWS IoT Core 정책 변수는 인증서 발급자가 설정한 발급자 대체 이름 속성을 기반으로 권한 부여 또는 거부를 지원합니다.

- `iot:Certificate.Issuer.AlternativeName.RFC822Name`
- `iot:Certificate.Issuer.AlternativeName.DNSName`
- `iot:Certificate.Issuer.AlternativeName.DirectoryName`
- `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier`
- `iot:Certificate.Issuer.AlternativeName.IPAddress`

제목 대체 이름 속성

다음 AWS IoT Core 정책 변수는 인증서 발급자가 설정한 주체 대체 이름 속성을 기반으로 권한 부여 또는 거부를 지원합니다.

- `iot:Certificate.Subject.AlternativeName.RFC822Name`
- `iot:Certificate.Subject.AlternativeName.DNSName`
- `iot:Certificate.Subject.AlternativeName.DirectoryName`
- `iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier`
- `iot:Certificate.Subject.AlternativeName.IPAddress`

기타 속성

`iot:Certificate.SerialNumber`를 사용하여 인증서의 일련 번호에 따라 AWS IoT Core 리소스에 대한 액세스를 허용하거나 거부할 수 있습니다. `iot:Certificate.AvailableKeys` 정책 변수는 값을 갖는 모든 인증서 정책 변수의 이름을 포함합니다.

X.509 인증서 정책 변수 사용

이 주제에서는 인증서 정책 변수를 사용하는 방법에 대한 세부 정보를 제공합니다. X.509 인증서 정책 변수는 X.509 인증서 속성을 기반으로 권한을 부여하는 AWS IoT Core 정책을 생성할 때 필수입니다. X.509 인증서에 특정 인증서 속성이 포함되어 있지 않지만 정책 문서에서 해당 인증서 정책 변수를 사용하는 경우 정책 평가가 예기치 않은 동작으로 이어질 수 있습니다. 이는 누락된 정책 변수가 정책 문에서 평가되지 않기 때문입니다.

이 주제에서 수행할 작업

- X.509 인증서 예제
- 인증서 발행자 속성을 인증서 정책 변수로 사용
- 인증서 주체 속성을 인증서 정책 변수로 사용
- 인증서 발행자 대체 이름 속성을 인증서 정책 변수로 사용
- 인증서 주체 대체 이름 속성을 인증서 정책 변수로 사용
- 다른 인증서 속성을 인증서 정책 변수로 사용
- X.509 인증서 정책 변수 제한 사항
- 인증서 정책 변수를 사용하는 정책 예제

X.509 인증서 예제

일반적인 X.509 인증서는 다음과 같이 표시될 수 있습니다. 이 예제 인증서에는 인증서 속성이 포함되어 있습니다. AWS IoT Core 정책을 평가하는 동안 `Serial Number`, `Issuer`, `Subject`, `X509v3 Issuer Alternative Name` 및 `X509v3 Subject Alternative Name` 인증서 정책 변수로 다음 인증서 속성이 채워집니다.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            92:12:85:cb:b7:a5:e0:86
```

```
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=IoT Devices, OU=SmartHome, ST=WA, CN=IoT Devices Primary CA,
    GN=Primary CA1/initials=XY/dnQualifier=Example corp,
    SN=SmartHome/ title=CA1/pseudonym=Primary_CA/generationQualifier=2/serialNumber=987

        Validity
            Not Before: Mar 26 03:25:40 2024 GMT
            Not After : Apr 28 03:25:40 2025 GMT
        Subject: C=US, O=IoT Devices, OU=LightBulb, ST=NY, CN=LightBulb Device Cert,
    GN=Bulb/initials=ZZ/dnQualifier=Bulb001,
    SN=Multi Color/title=RGB/pseudonym=RGB Device/generationQualifier=4/
serialNumber=123
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    << REDACTED >>
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:example.com, IP Address:1.2.3.4, URI:ResourceIdentifier001,
 email:device1@example.com, DirName:/C=US/O=IoT/OU=SmartHome/CN=LightBulbCert
            X509v3 Issuer Alternative Name:
                DNS:issuer.com, IP Address:5.6.7.8, URI:PrimarySignerCA,
 email:primary@issuer.com, DirName:/C=US/O=Issuer/OU=IoT Devices/CN=Primary Issuer CA
    Signature Algorithm: sha256WithRSAEncryption
         << REDACTED >>
```

인증서 발행자 속성을 인증서 정책 변수로 사용

다음 표에는 인증서 발급자 속성이 AWS IoT Core 정책에 채워지는 방법에 대한 세부 정보가 나와 있습니다.

정책에 채워질 발행자 속성

인증서 발행자 속성	인증서 정책 변수
• C=US • O=IoT Devices	• `iot:Certificate.Issuer.Country = US`

인증서 발행자 속성	인증서 정책 변수
• OU=SmartHome • ST=WA • CN=IoT Devices Primary CA • GN=Primary CA1 • initials=XY • dnQualifier=Example corp • SN=SmartHome • title=CA1 • pseudonym=Primary_CA • generationQualifier=2 • serialNumber=987	• `iot:Certificate.Issuer.Organization = IoT Devices` • `iot:Certificate.Issuer.OrganizationalUnit = SmartHome` • `iot:Certificate.Issuer.State = WA` • `iot:Certificate.Issuer.CommonName = IoT Devices Primary CA` • `iot:Certificate.Issuer.GivenName = Primary CA1` • `iot:Certificate.Issuer.initials = XY` • `iot:Certificate.Issuer.Distinguished NameQualifier = Example corp` • `iot:Certificate.Issuer.Surname = SmartHome` • `iot:Certificate.Issuer.Title = CA1` • `iot:Certificate.Issuer.Pseudonym = Primary_CA` • `iot:Certificate.Issuer.GenerationQualifier = 2` • `iot:Certificate.Issuer.SerialNumber = 987`

인증서 주체 속성을 인증서 정책 변수로 사용

다음 표에는 인증서 주체 속성이 AWS IoT Core 정책에 채워지는 방법에 대한 세부 정보가 나와 있습니다.

정책에 채워질 주체 속성

인증서 주체 속성	인증서 정책 변수
• C=US • O=IoT Devices • ST=NY	• `iot:Certificate.Subject.Country = US` • `iot:Certificate.Subject.Organization = IoT Devices` • `iot:Certificate.Subject.State = NY`

인증서 주체 속성	인증서 정책 변수
• CN=LightBulb Device Cert • GN=Bulb • initials=ZZ • dnQualifier=Bulb001 • SN=Multi Color • title=RGB • pseudonym=RGB Device • generationQualifier=4 • serialNumber=123	• `iot:Certificate.Subject.CommonName = LightBulb Device Cert` • `iot:Certificate.Subject.GivenName = Bulb` • `iot:Certificate.Subject.initials = ZZ` • `iot:Certificate.Subject.DistinguishedNameQualifier = Bulb001` • `iot:Certificate.Subject.Surname = Multi Color` • `iot:Certificate.Subject.Title = RGB` • `iot:Certificate.Subject.Pseudonym = RGB Device` • `iot:Certificate.Subject.GenerationQualifier = 4` • `iot:Certificate.Subject.SerialNumber = 123`

인증서 발행자 대체 이름 속성을 인증서 정책 변수로 사용

다음 표에는 인증서 발행자 대체 이름 속성이 AWS IoT Core 정책에 채워지는 방법에 대한 세부 정보가 나와 있습니다.

정책에 채워질 발행자 대체 이름 속성

X509v3 발행자 대체 이름	정책의 속성
• DNS:issuer.com • IP Address:5.6.7.8 • URI:PrimarySignerCA • email:primary@issuer.com • DirName:/C=US/O=Issuer/OU=IoT Devices/CN=Primary Issuer CA	• `iot:Certificate.Issuer.AlternativeName.DNSName = issuer.com` • `iot:Certificate.Issuer.AlternativeName.IPAddress = 5.6.7.8` • `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier = PrimarySignerCA` • `iot:Certificate.Issuer.AlternativeName.RFC822Name = primary@issuer.com`

X509v3 발행자 대체 이름	정책의 속성
	• `iot:Certificate.Issuer.AlternativeName.DirectoryName = cn=Primary Issuer CA,ou=IoT Devices,o=Issuer,c=US`

인증서 주체 대체 이름 속성을 인증서 정책 변수로 사용

다음 표에는 인증서 주체 대체 이름 속성이 AWS IoT Core 정책에 채워지는 방법에 대한 세부 정보가 나와 있습니다.

정책에 채워질 주체 대체 이름 속성

X509v3 주체 대체 이름	정책의 속성
• DNS:example.com • IP Address:1.2.3.4 • URI:ResourceIdentifier001 • email:device1@example.com • DirName :/C=US/O=IoT/OU=SmartHome/CN=LightBulbCert	• `iot:Certificate.Subject.AlternativeName.DNSName = example.com` • `iot:Certificate.Subject.AlternativeName.IPAddress = 1.2.3.4` • `iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier = ResourceIdentifier001` • `iot:Certificate.Subject.AlternativeName.RFC822Name = device1@example.com` • `iot:Certificate.Subject.AlternativeName.DirectoryName = cn=LightBulbCert,ou=SmartHome,o=IoT,c=US`

다른 인증서 속성을 인증서 정책 변수로 사용

다음 표에는 다른 인증서 속성이 AWS IoT Core 정책에 채워지는 방법에 대한 세부 정보가 나와 있습니다.

정책에 채워질 기타 속성

기타 인증서 속성	인증서 정책 변수
`Serial Number: 92:12:85:cb:b7:a5:e0:86`	`iot:Certificate.SerialNumber = 105256223 89124227206`

X.509 인증서 정책 변수 제한 사항

X.509 인증서 정책 변수에는 다음의 제한 사항이 적용됩니다.

누락된 정책 변수

X.509 인증서에 특정 인증서 속성이 포함되어 있지 않지만 정책 문서에서 해당 인증서 정책 변수를 사용하는 경우 정책 평가가 예기치 않은 동작으로 이어질 수 있습니다. 이는 누락된 정책 변수가 정책 문에서 평가되지 않기 때문입니다.

인증서 SerialNumber 형식

AWS IoT Core 는 인증서 일련 번호를 10진수 정수의 문자열 표현으로 처리합니다. 예를 들어 정책에서 인증서 일련 번호와 일치하는 클라이언트 ID와의 연결만 허용하는 경우 클라이언트 ID는 십진수 형식의 일련 번호여야 합니다.

와일드카드

인증서 속성에 와일드카드 문자가 있을 경우 정책 변수가 인증서 속성 값으로 대체되지 않습니다. 이렇게 하면 정책 문서에 `${policy-variable}` 텍스트가 남습니다. 이는 인증 실패를 초래할 수 있습니다. *, $, +, ?, # 등의 와일드카드 문자를 사용할 수 있습니다.

어레이 필드

어레이를 포함하는 인증서 속성은 5개 항목으로 제한됩니다. 추가 항목은 무시됩니다.

문자열 길이

모든 문자열 값은 1,024자로 제한됩니다. 인증서 속성이 1,024자를 초과하는 문자열을 포함할 경우 정책 변수가 인증서 속성 값으로 대체되지 않습니다. 이렇게 하면 정책 문서에 `${policy-variable}`이 남습니다. 이는 인증 실패를 초래할 수 있습니다.

특수 문자

`,`, `"`, `\`, +, =, <, >, `;` 등의 특수 문자는 정책 변수에 사용될 때 앞에 백슬래시(\)가 있어야 합니다. 예를 들어, `Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington`

C=US는 Amazon Web Service O\=Amazon.com Inc. L\=Seattle ST\=Washington C\=US가 됩니다.

인증서 정책 변수를 사용하는 정책 예제

다음 정책 문서에서는 인증서 일련번호와 일치하는 클라이언트 ID와의 연결 및 패턴과 일치하는 주제에 대한 게시를 허용합니다. `${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*`.

> **Important**
>
> X.509 인증서에 특정 인증서 속성이 포함되어 있지 않지만 정책 문서에서 해당 인증서 정책 변수를 사용하는 경우 정책 평가가 예기치 않은 동작으로 이어질 수 있습니다. 이는 누락된 정책 변수가 정책 문에서 평가되지 않기 때문입니다. 예를 들어 `iot:Certificate.Subject.Organization` 속성이 포함되지 않은 인증서에 다음 정책 문서를 연결하면 정책 평가 중에 `iot:Certificate.Subject.Organization` 인증서 정책 변수가 채워지지 않습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Certificate.SerialNumber}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*"
   ]
```

```
  }
 ]
}
```

Null 조건 연산자를 사용하여 정책 평가 중에 정책에 사용되는 인증서 정책 변수가 채워지도록 할 수도 있습니다. 다음 정책 문서는 인증서 일련번호 및 인증서 주체 일반 이름 속성이 있는 경우에만 인증서가 있는 `iot:Connect`를 허용합니다.

모든 인증서 정책 변수에는 문자열 값이 있으므로 모든 문자열 조건 연산자가 지원됩니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/*"
   ],
   "Condition": {
    "Null": {
     "iot:Certificate.SerialNumber": "false",
     "iot:Certificate.Subject.CommonName": "false"
    }
   }
  }
 ]
}
```

교차 서비스 혼동된 대리인 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 에서 AWS교차 서비스 가장은 혼동된 대리자 문제를 초래할 수 있습니다. 교차 서비스 가장은 한 서비스(직접적으로 호출하는 서비스)가 다른 서비스(직접적으로 호출되는 서비스)를 직접적으로 호출할 때 발생할 수 있습니다. 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위

해 AWS 에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 위탁자를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다.

리소스에 다른 서비스를 AWS IoT 제공하는 권한을 제한하려면 리소스 정책에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하는 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.

혼동된 대리자 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 Amazon 리소스 이름(ARN)이 포함된 aws:SourceArn 전역 조건 컨텍스트 키를 사용하는 것입니다. 의 경우 AWS IoT는 arn:*aws*:iot:*region*:*account-id*:*resource-type/resource-id* 리소스별 권한 또는의 형식을 준수해야 aws:SourceArn 합니다arn:*aws*:iot:*region*:*account-id*:*. resource-id는 허용된 리소스의 이름 또는 ID이거나 허용된 리소스 ID의 와일드카드 문입니다. *##*이 AWS IoT 리전과 일치하고 *account-id*가 고객 계정 ID와 일치하는지 확인합니다.

다음 예제에서는 AWS IoT 역할 신뢰 정책에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다. 더 많은 예시는 혼동된 대리자 방지의 세부 예제를 참조합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
               "aws:SourceArn":"arn:aws:iot:us-east-1:123456789012:*"
        }
         }
      }
   ]
}
```

> **Note**
>
> 액세스 거부 오류가 발생하는 경우 AWS 보안 토큰 서비스(STS)와의 서비스 통합이 `aws:SourceArn` 및 `aws:SourceAccount` 컨텍스트 키를 지원하지 않기 때문일 수 있습니다.

혼동된 대리자 방지의 세부 예제

이 섹션에서는 AWS IoT 역할 신뢰 정책에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법에 대한 자세한 예를 제공합니다.

- 플릿 프로비저닝
- JITP
- 보안 인증 제공업체

플릿 프로비저닝

프로비저닝 템플릿 리소스를 사용하여 플릿 프로비저닝을 구성할 수 있습니다. 프로비저닝 템플릿이 프로비저닝 역할을 참조하면 해당 역할의 신뢰 정책에 `aws:SourceArn` 및 `aws:SourceAccount` 조건 키가 포함될 수 있습니다. 이러한 키는 구성이 `sts:AssumeRole` 요청을 간접적으로 호출할 수 있는 리소스를 제한합니다.

다음 신뢰 정책을 사용하는 역할은 `SourceArn`에 지정된 프로비저닝 템플릿에 대한 IoT 위탁자(`iot.amazonaws.com`)만 수임할 수 있습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
```

```
                "aws:SourceArn":"arn:aws:iot:us-
east-1:123456789012:provisioningtemplate/example_template"
            }
             }
          }
       ]
}
```

JITP

적시 프로비저닝(JITP)에서는 프로비저닝 템플릿을 CA와 별도의 리소스로 사용하거나 템플릿 본문과 역할을 CA 인증서 구성의 일부로 정의할 수 있습니다. aws:SourceArn AWS IoT 역할 신뢰 정책의 값은 프로비저닝 템플릿을 정의하는 방법에 따라 달라집니다.

프로비저닝 템플릿을 별도의 리소스로 정의

프로비저닝 템플릿을 별도의 리소스로 정의하는 경우 aws:SourceArn의 값은 "arn:aws:iot:*region*:*account-id*:provisioningtemplate/*example_template*"가 될 수 있습니다. 플릿 프로비저닝에서 동일한 정책 예제를 사용할 수 있습니다.

CA 인증서에서 프로비저닝 템플릿 정의

CA 인증서 리소스 내에서 프로비저닝 템플릿을 정의하는 경우 aws:SourceArn의 값은 "arn:aws:iot:*region*:*account-id*:cacert/*cert_id*" 또는 "arn:aws:iot:*region*:*account-id*:cacert/*"가 될 수 있습니다. 생성 시 CA 인증서의 ID와 같은 리소스 식별자를 알 수 없는 경우 와일드카드를 사용할 수 있습니다.

다음 신뢰 정책이 있는 역할은 SourceArn에 지정된 CA 인증서의 IoT 위탁자(iot.amazonaws.com)만 수임할 수 있습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
```

```
                "aws:SourceAccount":"123456789012"
            },
                "ArnLike":{
                    "aws:SourceArn":"arn:aws:iot:us-
east-1:123456789012:cacert/
8ecde6884f3d87b1125ba31ac3fcb13d7016de7f57cc904fe1cb97c6ae98196e"
            }
             }
        }
    ]
}
```

CA 인증서를 생성할 때 등록 구성에서 프로비저닝 역할을 참조할 수 있습니다. 프로비저닝 역할의 신뢰 정책은 `aws:SourceArn`을 사용하여 역할을 수임할 수 있는 리소스를 제한할 수 있습니다. 그러나 CA 인증서를 등록하기 위한 최초 RegisterCACertificate 직접 호출 중에는 CA 인증서의 ARN이 `aws:SourceArn` 조건에 지정되지 않습니다.

이를 해결하려면, 즉에 등록된 특정 CA 인증서에 프로비저닝 역할 신뢰 정책을 지정하려면 다음을 수행할 AWS IoT Core수 있습니다.

- 먼저 `RegistrationConfig` 파라미터를 제공하지 않고 RegisterCACertificate를 직접적으로 호출합니다.
- CA 인증서가에 등록된 후 CA 인증서에서 UpdateCACertificate를 AWS IoT Core호출합니다.

  UpdateCACertificate 호출에서 새로 등록된 CA 인증서의 ARN으로 `aws:SourceArn` 설정된 프로비저닝 역할 신뢰 정책을 포함하는 `RegistrationConfig`를 제공합니다.

보안 인증 제공업체

AWS IoT Core 자격 증명 공급자의 경우에서 역할 별칭을 생성하는 데 AWS 계정 사용하는 것과 동일한를 사용하고 `aws:SourceAccount`에서 역할별 리소스 유형의 리소스 ARN과 일치하는 문을 지정합니다`aws:SourceArn`. AWS IoT Core 자격 증명 공급자와 함께 사용할 IAM 역할을 생성할 때 역할을 수임해야 할 수 있는 역할 별칭의 ARNs을 `aws:SourceArn` 조건에 포함시켜 교차 서비스 `sts:AssumeRole` 요청을 승인해야 합니다.

다음 신뢰 정책이 있는 역할은 `SourceArn`에 지정된 roleAlias에 대한 AWS IoT Core 자격 증명 공급자(`credentials.iot.amazonaws.com`)의 위탁자만 수임할 수 있습니다. 위탁자가 `aws:SourceArn` 조건에 지정된 것 이외의 역할 별칭에 대한 자격 증명을 검색하려고 하면 다른 역할 별칭이 동일한 IAM 역할을 참조하더라도 요청이 거부됩니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:iot:us-
east-1:123456789012:rolealias/example_rolealias"
        }
      }
    }
  ]
}
```

AWS IoT Core 정책 예제

이 섹션의 정책 예는 AWS IoT Core에서 일반적인 작업을 완료하는 데 사용되는 정책 문서를 보여줍니다. 이들을 솔루션에 대한 정책을 생성할 때부터 시작하는 예로 사용할 수 있습니다.

이 섹션의 예에서는 다음 정책 요소를 사용합니다.

- the section called “AWS IoT Core 정책 작업”
- the section called “AWS IoT Core 작업 리소스”
- the section called “자격 증명 기반 정책 예”
- the section called “기본 AWS IoT Core 정책 변수”
- the section called “X.509 인증서 AWS IoT Core 정책 변수”

이 단원의 정책 예:

- 연결 정책 예제
- 게시/구독 정책 예제

- 연결 및 게시 정책 예제
- 보관된 메시지 정책 예
- 인증서 정책 예제
- 사물 정책 예제
- 기본 작업 정책 예제

연결 정책 예제

다음 정책은 클라이언트 IDs `client1` 및에 대한 연결 권한을 거부합니다. `client2` AWS IoT Core클라이언트 ID는 AWS IoT Core 레지스트리에 등록되고 연결에 사용되는 보안 주체에 연결된 사물의 이름과 일치합니다.

> **Note**
>
> 등록된 디바이스의 경우 `Connect` 작업에 사물 정책 변수를 사용하고 연결에 사용되는 보안 주체에 사물을 연결하는 것이 좋습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1",
    "arn:aws:iot:us-east-1:123456789012:client/client2"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
```

```
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  }
 ]
}
```

다음 정책은 클라이언트 ID를 사용하여에 연결할 수 AWS IoT Core 있는 권한을 부여합니다`client1`. 이 정책 예시는 등록되지 않은 디바이스를 위한 것입니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1"
   ]
  }
 ]
}
```

MQTT 영구 세션 정책 예

`connectAttributes`을(를) 사용하면 IAM 정책의 연결 메시지에 사용할 속성(예: `PersistentConnect` 또는 `LastWill`)을 지정할 수 있습니다. 자세한 내용은 connectAttributes 사용 단원을 참조하십시오.

다음 정책은 `PersistentConnect` 기능이 있는 연결을 허용합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
```

```
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

다음 정책에서는 PersistentConnect을(를) 허용하지 않고 다른 기능은 허용됩니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringNotEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

위의 정책은 StringEquals을(를) 사용하여 표현할 수도 있습니다. 새 기능을 포함한 다른 모든 기능이 허용됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1"
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

다음 정책은 PersistentConnect 및 LastWill 둘 다에 의한 연결을 허용합니다. 다른 새로운 기능은 허용되지 않습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
```

```
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  }
 ]
}
```

다음 정책은 `LastWill`이(가) 있거나 없는 클라이언트에 의한 클린 연결을 허용합니다. 다른 기능은 허용되지 않습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iot:Connect"
        ],
        "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
        "Condition": {
            "StringEquals": {
                "iot:ConnectAttributes": "LastWill"
        }
        }
    }]
}
```

다음 정책은 기본 기능을 사용하는 연결만 허용합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
```

```
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

다음 정책은 PersistentConnect이 있어야만 연결을 허용합니다. 연결에 PersistentConnect을(를) 사용하는 한 새로운 기능이 허용됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

다음 정책은 연결에 PersistentConnect 및 LastWill 두 가지가 모두 있어야 함을 나타냅니다. 새로운 기능은 허용되지 않습니다.

```
{
 "Version":"2012-10-17",
```

```
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
  "Condition": {
   "ForAllValues:StringEquals": {
    "iot:ConnectAttributes": [
     "PersistentConnect",
     "LastWill"
    ]
   }
  }
 },
 {
  "Effect": "Deny",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "*",
  "Condition": {
   "ForAllValues:StringEquals": {
    "iot:ConnectAttributes": [
     "PersistentConnect"
    ]
   }
  }
 },
 {
  "Effect": "Deny",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "*",
  "Condition": {
   "ForAllValues:StringEquals": {
    "iot:ConnectAttributes": [
     "LastWill"
    ]
   }
  }
 },
```

```
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

다음 정책에는 PersistentConnect이(가) 없어야 하지만 LastWill은(는) 있을 수 있습니다. 다른 새로운 기능은 허용되지 않습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
```

```
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
            }
        }
        }
    ]
}
```

다음 정책은 "my/lastwill/topicName" 주제와 관련된 LastWill이 있는 클라이언트에 의해서만 연결을 허용합니다. LastWill 주제를 사용하는 한 모든 기능이 허용됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                "iot:LastWillTopic": "arn:aws:iot:us-east-1:123456789012:topic/my/
lastwill/topicName"
            }
        }
        }
    ]
}
```

다음 정책은 특정 LastWillTopic을(를) 사용하는 클린 연결만 허용합니다. LastWillTopic을(를) 사용하는 한 모든 기능이 허용됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                "iot:LastWillTopic": "arn:aws:iot:us-east-1:123456789012:topic/my/
lastwill/topicName"
            }
        }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

게시/구독 정책 예제

사용하는 정책은 연결하는 방법에 따라 달라집니다 AWS IoT Core. MQTT 클라이언트, HTTP 또는 WebSocket을 사용하여 AWS IoT Core 에 연결할 수 있습니다. MQTT 클라이언트를 사용하여 연결하는 경우 X.509 인증서를 사용하여 인증합니다. HTTP 또는 WebSocket 프로토콜을 통해 연결하는 경우 서명 버전 4 및 Amazon Cognito를 사용하여 인증합니다.

> **Note**
>
> 등록된 디바이스의 경우 `Connect` 작업에 사물 정책 변수를 사용하고 연결에 사용되는 보안 주체에 사물을 연결하는 것이 좋습니다.

이 섹션의 내용:

- MQTT 및 AWS IoT Core 정책에서 와일드카드 문자 사용
- 특정 주제에 대한 메시지 게시, 구독 및 수신 정책
- 특정 접두사가 붙은 주제에 대한 메시지 게시, 구독 및 수신 정책
- 각 디바이스와 관련된 주제에 대한 메시지 게시, 구독 및 수신 정책
- 주제 이름에 사물 속성이 포함된 주제에 대한 메시지 게시, 구독 및 수신 정책
- 주제 이름의 하위 주제에 대한 메시지 게시를 거부하는 정책
- 주제 이름의 하위 주제에 대한 메시지 수신을 거부하는 정책
- MQTT 와일드카드 문자를 사용하여 주제를 구독하는 정책
- HTTP 및 WebSocket 클라이언트용 정책

MQTT 및 AWS IoT Core 정책에서 와일드카드 문자 사용

MQTT와 AWS IoT Core 정책은 와일드카드 문자가 다르므로 신중하게 고려한 후 선택해야 합니다. MQTT에서 와일드카드 문자 + 및 #는 MQTT 주제 필터에서 여러 주제 이름을 구독하는 데 사용됩니다. AWS IoT Core 정책은 * 및를 와일드카드 문자?로 사용하고 IAM 정책의 규칙을 따릅니다. 정책 문서에서 *는 문자 조합을 나타내고 ?는 단일 문자를 나타냅니다. 정책 문서에서 MQTT 와일드카드 문자 + 및 #은 특별한 의미가 없는 문자로 취급됩니다. 정책의 `resource` 속성에서 여러 주제 이름과 주제 필터를 설명하려면 MQTT 와일드카드 문자 대신 * 및 ? 와일드카드 문자를 사용하세요.

정책 문서에서 사용할 와일드카드 문자를 선택할 때는 * 문자가 하나의 주제 수준에 국한되지 않는다는 점을 고려하세요. + 문자는 MQTT 주제 필터에서 단일 주제 수준으로 제한됩니다. 와일드카드 사양을 단일 MQTT 주제 필터 수준으로 제한하려면 여러 개의 ? 문자를 사용해 보세요. 정책 리소스에서 와일드카드 문자를 사용하는 방법과 와일드카드 문자가 무엇과 일치하는지에 대한 자세한 내용은 리소스 ARN에서 와일드카드 사용을 참조하세요.

아래 표는 MQTT 및 MQTT 클라이언트의 AWS IoT Core 정책에 사용되는 다양한 와일드카드 문자를 보여줍니다.

와일드카드 문자	MQTT 와일드카드 문자입니까	MQTT 예제	AWS IoT Core 정책 와일드카드 문자임	MQTT 클라이언트 AWS IoT Core 정책 예제
#	예	`some/#`	아니요	해당 사항 없음

와일드카드 문자	MQTT 와일드카드 문자입니까	MQTT 예제	AWS IoT Core 정책 와일드카드 문자임	MQTT 클라이언트 AWS IoT Core 정책 예제
+	예	`some/+/topic`	아니요	해당 사항 없음

*	아니요	해당 사항 없음	예	`topicfilter/some/*/topic`
				
				`topicfilter/some/sensor*/topic`

?	아니요	해당 사항 없음	예	`topic/some/?????/topic`
				
				`topicfilter/some/sensor???/topic`

특정 주제에 대한 메시지 게시, 구독 및 수신 정책

다음은 등록된 디바이스와 등록되지 않은 디바이스에서 'some_specific _topic'이라는 주제에 대해 메시지를 게시, 구독 및 수신하는 예를 보여줍니다. 또한 예제에서는 `Publish` 및 `Receive`가이 'topic'을 리소스로 사용하고 `Subscribe`가 topicfilter'를 리소스로 사용함을 강조합니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 또한 'some_specific_topic'이라는 주제에 대한 `Publish`, `Subscribe` 및 `Receive` 권한을 제공합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
```

```
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/some_specific_topic"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
   ]
  }
 ]
}
```

Unregistered devices

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스가 clientId1, clientId2 또는 clientId3을 사용하여 연결하도록 허용합니다. 또한 'some_specific_topic'이라는 주제에 대한 `Publish`, `Subscribe` 및 `Receive` 권한을 제공합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/
some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        }
    ]
}
```

특정 접두사가 붙은 주제에 대한 메시지 게시, 구독 및 수신 정책

다음은 등록된 디바이스와 등록되지 않은 디바이스에서 'topic_prefix'라는 접두사가 붙은 주제에 대해 메시지를 게시, 구독 및 수신하는 예를 보여줍니다.

> **Note**
>
> 이 예제에서는 와일드카드 문자 *를 사용한다는 점에 유의하세요. *는 단일 명령문에서 여러 주제 이름에 대한 권한을 제공하는 데 유용하지만 디바이스에 필요한 것보다 더 많은 권한을 제공하여 의도하지 않은 결과를 초래할 수 있습니다. 따라서 와일드카드 문자 *는 신중하게 고려한 후에만 사용하는 것이 좋습니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 또한 'topic_prefix' 접두사가 붙은 주제에 대한 `Publish`, `Subscribe` 및 `Receive` 권한을 제공합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
```

```
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish",
     "iot:Receive"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Subscribe"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
    ]
   }
 ]
}
```

Unregistered devices

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스가 clientId1, clientId2 또는 clientId3을 사용하여 연결하도록 허용합니다. 또한 'topic_prefix' 접두사가 붙은 주제에 대한 `Publish`, `Subscribe` 및 `Receive` 권한을 제공합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
            ]
        }
    ]
}
```

각 디바이스와 관련된 주제에 대한 메시지 게시, 구독 및 수신 정책

다음은 등록된 디바이스와 등록되지 않은 디바이스에서 해당 디바이스와 관련된 주제에 대해 메시지를 게시, 구독 및 수신하는 예를 보여줍니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 이 정책은 사물별 주제(`sensor/device/${iot:Connection.Thing.ThingName}`)에 게시하고 사물별 주제(`command/device/${iot:Connection.Thing.ThingName}`)를 구독 및 수신할 수 있는 권한을 제공합니다. 레지스트리의 사물 이름이 ‘thing1’인 경우 디바이스는 ‘sensor/device/thing1’ 주제에 게시할 수 있습니다. 또한 디바이스는 'command/device/thing1' 주제를 구독하고 수신할 수 있습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
```

```
    "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

Unregistered devices

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스가 clientId1, clientId2 또는 clientId3을 사용하여 연결하도록 허용합니다. 이 정책은 클라이언트별 주제(`sensor/device/${iot:ClientId}`)에 게시하고 클라이언트별 주제(`command/device/${iot:ClientId}`)를 구독 및 수신할 수 있는 권한을 제공합니다. 디바이스가 clientId1으로 clientId와 연결되는 경우 'sensor/device/clientId1' 주제에 게시할 수 있습니다. 또한 디바이스는 `device/clientId1/command` 주제를 구독하고 수신할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

주제 이름에 사물 속성이 포함된 주제에 대한 메시지 게시, 구독 및 수신 정책

다음은 등록된 디바이스에서 이름에 사물 속성이 포함된 주제를 게시, 구독 및 수신하는 예를 보여줍니다.

> **Note**
>
> 사물 속성은 AWS IoT Core 레지스트리에 등록된 디바이스에만 존재합니다. 등록되지 않은 디바이스에 해당하는 예제는 없습니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 이 정책은 주제 이름에 사물 속성이 포함된 경우 주제(`sensor/${iot:Connection.Thing.Attributes[version]}`)에 게시하고 주제(`command/${iot:Connection.Thing.Attributes[location]}`)를 구독 및 수신할 수 있는 권한을 제공합니다. 레지스트리에 있는 사물 이름에 `version=v1` 및 `location=Seattle`이 있는

경우 디바이스는 'sensor/v1' 주제에 게시하고 'command/Seattle' 주제를 구독 및 수신할 수 있습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/
${iot:Connection.Thing.Attributes[version]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
```

```
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  }
 ]
}
```

Unregistered devices

사물 속성은 AWS IoT Core 의 레지스트리에 등록된 디바이스에만 존재하므로 등록되지 않은 사물에 해당하는 예제는 없습니다.

주제 이름의 하위 주제에 대한 메시지 게시를 거부하는 정책

다음은 등록된 디바이스와 등록되지 않은 디바이스에서 특정 하위 주제를 제외한 모든 주제에 메시지를 게시하는 예를 보여줍니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 이 정책은 'department/' 접두사가 붙은 모든 주제에 게시할 수 있는 권한을 제공하지만 'department/admins' 하위 주제에 게시할 수 있는 권한은 제공하지 않습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
```

```
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/department/*"
    ]
   },
   {
    "Effect": "Deny",
    "Action": [
     "iot:Publish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
    ]
   }
 ]
}
```

Unregistered devices

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스가 clientId1, clientId2 또는 clientId3을 사용하여 연결하도록 허용합니다. 이 정책은 'department/' 접두사가 붙은 모든 주제에 게시할 수 있는 권한을 제공하지만 'department/admins' 하위 주제에 게시할 수 있는 권한은 제공하지 않습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
            ]
        }
    ]
}
```

주제 이름의 하위 주제에 대한 메시지 수신을 거부하는 정책

다음은 등록된 디바이스와 등록되지 않은 디바이스에서 특정 하위 주제를 제외한 특정 접두사가 붙은 주제에 대해 메시지를 구독하고 수신하는 예를 보여줍니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 이 정책은 디바이스가 'topic_prefix' 접두사가 붙은 모든 주제를 구독할 수 있도록 허용합니다. `iot:Receive` 명령문에서 `NotResource`를 사용하면 'topic_prefix/restricted' 접두사가 붙은 주제를 제외하고 디바이스가 구독한 모든 주제에서 메시지를 수신할 수 있습니다. 예를 들어, 이 정책에 따라 디바이스는 'topic_prefix/topic1' 및 'topic_prefix/restricted'를 구독할 수 있지만 'topic_prefix/topic1' 주제의 메시지만 수신하고 'topic_prefix/restricted' 주제의 메시지는 수신하지 않습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix/*"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Receive",
   "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix/
restricted/*"
  }
 ]
}
```

Unregistered devices

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스가 clientId1, clientId2 또는 clientId3을 사용하여 연결하도록 허용합니다. 이 정책은 디바이스가 'topic_prefix' 접두사가 붙은 모든 주제를 구독할 수 있도록 허용합니다. `iot:Receive` 명령문에서 `NotResource`를 사용하면 'topic_prefix/restricted' 접두사가 붙은 주제를 제외하고 디바이스가 구독한 모든 주제에서 메시지를 수신할 수 있습니다. 예를 들어 이 정책을 사용하면 디바이스가 'topic_prefix/topic1'과 'topic_prefix/restricted'을 구독할 수 있습니다. 하지만 주제 'topic_prefix/topic1'에서는 메시지만 수신하고 주제 'topic_prefix/restricted'에서는 메시지를 수신하지 않습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
topic_prefix/*"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/
topic_prefix/restricted/*"
        }
    ]
}
```

MQTT 와일드카드 문자를 사용하여 주제를 구독하는 정책

MQTT 와일드카드 문자 + 및 #은 리터럴 문자열로 처리되지만 AWS IoT Core 정책에서 사용할 때는 와일드카드로 처리되지 않습니다. MQTT에서 + 및 #은 주제 필터를 구독할 때만 와일드카드로 취급되고 다른 모든 컨텍스트에서는 리터럴 문자열로 취급됩니다. 이러한 MQTT 와일드카드는 신중하게 고려한 후에만 AWS IoT Core 정책의 일부로 사용하는 것이 좋습니다.

다음은 AWS IoT Core 정책에서 MQTT 와일드카드를 사용하는 등록된 사물과 등록되지 않은 사물의 예를 보여줍니다. 이러한 와일드카드는 리터럴 문자열로 취급됩니다.

Registered devices

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스가 레지스트리의 사물 이름과 일치하는 clientId에 연결하도록 허용합니다. 이 정책은 디바이스가 'department/+/employees' 및 'location/#' 주제를 구독할 수 있도록 허용합니다. + 및 #은 AWS IoT Core 정책에서 리터럴 문자열로 취급되기 때문에 디바이스가 'department/+/employees' 주제를 구독할 수 있지만 'department/engineering/employees' 주제는 구독할 수 없습니다. 마찬가지로, 디바이스는 'location/#' 주제를 구독할 수 있지만 'location/Seattle' 주제는 구독할 수 없습니다. 그러나 디바이스가 'department/+/employees' 주제를 구독하면 정책에 따라 'department/engineering/employees' 주제에서 메시지를 수신할 수 있습니다. 마찬가지로, 디바이스가 'location/#' 주제를 구독하면 'location/Seattle' 주제에서도 메시지를 수신하게 됩니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/department/+/
employees"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/#"
  },
  {
```

```
   "Effect": "Allow",
   "Action": "iot:Receive",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
  }
 ]
}
```

Unregistered devices

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스가 clientId1, clientId2 또는 clientId3을 사용하여 연결하도록 허용합니다. 이 정책은 디바이스가 'department/+/employees' 및 'location/#' 주제를 구독할 수 있도록 허용합니다. + 및 #은 AWS IoT Core 정책에서 리터럴 문자열로 취급되기 때문에 디바이스가 'department/+/employees' 주제를 구독할 수 있지만 'department/engineering/employees' 주제는 구독할 수 없습니다. 마찬가지로, 디바이스는 'location/#' 주제를 구독할 수 있지만 'location/Seattle' 주제는 구독할 수 없습니다. 그러나 디바이스가 'department/+/employees' 주제를 구독하면 정책에 따라 'department/engineering/employees' 주제에서 메시지를 수신할 수 있습니다. 마찬가지로, 디바이스가 'location/#' 주제를 구독하면 'location/Seattle' 주제에서도 메시지를 수신하게 됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
department/+/employees"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/
#"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        }
    ]
}
```

HTTP 및 WebSocket 클라이언트용 정책

HTTP 또는 WebSocket 프로토콜을 통해 연결하는 경우 서명 버전 4 및 Amazon Cognito를 사용하여 인증합니다. Amazon Cognito 자격 증명은 인증되거나 인증되지 않을 수 있습니다. 인증된 자격 증명은 지원되는 자격 증명 공급자가 인증한 사용자를 위한 것이고, 인증되지 않은 자격 증명은 대개 자격 증명 공급자를 통해 인증하지 않는 게스트 사용자에 속합니다. Amazon Cognito는 인증되지 않은 자격 증명을 지원하는 고유한 식별자 및 AWS 자격 증명을 제공합니다. 자세한 내용은 the section called “Amazon Cognito 자격 증명으로 권한 부여” 단원을 참조하십시오.

다음 작업의 경우 `AttachPolicy` API를 통해 Amazon Cognito 자격 증명에 연결된 AWS IoT Core 정책을 AWS IoT Core 사용합니다. 이렇게 하면 인증된 자격 증명이 있는 Amazon Cognito ID 풀에 연결된 권한의 범위가 줄어듭니다.

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`
- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

즉, Amazon Cognito 자격 증명에는 IAM 역할 정책 및 AWS IoT Core 정책의 권한이 필요합니다. `AttachPolicy` API를 통해 풀에 IAM 역할 정책을 연결하고 Amazon Cognito 자격 증명에 AWS IoT Core AWS IoT Core 정책을 연결합니다.

인증된 사용자와 인증되지 않은 사용자의 자격 증명 유형은 서로 다릅니다. Amazon Cognito 자격 증명에 AWS IoT 정책을 연결하지 않으면 인증된 사용자가에서 권한 부여에 실패 AWS IoT 하고 AWS IoT 리소스 및 작업에 액세스할 수 없습니다.

> **Note**
>
> 다른 AWS IoT Core 작업 또는 인증되지 않은 자격 증명의 경우 AWS IoT Core 는 Amazon Cognito 자격 증명 풀 역할에 연결된 권한의 범위를 좁히지 않습니다. 인증된 자격 증명과 인증되지 않은 자격 증명 모두에 대해 이 정책은 Amazon Cognito 풀 역할에 연결하도록 권장되는 가장 허용적인 정책입니다.

HTTP

인증되지 않은 Amazon Cognito 자격 증명이 Amazon Cognito 자격 증명별 주제에 HTTP를 통해 메시지를 게시하도록 허용하려면 Amazon Cognito 자격 증명 풀 역할에 다음 IAM 정책을 연결합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"
            ]
        }
    ]
}
```

인증된 사용자를 허용하려면 이전 정책을 Amazon Cognito 자격 증명 풀 역할과 AWS IoT Core AttachPolicy API를 사용하여 Amazon Cognito 자격 증명에 연결합니다.

> **Note**
>
> Amazon Cognito 자격 증명을 승인할 때는 정책을 모두 AWS IoT Core 고려하고 지정된 최소 권한을 부여합니다. 두 정책이 모두 요청된 작업을 허용하는 경우에만 작업이 허용됩니다. 어느 한 정책이 작업을 허용하지 않을 경우 해당 작업은 승인되지 않습니다.

MQTT

인증되지 않은 Amazon Cognito 자격 증명이 사용자 계정의 Amazon Cognito 자격 증명별 주제에 WebSocket를 통해 MQTT 메시지를 게시하도록 허용하려면 Amazon Cognito 자격 증명 풀 역할에 다음 IAM 정책을 연결합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

인증된 사용자를 허용하려면 이전 정책을 Amazon Cognito 자격 증명 풀 역할과 AWS IoT Core AttachPolicy API를 사용하여 Amazon Cognito 자격 증명에 연결합니다.

> **Note**
>
> Amazon Cognito 자격 증명을 승인할 때 AWS IoT Core 는 지정된 최소 권한을 고려하고 부여합니다. 두 정책이 모두 요청된 작업을 허용하는 경우에만 작업이 허용됩니다. 어느 한 정책이 작업을 허용하지 않을 경우 해당 작업은 승인되지 않습니다.

연결 및 게시 정책 예제

AWS IoT Core 레지스트리에 사물로 등록된 디바이스의 경우 다음 정책은 사물 이름과 일치하는 클라이언트 ID AWS IoT Core 를 사용하여에 연결할 수 있는 권한을 부여하고 디바이스가 클라이언트 ID 또는 사물 이름별 MQTT 주제에 게시하도록 제한합니다. 연결이 성공하려면 사물 이름을 AWS IoT Core 레지스트리에 등록하고 사물에 연결된 자격 증명 또는 보안 주체를 사용하여 인증해야 합니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/${iot:Connection.Thing.ThingName}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
      ]
    }
  ]
}
```

AWS IoT Core 레지스트리에 사물로 등록되지 않은 디바이스의 경우 다음 정책은 클라이언트 ID로에 연결할 수 AWS IoT Core 있는 권한을 부여`client1`하고 디바이스가 clientID별 MQTT 주제에 게시하도록 제한합니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/client1"
      ]
    }
  ]
}
```

보관된 메시지 정책 예

보관된 메시지를 사용하려면 특정 정책이 필요합니다. 보관된 메시지는 RETAIN 플래그가 설정되고에 의해 저장된 MQTT 메시지입니다 AWS IoT Core. 이 섹션에서는 보관된 메시지의 일반적인 사용을 허용하는 정책의 예를 보여줍니다.

이 섹션의 내용:

- 보관된 메시지를 연결하고 게시하는 정책
- 보관된 Will 메시지를 연결하고 게시하는 정책
- 보관된 메시지를 나열하고 가져오는 정책

보관된 메시지를 연결하고 게시하는 정책

디바이스에서 보관된 메시지를 게시하려면 디바이스가 MQTT 보관된 메시지를 연결하고 게시할 수 있어야 합니다. 다음 정책은 주제에 대해 이러한 권한을 부여합니다(클라이언트 **device1**에 대한 `device/sample/configuration`). 연결 권한을 부여하는 다른 예는 the section called “연결 및 게시 정책 예제” 섹션을 참조하세요.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:RetainPublish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/device/sample/configuration"
   ]
  }
 ]
}
```

보관된 Will 메시지를 연결하고 게시하는 정책

클라이언트는 클라이언트가 예기치 않게 연결 해제될 때 게시 AWS IoT Core 할 메시지를 구성할 수 있습니다. MQTT는 이러한 메시지를 Will 메시지라고 합니다. Will 메시지를 포함하려면 클라이언트의 연결 권한에 조건이 더 추가되어야 합니다.

다음 정책 문서는 AWS IoT Core 도 보관하는 `will`이라는 주제로 식별되는 Will 메시지를 연결하고 게시할 수 있는 권한을 모든 클라이언트에게 부여합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ],
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:RetainPublish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/will"
   ]
  }
 ]
}
```

보관된 메시지를 나열하고 가져오는 정책

서비스 및 애플리케이션은 ListRetainedMessages 및 GetRetainedMessage를 호출하여 MQTT 클라이언트를 지원할 필요 없이 보관된 메시지에 액세스할 수 있습니다. 이러한 작업을 호출하는 서비스 및 애플리케이션은 다음 예와 같은 정책을 사용하여 권한이 부여되어야 합니다.

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:ListRetainedMessages"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/device1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetRetainedMessage"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/foo"
            ]
        }
    ]
}
```

인증서 정책 예제

AWS IoT Core 레지스트리에 등록된 디바이스의 경우, 다음 정책은 사물 이름과 일치하는 클라이언트 ID AWS IoT Core 로에 연결하고 디바이스가 자체 인증에 사용한 인증서`certificateId`의와 이름이 동일한 주제에 게시할 수 있는 권한을 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 클라이언트 IDs, , `client2` `client3` 및 AWS IoT Core 를 사용하여 `client1`에 연결하고 디바이스가 자체 인증에 사용한 인증서`certificateId`의와 이름이 동일한 주제에 게시할 수 있는 권한을 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 사물 이름과 일치하는 클라이언트 ID AWS IoT Core 로에 연결하고 디바이스가 자체 인증에 사용한 인증서의 제목 CommonName 필드와 이름이 동일한 주제에 게시할 수 있는 권한을 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

> **Note**
>
> 이 예에서는 인증서 제목 공통 이름이 주제 식별자로 사용되고, 주제 공통 이름이 등록된 인증서마다 고유하다고 가정합니다. 인증서를 여러 디바이스에서 공유하는 경우, 제목 공통 이름은 이 인증서를 공유하는 모든 디바이스에서 동일하므로, 여러 디바이스에서 동일한 주제에 게시 권한을 허용하게 됩니다(권장되지 않음).

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 클라이언트 IDs, , client2및 AWS IoT Core 를 사용하여 client1에 연결하고 디바이스가 자체 인증에 사용한 인증서의 제목 CommonName 필드client3와 이름이 동일한 주제에 게시할 수 있는 권한을 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

Note

이 예에서는 인증서 제목 공통 이름이 주제 식별자로 사용되고, 주제 공통 이름이 등록된 인증서마다 고유하다고 가정합니다. 인증서를 여러 디바이스에서 공유하는 경우, 제목 공통 이름은 이 인증서를 공유하는 모든 디바이스에서 동일하므로, 여러 디바이스에서 동일한 주제에 게시 권한을 허용하게 됩니다(권장되지 않음).

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 사물 이름과 일치하는 클라이언트 ID AWS IoT Core 로에 연결하고 디바이스를 인증하는 데 사용된 인증서의 `Subject.CommonName.2` 필드가 로 설정된 `admin/` 경우 이름이 접두사로 지정된 주제에 게시할 수 있는 권한을 부여합니다. `Administrator`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
            }
        }
        }
    ]
}
```

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우, 다음 정책은 클라이언트 IDs `client1`, `client2` `client3` 및 AWS IoT Core 를 사용하여에 연결하고 디바이스를 인증하는 데 사용된 인증서의 `Subject.CommonName.2` 필드가 로 설정된 `admin/` 경우 이름이 접두사인 주제에 게시할 수 있는 권한을 부여합니다. `Administrator`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
            }
        }
        }
    ]
}
```

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 Administrator다음 정책은 디바이스 인증에 사용된 인증서에 Subject.CommonName 필드 중 하나가 로 설정된 ThingName 경우 디바이스가 사물 이름을 사용하여 admin/ 로 구성된 특정 주제에 게시할 수 있도록 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
```

```
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/
${iot:Connection.Thing.ThingName}"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
        }
    ]
}
```

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우, 다음 정책은 디바이스를 인증하는 데 사용되는 인증서에 Subject.CommonName 필드 중 하나가 AWS IoT Core 로 설정된 admin 경우 클라이언트 IDs client1, client2 client3 및를 사용하여에 연결하고 주제에 게시할 수 있는 권한을 부여합니다. Administrator

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
```

```
        }
    ]
}
```

사물 정책 예제

다음 정책은를 인증하는 데 사용되는 인증서 AWS IoT Core 가 정책이 평가되는 사물에 연결된 경우 디바이스가 연결되도록 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":["iot:Connect"],
            "Resource":[ "*" ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
            }
        }
        }
    ]
}
```

다음 정책은 인증서가 특정 사물 유형의 사물에 연결되어 있고 사물에 값 `attributeValue`가 있는 `attributeName`의 속성이 있는 경우 디바이스가 게시할 수 있도록 허용합니다. 사물 정책 변수에 대한 자세한 내용은 사물 정책 변수를 참조하세요.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/device/stats",
      "Condition": {
```

```
        "StringEquals": {
          "iot:Connection.Thing.Attributes[attributeName]": "attributeValue",
          "iot:Connection.Thing.ThingTypeName": "Thing_Type_Name"
        },
        "Bool": {
          "iot:Connection.Thing.IsAttached": "true"
        }
      }
    }
  ]
}
```

다음 정책은 디바이스가 사물의 속성으로 시작하는 주제에 게시할 수 있도록 허용합니다. 디바이스 인증서가 사물과 연결되어 있지 않으면 이 변수가 해결되지 않고 액세스 거부 오류가 발생합니다. 사물 정책 변수에 대한 자세한 내용은 사물 정책 변수를 참조하세요.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.Attributes[attributeName]}/*"
    }
  ]
}
```

기본 작업 정책 예제

이 샘플은 단일 디바이스인 작업 대상이 작업 요청을 수신하고 AWS IoT로 작업 실행 상태를 전달하기 위해 필요한 정책 문을 보여줍니다.

*us-west-2:57EXAMPLE833*을 AWS 리전, 콜론 문자(:) 및 12자리 AWS 계정 숫자로 바꾼 다음 *uniqueThingName*을 디바이스를 나타내는 사물 리소스의 이름으로 바꿉니다 AWS IoT.

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/uniqueThingName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/job/*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/jobExecution/
*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName"
            ]
        }
    ]
}
```

Amazon Cognito 자격 증명으로 권한 부여

Amazon Cognito 자격 증명에는 인증된 자격 증명과 인증되지 않은 자격 증명이라는 두 가지 유형이 있습니다. 앱이 인증되지 않은 Amazon Cognito 자격 증명을 지원하는 경우 인증이 수행되지 않으므로 사용자가 누구인지 알 수 없습니다.

인증되지 않은 자격 증명: 인증되지 않은 Amazon Cognito 자격 증명의 경우 인증되지 않은 자격 증명 풀에 IAM 역할을 연결하여 권한을 부여합니다. 알 수 없는 사용자가 사용할 수 있도록 하려는 리소스에 대해서만 액세스 권한을 부여하는 것이 좋습니다.

> **Important**
>
> 에 연결하는 인증되지 않은 Amazon Cognito 사용자의 경우 IAM 정책에서 매우 제한된 리소스에 대한 액세스 권한을 부여하는 AWS IoT Core것이 좋습니다.

인증된 자격 증명: 인증된 Amazon Cognito 자격 증명의 경우 다음 두 곳에서 권한을 지정해야 합니다.

- 인증된 Amazon Cognito 자격 증명 풀에 IAM 정책을 연결합니다.

- Amazon Cognito 자격 증명(인증된 사용자)에 AWS IoT Core 정책을 연결합니다.

에 연결하는 미인증 및 인증된 Amazon Cognito 사용자에 대한 정책 예제 AWS IoT Core

다음 예제는 Amazon Cognito 자격 증명의 IAM 정책과 IoT 정책에 있는 권한을 보여줍니다. 인증된 사용자가 디바이스별 주제(예: device/DEVICE_ID/status)에 게시하려고 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/Client_ID"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/device/Device_ID/status"
            ]
        }
    ]
}
```

다음 예제는 인증되지 않은 Amazon Cognito 역할의 IAM 정책에 있는 권한을 보여줍니다. 인증되지 않은 사용자가 인증이 필요 없는 디바이스별 주제 이외의 주제에 게시하려고 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/non_device_specific_topic"
            ]
        }
    ]
}
```

GitHub 예제

GitHub의 다음 예제 웹 애플리케이션은 인증된 사용자에 대한 정책 연결을 사용자 가입 및 인증 프로세스에 통합하는 방법을 보여줍니다.

- AWS Amplify 및를 사용한 MQTT React 웹 애플리케이션 게시/구독 AWS IoT Device SDK for JavaScript
- MQTT 게시/구독 AWS AmplifyAWS IoT Device SDK for JavaScript, 및 Lambda 함수를 사용한 React 웹 애플리케이션

Amplify는 서비스와 통합되는 웹 및 모바일 애플리케이션을 구축하는 데 도움이 되는 도구 및 AWS 서비스 세트입니다. Amplify에 대한 자세한 내용은 Amplify 프레임워크 설명서를 참조하세요.

두 예 모두 다음 단계를 수행합니다.

1. 사용자가 계정에 가입하면 애플리케이션에서 Amazon Cognito 사용자 풀과 자격 증명을 생성합니다.
2. 사용자가 인증하면 애플리케이션이 정책을 생성하여 자격 증명에 연결합니다. 이렇게 하면 사용자에게 게시 및 구독 권한이 부여됩니다.
3. 사용자는 애플리케이션을 사용하여 MQTT 주제를 게시하고 구독할 수 있습니다.

첫 번째 예는 인증 작업 내에서 직접 `AttachPolicy` API 작업을 사용합니다. 다음 예에서는 Amplify와 AWS IoT Device SDK for JavaScript를 사용하는 React 웹 애플리케이션 내에서 이 API 호출을 구현하는 방법을 보여줍니다.

```
function attachPolicy(id, policyName) {
    var Iot = new AWS.Iot({region: AWSConfiguration.region, apiVersion:
 AWSConfiguration.apiVersion, endpoint: AWSConfiguration.endpoint});
    var params = {policyName: policyName, target: id};

    console.log("Attach IoT Policy: " + policyName + " with cognito identity id: " +
 id);
    Iot.attachPolicy(params, function(err, data) {
         if (err) {
               if (err.code !== 'ResourceAlreadyExistsException') {
                  console.log(err);
               }
          }
         else  {
            console.log("Successfully attached policy with the identity", data);
         }
     });
}
```

이 코드는 AuthDisplay.js 파일에 나타납니다.

두 번째 예는 Lambda 함수에서 `AttachPolicy` API 작업을 구현합니다. 다음 예에서는 Lambda가 이 API 호출을 사용하는 방법을 보여줍니다.

```
iot.attachPolicy(params, function(err, data) {
     if (err) {
           if (err.code !== 'ResourceAlreadyExistsException') {
              console.log(err);
              res.json({error: err, url: req.url, body: req.body});
           }
      }
     else  {
        console.log(data);
        res.json({success: 'Create and attach policy call succeed!', url: req.url,
 body: req.body});
```

```
    }
});
```

이 코드는 app.js 파일의 `iot.GetPolicy` 함수 안에 나타납니다.

> **Note**
>
> Amazon Cognito 자격 증명 풀을 통해 얻은 AWS 자격 증명으로 함수를 호출하면 Lambda 함수의 컨텍스트 객체에에 대한 값이 포함됩니다`context.cognito_identity_id`. 자세한 내용은 다음을 참조하세요.
>
> - AWS Lambda Node.js의 컨텍스트 객체
> - AWS Lambda Python의 컨텍스트 객체
> - AWS Lambda Ruby의 컨텍스트 객체
> - AWS Lambda Java의 컨텍스트 객체
> - AWS Lambda Go의 컨텍스트 객체
> - AWS Lambda C#의 컨텍스트 객체
> - AWS Lambda PowerShell의 컨텍스트 객체

AWS IoT Core 자격 증명 공급자를 사용하여 AWS 서비스에 대한 직접 호출 권한 부여

디바이스는 X.509 인증서를 사용하여 TLS 상호 인증 프로토콜을 AWS IoT Core 사용하여에 연결할 수 있습니다. 다른 AWS 서비스는 인증서 기반 인증을 지원하지 않지만 AWS 서명 버전 4 형식의 AWS 자격 증명을 사용하여 호출할 수 있습니다. 서명 버전 4 알고리즘에서는 일반적으로 호출자에게 액세스 키 ID와 보안 액세스 키를 요구합니다. AWS IoT Core 에는 기본 제공 X.509 인증서를 고유한 디바이스 자격 증명으로 사용하여 AWS 요청을 인증할 수 있는 자격 증명 공급자가 있습니다. 따라서 디바이스에 액세스 키 ID와 보안 액세스 키를 저장할 필요가 없습니다.

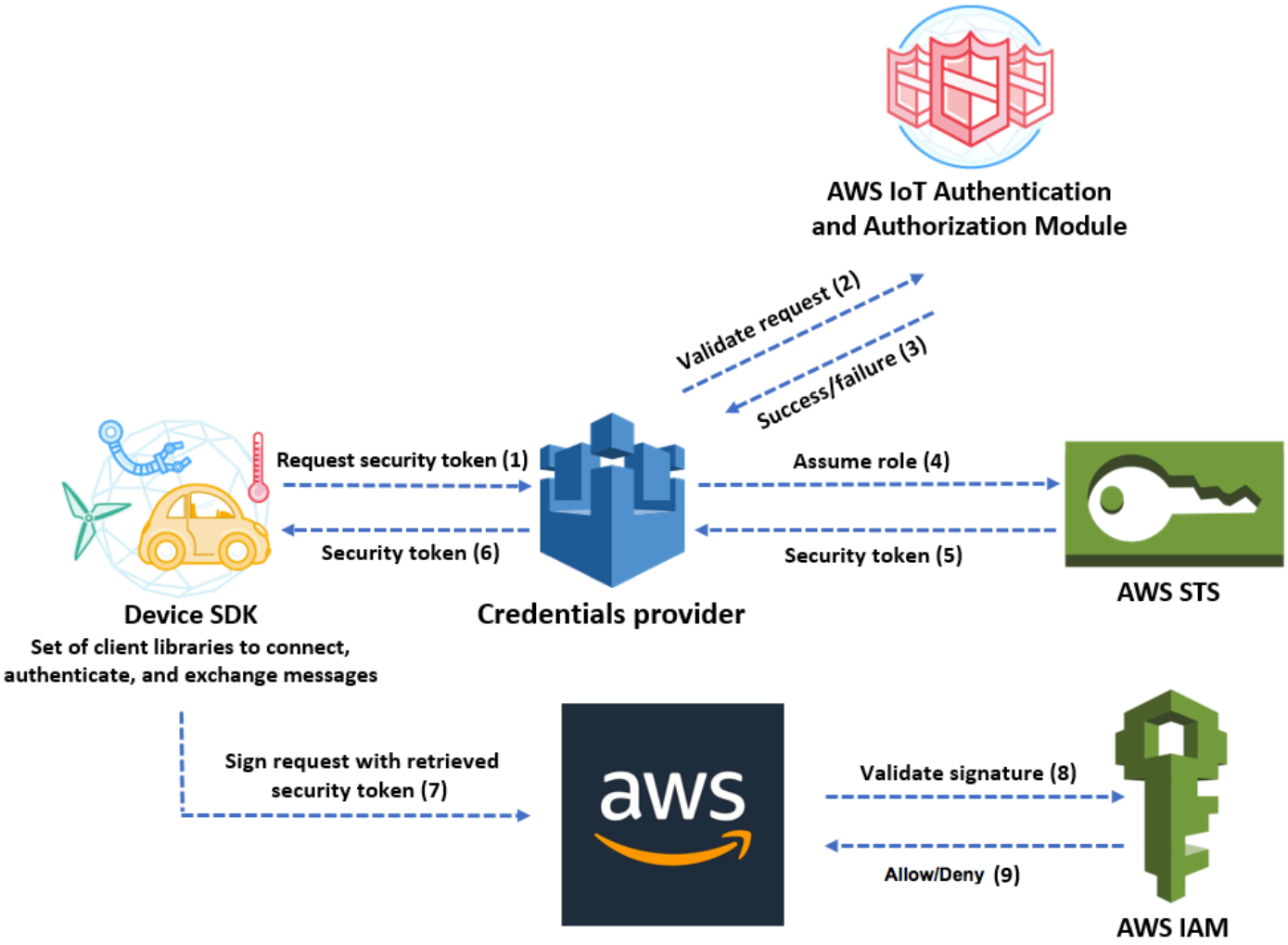
이 자격 증명 공급자는 X.509 인증서를 사용하여 호출자를 인증한 후 권한이 제한된 임시 보안 토큰을 발급합니다. 토큰을 사용하여 AWS 요청에 서명하고 인증할 수 있습니다. 이렇게 AWS 요청을 인증하려면 AWS Identity and Access Management (IAM) 역할을 생성 및 구성하고 자격 증명 공급자가 사용자를 대신하여 역할을 수임할 수 있도록 역할에 적절한 IAM 정책을 연결해야 합니다. AWS IoT Core 및 IAM 에 대한 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT 단원을 참조하세요.

AWS IoT 에서는 디바이스가 전송 계층 보안(TLS) 프로토콜로 SNI(Server Name Indication) 확장을 전송하고 host_name 필드에 전체 엔드포인트 주소를 제공해야 합니다. host_name 필드에는 호출하는 엔드포인트가 포함되어야 하며 다음과 같아야 합니다.

- aws iot describe-endpoint --endpoint-type iot:CredentialProvider에서 반환한 endpointAddress

올바른 host_name 값 없이 디바이스에서 시도하는 연결은 실패합니다.

다음 다이어그램은 자격 증명 공급자의 워크플로우를 보여줍니다.



1. AWS IoT Core 디바이스는 보안 토큰에 대해 자격 증명 공급자에게 HTTPS 요청을 합니다. 이 요청에는 인증을 위한 디바이스 X.509 인증서가 포함되어 있습니다.
2. 자격 증명 공급자는 AWS IoT Core 인증 및 권한 부여 모듈에 요청을 전달하여 인증서를 검증하고 디바이스에 보안 토큰을 요청할 권한이 있는지 확인합니다.

3. 인증서가 유효하고 보안 토큰을 요청할 권한이 있는 경우 AWS IoT Core 인증 및 권한 부여 모듈은 성공을 반환합니다. 그렇지 않은 경우, 디바이스에 예외를 전송합니다.
4. 인증서 검증에 성공하고 나면 자격 증명 공급자가 AWS Security Token Service (AWS STS)을(를) 호출하여 사용자가 생성한 IAM 역할을 수임합니다.
5. AWS STS 는 권한이 제한된 임시 보안 토큰을 자격 증명 공급자에게 반환합니다.
6. 자격 증명 공급자가 이 보안 토큰을 디바이스로 반환합니다.
7. 디바이스는 보안 토큰을 사용하여 서명 버전 4로 AWS AWS 요청에 서명합니다.
8. 요청된 서비스가 서명을 검증하고 사용자가 자격 증명 공급자를 위해 생성한 IAM 역할에 연결된 액세스 정책들에 대한 요청에 권한을 부여하기 위해 IAM을 호출합니다.
9. IAM이 서명을 성공적으로 검증하고 요청에 권한을 부여하면 요청은 성공합니다. 그렇지 않은 경우, IAM이 예외를 전송합니다.

다음 단원에서는 인증서를 사용하여 보안 토큰을 얻는 방법을 설명합니다. 이 단원은 사용자가 이미 디바이스를 등록했고 이 디바이스에 사용자의 인증서를 생성하여 활성화했다는 가정하에 작성되었습니다.

인증서를 사용하여 보안 토큰을 얻는 방법

1. 자격 증명 공급자가 사용자의 디바이스를 대신하여 수임하는 IAM 역할을 구성합니다. 다음 신뢰 정책을 역할에 연결합니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"Service": "credentials.iot.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }
}
```

   호출하려는 각 AWS 서비스에 대해 액세스 정책을 역할에 연결합니다. 자격 증명 공급자는 다음과 같은 정책 변수를 지원합니다.

   - `credentials-iot:ThingName`
   - `credentials-iot:ThingTypeName`

- `credentials-iot:AwsCertificateId`

  디바이스가 AWS 서비스에 대한 요청에서 사물 이름을 제시하는 경우, 자격 증명 공급자가 보안 토큰에 `credentials-iot:ThingName` 및 `credentials-iot:ThingTypeName`을(를) context 변수로 추가합니다. 디바이스가 요청에 사물 이름을 제시하지 않더라도 자격 증명 공급자는 context 변수로 `credentials-iot:AwsCertificateId`를 제공합니다. `x-amzn-iot-thingname` HTTP 요청 헤더의 값으로 그 사물 이름을 전달하세요.

  이 세 가지 변수는 AWS IoT Core 정책이 아니라 IAM 정책을 위해서만 작동합니다.

2. 다음 단계(역할 별칭 생성)를 수행하는 사용자가 새로 생성된 역할을 AWS IoT Core로 전달할 권한을 가지고 있는지 확인합니다. 다음 정책은 AWS 사용자에게 `iam:GetRole` 및 `iam:PassRole` 권한을 모두 부여합니다. `iam:GetRole` 권한은 방금 생성한 역할에 대한 정보를 사용자가 가져올 수 있도록 허용합니다. `iam:PassRole` 권한을 통해 사용자는 역할을 다른 AWS 서비스에 전달할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::123456789012:role/your role name"
    }
}
```

3. AWS IoT Core 역할 별칭을 생성합니다. AWS 서비스에 직접 호출할 디바이스는에 연결할 때 사용할 역할 ARN을 알아야 합니다 AWS IoT Core. 역할 ARN을 하드 코딩하면 역할 ARN이 바뀔 때마다 디바이스를 업데이트해야 하기 때문에 이것은 좋은 방법이 아닙니다. `CreateRoleAlias` API를 사용하여 역할 ARN을 가리키는 역할 별칭을 생성하는 것이 더 바람직합니다. 역할 ARN이 바뀔 경우 그 역할 별칭을 업데이트하면 됩니다. 디바이스에 어떤 변경도 할 필요가 없습니다. 이 API에서 사용하는 파라미터는 다음과 같습니다.

`roleAlias`

필수 사항입니다. 역할 별칭을 식별하는 임의 문자열입니다. 이것은 역할 별칭 데이터 모델에서 기본 키 역할을 합니다. 1-128자 길이이며 영숫자와 =, @, - 기호만 사용해야 합니다. 알파벳 대문자와 소문자가 허용됩니다. 역할 별칭은 대/소문자를 구분합니다.

`roleArn`

필수 사항입니다. 역할 별칭이 가리키는 역할의 ARN입니다.

`credentialDurationSeconds`

선택 사항. 자격 증명의 유효 시간(초)입니다. 최소값은 900초(15분)입니다. 최대값은 43,200초(12시간)입니다. 기본 값은 3,600초(1시간)입니다.

> **Important**
>
> AWS IoT Core 자격 증명 공급자는 최대 수명이 43,200초(12시간)인 자격 증명을 발급할 수 있습니다. 자격 증명이 최대 12시간 동안 유효하면 자격 증명을 더 오래 캐싱하여 자격 증명 공급자에 대한 호출 횟수를 줄일 수 있습니다. `credentialDurationSeconds` 값은 역할 별칭이 참조하는 IAM 역할의 최대 세션 기간보다 작거나 같아야 합니다. 자세한 내용은 Identity and Access Management 사용 설명서의 역할 최대 세션 기간(AWS API) 수정을 AWS 참조하세요.

이 API에 대한 자세한 내용은 CreateRoleAlias 단원을 참조하세요.

4. 디바이스 인증서에 정책을 연결합니다. 디바이스 인증서에 연결된 정책은 역할을 위임할 수 있는 디바이스 권한을 부여해야 합니다. 다음의 예와 같이 역할 별칭에 `iot:AssumeRoleWithCertificate` 작업 권한을 부여하면 됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/your role
 alias"
        }
```

```
    ]
}
```

5. 자격 증명 공급자에게 HTTPS 요청을 하여 보안 토큰을 얻습니다. 다음 정보를 제공합니다.

   - Certificate: 이 요청은 TLS 상호 인증을 통한 HTTP 요청이므로 이 요청 중에 인증서와 프라이빗 키를 클라이언트에 제공해야 합니다. 인증서를 등록할 때 사용한 것과 동일한 인증서와 프라이빗 키를 사용합니다 AWS IoT Core.

     디바이스가와 통신하고 있는지(AWS IoT Core 가장 가장하는 서비스가 아님) 확인하려면 서버 인증을 참조하고 링크를 따라 적절한 CA 인증서를 다운로드한 다음 디바이스에 복사합니다.
   - RoleAlias: 자격 증명 공급자를 위해 생성한 역할 별칭의 이름입니다. 역할 별칭 이름은 대/소문자를 구분하며 생성된 역할 별칭과 일치해야 합니다 AWS IoT Core.
   - ThingName: 사물을 등록할 때 생성한 AWS IoT Core 사물 이름입니다. 이것은 `x-amzn-iot-thingname` HTTP 헤더의 값으로 전달됩니다. 이 값은 AWS IoT Core 또는 IAM 정책에서 사물 속성을 정책 변수로 사용하는 경우에만 필요합니다.

     > **Note**
     >
     > `x-amzn-iot-thingname`에서 제공한 ThingName은 인증서에 할당된 AWS IoT 사물 리소스의 이름과 일치해야 합니다. 일치하지 않으면 403 오류가 반환됩니다.

   에서 다음 명령을 실행 AWS CLI 하여에 대한 자격 증명 공급자 엔드포인트를 가져옵니다 AWS 계정. 이 API에 대한 자세한 내용은 DescribeEndpoint 단원을 참조하세요. FIPS 지원 엔드포인트는 AWS IoT Core- 자격 증명 공급자 엔드포인트 섹션을 참조하세요.

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

   다음 JSON 객체는 describe-endpoint 명령의 샘플 출력입니다. 여기에는 보안 토큰을 요청하기 위해 사용하는 `endpointAddress`가 포함되어 있습니다.

   ```
   {
       "endpointAddress": "your_aws_account_specific_prefix.credentials.iot.your
    region.amazonaws.com"
   }
   ```

이 엔드포인트를 사용하여 HTTPS 요청을 통해 자격 증명 공급자에게 보안 토큰을 반환하라고 요청합니다. 다음 예제 명령은 curl을 사용하지만 모든 HTTP 클라이언트를 사용할 수 있습니다.

> **Note**
>
> roleAlias 이름은 대/소문자를 구분하며 생성된 역할 별칭과 일치해야 합니다 AWS IoT.

```
curl --cert your certificate --key your private key -H "x-amzn-iot-thingname: your thing name" --cacert AmazonRootCA1.pem https://your endpoint /role-aliases/your role alias/credentials
```

이 명령은 accessKeyId, secretAccessKey, sessionToken, 그리고 만료를 포함하는 보안 토큰 객체를 반환합니다. 다음 JSON 객체는 curl 명령의 샘플 출력입니다.

```
   {"credentials":{"accessKeyId":"access key","secretAccessKey":"secret access key","sessionToken":"session token","expiration":"2018-01-18T09:18:06Z"}}
```

그런 다음 accessKeyId, secretAccessKey및 sessionToken 값을 사용하여 AWS 서비스에 대한 요청에 서명할 수 있습니다. end-to-end 데모는 AWS 보안 블로그의 자격 AWS 증명 공급자 블로그 게시물을 사용하여 디바이스에서 하드 코딩된 AWS IoT 자격 증명의 필요성을 제거하는 방법을 참조하세요.

IAM을 통한 교차 계정 액세스

AWS IoT Core 를 사용하면 보안 주체가 소유 AWS 계정 하지 않은에 정의된 주제를 게시하거나 구독할 수 있습니다. 교차 계정 액세스는 IAM 정책 및 IAM 역할을 생성하고 정책을 역할에 연결하여 구성합니다.

먼저 AWS 계정에서 여러 사용자 및 인증서를 생성했을 때와 마찬가지로, IAM 정책 생성에 설명된 대로 고객 관리형 IAM 정책을 생성합니다.

AWS IoT Core 레지스트리에 등록된 디바이스의 경우 다음 정책은 디바이스의 사물 이름과 일치하는 클라이언트 ID를 AWS IoT Core 사용하여 디바이스에 연결하고 ## ### 디바이스의 사물 이름인에 게시할 수 my/topic/*thing-name* 있는 권한을 디바이스에 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

AWS IoT Core 레지스트리에 등록되지 않은 디바이스의 경우 다음 정책은 디바이스에 계정의 (123456789012) AWS IoT Core 레지스트리에 `client1` 등록된 사물 이름을 사용하여에 연결하고 이름이 접두사가 인 클라이언트 ID별 주제에 게시 AWS IoT Core 할 수 있는 권한을 부여합니다. `my/topic/`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
            ]
        }
    ]
}
```

다음으로 역할을 생성하여 IAM 사용자에게 권한 위임의 단계를 따릅니다. 액세스를 공유하려는 AWS 계정 의 계정 ID를 입력합니다. 마지막 단계로 방금 생성한 정책을 역할에 연결합니다. 나중에 액세스를 부여한 AWS 계정 ID를 수정해야 할 경우 다음의 신뢰 정책 형식을 사용할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::567890123456:user/MyUser"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

의 데이터 보호 AWS IoT Core

AWS 공동 책임 모델의 데이터 보호에 적용됩니다 AWS IoT Core. 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 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 데이터 프라이

버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 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.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 AWS IoT 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.

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

AWS IoT 디바이스는 데이터를 수집하고 해당 데이터에 대한 일부 조작을 수행한 다음 해당 데이터를 다른 웹 서비스로 전송합니다. 짧은 기간 동안 디바이스에 일부 데이터를 저장하도록 선택할 수 있습니다. 사용자는 해당 유휴 데이터에 대해 모든 데이터 보호를 제공할 책임이 있습니다. 디바이스가 데이터를 전송할 때이 섹션의 뒷부분에서 설명한 대로 TLS 연결을 통해 데이터를 AWS IoT전송합니다. AWS IoT 디바이스는 모든 AWS 서비스로 데이터를 전송할 수 있습니다. 각 서비스의 데이터 보안에 대한 자세한 내용은 해당 서비스에 대한 설명서를 참조하세요. CloudWatch Logs에 로그를 쓰고 AWS IoT API 호출을 로깅하도록 구성할 AWS IoT 수 있습니다 AWS CloudTrail. 이러한 서비스의 데이터 보

안에 대한 자세한 내용은 Amazon CloudWatch에 대한 인증 및 액세스 제어와 AWS KMS 관리형 키를 사용한 CloudTrail 로그 파일 암호화를 참조하세요.

의 데이터 암호화 AWS IoT

기본적으로 전송 AWS IoT 중 및 저장 데이터는 모두 암호화됩니다. 전송 중인 데이터는 TLS를 사용하여 암호화되고 저장 데이터는 AWS 소유 키를 사용하여 암호화됩니다.는 AWS Key Management Service AWS KMS keys ()의 고객 관리형(KMS 키)을 AWS IoT 지원합니다AWS KMS. 그러나 Device Advisor 및 AWS IoT Wireless는 만 사용하여 고객 데이터를 암호화 AWS 소유 키 합니다.

의 전송 보안 AWS IoT Core

전송 계층 보안(TLS)은 컴퓨터 네트워크를 통한 보안 통신을 위해 설계된 암호화 프로토콜입니다. AWS IoT Core Device Gateway를 사용하려면 디바이스에서 게이트웨이로의 연결에 TLS를 사용하여 전송 중 모든 통신을 암호화해야 합니다. TLS는 AWS IoT Core에서 지원하는 애플리케이션 프로토콜(MQTT, HTTP, WebSocket)의 기밀성을 달성하는 데 사용됩니다. TLS 지원은 다수의 프로그래밍 언어 및 운영 체제를 지원합니다. 내의 데이터는 특정 AWS 서비스에 의해 암호화 AWS 됩니다. 다른 AWS 서비스의 데이터 암호화에 대한 자세한 내용은 해당 서비스의 보안 설명서를 참조하세요.

내용

- TLS 프로토콜
- 보안 정책
- 의 전송 보안을 위한 중요 참고 사항 AWS IoT Core
- LoRaWAN 무선 디바이스를 위한 전송 보안

TLS 프로토콜

AWS IoT Core 는 다음 버전의 TLS 프로토콜을 지원합니다.

- TLS 1.3
- TLS 1.2

를 사용하면 도메인 구성에서 TLS 설정(TLS 1.2 및 TLS 1.3용)을 AWS IoT Core구성할 수 있습니다. 자세한 내용은 ??? 단원을 참조하십시오.

보안 정책

보안 정책은 클라이언트와 서버 간의 TLS 협상 중에 지원되는 프로토콜과 암호를 결정하는 TLS 프로토콜과 해당 암호의 조합입니다. 필요에 따라 미리 정의된 보안 정책을 사용하도록 디바이스를 구성할 수 있습니다. AWS IoT Core 는 사용자 지정 보안 정책을 지원하지 않습니다.

디바이스를 연결할 때 디바이스에 대해 사전 정의된 보안 정책 중 하나를 선택할 수 있습니다 AWS IoT Core. 에서 사전 정의된 최신 보안 정책의 이름에는 릴리스된 연도 및 월을 기반으로 한 버전 정보가 AWS IoT Core 포함됩니다. 기본 사전 정의 보안 정책은 `IoTSecurityPolicy_TLS13_1_2_2022_10`입니다. 보안 정책을 지정하려면 AWS IoT 콘솔 또는를 사용할 수 있습니다 AWS CLI. 자세한 내용은 ??? 단원을 참조하십시오.

다음 테이블에는 AWS IoT Core 에서 지원하는 가장 최근의 사전 정의된 보안 정책이 설명되어 있습니다. `IotSecurityPolicy_`가 제목 행의 정책 이름에서 제거되어 해당 위치에 맞게 조정되었습니다.

보안 정책	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
TCP 포트	443/8443/8883	443/8443/8883	443/8443/8883	443	8443/8883	443	8443/8883
TLS 프로토콜							
TLS 1.2		✓	✓	✓	✓	✓	✓
TLS 1.3	✓	✓					
TLS 암호							
TLS_AES_128_GCM_SHA256	✓	✓					
TLS_AES_256_GCM_SHA384	✓	✓					
TLS_CHACHA20_POLY	✓	✓					

보안 정책	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
305_SHA25 6							
ECDHE-RSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA		✓	✓	✓	✓	✓	✓

보안 정책	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
AES128-SHA256		✓	✓	✓		✓	✓
AES128-SHA		✓	✓	✓	✓	✓	✓
AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
AES256-SHA256		✓	✓	✓	✓	✓	✓
AES256-SHA		✓	✓	✓	✓	✓	✓
DHE-RSA-AES256-SHA						✓	✓
ECDHE-ECDSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA256		✓	✓	✓	✓	✓	✓

보안 정책	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
ECDHE-ECDSA-AES128-SHA		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-SHA		✓	✓	✓	✓	✓	✓

> **Note**
>
> TLS12_1_0_2016_01는 AWS 리전 ap-east-1, ap-northeast-2, ap-south-1, ap-southeast-2, ca-central-1, cn-north-1, cn-northwest-1, eu-north-1, eu-west-2, eu-west-3, me-south-1, sa-east-1, us-east-2, us-gov-west-1, us-gov-west-2, us-west-1에서만 사용할 수 있습니다.
> TLS12_1_0_2015_01는 AWS 리전 ap-northeast-1, ap-southeast-1, eu-central-1, eu-west-1, us-east-1, us-west-2에서만 사용할 수 있습니다.

의 전송 보안을 위한 중요 참고 사항 AWS IoT Core

MQTT를 AWS IoT Core 사용하여에 연결하는 디바이스의 경우 TLS는 디바이스와 브로커 간의 연결을 암호화하고 TLS 클라이언트 인증을 AWS IoT Core 사용하여 디바이스를 식별합니다. 자세한 내용

은 클라이언트 인증을 참조하세요. HTTP를 AWS IoT Core 사용하여에 연결하는 디바이스의 경우 TLS는 디바이스와 브로커 간의 연결을 암호화하고 인증은 AWS 서명 버전 4에 위임됩니다. 자세한 내용은 AWS 일반 참조에서 서명 버전 4를 사용하여 요청에 서명을 참조하세요.

디바이스를 연결할 때 SNI(Server Name Indication) 확장을 AWS IoT Core보낼 필요는 없지만 적극 권장됩니다. 다중 계정 등록, 사용자 지정 도메인, VPC 엔드포인트 및 구성된 TLS 정책과 같은 기능을 사용하려면, SNI 확장을 사용하고 `host_name` 필드에 전체 엔드포인트 주소를 제공해야 합니다. `host_name` 필드에는 호출하는 엔드포인트가 포함되어야 합니다. 이 엔드포인트는 다음 중 하나여야 합니다.

- `aws iot describe-endpoint --endpoint-type iot:Data-ATS`에서 반환한 `endpointAddress`
- `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"`에서 반환한 `domainName`

올바르지 않거나 잘못된 `host_name` 값을 가진 디바이스에서 시도한 연결은 실패합니다. AWS IoT Core 는 사용자 지정 인증의 인증 유형에 대해 CloudWatch에 실패를 기록합니다.

AWS IoT Core 는 SessionTicket TLS 확장을 지원하지 않습니다.

LoRaWAN 무선 디바이스를 위한 전송 보안

LoRaWAN 디바이스는 LoRaWAN™ 보안: LoRa Alliance™에 Gemalto, Actility 및 Semtech이 준비한 백서에 설명된 보안 연습을 따릅니다.

LoRaWAN 디바이스를 사용한 전송 보안에 대한 자세한 내용은 LoRaWAN 데이터 및 전송 보안 섹션을 참조하세요.

의 데이터 암호화 AWS IoT

데이터 보호는 전송 중(로/에서 이동하는 동안 AWS IoT Core) 및 저장 중(디바이스 또는 다른 AWS 서비스에 저장되는 동안) 데이터를 보호하는 것을 말합니다. 로 전송되는 모든 데이터는 MQTT, HTTPS 및 WebSocket 프로토콜을 사용하여 TLS 연결을 통해 전송되므로 전송 중에 기본적으로 안전합니다. AWS IoT Core 는 디바이스에서 데이터를 AWS IoT Core 수집한 다음 추가 처리를 위해 다른 AWS 서비스로 전송합니다. 다른 AWS 서비스의 데이터 암호화에 대한 자세한 내용은 해당 서비스의 보안 설명서를 참조하세요. 자세한 내용은 유휴 시 저장된 데이터 암호화를 참조하세요.

FreeRTOS는 키 스토리지를 추상화하여 암호화 객체에 액세스하고 세션을 관리하는 PKCS #11 라이브러리를 제공합니다. 이 라이브러리를 사용하여 디바이스에 저장된 데이터를 암호화하는 것은 사용

자의 책임입니다. 자세한 내용은 FreeRTOS퍼블릭 키 암호화 표준(PKCS) #11 라이브러리를 참조하세요.

에서 저장 데이터 암호화 AWS IoT Core

기본적으로 모든 저장 AWS IoT Core 데이터는 AWS 소유 키를 사용하여 암호화됩니다.는 AWS Key Management Service ()의 대칭 고객 관리형 키 AWS IoT Core 도 지원합니다AWS KMS. 고객 관리형 키를 사용하면 AWS 계정에서 AWS KMS 키를 생성, 소유 및 관리할 수 있습니다. AWS IoT Core 는 KMS 키를 사용하여 저장 데이터를 암호화합니다. 키 정책 생성 및 유지 관리를 포함하여, 이러한 KMS 키를 완전히 제어할 수 있습니다. 이러한 키에 대한 권한을 제어 AWS KMS 하기 위해에 액세스하는 역할에 대한 IAM 정책을 구성할 수도 있습니다.

AWS 소유 키

AWS 소유 키는 AWS 서비스가 여러 AWS 계정에서 사용하기 위해 소유하고 관리하는 KMS 키 모음입니다. AWS 서비스는 AWS 소유 키를 사용하여 데이터를 보호할 수 있습니다. 기본적으로는 AWS 소유 키를 사용하여 저장 데이터를 AWS IoT Core 암호화합니다. 이러한 키는 해당 서비스에서 관리합니다. AWS 소유 키를 보거나 관리하거나 사용할 수 없습니다. 그러나 이러한 키를 보호하기 위해 조치를 취할 필요는 없습니다.

AWS 소유 키에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 소유 키를 참조하세요.

고객 관리형 키

고객 관리형 키는 사용자가 생성, 소유 및 관리하는 AWS 계정의 KMS 키입니다. 키 정책 생성 및 유지 관리를 포함하여, 이러한 AWS KMS 키를 완전히 제어할 수 있습니다. 이러한 키에 대한 권한을 제어 AWS KMS 하기 위해에 액세스하는 역할에 대한 IAM 정책을 구성할 수도 있습니다. 고객 관리형 KMS 키를 사용하여 데이터를 암호화 AWS IoT Core 하도록를 구성할 수 있습니다.

고객 관리형 키에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드의 고객 관리형 키를 참조하세요.

고객 관리형 키를 옵트인하려면 다음 단계를 AWS IoT Core따르세요.

주제

- 1단계: 고객 관리형 키 만들기
- 2단계: KMS 키를 사용할 수 있는 AWS IoT Core 권한을 부여하는 IAM 역할 생성
- 3단계:에서 고객 관리형 키에 옵트인 AWS IoT Core
- 4단계: AWS IoT Core 컨트롤 플레인 작업에 필요한 추가 권한

- 5단계: 키 관리
- 6단계: 키 상태 모니터링

1단계: 고객 관리형 키 만들기

AWS KMS 콘솔 또는 AWS KMS CLI 명령을 사용하여 대칭 고객 관리형 키를 생성할 수 있습니다. keySpec는 SYMMETRIC_DEFAULT여야 하고, keyUsage는 ENCRYPT_DECRYPT여야 합니다.

> **Note**
>
> AWS IoT Core 는 고객 관리형 AWS KMS 키에 대한 SYMMETRIC_DEFAULT 키 사양 및 ENCRYPT_DECRYPT 키 사용이 있는 키만 지원합니다.

다음은 AWS IoT Core 고객 관리형 키에와 함께 사용할 수 있는 KMS 키를 생성하는 AWS CLI 명령의 예입니다.

```
aws kms create-key --key-spec SYMMETRIC_DEFAULT --key-usage ENCRYPT_DECRYPT --region
 us-west-2
```

다음은 명령의 출력 예제입니다.

```
{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": "2024-09-19T11:45:23.982000-07:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
```

```
        "MultiRegion": false
    }
}
```

자세한 내용은 AWS Key Management Service 개발자 안내서의 대칭 고객 관리형 키 생성을 참조하세요.

키 정책

고객 관리형 키를 만들 때 키 정책을 지정할 수 있습니다. 키 정책에서는 고객 관리형 키에 대한 액세스를 제어합니다. 모든 고객 관리형 키에는 키를 사용할 수 있는 사람과 키를 사용하는 방법을 결정하는 문장이 포함된 정확히 하나의 키 정책이 있어야 합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 키 정책을 참조하세요.

AWS IoT Core 는 계정의 IAM 역할을 사용하여 고객 관리형 키에 액세스합니다. 사용자 지정 키 정책을 사용하는 경우, 이 키에서 생성된 IAM 역할에 다음 권한이 있는지 확인합니다.

- `kms:DescribeKey`
- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:ReEncryptTo`
- `kms:ReEncryptFrom`

2단계: KMS 키를 사용할 수 있는 AWS IoT Core 권한을 부여하는 IAM 역할 생성

AWS IoT Core 가 저장 데이터를 암호화하기 위해 생성한 KMS 키를 사용하려면 계정에 IAM 역할도 생성해야 합니다.이 역할은가 KMS 키에 액세스하는 것으로 가정할 AWS IoT Core 수 있습니다.

가 역할을 AWS IoT Core 수임하도록 허용하려면 역할에 다음 신뢰 정책이 있어야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "iot.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
```

```
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:iot:us-west-2:111122223333:*"
            }
        }
    }
}
```

IAM 역할에 연결된 IAM 정책에 KMS 키에 대한 다음 권한이 있는지 확인합니다.

- kms:DescribeKey
- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo
- kms:ReEncryptFrom

다음은 고객 관리형 키에 필요한 권한이 있는 IAM 정책의 예입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowIoTToAccessKMSResource",
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:ReEncryptTo",
                "kms:ReEncryptFrom",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": [
                "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws-crypto-ec:vendor": "iot.amazonaws.com"
                }
            }
        }
    ]
}
```

자세한 정보는 AWS Identity and Access Management 사용 설명서의 IAM 사용자에게 권한을 위임할 역할 생성을 참조하세요.

3단계:에서 고객 관리형 키에 옵트인 AWS IoT Core

이전 단계를 모두 완료한 후, AWS IoT Core에서 고객 관리형 키를 사용해 `update-encryption-configuration` CLI 명령을 실행하여 옵트인합니다. 고객 관리형 키를 옵트인하면 AWS 계정의 모든 AWS IoT Core 리소스가 지정된 AWS KMS 키를 사용하여 암호화됩니다.

1. 를 AWS IoT Core 사용하여에서 고객 관리형 키를 옵트인하려면 `update-encryption-configuration` CLI 명령을 AWS CLI실행합니다.

   ```
   aws iot update-encryption-configuration --encryption-type "CUSTOMER_MANAGED_KMS_KEY"
    \
   --kms-access-role-arn "arn:aws:iam::111122223333:role/myrole" \
   --kms-key-arn "arn:aws:kms:us-
   west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab" --region us-west-2
   ```

2. 를 AWS IoT Core 사용하여에서 고객 관리형 키를 확인하려면 `describe-encryption-configuration` CLI 명령을 AWS CLI실행합니다.

   ```
   aws iot describe-encryption-configuration --region us-west-2
   ```

   에서 고객 관리형 키를 활성화한 경우 출력 AWS IoT Core은 다음과 같을 수 있습니다.

   ```
   {
       "encryptionType": "CUSTOMER_MANAGED_KMS_KEY",
       "kmsKeyArn": "arn:aws:kms:us-
   west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
       "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
       "configurationDetails": {
           "configurationStatus": "HEALTHY"
   ```

```
    },
    "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
}
```

`lastModifiedDate` 필드는 암호화 구성이 마지막으로 업데이트된 날짜를 나타냅니다.

고객 관리형 키를 활성화하지 않은 경우 출력은 다음과 같을 수 있습니다.

```
{
    "encryptionType": "AWS_OWNED_KMS_KEY",
    "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
}
```

4단계: AWS IoT Core 컨트롤 플레인 작업에 필요한 추가 권한

고객 관리형 키를 옵트인하면 AWS 계정에 속한 모든 AWS IoT Core 리소스가 제공된 KMS 키로 암호화됩니다. 이제 모든 컨트롤 플레인 작업을 수행하려면 호출자에게 AWS IoT Core 리소스의 특정 작업에 필요한 `kms:Decrypt` 권한 외에도 KMS 키에 대한 권한이 있어야 합니다. 호출자에게 `kms:Decrypt` 권한이 없고 데이터 암호화 또는 복호화가 필요한 API 직접 호출(예: `GetPolicy`)을 수행하는 경우, 호출자는 `UnauthorizedException`을 수신합니다.

예를 들어, `GetPolicy`를 직접 호출할 때, API 직접 호출이 성공하려면 고객 관리형 KMS 키에 대한 `iot:GetPolicy` 및 `kms:Decrypt` 권한이 모두 필요합니다.

> **Note**
>
> 암호화 구성에 사용되는 키에 대한 AWS KMS 권한을 부여하도록 IAM 사용자 또는 역할을 업데이트할 때 KMS 키 정책이 각 IAM 사용자 또는 역할에 필요한 권한도 부여하는지 확인합니다.

AWS KMS 에 대한 권한 **`UpdateEncryptionConfiguration`**

고객 관리형 키를 옵트인하거나 키 구성을 수정하려면 `UpdateEncryptionConfiguration` API 호출에 KMS 키에 대한 다음 AWS KMS 권한이 필요합니다.

- `kms:DescribeKey`
- `kms:Decrypt`
- `kms:Encrypt`

- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo
- kms:ReEncryptFrom

AWS KMS 다른 모든 컨트롤 플레APIs에 대한 권한

대부분의 컨트롤 플레인 API는 고객 관리형 키가 활성화된 경우 kms:Decrypt 권한이 필요합니다. 그러나 특정 API는 다음과 같은 추가 권한이 필요하지 않습니다.

AWS KMS 권한이 필요하지 않은 APIs

List* 및 Delete* API는 이 버킷에 포함되지 않습니다. 고객은 언제든지 List* 또는 Delete* 컨트롤 플레인 API를 간접적으로 호출할 수 있으며, 호출자에게 kms:Decrypt 권한이 없더라도 해당 API 직접 호출이 성공합니다. List* 및 Delete* API가 복호화를 수행하지 않아 고객 관리형 키가 비정상이더라도 이러한 API 직접 호출은 성공합니다.

- List* API - 모든 나열 작업(예: ListThings, ListPolicies, ListCertificates)
- Delete* API - 모든 삭제 작업(예: DeleteThing, DeletePolicy, DeleteCertificate)

5단계: 키 관리

AWS IoT Core 는 고객 관리형 키 구성을 정기적으로 확인하여 암호화 및 복호화 작업이 영향을 받지 않도록 합니다. 이러한 상태 확인은 1분마다 한 번씩 실행되며 암호화 및 복호화 작업을 위해 AWS KMS 키 및 연결된 IAM 역할에 액세스하고 사용할 수 AWS IoT Core있는지 확인합니다.

HEALTHY

AWS IoT Core 는 지정된 IAM 역할을 통해 AWS KMS 키에 성공적으로 액세스하고 암호화/복호화 작업을 수행할 수 있습니다. 모든 구성 요소가 올바르게 작동합니다.

UNHEALTHY

AWS IoT Core 는 AWS KMS 키에 액세스하거나 사용할 수 없습니다. 이로 인해 새 암호화 작업을 방해하고 서비스 기능에 영향을 미칠 수 있습니다. errorCode 필드는 문제가 해당 키 또는 IAM 역할에 있는지 여부를 나타냅니다.

키 상태에 영향을 미칠 수 있는 고객 작업

몇 가지 고객 작업으로 인해 키 상태가 HEALTHY에서 UNHEALTHY로 변경될 수 있습니다.

키 관련 작업

- AWS KMS 키 삭제 - 키 삭제를 예약하면 키가 `Pending deletion` 상태이므로 사용할 수 없습니다.
- AWS KMS 키 비활성화 - KMS 키를 비활성화하면 더 이상 암호화/복호화 작업에 사용할 수 없습니다.
- 키 삭제 예약 - 삭제가 완료되면 키를 사용할 수 없게 됩니다.
- 키 정책 수정 - AWS IoT Core 액세스에 필요한 권한 제거
- 키 사용 권한 변경 - 필요한 AWS KMS 작업 제한

IAM 역할 관련 작업

- IAM 역할 삭제 - 키에 액세스하기 위해 역할을 수임할 수 AWS IoT Core 없음
- 역할 권한 수정 - 역할 정책에서 필요한 AWS KMS 권한 제거
- 신뢰 정책 변경 - AWS IoT Core 서비스가 역할을 수임하지 못하도록 방지
- 제한적인 조건 추가 -가 역할을 AWS IoT Core 사용하지 못하게 하는 조건

계정 수준 작업

- 교차 계정 키 액세스 변경 - 서로 다른 계정의 키에 대한 권한 수정
- 서비스 제어 정책(SCPs) - AWS KMS 액세스를 제한하는 조직 수준 정책
- 계정 수준 IAM 정책 - 키 액세스를 재정의하거나 키 액세스와 충돌하는 정책

> **Important**
>
> 에서 사용하는 AWS KMS 키, IAM 역할 또는 정책에 대한 모든 변경 사항은 먼저 개발 환경에서 테스트해야 AWS IoT Core 합니다. 변경 후 키 상태를 면밀히 모니터링하여 AWS IoT Core 기능에 영향을 주지 않도록 합니다.

암호화 구성 업데이트

한 고객 관리형 키에서 다른 고객 관리형 키로 또는 AWS 소유 키와 고객 관리형 키 간에 변경 AWS IoT Core 하도록에서 암호화 구성을 업데이트합니다.

구성을 다른 고객 관리형 키로 변경하려면 다음을·수행하세요.

1. the section called “1단계: 고객 관리형 키 만들기”의 단계에 따라 새 고객 관리형 키를 생성합니다.

2. 업데이트 기간 동안 이전 키와 새 키 모두에 대한 권한을 포함하도록 IAM 역할 정책을 업데이트합니다.
3. 다음과 같이 새 키를 사용하도록 암호화 구성을 업데이트합니다.

```
aws iot update-encryption-configuration --encryption-type "CUSTOMER_MANAGED_KMS_KEY"
 \
--kms-access-role-arn "arn:aws:iam::111122223333:role/myrole" \
--kms-key-arn "arn:aws:kms:us-west-2:111122223333:key/new-key-id"
```

고객 관리형 키에서 AWS 소유 키로 구성을 변경하려면:

```
aws iot update-encryption-configuration --encryption-type "AWS_OWNED_KMS_KEY"
```

> **Note**
>
> 새 고객 관리형 키의 암호화 구성을 업데이트할 때, 이전 키와 새 키 모두에 계속 액세스할 수 있어야 작업이 성공할 수 있습니다.

일반적인 장애 시나리오 및 영향

다음 표에서는 키가 삭제되거나 비활성화되는 일반적인 장애 시나리오를 설명합니다.

시나리오	즉각적인 영향	장기적인 결과
키가 비활성화됨	모든 새 암호화/복호화 작업이 즉시 실패합니다.	키가 다시 활성화되거나 교체될 때까지 서비스 중단
키 삭제 예약	키 상태가 삭제 보류 중으로 변경되고 모든 암호화/복호화 작업이 실패합니다.	삭제 완료 시 자동 서비스 장애 발생
키가 영구적으로 삭제됨	모든 작업이 즉각적, 영구적으로 실패	영구 데이터 손실 및 암호화된 데이터 복구 불가
키 정책이 잘못 수정됨	AWS IoT Core 키에 대한 액세스 권한을 잃음	정책이 수정될 때까지 서비스 장애 발생

시나리오	즉각적인 영향	장기적인 결과
IAM 역할 삭제됨	AWS IoT Core 는 키에 액세스하기 위한 역할을 수임할 수 없습니다.	전체 암호화 서비스 장애 발생
IAM 역할이 잘못 수정됨	AWS IoT Core 는 역할을 수임하거나 역할을 사용하여 키에 액세스할 수 없습니다.	IAM 역할이 수정될 때까지 서비스 장애 발생

예방 및 모범 사례

우발적인 키 삭제 또는 비활성화를 방지하고 서비스 장애 위험을 최소화하려면 다음을 수행합니다.

키 수명 주기 정책 구현

키 생성, 교체 및 사용 중지에 대한 명확한 절차를 설정합니다. 어떤 키가 어떤 AWS IoT Core 리소스에서 사용되는지 문서화하고 활성 키의 인벤토리를 유지 관리합니다.

IAM 정책을 사용하여 키 삭제 제한

권한이 없는 사용자가 중요한 암호화 키를 삭제하거나 비활성화하지 못하도록 방지하는 IAM 정책을 생성합니다. 조건을 사용하여 키 삭제 작업에 대한 추가 승인이 필요합니다.

CloudTrail 로깅 활성화

CloudTrail을 통해 모든 AWS KMS 키 작업을 모니터링하여 무단 또는 우발적 키 관리 활동을 탐지합니다. 키 삭제, 비활성화 또는 정책 변경에 대한 알림을 설정합니다.

키 교체 절차 테스트

비프로덕션 환경에서 키 교체 절차를 정기적으로 테스트하여, 키 관련 장애로부터 신속하게 복구할 수 있는지 확인합니다.

키 백업 유지 관리

AWS KMS 키 구성 요소를 내보낼 수는 없지만 필요한 경우 키 ARNs, 정책 및 관련 AWS IoT Core 구성에 대한 세부 레코드를 유지하여 키 교체를 신속하게 수행할 수 있습니다.

키 상태 모니터링

`CMK.Health` 지표를 지속적으로 모니터링하고 주요 상태 변경에 대한 자동 알림을 설정합니다. 자동화된 응답을 구현하여 키 관련 문제를 신속하게 해결합니다.

> **Important**
>
> 프로덕션 환경에서 키 업데이트 절차를 구현하기 전에 항상 개발 환경에서 키 업데이트 절차를 테스트합니다. 문서화된 롤백 계획을 수립하고, 긴급 상황 발생 시 키 교체 절차를 신속하게 실행할 수 있는지 확인합니다.

6단계: 키 상태 모니터링

정기 점검 AWS IoT Core 실행의 일부로 CloudWatch 지표 및 로그가 생성되어 고객 관리형 키 구성의 상태를 파악할 수 있습니다.

AWS IoT Core 는 1분에 한 번 이상 CloudWatch에 `CMK.Health` 지표를 내보냅니다. 지표는가 데이터를 암호화하고 해독하는 데 사용하는 고객 관리형 키의 상태에 AWS IoT Core 대한 정보를 제공합니다.

`CMK.Health` 지표에는 다음과 같은 값이 있습니다.

- 값은 1: AWS IoT Core 이며 데이터 암호화 및 복호화에 암호화 키를 성공적으로 사용할 수 있습니다.
- 값은 입니다`0`.는 데이터 암호화 및 복호화에 암호화 키를 사용할 수 AWS IoT Core 없습니다.

AWS IoT Core 는 암호화 키의 상태가 변경될 때 AWS IoT 도 V2 로그를 내보냅니다. 이러한 로그는 상태 업데이트에 대한 추가 세부 정보를 제공합니다. 이러한 로그를 보려면 AWS IoT V2 로그를 활성화해야 합니다. `HEALTHY` 로그는 `INFO` 수준에서 내보내고 `UNHEALTHY` 로그는 `ERROR` 수준에서 내보냅니다. 로그 수준에 대한 자세한 내용은 로그 수준 섹션을 참조하세요.

다음 예는 고객 관리형 키의 상태 업데이트를 나타내기 AWS IoT Core 위해에서 내보낸 CloudWatch 로그 항목입니다.

주요 상태 변경을 효과적으로 모니터링하고 변경에 대응하려면 다음을·수행하세요.

1. `CMK.Health` 지표에 대한 CloudWatch 경보를 설정합니다.

```
aws cloudwatch put-metric-alarm --region us-west-2 \
  --alarm-name "IoTCore-CMK-Health-Alert" \
  --alarm-description "Alert when IoT Core CMK health is unhealthy" \
  --metric-name "CMK.Health" \
  --namespace "AWS/IoT" \
  --statistic "Minimum" \
```

```
  --period 300 \
  --evaluation-periods 1 \
  --threshold 1 \
  --comparison-operator "LessThanThreshold" \
  --alarm-actions "arn:aws:sns:us-west-2:111122223333:iot-alerts"
```

2. Enable AWS IoT V2 로깅을 사용하여 오류 코드 및 메시지와 함께 세부 상태 변경 이벤트를 캡처합니다.
3. 문제 해결을 위해 구성 상태를 확인합니다.

```
aws iot describe-encryption-configuration --region us-west-2
```

4. `errorCode` 필드를 검사하여 UNHEALTHY 상태를 조사합니다.
   - `KMS_KEY_VALIDATION_ERROR` - AWS KMS 키 관련 문제(비활성화, 삭제 또는 정책 문제)
   - `ROLE_VALIDATION_ERROR` - IAM 역할 관련 문제(삭제, 정책 문제 또는 신뢰 문제)

UNHEALTHY에서 HEALTHY로

암호화 키의 상태가에서 `UNHEALTHY`로 업데이트되면는 다음 형식으로 AWS IoT V2 로그 메시지를 내`HEALTHY` AWS IoT Core 보냅니다.

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "8421693b-f4f0-4e4a-9235-0cff8bab897d",
    "accountId": "111122223333",
    "status": "SUCCESS",
    "cmkStatus": "HEALTHY",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "eventType": "CmkHealthCheck"
}
```

HEALTHY에서 UNHEALTHY로

암호화 키의 상태가에서 `HEALTHY`로 업데이트되면는 다음 형식으로 AWS IoT V2 로그 메시지를 내`UNHEALTHY` AWS IoT Core 보냅니다.

```
{
    "timestamp": "2017-08-10 15:37:23.476",
```

```
    "logLevel": "ERROR",
    "traceId": "8421693b-f4f0-4e4a-9235-0cff8bab897d",
    "accountId": "111122223333",
    "status": "FAILURE",
    "cmkStatus": "UNHEALTHY",
    "errorCode": "KMS_KEY_VALIDATION_ERROR / ROLE_VALIDATION_ERROR",
    "errorMessage": "Error message on why there was a failure",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "eventType": "CmkHealthCheck"
}
```

> **Warning**
>
> 키 상태가 되면 UNHEALTHY AWS IoT Core 작업이 즉시 실패합니다. 이 경우에 키 구성, IAM 역할 권한 및 정책을 검토합니다. `CMK.Health` 지표에서 상태 변경을 모니터링합니다. 구성을 검토한 후에도 작업이 계속 실패하면, 계정 관리자 또는 AWS 지원 센터에 문의하여 추가 지원을 받으세요.

AWS CloudTrail 이벤트

암호화 복호화 작업에 대한 KMS 키 AWS IoT Core사용을 모니터링할 수도 있습니다. `Decrypt` AWS IoT Core 는 KMS 키에 대해 `DescribeKey`, `ReEncrypt`, 및 `GenerateDataKeyWithoutPlaintext` 작업을 수행하여 저장 시 저장된 AWS 계정에 속한 데이터를 암호화/복호화합니다.

`DescribeKey`, `Decrypt`, `ReEncrypt` 및 `GenerateDataKeyWithoutPlaintext`에 대한 CloudTrail 이벤트가 있습니다. 이러한 이벤트는에서 호출한 AWS KMS 작업을 모니터링 AWS IoT Core 하여 고객 관리형 키로 암호화된 데이터에 액세스합니다.

`Decrypt` 예

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
```

```
        "accessKeyId": "**********************",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "*****"
            },
            "attributes": {
                "creationDate": "2024-09-16T20:23:39Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "iot.amazonaws.com"
    },
    "eventTime": "2024-09-16T20:32:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "iot.amazonaws.com",
    "userAgent": "iot.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "kms-arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "aws-crypto-ec:vendor": "iot.amazonaws.com",
            "branch-key-id": "111122223333",
            "type": "branch:ACTIVE"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "1afb6d98-8388-455d-8b48-e62c9e0cf7f4",
    "eventID": "b59a5f16-0d98-46d8-a590-0e040a48b39b",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

에 대한 자격 증명 및 액세스 관리 AWS IoT

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 AWS IoT 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- IAM 자격 증명을 통한 인증
- 정책을 사용하여 액세스 관리
- AWS IoT 에서 IAM을 사용하는 방법
- AWS IoT 자격 증명 기반 정책 예제
- AWS 에 대한 관리형 정책 AWS IoT
- AWS IoT 자격 증명 및 액세스 문제 해결

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조AWS IoT 자격 증명 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(AWS IoT 에서 IAM을 사용하는 방법 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(AWS IoT 자격 증명 기반 정책 예제 참조)

IAM 자격 증명을 통한 인증

에서 AWS IoT ID는 디바이스(X.509) 인증서, Amazon Cognito ID 또는 IAM 사용자 또는 그룹일 수 있습니다. 이 주제에서는 IAM 자격 증명에 대해서만 설명합니다. 에서 AWS IoT 지원하는 다른 자격 증명에 대한 자세한 내용은 섹션을 참조하세요클라이언트 인증.

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증해야 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 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자가 필요한 작업 목록은 IAM 사용자 설명서의 루트 사용자 자격 증명이 필요한 작업을 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 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 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 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 정의를 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 지정해야 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 WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 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 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 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 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

AWS IoT 에서 IAM을 사용하는 방법

IAM을 사용하여에 대한 액세스를 관리하기 전에 사용할 수 있는 IAM 기능을 이해해야 AWS IoT합니다 AWS IoT. AWS IoT 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

주제

- AWS IoT 자격 증명 기반 정책
- AWS IoT 리소스 기반 정책

- AWS IoT 태그 기반 권한 부여
- AWS IoT IAM 역할

AWS IoT 자격 증명 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스 및 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. AWS IoT 는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 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 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

다음 표에는 IAM IoT 작업, 연결된 AWS IoT API 및 작업이 조작하는 리소스가 나열되어 있습니다.

정책 작업	AWS IoT API	리소스
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`* **Note** ARN에 AWS 계정 지정된는 인증서가 전송되는 계정이어야 합니다.
iot:AddThingToThingGroup	AddThingToThingGroup	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* `arn:aws:iot:` *`region`*`:`*`account-id`* `:thing/`*`thing-name`*
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	없음

정책 작업	AWS IoT API	리소스
iot:AttachPolicy	AttachPolicy	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 또는 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:AttachSecurityProfile	AttachSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:AttachThingPrincipal	AttachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:CancelCertificateTransfer	CancelCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id* **Note** ARN에 AWS 계정 지정된는 인증서가 전송되는 계정이어야 합니다.
iot:CancelJob	CancelJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:CancelJobExecution	CancelJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*

정책 작업	AWS IoT API	리소스
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	없음
iot:CreateAuthorizer	CreateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*
iot:CreateDimension	CreateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:CreateJob	CreateJob	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateJobTemplate	CreateJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateKeysAndCertificate	CreateKeysAndCertificate	*
iot:CreatePolicy	CreatePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*

정책 작업	AWS IoT API	리소스
iot:CreatePolicyVersion	CreatePolicyVersion	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name* **Note** 이는 IAM AWS IoT 정책이 아닌 정책이어야 합니다.
iot:CreateRoleAlias	CreateRoleAlias	(파라미터: roleAlias) `arn:aws:iot:` *region*`:`*account-id* `:rolealias/` *role-alias-name*
iot:CreateSecurityProfile	CreateSecurityProfile	`arn:aws:iot:` *region*`:`*account-id* `:securityprofile/` *security-profile-name* `arn:aws:iot:` *region*`:`*account-id* `:dimension/` *dimension-name*
iot:CreateThing	CreateThing	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
iot:CreateThingGroup	CreateThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* 사용할 경우 생성 그룹 및 상위 그룹에 필요
iot:CreateThingType	CreateThingType	`arn:aws:iot:` *region*`:`*account-id* `:thingtype/` *thing-type-name*
iot:CreateTopicRule	CreateTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
iot:DeleteAuthorizer	DeleteAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizer/` *authorizer-name*

정책 작업	AWS IoT API	리소스
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DeleteDimension	DeleteDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DeleteJobTemplate	DeleteJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-template-id*
iot:DeleteJobExecution	DeleteJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeleteRegistrationCode	DeleteRegistrationCode	*
iot:DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DeleteSecurityProfile	DeleteSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*

정책 작업	AWS IoT API	리소스
iot:DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeleteThingGroup	DeleteThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeleteThingType	DeleteThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DeleteTopicRule	DeleteTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeprecateThingType	DeprecateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name* (파라미터: authorizerName) 없음
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DescribeCertificate	DescribeCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	없음
iot:DescribeEndpoint	DescribeEndpoint	*

정책 작업	AWS IoT API	리소스
iot:DescribeEventConfigurations	DescribeEventConfigurations	없음
iot:DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DescribeJobExecution	DescribeJobExecution	없음
iot:DescribeJobTemplate	DescribeJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-template-id*
iot:DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DescribeThingGroup	DescribeThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DescribeThingRegistrationTask	DescribeThingRegistrationTask	없음
iot:DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* 또는 arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

정책 작업	AWS IoT API	리소스
iot:DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DetachSecurityProfile	DetachSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DisableTopicRule	DisableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:EnableTopicRule	EnableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:GetEffectivePolicies	GetEffectivePolicies	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:GetIndexingConfiguration	GetIndexingConfiguration	없음
iot:GetJobDocument	GetJobDocument	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:GetLoggingOptions	GetLoggingOptions	*
iot:GetPolicy	GetPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:GetPolicyVersion	GetPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*

정책 작업	AWS IoT API	리소스
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 또는 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	없음
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	없음
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	없음
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	없음

정책 작업	AWS IoT API	리소스
iot:ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* thingGroupName 파라미터를 사용하는 경우
iot:ListJobTemplates	ListJobs	없음
iot:ListOutgoingCertificates	ListOutgoingCertificates	*
iot:ListPolicies	ListPolicies	*
iot:ListPolicyPrincipals	ListPolicyPrincipals	*
iot:ListPolicyVersions	ListPolicyVersions	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListPrincipalPolicies	ListPrincipalPolicies	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
iot:ListPrincipalThings	ListPrincipalThings	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
iot:ListRoleAliases	ListRoleAliases	없음
iot:ListTargetsForPolicy	ListTargetsForPolicy	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListThingGroups	ListThingGroups	없음
iot:ListThingGroupsForThing	ListThingGroupsForThing	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thing/`*`thing-name`*

정책 작업	AWS IoT API	리소스
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	없음
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	없음
iot:ListThingTypes	ListThingTypes	*
iot:ListThings	ListThings	*
iot:ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:ListTopicRules	ListTopicRules	*
iot:ListV2LoggingLevels	ListV2LoggingLevels	없음
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	없음
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

정책 작업	AWS IoT API	리소스
iot:RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
iot:SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:SetLoggingOptions	SetLoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
iot:SetV2LoggingLevel	SetV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:SetV2LoggingOptions	SetV2LoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
iot:StartThingRegistrationTask	StartThingRegistrationTask	없음
iot:StopThingRegistrationTask	StopThingRegistrationTask	없음
iot:TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

정책 작업	AWS IoT API	리소스
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	없음
iot:TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateDimension	UpdateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:UpdateEventConfigurations	UpdateEventConfigurations	없음
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	없음
iot:UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:UpdateSecurityProfile	UpdateSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot:UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

정책 작업	AWS IoT API	리소스
iot:UpdateThingGroup	UpdateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

의 정책 작업은 작업 앞에 접두사를 AWS IoT 사용합니다`iot:`. 예를 들어 API를 사용하여에 등록된 모든 IoT 사물을 나열할 수 AWS 계정 있는 권한을 부여하려면 해당 정책에 `iot:ListThings` 작업을 `ListThings` 포함합니다. 정책 설명에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다.는이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 AWS IoT 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "ec2:action1",
      "ec2:action2"
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "iot:Describe*"
```

AWS IoT 작업 목록을 보려면 IAM 사용 설명서의 에서 정의한 작업을 AWS IoT 참조하세요.

Device Advisor 작업

다음 표에는 IAM IoT Device Advisor 작업, 연결된 AWS IoT Device Advisor API 및 해당 작업이 조작하는 리소스가 나열되어 있습니다.

정책 작업	AWS IoT API	리소스
iotdeviceadvisor:C	CreateSuiteDefinition	없음

정책 작업	AWS IoT API	리소스
reateSuiteDefinition		
iotdeviceadvisor:DeleteSuiteDefinition	DeleteSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:GetSuiteDefinition	GetSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:GetSuiteRun	GetSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-run-id*
iotdeviceadvisor:GetSuiteRunReport	GetSuiteRunReport	arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/ *suite-definition-id* /*suite-run-id*
iotdeviceadvisor:ListSuiteDefinitions	ListSuiteDefinitions	없음
iotdeviceadvisor:ListSuiteRuns	ListSuiteRuns	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor:ListTagsForResource	ListTagsForResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*

정책 작업	AWS IoT API	리소스
iotdevice advisor:StartSuiteRun	StartSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdevice advisor:TagResource	TagResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*
iotdevice advisor:UntagResource	UntagResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*
iotdevice advisor:UpdateSuiteDefinition	UpdateSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdevice advisor:StopSuiteRun	StopSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*

AWS IoT Device Advisor의 정책 작업은 작업 앞에 접두사를 사용합니다`iotdeviceadvisor:`. 예를 들어 ListSuiteDefinitions API를 사용하여에 등록된 모든 제품군 정의를 나열할 수 AWS 계정 있는 권한을 부여하려면 해당 정책에 `iotdeviceadvisor:ListSuiteDefinitions` 작업을 포함합니다.

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": "*"
```

AWS IoT 리소스

정책 작업	AWS IoT API	리소스
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** ARN에 AWS 계정 지정된는 인증서가 전송되는 계정이어야 합니다.
iot:AddThingToThingGroup	AddThingToThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	없음
iot:AttachPolicy	AttachPolicy	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* 또는 `arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*

정책 작업	AWS IoT API	리소스
iot:AttachThingPrincipal	AttachThingPrincipal	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
iot:CancelCertificateTransfer	CancelCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** ARN에 AWS 계정 지정된는 인증서가 전송되는 계정이어야 합니다.
iot:CancelJob	CancelJob	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id*
iot:CancelJobExecution	CancelJobExecution	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	없음
iot:CreateAuthorizer	CreateAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizer/` *authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*

정책 작업	AWS IoT API	리소스
iot:CreateJob	CreateJob	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateJobTemplate	CreateJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:CreateKeysAndCertificate	CreateKeysAndCertificate	*
iot:CreatePolicy	CreatePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
CreatePolicyVersion	iot:CreatePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** 이는 IAM AWS IoT 정책이 아닌 정책이어야 합니다.
iot:CreateRoleAlias	CreateRoleAlias	(파라미터: roleAlias) arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*

정책 작업	AWS IoT API	리소스
iot:CreateThing	CreateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:CreateThingGroup	CreateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 사용할 경우 생성 그룹 및 상위 그룹에 필요
iot:CreateThingType	CreateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:CreateTopicRule	CreateTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DeleteJobExecution	DeleteJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeleteJobTemplate	DeleteJobTemplate	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*

정책 작업	AWS IoT API	리소스
iot:DeleteRegistrationCode	DeleteRegistrationCode	*
iot:DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DeleteThingGroup	DeleteThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeleteThingType	DeleteThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DeleteTopicRule	DeleteTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DeprecateThingType	DeprecateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
iot:DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name* (파라미터: authorizerName) 없음
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:DescribeCertificate	DescribeCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

정책 작업	AWS IoT API	리소스
iot:DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	없음
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	없음
iot:DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:DescribeJobExecution	DescribeJobExecution	없음
iot:DescribeJobTemplate	DescribeJobTemplate	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot:DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:DescribeThingGroup	DescribeThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DescribeThingRegistrationTask	DescribeThingRegistrationTask	없음
iot:DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*

정책 작업	AWS IoT API	리소스
iot:DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* 또는 arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:DisableTopicRule	DisableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:EnableTopicRule	EnableTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:GetEffectivePolicies	GetEffectivePolicies	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:GetIndexingConfiguration	GetIndexingConfiguration	없음
iot:GetJobDocument	GetJobDocument	arn:aws:iot: *region*:*account-id* :job/*job-id*
iot:GetLoggingOptions	GetLoggingOptions	*
iot:GetPolicy	GetPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:GetPolicyVersion	GetPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*

정책 작업	AWS IoT API	리소스
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 또는 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	없음
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	없음
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	없음
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	없음

정책 작업	AWS IoT API	리소스
iot:ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* thingGroupName 파라미터를 사용하는 경우
iot:ListJobTemplates	ListJobTemplates	없음
iot:ListOutgoingCertificates	ListOutgoingCertificates	*
iot:ListPolicies	ListPolicies	*
iot:ListPolicyPrincipals	ListPolicyPrincipals	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListPolicyVersions	ListPolicyVersions	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListPrincipalPolicies	ListPrincipalPolicies	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
iot:ListPrincipalThings	ListPrincipalThings	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
iot:ListRoleAliases	ListRoleAliases	없음
iot:ListTargetsForPolicy	ListTargetsForPolicy	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
iot:ListThingGroups	ListThingGroups	없음
iot:ListThingGroupsForThing	ListThingGroupsForThing	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thing/`*`thing-name`*

정책 작업	AWS IoT API	리소스
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	없음
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	없음
iot:ListThingTypes	ListThingTypes	*
iot:ListThings	ListThings	*
iot:ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:ListTopicRules	ListTopicRules	*
iot:ListV2LoggingLevels	ListV2LoggingLevels	없음
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	없음
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

정책 작업	AWS IoT API	리소스
iot:RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
iot:SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
iot:SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:SetLoggingOptions	SetLoggingOptions	*
iot:SetV2LoggingLevel	SetV2LoggingLevel	*
iot:SetV2LoggingOptions	SetV2LoggingOptions	*
iot:StartThingRegistrationTask	StartThingRegistrationTask	없음
iot:StopThingRegistrationTask	StopThingRegistrationTask	없음
iot:TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

정책 작업	AWS IoT API	리소스
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	없음
iot:TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
iot:UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:UpdateEventConfigurations	UpdateEventConfigurations	없음
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	없음
iot:UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot:UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:UpdateThingGroup	UpdateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

ARN 형식에 대한 자세한 내용은 Amazon 리소스 이름(ARNs) 및 AWS 서비스 네임스페이스를 참조하세요.

리소스를 생성하기 위한 작업과 같은 일부 AWS IoT 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

AWS IoT 리소스 유형 및 해당 ARNs 목록을 보려면 IAM 사용 설명서의 에서 정의한 리소스를 AWS IoT 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS IoT가 정의한 작업을 참조하세요.

Device Advisor 리소스

AWS IoT Device Advisor IAM 정책에 대한 리소스 수준 제한을 정의하려면 제품군 정의 및 제품군 실행에 다음 리소스 ARN 형식을 사용합니다.

스위트 정의 리소스 ARN 형식

```
arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id
```

스위트 실행 리소스 ARN 형식

```
arn:aws:iotdeviceadvisor:region:account-id:suiterun/suite-definition-id/suite-run-id
```

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 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

AWS IoT 는 자체 조건 키 세트를 정의하고 일부 전역 조건 키 사용을 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

AWS IoT 조건 키

AWS IoT 조건 키	설명	형식
`aws:RequestTag/${tag-key}`	사용자가 AWS IoT에 수행하는 요청에 있는 태그 키입니다.	문자열
`aws:ResourceTag/${tag-key}`	AWS IoT 리소스에 연결된 태그의 태그 키 구성 요소입니다.	문자열
`aws:TagKeys`	요청의 리소스와 연결된 모든 태그 키 이름의 목록입니다.	문자열

AWS IoT 조건 키 목록을 보려면 IAM 사용 설명서의에 대한 조건 키를 참조하세요 AWS IoT. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 에서 정의한 작업을 AWS IoT 참조하세요.

예제

자격 AWS IoT 증명 기반 정책의 예를 보려면 섹션을 참조하세요AWS IoT 자격 증명 기반 정책 예제.

AWS IoT 리소스 기반 정책

리소스 기반 정책은 지정된 보안 주체가 AWS IoT 리소스에 대해 수행할 수 있는 작업과 조건을 지정하는 JSON 정책 문서입니다.

AWS IoT 는 IAM 리소스 기반 정책을 지원하지 않습니다. 그러나 AWS IoT 리소스 기반 정책을 지원합니다. 자세한 내용은 AWS IoT Core 정책 단원을 참조하십시오.

AWS IoT 태그 기반 권한 부여

AWS IoT 리소스에 태그를 연결하거나 요청에서 태그를에 전달할 수 있습니다 AWS IoT. 태그에 근거하여 액세스를 제어하려면 `iot:ResourceTag/key-name`, `aws:RequestTag/key-name`또는 `aws:TagKeys`조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. 자세한 내용은 IAM 정

책에 태그 사용 단원을 참조하십시오. AWS IoT 리소스 태그 지정에 대한 자세한 내용은 섹션을 참조하세요AWS IoT 리소스에 태그 지정.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그를 기반으로 AWS IoT 리소스 보기에서 확인할 수 있습니다.

AWS IoT IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 엔터티입니다.

에서 임시 자격 증명 사용 AWS IoT

임시 보안 인증을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. AssumeRole 또는 GetFederationToken과 같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 얻습니다.

AWS IoT 는 임시 자격 증명 사용을 지원합니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스가 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수 없습니다.

AWS IoT 는 서비스 연결 역할을 지원하지 않습니다.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 서비스 역할을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

AWS IoT 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 AWS IoT 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI또는 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 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- AWS IoT 콘솔 사용
- 사용자가 자신이 권한을 볼 수 있도록 허용
- 태그를 기반으로 AWS IoT 리소스 보기
- 태그를 기반으로 AWS IoT Device Advisor 리소스 보기

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 AWS IoT 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 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.

- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

AWS IoT 콘솔 사용

AWS IoT 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은의 AWS IoT 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

이러한 엔터티가 AWS IoT 콘솔을 계속 사용할 수 있도록 하려면 엔터티에 AWS 관리형 정책 도 연결합니다`AWSIoTFullAccess`. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자가 자신이 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
```

```
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

태그를 기반으로 AWS IoT 리소스 보기

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 AWS IoT 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 사물을 보도록 허용하는 정책을 생성할 수 있는 방법을 보여줍니다. 하지만 사물 태그 `Owner`에 해당 사용자의 사용자 이름 값이 있는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 태스크를 완료하는 데 필요한 권한도 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListBillingGroupsInConsole",
            "Effect": "Allow",
            "Action": "iot:ListBillingGroups",
            "Resource": "*"
        },
        {
            "Sid": "ViewBillingGroupsIfOwner",
            "Effect": "Allow",
           "Action": "iot:DescribeBillingGroup",
            "Resource": "arn:aws:iot:*:*:billinggroup/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
        }
```

```
        }
    ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. 라는 사용자가 AWS IoT 결제 그룹을 보richard-roe려고 하면 결제 그룹에 Owner=richard-roe 또는 태그가 지정되어야 합니다owner=richard-roe. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 Owner는 Owner 및 owner 모두와 일치합니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

태그를 기반으로 AWS IoT Device Advisor 리소스 보기

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 AWS IoT Device Advisor 리소스에 대한 액세스를 제어할 수 있습니다. 다음 예는 특정 스위트 정의를 볼 수 있는 정책을 생성하는 방법을 보여줍니다. 그러나 스위트 정의 태그가 SuiteType을(를) MQTT 값으로 설정하는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 작업을 완료하는 데 필요한 권한도 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewSuiteDefinition",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:GetSuiteDefinition",
            "Resource": "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/SuiteType": "MQTT"}
        }
        }
    ]
}
```

AWS 에 대한 관리형 정책 AWS IoT

사용자, 그룹 및 역할에 권한을 추가하려면 직접 정책을 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더 쉽습니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하기 위해서는 시간과

전문 지식이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이 정책은 일반적인 사용 사례를 다루며 사용자의 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스에서 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 작업을 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한 여러 서비스에 걸쳐 있는 직무에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 ReadOnlyAccess AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스를 제공합니다. 서비스에서 새 기능을 시작하면 AWS 가 새 작업 및 리소스에 대한 읽기 전용 권한을 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한AWS 관리형 정책을 참조하세요.

> **Note**
>
> AWS IoT 는 AWS IoT 및 IAM 정책 모두에서 작동합니다. 이 주제에서는 컨트롤 플레인 및 데이터 영역 API 작업에 대한 정책 작업을 정의하는 IAM 정책에만 대해서만 설명합니다. 또한 AWS IoT Core 정책 섹션도 참조하세요.

AWS 관리형 정책: AWSIoTConfigAccess

`AWSIoTConfigAccess` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 연결된 자격 증명에 모든 AWS IoT 구성 작업에 액세스할 수 있는 권한을 부여합니다. 이 정책은 데이터 처리와 저장에 영향을 줄 수 있습니다. 에서이 정책을 보려면 AWSIoTConfigAccess를 AWS Management Console참조하세요.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iot` - AWS IoT 데이터를 검색하고 IoT 구성 작업을 수행합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AcceptCertificateTransfer",
                "iot:AddThingToThingGroup",
                "iot:AssociateTargetsWithJob",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
                "iot:CancelCertificateTransfer",
                "iot:CancelJob",
                "iot:CancelJobExecution",
                "iot:ClearDefaultAuthorizer",
                "iot:CreateAuthorizer",
                "iot:CreateCertificateFromCsr",
                "iot:CreateJob",
                "iot:CreateKeysAndCertificate",
                "iot:CreateOTAUpdate",
                "iot:CreatePolicy",
                "iot:CreatePolicyVersion",
                "iot:CreateRoleAlias",
                "iot:CreateStream",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:CreateThingType",
                "iot:CreateTopicRule",
                "iot:DeleteAuthorizer",
                "iot:DeleteCACertificate",
                "iot:DeleteCertificate",
                "iot:DeleteJob",
                "iot:DeleteJobExecution",
                "iot:DeleteOTAUpdate",
                "iot:DeletePolicy",
                "iot:DeletePolicyVersion",
                "iot:DeleteRegistrationCode",
                "iot:DeleteRoleAlias",
```

```
                "iot:DeleteStream",
                "iot:DeleteThing",
                "iot:DeleteThingGroup",
                "iot:DeleteThingType",
                "iot:DeleteTopicRule",
                "iot:DeleteV2LoggingLevel",
                "iot:DeprecateThingType",
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachPrincipalPolicy",
                "iot:DetachThingPrincipal",
                "iot:DisableTopicRule",
                "iot:EnableTopicRule",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
```

```
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:RegisterCACertificate",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RejectCertificateTransfer",
                "iot:RemoveThingFromThingGroup",
                "iot:ReplaceTopicRule",
                "iot:SearchIndex",
                "iot:SetDefaultAuthorizer",
                "iot:SetDefaultPolicyVersion",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:SetV2LoggingOptions",
                "iot:StartThingRegistrationTask",
                "iot:StopThingRegistrationTask",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:TransferCertificate",
                "iot:UpdateAuthorizer",
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:UpdateEventConfigurations",
                "iot:UpdateIndexingConfiguration",
```

```
                "iot:UpdateRoleAlias",
                "iot:UpdateStream",
                "iot:UpdateThing",
                "iot:UpdateThingGroup",
                "iot:UpdateThingGroupsForThing",
                "iot:UpdateAccountAuditConfiguration",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DeleteAccountAuditConfiguration",
                "iot:StartOnDemandAuditTask",
                "iot:CancelAuditTask",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:CreateScheduledAudit",
                "iot:UpdateScheduledAudit",
                "iot:DeleteScheduledAudit",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:CreateSecurityProfile",
                "iot:DescribeSecurityProfile",
                "iot:UpdateSecurityProfile",
                "iot:DeleteSecurityProfile",
                "iot:AttachSecurityProfile",
                "iot:DetachSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: AWSIoTConfigReadOnlyAccess

`AWSIoTConfigReadOnlyAccess` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 연결된 자격 증명에 모든 AWS IoT 구성 작업에 읽기 전용으로 액세스할 수 있는 권한을 부여합니다. 에서이 정책을 보려면 AWSIoTConfigReadOnlyAccess를 AWS Management Console참조하세요.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iot` - IoT 구성 작업의 읽기 전용 작업을 수행합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
```

```
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:SearchIndex",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:DescribeSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
```

```
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: AWSIoTDataAccess

`AWSIoTDataAccess` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 모든 AWS IoT 데이터 작업에 대한 액세스를 허용하는 연결된 자격 증명 권한을 부여합니다. 데이터 작업을 사용하면 MQTT 또는 HTTP 프로토콜을 통해 데이터를 전송할 수 있습니다. AWS Management Console에서 이 정책을 보려면 `AWSIoTDataAccess`를 확인하세요.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iot` - AWS IoT 데이터를 검색하고 AWS IoT 메시징 작업에 대한 전체 액세스를 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
```

```
                "iot:ListNamedShadowsForThing"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: AWSIoTFullAccess

`AWSIoTFullAccess` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 연결된 자격 증명에 모든 AWS IoT 구성 및 메시징 작업에 액세할 수 있는 권한을 부여합니다. 에서이 정책을 보려면 섹션을 AWS Management Console참조하세요`AWSIoTFullAccess`.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iot` - AWS IoT 데이터를 검색하고 구성 및 메시징 작업에 대한 AWS IoT 전체 액세스를 허용합니다.
- `iotjobsdata` - AWS IoT 작업 데이터를 검색하고 AWS IoT 작업 데이터 영역 API 작업에 대한 전체 액세스를 허용합니다.

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

```
}
```

AWS 관리형 정책: AWSIoTLogging

`AWSIoTLogging` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 연결된 자격 증명에 Amazon CloudWatch Logs 그룹을 생성하고 해당 그룹에 로그를 스트리밍할 수 있는 권한을 부여합니다. 이 정책은 CloudWatch 로깅 역할에 연결됩니다. 에서이 정책을 보려면 섹션을 AWS Management Console참조하세요`AWSIoTLogging`.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `logs` - CloudWatch 로그를 검색합니다. 또한 CloudWatch Logs 그룹을 생성하고 해당 그룹에 로그를 스트리밍할 수 있도록 허용합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

```
}
```

AWS 관리형 정책: AWSIoTOTAUpdate

`AWSIoTOTAUpdate` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 연결된 자격 증명에 작업을 생성하고 AWS IoT , 서명 작업을 AWS IoT 코딩하고, AWS 코드 서명자 작업을 설명할 수 있는 액세스 권한을 부여합니다. 에서이 정책을 보려면 섹션을 AWS Management Console참조하세요`AWSIoTOTAUpdate.`

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iot` - AWS IoT 작업 및 코드 서명 작업을 생성합니다.
- `signer` - AWS 코드 서명자 작업 생성을 수행합니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateJob",
            "signer:DescribeSigningJob"
        ],
        "Resource": "*"
    }
}
```

AWS 관리형 정책: AWSIoTRuleActions

`AWSIoTRuleActions` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 AWS IoT 규칙 작업에서 AWS 서비스지원되는 모든에 대한 액세스를 허용하는 연결된 자격 증명 권한을 부여합니다. 에서이 정책을 보려면 섹션을 AWS Management Console참조하세요AWSIoTRuleActions.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- iot - 규칙 작업 메시지를 게시하기 위한 작업을 수행합니다.
- dynamodb - DynamoDB 테이블에 메시지를 삽입하거나 메시지를 DynamoDB 테이블의 여러 열로 분할합니다.
- s3 - Amazon S3 버킷에 객체를 저장합니다.
- kinesis - Amazon Kinesis 스트림 객체에 메시지를 전송합니다.
- firehose - Firehose 스트림 객체에 레코드를 삽입합니다.
- cloudwatch - CloudWatch 경보 상태를 변경하거나 메시지 데이터를 CloudWatch 지표로 전송합니다.
- sns - Amazon SNS를 사용하여 알림을 게시하기 위한 작업을 수행합니다. 이 작업은 AWS IoT SNS 주제로 범위가 지정됩니다.
- sqs - SQS 대기열에 추가할 메시지를 삽입합니다.
- es - OpenSearch Service 서비스에 메시지를 전송합니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "dynamodb:PutItem",
            "kinesis:PutRecord",
            "iot:Publish",
            "s3:PutObject",
            "sns:Publish",
            "sqs:SendMessage*",
            "cloudwatch:SetAlarmState",
            "cloudwatch:PutMetricData",
            "es:ESHttpPut",
```

```
            "firehose:PutRecord"
        ],
        "Resource": "*"
    }
}
```

AWS 관리형 정책: AWSIoTThingsRegistration

`AWSIoTThingsRegistration` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 연결된 자격 증명에 `StartThingRegistrationTask` API를 사용하여 사물을 대량으로 등록할 수 있는 권한을 부여합니다. 이 정책은 데이터 처리와 저장에 영향을 줄 수 있습니다. 에서이 정책을 보려면 섹션을 AWS Management Console참조하세요`AWSIoTThingsRegistration`.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iot` - 사물을 생성하고 대량으로 등록 시 정책 및 인증서를 연결하기 위한 작업을 수행합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateCertificateFromCsr",
                "iot:CreatePolicy",
                "iot:CreateThing",
                "iot:DescribeCertificate",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
```

```
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachThingPrincipal",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RemoveThingFromThingGroup",
                "iot:UpdateCertificate",
                "iot:UpdateThing",
                "iot:UpdateThingGroupsForThing",
                "iot:AddThingToBillingGroup",
                "iot:DescribeBillingGroup",
                "iot:RemoveThingFromBillingGroup"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS IoT AWS 관리형 정책에 대한 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 AWS IoT 이후부터의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 AWS IoT 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경	설명	Date
AWSIoTFullAccess - 기존 정책 업데이트	AWS IoT 는 사용자가 HTTP 프로토콜을 사용하여 AWS IoT 작업 데이터 영역 API 작업에	2022년 5월 11일

변경	설명	Date
	액세스할 수 있도록 허용하는 새 권한을 추가했습니다. 새로운 IAM 정책 접두사인 `iotjobsdata:` 는 AWS IoT Jobs 데이터 영역 엔드포인트에 액세스할 수 있는 보다 세분화된 액세스 제어를 제공합니다. 컨트롤 플레인 API 작업의 경우 여전히 `iot:` 접두사를 사용합니다. 자세한 내용은 AWS IoT Core HTTPS 프로토콜에 대한 정책 단원을 참조하십시오.	
AWS IoT 에서 변경 내용 추적 시작	AWS IoT 가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다.	2022년 5월 11일

AWS IoT 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS IoT 하고 수정할 수 있습니다.

주제

- 에서 작업을 수행할 권한이 없음 AWS IoT
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 AWS IoT 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

에서 작업을 수행할 권한이 없음 AWS IoT

작업을 수행할 권한이 없다는 오류가 표시되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예시 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 사물 리소스에 대한 세부 정보를 보려고 하지만 `iot:DescribeThing` 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iot:DescribeThing on resource: MyIoTThing
```

이 경우 iot:*DescribeThing* 작업을 사용하여 사물 리소스에 액세스할 수 있도록 mateojackson 사용자에 대한 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

AWS IoT Device Advisor 사용

AWS IoT Device Advisor를 사용하는 경우 다음 예제 오류는 사용자가 콘솔을 사용하여 제품군 정의에 대한 세부 정보를 보mateojackson려고 하지만 iotdeviceadvisor:*GetSuiteDefinition* 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotdeviceadvisor:GetSuiteDefinition on resource: MySuiteDefinition
```

이 경우 iotdeviceadvisor:*GetSuiteDefinition* 작업을 사용하여 *MySuiteDefinition* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS IoT에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 AWS IoT에서 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:PassRole작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 AWS IoT 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- 에서 이러한 기능을 AWS IoT 지원하는지 여부를 알아보려면 섹션을 참조하세요AWS IoT 에서 IAM을 사용하는 방법.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

로깅 및 모니터링

모니터링은 AWS IoT 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다중 지점 장애가 발생할 경우 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. 로깅 및 모니터링 절차에 대한 자세한 내용은 단원을 참조하세요모니터링 AWS IoT

모니터링 도구

AWS 는 모니터링에 사용할 수 있는 도구를 제공합니다 AWS IoT. 이러한 도구 중 일부를 구성하여 모니터링을 수행할 수 있습니다. 일부 도구는 수동 개입이 필요합니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 문제 발생 시 모니터링하고 AWS IoT 보고할 수 있습니다.

- Amazon CloudWatch 경보 – 지정한 기간 동안 단일 지표를 감시하고, 여러 기간에 대해 지정된 임곗값과 관련하여 지표 값을 기준으로 하나 이상의 작업을 수행합니다. 이 작업은 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책에 전송되는 알림입니다. CloudWatch 경보는 단순히 특정 상태에 있다고 해서 작업을 호출하지 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다. 자세한 내용은 Amazon CloudWatch를 사용하여 AWS IoT 경보 및 지표 모니터링 단원을 참조하십시오.
- Amazon CloudWatch Logs - AWS CloudTrail 또는 기타 소스에서 로그 파일을 모니터링, 저장 및 액세스합니다. 또한 Amazon CloudWatch Logs를 사용하면 Device Advisor 테스트 사례가 수행하는 중요한 단계 AWS IoT , 생성된 이벤트 및 디바이스에서 전송되거나 테스트 실행 AWS IoT Core 중에 전송되는 MQTT 메시지를 볼 수 있습니다. 이러한 로그를 사용하면 장치에서 디버깅하고 수정 조치를 취할 수 있습니다. 자세한 내용은 CloudWatch Logs AWS IoT 를 사용하여 모니터링 단원을 참조하세요. Amazon CloudWatch 사용에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 로그 파일 모니터링을 참조하세요.
- Amazon CloudWatch Events – 이벤트를 일치시키고 하나 이상의 대상 함수 또는 스트림으로 라우팅하여 값을 변경하거나 상태 정보를 캡처하거나 수정 작업을 수행합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch Events란 무엇입니까를 참조하세요.
- AWS CloudTrail 로그 모니터링 - 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs로 전송하여 실시간으로 모니터링하고, Java로 로그 처리 애플리케이션을 작성하고, CloudTrail에서 전송한 후 로그 파일이 변경되지 않았는지 확인합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 를 사용하여 AWS IoT API 호출 로깅 AWS CloudTrail 및 CloudTrail 로그 파일 작업을 참조하세요.

수동 모니터링 도구

모니터링의 또 다른 중요한 부분은 CloudWatch 경보 AWS IoT 가 다루지 않는 항목을 수동으로 모니터링하는 것입니다. AWS IoT, CloudWatch 및 기타 AWS 서비스 콘솔 대시보드는 AWS 환경 상태를 at-a-glance 볼 수 있습니다. 로그 파일도 확인하는 것이 좋습니다 AWS IoT.

- AWS IoT 대시보드에는 다음이 표시됩니다.
  - CA 인증서
  - 인증서
  - 정책
  - 규칙
  - 사물
- CloudWatch 홈 페이지에 표시되는 항목은 다음과 같습니다.

- 현재 경보 및 상태.
- 경보 및 리소스 그래프.
- 서비스 상태.

CloudWatch를 사용하여 다음을 수행할 수 있습니다.

- 사용자 지정 대시보드를 만들어 원하는 서비스 모니터링.
- 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악.
- 모든 AWS 리소스 지표를 검색하고 검색합니다.
- 문제에 대해 알려주는 경보 생성 및 편집

AWS IoT 코어에 대한 규정 준수 검증

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 내를 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact을 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 AWS 서비스 결정됩니다. 사용 시 규정 준수 책임에 대한 자세한 내용은 AWS 보안 설명서를 AWS 서비스참조하세요.

AWS IoT Core의 복원력

AWS 글로벌 인프라는 및 가용 영역을 중심으로 구축됩니다.는 물리적으로 분리되고 격리된 여러 가용 영역을 AWS 리전제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 AWS 리전높으며 중복성이 높은 네트워킹과 연결됩니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS IoT Core 는 디바이스에 대한 정보를 디바이스 레지스트리에 저장합니다. CA 인증서, 디바이스 인증서 및 디바이스 섀도우 데이터도 저장합니다. 이 데이터는 하드웨어 또는 네트워크 장애가 발생할 경우 자동으로 복제되지 않습니다.

AWS IoT Core 는 디바이스 레지스트리가 업데이트될 때 MQTT 이벤트를 게시합니다. 이러한 메시지를 사용하여 레지스트리 데이터를 백업하고 DynamoDB 테이블과 같은 특정 장소에 저장할 수 있습니다. 에서 AWS IoT Core 자동으로 생성하거나 직접 생성한 인증서를 저장할 책임은 사용자에게 있습니다. 디바이스 섀도우는 디바이스에 대한 상태 데이터를 저장하고 디바이스가 다시 온라인 상태가 되면 재전송할 수 있습니다. AWS IoT Device Advisor는 테스트 제품군 구성에 대한 정보를 저장합니다. 이 데이터는 하드웨어 또는 네트워크 장애가 발생할 경우 자동으로 복제됩니다.

AWS IoT Core 리소스는 리전별로 다르며 특별히 그렇게 하지 않는 AWS 리전 한 간에 복제되지 않습니다.

보안 모범 사례에 대한 자세한 내용은 의 보안 모범 사례 AWS IoT Core 단원을 참조하세요.

인터페이스 VPC 엔드포인트 AWS IoT Core 와 함께 사용

를 사용하면 인터페이스 VPC 엔드포인트를 사용하여 Virtual Private Cloud(VPC) 내에 IoT 컨트롤 플레인 엔드포인트와 IoT 데이터 엔드포인트를 생성할 AWS IoT Core수 있습니다. https://docs.aws.amazon.com//vpc/latest/userguide/vpce-interface.html#create-interface-endpoint 인터페이스 VPC 엔드포인트는 프라이빗 IP 주소를 사용하여 AWS 에서 실행되는 서비스에 액세스하는 데 사용할 수 있는 AWS PrivateLink기술로 구동 AWS 됩니다. 자세한 내용은 Amazon Virtual Private Cloud(VPC)를 참조하세요.

기업 네트워크와 같은 원격 네트워크의 현장에서 디바이스를 Amazon VPC에 연결하려면 네트워크와 Amazon VPC 간 연결 매트릭스에 명시된 옵션 목록을 참조하세요.

내용

- AWS IoT Core 컨트롤 플레인용 VPC 엔드포인트 생성
- AWS IoT Core 데이터 영역에 대한 VPC 엔드포인트 생성
- AWS IoT Core 자격 증명 공급자를 위한 VPC 엔드포인트 생성
- Amazon VPC 엔드포인트 생성
- 프라이빗 호스팅 영역 구성
- VPC 엔드포인트를 AWS IoT Core 통해에 대한 액세스 제어
- 제한 사항
- 를 사용하여 VPC 엔드포인트 확장 AWS IoT Core
- VPC 엔드포인트에 사용자 지정 도메인 사용
- 에 대한 VPC 엔드포인트의 가용성 AWS IoT Core
- 인터페이스 VPC 엔드포인트에서 AWS IoT Device Management 보안 터널링 사용

AWS IoT Core 컨트롤 플레인용 VPC 엔드포인트 생성

AWS IoT Core 컨트롤 플레인 API용 VPC 엔드포인트를 생성하여 디바이스를 AWS IoT 서비스 및 기타 AWS 서비스에 연결할 수 있습니다. VPC 엔드포인트를 시작하려면 인터페이스 VPC 엔드포인트를 생성하고 AWS IoT Core 를 AWS 서비스로 선택합니다. CLI를 사용하는 경우 먼저 describe-vpc-endpoint-services를 호출하여 AWS IoT Core 가 특정에 있는 가용 영역을 선택하는지 확인합니다 AWS 리전. 예를 들어 us-east-1에서 이 명령은 다음과 같습니다.

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.api
```

아래 AWS IoT Core 컨트롤 플레인용 Amazon VPC 인터페이스 엔드포인트 생성에 대한 자세한 지침을 참조하세요.

AWS IoT Core 데이터 영역에 대한 VPC 엔드포인트 생성

AWS IoT Core 데이터 영역 API용 VPC 엔드포인트를 생성하여 디바이스를 AWS IoT 서비스 및 기타 AWS 서비스에 연결할 수 있습니다. VPC 엔드포인트를 시작하려면 인터페이스 VPC 엔드포인트를 생성하고 AWS IoT Core 를 AWS 서비스로 선택합니다. CLI를 사용하는 경우 먼저 describe-vpc-endpoint-services를 호출하여 AWS IoT Core 가 특정에 있는 가용 영역을 선택하는지 확인합니다 AWS 리전. 예를 들어 us-east-1에서 이 명령은 다음과 같습니다.

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.data
```

> **Note**
>
> DNS 레코드를 자동으로 생성하는 VPC 기능이 사용 중지됩니다. 이러한 엔드포인트에 연결하려면 프라이빗 DNS 레코드를 수동으로 생성해야 합니다. 프라이빗 VPC DNS 레코드에 대한 자세한 내용은 인터페이스 엔드포인트용 프라이빗 DNS를 참조하세요. AWS IoT Core VPC 제한 사항에 대한 자세한 내용은 섹션을 참조제한 사항하세요.

MQTT 클라이언트를 VPC 엔드포인트 인터페이스에 연결하려면:

- VPC에 연결된 프라이빗 호스팅 영역에 DNS 레코드를 수동으로 생성해야 합니다. 시작하려면 프라이빗 호스팅 영역 생성을 참조하세요.

- 프라이빗 호스팅 영역 내에서 VPC 엔드포인트에 대한 각 탄력적 네트워크 인터페이스 IP에 대한 별칭 레코드를 생성합니다. 여러 VPC 엔드포인트에 대해 여러 네트워크 인터페이스 IP가 있는 경우 모

든 가중치 레코드에서 가중치가 동일한 가중치 DNS 레코드를 생성합니다. 이러한 IP 주소는 설명 필드에서 VPC 엔드포인트 ID로 필터링되는 경우 DescribeNetworkInterfaces API 호출에서 이용 가능합니다.

Amazon VPC 인터페이스 엔드포인트 생성 및 AWS IoT Core 데이터 영역에 대한 프라이빗 호스팅 영역 구성에 대한 자세한 지침은 아래를 참조하세요.

AWS IoT Core 자격 증명 공급자를 위한 VPC 엔드포인트 생성

VPC 엔드포인트 AWS IoT Core forcredential 공급자를 생성하여 클라이언트 인증서 기반 인증을 사용하여 디바이스를 연결하고 AWS 서명 버전 4 형식으로 임시 AWS 자격 증명을 가져올 수 있습니다. AWS IoT Core 자격 증명 공급자의 VPC 엔드포인트를 시작하려면 create-vpc-endpoint CLI 명령을 실행하여 인터페이스 VPC 엔드포인트를 생성하고 자격 AWS IoT Core 증명 공급자를 AWS 서비스로 선택합니다. 가 특정에 AWS IoT Core 있는 가용 영역을 선택하도록 하려면 AWS 리전먼저 describe-vpc-endpoint-services 명령을 실행합니다. 예를 들어 us-east-1에서 이 명령은 다음과 같습니다.

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.iot.credentials
```

> **Note**
>
> DNS 레코드를 자동으로 생성하는 VPC 기능이 사용 중지됩니다. 이러한 엔드포인트에 연결하려면 프라이빗 DNS 레코드를 수동으로 생성해야 합니다. 프라이빗 VPC DNS 레코드에 대한 자세한 내용은 인터페이스 엔드포인트용 프라이빗 DNS를 참조하세요. AWS IoT Core VPC 제한 사항에 대한 자세한 내용은 섹션을 참조제한 사항하세요.

HTTP 클라이언트를 VPC 엔드포인트 인터페이스에 연결하려면:

- VPC에 연결된 프라이빗 호스팅 영역에 DNS 레코드를 수동으로 생성해야 합니다. 시작하려면 프라이빗 호스팅 영역 생성을 참조하세요.

- 프라이빗 호스팅 영역 내에서 VPC 엔드포인트에 대한 각 탄력적 네트워크 인터페이스 IP에 대한 별칭 레코드를 생성합니다. 여러 VPC 엔드포인트에 대해 여러 네트워크 인터페이스 IP가 있는 경우 모든 가중치 레코드에서 가중치가 동일한 가중치 DNS 레코드를 생성합니다. 이러한 IP 주소는 설명 필드에서 VPC 엔드포인트 ID로 필터링되는 경우 DescribeNetworkInterfaces API 호출에서 이용 가능합니다.

Amazon VPC 인터페이스 엔드포인트 생성 및 AWS IoT Core 자격 증명 공급자를 위한 프라이빗 호스팅 영역 구성에 대한 자세한 지침은 아래를 참조하세요.

Amazon VPC 엔드포인트 생성

인터페이스 VPC 엔드포인트를 생성하여 로 구동되는 AWS 서비스에 연결할 수 있습니다 AWS PrivateLink. 다음 절차에 따라 AWS IoT Core 데이터 영역 또는 AWS IoT Core 자격 증명 공급자에 연결하는 인터페이스 VPC 엔드포인트를 생성합니다. 자세한 내용은 인터페이스 VPC 엔드포인트를 사용하여 AWS 서비스 액세스를 참조하세요.

> **Note**
>
> AWS IoT Core 데이터 영역 및 AWS IoT Core 자격 증명 공급자에 대한 Amazon VPC 인터페이스 엔드포인트를 생성하는 프로세스는 비슷하지만 연결을 작동하려면 엔드포인트별 변경을 수행해야 합니다.

VPC 엔드포인트 콘솔을 사용해 인터페이스 VPC 엔드포인트 생성

1. VPC 엔드포인트 콘솔로 이동한 다음 왼쪽 메뉴의 Virtual Private Cloud(VPC)에서 엔드포인트를 선택한 다음 엔드포인트 생성을 선택합니다.
2. 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 컨트롤 플레인에 대한 VPC 엔드포인트를 생성하는 경우에 대한 AWS IoT Core 컨트롤 플레인 API 엔드포인트를 선택합니다 AWS 리전. 엔드포인트 형식은 `com.amazonaws.`*`region`*`.iot.api`이(가) 될 것입니다.

     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.

FIPS(연방 정보 처리 표준) 리전에 대한 VPC 엔드포인트를 생성하는 경우에 대한 FIPS API 엔드포인트를 선택합니다 AWS 리전. 엔드포인트 형식은 `com.amazonaws.region.iot-fips.api`이(가) 될 것입니다. 이는 컨트롤 플레인에만 해당됩니다.

> **Note**
>
> 중국 리전의 AWS IoT Core 데이터 영역에 대한 서비스 이름은 형식입니다`cn.com.amazonaws.region.iot.data`. 중국 리전에서 AWS IoT Core 컨트롤 플레인의 서비스 이름은 형식입니다`com.amazonaws.region.iot.api`.

- VPC 및 서브넷에서 엔드포인트를 생성하려는 VPC 및 엔드포인트 네트워크를 생성하려는 가용 영역(AZ)을 선택합니다.
- DNS 이름 활성화에서 AWS IoT Core 데이터 영역 및 AWS IoT Core 자격 증명 공급자에 대해 이 엔드포인트에 대한 활성화가 선택되지 않았는지 확인합니다. AWS IoT Core 데이터 영역이나 AWS IoT Core 자격 증명 공급자 모두 프라이빗 DNS 이름을 아직 지원하지 않습니다.

  AWS IoT Core 컨트롤 플레인의 경우 DNS 이름 활성화가 기본적으로 선택됩니다. 이렇게 하면 컨트롤 플레인 퍼블릭 엔드포인트에 AWS IoT Core 대한 모든 요청이 대신 VPC 엔드포인트를 통해 라우팅됩니다. 이 옵션을 활성화하면 프라이빗 호스팅 영역을 구성할 필요가 없습니다.
- 보안 그룹에서 엔드포인트 네트워크 인터페이스와 연결하려는 보안 그룹을 선택합니다.
- 선택적으로 태그를 추가하거나 제거할 수 있습니다. 태그는 엔드포인트와 연결하는 데 사용하는 이름-값 페어입니다.

3. VPC 엔드포인트를 생성하려면 엔드포인트 생성을 선택합니다.

AWS PrivateLink 엔드포인트를 생성한 후 엔드포인트의 세부 정보 탭에 DNS 이름 목록이 표시됩니다. 이러한 DNS 이름 중 하나를 사용하여 프라이빗 호스팅 영역을 구성할 수 있습니다. AWS IoT Core 컨트롤 플레인을 사용하는 경우 프라이빗 호스팅 영역을 구성할 필요가 없습니다.

프라이빗 호스팅 영역 구성

> **Note**
>
> AWS IoT Core 컨트롤 플레인을 사용하고 DNS 이름 활성화를 선택한 경우 프라이빗 호스팅 영역을 구성할 필요가 없습니다. 비활성화한 경우이 절차에 따라 프라이빗 호스팅 영역을 구성해야 합니다.

이전 섹션에서 생성한 DNS 이름 중 하나를 사용하여 프라이빗 호스팅 영역을 구성할 수 있습니다.

AWS IoT Core 데이터 영역의 경우

DNS 이름은 도메인 구성 이름 또는 사용자의 `IoT:Data-ATS` 엔드포인트여야 합니다. DNS 이름 예시: *xxx*`-ats.data.iot.`*region*`.amazonaws.com`

AWS IoT Core 자격 증명 공급자의 경우

DNS 이름은 사용자의 `iot:CredentialProvider` 엔드포인트여야 합니다. DNS 이름 예시: *xxxx*`.credentials.iot.`*region*`.amazonaws.com`

AWS IoT Core 컨트롤 플레인용

DNS 이름은 AWS IoT Core 컨트롤 플레인 엔드포인트여야 합니다. AWS IoT Core 컨트롤 플레인의 DNS 이름 예는 입니다 *xxxx*`.api.iot.`*region*`.amazonaws.com`.

> **Note**
>
> AWS IoT Core 데이터 영역 및 AWS IoT Core 자격 증명 공급자에 대한 프라이빗 호스팅 영역을 구성하는 프로세스는 비슷하지만 연결을 작동하려면 엔드포인트별 변경을 수행해야 합니다.

프라이빗 호스팅 영역 생성

Route 53 콘솔로 프라이빗 호스팅 영역 생성

1. Route 53 호스팅 영역 콘솔로 이동하여 호스팅 영역 생성을 선택합니다.
2. 호스팅 영역 생성 페이지에서 다음 정보를 지정합니다.
   - 도메인 이름에는 사용자 `iot:Data-ATS` 또는 `iot:CredentialProvider` 엔드포인트의 엔드포인트 주소를 입력합니다. 다음 AWS CLI 명령으로 퍼블릭 네트워크(`aws iot describe-endpoint --endpoint-type iot:Data-ATS` 또는 `aws iot describe-endpoint --endpoint-type iot:CredentialProvider`)를 통해 엔드포인트를 가져올 수 있습니다.

     > **Note**
     >
     > 사용자 지정 도메인을 사용하는 경우 VPC 엔드포인트에 사용자 지정 도메인 사용을 참조하세요. 사용자 지정 도메인은 AWS IoT Core 자격 증명 공급자에 대해 지원되지 않습니다.

- 유형 목록에서 프라이빗 호스팅 영역을 선택합니다.
- 필요에 따라 호스팅 영역과 연결할 태그를 추가하거나 제거할 수 있습니다.

3. 프라이빗 호스팅 영역을 생성하려면 호스팅 영역 생성을 선택합니다.

자세한 내용은 프라이빗 호스팅 영역 생성을 참조하세요.

레코드 생성

프라이빗 호스팅 영역을 생성한 후 DNS에 트래픽을 해당 도메인으로 라우팅할 방법을 알려주는 레코드를 생성할 수 있습니다.

레코드 생성

1. 표시된 호스팅 영역 목록에서 이전에 생성한 프라이빗 호스팅 영역을 선택하고 레코드 생성을 선택합니다.
2. 마법사 메서드를 사용하여 레코드를 만듭니다. 콘솔이 빠른 생성 메서드를 제공하는 경우 마법사로 전환을 선택합니다.
3. 라우팅 정책에서 단순 라우팅을 선택한 후 다음을 선택합니다.
4. 레코드 구성 페이지에서 단순 레코드 정의를 선택합니다.
5. 단순 레코드 정의 페이지에서:
   - 레코드 이름에 `iot:Data-ATS` 또는 `iot:CredentialProvider` 엔드포인트를 입력합니다. 프라이빗 호스팅 영역 이름과 같아야 합니다.
   - 레코드 유형의 경우 IPv4만 지원하려면 값을 `A - Routes traffic to an IPv4 address and some AWS resources`로 유지합니다. IPv6만 지원하려면 값을 `AAAA - Routes traffic to an IPv6 address and some AWS resources`로 유지합니다. 듀얼 스택 지원(IPv4 및 IPv6 모두)을 원하는 경우, 호스팅 영역에 레코드 이름과 값/라우팅 트래픽 대상이 동일한 두 개의 레코드(`A` 및 `AAAA`)를 생성합니다.
   - 값/트래픽 라우팅 대상(Value/Route traffic to)에서 VPC 엔드포인트에 대한 별칭(Alias to VPC endpoint)을 선택합니다. 그리고 나서 리전을 선택한 다음 ???에 설명된 대로 표시된 엔드포인트 목록에서 이전에 생성한 엔드포인트를 선택합니다.
6. 단순 레코드 정의를 선택하여 레코드를 만듭니다.

VPC 엔드포인트를 AWS IoT Core 통해에 대한 액세스 제어

VPC 조건 컨텍스트 키를 사용하여 디바이스 액세스를 VPC 엔드포인트를 통해서만 허용 AWS IoT Core 하도록 제한할 수 있습니다.는 다음 VPC 관련 컨텍스트 키를 AWS IoT Core 지원합니다.

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core 는 VPC 엔드포인트에 대한 엔드포인트 정책을 지원하지 않습니다.

예를 들어 다음 정책은 사물 이름과 일치하는 클라이언트 ID를 AWS IoT Core 사용하여에 연결하고 사물 이름 접두사가 붙은 주제에 게시할 수 있는 권한을 부여합니다. 단, 디바이스가 특정 VPC 엔드포인트 ID를 사용하여 VPC 엔드포인트에 연결하는 것을 조건으로 합니다. 이 정책은 퍼블릭 IoT 데이터 엔드포인트에 대한 연결 시도를 거부합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
            }
        }

        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

제한 사항

VPC 엔드포인트는 현재 AWS IoT Core 컨트롤 플레인 엔드포인트, AWS IoT Core 데이터 엔드포인트 및 AWS IoT Core 자격 증명 공급자 엔드포인트에 대해 지원됩니다. VPC 엔드포인트는 컨트롤 플레인을 사용할 때 Federal Information Processing Standard(FIPS) 엔드포인트에서만 지원됩니다 AWS IoT Core .

IoT 컨트롤 플레인 VPC 엔드포인트의 제한 사항

이 섹션에서는 IoT 컨트롤 플레인 VPC 엔드포인트의 제한 사항을 다룹니다.

- VPC 엔드포인트는 ATS 인증서만 사용할 수 있습니다.
- 컨트롤 플레인 엔드포인트에는 사용자 지정 도메인이 지원되지 않습니다.
- FIPS 보안 정책에 대한 자세한 내용은 FIPS 보안 정책을 참조하세요.

IoT 데이터 VPC 엔드포인트 제한 사항

본 섹션에서는 IoT 데이터 VPC 엔드포인트의 제한 사항을 다룹니다.

- MQTT 유지 기간은 230초로 제한됩니다. 해당 연결 유지 시간 초과 시 자동으로 230초로 줄어듭니다.
- 각 VPC 엔드포인트는 총 100,000개의 동시 연결 디바이스를 지원합니다. 더 많은 연결이 필요한 경우 를 사용하여 VPC 엔드포인트 확장 AWS IoT Core 섹션을 참조하세요.
- VPC 엔드포인트는 ATS 인증서만 사용할 수 있습니다(사용자 지정 도메인 제외).
- VPC 엔드포인트 정책은 지원되지 않습니다.

- AWS IoT Core 데이터 영역에 AWS IoT Core 대해 생성된 VPC 엔드포인트의 경우는 영역 또는 리전 퍼블릭 DNS 레코드 사용을 지원하지 않습니다.

자격 증명 공급자 엔드포인트 제한 사항

본 섹션에서는 자격 증명 공급자 VPC 엔드포인트의 제한 사항을 다룹니다.

- VPC 엔드포인트는 ATS 인증서만 사용할 수 있습니다.
- VPC 엔드포인트 정책은 지원되지 않습니다.
- 사용자 지정 도메인은 자격 증명 공급자에 지원되지 않습니다.
- AWS IoT Core 자격 증명 공급자를 위해 생성된 VPC 엔드포인트의 경우는 영역 또는 리전 퍼블릭 DNS 레코드 사용을 지원하지 AWS IoT Core 않습니다.

를 사용하여 VPC 엔드포인트 확장 AWS IoT Core

AWS IoT Core 인터페이스 VPC 엔드포인트는 단일 인터페이스 엔드포인트를 통해 100,000개의 연결된 디바이스로 제한됩니다. 사용 사례에서 브로커에 대한 추가 동시 연결을 요구하는 경우 여러 VPC 엔드포인트를 사용하고 인터페이스 엔드포인트에서 디바이스를 수동으로 라우팅하는 것이 좋습니다. VPC 엔드포인트로 트래픽을 라우팅하기 위해 프라이빗 DNS 레코드를 생성할 때 여러 엔드포인트에 트래픽을 분산할 VPC 엔드포인트가 있는 만큼의 가중치 레코드를 생성해야 합니다.

VPC 엔드포인트에 사용자 지정 도메인 사용

VPC 엔드포인트와 함께 사용자 정의 도메인을 사용하려면 프라이빗 호스팅 영역에서 사용자 정의 도메인 이름 레코드를 생성하고 Route53에서 라우팅 레코드를 생성해야 합니다. 자세한 내용은 프라이빗 호스팅 영역 생성을 참조하세요.

> **Note**
>
> 사용자 지정 도메인은 AWS IoT Core 데이터 엔드포인트에서만 지원됩니다.

에 대한 VPC 엔드포인트의 가용성 AWS IoT Core

AWS IoT Core 인터페이스 VPC 엔드포인트는 AWS IoT Core 지원되는 모든 리전에서 사용할 수 있습니다. 자격 증명 공급자의 AWS IoT Core AWS IoT Core 인터페이스 VPC 엔드포인트는 중국 리전 및 에서 지원되지 않습니다 AWS GovCloud (US) Regions.

인터페이스 VPC 엔드포인트에서 AWS IoT Device Management 보안 터널링 사용

AWS IoT Device Management 보안 터널링은 인터페이스 VPC 엔드포인트를 지원합니다. VPC 엔드포인트를 사용하면 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 VPC와 AWS 네트워크 AWS IoT Secure Tunneling 간의 트래픽을 유지할 수 있습니다.

인터페이스 VPC 엔드포인트는 프라이빗 IP 주소를 사용하여 서비스에 비공개로 액세스할 수 있는 기술AWS PrivateLink인 로 구동됩니다. 자세한 내용은 AWS PrivateLink 가이드의 인터페이스 VPC 엔드포인트를 사용하여 AWS 서비스 액세스를 참조하세요.

내용

- 사전 조건
- VPC 엔드포인트를 통해 터널 알림 수신
- 보안 터널링을 위한 VPC 엔드포인트 생성
- 프록시 서버에서 VPC 엔드포인트 정책 구성
- 다음 단계

사전 조건

에 대한 VPC 엔드포인트를 생성하기 전에 다음이 있는지 AWS IoT Secure Tunneling확인합니다.

- VPC 엔드포인트를 생성하는 데 필요한 권한이 있는 AWS 계정입니다.
- AWS 계정의 VPC입니다.
- AWS IoT Device Management 보안 터널링 개념에 대한 이해.
- VPC 엔드포인트 정책 및 AWS Identity and Access Management (IAM)에 대한 지식

VPC 엔드포인트를 통해 터널 알림 수신

VPC 엔드포인트를 통해 터널 알림을 수신하기 위해 디바이스는 VPC 엔드포인트를 통해 AWS IoT Core 데이터 영역에 연결하고 보안 터널링 예약 MQTT 주제를 구독할 수 있습니다.

AWS IoT Core 데이터 영역에서 VPC 엔드포인트를 생성하고 구성하는 방법에 대한 지침은 AWS IoT 개발자 안내서의 인터페이스 VPC 엔드포인트와 AWS IoT Core 함께 사용을 참조하세요.

보안 터널링을 위한 VPC 엔드포인트 생성

보안 터널링 컨트롤 플레인과 프록시 서버 모두에 대한 VPC 엔드포인트를 생성할 수 있습니다.

보안 터널링을 위한 VPC 엔드포인트를 생성하려면

1. Amazon VPC 개발자 안내서의 인터페이스 엔드포인트 생성의 단계를 따릅니다.
2. 서비스 이름에서 엔드포인트 유형에 따라 다음 옵션 중 하나를 선택합니다.

   컨트롤 플레인

   - 표준: `com.amazonaws.<region>.iot.tunneling.api`
   - FIPS(FIPS 리전에서 사용 가능): `com.amazonaws.<region>.iot-fips.tunneling.api`

   프록시 서버

   - 표준: `com.amazonaws.<region>.iot.tunneling.data`
   - FIPS(FIPS 리전에서 사용 가능): `com.amazonaws.<region>.iot-fips.tunneling.data`

   *<region>*을 로 바꿉니다 AWS 리전. 예를 들어 `us-east-1`입니다.
3. 네트워크 요구 사항에 따라 VPC 엔드포인트 생성 프로세스의 나머지 단계를 완료합니다.

프록시 서버에서 VPC 엔드포인트 정책 구성

터널에 대한 연결을 승인하는 데 사용되는 클라이언트 액세스 토큰 기반 권한 부여 외에도 VPC 엔드포인트 정책을 사용하여 디바이스가 VPC 엔드포인트를 사용하여 보안 터널링 프록시 서버에 연결하는 방법을 추가로 제한할 수 있습니다. VPC 엔드포인트 정책은 IAM과 유사한 구문을 따르며 VPC 엔드포인트 자체에 구성됩니다.

프록시 서버 VPC 엔드포인트 정책에 대해 지원되는 유일한 IAM 작업은 입니다`iot:ConnectToTunnel`.

다음은 다양한 VPC 엔드포인트 정책의 예입니다.

프록시 서버 VPC 엔드포인트 정책 예제

다음 예제에서는 일반적인 사용 사례에 대한 프록시 서버 VPC 엔드포인트 정책 구성을 보여줍니다.

Example- 기본 정책

이 정책은 VPC 내의 디바이스 AWS 리전 가 엔드포인트가 생성된 동일한의 모든 터널에 모든 AWS 계정에서 연결할 수 있도록 허용합니다.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Example- 특정 AWS 계정에 대한 액세스 제한

이 정책은 VPC 엔드포인트가 특정 AWS 계정의 터널에만 연결하도록 허용합니다.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*",
                "arn:aws:iot:us-east-1:444455556666:tunnel/*"
            ]
        }
    ]
}
```

Example- 터널 엔드포인트별 연결 제한

디바이스가 터널의 소스 또는 대상 끝에 연결할 수 있도록 VPC 엔드포인트 액세스를 제한할 수 있습니다.

소스만 해당:

```
{
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "source"
                }
            }
        }
    ]
}
```

대상만 해당:

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
            }
        }
    ]
}
```

Example- 리소스 태그를 기반으로 액세스 제한

이 정책은 VPC 엔드포인트가 특정 키-값 페어로 태그가 지정된 터널에만 연결하도록 허용합니다.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
```

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Environment": "Production"
                }
            }
        }
    ]
}
```

Example- 결합된 정책 조건

이 정책은 여러 정책 요소를 결합하는 방법을 보여줍니다. 터널에가 로 `AllowConnectionsThroughPrivateLink` 설정되어 `true` 있고 클라이언트가 터널의 대상 끝에 연결되지 않는 경우에만 특정 AWS 계정의 터널에 대한 연결을 허용합니다.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/AllowConnectionsThroughPrivateLink": "true"
                }
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*"
            ],
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
            }
```

```
        }
    ]
}
```

다음 단계

에 대한 VPC 엔드포인트를 생성하고 구성한 후에는 다음 사항을 AWS IoT Secure Tunneling고려하세요.

- 엔드포인트를 통해 디바이스를 연결하여 VPC 엔드포인트 구성을 테스트합니다.
- Amazon CloudWatch 지표를 통해 VPC 엔드포인트 사용량을 모니터링합니다.
- 보안 요구 사항에 따라 VPC 엔드포인트 정책을 검토하고 업데이트합니다.

AWS IoT Device Management 보안 터널링에 대한 자세한 내용은 섹션을 참조하세요AWS IoT Secure Tunneling.

의 인프라 보안 AWS IoT

관리형 서비스 모음으로서 AWS IoT 는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

AWS 에서 게시한 API 호출을 사용하여 네트워크를 AWS IoT 통해에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.2 이상을 지원해야 합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 완전 전송 보안(PFS)이 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다. 자세한 내용은 의 전송 보안 AWS IoT Core 단원을 참조하세요.

요청은 액세스 키 ID 및 IAM 보안 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

AWS IoT Core를 사용한 프로덕션 플릿 또는 디바이스의 보안 모니터링

IoT 플릿은 다양한 기능을 수행하고 장기적으로 사용되며 지리적으로 분산된 다수의 디바이스로 구성될 수 있습니다. 이러한 특성으로 인해 플릿 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 플릿이 해커의 매력적인 대상이 되며, 디바이스 플릿을 지속적으로 보호하기 어렵게 됩니다.

AWS IoT Device Defender 는 보안 문제와 모범 사례와의 편차를 식별하는 도구를 제공하여 이러한 문제를 해결합니다. AWS IoT Device Defender를 사용하여 연결된 디바이스를 분석, 감사 및 모니터링하여 비정상적인 동작을 감지하고 보안 위험을 완화할 수 있습니다. AWS IoT Device Defender 는 디바이스 플릿을 감사하여 보안 모범 사례를 준수하고 디바이스에서 비정상적인 동작을 감지할 수 있습니다. 이를 통해 AWS IoT 디바이스 플릿에 일관된 보안 정책을 적용하고 디바이스가 손상되면 신속하게 대응할 수 있습니다. 자세한 내용은 AWS IoT Device Defender 단원을 참조하십시오.

AWS IoT Device Advisor는 필요에 따라 업데이트를 푸시하고 플릿을 패치합니다. AWS IoT Device Advisor는 테스트 사례를 자동으로 업데이트합니다. 선택한 테스트 케이스는 항상 최신 버전입니다. 자세한 내용은 디바이스 어드바이저 단원을 참조하십시오.

의 보안 모범 사례 AWS IoT Core

이 섹션에는의 보안 모범 사례에 대한 정보가 포함되어 있습니다 AWS IoT Core. 산업용 IoT 솔루션의 보안 규칙에 대한 자세한 내용은 산업용 IoT 솔루션을 위한 10가지 보안 황금률을 참조하세요.

에서 MQTT 연결 보호 AWS IoT

AWS IoT Core는 연결된 디바이스가 클라우드 애플리케이션 및 기타 디바이스와 쉽고 안전하게 상호 작용할 수 있도록 하는 관리형 클라우드 서비스입니다.는 간헐적 연결을 허용하도록 특별히 설계된 경량 통신 프로토콜인 HTTP, WebSocket 및 MQTT를 AWS IoT Core 지원합니다. MQTT를 AWS IoT 사용하여에 연결하는 경우 각 연결은 클라이언트 ID라고 하는 식별자와 연결되어야 합니다. MQTT 클라이언트 ID는 MQTT 연결을 고유하게 식별합니다. 다른 연결에 대해 이미 신청된 클라이언트 ID를 사용하여 새 연결이 설정되면 AWS IoT 메시지 브로커는 이전 연결을 삭제하여 새 연결을 허용합니다. 클라이언트 IDs 각각 AWS 계정 내에서 고유해야 합니다 AWS 리전. 즉, 내 외부 AWS 계정 또는 내 리전 간에 클라이언트 IDs의 글로벌 고유성을 적용할 필요가 없습니다 AWS 계정.

디바이스 플릿에서 MQTT 연결 해제의 영향 및 심각도는 여러 요인에 좌우됩니다. 다음이 포함됩니다.

- 사용 사례(예: 디바이스가 전송하는 데이터 AWS IoT, 데이터 양 및 데이터 전송 빈도).
- MQTT 클라이언트 구성(예: 자동 재연결 설정, 연결된 백오프 타이밍, MQTT 영구 세션 사용)
- 디바이스 리소스 제한 사항
- 연결 해제의 근본 원인, 강도 및 지속성

클라이언트 ID 충돌과 잠재적인 부정적 영향을 방지하려면 각 디바이스 또는 모바일 애플리케이션에 AWS IoT 메시지 브로커에 대한 MQTT 연결에 사용할 수 있는 클라이언트 IDs를 제한하는 AWS IoT 또는 IAM 정책이 있는지 확인합니다. 예를 들어 IAM 정책을 사용하면 이미 사용 중인 클라이언트 ID를 사용하여 디바이스가 의도치 않게 다른 디바이스의 연결을 종료하지 않도록 할 수 있습니다. 자세한 내용은 권한 부여 단원을 참조하십시오.

플릿의 모든 디바이스에는 메시지 게시 또는 특정 범위 및 컨텍스트가 있는 주제 구독과 같은 AWS IoT MQTT 작업을 포함하는(이에 국한되지 않음) 의도한 작업만 승인하는 권한이 있는 자격 증명이 있어야 합니다. 특정 권한 정책은 사용 사례에 따라 다를 수 있습니다. 비즈니스 및 보안 요구 사항에 가장 적합한 권한 정책을 식별합니다.

권한 정책 생성 및 관리를 간소화하기 위해 AWS IoT Core 정책 변수 및 IAM 정책 변수를 사용할 수 있습니다. 정책에 정책 변수를 삽입할 수 있으며, 정책이 평가될 때 해당 변수가 디바이스의 요청에서 나온 값으로 대체됩니다. 정책 변수를 사용하여 복수의 디바이스에 권한을 부여하는 단일 정책을 생성할 수 있습니다. AWS IoT 메시지 브로커에 연결하는 데 사용되는 AWS IoT 계정 구성, 인증 메커니즘 및 네트워크 프로토콜을 기반으로 사용 사례에 맞는 관련 정책 변수를 식별할 수 있습니다. 그러나 최선의 권한 정책을 작성하려면 사용 사례 및 위협 모델의 구체적 내용을 고려해야 합니다.

예를 들어 AWS IoT , 레지스트리에 디바이스를 등록한 경우 정책의 사물 정책 변수를 사용하여 사물 이름, 사물 유형 및 사물 속성 값과 같은 사물 속성을 기반으로 권한을 AWS IoT 부여하거나 거부할 수 있습니다. 사물 이름은 사물이 연결될 때 전송되는 MQTT 연결 메시지의 클라이언트 ID에서 가져옵니다 AWS IoT. 사물 정책 변수는 사물이 TLS 상호 인증을 사용하여 MQTT AWS IoT 를 통해에 연결되거나 인증된 Amazon Cognito 자격 증명을 사용하여 WebSocket 프로토콜을 통해 MQTT에 연결될 때 대체됩니다. AttachThingPrincipal API를 사용하여 인증서 및 인증된 Amazon Cognito 자격 증명을 사물에 연결할 수 있습니다. `iot:Connection.Thing.ThingName`은(는) 클라이언트 ID 제한을 적용할 수 있는 유용한 사물 정책 변수입니다. 다음 예제 AWS IoT 정책은 등록된 사물의 이름을 AWS IoT 메시지 브로커에 대한 MQTT 연결의 클라이언트 ID로 사용해야 합니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
```

```
}
```

지속적인 클라이언트 ID 충돌을 식별하려면 CloudWatch Logs를 AWS IoT 활성화하고 사용할 수 있습니다. 클라이언트 ID 충돌로 인해 AWS IoT 메시지 브로커가 연결을 해제하는 모든 MQTT 연결에 대해 다음과 유사한 로그 레코드가 생성됩니다.

```
{
    "timestamp": "2019-04-28 22:05:30.105",
    "logLevel": "ERROR",
    "traceId": "02a04a93-0b3a-b608-a27c-1ae8ebdb032a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "clientId01",
    "principalId": "1670fcf6de55adc1930169142405c4a2493d9eb5487127cd0091ca0193a3d3f6",
    "sourceIp": "203.0.113.1",
    "sourcePort": 21335,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID"
}
```

CloudWatch Logs 필터(예: `{$.reason= "DUPLICATE_CLIENT_ID" }`)를 사용하여 클라이언트 ID 충돌 인스턴스를 검색하거나 지속적인 모니터링 및 보고를 위해 CloudWatch 지표 필터 및 해당 CloudWatch 경보를 설정할 수 있습니다.

AWS IoT Device Defender를 사용하여 지나치게 허용적인 정책 AWS IoT 및 IAM 정책을 식별할 수 있습니다. AWS IoT 또한 Device Defender는 플릿의 여러 디바이스가 동일한 클라이언트 ID를 사용하여 AWS IoT 메시지 브로커에 연결되어 있는지 여부를 알려주는 감사 검사를 제공합니다.

AWS IoT Device Advisor를 사용하여 디바이스가 보안 모범 사례에 안정적으로 연결하고 AWS IoT Core 따를 수 있는지 확인할 수 있습니다.

다음 사항도 참조하세요.

- AWS IoT Core
- AWS IoT의 보안 기능
- AWS IoT Core 정책 변수
- IAM 정책 변수

- Amazon Cognito 자격 증명
- AWS IoT Device Defender
- 에 대한 CloudWatch Logs AWS IoT

디바이스의 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. 상업용 IoT 디바이스를 제작하는 경우 제품을 판매하기 전에 장기간 보관할 수 있습니다. 이 시간 동안 실시간 시계가 드리프트되고 배터리가 방전될 수 있으므로 출고 시 시간 설정으로는 충분하지 않습니다.

다시 말해 대부분의 시스템에서는 디바이스의 소프트웨어에 NTP(네트워크 시간 프로토콜) 클라이언트가 포함되어 있어야 합니다. 디바이스는 AWS IoT Core에 연결하려면 NTP 서버와 동기화될 때까지 기다려야 합니다. 이렇게 할 수 없으면 시스템은 후속 연결이 성공하도록 사용자가 디바이스의 시간을 설정할 수 있는 방법을 제공해야 합니다.

디바이스가 NTP 서버와 동기화되면 AWS IoT Core와의 연결을 열 수 있습니다. 허용되는 클럭 스큐 양은 연결로 수행하려는 작업에 따라 다릅니다.

서버 인증서 검증

디바이스가 상호 작용하는 첫 번째 방법은 보안 연결을 여는 AWS IoT 것입니다. 디바이스를에 연결할 때 다른 서버 가장이 AWS IoT 아닌와 대화하고 있는지 AWS IoT확인합니다 AWS IoT. 각 AWS IoT 서버는 `iot.amazonaws.com` 도메인에 대해 발급된 인증서로 프로비저닝됩니다. 이 인증서는 도메인의 자격 증명과 소유권을 확인한 신뢰할 수 있는 인증 기관에서 AWS IoT 에 발급되었습니다.

디바이스가 연결될 때 첫 번째 AWS IoT Core 작업 중 하나는 디바이스에 서버 인증서를 보내는 것입니다. 디바이스는 `iot.amazonaws.com`에 연결하는 것을 예상하고 있었는지, 해당 연결이 끝날 때 서버에 해당 도메인에 대한 신뢰할 수 있는 기관의 인증서가 있는지 확인할 수 있습니다.

TLS 인증서는 X.509 형식이며 조직의 이름, 위치, 도메인 이름 및 유효 기간과 같은 다양한 정보가 포함되어 있습니다. 유효 기간은 `notBefore` 및 `notAfter`라는 시간 값 페어로 지정됩니다. 와 같은 서비스는 서버 인증서에 제한된 유효 기간(예: 1년)을 AWS IoT Core 사용하고 이전 인증서가 만료되기 전에 새 인증서를 제공하기 시작합니다.

디바이스별로 단일 자격 증명 사용

클라이언트별로 단일 자격 증명을 사용합니다. 디바이스는 일반적으로 X.509 클라이언트 인증서를 사용합니다. 웹 및 모바일 애플리케이션은 Amazon Cognito 자격 증명을 사용합니다. 이렇게 하면 세분화된 권한을 디바이스에 적용할 수 있습니다.

예를 들어, 전구와 온도 조절기라는 두 가지 스마트 홈 객체에서 상태 업데이트를 받는 휴대폰 디바이스로 구성된 애플리케이션을 가지고 있습니다. 전구는 배터리 수준 상태를 전송하고, 온도 조절기는 온도를 보고하는 메시지를 전송합니다.

AWS IoT 는 디바이스를 개별적으로 인증하고 각 연결을 개별적으로 처리합니다. 권한 부여 정책을 사용하여 세분화된 액세스 제어를 적용할 수 있습니다. 주제 공간에 게시하도록 허용하는 정책을 온도 조절기에 대해 정의할 수 있습니다. 다른 주제 공간에 게시하도록 허용하는 별도의 정책을 전구에 대해 정의할 수 있습니다. 마지막으로 온도 조절기와 전구에 대한 주제에 연결하고 구독하여 이러한 디바이스로에서만 메시지를 수신하도록 허용하는 정책을 모바일 앱에 대해 정의할 수 있습니다.

최소 권한의 원칙을 적용하고 디바이스별 권한을 최대한 좁은 범위로 줄입니다. 모든 디바이스 또는 사용자는에 알려진 클라이언트 ID로만 연결하고 식별되고 고정된 주제 세트를 게시하고 구독할 수 AWS IoT 있는 AWS IoT 정책이 있어야 합니다.

백업 AWS 리전 으로 두 번째 사용

데이터 사본을 백업 AWS 리전 으로 1초에 저장하는 것이 좋습니다. 에 대한 재해 복구 AWS IoT라는 AWS 솔루션을 더 이상 사용할 수 없습니다. 연결된 GitHub 라이브러리는 계속 액세스할 수 있지만 2023년 7월에 AWS 사용되지 않으며 더 이상 유지 관리 또는 지원을 제공하지 않습니다. 자체 솔루션을 구현하거나 추가 지원 옵션을 알아보려면 AWS에 문의를 참조하세요. 계정과 연결된 AWS 기술 계정 관리자가 있는 경우 해당 계정에 문의하여 도움을 받으세요.

정시 프로비저닝 사용

각 디바이스를 수동으로 생성하고 프로비저닝하는 것은 시간이 많이 걸릴 수 있습니다.는 디바이스를 처음 연결할 때 프로비저닝할 템플릿을 정의하는 방법을 AWS IoT 제공합니다 AWS IoT. 자세한 내용은 JIT 프로비저닝 단원을 참조하십시오.

AWS IoT Device Advisor 테스트를 실행할 수 있는 권한

다음 정책 템플릿은 AWS IoT Device Advisor 테스트 사례를 실행하는 데 필요한 최소 권한 및 IAM 엔터티를 보여줍니다. *your-device-role-arn*을 사전 조건에서 생성한 디바이스 역할 Amazon 리소스 이름(ARN)으로 바꿉니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thinggroup/your-thing-
group",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iotdeviceadvisor.amazonaws.com"
            }
        }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iotjobsdata:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iotjobsdata:GetPendingJobExecutions",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:UpdateJobExecution",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
```

```
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        }
    ]
}
```

Device Advisor에 대한 교차 서비스 혼동된 대리자 예방

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 위해는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 AWS 제공합니다.

Device Advisor가 리소스에 다른 서비스를 제공하는 권한을 제한하려면 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하는 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 스위츠 정의 리소스의 ARN이어야 합니다. 스위트 정의 리소스는 Device Advisor를 사용하여 생성한 테스트 스위트를 나타냅니다.

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. 예: `arn:aws:iotdeviceadvisor:*:account-id:suitedefinition/*`

다음 예는 Device Advisor에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "ConfusedDeputyPreventionExamplePolicy",
        "Effect": "Allow",
        "Principal": {
            "Service": "iotdeviceadvisor.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:iotdeviceadvisor:us-
east-1:123456789012:suitedefinition/ygp6rxa3tzvn"
        },
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
        }
        }
    }
}
```

AWS 교육 및 인증

AWS IoT 보안의 주요 개념인 AWS IoT Security Primer에 대해 알아보려면 다음 과정을 수강하세요.

모니터링 AWS IoT

모니터링은 AWS IoT 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다.

다중 지점 장애가 발생할 경우 더 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집하는 것이 좋습니다. 먼저 다음 질문에 답하는 모니터링 계획을 수립합니다. 어떻게 답해야 할지 잘 모르는 경우에도 계속해서 로깅을 활성화하고 기준 성능을 설정할 수 있습니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계는 로깅을 활성화하고 다양한 시간과 다양한 로드 조건에서 AWS IoT 성능을 측정하여 환경에서 정상 성능의 기준을 설정하는 것입니다. 모니터링할 때 현재 성능 데이터와 비교할 수 있도록 모니터링 데이터를 기록 AWS IoT해 둡니다. 이를 통해 정상적인 성능 패턴과 성능 이상을 식별하고 문제 해결 방법을 고안할 수 있습니다.

에 대한 기준 성능을 설정하려면 이러한 지표를 모니터링하여 시작해야 AWS IoT합니다. 나중에 언제든 더 많은 지표를 모니터링할 수 있습니다.

- `PublishIn.Success`
- `PublishOut.Success`
- `Subscribe.Success`
- `Ping.Success`
- `Connect.Success`
- `GetThingShadow.Accepted`
- `UpdateThingShadow.Accepted`
- `DeleteThingShadow.Accepted`
- `RulesExecuted`

이 섹션의 주제는 AWS IoT로깅 및 모니터링을 시작하는 데 도움이 될 수 있습니다.

주제

- AWS IoT 로깅 구성
- Amazon CloudWatch를 사용하여 AWS IoT 경보 및 지표 모니터링
- CloudWatch Logs AWS IoT 를 사용하여 모니터링
- 디바이스 측 로그를 Amazon CloudWatch에 업로드합니다.
- 를 사용하여 AWS IoT API 호출 로깅 AWS CloudTrail

AWS IoT 로깅 구성

AWS IoT 활동을 모니터링하고 로깅하려면 먼저 AWS IoT 콘솔, CLI 또는 API를 사용하여 로깅을 활성화해야 합니다. 계정 수준, 이벤트 수준 또는 리소스별 수준의 세 AWS IoT 가지 수준에서에 대한 로깅을 구성할 수 있습니다. 이벤트 수준 및 리소스별 로깅은 V2 로깅에서만 사용할 수 있습니다. V1 로깅을 사용하는 고객은 이러한 기능에 액세스하려면 V2로 마이그레이션해야 합니다. 세부 정보를 참조하세요.

AWS IoT 로깅을 구성하는 방법을 고려할 때 달리 지정하지 않는 한 계정 수준 로깅 구성에 따라 AWS IoT 활동이 로깅되는 방법이 결정됩니다. 먼저 기본 로그 수준이 INFO 또는 DEBUG인 세부 로그를 가져올 수 있습니다. 초기 로그를 검토한 후 계정 또는 이벤트 수준에서 WARN 또는 ERROR와 같은 덜 명확한 수준으로 기본 로그 수준을 변경하고 더 많은 주의가 필요할 수 있는 리소스에 대해 보다 구체적인 리소스별 로그 수준을 설정할 수 있습니다. 로그 수준은 언제든 변경할 수 있습니다.

이 주제에서는 클라우드 측 로그인을 다룹니다 AWS IoT. 디바이스 측 로깅 및 모니터링에 대한 자세한 내용은 CloudWatch에 디바이스 측 로그 업로드를 참조하세요.

로깅 및 모니터링에 대한 자세한 내용은 의 로깅 및 모니터링을 AWS IoT Greengrass AWS IoT Greengrass참조하세요.

에서 V2 로깅 구성 AWS IoT

로깅 버전 확인

V2 로깅이 활성화되지 않은 경우 GetV2LoggingOptions API는 NotConfiguredException을 반환합니다. V2 이 오류는 V1 로깅이 사용 중이거나 로깅이 구성되지 않은 경우에 발생합니다.

V2 로깅 기능 이해

V2 로깅은 이벤트 수준 로깅과 리소스별 로깅이라는 두 가지 주요 기능을 제공합니다. 이벤트 수준 로깅을 사용하면 사용자 지정 가능한 로그 수준 및 CloudWatch 로그 그룹 대상으로 대상 로깅 구성을 사

용할 수 있습니다. 리소스별 로깅을 사용하면 사물 그룹, 소스 IP, 클라이언트 ID 또는 보안 주체 ID별로 로그를 필터링할 수 있습니다. 이러한 기능을 함께 사용하면 IoT 작업에 대한 세분화된 제어와 포괄적인 가시성을 제공하여 로그 검색 가능성을 개선하고 불필요한 로깅 활동을 제거하여 비용을 절감할 수 있습니다.

V1에서 V2로 마이그레이션

AWS CLI 또는 SDK를 통해 SetV2LoggingOptions API를 사용하여 V2 로깅으로 마이그레이션 할 수 있습니다. SetV2LoggingOptions 마이그레이션 후는 모든 로그를 CloudWatch 로그 그룹 'AWSIotLogsV2'로 AWS IoT 자동 라우팅합니다. 중요: 다운스트림 애플리케이션 또는 리소스가 'AWSIotLogs'의 정보를 사용하는 경우 해당 로그 그룹 경로를 사용하도록 업데이트합니다.

로깅 역할 및 정책 구성

로그인을 활성화하려면 먼저 사용자를 대신하여 CloudWatch AWS IoT 로그 그룹에 로그 활동을 쓸 수 있는 AWS IoT 권한을 부여하는 IAM 역할과 정책을 생성 AWS IoT해야 합니다. AWS IoT 콘솔의 로그 섹션에 필요한 정책을 사용하여 IAM 역할을 생성할 수도 있습니다.

> **Note**
>
> AWS IoT 로깅을 활성화하기 전에 CloudWatch Logs 액세스 권한을 이해해야 합니다. CloudWatch Logs 액세스 권한이 있는 사용자는 디바이스에서 디버깅 정보를 볼 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs에 대한 인증 및 액세스 제어 단원을 참조하세요. 로드 테스트 AWS IoT Core 로 인해에서 트래픽 패턴이 많을 것으로 예상되는 경우 제한을 방지하려면 IoT 로깅을 비활성화하는 것이 좋습니다. 트래픽이 많이 감지되면 서비스에서 계정 로그인을 비활성화할 수 있습니다.

다음은 AWS IoT Core 리소스에 대한 로깅 역할 및 정책을 생성하는 방법을 보여줍니다.

로깅 역할 생성

로깅 역할을 만들려면 IAM 콘솔의 역할 허브를 열고 역할 생성을 선택합니다.

1. Select trusted entity(신뢰할 수 있는 엔터티 선택)에서 AWS Service를 선택합니다. 그런 다음 Use case(사용 사례)에서 IoT를 선택합니다. IoT가 표시되지 않는 경우 다른 AWS 서비스 사용 사례: 드롭다운 메뉴에서 IoT를 입력하고 검색합니다. 다음을 선택합니다.
2. Add permissions(권한 추가) 페이지에서 서비스 역할에 자동으로 연결되는 정책이 표시됩니다. 다음을 선택합니다.

3. Name, review, and create(이름 지정, 검토 및 생성) 페이지에서 해당 역할의 Role name(역할 이름) 및 Role description(역할 설명)을 입력하고 Create role(역할 생성)을 선택합니다.

로깅 역할 정책

다음 정책 문서는 AWS IoT 가 사용자 대신 CloudWatch에 로그 항목을 제출하도록 허용하는 역할 정책 및 신뢰 정책을 제공합니다. 사용자 지정 CloudWatch 로그 그룹을 사용하여 이벤트 수준 로깅을 구성하는 경우 사용자 지정 리소스 ARN을 포함하도록 역할 정책을 업데이트해야 합니다.

로그 항목도 제출 AWS IoT Core for LoRaWAN 하도록 허용한 경우 두 활동을 모두 로깅하는 정책 문서가 생성됩니다.

> **Note**
>
> 정책 문서는 로깅 역할을 생성할 때 사용자를 위해 작성됩니다. 문서에는 변수 *${partition}* *${region}* 가 있으며*${accountId}*, 이를 값으로 바꿔야 합니다.
>
> - 파티션을 리전의 파티션으로 바꿉니다.
> - region을 사용하는 AWS 리전으로 바꿉니다. 디바이스에서 AWS CLI를 구성하는 데 사용한 것과 동일한 AWS 리전을 사용해야 합니다.
> - account-id를 AWS 계정 ID로 바꿉니다.

역할 정책:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "iot:GetLoggingOptions",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingOptions",
```

```
                "iot:GetV2LoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:ListV2LoggingLevels",
                "iot:DeleteV2LoggingLevel"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:AWSIotLogsV2:*"
            ]
        }
    ]
}
```

AWS IoT Core 활동만 로깅하는 신뢰 정책:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

AWS IoT IAM 역할 권한이 부족하여 로깅이 CloudWatch Logs에 로깅을 게시하지 못할 수 있습니다. 이 경우 CloudWatch 로깅 지표를 확인하여 실패를 조사하고 문제를 해결합니다.

에서 로깅 구성 AWS IoT (콘솔)

이 섹션에서는 AWS IoT 콘솔을 사용하여 AWS IoT 로깅을 구성하는 방법을 설명합니다. 계정 수준, 이벤트 수준 및 리소스별 로깅을 설정할 수 있습니다.

AWS IoT 로깅을 구성하려면:

1. AWS IoT 콘솔에 로그인합니다. 자세한 내용은 AWS IoT 콘솔 열기 단원을 참조하십시오.

2. 왼쪽 탐색 창에서 로그(이전에는 설정 아래의 섹션)를 선택합니다.
3. 계정 수준 로깅 구성: 이벤트 수준 또는 리소스별 설정으로 재정의되지 않는 한 계정 수준 로깅은 모든 AWS IoT 플릿(디바이스 또는 엔드포인트)에 적용됩니다.

   a. 계정 수준 로깅에서 계정 수준 로깅 관리를 선택하여 업데이트합니다.

   b. "로깅 활성화" 확인란을 선택하여 CloudWatch로 로그 전송을 시작합니다. “로깅 활성화”를 선택하지 않으면는 이벤트 수준 또는 리소스 수준 로깅 구성에 관계없이 CloudWatch 로그 그룹에 로그를 보내지 AWS IoT 않습니다.

   c. IAM 로그 역할의 드롭다운 목록에서 기존 역할을 선택합니다. 역할 세부 정보를 보고 역할 권한을 검사할 수 있습니다. 또는 새 역할 생성을 선택하여 새 IAM 역할을 설정합니다. 로깅 역할은가 사용자를 대신하여 CloudWatch에 로그 항목을 AWS IoT 제출하도록 허용하는 정책을 제공합니다. 사용자 지정 CloudWatch 로그 그룹을 사용하여 이벤트 수준 로깅을 구성하는 경우이 로그 그룹의 ARN을 포함하도록 역할 정책을 업데이트해야 합니다.

   d. CloudWatch Logs에 표시할 로그 항목의 세부 수준에 해당하는 기본 로그 수준을 선택합니다. https://docs.aws.amazon.com/iot/latest/developerguide/configure-logging.html#log-level 참고: 로그 수준 “DEBUG”는 가장 세부적인 정보를 제공하지만 CloudWatch 비용은 증가합니다. CloudWatch 로그 그룹 대상은 계정 수준에서 구성할 수 없습니다. 그러나 다음 섹션에 설명된 대로 개별 이벤트 유형에 대한 사용자 지정 로그 그룹을 지정할 수 있습니다.

   e. 로깅 업데이트를 선택하여 변경 사항을 저장합니다.

4. 이벤트 수준 로깅을 사용하면 관련 이벤트에 대한 로그를 선택적으로 캡처하여 전용 CloudWatch 로그 그룹으로 전달할 수 있습니다. 이를 통해 검색 가능성을 높이기 위해 사용 사례별로 로그를 구성하고, 다양한 대상과 공유하며, 로그를 활성화하고 이벤트 중요도에 따라 로그 수준을 설정하여 CloudWatch 비용을 절감할 수 있습니다.

   이벤트 수준 로깅 구성: 이벤트 수준 로깅은 클라이언트 인증 시도와 같은 특정 AWS IoT 이벤트를 캡처합니다. 이러한 설정은 계정 수준 로깅을 재정의합니다.

   a. 이벤트 수준 로깅 섹션에서 이벤트 수준 로깅 관리를 선택하여 업데이트합니다.

   b. 기본적으로 이벤트 유형은 계정 수준 로깅 구성을 상속합니다. 참고: 리소스별 로깅이 구성된 경우 계정 및 이벤트 수준 설정을 재정의합니다.

   c. 개별 이벤트에 대한 설정을 수정하려면 해당 이벤트 행의 값을 클릭합니다. 로그 수준과 CloudWatch 로그 그룹 대상을 모두 조정할 수 있습니다. 사용자 지정 CloudWatch 로그 그룹 대상을 지정할 때 IAM 역할 정책에 새 로그 그룹에 대한 권한이 포함되어 있는지 확인해야 합니다. 역할 정책을 업데이트하지 않으면가 사용자 지정 로그 그룹에 로그를 AWS IoT 쓰지 못합니다. 선택한 후 확인 표시를 클릭하여 선택을 확인합니다. '수정됨' 열에 보류 중인 변경 사항을 나타내는 '예'가 표시됩니다.

d. 로깅 업데이트를 클릭하여 변경 사항을 적용하거나 취소를 선택하여 취소합니다.

5. 리소스별 재정의 구성: 리소스별 재정의는 선택한 리소스에 로깅 설정을 적용합니다. 리소스는 사물 그룹, 소스 IP, 클라이언트 ID 또는 보안 주체 ID일 수 있습니다. 리소스별 로깅 구성은 계정 수준 및 이벤트 수준 설정을 모두 재정의합니다. 활성화하면 지정된 리소스에 대해 구성된 로깅 수준에서 모든 이벤트 유형에 대한 로그가 생성됩니다. 예를 들어, 다른 모든 사물에 대한 정보 수준 로깅을 유지하면서 특정 사물에 대한 디버그 수준 로깅을 설정할 수 있습니다.

   a. 리소스별 재정의 섹션에서 리소스별 재정의 추가를 선택합니다.
   b. 사물 그룹, 소스 IP, 클라이언트 ID 또는 보안 주체 ID 중에서 로그 대상을 선택합니다.
   c. 선택한 대상 유형에 해당하는 로그 대상 값을 입력합니다.
   d. 리소스별 로그 수준 섹션의 드롭다운 메뉴에서 원하는 로그 수준을 선택합니다.
   e. 제출을 클릭하여 재정의를 추가하거나 취소를 클릭하여 변경 사항을 취소합니다.
   f. 기존 리소스별 재정의를 수정하려면 리소스 옆의 확인란을 선택하고 “제거”를 클릭하여 재정의를 삭제하거나 “편집”을 클릭하여 수정합니다.

로깅을 활성화한 후 CloudWatch 콘솔에서 AWS IoT 로그 보기로 이동하여 로그 항목을 보는 방법에 대해 자세히 알아보세요.

에서 계정 및 이벤트 수준 로깅 구성 AWS IoT (CLI)

이 섹션에서는 CLI를 사용하여 AWS IoT 에 대한 글로벌 로깅을 구성하는 방법을 설명합니다.

선택적으로 이벤트 수준 로깅을 구성할 수 있습니다. 이벤트 수준 로깅은 인증 및 권한 부여 또는 인증서 생성 이벤트와 같은 이벤트 수준에서 로깅 정보를 캡처합니다. 이벤트 수준에서 로그 수준과 CloudWatch 로그 그룹 대상을 모두 사용자 지정할 수 있습니다. 이벤트 수준 로깅은 계정 수준 로깅에 비해 더 대상화된 수준에서 작동하므로 계정 수준 로깅 설정을 재정의합니다. 이 계층적 접근 방식을 사용하면 운영 중요도 및 비용 고려 사항에 따라 다양한 유형의 이벤트에 대해 다양한 로깅 전략을 유지할 수 있습니다.

> **Note**
>
> 사용하려는 역할의 Amazon 리소스 이름(ARN)이 필요합니다. 로깅에 사용할 역할을 만들어야 하는 경우 계속하기 전에 로깅 역할 생성 단원을 참조하세요. 모든 이벤트 유형에 대해 사용자 지정 CloudWatch 로그 그룹을 지정할 때 로깅 역할에 대상 로그 그룹에 필요한 권한이 있는지 확인합니다.
>
> API를 호출하는 데 사용되는 보안 주체는 로깅 역할에 대한 역할 전달 권한이 있어야 합니다.

API에서 여기에 표시된 CLI 명령에 해당하는 메서드를 사용하여 AWS API로이 절차를 수행할 수도 있습니다.

CLI를 사용하여에 대한 기본 로깅을 구성하려면 AWS IoT

1. set-v2-logging-options 명령을 사용하여 계정에 대한 로깅 옵션을 설정합니다.

```
aws iot set-v2-logging-options \
    --event-configurations event-configuration-list \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

여기서 각 항목은 다음과 같습니다.

--role-arn

CloudWatch Logs의 로그에 쓸 수 있는 AWS IoT 권한을 부여하는 역할 ARN입니다. 초기 설정에는 Role-arn 구성이 필요합니다.

--default-log-level

사용할 로그 수준입니다. 유효한 값은 ERROR, WARN, INFO, DEBUG또는 입니다DISABLED. 초기 설정에는 Default-log-level 구성이 필요합니다.

--no-disable-all-logs

모든 AWS IoT 로깅을 활성화하는 선택적 파라미터입니다. 이 파라미터를 사용하여 현재 비활성화되어 있는 로깅을 활성화합니다.

--disable-all-logs

모든 AWS IoT 로깅을 비활성화하는 선택적 파라미터입니다. 현재 활성화된 로깅을 비활성화하려면 이 파라미터를 사용합니다.

--event-configurations

이 파라미터는 선택 사항이며 개별 이벤트 유형에 대한 로깅 설정을 사용자 지정할 수 있습니다.

- eventType: 계정 수준 설정을 재정의하는 이벤트 유형입니다.
- logLevel: DEBUG, INFO, ERROR, WARN 또는 DISABLED로 계정 수준 설정 재정의
- logDestination: 로그 전송을 위한 사용자 지정 CloudWatch 로그 그룹 지정

각 이벤트 유형에 대해 로깅 수준 및 로그 대상을 독립적으로 구성할 수 있습니다. 지정하지 않으면 이벤트가 계정 수준 설정을 상속합니다.

```
aws iot set-v2-logging-options \
    --event-configurations "[{\"eventType\":\"Publish-In\",\"logLevel\":\"INFO\",
\"logDestination\":\"examplePublishInLogGroup\"}]"
```

2. get-v2-logging-options 명령을 사용하여 현재 로깅 옵션을 가져옵니다.

```
aws iot get-v2-logging-options \
    --verbose
```

여기서 각 항목은 다음과 같습니다.

--verbose

모든 eventTypes 및 해당 구성을 검색할 수 있는 선택적 파라미터입니다.

로깅을 활성화한 후 CloudWatch 콘솔에서 AWS IoT 로그 보기로 이동하여 로그 항목을 보는 방법에 대해 자세히 알아보세요.

> **Note**
>
> AWS IoT 는 계정에 대한 글로벌 로깅을 설정하고 가져오기 위해 이전 명령(set-logging-options 및 get-logging-options)을 계속 지원합니다. 이들 명령어를 사용할 경우 결과 로그는 JSON 페이로드 대신 일반 텍스트가 포함되며 일반적으로 로깅 지연 시간이 길어질 수 있음에 유의하세요. 이러한 이전 명령의 구현은 더 이상 개선되지 않을 것입니다. "v2" 버전을 사용하여 로깅 옵션을 구성하고 가능하면 이전 버전을 사용하는 레거시 애플리케이션을 변경하는 것이 좋습니다.

AWS IoT (CLI)에서 리소스별 재정의 구성

이 섹션에서는 CLI를 AWS IoT 사용하여에 대한 리소스별 재정의를 구성하는 방법을 설명합니다. 리소스별 재정의를 사용하면 사물 그룹, 클라이언트 ID, 소스 IP 또는 보안 주체 ID로 식별되는 특정 리소스에 대한 로깅 수준을 지정할 수 있습니다. 리소스별 로깅이 활성화되면 계정 수준 및 이벤트 수준 설정

을 모두 재정의합니다. 모든 이벤트 유형은 이벤트 수준 구성에서 해당 이벤트가 비활성화된 경우에도 구성된 로깅 수준에서 지정된 리소스에 대한 로그를 생성합니다.

사물 그룹에는 계층적 관계를 만들기 위해 다른 사물 그룹이 포함될 수 있습니다. 이 절차에서는 단일 사물 그룹의 로깅을 구성하는 방법에 대해 설명합니다. 계층 구조의 상위 사물 그룹에 이 절차를 적용하여 해당 계층 구조의 모든 사물 그룹에 대한 로깅을 구성할 수 있습니다. 하위 사물 그룹에 이 절차를 적용하여 상위 그룹의 로깅 구성을 재정의할 수도 있습니다.

사물은 사물 그룹의 멤버일 수 있습니다. 이 멤버십을 통해 사물은 사물 그룹에 적용된 구성, 정책 및 설정을 상속할 수 있습니다. 사물 그룹은 각 사물을 개별적으로 처리하는 대신 설정을 관리하고 여러 사물에 집합적으로 적용하는 데 사용됩니다. 클라이언트 ID가 사물 이름과 일치하면 AWS IoT Core 는 클라이언트 세션을 해당 사물 리소스와 자동으로 연결합니다. 이렇게 하면 클라이언트 세션이 로깅 수준을 포함하여 사물이 속한 사물 그룹에 적용된 구성 및 설정을 상속할 수 있습니다. 클라이언트 ID가 사물 이름과 일치하지 않는 경우 독점 사물 연결을 활성화하여 연결을 설정할 수 있습니다. 자세한 정보는 ???을 참조하십시오.

사물 그룹 외에도 디바이스의 클라이언트 ID, 소스 IP 및 보안 주체 ID와 같은 대상을 로깅할 수 있습니다.

> **Note**
>
> 사용하려는 역할의 Amazon 리소스 이름(ARN)이 필요합니다. 로깅에 사용할 역할을 만들어야 하는 경우 계속하기 전에 로깅 역할 생성 단원을 참조하세요.
> API를 호출하는 데 사용되는 보안 주체는 로깅 역할에 대한 역할 전달 권한이 있어야 합니다.

API에서 여기에 표시된 CLI 명령에 해당하는 메서드를 사용하여 AWS API로이 절차를 수행할 수도 있습니다.

CLI를 사용하여에 대한 리소스별 재정의를 구성하려면 AWS IoT

1. aws iot set-v2-logging-options 명령을 사용하여 리소스별 로깅을 구성하기 전에 계정 수준 로깅을 활성화합니다.
2. set-v2-logging-level 명령을 사용하여 리소스별 재정의를 구성합니다. 사물 그룹 구성은 다음 예제를 참조하세요.

```
aws iot set-v2-logging-level \
              --log-target targetType=THING_GROUP,targetName=thing_group_name \
              --log-level log_level
```

--log-target

로깅을 구성하는 리소스의 유형과 이름입니다. `targetType` 값은 `THING_GROUP` | `CLIENT_ID` | `SOURCE_IP` | `PRINCIPAL_ID` 중 하나여야 합니다. 로그 대상 파라미터 값은 앞의 예제 명령과 같이 텍스트이거나 다음 예와 같이 JSON 문자열일 수 있습니다.

```
aws iot set-v2-logging-level \
              --log-target '{"targetType": "THING_GROUP","targetName":
 "thing_group_name"}' \
              --log-level log_level
```

--log-level

특정 리소스에 대한 로그를 작성할 때 사용하는 로깅 수준입니다. 유효한 값은 DEBUG, INFO, ERROR, WARN, DISABLED입니다.

3. list-v2-logging-levels 명령을 사용하여 현재 구성된 로깅 수준을 나열합니다.

```
aws iot list-v2-logging-levels
```

4. delete-v2-logging-level 명령을 사용하여 다음 예와 같이 리소스별 로깅 수준을 삭제합니다.

```
aws iot delete-v2-logging-level \
              --target-type "THING_GROUP" \
              --target-name "thing_group_name"
```

```
aws iot delete-v2-logging-level \
              --target-type=CLIENT_ID
              --target-name=ClientId1
```

--target-type

`target-type` 값은 `THING_GROUP` | `CLIENT_ID` | `SOURCE_IP` | `PRINCIPAL_ID` 중 하나여야 합니다.

--target-name

로깅 수준을 제거할 사물 그룹의 이름입니다.

로그 수준

이러한 로그 수준은 기록되는 이벤트를 결정하고, 기본 및 리소스별 로그 수준에 적용됩니다.

오류

작업을 실패하게 만든 오류입니다.

예: 인증서가 만료되어 디바이스를 인증하지 못했습니다.

로그에 ERROR 정보만 포함됩니다.

WARN

시스템에서 불일치를 초래할 수는 있지만 반드시 작업이 실패하지는 않는 모든 것입니다.

예: 메시지 속도 제한에 근접합니다.

로그에 ERROR 및 WARN 정보가 포함됩니다.

INFO

사물 흐름에 대한 상위 수준 정보입니다.

예: 클라이언트가 MQTT 주제를 성공적으로 구독했습니다.

로그에 INFO, ERROR 및 WARN 정보가 포함됩니다.

DEBUG

문제 디버깅에 도움이 될 수 있는 정보입니다.

예: IoT Rules Engine이 규칙 주제 “rule/test”에 게시된 메시지를 감지하고 성공적으로 실행을 시작했습니다. 규칙은 RepublishAction으로 구성됩니다.

로그에 DEBUG, INFO, ERROR 및 WARN 정보가 포함됩니다.

DISABLED

모든 로깅이 비활성화됩니다.

Amazon CloudWatch를 사용하여 AWS IoT 경보 및 지표 모니터링

에서 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표 AWS IoT 로 처리하는 CloudWatch 를 AWS IoT 사용하여 모니터링할 수 있습니다. 이러한 통계는 2주간 기록되므로 기록 정보를 보고 웹 애플리케이션이나 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다.

AWS IoT 지표 데이터는 기본적으로 1분 간격으로 CloudWatch에 자동 전송됩니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch, Amazon CloudWatch Events 및 Amazon CloudWatch Logs란 무엇입니까?를 참조하세요.

AWS IoT 지표 사용

에서 보고하는 지표는 다양한 방식으로 분석할 수 있는 정보를 AWS IoT 제공합니다. 다음 사용 사례는 10개의 사물이 하루 한 번 인터넷에 연결하는 시나리오를 기반으로 합니다. 매일:

- 10개의 사물이 거의 동시에 AWS IoT 에 연결됩니다.
- 각 사물이 주제 필터를 구독한 후 연결 해제 전에 1시간을 대기합니다. 이 시간 동안 사물이 서로 통신하고 세계의 상태를 더 많이 학습합니다.
- 각 사물이 `UpdateThingShadow`를 사용하여 새로 발견된 데이터에 기반한 일부 인식을 게시합니다.
- 각 사물의 연결이 끊어집니다 AWS IoT.

이 주제에서는 시작하는 데 도움이 될 수 있는 몇 가지 질문을 살펴봅니다.

- 내 사물이 매일 성공적으로 연결되지 않을 경우 어떻게 통보를 받을 수 있습니까?
- 내 사물이 매일 데이터를 게시하지 않을 경우 어떻게 통보를 받을 수 있습니까?
- 내 사물 섀도우 업데이트가 매일 거부될 경우 어떻게 통보를 받을 수 있습니까?
- 작업에 대한 CloudWatch 경보를 생성하려면 어떻게 해야 합니까?

CloudWatch 경보 및 지표에 대한 추가 정보

- 모니터링할 CloudWatch 경보 생성 AWS IoT
- AWS IoT 지표 및 차원

모니터링할 CloudWatch 경보 생성 AWS IoT

경보로 인해 상태가 변경되면 Amazon SNS 메시지를 전송하는 CloudWatch 경보를 생성할 수 있습니다. 경보는 지정한 기간 동안 단일 지표를 감시합니다. 지표 값이 장기간 동안 지정된 임계값을 초과하면 하나 이상의 작업이 수행됩니다. 이 작업은 Amazon SNS 주제 또는 Auto Scaling 정책에 전송되는 알림일 수 있습니다. 경보는 지속적인 상태 변경에 대해서만 작업을 트리거합니다. CloudWatch 경보는 특정 상태에 있다는 이유만으로는 작업을 트리거하지 않습니다. 상태가 변경되고 지정한 기간 동안 유지되어야 합니다.

다음 주제에서는 CloudWatch 경보를 사용하는 몇 가지 예를 설명합니다.

- 내 사물이 매일 성공적으로 연결되지 않을 경우 어떻게 통보를 받을 수 있습니까?
- 내 사물이 매일 데이터를 게시하지 않을 경우 어떻게 통보를 받을 수 있습니까?
- 내 사물 섀도우 업데이트가 매일 거부될 경우 어떻게 통보를 받을 수 있습니까?
- 작업에 대한 CloudWatch 경보를 생성하려면 어떻게 해야 합니까?

CloudWatch 경보가 모니터링할 수 있는 모든 지표는 AWS IoT 지표 및 차원에서 확인할 수 있습니다.

내 사물이 매일 성공적으로 연결되지 않을 경우 어떻게 통보를 받을 수 있습니까?

1. `things-not-connecting-successfully`라는 Amazon SNS 주제를 생성하고 해당 Amazon 리소스 이름(ARN)을 기록합니다. 이 절차에서는 주제의 ARN을 *`sns-topic-arn`* 이라고 지칭하겠습니다.

   Amazon SNS 알림 생성 방법에 대한 자세한 내용은 Amazon SNS 시작하기를 참조하세요.
2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name ConnectSuccessAlarm \
    --alarm-description "Alarm when my Things don't connect successfully" \
    --namespace AWS/IoT \
    --metric-name Connect.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. CloudWatch 콘솔에 경보가 나타나는지 확인합니다.

내 사물이 매일 데이터를 게시하지 않을 경우 어떻게 통보를 받을 수 있습니까?

1. `things-not-publishing-data`라는 Amazon SNS 주제를 생성하고 해당 Amazon 리소스 이름(ARN)을 기록합니다. 이 절차에서는 주제의 ARN을 *sns-topic-arn* 이라고 지칭하겠습니다.

   Amazon SNS 알림 생성 방법에 대한 자세한 내용은 Amazon SNS 시작하기를 참조하세요.
2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name PublishInSuccessAlarm\
    --alarm-description "Alarm when my Things don't publish their data \
    --namespace AWS/IoT \
    --metric-name PublishIn.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. CloudWatch 콘솔에 경보가 나타나는지 확인합니다.

내 사물 섀도우 업데이트가 매일 거부될 경우 어떻게 통보를 받을 수 있습니까?

1. `things-shadow-updates-rejected`라는 Amazon SNS 주제를 생성하고 해당 Amazon 리소스 이름(ARN)을 기록합니다. 이 절차에서는 주제의 ARN을 *sns-topic-arn* 이라고 지칭하겠습니다.

   Amazon SNS 알림 생성 방법에 대한 자세한 내용은 Amazon SNS 시작하기를 참조하세요.
2. 경보를 만듭니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name UpdateThingShadowSuccessAlarm \
    --alarm-description "Alarm when my Things Shadow updates are getting rejected"
 \
    --namespace AWS/IoT \
    --metric-name UpdateThingShadow.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 경보를 테스트합니다.

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value ALARM
```

4. CloudWatch 콘솔에 경보가 나타나는지 확인합니다.

작업에 대한 CloudWatch 경보를 생성하려면 어떻게 해야 합니까?

작업 서비스는 작업을 모니터링할 수 있는 CloudWatch 지표를 제공합니다. CloudWatch 경보를 생성하여 Jobs 지표를 모니터링할 수 있습니다.

다음 명령은 CloudWatch 경보를 생성하여 *SampleOTAJob* 작업에 대해 실패한 총 작업 실행 수를 모니터링하고 20회 이상 작업 실행이 실패한 경우 이를 알립니다. 경보는 보고된 값을 300초마다 확인하여 작업 지표 `FailedJobExecutionTotalCount`을 모니터링합니다. 보고된 단일 값이 20보다 클 때, 즉 작업이 시작된 이후 20개 이상의 작업 실행이 실패했을 때 활성화합니다. 경보가 울리면 제공된 Amazon SNS 주제로 알림이 전송됩니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name TotalFailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when total number of failed job execution exceeds the
 threshold for SampleOTAJob" \
```

```
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionTotalCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 20 \
    --comparison-operator GreaterThanThreshold \
    --period 300 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions
```

다음 명령은 CloudWatch 경보를 생성하여 주어진 기간 동안 *SampleOTAJob* 작업에 대해 실패한 작업 실행 수를 모니터링합니다. 그런 다음 해당 기간 동안 5회 이상 작업 실행이 실패한 경우 이를 알립니다. 경보는 3600초마다 보고된 값을 확인하여 작업 지표 FailedJobExecutionCount을 모니터링합니다. 보고된 단일 값이 5보다 클 때, 즉 작업이 시작된 이후 5개 이상의 작업 실행이 실패했을 때 활성화합니다. 경보가 울리면 제공된 Amazon SNS 주제로 알림이 전송됩니다.

```
aws cloudwatch put-metric-alarm \
    --alarm-name FailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when number of failed job execution per hour exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --period 3600 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions-per-hour
```

AWS IoT 지표 및 차원

와 상호 작용하면 AWS IoT서비스는 1분마다 지표와 차원을 CloudWatch로 전송합니다. CloudWatch 콘솔 또는를 AWS IoT사용하여 이러한 지표 AWS CLI 를 볼 수 있습니다.

CloudWatch 콘솔을 사용하여 지표를 보려면 CloudWatch 콘솔을 엽니다. 탐색 창에서 지표(Metrics)를 선택한 다음 모든 지표(All metrics)를 선택합니다. 찾아보기 탭에서를 검색 AWS IoT 하여 지표 목록을

봅니다. 지표는 먼저 서비스 네임스페이스별로 그룹화된 다음 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

를 사용하여 지표를 보려면 다음 명령을 AWS CLI실행합니다.

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch는 AWS IoT다음과 같은 지표 그룹을 표시합니다.

- AWS IoT 지표
- AWS IoT Core 자격 증명 공급자 지표
- 인증 지표
- 서버 인증서 OCSP 스테이플링 지표
- 규칙 지표
- 규칙 작업 지표
- HTTP 작업별 지표
- 메시지 브로커 지표
- 디바이스 섀도우 지표
- 지표 로깅
- Jobs 지표
- Device Defender Audit 지표
- Device Defender Detect 지표
- 디바이스 프로비저닝 지표
- LoRaWAN 지표
- 플릿 인덱싱 지표
- 지표 차원

AWS IoT 지표

지표	설명
AddThingToDynamicThingGroupsFailed	동적 사물 그룹에 사물을 추가하는 것과 관련된 실패 이벤트 수입니다. DynamicThingGroupName 차

지표	설명
	원에는 사물을 추가하지 못한 동적 그룹의 이름이 포함됩니다.
`NumLogBatchesFailedToPublishThrottled`	조절 오류로 인해 게시 실패한 단일 배치.
`NumLogEventsFailedToPublishThrottled`	조절 오류로 인해 게시 실패한 배치 내 로그 이벤트 수.

AWS IoT Core 자격 증명 공급자 지표

지표	설명
`CredentialExchangeSuccess`	AWS IoT Core 자격 증명 공급자에 대한 성공적인 `AssumeRoleWithCertificate` 요청의 수입니다.

인증 지표

> **Note**
>
> 인증 지표는 프로토콜 지표 아래의 CloudWatch 콘솔에 표시됩니다.

지표	설명
`Connection.AuthNError`	인증 실패로 인해가 AWS IoT Core 거부하는 연결 시도 횟수입니다. 이 지표는 AWS 계정의 엔드포인트와 일치하는 서버 이름 표시(SNI) 문자열을 보내는 연결만 고려합니다. 이 지표에는 인터넷 스캔 도구 또는 검사 활동과 같은 외부 소스의 연결 시도가 포함됩니다. `Protocol` 차원에는 연결 시도를 전송하는 데 사용된 프로토콜이 포함됩니다.

서버 인증서 OCSP 스테이플링 지표

지표	설명
RetrieveOCSPStapleData.Success	OCSP 응답을 성공적으로 수신하고 처리했습니다. 이 응답은 구성된 도메인에 대한 TLS 핸드셰이크 중에 포함됩니다. `DomainConfigurationName` 차원에는 서버 인증서 OCSP 스테이플링이 활성화된 구성된 도메인의 이름이 포함됩니다.

규칙 지표

지표	설명
`ParseError`	규칙이 수신 대기 중인 주제에 게시되는 메시지에서 발생한 JSON 구문 분석 오류 수. `RuleName` 차원에는 규칙 이름이 포함됩니다.
`RuleExecutionThrottled`	악의적인 행위 때문에 또는 메시지 수가 규칙 엔진의 조절 제한을 초과했기 때문에 규칙 엔진이 제한하는 메시지 수입니다. `RuleName` 차원에는 트리거할 규칙 이름이 포함됩니다.
`RuleNotFound`	트리거할 규칙을 찾을 수 없습니다. `RuleName` 차원에는 규칙 이름이 포함됩니다.
`RulesExecuted`	실행된 AWS IoT 규칙 수입니다.
`TopicMatch`	규칙이 수신 대기 중인 주제에 게시되는 수신 메시지 수. `RuleName` 차원에는 규칙 이름이 포함됩니다.

규칙 작업 지표

지표	설명
Failure	실패한 규칙 작업 호출 수. RuleName 차원에는 작업을 지정하는 규칙 이름이 포함됩니다. ActionType 차원에는 호출된 작업 유형이 포함됩니다.
Success	성공한 규칙 작업 호출 수. RuleName 차원에는 작업을 지정하는 규칙 이름이 포함됩니다. ActionType 차원에는 호출된 작업 유형이 포함됩니다.
ErrorActionFailure	실패한 오류 작업 수 RuleName 차원에는 작업을 지정하는 규칙 이름이 포함됩니다. ActionType 차원에는 호출된 작업 유형이 포함됩니다.
ErrorActionSuccess	성공한 오류 작업 수 RuleName 차원에는 작업을 지정하는 규칙 이름이 포함됩니다. ActionType 차원에는 호출된 작업 유형이 포함됩니다.

HTTP 작업별 지표

지표	설명
HttpCode_Other	다운스트림 웹 서비스/애플리케이션의 응답 상태 코드가 2xx, 4xx 또는 5xx가 아닌 경우 생성됩니다.
HttpCode_4XX	다운스트림 웹 서비스/애플리케이션의 응답 상태 코드가 400에서 499 사이인 경우 생성됩니다.
HttpCode_5XX	다운스트림 웹 서비스/애플리케이션의 응답 상태 코드가 500에서 599 사이인 경우 생성됩니다.
HttpInvalidUrl	대체 템플릿이 바뀐 후 엔드포인트 URL이 https://로 시작하지 않는 경우 생성됩니다.

지표	설명
HttpRequestTimeout	다운스트림 웹 서비스/애플리케이션이 요청 제한 시간 내에 응답을 반환하지 않는 경우 생성됩니다. 자세한 내용은 Service Quotas를 참조하세요.
HttpUnknownHost	URL이 유효하지만 서비스가 존재하지 않거나 연결할 수 없는 경우 생성됩니다.

메시지 브로커 지표

> **Note**
>
> 메시지 브로커 지표는 프로토콜 지표 아래의 CloudWatch 콘솔에 표시됩니다.

지표	설명
Connect.AuthError	메시지 브로커에서 권한을 부여할 수 없는 연결 요청 수. Protocol 차원에는 CONNECT 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Connect.ClientError	MQTT 메시지가 AWS IoT 할당량에 정의된 요구 사항을 충족하지 않아 거부된 연결 요청 수입니다. Protocol 차원에는 CONNECT 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Connect.ClientIDThrottle	클라이언트가 특정 클라이언트 ID에 대해 허용되는 연결 요청 빈도를 초과하기 때문에 제한된 연결 요청 수입니다. Protocol 차원에는 CONNECT 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Connect.ServerError	내부 오류가 발생하여 실패한 연결 요청 수. Protocol 차원에는 CONNECT 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.

지표	설명
Connect.Success	메시지 브로커에 대한 성공적인 연결 수. Protocol 차원에는 CONNECT 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Connect.Throttle	계정이 허용되는 연결 요청 빈도를 초과하여 병목 현상이 발생한 연결 요청 수. Protocol 차원에는 CONNECT 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Ping.Success	메시지 브로커에 수신된 ping 메시지 수. Protocol 차원에는 ping 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishIn.AuthError	메시지 브로커가 권한을 부여할 수 없는 게시 요청 수. Protocol 차원에는 메시지를 게시하는 데 사용된 프로토콜이 포함됩니다. HTTP 게시에서는 이 지표가 지원되지 않습니다.
PublishIn.ClientError	메시지가에 정의된 요구 사항을 충족하지 않아 메시지 브로커가 거부한 게시 요청 수입니다AWS IoT 할당량. Protocol 차원에는 메시지를 게시하는 데 사용된 프로토콜이 포함됩니다.
PublishIn.ServerError	내부 오류로 인해 메시지 브로커가 처리하지 못한 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishIn.Success	메시지 브로커가 성공적으로 처리한 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishIn.Throttle	클라이언트가 허용되는 인바운드 메시지 빈도를 초과하여 병목 현상이 발생한 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다. HTTP 게시에서는 이 지표가 지원되지 않습니다.

지표	설명
PublishOut.AuthError	AWS IoT가 권한을 부여할 수 없는 메시지 브로커의 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishOut.ClientError	메시지가에 정의된 요구 사항을 충족하지 않아 거부된 메시지 브로커의 게시 요청 수입니다AWS IoT 할당량. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishOut.Success	메시지 브로커의 성공적인 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishOut.Throttle	클라이언트가 허용되는 아웃바운드 메시지 빈도를 초과하여 병목 현상이 발생한 게시 요청 수입니다. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishRetained.AuthError	메시지 브로커가 권한을 부여할 수 없고 RETAIN 플래그가 설정된 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishRetained.ServerError	내부 오류로 인해 메시지 브로커가 처리하지 못한 유지된 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishRetained.Success	메시지 브로커가 성공적으로 처리하고 RETAIN 플래그가 설정된 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
PublishRetained.Throttle	클라이언트가 허용되는 인바운드 메시지 빈도를 초과하여 병목 현상이 발생하고 RETAIN 플래그가 설정된 게시 요청 수. Protocol 차원에는 PUBLISH 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.

지표	설명
Queued.Success	영구 세션에서 연결이 끊긴 클라이언트에 대해 메시지 브로커가 성공적으로 처리한 저장된 메시지 수입니다. QoS가 1인 메시지는 영구 세션이 있는 클라이언트의 연결이 끊긴 동안 저장됩니다.
Queued.Throttle	영구 세션이 있는 클라이언트의 연결이 끊긴 동안 저장할 수 없고 제한된 메시지 수입니다. 클라이언트가 계정당 초당 대기 중 메시지 제한을 초과할 때 발생합니다. QoS가 1인 메시지는 영구 세션이 있는 클라이언트의 연결이 끊긴 동안 저장됩니다.
Queued.ServerError	내부 오류로 인해 영구 세션에 대해 저장되지 않은 메시지 수입니다. 영구 세션이 있는 클라이언트의 연결이 끊어지면 서비스 품질(QoS)이 1인 메시지가 저장됩니다.
Subscribe.AuthError	권한을 부여할 수 없는 클라이언트의 구독 요청 수. Protocol 차원에는 SUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Subscribe.ClientError	SUBSCRIBE 메시지가 AWS IoT 할당량에 정의된 요구 사항을 충족하지 않아 거부된 구독 요청 수입니다. Protocol 차원에는 SUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Subscribe.ServerError	내부 오류가 발생하여 거부된 구독 요청 수. Protocol 차원에는 SUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Subscribe.Success	메시지 브로커가 성공적으로 처리한 구독 요청 수. Protocol 차원에는 SUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.

지표	설명
Subscribe.Throttle	AWS 계정의 허용되는 구독 요청 빈도 제한을 초과하여 스로틀링이 발생한 구독 요청 수입니다. 이러한 제한에는 계정당 초당 구독, 계정당 구독, AWS IoT Core message broker and protocol limits and quotas에 설명된 연결당 구독이 포함됩니다. Protocol 차원에는 SUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Throttle.Exceeded	이 지표는 MQTT 클라이언트가 연결 수준당 초당 패킷에서 제한될 때 CloudWatch에 표시됩니다. 이 지표는 HTTP 연결에는 적용되지 않습니다.
Unsubscribe.ClientError	UNSUBSCRIBE 메시지가 AWS IoT 할당량에 정의된 요구 사항을 충족하지 않아 거부된 구독 취소 요청 수입니다. Protocol 차원에는 UNSUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Unsubscribe.ServerError	내부 오류가 발생하여 거부된 구독 취소 요청 수. Protocol 차원에는 UNSUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Unsubscribe.Success	메시지 브로커가 성공적으로 처리한 구독 취소 요청 수. Protocol 차원에는 UNSUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.
Unsubscribe.Throttle	클라이언트가 허용되는 구독 취소 요청 빈도를 초과하여 거부된 구독 취소 요청 수. Protocol 차원에는 UNSUBSCRIBE 메시지를 전송하는 데 사용된 프로토콜이 포함됩니다.

디바이스 섀도우 지표

> **Note**
>
> 디바이스 섀도우 지표는 프로토콜 지표 아래의 CloudWatch 콘솔에 표시됩니다.

지표	설명
DeleteThingShadow.Accepted	성공적으로 처리된 DeleteThingShadow 요청 수입니다. Protocol 차원에는 요청하는 데 사용된 프로토콜이 포함됩니다.
GetThingShadow.Accepted	성공적으로 처리된 GetThingShadow 요청 수입니다. Protocol 차원에는 요청하는 데 사용된 프로토콜이 포함됩니다.
ListThingShadow.Accepted	성공적으로 처리된 ListThingShadow 요청 수입니다. Protocol 차원에는 요청하는 데 사용된 프로토콜이 포함됩니다.
UpdateThingShadow.Accepted	성공적으로 처리된 UpdateThingShadow 요청 수입니다. Protocol 차원에는 요청하는 데 사용된 프로토콜이 포함됩니다.

지표 로깅

지표	설명
CloudwatchLogs:LogGroupCreationFailed	CreateLogGroup 요청 실패 횟수입니다. AWS IoT Logging은 존재하지 않는 경우 CloudWatch 로그 그룹을 생성합니다. LogGroup 차원에는 요청을 수행하는 데 사용되는 로그 그룹이 포함됩니다.
CloudwatchLogs:LogStreamCreationFailed	CreateLogStream 요청 실패 횟수입니다. AWS IoT 로깅은 CloudWatch 로그 스트림을 생성하여 로그 이벤트를 푸시합니다. LogGroup 차원에는 요청을 수행하는 데 사용되는 로그 그룹이 포함됩니다.
CloudwatchLogs:PutRetentionPolicyFailed	PutRetentionPolicy 요청 실패 횟수입니다. AWS IoT Logging은 존재하지 않는 경우 CloudWatch 로그 그룹을 생성하고 보존 정책은 30일로 설정됩니다. LogGroup 차원에는 요청을 수행하는 데 사용되는 로그 그룹이 포함됩니다.

지표	설명
CloudwatchLogs:PutLogEvents Failed	PutLogEvents 요청 실패 횟수입니다. LogGroup 차원에는 요청을 수행하는 데 사용되는 CloudWatch 로그 그룹이 포함됩니다.

Jobs 지표

지표	설명
CanceledJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 CANCELED로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.
CanceledJobExecutionTotalCount	해당 작업에 대해 상태가 CANCELED인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.
ClientErrorCount	작업을 실행하는 동안 발생한 클라이언트 오류 수. JobId 차원에는 작업 ID가 포함됩니다.
FailedJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 FAILED로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.
FailedJobExecutionTotalCount	해당 작업에 대해 상태가 FAILED인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.
InProgressJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 IN_PROGRESS 로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.

지표	설명
InProgressJobExecutionTotalCount	해당 작업에 대해 상태가 IN_PROGRESS 인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.
RejectedJobExecutionTotalCount	해당 작업에 대해 상태가 REJECTED인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.
RemovedJobExecutionTotalCount	해당 작업에 대해 상태가 REMOVED인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.
QueuedJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 QUEUED로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.
QueuedJobExecutionTotalCount	해당 작업에 대해 상태가 QUEUED인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.
RejectedJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 REJECTED로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.
RemovedJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 REMOVED로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.
ServerErrorCount	작업을 실행하는 동안 발생한 서버 오류 수. JobId 차원에는 작업 ID가 포함됩니다.

지표	설명
SucceededJobExecutionCount	CloudWatch에 의해 결정된 기간 안에 상태가 SUCCESS로 변경된 작업 실행의 수입니다. (CloudWatch 지표에 대한 자세한 정보는 Amazon CloudWatch 지표를 참조하세요.) JobId 차원에는 작업 ID가 포함됩니다.
SucceededJobExecutionTotalCount	해당 작업에 대해 상태가 SUCCESS인 총 작업 실행 수. JobId 차원에는 작업 ID가 포함됩니다.

Device Defender Audit 지표

지표	설명
NonCompliantResources	점검을 통해 규정 미준수로 확인된 리소스 수입니다. 시스템은 수행된 감사의 각 점검 항목에 대해 규정 미준수 리소스 수를 보고합니다.
ResourcesEvaluated	규정 준수에 대해 평가된 리소스 수입니다. 시스템은 수행된 감사의 각 점검 항목에 대해 평가된 리소스 수를 보고합니다.
MisconfiguredDeviceDefenderNotification	에 대한 SNS 구성 AWS IoT Device Defender 이 잘못 구성되면 알려줍니다. Dimensions

Device Defender Detect 지표

지표	설명
NumOfMetricsExported	클라우드 측, 디바이스 측 또는 사용자 지정 지표에 대해 내보낸 지표의 수입니다. 시스템은 특정 지표를 바탕으로 계정에 대해 내보낸 지표의 수를 보고합니다.

지표	설명
	이 지표는 지표 내보내기를 이용하는 고객에게만 제공됩니다.
NumOfMetricsSkipped	클라우드 측, 디바이스 측 또는 사용자 지정 지표에 대해 건너뛴 지표의 수입니다. 시스템은 Device Defender Detect에 mqtt 주제에 게시할 수 있는 권한이 충분하지 않아 특정 지표에 대해 계정에서 건너뛰는 지표의 수를 보고합니다. 이 지표는 지표 내보내기를 이용하는 고객에게만 제공됩니다.
NumOfMetricsExceedingSizeLimit	크기가 MQTT 메시지 크기 제한을 초과하여 클라우드 측, 디바이스 측 또는 사용자 지정 지표에 대해 내보내기를 건너뛴 지표 수입니다. 시스템은 크기가 MQTT 메시지 크기 제한을 초과하여 특정 지표에 대해 계정 내보내기를 건너뛰는 지표의 수를 보고합니다. 이 지표는 지표 내보내기를 이용하는 고객에게만 제공됩니다.
Violations	마지막으로 평가가 수행된 시간 이후에 발견된 보안 프로필 동작의 새로운 위반 수입니다. 시스템은 계정, 특정 보안 프로필 및 특정 보안 프로필의 특정 동작에 대해 새로운 위반 수를 보고합니다.
ViolationsCleared	마지막으로 평가가 수행된 시간 이후에 해결된 보안 프로필 동작의 위반 수입니다. 시스템은 계정, 특정 보안 프로필과 보안 프로필의 특정 동작에 대해 해결된 위반 수를 보고합니다.
ViolationsInvalidated	마지막으로 평가가 수행된 시간 이후에 더 이상 정보를 사용할 수 없는(보고 디바이스가 보고를 중단했거나 어떠한 이유로든 더 이상 모니터링되지 않기 때문에) 보안 프로필 동작의 위반 수입니다. 시스템은 전체 계정, 특정 보안 프로필과 보안 프로필의 특정 동작에 대해 무효화된 위반 수를 보고합니다.

지표	설명
MisconfiguredDeviceDefenderNotification	에 대한 SNS 구성 AWS IoT Device Defender 이 잘못 구성되면 알려줍니다. Dimensions

디바이스 프로비저닝 지표

AWS IoT 플릿 프로비저닝 지표

지표	설명
ApproximateNumberOfThingsRegistered	플릿 프로비저닝에 의해 등록된 사물의 개수입니다. 카운트는 일반적으로 정확하지만 AWS IoT Core 의 분산 아키텍처로 인해 등록된 사물의 정확한 개수를 유지하기 어렵습니다. 이 지표에 사용하는 통계는 다음과 같습니다. • Max를 사용하여 등록된 총 사물 수를 보고합니다. CloudWatch 집계 기간 동안 등록된 사물의 수는 RegisterThingFailed 지표를 참조하세요. 차원: ClaimCertificateId
CreateKeysAndCertificateFailed	CreateKeysAndCertificate MQTT API를 호출할 때 발생한 실패 횟수입니다. 지표는 성공(값 = 0) 및 실패(값 = 1) 사례 모두에서 내보내집니다. 이 지표는 CloudWatch 지원 집계 기간 동안(예: 5분 또는 1시간) 생성 및 등록된 인증서 수를 추적하는 데 사용할 수 있습니다. 이 지표에 사용할 수 있는 통계는 다음과 같습니다. • Sum을 사용하여 실패한 호출 수를 보고합니다.

지표	설명
	• SampleCount를 사용하여 성공한 호출 및 실패한 호출 수를 보고합니다.
CreateCertificateFromCsrFailed	CreateCertificateFromCsr MQTT API를 호출할 때 발생한 실패 횟수입니다. 지표는 성공(값 = 0) 및 실패(값 = 1) 사례 모두에서 내보내집니다. 이 지표는 CloudWatch 지원 집계 기간(예: 5분 또는 1시간) 동안 등록된 사물 수를 추적하는 데 사용할 수 있습니다. 이 지표에 사용할 수 있는 통계는 다음과 같습니다. • Sum을 사용하여 실패한 호출 수를 보고합니다. • SampleCount를 사용하여 성공한 호출 및 실패한 호출 수를 보고합니다.
RegisterThingFailed	RegisterThing MQTT API를 호출할 때 발생한 실패 횟수입니다. 지표는 성공(값 = 0) 및 실패(값 = 1) 사례 모두에서 내보내집니다. 이 지표는 CloudWatch 지원 집계 기간(예: 5분 또는 1시간) 동안 등록된 사물 수를 추적하는 데 사용할 수 있습니다. 등록된 총 사물 수는 ApproximateNumberOfThingsRegistered 지표를 참조하세요. 이 지표에 사용할 수 있는 통계는 다음과 같습니다. • Sum을 사용하여 실패한 호출 수를 보고합니다. • SampleCount를 사용하여 성공한 호출 및 실패한 호출 수를 보고합니다. 차원: TemplateName

JIT 프로비저닝 지표

지표	설명
`ProvisionThing.ClientError`	클라이언트 오류로 인해 디바이스를 프로비저닝하지 못한 횟수입니다. 예를 들어 템플릿에 지정된 정책이 존재하지 않습니다.
`ProvisionThing.ServerError`	서버 오류로 인해 디바이스를 프로비저닝하지 못한 횟수입니다. 고객은 대기 후 디바이스 프로비저닝을 다시 시도할 수 있으며 문제가 동일하게 유지되는 경우 AWS IoT 에 문의할 수 있습니다.
`ProvisionThing.Success`	디바이스가 성공적으로 프로비저닝된 횟수입니다.

LoRaWAN 지표

다음 표에는에 대한 지표가 나와 있습니다 AWS IoT Core for LoRaWAN. 자세한 내용은 AWS IoT Core for LoRaWAN 지표를 참조하세요.

AWS IoT Core for LoRaWAN 지표

지표	설명
활성 디바이스/게이트웨이	계정의 활성 LoRaWAN 디바이스 및 게이트웨이 수입니다.
업링크 메시지 수	AWS 계정의 모든 활성 게이트웨이 및 디바이스에 대해 지정된 기간 내에 전송되는 업링크 메시지 수입니다. 업링크 메시지는 디바이스에서 로 전송되는 메시지입니다 AWS IoT Core for LoRaWAN.
다운링크 메시지 수	AWS 계정의 모든 활성 게이트웨이 및 디바이스에 대해 지정된 기간 내에 전송되는 다운링크 메시지 수입니다. 다운링크 메시지는에서 디바이스 AWS IoT Core for LoRaWAN 로 전송되는 메시지입니다.
메시지 손실률	디바이스를 추가하고에 연결한 후 디바이스는 업링크 메시지를 시작하여 클라우드와 메시지 교환을 시작할

지표	설명
	수 AWS IoT Core for LoRaWAN있습니다. 이 지표를 사용하여 손실된 업링크 메시지의 속도를 추적할 수 있습니다.
조인 지표	디바이스와 게이트웨이를 추가한 후 디바이스가 업링크 데이터를 전송하고 통신할 수 있도록 조인 절차를 수행합니다 AWS IoT Core for LoRaWAN. 이 지표를 사용하여 AWS 계정의 모든 활성 디바이스의 조인 지표에 대한 정보를 얻을 수 있습니다.
평균 수신 신호 강도 표시기(RSSI)	이 지표를 사용하여 지정된 기간 내에 평균 RSSI(수신 신호 강도 표시기)를 모니터링할 수 있습니다. RSSI는 신호가 양호한 무선 연결을 위해 충분히 강력한지 여부를 나타내는 측정치입니다. 이 값은 음수이며 강력한 연결을 위해서는 0에 가까워야 합니다.
평균 신호 대 잡음비(SNR)	이 지표를 사용하여 지정된 기간 내에 평균 SNR(신호 대 잡음비)을 모니터링할 수 있습니다. SNR은 수신된 신호가 양호한 무선 연결을 위한 노이즈 수준에 비해 충분히 강력한지 여부를 나타내는 측정치입니다. SNR 값은 양수이며 신호 출력이 노이즈 출력보다 강력함을 나타내려면 0보다 커야 합니다.
게이트웨이 가용성	이 지표를 사용하여 지정된 기간 내에 이 게이트웨이의 가용성에 대한 정보를 얻을 수 있습니다. 이 지표는 지정된 기간 동안 이 게이트웨이의 웹 소켓 연결 시간을 표시합니다.

JIT 프로비저닝 지표

지표	설명
`ProvisionThing.ClientError`	클라이언트 오류로 인해 디바이스를 프로비저닝하지 못한 횟수입니다. 예를 들어 템플릿에 지정된 정책이 존재하지 않습니다.

지표	설명
ProvisionThing.ServerError	서버 오류로 인해 디바이스를 프로비저닝하지 못한 횟수입니다. 고객은 대기 후 디바이스 프로비저닝을 다시 시도할 수 있으며 문제가 동일하게 유지되는 경우 AWS IoT 에 문의할 수 있습니다.
ProvisionThing.Success	디바이스가 성공적으로 프로비저닝된 횟수입니다.

플릿 인덱싱 지표

AWS IoT 플릿 인덱싱 지표

지표	설명
NamedShadowCountForDynamicGroupQueryLimitExceeded	동적 사물 그룹에서 특정 데이터 소스가 아닌 쿼리 용어에 대해 사물당 최대 25개의 명명된 섀도우가 처리됩니다. 사물에 대해 이 제한이 위반되면 NamedShadowCountForDynamicGroupQueryLimitExceeded 이벤트 유형이 내보내집니다.

지표 차원

지표는 네임스페이스를 사용하며 다음 차원의 지표를 제공합니다.

차원	설명
ActionType	요청에 따라 트리거되는 규칙에서 지정한 작업 유형.
BehaviorName	모니터링되고 있는 Device Defender Detect 보안 프로필 동작의 이름입니다.
ClaimCertificateId	장치를 프로비저닝하는 데 사용된 클레임의 certificateId 입니다.
CheckName	결과가 모니터링되고 있는 Device Defender 감사 점검 항목의 이름입니다.

차원	설명
JobId	진행 상황이나 메시지 연결 성공/실패를 모니터링 중인 작업의 ID.
Protocol	요청에 사용된 프로토콜. 유효 값: MQTT 또는 HTTP
RuleName	요청에 따라 트리거되는 규칙 이름
ScheduledAuditName	점검 결과가 모니터링되고 있는 Device Defender 예정된 감사의 이름입니다. 보고된 결과가 온디맨드로 수행된 감사에 대한 결과인 경우 이 이름에 OnDemand라는 값이 있습니다.
SecurityProfileName	동작이 모니터링되고 있는 Device Defender Detect 보안 프로필의 이름입니다.
TemplateName	프로비저닝 템플릿의 이름입니다.
SourceArn	탐지를 위한 보안 프로필 또는 감사를 위한 계정 arn을 참조합니다.
RoleArn	Device Defender가 수임하려고 시도한 역할을 나타냅니다.
TopicArn	Device Defender가 게시하려고 시도한 SNS 주제를 참조합니다.

차원	설명
`Error`	SNS 주제에 게시하려고 시도하는 동안 수신된 오류에 대한 간단한 설명을 제공합니다. 가능한 값은 다음과 같습니다. • 'KMSKeyNotFound': 주제에 대한 KMS 키가 존재하지 않음을 나타냅니다. • 'InvalidTopicName': SNS 주제가 유효하지 않음을 나타냅니다. • 'KMSAccessDenied': 역할에 주제의 KMS 키에 대한 권한이 없음을 나타냅니다. • 'AuthorizationError': 제공된 역할이 Device Defender가 SNS 주제에 게시할 수 있는 권한을 부여하지 않음을 나타냅니다. • 'SNSTopicNotFound': 제공된 SNS 주제가 존재하지 않음을 나타냅니다. • 'FailureToAssumeRole': 제공된 역할이 Device Defender가 역할을 수임할 권한을 부여하지 않음을 나타냅니다. • 'CrossRegionSNSTopic': SNS 주제가 다른 리전에 있음을 나타냅니다.

CloudWatch Logs AWS IoT 를 사용하여 모니터링

AWS IoT 로깅이 활성화되면는 메시지 브로커 및 규칙 엔진을 통해 디바이스에서 전달되는 각 메시지에 대한 진행 이벤트를 AWS IoT 보냅니다. CloudWatch 콘솔에서 CloudWatch Logs는 AWSIotLogs라는 로그 그룹에 나타납니다.

CloudWatch Logs에 대한 자세한 내용은 CloudWatch Logs를 참조하세요. 지원되는 AWS IoT CloudWatch Logs에 대한 자세한 내용은 CloudWatch Logs AWS IoT 로그 항목 섹션을 참조하세요.

CloudWatch 콘솔에서 AWS IoT 로그 보기

> **Note**
>
> `AWSIotLogsV2` 로그 그룹은 다음에 해당될 때까지 CloudWatch 콘솔에 표시되지 않습니다.
>
> - 로그인을 활성화했습니다 AWS IoT. 로그인을 활성화하는 방법에 대한 자세한 내용은 섹션을 참조하세요 AWS IoT. AWS IoT 로깅 구성
> - 일부 로그 항목은 AWS IoT 작업에 의해 작성되었습니다.

CloudWatch 콘솔에서 AWS IoT 로그를 보려면

1. https://console.aws.amazon.com/cloudwatch/로 이동합니다. 탐색 창에서 로그 그룹을 선택합니다.
2. 필터 텍스트 상자에 **`AWSIotLogsV2`**을 입력한 후 Enter 키를 누릅니다.
3. `AWSIotLogsV2` 로그 그룹을 두 번 클릭합니다.
4. 모두 검색(Search All)을 선택합니다. 계정에 대해 생성된 AWS IoT 로그의 전체 목록이 표시됩니다.
5. 확장 아이콘을 선택하여 개별 스트림을 확인합니다.

이벤트 필터링 텍스트 상자에 쿼리를 입력할 수도 있습니다. 몇 가지 시도해볼 만한 쿼리는 다음과 같습니다.

- `{ $.logLevel = "INFO" }`

  로그 수준이 `INFO`인 로그를 모두 찾습니다.
- `{ $.status = "Success" }`

  상태가 `Success`인 로그를 모두 찾습니다.
- `{ $.status = "Success" && $.eventType = "GetThingShadow" }`

  상태가 `Success`이고, 이벤트 유형이 `GetThingShadow`인 로그를 모두 찾습니다.

필터 표현식 작성에 대한 자세한 내용은 CloudWatch Logs 쿼리 단원을 참조하세요.

CloudWatch Logs AWS IoT 로그 항목

의 각 구성 요소는 고유한 로그 항목을 AWS IoT 생성합니다. 각 로그 항목에는 로그 항목이 생성되도록 한 작업을 지정하는 eventType이 있습니다. 이번 단원에서는 다음 AWS IoT 구성 요소에서 작성되는 로그 항목에 대해 설명합니다.

주제

- 메시지 브로커 로그 항목
- 서버 인증서 OCSP 로그 항목
- Device Shadow 로그 항목
- Rules engine 로그 항목
- Job 로그 항목
- Device provisioning 로그 항목
- Dynamic thing group 로그 항목
- 플릿 인덱싱 로그 항목
- 일반적인 CloudWatch Logs 속성

메시지 브로커 로그 항목

AWS IoT 메시지 브로커는 다음 이벤트에 대한 로그 항목을 생성합니다.

주제

- Connect 로그 항목
- Disconnect 로그 항목
- DeleteConnection 로그 항목
- GetRetainedMessage 로그 항목
- ListRetainedMessage 로그 항목
- Publish-In 로그 항목
- Publish-Out 로그 항목
- 대기 중 로그 항목
- Subscribe 로그 항목
- 로그 항목 구독 취소

Connect 로그 항목

AWS IoT 메시지 브로커는 MQTT 클라이언트가 연결Connect될 때 eventType이 인 로그 항목을 생성합니다.

Connect 로그 항목 예제

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

일반적인 CloudWatch Logs 속성 외에도Connect 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 MQTT 또는 HTTP입니다.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

Disconnect 로그 항목

AWS IoT 메시지 브로커는 MQTT 클라이언트가 연결 해제될 `Disconnect` 때 `eventType`이 인 로그 항목을 생성합니다.

Disconnect 로그 항목 예제

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID",
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT"
}
```

일반적인 CloudWatch Logs 속성 외에도`Disconnect` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 `MQTT` 또는 `HTTP`입니다.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

reason

클라이언트가 연결을 해제하는 이유입니다.

details

오류에 대한 간단한 설명입니다.

disconnectReason

클라이언트가 연결을 해제하는 이유입니다.

DeleteConnection 로그 항목

AWS IoT 메시지 브로커는 MQTT 클라이언트 연결이 삭제될 `DeleteConnection` 때 `eventType`이인 로그 항목을 생성합니다.

DeleteConnection 로그 항목 예제

```
{
    "timestamp": "2025-08-09 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteConnection",
    "protocol": "HTTP",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

일반적인 CloudWatch Logs 속성 외에도`DeleteConnection` 로그 항목에는 다음 속성이 포함됩니다.

clientId

해제될 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다. 위탁자 ID를 사용하여 위탁자를 식별하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM ID 및 자격 증명 비교를 참조하세요.

protocol

요청에 사용된 프로토콜. 유효한 값은 HTTP입니다.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

GetRetainedMessage 로그 항목

AWS IoT 메시지 브로커는가 GetRetainedMessage 호출될 GetRetainedMessage 때 eventType이 인 로그 항목을 생성합니다.

GetRetainedMessage 로그 항목 예제

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetRetainedMessage",
    "protocol": "HTTP",
    "topicName": "a/b/c",
    "qos": "1",
    "lastModifiedDate": "2017-08-07 18:47:56.664"
}
```

일반적인 CloudWatch Logs 속성 외에도GetRetainedMessage 로그 항목에는 다음 속성이 포함됩니다.

lastModifiedDate

보관된 메시지가 저장된 밀리초 단위의 Epoch 날짜 및 시간입니다 AWS IoT.

protocol

요청에 사용된 프로토콜. 유효한 값: HTTP.

qos

게시 요청에 사용되는 서비스 품질(QoS) 수준입니다. 유효한 값은 0 또는 1입니다.

topicName

구독하는 주제의 이름입니다.

ListRetainedMessage 로그 항목

AWS IoT 메시지 브로커는가 ListRetainedMessages 호출될 ListRetainedMessage 때 eventType이 인 로그 항목을 생성합니다.

ListRetainedMessage 로그 항목 예제

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ListRetainedMessage",
    "protocol": "HTTP"
}
```

일반적인 CloudWatch Logs 속성 로그ListRetainedMessage 항목에는 외에도 다음 속성이 포함됩니다.

protocol

요청에 사용된 프로토콜. 유효한 값: HTTP.

Publish-In 로그 항목

AWS IoT 메시지 브로커는 MQTT 메시지를 수신하면 eventType이 인 로그 항목을 생성합니다Publish-In.

Publish-In 로그 항목 예제

```
{
        "timestamp": "2017-08-10 15:39:30.961",
```

```
        "logLevel": "INFO",
        "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
        "accountId": "123456789012",
        "status": "Success",
        "eventType": "Publish-In",
        "protocol": "MQTT",
        "topicName": "$aws/things/MyThing/shadow/get",
        "clientId": "abf27092886e49a8a5c1922749736453",
        "principalId":
  "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
        "sourceIp": "205.251.233.181",
        "sourcePort": 13490,
        "retain": "True"
    }
```


일반적인 CloudWatch Logs 속성 외에도`Publish-In` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 `MQTT` 또는 `HTTP`입니다.

retain

메시지에 RETAIN 플래그가 `True` 값으로 설정되어 있을 때 사용되는 속성입니다. 메시지에 RETAIN 플래그가 설정되어 있지 않으면 이 속성은 로그 항목에 나타나지 않습니다. 자세한 정보는 MQTT 보존 메시지을 참조하십시오.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

topicName

구독하는 주제의 이름입니다.

Publish-Out 로그 항목

메시지 브로커는 MQTT 메시지를 게시하면 `eventType`이 `Publish-Out`인 로그 항목을 생성합니다.

Publish-Out 로그 항목 예제

```
{
    "timestamp": "2017-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Publish-Out",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/get",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

일반적인 CloudWatch Logs 속성 외에도`Publish-Out` 로그 항목에는 다음 속성이 포함됩니다.

clientId

해당 MQTT 주제에 대한 메시지를 수신하는 구독 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 `MQTT` 또는 `HTTP`입니다.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

topicName

구독하는 주제의 이름입니다.

대기 중 로그 항목

영구 세션이 있는 디바이스의 연결이 끊어지면 MQTT 메시지 브로커는 디바이스의 메시지를 저장하고 eventType이 인 로그 항목을 AWS IoT 생성합니다Queued. MQTT 영구 세션에 대한 자세한 내용은 섹션을 참조MQTT 지속적 세션하세요.

대기 중 서버 오류 로그 항목 예제

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "details": "Server Error"
}
```

일반적인 CloudWatch Logs 속성 서버Queued 오류 로그 항목에는 외에도 다음 속성이 포함됩니다.

clientId

메시지가 대기할 클라이언트의 ID입니다.

details

`Server Error`

서버 오류로 인해 메시지가 저장되지 않았습니다.

protocol

요청에 사용된 프로토콜. 값은 항상 `MQTT`입니다.

qos

요청의 서비스 품질(QoS) 수준입니다. QoS가 0인 메시지는 저장되지 않으므로 값은 항상 1입니다.

topicName

구독하는 주제의 이름입니다.

대기 중 성공 로그 항목 예시

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Success"
}
```

일반적인 CloudWatch Logs 속성 성공Queued 로그 항목에는 외에도 다음 속성이 포함됩니다.

clientId

메시지가 대기할 클라이언트의 ID입니다.

protocol

요청에 사용된 프로토콜. 값은 항상 MQTT입니다.

qos

요청의 서비스 품질(QoS) 수준입니다. QoS가 0인 메시지는 저장되지 않으므로 값은 항상 1입니다.

topicName

구독하는 주제의 이름입니다.

대기 중 제한된 로그 항목 예시

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
```

```
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "details": "Throttled while queueing offline message"
}
```

일반적인 CloudWatch Logs 속성 제한 로그 항목에는 외에도Queued 다음 속성이 포함됩니다.

clientId

메시지가 대기할 클라이언트의 ID입니다.

details

`Throttled while queueing offline message`

클라이언트가 `Queued messages per second per account` 제한을 초과하여 메시지가 저장되지 않았습니다.

protocol

요청에 사용된 프로토콜. 값은 항상 `MQTT`입니다.

qos

요청의 서비스 품질(QoS) 수준입니다. QoS가 0인 메시지는 저장되지 않으므로 값은 항상 1입니다.

topicName

구독하는 주제의 이름입니다.

Subscribe 로그 항목

AWS IoT 메시지 브로커는 MQTT 클라이언트가 주제를 구독할 `Subscribe` 때 `eventType`이 인 로그 항목을 생성합니다.

MQTT 3 구독 로그 항목 예시

```
{
    "timestamp": "2017-08-10 15:39:04.413",
    "logLevel": "INFO",
    "traceId": "7aa5c38d-1b49-3753-15dc-513ce4ab9fa6",
    "accountId": "123456789012",
    "status": "Success",
```

```
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/#",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

일반적인 CloudWatch Logs 속성 외에도`Subscribe` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

protocol

요청에 사용된 프로토콜. 값은 항상 `MQTT`입니다.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

topicName

구독하는 주제의 이름입니다.

MQTT 5 구독 로그 항목 예시

```
{
 "timestamp": "2022-11-30 16:24:15.628",
 "logLevel": "INFO",
 "traceId": "7aa5c38d-1b49-3753-15dc-513ce4ab9fa6",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "Subscribe",
 "protocol": "MQTT",
```

```
 "topicName": "test/topic1,$invalid/reserved/topic",
 "subscriptions": [
  {
   "topicName": "test/topic1",
   "reasonCode": 1
  },
  {
   "topicName": "$invalid/reserved/topic",
   "reasonCode": 143
  }
 ],
 "clientId": "abf27092886e49a8a5c1922749736453",
 "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
 "sourceIp": "205.251.233.181",
 "sourcePort": 13490
}
```

MQTT 5 구독 작업의 경우, 일반적인 CloudWatch Logs 속성 및 MQTT 3 구독 로그 항목 속성 외에도 MQTT 5 Subscribe 로그 항목에는 다음 속성이 포함됩니다.

구독

구독 요청의 요청된 주제와 개별 MQTT 5 사유 코드 간의 매핑 목록입니다. 자세한 내용은 MQTT 사유 코드를 참조하세요.

로그 항목 구독 취소

AWS IoT 메시지 브로커는 MQTT 클라이언트가 MQTT 주제 구독을 취소할 Unsubscribe 때 eventType이 인 로그 항목을 생성합니다.

MQTT 구독 취소 로그 항목 예시

```
{
    "timestamp": "2024-08-20 22:53:32.844",
    "logLevel": "INFO",
    "traceId": "db6bd09a-2c3f-1cd2-27cc-fd6b1ce03b58",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Unsubscribe",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
```

```
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

일반적인 CloudWatch Logs 속성 외에도`Unsubscribe` 로그 항목에는 다음 속성이 포함됩니다.

protocol

요청에 사용된 프로토콜. 값은 항상 `MQTT`입니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

sourceIp

요청이 시작된 IP 주소입니다.

sourcePort

요청이 시작된 포트입니다.

서버 인증서 OCSP 로그 항목

AWS IoT Core 는 다음 이벤트에 대한 로그 항목을 생성합니다.

주제

- RetrieveOCSPStapleData 로그 항목
- 프라이빗 엔드포인트에 대한 RetrieveOCSPStapleData 로그 항목

RetrieveOCSPStapleData 로그 항목

AWS IoT Core 는 서버가 OCSP 스테이플 데이터를 검색할 `eventType` `RetrieveOCSPStapleData` 때이 인 로그 항목을 생성합니다.

RetrieveOCSPStapleData 로그 항목 예시

다음은 `Success`의 로그 항목 예시입니다.

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "200",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
 "ocspResponseDetails": {
  "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```

다음은 Failure의 로그 항목 예시입니다.

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "A non 2xx HTTP response was received from the OCSP responder.",
 "eventType": "RetrieveOCSPStapleData",
```

```
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "444",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```


`RetrieveOCSPStaple` 작업의 경우 일반적인 CloudWatch Logs 속성 외에도 로그 항목에는 다음 속성이 포함됩니다.

reason

작업이 실패하는 이유입니다.

domainConfigName

도메인 구성의 이름입니다.

connectionDetails

연결 세부 정보에 대한 간략한 설명입니다.

- httpStatusCode

  서버에 대한 클라이언트의 요청에 따라 OCSP 응답기가 반환하는 HTTP 상태 코드입니다.

- ocspResponderUri

  서버 인증서에서 AWS IoT Core 가져오는 OCSP 응답기 URI입니다.

- sourceIp

  AWS IoT Core 서버의 소스 IP 주소입니다.

- targetIp

  OCSP 응답기의 대상 IP 주소입니다.

ocspRequestDetails

OCSP 요청의 세부 정보입니다.

- requesterName

  OCSP 응답기에 요청을 보내는 AWS IoT Core 서버의 식별자입니다.

- requestCertId

  요청의 인증서 ID입니다. OCSP 응답이 요청되는 인증서의 ID입니다.

ocspResponseDetails

OCSP 응답의 세부 정보입니다.

- responseCertId

  OCSP 응답의 인증서 ID입니다.

- ocspResponseStatus

  OCSP 응답의 상태입니다.

- certStatus

  인증서의 상태입니다.

- signature

  신뢰할 수 있는 엔터티가 응답에 적용하는 서명입니다.

- thisUpdateTime

  표시되는 상태가 올바른 것으로 알려진 시간입니다.

- nextUpdateTime

  인증서 상태에 대한 최신 정보를 사용할 수 있는 시점 또는 그 이전입니다.

- producedAtTime

  OCSP 응답자가 이 응답에 서명한 시간입니다.

- stapledDataPayloadSize

  스테이플링된 데이터의 페이로드 크기입니다.

프라이빗 엔드포인트에 대한 RetrieveOCSPStapleData 로그 항목

AWS IoT Core 는 서버가 OCSP 스테이플 데이터를 검색할 `eventType` `RetrieveOCSPStapleData` 때이 인 로그 항목을 생성합니다.

프라이빗 엔드포인트에 대한 RetrieveOCSPStapleData 로그 항목 예시

다음은 Success의 로그 항목 예시입니다.


```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
        "lambdaDetails": {
        "lambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "sourceArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
testDomainConfigure/6bzfg"
      },
        "authorizedResponderArn": "arn:aws:acm:us-west-2:123456789012:certificate/
certificate_ID",
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
 "ocspResponseDetails": {
  "responderId": "04:C1:3F:8F:27:D6:49:13:F8:DE:B2:36:9D:85:8E:F8:31:3B:A6:D0"
               "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```


다음은 Failure의 로그 항목 예시입니다.


```
{
 "timestamp": "2024-01-30 15:39:30.961",
```

```
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "The payload returned by the Lambda function exceeds the maximum response
 size of 7 kilobytes.",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
        "lambdaDetails": {
        "lambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "sourceArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
testDomainConfigure/6bzfg"
      },
        "authorizedResponderArn": "arn:aws:acm:us-west-2:123456789012:certificate/
certificate_ID",
  "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```


`RetrieveOCSPStaple` 작업의 경우, 일반적인 CloudWatch Logs 속성 및 RetrieveOCSPStapleData 로그 항목의 속성 외에도 프라이빗 엔드포인트의 로그 항목에는 다음 속성이 포함됩니다.

lambdaDetails

Lambda 함수의 세부 정보입니다.

- lambdaArn

  Lambda 함수의 ARN입니다.

- sourceArn

  도메인 구성의 ARN입니다.

authorizedResponderArn

도메인 구성에 구성된 ARN이 있는 경우, 권한 부여자 응답자의 ARN입니다.

Device Shadow 로그 항목

AWS IoT 디바이스 섀도우 서비스는 다음 이벤트에 대한 로그 항목을 생성합니다.

주제

- DeleteThingShadow 로그 항목
- GetThingShadow 로그 항목
- UpdateThingShadow 로그 항목

DeleteThingShadow 로그 항목

디바이스 섀도우 서비스는 디바이스 섀도우 삭제 요청이 수신되면 `eventType`이 `DeleteThingShadow`인 로그 항목을 작성합니다.

DeleteThingShadow 로그 항목 예제

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/delete"
}
```

외에도 일반적인 CloudWatch Logs 속성 `DeleteThingShadow` 로그 항목에는 다음 속성이 포함됩니다.

deviceShadowName

업데이트할 섀도우의 이름입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 `MQTT` 또는 `HTTP`입니다.

topicName

요청이 게시된 주제의 이름입니다.

GetThingShadow 로그 항목

디바이스 섀도우 서비스는 섀도우 가져오기 요청이 수신되면 `eventType`이 `GetThingShadow`인 로그 항목을 작성합니다.

GetThingShadow 로그 항목 예제

```
{
    "timestamp": "2017-08-09 17:56:30.941",
    "logLevel": "INFO",
    "traceId": "b575f19a-97a2-cf72-0ed0-c64a783a2504",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "MyThing",
    "topicName": "$aws/things/MyThing/shadow/get"
}
```

외에도 일반적인 CloudWatch Logs 속성 `GetThingShadow` 로그 항목에는 다음 속성이 포함됩니다.

deviceShadowName

요청한 섀도우의 이름입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 `MQTT` 또는 `HTTP`입니다.

topicName

요청이 게시된 주제의 이름입니다.

UpdateThingShadow 로그 항목

디바이스 섀도우 서비스는 디바이스 섀도우 업데이트 요청이 수신되면 `eventType`이 `UpdateThingShadow`인 로그 항목을 작성합니다.

UpdateThingShadow 로그 항목 예제

```
{
    "timestamp": "2017-08-07 18:43:59.436",
    "logLevel": "INFO",
    "traceId": "d0074ba8-0c4b-a400-69df-76326d414c28",
```

```
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/update"
}
```

외에도 일반적인 CloudWatch Logs 속성 UpdateThingShadow 로그 항목에는 다음 속성이 포함됩니다.

deviceShadowName

업데이트할 섀도우의 이름입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 MQTT 또는 HTTP입니다.

topicName

요청이 게시된 주제의 이름입니다.

Rules engine 로그 항목

AWS IoT 규칙 엔진은 다음 이벤트에 대한 로그를 생성합니다.

주제

- FunctionExecution 로그 항목
- RuleExecution 로그 항목
- RuleMatch 로그 항목
- RuleExecutionThrottled 로그 항목
- RuleNotFound 로그 항목
- StartingRuleExecution 로그 항목

FunctionExecution 로그 항목

규칙 엔진은 규칙의 SQL 쿼리가 외부 함수를 호출할 때 eventType이 FunctionExecution인 로그 항목을 작성합니다. 외부 함수는 규칙의 작업이 AWS IoT 또는 다른 웹 서비스에 HTTP 요청을 할 때 호출됩니다(예: get_thing_shadow 또는 호출machinelearning_predict).

FunctionExecution 로그 항목 예제

```
{
    "timestamp": "2017-07-13 18:33:51.903",
    "logLevel": "DEBUG",
    "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
    "status": "Success",
    "eventType": "FunctionExecution",
    "clientId": "N/A",
    "topicName":"rules/test",
    "ruleName": "ruleTestPredict",
    "ruleAction": "MachinelearningPredict",
    "resources": {
        "ModelId": "predict-model"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

일반적인 CloudWatch Logs 속성 외에도FunctionExecution 로그 항목에는 다음 속성이 포함됩니다.

clientId

FunctionExecution 로그에서는 N/A입니다.

principalId

요청하는 위탁자의 ID입니다.

리소스

규칙의 작업에서 사용하는 리소스 모음입니다.

ruleName

일치하는 규칙의 이름입니다.

topicName

구독하는 주제의 이름입니다.

RuleExecution 로그 항목

AWS IoT 규칙 엔진은 규칙의 작업을 트리거할 때 RuleExecution 로그 항목을 생성합니다.

RuleExecution 로그 항목 예제

```
{
    "timestamp": "2017-08-10 16:32:46.070",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "resources": {
        "RepublishTopic": "rules/republish"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

일괄 처리 로그 항목을 사용한 RuleExecution 예제

```
{
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "ruleName": "rule_test",
    "ruleAction": "HttpAction",
    "resources": {
        "Url": "https://example.com",
        "ConfirmationUrl": "https://example.com"
    },
    "details": "HttpAction made a request to the specified endpoint",
    "batchDetails": {
        "timestamps": [
            "1234567890123",
            "1234567890123",
            "1234567890123"
        ],
        "traceIds": [
            "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
            "30aa7ccc-1d23-0b97-aa7b-76196d83537c",
            "30aa7ccc-1d23-0b97-aa7b-76196d83537d"
```

```
        ],
        "clientIds": [
            "N/A",
            "N/A",
            "N/A"
        ],
        "topicNames": [
            "topic/ruletest",
            "topic/ruletest",
            "topic/ruletest"
        ],
        "principalIds": [
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
        ],
        "batchSize": 3,
        "batchSizeInBytes": 114
    }
}
```

일반적인 CloudWatch Logs 속성 뿐만 아니라 `RuleExecution` 로그 항목에는 다음과 같은 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

리소스

규칙의 작업에서 사용하는 리소스 모음입니다.

ruleAction

트리거된 작업의 이름입니다.

ruleName

일치하는 규칙의 이름입니다.

topicName

구독하는 주제의 이름입니다.

RuleMatch 로그 항목

AWS IoT 규칙 엔진은 메시지 브로커가 규칙과 일치하는 메시지를 수신할 `RuleMatch` 때 `eventType`이 인 로그 항목을 생성합니다.

RuleMatch 로그 항목 예제

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleMatch",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

외에도 일반적인 CloudWatch Logs 속성 `RuleMatch` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

ruleName

일치하는 규칙의 이름입니다.

topicName

구독하는 주제의 이름입니다.

RuleExecutionThrottled 로그 항목

실행이 제한되면 AWS IoT 규칙 엔진은 `eventType`가 인 로그 항목을 생성합니다`RuleExecutionThrottled`.

RuleExecutionThrottled 로그 항목 예시

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecutionThrottled",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleExecutionThrottled",
    "details": "Exection of Rule example_rule throttled"
}
```

일반적인 CloudWatch Logs 속성 외에도`RuleExecutionThrottled` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

details

오류에 대한 간단한 설명입니다.

principalId

요청하는 위탁자의 ID입니다.

reason

'RuleExecutionThrottled' 문자열

ruleName

트리거할 규칙의 이름.

topicName

게시된 주제의 이름입니다.

RuleNotFound 로그 항목

AWS IoT 규칙 엔진이 지정된 이름의 규칙을 찾을 수 없는 경우 `eventType`이 인 로그 항목을 생성합니다`RuleNotFound`.

RuleNotFound 로그 항목 예제

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleNotFound",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleNotFound",
    "details": "Rule example_rule not found"
}
```

일반적인 CloudWatch Logs 속성 외에도`RuleNotFound` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

details

오류에 대한 간단한 설명입니다.

principalId

요청하는 위탁자의 ID입니다.

reason

문자열 "RuleNotFound".

ruleName

찾을 수 없는 규칙의 이름입니다.

topicName

게시된 주제의 이름입니다.

StartingRuleExecution 로그 항목

AWS IoT 규칙 엔진이 규칙의 작업을 트리거하기 시작하면 `eventType`이 인 로그 항목이 생성됩니다`StartingRuleExecution`.

StartingRuleExecution 로그 항목 예제

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "DEBUG",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartingRuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

일반적인 CloudWatch Logs 속성 외에도`rule-` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

principalId

요청하는 위탁자의 ID입니다.

ruleAction

트리거된 작업의 이름입니다.

ruleName

일치하는 규칙의 이름입니다.

topicName

구독하는 주제의 이름입니다.

Job 로그 항목

AWS IoT 작업 서비스는 다음 이벤트에 대한 로그 항목을 생성합니다. 디바이스로부터 MQTT 또는 HTTP 요청이 수신되면 로그 항목이 생성됩니다.

주제

- DescribeJobExecution 로그 항목
- GetPendingJobExecution 로그 항목
- ReportFinalJobExecutionCount 로그 항목
- StartNextPendingJobExecution 로그 항목
- UpdateJobExecution 로그 항목

DescribeJobExecution 로그 항목

AWS IoT 작업 서비스는 서비스가 작업 실행을 설명하라는 요청을 수신할 `DescribeJobExecution` 때 `eventType`이 인 로그 항목을 생성합니다.

DescribeJobExecution 로그 항목 예제

```
{
    "timestamp": "2017-08-10 19:13:22.841",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DescribeJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/get",
    "clientToken": "myToken",
    "details": "The request status is SUCCESS."
}
```

일반적인 CloudWatch Logs 속성 외에도`GetJobExecution` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

clientToken

요청 멱등성을 보장하는 고유한 대/소문자 구분 식별자입니다. 자세한 내용은 멱등성 보장 방법 (How to Ensure Idempotency) 단원을 참조하세요.

details

작업 서비스의 다른 정보입니다.

jobId

작업 실행의 작업 ID입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 `MQTT` 또는 `HTTP`입니다.

topicName

요청에 사용된 주제입니다.

GetPendingJobExecution 로그 항목

AWS IoT 작업 서비스는 서비스가 작업 실행 요청을 수신할 `GetPendingJobExecution` 때 `eventType`이 인 로그 항목을 생성합니다.

GetPendingJobExecution 로그 항목 예제

```
{
    "timestamp": "2018-06-13 17:45:17.197",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "299966ad-54de-40b4-99d3-4fc8b52da0c5",
    "topicName": "$aws/things/299966ad-54de-40b4-99d3-4fc8b52da0c5/jobs/get",
    "clientToken": "24b9a741-15a7-44fc-bd3c-1ff2e34e5e82",
    "details": "The request status is SUCCESS."
}
```

일반적인 CloudWatch Logs 속성 외에도`GetPendingJobExecution` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

clientToken

요청 멱등성을 보장하는 고유한 대/소문자 구분 식별자입니다. 자세한 내용은 멱등성 보장 방법 (How to Ensure Idempotency) 단원을 참조하세요.

details

작업 서비스의 다른 정보입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 MQTT 또는 HTTP입니다.

topicName

구독하는 주제의 이름입니다.

ReportFinalJobExecutionCount 로그 항목

AWS IoT 작업 서비스는 ReportFinalJobExecutionCount 작업이 완료되면 entryType이 인 로그 항목을 생성합니다.

ReportFinalJobExecutionCount 로그 항목 예제

```
{
    "timestamp": "2017-08-10 19:44:16.776",
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ReportFinalJobExecutionCount",
    "jobId": "002",
    "details": "Job 002 completed. QUEUED job execution count: 0 IN_PROGRESS job
 execution count: 0 FAILED job execution count: 0 SUCCEEDED job execution count: 1
 CANCELED job execution count: 0 REJECTED job execution count: 0 REMOVED job execution
 count: 0"
}
```

일반적인 CloudWatch Logs 속성 외에도ReportFinalJobExecutionCount 로그 항목에는 다음 속성이 포함됩니다.

details

작업 서비스의 다른 정보입니다.

jobId

작업 실행의 작업 ID입니다.

StartNextPendingJobExecution 로그 항목

대기 중인 다음 작업 실행을 시작하라는 요청을 수신하면 AWS IoT 작업 서비스는 `eventType`이 인 로그 항목을 생성합니다`StartNextPendingJobExecution`.

StartNextPendingJobExecution 로그 항목 예제

```
{
    "timestamp": "2018-06-13 17:49:51.036",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartNextPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "95c47808-b1ca-4794-bc68-a588d6d9216c",
    "topicName": "$aws/things/95c47808-b1ca-4794-bc68-a588d6d9216c/jobs/start-next",
    "clientToken": "bd7447c4-3a05-49f4-8517-dd89b2c68d94",
    "details": "The request status is SUCCESS."
}
```

일반적인 CloudWatch Logs 속성 외에도`StartNextPendingJobExecution` 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

clientToken

요청 멱등성을 보장하는 고유한 대/소문자 구분 식별자입니다. 자세한 내용은 멱등성 보장 방법 (How to Ensure Idempotency) 단원을 참조하세요.

details

작업 서비스의 다른 정보입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 MQTT 또는 HTTP입니다.

topicName

요청에 사용된 주제입니다.

UpdateJobExecution 로그 항목

AWS IoT 작업 서비스는 서비스가 작업 실행 업데이트 요청을 수신할 UpdateJobExecution 때 eventType이 인 로그 항목을 생성합니다.

UpdateJobExecution 로그 항목 예제

```
{
    "timestamp": "2017-08-10 19:25:14.758",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/update",
    "clientToken": "myClientToken",
    "versionNumber": "1",
    "details": "The destination status is IN_PROGRESS. The request status is SUCCESS."
}
```

일반적인 CloudWatch Logs 속성 외에도UpdateJobExecution 로그 항목에는 다음 속성이 포함됩니다.

clientId

요청하는 클라이언트의 ID입니다.

clientToken

요청 멱등성을 보장하는 고유한 대/소문자 구분 식별자입니다. 자세한 내용은 멱등성 보장 방법 (How to Ensure Idempotency) 단원을 참조하세요.

details

작업 서비스의 다른 정보입니다.

jobId

작업 실행의 작업 ID입니다.

protocol

요청에 사용된 프로토콜. 유효한 값은 MQTT 또는 HTTP입니다.

topicName

요청에 사용된 주제입니다.

versionNumber

작업 실행 버전입니다.

Device provisioning 로그 항목

AWS IoT 디바이스 프로비저닝 서비스는 다음 이벤트에 대한 로그를 생성합니다.

주제

- GetDeviceCredentials 로그 항목
- ProvisionDevice 로그 항목

GetDeviceCredentials 로그 항목

AWS IoT 디바이스 프로비저닝 서비스는 클라이언트가를 호출할 `GetDeviceCredential` 때이 `eventType`인 로그 항목을 생성합니다`GetDeviceCredential`.

GetDeviceCredentials 로그 항목 예제

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "GetDeviceCredentials",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
}
```

일반적인 CloudWatch Logs 속성 외에도GetDeviceCredentials 로그 항목에는 다음 속성이 포함됩니다.

details

오류에 대한 간단한 설명입니다.

deviceCertificateId

디바이스 인증서의 ID입니다.

ProvisionDevice 로그 항목

AWS IoT 디바이스 프로비저닝 서비스는 클라이언트가를 호출할 ProvisionDevice 때이 eventType인 로그 항목을 생성합니다ProvisionDevice.

ProvisionDevice 로그 항목 예제

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "ProvisionDevice",
  "provisioningTemplateName" : "myTemplate",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
 }
```

외에도 일반적인 CloudWatch Logs 속성 ProvisionDevice 로그 항목에는 다음 속성이 포함됩니다.

details

오류에 대한 간단한 설명입니다.

deviceCertificateId

디바이스 인증서의 ID입니다.

provisioningTemplateName

프로비저닝 템플릿의 이름입니다.

Dynamic thing group 로그 항목

AWS IoT 동적 사물 그룹은 다음 이벤트에 대한 로그를 생성합니다.

주제

- AddThingToDynamicThingGroupsFailed 로그 항목

AddThingToDynamicThingGroupsFailed 로그 항목

AWS IoT 가 지정된 동적 그룹에 사물을 추가할 수 없는 경우 `eventType`이 인 로그 항목을 생성합니다`AddThingToDynamicThingGroupsFailed`. 사물이 동적 사물 그룹에 추가되기 위한 조건을 충족한 경우 발생하지만 동적 그룹에 추가할 수 없거나 동적 그룹에서 제거되었습니다. 이는 다음과 같은 이유로 발생할 수 있습니다.

- 사물이 이미 최대 그룹 수에 속합니다.
- --override-dynamic-groups 옵션은 사물을 정적 사물 그룹에 추가하는 데 사용되었습니다. 이를 가능하게 하기 위해 동적 사물 그룹에서 제거되었습니다.

자세한 내용은 동적 사물 그룹 제한 및 충돌을 참조하세요.

AddThingToDynamicThingGroupsFailed 로그 항목 예제

이 예제에서는 `AddThingToDynamicThingGroupsFailed` 오류의 로그 항목을 보여줍니다. 다음 예제에서 TestThing은 `dynamicThingGroupNames`에 나열된 동적 사물 그룹에 추가되기 위한 조건을 충족했지만 `reason`에 설명된 대로 해당 동적 그룹에 추가할 수 없습니다.

```
{
 "timestamp": "2020-03-16 22:24:43.804",
 "logLevel": "ERROR",
 "traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
 "accountId": "57EXAMPLE833",
 "status": "Failure",
 "eventType": "AddThingToDynamicThingGroupsFailed",
 "thingName": "TestThing",
 "dynamicThingGroupNames": [
  "DynamicThingGroup11",
  "DynamicThingGroup12",
  "DynamicThingGroup13",
  "DynamicThingGroup14"
 ],
```

```
 "reason": "The thing failed to be added to the given dynamic thing group(s) because
 the thing already belongs to the maximum allowed number of groups."
}
```

외에도 일반적인 CloudWatch Logs 속성 AddThingToDynamicThingGroupsFailed 로그 항목에는 다음 속성이 포함됩니다.

dynamicThingGroupNames

사물을 추가할 수 없는 동적 사물 그룹의 배열입니다.

reason

사물을 동적 사물 그룹에 추가할 수 없는 이유입니다.

thingName

동적 사물 그룹에 추가할 수 없는 사물의 이름입니다.

플릿 인덱싱 로그 항목

AWS IoT 플릿 인덱싱은 다음 이벤트에 대한 로그 항목을 생성합니다.

주제

- NamedShadowCountForDynamicGroupQueryLimitExceeded 로그 항목

NamedShadowCountForDynamicGroupQueryLimitExceeded 로그 항목

동적 그룹에서 특정 데이터 소스가 아닌 쿼리 용어에 대해 사물당 최대 25개의 명명된 섀도우가 처리됩니다. 사물에 대해 이 제한이 위반되면 NamedShadowCountForDynamicGroupQueryLimitExceeded 이벤트 유형이 내보내집니다.

NamedShadowCountForDynamicGroupQueryLimitExceeded 로그 항목 예제

이 예에서는 NamedShadowCountForDynamicGroupQueryLimitExceeded 오류의 로그 항목을 보여줍니다. 이 예에서는 reason 필드에 설명된 대로 모든 값 기반 DynamicGroup 결과가 정확하지 않을 수 있습니다.

```
{
"timestamp": "2020-03-16 22:24:43.804",
"logLevel": "ERROR",
"traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
```

```
"accountId": "571032923833",
"status": "Failure",
"eventType": "NamedShadowCountForDynamicGroupQueryLimitExceeded",
"thingName": "TestThing",
"reason": "A maximum of 25 named shadows per thing are processed for non-data source
 specific query terms in dynamic groups."
}
```

일반적인 CloudWatch Logs 속성

모든 CloudWatch Logs 로그 항목에는 다음 속성이 포함됩니다.

accountId

AWS 계정 ID.

eventType

로그가 작성된 이벤트 유형입니다. 이벤트 유형의 값은 로그 항목을 생성한 이벤트에 따라 다릅니다. 각 로그 항목 설명에는 해당 로그 항목의 `eventType` 값이 포함됩니다.

logLevel

사용 중인 로그 수준입니다. 자세한 정보는 the section called “로그 수준”을 참조하십시오.

status

요청 상태입니다.

timestamp

클라이언트가 AWS IoT 메시지 브로커에 연결된 시점의 사람이 읽을 수 있는 UTC 타임스탬프입니다.

traceId

특정 요청에서 모든 로그의 연관성을 나타내는 데 사용할 수 있도록 무작위로 생성되는 식별자입니다.

디바이스 측 로그를 Amazon CloudWatch에 업로드합니다.

기록 디바이스 측 로그를 Amazon CloudWatch에 업로드하여 현장에서의 디바이스 활동을 모니터링하고 분석할 수 있습니다. 디바이스 측 로그에는 시스템, 애플리케이션 및 디바이스 로그 파일이 포함될 수 있습니다. 이 프로세스는 CloudWatch Logs 규칙 작업 파라미터를 사용하여 디바이스 측 로그를 고객이 정의한 로그 그룹에 게시합니다.

작동 방식

이 프로세스는 AWS IoT 디바이스가 형식이 지정된 로그 파일이 포함된 MQTT 메시지를 AWS IoT 주제에 전송할 때 시작됩니다. AWS IoT 규칙은 메시지 주제를 모니터링하고 사용자가 정의한 CloudWatch Logs 그룹에 로그 파일을 전송합니다. 그런 다음 정보를 검토하고 분석할 수 있습니다.

주제

- MQTT 주제
- 규칙 작업

MQTT 주제

로그를 게시하는 데 사용할 MQTT 주제 네임스페이스를 선택합니다. 일반 주제 공간에는 `$aws/rules/things/thing_name/logs` 형식을, 오류 주제에는 `$aws/rules/things/thing_name/logs/errors` 형식을 사용하는 것이 좋습니다. 로그 및 오류 주제의 명명 구조는 권장되지만 필수는 아닙니다. 자세한 내용은 AWS IoT Core에 대한 MQTT 주제 설계를 참조하세요.

권장 공통 주제 공간을 사용하면 AWS IoT 기본 수집 예약 주제를 활용할 수 있습니다. AWS IoT 기본 수집은 AWS IoT 규칙 작업에서 지원하는 AWS 서비스로 디바이스 데이터를 안전하게 전송합니다. 수집 경로에서 게시/구독 메시지 브로커를 제거해 비용 효율성이 더 커집니다. 자세한 내용은 Basic Ingest를 사용하여 메시징 비용 절감을 참조하세요.

BatchMode를 사용하여 로그 파일을 업로드하는 경우 메시지는 UNIX 타임스탬프 및 메시지를 포함하는 특정 형식을 따라야 합니다. 자세한 내용은 CloudWatch Logs 규칙 작업의 BatchMode에 대한 MQTT 메시지 형식 요구 사항 주제를 참조하세요.

규칙 작업

가 클라이언트 디바이스에서 MQTT 메시지를 AWS IoT 수신하면 AWS IoT 규칙은 고객 정의 주제를 모니터링하고 사용자가 정의한 CloudWatch 로그 그룹에 콘텐츠를 게시합니다. 이 프로세스에서는 CloudWatch Logs 규칙 작업을 사용하여 MQTT에 로그 파일 배치가 있는지 모니터링합니다. 자세한 내용은 CloudWatch Logs AWS IoT 규칙 작업을 참조하세요.

BatchMode

`batchMode`는 AWS IoT CloudWatch Logs 규칙 작업 내의 부울 파라미터입니다. 이 파라미터는 선택 사항이며 기본적으로 꺼짐(`false`) 상태입니다. 디바이스 측 로그 파일을 배치로 업로드하려면 AWS IoT 규칙을 생성할 때이 파라미터를 켜야 합니다(`true`). 자세한 내용은 AWS IoT 규칙 작업 섹션의 CloudWatch Logs를 참조하세요.

AWS IoT 규칙을 사용하여 디바이스 측 로그 업로드

AWS IoT 규칙 엔진을 사용하여 기존 디바이스 측 로그 파일(시스템, 애플리케이션 및 디바이스 클라이언트 로그)의 로그 레코드를 Amazon CloudWatch에 업로드할 수 있습니다. 디바이스 측 로그가 MQTT 주제에 게시되면 CloudWatch Logs 규칙 작업이 메시지를 CloudWatch Logs로 전송합니다. 이 프로세스에서는 규칙 작업 `batchMode` 파라미터가 켜진 상태에서(`true`로 설정) 이를 사용하여 디바이스 로그를 일괄적으로 업로드하는 방법을 설명합니다.

CloudWatch에 디바이스 측 로그 업로드를 시작하려면 다음 사전 조건을 완료하세요.

사전 조건

시작하기 전에 다음을 수행하십시오.

- 사물 AWS IoT Core 로에 등록된 대상 IoT 디바이스를 AWS IoT 하나 이상 생성합니다. 자세한 내용은 사물 객체 생성을 참조하세요.
- 수집 및 오류에 대한 MQTT 주제 공간을 결정합니다. MQTT 주제 및 권장 명명 규칙에 대한 자세한 내용은 Amazon CloudWatch에 디바이스 측 로그 업로드의 MQTT 주제 MQTT 주제 섹션을 참조하세요.

이 사전 조건에 대한 자세한 내용은 CloudWatch에 디바이스 측 로그 업로드를 참조하세요.

CloudWatch 로그 그룹 생성

CloudWatch 로그 그룹을 생성하려면 다음 단계를 수행합니다. AWS Management Console 또는 AWS Command Line Interface ()를 통해 단계를 수행할지 여부에 따라 적절한 탭을 선택합니다AWS CLI.

AWS Management Console

를 사용하여 CloudWatch 로그 그룹을 생성하려면 AWS Management Console

1. 를 열고 CloudWatch로 AWS Management Console 이동합니다.
2. 탐색 모음에서 Logs(로그)를 선택한 다음, Log groups(로그 그룹)를 선택합니다.
3. 로그 그룹 생성을 선택합니다.
4. Log group name(로그 그룹 이름)을 업데이트하고 필요에 따라 Retention setting(보존 설정) 필드를 업데이트합니다.
5. 생성(Create)을 선택합니다.

AWS CLI

를 사용하여 CloudWatch 로그 그룹을 생성하려면 AWS CLI

1. 다음 명령을 실행하여 로그 그룹을 생성합니다. 자세한 내용은 AWS CLI v2 명령 참조 create-log-group 의 섹션을 참조하세요.

   예제(uploadLogsGroup)의 로그 그룹 이름을 원하는 이름으로 바꿉니다.

   ```
   aws logs create-log-group --log-group-name uploadLogsGroup
   ```

2. 로그 그룹이 제대로 생성되었는지 확인하려면 다음 명령을 실행합니다.

   ```
   aws logs describe-log-groups --log-group-name-prefix uploadLogsGroup
   ```

   샘플 출력:

   ```
   {
       "logGroups": [
           {
               "logGroupName": "uploadLogsGroup",
               "creationTime": 1674521804657,
               "metricFilterCount": 0,
               "arn": "arn:aws:logs:us-east-1:111122223333:log-
   group:uploadLogsGroup:*",
               "storedBytes": 0
           }
       ]
   }
   ```

주제 규칙 생성

AWS IoT 규칙을 생성하려면 다음 단계를 완료합니다. AWS Management Console 또는 AWS Command Line Interface ()를 통해 단계를 수행할지 여부에 따라 적절한 탭을 선택합니다AWS CLI.

AWS Management Console

를 사용하여 주제 규칙을 생성하려면 AWS Management Console

1. 규칙 허브를 엽니다.

a. AWS Management Console 을 열고 AWS IoT로 이동합니다.
b. 탐색 모음에서 Message routing(메시지 라우팅)을 선택한 다음 Rules(규칙)을 선택합니다.
c. 규칙 생성을 선택합니다.

2. 규칙 속성을 입력합니다.

a. 영숫자 Rule name(규칙 이름)을 입력합니다.
b. (선택 사항) Rule description(규칙 설명) 및 Tags(태그)를 입력합니다.
c. 다음을 선택합니다.

3. SQL 문을 입력합니다.

a. 수집을 위해 정의한 MQTT 주제를 사용하여 SQL 문을 입력합니다.

   예: `SELECT * FROM '$aws/rules/things/thing_name/logs'`
b. 다음을 선택합니다.

4. 규칙 작업을 입력합니다.

a. Actions 1(작업 1) 메뉴에서 CloudWatch logs(CloudWatch 로그)를 선택합니다.
b. Log group name(로그 그룹 이름)을 선택한 다음 생성한 로그 그룹을 선택합니다.
c. Use batch mode(배치 모드 사용)을 선택합니다.
d. 규칙의 IAM 역할을 지정합니다.

   규칙에 대한 IAM 역할이 있는 경우 다음을 수행합니다.

   1. IAM role(IAM 역할) 메뉴에서 IAM 역할을 선택합니다.

   규칙에 대한 IAM 역할이 없는 경우 다음을 수행합니다.

   1. Create new role(새 역할 생성)을 선택합니다.
   2. Role name(역할 이름)에 고유한 이름을 입력하고 Create(생성)를 선택합니다.
   3. IAM role(IAM 역할) 필드에서 IAM 역할 이름이 올바른지 확인합니다.
e. 다음을 선택합니다.

5. 템플릿 구성을 검토합니다.

a. 작업 템플릿의 설정을 검토하여 해당 설정이 올바른지 확인합니다.
b. 완료했으면 Create(생성)를 선택합니다.

AWS CLI

를 사용하여 IAM 역할 및 주제 규칙을 생성하려면 AWS CLI

1. AWS IoT 규칙에 권한을 부여하는 IAM 역할을 생성합니다.

   a. IAM 정책을 생성합니다.

   IAM 정책을 생성하려면 다음 명령을 실행합니다. `policy-name` 파라미터 값을 업데이트해야 합니다. 자세한 내용은 AWS CLI v2 명령 참조`create-policy`의 섹션을 참조하세요.

   > **Note**
   >
   > Microsoft Windows 운영 체제를 사용하는 경우 줄 끝 표시자 (\) 기호를 틱(`)이나 다른 문자로 바꿔야 할 수 있습니다.

```
aws iam create-policy \
    --policy-name uploadLogsPolicy \
    --policy-document \
'{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateTopicRule",
            "iot:Publish",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:GetLogEvents"
        ],
        "Resource": "*"
    }
}'
```

   b. 출력의 정책 ARN을 텍스트 편집기에 복사합니다.

   샘플 출력:

```
{
    "Policy": {
        "PolicyName": "uploadLogsPolicy",
        "PermissionsBoundaryUsageCount": 0,
        "CreateDate": "2023-01-23T18:30:10Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "AAABBBCCCDDDEEEFFFGGG",
        "DefaultVersionId": "v1",
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:policy/uploadLogsPolicy",
        "UpdateDate": "2023-01-23T18:30:10Z"
    }
}
```

c. IAM 역할 및 신뢰 정책을 생성합니다.

IAM 정책을 생성하려면 다음 명령을 실행합니다. `role-name` 파라미터 값을 업데이트해야 합니다. 자세한 내용은 AWS CLI v2 명령 참조`create-role`의 섹션을 참조하세요.

```
aws iam create-role \
--role-name uploadLogsRole \
--assume-role-policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

d. IAM 정책을 규칙에 연결합니다.

IAM 정책을 생성하려면 다음 명령을 실행합니다. `role-name` 및 `policy-arn` 파라미터 값을 업데이트해야 합니다. 자세한 내용은 AWS CLI v2 명령 참조`attach-role-policy`의 섹션을 참조하세요.

```
aws iam attach-role-policy \
--role-name uploadLogsRole \
--policy-arn arn:aws:iam::111122223333:policy/uploadLogsPolicy
```

e. 역할을 검토합니다.

IAM 역할이 제대로 생성되었는지 확인하려면 다음 명령을 실행합니다. `role-name` 파라미터 값을 업데이트해야 합니다. 자세한 내용은 AWS CLI v2 명령 참조[get-role](#)의 섹션을 참조하세요.

```
aws iam get-role --role-name uploadLogsRole
```

샘플 출력:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "uploadLogsRole",
        "RoleId": "AAABBBCCCDDDEEEFFFGGG",
        "Arn": "arn:aws:iam::111122223333:role/uploadLogsRole",
        "CreateDate": "2023-01-23T19:17:15+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "Statement1",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "iot.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Description": "",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {}
    }
}
```

2. 에서 AWS IoT 주제 규칙을 생성합니다 AWS CLI.

a. AWS IoT 주제 규칙을 생성하려면 다음 명령을 실행합니다. `--rule-name`,`sql` 문, `descriptionroleARN` 및 `logGroupName` 파라미터 값을 업데이트해야 합니다. 자세한 내용은 AWS CLI v2 명령 참조의 create-topic-rule을 참조하세요.

```
aws iot create-topic-rule \
--rule-name uploadLogsRule \
--topic-rule-payload \
 '{
  "sql":"SELECT * FROM 'rules/things/thing_name/logs'",
  "description":"Upload logs test rule",
  "ruleDisabled":false,
  "awsIotSqlVersion":"2016-03-23",
  "actions":[
   {"cloudwatchLogs":
    {"roleArn":"arn:aws:iam::111122223333:role/uploadLogsRole",
    "logGroupName":"uploadLogsGroup",
    "batchMode":true}
    }
   ]
}'
```

b. 규칙이 제대로 생성되었는지 확인하려면 다음 명령을 실행합니다. `role-name` 파라미터 값을 업데이트해야 합니다. 자세한 내용은 AWS CLI v2 명령 참조의 get-topic-rule을 참조하세요.

```
aws iot get-topic-rule --rule-name uploadLogsRule
```

샘플 출력:

```
{
    "ruleArn": "arn:aws:iot:us-east-1:111122223333:rule/uploadLogsRule",
    "rule": {
        "ruleName": "uploadLogsRule",
        "sql": "SELECT * FROM rules/things/thing_name/logs",
        "description": "Upload logs test rule",
        "createdAt": "2023-01-24T16:28:15+00:00",
        "actions": [
            {
                "cloudwatchLogs": {
                    "roleArn": "arn:aws:iam::111122223333:role/
uploadLogsRole",
```

```
                    "logGroupName": "uploadLogsGroup",
                    "batchMode": true
                }
            }
        ],
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23"
    }
}
```

로 디바이스 측 로그 전송 AWS IoT

디바이스 측 로그를 로 전송하려면 AWS IoT

1. 기록 로그를 로 보내려면 디바이스와 AWS IoT통신하여 다음을 확인합니다.

   - 로그 정보는 이 절차의 사전 요구 사항 섹션에 명시된 것처럼 올바른 주제 네임스페이스로 전송됩니다.

     예: `$aws/rules/things/thing_name/logs`

   - MQTT 메시지 페이로드 형식이 올바르게 지정되었습니다. MQTT 주제 및 권장 이름 지정 규칙에 대한 자세한 내용은 내의 MQTT 주제 섹션을 참조하세요디바이스 측 로그를 Amazon CloudWatch에 업로드합니다..

2. MQTT 클라이언트 내에서 AWS IoT MQTT 메시지가 수신되는지 확인합니다.

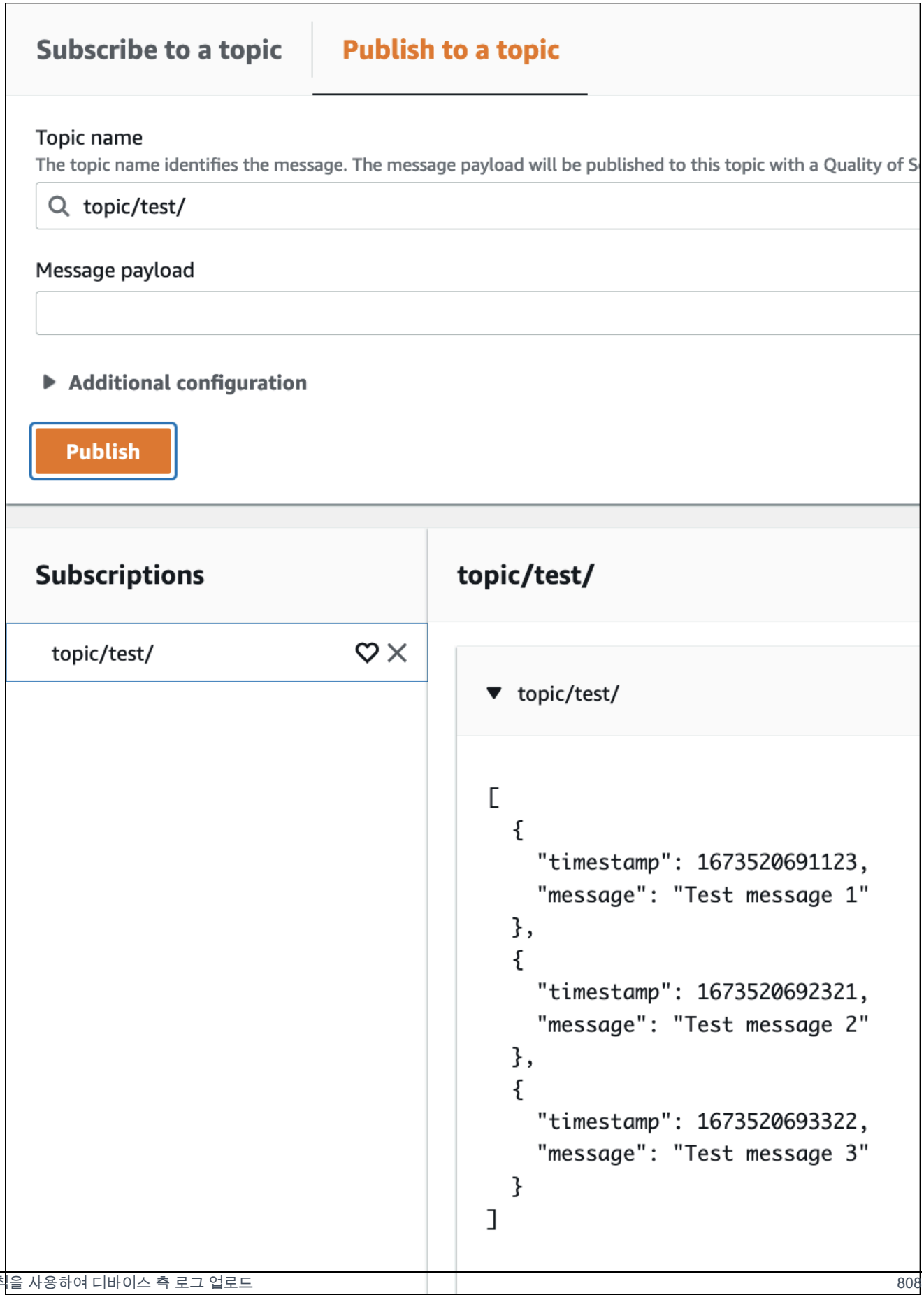
   a. 를 AWS Management Console 열고 로 이동합니다AWS IoT.

   b. MQTT 테스트 클라이언트를 보려면 탐색 모음에서 테스트, MQTT 테스트 클라이언트를 선택합니다.

   c. 주제 구독, 주제 필터에 주제 네임스페이스를 입력합니다.

   d. 구독을 선택합니다.

      MQTT 메시지는 다음과 같이 Subscriptions(구독) 및 Topic(주제) 테이블에 나타납니다. 이 메시지가 표시되려면 최대 5분이 걸릴 수 있습니다.


Subscribe to a topic
Publish to a topic
Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of S
topic/test/
Message payload
Additional configuration
Publish
Subscriptions
topic/test/
topic/test/
topic/test/
[
{
"timestamp": 1673520691123,
"message": "Test message 1"
},
{
"timestamp": 1673520692321,
"message": "Test message 2"
},
{
"timestamp": 1673520693322,
"message": "Test message 3"
}
]

로그 데이터 보기

CloudWatch Logs에서 로그 레코드 검토

1. 를 열고 CloudWatch로 AWS Management Console이동합니다.
2. 탐색 모음에서 로그, 로그 인사이트를 선택합니다.
3. 로그 그룹 선택(Select log group) 메뉴에서 AWS IoT 규칙에 지정한 로그 그룹을 선택합니다.
4. Logs insights 페이지에서 Run query(쿼리 실행)를 선택합니다.

를 사용하여 AWS IoT API 호출 로깅 AWS CloudTrail

AWS IoT 는 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다 AWS IoT. CloudTrail은 AWS IoT 콘솔의 호출 및 API에 대한 코드 호출을 포함하여에 대한 모든 API 호출을 이벤트 AWS IoT 로 캡처합니다. AWS IoT APIs 추적을 생성하면 이벤트를 포함하여 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다 AWS IoT. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 요청이 수행된 AWS IoT IP 주소, 요청이 수행된 사람, 요청이 수행된 시간 및 기타 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 설명은 AWS CloudTrail 사용자 가이드를 참조하십시오.

AWS IoT CloudTrail의 정보

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화됩니다. 활동이 발생하면 AWS IoT해당 활동은 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다 AWS 계정. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

AWS IoT에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 기본적으로 콘솔에서 추적을 생성하면 추적이 모든 AWS 리전에 적용됩니다. 추적은 AWS 파티션의 모든 AWS 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- 트레일 생성 개요
- CloudTrail 지원 서비스 및 통합

- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기

> **Note**
>
> AWS IoT 데이터 영역 작업(디바이스 측)은 CloudTrail에서 로깅되지 않습니다. CloudWatch를 사용하여 이러한 작업을 모니터링합니다.

일반적으로 변경을 수행하는 AWS IoT 컨트롤 플레인 작업은 CloudTrail에서 로깅합니다. CreateThing, CreateKeysAndCertificate 및 UpdateCertificate와 같은 호출은 CloudTrail 항목을 남기지만, ListThings 및 ListTopicRules와 같은 호출은 그렇지 않습니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 설명은 CloudTrail userIdentity 요소를 참조하세요.

AWS IoT 작업은 AWS IoT API 참조에 설명되어 있습니다. AWS IoT 무선 작업은 AWS IoT 무선 API 참조에 설명되어 있습니다.

AWS IoT 로그 파일 항목 이해

트레일이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함하고 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출에 대한 순서 지정된 스택 추적이 아니기 때문에 특정 순서로 표시되지 않습니다.

다음은 `AttachPolicy` 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예시입니다.

```
{
    "timestamp":"1460159496",
```

```
    "AdditionalEventData":"",
    "Annotation":"",
    "ApiVersion":"",
    "ErrorCode":"",
    "ErrorMessage":"",
    "EventID":"8bff4fed-c229-4d2d-8264-4ab28a487505",
    "EventName":"AttachPolicy",
    "EventTime":"2016-04-08T23:51:36Z",
    "EventType":"AwsApiCall",
    "ReadOnly":"",
    "RecipientAccountList":"",
    "RequestID":"d4875df2-fde4-11e5-b829-23bf9b56cbcd",
    "RequestParamters":{
        "principal":"arn:aws:iot:us-
east-1:123456789012:cert/528ce36e8047f6a75ee51ab7beddb4eb268ad41d2ea881a10b67e8e76924d894",
        "policyName":"ExamplePolicyForIoT"
    },
    "Resources":"",
    "ResponseElements":"",
    "SourceIpAddress":"52.90.213.26",
    "UserAgent":"aws-internal/3",
    "UserIdentity":{
        "type":"AssumedRole",
        "principalId":"AKIAI44QH8DHBEXAMPLE",
        "arn":"arn:aws:sts::12345678912:assumed-role/iotmonitor-us-east-1-beta-
InstanceRole-1C5T1YCYMHPYT/i-35d0a4b6",
        "accountId":"222222222222",
        "accessKeyId":"access-key-id",
        "sessionContext":{
            "attributes":{
                "mfaAuthenticated":"false",
                "creationDate":"Fri Apr 08 23:51:10 UTC 2016"
            },
            "sessionIssuer":{
                "type":"Role",
                "principalId":"AKIAI44QH8DHBEXAMPLE",
                "arn":"arn:aws:iam::123456789012:role/executionServiceEC2Role/
iotmonitor-us-east-1-beta-InstanceRole-1C5T1YCYMHPYT",
                "accountId":"222222222222",
                "userName":"iotmonitor-us-east-1-InstanceRole-1C5T1YCYMHPYT"
            }
        },
        "invokedBy":{
            "serviceAccountId":"111111111111"
```

```
        }
    },
    "VpcEndpointId":""
}
```

에 대한 규칙 AWS IoT

규칙은 디바이스가 상호 작용할 수 있는 기능을 제공합니다 AWS 서비스. MQTT 주제 스트림을 기반으로 규칙이 분석되고 작업이 수행됩니다. 규칙을 사용하여 다음 작업을 지원할 수 있습니다.

- 디바이스로부터 수신한 데이터를 보강 또는 필터링합니다.
- 디바이스로부터 수신한 데이터를 Amazon DynamoDB 데이터베이스에 기록합니다.
- Amazon S3에 파일을 저장합니다.
- Amazon SNS를 사용하는 모든 사용자에게 푸시 알림을 발송합니다.
- Amazon SQS 대기열에 데이터를 게시합니다.
- Lambda 함수를 호출하여 데이터를 추출합니다.
- Amazon Kinesis를 사용하여 다수의 디바이스로부터 메시지를 처리합니다.
- Amazon OpenSearch Service로 데이터를 전송합니다.
- CloudWatch 지표를 수집합니다.
- CloudWatch 경보를 변경합니다.
- 기계 학습(ML) 모델을 기반으로 한 예측을 위해 MQTT 메시지의 데이터를 Amazon SageMaker AI에 전송합니다.
- Salesforce IoT 입력 스트림에 메시지 전송
- Step Functions 상태 머신 프로세스를 시작합니다.
- 메시지 데이터를 AWS IoT Events 입력으로 전송합니다.
- AWS IoT SiteWise의 자산 속성에 메시지 데이터를 전송합니다.
- 메시지 데이터를 웹 애플리케이션 또는 서비스에 전송합니다.

규칙은 the section called “디바이스 통신 프로토콜”에서 지원하는 게시/구독 프로토콜을 통과하는 MQTT 메시지를 사용할 수 있습니다. 기본 수집 기능을 사용하여 메시징 비용 없이 디바이스 데이터를 이전에 AWS 서비스 나열된 로 안전하게 전송할 수도 있습니다. Basic Ingest(기본 수집) 기능은 수집 경로에서 게시/구독 메시지 브로커를 제거해 데이터 흐름을 최적화합니다. 따라서의 보안 및 데이터 처리 기능을 유지하면서 비용 효율적입니다 AWS IoT.

AWS IoT 가 이러한 작업을 수행하기 전에 사용자를 대신하여 리소스에 액세스할 수 AWS 있는 권한을 부여해야 합니다. 작업이 수행되면 AWS 서비스 사용하는에 대한 표준 요금이 발생합니다.

내용

- AWS IoT 규칙에 필요한 액세스 권한 부여
- 역할 전달 권한
- AWS IoT 규칙 생성
- AWS IoT 규칙 관리
- AWS IoT 규칙 작업
- 규칙 문제 해결
- AWS IoT 규칙을 사용하여 교차 계정 리소스에 액세스
- 오류 처리(오류 작업)
- Basic Ingest를 통한 메시징 비용 절감
- AWS IoT SQL 참조

AWS IoT 규칙에 필요한 액세스 권한 부여

IAM 역할을 사용하여 각 규칙이 액세스할 수 있는 AWS 리소스를 제어합니다. 규칙을 생성하기 전에 필요한 AWS 리소스에 대한 액세스를 허용하는 정책이 있는 IAM 역할을 생성해야 합니다. 규칙을 구현할 때이 역할을 수 AWS IoT 임합니다.

다음 단계를 완료하여 AWS IoT 규칙에 필요한 액세스 권한을 부여하는 IAM 역할 및 AWS IoT 정책을 생성합니다(AWS CLI).

1. 역할을 수임할 수 있는 AWS IoT 권한을 부여하는 다음 신뢰 정책 문서를 이라는 파일에 저장합니다`iot-role-trust.json`.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
            },
```

```
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:iot:us-east-1:123456789012:rule/
rulename"
            }
        }
        }
    ]
}
```

create-role 명령을 사용하여 iot-role-trust.json 파일을 지정하는 IAM 역할을 생성합니다.

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document
 file://iot-role-trust.json
```

이 명령의 출력은 다음과 같습니다.

```
{
 "Role": {
  "AssumeRolePolicyDocument": "url-encoded-json",
  "RoleId": "AKIAIOSFODNN7EXAMPLE",
  "CreateDate": "2015-09-30T18:43:32.821Z",
  "RoleName": "my-iot-role",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:role/my-iot-role"
 }
}
```

2. 다음 JSON을 my-iot-policy.json이라는 파일에 저장합니다.

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

이 JSON은 AWS IoT 관리자에게 DynamoDB에 대한 액세스 권한을 부여하는 정책 문서의 예입니다.

create-policy 명령을 사용하여 역할을 수임하고 `my-iot-policy.json` 파일을 전달할 때 AWS 리소스에 대한 AWS IoT 액세스 권한을 부여합니다.

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-policy.json
```

정책 AWS 서비스 에서에 대한 액세스 권한을 부여하는 방법에 대한 자세한 내용은 섹션을 AWS IoT참조하세요AWS IoT 규칙 생성.

create-policy 명령의 출력에는 정책의 ARN이 포함됩니다. 역할에 정책을 연결합니다.

```
{
 "Policy": {
  "PolicyName": "my-iot-policy",
  "CreateDate": "2015-09-30T19:31:18.620Z",
  "AttachmentCount": 0,
  "IsAttachable": true,
  "PolicyId": "ZXR6A36LTYANPAI7NJ5UV",
  "DefaultVersionId": "v1",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:policy/my-iot-policy",
  "UpdateDate": "2015-09-30T19:31:18.620Z"
 }
}
```

3. attach-role-policy 명령을 사용하여 정책을 역할에 연결합니다.

```
aws iam attach-role-policy --role-name my-iot-role --policy-arn
 "arn:aws:iam::123456789012:policy/my-iot-policy"
```

규칙 엔진 액세스 취소

규칙 엔진 액세스를 즉시 취소하려면 다음을 수행합니다.

1. 신뢰 정책에서 iot.amazonaws.com 제거

2. 단계에 따라 IoT 역할 세션을 취소

역할 전달 권한

규칙의 작업에 지정된 리소스에 액세스할 권한을 부여하는 IAM 역할은 규칙 정의의 일부입니다. 규칙 작업이 호출되면 규칙 엔진이 해당 역할을 수임합니다. 역할은 규칙 AWS 계정 과 동일한에 정의되어야 합니다.

규칙을 생성 또는 교체하는 것은 규칙 엔진에 역할을 전달하는 것과 동일한 것입니다. 이 작업을 실행하려면 `iam:PassRole` 권한이 필요합니다. 이 권한이 있는지 확인하려면 `iam:PassRole` 권한을 부여하는 정책을 생성하여 IAM 사용자에게 연결합니다. 다음 정책은 역할에 `iam:PassRole` 권한을 허용하는 방법을 보여줍니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt1",
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": [
    "arn:aws:iam::123456789012:role/myRole"
   ]
  }
 ]
}
```

이 정책 예제에서는 역할 `myRole`에 대한 `iam:PassRole` 권한이 부여됩니다. 역할은 역할의 ARN을 사용하여 지정됩니다. 이 정책을 IAM 사용자 또는 사용자가 속한 역할에 연결합니다. 자세한 내용은 관리형 정책 작업을 참조하세요.

> **Note**
>
> Lambda 함수는 리소스 기반 정책을 사용합니다. 이때 정책은 Lambda 함수 자체에 직접 연결됩니다. Lambda 함수를 호출하는 규칙을 생성하는 경우 역할은 전달하지 않습니다. 따라서 규칙을 생성하는 사용자에게 `iam:PassRole` 권한이 필요하지 않습니다. Lambda 함수 권한

부여에 대한 자세한 내용은 리소스 정책을 사용하여 권한 부여(Granting Permissions Using a Resource Policy) 단원을 참조하세요.

AWS IoT 규칙 생성

연결된 사물의 데이터를 라우팅하여 다른 AWS 서비스와 상호 작용하는 AWS IoT 규칙을 생성할 수 있습니다. AWS IoT 규칙은 다음 구성 요소로 구성됩니다.

규칙의 구성 요소

구성 요소	설명	필수 또는 선택
규칙 이름	규칙의 이름입니다. 규칙 이름에 개인 식별 정보를 사용하지 않는 것이 좋습니다.	필수 사항입니다.
규칙 설명	규칙에 대한 텍스트 설명입니다. 규칙 설명에 개인 식별 정보를 사용하지 않는 것이 좋습니다.	선택 사항.
SQL 문	MQTT 주제에서 수신된 메시지를 필터링하고 데이터를 다른 위치로 푸시하기 위해 간소화된 SQL 구문입니다. 자세한 내용은 AWS IoT SQL 참조 단원을 참조하십시오.	필수 사항입니다.
SQL 버전	규칙을 평가할 때 사용할 SQL 규칙의 버전입니다. 이 속성은 선택 사항이지만 SQL 버전을 지정하는 것이 좋습니다. AWS IoT Core 콘솔은 `2016-03-23` 기본적으로이 속성을 로 설정합니다. AWS CLI 명령 또는 CloudFormation 템플릿과 같이이 속성이 설정되지 않은 경우 `2015-10-08` 가 사용됩니다. 자세한 내용은 SQL 버전 단원을 참조하십시오.	필수 사항입니다.
하나 이상의 작업	작업은 규칙을 실행할 때 AWS IoT 수행됩니다. 예를 들어 DynamoDB 테이블에 데이터를 삽입하거나, Amazon S3 버킷에 데이터를 쓰거나, Amazon SNS 주제에 게시하거나, Lambda 함수를 호출할 수 있습니다.	필수 사항입니다.

구성 요소	설명	필수 또는 선택
오류 작업	작업은 규칙의 작업을 수행할 수 없을 때 AWS IoT 수행됩니다.	선택 사항.

AWS IoT 규칙을 생성하기 전에 필요한 AWS 리소스에 대한 액세스를 허용하는 정책을 사용하여 IAM 역할을 생성해야 합니다. 규칙을 구현할 때는이 역할을 수 AWS IoT 임합니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 및 역할 권한 전달을 참조하세요.

규칙을 생성할 때 주제에 얼마나 많은 데이터를 게시하는지 알고 있어야 합니다. 와일드카드 주제 패턴을 포함하는 규칙을 만들면 메시지의 많은 부분과 매칭될 수 있습니다. 이 경우 대상 작업에 사용되는 AWS 리소스의 용량을 늘려야 할 수 있습니다. 중복 처리를 방지하고 비용을 절감하려면, 재게시 규칙에서 와일드카드 주제 패턴을 피하는 것이 좋습니다.

> **Note**
>
> 규칙 생성 및 업데이트는 관리자 수준 작업입니다. 규칙을 생성 또는 업데이트할 수 있는 권한이 있는 모든 사용자는 규칙이 처리한 데이터에 액세스할 수 있습니다.

규칙 생성(콘솔)

규칙(AWS Management Console)을 생성하려면

AWS Management Console 명령을 사용하여 규칙을 생성합니다.

1. AWS IoT 콘솔을 엽니다.
2. 왼쪽 탐색에서 관리 섹션에서 메시지 라우팅을 선택합니다. 그런 다음 규칙을 선택합니다.
3. 규칙 페이지에서 규칙 생성을 선택합니다.
4. 규칙 속성 지정 페이지에서 규칙의 이름을 입력합니다. 규칙 설명 및 태그는 선택 사항입니다. 다음을 선택합니다.
5. SQL 문 구성 페이지에서 SQL 버전을 선택하고 SQL 문을 입력합니다. 예제 SQL 문은 `SELECT temperature FROM 'iot/topic' WHERE temperature > 50`일 수 있습니다. 자세한 내용은 SQL versions 및 AWS IoT SQL reference를 참조하세요.
6. 규칙 작업 연결 페이지에서 다른 AWS 서비스로 데이터를 라우팅하는 규칙 작업을 추가합니다.

1. 규칙 작업의 드롭다운 목록에서 규칙 작업을 선택합니다. 예를 들어 Kinesis Stream을 선택할 수 있습니다. 규칙 작업에 대한 자세한 내용은 AWS IoT rule actions를 참조하세요.
2. 선택한 규칙 작업에 따라 관련 구성 세부 정보를 입력합니다. 예를 들어 Kinesis Stream을 선택하는 경우 데이터 스트림 리소스를 선택하거나 생성하고 선택적으로 스트림의 샤드별로 데이터를 그룹화하는 데 사용되는 파티션 키와 같은 구성 세부 정보를 입력해야 합니다.
3. IAM 역할에서 엔드포인트에 대한 AWS IoT 액세스 권한을 부여할 역할을 선택하거나 생성합니다. AWS IoT 는 선택한 IAM 역할 `aws-iot-rule` 아래에 접두사가 인 정책을 자동으로 생성합니다. 보기를 선택하여 IAM 콘솔에서 IAM 역할과 정책을 볼 수 있습니다. 오류 작업은 선택 사항입니다. Error handling (error action)에서 자세한 내용을 확인할 수 있습니다. 규칙의 IAM 역할 생성에 대한 자세한 내용은 Grant a rule the access it requires를 참조하세요. 다음을 선택합니다.

7. 검토 및 생성 페이지에서 모든 구성을 검토하고 필요한 경우 편집합니다. 생성(Create)을 선택합니다.

규칙을 성공적으로 생성하면 규칙 페이지에 나열된 규칙이 표시됩니다. 규칙을 선택하여 규칙을 보고, 규칙을 편집하고, 규칙을 비활성화하고, 규칙을 삭제할 수 있는 세부 정보 페이지를 열 수 있습니다.

규칙 생성(CLI)

규칙(AWS CLI)을 생성하려면

규칙을 생성하려면 create-topic-rule 명령을 사용합니다.

```
aws iot create-topic-rule --rule-name myrule --topic-rule-payload file://myrule.json
```

다음 예제는 `iot/test` 주제에 전송된 모든 메시지를 지정된 DynamoDB 테이블에 삽입하는 규칙을 포함한 페이로드 파일입니다. SQL 문은 메시지를 필터링하고 역할 ARN은 DynamoDB 테이블에 쓸 수 있는 AWS IoT 권한을 부여합니다.

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "dynamoDB": {
    "tableName": "my-dynamodb-table",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
```

```
    "hashKeyField": "topic",
    "hashKeyValue": "${topic(2)}",
    "rangeKeyField": "timestamp",
    "rangeKeyValue": "${timestamp()}"
   }
  }
 ]
}
```

다음 예제는 iot/test 주제에 전송된 모든 메시지를 지정된 S3 버킷에 삽입하는 규칙을 포함한 페이로드 파일입니다. SQL 문은 메시지를 필터링하고 역할 ARN은 Amazon S3 버킷에 쓸 수 있는 AWS IoT 권한을 부여합니다.

```
{
 "awsIotSqlVersion": "2016-03-23",
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "actions": [
  {
   "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
    "bucketName": "amzn-s3-demo-bucket",
    "key": "myS3Key"
   }
  }
 ]
}
```

다음 예제는 데이터를 Amazon OpenSearch Service로 푸시하는 규칙을 포함하는 페이로드 파일입니다.

```
{
 "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "OpenSearch": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es",
    "endpoint": "https://my-endpoint",
    "index": "my-index",
    "type": "my-type",
```

```
    "id": "${newuuid()}"
   }
  }
 ]
}
```

다음 예제는 Lambda 함수를 호출하는 규칙을 포함한 페이로드 파일입니다.

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function"
   }
  }
 ]
}
```

다음 예제는 Amazon SNS 주제에 게시하는 규칙을 포함한 페이로드 파일입니다.

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sns": {
    "targetArn": "arn:aws:sns:us-west-2:123456789012:my-sns-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

다음 예제는 다른 MQTT 주제에 재게시하는 규칙을 포함한 페이로드 파일입니다.

```
{
 "sql": "expression",
 "ruleDisabled": false,
```

```
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "topic": "my-mqtt-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

다음 예제는 데이터를 Amazon Data Firehose 스트림으로 푸시하는 규칙을 포함한 페이로드 파일입니다.

```
{
 "sql": "SELECT * FROM 'my-topic'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "firehose": {
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "deliveryStreamName": "my-stream-name"
   }
  }
 ]
}
```

다음 예제는 MQTT 페이로드의 데이터가 1로 분류될 경우 Amazon SageMaker AI `machinelearning_predict` 함수를 사용하여 주제에 재게시하는 규칙을 포함한 예시 페이로드 파일입니다.

```
{
 "sql": "SELECT * FROM 'iot/test' where machinelearning_predict('my-model',
 'arn:aws:iam::123456789012:role/my-iot-aml-role', *).predictedLabel=1",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "topic": "my-mqtt-topic"
```

```
   }
  }
 ]
}
```

다음은 Salesforce IoT 클라우드 입력 스트림에 메시지를 게시하는 규칙을 가진 페이로드 파일의 예입니다.

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "salesforce": {
    "token": "ABCDEFGHI123456789abcdefghi123456789",
    "url": "https://ingestion-cluster-id.my-env.sfdcnow.com/streams/stream-id/
connection-id/my-event"
   }
  }
 ]
}
```

다음 예제는 Step Functions 상태 시스템 실행을 시작하는 규칙을 포함한 페이로드 파일입니다.

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "stepFunctions": {
    "stateMachineName": "myCoolStateMachine",
    "executionNamePrefix": "coolRunning",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

AWS IoT 규칙 관리

다음 작업을 사용하여 AWS IoT 규칙을 관리할 수 있습니다.

이 주제에서 수행할 작업

- 규칙 태그 지정
- 규칙 보기
- 규칙 삭제

규칙 태그 지정

새 규칙이나 기존 규칙에 다른 특수성을 더하기 위해 태깅을 적용할 수 있습니다. 태그 지정은 규칙의 키-값 페어를 활용하여 규칙이 AWS IoT 리소스 및 서비스에 적용되는 방식과 위치를 더 잘 제어할 수 있습니다. 예를 들어, 규칙이 베타 환경에만 적용되도록 범위를 제한하여 릴리스 전 테스팅(`Key=environment, Value=beta`)을 하거나 특정 엔드포인트에서 `iot/test` 주제로 전송된 모든 메시지를 캡처하여 Amazon S3 버킷에 저장하도록 할 수 있습니다.

IAM 정책 예시

규칙에 태깅 권한을 부여하는 방법을 보여주는 예로, 규칙을 만들고 베타 환경에만 태깅하기 위해 다음 명령을 실행하는 사용자를 생각해 보세요.

이 예에서 다음과 같이 대체합니다.

- 규칙 이름이 포함된 *MyTopicRuleName*
- 정책 문서의 이름이 포함된 *myrule.json*

```
aws iot create-topic-rule
    --rule-name MyTopicRuleName
    --topic-rule-payload file://myrule.json
    --tags "environment=beta"
```

이 예시에서는 다음 IAM 정책을 사용해야 합니다.

```
{
    "Version":"2012-10-17",
```

```
    "Statement":
    {
        "Action": [ "iot:CreateTopicRule", "iot:TagResource" ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:rule/MyTopicRuleName"
        ]
    }
}
```

위 예시는 베타 환경에만 적용되는 새로 만든 규칙인 MyTopicRuleName을 보여줍니다. MyTopicRuleName이 강조된 정책 설명의 iot:TagResource는 MyTopicRuleName 생성 또는 업데이트 시 태깅을 허용합니다. 규칙을 만들 때 사용되는 --tags "environment=beta" 파라미터는 MyTopicRuleName의 범위를 베타 환경으로만 제한합니다. --tags "environment=beta" 파라미터를 제거하면 MyTopicRuleName이 모든 환경에 적용됩니다.

AWS IoT 규칙에 대한 IAM 역할 및 정책 생성에 대한 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 섹션을 참조하세요.

리소스 태깅에 대한 자세한 내용은 AWS IoT 리소스에 태그 지정 섹션을 참조하세요.

규칙 보기

규칙을 나열하려면 list-topic-rules 명령을 사용합니다.

```
aws iot list-topic-rules
```

규칙에 대한 정보를 가져오려면 get-topic-rule 명령을 사용합니다.

```
aws iot get-topic-rule --rule-name myrule
```

규칙 삭제

규칙 사용을 마치면 이를 삭제할 수 있습니다.

규칙(AWS CLI)을 삭제하려면

규칙을 삭제하려면 delete-topic-rule 명령을 사용합니다.

```
aws iot delete-topic-rule --rule-name myrule
```

AWS IoT 규칙 작업

AWS IoT 규칙 작업은 규칙이 호출될 때 수행할 작업을 지정합니다. Amazon DynamoDB 데이터베이스로 데이터를 전송하고, Amazon Kinesis Data Streams로 데이터를 전송하고, 함수를 AWS Lambda 호출하는 작업을 정의할 수 있습니다.는 작업의 서비스를 사용할 수 AWS 리전 있는에서 다음 작업을 AWS IoT 지원합니다.

규칙 작업	설명	API의 이름
Apache Kafka	Apache Kafka 클러스터에 메시지를 전송합니다.	`kafka`
CloudWatch 경보	Amazon CloudWatch 경보 상태를 변경합니다.	`cloudwatchAlarm`
CloudWatch Logs	Amazon CloudWatch Logs로 메시지를 전송합니다.	`cloudwatchLogs`
CloudWatch 지표	CloudWatch 지표에 메시지를 전송합니다.	`cloudwatchMetric`
DynamoDB	DynamoDB 테이블에 메시지를 전송합니다.	`dynamoDB`
DynamoDBv2	메시지 데이터를 DynamoDB 테이블의 여러 열에 전송합니다.	`dynamoDBv2`
Elasticsearch	OpenSearch 엔드포인트에 메시지를 전송합니다.	`OpenSearch`
HTTP	HTTPS 엔드포인트에 메시지를 게시합니다.	`http`
AWS IoT Events	AWS IoT Events 입력에 메시지를 전송합니다.	`iotEvents`

규칙 작업	설명	API의 이름
AWS IoT SiteWise	메시지 데이터를 AWS IoT SiteWise 자산 속성으로 전송합니다.	`iotSiteWise`
Firehose	메시지를 Firehose 전송 스트림에 전송합니다.	`firehose`
Kinesis Data Streams	메시지를 Kinesis 데이터 스트림에 전송합니다.	`kinesis`
Lambda	메시지 데이터를 입력으로 사용하여 Lambda 함수를 호출합니다.	`lambda`
Location	Amazon Location Service에 위치 데이터를 전송합니다.	`location`
OpenSearch	Amazon OpenSearch Service 엔드포인트에 메시지를 전송합니다.	`OpenSearch`
재게시	메시지를 다른 MQTT 주제에 재게시합니다.	`republish`
S3	메시지를 Amazon Simple Storage Service(Amazon S3) 버킷에 저장합니다.	`s3`
Salesforce IoT	메시지를 Salesforce IoT 입력 스트림에 전송합니다.	`salesforce`
SNS	메시지를 Amazon Simple Notification Service(Amazon SNS) 푸시 알림으로 게시합니다.	`sns`

규칙 작업	설명	API의 이름
SQS	메시지를 Amazon Simple Queue Service(Amazon SQS) 대기열에 전송합니다.	`sqs`
단계 함수	AWS Step Functions 상태 시스템을 시작합니다.	`stepFunctions`
the section called “Timestream”	메시지를 Amazon Timestream 데이터베이스 테이블에 전송합니다.	`timestream`

참고

- 규칙 작업이 해당 리소스와 상호 작용할 수 있도록 AWS 리전 다른 서비스의 리소스와 동일한에서 규칙을 정의합니다.
- AWS IoT 규칙 엔진은 간헐적인 오류가 발생할 경우 작업을 여러 번 시도할 수 있습니다. 모든 시도가 실패하면 메시지가 삭제되고 CloudWatch Logs에서 오류를 확인할 수 있습니다. 각 규칙에 대해 실패 발생 후 간접 호출되는 오류 작업을 지정할 수 있습니다. 자세한 내용은 오류 처리(오류 작업) 단원을 참조하십시오.
- 일부 규칙 작업은 AWS Key Management Service (AWS KMS)와 통합되는 서비스에서 작업을 활성화하여 저장 데이터 암호화를 지원합니다. 고객 관리형 AWS KMS key (KMS 키)를 사용하여 저장 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 고객 관리형 KMS 키에 대한 권한을 관리하는 방법을 알아보려면 해당 서비스 가이드의 데이터 암호화 주제를 참조하세요. 고객 관리형 KMS 키에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS Key Management Service 개념을 참조하세요.

  , `aws_lambda()`, , , , , , 등의 외부 함수를 포함하여 오류 작업의 SQL 문에서 함수 또는 대체 템플릿을 사용할 수 `get_dynamodb()` `get_registry_data()` `get_thing_shadow()` `get_secret()` `machinelearning_predict()`있습니다`decode()`. 오류 작업에서 외부 함수를 호출해야 하는 경우 오류 작업을 직접적으로 호출하면 외부 함수에 대한 추가 요금이 부과될 수 있습니다.

Apache Kafka

Apache Kafka(Kafka) 작업은 Amazon Managed Streaming for Apache Kafka(Amazon MSK), Confluent Cloud와 같은 서드파티 공급자가 관리하는 Apache Kafka 클러스터 또는 자체 관리형 Apache Kafka 클러스터로 직접 메시지를 전송합니다. Kafka 규칙 작업을 사용하면 IoT 데이터를 Kafka 클러스터로 라우팅할 수 있습니다. 이를 통해 스트리밍 분석, 데이터 통합, 시각화 및 미션 크리티컬 비즈니스 애플리케이션과 같은 다양한 목적을 위해 고성능 데이터 파이프라인을 구축할 수 있습니다.

> **Note**
>
> 이 주제에서는 Apache Kafka 플랫폼 및 관련 개념에 대해 잘 알고 있다고 가정합니다. Apache Kafka에 대한 자세한 내용은 Apache Kafka 섹션을 참조하세요. MSK Serverless는 지원되지 않습니다. MSK Serverless 클러스터는 현재 Apache Kafka 규칙 작업이 지원하지 않는 IAM 인증을 통해서만 수행할 수 있습니다. Confluent AWS IoT Core 를 사용하여를 구성하는 방법에 대한 자세한 내용은 Confluent 활용 및 IoT 디바이스 및 데이터 관리 문제 AWS 해결을 참조하세요.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- 가 `ec2:CreateNetworkInterface`, , , , `ec2:DescribeNetworkInterfaces`, `ec2:CreateNetworkInterfacePermission`, `ec2:DescribeVpcAttribute`및 `ec2:DescribeSecurityGroups` 작업을 수행하기 위해 `ec2:DeleteNetworkInterface` `ec2:DescribeSubnets` `ec2:DescribeVpcs`수임할 AWS IoT 수 있는 IAM 역할입니다. 이 역할은 Amazon Virtual Private Cloud에 대한 탄력적인 네트워크 인터페이스를 생성 및 관리하여 Kafka 브로커에 도달할 수 있습니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT Core 있습니다.

  네트워크 인터페이스에 대한 자세한 내용은 Amazon EC2 사용 설명서의 탄력적 네트워크 인터페이스를 참조하세요.

  지정한 역할에 연결된 정책은 다음 예시와 같습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcAttribute",
            "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

- AWS Secrets Manager 를 사용하여 Kafka 브로커에 연결하는 데 필요한 자격 증명을 저장하는 경우 `secretsmanager:GetSecretValue` 및 `secretsmanager:DescribeSecret` 작업을 수행하기 위해가 맡을 AWS IoT Core 수 있는 IAM 역할을 생성해야 합니다.

  지정한 역할에 연결된 정책은 다음 예시와 같습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:kafka_client_truststore-*",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:kafka_keytab-*"
            ]
        }
```

```
    ]
}
```

- Amazon Virtual Private Cloud(VPC) 내에서 Apache Kafka 클러스터를 실행할 수 있습니다. Apache Kafka Virtual Private Cloud(VPC) 대상을 생성하고 서브넷에서 NAT 게이트웨이를 사용하여의 메시지를 퍼블릭 Kafka 클러스터 AWS IoT 로 전달해야 합니다. 규칙 엔진은 AWS IoT 대상에 나열된 각 서브넷에 네트워크 인터페이스를 생성하여 트래픽을 VPC로 직접 라우팅합니다. 대상 지정 시 AWS IoT 규칙 엔진은 VPC 규칙 작업을 자동으로 생성합니다. VPC 규칙 작업에 대한 자세한 내용은 Apache Kafka Virtual Private Cloud(VPC) 대상 단원을 참조하세요.
- 고객 관리형 AWS KMS key (KMS 키)를 사용하여 저장 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 Amazon MSK 암호화를 참조하세요.

파라미터

이 작업을 사용하여 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

destinationArn

Apache Kafka Virtual Private Cloud(VPC) 대상의 Amazon 리소스 이름(ARN)입니다. 대상 생성에 대한 자세한 내용은 섹션을 참조하세요Apache Kafka Virtual Private Cloud(VPC) 대상.

주제

메시지에 대한 Kafka 주제는 Kafka 브로커로 전송됩니다.

대체 템플릿을 사용하여 이 필드를 대체할 수 있습니다. 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요.

key(선택 사항)

Kafka 메시지 키입니다.

대체 템플릿을 사용하여 이 필드를 대체할 수 있습니다. 자세한 내용은 the section called “대체 템플릿” 단원을 참조하십시오.

헤더(선택 사항)

지정한 Kafka 헤더 목록입니다. 각 헤더는 Kafka 작업을 생성할 때 지정할 수 있는 키-값 페어입니다. 메시지 페이로드를 수정하지 않고도 이러한 헤더를 사용하여 IoT 클라이언트의 데이터를 다운스트림 Kafka 클러스터로 라우팅할 수 있습니다.

대체 템플릿을 사용하여 이 필드를 대체할 수 있습니다. Kafka 작업의 헤더에서 인라인 규칙의 함수를 대체 템플릿으로 전달하는 방법을 이해하려면 예시를 참조하세요. 자세한 내용은 the section called “대체 템플릿” 단원을 참조하십시오.

> **Note**
>
> 바이너리 형식의 헤더는 지원되지 않습니다.

partition(선택 사항)

Kafka 메시지 파티션입니다.

대체 템플릿을 사용하여 이 필드를 대체할 수 있습니다. 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요.

clientProperties

Apache Kafka 프로듀서 클라이언트의 속성을 정의하는 객체입니다.

acks(선택 사항)

프로듀서가 요청이 완료되었음을 고려하기 전에 서버가 수신해야 하는 승인 수입니다.

값으로 0을 지정하면 프로듀서는 서버의 승인을 기다리지 않습니다. 서버에서 메시지를 받지 못하면 프로듀서는 메시지를 전송하려고 다시 시도하지 않습니다.

유효한 값: `-1`, `0`, `1`, `all`. 기본값은 `1`입니다.

bootstrap.servers

Kafka 클러스터에 대한 초기 연결을 설정하는 데 사용되는 호스트 및 포트 페어 목록(예: `host1:port1`, `host2:port2`)입니다.

compression.type(선택 사항)

생산자가 생성한 모든 데이터에 대한 압축 유형입니다.

유효한 값:none ,`gzip` ,`snappy` ,`lz4` .`zstd` 기본값은 none입니다.

security.protocol

Kafka 브로커에 연결하는 데 사용되는 보안 프로토콜입니다.

유효한 값: `SSL`, `SASL_SSL`. 기본값은 `SSL`입니다.

key.serializer

`ProducerRecord`와 함께 제공되는 키 객체를 바이트로 변환하는 방법을 지정합니다.

유효한 값: `StringSerializer`.

value.serializer

`ProducerRecord`와 함께 제공하는 값 객체를 바이트로 변환하는 방법을 지정합니다.

유효한 값: `ByteBufferSerializer`.

ssl.truststore

base64 형식의 truststore 파일 또는 AWS Secrets Manager의 truststore 파일 위치입니다. Amazon 인증 기관(CA)에서 사용자의 트러스트 스토어를 신뢰하는 경우에는 이 값이 필요하지 않습니다.

이 필드는 대체 템플릿을 지원합니다. Kafka 브로커에 연결하는 데 필요한 보안 인증 정보를 저장하는 데 Secrets Manager를 사용하는 경우, `get_secret` SQL 함수를 사용하여 이 필드의 값을 검색할 수 있습니다. 대체 변수에 대한 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요. `get_secret` SQL 함수에 대한 자세한 내용은 the section called “get_secret(secretId, secretType, key, roleArn)” 섹션을 참조하세요. truststore가 파일 형식인 경우 `SecretBinary` 파라미터를 사용합니다. truststore가 문자열 형식인 경우 `SecretString` 파라미터를 사용합니다.

이 값의 최대 크기는 65KB입니다.

ssl.truststore.password

truststore의 암호입니다. 이 값은 truststore의 암호를 만든 경우에만 필요합니다.

ssl.keystore

keystore 파일입니다. `SSL`을 `security.protocol`의 값으로 지정할 때 이 값이 필요합니다.

이 필드는 대체 템플릿을 지원합니다. Secrets Manager를 사용하여 Kafka 브로커에 연결하는 데 필요한 보안 인증 정보를 저장합니다. `get_secret` SQL 함수를 사용하여 이 필드의 값을 검색합니다. 대체 변수에 대한 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요. `get_secret` SQL 함수에 대한 자세한 내용은 the section called “get_secret(secretId, secretType, key, roleArn)” 섹션을 참조하세요. `SecretBinary` 파라미터를 사용합니다.

ssl.keystore.password

keystore 파일의 스토어 암호입니다. `ssl.keystore`에 값이 지정된 경우 이 값이 필요합니다.

이 필드의 값은 일반 텍스트일 수 있습니다. 이 필드는 대체 템플릿도 지원합니다. Secrets Manager를 사용하여 Kafka 브로커에 연결하는 데 필요한 보안 인증 정보를 저장합니다. `get_secret` SQL 함수를 사용하여 이 필드의 값을 검색합니다. 대체 변수에 대한 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요. `get_secret` SQL 함수에 대한 자세한 내용은 the section called “get_secret(secretId, secretType, key, roleArn)” 섹션을 참조하세요. `SecretString` 파라미터를 사용합니다.

ssl.key.password

keystore 파일에 있는 프라이빗 키의 암호입니다.

이 필드는 대체 템플릿을 지원합니다. Secrets Manager를 사용하여 Kafka 브로커에 연결하는 데 필요한 보안 인증 정보를 저장합니다. `get_secret` SQL 함수를 사용하여 이 필드의 값을 검색합니다. 대체 변수에 대한 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요. `get_secret` SQL 함수에 대한 자세한 내용은 the section called “get_secret(secretId, secretType, key, roleArn)” 섹션을 참조하세요. `SecretString` 파라미터를 사용합니다.

sasl.mechanism

Kafka 브로커에 연결하는 데 사용되는 보안 메커니즘입니다. `security.protocol`에 `SASL_SSL`을 지정할 때 이 값이 필요합니다.

유효한 값: `PLAIN`, `SCRAM-SHA-512`, `GSSAPI`.

> **Note**
>
> `SCRAM-SHA-512`는 cn-north-1, cn-northwest-1, us-gov-east-1, and us-gov-west-1 리전에서 유일하게 지원되는 보안 메커니즘입니다.

sasl.plain.username

Secrets Manager에서 보안 문자열을 검색하는 데 사용되는 사용자 이름입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `PLAIN`을 지정할 때 이 값이 필요합니다.

sasl.plain.password

Secrets Manager에서 보안 문자열을 검색하는 데 사용되는 암호입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `PLAIN`을 지정할 때 이 값이 필요합니다.

sasl.scram.username

Secrets Manager에서 보안 문자열을 검색하는 데 사용되는 사용자 이름입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `SCRAM-SHA-512`을 지정할 때 이 값이 필요합니다.

sasl.scram.password

Secrets Manager에서 보안 문자열을 검색하는 데 사용되는 암호입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `SCRAM-SHA-512`을 지정할 때 이 값이 필요합니다.

sasl.kerberos.keytab

Secrets Manager의 Kerberos 인증을 위한 keytab 파일입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `GSSAPI`을 지정할 때 이 값이 필요합니다.

이 필드는 대체 템플릿을 지원합니다. Secrets Manager를 사용하여 Kafka 브로커에 연결하는 데 필요한 보안 인증 정보를 저장합니다. `get_secret` SQL 함수를 사용하여 이 필드의 값을 검색합니다. 대체 변수에 대한 자세한 내용은 the section called “대체 템플릿” 단원을 참조하세요. `get_secret` SQL 함수에 대한 자세한 내용은 the section called “get_secret(secretId, secretType, key, roleArn)” 섹션을 참조하세요. `SecretBinary` 파라미터를 사용합니다.

sasl.kerberos.service.name

Apache Kafka가 실행되는 Kerberos 보안 주체 이름입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `GSSAPI`을 지정할 때 이 값이 필요합니다.

sasl.kerberos.krb5.kdc

Apache Kafka 프로듀서 클라이언트가 연결하는 키 배포 센터(KDC)의 호스트 이름입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `GSSAPI`을 지정할 때 이 값이 필요합니다.

sasl.kerberos.krb5.realm

Apache Kafka 프로듀서 클라이언트가 연결되는 영역입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `GSSAPI`을 지정할 때 이 값이 필요합니다.

sasl.kerberos.principal

Kerberos가 Kerberos 인식 서비스에 액세스하기 위해 티켓을 할당할 수 있는 고유한 Kerberos ID입니다. `security.protocol`에 `SASL_SSL`, `sasl.mechanism`에 `GSSAPI`을 지정할 때 이 값이 필요합니다.

예제

다음 JSON 예제는 AWS IoT 규칙에서 Apache Kafka 작업을 정의합니다. 다음 예시에서는 sourceIp() 인라인 함수를 Kafka 작업 헤더의 대체 템플릿으로 전달합니다.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "kafka": {
     "destinationArn": "arn:aws:iot:region:123456789012:ruledestination/vpc/
VPCDestinationARN",
     "topic": "TopicName",
     "clientProperties": {
      "bootstrap.servers": "kafka.com:9092",
      "security.protocol": "SASL_SSL",
      "ssl.truststore": "${get_secret('kafka_client_truststore',
 'SecretBinary','arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "ssl.truststore.password": "kafka password",
      "sasl.mechanism": "GSSAPI",
      "sasl.kerberos.service.name": "kafka",
      "sasl.kerberos.krb5.kdc": "kerberosdns.com",
      "sasl.kerberos.keytab": "${get_secret('kafka_keytab','SecretBinary',
 'arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "sasl.kerberos.krb5.realm": "KERBEROSREALM",
      "sasl.kerberos.principal": "kafka-keytab/kafka-keytab.com"
     },
     "headers": [
      {
       "key": "static_header_key",
       "value": "static_header_value"
      },
      {
       "key": "substitutable_header_key",
       "value": "${value_from_payload}"
      },
      {
       "key": "source_ip",
       "value": "${sourceIp()}"
      }
```

```
      ]
     }
    }
   ]
  }
 }
```

Kerberos 설정에 대한 중요 참고 사항

- 대상 VPC 내의 프라이빗 도메인 이름 시스템(DNS)을 통해 키 배포 센터(KDC) 를 확인할 수 있어야 합니다. 한 가지 가능한 방법은 KDC DNS 항목을 프라이빗 호스팅 영역에 추가하는 것입니다. 이 접근법에 대한 자세한 내용은 프라이빗 호스팅 영역 작업을 참조하세요.
- 각 VPC에는 DNS 확인이 활성화되어 있어야 합니다. 자세한 내용은 VPC에서 DNS 사용하기 단원을 참조하세요.
- VPC 대상의 네트워크 인터페이스 보안 그룹과 인스턴스 수준 보안 그룹은 다음 포트에서 VPC 내의 트래픽을 허용해야 합니다.
  - 부트스트랩 브로커 리스너 포트의 TCP 트래픽(대개 9092이지만 9000~9100 범위 내에 있어야 함)
  - 포트 88에서 KDC에 대한 TCP 및 UDP 트래픽
- `SCRAM-SHA-512`는 cn-north-1, cn-northwest-1, us-gov-east-1, and us-gov-west-1 리전에서 유일하게 지원되는 보안 메커니즘입니다.

Apache Kafka Virtual Private Cloud(VPC) 대상

Apache Kafka 규칙 작업은 Amazon Virtual Private Cloud(Amazon VPC)의 Apache Kafka 클러스터로 데이터를 라우팅합니다. Apache Kafka 규칙 작업에서 사용하는 VPC 구성은 규칙 작업의 VPC 대상을 지정할 때 자동으로 사용됩니다.

Apache Kafka Virtual Private Cloud(VPC) 대상에는 VPC 내의 서브넷 목록이 포함됩니다. 규칙 엔진은 이 목록에서 지정한 각 서브넷에 탄력적 네트워크 인터페이스를 생성합니다. 네트워크 인터페이스에 대한 자세한 내용은 Amazon EC2 사용 설명서의 탄력적 네트워크 인터페이스를 참조하세요.

요구 사항 및 고려 사항

- 인터넷을 통해 퍼블릭 엔드포인트를 사용하여 액세스될 자체 관리형 Apache Kafka 클러스터를 사용하는 경우:
  - 서브넷의 인스턴스에 대해 NAT 게이트웨이를 생성합니다. NAT 게이트웨이에는 인터넷에 연결할 수 있는 퍼블릭 IP 주소가 있으므로 규칙 엔진이 메시지를 퍼블릭 Kafka 클러스터로 전달할 수 있습니다.

- Apache Kafka Virtual Private Cloud(VPC) 대상에서 생성한 탄력적 네트워크 인터페이스(ENIs)를 사용하여 탄력적 IP 주소를 할당합니다. 사용하는 보안 그룹은 수신 트래픽을 차단하도록 구성해야 합니다.

  > **Note**
  >
  > Apache Kafka Virtual Private Cloud(VPC) 대상이 비활성화되었다가 다시 활성화된 경우 탄력적 IPs를 새 ENIs와 다시 연결해야 합니다.

- Apache Kafka Virtual Private Cloud(VPC) 대상이 30일 연속으로 트래픽을 수신하지 않으면 비활성화됩니다.
- Apache Kafka Virtual Private Cloud(VPC) 대상에서 사용하는 리소스가 변경되면 대상이 비활성화되고 사용할 수 없습니다.
- Apache Kafka Virtual Private Cloud(VPC) 대상을 비활성화할 수 있는 몇 가지 변경 사항은 다음과 같습니다.
  - VPC, 서브넷, 보안 그룹 또는 사용된 역할 삭제.
  - 더 이상 필요한 권한이 없도록 역할을 수정합니다.
  - 거의 서브넷 용량에 도달하여 FedRAMP 패치를 적용할 수 없습니다.
  - 대상을 비활성화합니다.

가격 책정

가격 책정을 위해 리소스가 VPC에 있을 때 리소스에 메시지를 전송하는 작업 외에도 VPC 규칙 작업이 측정됩니다. 요금 정보는 AWS IoT Core 요금을 참조하세요.

Apache Kafka Virtual Private Cloud(VPC) 대상 생성

CreateTopicRuleDestination API 또는 AWS IoT Core 콘솔을 사용하여 Apache Kafka Virtual Private Cloud(VPC) 대상을 생성합니다.

대상을 생성할 때 다음 정보를 지정해야 합니다.

vpcId

Amazon VPC의 고유 ID입니다.

subnetIds

규칙 엔진이 탄력적 네트워크 인터페이스를 만드는 서브넷 목록입니다. 규칙 엔진은 목록의 각 서브넷에 대해 단일 네트워크 인터페이스를 할당합니다.

securityGroups(선택 사항)

네트워크 인터페이스에 적용할 보안 그룹 목록입니다.

roleArn

사용자를 대신하여 네트워크 인터페이스를 생성할 권한이 있는 역할의 Amazon 리소스 이름(ARN)입니다.

이 ARN에는 다음 예와 같은 정책이 연결되어 있어야 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/VPCDestinationENI": "true"
            }
        }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "CreateNetworkInterface",
                    "aws:RequestTag/VPCDestinationENI": "true"
                }
        }
        }
    ]
}
```

를 사용하여 Apache Kafka Virtual Private Cloud(VPC) 대상 생성 AWS CLI

다음 예제에서는를 사용하여 대상을 생성하는 방법을 보여줍니다 AWS CLI.

```
aws --region regions iot create-topic-rule-destination --destination-configuration
 'vpcConfiguration={subnetIds=["subnet-
123456789101230456"],securityGroups=[],vpcId="vpc-
123456789101230456",roleArn="arn:aws:iam::123456789012:role/role-name"}'
```

이 명령을 실행하면 대상 상태는가 됩니다IN_PROGRESS. 몇 분 후 상태가 ERROR(명령이 성공하지 못한 경우) 또는 ENABLED로 변경됩니다. 대상 상태가 ENABLED이면 사용할 준비가 된 것입니다.

다음 명령을 사용하여 Apache Kafka Virtual Private Cloud(VPC) 대상의 상태를 가져올 수 있습니다.

```
aws --region region iot get-topic-rule-destination --arn "VPCDestinationARN"
```

AWS IoT Core 콘솔을 사용하여 Apache Kafka Virtual Private Cloud(VPC) 대상 생성

다음 단계에서는 AWS IoT Core 콘솔을 사용하여 대상을 생성하는 방법을 설명합니다.

1. AWS IoT Core 콘솔로 이동합니다. 왼쪽 창의 동작 탭에서 대상을 선택합니다.

2. 다음 필드에 값을 입력합니다.

   - VPC ID
   - 서브넷 ID
   - 보안 그룹
3. 네트워크 인터페이스를 생성하는 데 필요한 권한이 있는 역할을 선택합니다. 앞의 예제 정책에는 이러한 사용 권한이 포함되어 있습니다.

Apache Kafka Virtual Private Cloud(VPC) 대상 상태가 ENABLED이면 바로 사용할 수 있습니다.

CloudWatch 경보

CloudWatch 경보(`cloudWatchAlarm`) 작업은 Amazon CloudWatch 경보의 상태를 변경합니다. 이 호출에서 상태 변경 이유 및 값을 지정할 수 있습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `cloudwatch:SetAlarmState` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.

파라미터

이 작업을 사용하여 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`alarmName`

CloudWatch 경보 이름입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`stateReason`

경보 변경 이유입니다.

대체 템플릿 지원: 예

`stateValue`

경보 상태의 값입니다. 유효한 값: `OK`, `ALARM`, `INSUFFICIENT_DATA`.

대체 템플릿 지원: 예

`roleArn`

CloudWatch 경보에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 CloudWatch 경보 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchAlarm": {
                    "alarmName": "IotAlarm",
                    "stateReason": "Temperature stabilized.",
                    "stateValue": "OK",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?
- Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 경보 사용

CloudWatch Logs

CloudWatch Logs(`cloudwatchLogs`) 작업은 데이터를 Amazon CloudWatch Logs에 전송합니다. `batchMode`를 사용하여 하나의 메시지에 여러 디바이스 로그 레코드를 업로드하고 타임스탬프를 지정할 수 있습니다. 작업에서 데이터를 전송할 로그 그룹을 지정할 수도 있습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- 가 `logs:CreateLogStream`, `logs:DescribeLogStreams`및 `logs:PutLogEvents` 작업을 수행하기 위해 수임할 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.
- 고객 관리형 AWS KMS key (KMS 키)를 사용하여 CloudWatch Logs에서 로그 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 AWS KMS를 사용하여 CloudWatch Logs에서 로그 데이터 암호화를 참조하세요.

batchMode에 대한 MQTT 메시지 형식 요구 사항

`batchMode`를 끄고 CloudWatch Logs 규칙 작업을 사용하는 경우 MQTT 메시지 형식 지정 요구 사항이 없습니다. (참고: `batchMode` 파라미터의 기본값은 `false`입니다.) 그러나 `batchMode`가 켜진 상태에서 CloudWatch Logs 규칙 작업을 사용하는 경우(파라미터 값은 `true`), 디바이스 측 로그를 포함하는 MQTT 메시지는 타임스탬프와 메시지 페이로드를 포함하도록 형식을 지정해야 합니다. 참고: `timestamp`는 이벤트가 발생한 시간을 나타내며, 1970년 1월 1일 00:00:00 UTC 이후 경과된 밀리초 수로 표시됩니다.

다음은 게시 형식의 예입니다.

```
[
  {"timestamp": 1673520691093, "message": "Test message 1"},
  {"timestamp": 1673520692879, "message": "Test message 2"},
  {"timestamp": 1673520693442, "message": "Test message 3"}
]
```

이 요구 사항을 준수하려면 기기 측 로그가 생성되는 방식에 따라 로그를 전송하기 전에 해당 로그를 필터링하고 형식을 다시 지정해야 할 수 있습니다. 자세한 내용은 MQTT 메시지 페이로드를 참조하세요.

batchMode 파라미터와 관계없이 message 콘텐츠는 AWS IoT 메시지 크기 제한을 준수해야 합니다. 자세한 내용은 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

파라미터

이 작업을 사용하여 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

logGroupName

작업에서 데이터를 전송할 CloudWatch 로그 그룹입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

roleArn

CloudWatch 로그 그룹에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니오

batchMode (선택 사항)

로그 레코드 배치를 추출하여 CloudWatch에 업로드할지 여부를 나타냅니다. 값에는 true 또는 false(기본값)가 포함됩니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 CloudWatch Logs 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchLogs": {
```

```
                    "logGroupName": "IotLogs",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw",
                    "batchMode": false
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon CloudWatch Logs 사용 설명서의 Amazon CloudWatch Logs란 무엇입니까?

CloudWatch 지표

CloudWatch 지표(`cloudwatchMetric`) 작업은 Amazon CloudWatch 지표를 포착합니다. 지표 네임스페이스, 이름, 값 및 타임스탬프를 지정할 수 있습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `cloudwatch:PutMetricData` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`metricName`

CloudWatch 지표 이름입니다.

대체 템플릿 지원: 예

`metricNamespace`

CloudWatch 지표 네임스페이스 이름입니다.

대체 템플릿 지원: 예

`metricUnit`

CloudWatch에서 지원되는 지표 단위입니다.

대체 템플릿 지원: 예

`metricValue`

CloudWatch 지표 값을 포함하는 문자열입니다.

대체 템플릿 지원: 예

`metricTimestamp`

(선택 사항) Unix Epoch 시간으로 표시되는(초 단위) 타임스탬프가 포함된 문자열입니다. 기본값은 현재 Unix Epoch 시간입니다.

대체 템플릿 지원: 예

`roleArn`

CloudWatch 지표에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 CloudWatch 지표 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "IotMetric",
                    "metricNamespace": "IotNamespace",
                    "metricUnit": "Count",
                    "metricValue": "1",
                    "metricTimestamp": "1456821314",
```

```
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 CloudWatch 지표 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "${topic()}",
                    "metricNamespace": "${namespace}",
                    "metricUnit": "${unit}",
                    "metricValue": "${value}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon CloudWatch 사용 설명서의 Amazon CloudWatch란 무엇입니까?
- Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 지표 사용

DynamoDB

DynamoDB(dynamoDB) 작업은 MQTT 메시지의 전체 또는 일부를 Amazon DynamoDB 테이블에 씁니다.

DynamoDB 작업을 사용하여 규칙을 생성하고 테스트하는 방법을 자습서에서 확인할 수 있습니다. 자세한 내용은 자습서: DynamoDB 테이블에 디바이스 데이터 저장 섹션을 참조하세요.

> **Note**
>
> 이 규칙은 비 JSON 데이터를 이진 데이터로 DynamoDB에 씁니다. DynamoDB 콘솔은 데이터를 Base64 인코딩된 텍스트로 표시합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `dynamodb:PutItem` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.
- 고객 관리형 AWS KMS key (KMS 키)를 사용하여 DynamoDB에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon DynamoDB 시작 안내서의 고객 관리형 KMS 키를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`tableName`

DynamoDB 테이블의 이름입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`hashKeyField`

해시 키(파티션 키라고도 함)의 이름입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`hashKeyType`

(선택 사항) 해시 키(파티션 키라고도 함)의 데이터 형식입니다. 유효한 값: `STRING`, `NUMBER`.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`hashKeyValue`

해시 키의 값입니다. `${topic()}` 또는 `${timestamp()}` 같은 대체 템플릿 사용을 고려하세요.

대체 템플릿 지원: 예

`rangeKeyField`

(선택 사항) 범위 키(정렬 키라고도 함)의 이름입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`rangeKeyType`

(선택 사항) 범위 키(정렬 키라고도 함)의 데이터 형식입니다. 유효한 값: `STRING`, `NUMBER`.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`rangeKeyValue`

(선택 사항) 범위 키의 값입니다. `${topic()}` 또는 `${timestamp()}` 같은 대체 템플릿 사용을 고려하세요.

대체 템플릿 지원: 예

`payloadField`

(선택 사항) 페이로드가 기록될 열 이름입니다. 이 값을 생략할 경우 페이로드가 `payload`라는 열에 기록됩니다.

대체 템플릿 지원: 예

`operation`

(선택 사항) 수행할 작업의 유형입니다. 유효한 값: `INSERT`, `UPDATE`, `DELETE`.

대체 템플릿 지원: 예

`roleARN`

DynamoDB 테이블에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니요

DynamoDB 테이블에 기록된 데이터는 규칙 내 SQL 문의 결과입니다.

예제

다음 JSON 예제는 AWS IoT 규칙에서 DynamoDB 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDB": {
                    "tableName": "my_ddb_table",
                    "hashKeyField": "key",
                    "hashKeyValue": "${topic()}",
                    "rangeKeyField": "timestamp",
                    "rangeKeyValue": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon DynamoDB 개발자 안내서의 Amazon DynamoDB란 무엇입니까?
- Amazon DynamoDB 개발자 안내서의 DynamoDB 시작하기
- 자습서: DynamoDB 테이블에 디바이스 데이터 저장

DynamoDBv2

DynamoDBv2(dynamoDBv2) 작업은 MQTT 메시지의 전체 또는 일부를 Amazon DynamoDB 테이블에 전송합니다. 페이로드의 각 속성은 DynamoDB 데이터베이스의 개별 열에 기록됩니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `dynamodb:PutItem` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.
- MQTT 메시지 페이로드는 테이블의 기본 파티션 키와 일치하는 루트 수준 키, 그리고 정의된 경우 테이블의 기본 정렬 키와 일치하는 루트 수준 키를 포함해야 합니다.
- 고객 관리형 AWS KMS key (KMS 키)를 사용하여 DynamoDB에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon DynamoDB 시작 안내서의 고객 관리형 KMS 키를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`putItem`

메시지 데이터를 작성할 DynamoDB 테이블을 지정하는 객체입니다. 이 객체에는 다음 정보가 포함되어 있어야 합니다.

`tableName`

DynamoDB 테이블의 이름입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`roleARN`

DynamoDB 테이블에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니요

DynamoDB 테이블에 기록된 데이터는 규칙 내 SQL 문의 결과입니다.

예제

다음 JSON 예제는 AWS IoT 규칙에서 DynamoDBv2 작업을 정의합니다.

```
{
```

```
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "my_ddb_table"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2",
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 DynamoDB 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2015-10-08",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "${topic()}"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon DynamoDB 개발자 안내서의 Amazon DynamoDB란 무엇입니까?
- Amazon DynamoDB 개발자 안내서의 DynamoDB 시작하기

Elasticsearch

Elasticsearch(`elasticsearch`) 작업을 사용하여 MQTT 메시지의 데이터를 Amazon OpenSearch Service 도메인에 전송합니다. 그런 다음 OpenSearch 대시보드와 같은 도구를 사용하여 OpenSearch Service에서 데이터를 쿼리하고 시각화할 수 있습니다.

> **Warning**
>
> `Elasticsearch` 작업은 기존 규칙 작업에서만 사용할 수 있습니다. 새 규칙 작업을 생성하거나 기존 규칙 작업을 업데이트하려면 대신 `OpenSearch` 규칙 작업을 사용합니다. 자세한 내용은 OpenSearch 단원을 참조하십시오.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `es:ESHttpPut` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.
- 고객 관리형 AWS KMS key (KMS 키)를 사용하여 OpenSearch에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon OpenSearch Service 개발자 안내서의 Amazon OpenSearch Service의 저장 시 데이터 암호화를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`endpoint`

서비스 도메인의 엔드포인트입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`index`

데이터를 저장할 인덱스입니다.

대체 템플릿 지원: 예

`type`

저장 중인 문서의 유형입니다.

대체 템플릿 지원: 예

`id`

각 문서의 고유 식별자입니다.

대체 템플릿 지원: 예

`roleARN`

OpenSearch 도메인에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 Elasticsearch 작업을 정의하고 `elasticsearch` 작업에 대한 필드를 지정하는 방법을 정의합니다. 자세한 내용은 ElasticsearchAction 섹션을 참조하세요.

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "my-type",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
```

```
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 Elasticsearch 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- OpenSearch
- Amazon OpenSearch Service란 무엇입니까?

HTTP

HTTPS(http) 작업은 MQTT 메시지의 데이터를 HTTPS 엔드포인트로 전송합니다. HTTPS 엔드포인트는 웹 애플리케이션 또는 서비스를 가리킬 수 있습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- 규칙 엔진에서 사용하려면 먼저 HTTPS 엔드포인트를 확인하고 활성화해야 합니다. 자세한 내용은 HTTP 작업 대상 단원을 참조하십시오.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`url`

메시지가 HTTP POST 메서드를 사용하여 전송되는 HTTPS 엔드포인트입니다. 호스트 이름 대신 IP 주소를 사용하는 경우 IPv4 주소여야 합니다. IPv6 주소는 지원하지 않습니다.

대체 템플릿 지원: 예

`confirmationUrl`

(선택 사항) 지정된 경우 확인 URL을 AWS IoT 사용하여 일치하는 주제 규칙 대상을 생성합니다. HTTP 작업에서 사용하기 전에 HTTP 작업 대상을 활성화해야 합니다. 자세한 내용은 HTTP 작업 대상 단원을 참조하십시오. 대체 템플릿을 사용하는 경우 작업을 `http` 사용하려면 먼저 HTTP 작업 대상을 수동으로 생성해야 합니다.는 접두사여야 `confirmationUrl` 합니다`url`.

`url`과 `confirmationUrl` 사이의 관계는 다음으로 설명됩니다.

- `url`가 하드코딩되고 `confirmationUrl`가 제공되지 않은 경우 `url` 필드를 로 암시적으로 취급합니다`confirmationUrl`.는에 대한 주제 규칙 대상을 AWS IoT 생성합니다`url`.
- `url` 및 `confirmationUrl`가 하드코딩된 경우는 로 시작해야 `url` 합니다`confirmationUrl`.는에 대한 주제 규칙 대상을 AWS IoT 생성합니다`confirmationUrl`.
- `url`에 대체 템플릿이 포함되어 있는 경우 `confirmationUrl`을 지정하고 `url`은 `confirmationUrl`로 시작해야 합니다. 에 대체 템플릿이 `confirmationUrl` 포함된 경우 HTTP 작업 대상을 수동으로 생성해야 `http` 작업을 사용할 수 있습니다. `confirmationUrl`에 대체 템플릿이 포함되어 있지 않은 경우는에 대한 주제 규칙 대상을 AWS IoT 생성합니다`confirmationUrl`.

대체 템플릿 지원: 예

`headers`

(선택 사항) 엔드포인트에 대한 HTTP 요청에 포함할 헤더 목록입니다. 각 헤더에는 다음 정보가 포함되어야 합니다.

`key`

헤더의 키입니다.

대체 템플릿 지원: 아니요

`value`

헤더의 값

대체 템플릿 지원: 예

> **Note**
>
> 페이로드가 JSON 형식일 때 기본 콘텐츠 유형은 application/json입니다. 그렇지 않으면 application/octet-stream입니다. 키 콘텐츠 유형(대/소문자 구분 없음)으로 헤더에서 정확한 콘텐츠 유형을 지정하여 덮어쓸 수 있습니다.

`auth`

(선택 사항) `url` 인수에 지정된 엔드포인트 URL에 연결하기 위해 규칙 엔진에서 사용하는 인증입니다. 현재 서명 버전 4는 유일하게 지원되는 인증 유형입니다. 자세한 내용은 HTTP 권한 부여를 참조하세요.

대체 템플릿 지원: 아니요

`enableBatching`

(선택 사항) HTTP 작업 메시지를 지정된 URL에 대한 단일 요청으로 처리할지 여부입니다. 값은 true 또는 false일 수 있습니다. 일괄 처리에 대한 자세한 내용은 HTTP 작업 메시지 일괄 처리를 참조하세요.

부울 값

대체 템플릿 지원: 아니오

`batchConfig`

(선택 사항) 일괄 처리를 위한 구성 설정입니다. 활성화되면 `batchConfig` 파라미터를 지정해야 합니다. `batchConfig` 파라미터를 지정하지 않으면 기본값이 사용됩니다.

`maxBatchOpenMs`

발신 메시지가 다른 메시지가 배치를 생성할 때까지 대기하는 최대 시간(밀리초)입니다. 설정이 높을수록 일괄 HTTP 작업의 지연 시간이 길어집니다.

최소값: 5ms. 최댓값: 200ms.

기본값: 20ms

대체 템플릿 지원: 아니오

`maxBatchSize`

단일 작업 실행에서 함께 일괄 처리되는 최대 메시지 수입니다.

대체 템플릿 지원: 아니오

최솟값: 메시지 2개. 최대값: 메시지 10개

기본값: 메시지 10개

`maxBatchSizeBytes`

메시지 배치의 최대 크기, 바이트 단위.

최소값: 100바이트. 최대값: 131,072바이트

기본값: 5,120바이트

대체 템플릿 지원: 아니오

> **Note**
>
> 페이로드가 JSON 형식일 때 기본 콘텐츠 유형은 application/json입니다. 그렇지 않으면 application/octet-stream입니다. 키 콘텐츠 유형(대/소문자 구분 없음)으로 헤더에서 정확한 콘텐츠 유형을 지정하여 덮어쓸 수 있습니다.

예제

다음 JSON 예제에서는 HTTP 작업을 사용하여 AWS IoT 규칙을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
```

```
                            "key": "static_header_key",
                            "value": "static_header_value"
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                        }
                    ]
                }
            }
        ]
    }
}
```

```
"http": {
    "url": "https://www.example.com/subpath",
    "confirmationUrl": "https://www.example.com",
    "headers": [
        {
            "key": "Content-Type",
            "value": "application/json"
          }
    ],
    "enableBatching": true,
    "batchConfig": {
      "maxBatchOpenMs": 123,
      "maxBatchSize": 5,
      "maxBatchSizeBytes": 131072,
     }
 },
 "errorAction": {
        "http": {
            "url": "https://www.example.com/subpath",
            "confirmationUrl": "https://www.example.com"
            // batchConfig is not allowed here
        }
}
```

HTTP 작업 재시도 논리

AWS IoT 규칙 엔진은 다음 규칙에 따라 HTTP 작업을 재시도합니다.

- 규칙 엔진은 메시지를 한 번 이상 전송하려고 시도합니다.

- 규칙 엔진은 최대 두 번 재시도합니다. 최대 시도 횟수는 3회입니다.
- 규칙 엔진은 다음과 같은 경우 재시도를 수행하지 않습니다.
  - 이전 시도에서 16,384바이트보다 큰 응답을 제공했습니다.
  - 시도 후 다운스트림 웹 서비스 또는 애플리케이션에서 TCP 연결을 닫습니다.
  - 재시도로 요청을 완료하는 데 걸리는 총 시간이 요청 제한 시간 한도를 초과했습니다.
  - 이 요청이 429, 500-599 이외의 HTTP 상태 코드를 반환합니다.

> **Note**
>
> 표준 데이터 전송 비용이 재시도에 적용됩니다.

다음 사항도 참조하세요.

- HTTP 작업 메시지 배치 처리
- HTTP 작업 대상
- 블로그의 사물 인터넷 AWS에서 에서 AWS IoT Core 웹 서비스로 직접 데이터 라우팅

HTTP 작업 메시지 배치 처리

일괄 처리를 사용하여 단일 요청으로 여러 HTTP 작업 메시지를 보낼 수 있습니다.

개요

일괄 처리를 사용하면 AWS IoT Core 규칙 엔진에서 HTTP 엔드포인트로 메시지를 일괄 전송할 수 있습니다. 이 기능을 사용하면 HTTP 작업 실행 수를 줄이고 새 연결 설정과 관련된 오버헤드를 줄여 효율성을 개선할 수 있습니다.

> **Note**
>
> 일괄 HTTP 작업은 단일 작업으로 측정됩니다. AWS IoT Core 규칙 엔진에서 다운스트림 서비스로 내보내는 아웃바운드 배치 페이로드의 크기를 기준으로 5kiB 단위로 측정됩니다. 자세한 내용은 AWS IoT Core 요금 페이지를 참조하세요.

IoT 규칙 작업의 정의에서 일괄 처리를 활성화하면 구성에 다음 파라미터를 사용할 수 있습니다.

`maxBatchOpenMs`

발신 메시지가 다른 메시지가 배치를 생성할 때까지 대기하는 최대 시간(밀리초)입니다. 설정이 높을수록 일괄 HTTP 작업의 지연 시간이 길어집니다.

최소값: 5ms. 최댓값: 200ms.

기본값: 20ms

대체 템플릿 지원: 아니오

`maxBatchSize`

단일 IoT 규칙 작업 실행에서 함께 일괄 처리되는 최대 메시지 수입니다.

최솟값: 메시지 2개. 최대값: 메시지 10개

기본값: 메시지 10개

대체 템플릿 지원: 아니오

`maxBatchSizeBytes`

메시지 배치의 최대 크기, 바이트 단위.

최소값: 100바이트. 최대값: 131,072바이트

기본값: 5120바이트

대체 템플릿 지원: 아니오

> **Important**
>
> 여러 배치 파라미터를 지정하면 첫 번째 한도에 도달하면 일괄 처리가 완료됩니다. 예를 들어 최대 배치 열기 시간으로 100ms를 지정하고 최대 배치 크기로 5kiB를 지정하고 규칙 엔진이 100ms 이내에 2kiB만 배치하는 경우 2kiB 배치가 생성되고 전송됩니다.

배치에서 HTTP 헤더 사용

HTTP 작업에서 헤더를 사용하는 경우 일괄 처리 요청은 배치에 추가된 마지막 메시지의 헤더 값(반드시 게시한 마지막 메시지가 아님)을 사용합니다. 다음 중 하나인 헤더 값을 사용하는 것이 좋습니다.

- 배치의 모든 메시지에서 동일

- 모든 메시지에 적용 가능(예: 인증 자격 증명)

헤더는 HTTP 요청과 함께 전송되며 메시지 본문의 일부가 아닙니다.

> **Note**
>
> 일괄 처리가 활성화된 경우:
>
> - 배치 요청은 배치가 JSON 배열로 전송되므로 `Content-Type: application/json` 헤더를 자동으로 포함합니다.
> - 배치의 마지막 메시지가 게시한 마지막 메시지임을 보장할 수는 없습니다. 배치로 만든 마지막 메시지입니다.

페이로드 예제

다음 예제는 HTTP 엔드포인트로 전송된 일괄 메시지 페이로드의 구조를 보여줍니다.

```
[
  {
    "user_id": "user1",
    "steps_today": 1000
  },
  {
    "user_id": "user2",
    "steps_today": 21000
  },
  {
    "user_id": "user8",
    "steps_today": 1500
  },
  ...
]
```

제한 사항

다음은 일괄 처리의 제한 사항입니다.

- AWS IoT Core 는 전체 메시지 순서를 보장하지 않습니다. 배치는 각 호스트에서 로컬로 수행되므로 배치 내의 메시지가 수신된 순서와 다른 순서로 처리될 수 있습니다.

- AWS IoT Core 는 수신자 측에서 메시지 처리 지원을 제공하지 않습니다. 다운스트림 서비스가 데이터를 일괄적으로 수락하고 처리하도록 구성되어 있는지 확인하는 것은 사용자의 책임입니다.
- 메시지가 동일한 리소스 식별자(HTTP URL 또는 리소스 ARN)로 향하는 경우에도 교차 계정 일괄 처리는 지원되지 않습니다.
- AWS IoT Core 는 배치 크기가 지정한 구성을 충족할 것이라고 보장하지 않습니다. 배치는 타이밍 및 메시지 흐름에 따라 구성된 제한보다 작을 수 있습니다.
- 일괄 처리가 활성화된 경우 이진 페이로드(UTF-8이 아닌 데이터)는 지원되지 않습니다. UTF-8 텍스트 페이로드(예: JSON)만 허용됩니다. 바이너리 데이터를 전송하려면 HTTP 작업으로 전송하기 전에 base64로 인코딩한 다음 수신 엔드포인트에서 디코딩합니다. 예를 들어 IoT 규칙의 인코딩 함수를 사용하여 바이너리 페이로드를 인코딩할 수 있습니다. 또는 IoT 디바이스에서 바이너리 페이로드를 인코딩하여에 게시할 수 있습니다 AWS IoT Core.

일괄 처리를 위한 오류 작업

오류 작업 정의에서 별도의 일괄 처리 로직을 정의할 수 없습니다. 그러나 기본 작업에서 일괄 처리 로직을 정의한 경우 오류 작업은 일괄 처리를 지원합니다.

배치 요청이 실패하면 AWS IoT Core 규칙 엔진은 HTTP 작업 재시도 로직을 따릅니다. 마지막 재시도 후 실패한 전체 배치에 대해 오류 작업이 호출됩니다.

다음은 일괄 처리가 활성화된 오류 작업 메시지의 예입니다.

```
{
    "ruleName": "FailedTopicRule",
    "topic": "topic/rulesengine",
    "payloadsWithMetadata": [
        {
            "id": 1,
            "cloudwatchTraceId": "bebd6d93-6d4a-899e-9e40-56e82252d2be",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wC
        },
        {
            "id": 2,
            "cloudwatchTraceId": "af94d3b8-0b18-1dbf-2c7d-513f5cb9e2e1",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
```

```
            "base64OriginalPayload":
"eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        },
        {
            "id": 3,
            "cloudwatchTraceId": "ca441266-c2ce-c916-6aee-b9e5c7831675",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
"eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        }
    ],
    "failures": [
        {
            "affectedIds": [
                1,
                2,
                3
            ],
            "failedAction": "HttpAction",
            "failedResource": "https://example.foobar.com/HttpAction",
            "errorMessage": "HttpAction failed to make a request to the specified
endpoint. StatusCode: 500. Reason: Internal Server Error."
        },
        {
            "affectedIds": [
                3
            ],
            "failedAction": "S3Action",
            "failedResource": "amzn-s3-demo-bucket",
            "errorMessage": "Failed to put S3 object. The error received was The
specified bucket does not exist"
        },
        {
            "affectedIds": [
                3
            ],
            "failedAction": "LambdaAction",
            "failedResource": "arn:aws:lambda:us-west-2:123456789012:function:dummy",
            "errorMessage": "Failed to invoke lambda function. Received Server error
from Lambda. The error code is 403"
        }
    ]
```

```
}
```

> **Note**
>
> 또한 배치 작업 실패는 더 큰 오류 작업 페이로드를 생성하여 크기로 인한 오류 작업 실패 가능성을 높일 수 있습니다. `ErrorActionFailure` 지표를 사용하여 오류 작업 실패를 모니터링할 수 있습니다. 자세한 정보는 규칙 작업 지표을 참조하세요.

를 사용하여 HTTP 작업 메시지 일괄 처리 AWS CLI

일괄 처리를 사용하여 규칙 작업 생성 또는 업데이트

1. 적절한 AWS CLI 명령을 사용하여 규칙을 생성하거나 업데이트합니다.
   - 새 규칙을 생성하려면 create-topic-rule 명령을 사용합니다.

     ```
     aws iot create-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

   - 기존 규칙을 업데이트하려면 replace-topic-rule 명령을 사용합니다.

     ```
     aws iot replace-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

2. 주제 규칙 페이로드에서 enableBatching 파라미터를 true로 설정하여 일괄 처리 기능을 활성화합니다.

```
{
        "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
                            "key": "static_header_key",
                            "value": "static_header_value"
```

```
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                         }
                    ],
                    "enableBatching": true,
                    "batchConfig": {
                       "maxBatchOpenMs": 100,
                       "maxBatchSize": 5,
                       "maxBatchSizeBytes": 1024
                    }
                }
            }
        ]
}
```

3. 일괄 처리 파라미터를 구성합니다. 모든 배치 파라미터를 지정할 필요는 없습니다. 1, 2 또는 3개의 배치 파라미터를 모두 지정하도록 선택할 수 있습니다. 배치 파라미터를 지정하지 않으면 규칙 엔진은 해당 파라미터를 기본값으로 업데이트합니다. 파라미터 일괄 처리 및 기본값에 대한 자세한 내용은 HTTP 파라미터를 참조하세요.

HTTP 작업 대상

HTTP 작업 대상은 규칙 엔진이 주제 규칙에서 데이터를 라우팅할 수 있는 웹 서비스입니다. AWS IoT Core 리소스는에 대한 웹 서비스를 설명합니다 AWS IoT. 대상 리소스는 다른 규칙에서 공유할 수 있습니다.

AWS IoT Core 가 다른 웹 서비스로 데이터를 전송하려면 먼저 서비스의 엔드포인트에 액세스할 수 있는지 확인해야 합니다.

개요

HTTP 작업 대상은 확인 URL과 하나 이상의 데이터 수집 URLs을 지원하는 웹 서비스를 말합니다. 대상 리소스에는 웹 서비스의 확인 URL이 포함되어 있습니다. HTTP 작업을 구성할 때 웹 서비스의 확인 URL과 함께 데이터를 수신해야 하는 엔드포인트의 실제 URL을 지정합니다. 대상이 확인되면 주제 규칙은 SQL 문의 결과를 확인 URL이 아닌 HTTPS 엔드포인트로 전송합니다.

HTTP 작업 대상은 다음 상태 중 하나일 수 있습니다.

ENABLED

대상이 확인되었으며 규칙 작업에서 사용할 수 있습니다. 대상을 규칙에서 사용하려면 대상의 상태가 ENABLED여야 합니다. DISABLED 상태인 대상만 활성화할 수 있습니다.

DISABLED

대상이 확인되었지만 규칙 작업에서 사용할 수 없습니다. 이는 확인 프로세스를 다시 거치지 않고도 엔드포인트에 대한 트래픽을 일시적으로 차단하려는 경우에 유용합니다. ENABLED 상태인 대상만 비활성화할 수 있습니다.

IN_PROGRESS

대상 확인이 진행 중입니다.

ERROR

대상 확인 시간이 초과되었습니다.

HTTP 작업 대상이 확인되고 활성화되면 계정의 모든 규칙과 함께 사용할 수 있습니다.

HTTP 작업 대상 관리

다음 작업을 사용하여 HTTP 작업 대상을 관리할 수 있습니다.

HTTP 작업 대상 생성

CreateTopicRuleDestination 작업을 호출하거나 AWS IoT 콘솔을 사용하여 HTTP 작업 대상을 생성합니다.

대상을 생성한 후는 확인 URL로 확인 요청을 AWS IoT 보냅니다. 확인 요청의 형식은 다음과 같습니다.

```
HTTP POST {confirmationUrl}/?confirmationToken={confirmationToken}
Headers:
x-amz-rules-engine-message-type: DestinationConfirmation
x-amz-rules-engine-destination-arn:"arn:aws:iot:us-east-1:123456789012:ruledestination/
http/7a280e37-b9c6-47a2-a751-0703693f46e4"
Content-Type: application/json
Body:
{
    "arn":"arn:aws:iot:us-east-1:123456789012:ruledestination/http/7a280e37-b9c6-47a2-
a751-0703693f46e4",
```

```
    "confirmationToken": "AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "enableUrl": "https://iot.us-east-1.amazonaws.com/confirmdestination/
AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "messageType": "DestinationConfirmation"
}
```

확인 요청의 내용에는 다음 정보가 포함됩니다.

arn

확인할 HTTP 작업 대상의 Amazon 리소스 이름(ARN)입니다.

confirmationToken

에서 전송한 확인 토큰입니다 AWS IoT Core. 이 예제의 토큰은 잘려져 있습니다. 사용자의 토큰은 더 길어집니다. AWS IoT Core와 대상을 확인하려면 이 토큰이 필요합니다.

enableUrl

주제 규칙 대상을 확인하기 위해 찾아보는 URL입니다.

messageType

메시지 유형.

HTTP 작업 대상 확인

AWS CLI를 사용하는 경우 엔드포인트 확인 프로세스를 완료하려면 확인 URL에서 확인 요청을 받은 후 다음 중 하나를 수행해야 합니다.

1. 대상이 메시지를 수신할 준비가 되었는지 확인

   HTTP 작업 대상이 IoT 메시지를 수신할 준비가 되었는지 확인하려면 확인 요청`enableUrl`에서를 호출하거나 `ConfirmTopicRuleDestination` API 작업을 수행하고 확인 요청`confirmationToken`에서를 전달합니다.
2. 주제 규칙 상태를 활성화됨으로 설정

   대상에서 메시지를 수신할 수 있는지 확인한 후 `UpdateTopicRuleDestination` API 작업을 수행하여 주제 규칙의 상태를 `ENABLED`로 설정해야 합니다.

AWS IoT 콘솔을 사용하는 경우를 복사`confirmationToken`하여 AWS IoT 콘솔의 대상 확인 대화 상자에 붙여 넣습니다. 그런 다음 주제 규칙을 활성화할 수 있습니다.

새 확인 요청 전송하기

대상에 대해 새 확인 메시지를 활성화하려면 `UpdateTopicRuleDestination`을 호출하고 주제 규칙 대상의 상태를 `IN_PROGRESS`로 설정합니다.

새 확인 요청을 전송한 후 확인 프로세스를 반복합니다.

HTTP 작업 대상 비활성화 및 삭제

대상을 비활성화하려면 `UpdateTopicRuleDestination`을 호출하고 주제 규칙 대상의 상태를 `DISABLED`로 설정합니다. 새 확인 요청을 전송할 필요 없이 DISABLED 상태의 주제 규칙을 다시 사용할 수 있습니다.

HTTP 작업 대상을 삭제하려면를 호출합니다`DeleteTopicRuleDestination`.

인증 기관 지원

> **Note**
>
> 자체 서명된 인증서는 지원되지 않습니다.

HTTP 작업 대상의 HTTPS 엔드포인트는 AWS Private Certificate Authority와 Lets Encrypt 모두에서 발급된 인증서를 지원합니다.

AWS IoT Events

AWS IoT Events (`iotEvents`) 작업은 MQTT 메시지의 데이터를 AWS IoT Events 입력으로 전송합니다.

> **Important**
>
> 페이로드가 AWS IoT Core 없이 로 전송`Input attribute Key`되거나 키가 키에 지정된 것과 동일한 JSON 경로에 있지 않으면 오류와 함께 IoT 규칙이 실패합니다`Failed to send message to Iot Events`.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `iotevents:BatchPutMessage` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.

파라미터

이 작업을 사용하여 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`batchMode`

(선택 사항) 이벤트 작업을 배치로 처리할지 여부입니다. 기본값은 `false`입니다.

`batchMode`가 `true` 이고 규칙 SQL 문이 배열로 평가되면를 호출하여 AWS IoT 이벤트로 전송할 때 각 배열 요소가 별도의 메시지로 처리됩니다`BatchPutMessage`. 결과 배열의 메시지는 10개를 초과할 수 없습니다.

`batchMode`가 `true`인 경우 `messageId`를 지정할 수 없습니다.

대체 템플릿 지원: 아니요

`inputName`

AWS IoT Events 입력의 이름입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`messageId`

(선택 사항) 지정된가 있는 하나의 입력(메시지)만 AWS IoT Events 감지기에 의해 `messageId` 처리되는지 확인하려면이 옵션을 사용합니다. `${newuuid()}` 대체 템플릿을 사용하여 각 요청에 대해 고유한 ID를 생성할 수 있습니다.

`batchMode`가 `true`인 경우 `messageId`을 지정할 수 없습니다--새 UUID 값이 할당됩니다.

대체 템플릿 지원: 예

`roleArn`

가 감지 AWS IoT 기에 입력을 전송하도록 허용하는 IAM 역할입니다 AWS IoT Events . 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 IoT Events 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotEvents": {
                    "inputName": "MyIoTEventsInput",
                    "messageId": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_events"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- AWS IoT Events 개발자 안내서의 란 무엇입니까 AWS IoT Events?

AWS IoT SiteWise

AWS IoT SiteWise (`iotSiteWise`) 작업은 MQTT 메시지의 데이터를의 자산 속성으로 전송합니다 AWS IoT SiteWise.

사물에서 데이터를 수집하는 방법을 보여주는 자습서를 따를 수 AWS IoT 있습니다. 자세한 내용은 사용 AWS IoT SiteWise 설명서의 AWS IoT 사물에서 로 AWS IoT SiteWise 데이터 수집 자습서 또는 AWS IoT 코어 규칙을 사용하여 데이터 수집 섹션을 참조하세요.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `iotsitewise:BatchPutAssetPropertyValue` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

아래 예제는 역할에 연결할 수 있는 신뢰 정책을 보여줍니다.

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

보안을 강화하기 위해 `Condition` 속성에서 AWS IoT SiteWise 자산 계층 구조 경로를 지정할 수 있습니다. 다음 예제는 자산 계층 구조 경로를 지정하는 신뢰 정책입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
            }
        }
        }
    ]
}
```

- 이 작업을 AWS IoT SiteWise 사용하여에 데이터를 전송할 때 데이터는 `BatchPutAssetPropertyValue` 작업의 요구 사항을 충족해야 합니다. 자세한 내용은 AWS IoT SiteWise API 참조의 BatchPutAssetPropertyValue를 참조하세요.

파라미터

이 작업을 사용하여 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`putAssetPropertyValueEntries`

각각 다음 정보를 포함하는 자산 속성 값 항목 목록입니다.

`propertyAlias`

(선택 사항) 자산 속성과 연결된 속성 별칭입니다. `propertyAlias` 또는 `assetId`와 `propertyId` 둘 다 지정합니다. 속성 별칭에 대한 자세한 내용은 AWS IoT SiteWise 사용 설명서의 자산 속성에 산업 데이터 스트림 매핑을 참조하세요.

대체 템플릿 지원: 예

`assetId`

(선택 사항) AWS IoT SiteWise 자산의 ID입니다. `propertyAlias` 또는 `assetId`와 `propertyId` 둘 다 지정합니다.

대체 템플릿 지원: 예

`propertyId`

(선택 사항) 자산 속성 ID입니다. `propertyAlias` 또는 `assetId`와 `propertyId` 둘 다 지정합니다.

대체 템플릿 지원: 예

`entryId`

(선택 사항) 이 항목의 고유 식별자입니다. 오류 발생 시 오류를 유발한 메시지를 더 잘 추적하도록 `entryId`를 정의합니다. 기본값은 새 UUID입니다.

대체 템플릿 지원: 예

`propertyValues`

각각 다음 형식의 타임스탬프, 품질 및 값(TQV)을 포함하는 삽입할 속성 값 목록입니다.

`timestamp`

다음 정보가 포함된 타임스탬프 구조입니다.

`timeInSeconds`

Unix Epoch 시간에 초 단위 시간을 포함하는 문자열입니다. 메시지 페이로드에 타임스탬프가 없는 경우 현재 시간을 밀리초 단위로 반환하는 timestamp()를 사용할 수 있습니다. 이 시간을 초로 변환하려면 대체 템플릿 **`${floor(timestamp() / 1E3)}`**을 사용할 수 있습니다.

대체 템플릿 지원: 예

`offsetInNanos`

(선택 사항) 초 단위 시간에서 나노초 시간 오프셋을 포함하는 문자열입니다. 메시지 페이로드에 타임스탬프가 없는 경우 현재 시간을 밀리초 단위로 반환하는 timestamp()를 사용할 수 있습니다. 해당 시간에서 나노초 시간 오프셋을 계산하려면 대체 템플릿 **`${(timestamp() % 1E3) * 1E6}`**을 사용할 수 있습니다.

대체 템플릿 지원: 예

Unix epoch 시간과 관련하여는 타임스탬프가 과거 최대 7일, 향후 최대 5분인 항목만 AWS IoT SiteWise 허용합니다.

`quality`

(선택 사항) 값의 품질을 설명하는 문자열입니다. 유효한 값: `GOOD`, `BAD`, `UNCERTAIN`.

대체 템플릿 지원: 예

`value`

자산 속성의 데이터 형식에 따라 다음 값 필드 중 하나를 포함하는 값 구조입니다.

`booleanValue`

(선택 사항) 값 항목의 부울 값을 포함하는 문자열입니다.

대체 템플릿 지원: 예

`doubleValue`

(선택 사항) 값 항목의 실수 값을 포함하는 문자열입니다.

대체 템플릿 지원: 예

`integerValue`

(선택 사항) 값 항목의 정수 값을 포함하는 문자열입니다.

대체 템플릿 지원: 예

`stringValue`

(선택 사항) 값 항목의 문자열 값입니다.

대체 템플릿 지원: 예

`roleArn`

자산 속성 값을 보낼 수 있는 AWS IoT 권한을 부여하는 IAM 역할의 ARN입니다 AWS IoT SiteWise. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 기본 IoT SiteWise 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "/some/property/alias",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${my.payload.timeInSeconds}"
                                    },
                                    "value": {
                                        "integerValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
```

```
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 IoT SiteWise 작업을 정의합니다. 이 예제에서는 주제를 속성 별칭 및 `timestamp()` 함수로 사용합니다. 예를 들어 에 데이터를 `/company/windfarm/3/turbine/7/rpm`에 게시하는 경우 이 작업은 지정한 주제와 동일한 속성 별칭을 사용하여 데이터를 자산 속성에 전송합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "${topic()}",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${floor(timestamp() / 1E3)}",
                                        "offsetInNanos": "${(timestamp() % 1E3) * 1E6}"
                                    },
                                    "value": {
                                        "doubleValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- AWS IoT SiteWise 사용 설명서의 AWS IoT SiteWise란 무엇입니까?

- AWS IoT SiteWise 사용 설명서의 AWS IoT Core 규칙을 사용하여 데이터 수집
- AWS IoT SiteWise 사용 설명서의 AWS IoT 사물 AWS IoT SiteWise 에서 로 데이터 수집
- AWS IoT SiteWise 사용 설명서의 AWS IoT SiteWise 규칙 작업 문제 해결

Firehose

Firehose(`firehose`) 작업을 사용하여 MQTT 메시지의 데이터를 Amazon Data Firehose 스트림에 전송합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `firehose:PutRecord` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서 역할을 선택하거나 생성하여가이 규칙 작업을 수행하도록 허용할 수 AWS IoT 있습니다.
- Firehose를 사용하여 Amazon S3 버킷으로 데이터를 전송하고 고객 관리형를 AWS KMS 사용하여 Amazon S3에 저장된 데이터를 AWS KMS key 암호화하는 경우 Firehose는 버킷에 대한 액세스 권한과 호출자를 대신하여를 사용할 수 있는 AWS KMS key 권한이 있어야 합니다. 자세한 내용은 Amazon Data Firehose 개발자 안내서의 Firehose에 Amazon S3 대상에 대한 액세스 권한 부여를 참조하세요.

파라미터

이 작업을 사용하여 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`batchMode`

(선택 사항) `PutRecordBatch`를 사용하여 Firehose 스트림을 배치로 전송할지 여부입니다. 기본값은 `false`입니다.

`batchMode`가 `true`이고 규칙의 SQL 문이 Array로 평가되는 경우 각 Array 요소는 `PutRecordBatch` 요청에서 하나의 레코드를 형성합니다. 결과 배열의 레코드는 500개를 초과할 수 없습니다.

대체 템플릿 지원: 아니요

`deliveryStreamName`

메시지 데이터를 기록할 Firehose 스트림입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`separator`

(선택 사항) Firehose 스트림에 기록된 레코드를 구분하는 데 사용할 문자 구분 기호입니다. 이 파라미터를 생략하면 스트림에서 구분 기호를 사용하지 않습니다. 유효한 값: `,`(쉼표) `\t`(탭), `\n`(줄 바꿈) `\r\n`(Windows 줄 바꿈).

대체 템플릿 지원: 아니요

`roleArn`

Firehose 스트림에 대한 액세스 권한을 부여하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 Firehose 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "my_firehose_stream",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 Firehose 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon Kinesis Data Firehose 개발자 가이드의 Amazon Kinesis Data Firehose란?

Kinesis Data Streams

Kinesis Data Streams(`kinesis`) 규칙 작업은 MQTT 메시지의 데이터를 Amazon Kinesis Data Streams에 전송합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `kinesis:PutRecord` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.
- AWS KMS 고객 관리형 AWS KMS key (KMS 키)를 사용하여 Kinesis Data Streams에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 AWS KMS key 를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 사용자 생성 AWS KMS keys를 사용할 수 있는 권한을 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`stream`

데이터를 기록할 Kinesis 데이터 스트림입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`partitionKey`

어느 샤드에 데이터를 기록할지 결정하는 데 사용되는 파티션 키입니다. 파티션 키는 일반적으로 표현식(예: `${topic()}` 또는 `${timestamp()}`)으로 구성됩니다.

대체 템플릿 지원: 예

`roleArn`

Kinesis 데이터 스트림에 액세스할 수 있는 AWS IoT 권한을 부여하는 IAM 역할의 ARN입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 Kinesis Data Streams 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "my_kinesis_stream",
                    "partitionKey": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
```

```
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 Kinesis 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "${topic()}",
                    "partitionKey": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams란 무엇입니까?

Lambda

Lambda(`lambda`) 작업은 AWS Lambda 함수를 호출하여 MQTT 메시지를 전달합니다. Lambda 함수를 비동기적으로 AWS IoT 호출합니다.

Lambda 작업을 사용하여 규칙을 생성하고 테스트하는 방법을 보여주는 자습서를 따라할 수 있습니다. 자세한 내용은 자습서: AWS Lambda 함수를 사용하여 알림 형식 지정 단원을 참조하십시오.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- 가 Lambda 함수를 호출 AWS IoT 하려면 `lambda:InvokeFunction` 권한을 부여하는 정책을 구성해야 합니다 AWS IoT. Lambda 정책이 있는 AWS 리전 동일한에 정의된 Lambda 함수만 호출할 수

있습니다. Lambda 함수는 리소스 기반 정책을 사용합니다. 이때 정책은 Lambda 함수 자체에 직접 연결되어야 합니다.

다음 AWS CLI 명령을 사용하여 `lambda:InvokeFunction` 권한을 부여하는 정책을 연결합니다. 이 명령에서 다음과 같이 바꿉니다.

- *`function_name`*을 Lambda 함수의 이름으로 대체합니다. 함수의 리소스 정책을 업데이트하려면 새 권한을 추가 합니다.
- 함수 AWS 리전 의이 있는 *`region`*.
- 규칙이 정의된 AWS 계정 번호가 포함된 *`account-id`*입니다.
- *`rule-name`*을 Lambda 작업을 정의하려는 AWS IoT 규칙의 이름으로 바꿉니다.
- *`unique_id`*를 고유 문 식별자로 대체합니다.

> **Important**
>
> `source-arn` 또는를 제공하지 않고 AWS IoT 보안 주체에 대한 권한을 추가하는 경우 Lambda 작업으로 규칙을 생성하는 `source-account`는 Lambda 함수를 호출 AWS 계정 하는 규칙을 활성화할 수 있습니다 AWS IoT.

자세한 내용은 AWS Lambda 권한을 참조하세요.

```
aws lambda add-permission \
    --function-name function_name \
    --region region \
    --principal iot.amazonaws.com \
    --source-arn arn:aws:iot:region:account-id:rule/rule_name \
    --source-account account-id
    --statement-id unique_id
    --action "lambda:InvokeFunction"
```

- AWS IoT 콘솔을 사용하여 Lambda 규칙 작업에 대한 규칙을 생성하면 Lambda 함수가 자동으로 트리거됩니다. AWS CloudFormation 대신를와 함께 사용하는 경우 `AWS::lambda::Permission` 리소스를 추가`AWS::IoT::TopicRule LambdaAction`해야 합니다. 그런 다음 추가된 리소스는 사용자에게 Lambda 함수를 트리거할 수 있는 권한을 부여합니다.

  다음 코드는 이 리소스를 추가하는 방법을 보여주는 예제입니다. 대체 예시:

  - *`function_name`*을 Lambda 함수의 이름으로 대체합니다.
  - 함수 AWS 리전 의이 있는 *`region`*.

- 규칙이 정의된 AWS 계정 번호가 포함된 *account-id*입니다.
- *rule-name*을 Lambda 작업을 정의하려는 AWS IoT 규칙의 이름으로 바꿉니다.

```
Type: AWS::Lambda::Permission
Properties:
  Action: lambda:InvokeFunction
  FunctionName: !Ref function_name
  Principal: "iot.amazonaws.com"
  SourceAccount: account-id
  SourceArn: arn:aws:iot:region:account-id:rule/rule_name
```

- AWS KMS 고객 관리형 AWS KMS key 를 사용하여 Lambda에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 AWS KMS key 를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 AWS Lambda 개발자 안내서의 저장 시 암호화를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`functionArn`

간접 호출할 Lambda 함수의 ARN은 함수를 간접 호출할 권한이 있어야 AWS IoT 합니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

Lambda 함수의 버전 또는 별칭을 지정하지 않은 경우, 가장 최신 버전의 함수가 종료됩니다. 특정 버전의 Lambda 함수를 종료하고자 하는 경우 버전 또는 별칭을 지정하면 됩니다. 버전 또는 별칭을 지정하려면 Lambda 함수의 ARN에 버전 또는 별칭을 추가합니다.

```
arn:aws:lambda:us-east-2:123456789012:function:myLambdaFunction:someAlias
```

버전 관리 및 별칭에 대한 자세한 내용은 AWS Lambda 함수 버전 관리 및 별칭을 참조하세요.

대체 템플릿 지원: API 및 AWS CLI 만 해당

예제

다음 JSON 예제는 AWS IoT 규칙에서 Lambda 작업을 정의합니다.

```
{
```

```
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-
east-2:123456789012:function:myLambdaFunction"
                 }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 Lambda 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:
${topic()}"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- AWS Lambda 개발자 안내서의 란 무엇입니까 AWS Lambda?
- 자습서: AWS Lambda 함수를 사용하여 알림 형식 지정

Location

Location(`location`) 작업은 지리적 위치 데이터를 Amazon Location Service로 전송합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `geo:BatchUpdateDevicePosition` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 AWS IoT 역할을 선택하거나 생성할 수 있습니다.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`deviceId`

위치 데이터를 제공하는 디바이스의 고유 ID입니다. 자세한 내용은 Amazon Location Service API 참조의 `DeviceId` 섹션을 참조하세요.

대체 템플릿 지원: 예

`latitude`

디바이스 위치의 위도를 나타내는 double 값으로 평가되는 문자열입니다.

대체 템플릿 지원: 예

`longitude`

디바이스 위치의 경도를 나타내는 double 값으로 평가되는 문자열입니다.

대체 템플릿 지원: 예

`roleArn`

Amazon Location Service 도메인에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

`timestamp`

위치 데이터가 샘플링된 시간입니다. 기본값은 MQTT 메시지가 처리된 시간입니다.

`timestamp` 값은 다음 두 값으로 구성됩니다.

- `value`: 긴 Epoch 시간 값을 반환하는 표현식입니다. the section called “time_to_epoch(String, String)” 함수를 사용하여 메시지 페이로드에 전달된 날짜 또는 시간 값에서 유효한 타임스탬프를 만들 수 있습니다. 대체 템플릿 지원: 예.
- `unit`: (선택 사항) `value`에 설명된 표현식에서 기인하는 타임스탬프 값의 정밀도입니다. 유효한 값: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`. 기본값은 `MILLISECONDS`입니다. 대체 템플릿: API 및 AWS CLI 만 지원합니다.

`trackerName`

위치가 업데이트되는 Amazon Location의 추적기 리소스 이름입니다. 자세한 정보는 Amazon Location Service 개발자 안내서의 추적기를 참조하세요.

대체 템플릿 지원: API 및 AWS CLI 만 해당

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 Location 작업을 정의합니다.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123454962127:role/service-role/ExampleRole",
     "trackerName": "MyTracker",
     "deviceId": "001",
     "sampleTime": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "-12.3456",
     "longitude": "65.4321"
    }
   }
  ]
 }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿이 있는 위치 작업을 정의합니다.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123456789012:role/service-role/ExampleRole",
     "trackerName": "${TrackerName}",
     "deviceId": "${DeviceID}",
     "timestamp": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "${get(position, 0)}",
     "longitude": "${get(position, 1)}"
    }
   }
  ]
 }
}
```

다음 MQTT 페이로드 예시는 위 예시의 대체 템플릿이 데이터에 액세스하는 방법을 보여줍니다. get-device-position-history CLI 명령을 사용하여 MQTT 페이로드 데이터가 위치 추적기에 전달되는지 확인할 수 있습니다.

```
{
 "TrackerName": "mytracker",
 "DeviceID": "001",
 "position": [
  "-12.3456",
  "65.4321"
 ]
}
```

```
aws location get-device-position-history --device-id 001 --tracker-name mytracker
```

```
{
 "DevicePositions": [
  {
```

```
    "DeviceId": "001",
    "Position": [
     -12.3456,
     65.4321
    ],
    "ReceivedTime": "2022-11-11T01:31:54.464000+00:00",
    "SampleTime": "2022-11-11T01:31:54.308000+00:00"
   }
  ]
}
```

다음 사항도 참조하세요.

- Amazon Location Service Developer Guide(Amazon Location Service 개발자 안내서)의 What is Amazon Location Service?(Amazon Location Service란 무엇인가요?)

OpenSearch

OpenSearch(`openSearch`) 작업을 사용하여 MQTT 메시지의 데이터를 Amazon OpenSearch Service 도메인에 전송합니다. 그런 다음 OpenSearch 대시보드와 같은 도구를 사용하여 OpenSearch Service에서 데이터를 쿼리하고 시각화할 수 있습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `es:ESHttpPut` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.
- 고객 관리형 AWS KMS key 를 사용하여 OpenSearch Service에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 KMS 키를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon OpenSearch Service 개발자 안내서의 Amazon OpenSearch Service의 저장 시 데이터 암호화를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`endpoint`

Amazon OpenSearch Service 도메인의 엔드포인트입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`index`

데이터를 저장할 OpenSearch 인덱스입니다.

대체 템플릿 지원: 예

`type`

저장 중인 문서의 유형입니다.

> **Note**
>
> OpenSearch 버전이 1.0 이상인 경우 `type` 파라미터 값은 `_doc`여야 합니다. 자세한 내용은 OpenSearch 설명서를 참조하세요.

대체 템플릿 지원: 예

`id`

각 문서의 고유 식별자입니다.

대체 템플릿 지원: 예

`roleARN`

OpenSearch 도메인에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

제한 사항

OpenSearch(`openSearch`) 작업은 VPC Elasticsearch 클러스터로 데이터를 전달하는 데 사용할 수 없습니다.

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 OpenSearch 작업을 정의하고 OpenSearch 작업에 대한 필드를 지정하는 방법을 정의합니다. 자세한 내용은 OpenSearchAction을 참조하세요.

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "_doc",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 OpenSearch 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
```

```
}
```

> **Note**
>
> 대체 `type` 필드는 OpenSearch 버전 1.0에서 작동합니다. 1.0 이후 버전에서는 `type`의 값이 `_doc`여야 합니다.

다음 사항도 참조하세요.

Amazon OpenSearch Service 개발자 가이드의 Amazon OpenSearch Service란 무엇입니까?

재게시

다시 게시(`republish`) 작업은 MQTT 메시지를 다른 MQTT 주제에 다시 게시합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `iot:Publish` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`headers`

MQTT 버전 5.0 헤더 정보입니다.

자세한 내용은 AWS API 참조의 RepublishAction 및 MqttHeaders를 참조하세요.

`topic`

메시지를 재게시할 MQTT 주제입니다.

$로 시작하는 예약된 주제에 게시하려면 대신 $$를 사용하세요. 예를 들어 디바이스 섀도우 주제 $aws/things/MyThing/shadow/update에 다시 게시하려면 주제를 $$aws/things/MyThing/shadow/update로 지정합니다.

> **Note**
> 예약된 작업 주제에 다시 게시하는 것은 지원되지 않습니다.
> AWS IoT Device Defender 예약 주제는 HTTP 게시를 지원하지 않습니다.

대체 템플릿 지원: 예

`qos`

(선택 사항) 메시지를 다시 게시할 때 사용할 서비스 품질(QoS) 수준입니다. 유효한 값: 0, 1. 기본값은 0입니다. MQTT QoS에 대한 자세한 내용은 MQTT 섹션을 참조하세요.

대체 템플릿 지원: 아니요

`roleArn`

가 MQTT 주제에 AWS IoT 게시하도록 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 재게시 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "another/topic",
                    "qos": 1,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
```

```
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 재게시 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙headers에서를 사용하여 재게시 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
                    "headers": {
                        "payloadFormatIndicator": "UTF8_DATA",
                        "contentType": "rule/contentType",
                        "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
                        "userProperties": [
                            {
                                "key": "ruleKey1",
                                "value": "ruleValue1"
```

```
                        },
                        {
                            "key": "ruleKey2",
                            "value": "ruleValue2"
                        }
                    ]
                }
            }
        }
    ]
  }
}
```

> **Note**
>
> 원본 소스 IP는 재게시 작업을 거쳐 전달되지 않습니다.

S3

S3(s3) 작업은 MQTT 메시지의 데이터를 Amazon Simple Storage Service(Amazon S3) 버킷에 기록합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `s3:PutObject` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.
- AWS KMS customermanaged를 사용하여 Amazon S3의 저장 데이터를 암호화하는 경우 호출자를 대신하여 AWS KMS key 를 AWS KMS key 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 AWS 관리형 AWS KMS keys 및 고객 관리 AWS KMS keys형을 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`bucket`

데이터를 기록할 Amazon S3 버킷입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`cannedacl`

(선택 사항) 객체 키로 식별된 객체에 대한 액세스를 제어할 수 있도록 Amazon S3에서 미리 정의된 ACL입니다. 허용 값을 포함하여 자세한 내용은 미리 정의된 ACL을 참조하세요.

대체 템플릿 지원: 아니요

`key`

데이터가 기록되는 파일의 경로입니다.

이 파라미터가 `${topic()}/${timestamp()}`이고 규칙이 주제가 `some/topic`인 메시지를 수신하는 예를 고려하세요. 현재 타임스탬프가 `1460685389`인 경우 이 작업은 S3 버킷의 `some/topic` 폴더에 있는 `1460685389`라는 파일에 데이터를 씁니다.

> **Note**
>
> 정적 키를 사용하는 경우는 규칙이 호출할 때마다 단일 파일을 AWS IoT 덮어씁니다. 메시지 타임스탬프 또는 다른 고유한 메시지 식별자를 사용하는 것이 좋습니다. 그러면 각 메시지가 수신될 때마다 Amazon S3에 새 파일이 저장됩니다.

대체 템플릿 지원: 예

`roleArn`

Amazon S3 버킷에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 S3 작업을 정의합니다.

```
{
```

```
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "bucketName": "amzn-s3-demo-bucket",
                    "cannedacl": "public-read",
                    "key": "${topic()}/${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon Simple Storage Service 사용 설명서의 Amazon S3란 무엇입니까?

Salesforce IoT

Salesforce IoT(`salesforce`) 작업은 규칙을 트리거한 MQTT 메시지의 데이터를 Salesforce IoT 입력 스트림에 전송합니다.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`url`

Salesforce IoT 입력 스트림에서 공개한 URL입니다. 이 URL은 입력 스트림을 만들 때 Salesforce IoT 플랫폼에서 제공합니다. 자세한 내용은 Salesforce IoT 설명서를 참조하세요.

대체 템플릿 지원: 아니요

`token`

지정된 Salesforce IoT 입력 스트림에 대한 액세스를 인증하는 데 사용되는 토큰입니다. 이 토큰은 입력 스트림을 만들 때 Salesforce IoT 플랫폼에서 제공합니다. 자세한 내용은 Salesforce IoT 설명서를 참조하세요.

대체 템플릿 지원: 아니요

예제

다음 JSON 예제는 AWS IoT 규칙에서 Salesforce IoT 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "salesforce": {
                    "token": "ABCDEFGHI123456789abcdefghi123456789",
                    "url": "https://ingestion-cluster-id.my-env.sfdcnow.com/streams/
stream-id/connection-id/my-event"
                }
            }
        ]
    }
}
```

SNS

SNS(sns) 작업은 MQTT 메시지의 데이터를 Amazon Simple Notification Service(Amazon SNS) 푸시 알림으로서 전송합니다.

SNS 작업을 사용하여 규칙을 생성하고 테스트하는 방법을 보여주는 자습서를 따라할 수 있습니다. 자세한 내용은 자습서: Amazon SNS 알림 전송 단원을 참조하세요.

> **Note**
>
> SNS 작업은 Amazon SNS 주제 FIFO(선입선출)를 지원하지 않습니다. 규칙 엔진은 완전 분산형 서비스이므로 SNS 작업이 호출되는 메시지 순서가 보장되지 않습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `sns:Publish` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.
- AWS KMS 고객 관리형 AWS KMS key 를 사용하여 Amazon SNS에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 AWS KMS key 를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 키 관리를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`targetArn`

푸시 알림이 전송될 SNS 주제 또는 개별 디바이스입니다.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`messageFormat`

(선택 사항) 메시지 형식입니다. Amazon SNS는 이 설정을 사용하여 페이로드를 구문 분석하고 페이로드의 해당 플랫폼별 부분을 추출할지 여부를 결정합니다. 유효한 값: `JSON`, `RAW`. 기본값은 `RAW`입니다.

대체 템플릿 지원: 아니요

`roleArn`

SNS에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제는 AWS IoT 규칙에서 SNS 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
```

```
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-2:123456789012:my_sns_topic",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 SNS 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-1:123456789012:${topic()}",
                    "messageFormat": "JSON",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon Simple Notification Service 개발자 가이드의 Amazon Simple Notification Service란?
- 자습서: Amazon SNS 알림 전송

SQS

SQS(sqs) 작업은 MQTT 메시지의 데이터를 Amazon Simple Queue Service(Amazon SQS) 대기열에 전송합니다.

> **Note**
>
> SQS 작업은 Amazon SQS 대기열 FIFO(선입선출)를 지원하지 않습니다. 규칙 엔진은 완전 분산형 서비스이므로 SQS 작업이 트리거되는 메시지 순서가 보장되지 않습니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `sqs:SendMessage` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.
- AWS KMS 고객 관리형를 사용하여 Amazon AWS KMS key SQS에서 저장 데이터를 암호화하는 경우 호출자를 대신하여 AWS KMS key 를 사용할 수 있는 권한이 서비스에 있어야 합니다. Amazon SQS 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 키 관리를 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`queueUrl`

데이터를 기록할 Amazon SQS 대기열의 URL입니다. 이 URL의 리전은 AWS IoT 규칙 AWS 리전과 같을 필요가 없습니다.

> **Note**
>
> SQS 규칙 작업을 사용하여 데이터 전송 크로스 AWS 리전 에 대한 추가 요금이 발생할 수 있습니다. 자세한 내용은 Amazon SQS 요금을 참조하세요.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`useBase64`

이 파라미터를 `true`로 설정하여 메시지 데이터가 Amazon SQS 대기열에 기록되기 전에 메시지 데이터를 base64로 인코딩하도록 규칙 작업을 구성합니다. 기본값은 `false`입니다.

대체 템플릿 지원: 아니요

`roleArn`

Amazon SQS 대기열에 대한 액세스를 허용하는 IAM 역할입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 SQS 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
my_sqs_queue",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

다음 JSON 예제에서는 AWS IoT 규칙에서 대체 템플릿을 사용하여 SQS 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
${topic()}",
                    "useBase64": true,
```

```
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- Amazon Simple Queue Service 개발자 안내서의 Amazon Simple Queue Service 란?

단계 함수

Step Functions(`stepFunctions`) 작업은 AWS Step Functions 상태 시스템을 시작합니다.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `states:StartExecution` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택하거나 생성할 수 AWS IoT 있습니다.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`stateMachineName`

시작할 Step Functions 상태 머신 이름.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`executionNamePrefix`

(선택 사항) 뒤에 UUID가 붙는 이 접두사로 이루어진 상태 머신 실행에 부여된 이름. Step Functions는 각 상태 머신 실행에 제공된 고유한 이름이 없는 경우 고유한 이름을 생성합니다.

대체 템플릿 지원: 예

`roleArn`

상태 시스템을 시작할 수 있는 AWS IoT 권한을 부여하는 역할의 ARN입니다. 자세한 내용은 요구 사항 단원을 참조하십시오.

대체 템플릿 지원: 아니오

예제

다음 JSON 예제에서는 AWS IoT 규칙에서 Step Functions 작업을 정의합니다.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "stepFunctions": {
                    "stateMachineName": "myStateMachine",
                    "executionNamePrefix": "myExecution",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_step_functions"
                }
            }
        ]
    }
}
```

다음 사항도 참조하세요.

- AWS Step Functions 개발자 안내서의 란 무엇입니까 AWS Step Functions?

Timestream

Timestream 규칙 작업은 MQTT 메시지의 속성(측정값)을 Amazon Timestream 테이블에 기록합니다. Amazon Timestream 에 대한 자세한 내용은 Amazon Timestream 이란?을 참조하세요.

> **Note**
>
> 일부 AWS 리전에서는 Amazon Timestream을 사용할 수 없습니다. 리전에서 Amazon Timestream을 사용할 수 없는 경우 규칙 작업 목록에 해당 Timestream이 나타나지 않습니다.

이 규칙이 Timestream 데이터베이스에 저장하는 속성은 규칙의 쿼리 문에서 가져온 속성입니다. 쿼리 문의 결과에 있는 각 속성의 값은 데이터 유형을 추론하기 위해 구문 분석됩니다(예: the section called "DynamoDBv2" 작업). 각 속성의 값은 Timestream 테이블의 자체 레코드에 기록됩니다. 속성의 데이터 유형을 지정하거나 변경하려면 쿼리 문에서 `cast()` 함수를 사용합니다. 각 Timestream 레코드의 내용에 대한 자세한 내용은 the section called "Timestream 레코드 내용" 섹션을 참조하세요.

> **Note**
>
> SQL V2(2016-03-23)에서는 `10.0`과 같은 정수인 숫자 값이 정수 표현(`10`)으로 변환됩니다. cast() 함수를 사용하는 것과 같이 명시적으로 `Decimal` 값으로 캐스팅해도 결과는 바뀌지 않습니다. 결과는 여전히 `Integer` 값입니다. 이로 인해 형식 불일치 오류가 발생하여 Timestream 데이터베이스에 데이터가 기록되지 않을 수 있습니다. 정수 숫자 값을 `Decimal` 값으로 처리하려면 규칙 쿼리 문에 SQL V1(2015-10-08)을 사용하세요.

> **Note**
>
> Timestream 규칙 작업이 Amazon Timestream 테이블에 쓸 수 있는 최대 값 수는 100입니다. 자세한 내용은 Amazon Timestream 할당량 참조를 참조하세요.

요구 사항

이 규칙 작업은 다음 요구 사항을 충족해야 합니다.

- `timestream:DescribeEndpoints` 및 `timestream:WriteRecords` 작업을 수행하기 위해가 맡을 AWS IoT 수 있는 IAM 역할입니다. 자세한 내용은 AWS IoT 규칙에 필요한 액세스 권한 부여 단원을 참조하십시오.

  AWS IoT 콘솔에서가이 규칙 작업을 수행하도록 허용하는 역할을 선택, 업데이트 또는 생성할 수 AWS IoT 있습니다.

- 고객-를 사용하여 Timestream에서 저장 데이터를 암호화하는 경우 호출 AWS KMS key 자를 대신하여 AWS KMS 를 사용할 수 있는 권한이 서비스에 있어야 합니다. 자세한 내용은 AWS 서비스에서 AWS KMS를 사용하는 방법을 참조하세요.

파라미터

이 작업으로 AWS IoT 규칙을 생성할 때 다음 정보를 지정해야 합니다.

`databaseName`

이 작업에서 생성되는 레코드를 수신할 테이블이 있는 Amazon Timestream 데이터베이스의 이름입니다. 또한 **`tableName`** 단원도 참조하세요.

대체 템플릿 지원: API 및 AWS CLI 만 해당

`dimensions`

각 측정값 레코드에 기록되는 시계열의 메타데이터 속성입니다. 예를 들어 EC2 인스턴스의 이름 및 가용 영역 또는 풍력 터빈 제조업체 이름은 차원입니다.

`name`

메타데이터 차원 이름입니다. 데이터베이스 테이블 레코드 열의 이름입니다.

차원 이름은 `measure_name`, `measure_value` 또는 `time`으로 지정할 수 없습니다. 이러한 이름은 예약되어 있습니다. 차원 이름은 `ts_` 또는 `measure_value`로 시작할 수 없으며 콜론(`:`) 문자를 포함할 수 없습니다.

대체 템플릿 지원: 아니요

`value`

데이터베이스 레코드의 이 열에 쓸 값입니다.

대체 템플릿 지원: 예

`roleArn`

Timestream 데이터베이스 테이블에 쓸 수 있는 권한을 AWS IoT 에 부여하는 역할의 Amazon 리소스 이름(ARN)입니다. 자세한 내용은 요구 사항 단원을 참조하세요.

대체 템플릿 지원: 아니요

tableName

측정값 레코드를 기록할 데이터베이스 테이블의 이름입니다. 또한 **databaseName** 단원도 참조하세요.

대체 템플릿 지원: API 및 AWS CLI 만 해당

timestamp

항목의 타임스탬프에 사용할 값입니다. 비워 두면 항목이 처리된 시간이 사용됩니다.

`unit`

`value`에 설명된 표현식에서 기인하는 타임스탬프 값의 정밀도입니다.

유효한 값: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`. 기본값은 `MILLISECONDS`입니다.

`value`

긴 Epoch 시간 값을 반환하는 표현식입니다.

the section called “time_to_epoch(String, String)” 함수를 사용하여 메시지 페이로드에 전달된 날짜 또는 시간 값에서 유효한 타임스탬프를 만들 수 있습니다.

Timestream 레코드 내용

이 작업을 통해 Amazon Timestream 테이블에 기록된 데이터에는 타임스탬프, Timestream 규칙 작업의 메타데이터 및 규칙 쿼리 문의 결과가 포함됩니다.

쿼리 문의 결과에 있는 각 속성(측정값)에 대해 이 규칙 작업은 지정된 Timestream 테이블에 레코드를 기록합니다.

열 이름	속성 유형	값	설명
dimension-name	DIMENSION	Timestream 규칙 작업 항목에 지정된 값입니다.	규칙 작업 항목에 지정된 각 차원은 차원 이름을 사용하여 Timestream 데이터베이스에 열을 생성합니다.

열 이름	속성 유형	값	설명
measure_name	MEASURE_NAME	속성의 이름	`measure_value::` *data-type* 열에 값이 지정된 쿼리 문의 결과에 있는 속성의 이름입니다.
measure_value::*data-type*	MEASURE_VALUE	쿼리 문의 결과에 있는 속성 값입니다. 속성의 이름은 `measure_name` 열에 있습니다.	값은 해석되고* `bigint`, `boolean`, `double` 또는 `varchar` 중 가장 일치하는 항목으로 캐스팅됩니다. Amazon Timestream은 각 데이터 유형에 대해 별도의 열을 생성합니다. 규칙의 쿼리 문에서 `cast()` 함수를 사용하여 메시지의 값을 다른 데이터 형식으로 캐스팅할 수 있습니다.
시간	TIMESTAMP	데이터베이스에 있는 레코드의 날짜 및 시간입니다.	이 값은 정의된 경우 규칙 엔진 또는 `timestamp` 속성에 의해 지정됩니다.

* 메시지 페이로드에서 읽은 속성 값은 다음과 같이 해석됩니다. 이러한 각 경우에 대한 설명은 the section called “예제”를 참조하세요.

- 인용되지 않은 값 `true` 또는 `false`는 `boolean` 유형으로 해석됩니다.
- 십진수는 `double` 유형으로 해석됩니다.
- 소수점이 없는 숫자 값은 `bigint` 유형으로 해석됩니다.
- 따옴표로 묶인 문자열은 `varchar` 유형으로 해석됩니다.
- 객체 및 배열 값은 JSON 문자열로 변환되어 `varchar` 유형으로 저장됩니다.

예제

다음 JSON 예제에서는 규칙에서 대체 템플릿을 사용하여 Timestream AWS IoT 규칙 작업을 정의합니다.

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "timestream": {
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_timestream",
          "tableName": "devices_metrics",
          "dimensions": [
            {
              "name": "device_id",
              "value": "${clientId()}"
            },
            {
              "name": "device_firmware_sku",
              "value": "My Static Metadata"
            }
          ],
          "databaseName": "record_devices"
        }
      }
    ]
  }
}
```

다음 메시지 페이로드와 함께 이전 예에서 정의한 Timestream 주제 규칙 작업을 사용하면 다음 테이블에 Amazon Timestream 레코드가 작성됩니다.

```
{
  "boolean_value": true,
  "integer_value": 123456789012,
  "double_value": 123.456789012,
  "string_value": "String value",
  "boolean_value_as_string": "true",
  "integer_value_as_string": "123456789012",
```

```
  "double_value_as_string": "123.456789012",
  "array_of_integers": [23,36,56,72],
  "array of strings": ["red", "green","blue"],
  "complex_value": {
    "simple_element": 42,
    "array_of_integers": [23,36,56,72],
    "array of strings": ["red", "green","blue"]
  }
}
```

다음 테이블에는 지정된 주제 규칙 작업을 사용하여 이전 메시지 페이로드를 처리하는 데이터베이스 열과 레코드가 표시됩니다. `device_firmware_sku` 및 `device_id` 열은 주제 규칙 작업에 정의된 DIMENSIONS입니다. Timestream 주제 규칙 작업은 `time` 열과 `measure_name` 및 `measure_value::*` 열을 생성하며, 이를 주제 규칙 작업의 쿼리 문의 결과 값으로 채웁니다.

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	시간
My Static Metadata	iotconsole-159EXAMPLE738-0	complex_value	-	{"simple_element":42,"array_of_integers":[23,36,56,72],"array of strings":["red","green","blue"]}	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	integer_value_as_string	-	123456789012	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	boolean_value	-	-	-	TRUE	2020-08-26 22:42:16.423000000

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	시간
My Static Metadata	iotconsole-159EXAMPLE738-0	integer_value	123456789012	-	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	string_value	-	문자열 값	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	array_of_integers	-	[23,36,56,72]	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	문자열 배열	-	["red","green","blue"]	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	boolean_value_as_string	-	TRUE	-	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	double_value	-	-	123.456789012	-	2020-08-26 22:42:16.423000000
My Static Metadata	iotconsole-159EXAMPLE738-0	double_value_as_string	-	123.45679	-	-	2020-08-26 22:42:16.423000000

규칙 문제 해결

규칙에 문제가 있는 경우 CloudWatch Logs를 활성화하는 것이 좋습니다. 로그를 분석하여 문제가 권한 부여인지 또는 예를 들어 WHERE 절 조건이 일치하지 않는지 여부를 확인할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 설정을 참조하세요.

AWS IoT 규칙을 사용하여 교차 계정 리소스에 액세스

한 계정의 MQTT 주제에 수집된 데이터를 다른 계정의 Amazon SQS 및 Lambda와 같은 AWS 서비스로 라우팅할 수 있도록 교차 계정 액세스 AWS IoT 규칙을 구성할 수 있습니다. 다음은 한 계정의 MQTT 주제에서 다른 계정의 대상으로 교차 계정 데이터 수집 AWS IoT 규칙을 설정하는 방법을 설명합니다.

대상 리소스에 대한 리소스 기반 권한을 사용하여 교차 계정 규칙을 구성할 수 있습니다. 따라서 AWS IoT 규칙을 사용하여 교차 계정 액세스에는 리소스 기반 권한을 지원하는 대상만 활성화할 수 있습니다. 지원되는 대상에는 Amazon SQS, Amazon SNS, Amazon S3 및 AWS Lambda가 있습니다.

> **Note**
>
> Amazon SQS를 제외한 지원되는 대상의 경우 규칙 작업이 해당 리소스와 상호 작용할 수 있도록 다른 서비스의 리소스 AWS 리전 와 동일한에서 규칙을 정의해야 합니다. AWS IoT 규칙 작업에 대한 자세한 내용은 AWS IoT 규칙 작업을 참조하세요. 규칙의 SQS 작업에 대한 자세한 내용은 ??? 섹션을 참조하세요.

사전 조건

- AWS IoT 규칙 숙지
- IAM 사용자, 역할 및 리소스 기반 권한에 대한 이해
- AWS CLI가 설치되어 있음

Amazon SQS에 대한 교차 계정 설정

시나리오: 계정 A는 MQTT 메시지의 데이터를 계정 B의 Amazon SQS 대기열에 전송합니다.

AWS 계정	계정 이름	설명
1111-1111-1111	계정 A	규칙 작업: `sqs:SendMessage`
2222-2222-2222	계정 B	Amazon SQS 대기열 • ARN: *arn:aws:sqs:region:2222-2222-2222:ExampleQueue*

AWS 계정	계정 이름	설명
		• URL: *https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue*

> **Note**
>
> 대상 Amazon SQS 대기열이 AWS IoT 규칙 AWS 리전 과 동일한에 있을 필요는 없습니다. 규칙의 SQS 작업에 대한 자세한 내용은 ??? 섹션을 참조하세요.

계정 A 태스크 수행

> **Note**
>
> 다음 명령을 실행하려면 규칙의 Amazon 리소스 이름(ARN)을 리소스로 사용하는 `iot:CreateTopicRule`에 대한 권한과 리소스를 역할의 ARN으로 사용하는 `iam:PassRole` 작업에 대한 권한이 IAM 사용자에게 있어야 합니다.

1. 계정 A의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. AWS IoT 규칙 엔진을 신뢰하고 계정 B의 Amazon SQS 대기열에 대한 액세스를 허용하는 정책을 연결하는 IAM 역할을 생성합니다. 필요한 액세스 권한 부여의 명령 및 정책 문서 예제 AWS IoT 를 참조하세요.
3. 주제에 연결된 규칙을 생성하려면 create-topic-rule 명령을 실행합니다.

   ```
   aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
   ```

   다음은 `iot/test` 주제에 전송된 모든 메시지를 지정된 Amazon SQS 대기열에 삽입하는 규칙이 있는 페이로드 파일의 예입니다. SQL 문은 메시지를 필터링하고 역할 ARN은 Amazon SQS 대기열에 메시지를 추가할 수 있는 권한을 AWS IoT 에 부여합니다.

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
   ```

```
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sqs": {
    "queueUrl": "https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role",
    "useBase64": false
   }
  }
 ]
}
```

AWS IoT 규칙에서 Amazon SQS 작업을 정의하는 방법에 대한 자세한 내용은 AWS IoT 규칙 작업 - Amazon SQS를 참조하세요.

계정 B 태스크 수행

1. 계정 B의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. Amazon SQS 대기열 리소스에 대한 권한을 계정 A에 부여하려면 add-permission 명령을 실행합니다.

```
aws sqs add-permission --queue-url https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue --label SendMessagesToMyQueue --aws-account-ids 1111-1111-1111 --actions SendMessage
```

Amazon SNS를 위한 교차 계정 설정

시나리오: 계정 A는 MQTT 메시지의 데이터를 계정 B의 Amazon SNS 주제에 전송합니다.

AWS 계정	계정 이름	설명
1111-1111-1111	계정 A	규칙 작업: `sns:Publish`
2222-2222-2222	계정 B	Amazon SNS 주제 ARN: *arn:aws:sns:region:2222-2222-2222:ExampleTopic*

계정 A 태스크 수행

> **참고**
>
> 다음 명령을 실행하려면 규칙 ARN을 리소스로 사용하는 `iot:CreateTopicRule`에 대한 권한과 리소스를 역할 ARN으로 사용하는 `iam:PassRole` 작업에 대한 권한이 IAM 사용자에게 있어야 합니다.

1. 계정 A의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. AWS IoT 규칙 엔진을 신뢰하는 IAM 역할을 생성하고 계정 B의 Amazon SNS 주제에 대한 액세스를 허용하는 정책을 연결합니다. 예제 명령 및 정책 문서는 필요한 액세스 권한 부여 AWS IoT 를 참조하세요.
3. 주제에 연결된 규칙을 생성하려면 create-topic-rule 명령을 실행합니다.

```
aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
```

다음은 `iot/test` 주제에 전송된 모든 메시지를 지정된 Amazon SNS 주제에 삽입하는 규칙이 있는 예제 페이로드 파일입니다. SQL 문은 메시지를 필터링하고 역할 ARN은 Amazon SNS 주제에 메시지를 전송할 수 있는 권한을 AWS IoT 에 부여합니다.

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sns": {
    "targetArn": "arn:aws:sns:region:2222-2222-2222:ExampleTopic",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
   }
  }
 ]
}
```

AWS IoT 규칙에서 Amazon SNS 작업을 정의하는 방법에 대한 자세한 내용은 AWS IoT 규칙 작업 - Amazon SNS를 참조하세요.

계정 B 태스크 수행

1. 계정 B의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. Amazon SNS 주제 리소스에 대한 권한을 계정 A에 부여하려면 add-permission 명령을 실행합니다.

```
aws sns add-permission --topic-arn arn:aws:sns:region:2222-2222-2222:ExampleTopic
 --label Publish-Permission --aws-account-id 1111-1111-1111 --action-name Publish
```

Amazon S3를 위한 교차 계정 설정

시나리오: 계정 A는 MQTT 메시지의 데이터를 계정 B의 Amazon S3 버킷에 전송합니다.

AWS 계정	계정 이름	설명
1111-1111-1111	계정 A	규칙 작업: `s3:PutObject`
2222-2222-2222	계정 B	Amazon S3 버킷 ARN: *arn:aws:s3:::amzn-s3-demo-bucket*

계정 A 태스크 수행

> **Note**
>
> 다음 명령을 실행하려면 규칙 ARN을 리소스로 사용하는 `iot:CreateTopicRule`에 대한 권한과 리소스를 역할 ARN으로 사용하는 `iam:PassRole` 작업에 대한 권한이 IAM 사용자에게 있어야 합니다.

1. 계정 A의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. AWS IoT 규칙 엔진을 신뢰하고 계정 B의 Amazon S3 버킷에 대한 액세스를 허용하는 정책을 연결하는 IAM 역할을 생성합니다. 예제 명령 및 정책 문서는 필요한 액세스 권한 부여 AWS IoT 를 참조하세요.
3. 대상 S3에 연결된 규칙을 생성하려면 create-topic-rule 명령을 실행합니다.

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
```

다음 예제는 `iot/test` 주제에 전송된 모든 메시지를 지정된 Amazon S3 버킷에 삽입하는 규칙을 포함한 페이로드 파일입니다. SQL 문은 메시지를 필터링하고 역할 ARN은 Amazon S3 버킷에 메시지를 추가할 수 있는 권한을 AWS IoT 에 부여합니다.

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "s3": {
    "bucketName": "amzn-s3-demo-bucket",
    "key": "${topic()}/${timestamp()}",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
   }
  }
 ]
}
```

AWS IoT 규칙에서 Amazon S3 작업을 정의하는 방법에 대한 자세한 내용은 AWS IoT 규칙 작업 - Amazon S3를 참조하세요.

계정 B 태스크 수행

1. 계정 B의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. 계정 A의 보안 주체를 신뢰하는 버킷 정책을 만듭니다.

   다음 예제는 다른 계정의 보안 주체를 신뢰하는 버킷 정책을 정의하는 페이로드 파일입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AddCannedAcl",
            "Effect": "Allow",
```

```
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:root"
                ]
        },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

자세한 정보는 버킷 정책 예제 단원을 참조하세요.

3. 버킷 정책을 지정된 버킷에 연결하려면 put-bucket-policy 명령을 실행합니다.

```
aws s3api put-bucket-policy --bucket amzn-s3-demo-bucket --policy file://./amzn-s3-demo-bucket-policy.json
```

4. 교차 계정 액세스가 작동하도록 하려면 모든 퍼블릭 액세스 차단(Block all public access) 설정이 올바른지 확인하세요. 자세한 내용은 Amazon S3 보안 모범 사례를 참조하세요.

에 대한 교차 계정 설정 AWS Lambda

시나리오: 계정 A는 계정 B의 AWS Lambda 함수를 호출하여 MQTT 메시지를 전달합니다.

AWS 계정	계정 이름	설명
1111-1111-1111	계정 A	규칙 작업: `lambda:InvokeFunction`
2222-2222-2222	계정 B	Lambda 함수 ARN: *arn:aws:lambda:region:2222-2222-2222:function:example-function*

계정 A 태스크 수행

> **참고**
>
> 다음 명령을 실행하려면 규칙 ARN을 리소스로 사용하는 `iot:CreateTopicRule`에 대한 권한과 리소스를 역할 ARN으로 사용하는 `iam:PassRole` 작업에 대한 권한이 IAM 사용자에게 있어야 합니다.

1. 계정 A의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.
2. create-topic-rule 명령을 실행하여 계정 B의 Lambda 함수에 대한 교차 계정 액세스를 정의하는 규칙을 생성합니다.

   ```
   aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
   ```

   다음 예제는 `iot/test` 주제에 전송된 모든 메시지를 지정된 Lambda 함수에 삽입하는 규칙을 포함한 페이로드 파일입니다. SQL 문은 메시지를 필터링하고 역할 ARN은 Lambda 함수에 데이터를 전달할 수 있는 권한을 AWS IoT 에 부여합니다.

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
     {
      "lambda": {
       "functionArn": "arn:aws:lambda:region:2222-2222-2222:function:example-function"
      }
     }
    ]
   }
   ```

   규칙에서 AWS Lambda 작업을 정의하는 방법에 대한 자세한 내용은 AWS IoT 규칙 작업 - Lambda를 AWS IoT 참조하십시오.

계정 B 태스크 수행

1. 계정 B의 IAM 사용자를 사용하여 AWS CLI를 구성합니다.

2. Lambda의 add-permission 명령을 실행하여 Lambda 함수를 활성화할 수 있는 권한을 AWS IoT 규칙에 부여합니다. 다음 명령을 실행하려면 IAM 사용자에게 `lambda:AddPermission` 작업에 대한 권한이 있어야 합니다.

```
aws lambda add-permission --function-name example-function --region us-east-1 --
principal iot.amazonaws.com --source-arn arn:aws:iot:region:1111-1111-1111:rule/
example-rule --source-account 1111-1111-1111 --statement-id "unique_id" --action
 "lambda:InvokeFunction"
```

옵션:

--principal

이 필드는 AWS IoT (로 표시됨`iot.amazonaws.com`)에 Lambda 함수를 호출할 수 있는 권한을 부여합니다.

--source-arn

이 필드는 AWS IoT 의 `arn:aws:iot:region:1111-1111-1111:rule/example-rule`만 이 Lambda 함수를 트리거하고 동일하거나 다른 계정의 다른 규칙이 이 Lambda 함수를 활성화할 수 없도록 확정합니다.

--source-account

이 필드는이 `1111-1111-1111` 계정을 대신하여이 Lambda 함수를 AWS IoT 활성화함을 확인합니다.

> **참고**
>
> 구성 아래의 AWS Lambda 함수 콘솔에서 "규칙을 찾을 수 없습니다"라는 오류 메시지가 표시되면 오류 메시지를 무시하고 연결 테스트를 진행하세요.

오류 처리(오류 작업)

가 디바이스에서 메시지를 AWS IoT 수신하면 규칙 엔진은 메시지가 규칙과 일치하는지 확인합니다. 일치할 경우에는 규칙의 쿼리 문을 평가하고, 규칙의 작업을 활성화한 다음 쿼리 문의 결과를 전달합니다.

작업 활성화 시 문제가 발생하면 해당 규칙에 지정되어 있는 경우에 한해 규칙 엔진이 오류 작업을 활성화합니다. 다음과 같은 경우가 이에 해당합니다.

- Amazon S3 버킷에 액세스할 수 있는 권한이 규칙에 없는 경우.
- 사용자 실수로 DynamoDB 프로비저닝 처리량을 초과하는 경우.

> **Note**
>
> 이 주제에서 다루는 오류 처리는 규칙 작업을 위한 것입니다. 외부 함수를 포함한 SQL 문제를 디버깅하려면 AWS IoT 로깅을 설정할 수 있습니다. 자세한 내용은 ??? 단원을 참조하십시오.

오류 작업 메시지 형식

규칙마다 생성되는 메시지는 1개입니다. 예를 들어 동일한 규칙에서 2개의 규칙 작업이 오류로 중단되더라도 오류 작업은 오류 2개가 모두 포함된 메시지 1개를 수신합니다.

오류 작업 메시지는 다음 예제와 같습니다.

```
{
  "ruleName": "TestAction",
  "topic": "testme/action",
  "cloudwatchTraceId": "7e146a2c-95b5-6caf-98b9-50e3969734c7",
  "clientId": "iotconsole-1511213971966-0",
  "base64OriginalPayload":
 "ewogICJtZXNzYWdlIjogIkhlbGxvIHZyb20gQVdTIElvVCBjb25zb2xlIgp9",
  "failures": [
    {
      "failedAction": "S3Action",
      "failedResource": "us-east-1-s3-verify-user",
      "errorMessage": "Failed to put S3 object. The error received was The
 specified bucket does not exist (Service: Amazon S3; Status Code: 404; Error
 Code: NoSuchBucket; Request ID: 9DF5416B9B47B9AF; S3 Extended Request ID:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y=).
 Message arrived on: error/action, Action: s3, Bucket: us-
east-1-s3-verify-user, Key: \"aaa\". Value of x-amz-id-2:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y="
    }
  ]
}
```

ruleName

오류 작업을 트리거한 규칙의 이름입니다.

주제

최초 메시지가 수신된 주제입니다.

cloudwatchTraceId

CloudWatch의 오류 로그를 참조하는 고유 ID입니다.

clientId

메시지 게시자의 클라이언트 ID입니다.

base64OriginalPayload

Base64로 인코딩된 최초의 메시지 페이로드입니다.

failures

failedAction

오류로 중단된 작업의 이름입니다(예: "S3Action").

failedResource

리소스 이름입니다(예: S3 버킷 이름).

errorMessage

오류에 대한 설명입니다.

오류 작업 예제

아래는 추가된 오류 작업을 포함한 규칙 예제입니다. 이 규칙에는 메시지 데이터를 DynamoDB 테이블에 기록하는 작업과 데이터를 Amazon S3 버킷에 기록하는 오류 작업이 있습니다.

```
{
    "sql" : "SELECT * FROM ..."
    "actions" : [{
        "dynamoDB" : {
            "table" : "PoorlyConfiguredTable",
            "hashKeyField" : "AConstantString",
```

```
            "hashKeyValue" : "AHashKey"}}
    ],
    "errorAction" : {
        "s3" : {
            "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
            "bucketName" : "message-processing-errors",
            "key" : "${replace(topic(), '/', '-') + '-' + timestamp() + '-' +
 newuuid()}"
        }
    }
}
```

, aws_lambda(), , , , , , 등의 외부 함수를 포함하여 오류 작업의 SQL 문에서 모든 함수 또는 대체 템플릿을 사용할 수 get_dynamodb() get_registry_data() get_thing_shadow() get_secret() machinelearning_predict()있습니다decode(). 오류 작업에서 외부 함수를 호출해야 하는 경우 오류 작업을 직접적으로 호출하면 외부 함수에 대한 추가 요금이 부과될 수 있습니다.

규칙 및 오류 작업을 지정하는 방법에 대한 자세한 내용은 AWS IoT 규칙 생성을 참조하세요.

CloudWatch를 사용하여 규칙의 성공 또는 실패 여부를 모니터링하는 방법에 대한 자세한 내용은 AWS IoT 지표 및 차원 단원을 참조하세요.

Basic Ingest를 통한 메시징 비용 절감

기본 수집을 사용하면 AWS IoT 규칙 작업에서 지원하는 AWS 서비스로 디바이스 데이터를 안전하게 전송할 수 있으며, 이때 메시징 요금이 발생하지 않습니다. 기본 수집은 수집 경로에서 게시/구독 메시지 브로커를 제거해 데이터 흐름을 최적화합니다.

기본 수집은 디바이스 또는 애플리케이션에서 메시지를 전송할 수 있습니다. 메시지에는 처음 3개 수준에 대한 $aws/rules/*rule_name*으로 시작하는 주제 이름이 있으며, 여기서 *rule_name*은 호출하려는 AWS IoT 규칙의 이름입니다.

일반적으로 규칙을 호출하는 데 사용하는 메시지 주제에 기본 수집 접두사($aws/rules/*rule_name*)를 추가하면 기본 수집과 함께 기존 규칙을 사용할 수 있습니다. 예를 들어, Buildings/Building5/Floor2/Room201/Lights("sql": "SELECT * FROM 'Buildings/#'")와 같은 주제가 포함된 메시지로 호출되는 BuildingManager라는 규칙이 있는 경우 $aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights 주제가 포함된 메시지를 전송해 기본 수집과 함께 동일한 규칙을 호출할 수 있습니다.

> **Note**
>
> - 디바이스 및 규칙은 Basic Ingest 예약 주제를 구독할 수 없습니다. 예를 들어 AWS IoT Device Defender 지표 `num-messages-received` 지표는 주제 구독을 지원하지 않으므로 내보내지지 않습니다. 자세한 내용은 예약된 주제 단원을 참조하십시오.
> - 메시지를 여러 구독자에게 배포하기 위해(예: 메시지를 다른 디바이스와 규칙 엔진으로 전달하기 위해) 게시/구독 브로커가 필요한 경우 계속해서 AWS IoT 메시지 브로커를 사용해 메시지 배포를 처리해야 합니다. 그러나 기본 수집 주제 이외의 주제에 대해 메시지를 게시해야 합니다.

Basic Ingest 사용

기본 수집을 사용하기 전에 디바이스 또는 애플리케이션이 `$aws/rules/*`에 대한 게시 권한이 있는 정책을 사용하는지 확인하세요. 또는 정책에서 `$aws/rules/`*`rule_name`*`/*`을(를) 사용해 개별 규칙에 대한 권한을 지정할 수 있습니다. 그렇지 않은 경우 디바이스 및 애플리케이션은 계속해서 AWS IoT Core와의 기존 연결을 사용할 수 있습니다.

메시지가 규칙 엔진에 도달하면 기본 수집에서 호출된 규칙과 메시지 브로커 구독을 통해 호출된 규칙 간에 구현 또는 오류 처리에 아무런 차이가 없습니다.

Basic Ingest와 함께 사용할 규칙을 생성할 수 있습니다. 다음 사항에 유의하세요.

- Basic Ingest 주제(`$aws/rules/`*`rule_name`*)의 최초 접두사는 topic(Decimal) 함수에 사용할 수 없습니다.
- 기본 수집으로만 호출되는 규칙을 정의하는 경우 `FROM` 절은 `rule` 정의의 `sql` 필드에서 선택적입니다. 이는 (예를 들어, 다른 메시지를 여러 구독자에게 배포해야 하기 때문에) 메시지 브로커를 통해 전송해야 하는 다른 메시지로도 해당 규칙이 호출될 경우에도 필요합니다. 자세한 내용은 AWS IoT SQL 참조 단원을 참조하십시오.
- Basic Ingest 주제(`$aws/rules/`*`rule_name`*)의 처음 3개 수준은 주제에 대한 8개 세그먼트 길이 제한 또는 256자의 총 문자 제한에 포함되지 않습니다. 그렇지 않으면 AWS IoT 제한의 설명과 동일한 제한이 적용됩니다.
- 비활성 규칙 또는 존재하지 않는 규칙을 지정하는 Basic Ingest 주제가 포함된 메시지가 수신되면 디버깅할 수 있도록 Amazon CloudWatch 로그에 오류 로그가 생성됩니다. 자세한 내용은 Rules engine 로그 항목 단원을 참조하세요. `RuleNotFound` 측정치가 표시되어 이 측정치에 대한 경보를 생성할 수 있습니다. 자세한 내용은 규칙 지표의 규칙 지표를 참조하세요.

- QoS 1을 사용해 Basic Ingest 주제에 대해 게시할 수 있습니다. 메시지가 규칙 엔진으로 성공적으로 전달되면 PUBACK이 수신됩니다. PUBACK을 수신했다고 해서 규칙 작업이 성공적으로 완료된 것은 아닙니다. 작업이 실행될 때 오류를 처리하도록 오류 작업을 구성할 수 있습니다. 자세한 내용은 오류 처리(오류 작업) 단원을 참조하십시오.

AWS IoT SQL 참조

에서 AWS IoT규칙은 SQL과 유사한 구문을 사용하여 정의됩니다. SQL 문은 세 유형의 절로 구성됩니다.

SET

(선택 사항) SQL 문 및 대체 템플릿 전체에서 재사용할 수 있는 변수를 정의합니다. 표현식을 사용하여 변수에 값을 할당합니다. SELECT 및 WHERE 절과 작업 대체 템플릿에서 이러한 변수를 참조하세요.

SET 절은 데이터 타입, , 연산자, 함수, 리터럴Case 문, JSON 확장, 변수 및를 지원합니다중첩된 객체 쿼리.

SELECT

(필수 사항) 수신 메시지 페이로드에서 정보를 추출하고 정보 변환을 수행합니다. 사용할 메시지는 FROM 절에서 지정한 주제 필터에 의해 식별됩니다.

SELECT 절은 데이터 타입, , 연산자, 함수리터럴, Case 문, JSON 확장, 대체 템플릿, 변수, 중첩된 객체 쿼리및를 지원합니다이진 페이로드.

FROM

데이터를 추출할 메시지를 식별하는 MQTT 메시지 주제 필터입니다. 여기에 지정된 주제 필터와 일치하는 MQTT 주제로 전송된 각 메시지가 규칙을 활성화합니다. 메시지 브로커를 통과하는 메시지가 활성화하는 규칙에서는 필수입니다. 기본 수집 기능을 사용해서만 활성화되는 규칙에서는 선택 사항입니다.

WHERE

(선택 사항) 규칙에서 지정된 작업이 수행되었는지 여부를 확인하는 조건부 논리를 추가합니다.

WHERE 절은 데이터 타입, , 연산자, 함수, 리터럴Case 문, JSON 확장, 변수 및를 지원합니다중첩된 객체 쿼리.

다음은 SQL 문 예제입니다.

```
SELECT color AS rgb FROM 'topic/subtopic' WHERE temperature > 50
```

다음은 MQTT 메시지(수신 페이로드라고도 함) 예제입니다.

```
{
    "color":"red",
    "temperature":100
}
```

이 메시지가 'topic/subtopic' 주제에 게시될 경우 규칙이 트리거되고 SQL 문이 실행됩니다. "temperature" 속성이 50을 초과할 경우 SQL 문이 color 속성의 값을 추출합니다. WHERE 절은 조건 temperature > 50을 지정합니다. AS 키워드는 "color" 속성의 이름을 "rgb"로 변경합니다. 그 결과(송신 페이로드)는 다음과 같습니다.

```
{
    "rgb":"red"
}
```

그런 다음 이 데이터가 규칙의 작업으로 전달되고, 작업이 추가 처리를 위해 데이터를 전송합니다. 규칙 작업에 대한 자세한 내용은 AWS IoT 규칙 작업 섹션을 참조하세요.

> **Note**
>
> 주석은 현재 AWS IoT SQL 구문에서 지원되지 않습니다.
> 공백이 포함된 속성 이름은 SQL 문에서 필드 이름으로 사용할 수 없습니다. 수신되는 페이로드는 공백이 있는 속성 이름을 가질 수 있지만 SQL 문에서는 이러한 이름을 사용할 수 없습니다. 그러나 와일드카드(*) 필드 이름 사양을 사용하는 경우 송신하는 페이로드로 전달됩니다.

SELECT 절

AWS IoT SELECT 절은 기본적으로 ANSI SQL SELECT 절과 동일하지만 약간의 차이가 있습니다.

SELECT 절은 데이터 타입, , 연산자, 함수리터럴, Case 문, JSON 확장, 변수, 중첩된 객체 쿼리및를 지원합니다이진 페이로드.

SELECT 절을 사용하여 수신 MQTT 메시지에서 정보를 추출할 수 있습니다. SELECT *를 사용하여 전체 수신 메시지 페이로드를 검색할 수도 있습니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT * FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50}
```

페이로드가 JSON 객체일 경우 객체 내 키를 참조할 수 있습니다. 송신 페이로드에는 키-값 페어가 포함됩니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT color FROM 'topic/subtopic'
Outgoing payload: {"color":"red"}
```

AS 키워드를 사용하여 키 이름을 바꿀 수 있습니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic':{"color":"red", "temperature":50}
SQL:SELECT color AS my_color FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red"}
```

쉼표로 구분하여 여러 항목을 선택할 수 있습니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT color as my_color, temperature as fahrenheit FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red","fahrenheit":50}
```

'*'를 포함하여 여러 항목을 선택하여 수신 페이로드에 항목을 추가할 수 있습니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT *, 15 as speed FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50, "speed":15}
```

"VALUE" 키워드를 사용하여 JSON 객체가 아닌 송신 페이로드를 생성할 수 있습니다. SQL 버전 2015-10-08에서는 한 항목만 선택할 수 있습니다. SQL 버전 2016-03-23 이상에서는 최상위 객체로 출력할 배열을 선택할 수도 있습니다.

Example

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT VALUE color FROM 'topic/subtopic'
Outgoing payload: "red"
```

'.' 구문을 사용하여 수신 페이로드에서 중첩된 JSON 객체를 자세히 확인할 수 있습니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic': {"color":
{"red":255,"green":0,"blue":0}, "temperature":50}
SQL: SELECT color.red as red_value FROM 'topic/subtopic'
Outgoing payload: {"red_value":255}
```

숫자 또는 하이픈(-) 문자와 같은 예약된 문자가 포함된 JSON 객체 및 속성 이름을 사용하는 방법에 대한 자세한 내용은 JSON 확장 단원을 참조하세요.

함수(함수 섹션 참조)를 사용하여 수신 페이로드를 변환할 수 있습니다. 그룹화에 괄호를 사용할 수 있습니다. 다음 예를 참조하세요.

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT (temperature - 32) * 5 / 9 AS celsius, upper(color) as my_color FROM
 'topic/subtopic'
Outgoing payload: {"celsius":10,"my_color":"RED"}
```

FROM 절

FROM 절은 규칙이 주제 또는 주제 필터를 구독하도록 지정합니다. 주제 또는 주제 필터를 작은따옴표(')로 묶으세요. 여기에 지정된 주제 필터와 일치하는 MQTT 주제로 전송된 각 메시지가 규칙을 트리거합니다. 주제 필터를 사용하여 비슷한 주제의 그룹을 구독할 수 있습니다.

예:

주제 'topic/subtopic'에 게시된 수신 페이로드: {temperature: 50}

주제 'topic/subtopic-2'에 게시된 수신 페이로드: {temperature: 50}

SQL: "SELECT temperature AS t FROM 'topic/subtopic'".

규칙이 'topic/subtopic'를 구독하므로 수신 페이로드가 규칙에 전달됩니다. 규칙 작업에 전달되는 송신 페이로드는 {t: 50}입니다. 규칙이 'topic/subtopic-2'를 구독하지 않으며, 따라서 'topic/subtopic-2'에 게시된 메시지가 규칙을 트리거하지 않습니다.

와일드카드 예:

'#'(다중 레벨) 와일드카드 문자를 사용하여 하나 이상의 특정 경로 요소와 일치시킬 수 있습니다.

주제 'topic/subtopic'에 게시된 수신 페이로드: {temperature: 50}

주제 'topic/subtopic-2'에 게시된 수신 페이로드: {temperature: 60}

주제 'topic/subtopic-3/details'에 게시된 수신 페이로드: {temperature: 70}

주제 'topic-2/subtopic-x'에 게시된 수신 페이로드: {temperature: 80}

SQL: "SELECT temperature AS t FROM 'topic/#'".

규칙이 'topic'으로 시작하는 모든 주제를 구독하므로 세 번 실행되어 {t: 50}(topic/subtopic), {t: 60}(topic/subtopic-2) 및 {t: 70}(topic/subtopic-3/details)의 송신 페이로드를 해당 작업으로 전송합니다. 규칙이 'topic-2/subtopic-x'를 구독하지 않으며, 따라서 {temperature: 80} 메시지가 규칙을 트리거하지 않습니다.

+ 와일드카드 예:

'+'(단일 레벨) 와일드카드 문자를 사용하여 어느 하나의 특정 경로 요소와 일치시킬 수 있습니다.

주제 'topic/subtopic'에 게시된 수신 페이로드: {temperature: 50}

주제 'topic/subtopic-2'에 게시된 수신 페이로드: {temperature: 60}

주제 'topic/subtopic-3/details'에 게시된 수신 페이로드: {temperature: 70}

주제 'topic-2/subtopic-x'에 게시된 수신 페이로드: {temperature: 80}

SQL: "SELECT temperature AS t FROM 'topic/+'".

규칙이 첫 번째 요소가 'topic'인 2개의 경로 요소를 갖는 모든 주제를 구독합니다. 규칙은 'topic/subtopic' 및 'topic/subtopic-2'에 전송된 메시지에 대해 실행되지만, 'topic/subtopic-3/details'(주제(topic) 필터보다 더 많은 레벨이 있음) 또는 'topic-2/subtopic-x'(topic으로 시작하지 않음)에 전송된 메시지에 대해서는 실행되지 않습니다.

SET 절

SET 절을 사용하여 표현식 결과를 저장하는 변수를 정의합니다. SELECT 및 WHERE 절과 대체 템플릿에서 이러한 변수를 재사용할 수 있습니다. 이렇게 하면 복잡한 표현식을 복제하지 않고 SQL 문에서 함수 호출 수를 줄일 수 있습니다.

SET 절은 데이터 타입, , 연산자, 함수, 리터럴Case 문, JSON 확장, 변수 및를 지원합니다중첩된 객체 쿼리.

SET 절 구문

SET 절은 SQL 문의 SELECT 절 앞에 나타나야 합니다. 다음 구문을 사용합니다.

```
SET @variable_name = expression [, @variable_name2 = expression2]
```

구문 규칙:

- 를 사용하여 변수 이름 시작 @
- 변수 이름에는 문자, 숫자 및 밑줄이 포함될 수 있습니다.
- 변수 이름은 최대 64자까지 가능합니다.
- 쉼표로 구분된 단일 SET 절에서 여러 변수를 설정할 수 있습니다.
- 각 변수는 한 번만 할당할 수 있습니다(변수는 변경할 수 없음).
- SET 키워드는 SQL 문당 한 번만 사용할 수 있습니다.

변수 사용

변수를 정의한 후 다음에서 사용할 수 있습니다.

- SELECT 절
- WHERE 절
- 기타 SET 변수 할당
- 작업 대체 템플릿
- 오류 작업 대체 템플릿
- 중첩된 SELECT 쿼리
- 함수 파라미터(roleArn 파라미터와 같은 특정 파라미터 및와 유사한 함수 모드를 전환하는 파라미터는 변수를 지원하지 `transform("enrichArray", attributes, values)` 않음)

변수는 `@variable_name` SET 절에 사용된 것과 동일한 구문을 사용하여 참조됩니다.
JSON 확장 구문을 사용하여와 같이 객체가 포함된 변수의 속성에 액세스할 수도 있습니다`@variable_name.property`.

SET 절 예제

기본 변수 사용

다음 예제는 주제에 게시된 페이로드를 보여줍니다device/data. {"temp_fahrenheit": 75, "humidity": 60}

SQL 문

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

발신 페이로드: {"celsius": 23.89, "humidity": 60}

임베디드 JSON 객체의 멤버 액세스

다음 예제는 주제에 게시된 페이로드를 보여줍니다device/data. {"device1": {"deviceId":"weather_sensor", "deviceData": {"sensors": {"temp_fahrenheit": 75, "humidity": 60}, "location": [47.606,-122.332]}}}

SQL 문

```
SET @device_sensor_data = device1.deviceData.sensors
SELECT @device_sensor_data.temp_fahrenheit AS temp_fahrenheit,
 @device_sensor_data.humidity as humidity, device1.deviceId as deviceId FROM 'device/
data'
```

발신 페이로드:
{"temp_fahrenheit":75,"humidity":60,"deviceId":"weather_sensor"}

JSON 확장을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요. JSON 확장

중복 함수 호출 방지

SET 변수는 복잡한 디코딩 작업의 중복을 방지하는 데 도움이 됩니다.

```
SET @decoded_data = decode(encode(*, 'base64'), 'proto', 'schema', 'schema.desc',
 'message.proto', 'Message')
SELECT @decoded_data.sensor_id, @decoded_data.reading FROM 'device/protobuf'
WHERE @decoded_data.reading > 100
```

SET 변수가 없으면 디코딩 함수를 세 번 반복해야 하며, 이는 함수 호출 제한을 초과합니다.

여러 변수

쉼표로 구분하여 단일 SET 절에서 여러 변수를 정의할 수 있습니다.

```
SET @user_data = get_user_properties(device_id), @threshold = 50
SELECT @user_data.name, temp_fahrenheit FROM 'sensors/+'
WHERE temp_fahrenheit > @threshold AND @user_data.active = true
```

대체 템플릿에서 변수 사용

변수는 작업 대체 템플릿에도 사용할 수 있으므로 SQL 문과 규칙 작업 모두에서 계산된 값을 재사용할 수 있습니다.

SQL 문

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

작업 구성:

```
{
  "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/testRuleRole",
    "bucketName": "bucket",
    "key": "temperature-data/${device_id}/temp-${@temp_celsius}C.json"
  }
}
```

이 예제에서는 대체 템플릿에서 SET 변수를 `@temp_celsius` 사용하여 S3 작업의 키 필드를 구성합니다.

비 JSON 페이로드 사용량

SET 변수는 비 JSON 페이로드를 직접 지원하지 않으므로 먼저 페이로드를 인코딩하거나 디코딩해야 합니다.

```
SET @encoded_payload = encode(*, 'base64')
SELECT @encoded_payload AS raw_data FROM 'device/binary'
```

비 JSON 페이로드 작업 방법에 대한 자세한 내용은 섹션을 참조하세요. 이진 페이로드 작업

SET 절 제한

SET 변수에는 다음 제한이 적용됩니다.

- SQL 문당 최대 10개의 고유 변수
- 최대 변수 값 크기 128KiB(최소 UTF-8 JSON 문자열)
- 모든 변수에 대해 128KiB의 최대 총 값 크기
- 변수 이름은 64자로 제한됩니다.
- 변수는 JSON 페이로드를 그대로 직접 수락할 수 있습니다(JSON이 아닌 페이로드를 먼저 인코딩/디코딩해야 함).

WHERE 절

WHERE 절은 규칙에서 지정된 작업이 수행되었는지 여부를 확인합니다. WHERE 절이 true로 평가될 경우 규칙 작업이 수행됩니다. 그렇지 않으면 규칙 작업이 수행되지 않습니다.

WHERE 절은 데이터 타입, , 연산자, 함수, 리터럴Case 문, JSON 확장, 변수 및를 지원합니다중첩된 객체 쿼리.

예:

`topic/subtopic`에 게시된 수신 페이로드: `{"color":"red", "temperature":40}`

SQL: `SELECT color AS my_color FROM 'topic/subtopic' WHERE temperature > 50 AND color <> 'red'`.

이 경우 규칙이 트리거되지만 규칙에서 지정된 작업은 수행되지 않습니다. 송신 페이로드가 없습니다.

WHERE 절에서 함수 및 연산자를 사용할 수 있습니다. 하지만 SELECT 절의 AS 키워드를 사용하여 생성된 별칭은 참조할 수 없습니다. SELECT 절 평가 여부를 결정하기 위해 WHERE 절이 먼저 평가됩니다.

비 JSON 페이로드 예제:

'topic/subtopic'에 게시된 들어오는 비 JSON 페이로드: '80'

SQL: `` `SELECT decode(encode(*, 'base64'), 'base64') AS value FROM 'topic/subtopic' WHERE decode(encode(*, 'base64'), 'base64') > 50 ``

이 경우 규칙이 트리거되고 규칙에서 지정된 작업이 수행됩니다. 나가는 페이로드는 SELECT 절에 의해 JSON 페이로드 `{"value":80}`으로 변환됩니다.

데이터 타입

AWS IoT 규칙 엔진은 모든 JSON 데이터 형식을 지원합니다.

지원되는 데이터 유형

Type	의미
`Int`	이산 `Int`. 최대 34자리.
`Decimal`	정밀도 34자리, 0이 아닌 최소 크기 1E-999, 최대 크기 9.999...E999의 `Decimal` **Note** 일부 함수는 34자리 정밀도 대신 배정밀도의 `Decimal`을 반환합니다. SQL V2(2016-03-23)에서 `10.0`와 (과) 같은 정수인 숫자 값은 예상되는 `Decimal` 값(`10.0`) 대신 `Int` 값(`10`)으로 처리됩니다. 정수 숫자 값을 `Decimal` 값으로 안정적으로 처리하려면 규칙 쿼리 문에 SQL V1(2015-10-08)을 사용하세요.
`Boolean`	`True` 또는 `False`
`String`	UTF-8 문자열
`Array`	동일한 형식일 필요가 없는 일련의 값
`Object`	키 및 값으로 구성된 JSON 값입니다. 키는 문자열이어야 합니다. 값은 임의의 형식일 수 있습니다.
`Null`	JSON에 의해 정의된 `Null`입니다. 이는 값 없음을 나타내는 실제 값입니다. SQL 문에서 `Null` 키워드를 사용하여 명시적으로 `Null` 값을 생성할 수 있습니다. 예: `"SELECT NULL AS n FROM 'topic/subtopic'"`
`Undefined`	값이 아닙니다. 값을 생략하는 것 이외에는 JSON에서 이를 명시적으로 표현할 수 없습

Type	의미

	니다. 예를 들어 객체 {"foo": null}에서 키 "foo"는 NULL을 반환하지만 키 "bar"는 Undefined 를 반환합니다. 내부적으로는 SQL 언어가 Undefined 를 값으로 취급하지만 JSON에서는 표현이 불가능합니다. 따라서 JSON으로 직렬화될 경우 결과는 Undefined 입니다.
	
	`{"foo":null, "bar":undefined}`
	
	다음과 같이 JSON으로 직렬화됩니다.
	
	`{"foo":null}`
	
	마찬가지로, Undefined 는 자체에 의해 직렬화될 경우 빈 문자열로 변환됩니다. 잘못된 인수(예: 잘못된 형식, 잘못된 인수 개수 등)로 호출된 함수는 Undefined 를 반환합니다.

변환

다음 표에는 (함수에 잘못된 형식의 값이 입력된 경우) 값이 한 형식에서 다른 형식으로 변환될 경우의 결과가 나와 있습니다. 예를 들어 절대값 함수 "abs"(Int 또는 Decimal이 필요함)에 String이 주어질 경우 이 함수는 다음 규칙에 따라 String을 Decimal로 변환하려고 시도합니다. 이 경우 'abs("-5.123")'는 'abs(-5.123)'으로 취급됩니다.

> **Note**
>
> Array, Object, Null 또는 Undefined로는 변환이 시도되지 않습니다.

소수로 변환

인수 유형	결과
`Int`	소수점이 없는 `Decimal`
`Decimal`	소스 값
`Boolean`	`Undefined` (명시적으로 cast 함수를 사용하여 true = 1.0, false = 0.0으로 변환할 수 있습니다.)
`String`	SQL 엔진은 문자열을 로 구문 분석하려고 시도합니다`Decimal`.는 정규식과 일치하는 문자열을 구문 분석하려고 AWS IoT 시도합니다`^-?\d+(\.\d+)?((?i)E-?\d+)?$` . "0", "-1.2", "5E-12"는 모두 자동으로 `Decimal`로 변환되는 문자열의 예입니다.
배열	`Undefined` .
객체	`Undefined` .
Null	`Null`.
정의되지 않음	`Undefined` .

정수로 변환

인수 유형	결과
`Int`	소스 값
`Decimal`	가장 가까운 `Int`로 반올림한 소스 값
`Boolean`	`Undefined` (명시적으로 cast 함수를 사용하여 true = 1.0, false = 0.0으로 변환할 수 있습니다.)
`String`	SQL 엔진은 문자열을 로 구문 분석하려고 시도합니다`Decimal`.는 정규식과 일치하는 문자열을 구문 분석하려고 AWS IoT 시도합니다`^-?\d`

인수 유형	결과
	`+(\.\d+)?((?i)E-?\d+)?$` . "0", "-1.2", "5E-12"는 모두 자동으로 `Decimals`. AWS IoT attempts로 변환되어를 `String`로 변환하는 문자열의 예`Decimal`입니다. 그런 다음 해당 문자열의 소수점을 잘라`Decimal`내어를 만듭니다`Int`.
배열	`Undefined` .
객체	`Undefined` .
Null	`Null`.
정의되지 않음	`Undefined` .

부울로 변환

인수 유형	결과
`Int`	`Undefined` (명시적으로 `cast` 함수를 사용하여 0 = False, 0이 아닌 값 = True로 변환할 수 있습니다.)
`Decimal`	`Undefined` (명시적으로 cast 함수를 사용하여 0 = False, 0이 아닌 값 = True로 변환할 수 있습니다.)
`Boolean`	원래 값
`String`	"true"=True, "false"=False(대/소문자를 구분하지 않음). 다른 문자열 값은 `Undefined` 입니다.
배열	`Undefined` .
객체	`Undefined` .

인수 유형	결과
Null	`Undefined` .
정의되지 않음	`Undefined` .

문자열로 변환

인수 유형	결과
`Int`	표준 표기법을 따른 `Int`의 문자열 표현.
`Decimal`	유효숫자 표기법을 따를 수 있는 `Decimal`의 문자열 표현.
`Boolean`	"true" 또는 "false". 모두 소문자.
`String`	원래 값
배열	JSON으로 직렬화된 `Array`입니다. 결과 문자열은 대괄호 안의 쉼표로 구분된 목록입니다. `String`은 따옴표로 묶입니다. `Decimal`, `Int`, `Boolean` 및 `Null`은 따옴표로 묶이지 않습니다.
객체	JSON으로 직렬화된 객체입니다. 결과 문자열은 키-값 페어의 쉼표로 구분된 목록이며 중괄호로 묶입니다. `String`은 따옴표로 묶입니다. `Decimal`, `Int`, `Boolean` 및 `Null`은 따옴표로 묶이지 않습니다.
Null	`Undefined` .
정의되지 않음	Undefined

연산자

SELECT 및 WHERE 절에서 다음 연산자를 사용할 수 있습니다.

AND 연산자

Boolean 결과를 반환합니다. 논리적 AND 연산을 수행합니다. 왼쪽 및 오른쪽 피연산자가 true일 경우 true를, 그렇지 않을 경우 false를 반환합니다. Boolean 피연산자 또는 대/소문자를 구분하지 않는 "true" 또는 "false" 문자열 피연산자가 필요합니다.

구문: *expression* AND *expression*.

AND 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Boolean	Boolean	Boolean. 양쪽 피연산자가 true일 경우 true입니다. 그렇지 않으면 false입니다.
String/Boolean	String/Boolean	모든 문자열이 "true" 또는 "false"(대/소문자를 구분하지 않음)일 경우 문자열은 Boolean로 변환된 후 정상적으로 *boolean* AND *boolean*로 처리됩니다.
기타 값	기타 값	Undefined .

OR 연산자

Boolean 결과를 반환합니다. 논리적 OR 연산을 수행합니다. 왼쪽 또는 오른쪽 피연산자가 true일 경우 true를, 그렇지 않을 경우 false를 반환합니다. 그렇지 않을 경우 false를 반환합니다. Boolean 피연산자 또는 대/소문자를 구분하지 않는 "true" 또는 "false" 문자열 피연산자가 필요합니다.

구문: *expression* OR *expression*.

OR 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Boolean	Boolean	Boolean. 어느 한 피연산자가 true일 경우 true입니다. 그렇지 않으면 false입니다.
String/Boolean	String/Boolean	모든 문자열이 "true" 또는 "false"(대/소문자를 구분하지 않음)인 경우 문자열은 부울로 변환되고 정상적으로 *boolean* OR *boolean*으로 처리됩니다.

왼쪽 피연산자	오른쪽 피연산자	출력
기타 값	기타 값	`Undefined` .

NOT 연산자

Boolean 결과를 반환합니다. 논리적 NOT 연산을 수행합니다. 피연산자가 false일 경우 true를, 그렇지 않은 경우 false를 반환합니다. Boolean 피연산자 또는 대/소문자를 구분하지 않는 "true" 또는 "false" 문자열 피연산자가 필요합니다.

구문: `NOT` *`expression`*.

NOT 연산자

피연산자	출력
`Boolean`	Boolean. 피연산자가 false일 경우 true입니다. 그렇지 않은 경우 true입니다.
`String`	문자열이 'true' 또는 'false'(대/소문자를 구분하지 않음)일 경우 문자열은 해당하는 부울 값으로 변환되고 반대 값이 반환됩니다.
기타 값	`Undefined` .

IN 연산자

Boolean 결과를 반환합니다. WHERE 절의 IN 연산자를 사용하여 값이 배열의 값과 일치하는지 확인할 수 있습니다. 일치 항목이 발견되면 true를 반환하고 그렇지 않으면 false를 반환합니다.

구문: *`expression`* `IN` *`expression`*.

IN 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
`Int/Decimal/String/A`	`Array`	배열에서 `Integer/Decimal/String/Array/Object` 요소가 발견되면 true입니다. 그렇지 않으면 false입니다.

예:

```
SQL: "select * from 'a/b' where 3 in arr"

JSON: {"arr":[1, 2, 3, "three", 5.7, null]}
```

이 예제에서는 arr이라는 배열에 3이 존재하므로 `where 3 in arr` 조건 절이 true로 평가됩니다. 따라서 SQL 문에서 `select * from 'a/b'`가 실행됩니다. 이 예제는 배열이 이기종일 수 있음을 보여줍니다.

EXISTS 연산자

Boolean 결과를 반환합니다. 조건부 절에서 EXISTS 연산자를 사용하여 하위 쿼리에 요소가 있는지 테스트할 수 있습니다. 하위 쿼리가 하나 이상의 요소를 반환하는 경우 true를 반환하고 하위 쿼리가 요소를 반환하지 않는 경우 false를 반환합니다.

구문: *expression*.

예:

```
SQL: "select * from 'a/b' where exists (select * from arr as a where a = 3)"

JSON: {"arr":[1, 2, 3]}
```

이 예제에서는 arr이라는 배열에 3이 존재하므로 `where exists (select * from arr as a where a = 3)` 조건 절이 true로 평가됩니다. 따라서 SQL 문에서 `select * from 'a/b'`가 실행됩니다.

예:

```
SQL: select * from 'a/b' where exists (select * from e as e where foo = 2)

JSON: {"foo":4,"bar":5,"e":[{"foo":1},{"foo":2}]}
```

이 예제에서는 JSON 객체 e 내의 배열에 객체 `{"foo":2}`가 포함되어 있기 때문에 `where exists (select * from e as e where foo = 2)` 조건 절이 true로 평가됩니다. 따라서 SQL 문에서 `select * from 'a/b'`가 실행됩니다.

> 연산자

Boolean 결과를 반환합니다. 왼쪽 피연산자가 오른쪽 피연산자보다 클 경우 true를 반환합니다. 두 피연산자 모두 Decimal로 변환된 후 비교됩니다.

구문: *expression* > *expression*.

> 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int/Decimal	Int/Decimal	Boolean. 왼쪽 피연산자가 오른쪽 피연산자보다 클 경우 true입니다. 그렇지 않으면 false입니다.
String/Int/Deci	String/Int/Deci	모든 문자열을 Decimal로 변환할 수 있는 경우 Boolean입니다. 왼쪽 피연산자가 오른쪽 피연산자보다 클 경우 true를 반환합니다. 그렇지 않으면 false입니다.
기타 값	Undefined .	Undefined .

>= 연산자

Boolean 결과를 반환합니다. 왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 경우 true, 그렇지 않을 경우 false를 반환합니다. 두 피연산자 모두 Decimal로 변환된 후 비교됩니다.

구문: *expression* >= *expression*.

>= 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int/Decimal	Int/Decimal	Boolean. 왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 경우 true입니다. 그렇지 않으면 false입니다.
String/Int/Deci	String/Int/Deci	모든 문자열을 Decimal로 변환할 수 있는 경우 Boolean입니다. 왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 경우 true, 그렇지 않을 경우 false를 반환합니다. 그렇지 않으면 false입니다.
기타 값	Undefined .	Undefined .

< 연산자

Boolean 결과를 반환합니다. 왼쪽 피연산자가 오른쪽 피연산자보다 작을 경우 true를 반환합니다. 두 피연산자 모두 Decimal로 변환된 후 비교됩니다.

구문: *expression* < *expression*.

< 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int/Decimal	Int/Decimal	Boolean. 왼쪽 피연산자가 오른쪽 피연산자보다 작을 경우 true입니다. 그렇지 않으면 false입니다.
String/Int/Deci	String/Int/Deci	모든 문자열을 Decimal로 변환할 수 있는 경우 Boolean입니다. 왼쪽 피연산자가 오른쪽 피연산자보다 작을 경우 true를 반환합니다. 그렇지 않으면 false입니다.
기타 값	Undefined	Undefined

<= 연산자

Boolean 결과를 반환합니다. 왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 경우 true, 그렇지 않을 경우 false를 반환합니다. 두 피연산자 모두 Decimal로 변환된 후 비교됩니다.

구문: *expression* <= *expression*.

<= 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int/Decimal	Int/Decimal	Boolean. 왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 경우 true입니다. 그렇지 않으면 false입니다.
String/Int/Deci	String/Int/Deci	모든 문자열을 Decimal로 변환할 수 있는 경우 Boolean입니다. 왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 경우 true, 그렇지 않을 경우 false를 반환합니다. 그렇지 않으면 false입니다.
기타 값	Undefined	Undefined

<> 연산자

Boolean 결과를 반환합니다. 왼쪽 및 오른쪽 피연산자가 다를 경우 true를, 그렇지 않을 경우 false를 반환합니다.

구문: *expression* <> *expression*.

<> 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	왼쪽 피연산자가 오른쪽 피연산자와 다를 경우 true입니다. 그렇지 않으면 false입니다.
Decimal	Decimal	왼쪽 피연산자가 오른쪽 피연산자와 다를 경우 true입니다. 그렇지 않으면 false입니다. Int은(는) Decimal(으)로 변환된 후 비교됩니다.
String	String	왼쪽 피연산자가 오른쪽 피연산자와 다를 경우 true입니다. 그렇지 않으면 false입니다.
배열	배열	각 피연산자의 항목이 같지 않고 동일한 순서가 아닐 경우 true입니다. 그렇지 않으면 false입니다.
객체	객체	각 피연산자의 키 및 값이 다를 경우 true입니다. 그렇지 않으면 false입니다. 키/값의 순서는 무시됩니다.
Null	Null	False.
임의의 값	Undefined	Undefined
Undefined	임의의 값	Undefined
일치하지 않은 형식	일치하지 않은 형식	True.

= 연산자

Boolean 결과를 반환합니다. 왼쪽 및 오른쪽 피연산자가 동일한 경우 true를, 그렇지 않을 경우 false를 반환합니다.

구문: *expression* = *expression*.

= 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	왼쪽 피연산자가 오른쪽 피연산자와 같을 경우 true입니다. 그렇지 않으면 false입니다.
Decimal	Decimal	왼쪽 피연산자가 오른쪽 피연산자와 같을 경우 true입니다. 그렇지 않으면 false입니다. Int은(는) Decimal(으)로 변환된 후 비교됩니다.
String	String	왼쪽 피연산자가 오른쪽 피연산자와 같을 경우 true입니다. 그렇지 않으면 false입니다.
배열	배열	각 피연산자의 항목이 같고 동일한 순서일 경우 true입니다. 그렇지 않으면 false입니다.
객체	객체	각 피연산자의 키 및 값이 같을 경우 true입니다. 그렇지 않으면 false입니다. 키/값의 순서는 무시됩니다.
임의의 값	Undefined	Undefined .
Undefined	임의의 값	Undefined .
일치하지 않은 형식	일치하지 않은 형식	False.

+ 연산자

"+"는 오버로드된 연산자입니다. 문자열 연결 또는 추가에 사용할 수 있습니다.

구문: *expression* + *expression*.

+ 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
String	임의의 값	오른쪽 피연산자를 문자열로 변환하여 왼쪽 피연산자의 끝에 연결합니다.

왼쪽 피연산자	오른쪽 피연산자	출력
임의의 값	String	왼쪽 피연산자를 문자열로 변환하고 오른쪽 피연산자를 변환된 왼쪽 피연산자의 끝에 연결합니다.
Int	Int	Int 값 피연산자를 함께 더합니다.
Int/Decimal	Int/Decimal	Decimal 값 피연산자를 함께 더합니다.
기타 값	기타 값	Undefined .

- 연산자

왼쪽 피연산자에서 오른쪽 피연산자를 뺍니다.

구문: *expression* - *expression*.

- 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	Int 값 왼쪽 피연산자에서 오른쪽 피연산자를 뺍니다.
Int/Decimal	Int/Decimal	Decimal 값 왼쪽 피연산자에서 오른쪽 피연산자를 뺍니다.
String/Int/Deci	String/Int/Deci	모든 문자열이 올바르게 10진수로 변환된 경우 Decimal 값이 반환됩니다. 왼쪽 피연산자에서 오른쪽 피연산자를 뺍니다. 그렇지 않은 경우 Undefined 를 반환합니다.
기타 값	기타 값	Undefined .
기타 값	기타 값	Undefined .

* 연산자

왼쪽 피연산자에 오른쪽 피연산자를 곱합니다.

구문: *expression* * *expression*.

* 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	Int 값 왼쪽 피연산자에 오른쪽 피연산자를 곱합니다.
Int/Decimal	Int/Decimal	Decimal 값 왼쪽 피연산자에 오른쪽 피연산자를 곱합니다.
String/Int/Deci	String/Int/Deci	모든 문자열이 올바르게 10진수로 변환된 경우 Decimal 값이 반환됩니다. 왼쪽 피연산자에 오른쪽 피연산자를 곱합니다. 그렇지 않은 경우 Undefined 를 반환합니다.
기타 값	기타 값	Undefined .

/ 연산자

왼쪽 피연산자를 오른쪽 피연산자로 나눕니다.

구문: *expression* / *expression*.

/ 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	Int 값 왼쪽 피연산자를 오른쪽 피연산자로 나눕니다.
Int/Decimal	Int/Decimal	Decimal 값 왼쪽 피연산자를 오른쪽 피연산자로 나눕니다.
String/Int/Deci	String/Int/Deci	모든 문자열이 올바르게 10진수로 변환된 경우 Decimal 값이 반환됩니다. 왼쪽 피연산자를 오른쪽 피연산자로 나눕니다. 그렇지 않은 경우 Undefined 를 반환합니다.
기타 값	기타 값	Undefined .

% 연산자

왼쪽 피연산자를 오른쪽 피연산자로 나눈 나머지를 반환합니다.

구문: *expression* % *expression*.

% 연산자

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	Int 값 왼쪽 피연산자를 오른쪽 피연산자로 나눈 나머지를 반환합니다.
String/Int/Deci	String/Int/Deci	모든 문자열이 올바르게 10진수로 변환된 경우 Decimal 값이 반환됩니다. 왼쪽 피연산자를 오른쪽 피연산자로 나눈 나머지를 반환합니다. 그렇지 않을 경우 Undefined 입니다.
기타 값	기타 값	Undefined .

함수

SQL 표현식의 SELECT 또는 WHERE 절에서 다음 내장 함수를 사용할 수 있습니다.

, aws_lambda, get_dynamodb() get_registry_data()및 외부 함수는 규칙 작업과 동일한 요금이 청구됩니다get_thing_shadow(). 또한 JSON에 대한 Protobuf 메시지를 디코딩할 때만 decode() 함수에 대한 요금이 청구됩니다. 자세한 내용은 AWS IoT Core pricing page를 참조하세요.

abs(Decimal)

숫자의 절대값을 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: abs(-5)= 5.

인수 유형	결과
Int	Int, 인수의 절대값
Decimal	Decimal, 인수의 절대값
Boolean	Undefined .

인수 유형	결과
String	Decimal. 결과는 인수의 절대값입니다. 문자열을 변환할 수 없는 경우 결과는 Undefined 입니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

accountid()

이 규칙을 소유하는 계정의 ID를 String으로 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

accountid() = "123456789012"

acos(Decimal)

숫자의 역코사인을 라디안 단위로 반환합니다. Decimal 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: acos(0) = 1.5707963267948966

인수 유형	결과
Int	Decimal(배정밀도), 인수의 역코사인. 가상 결과는 Undefined 로 반환됩니다.
Decimal	Decimal(배정밀도), 인수의 역코사인. 가상 결과는 Undefined 로 반환됩니다.
Boolean	Undefined .

인수 유형	결과
String	Decimal, 인수의 역코사인. 문자열을 변환할 수 없는 경우 결과는 Undefined 입니다. 가상 결과는 Undefined 로 반환됩니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

asin(Decimal)

숫자의 역사인을 라디안 단위로 반환합니다. Decimal 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: asin(0) = 0.0

인수 유형	결과
Int	Decimal(배정밀도), 인수의 역사인. 가상 결과는 Undefined 로 반환됩니다.
Decimal	Decimal(배정밀도), 인수의 역사인. 가상 결과는 Undefined 로 반환됩니다.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 역사인. 문자열을 변환할 수 없는 경우 결과는 Undefined 입니다. 가상 결과는 Undefined 로 반환됩니다.
배열	Undefined .

인수 유형	결과
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

atan(Decimal)

숫자의 역탄젠트를 라디안 단위로 반환합니다. Decimal 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: atan(0) = 0.0

인수 유형	결과
Int	Decimal(배정밀도), 인수의 역탄젠트. 가상 결과는 Undefined 로 반환됩니다.
Decimal	Decimal(배정밀도), 인수의 역탄젠트. 가상 결과는 Undefined 로 반환됩니다.
Boolean	Undefined .
String	Decimal, 인수의 역탄젠트. 문자열을 변환할 수 없는 경우 결과는 Undefined 입니다. 가상 결과는 Undefined 로 반환됩니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

atan2(Decimal, Decimal)

양의 x축과 두 인수로 정의된 점(x, y) 사이의 각도(라디안)를 반환합니다. 이 각도는 시계 반대 방향 각도(상반면, y > 0)의 경우 양수이고, 시계 방향 각도(하반면, y < 0). `Decimal` 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `atan2(1, 0)` = 1.5707963267948966

인수 유형	인수 유형	결과
`Int/Decimal`	`Int/Decimal`	`Decimal`(배정밀도), x축과 지정 사이의 각도.
`Int/Decimal/String`	`Int/Decimal/String`	`Decimal`, 설명된 점의 역탄젠트 변환할 수 없는 경우 결과는 Und 니다.
기타 값	기타 값	`Undefined` .

aws_lambda(functionArn, inputJson)

`inputJson`을 Lambda 함수로 전달하도록 지정된 Lambda 함수를 호출한 다음 Lambda 함수에서 생성되는 JSON 데이터를 반환합니다.

인수

인수	설명
`functionArn`	호출할 Lambda 함수의 ARN입니다. Lambda 함수는 JSON 데이터를 반환해야 합니다.
`inputJson`	Lambda 함수로 전달되는 JSON 입력 값입니다. 중첩된 객체 쿼리와 리터럴을 전달하려면 SQL 버전 2016-03-23을 사용해야 합니다.

지정된 Lambda 함수를 호출할 수 있는 권한을 부여 AWS IoT `lambda:InvokeFunction`해야 합니다. 다음은 AWS CLI를 사용하여 `lambda:InvokeFunction` 권한을 부여하는 방법을 보여주는 예제입니다.

```
aws lambda add-permission --function-name "function_name"
```

```
--region "region"
--principal iot.amazonaws.com
--source-arn arn:aws:iot:us-east-1:account_id:rule/rule_name
--source-account "account_id"
--statement-id "unique_id"
--action "lambda:InvokeFunction"
```

다음은 add-permission 명령의 인수입니다.

--function-name

Lambda 함수의 이름입니다. 함수의 리소스 정책을 업데이트하기 위한 새 권한을 추가합니다.

--region

계정 AWS 리전 의 입니다.

--principal

권한을 부여받는 보안 주체입니다. 이는 Lambda 함수`iot.amazonaws.com`를 호출할 수 있는 AWS IoT 권한을 허용하기 위한 것이어야 합니다.

--source-arn

규칙의 ARN입니다. get-topic-rule AWS CLI 명령을 사용하여 규칙의 ARN을 가져올 수 있습니다.

--source-account

규칙이 정의된 AWS 계정 입니다.

--statement-id

고유한 문 식별자입니다.

--action

이 문에서 허용할 Lambda 작업입니다. AWS IoT 가 Lambda 함수를 호출하도록 허용하려면 `lambda:InvokeFunction`을 지정합니다.

> **Important**
>
> `source-arn` 또는를 제공하지 않고 AWS IoT 보안 주체에 대한 권한을 추가하는 경우 Lambda 작업으로 규칙을 생성하는 `source-account` AWS 계정 는 Lambda 함수를 호출하는 규칙을 트리거할 수 있습니다 AWS IoT. 자세한 내용은 Lambda 권한 모델을 참조하세요.

다음과 같은 JSON 메시지 페이로드가 있을 경우:

```
{
    "attribute1": 21,
    "attribute2": "value"
}
```

다음과 같이 aws_lambda 함수를 사용하여 Lambda 함수를 호출할 수 있습니다.

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function",
 {"payload":attribute1}) as output FROM 'topic-filter'
```

전체 MQTT 메시지 페이로드를 전달하려면 다음 예와 같이 '*'를 사용하여 JSON 페이로드를 지정할 수 있습니다.

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function", *) as output
 FROM 'topic-filter'
```

payload.inner.element는 'topic/subtopic' 주제에 게시되는 메시지에서 데이터를 선택합니다.

some.value는 Lambda 함수에서 생성되는 출력에서 데이터를 선택합니다.

> **Note**
>
> 규칙 엔진은 Lambda 함수의 실행 시간을 제한합니다. 규칙의 Lambda 함수 호출은 2,000ms 이내에 완료되어야 합니다.

bitand(Int, Int)

두 Int(변환) 인수의 비트 표현에 대해 비트 단위 AND를 수행합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: bitand(13, 5) = 5

인수 유형	인수 유형	결과
Int	Int	Int, 두 인수의 비트 단위 AND.

인수 유형	인수 유형	결과
Int/Decimal	Int/Decimal	Int, 두 인수의 비트 단위 AND. 숫자는 가장 가까운 Int로 내림 수를 Int로 변환할 수 없는 경우 Undefined 입니다.
Int/Decimal/String	Int/Decimal/String	Int, 두 인수의 비트 단위 AND. 은 10진수로 변환된 후 가장 가까 내림됩니다. 변환이 실패할 경우 Undefined 입니다.
기타 값	기타 값	Undefined .

bitor(Int, Int)

두 인수의 비트 표현에 대해 비트 단위 OR을 수행합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `bitor(8, 5)` = 13

인수 유형	인수 유형	결과
Int	Int	Int, 두 인수의 비트 단위 OR.
Int/Decimal	Int/Decimal	Int, 두 인수의 비트 단위 OR. 5 숫자는 가장 가까운 Int로 내림 이 실패할 경우 결과는 Undefin
Int/Decimal/String	Int/Decimal/String	Int, 두 인수의 비트 단위 OR. 5 은 10진수로 변환된 후 가장 가까 내림됩니다. 변환이 실패할 경우 Undefined 입니다.
기타 값	기타 값	Undefined .

bitxor(Int, Int)

두 Int(변환) 인수의 비트 표현에 대해 비트 단위 XOR을 수행합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `bitor(13, 5) = 8`

인수 유형	인수 유형	결과
Int	Int	Int, 두 인수에 대한 비트 단위 X
Int/Decimal	Int/Decimal	Int, 두 인수에 대한 비트 단위 X 숫자는 가장 가까운 Int로 내림
Int/Decimal/String	Int/Decimal/String	Int, 두 인수에 대한 비트 단위 X 열은 10진수로 변환되어 가장 가 내림됩니다. 변환이 실패할 경우 Undefined 입니다.
기타 값	기타 값	Undefined .

bitnot(Int)

Int(변환) 인수의 비트 표현에 대해 비트 단위 NOT을 수행합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `bitnot(13) = 2`

인수 유형	결과
Int	Int, 인수의 비트 단위 NOT.
Decimal	Int, 인수의 비트 단위 NOT. Decimal 값은 가장 가까운 Int로 내림됩니다.
String	Int, 인수의 비트 단위 NOT. 문자열은 10진수로 변환된 후 가장 가까운 Int로 내림됩니다. 변환이 실패할 경우 결과는 Undefined 입니다.

인수 유형	결과
기타 값	기타 값

cast()

값을 한 데이터 형식에서 다른 형식으로 변환합니다. cast는 일반 변환과 거의 비슷하게 동작하지만 숫자와 부울 간 캐스팅이 추가됩니다. 가 한 유형을 다른 유형으로 캐스팅하는 방법을 결정할 AWS IoT 수 없는 경우 결과는 입니다`Undefined`. SQL 버전 2015-10-08 이상에서 지원됩니다. 형식: cast(*`value`* as *`type`*).

예제:

```
cast(true as Int) = 1
```

`cast` 호출 시 다음 키워드가 "as" 뒤에 나올 수 있습니다.

SQL 버전 2015-10-08 및 2016-03-23의 경우

키워드	결과
`String`	값을 `String`로 캐스팅합니다.
Nvarchar	값을 `String`로 캐스팅합니다.
텍스트	값을 `String`로 캐스팅합니다.
Ntext	값을 `String`로 캐스팅합니다.
varchar	값을 `String`로 캐스팅합니다.
`Int`	값을 `Int`로 캐스팅합니다.
Integer	값을 `Int`로 캐스팅합니다.
배정밀도 실수	값을 `Decimal`(배정밀도)로 캐스팅합니다.

SQL 버전 2016-03-23의 경우

키워드	결과
`Decimal`	값을 `Decimal`로 캐스팅합니다.
부울	값을 `Boolean`로 캐스팅합니다.
`Boolean`	값을 `Boolean`로 캐스팅합니다.

캐스팅 규칙:

소수로 캐스팅

인수 유형	결과
`Int`	소수점이 없는 `Decimal`
`Decimal`	소스 값 **Note** SQL V2(2016-03-23)에서 `10.0`와 (과) 같은 정수인 숫자 값은 예상되는 `Decimal` 값(`10.0`) 대신 `Int` 값(`10`)을 반환합니다. 정수 숫자 값을 `Decimal` 값으로 안정적으로 캐스팅하려면 규칙 쿼리 문에 SQL V1(2015-10-08)을 사용하세요.
`Boolean`	true = 1.0, false = 0.0.
`String`	문자열을 `Decimal`로 구문 분석하려고 시도합니다. AWS IoT 는 다음 정규식 ^-?\d+(\.\d+)?((?i)E-?\d+)?$와 일치하는 문자열을 구문 분석하려고 시도합니다. "0", "-1.2", "5E-12"는 모두 자동으로 10진수로 변환되는 문자열의 예입니다.

인수 유형	결과
배열	`Undefined .`
객체	`Undefined .`
Null	`Undefined .`
정의되지 않음	`Undefined .`

정수로 캐스팅

인수 유형	결과
`Int`	소스 값
`Decimal`	가장 가까운 `Int`로 내림한 소스 값
`Boolean`	true = 1.0, false = 0.0.
`String`	문자열을 `Decimal`로 구문 분석하려고 시도합니다. AWS IoT 는 다음 정규식 ^-?\d+(\.\d+)?((?i)E-?\d+)?$와 일치하는 문자열을 구문 분석하려고 시도합니다. "0", "-1.2", "5E-12"는 모두 자동으로 10진수로 변환되는 문자열의 예입니다. AWS IoT 는 문자열을 `Decimal`로 변환한 후 가장 가까운 `Int`로 내림하려고 시도합니다.
배열	`Undefined .`
객체	`Undefined .`
Null	`Undefined .`
정의되지 않음	`Undefined .`

Boolean로 캐스팅

인수 유형	결과
Int	0 = False, 0이 아닌 값 = True.
Decimal	0 = False, 0이 아닌 값 = True.
Boolean	소스 값
String	"true" = True, "false" = False(대/소문자를 구분하지 않음). 다른 문자열 값 = Undefined .
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

문자열로 캐스팅

인수 유형	결과
Int	표준 표기법을 따른 Int의 문자열 표현.
Decimal	유효숫자 표기법을 따를 수 있는 Decimal의 문자열 표현.
Boolean	"true" 또는 "false", 모두 소문자.
String	소스 값
배열	JSON으로 직렬화된 배열입니다. 결과 문자열은 대괄호 안의 쉼표로 구분된 목록입니다. String은 따옴표로 묶입니다. Decimal, Int 및 Boolean은 따옴표로 묶이지 않습니다.

인수 유형	결과
객체	JSON으로 직렬화된 객체입니다. JSON 문자열은 키-값 페어의 쉼표로 구분된 목록이며 중괄호로 묶입니다. String은 따옴표로 묶입니다. Decimal, Int, Boolean 및 Null은 따옴표로 묶이지 않습니다.
Null	Undefined .
정의되지 않음	Undefined .

ceil(Decimal)

지정된 Decimal을 가장 가까운 Int로 올림합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

ceil(1.2) = 2

ceil(-1.2) = -1

인수 유형	결과
Int	Int, 인수 값
Decimal	Int, 가장 가까운 Int로 올림된 Decimal 값
String	Int. 문자열은 Decimal로 변환된 후 가장 가까운 Int로 올림됩니다. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
기타 값	Undefined .

chr(String)

지정된 Int 인수에 대응하는 ASCII 문자를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`chr(65)` = "A".

`chr(49)` = "1".

인수 유형	결과
`Int`	지정된 ASCII 값에 대응하는 문자입니다. 인수가 유효한 ASCII 값이 아닐 경우 결과는 `Undefined` 입니다.
`Decimal`	지정된 ASCII 값에 대응하는 문자입니다. `Decimal` 인수는 가장 가까운 `Int`로 내림됩니다. 인수가 유효한 ASCII 값이 아닐 경우 결과는 `Undefined` 입니다.
`Boolean`	`Undefined` .
`String`	`String`을 `Decimal`로 변환할 수 있는 경우 가장 가까운 `Int`로 내림됩니다. 인수가 유효한 ASCII 값이 아닐 경우 결과는 `Undefined` 입니다.
배열	`Undefined` .
객체	`Undefined` .
Null	`Undefined` .
기타 값	`Undefined` .

clientid()

메시지를 전송하는 MQTT 클라이언트의 ID, 또는 메시지가 MQTT를 통해 전송되지 않은 경우 `n/a`를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`clientid()` = "123456789012"

concat()

배열 또는 문자열을 연결합니다. 이 함수는 인수의 수를 제한하지 않으며 `String` 또는 `Array`를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`concat()` = Undefined.

`concat(1)` = "1".

`concat([1, 2, 3], 4)` = [1, 2, 3, 4].

`concat([1, 2, 3], "hello")` = [1, 2, 3, "hello"]

`concat("con", "cat")` = "concat"

`concat(1, "hello")` = "1hello"

`concat("he","is","man")` = "heisman"

`concat([1, 2, 3], "hello", [4, 5, 6])` = [1, 2, 3, "hello", 4, 5, 6]

인수의 수	결과
0	`Undefined` .
1	인수가 수정 없이 반환됩니다.
2+	인수가 `Array`일 경우 결과는 모든 인수를 포함하는 단일 배열입니다. 모든 인수가 배열이 아니고 하나 이상의 인수가 `String`일 경우 결과는 모든 인수에 대한 `String` 표현의 연결입니다. 인수는 이전에 나열된 표준 변환을 사용하여 문자열로 변환됩니다.

cos(Decimal)

숫자의 코사인을 라디안 단위로 반환합니다. `Decimal` 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

cos(0) = 1.

인수 유형	결과
Int	Decimal(배정밀도), 인수의 코사인. 가상 결과는 Undefined 로 반환됩니다.
Decimal	Decimal(배정밀도), 인수의 코사인. 가상 결과는 Undefined 로 반환됩니다.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 코사인. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다. 가상 결과는 Undefined 로 반환됩니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

cosh(Decimal)

숫자의 쌍곡코사인을 라디안 단위로 반환합니다. Decimal 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: cosh(2.3) = 5.037220649268761.

인수 유형	결과
Int	Decimal(배정밀도), 인수의 쌍곡코사인. 가상 결과는 Undefined 로 반환됩니다.
Decimal	Decimal(배정밀도), 인수의 쌍곡코사인. 가상 결과는 Undefined 로 반환됩니다.

인수 유형	결과
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 쌍곡코사인. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다. 가상 결과는 Undefined 로 반환됩니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

decode(value, decodingScheme)

decode 함수를 사용하여 인코딩된 값을 디코딩합니다. 디코딩된 문자열이 JSON 문서인 경우, 주소 지정 가능한 객체가 반환됩니다. 그렇지 않으면 디코딩된 문자열이 문자열로 반환됩니다. 문자열을 디코딩할 수 없는 경우 함수는 NULL을 반환합니다. 이 함수는 base64로 인코딩된 문자열과 프로토콜 버퍼(protobuf) 메시지 형식의 디코딩을 지원합니다.

SQL 버전 2016-03-23 이상에서 지원됩니다.

값

문자열 값, 또는 문자열을 반환하는 (AWS IoT SQL 참조에 정의된) 유효한 표현식입니다.

decodingScheme

값을 디코딩하는 데 사용되는 체계를 나타내는 리터럴 문자열입니다. 현재 'base64' 및 'proto'만 지원됩니다.

base64로 인코딩된 문자열 디코딩

이 예에서 메시지 페이로드는 인코딩된 값을 포함합니다.

```
{
    encoded_temp: "eyAidGVtcGVyYXR1cmUiOiAzMyB9Cg=="
```

```
}
```

이 SQL 문의 decode 함수는 메시지 페이로드의 값을 디코딩합니다.

```
SELECT decode(encoded_temp,"base64").temperature AS temp from 'topic/subtopic'
```

encoded_temp 값을 디코딩하면 SELECT 문에서 온도 값을 읽을 수 있는 다음과 같은 유효한 JSON 문서가 생성됩니다.

```
{ "temperature": 33 }
```

이 예제에서 SELECT 문의 결과는 다음과 같습니다.

```
{ "temp": 33 }
```

디코딩된 값이 유효한 JSON 문서가 아닌 경우 디코딩된 값은 문자열로 반환됩니다.

protobuf 메시지 페이로드 디코딩

decode SQL 함수를 사용하여 protobuf 메시지 페이로드를 디코딩할 수 있는 규칙을 구성할 수 있습니다. 자세한 내용은 protobuf 메시지 페이로드 디코딩을 참조하세요.

> **Important**
>
> AWS IoT 보안 주체에 대한 권한을 설정할 source-account 때 source-arn 또는를 생략하면에서 다른 AWS IoT 규칙을 통해 디코딩 함수를 호출할 AWS 계정 수 있습니다. 함수를 보호하려면 Amazon Simple Storage Service 사용 설명서의 버킷 정책을 참조하세요.

함수 서명은 다음과 같습니다.

```
decode(<ENCODED DATA>, 'proto', '<S3 BUCKET NAME>', '<S3 OBJECT KEY>', '<PROTO NAME>',
 '<MESSAGE TYPE>')
```

ENCODED DATA

디코딩할 protobuf 인코딩 데이터를 지정합니다. 규칙으로 전송된 전체 메시지가 protobuf로 인코딩된 데이터인 경우 *를 사용하여 원시 바이너리 수신 페이로드를 참조할 수 있습니다. 그렇지 않은 경우 이 필드는 base-64로 인코딩된 JSON 문자열이어야 하며 문자열에 대한 참조를 직접 전달할 수 있습니다.

1) 원시 바이너리 protobuf 수신 페이로드를 디코딩하는 방법

```
decode(*, 'proto', ...)
```

2) base64로 인코딩된 문자열 'a.b'로 표시되는 protobuf로 인코딩된 메시지를 디코딩하는 방법

```
decode(a.b, 'proto', ...)
```

`proto`

디코딩할 데이터를 protobuf 메시지 형식으로 지정합니다. `proto` 대신 `base64`를 지정하면 이 함수는 base64로 인코딩된 문자열을 JSON으로 디코딩합니다.

`S3 BUCKET NAME`

`FileDescriptorSet` 파일을 업로드한 Amazon S3 버킷의 이름입니다.

`S3 OBJECT KEY`

Amazon S3 버킷 내의 `FileDescriptorSet` 파일을 지정하는 객체 키입니다.

`PROTO NAME`

`FileDescriptorSet` 파일을 생성하는 데 사용된 `.proto` 파일의 이름(확장명 제외)입니다.

`MESSAGE TYPE`

디코딩할 데이터가 준수해야 하는 `FileDescriptorSet` 파일 내 protobuf 메시지 구조의 이름입니다.

decode SQL 함수를 사용한 SQL 표현식의 예는 다음과 같습니다.

```
SELECT VALUE decode(*, 'proto', 's3-bucket', 'messageformat.desc', 'myproto',
 'messagetype') FROM 'some/topic'
```

- `*`

  `mymessagetype`이라는 protobuf 메시지 유형을 준수하는 이진 수신 페이로드를 나타냅니다.
- `messageformat.desc`

  `s3-bucket`이라는 Amazon S3 버킷에 저장된 `FileDescriptorSet` 파일입니다.

- `myproto`

  `myproto.proto`라는 `FileDescriptorSet` 파일을 생성하는 데 사용된 원본 `.proto` 파일입니다.

- `messagetype`

  `myproto.proto`에 정의된 `messagetype`이라는 메시지 유형(가져온 종속 항목 포함)입니다.

encode(value, encodingScheme)

`encode` 함수를 사용하여 페이로드(비 JSON 데이터일 수 있음)를 인코딩 체계를 기반으로 한 문자열 표현으로 인코딩합니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

값

AWS IoT SQL 참조에서 정의된 임의의 유효한 표현식. *를 지정하여 JSON 형식 여부와 상관없이 전체 페이로드를 인코딩할 수 있습니다. 사용자가 표현식을 제공할 경우 식 결과가 문자열로 변환된 후 인코딩됩니다.

encodingScheme

사용할 인코딩 체계를 나타내는 리터럴 문자열. 현재 `'base64'`만 지원됩니다.

endswith(String, String)

첫 번째 `String` 인수가 두 번째 `String` 인수로 끝나는지 여부를 나타내는 `Boolean`을 반환합니다. 인수가 `Null` 또는 `Undefined`일 경우 결과는 `Undefined`입니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `endswith("cat","at")` = true.

인수 형식 1	인수 형식 2	결과
`String`	`String`	첫 번째 인수가 두 번째 인수로 끝 true입니다. 그렇지 않으면 false
기타 값	기타 값	두 인수는 모두 표준 변환 규칙을 여 문자열로 변환됩니다. 첫 번째 두 번째 인수로 끝날 경우 true입

인수 형식 1	인수 형식 2	결과
		지 않으면 false입니다. 인수가 N Undefined 일 경우 결과는 Un 입니다.

exp(Decimal)

Decimal 인수로 거듭제곱된 e를 반환합니다. Decimal 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: exp(1) = e.

인수 유형	결과
Int	Decimal(배정밀도), e ^ 인수.
Decimal	Decimal(배정밀도), e ^ 인수.
String	Decimal(배정밀도), e ^ 인수. String을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
기타 값	Undefined .

floor(십진수)

지정된 Decimal을 가장 가까운 Int로 내림합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

floor(1.2) = 1

floor(-1.2) = -2

인수 유형	결과
Int	Int, 인수 값

인수 유형	결과
Decimal	가장 가까운 Int로 내림된 Decimal 값은 Int입니다.
String	Int. 문자열은 Decimal로 변환된 후 가장 가까운 Int로 내림됩니다. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
기타 값	Undefined .

시작

모음과 같은 형식(배열, 문자열, 객체)에서 값을 추출합니다. 첫 번째 인수에는 변환이 적용되지 않습니다. 변환은 두 번째 인수에 표에 설명된 대로 적용됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

```
get(["a", "b", "c"], 1) = "b"
```

```
get({"a":"b"}, "a") = "b"
```

```
get("abc", 0) = "a"
```

인수 형식 1	인수 형식 2	결과
배열	임의의 형식(Int로 변환됨)	두 번째 인수(Int로 변환됨)가 터 시작하는 Array 인덱스의 항 실패할 경우 결과는 Undefined 덱스가 Array의 범위를 벗어날 또는 >= array.length) 결과는 Un 입니다.
문자열	임의의 형식(Int로 변환됨)	두 번째 인수(Int로 변환됨)가 터 시작하는 문자열 인덱스의 문 실패할 경우 결과는 Undefined 덱스가 문자열의 범위를 벗어날

인수 형식 1	인수 형식 2	결과
		또는 >= string.length) 결과는 Un 입니다.
객체	String(변환이 적용되지 않음)	두 번째 인수가 제공하는 문자열 되는 첫 번째 인수 객체에 저장된
기타 값	임의의 값	Undefined .

get_dynamodb(tableName, partitionKeyName, partitionKeyValue, sortKeyName, sortKeyValue, roleArn)

DynamoDB 테이블에서 데이터를 검색합니다. `get_dynamodb()`를 사용하면 규칙이 평가되는 동안 DynamoDB 테이블을 쿼리할 수 있습니다. DynamoDB에서 검색된 데이터를 사용하여 메시지 페이로드를 필터링하거나 증가시킬 수 있습니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

`get_dynamodb()`는 다음 파라미터를 사용합니다.

tableName

쿼리할 DynamoDB 테이블의 이름입니다.

partitionKeyName

파티션 키의 이름입니다. 자세한 내용은 DynamoDB 키를 참조하세요.

partitionKeyValue

레코드를 식별하는 데 사용되는 파티션 키의 값입니다. 자세한 내용은 DynamoDB 키를 참조하세요.

sortKeyName

(선택 사항) 정렬 키의 이름입니다. 이 파라미터는 쿼리된 DynamoDB 테이블에서 복합 키를 사용하는 경우에만 필요합니다. 자세한 내용은 DynamoDB 키를 참조하세요.

sortKeyValue

(선택 사항) 정렬 키의 값입니다. 이 파라미터는 쿼리된 DynamoDB 테이블에서 복합 키를 사용하는 경우에만 필요합니다. 자세한 내용은 DynamoDB 키를 참조하세요.

roleArn

DynamoDB 테이블에 대한 액세스 권한을 부여하는 IAM 역할의 ARN입니다. 규칙 엔진은 사용자를 대신하여 DynamoDB 테이블에 액세스하기 위해 이 역할을 수임합니다. 지나치게 허용적인 역할을 사용하지 마세요. 규칙에 필요한 권한만 역할에 부여합니다. 다음은 하나의 DynamoDB 테이블에 대한 액세스 권한을 부여하는 정책 예제입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/table-
name"
        }
    ]
}
```

`get_dynamodb()`를 사용하는 방법의 예로, AWS IoT에 연결된 모든 장치에 대한 디바이스 ID와 위치 정보가 포함된 DynamoDB 테이블이 있다고 가정해 보겠습니다. 다음 SELECT 문은 `get_dynamodb()` 함수를 사용하여 지정된 디바이스 ID의 위치를 검색합니다.

```
SELECT *, get_dynamodb("InServiceDevices", "deviceId", id,
"arn:aws:iam::12345678910:role/getdynamo").location AS location FROM 'some/
topic'
```

> **Note**
>
> - SQL 문 1개당 최대 한 번 `get_dynamodb()`를 호출할 수 있습니다. 단일 SQL 문에서 여러 번 `get_dynamodb()`를 호출하면 작업을 호출하지 않고 규칙이 종료됩니다.

get_mqtt_property(name)

MQTT5 헤더 `contentType`, `payLoadFormatIndicator`, `responseTopic`, `correlationData` 중 하나를 참조합니다. 이 함수는 리터럴 문자열 `content_type`, `format_indicator`, `response_topic`, `correlation_data` 중 하나를 인수로 사용합니다. 자세한 내용은 다음 함수 인수 테이블을 참조하세요.

contentType

문자열: 게시 메시지의 내용을 설명하는 UTF-8으로 인코딩된 문자열입니다.

payLoadFormatIndicator

문자열: 페이로드가 UTF-8 형식으로 지정되었는지 여부를 나타내는 Enum 문자열 값입니다. 유효 값은 `UNSPECIFIED_BYTES` 및 `UTF8_DATA`입니다.

responseTopic

문자열: 응답 메시지의 주제 이름으로 사용되는 UTF-8 인코딩 문자열입니다. 응답 주제는 수신자가 요청-응답 흐름의 일부로 게시해야 하는 주제를 설명하는 데 사용됩니다. 주제에는 와일드카드 문자가 포함되어서는 안 됩니다.

correlationData

문자열: 요청 메시지 발신자가 응답 메시지를 수신할 때 어떤 요청에 대한 응답 메시지인지 식별하는 데 사용되는 base64 인코딩 이진 데이터입니다.

다음 표에는 `get_mqtt_property` 함수에 사용할 수 있는 함수 인수와 관련 반환 유형이 나와 있습니다.

함수 인수

SQL	반환되는 데이터 유형(있는 경우)	반환되는 데이터 유형(없는 경우)
`get_mqtt_property("format_indicator")`	문자열(UNSPECIFIED_BYTES 또는 UTF8_DATA)	문자열(UNSPECIFIED_BYTES)
`get_mqtt_property("content_type")`	문자열	정의되지 않음
`get_mqtt_property("response_topic")`	문자열	정의되지 않음
`get_mqtt_property("correlation_data")`	base64로 인코딩된 문자열	정의되지 않음

SQL	반환되는 데이터 유형(있는 경우)	반환되는 데이터 유형(없는 경우)
`get_mqtt_property( "some_invalid_name ")`	정의되지 않음	정의되지 않음

다음 예제 규칙 SQL은 MQTT5 헤더 `contentType`, `payLoadFormatIndicator`, `responseTopic`, `correlationData` 중 하나를 참조합니다.

```
SELECT *, get_mqtt_property('content_type') as contentType,
          get_mqtt_property('format_indicator') as payloadFormatIndicator,
          get_mqtt_property('response_topic') as responseTopic,
          get_mqtt_property('correlation_data') as correlationData
FROM 'some/topic'
```

get_or_default(식, defaultValue)

지정된 경우 두 번째 파라미터의 기본값을 반환하고, 그렇지 않으면 첫 번째 파라미터의 표현식이 null, 정의되지 않음 또는 실패를 반환합니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

> **Important**
>
> `get_or_default`는 JSON이 아닌 페이로드를 있는 그대로 직접 지원하지 않습니다. 비 JSON 페이로드를 사용하는 경우 `encode` 또는 `decode` 함수를 사용합니다.

`get_or_default()`는 다음 파라미터를 사용합니다.

expression

데이터 타입, , 함수, 리터럴 변수, 중첩된 객체 쿼리또는를 포함하는 모든 유효한 표현식입니다JSON 확장.

defaultValue

(선택 사항) 데이터 타입, , 함수, 리터럴 변수, 중첩된 객체 쿼리또는를 포함하는 모든 유효한 표현식입니다JSON 확장. 첫 번째 인수가 null, 정의되지 않음 또는 실패를 반환할 때마다 반환되는 값입니다.

> **Note**
>
> get_secret, get_dynamodb, aws_lambda, get_thing_shadow, decode-protobuf, machinelearning_predict와 같은 고객 소유 리소스에서 데이터를 가져오는 함수는 defaultValue 파라미터에 허용되지 않습니다.

다음 표에는 각 인수 및 관련 출력에 허용되는 함수 인수가 나와 있습니다.

첫 번째 인수	두 번째 인수	출력
성공적인 평가	모든 값 또는 지정되지 않음	첫 번째 인수 값입니다.
정의되지 않음, Null 또는 실패	정의되지 않음 또는 Null을 포함한 모든 값	두 번째 인수 값입니다.
정의되지 않음, Null 또는 실패	지정되지 않음	`Undefined`

예:

예제 1:

다음 예제에서는 DynamoDB 테이블 또는 쿼리가 실패할 경우 defaultValue 값을 제공합니다.

```
SELECT
    device_id,
    get_or_default(
        get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME"),
        {"mode": "standard", "timeout": 30, "enabled": true }
    ) as config
FROM 'device/telemetry'
```

예제 2:

다음 예제에서는 상태가 정의되지 않은 경우 안전한 기본값 "UNKNOWN"을 제공합니다.

```
SELECT
  get_or_default( CASE status
```

```
    WHEN 'active' THEN 'GOOD'
    WHEN 'inactive' THEN 'BAD'/
    ELSE 'UNKNOWN'
  END, 'UNKNOWN') as status_category
FROM 'topic/subtopic'
```

예제 3:

다음 예제에서는 단일 파라미터와 함께 get_or_default를 사용하는 방법을 보여줍니다. 이는 명확한 기본값이 없을 수 있지만 규칙 실행이 실패하지 않도록 하려는 시나리오에서 유용합니다.

```
SELECT
  get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME") as config
FROM 'device/telemetry'
```

DynamoDB 조회가 실패하면 규칙 실행이 실패하고 작업이 실행되지 않습니다. 대신 다음 SQL을 사용하는 경우:

```
SELECT
  get_or_default(get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME")) as config
FROM 'device/telemetry'
```

get_or_default 문은 로 평가Undefined되므로이 예제에서는 SELECT 문 전체가 로 평가{}되고 규칙 작업이 시도됩니다.

> **Important**
>
> 이 함수를 사용할 때 보안을 유지하려면 다음 모범 사례를 따르는 것이 좋습니다.
>
> - 기본값을 포함한 규칙 정의에서 하드 코딩된 보안 암호 사용 금지
> - 민감한 정보 관리에 AWS Secrets Manager 사용

get_registry_data(registryAPI, thingName, roleArn)

AWS IoT 규칙에서 AWS IoT 사물 레지스트리 데이터를 검색합니다. 레지스트리 데이터(예: 디바이스가 속한 속성, 사물 유형 및 사물 그룹)를 읽고이 정보를 사용하여 메시지를 필터링, 보강 또는 동적으로 라우팅할 수 있습니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

`get_registry_data()`는 다음 파라미터를 사용합니다.

registryAPI

호출 중인 레지스트리 API입니다. 유효 값은 `DescribeThing` 및 `ListThingGroupsForThing`입니다. 이러한 값은 상수 문자열이어야 합니다.

thingName

문자열: 레지스트리 데이터를 검색하려는 사물의 이름입니다.

roleArn

문자열: 호출 중인 API에 따라 `iot:DescribeThing` 권한 및/또는 `iot:ListThingGroupsForThing` 권한이 있는 역할 ARN입니다.

`get_registry_data` 함수의 응답 형식은 라는 레지스트리 API와 동일합니다. 자세한 내용은 DescribeThing 및 ListThingGroupsForThing APIs.

예제:

사물 유형 정보를 검색하여 사물 유형이 인 사물(사물 이름이 MQTT 클라이언트 ID와 일치)에 대한 AWS IoT Core 수명 주기 이벤트 메시지를 필터링할 수 있습니다`testenv`.

```
SELECT *
FROM '$aws/events/lifecycle/+'
WHERE
    get_registry_data("DescribeThing",clientId,[roleArn]).thingTypeName='testenv'
```

예제:

게이트웨이 디바이스에서 보낸 모든 메시지에 대해 사물 이름을 가진 디바이스`sensor1`의 사물 속성을 검색할 수 있습니다`gateway1`.

```
SELECT *, get_registry_data("DescribeThing","sensor1",
[roleArn]).attributes.temperature_threhold AS device1_tempthreshold
FROM home1/gateway1/sensor1/#
```

> **Note**
>
> - 작업 및 오류 작업에 대해 SQL 문 및 대체 템플릿당 `get_registry_data()` 최대 한 번 호출할 수 있습니다.

get_secret(secretId, secretType, key, roleArn)

AWS Secrets Manager에서 현재 버전의 보안 암호에 대한 암호화된 `SecretString` 또는 `SecretBinary` 필드의 값을 검색합니다. 보안 암호 생성 및 유지 관리에 대한 자세한 내용은 CreateSecret, UpdateSecret 및 PutSecretValue를 참조하세요.

`get_secret()`는 다음 파라미터를 사용합니다.

SecretId

문자열: 검색할 보안 암호의 Amazon 리소스 이름(ARN) 또는 친숙한 이름입니다.

secretType

문자열: 보안 암호 유형입니다. 유효한 값: `SecretString` | `SecretBinary`.

SecretString

- APIs, AWS CLI또는 AWS Secrets Manager 콘솔을 사용하여 JSON 객체로 생성하는 보안 암호의 경우:
  - `key` 파라미터에 값을 지정한 경우, 이 함수는 지정된 키의 값을 반환합니다.
  - `key` 파라미터에 값을 지정하지 않은 경우 이 함수는 전체 JSON 객체를 반환합니다.
- API, 또는 AWS CLI를 사용하여 JSON 객체로 생성하는 보안 암호의 경우:
  - `key` 파라미터에 값을 지정한 경우 이 함수는 예외와 함께 실패합니다.
  - `key` 파라미터에 값을 지정하지 않은 경우 이 함수는 보안 암호의 내용을 반환합니다.

SecretBinary

- `key` 파라미터에 값을 지정한 경우 이 함수는 예외와 함께 실패합니다.
- `key` 파라미터에 값을 지정하지 않은 경우 이 함수는 보안 암호 값을 Base64로 인코딩된 UTF-8 문자열로 반환합니다.

키

(선택 사항) 문자열: `SecretString` 보안 암호 필드에 저장된 JSON 객체 내부의 키 이름입니다. 전체 JSON 객체 대신 보안 암호에 저장된 키의 값만 검색하려는 경우 이 값을 사용합니다.

이 파라미터에 값을 지정하고 보안 암호의 `SecretString` 필드에 JSON 객체가 포함되어 있지 않으면 이 함수는 예외와 함께 실패합니다.

roleArn

문자열: `secretsmanager:GetSecretValue` 및 `secretsmanager:DescribeSecret` 권한을 갖는 역할 ARN입니다.

> **Note**
>
> 이 함수는 항상 현재 버전의 보안 암호(`AWSCURRENT` 태그가 있는 버전)를 반환합니다 AWS IoT 규칙 엔진은 각 보안 암호를 최대 15분 동안 캐싱합니다. 따라서 규칙 엔진이 보안 암호를 업데이트하는 데 최대 15분이 걸릴 수 있습니다. 즉 AWS Secrets Manager, 업데이트 후 최대 15분 이내에 보안 암호를 검색하면이 함수가 이전 버전을 반환할 수 있습니다.
> 이 함수는 측정되지 않지만 AWS Secrets Manager 요금이 적용됩니다. 보안 암호 캐싱 메커니즘으로 인해 규칙 엔진은 때때로 AWS Secrets Manager을(를) 호출합니다. 규칙 엔진은 완전히 분산된 서비스이므로 15분 캐싱 기간 동안 규칙 엔진에서 여러 Secrets Manager API 호출을 볼 수 있습니다.

예시:

`get_secret` 함수를 다음 API 키 인증 예제와 같이 HTTPS 규칙 동작의 인증 헤더에서 사용할 수 있습니다.

```
"API_KEY": "${get_secret('API_KEY', 'SecretString', 'API_KEY_VALUE',
 'arn:aws:iam::12345678910:role/getsecret')}"
```

HTTPS 규칙 작업에 대한 자세한 내용은 the section called “HTTP” 단원을 참조하세요.

get_thing_shadow(thingName, shadowName, roleArn)

지정된 사물의 지정된 섀도우를 반환합니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

thingName

문자열: 섀도우를 검색할 사물의 이름입니다.

shadowName

(선택 사항) 문자열: 섀도우의 이름입니다. 이 파라미터는 명명된 섀도우를 참조하는 경우에만 필요합니다.

roleArn

문자열: `iot:GetThingShadow` 권한을 갖는 역할 ARN입니다.

예시:

명명된 섀도우와 함께 사용할 경우 `shadowName` 파라미터를 제공합니다.

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","MyThingShadow","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

명명되지 않은 섀도우와 함께 사용할 경우 `shadowName` 파라미터를 생략합니다.

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

get_user_properties(userPropertyKey)

MQTT5에서 지원되는 속성 헤더 유형 중 하나인 사용자 속성을 참조합니다.

userProperty

문자열: 사용자 속성은 키-값 페어입니다. 이 함수는 키를 인수로 사용하여 관련 키와 일치하는 모든 값의 배열을 반환합니다.

함수 인수

메시지 헤더에 있는 다음 사용자 속성의 경우:

Key(키)	값
some key	some value
a different key	a different value
some key	value with duplicate key

다음 표에는 예상 SQL 동작이 나와 있습니다.

SQL	반환되는 데이터 형식	반환되는 데이터 값
get_user_properties('some key')	문자열 배열	`['some value', 'value with duplicate key']`
get_user_properties('other key')	문자열 배열	`['a different value']`
get_user_properties( )	키-값 페어 객체의 배열	`[{'"some key": "some value"'}, {"other key": "a different value"}, {"some key": "value with duplicate key"}]`
get_user_properties('non-existent key')	정의되지 않음	

다음 예제 규칙 SQL은 사용자 속성(MQTT5 속성 헤더 유형)을 페이로드로 참조합니다.

```
SELECT *, get_user_properties('user defined property key') as userProperty
FROM 'some/topic'
```

해시 함수

AWS IoT 는 다음과 같은 해싱 함수를 제공합니다.

- md2

- md5
- sha1
- sha224
- sha256
- sha384
- sha512

모든 해시 함수에는 문자열 인수 1개가 필요합니다. 결과는 해당 문자열의 해시된 값입니다. 표준 문자열 변환이 비 문자열 인수에 적용됩니다. 모든 해시 함수는 SQL 버전 2015-10-08 이상에서 지원합니다.

예시:

`md2("hello")` = "a9046c73e00331af68917d3804f70655"

`md5("hello")` = "5d41402abc4b2a76b9719d911017c592"

indexof(String, String)

두 번째 인수의 첫 번째 인덱스(0부터 시작)을 첫 번째 인수의 하위 문자열로 반환합니다. 두 인수 모두 문자열이 필요합니다. 문자열이 아닌 인수에는 표준 문자열 변환 규칙이 적용됩니다. 이 함수는 배열에는 적용되지 않고 문자열에만 적용됩니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

예시:

`indexof("abcd", "bc")` = 1

isNull()

인수가 `Null` 값이면 true를 반환합니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

예시:

`isNull(5)` = false.

`isNull(Null)` = true.

인수 유형	결과
`Int`	false

인수 유형	결과
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	true
`Undefined`	false

isUndefined()

인수가 `Undefined`이면 true를 반환합니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

예시:

`isUndefined(5)` = false.

`isUndefined(floor([1,2,3]))` = true.

인수 유형	결과
`Int`	false
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	false

인수 유형	결과
Undefined	true

length(String)

제공된 문자열의 문자 수를 반환합니다. 비 String 인수에는 표준 변환 규칙이 적용됩니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

예시:

length("hi") = 2

length(false) = 5

ln(Decimal)

인수의 자연 로그를 반환합니다. Decimal 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: ln(e) = 1.

인수 유형	결과
Int	Decimal(배정밀도), 인수의 자연 로그.
Decimal	Decimal(배정밀도), 인수의 자연 로그.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 자연 로그. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .

인수 유형	결과
정의되지 않음	`Undefined` .

log(Decimal)

인수의 기수 10 로그를 반환합니다. `Decimal` 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `log(100)` = 2.0.

인수 유형	결과
`Int`	`Decimal`(배정밀도), 인수의 기수 10 로그.
`Decimal`	`Decimal`(배정밀도), 인수의 기수 10 로그.
`Boolean`	`Undefined` .
`String`	`Decimal`(배정밀도), 인수의 기수 10 로그. `String`을 `Decimal`로 변환할 수 없는 경우 결과는 `Undefined` 입니다.
배열	`Undefined` .
객체	`Undefined` .
Null	`Undefined` .
정의되지 않음	`Undefined` .

lower(String)

지정된 `String`의 소문자 버전을 반환합니다. 비 문자열 인수는 표준 변환 규칙을 사용하여 문자열로 변환됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`lower("HELLO")` = "hello".

`lower(["HELLO"])` = "[\"hello\"]".

lpad(String, Int)

두 번째 인수로 지정된 수의 공백이 왼쪽에 추가된 `String` 인수를 반환합니다. `Int` 인수는 0부터 1000 사이여야 합니다. 제공된 값이 이 유효한 범위를 벗어날 경우 인수가 가장 가까운 유효한 값(0 또는 1000)으로 설정됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`lpad("hello", 2)` = " hello".

`lpad(1, 3)` = " 1"

인수 형식 1	인수 형식 2	결과
`String`	`Int`	`String`. 제공된 `Int`와 동일한 수 지정된 `String` 왼쪽에 추가됩니
`String`	`Decimal`	`Decimal` 인수는 가장 가까운 `In` 고 `String`은 지정된 수의 공백으 가됩니다.
`String`	`String`	두 번째 인수는 `Decimal`로 변환 장 가까운 `Int`로 내림되고, `Str` 된 수의 공백이 왼쪽에 추가됩니 인수를 `Int`로 변환할 수 없는 경 `Undefined` 입니다.
기타 값	`Int/Decimal/String`	첫 번째 값이 표준 변환을 통해 `S` 변환된 후, 이 `String`에 LPAD 함 용됩니다. 변환이 불가능한 경우 `Undefined` 입니다.
임의의 값	기타 값	`Undefined` .

ltrim(String)

제공된 `String`에서 모든 선행 공백(탭 및 공백)을 제거합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`Ltrim(" h i ")` = "hi ".

인수 유형	결과
`Int`	모든 선행 공백이 제거된 `Int`의 `String` 표현.
`Decimal`	모든 선행 공백이 제거된 `Decimal`의 `String` 표현.
`Boolean`	모든 선행 공백이 제거된 부울('true' 또는 'false')의 `String` 표현.
`String`	모든 선행 공백이 제거된 인수.
배열	모든 선행 공백이 제거된 `Array`의 `String`표현(표준 변환 규칙 사용).
객체	모든 선행 공백이 제거된 객체의 `String` 표현(표준 변환 규칙 사용).
Null	`Undefined` .
정의되지 않음	`Undefined` .

machinelearning_predict(modelId, roleArn, record)

`machinelearning_predict` 함수를 사용하여 Amazon SageMaker AI 모델을 기반으로 MQTT 메시지의 데이터를 사용하여 예측합니다. SQL 버전 2015-10-08 이상에서 지원됩니다. `machinelearning_predict` 함수의 인수는 다음과 같습니다.

modelId

예측을 실행할 모델의 ID. 모델의 실시간 엔드포인트가 활성화되어야 합니다.

roleArn

`machinelearning:Predict` 및 `machinelearning:GetMLModel` 권한을 갖는 정책이 연결되고 예측을 실행할 모델에 대한 액세스를 허용하는 IAM 역할입니다.

record

SageMaker AI 예측 API로 전달할 데이터입니다. 이 인수는 단일 계층 JSON 객체로 표현되어야 합니다. 레코드가 다중 수준 JSON 객체일 경우 레코드가 값 직렬화를 통해 평면화됩니다. 예를 들어 다음 JSON이

```
{ "key1": {"innerKey1": "value1"}, "key2": 0}
```

다음과 같이 평면화됩니다.

```
{ "key1": "{\"innerKey1\": \"value1\"}", "key2": 0}
```

이 함수는 다음 필드를 포함하는 JSON 객체를 반환합니다.

predictedLabel

모델을 기반으로 한 입력의 분류입니다.

details

다음 속성을 포함합니다.

PredictiveModelType

모델 유형입니다. 유효한 값은 REGRESSION, BINARY, MULTICLASS입니다.

Algorithm

SageMaker AI가 예측에 사용하는 알고리즘입니다. 값은 SGD이어야 합니다.

predictedScores

각 레이블에 해당하는 원시 분류 점수를 포함합니다.

predictedValue

SageMaker AI가 예측한 값입니다.

mod(Decimal, Decimal)

첫 번째 인수를 두 번째 인수로 나눈 나머지를 반환합니다. 나머지(십진수, 십진수)와 동일합니다. 또한 "%"를 동일한 모듈로 기능의 중위 연산자로 사용할 수도 있습니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `mod(8, 3)` = 2.

왼쪽 피연산자	오른쪽 피연산자	출력
`Int`	`Int`	`Int`, 두 번째 인수를 법으로 하는 수.
`Int/Decimal`	`Int/Decimal`	`Decimal`, 두 번째 피연산자를 법 첫 번째 인수.
`String/Int/Decimal`	`String/Int/Decimal`	모든 문자열이 10진수로 변환될 는 두 번째 인수를 법으로 하는 첫 입니다. 그렇지 않을 경우 Undef 다.
기타 값	기타 값	`Undefined` .

nanvl(AnyValue, AnyValue)

유효한 `Decimal`일 경우 첫 번째 인수를 반환합니다. 그렇지 않으면 두 번째 인수를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `Nanvl(8, 3)` = 8.

인수 형식 1	인수 형식 2	출력
정의되지 않음	임의의 값	두 번째 인수.
Null	임의의 값	두 번째 인수.
`Decimal`(NaN)	임의의 값	두 번째 인수.
`Decimal`(NaN 아님)	임의의 값	첫 번째 인수.

인수 형식 1	인수 형식 2	출력
기타 값	임의의 값	첫 번째 인수.

newuuid()

임의의 16바이트 UUID를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `newuuid()` = `123a4567-b89c-12d3-e456-789012345000`

numbytes(String)

지정된 문자열의 UTF-8 인코딩 내 바이트 수를 반환합니다. 비 `String` 인수에는 표준 변환 규칙이 적용됩니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

예시:

`numbytes("hi")` = 2

`numbytes("€")` = 3

parse_time(String, Long[, String])

`parse_time` 함수는 타임스탬프를 사람이 읽을 수 있는 날짜/시간 형식으로 만듭니다. SQL 버전 2016-03-23 이상에서 지원됩니다. 타임스탬프 문자열을 밀리초로 변환하려면 time_to_epoch(String, String) 단원을 참조하세요.

`parse_time` 함수는 다음 인수를 사용합니다.

pattern

(문자열) Joda-Time 형식을 따르는 날짜/시간 패턴입니다.

timestamp

(Long) Unix Epoch부터의 시간을 밀리초로 변환한 시간입니다. timestamp() 함수를 참조하세요.

timezone

(문자열) 형식 지정된 날짜/시간의 시간대입니다. 기본값은 "UTC"입니다. 이 함수는 Joda-Time 시간대를 지원합니다. 이 인수는 선택 사항입니다.

예시:

이 메시지가 주제 'A/B'에 게시되면 페이로드 {"ts": "1970.01.01 AD at 21:46:40 CST"}가 S3 버킷으로 전송됩니다.

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", 100000000,
 'America/Belize' ) as ts FROM 'A/B'",

        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

이 메시지가 주제 'A/B'에 게시되면 페이로드 {"ts": "2017.06.09 AD at 17:19:46 UTC"}가 S3 버킷으로 전송됩니다.

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", timestamp() ) as ts
 FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
```

```
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

parse_time()은 대체 템플릿으로 사용할 수도 있습니다. 예를 들어 이 메시지가 주제 'A/B'에 게시되면 페이로드가 키 값이 "2017"인 S3 버킷으로 전송됩니다.

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT * FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [{
            "s3": {
                "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                "bucketName": "BUCKET_NAME",
                "key": "${parse_time('yyyy', timestamp(), 'UTC')}"
            }
        }],
        "ruleName": "RULE_NAME"
    }
}
```

power(Decimal, Decimal)

두 번째 인수로 거듭제곱된 첫 번째 인수를 반환합니다. Decimal인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: power(2, 5) = 32.0.

인수 형식 1	인수 형식 2	출력
Int/Decimal	Int/Decimal	Decimal(배정밀도), 두 번째 인 거듭제곱된 첫 번째 인수.
Int/Decimal/String	Int/Decimal/String	Decimal(배정밀도), 두 번째 인 로 거듭제곱된 첫 번째 인수. 모든

인수 형식 1	인수 형식 2	출력
		은 10진수로 변환됩니다. Strin Decimal(으)로 변환하지 못한 경 Undefined 입니다.
기타 값	기타 값	Undefined .

principal()

트리거하는 메시지가 게시된 방식에 따라 디바이스가 인증에 사용하는 위탁자를 반환합니다. 다음 표에서는 각 게시 방법과 프로토콜에 대해 반환된 보안 주체를 설명합니다.

메시지 게시 방법	프로토콜	자격 증명 유형
MQTT 클라이언트	MQTT	X.509 디바이스 인증서
AWS IoT 콘솔 MQTT 클라이언트	MQTT	IAM 사용자 또는 역할
AWS CLI	HTTP	IAM 사용자 또는 역할
AWS IoT 디바이스 SDK	MQTT	X.509 디바이스 인증서
AWS IoT 디바이스 SDK	MQTT over WebSocket	IAM 사용자 또는 역할

다음은 `principal()`이 반환할 수 있는 다양한 유형의 값을 보여주는 예입니다.

- X.509 인증서 지문: `ba67293af50bf2506f5f93469686da660c7c844e7b3950bfb16813e0d31e9373`
- IAM 역할 ID 및 세션 이름: `ABCD1EFG3HIJK2LMNOP5:my-session-name`
- 사용자 ID 반환: `ABCD1EFG3HIJK2LMNOP5`

rand()

0.0~1.0 범위의 균등 분포된 배정밀도 의사 난수를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`rand()` = 0.8231909191640703

regexp_matches(String, String)

문자열(첫 번째 인수)에 정규식(두 번째 인수)과 일치하는 항목이 포함되어 있으면 true를 반환합니다. 정규 표현식에서 | 코드를 사용하는 경우 ()와 함께 사용해야 합니다.

예시:

`regexp_matches("aaaa", "a{2,}")` = true.

`regexp_matches("aaaa", "b")` = false.

`regexp_matches("aaa", "(aaa|bbb)")` = true.

`regexp_matches("bbb", "(aaa|bbb)")` = true.

`regexp_matches("ccc", "(aaa|bbb)")` = false.

첫 번째 인수:

인수 유형	결과
`Int`	`Int`의 `String` 표현.
`Decimal`	`Decimal`의 `String` 표현.
`Boolean`	부울('true' 또는 'false')의 `String` 표현
`String`	`String`.
배열	`Array`의 `String` 표현(표준 변환 규칙 사용).
객체	객체의 `String` 표현(표준 변환 규칙 사용).
Null	`Undefined` .
정의되지 않음	`Undefined` .

두 번째 인수:

유효한 정규식이어야 합니다. 비 문자열 형식은 표준 변환 규칙을 사용하여 String으로 변환됩니다. 형식에 따라 결과 문자열이 유효한 정규식이 아닐 수도 있습니다. (변환된) 인수가 유효한 정규식이 아닐 경우 결과는 Undefined입니다.

regexp_replace(String, String, String)

첫 번째 인수에서 모든 두 번째 인수(정규식)를 세 번째 인수로 대체합니다. "$"를 사용하여 캡처 그룹을 참조합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`regexp_replace("abcd", "bc", "x")` = "axd".

`regexp_replace("abcd", "b(.*)d", "$1")` = "ac".

첫 번째 인수:

인수 유형	결과
Int	Int의 String 표현.
Decimal	Decimal의 String 표현.
Boolean	부울('true' 또는 'false')의 String 표현
String	소스 값
배열	Array의 String 표현(표준 변환 규칙 사용).
객체	객체의 String 표현(표준 변환 규칙 사용).
Null	Undefined .
정의되지 않음	Undefined .

두 번째 인수:

유효한 정규식이어야 합니다. 비 문자열 형식은 표준 변환 규칙을 사용하여 String으로 변환됩니다. 형식에 따라 결과 문자열이 유효한 정규식이 아닐 수도 있습니다. (변환된) 인수가 유효한 정규식이 아닐 경우 결과는 Undefined입니다.

세 번째 인수:

유효한 정규식 대체 문자열이어야 합니다. (캡처 그룹을 참조할 수 있습니다.) 비 문자열 형식은 표준 변환 규칙을 사용하여 String으로 변환됩니다. (변환된) 인수가 유효한 정규식 대체 문자열이 아닐 경우 결과는 Undefined입니다.

regexp_substr(String, String)

첫 번째 파라미터에서 두 번째 파라미터(정규식)의 첫 번째 일치를 찾습니다. "$"를 사용하여 캡처 그룹을 참조합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`regexp_substr("hihihello", "hi")` = "hi"

`regexp_substr("hihihello", "(hi)*")` = "hihi"

첫 번째 인수:

인수 유형	결과
Int	Int의 String 표현.
Decimal	Decimal의 String 표현.
Boolean	부울('true' 또는 'false')의 String 표현
String	String 인수.
배열	Array의 String 표현(표준 변환 규칙 사용).
객체	객체의 String 표현(표준 변환 규칙 사용).
Null	Undefined .
정의되지 않음	Undefined .

두 번째 인수:

유효한 정규식이어야 합니다. 비 문자열 형식은 표준 변환 규칙을 사용하여 String으로 변환됩니다. 형식에 따라 결과 문자열이 유효한 정규식이 아닐 수도 있습니다. (변환된) 인수가 유효한 정규식이 아닐 경우 결과는 Undefined입니다.

나머지(십진수, 십진수)

첫 번째 인수를 두 번째 인수로 나눈 나머지를 반환합니다. mod(Decimal, Decimal)와 동일합니다. 또한 "%"를 동일한 모듈로 기능의 중위 연산자로 사용할 수도 있습니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `remainder(8, 3)` = 2.

왼쪽 피연산자	오른쪽 피연산자	출력
Int	Int	Int, 두 번째 인수를 법으로 하는 수.
Int/Decimal	Int/Decimal	Decimal, 두 번째 피연산자를 법 첫 번째 인수.
String/Int/Decimal	String/Int/Decimal	모든 문자열이 10진수로 변환될 는 두 번째 인수를 법으로 하는 첫 입니다. 그렇지 않을 경우 Undef 다.
기타 값	기타 값	Undefined .

바꾸기(문자열, 문자열, 문자열)

첫 번째 인수에서 모든 두 번째 인수를 세 번째 인수로 대체합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`replace("abcd", "bc", "x")` = `"axd"`.

`replace("abcdabcd", "b", "x")` = `"axcdaxcd"`.

모든 인수

인수 유형	결과
Int	Int의 String 표현.
Decimal	Decimal의 String 표현.
Boolean	부울('true' 또는 'false')의 String 표현
String	소스 값
배열	Array의 String 표현(표준 변환 규칙 사용).
객체	객체의 String 표현(표준 변환 규칙 사용).
Null	Undefined .
정의되지 않음	Undefined .

rpad(String, Int)

두 번째 인수로 지정된 수의 공백이 오른쪽에 추가된 문자열 인수를 반환합니다. Int 인수는 0부터 1000 사이여야 합니다. 제공된 값이 이 유효한 범위를 벗어날 경우 인수가 가장 가까운 유효한 값(0 또는 1000)으로 설정됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`rpad("hello", 2) = "hello  ".`

`rpad(1, 3) = "1  ".`

인수 형식 1	인수 형식 2	결과
String	Int	제공된 Int와 동일한 수의 공백이 String의 오른쪽에 추가됩니다.
String	Decimal	Decimal 인수는 가장 가까운 Int로 내림되

인수 형식 1	인수 형식 2	결과
		고, 문자열은 제공된 Int와 동일한 수의 공백이 오른쪽에 추가됩니다.
String	String	두 번째 인수는 Decimal로 변환된 후 가장 가까운 Int로 내림됩니다. Int 값과 동일한 수의 공백이 String의 오른쪽에 추가됩니다.
기타 값	Int/Decimal/String	첫 번째 값이 표준 변환을 통해 String로 변환된 후, 이 String에 rpad 함수가 적용됩니다. 변환이 불가능한 경우 결과는 Undefined 입니다.
임의의 값	기타 값	Undefined .

round(Decimal)

지정된 Decimal을 가장 가까운 Int로 반올림합니다. Decimal이 두 Int 값과 등거리일 경우(예: 0.5) Decimal은 올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: Round(1.2) = 1.

Round(1.5) = 2.

Round(1.7) = 2.

Round(-1.1) = -1.

Round(-1.5) = -2.

인수 유형	결과
Int	인수.
Decimal	Decimal은 가장 가까운 Int로 내림됩니다.
String	Decimal은 가장 가까운 Int로 내림됩니다. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
기타 값	Undefined .

rtrim(String)

제공된 String에서 모든 후행 공백(탭 및 공백)을 제거합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

```
rtrim(" h i ") = " h i"
```

인수 유형	결과
Int	Int의 String 표현.
Decimal	Decimal의 String 표현.
Boolean	부울('true' 또는 'false')의 String 표현
배열	Array의 String 표현(표준 변환 규칙 사용).
객체	객체의 String 표현(표준 변환 규칙 사용).
Null	Undefined .
정의되지 않음	Undefined

sign(Decimal)

지정된 숫자의 부호를 반환합니다. 인수의 부호가 플러스일 경우 1이 반환됩니다. 인수의 부호가 마이너스일 경우 -1이 반환됩니다. 인수가 0일 경우 0이 반환됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`sign(-7)` = -1.

`sign(0)` = 0.

`sign(13)` = 1.

인수 유형	결과
`Int`	`Int`, `Int` 값의 부호.
`Decimal`	`Int`, `Decimal` 값의 부호.
`String`	`Int`, `Decimal` 값의 부호. 문자열은 `Decimal` 값으로 변환된 후 `Decimal` 값의 부호가 반환됩니다. `String`을 `Decimal`로 변환할 수 없는 경우 결과는 `Undefined` 입니다. SQL 버전 2015-10-08 이상에서 지원됩니다.
기타 값	`Undefined` .

sin(Decimal)

숫자의 사인을 라디안 단위로 반환합니다. `Decimal` 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `sin(0)` = 0.0

인수 유형	결과
`Int`	`Decimal`(배정밀도), 인수의 사인.

인수 유형	결과
Decimal	Decimal(배정밀도), 인수의 사인.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 사인. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
Undefined	Undefined .

sinh(Decimal)

숫자의 쌍곡사인을 반환합니다. Decimal 값은 함수 적용 전에 배정밀도로 반올림됩니다. 결과는 배정밀도의 Decimal 값입니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: sinh(2.3) = 4.936961805545957

인수 유형	결과
Int	Decimal(배정밀도), 인수의 쌍곡사인.
Decimal	Decimal(배정밀도), 인수의 쌍곡사인.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 쌍곡사인. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
배열	Undefined .
객체	Undefined .

인수 유형	결과
Null	`Undefined .`
정의되지 않음	`Undefined .`

sourceip()

연결되는 디바이스 또는 라우터의 IP 주소를 검색합니다. 디바이스가 인터넷에 직접 연결되어 있는 경우 함수는 디바이스의 소스 IP 주소를 반환합니다. 디바이스가 인터넷에 연결되는 라우터에 연결되어 있는 경우 함수는 라우터의 소스 IP 주소를 반환합니다. SQL 버전 2016-03-23에서 지원됩니다. `sourceip()`는 파라미터를 사용하지 않습니다.

> **Important**
>
> 디바이스의 퍼블릭 소스 IP 주소는 대개 마지막 Network Address Translation(NAT) Gateway(예: 인터넷 서비스 공급자의 라우터 또는 케이블 모뎀)의 IP 주소입니다.

예시:

```
sourceip()="192.158.1.38"
```

```
sourceip()="1.102.103.104"
```

```
sourceip()="2001:db8:ff00::12ab:34cd"
```

SQL 예시

```
SELECT *, sourceip() as deviceIp FROM 'some/topic'
```

AWS IoT Core 규칙 작업에서 sourceip() 함수를 사용하는 방법의 예:

예제 1.

다음 예시는 DynamoDB 작업에서 () 함수를 대체 템플릿으로 호출하는 방법을 보여줍니다.

```
{
 "topicRulePayload": {
  "sql": "SELECT * AS message FROM 'some/topic'",
```

```
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "dynamoDB": {
     "tableName": "my_ddb_table",
     "hashKeyField": "key",
     "hashKeyValue": "${sourceip()}",
     "rangeKeyField": "timestamp",
     "rangeKeyValue": "${timestamp()}",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
    }
   }
  ]
 }
}
```

예제 2.

다음 예시는 대체 템플릿을 사용하여 sourceip() 함수를 MQTT 사용자 속성으로 추가하는 방법을 보여 줍니다.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "republish": {
     "topic": "${topic()}/republish",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
     "headers": {
      "payloadFormatIndicator": "UTF8_DATA",
      "contentType": "rule/contentType",
      "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
      "userProperties": [
       {
        "key": "ruleKey1",
        "value": "ruleValue1"
       },
       {
        "key": "sourceip",
```

```
        "value": "${sourceip()}"
       }
      ]
     }
    }
   }
  ]
 }
}
```

메시지 브로커 및 기본 수집 경로 모두에서 AWS IoT Core 규칙에 전달되는 메시지에서 소스 IP 주소를 검색할 수 있습니다. IPv4와 IPv6 메시지에 대한 소스 IP도 검색할 수 있습니다. 소스 IP는 다음과 같이 표시됩니다.

IPv6: `yyyy:yyyy:yyyy::yyyy:yyyy`

IPv4: `xxx.xxx.xxx.xxx`

> **Note**
>
> 원본 소스 IP는 재게시 작업을 거쳐 전달되지 않습니다.

substring(String, Int[, Int])

`String` 이후에 하나 또는 두 개의 `Int` 값이 필요합니다. `String` 및 단일 `Int` 인수일 경우 이 함수는 지정된 `Int` 인덱스(0부터 시작, 포함)부터 `String`의 끝까지의 지정된 `String`의 하위 문자열을 반환합니다. `String` 및 2개 `Int` 인수일 경우 이 함수는 첫 번째 `Int` 인덱스 인수(0부터 시작, 포함)부터 두 번째 `Int` 인덱스 인수(0부터 시작, 미포함)까지의 지정된 `String`의 하위 문자열을 반환합니다. 0보다 작은 인덱스는 0으로 설정됩니다. `String` 길이보다 큰 인덱스는 `String` 길이로 설정됩니다. 3개 인수 버전의 경우, 첫 번째 인덱스가 두 번째 인덱스보다 크거나 같을 경우 결과는 빈 `String`입니다.

제공된 인수가 (*`String`*, *`Int`*) 또는 (*`String`*, *`Int`*, *`Int`*)가 아닐 경우 인수에 표준 변환을 적용하여 올바른 형식으로 변환합니다. 형식을 변환할 수 없을 경우 함수 결과는 `Undefined`입니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`substring("012345", 0)` = "012345".

`substring("012345", 2)` = "2345".

`substring("012345", 2.745)` = "2345".

`substring(123, 2)` = "3".

`substring("012345", -1)` = "012345".

`substring(true, 1.2)` = "rue".

`substring(false, -2.411E247)` = "false".

`substring("012345", 1, 3)` = "12".

`substring("012345", -50, 50)` = "012345".

`substring("012345", 3, 1)` = "".

sql_version()

이 규칙에 지정된 SQL 버전을 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`sql_version()` = "2016-03-23"

sqrt(Decimal)

숫자의 제곱근을 반환합니다. `Decimal` 인수는 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `sqrt(9)` = 3.0.

인수 유형	결과
`Int`	인수의 제곱근.
`Decimal`	인수의 제곱근.
`Boolean`	`Undefined` .
`String`	인수의 제곱근. 문자열을 `Decimal`로 변환할 수 없는 경우 결과는 `Undefined` 입니다.

인수 유형	결과
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

startswith(String, String)

첫 번째 문자열 인수가 두 번째 문자열 인수로 시작하는지 여부를 나타내는 Boolean을 반환합니다. 인수가 Null 또는 Undefined일 경우 결과는 Undefined입니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

startswith("ranger","ran") = true

인수 형식 1	인수 형식 2	결과
String	String	첫 번째 문자열이 두 번째 문자열는지 여부.
기타 값	기타 값	두 인수는 모두 표준 변환 규칙을 문자열로 변환됩니다. 첫 번째 문번째 문자열로 시작하면 true를 인수가 Null 또는 Undefined 과는 Undefined 입니다.

tan(Decimal)

숫자의 탄젠트를 라디안 단위로 반환합니다. Decimal 값은 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: tan(3) = -0.1425465430742778

인수 유형	결과
Int	Decimal(배정밀도), 인수의 탄젠트.
Decimal	Decimal(배정밀도), 인수의 탄젠트.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 탄젠트. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.
배열	Undefined .
객체	Undefined .
Null	Undefined .
정의되지 않음	Undefined .

tanh(Decimal)

숫자의 쌍곡탄젠트를 라디안 단위로 반환합니다. Decimal 값은 함수 적용 전에 배정밀도로 반올림됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: tanh(2.3) = 0.9800963962661914

인수 유형	결과
Int	Decimal(배정밀도), 인수의 쌍곡탄젠트.
Decimal	Decimal(배정밀도), 인수의 쌍곡탄젠트.
Boolean	Undefined .
String	Decimal(배정밀도), 인수의 쌍곡탄젠트. 문자열을 Decimal로 변환할 수 없는 경우 결과는 Undefined 입니다.

인수 유형	결과
배열	`Undefined .`
객체	`Undefined .`
Null	`Undefined .`
정의되지 않음	`Undefined .`

time_to_epoch(String, String)

`time_to_epoch` 함수를 사용하여 타임스탬프 문자열을 Unix Epoch 시간의 밀리초 수로 변환합니다. SQL 버전 2016-03-23 이상에서 지원됩니다. 밀리초를 형식이 지정된 타임스탬프 문자열로 변환하려면 parse_time(String, Long[, String]) 단원을 참조하세요.

`time_to_epoch` 함수는 다음 인수를 사용합니다.

timestamp

(문자열) Unix Epoch 이후 밀리초로 변환할 타임스탬프 문자열입니다. 타임스탬프 문자열에서 시간대를 지정하지 않으면 함수는 UTC 시간대를 사용합니다.

pattern

(문자열) JDK11 Time 형식을 따르는 날짜/시간 패턴입니다.

예시:

`time_to_epoch("2020-04-03 09:45:18 UTC+01:00", "yyyy-MM-dd HH:mm:ss VV")` = 1585903518000

`time_to_epoch("18 December 2015", "dd MMMM yyyy")` = 1450396800000

`time_to_epoch("2007-12-03 10:15:30.592 America/Los_Angeles", "yyyy-MM-dd HH:mm:ss.SSS z")` = 1196705730592

timestamp()

AWS IoT 규칙 엔진에서 관찰한 대로 1970년 1월 1일 목요일 00:00:00 협정 세계시(UTC)부터 현재 타임스탬프를 밀리초 단위로 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예: `timestamp()` = `1481825251155`

topic(Decimal)

규칙을 트리거한 메시지가 전송된 주제를 반환합니다. 지정된 파라미터가 없을 경우 전체 주제가 반환됩니다. `Decimal` 파라미터는 특정 주제 세그먼트를 지정하는데 사용되며, 첫 번째 세그먼트는 1로 지정됩니다. 주제 `foo/bar/baz`의 경우, topic(1)이 `foo`를 반환하고, topic(2)가 `bar`를 반환하는 식입니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

`topic()` = "things/myThings/thingOne"

`topic(1)` = "things"

Basic Ingest를 사용하는 경우 주제(`$aws/rules/`*`rule-name`*)의 최초 접두사는 topic() 함수에 사용할 수 없습니다. 예를 들어 주제는 다음과 같습니다.

`$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`

`topic()` = "Buildings/Building5/Floor2/Room201/Lights"

`topic(3)` = "Floor2"

traceid()

MQTT 메시지의 트레이스 ID(UUID), 또는 메시지가 MQTT를 통해 전송되지 않은 경우 `Undefined`를 반환합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

`traceid()` = "12345678-1234-1234-1234-123456789012"

transform(String, Object, Array)

`Object` 파라미터에서 `Array` 파라미터의 지정된 변환 결과를 포함하는 객체 배열을 반환합니다.

SQL 버전 2016-03-23 이상에서 지원됩니다.

문자열

사용할 변환 모드입니다. 지원되는 변환 모드와 `Object` 및 `Array` 파라미터에서 `Result`를 생성하는 방법은 다음 표를 참조하세요.

객체

`Array`의 각 요소에 적용할 속성을 포함하는 객체입니다.

배열

`Object`의 속성이 적용되는 객체의 배열입니다.

이 배열의 각 객체는 함수의 응답에 있는 객체에 해당합니다. 함수 응답의 각 객체에는 원래 객체에 있는 속성과, `String`에 지정된 변환 모드에 의해 결정되어 `Object`에서 제공한 속성이 포함됩니다.

`String` 파라미터	**`Object`** 파라미터	**`Array`** 파라미터	결과
`enrichArray`	객체	객체 배열	각 객체에 `Array` 파라미터의 요소 특성과 `Object` 파라미터의 특성이 포함된 객체 배열입니다.
기타 값	임의의 값	임의의 값	정의되지 않음

> **Note**
>
> 이 함수에 의해 반환된 배열은 128KiB로 제한됩니다.

변환 함수 예제 1

이 예제에서는 transform() 함수가 데이터 객체와 배열에서 단일 객체 배열을 생성하는 방법을 보여줍니다.

이 예에서 다음 메시지는 MQTT 주제 `A/B`에 게시됩니다.

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
```

```
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

주제 규칙 작업에 대한 이 SQL 문은 `String` 값이 `enrichArray`인 transform() 함수를 사용합니다. 이 예에서 `Object`는 메시지 페이로드의 `attributes` 속성이고 `Array`는 세 개의 객체를 포함하는 `values` 배열입니다.

```
select value transform("enrichArray", attributes, values) from 'A/B'
```

메시지 페이로드를 수신하면 SQL 문은 다음과 같은 응답으로 평가됩니다.

```
[
  {
    "a": 3,
    "data1": 1,
    "data2": 2
  },
  {
    "b": 4,
    "data1": 1,
    "data2": 2
  },
  {
    "c": 5,
    "data1": 1,
    "data2": 2
  }
]
```

변환 함수 예제 2

이 예에서는 transform() 함수가 리터럴 값을 사용하여 메시지 페이로드의 개별 속성을 포함시키고 이름을 바꾸는 방법을 보여줍니다.

이 예에서 다음 메시지는 MQTT 주제 A/B에 게시됩니다. 이는 the section called “변환 함수 예제 1”에 사용된 것과 동일한 메시지입니다.

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

주제 규칙 작업에 대한 이 SQL 문은 String 값이 enrichArray인 transform() 함수를 사용합니다. Object 함수의 transform()에는 메시지 페이로드의 값이 key인 attributes.data1라는 단일 속성이 있고 Array는 이전 예에서 사용된 것과 동일한 세 개의 객체를 포함하는 values 배열입니다.

```
select value transform("enrichArray", {"key": attributes.data1}, values) from 'A/B'
```

메시지 페이로드를 수신하면 이 SQL 문은 다음 응답으로 평가됩니다. 응답에서 data1 속성의 이름이 key로 지정되는 방법에 주목하세요.

```
[
  {
    "a": 3,
    "key": 1
  },
  {
    "b": 4,
    "key": 1
  },
  {
    "c": 5,
```

```
    "key": 1
  }
]
```

변환 함수 예제 3

이 예에서는 중첩된 SELECT 절에서 transform() 함수를 사용하여 여러 속성을 선택하고 후속 처리를 위해 새 객체를 만드는 방법을 보여줍니다.

이 예에서 다음 메시지는 MQTT 주제 A/B에 게시됩니다.

```
{
  "data1": "example",
  "data2": {
    "a": "first attribute",
    "b": "second attribute",
    "c": [
      {
        "x": {
          "someInt": 5,
          "someString": "hello"
        },
        "y": true
      },
      {
        "x": {
          "someInt": 10,
          "someString": "world"
        },
        "y": false
      }
    ]
  }
}
```

이 변환 함수의 Object는 메시지의 data2 객체의 a 및 b 요소를 포함하는, SELECT 문에서 반환된 객체입니다. Array 파라미터는 원본 메시지에서 data2.c 배열의 두 객체로 구성됩니다.

```
select value transform('enrichArray', (select a, b from data2), (select value c from
 data2)) from 'A/B'
```

앞의 메시지를 사용하여 SQL 문은 다음 응답으로 평가됩니다.

```
[
  {
    "x": {
      "someInt": 5,
      "someString": "hello"
    },
    "y": true,
    "a": "first attribute",
    "b": "second attribute"
  },
  {
    "x": {
      "someInt": 10,
      "someString": "world"
    },
    "y": false,
    "a": "first attribute",
    "b": "second attribute"
  }
]
```

이 응답에서 반환된 배열은 batchMode를 지원하는 주제 규칙 작업과 함께 사용할 수 있습니다.

trim(String)

제공된 String에서 모든 선행 및 후행 공백을 제거합니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예제:

Trim(" hi ") = "hi"

인수 유형	결과
Int	모든 선행 및 후행 공백이 제거된 Int의 String 표현.
Decimal	모든 선행 및 후행 공백이 제거된 Decimal의 String 표현.

인수 유형	결과
Boolean	모든 선행 및 후행 공백이 제거된 Boolean("true" 또는 "false")의 String 표현.
String	모든 선행 및 후행 공백이 제거된 String.
배열	Array의 String 표현(표준 변환 규칙 사용).
객체	객체의 String 표현(표준 변환 규칙 사용).
Null	Undefined .
정의되지 않음	Undefined .

trunc(Decimal, Int)

첫 번째 인수를 두 번째 인수로 지정된 Decimal 자리수로 절사합니다. 두 번째 인수가 0보다 작을 경우 0으로 설정됩니다. 두 번째 인수가 34보다 클 경우 34로 설정됩니다. 끝의 0은 결과에서 제거됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

trunc(2.3, 0) = 2.

trunc(2.3123, 2) = 2.31.

trunc(2.888, 2) = 2.88.

trunc(2.00, 5) = 2.

인수 형식 1	인수 형식 2	결과
Int	Int	소스 값
Int/Decimal	Int/Decimal	첫 번째 인수가 두 번째 인수로 지 로 절사됩니다. 두 번째 인수는 I 경우 가까운 Int로 내림됩니다.

인수 형식 1	인수 형식 2	결과
Int/Decimal/String	Int/Decimal	첫 번째 인수가 두 번째 인수로 ㅈ 로 절사됩니다. 두 번째 인수는 I 경우 가까운 Int로 내림됩니다. Decimal 값으로 변환됩니다. 둔 이 실패할 경우 결과는 Undefin
기타 값		Undefined .

upper(String)

지정된 String의 대문자 버전을 반환합니다. 비 String 인수는 표준 변환 규칙을 사용하여 String으로 변환됩니다. SQL 버전 2015-10-08 이상에서 지원됩니다.

예시:

upper("hello") = "HELLO"

upper(["hello"]) = "[\"HELLO\"]"

리터럴

규칙 SQL의 SELECT 및 WHERE 절에서 리터럴 객체를 직접 지정할 수 있습니다. 이는 정보 전달에 유용할 수 있습니다.

> **Note**
>
> 리터럴은 SQL 버전 2016-03-23 이상에서만 사용 가능합니다.

JSON 객체 구문을 사용합니다(키-값 페어, 쉼표로 구분, 키는 문자열이고 값은 JSON 값, 중괄호 {}로 묶임). 다음 예를 참조하세요.

주제 topic/subtopic에 게시된 수신 페이로드: {"lat_long": [47.606,-122.332]}

SQL 문: SELECT {'latitude': get(lat_long, 0),'longitude':get(lat_long, 1)} as lat_long FROM 'topic/subtopic'

결과 송신 페이로드: {"lat_long":{"latitude":47.606,"longitude":-122.332}}.

규칙 SQL의 SELECT 및 WHERE 절에서 배열도 직접 지정할 수 있습니다. 그러면 정보를 그룹화할 수 있습니다. JSON 구문을 사용합니다(쉼표로 구분된 항목을 대괄호 []로 묶어 배열 리터럴을 생성). 다음 예를 참조하세요.

주제 `topic/subtopic`에 게시된 수신 페이로드: `{"lat": 47.696, "long": -122.332}`

SQL 문: `SELECT [lat,long] as lat_long FROM 'topic/subtopic'`

결과 출력 페이로드: `{"lat_long": [47.606,-122.332]}`.

Case 문

Case 문은 switch 문과 같은 분기 실행에 사용할 수 있습니다.

구문:

```
CASE v WHEN t[1] THEN r[1]
  WHEN t[2] THEN r[2] ...
  WHEN t[n] THEN r[n]
  ELSE r[e] END
```

표현식 *v*는 각 WHEN 절의 *t[i]* 값과 일치 여부가 평가됩니다. 일치가 발견될 경우 해당하는 *r[i]* 표현식이 CASE 문의 결과가 됩니다. WHEN 절은 일치하는 절이 두 개 이상 있는 경우 첫 번째 일치 절의 결과가 CASE 문의 결과가 되도록 순서대로 평가됩니다. 일치하는 항목이 없는 경우 ELSE 절의 *r[e]*가 결과입니다. 일치하는 항목과 ELSE 절이 없는 경우 결과는 Undefined입니다.

CASE 문에는 적어도 하나의 WHEN 절이 필요합니다. ELSE 절은 선택 사항입니다.

예:

주제 `topic/subtopic`에 게시된 수신 페이로드:

```
{
    "color":"yellow"
}
```

SQL 문

```
SELECT CASE color
        WHEN 'green' THEN 'go'
```

```
        WHEN 'yellow' THEN 'caution'
        WHEN 'red' THEN 'stop'
        ELSE 'you are not at a stop light' END as instructions
    FROM 'topic/subtopic'
```

결과 출력 페이로드:

```
{
    "instructions":"caution"
}
```

> **Note**
>
> *v*가 Undefined일 경우 case 문의 결과는 Undefined입니다.

JSON 확장

다음과 같은 ANSI SQL 구문 확장을 사용하여 중첩된 JSON 객체 작업을 용이하게 할 수 있습니다.

'.' 연산자

이 연산자는 포함된 JSON 객체의 멤버에 액세스하고 ANSI SQL 및 JavaScript와 동일하게 기능합니다. 다음 예를 참조하세요.

```
SELECT foo.bar AS bar.baz FROM 'topic/subtopic'
```

topic/subtopic 주제로 전송된 다음 메시지 페이로드에서 foo 객체의 bar 속성 값을 선택합니다.

```
{
  "foo": {
    "bar": "RED",
    "bar1": "GREEN",
    "bar2": "BLUE"
  }
}
```

JSON 속성 이름에 하이픈 문자 또는 숫자 문자가 포함되어 있으면 '점' 표기법이 작동하지 않습니다. 대신, 가져오기 함수(get function)를 사용하여 속성 값을 추출할 수 있습니다.

이 예에서 다음 메시지는 `iot/rules` 주제로 전송됩니다.

```
{
  "mydata": {
    "item2": {
      "0": {
        "my-key": "myValue"
      }
    }
  }
}
```

일반적으로, `my-key`의 값은 이 쿼리와 같이 식별됩니다.

```
SELECT * from iot/rules WHERE mydata.item2.0.my-key= "myValue"
```

그러나 속성 이름 `my-key`에는 하이픈이 있고 `item2`에는 숫자 문자가 포함되어 있으면 가져 오기 함수(get function)는 다음 쿼리에서 보여 주는 대로 사용해야 합니다.

```
SELECT * from 'iot/rules' WHERE get(get(get(mydata,"item2"),"0"),"my-key") = "myValue"
```

* 연산자

이 연산자는 ANSI SQL의 * 와일드카드와 동일하게 기능합니다. SELECT 절에서만 사용되며 메시지 데이터를 포함하는 새 JSON 객체를 생성합니다. 메시지 페이로드가 JSON 형식이 아닐 경우 *는 전체 메시지 페이로드를 원시 바이트로 반환합니다. 다음 예를 참조하세요.

```
SELECT * FROM 'topic/subtopic'
```

속성 값에 함수 적용

다음 예는 디바이스가 게시할 수 있는 JSON 페이로드입니다.

```
{
    "deviceid" : "iot123",
    "temp" : 54.98,
    "humidity" : 32.43,
    "coords" : {
        "latitude" : 47.615694,
```

```
        "longitude" : -122.3359976
    }
}
```

다음 예제는 JSON 페이로드의 속성 값에 함수를 적용합니다.

```
SELECT temp, md5(deviceid) AS hashed_id FROM topic/#
```

이 쿼리의 결과는 다음 JSON 객체입니다.

```
{
   "temp": 54.98,
   "hashed_id": "e37f81fb397e595c4aeb5645b8cbbbd1"
}
```

대체 템플릿

대체 템플릿을 사용하여 규칙이 트리거되고 작업을 AWS IoT 수행할 때 반환되는 JSON 데이터를 보강할 수 있습니다. 대체 템플릿의 구문은 ${표현식}입니다. 여기서 표현식은 SELECT 절, WHERE 절 및 AWS IoT 에서가 지원하는 모든 표현식일 수 있습니다AWS IoT 규칙 작업. 이 표현식을 규칙의 작업 필드에 연결하여 작업을 동적으로 구성할 수 있습니다. 실제로 이 기능은 작업의 정보를 대체합니다. 이러한 표현식에는 함수, 연산자, 그리고 원본 메시지 페이로드에서 제공되는 정보가 포함됩니다.

> **Important**
>
> 대체 템플릿의 표현식은 'SELECT ...' 문과 별도로 평가되므로 AS 절을 사용하여 생성한 별칭은 참조할 수 없습니다. 원래 페이로드에 있는 정보, 함수 및 연산자만 참조할 수 있습니다.

지원되는 표현식에 대한 자세한 내용은 AWS IoT SQL 참조 섹션을 참조하세요.

다음 규칙 작업은 대체 템플릿을 지원합니다. 각 작업은 대체할 수 있는 다른 필드를 지원합니다.

- Apache Kafka
- CloudWatch 경보
- CloudWatch Logs
- CloudWatch 지표

- DynamoDB
- DynamoDBv2
- Elasticsearch
- HTTP
- AWS IoT Events
- AWS IoT SiteWise
- Kinesis Data Streams
- Firehose
- Lambda
- Location
- OpenSearch
- 재게시
- S3
- SNS
- SQS
- 단계 함수
- Timestream

대체 템플릿은 규칙 내의 작업 파라미터에 나타납니다.

```
{
    "sql": "SELECT *, timestamp() AS timestamp FROM 'my/iot/topic'",
    "ruleDisabled": false,
    "actions": [{
        "republish": {
            "topic": "${topic()}/republish",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

이 규칙이 `my/iot/topic`에 게시된 다음 JSON에 의해 트리거될 경우:

```
{
```

```
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    }
}
```

그런 ${topic()}/republish다음이 규칙은에서 AWS IoT 대체my/iot/topic/republish하는 다음 JSON을에 게시합니다.

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    },
    "timestamp": 1579637878451
}
```

중첩된 객체 쿼리

중첩된 SELECT 절을 사용하여 배열 및 내부 JSON 객체에서 속성을 쿼리할 수 있습니다. SQL 버전 2016-03-23 이상에서 지원됩니다.

다음 MQTT 메시지를 살펴보겠습니다.

```
{
    "e": [
        { "n": "temperature", "u": "Cel", "t": 1234, "v": 22.5 },
        { "n": "light", "u": "lm", "t": 1235, "v": 135 },
        { "n": "acidity", "u": "pH", "t": 1235, "v": 7 }
    ]
}
```

Example

다음 규칙을 사용하여 값을 새 배열로 변환할 수 있습니다.

```
SELECT (SELECT VALUE n FROM e) as sensors FROM 'my/topic'
```

이 규칙은 다음과 같은 출력을 생성합니다.

```
{
    "sensors": [
        "temperature",
        "light",
        "acidity"
    ]
}
```

Example

동일한 MQTT 메시지에 대해, 다음 규칙을 사용하여 중첩된 객체에서 특정 값을 쿼리할 수도 있습니다.

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

이 규칙은 다음과 같은 출력을 생성합니다.

```
{
    "temperature": [
        {
            "v": 22.5
        }
    ]
}
```

Example

더 복잡한 규칙으로 출력을 평면화할 수도 있습니다.

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'), 0).v as temperature FROM 'topic'
```

이 규칙은 다음과 같은 출력을 생성합니다.

```
{
    "temperature": 22.5
}
```

이진 페이로드 작업

메시지 페이로드를 JSON 객체가 아닌 원시 이진 데이터로 처리해야 하려면 * 연산자를 사용하여 SELECT 절에서 참조할 수 있습니다.

이 주제에서 수행할 작업

- 이진 페이로드 예
- protobuf 메시지 페이로드 디코딩

이진 페이로드 예

*를 사용하여 메시지 페이로드를 원시 이진 데이터로 참조하면 규칙에 데이터를 추가할 수 있습니다. 비어 있거나 JSON 페이로드가 있는 경우 결과 페이로드에 규칙을 사용하여 데이터를 추가할 수 있습니다. 다음은 지원되는 SELECT 절의 예를 보여줍니다.

- 다음 이진 페이로드에 *만 있는 다음 SELECT 절을 사용할 수 있습니다.
  - ```
    SELECT * FROM 'topic/subtopic'
    ```
  - ```
    SELECT * FROM 'topic/subtopic' WHERE timestamp() % 12 = 0
    ```
- 데이터를 추가하고 다음 SELECT 절을 사용할 수도 있습니다.
  - ```
    SELECT *, principal() as principal, timestamp() as time FROM 'topic/subtopic'
    ```
  - ```
    SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'topic/subtopic'
    ```
- 이진 페이로드에 이러한 SELECT 절을 사용할 수도 있습니다.
  - 다음은 WHERE 절에서 `device_type`을 참조합니다.

    ```
    SELECT * FROM 'topic/subtopic' WHERE device_type = 'thermostat'
    ```
  - 다음도 지원됩니다.

    ```
    {
     "sql": "SELECT * FROM 'topic/subtopic'",
     "actions": [
      {
       "republish": {
        "topic": "device/${device_id}"
    ```

```
    }
   }
  ]
}
```

다음 규칙 작업은 이진 페이로드를 지원하지 않으므로 디코딩해야 합니다.

- Lambda 작업과 같은 일부 규칙 작업은 이진 페이로드 입력을 지원하지 않으므로 이진 페이로드를 디코딩해야 합니다. Lambda 규칙 작업은 base64로 인코딩되고 JSON 페이로드에 있는 경우 이진 데이터를 수신할 수 있습니다. 규칙을 다음으로 변경하여 이 작업을 수행할 수 있습니다.

  ```
  SELECT encode(*, 'base64') AS data FROM 'my_topic'
  ```

- SQL 문은 문자열을 입력으로 지원하지 않습니다. 문자열 입력을 JSON으로 변환하려면 다음 명령을 실행합니다.

  ```
  SELECT decode(encode(*, 'base64'), 'base64') AS payload FROM 'topic'
  ```

protobuf 메시지 페이로드 디코딩

프로토콜 버퍼(protobuf)는 구조화된 데이터를 압축 이진 형식으로 직렬화하는 데 사용되는 오픈 소스 데이터 형식입니다. 이는 데이터를 네트워크를 통해 전송하거나 파일에 저장하는 데 사용됩니다. Protobuf를 사용하면 다른 메시징 형식보다 작은 패킷 크기와 빠른 속도로 데이터를 전송할 수 있습니다. AWS IoT Core 규칙은 디코딩(값, 디decodingScheme) SQL 함수를 제공하여 protobuf를 지원하므로 protobuf 인코딩 메시지 페이로드를 JSON 형식으로 디코딩하고 다운스트림 서비스로 라우팅할 수 있습니다. 이 섹션에서는 AWS IoT Core 규칙에서 protobuf 디코딩을 구성하는 단계별 프로세스를 자세히 설명합니다.

이 섹션의 내용:

- 사전 조건
- 설명자 파일 생성
- S3 버킷에 설명자 파일 업로드
- 규칙에서 protobuf 디코딩 구성
- 제한 사항
- 모범 사례

사전 조건

- 프로토콜 버퍼(protobuf) 에 대한 기본적인 이해
- 메시지 유형 및 관련 종속 항목을 정의하는 `.proto` 파일
- 시스템에 Protobuf 컴파일러(protoc) 설치

설명자 파일 생성

설명자 파일이 이미 있는 경우 이 단계를 건너뛰어도 됩니다. 설명자 파일(`.desc`)은 protobuf 직렬화에 사용할 데이터 구조와 메시지 유형을 정의하는 텍스트 파일인 `.proto` 파일의 컴파일된 버전입니다. 설명자 파일을 생성하려면 `.proto` 파일을 정의하고 protoc 컴파일러를 사용하여 컴파일해야 합니다.

1. 메시지 유형을 정의하는 `.proto` 파일을 생성합니다. 예제 `.proto` 파일은 다음과 같습니다.

```
syntax = "proto3";

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

   이 예제 `.proto` 파일에서는 proto3 구문을 사용하고 `Person`이라는 메시지 유형을 정의합니다. `Person` 메시지 정의는 3개의 필드(이름, ID 및 이메일)를 지정합니다. `.proto` 파일 메시지 형식에 대한 자세한 내용은 Language Guide(proto3)(언어 가이드(proto3))를 참조하세요.

2. protoc 컴파일러를 사용하여 `.proto` 파일을 컴파일하고 설명자 파일을 생성합니다. 설명자 (`.desc`) 파일을 생성하는 예제 명령은 다음과 같습니다.

```
protoc --descriptor_set_out=<FILENAME>.desc \
    --proto_path=<PATH_TO_IMPORTS_DIRECTORY> \
    --include_imports \
    <PROTO_FILENAME>.proto
```

   이 예제 명령은에 정의된 데이터 구조를 준수하는 protobuf 페이로드를 디코딩하는 데 사용할 수 있는 `<FILENAME>.desc` AWS IoT Core 설명자 파일을 생성합니다`<PROTO_FILENAME>.proto`.

   - `--descriptor_set_out`

생성할 설명자 파일(<FILENAME>.desc)의 이름을 지정합니다.

- --proto_path

  컴파일되는 파일에서 참조하는 가져온 .proto 파일의 위치를 지정합니다. 가져온 .proto 파일이 여러 개 있고 파일의 위치가 서로 다른 경우 플래그를 여러 번 지정할 수 있습니다.

- --include_imports

  가져온 .proto 파일도 컴파일하여 <FILENAME>.desc 설명자 파일에 포함하도록 지정합니다.

- <PROTO_FILENAME>.proto

  컴파일할 .proto 파일의 이름을 지정합니다.

protoc 참조에 대한 자세한 내용은 API 참조를 참조하세요.

S3 버킷에 설명자 파일 업로드

설명자 파일을 생성한 후 AWS API<FILENAME>.desc, AWS SDK 또는 <FILENAME>.desc를 사용하여 설명자 파일을 Amazon S3 버킷에 업로드합니다 AWS Management Console.

중요 고려 사항

- 규칙을 구성하려는 AWS 리전 것과 동일한의 Amazon S3 버킷 AWS 계정 에 설명자 파일을 업로드해야 합니다.
- S3FileDescriptorSet에서를 읽을 수 있는 AWS IoT Core 액세스 권한을 부여해야 합니다. S3 버킷에서 서버 측 암호화(SSE)가 비활성화되었거나 S3 버킷이 Amazon S3 관리형 키(SSE-S3)를 사용하여 암호화된 경우 추가 정책 구성이 필요하지 않습니다. 이 작업은 다음과 같은 예제 버킷 정책을 사용하여 수행할 수 있습니다.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Statement1",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
```

```
    "Action": "s3:Get*",
                        "Resource": "arn:aws:s3:::<BUCKET NAME>/<FILENAME>.desc"
   }
  ]
}
```

- S3 버킷이 AWS Key Management Service 키(SSE-KMS)를 사용하여 암호화된 경우 S3 버킷에 액세스할 때 키를 사용할 수 있는 AWS IoT Core 권한을 부여해야 합니다. 키 정책에 다음 문을 추가하면 됩니다.

```
{
 "Sid": "Statement1",
 "Effect": "Allow",
 "Principal": {
  "Service": "iot.amazonaws.com"
 },
 "Action": [
  "kms:Decrypt",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
 ],
        "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

}
```

규칙에서 protobuf 디코딩 구성

설명자 파일을 Amazon S3 버킷에 업로드한 후 decode(value, decodingScheme) SQL 함수를 사용하여 protobuf 메시지 페이로드 형식을 디코딩할 수 있는 규칙을 구성합니다. 자세한 함수 서명 및 예시는 AWS IoT SQL 참조의 decode(value, decodingScheme) SQL 함수 섹션에서 찾을 수 있습니다.

다음은 decode(value, decodingScheme) 함수를 사용하는 예시 SQL 표현식입니다.

```
SELECT VALUE decode(*, 'proto', '<BUCKET NAME>', '<FILENAME>.desc', '<PROTO_FILENAME>',
 '<PROTO_MESSAGE_TYPE>') FROM '<MY_TOPIC>'
```

이 예제 표현식에서는

- decode(value, decodingScheme) SQL 함수를 사용하여 *에서 참조하는 바이너리 메시지 페이로드를 디코딩합니다. 이는 바이너리 protobuf로 인코딩된 페이로드이거나 base64로 인코딩된 protobuf 페이로드를 나타내는 JSON 문자열일 수 있습니다.
- 제공된 메시지 페이로드는 `PROTO_FILENAME.proto`에 정의된 `Person` 메시지 유형을 사용하여 인코딩됩니다.
- `BUCKET NAME`이라는 Amazon S3 버킷에는 `PROTO_FILENAME.proto`에서 생성된 `FILENAME.desc`가 포함되어 있습니다.

구성을 완료한 후 규칙이 구독되는 주제에 메시지를 AWS IoT Core 에 게시합니다.

제한 사항

AWS IoT Core 규칙은 다음과 같은 제한 사항이 있는 protobuf를 지원합니다.

- 대체 템플릿 내에서 protobuf 메시지 페이로드를 디코딩하는 것은 지원되지 않습니다.
- protobuf 메시지 페이로드를 디코딩할 때 단일 SQL 표현식 내에서 decode SQL 함수를 최대 두 번 사용할 수 있습니다.
- 최대 인바운드 페이로드 크기는 128KiB(1KiB=1024바이트)이고, 최대 아웃바운드 페이로드 크기는 128KiB이며, Amazon S3 버킷에 저장되는 `FileDescriptorSet` 객체의 최대 크기는 32KiB입니다.
- SSE-C로 암호화된 Amazon S3 버킷은 지원되지 않습니다.

모범 사례

다음은 몇 가지 모범 사례와 문제 해결 팁입니다.

- Amazon S3 버킷에 proto 파일을 백업합니다.

  문제가 발생할 경우에 대비하여 proto 파일을 백업하는 것이 좋습니다. 예를 들어, protoc을 실행할 때 백업 없이 proto 파일을 잘못 수정하면 프로덕션 스택에 문제가 발생할 수 있습니다. Amazon S3 버킷에 파일을 백업하는 방법에는 여러 가지가 있습니다. 예를 들어, S3 버킷에서 버전 관리를 사용할 수 있습니다. Amazon S3 버킷에 파일을 백업하는 방법에 대한 자세한 내용은 Amazon S3 개발자 안내서를 참조하세요.
- 로그 항목을 보도록 AWS IoT 로깅을 구성합니다.

  CloudWatch에서 계정의 AWS IoT 로그를 확인할 수 있도록 AWS IoT 로깅을 구성하는 것이 좋습니다. 규칙의 SQL 쿼리가 외부 함수를 호출하면 AWS IoT Core 규칙은 실패 문제를 해결하는 데 도움

이 FunctionExecution되는 이유 필드가 포함된 eventType의를 사용하여 로그 항목을 생성합니다. 발생할 수 있는 오류에는 Amazon S3 객체를 찾을 수 없거나 protobuf 파일 설명자가 잘못된 것 등이 포함됩니다. AWS IoT 로깅을 구성하고 로그 항목을 확인하는 방법에 대한 자세한 내용은 AWS IoT 로깅 구성 및 규칙 엔진 로그 항목을 참조하세요.

- 새 객체 키를 사용하여 FileDescriptorSet를 업데이트하고 규칙에서 객체 키를 업데이트합니다.

  업데이트된 설명자 파일을 Amazon S3 버킷에 업로드하여 FileDescriptorSet를 업데이트할 수 있습니다. FileDescriptorSet에 대한 업데이트 내용이 반영되는 데 최대 15분이 걸릴 수 있습니다. 이러한 지연을 피하려면 새 객체 키를 사용하여 업데이트된 FileDescriptorSet를 업로드하고 규칙에서 객체 키를 업데이트하는 것이 좋습니다.

SQL 버전

AWS IoT 규칙 엔진은 SQL과 유사한 구문을 사용하여 MQTT 메시지에서 데이터를 선택합니다. SQL 문은 규칙을 설명하는 JSON 문서에서 awsIotSqlVersion 속성으로 지정된 SQL 버전을 기반으로 해석됩니다. JSON 규칙 문서의 구조에 대한 자세한 내용은 규칙 생성 섹션을 참조하세요. awsIotSqlVersion 속성을 사용하면 사용할 AWS IoT SQL 규칙 엔진 버전을 지정할 수 있습니다. 새 버전이 개발될 경우 이전 버전을 계속 사용할 수도 있고 새 버전을 사용하도록 규칙을 변경할 수도 있습니다. 현재 규칙은 규칙 생성 당시의 버전을 계속 사용합니다.

다음 JSON 예제는 awsIotSqlVersion 속성을 사용하여 SQL 버전을 지정하는 방법을 보여줍니다.

```
{
    "sql": "expression",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
        "republish": {
            "topic": "my-mqtt-topic",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

AWS IoT 는 현재 다음 SQL 버전을 지원합니다.

- 2016-03-23 – 2016년 3월 23일 작성된 SQL 버전(권장됨).
- 2015-10-08 – 2015년 10월 8일 작성된 원래 SQL 버전.
- beta – 최신의 베타 SQL 버전. 이 버전으로 인해 규칙에 영향을 미치는 변화가 발생할 수 있습니다.

2016-03-23 SQL 규칙 엔진 버전의 새로운 기능

- 중첩된 JSON 객체 선택에 대한 수정 사항
- 배열 쿼리에 대한 수정 사항
- 객체 내 쿼리 지원. 자세한 내용은 중첩된 객체 쿼리 단원을 참조하세요.
- 최상위 객체로 배열 출력 지원
- JSON 및 비 JSON 형식 데이터에 적용할 수 있는 `encode(value, encodingScheme)` 기능 추가 자세한 내용은 encode 함수를 참조하세요.

최상위 객체로 **`Array`** 출력

이 기능을 사용하여 배열을 최상위 객체로 반환할 수 있습니다. 예를 들어 다음과 같은 MQTT 메시지가 있다고 가정합니다.

```
{
    "a": {"b":"c"},
    "arr":[1,2,3,4]
}
```

그리고 다음과 같은 규칙이 있다고 가정합니다.

```
SELECT VALUE arr FROM 'topic'
```

이 규칙은 다음과 같은 출력을 생성합니다.

```
[1,2,3,4]
```

AWS IoT 디바이스 섀도우 서비스

AWS IoT 디바이스 섀도우 서비스는 AWS IoT 사물 객체에 섀도우를 추가합니다. 섀도우를 사용하면 디바이스가 연결되어 AWS IoT 있는지 여부에 관계없이 앱 및 기타 서비스에서 디바이스의 상태를 사용할 수 있습니다. AWS IoT 사물 객체에는 여러 개의 명명된 섀도우가 있을 수 있으므로 IoT 솔루션에서 디바이스를 다른 앱 및 서비스에 연결할 수 있는 더 많은 옵션이 제공됩니다.

AWS IoT 사물 객체는 명시적으로 생성될 때까지 섀도우가 없습니다. AWS IoT 콘솔을 사용하여 섀도우를 생성, 업데이트 및 삭제할 수 있습니다. 디바이스, 기타 웹 클라이언트 및 서비스는 MQTT 및 예약된 MQTT 주제, 디바이스 섀도우 REST API를 사용한 HTTP 및 AWS IoT의AWS CLI를 사용하여 섀도우를 생성, 업데이트 및 삭제할 수 있습니다. 섀도우는에 의해 클라우드 AWS 에 저장되므로 디바이스 연결 여부에 관계없이 앱 및 기타 클라우드 서비스에서 디바이스 상태 데이터를 수집하고 보고할 수 있습니다.

섀도우 사용

섀도우는 디바이스, 앱 및 기타 클라우드 서비스가 데이터를 공유할 수 있는 신뢰할 수 있는 데이터 스토어를 제공합니다. 이를 통해 디바이스, 앱 및 기타 클라우드 서비스가 디바이스 상태를 유지하면서 연결하거나 연결을 끊을 수 있습니다.

디바이스, 앱 및 기타 클라우드 서비스가 연결되어 AWS IoT있는 동안 섀도우를 통해 디바이스의 현재 상태에 액세스하고 제어할 수 있습니다. 예를 들어 앱은 섀도우를 업데이트하여 디바이스 상태 변경을 요청할 수 있습니다.는 디바이스 변경을 나타내는 메시지를 AWS IoT 게시합니다. 디바이스는 이 메시지를 수신하고, 메시지와 일치하도록 상태를 업데이트하고, 업데이트된 상태로 메시지를 게시합니다. 디바이스 섀도우 서비스는 해당 섀도우에 이 업데이트된 상태를 반영합니다. 앱은 섀도우의 업데이트를 구독하거나 섀도우에 현재 상태를 쿼리할 수 있습니다.

디바이스가 오프라인 상태가 되더라도 앱은 여전히 AWS IoT 및 디바이스의 섀도우와 통신할 수 있습니다. 디바이스가 다시 연결되면 섀도우의 현재 상태를 수신하여 섀도우와 일치하도록 상태를 업데이트한 다음 업데이트된 상태로 메시지를 게시할 수 있습니다. 마찬가지로 앱이 오프라인으로 전환되고 오프라인 상태인 동안 디바이스 상태가 변경된 경우, 디바이스는 섀도우를 업데이트하여 앱이 다시 연결될 때 섀도우에 현재 상태를 쿼리하도록 할 수 있습니다.

디바이스가 자주 오프라인 상태이고 디바이스가 다시 연결된 후 델타 메시지를 수신하도록 디바이스를 구성하려는 경우 영구 세션 기능을 사용할 수 있습니다. 영구 세션 만료 기간에 대한 자세한 내용은 영구 세션 만료 기간을 참조하세요.

명명된 섀도우 또는 명명되지 않은 섀도우 사용 선택

디바이스 섀도우 서비스는 명명된 섀도우 및 명명되지 않은 섀도우, 즉 클래식 섀도우를 지원합니다. 사물 객체의 경우 명명된 섀도우는 여러 개 가질 수 있으며 명명되지 않은 섀도우는 하나만 가질 수 있습니다. 사물 객체는 명명된 예약 섀도우를 가질 수 있습니다. 이 섀도우는 이름을 업데이트할 수 없다는 점을 제외하면 명명된 섀도우와 비슷하게 작동합니다. 자세한 내용은 명명된 예약 섀도우를 참조하세요.

사물 객체는 명명된 섀도우와 명명되지 않은 섀도우를 동시에 가질 수 있습니다. 그러나 각각에 액세스하는 데 사용되는 API는 약간 다르므로 솔루션에 가장 적합한 섀도우 유형을 결정하고 해당 유형만 사용하는 것이 더 효율적일 수 있습니다. 섀도우에 액세스하기 위한 API에 대한 자세한 내용은 섀도우 주제 단원을 참조하세요.

명명된 섀도우를 사용하여 사물 객체의 상태에 대한 다양한 보기를 생성할 수 있습니다. 예를 들어, 많은 속성을 가진 사물 객체를 각각 섀도우 이름으로 식별된 논리적 속성 그룹이 있는 섀도우로 분리할 수 있습니다. 속성을 섀도우별로 그룹화하고 정책을 사용하여 액세스를 제어함으로써 속성에 대한 액세스를 제한할 수도 있습니다. 디바이스 섀도우에 사용하는 정책에 대한 자세한 내용은 AWS IoT에 사용되는 작업, 리소스 및 조건 키와 AWS IoT Core 정책을 참조하세요.

명명되지 않은 클래식 섀도우는 명명된 섀도우보다 더 간단하지만 다소 제한적입니다. 각 AWS IoT 사물 객체에는 이름이 지정되지 않은 섀도우가 하나만 있을 수 있습니다. IoT 솔루션의 섀도우 데이터 사용이 제한적일 것으로 예상되는 경우 이 방법을 사용하여 섀도우 사용을 시작할 수 있습니다. 그러나 나중에 섀도우를 추가하려는 경우 처음부터 명명된 섀도우를 사용하는 것이 좋습니다.

플릿 인덱싱은 명명되지 않은 섀도우와 명명된 섀도우를 다르게 지원합니다. 자세한 정보는 플릿 인덱싱 관리를 참조하세요.

섀도우 액세스

모든 섀도우는 예약된 MQTT 주제 및 섀도우에서 `get`, `update` 및 `delete` 작업을 지원하는 HTTP URL을 보유합니다.

섀도우는 JSON 섀도우 문서를 사용하여 데이터를 저장하고 검색합니다. 섀도우 문서에는 디바이스 상태의 다음 측면을 설명하는 상태 속성이 포함되어 있습니다.

- `desired`

  앱은 `desired` 객체를 업데이트하여 디바이스 속성에 대해 원하는 상태를 지정합니다.

- `reported`

디바이스는 `reported` 객체에 현재 상태를 보고합니다.

- `delta`

  AWS IoT 는 `delta` 객체에서 원하는 상태와 보고된 상태 간의 차이를 보고합니다.

섀도우에 저장된 데이터는 업데이트 작업 메시지 본문의 상태 속성에 의해 결정됩니다. 후속 업데이트 작업은 기존 데이터 객체의 값을 수정하고 섀도우의 상태 객체에서 키 및 기타 요소를 추가 및 삭제할 수 있습니다. 섀도우 액세스에 대한 자세한 내용은 디바이스에서 섀도우 사용 및 앱 및 서비스에서 섀도우 사용 단원을 참조하세요.

> **Important**
>
> 업데이트 요청 권한은 신뢰할 수 있는 앱 및 디바이스로 제한되어야 합니다. 이렇게 하면 섀도우의 상태 속성이 예기치 않게 변경되는 것을 방지할 수 있습니다. 그렇지 않으면 섀도우를 사용하는 디바이스 및 앱이 상태 속성의 키가 변경될 수 있도록 설계되어야 합니다.

디바이스, 앱 및 기타 클라우드 서비스에서 섀도우 사용

디바이스, 앱 및 기타 클라우드 서비스에서 섀도우를 사용하려면 이러한 모든 작업 간에 일관성과 조정이 필요합니다. AWS IoT 디바이스 섀도우 서비스는 섀도우 상태를 저장하고, 섀도우 상태가 변경될 때 메시지를 보내고, 상태를 변경하는 메시지에 응답합니다. IoT 솔루션의 디바이스, 앱 및 기타 클라우드 서비스는 해당 상태를 관리하고 디바이스 섀도우의 상태와 일관성을 유지해야 합니다.

섀도우 상태 데이터는 동적이며 섀도우 액세스 권한이 있는 디바이스, 앱 및 기타 클라우드 서비스에 의해 변경될 수 있습니다. 이러한 이유로 각 디바이스, 앱 및 기타 클라우드 서비스가 섀도우와 상호 작용하는 방식을 고려하는 것이 중요합니다. 다음 예를 참조하세요.

- 섀도우와 상태 데이터를 통신할 때 디바이스는 섀도우 상태의 `reported` 속성에만 기록해야 합니다.
- 앱 및 기타 클라우드 서비스는 섀도우를 통해 디바이스에 상태 변경 요청을 전달할 때 `desired` 속성에만 기록해야 합니다.

> **Important**
>
> 섀도우 데이터 객체에 포함된 데이터는 다른 섀도우 및 사물의 속성 및 사물 객체의 디바이스가 게시할 수 있는 MQTT 메시지의 콘텐츠와 같은 기타 사물 객체 속성과는 독립적입니다. 그

러나 필요한 경우 디바이스는 서로 다른 MQTT 주제 및 섀도우에서 동일한 데이터를 보고할 수 있습니다.
여러 섀도우를 지원하는 디바이스는 서로 다른 섀도우에서 보고하는 데이터의 일관성을 유지해야 합니다.

메시지 순서

AWS IoT 서비스의 메시지가 특정 순서로 디바이스에 도착한다는 보장은 없습니다. 다음 시나리오는 이 경우에 어떤 일이 발생하는지 보여줍니다.

초기 상태 문서:

```
{
  "state": {
    "reported": {
      "color": "blue"
    }
  },
  "version": 9,
  "timestamp": 123456776
}
```

업데이트 1:

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

업데이트 2:

```
{
  "state": {
    "desired": {
```

```
      "color": "GREEN"
    }
  },
  "version": 11,
  "timestamp": 123456778
}
```

최종 상태 문서:

```
{
  "state": {
    "reported": {
      "color": "GREEN"
    }
  },
  "version": 12,
  "timestamp": 123456779
}
```

이로 인해 델타 메시지 2개가 생성됩니다.

```
{
  "state": {
    "color": "RED"
  },
  "version": 11,
  "timestamp": 123456778
}
```

```
{
  "state": {
    "color": "GREEN"
  },
  "version": 12,
  "timestamp": 123456779
}
```

디바이스가 이들 메시지를 틀린 순서로 수신할 수도 있습니다. 이들 메시지에서의 상태는 누적적이므로 디바이스는 현재 추적 중인 것보다 빠른 버전 번호는 무시할 수 있습니다. 디바이스가 버전 11보다 먼저 버전 12의 델타를 수신할 경우 버전 11 메시지는 무시할 수 있습니다.

섀도우 메시지 트리밍

디바이스로 전송되는 섀도우 메시지의 크기를 줄이려면 디바이스에 필요한 필드만 선택하고 디바이스가 수신 대기하는 MQTT 주제에 메시지를 다시 게시하는 규칙을 정의합니다.

이 규칙은 JSON 형식으로 지정되며 다음과 같아야 합니다.

```
{
  "sql": "SELECT state, version FROM '$aws/things/+/shadow/update/delta'",
  "ruleDisabled": false,
  "actions": [
    {
      "republish": {
        "topic": "${topic(3)}/delta",
        "roleArn": "arn:aws:iam:123456789012:role/my-iot-role"
      }
    }
  ]
}
```

SELECT 문은 메시지 가운데 지정된 주제로 재게시할 필드를 결정합니다. "+" 와일드 카드는 모든 섀도우 이름과 일치시키는 데 사용됩니다. 이 규칙은 모든 일치하는 메시지를 지정된 주제로 재게시하도록 지정합니다. 메시지를 재게시할 주제를 지정하는 데는 "topic()" 함수가 사용됩니다. topic(3)은 원래 주제의 사물 이름으로 평가됩니다. 규칙 생성에 대한 자세한 내용은 에 대한 규칙 AWS IoT 단원을 참조하세요.

디바이스에서 섀도우 사용

이 섹션에서는 디바이스가 디바이스 섀도우 서비스와 통신하는 데 선호되는 방법인 MQTT 메시지를 사용한 섀도우와의 AWS IoT 디바이스 통신에 대해 설명합니다.

섀도우 통신은 MQTT의 게시/구독 통신 모델을 사용하여 요청/응답 모델을 에뮬레이트합니다. 모든 섀도우 작업은 요청 주제, 성공한 응답 주제(accepted) 및 오류 응답 주제(rejected)로 구성됩니다.

앱 및 서비스에서 디바이스가 연결되어 있는지 여부를 확인할 수 있도록 하려면 디바이스 연결 상태 감지 단원을 참조하세요.

> **Important**
>
> MQTT는 게시/구독 통신 모델을 사용하기 때문에 요청 주제를 게시하기 전에 응답 주제를 구독해야 합니다. 그렇지 않으면 게시한 요청에 대한 응답을 받지 못합니다.

AWS IoT Device SDK를 사용하여 디바이스 섀도우 서비스 API를 호출하면 이 작업이 자동으로 처리됩니다.

이 섹션의 예에서는 이 표에 설명된 대로 *ShadowTopicPrefix*가 명명된 섀도우 또는 명명되지 않은 섀도우를 참조할 수 있는 주제의 축약 형식을 사용합니다.

섀도우는 명명되거나 명명되지 않을(클래식) 수 있습니다. 각 섀도우에 사용되는 주제는 주제 접두사만 다릅니다. 이 표에서는 각 섀도우 유형에서 사용하는 주제 접두사를 보여줍니다.

ShadowTopicPrefix 값	섀도우 유형
`$aws/things/` *thingName* `/shadow`	명명되지 않은(클래식) 섀도우
`$aws/things/` *thingName* `/shadow/name/` *shadowName*	명명된 섀도우

Important

앱 또는 서비스의 섀도우 사용이 일관되고 디바이스의 해당 구현에 의해 지원되는지 확인합니다. 예를 들어 섀도우가 생성, 업데이트 및 삭제되는 방법을 고려합니다. 또한 디바이스와, 섀도우를 통해 해당 디바이스에 액세스하는 앱 또는 서비스에서 업데이트가 처리되는 방식을 고려합니다. 디바이스의 상태가 업데이트되고 보고되는 방식과 앱 및 서비스가 디바이스 및 섀도우와 상호 작용하는 방식에 대한 설계가 명확해야 합니다.

전체 주제를 생성하려면 참조할 섀도우 유형에 대해 *ShadowTopicPrefix*를 선택하고 *thingName* 및 *shadowName*(해당하는 경우)을 해당 값으로 바꾸고 다음 표에 표시된 대로 주제 스텁을 추가합니다. 주제는 대/소문자를 구분합니다.

섀도우에 대해 예약된 주제에 대한 자세한 내용은 섀도우 주제 단원을 참조하세요.

에 처음 연결할 때 디바이스 초기화 AWS IoT

디바이스가에 등록한 후에는 지원하는 섀도우에 대해 이러한 MQTT 메시지를 구독 AWS IoT해야 합니다.

Topic	의미	이 주제를 수신할 때 디바이스가 수행해야 하는 작업
ShadowTopicPrefix / delete/accepted	delete 요청이 수락되고 섀도우가 AWS IoT 삭제되었습니다.	삭제된 섀도우를 수용하는 데 필요한 작업(예: 업데이트 게시 중지)
ShadowTopicPrefix / delete/rejected	에서 delete 요청을 거부 AWS IoT 했으며 섀도우가 삭제되지 않았습니다. 메시지 본문에 오류 정보가 포함되어 있습니다.	메시지 본문의 오류 메시지에 대한 응답
ShadowTopicPrefix / get/accepted	에서 get 요청을 수락 AWS IoT했으며 메시지 본문에 현재 섀도우 문서가 포함되어 있습니다.	메시지 본문에서 상태 문서를 처리하는 데 필요한 작업
ShadowTopicPrefix / get/rejected	에서 get 요청을 거부 AWS IoT했으며 메시지 본문에 오류 정보가 포함되어 있습니다.	메시지 본문의 오류 메시지에 대한 응답
ShadowTopicPrefix / update/accepted	에서 update 요청을 수락 AWS IoT했으며 메시지 본문에 현재 섀도우 문서가 포함되어 있습니다.	메시지 본문의 업데이트된 데이터가 디바이스 상태와 일치하는지 확인
ShadowTopicPrefix / update/rejected	에서 update 요청을 거부 AWS IoT했으며 메시지 본문에 오류 정보가 포함되어 있습니다.	메시지 본문의 오류 메시지에 대한 응답
ShadowTopicPrefix / update/delta	에 대한 요청에 의해 섀도우 문서가 업데이트되었으며 메시지 본문 AWS IoT에 요청된 변경 사항이 포함되어 있습니다.	메시지 본문의 원하는 상태와 일치하도록 디바이스의 상태 업데이트

Topic	의미	이 주제를 수신할 때 디바이스가 수행해야 하는 작업
ShadowTopicPrefix /update/documents	섀도우 업데이트가 최근에 완료되었으며 메시지 본문에 현재 섀도우 문서가 포함되어 있습니다.	메시지 본문의 업데이트된 상태가 디바이스의 상태와 일치하는지 확인

각 섀도우에 대해 위 표의 메시지를 구독한 후 디바이스는 각 섀도우에 /get 주제를 게시하여 지원하는 섀도우가 이미 생성되었는지 테스트해야 합니다. /get/accepted 메시지가 수신되면 메시지 본문에 섀도우 문서가 포함되며, 디바이스에서 이 문서를 사용하여 해당 상태를 초기화할 수 있습니다. /get/rejected 메시지가 수신되면 현재 디바이스 상태와 함께 /update 메시지를 게시하여 섀도우를 생성해야 합니다.

예를 들어, 클래식 섀도우나 명명된 섀도우가 없는 사물 My_IoT_Thing이 있다고 가정합니다. 이제 예약된 주제 $aws/things/My_IoT_Thing/shadow/get에 대한 /get 요청을 게시하면 사물에 섀도우가 없기 때문에 $aws/things/My_IoT_Thing/shadow/get/rejected 주제에 대한 오류가 반환됩니다. 이 오류를 해결하려면 먼저 다음 페이로드와 같은 현재 디바이스 상태를 포함하는 $aws/things/My_IoT_Thing/shadow/update 주제를 사용하여 /update 메시지를 게시합니다.

```
{
 "state": {
  "reported": {
   "welcome": "aws-iot",
   "color": "yellow"
  }
 }
}
```

이제 사물에 대한 클래식 섀도우가 생성되고 메시지가 $aws/things/My_IoT_Thing/shadow/update/accepted 주제에 게시됩니다. 주제 $aws/things/My_IoT_Thing/shadow/get에 게시하는 경우 디바이스 상태와 함께 $aws/things/My_IoT_Thing/shadow/get/accepted 주제에 대한 응답이 반환됩니다.

명명된 섀도우의 경우 get 요청을 사용하기 전에 먼저 명명된 섀도우를 생성하거나 섀도우 이름이 포함된 업데이트를 게시해야 합니다. 예를 들어 명명된 섀도우 namedShadow1을 생성하려면 먼저 디바이스 상태 정보를 주제 $aws/things/My_IoT_Thing/shadow/name/namedShadow1/update에 게

시합니다. 상태 정보를 검색하려면 명명된 섀도우 `$aws/things/My_IoT_Thing/shadow/name/namedShadow1/get`에 대한 `/get` 요청을 사용합니다.

디바이스가 연결되어 있는 동안 메시지 처리 AWS IoT

디바이스가 연결되어 있는 동안 /update/delta 메시지를 수신할 AWS IoT수 있으며 다음을 통해 디바이스 상태를 섀도우의 변경 사항과 일치해야 합니다.

1. 수신된 모든 /update/delta 메시지를 읽고 일치하도록 디바이스 상태를 동기화합니다.
2. 디바이스의 상태가 변경될 때마다 디바이스의 현재 상태가 포함된 `reported` 메시지 본문과 함께 /update 메시지 게시.

디바이스가 연결되어 있는 동안 표시된 경우 이러한 메시지를 게시해야 합니다.

표시	Topic	Payload
디바이스의 상태가 변경되었습니다.	*ShadowTopicPrefix* `/update`	`reported` 속성이 있는 섀도우 문서
디바이스가 섀도우와 동기화되지 않았을 수 있습니다.	*ShadowTopicPrefix* `/get`	(비어 있음)
디바이스에 대한 작업은 디바이스가 제거되거나 교체되는 경우와 같이 디바이스에서 더 이상 섀도우를 지원하지 않음을 나타냅니다.	*ShadowTopicPrefix* `/delete`	(비어 있음)

디바이스가에 다시 연결될 때 메시지 처리 AWS IoT

하나 이상의 섀도우가 있는 디바이스가 연결되면 다음을 통해 지원하는 모든 섀도우의 상태와 동기화 AWS IoT해야 합니다.

1. 수신된 모든 /update/delta 메시지를 읽고 일치하도록 디바이스 상태를 동기화합니다.
2. 디바이스의 현재 상태가 포함된 `reported` 메시지 본문과 함께 /update 메시지 게시.

앱 및 서비스에서 섀도우 사용

이 섹션에서는 앱 또는 서비스가 AWS IoT 디바이스 섀도우 서비스와 상호 작용하는 방법을 설명합니다. 이 예에서는 앱 또는 서비스가 섀도우만 대상으로 상호 작용하거나 섀도우를 통해 디바이스만 대상으로 상호 작용한다고 가정합니다. 이 예에는 섀도우 생성 또는 삭제와 같은 관리 작업이 포함되지 않습니다.

이 예제에서는 AWS IoT 디바이스 섀도우 서비스의 REST API를 사용하여 섀도우와 상호 작용합니다. 게시/구독 통신 모델을 사용하는 디바이스에서 섀도우 사용의 예제와 달리, 이 예제에서는 REST API의 요청/응답 통신 모델을 사용합니다. 즉, 앱 또는 서비스가 요청을 해야 응답을 받을 수 있습니다 AWS IoT. 그러나 이 모델의 단점은 알림을 지원하지 않는다는 것입니다. 앱 또는 서비스에서 디바이스 상태 변경에 대한 시기 적절한 알림이 필요한 경우 디바이스에서 섀도우 사용에 설명된 대로 게시/구독 통신 모델을 지원하는 MQTT 또는 MQTT over WSS 프로토콜을 고려하세요.

> **Important**
>
> 앱 또는 서비스의 섀도우 사용이 일관되고 디바이스의 해당 구현에 의해 지원되는지 확인합니다. 예를 들어, 섀도우를 생성, 업데이트 및 삭제하는 방법과 디바이스 및 섀도우에 액세스하는 앱 또는 서비스에서 업데이트가 처리되는 방법을 고려합니다. 디바이스의 상태가 업데이트되고 보고되는 방식과 앱 및 서비스가 디바이스 및 섀도우와 상호 작용하는 방식에 대한 설계가 명확해야 합니다.

명명된 섀도우에 대한 REST API의 URL은 다음과 같습니다.

```
https://endpoint/things/thingName/shadow?name=shadowName
```

또한 명명되지 않은 섀도우에 대한 REST API의 URL은 다음과 같습니다.

```
https://endpoint/things/thingName/shadow
```

여기서 각 항목은 다음과 같습니다.

엔드포인트

CLI 명령에 의해 반환된 엔드포인트는 다음과 같습니다.

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

thingName

섀도우가 속한 사물 객체의 이름입니다.

shadowName

명명된 섀도우의 이름입니다. 이 파라미터는 명명되지 않은 섀도우와 함께 사용되지 않습니다.

에 연결할 때 앱 또는 서비스 초기화 AWS IoT

앱이 처음 연결되면 사용 중인 섀도우의 현재 상태를 가져오는 데 사용하는 섀도우의 URLs로 HTTP GET 요청을 보내야 AWS IoT합니다. 이를 통해 앱이나 서비스를 섀도우와 동기화할 수 있습니다.

앱 또는 서비스가에 연결되어 있는 동안 처리 상태 변경 AWS IoT

앱 또는 서비스가 연결되어 있는 동안 사용하는 섀도우의 URLs에 HTTP GET 요청을 전송하여 현재 상태를 주기적으로 쿼리할 AWS IoT수 있습니다.

최종 사용자가 앱 또는 서비스와 상호 작용하여 디바이스의 상태를 변경하는 경우 앱이나 서비스는 섀도우의 `desired` 상태를 업데이트하는 데 사용되는 섀도우의 URL로 HTTP POST 요청을 전송할 수 있습니다. 이 요청은 허용된 변경 사항을 반환하지만 디바이스가 새 상태로 섀도우를 업데이트할 때까지 HTTP GET 요청을 만들어 섀도우를 폴링해야 할 수 있습니다.

디바이스 연결 상태 감지

현재 디바이스가 연결되어 있는지 확인하려면 섀도우 문서에 `connected` 속성을 포함시키고 MQTT LWT(Last Will and Testament) 메시지를 사용하세요. 이 메시지는 디바이스가 오류로 인해 연결 해제될 경우 `connected` 속성을 `false`로 설정합니다.

> **Note**
>
> AWS IoT 예약된 주제($로 시작하는 주제)로 전송된 MQTT LWT 메시지는 AWS IoT 디바이스 섀도우 서비스에서 무시됩니다. 그러나 구독한 클라이언트와 AWS IoT 규칙 엔진에서 처리하므로 예약되지 않은 주제로 전송되는 LWT 메시지와 MQTT LWT 메시지를 섀도우의 예약된 업데이트 주제인에 섀도우 업데이트 메시지로 다시 게시하는 규칙을 생성해야 합니다*ShadowTopicPrefix*/update.

디바이스 섀도우 서비스에 LWT 메시지를 전송하려면

1. 예약된 주제에 대해 MQTT LWT 메시지를 재게시하는 규칙을 생성합니다. 다음 예는 `my/things/myLightBulb/update` 주제에 대한 메시지를 수신하고 `$aws/things/myLightBulb/shadow/update`에 재게시하는 규칙입니다.

```
{
    "rule": {
    "ruleDisabled": false,
    "sql": "SELECT * FROM 'my/things/myLightBulb/update'",
    "description": "Turn my/things/ into $aws/things/",
    "actions": [
        {
        "republish": {
            "topic": "$$aws/things/myLightBulb/shadow/update",
            "roleArn": "arn:aws:iam:123456789012:role/aws_iot_republish"
            }
        }
     ]
   }
}
```

2. 디바이스가 연결되면 재게시 규칙이 인식할 수 있도록 예약되지 않은 주제에 LWT 메시지를 AWS IoT등록합니다. 이 예제에서 해당 주제는 `my/things/myLightBulb/update`이고 연결된 속성을 `false`로 설정합니다.

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

3. 연결 후 디바이스는 섀도우 업데이트 주제인 `$aws/things/myLightBulb/shadow/update`에 메시지를 게시하여 `connected` 속성을 `true`로 설정하는 등 현재 상태를 보고합니다.

```
{
     "state": {
        "reported": {
            "connected":"true"
        }
```

```
    }
}
```

4. 디바이스가 정상적으로 연결 해제되기 전에, 섀도우 업데이트 주제인 `$aws/things/myLightBulb/shadow/update`에 메시지를 게시하여 `connected` 속성을 `false`로 설정하는 등 최신 상태를 보고합니다.

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

5. 오류로 인해 디바이스 연결이 끊어지면 AWS IoT 메시지 브로커가 디바이스를 대신하여 디바이스의 LWT 메시지를 게시합니다. 재게시 규칙은 이 메시지를 감지하고 섀도우 업데이트 메시지를 게시하여 디바이스 섀도우의 `connected` 속성을 업데이트합니다.

> **Note**
>
> 연결 해제 처리의 비동기적 특성으로 인해 LWT 메시지는 재연결 중에 순서대로 디스패치되지 않습니다. 수명 주기 이벤트를 사용하여 연결 상태 감지의 정확도를 높이는 것이 좋습니다. 이러한 이벤트는 out-of-order 이벤트를 관리하는 속성을 제공하기 때문입니다.

디바이스 섀도우 서비스 통신 시뮬레이션

이 주제에서는 디바이스 섀도우 서비스가 중개자 역할을 하는 방법을 보여 주며 디바이스 및 앱에서 섀도우를 사용하여 디바이스 상태를 업데이트, 저장 및 검색할 수 있도록 합니다.

이 주제에 설명된 상호 작용을 시연하고 자세히 살펴보려면 AWS 계정 및를 실행할 수 있는 시스템이 필요합니다 AWS CLI. 이러한 계정과 시스템이 없으면 코드 예제에서 상호 작용을 볼 수 있습니다.

이 예제에서 AWS IoT 콘솔은 디바이스를 나타냅니다. 는 섀도우를 통해 디바이스에 액세스하는 앱 또는 서비스를 AWS CLI 나타냅니다. AWS CLI 인터페이스는 앱이 통신에 사용할 수 있는 API와 매우 유사합니다 AWS IoT. 이 예제의 디바이스는 스마트 전구이며 앱은 전구의 상태를 표시하고 전구의 상태를 변경할 수 있습니다.

시뮬레이션 설정

이 절차에서는 디바이스를 시뮬레이션하는 AWS IoT 콘솔 및 앱을 시뮬레이션하는 명령줄 창을 열어 시뮬레이션을 초기화합니다.

시뮬레이션 환경을 설정하려면

1. 이 주제의 예제를 직접 실행 AWS 계정 하려면이 필요합니다. 이 없는 경우에 설명된 대로 AWS 계정생성합니다설정 AWS 계정.
2. AWS IoT 콘솔을 열고 왼쪽 메뉴에서 테스트를 선택하여 MQTT 클라이언트를 엽니다.
3. 다른 창을 열고 AWS CLI 가 설치된 시스템에서 터미널 창을 엽니다.

두 개의 창이 열려 있어야 합니다. 하나는 테스트 페이지에 AWS IoT 콘솔이 있고 다른 하나는 명령줄 프롬프트가 있습니다.

디바이스 초기화

이 시뮬레이션에서는 mySimulatedThing이라는 이름의 사물 객체와 simShadow1이라는 해당 섀도우로 작업합니다.

사물 객체 및 해당 IoT 정책 생성

사물을 생성하려면, AWS IoT 콘솔에서:

1. 관리(Manage)를 선택한 후 사물(Things)을 선택합니다.
2. 사물이 나열되면 생성 버튼을 클릭합니다. 그렇지 않으면 단일 사물 등록을 클릭하여 단일 AWS IoT 사물을 생성합니다.
3. 이름 `mySimulatedThing`을 입력하고 다른 설정은 기본값으로 그대로 둔 채 다음(Next)을 클릭합니다.
4. 원 클릭 인증서 생성을 사용하여 AWS IoT에 대한 디바이스 연결을 인증할 인증서를 생성합니다. 인증서를 활성화하려면 활성화(Activate)를 클릭합니다.
5. MQTT 예약 주제를 게시 및 구독할 수 있는 권한을 디바이스에 부여하는 정책 `My_IoT_Policy`를 연결할 수 있습니다. AWS IoT 사물을 생성하는 방법과이 정책을 생성하는 방법에 대한 자세한 단계는 섹션을 참조하세요사물 객체 만들기.

사물 객체에 대한 명명된 섀도우 생성

아래에 설명된 대로 주제 `$aws/things/mySimulatedThing/shadow/name/simShadow1/update`에 업데이트 요청을 게시하여 사물에 대한 명명된 섀도우를 생성할 수 있습니다.

또는 명명된 섀도를 생성하려면:

1. AWS IoT 콘솔의 표시된 사물 목록에서 사물 객체를 선택한 다음 섀도우(Shadows)를 선택합니다.
2. 섀도우 추가(Add a shadow)를 선택하고 이름 `simShadow1`을 입력한 다음 생성(Create)을 선택하여 명명된 섀도우를 추가합니다.

예약된 MQTT 주제를 구독하고 주제에 게시

콘솔에서, 예약된 MQTT 섀도우 주제를 구독합니다. 이러한 주제는 `get`, `update` 및 `delete` 작업에 대한 응답으로, 디바이스에서 작업을 게시한 후 응답을 받을 수 있습니다.

MQTT 클라이언트에서 MQTT 주제를 구독하려면

1. MQTT 클라이언트(MQTT client)에서 주제 구독(Subscribe to a topic)을 선택합니다.
2. 구독할 `get`, `update` 및 `delete` 주제를 입력합니다. 다음 목록에서 한 번에 하나의 주제를 복사하여 주제 필터(Topic filter) 필드에 붙여넣고 구독(Subscribe)을 클릭합니다. 구독(Subscriptions) 아래에 주제가 표시되어야 합니다.

   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/documents`

   이 시점에서 시뮬레이션된 디바이스는 AWS IoT에서 게시하는 주제를 수신할 준비가 되었습니다.

MQTT 클라이언트에서 MQTT 주제에 게시하려면

디바이스가 자체적으로 초기화되고 응답 주제를 구독한 후에는 지원하는 섀도우를 쿼리해야 합니다. 이 시뮬레이션은 mySimulatedThing이라는 사물 객체를 지원하는 simShadow1이라는 이름의 섀도우 하나만 지원합니다.

MQTT 클라이언트에서 현재 섀도우 상태를 가져오려면

1. MQTT 클라이언트에서 주제에 게시(Publish to a topic)를 선택합니다.
2. 게시(Publish) 아래에서, 다음 주제를 입력하고 메시지 본문 창에서 가져올 주제의 입력 위치 아래에 있는 내용을 모두 삭제합니다. 그런 다음 주제에 게시(Publish to topic)를 선택하여 요청을 게시합니다. `$aws/things/mySimulatedThing/shadow/name/simShadow1/get`.

   명명된 섀도우 `simShadow1`을 생성하지 않은 경우, 섀도우가 생성되지 않았으므로 이 예제와 같이 `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected` 주제에 메시지가 수신되고 `code`는 `404`입니다. 이것을 다음에 생성하겠습니다.

   ```
   {
     "code": 404,
     "message": "No shadow exists with name: 'simShadow1'"
   }
   ```

디바이스의 현재 상태로 섀도우를 생성하려면

1. MQTT 클라이언트에서, 주제에 게시(Publish to a topic)를 선택하여 다음 주제를 입력합니다.

   ```
   $aws/things/mySimulatedThing/shadow/name/simShadow1/update
   ```

2. 메시지 본문 창에서 주제의 입력 위치 아래에 다음 섀도우 문서를 입력하여 디바이스가 해당 ID와 현재 색상(RGB 값)을 보고하고 있음을 표시합니다. 게시(Publish)를 선택하여 요청을 게시합니다.

   ```
   {
     "state": {
       "reported": {
         "ID": "SmartLamp21",
         "ColorRGB": [
           128,
           128,
           128
         ]
       }
     },
   ```

```
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

다음 주제에서 메시지를 수신한 경우:

- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted: 섀도우가 생성되고 메시지 본문에 현재 섀도우 문서가 포함되어 있음을 의미합니다.
- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected: 메시지 본문의 오류를 검토합니다.
- $aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted: 섀도우가 이미 존재하며 메시지 본문에 다음 예와 같은 현재 섀도우 상태가 포함됩니다. 이를 통해 디바이스를 설정하거나 섀도우 상태와 일치하는지 확인할 수 있습니다.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
```

```
  },
  "version": 3,
  "timestamp": 1591140517,
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

앱에서 업데이트 전송하기

이 섹션에서는 AWS CLI 를 사용하여 앱이 섀도우와 상호 작용하는 방법을 보여줍니다.

를 사용하여 섀도우의 현재 상태를 가져오려면 AWS CLI

명령줄에서 다음 명령을 입력합니다.

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1 /
dev/stdout
```

Windows 플랫폼에서는 /dev/stdout 대신 con을 사용할 수 있습니다.

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1
 con
```

섀도우가 존재하고 현재 상태를 반영하기 위해 디바이스에 의해 초기화되었기 때문에 다음 섀도우 문서를 반환해야 합니다.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
```

```
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591141111
}
```

앱은 이 응답을 사용하여 디바이스 상태의 표현을 초기화할 수 있습니다.

앱이 상태를 업데이트하면(예: 최종 사용자가 스마트 전구의 색상을 노란색으로 변경한 경우) 앱에서 update-thing-shadow 명령을 전송합니다. 이 명령은 UpdateThingShadow REST API에 해당합니다.

앱에서 섀도우를 업데이트하려면

명령줄에서 다음 명령을 입력합니다.

AWS CLI v2.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --cli-binary-format raw-in-base64-out \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

AWS CLI v1.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

성공한 경우 이 명령은 다음 섀도우 문서를 반환해야 합니다.

```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    }
  },
  "version": 4,
  "timestamp": 1591141596,
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

디바이스의 업데이트에 응답

AWS 콘솔에서 MQTT 클라이언트로 돌아가면 이전 섹션에서 실행된 업데이트 명령을 반영하기 위해 가 AWS IoT 게시한 메시지가 표시됩니다.

MQTT 클라이언트에서 업데이트 메시지를 보려면

MQTT 클라이언트의 구독(Subscriptions) 열에서 $aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta를 선택합니다. 주제 이름이 잘리는 경우 커서를 그 위에 잠시 두면 전체 주제 이름이 표시됩니다. 이 주제의 주제 로그에 다음과 유사한 `/delta` 메시지가 표시되어야 합니다.

```
{
```

```
  "version": 4,
  "timestamp": 1591141596,
  "state": {
    "ColorRGB": [
      255,
      255,
      0
    ]
  },
  "metadata": {
    "ColorRGB": [
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      }
    ]
  },
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

디바이스에서 이 메시지의 콘텐츠를 처리하여 디바이스 상태를 메시지의 `desired` 상태와 일치하도록 설정합니다.

디바이스가 메시지의 상태와 일치하도록 `desired` 상태를 업데이트한 후에는 업데이트 메시지를 게시 AWS IoT 하여 새로 보고된 상태를 로 다시 보내야 합니다. 이 절차는 MQTT 클라이언트에서 시뮬레이션합니다.

디바이스에서 섀도우를 업데이트하려면

1. MQTT 클라이언트에서 주제에 게시(Publish to a topic)를 선택합니다.
2. 메시지 본문 창에서 메시지 본문 창 위의 주제 필드에 섀도우의 주제와 `/update` 작업: `$aws/things/mySimulatedThing/shadow/name/simShadow1/update`를 입력하고 메시지 본문에 디바이스의 현재 상태를 설명하는 다음과 같은 업데이트된 섀도우 문서를 입력합니다. 게시(Publish)를 클릭하여 업데이트된 디바이스 상태를 게시합니다.

   ```
   {
     "state": {
   ```

```
    "reported": {
      "ColorRGB": [255,255,0]
      }
  },
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

에서 메시지를 성공적으로 수신한 경우이 예제와 같이 MQTT 클라이언트의 $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted 메시지 로그에 현재 섀도우 상태가 포함된 새 응답이 AWS IoT표시됩니다.

```
{
  "state": {
    "reported": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "reported": {
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

또한 디바이스의 보고된 상태를 성공적으로 업데이트하면 AWS IoT 가 이전 절차에서 디바이스가 수행한 섀도우 업데이트로 인해 발생한이 메시지 본문과 같은 메시지의 섀도우 상태에 대한 포괄적인 설명을 update/documents 주제에 전송합니다.

```
{
  "previous": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          128,
          128,
          128
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
```

```
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          }
        ]
      }
    },
    "version": 4
  },
  "current": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          255,
          255,
          0
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
```

```
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          }
        ]
      }
    },
    "version": 5
  },
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

앱에서 업데이트 관찰

이제 앱이 디바이스에서 보고한 대로 섀도우에 현재 상태를 쿼리할 수 있습니다.

를 사용하여 섀도우의 현재 상태를 가져오려면 AWS CLI

1. 명령줄에서 다음 명령을 입력합니다.

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 /dev/stdout
   ```

   Windows 플랫폼에서는 /dev/stdout 대신 con을 사용할 수 있습니다.

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 con
   ```

2. 현재 상태를 반영하기 위해 디바이스에 의해 섀도우가 방금 업데이트되었기 때문에 다음 섀도우 문서를 반환해야 합니다.

```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    },
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    },
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
```

```
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591143269
}
```

시뮬레이션 이후의 작업

AWS CLI (앱을 나타냄)와 콘솔(디바이스를 나타냄) 사이의 상호 작용을 실험하여 IoT 솔루션을 모델링합니다.

섀도우와의 상호 작용

이 주제에서는 섀도우 작업을 위해 AWS IoT 에서 제공하는 세 가지 메서드와 관련된 메시지에 대해 설명합니다. 이러한 메서드는 다음과 같습니다.

`UPDATE`

섀도우가 없는 경우 섀도우를 생성하거나, 메시지 본문에 제공된 상태 정보로 기존 섀도우의 콘텐츠를 업데이트합니다. AWS IoT 는 각 업데이트와 함께 타임스탬프를 기록하여 상태가 마지막으로 업데이트된 시점을 나타냅니다. 섀도우의 상태가 변경되면는 `desired`와 `reported` 상태 간의 차이가 있는 `/delta` 메시지를 모든 MQTT 구독자에게 AWS IoT 보냅니다. `/delta` 메시지를 수신한 디바이스 또는 앱은 차이에 따라 작업을 수행할 수 있습니다. 예를 들어 디바이스가 원하는 상태로 업데이트할 수도 있고, 앱이 디바이스의 상태 변경을 반영하도록 UI를 업데이트할 수도 있습니다.

`GET`

메타데이터를 포함하여 섀도우의 전체 상태가 포함된 현재 섀도우 문서를 검색합니다.

`DELETE`

디바이스 섀도우와 해당 콘텐츠를 삭제합니다.

삭제된 디바이스 섀도우 문서는 복원할 수 없지만 삭제된 디바이스 섀도우 문서의 이름으로 새 디바이스 섀도우를 생성할 수 있습니다. 지난 48시간 이내에 삭제된 것과 동일한 이름의 디바이스 섀도우 문서를 생성하는 경우 새 디바이스 섀도우 문서의 버전 번호는 삭제된 문서의 버전 번호를 따

릅니다. 디바이스 섀도우 문서가 48시간 이상 삭제된 경우 동일한 이름을 가진 새 디바이스 섀도우 문서의 버전 번호는 0이 됩니다.

프로토콜 지원

AWS IoT 는 섀도우와 상호 작용하기 위해 HTTPS 프로토콜을 통한 MQTT 및 REST API를 지원합니다.는 MQTT 게시 및 구독 작업에 대한 예약 요청 및 응답 주제 세트를 AWS IoT 제공합니다. 디바이스와 앱은 요청 주제를 게시하기 전에 응답 주제를 구독하여에서 요청을 AWS IoT 처리하는 방법에 대한 정보를 확인해야 합니다. 자세한 내용은 디바이스 섀도우 MQTT 주제 및 디바이스 섀도우 REST API 섹션을 참조하세요.

상태 요청 및 보고

AWS IoT 및 섀도우를 사용하여 IoT 솔루션을 설계할 때는 변경을 요청할 앱 또는 디바이스와 이를 구현할 앱을 결정해야 합니다. 일반적으로 디바이스는 변경 사항을 구현하고 섀도우에 다시 보고하며, 앱과 서비스는 섀도우에 응답하고 변경을 요청합니다. 솔루션은 다를 수 있지만 이 주제의 예에서는 클라이언트 앱 또는 서비스가 섀도우의 변경을 요청하고 디바이스가 변경을 수행하며 이를 섀도우에 다시 보고한다고 가정합니다.

섀도우 업데이트

앱이나 서비스는 UpdateThingShadow API를 사용하거나 /update 주제에 게시하여 섀도우의 상태를 업데이트할 수 있습니다. 업데이트는 요청에 지정된 필드에만 영향을 미칩니다.

클라이언트가 상태 변경을 요청할 때 섀도우 업데이트

클라이언트가 MQTT 프로토콜을 사용하여 섀도우의 상태 변경을 요청하는 경우

1. 클라이언트가 변경할 속성을 식별할 수 있도록 현재 섀도우 문서가 있어야 합니다. 현재 섀도우 문서를 가져오는 방법은 /get 작업을 참조하세요.
2. 클라이언트는 다음 MQTT 주제를 구독합니다.

   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`
   - `$aws/things/thingName/shadow/name/shadowName/update/documents`

3. 클라이언트는 원하는 섀도우 상태를 포함하는 상태 문서와 함께 $aws/things/*thingName*/shadow/name/*shadowName*/update 요청 주제를 게시합니다. 변경할 속성만 문서에 포함시켜야 합니다. 다음은 원하는 상태의 문서에 대한 예입니다.

```
{
  "state": {
    "desired": {
      "color": {
        "r": 10
      },
      "engine": "ON"
    }
  }
}
```

4. 업데이트 요청이 유효한 경우는 섀도우에서 원하는 상태를 AWS IoT 업데이트하고 다음 주제에 메시지를 게시합니다.

   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/delta

   /update/accepted 메시지에는 /accepted response state document 섀도우 문서가 포함되어 있으며 /update/delta 메시지에는 /delta response state document 섀도우 문서가 포함되어 있습니다.

5. 업데이트 요청이 유효하지 않은 경우는 오류를 설명하는 오류 응답 문서 섀도우 문서와 함께 $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected 주제가 포함된 메시지를 AWS IoT 게시합니다.

클라이언트가 API를 사용하여 섀도우에서 상태 변경을 요청하는 경우

1. 클라이언트는 요청 상태 문서 상태 문서를 메시지 본문으로 사용하여 UpdateThingShadow API를 호출합니다.
2. 요청이 유효하면는 HTTP 성공 응답 코드와 /accepted response state document 섀도우 문서를 응답 메시지 본문으로 AWS IoT 반환합니다.

   AWS IoT 또한는 구독하는 모든 디바이스 또는 클라이언트에 대한 /delta response state document 섀도우 문서와 함께 $aws/things/*thingName*/shadow/name/*shadowName*/update/delta 주제에 MQTT 메시지를 게시합니다.

3. 요청이 유효하지 않은 경우는 HTTP 오류 응답 코드를 응답 메시지 본문오류 응답 문서으로 AWS IoT 반환합니다.

디바이스가 /update/delta 주제에 대한 /desired 상태를 수신하면 디바이스에서 원하는 변경을 수행합니다. 그런 다음 /update 주제에 메시지를 전송해 현재 상태를 섀도우에 보고합니다.

디바이스가 현재 상태를 보고할 때 섀도우 업데이트

디바이스가 MQTT 프로토콜을 사용하여 섀도우에 현재 상태를 보고하는 경우

1. 디바이스는 섀도우를 업데이트하기 전에 다음 MQTT 주제를 구독해야 합니다.
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/delta
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/documents
2. 디바이스는 이 예에서와 같이 현재 상태를 보고하는 $aws/things/*thingName*/shadow/name/*shadowName*/update 주제에 메시지를 게시하여 현재 상태를 보고합니다.

```
{
    "state": {
        "reported" : {
            "color" : { "r" : 10 },
            "engine" : "ON"
        }
    }
}
```

3. 가 업데이트를 AWS IoT 수락하면 /accepted response state document섀도우 문서와 함께 $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted 주제에 메시지를 게시합니다.
4. 업데이트 요청이 유효하지 않은 경우는 오류를 설명하는 오류 응답 문서 섀도우 문서와 함께 $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected 주제가 포함된 메시지를 AWS IoT 게시합니다.

디바이스가 API를 사용하여 섀도우에 현재 상태를 보고하는 경우

1. 디바이스는 요청 상태 문서 상태 문서를 메시지 본문으로 사용하여 UpdateThingShadow API를 호출합니다.
2. 요청이 유효하면는 섀도우를 AWS IoT 업데이트하고 /accepted response state document 섀도우 문서와 함께 HTTP 성공 응답 코드를 응답 메시지 본문으로 반환합니다.

   AWS IoT 또한는 구독하는 모든 디바이스 또는 클라이언트에 대한 /delta response state document 섀도우 문서와 함께 $aws/things/*thingName*/shadow/name/*shadowName*/update/delta 주제에 MQTT 메시지를 게시합니다.
3. 요청이 유효하지 않은 경우는 HTTP 오류 응답 코드를 응답 메시지 본문오류 응답 문서으로 AWS IoT 반환합니다.

낙관적 잠금

상태 문서 버전을 사용하면 업데이트하는 디바이스 섀도우 문서가 최신 버전인지 확인할 수 있습니다. 업데이트 요청에 버전을 입력하면 상태 문서의 현재 버전이 입력된 버전과 일치하지 않을 경우 서비스가 HTTP 409 충돌 응답 코드를 포함하는 요청을 거부합니다. 충돌 응답 코드는 DeleteThingShadow를 포함하여 ThingShadow를 수정하는 API에서도 발생할 수 있습니다.

예제:

초기 문서:

```
{
  "state": {
    "desired": {
      "colors": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

업데이트: (버전이 일치하지 않음, 이 요청이 거부됨)

```
{
```

```
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 9
}
```

결과:

```
{
  "code": 409,
  "message": "Version conflict",
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

업데이트: (버전이 일치함, 이 요청이 수락됨)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

최종 상태:

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 11
```

```
}
```

섀도우 문서 검색

GetThingShadow API를 사용하거나 /get 주제에 대해 구독 및 게시하여 섀도우 문서를 검색할 수 있습니다. 그러면 desired 및 reported 상태 사이의 델타를 포함하여 전체 섀도우 문서가 검색됩니다. 이 작업의 절차는 요청하는 주체가 디바이스인지 클라이언트인지에 관계없이 동일합니다.

MQTT 프로토콜을 사용하여 섀도우 문서를 검색하려면

1. 디바이스 또는 클라이언트는 섀도우를 업데이트하기 전에 다음 MQTT 주제를 구독해야 합니다.
   - `$aws/things/thingName/shadow/name/shadowName/get/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/get/rejected`
2. 디바이스나 클라이언트는 메시지 본문이 비어 있는 메시지를 `$aws/things/thingName/shadow/name/shadowName/get` 주제에 게시합니다.
3. 요청이 성공하면는 메시지 본문에가 있는 `$aws/things/thingName/shadow/name/shadowName/get/accepted` 주제에 메시지를 AWS IoT 게시/accepted response state document합니다.
4. 요청이 유효하지 않은 경우는 메시지 본문에가 있는 `$aws/things/thingName/shadow/name/shadowName/get/rejected` 주제에 메시지를 AWS IoT 게시오류 응답 문서합니다.

REST API를 사용하여 섀도우 문서를 검색하려면

1. 디바이스 또는 클라이언트는 빈 메시지 본문을 사용하여 GetThingShadow API를 호출합니다.
2. 요청이 유효한 경우는 /accepted response state document 섀도우 문서가 포함된 HTTP 성공 응답 코드를 응답 메시지 본문으로 AWS IoT 반환합니다.
3. 요청이 유효하지 않은 경우는 HTTP 오류 응답 코드를 응답 메시지 본문오류 응답 문서으로 AWS IoT 반환합니다.

섀도우 데이터 삭제

섀도우 데이터를 삭제하는 방법에는 두 가지가 있습니다. 섀도우 문서에서 특정 속성을 삭제하거나 섀도우를 완전히 삭제할 수 있습니다.

- 섀도우에서 특정 속성을 삭제하려면 삭제할 속성의 값을 null로 설정하여 섀도우를 업데이트합니다. 섀도우 문서에서 값이 null인 필드가 제거됩니다.

- 전체 섀도우를 삭제하려면 DeleteThingShadow API를 사용하거나 /delete 주제에 게시합니다.

> **Note**
>
> 섀도를 삭제해도 버전 번호가 0으로 즉시 재설정되지는 않습니다. 48시간 후에 0으로 재설정됩니다.

섀도우 문서에서 속성 삭제

MQTT 프로토콜을 사용하여 섀도우에서 속성을 삭제하려면

1. 디바이스 또는 클라이언트가 변경할 속성을 식별할 수 있도록 현재 섀도우 문서가 있어야 합니다. 현재 섀도우 문서를 가져오는 방법은 섀도우 문서 검색 단원을 참조하세요.
2. 디바이스 또는 클라이언트는 다음 MQTT 주제를 구독합니다.
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
3. 디바이스나 클라이언트는 삭제할 섀도우 속성에 `null` 값을 할당하는 상태 문서와 함께 `$aws/things/thingName/shadow/name/shadowName/update` 요청 주제를 게시합니다. 변경할 속성만 문서에 포함시켜야 합니다. 다음은 `engine` 속성을 삭제하는 문서의 예입니다.

   ```
   {
     "state": {
       "desired": {
         "engine": null
       }
     }
   }
   ```

4. 업데이트 요청이 유효한 경우는 섀도우에서 지정된 속성을 AWS IoT 삭제하고 메시지 본문에 / accepted response state document섀도우 문서가 있는 `$aws/things/thingName/shadow/name/shadowName/update/accepted` 주제와 함께 메시지를 게시합니다.
5. 업데이트 요청이 유효하지 않은 경우는 오류를 설명하는 오류 응답 문서 섀도우 문서와 함께 `$aws/things/thingName/shadow/name/shadowName/update/rejected` 주제가 포함된 메시지를 AWS IoT 게시합니다.

REST API를 사용하여 섀도우에서 속성을 삭제하려면

1. 디바이스 또는 클라이언트는 삭제할 섀도우의 속성에 null 값을 할당하는 요청 상태 문서을 사용하여 UpdateThingShadow API를 호출합니다. 문서에서 삭제할 속성만 포함합니다. 다음은 engine 속성을 삭제하는 문서의 예입니다.

   ```
   {
     "state": {
       "desired": {
         "engine": null
       }
     }
   }
   ```

2. 요청이 유효하면는 HTTP 성공 응답 코드와 /accepted response state document 섀도우 문서를 응답 메시지 본문으로 AWS IoT 반환합니다.
3. 요청이 유효하지 않은 경우는 HTTP 오류 응답 코드를 응답 메시지 본문오류 응답 문서으로 AWS IoT 반환합니다.

섀도우 삭제

다음은 디바이스의 섀도우를 삭제할 때 고려해야 할 몇 가지 사항입니다.

- 디바이스의 섀도우 상태를 null로 설정해도 섀도우가 삭제되지는 않습니다. 섀도우 버전은 다음 업데이트 시 증가합니다.
- 디바이스 섀도우를 삭제해도 사물 객체가 삭제되지는 않습니다. 사물 객체를 삭제해도 해당 디바이스 섀도우는 삭제되지 않습니다.
- 섀도를 삭제해도 버전 번호가 0으로 즉시 재설정되지는 않습니다. 48시간 후에 0으로 재설정됩니다.

MQTT 프로토콜을 사용하여 섀도우를 삭제하려면

1. 디바이스 또는 클라이언트는 다음 MQTT 주제를 구독합니다.

   - $aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected

2. 디바이스 또는 클라이언트는 빈 메시지 버퍼를 사용하여 $aws/things/*thingName*/shadow/name/*shadowName*/delete를 게시합니다.

3. 삭제 요청이 유효한 경우는 섀도우를 AWS IoT 삭제하고 $aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted 주제 및 축약된 /accepted response state document 섀도우 문서가 포함된 메시지를 메시지 본문에 게시합니다. 다음은 수락된 삭제 메시지의 예입니다.

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

4. 업데이트 요청이 유효하지 않은 경우는 오류를 설명하는 오류 응답 문서 섀도우 문서가 포함된 $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected 주제가 포함된 메시지를 AWS IoT 게시합니다.

REST API를 사용하여 섀도우를 삭제하려면

1. 디바이스 또는 클라이언트는 빈 메시지 버퍼를 사용하여 DeleteThingShadow API를 호출합니다.
2. 요청이 유효하면 메시지 본문에 HTTP 성공 응답 코드와 /accepted response state document 및 축약된 /accepted response state document 섀도우 문서를 AWS IoT 반환합니다. 다음은 수락된 삭제 메시지의 예입니다.

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

3. 요청이 유효하지 않은 경우는 HTTP 오류 응답 코드를 응답 메시지 본문오류 응답 문서으로 AWS IoT 반환합니다.

디바이스 섀도우 REST API

섀도우는 상태 정보 업데이트를 위해 다음 URI를 노출시킵니다.

```
https://account-specific-prefix-ats.iot.region.amazonaws.com/things/thingName/shadow
```

엔드포인트는에 고유합니다 AWS 계정. 엔드포인트를 찾으려면 다음을 수행할 수 있습니다.

- AWS CLI에서 describe-endpoint 명령을 사용합니다.

- AWS IoT 콘솔 설정을 사용합니다. 설정(Settings)에서 사용자 지정 엔드포인트(Custom endpoint) 아래에 엔드포인트가 나열됩니다.
- AWS IoT 콘솔 사물 세부 정보 페이지를 사용합니다. 콘솔에서:
  1. 관리(Manage)를 열고 관리(Manage) 아래에서 사물(Things)을 선택합니다.
  2. 사물 목록에서 엔드포인트 URI를 가져올 항목을 선택합니다.
  3. 디바이스 섀도우(Device Shadows) 탭을 선택하고 섀도우를 선택합니다. 디바이스 섀도우 세부 정보 페이지의 디바이스 섀도우 URL 섹션에서 엔드포인트 URI를 볼 수 있습니다.

엔드포인트의 형식은 다음과 같습니다.

```
identifier.iot.region.amazonaws.com
```

섀도우 REST API는 디바이스 통신 프로토콜에서 설명한 것과 동일한 HTTPS 프로토콜/포트 매핑을 따릅니다.

> **Note**
>
> API를 사용하려면 `iotdevicegateway`를 인증을 위한 서비스 이름으로 사용해야 합니다. 자세한 내용은 IoTDataPlane을 참조하세요.

API 작업

- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

API를 사용하여 API의 쿼리 파라미터의 일부로 `name=shadowName`을 제공하여 명명된 섀도우를 생성할 수도 있습니다.

GetThingShadow

지정된 사물의 섀도우를 가져옵니다.

응답 상태 문서는 `desired` 상태와 `reported` 상태 간 델타를 포함합니다.

요청

요청은 표준 HTTP 헤더와 다음 URI를 포함합니다.

```
HTTP GET https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

명명되지 않은(클래식) 섀도우에는 name 쿼리 파라미터가 필요하지 않습니다.

응답

성공 시, 응답은 표준 HTTP 헤더와 다음 코드 및 본문을 포함합니다.

```
HTTP 200
Response Body: response state document
```

자세한 내용은 응답 상태 문서 예제 섹션을 참조하세요.

권한 부여

섀도우를 검색하려면 호출자가 iot:GetThingShadow 작업을 수행하도록 허용하는 정책이 필요합니다. 디바이스 섀도우 서비스는 두 가지 인증 양식, 즉 IAM 자격 증명을 사용하는 서명 버전 4 또는 클라이언트 인증서를 사용하는 TLS 상호 인증을 수용합니다.

다음은 호출자가 디바이스 섀도우를 검색하도록 허용하는 정책 예제입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:GetThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

UpdateThingShadow

지정된 사물의 섀도우를 업데이트합니다.

업데이트는 요청 상태 문서에 지정된 필드에만 영향을 미칩니다. 디바이스 섀도우에서 null 값의 필드가 모두 제거됩니다.

요청

요청은 표준 HTTP 헤더와 다음 URI 및 본문을 포함합니다.

```
HTTP POST https://endpoint/things/thingName/shadow?name=shadowName
Request body: request state document
```

명명되지 않은(클래식) 섀도우에는 name 쿼리 파라미터가 필요하지 않습니다.

자세한 내용은 요청 상태 문서 예제 섹션을 참조하세요.

응답

성공 시, 응답은 표준 HTTP 헤더와 다음 코드 및 본문을 포함합니다.

```
HTTP 200
Response body: response state document
```

자세한 내용은 응답 상태 문서 예제 섹션을 참조하세요.

권한 부여

섀도우를 업데이트하려면 호출자가 iot:UpdateThingShadow 작업을 수행하도록 허용하는 정책이 필요합니다. 디바이스 섀도우 서비스는 두 가지 인증 양식, 즉 IAM 자격 증명을 사용하는 서명 버전 4 또는 클라이언트 인증서를 사용하는 TLS 상호 인증을 수용합니다.

다음은 호출자가 디바이스 섀도우를 업데이트하도록 허용하는 정책 예제입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:UpdateThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
```

```
    ]
}
```

DeleteThingShadow

지정된 사물의 섀도를 삭제합니다.

요청

요청은 표준 HTTP 헤더와 다음 URI를 포함합니다.

```
HTTP DELETE https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

명명되지 않은(클래식) 섀도우에는 name 쿼리 파라미터가 필요하지 않습니다.

응답

성공 시, 응답은 표준 HTTP 헤더와 다음 코드 및 본문을 포함합니다.

```
HTTP 200
Response body: Empty response state document
```

섀도우를 삭제해도 버전 번호가 0으로 재설정되지는 않습니다.

권한 부여

디바이스 섀도우를 삭제하려면 호출자가 `iot:DeleteThingShadow` 작업을 수행하도록 허용하는 정책이 필요합니다. 디바이스 섀도우 서비스는 두 가지 인증 양식, 즉 IAM 자격 증명을 사용하는 서명 버전 4 또는 클라이언트 인증서를 사용하는 TLS 상호 인증을 수용합니다.

다음은 호출자가 디바이스 섀도우를 삭제하도록 허용하는 정책 예제입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteThingShadow",
```

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

ListNamedShadowsForThing

지정한 사물에 대한 섀도우를 나열합니다.

요청

요청은 표준 HTTP 헤더와 다음 URI를 포함합니다.

```
HTTP GET /api/things/shadow/ListNamedShadowsForThing/thingName?
nextToken=nextToken&pageSize=pageSize
Request body: (none)
```

nextToken

다음 결과 집합을 가져오기 위한 토큰입니다.

이 값은 페이징된 결과에서 반환되며 다음 페이지를 반환하는 호출에 사용됩니다.

pageSize

각 호출에서 반환할 섀도우 이름의 수입니다. 또한 nextToken 단원도 참조하세요.

thingName

명명된 섀도우를 나열할 사물의 이름입니다.

응답

성공 시, 응답은 표준 HTTP 헤더와 다음 코드 및 섀도우 이름 목록 응답 문서를 포함합니다.

> **Note**
>
> 명명되지 않은(클래식) 섀도우는 이 목록에 나타나지 않습니다. 클래식 섀도우만 있거나 지정한 thingName이 없는 경우 응답은 빈 목록입니다.

```
HTTP 200
Response body: Shadow name list document
```

권한 부여

디바이스 섀도우 목록을 표시하려면 호출자가 `iot:ListNamedShadowsForThing` 작업을 수행하도록 허용하는 정책이 필요합니다. 디바이스 섀도우 서비스는 두 가지 인증 양식, 즉 IAM 자격 증명을 사용하는 서명 버전 4 또는 클라이언트 인증서를 사용하는 TLS 상호 인증을 수용합니다.

다음은 호출자가 사물의 명명된 섀도우를 나열하도록 허용하는 정책 예제입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:ListNamedShadowsForThing",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

디바이스 섀도우 MQTT 주제

디바이스 섀도우 서비스는 디바이스 및 앱이 디바이스(섀도우)의 상태 정보를 가져오거나 업데이트하거나 삭제할 수 있도록 예약된 MQTT 주제를 사용합니다.

섀도우 주제에 대해 게시 또는 구독하려면 주제 기반 인증이 필요합니다. AWS IoT 은(는) 기존 주제 구조에 새 주제를 추가할 권리를 보유합니다. 그러므로 섀도우 주제에 대한 와일드 카드 구독은 삼가도록 권장합니다. 예를 들어가 새 섀도우 주제를 AWS IoT 도입함에 따라이 주제 필터와 일치하는 주제 수가 증가할 수 `$aws/things/thingName/shadow/#` 있으므로와 같은 주제 필터를 구독하지 마세요. 이러한 주제에 게시되는 메시지의 예는 섀도우와의 상호 작용 섹션을 참조하세요.

섀도우는 명명되거나 명명되지 않을(클래식) 수 있습니다. 각 섀도우에 사용되는 주제는 주제 접두사만 다릅니다. 이 표에서는 각 섀도우 유형에서 사용하는 주제 접두사를 보여줍니다.

ShadowTopicPrefix 값	섀도우 유형
$aws/things/ *thingName* /shadow	명명되지 않은(클래식) 섀도우
$aws/things/ *thingName* /shadow/name/ *shadowName*	명명된 섀도우

전체 주제를 생성하려면 참조할 섀도우 유형에 대해 *ShadowTopicPrefix*를 선택하고 *thingName* 및 *shadowName*(해당하는 경우)을 해당 값으로 바꾸고 다음 섹션에 표시된 대로 주제 스텁을 추가합니다.

다음은 섀도우와 상호 작용하기 위해 사용되는 MQTT 주제입니다.

주제

- /get
- /get/accepted
- /get/rejected
- /update
- /update/delta
- /update/accepted
- /update/documents
- /update/rejected
- /delete
- /delete/accepted
- /delete/rejected

/get

디바이스 섀도우를 가져오려면 이 주제에 빈 메시지를 게시합니다.

```
ShadowTopicPrefix/get
```

AWS IoT 는 /get/accepted 또는에 게시하여 응답합니다/get/rejected.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/get"
            ]
        }
    ]
}
```

/get/accepted

AWS IoT 는 디바이스의 섀도우를 반환할 때이 주제에 응답 섀도우 문서를 게시합니다.

```
ShadowTopicPrefix/get/accepted
```

자세한 내용은 응답 상태 문서 단원을 참조하십시오.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/get/accepted"
            ]
        }
    ]
}
```

/get/rejected

AWS IoT 는 디바이스의 섀도우를 반환할 수 없을 때이 주제에 오류 응답 문서를 게시합니다.

```
ShadowTopicPrefix/get/rejected
```

자세한 내용은 오류 응답 문서 단원을 참조하십시오.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
```

```
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/thingName/
shadow/get/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/shadow/get/
rejected"
      ]
    }
  ]
}
```

/update

디바이스 섀도우를 업데이트하려면 이 주제에 요청 상태 문서를 게시합니다.

```
ShadowTopicPrefix/update
```

메시지 본문에 부분 요청 상태 문서가 포함되어 있습니다.

디바이스의 상태를 업데이트하려고 시도하는 클라이언트는 다음과 같은 `desired` 속성을 가진 JSON 요청 상태 문서를 전송합니다.

```
{
  "state": {
    "desired": {
      "color": "red",
      "power": "on"
    }
  }
}
```

섀도우를 업데이트하는 디바이스는 다음과 같은 `reported` 속성을 가진 JSON 요청 상태 문서를 전송합니다.

```
{
```

```
  "state": {
    "reported": {
      "color": "red",
      "power": "on"
    }
  }
}
```

AWS IoT 는 /update/accepted 또는에 게시하여 응답합니다/update/rejected.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update"
            ]
        }
    ]
}
```

/update/delta

AWS IoT 는 디바이스의 섀도우에 대한 변경을 수락하고 응답 상태 문서에 desired 및 상태에 대한 다양한 값이 포함되어 있을 때이 주제에 응답 reported 상태 문서를 게시합니다.

```
ShadowTopicPrefix/update/delta
```

메시지 버퍼에 /delta response state document가 포함되어 있습니다.

메시지 본문 세부 정보

- update/delta에 게시되는 메시지에는 desired 및 reported 섹션이 서로 다른 경우에만 원하는 속성이 포함됩니다. 이 메시지는 이러한 속성이 현재 업데이트 메시지에 포함되어 있든지, 또는 이미 AWS IoT에 저장되어 있든지 상관없이 모두 포함합니다. desired 및 reported 섹션이 서로 다르지 않은 속성은 포함하지 않습니다.
- 속성이 reported 섹션에는 있지만 desired 섹션에는 동일한 속성이 없는 경우 이 속성은 포함되지 않습니다.
- 속성이 desired 섹션에는 있지만 reported 섹션에는 동일한 속성이 없는 경우 이 속성은 포함됩니다.
- 속성이 reported 섹션에서 삭제되었지만 desired 섹션에는 계속 존재하는 경우 이 속성은 포함됩니다.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/delta"
            ]
```

```
        }
    ]
}
```

/update/accepted

AWS IoT 는 디바이스 섀도우에 대한 변경을 수락할 때이 주제에 응답 상태 문서를 게시합니다.

```
ShadowTopicPrefix/update/accepted
```

메시지 버퍼에 /accepted response state document가 포함되어 있습니다.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/accepted"
            ]
        }
    ]
```

```
}
```

/update/documents

AWS IoT 는 섀도우에 대한 업데이트가 성공적으로 수행될 때마다 상태 문서를이 주제에 게시합니다.

```
ShadowTopicPrefix/update/documents
```

메시지 본문에 /documents response state document가 포함되어 있습니다.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/documents"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/documents"
            ]
        }
    ]
}
```

/update/rejected

AWS IoT 는 디바이스 섀도우에 대한 변경을 거부할 때 오류 응답 문서를이 주제에 게시합니다.

```
ShadowTopicPrefix/update/rejected
```

메시지 본문에 오류 응답 문서가 포함되어 있습니다.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/rejected"
            ]
        }
    ]
}
```

/delete

디바이스 섀도우를 삭제하려면 삭제 주제에 빈 메시지를 게시합니다.

```
ShadowTopicPrefix/delete
```

메시지의 내용은 무시됩니다.

섀도우를 삭제해도 버전 번호가 0으로 재설정되지는 않습니다.

AWS IoT 는 /delete/accepted 또는에 게시하여 응답합니다/delete/rejected.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete"
            ]
        }
    ]
}
```

/delete/accepted

AWS IoT 는 디바이스의 섀도우가 삭제될 때이 주제에 메시지를 게시합니다.

```
ShadowTopicPrefix/delete/accepted
```

예제 정책

다음은 필요한 정책의 예입니다.

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/delete/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete/accepted"
            ]
        }
    ]
}
```

/delete/rejected

AWS IoT 는 디바이스의 섀도우를 삭제할 수 없을 때이 주제에 오류 응답 문서를 게시합니다.

```
ShadowTopicPrefix/delete/rejected
```

메시지 본문에 오류 응답 문서가 포함되어 있습니다.

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete/rejected"
            ]
        }
    ]
}
```

디바이스 섀도우 서비스 문서

디바이스 섀도우 서비스는 JSON 사양의 모든 규칙을 준수합니다. 값, 객체 및 배열이 디바이스 섀도우 문서에 저장됩니다.

내용

- 섀도우 문서 예제
- 문서 속성
- 델타 상태
- 섀도우 문서 버전 관리
- 섀도우 문서의 클라이언트 토큰
- 빈 섀도우 문서 속성
- 섀도우 문서의 배열 값

섀도우 문서 예제

디바이스 섀도우 서비스는 REST API 또는 MQTT Pub/Sub 메시지를 사용한 UPDATE, GET 및 DELETE 작업에서 다음 문서를 사용합니다.

예제

- 요청 상태 문서
- 응답 상태 문서
- 오류 응답 문서
- 섀도우 이름 목록 응답 문서

요청 상태 문서

요청 상태 문서의 형식은 다음과 같습니다.

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "clientToken": "token",
    "version": version
}
```

- `state` — 업데이트는 지정된 필드에만 영향을 미칩니다. 일반적으로 `desired` 또는 `reported` 속성 중 하나를 사용하지만 동일한 요청에서는 둘 다 사용하지는 않습니다.
  - `desired` — 디바이스에서 업데이트하도록 요청된 상태 속성 및 값입니다.
  - `reported` — 디바이스에서 보고한 상태 속성 및 값입니다.
- `clientToken` — 사용할 경우, 클라이언트 토큰으로 요청과 해당 응답을 일치시킬 수 있습니다.

- `version` — 사용할 경우, 디바이스 섀도우 서비스는 지정된 버전이 서비스가 보유하는 최신 버전과 일치하는 경우에만 업데이트를 처리합니다.

응답 상태 문서

응답 상태 문서의 형식은 응답 유형에 따라 다음과 같습니다.

/accepted response state document

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/delta response state document

```
{
    "state": {
        "attribute1": integer2,
```

```
        "attribute2": "string2",
        ...
        "attributeN": boolean2
    },
    "metadata": {
        "attribute1": {
            "timestamp": timestamp
        },
        "attribute2": {
            "timestamp": timestamp
        },
        ...
        "attributeN": {
            "timestamp": timestamp
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/documents response state document

```
{
  "previous" : {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
```

```
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version-1
},
"current": {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
```

```
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version
  },
  "timestamp": timestamp,
  "clientToken": "token"
}
```

응답 상태 문서 속성

- previous — 성공적으로 업데이트되면 업데이트 전 객체의 state를 포함합니다.
- current — 성공적으로 업데이트되면 업데이트 후 객체의 state를 포함합니다.
- state
  - reported — 사물이 reported 섹션의 데이터를 보고하고 요청 상태 문서에 있는 필드만 포함하는 경우에만 존재합니다.
  - desired — 디바이스가 desired 섹션의 데이터를 보고하고 요청 상태 문서에 있는 필드만 포함하는 경우에만 존재합니다.
- metadata — 언제 상태가 업데이트되었는지 확인할 수 있도록 desired 및 reported 섹션의 각 속성에 대한 타임스탬프를 포함합니다.
- timestamp - 응답이 생성된 Epoch 날짜 및 시간입니다 AWS IoT.

- clientToken — 유효한 JSON을 /update 주제에 게시할 때 클라이언트 토큰이 사용된 경우에만 존재합니다.
- version — AWS IoT에서 공유되는 디바이스 섀도우에 대한 문서의 현재 버전입니다. 현재 버전은 이전 문서 버전보다 1이 증가합니다.

오류 응답 문서

오류 응답 문서의 형식은 다음과 같습니다.

```
{
    "code": error-code,
    "message": "error-message",
    "timestamp": timestamp,
    "clientToken": "token"
}
```

- code — 오류 유형을 나타내는 HTTP 응답 코드입니다.
- message — 추가 정보를 제공하는 텍스트 메시지입니다.
- timestamp - 응답이 생성된 날짜와 시간입니다 AWS IoT. 이 속성은 모든 오류 응답 문서에 존재하지 않습니다.
- clientToken — 게시된 메시지에 클라이언트 토큰이 사용된 경우에만 존재합니다.

자세한 내용은 디바이스 섀도우 오류 메시지 단원을 참조하세요.

섀도우 이름 목록 응답 문서

섀도우 이름 목록 응답 문서의 형식은 다음과 같습니다.

```
{
    "results": [
        "shadowName-1",
        "shadowName-2",
        "shadowName-3",
        "shadowName-n"
    ],
    "nextToken": "nextToken",
    "timestamp": timestamp
}
```

- `results` — 섀도우 이름의 배열입니다.
- `nextToken` — 시퀀스의 다음 페이지를 가져오기 위해 페이징된 요청에 사용할 토큰 값입니다. 반환할 섀도우 이름이 더 이상 없는 경우에는 이 속성이 존재하지 않습니다.
- `timestamp` - 응답이 생성된 날짜와 시간입니다 AWS IoT.

문서 속성

디바이스 섀도우 문서의 속성은 다음과 같습니다.

`state`

`desired`

디바이스의 원하는 상태입니다. 앱은 문서의 이 부분에 기록하여, 디바이스와 연결할 필요 없이 디바이스의 상태를 직접 업데이트할 수 있습니다.

`reported`

디바이스의 보고된 상태입니다. 디바이스는 문서의 이 부분을 기록하여 새 상태를 보고합니다. 앱은 문서의 이 부분을 읽어 디바이스의 마지막 보고 상태를 확인합니다.

`metadata`

문서의 `state` 섹션에 저장된 데이터에 대한 정보입니다. 여기에는 `state` 섹션의 각 속성에 대한 Epoch 시간 형식의 타임스탬프가 포함되는데, 이를 통해 각 속성이 업데이트된 시간을 알 수 있습니다.

> **Note**
>
> 메타데이터는 서비스 제한 또는 요금에 대한 문서 크기에 도움이 되지 않습니다. 자세한 내용은 AWS IoT 서비스 한도 단원을 참조하세요.

`timestamp`

메시지가 전송된 시기를 나타냅니다 AWS IoT. 메시지 내 타임스탬프와 `desired` 또는 `reported` 섹션의 개별 속성에 대한 타임스탬프를 사용하면 디바이스에 내장 클록이 없어도 디바이스에서 속성의 사용 기간을 확인할 수 있습니다.

`clientToken`

MQTT 환경에서 응답을 요청에 연결할 수 있는 디바이스 고유 문자열입니다.

`version`

문서 버전입니다. 문서가 업데이트될 때마다 이 버전 번호가 증가합니다. 업데이트되는 문서의 버전이 최신인지 확인하는 데 사용됩니다.

자세한 내용은 섀도우 문서 예제 단원을 참조하세요.

델타 상태

델타 상태는 `desired` 상태와 `reported` 상태 간 차이를 포함하는 가상의 상태 유형입니다. `desired` 섹션에는 있지만 `reported` 섹션에는 없는 필드가 델타에 포함됩니다. `reported` 섹션에는 있고 `desired` 섹션에는 없는 필드는 델타에 포함되지 않습니다. 델타는 메타데이터를 포함하며, 해당 값은 `desired` 필드 내 메타데이터와 동일합니다. 다음 예를 참조하세요.

```
{
  "state": {
    "desired": {
      "color": "RED",
      "state": "STOP"
    },
    "reported": {
      "color": "GREEN",
      "engine": "ON"
    },
    "delta": {
      "color": "RED",
      "state": "STOP"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 12345
      },
      "state": {
        "timestamp": 12345
      }
      },
      "reported": {
        "color": {
          "timestamp": 12345
        },
```

```
        "engine": {
          "timestamp": 12345
        }
      },
      "delta": {
        "color": {
          "timestamp": 12345
        },
        "state": {
          "timestamp": 12345
        }
      }
    },
    "version": 17,
    "timestamp": 123456789
  }
}
```

중첩된 객체가 다를 경우 델타는 루트까지의 전체 경로를 포함합니다.

```
{
  "state": {
    "desired": {
      "lights": {
        "color": {
          "r": 255,
          "g": 255,
          "b": 255
        }
      }
    },
    "reported": {
      "lights": {
        "color": {
          "r": 255,
          "g": 0,
          "b": 255
        }
      }
    },
    "delta": {
      "lights": {
        "color": {
```

```
          "g": 255
        }
      }
    }
  },
  "version": 18,
  "timestamp": 123456789
}
```

디바이스 섀도우 서비스는 desired 상태의 각 필드를 reported 상태와 비교하여 델타를 계산합니다.

배열은 값처럼 취급됩니다. desired 섹션의 한 배열이 reported 섹션의 배열과 일치하지 않을 경우 원하는 배열 전체가 델타로 복사됩니다.

섀도우 문서 버전 관리

디바이스 섀도우 서비스는 요청과 응답의 모든 업데이트 메시지에 대한 버전 관리를 지원합니다. 즉, 섀도우가 업데이트될 때마다 JSON 문서의 버전이 증가합니다. 이를 통해 2가지가 보장됩니다.

- 클라이언트가 구 버전을 사용하여 섀도우를 덮어쓰려 할 경우 오류 응답을 받을 수 있습니다. 클라이언트에 디바이스 섀도우를 업데이트하기 전에 재동기화해야 한다는 알림이 제공됩니다.
- 수신된 메시지가 클라이언트에 저장된 것보다 낮은 버전일 경우 클라이언트는 메시지에 대한 작업을 수행하지 않기로 결정할 수 있습니다.

클라이언트는 섀도우 문서에 버전을 포함하지 않음으로써 버전 일치를 무시할 수 있습니다.

섀도우 문서의 클라이언트 토큰

클라이언트 토큰을 MQTT 기반 메시징과 함께 사용하여 요청 및 요청 응답에 동일한 클라이언트 토큰이 포함되었는지 확인할 수 있습니다. 이는 응답과 요청이 연결되도록 합니다.

> **Note**
>
> 클라이언트 토큰은 64바이트보다 길면 안 됩니다. 클라이언트 토큰이 64바이트보다 길면 400(잘못된 요청) 응답과 잘못된 클라이언트 토큰 오류 메시지가 표시됩니다.

빈 섀도우 문서 속성

섀도우 문서의 reported 및 desired 속성은 비어 있거나 현재 섀도우 상태에 적용되지 않을 경우 생략할 수 있습니다. 예를 들어 섀도우 문서는 원하는 상태가 있을 경우에만 desired 속성을 포함합니다. 예를 들어 다음은 유효한 상태 문서이지만 desired 속성이 없습니다.

```
{
    "reported" : { "temp": 55 }
}
```

reported 속성은 디바이스에 의해 섀도우가 업데이트되지 않은 경우와 같이 비어 있을 수도 있습니다.

```
{
    "desired" : { "color" : "RED" }
}
```

업데이트로 인해 desired 또는 reported 속성이 null이 될 경우 해당 속성이 문서에서 제거됩니다. 다음은 null로 설정하여 desired 속성을 제거하는 방법을 보여줍니다. 예를 들어 디바이스가 상태를 업데이트할 때 이 작업을 수행할 수 있습니다.

```
{
    "state": {
        "reported": {
            "color": "red"
        },
        "desired": null
    }
}
```

섀도우 문서에는 desired 또는 reported 속성이 없을 수도 있으므로 섀도우 문서를 비워 둘 수 있습니다. 비어 있지만 유효한 섀도우 문서의 예입니다.

```
{
}
```

섀도우 문서의 배열 값

섀도우는 배열을 지원하지만, 배열에 대한 업데이트가 전체 배열을 대체한다는 점에서 배열을 일반적인 값으로 취급합니다. 배열을 일부만 업데이트할 수는 없습니다.

Initial state:

```
{
    "desired" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

업데이트:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

최종 상태:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

배열은 null 값을 가질 수 없습니다. 예를 들어 다음 배열은 유효하지 않으므로 거부됩니다.

```
{
    "desired" : {
        "colors" : [ null, "RED", "GREEN" ]
    }
}
```

디바이스 섀도우 오류 메시지

디바이스 섀도우 서비스는 상태 문서에 대한 변경 시도가 실패할 경우 (MQTT를 통해) 메시지를 오류 주제에 게시합니다. 이 메시지는 예약된 $aws 주제 중 하나로의 게시 요청에 대한 응답으로만 생성됩니다. 클라이언트가 REST API를 사용하여 문서를 업데이트하는 경우에는 클라이언트가 응답의 일부로 HTTP 오류 코드를 수신하며 MQTT 오류 메시지는 생성되지 않습니다.

HTTP 오류 코드	오류 메시지
400(잘못된 요청)	• 잘못된 JSON • 필수 노드 누락: 상태 • 상태 노드는 객체여야 함

HTTP 오류 코드	오류 메시지
	• 원하는 노드는 객체여야 함 • 보고된 노드는 객체여야 함 • 잘못된 버전 • 잘못된 클라이언트 토큰 **Note** 클라이언트 토큰이 64바이트보다 길면 이 응답이 표시됩니다. • JSON이 너무 많은 수준의 중첩을 포함. 최대 6개 수준 • 상태가 잘못된 노드를 포함
401(권한이 없음)	• 권한이 없음
403(금지됨)	• 금지됨
404(찾을 수 없음)	• 사물을 찾을 수 없음 • 이름이 *shadowName* 인 섀도우 없음
409(충돌)	• 버전 충돌
413(페이로드가 너무 큼)	• 페이로드가 허용된 최대 크기를 초과함
415(지원되지 않는 미디어 유형)	• 지원되지 않는 문서 인코딩. 지원되는 인코딩은 UTF-8
429(요청이 너무 많음)	• 단일 연결에서 처리 중 요청이 10개를 초과할 경우 디바이스 섀도우 서비스가 이 오류 메시지를 생성합니다. 인플라이트 요청은 시작되었지만 아직 완료되지 않고 진행 중인 요청입니다.
500(내부 서버 오류)	• 내부 서비스 실패

AWS IoT Device Management 소프트웨어 패키지 카탈로그

AWS IoT Device Management 소프트웨어 패키지 카탈로그를 사용하면 소프트웨어 패키지 및 해당 버전의 인벤토리를 유지할 수 있습니다. 패키지 버전을 개별 사물 및 AWS IoT 동적 사물 그룹에 연결하고 사내 프로세스 또는 AWS IoT 작업을 통해 배포할 수 있습니다.

소프트웨어 패키지는 하나의 단위로 배포할 수 있는 파일 모음인 패키지 버전을 하나 이상 포함합니다. 패키지 버전에는 펌웨어, 운영 체제 업데이트, 디바이스 애플리케이션, 구성 및 보안 패치가 포함될 수 있습니다. 시간이 흘러 소프트웨어가 발전하면 새 패키지 버전을 생성하여 플릿에 배포할 수 있습니다.

AWS IoT 소프트웨어 패키지 허브는 내부에 있습니다 AWS IoT Core. 이 허브를 사용하여 소프트웨어 패키지 인벤토리 및 메타데이터를 중앙에서 등록 및 유지 관리할 수 있으며, 이렇게 하면 소프트웨어 패키지 및 해당 버전의 카탈로그가 생성됩니다. 디바이스에 배포된 소프트웨어 패키지 및 패키지 버전을 기준으로 디바이스를 그룹화할 수 있습니다. 이 기능을 사용하면 디바이스 측 패키지 인벤토리를 명명된 섀도우로 유지하고, 버전을 기반으로 디바이스를 연결 및 그룹화하고, 플릿 메트릭을 사용하여 플릿 전체의 패키지 버전 분포를 시각화할 수 있습니다.

사내 소프트웨어 배포 시스템을 구축한 경우 해당 프로세스를 계속 사용하여 패키지 버전을 배포할 수 있습니다. 배포 프로세스를 설정하지 않았거나 원하는 경우 AWS IoT 작업을 사용하여 소프트웨어 패키지 카탈로그의 기능을 사용하는 것이 좋습니다. 자세한 내용은 AWS IoT 작업 준비를 참조하세요.

이번 장은 다음과 같은 단원들로 구성되어 있습니다.

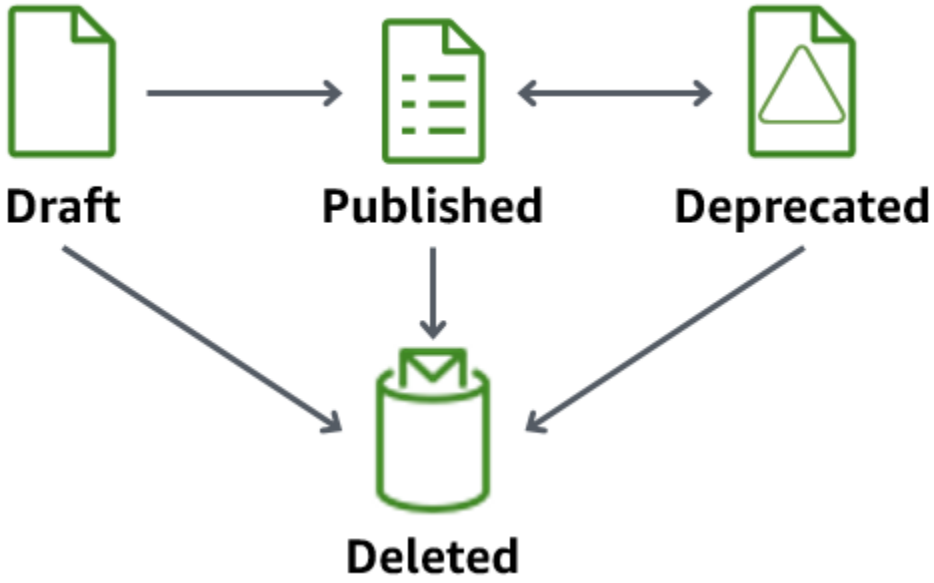
- 소프트웨어 패키지 카탈로그 사용 준비
- 보안 준비
- 플릿 인덱싱 준비
- AWS IoT 작업 준비
- 소프트웨어 패키지 카탈로그 시작

소프트웨어 패키지 카탈로그 사용 준비

다음 섹션에서는 패키지 버전 수명 주기에 대한 개요와 AWS IoT Device Management 소프트웨어 패키지 카탈로그 사용에 대한 정보를 제공합니다.

패키지 버전 수명 주기

패키지 버전은 `draft`, `published`, 및 `deprecated`와 같은 수명 주기 상태를 따라 변화할 수 있습니다. `deleted` 상태일 수도 있습니다.



- Draft

  패키지 버전이 생성되면 `draft` 상태가 됩니다. 이 상태는 소프트웨어 패키지가 준비 중이거나 불완전함을 나타냅니다.

  패키지 버전이 이 상태인 동안에는 배포할 수 없습니다. 패키지 버전의 설명, 속성 및 태그를 편집할 수 있습니다.

  콘솔을 사용하거나 UpdatePackageVersion 또는 DeletePackageVersion API 작업을 실행하여, 현재 `draft` 상태에 있는 패키지 버전을 `published` 상태로 전환하거나 `deleted` 상태로 만들 수 있습니다.

- Published

  패키지 버전을 배포할 준비가 되면 패키지 버전을 `published` 상태로 전환하세요. 이 상태에서는 콘솔에서 소프트웨어 패키지를 편집하거나 UpdatePackage API 작업을 통해 패키지 버전을 기본 버전으로 지정할 수 있습니다. 이 상태에서는 설명과 태그만 편집할 수 있습니다.

  콘솔을 사용하거나 UpdatePackageVersion 또는 DeletePackageVersion API 작업을 실행하여, 현재 `published` 상태에 있는 패키지 버전을 `deprecated` 상태로 전환하거나 `deleted` 상태로 만들 수 있습니다.

- Deprecated

새 패키지 버전을 사용할 수 있는 경우 이전 패키지 버전을 deprecated로 전환할 수 있습니다. 더 이상 사용되지 않는 패키지 버전으로 작업을 배포할 수 있습니다. 사용되지 않는 패키지 버전의 이름을 기본 버전으로 지정하고 설명 및 태그만 편집할 수도 있습니다.

버전이 오래된 경우 패키지 버전을 deprecated로 전환하는 것이 좋지만, 이전 버전을 사용 중이거나 런타임 종속성으로 인해 유지 관리해야 하는 디바이스가 여전히 필드에 있습니다.

콘솔을 사용하거나 UpdatePackageVersion 또는 DeletePackageVersion API 작업을 실행하여, 현재 deprecated 상태에 있는 패키지 버전을 published 상태로 전환하거나 deleted 상태로 만들 수 있습니다.

- Deleted

  패키지 버전을 더 이상 사용하지 않으려는 경우 콘솔이나 DeletePackageVersion API 작업을 사용하여 삭제할 수 있습니다.

  > **Note**
  >
  > 패키지 버전을 참조하는 보류 중인 작업이 있을 때 해당 패키지 버전을 삭제하면 작업이 성공적으로 완료되고 명명된 예약 섀도우를 업데이트하려고 하면 오류 메시지가 표시됩니다. 삭제하려는 소프트웨어 패키지 버전이 기본 패키지 버전으로 명명된 경우 먼저 패키지를 업데이트하여 다른 버전을 기본 버전으로 명명하거나 필드에 이름을 지정하지 않은 상태로 두어야 합니다. 이 작업은 콘솔 또는 UpdatePackageVersion API 작업을 사용하여 수행할 수 있습니다. 명명된 패키지 버전을 기본값에서 제거하려면 UpdatePackage API 작업을 실행할 때 unsetDefaultVersion 파라미터를 참으로 설정하세요.

  콘솔을 통해 소프트웨어 패키지를 삭제하면 기본 버전으로 명명되지 않은 한 해당 패키지와 관련된 모든 패키지 버전이 삭제됩니다.

패키지 버전 명명 규칙

패키지 버전의 이름을 지정할 때는 사용자와 다른 사람들이 최신 패키지 버전과 버전 진행 상황을 쉽게 식별할 수 있도록 논리적인 이름 지정 전략을 계획하고 적용하는 것이 중요합니다. 패키지 버전을 만들 때 버전 이름을 입력해야 하지만 전략과 형식은 주로 비즈니스 사례에 따라 달라집니다.

모범 사례로 시맨틱 버전 관리 SemVer 형식을 사용하는 것이 좋습니다. 예를 들어, 1.2.3에서 1은 기능적으로 호환되지 않는 변경의 메이저 버전이고 2는 기능적으로 호환되는 변경의 메이저 버전이며

3은 패치 버전(버그 수정용)입니다. 자세한 내용을 알아보려면 의미 체계 버전 관리 2.0.0을 참조하세요. 패키지 버전 이름 요구 사항에 대한 자세한 내용은 AWS IoT API 참조 가이드의 versionName을 참조하세요.

기본 버전

버전을 기본값으로 설정하는 것은 선택 사항입니다. 기본 패키지 버전을 추가하거나 제거할 수 있습니다. 기본 버전으로 명명되지 않은 패키지 버전을 배포할 수도 있습니다.

패키지 버전을 만들면 `draft` 상태가 되며 패키지 버전을 Published로 전환하기 전까지는 기본 버전으로 명명할 수 없습니다. 소프트웨어 패키지 카탈로그는 자동으로 버전을 기본 버전으로 선택하거나 새 패키지 버전을 기본 버전으로 업데이트하지 않습니다. 콘솔을 통해 또는 UpdatePackageVersion API 작업을 실행하여 선택한 패키지 버전의 이름을 의도적으로 지정해야 합니다.

버전 속성

버전 속성 및 해당 값에는 패키지 버전에 대한 중요한 정보가 들어 있습니다. 패키지 또는 패키지 버전의 범용 속성을 정의하는 것이 좋습니다. 예를 들어 플랫폼, 아키텍처, 운영 체제, 출시일, 작성자 또는 Amazon S3 URL에 대한 이름-값 페어를 생성할 수 있습니다.

AWS IoT 작업 문서로 작업을 생성할 때 속성 값을 참조하는 대체 변수(`$parameter`)를 사용하도록 선택할 수도 있습니다. 자세한 내용은 AWS IoT 작업 준비를 참조하세요.

패키지 버전에서 사용되는 버전 속성은 예약 및 명명된 섀도우에 자동으로 추가되지 않으며 플릿 인덱싱을 통해 직접 인덱싱하거나 쿼리할 수 없습니다. 플릿 인덱싱을 통해 패키지 버전 속성을 인덱싱하거나 쿼리하려면 예약 및 명명된 섀도우에 버전 속성을 채우면 됩니다.

예약 및 명명된 섀도우 캡처 디바이스에서 보고한 속성(예: 운영 체제 및 설치 시간)의 버전 속성 파라미터를 사용하는 것이 좋습니다. 플릿 인덱싱을 통해 인덱싱하고 쿼리할 수도 있습니다.

버전 속성에 대한 특정 명명 규칙을 따를 필요는 없습니다. 비즈니스 요구 사항에 맞게 이름-값 페어를 만들 수 있습니다. 패키지 버전의 모든 속성을 합친 크기는 3KB로 제한됩니다. 자세한 내용은 소프트웨어 패키지 카탈로그 소프트웨어 패키지 및 패키지 버전 제한을 참조하세요.

작업 문서의 모든 속성 사용

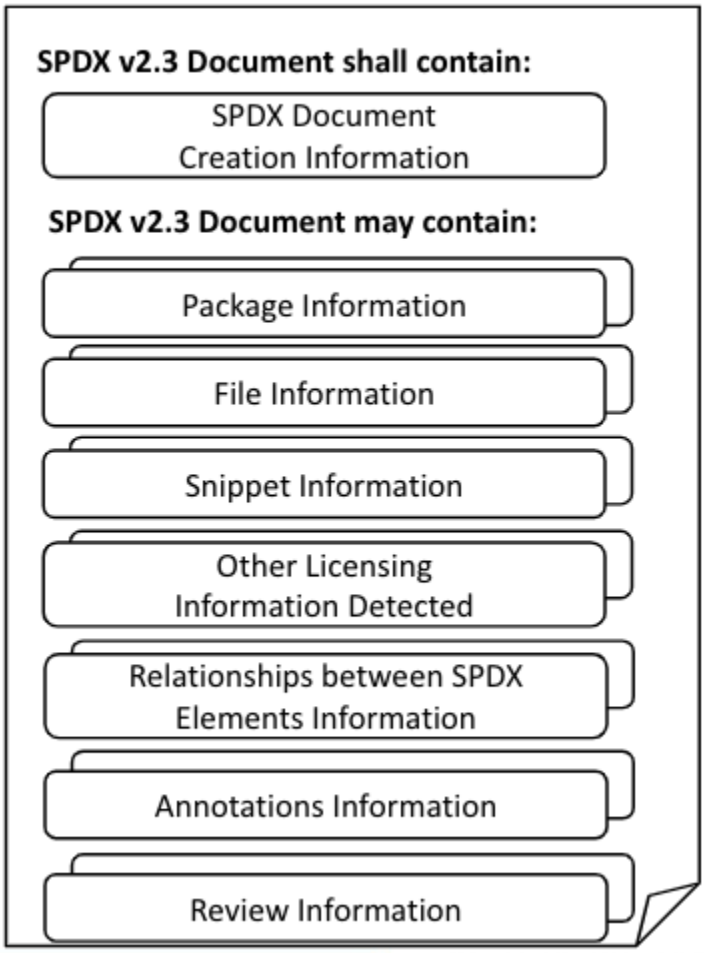
선택한 디바이스의 작업 배포에 모든 패키지 버전 속성을 자동으로 추가할 수 있습니다. API 또는 CLI 명령에서 프로그래밍 방식으로 모든 패키지 버전 속성을 자동으로 사용하려면 다음 작업 문서 예제를 참조하세요.

```
"TestPackage": "${aws:iot:package:TestPackage:version:PackageVersion:attributes}"
```

소프트웨어 재료표

소프트웨어 재료표(SBOM)는 소프트웨어 패키지의 모든 측면을 위한 중앙 리포지토리를 제공합니다. 소프트웨어 패키지 및 패키지 버전을 저장하는 것 외에도 각 패키지 버전과 연결된 소프트웨어 재료표(SBOM)를 AWS IoT Device Management 소프트웨어 패키지 카탈로그에 저장할 수 있습니다. 소프트웨어 패키지에는 하나 이상의 패키지 버전이 포함되어 있으며 각 패키지 버전은 하나 이상의 구성 요소로 구성됩니다. 특정 패키지 버전의 구성을 지원하는 각 구성 요소는 소프트웨어 재료표를 사용하여 설명하고 카탈로그화할 수 있습니다. 지원되는 소프트웨어 재료표에 대한 업계 표준은 SPDX 및 CycloneDX 입니다. SBOM이 처음 생성되면 SPDX 및 CycloneDX 산업 표준 형식에 대한 검증을 거칩니다. SPDX에 대한 자세한 내용은 시스템 패키지 데이터 교환을 참조하세요. CycloneDX에 대한 자세한 내용은 CycloneDX를 참조하세요.

소프트웨어 자료표는 패키지 정보, 파일 정보 및 기타 관련 메타데이터와 같은 특정 패키지 버전 구성 요소의 모든 측면을 설명합니다. SPDX 형식의 소프트웨어 재료표 문서 구조의 아래 예제를 참조하세요.



소프트웨어 재료표 이점

Software Package Catalog의 패키지 버전에 대한 소프트웨어 재료 명세서를 추가할 때 얻을 수 있는 주요 이점 중 하나는 취약성 관리입니다.

취약성 관리

소프트웨어 구성 요소의 명백한 보안 위험에 대한 취약성을 평가하고 완화하는 것은 디바이스 플릿의 무결성을 보호하는 데 여전히 중요합니다. 각 패키지 버전에 대해 Software Package Catalog에 저장된 소프트웨어 자료 명세서를 추가하면 자체 취약성 관리 솔루션을 사용하여 패키지 버전 및 SBOM을 기반으로 어떤 디바이스가 위험에 처했는지 파악하여 보안 격차를 사전에 파악할 수 있습니다. 영향을 받는 디바이스에 수정 사항을 배포하고 디바이스 플릿을 보호할 수 있습니다.

소프트웨어 재료표 스토리지

각 소프트웨어 패키지 버전의 소프트웨어 재료표(SBOM)는 Amazon S3 버전 관리 기능을 사용하여 Amazon S3 버킷에 저장됩니다. SBOM을 저장하는 Amazon S3 버킷은 패키지 버전이 생성된 동일한 리전에 있어야 합니다. 버전 관리 기능을 사용하는 Amazon S3 버킷은 동일한 버킷에 객체의 여러 변형을 유지합니다. Amazon S3 버킷에서 버전 관리를 사용하는 방법에 대한 자세한 내용은 Amazon S3 버킷에서 버전 관리를 참조하세요.

> **Note**
>
> 각 소프트웨어 패키지 버전에는 여러 SBOM 파일이 연결될 수 있지만 이러한 SBOM 파일은 단일 zip 아카이브 파일에 저장되어야 합니다.

버킷의 특정 Amazon S3 키 및 버전 ID는 패키지 버전에 대한 소프트웨어 재료표의 각 버전을 고유하게 식별하는 데 사용됩니다.

> **Note**
>
> 단일 SBOM 파일이 있는 패키지 버전의 경우 해당 SBOM 파일을 Amazon S3 버킷에 zip 아카이브 파일로 저장할 수 있습니다.
> 여러 SBOM 파일이 있는 패키지 버전의 경우 모든 SBOM 파일을 단일 zip 아카이브 파일에 배치한 다음 해당 zip 아카이브 파일을 Amazon S3 버킷에 저장해야 합니다.
> 두 시나리오 모두에서 단일 zip 아카이브 파일에 저장된 모든 SBOM 파일은 SPDX 또는 CycloneDX .json 파일로 형식이 지정됩니다.

권한 정책

지정된 보안 주체 AWS IoT 역할을 수행하여 Amazon S3 버킷에 저장된 SBOM zip 아카이브 파일에 액세스하려면 리소스 기반 권한 정책이 필요합니다. 올바른 리소스 기반 권한 정책은 다음 예제를 참조하세요.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iot.amazonaws.com"
                ]
        },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::bucketName/*"
        }
    ]
}
```

리소스 기반 정책에 대한 자세한 내용은 AWS IoT 리소스 기반 정책 섹션을 참조하세요.

SBOM 업데이트

디바이스 플릿을 보호하고 강화하는 데 필요한 만큼 자주 소프트웨어 재료표를 업데이트할 수 있습니다. Amazon S3 버킷에서 소프트웨어 재료표가 업데이트될 때마다 버전 ID가 변경되고, 새 Amazon S3 버킷 URL을 적절한 소프트웨어 패키지 버전과 연결해야 합니다. AWS Management Console의 패키지 버전 페이지에 있는 Amazon S3 객체 버전 ID 열에 새 버전 ID가 표시됩니다. 또한 API 작업 `GetPackageVersion` 또는 CLI 명령 `get-package-version`을 사용하여 새 버전 ID를 볼 수 있습니다.

> **Note**
>
> 새 버전 ID를 발생시키는 소프트웨어 재료 명세서를 업데이트해도 새 패키지 버전이 생성되지 않습니다.

Amazon S3 객체 키에 대한 자세한 내용은 객체 키 이름 생성을 참조하세요.

AWS IoT 플릿 인덱싱 활성화

AWS IoT 플릿 인덱싱을 활성화하려면 AWS IoT Device Management 소프트웨어 패키지 카탈로그를 사용해야 합니다. Software Package Catalog를 사용하여 AWS IoT 플릿 인덱싱을 활용하려면 예약된 명명 섀도($package)를 인덱싱하고 지표를 수집하려는 각 디바이스의 데이터 소스로 설정합니다. 예약된 명명 섀도에 대한 자세한 내용은 명명된 예약 섀도우 섹션을 참조하세요.

플릿 인덱싱은 소프트웨어 패키지 버전으로 필터링되는 동적 사물 그룹을 통해 AWS IoT 사물을 그룹화할 수 있는 지원을 제공합니다. 예를 들어 플릿 인덱싱은 특정 패키지 버전이 설치되어 있거나 설치되지 않은 사물, 패키지 버전이 설치되지 않은 사물, 특정 이름-값 페어와 일치하는 사물을 식별할 수 있습니다. 마지막으로, 플릿 인덱싱은 디바이스 플릿 상태에 대한 인사이트를 얻는 데 사용할 수 있는 표준 및 사용자 지정 지표를 제공합니다. 자세한 내용은 플릿 인덱싱 준비 단원을 참조하십시오.

> **Note**
>
> 소프트웨어 패키지 카탈로그에 플릿 인덱싱을 활성화하면 표준 서비스 비용이 발생합니다. 자세한 내용은 AWS IoT Device Management요금을 참조하세요.

명명된 예약 섀도우

명명된 예약 섀도우인 $package는 디바이스에 설치된 소프트웨어 패키지 및 패키지 버전의 상태를 반영합니다. 플릿 인덱싱은 명명된 예약 섀도우를 데이터 소스로 사용하여 플릿 상태를 쿼리할 수 있도록 표준 및 사용자 지정 지표를 구축합니다. 자세한 내용은 플릿 인덱싱 준비를 참조하세요.

명명된 예약 섀도우는 이름이 미리 정의되어 있고 변경할 수 없다는 점을 제외하면 명명된 섀도우와 비슷합니다. 또한 명명된 예약 섀도우는 메타데이터로 업데이트되지 않고 `version` 및 `attributes` 키워드만 사용합니다.

`description`과 같은 다른 키워드가 포함된 업데이트 요청은 `rejected` 주제에서 오류 응답을 받게 됩니다. 자세한 내용은 디바이스 섀도우 오류 메시지지를 참조하세요.

콘솔을 통해 AWS IoT 사물을 생성할 때, AWS IoT 작업이 성공적으로 완료되고 섀도우를 업데이트할 때, `UpdateThingShadow` API 작업을 실행할 때 생성할 수 있습니다. 자세한 내용은 AWS IoT Core 개발자 안내서의 UpdateThingShadow를 참조하세요.

> **Note**
>
> 명명된 예약 섀도우를 인덱싱해도 플릿 인덱싱이 인덱싱할 수 있는 명명된 섀도우 개수에는 포함되지 않습니다. 자세한 내용은 AWS IoT Device Management 플릿 인덱싱 제한 및 할당량을 참조하세요. 또한 AWS IoT 작업이 성공적으로 완료될 때 예약된 명명된 섀도우를 업데이트하도록 선택한 경우 API 호출은 디바이스 섀도우 및 레지스트리 작업에 포함되며 비용이 발생할 수 있습니다. 자세한 내용은 AWS IoT Device Management 작업 제한 및 할당량 및 IndexingFilter API 데이터 유형을 참조하세요.

$package 섀도우의 구조

명명된 예약 섀도우에는 다음이 포함됩니다.

```
{
    "state": {
        "reported": {
            "<packageName>": {
                "version": "",
                "attributes": {
                }
            }
        }
    },
    "version" : 1
    "timestamp" : 1672531201
}
```

섀도우 속성이 다음 정보로 업데이트됩니다.

- <packageName>: 설치된 소프트웨어 패키지의 이름으로, packageName 파라미터로 업데이트됩니다.
- version: 설치된 소프트웨어 패키지의 버전으로, versionName 파라미터로 업데이트됩니다.
- attributes: 디바이스가 저장하고 플릿 인덱싱이 인덱싱하는 선택적 메타데이터입니다. 이를 통해 고객은 저장 데이터를 기반으로 인덱스를 쿼리할 수 있습니다.
- version: 섀도우의 버전 번호입니다. 섀도우가 업데이트되고 1에서 시작될 때마다 자동으로 증가합니다.
- timestamp: 섀도우가 마지막으로 업데이트된 시간을 나타내며 Unix 시간으로 기록됩니다.

명명된 섀도우의 형식 및 동작에 대한 자세한 내용은 AWS IoT 디바이스 섀도우 서비스 메시지 순서를 참조하세요.

소프트웨어 패키지 및 패키지 버전 삭제

소프트웨어 패키지를 삭제하기 전에 다음 작업을 수행하세요.

- 패키지와 해당 버전이 활발하게 배포되고 있지 않은지 확인합니다.
- 먼저 관련 버전을 모두 삭제합니다. 버전 중 하나가 기본 버전으로 지정된 경우 패키지에서 명명된 기본 버전을 제거해야 합니다. 기본 버전 지정은 선택 사항이므로 기본 버전을 제거해도 충돌이 발생하지 않습니다. 소프트웨어 패키지에서 기본 버전을 제거하려면 콘솔을 통해 패키지를 편집하거나 UpdatePackageVersion API 작업을 사용하세요.

명명된 기본 패키지 버전이 없는 한 콘솔을 사용하여 소프트웨어 패키지를 삭제할 수 있으며 해당 패키지 버전도 모두 삭제됩니다. API 호출을 사용하여 소프트웨어 패키지를 삭제하는 경우 먼저 패키지 버전을 삭제한 다음 소프트웨어 패키지를 삭제해야 합니다.

보안 준비

이 섹션에서는 AWS IoT Device Management 소프트웨어 패키지 카탈로그의 주요 보안 요구 사항에 대해 설명합니다.

리소스 기반 인증

소프트웨어 패키지 카탈로그는 리소스 기반 인증을 사용하여 플릿의 소프트웨어를 업데이트할 때 보안을 강화합니다. 즉, 소프트웨어 패키지 및 패키지 버전에 대해 `create`, `read`, `updatedelete`, 및 `list` 작업을 수행할 권한을 부여하는 AWS Identity and Access Management (IAM) 정책을 생성하고 `Resources` 섹션에서 배포하려는 특정 소프트웨어 패키지 및 패키지 버전을 참조해야 합니다. 명명된 예약 섀도우를 업데이트할 때도 이러한 권한이 필요합니다. 각 엔터티에 Amazon 리소스 이름(ARN)을 포함하여 소프트웨어 패키지 및 패키지 버전을 참조합니다.

> **Note**
>
> 정책에서 패키지 버전 API 직접 호출(예: CreatePackageVersion, UpdatePackageVersion, DeletePackageVersion)에 대한 권한을 부여하려는 경우, 소프트웨어 패키지와 패키지 버전 ARN을 둘 다 정책에 포함해야 합니다. 정책에서 패키지 버전 API 직접 호출(예:

CreatePackage, UpdatePackage, DeletePackage)에 대한 권한을 부여하려는 경우 소프트웨어 패키지 ARN만 정책에 포함해야 합니다.

소프트웨어 패키지 및 패키지 버전 ARN을 다음과 같이 구성하세요.

- 소프트웨어 패키지: `arn:aws:iot:<region>:<accountID>:package/<packageName>/package`
- 패키지 버전: `arn:aws:iot:<region>:<accountID>:package/<packageName>/version/<versionName>`

> **Note**
>
> 이 정책에 포함할 수 있는 기타 관련 권한도 있습니다. 예를 들어, `job`, `thinggroup` 및 `jobtemplate`에 대한 ARN을 포함할 수 있습니다. 자세한 내용과 정책 옵션의 전체 목록은 AWS IoT 작업을 사용하여 사용자 및 디바이스 보안을 참조하세요.

예를 들어 다음과 같이 명명된 소프트웨어 패키지 및 패키지 버전이 있는 경우:

- AWS IoT 사물: `myThing`
- 패키지 이름: `samplePackage`
- 버전: `1.0.0`

정책은 다음과 같을 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:createPackage",
                "iot:createPackageVersion",
                "iot:updatePackage",
                "iot:updatePackageVersion"
            ],
```

```
            "Resource": [
               "arn:aws:iot:us-east-1:111122223333:package/samplePackage",
               "arn:aws:iot:us-east-1:111122223333:package/samplePackage/
version/1.0.0"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": "arn:aws:iot:us-east-1:111122223333:thing/myThing/$package"
        }
    ]
}
```

AWS IoT 패키지 버전을 배포할 수 있는 작업 권한

보안을 위해 패키지 및 패키지 버전을 배포할 권한을 부여하고 배포가 허용된 특정 패키지 및 패키지 버전의 이름을 지정하는 것이 중요합니다. 이를 위해서는 패키지 버전으로 작업을 배포할 권한을 부여하는 IAM 역할과 정책을 만듭니다. 정책은 대상 패키지 버전을 리소스로 지정해야 합니다.

IAM 정책

IAM 정책은 `Resource` 섹션에 명명된 패키지 및 버전을 포함하는 작업을 생성할 권한을 부여합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob",
                "iot:CreateJobTemplate"
            ],
            "Resource":[
                "arn:aws:iot:*:111122223333:job/<jobId>",
                "arn:aws:iot:*:111122223333:thing/<thingName>/$package",
                "arn:aws:iot:*:111122223333:thinggroup/<thingGroupName>",
                "arn:aws:iot:*:111122223333:jobtemplate/<jobTemplateName>",
```

```
                "arn:aws:iot:*:111122223333:package/<packageName>/
version/<versionName>"
            ]
        }
    ]
}
```

> **Note**
>
> 소프트웨어 패키지와 패키지 버전을 제거하는 작업을 배포하려면 다음과 같이 패키지 버전이 $null인 ARN을 승인해야 합니다.

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

AWS IoT 예약된 명명된 섀도우를 업데이트하는 작업 권한

작업이 성공적으로 완료되었을 때 작업이 사물의 명명된 예약 섀도우를 업데이트할 수 있게 하려면 IAM 역할 및 정책을 생성해야 합니다. AWS IoT 콘솔에서 이 작업을 수행하는 방법은 두 가지입니다. 첫 번째는 콘솔에서 소프트웨어 패키지를 만드는 것입니다. 패키지 관리를 위한 종속성 활성화 대화 상자가 표시되면 기존 역할을 사용하거나 새 역할을 만들 수 있습니다. 또는 AWS IoT 콘솔에서 설정을 선택하고 인덱싱 관리를 선택한 다음 디바이스 패키지 및 버전 인덱싱 관리를 선택합니다.

> **Note**
>
> AWS IoT 작업이 성공적으로 완료되면 작업 서비스가 예약된 명명된 섀도우를 업데이트하도록 선택하면 API 호출이 디바이스 섀도우 및 레지스트리 작업에 포함되며 비용이 발생할 수 있습니다. 자세한 내용은 AWS IoT Core 요금을 참조하십시오.

역할 생성 옵션을 사용하는 경우 생성된 역할 이름은 `aws-iot-role-update-shadows`로 시작되며 다음 정책을 포함합니다.

역할 설정

권한

권한 정책은 사물 섀도우를 쿼리하고 업데이트할 권한을 부여합니다. 리소스 ARN의 $package 파라미터는 명명된 예약 섀도우를 대상으로 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DescribeEndpoint",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:111122223333:thing/<thingName>/$package"
            ]
        }
    ]
}
```

신뢰 관계

권한 정책 외에도 역할에는 엔터티가 역할을 수임하고 명명된 예약 섀도우를 업데이트할 수 있도록 AWS IoT Core 와의 신뢰 관계가 필요합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

사용자 정책 설정

iam:PassRole 권한

마지막으로 UpdatePackageConfiguration API 작업을 호출할 AWS IoT Core 때 역할을에 전달할 권한이 있어야 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iot:UpdatePackageConfiguration"
            ],
            "Resource": "arn:aws:iam::111122223333:role/<roleName>"
        }
    ]
}
```

AWS IoT Amazon S3에서 다운로드할 수 있는 작업 권한

작업 문서는 Amazon S3에 저장됩니다. AWS IoT 작업을 통해 디스패치할 때이 파일을 참조합니다. AWS IoT 작업에게 파일()을 다운로드할 수 있는 권한을 제공해야 합니다`s3:GetObject`. 또한 Amazon S3와 AWS IoT 작업 간에 신뢰 관계를 설정해야 합니다. 이러한 정책을 생성하는 방법에 대한 지침은 작업 관리에서 미리 서명된 URL을 참조하세요.

패키지 버전의 소프트웨어 재료 명세서를 업데이트할 수 있는 권한

`Draft`, `Published`또는 `Deprecated` 수명 주기 상태에서 패키지 버전의 소프트웨어 재료표를 업데이트하려면 Amazon S3에서 새 소프트웨어 재료표를 찾고 패키지 버전을 업데이트하기 위한 AWS Identity and Access Management 역할 및 정책이 필요합니다 AWS IoT Core.

먼저 업데이트된 소프트웨어 재료표를 버전이 지정된 Amazon S3 버킷에 넣고 `sboms` 파라미터가 포함된 `UpdatePackageVersion` API 작업을 직접적으로 호출합니다. 그런 다음 권한 있는 보안 주체가 생성한 IAM 역할을 수임하고, Amazon S3에서 업데이트된 소프트웨어 재료표를 찾고, AWS IoT Core for Software Package Catalog에서 패키지 버전을 업데이트합니다.

이 업데이트를 수행하려면 다음 정책이 필요합니다.

정책

- Amazon S3의 버전 버킷에서 업데이트된 소프트웨어 재료표를 찾고 AWS IoT Core에서 패키지 버전을 업데이트할 수 있도록 IAM 역할을 수임하는 권한 있는 위탁자와 신뢰 관계를 설정하는 신뢰 정책 정책입니다.

  -
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

  -
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- 권한 정책: 패키지 버전에 대한 소프트웨어 자재 명세서가 저장되는 Amazon S3 버전 버킷에 액세스하고 AWS IoT Core에서 패키지 버전을 업데이트하는 정책입니다.

  -
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1"
            ]
        }
    ]
}
```

- ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "iot:UpdatePackageVersion"
              ],
              "Resource": [
                  "arn:aws:iot:*:111122223333:package/<packageName>/
  version/<versionName>"
              ]
          }
      ]
  }
  ```

- 역할 권한 전달: UpdatePackageVersion API 작업을 호출할 AWS IoT Core 때 IAM 역할을 Amazon S3 및에 전달할 수 있는 권한을 부여하는 정책입니다.

  - ```
    {
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "iam:PassRole",
            "s3:GetObject"
          ],
          "Resource": [
            "arn:aws:s3:::awsexamplebucket1"
          ]
        }
      ]
    }
    ```

- ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "iam:PassRole",
                  "iot:UpdatePackageVersion"
              ],
              "Resource": "arn:aws:iam::111122223333:role/<roleName>"
          }
      ]
  }
  ```

> **Note**
>
> `Deleted` 수명 주기 상태로 전환된 패키지 버전에서는 소프트웨어 재료표를 업데이트할 수 없습니다.

AWS 서비스에 대한 IAM 역할 생성에 대한 자세한 내용은 AWS 서비스에 권한을 위임할 역할 생성을 참조하세요.

Amazon S3 버킷 생성 및 객체 업로드에 대한 자세한 내용은 버킷 생성 및 객체 업로드를 참조하세요.

플릿 인덱싱 준비

AWS IoT 플릿 인덱싱을 사용하면 예약된 명명된 섀도우()를 사용하여 데이터를 검색하고 집계할 수 있습니다$package. 명명된 예약 섀도우 및 동적 AWS IoT 사물 그룹을 쿼리하여 사물을 그룹화할 수도 있습니다. https://docs.aws.amazon.com/iot/latest/developerguide/dynamic-thing-groups.html 예를 들어 특정 패키지 버전을 사용하거나, 특정 패키지 버전이 설치되어 있지 않거나, 패키지 버전이 설치되어 있지 않은 AWS IoT 사물에 대한 정보를 찾을 수 있습니다. 속성을 결합하여 더 많은 인사이트를 얻을 수 있습니다. 예를 들어, 특정 버전이 설치되었고 특정 사물 유형(예: 버전 1.0.0, 사물 유형 pump_sensor)인 사물을 식별할 수 있습니다. 자세한 내용은 플릿 인덱싱을 참조하세요.
  ```

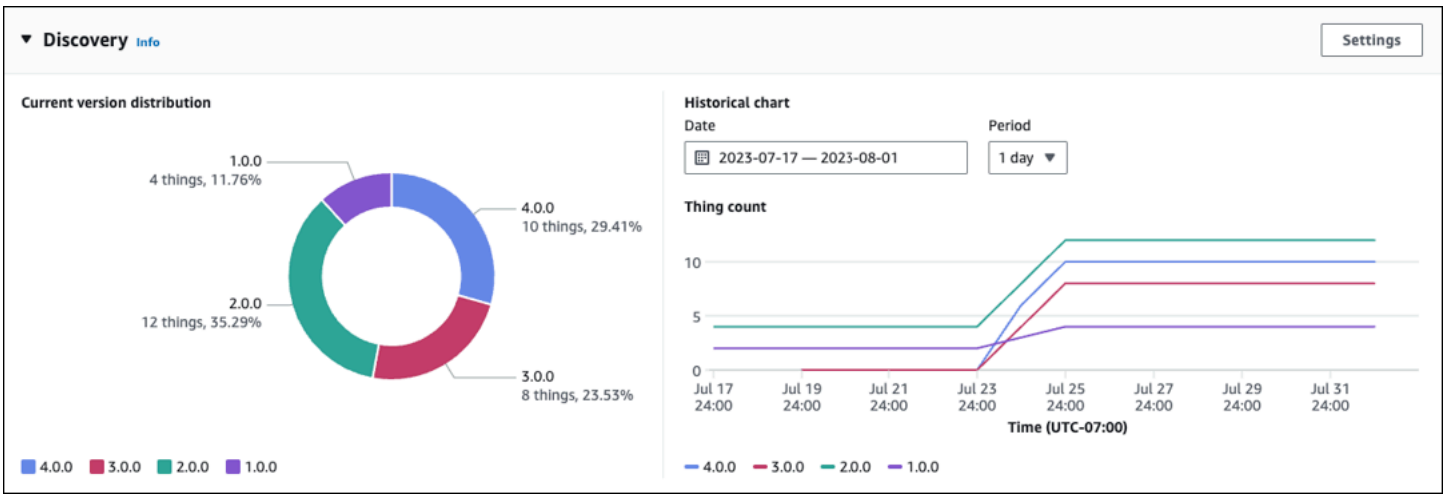
$package 섀도우를 데이터 소스로 설정

소프트웨어 패키지 카탈로그에서 플릿 인덱싱을 사용하려면 플릿 인덱싱을 활성화하고, 명명된 섀도우를 데이터 소스로 설정하고, $package를 명명된 섀도우 필터로 정의해야 합니다. 플릿 인덱싱을 활성화하지 않은 경우 이 프로세스에서 활성화할 수 있습니다. 콘솔의 AWS IoT Core에서 설정을 열고 인덱싱 관리를 선택한 다음 명명된 섀도우 추가, 디바이스 소프트웨어 패키지 및 버전 추가, 업데이트를 선택합니다. 자세한 내용은 사물 인덱싱 관리를 참조하세요.

또는 첫 번째 패키지를 생성할 때 플릿 인덱싱을 활성화할 수 있습니다. 패키지 관리를 위한 종속성 활성화 대화 상자가 나타나면 디바이스 소프트웨어 패키지 및 버전을 플릿 인덱싱에 데이터 소스로 추가하는 옵션을 선택합니다. 이 옵션을 선택하면 플릿 인덱싱도 활성화됩니다.

> **Note**
>
> 소프트웨어 패키지 카탈로그에 플릿 인덱싱을 활성화하면 표준 서비스 비용이 발생합니다. 자세한 내용은 AWS IoT Device Management요금을 참조하세요.

콘솔에 표시된 지표



AWS IoT 콘솔 소프트웨어 패키지 세부 정보 페이지의 검색 패널에는 $package 섀도우를 통해 수집된 표준 지표가 표시됩니다.

- 현재 버전 배포 차트는이 소프트웨어 패키지와 연결된 모든 디바이스의 AWS IoT 사물과 연결된 10개의 최신 패키지 버전에 대한 디바이스 수와 비율을 보여줍니다. 참고: 소프트웨어 패키지의 패키지 버전이 차트에 레이블이 지정된 버전보다 많으면 기타 내에 그룹화되어 있는 것을 확인할 수 있습니다.

- 기록 차트는 지정된 기간 동안 선택한 패키지 버전과 관련된 디바이스 수를 보여줍니다. 처음부터 패키지 버전을 최대 5개까지 선택하고 날짜 범위와 시간 간격을 정의할 때까지 차트는 비어 있습니다. 차트의 파라미터를 선택하려면 설정을 선택합니다. 표시되는 패키지 버전 수에 차이가 있고 기록 차트에서 분석할 패키지 버전을 선택할 수 있기 때문에 기록 차트에 표시되는 데이터가 현재 버전 배포 차트와 다를 수 있습니다. 참고: 시각화할 패키지 버전을 선택하면 해당 패키지 버전이 플릿 지표 최대 한도에 포함됩니다. 자세한 내용은 플릿 인덱싱 제한 및 할당량을 참조하세요.

패키지 버전 배포 수집에 대한 인사이트를 얻을 수 있는 또 다른 방법은 getBucketsAggregation을 통한 패키지 버전 배포 수집을 참조하세요.

쿼리 패턴

Software Package Catalog를 사용한 플릿 인덱싱은 표준으로 지원되는 대부분의 기능(예: 용어, 구문, 검색 필드)을 사용합니다. 그러나 명명된 예약 섀도우($package) `version` 키에는 비교 연산자(예: <보다 작거나 >보다 큼)와 `range` 쿼리를 사용할 수 없습니다. 이러한 쿼리는 `attributes` 키에 사용할 수 있습니다. 자세한 내용은 쿼리 구문을 참조하세요.

예시 데이터

참고: 명명된 예약 섀도우와 그 구조에 대한 자세한 내용은 명명된 예약 섀도우를 참조하세요.

이 예시에서는 첫 번째 디바이스의 이름이 `AnyThing`이고 다음 패키지가 설치되어 있습니다.

- 소프트웨어 패키지: `SamplePackage`

  패키지 버전: `1.0.0`

  패키지 ID: `1111`

섀도우는 다음과 같습니다.

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
```

```
                    "packageID": "1111"
                    }
            }
        }
    }
}
```

두 번째 디바이스는 이름이 AnotherThing이고며 다음 패키지가 설치되어 있습니다.

- 소프트웨어 패키지: SamplePackage

  패키지 버전: 1.0.0

  패키지 ID: 1111
- 소프트웨어 패키지: OtherPackage

  패키지 버전: 1.2.5

  패키지 ID: 2222

섀도우는 다음과 같습니다.

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                }
            },
            "OtherPackage": {
                "version": "1.2.5",
                "attributes": {
                    "s3UrlForOtherPackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile2",
                    "packageID": "2222"
                    }
            },
        }
```

```
    }
}
```

샘플 쿼리

다음 테이블에는 AnyThing 및 AnotherThing에 대한 예시 디바이스 섀도우를 기반으로 한 샘플 쿼리가 나열되어 있습니다. 자세한 내용은 예시 사물 쿼리를 참조하세요.

FreeRTOS AWS IoT Device Tester 용의 최신 버전

요청된 정보	Query	결과
특정 패키지 버전이 설치된 사물	`shadow.name.$package.reported.SamplePackage.version:1.0.0`	`AnyThing, OtherThing`
특정 패키지 버전이 설치되지 않은 사물	`NOT shadow.name.$package.reported.OtherPackage.version:1.2.5`	`AnyThing`
패키지 ID가 1500보다 큰 패키지 버전을 사용하는 모든 디바이스	`shadow.name.$package.reported.*.attributes.packageID>1500"`	`OtherThing`
특정 패키지가 설치되어 있고 두 개 이상의 패키지가 설치된 사물	`shadow.name.$package.reported.SamplePackage.version:1.0.0 AND shadow.name.$package.reported.totalCount:2`	`OtherThing`

`getBucketsAggregation`을 통한 패키지 버전 배포 수집

AWS IoT 콘솔 내의 검색 패널 외에도 GetBucketsAggregation API 작업을 사용하여 패키지 버전 배포 정보를 가져올 수도 있습니다. 패키지 버전 배포 정보를 가져오려면 다음을 수행해야 합니다.

- 각 소프트웨어 패키지의 플릿 인덱싱 내에서 사용자 지정 필드를 정의합니다. 참고: 사용자 지정 필드를 생성하는 것은 AWS IoT 플릿 인덱싱 서비스 할당량에 포함됩니다.
- 다음과 같이 사용자 지정 필드의 형식을 지정합니다.

  `shadow.name.$package.reported.<packageName>.version`

자세한 내용은 AWS IoT 플릿 인덱싱의 사용자 지정 필드 섹션을 참조하세요.

AWS IoT 작업 준비

AWS IoT Device Management 소프트웨어 패키지 카탈로그는 대체 파라미터, AWS IoT 플릿 인덱싱, 동적 사물 그룹 및 AWS IoT 사물의 예약된 명명된 섀도우와의 통합을 통해 AWS IoT 작업을 확장합니다.

> **Note**
>
> 소프트웨어 패키지 카탈로그에서 제공하는 모든 기능을 사용하려면 다음과 같은 AWS Identity and Access Management (IAM) 역할 및 정책을 생성해야 합니다. AWS IoT 패키지 버전을 배포할 수 있는 작업 권한 및 AWS IoT 예약된 명명된 섀도우를 업데이트할 수 있는 작업 권한. 자세한 내용은 보안 준비를 참조하세요.

AWS IoT 작업의 대체 파라미터

대체 파라미터를 AWS IoT 작업 문서 내에서 자리 표시자로 사용할 수 있습니다. 작업 서비스는 대체 파라미터를 발견하면 작업이 해당 파라미터 값에 대한 명명된 소프트웨어 버전의 속성을 가리키도록 합니다. 이 프로세스를 사용하여 단일 작업 문서를 만들고 범용 속성을 통해 메타데이터를 작업에 전달할 수 있습니다. 예를 들어 패키지 버전 속성을 통해 Amazon Simple Storage Service(S3) URL, 소프트웨어 패키지 Amazon 리소스 이름(ARN) 또는 서명을 작업 문서에 전달할 수 있습니다.

작업 문서에서 대체 파라미터의 형식을 다음과 같이 지정해야 합니다.

- 소프트웨어 패키지 이름 및 패키지 버전
  - `package::version` 사이의 빈 문자열은 소프트웨어 패키지 이름 대체 파라미터를 나타냅니다. `version::attribute` 사이의 빈 문자열은 소프트웨어 패키지 버전 대체 파라미터를 나타냅니다. 작업 문서에서 패키지 이름 및 패키지 버전 대체 파라미터를 사용하려면 다음 예시를 참조하세요. `${aws:iot:package::version::attributes:<attributekey>}`.

  - 작업 문서는 패키지 버전 세부 정보의 버전 ARN을 사용하여 이러한 대체 파라미터를 자동으로 채웁니다. API 또는 CLI 명령을 사용하여 단일 패키지 배포에 대한 작업 또는 작업 템플릿을 생성하는 경우 패키지 버전의 버전 ARN은 `CreateJob` 및 `DescribeJob`의 `destinationPackageVersions` 파라미터로 표시됩니다.
- 소프트웨어 패키지 버전의 모든 속성
  - 작업 문서에서 소프트웨어 패키지 버전 대체 파라미터의 모든 속성을 사용하려면 다음 예제를 참조하세요. `${aws:iot:package:<packageName>:version:<versionName>:attributes}`.

> **Note**
>
> 패키지 이름, 패키지 버전 및 모든 속성 대체 파라미터를 함께 사용할 수 있습니다. 작업 문서에서 세 가지 대체 파라미터를 모두 사용하려면 다음 예제를 참조하세요. `${aws:iot:package::version::attributes}`.

이 예시에는 이름이 `samplePackage`인 소프트웨어 패키지가 있고 이 패키지에는 이름이 `2.1.5`이고 속성이 다음과 같은 패키지 버전이 있습니다.

- 이름: `s3URL`, 값: `https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/exampleCodeFile`
  - 이 속성은 Amazon S3에 저장된 코드 파일의 위치를 식별합니다.
- 이름: `signature`, 값: `aaaaabbbbbccccddddeeeeeffffffgggggghhhhhiiiiijjjj`
  - 이 속성은 디바이스에 보안 조치로 필요한 코드 서명 값을 제공합니다. 자세한 내용은 작업용 코드 서명을 참조하세요. 참고: 이 속성은 예시이며 소프트웨어 패키지 카탈로그 또는 작업에 필요하지 않습니다.

`s3URL`의 경우, 작업 문서 파라미터는 다음과 같이 작성됩니다.

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
}
```

`signature`의 경우, 작업 문서 파라미터는 다음과 같이 작성됩니다.

```
{
```

```
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
}
```

전체 작업 문서는 다음과 같이 작성됩니다.

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage":
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
      },
    ],
    "signature": [
      "samplePackage" :
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
    ]
  }
}
```

대체가 이루어지면 다음 작업 문서가 디바이스에 배포됩니다.

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage": "https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/
exampleCodeFile"
      },
    ],
    "signature": [
      "samplePackage" : "aaaaabbbbbcccccdddddeeeeeffffffgggggghhhhhiiiiijjjj"
    ]
  }
}
```

대체 파라미터(전/후 보기)

대체 파라미터는 기본 패키지 버전의 $default와 같은 다양한 플래그를 사용하여 작업 문서 생성을 간소화합니다. 이렇게 하면 해당 플래그가 특정 패키지 버전에서 참조된 메타데이터로 자동 채워지므로 각 작업 배포에 대해 특정 패키지 버전 메타데이터를 수동으로 입력할 필요가 없습니다. 기본 패키지 버전의 $default와 같은 패키지 버전 속성에 대한 자세한 내용은 배포를 위한 작업 문서 및 패키지 버전 준비 섹션을 참조하세요.

에서 패키지 버전의 작업 배포 중에 배포 지침 파일 편집기 창의 대체 미리 보기 버튼을 AWS Management Console전환하여 대체 파라미터가 있거나 없는 작업 문서를 봅니다.

DescribeJob 및 GetJobDocument API의 'before-substitution' 파라미터를 사용하면 대체 파라미터가 제거되기 전과 후에 API 응답을 볼 수 있습니다. DescribeJob 및 GetJobDocument API에 대한 다음 예시를 참조하세요.

- DescribeJob
  - 기본 보기

```
{
    "jobId": "<jobId>",
    "description": "<description>",
    "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/1.0.2"]
}
```

  - 대체 보기 전

```
{
    "jobId": "<jobId>",
    "description": "<description>",
    "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/$default"]
}
```

- GetJobDocument
  - 기본 보기

```
{
    "attributes": {
        "location": "prod-artifacts.s3.us-east-1.amazonaws.com/mqtt-core",
        "signature": "IQoJb3JpZ2luX2VjEIrwEaCXVzLWVhc3QtMSJHMEUCIAofPNPpZ9cI",
        "streamName": "mqtt-core",
        "fileId": "0"
```

```
    },
}
```

- 대체 보기 전

```
{
    "attributes": "${aws:iot:package:TestPackage:version:$default:attributes}",
}
```

AWS IoT 작업, 작업 문서 생성 및 작업 배포에 대한 자세한 내용은 작업을 참조하세요.

배포를 위한 작업 문서 및 패키지 버전 준비

패키지 버전이 생성되면 배포 준비 중임을 나타내기 위해 `draft` 상태가 됩니다. 배포를 위해 패키지 버전을 준비하려면 반드시 작업 문서를 생성하고, 작업이 액세스할 수 있는 위치(예: Amazon S3)에 문서를 저장하고, 패키지 버전에 작업 문서에서 사용할 속성 값이 있는지 확인하고, 패키지 버전을 게시해야 합니다. (참고: `draft` 상태에 있는 동안에만 패키지 버전의 속성을 업데이트할 수 있습니다.)

단일 패키지 배포를 위한 AWS IoT 작업 또는 작업 템플릿을 생성할 때 작업 문서를 사용자 지정하는 다음과 같은 옵션이 있습니다.

배포 지침 파일(**`recipe`**)

- 패키지 버전의 배포 지침 파일에는 패키지 버전을 여러 디바이스에 배포하기 위한 인라인 작업 문서를 포함한 배포 지침이 포함되어 있습니다. 파일은 빠르고 효율적인 작업 배포를 위해 패키지 버전에 특정 배포 지침을 연결합니다.

  에서 새 패키지 생성 워크플로의 버전 배포 구성 탭에 있는 배포 지침 파일 미리 보기 창에서 파일을 생성할 AWS Management Console수 있습니다. AWS IoT 를 사용하여 권장 파일에서 시작을 사용하여 패키지 버전 속성에서 AWS IoT 지침 파일을 자동으로 생성하거나 자체 배포 지침 파일 사용을 사용하여 Amazon S3 버킷에 저장된 기존 작업 문서를 사용할 수 있습니다.

  > **Note**
  >
  > 자체 작업 문서를 사용하는 경우 배포 지침 파일 미리 보기 창에서 직접 업데이트할 수 있지만 Amazon S3 버킷에 저장된 원본 작업 문서는 자동으로 업데이트되지 않습니다.

AWS CLI 또는 `CreatePackageVersion`, `GetPackageVersion`또는와 같은 API 명령을 사용하는 경우는 인라인 작업 문서가 포함된 배포 지침 파일을 `UpdatePackageVersion` `recipe` 나타냅니다.

작업 문서가 무엇인지에 대한 자세한 내용은 기본 개념 섹션을 참조하세요.

`recipe`로 표시된 배포 지침 파일은 다음 예시를 참조하세요.

```
{
    "packageName": "sample-package-name",
    "versionName": "sample-package-version",
    ...
    "recipe": "{...}"
}
```

> **Note**
>
> 패키지 버전이 패키지 버전 메타데이터와 별개인 `published` 상태일 때 `recipe`로 표시되는 배포 지침 파일을 업데이트할 수 있습니다. 작업 배포 중에는 변경할 수 없습니다.

`Artifact` 버전 속성

- 소프트웨어 패키지 버전의 버전 속성 `artifact`를 사용하여 패키지 버전 아티팩트의 Amazon S3 위치를 추가할 수 있습니다. AWS IoT 작업을 사용하여 패키지 버전의 작업 배포가 트리거되면 `${aws:iot:package:<packageName>:version:<versionName>:artifact-location:s3-presigned-url}` 작업 문서의 미리 서명된 URL 자리 표시자는 Amazon S3 버킷, 버킷 키 및 Amazon S3 버킷에 저장된 파일의 버전을 사용하여 업데이트됩니다. 패키지 버전 아티팩트를 저장하는 Amazon S3 버킷은 패키지 버전이 생성된 동일한 리전에 있어야 합니다.

> **Note**
>
> Amazon S3 버킷에 동일한 파일의 여러 객체 버전을 저장하려면 버킷에서 버전 관리를 활성화해야 합니다. 자세한 내용은 버킷에 버전 관리 사용 설정을 참조하세요.

CreatePackageVersion 또는 UpdatePackageVersion API 작업을 사용할 때 Amazon S3 버킷의 패키지 버전 아티팩트에 액세스하려면 다음 권한이 있어야 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObjectVersion",
            "Resource": "arn:aws:s3:::bucket-name/key-name"
        }
    ]
}
```

CreatePackageVersion 및 UpdatePackageVersion API 작업의 버전 속성 artifact에 대한 자세한 내용은 CreatePackageVersion 및 UpdatePackageVersion을 참조하세요.

새 패키지 버전을 생성할 때 Amazon S3에서 아티팩트 위치를 지원하는 버전 속성 artifact를 보여주는 다음 예시를 참조하세요.

```
{
    "packageName": "sample package name",
    "versionName": "1.0",
    "artifact": {
        "s3Location": {
            "bucket": "firmware",
            "key": "image.bin",
            "version": "12345"
        }
    }
}
```

> **Note**
>
> 패키지 버전이 draft 상태에서 published 상태로 업데이트되면 패키지 버전 속성과 아티팩트 위치를 변경할 수 없게 됩니다. 이 정보를 업데이트하려면 새 패키지 버전을 생성하고 draft 상태에서 해당 업데이트를 수행해야 합니다.

패키지 버전

- 기본 소프트웨어 패키지 버전은 안전하고 안정적인 패키지 버전을 제공하는 소프트웨어 패키지의 사용 가능한 버전에 표시될 수 있습니다. 이는 AWS IoT 작업을 사용하여 디바이스 플릿에 기본 패키지 버전을 배포할 때 소프트웨어 패키지의 기본 버전으로 사용됩니다. 소프트웨어 `$default` 패키지의 패키지 버전을 배포하는 작업을 생성할 때 작업 문서와 새 작업 배포의 패키지 버전이 `$default`와 일치해야 합니다. 작업 배포의 패키지 버전은 API 및 CLI 명령의 경우 `destinationPackageVersions`로 표시되고 `VersionARN`에서는 AWS Management Console로 표시됩니다. 작업 문서의 패키지 버전은 아래에 표시된 다음 작업 문서 자리 표시자로 표시됩니다.

  ```
  arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$default
  ```

  기본 패키지 버전을 사용하여 작업 또는 작업 템플릿을 생성하려면 아래와 같이 `CreateJob` 또는 `CreateJobTemplate` API 명령에서 `$default` 플래그를 사용합니다.

  ```
  "$ aws iot create-job \
      --destination-package-versions "arn:aws:iot:us-west-2:123456789012:package/
  TestPackage/version/$default"
      --document file://jobdoc.json
  ```

  > **Note**
  >
  > 기본 버전을 참조하는 `$default` 패키지 버전 속성은 AWS IoT 작업을 통한 작업 배포에 대한 기본 패키지 버전을 참조할 때만 필요한 선택적 속성입니다.

패키지 버전이 만족스러우면 AWS IoT 콘솔의 소프트웨어 패키지 세부 정보 페이지를 통해 게시하거나 UpdatePackageVersion API 작업을 실행하여 게시합니다. 그런 다음 AWS IoT 콘솔을 통해 또는 CreateJob API 작업을 실행하여 작업을 생성할 때 패키지 버전을 참조할 수 있습니다.

배포 시 패키지 및 버전 이름 지정

소프트웨어 패키지 버전을 디바이스에 배포하려면 작업 문서에 참조된 소프트웨어 패키지 및 패키지 버전이 `CreateJob` API 작업의 `destinationPackageVersions` 파라미터에 명시된 소프트웨어 패키지 및 패키지 버전과 일치하는지 확인합니다. 일치하지 않는 경우 두 참조를 모두 일치하라는 오류 메시지가 표시됩니다. Software Package Catalog 오류 메시지에 대한 자세한 내용은 오류 메시지 일반 문제 해결 섹션을 참조하세요.

작업 문서에 참조된 소프트웨어 패키지 및 패키지 버전 외에도 작업 문서에 참조되지 않은 `CreateJob` API 작업의 `destinationPackageVersions` 파라미터에 추가 소프트웨어 패키지 및 패키지 버전을 포함할 수 있습니다. 디바이스가 추가 소프트웨어 패키지 버전을 올바르게 설치하는 데 필요한 설치 정보가 작업 문서에 포함되어 있는지 확인합니다. `CreateJob` API 작업에 대한 자세한 내용은 CreateJob 섹션을 참조하세요.

AWS IoT 동적 사물 그룹을 통한 작업 대상 지정

Software Package Catalog는 플릿 인덱싱, AWS IoT 작업 및 AWS IoT 동적 사물 그룹과 함께 작동하여 플릿 내의 디바이스를 필터링하고 타게팅하여 디바이스에 배포할 패키지 버전을 선택합니다. 디바이스의 현재 패키지 정보를 기반으로 플릿 인덱싱 쿼리를 실행하고 해당 작업을 대상으로 지정할 수 있습니다 AWS IoT . 소프트웨어 업데이트를 릴리스할 수도 있지만 적합한 대상 디바이스에만 릴리스할 수 있습니다. 예를 들어, 현재 `iot-device-client 1.5.09`를 실행하는 디바이스에만 구성을 배포하도록 지정할 수 있습니다. 자세한 내용은 동적 사물 그룹 생성을 참조하세요.

명명된 예약 섀도우 및 패키지 버전

구성된 경우 AWS IoT 작업이 성공적으로 완료되면 작업에서 예약된 섀도우($package)를 업데이트할 수 있습니다. 이렇게 하면 패키지 버전을 사물의 명명된 예약 섀도우에 수동으로 연결할 필요가 없습니다.

다음과 같은 상황에서는 패키지 버전을 사물의 명명된 예약 섀도우에 수동으로 연결하거나 업데이트하도록 선택할 수 있습니다.

- 설치된 패키지 버전을 연결 AWS IoT Core 하지 않고에 사물을 등록합니다.
- AWS IoT 사물의 예약된 명명된 섀도우를 업데이트하도록 작업이 구성되지 않았습니다.
- 사내 프로세스를 사용하여 패키지 버전을 플릿으로 디스패치하고 해당 프로세스가 AWS IoT Core 완료되면 업데이트되지 않습니다.

> **Note**
>
> AWS IoT 작업을 사용하여 예약된 명명된 섀도우()에서 패키지 버전을 업데이트하는 것이 좋습니다$package. AWS IoT 작업이 $package 섀도우를 업데이트하도록 구성된 경우 다른 프로세스(예: 수동 또는 프로그래밍 방식 API 호출)를 통해 섀도우의 버전 파라미터를 업데이트하면 디바이스의 실제 버전과 예약된 명명된 섀도우에 보고된 버전 간에 불일치가 발생할 수 있습니다.

콘솔 또는 UpdateThingShadow API 작업을 통해 사물의 명명된 예약 섀도우($package)에 패키지 버전을 추가하거나 업데이트할 수 있습니다. 자세한 내용은 패키지 버전을 AWS IoT 사물에 연결을 참조하세요.

> **Note**
>
> 패키지 버전을 AWS IoT 사물에 연결해도 디바이스 소프트웨어가 직접 업데이트되지는 않습니다. 디바이스 소프트웨어를 업데이트하려면 패키지 버전을 디바이스에 배포해야 합니다.

소프트웨어 패키지 및 패키지 버전 제거

$null는 AWS IoT Jobs 서비스에 디바이스의 예약된 명명된 섀도우에서 기존 소프트웨어 패키지 및 패키지 버전을 제거하라는 메시지를 표시하는 예약 자리 표시자입니다$package. 자세한 내용은 명명된 예약 섀도우를 참조하세요.

이 기능을 사용하려면 버전명의 끝에 있는 destinationPackageVersion Amazon Resource Name(ARN)을 $null로 바꿔야 합니다. 그런 다음 디바이스에서 소프트웨어를 제거하도록 서비스에 지시합니다.

승인된 ARN은 다음의 형식을 사용합니다.

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

예:

```
$ aws iot create-job \
    ... \
    --destinationPackageVersions ["arn:aws:iot:us-east-1:111122223333:package/
samplePackage/version/$null"]
```

소프트웨어 패키지 카탈로그 시작

, AWS Management Console AWS IoT Core API 작업 및 AWS Command Line Interface ()를 통해 AWS IoT Device Management 소프트웨어 패키지 카탈로그를 빌드하고 유지 관리할 수 있습니다AWS CLI.

> **Note**
>
> AWS IoT 플릿 인덱싱을 활성화하려면 소프트웨어 패키지 카탈로그를 사용해야 합니다. 에서 소프트웨어 패키지 버전을 생성하고 `CreatePackage` API 명령을 AWS Management Console 사용하는 등의 기본 작업은 AWS IoT 플릿 인덱싱을 활성화하지 않으면 실패합니다. 소프트웨어 패키지 카탈로그에서 AWS IoT 플릿 인덱싱을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요플릿 인덱싱 준비.

콘솔 사용

를 사용하려면 AWS 계정에 AWS Management Console로그인하고 로 이동합니다AWS IoT Core. 탐색 창에서 소프트웨어 패키지를 선택합니다. 그런 다음 이 섹션에서 패키지와 해당 버전을 만들고 관리할 수 있습니다.

API 또는 CLI 작업 사용

AWS IoT Core API 작업을 사용하여 소프트웨어 패키지 카탈로그 기능을 생성하고 관리할 수 있습니다. 자세한 내용은 AWS IoT API 참조와AWS SDK 및 도구를 참조하세요. AWS CLI 명령은 카탈로그도 관리합니다. 자세한 내용은 AWS IoT CLI 명령 참조를 참조하세요.

이번 장은 다음과 같은 단원들로 구성되어 있습니다.

- 소프트웨어 패키지 및 패키지 버전 생성
- AWS IoT 작업을 통해 패키지 버전 배포
- 패키지 버전을 AWS IoT 사물에 연결

소프트웨어 패키지 및 패키지 버전 생성

다음 단계에 따라 AWS Management Console을 통해 패키지와 초기 버전을 만들 수 있습니다.

소프트웨어 패키지를 만드는 방법

1. AWS 계정에 로그인하고 AWS IoT 콘솔로 이동합니다.
2. 탐색 창에서 소프트웨어 패키지를 선택합니다.
3. AWS IoT 소프트웨어 패키지 페이지에서 패키지 생성을 선택합니다. 패키지 관리를 위한 종속성 활성화 대화 상자가 나타납니다.
4. 플릿 인덱싱에서 디바이스 소프트웨어 패키지 및 버전 추가를 선택합니다. 이는 소프트웨어 패키지 카탈로그에 필요하며 플릿에 대한 플릿 인덱싱 및 지표를 제공합니다.

5. [선택 사항] AWS IoT 작업이 성공적으로 완료되면 예약된 명명된 섀도우를 업데이트하도록 하려면 작업에서 섀도우 자동 업데이트를 선택합니다. AWS IoT 작업이이 업데이트를 수행하지 않도록 하려면이 확인란을 선택하지 않은 상태로 둡니다.
6. [선택 사항] 예약된 명명된 섀도우를 업데이트할 수 있는 권한을 AWS IoT 작업에 부여하려면 역할 선택에서 역할 생성을 선택합니다. AWS IoT 작업이이 업데이트를 수행하지 않도록 하려면이 역할이 필요하지 않습니다.
7. 역할을 생성하거나 선택합니다.

   a. 이 목적을 위한 역할이 없는 경우: 역할 생성생성 대화 상자가 나타나면 역할 이름을 입력한 다음 생성을 선택합니다.

   b. 이 목적을 위한 역할이 있는 경우: 역할 선택에서 역할을 선택한 다음 IAM 역할에 정책 연결 확인란이 선택되어 있는지 확인합니다.
8. 확인을 선택합니다. 새 패키지 생성 페이지가 나타납니다.
9. 패키지 세부 정보에서 패키지 이름을 입력합니다.
10. 패키지 설명에서 이 패키지를 식별하고 관리하는 데 도움이 되는 정보를 입력합니다.
11. [선택 사항] 태그를 사용하여 이 패키지를 분류하고 관리할 수 있습니다. 태그를 추가하려면 태그를 확장하고 태그 추가를 선택한 다음 키-값 페어를 입력합니다. 최대 50개의 태그를 추가할 수 있습니다. 자세한 내용은 AWS IoT 리소스 태그 지정을 참조하세요.

새 패키지를 만들 때 패키지 버전을 추가하는 방법

1. 첫 번째 버전에서 버전 이름을 입력합니다.

   패키지 버전을 고유하게 식별하려면 SemVer 형식(예:`1.0.0.0`)을 사용하는 것이 좋습니다. 사용 사례에 더 적합한 다른 형식 지정 전략을 사용할 수도 있습니다. 자세한 내용은 패키지 버전 수명 주기 단원을 참조하십시오.
2. 버전 설명에서 이 패키지 버전을 식별하고 관리하는 데 도움이 되는 정보를 입력합니다.

> **Note**
>
> 패키지 버전이 `draft` 상태에서 생성되므로 기본 버전 확인란은 비활성화되어 있습니다. 패키지 버전을 만든 후 상태를 `published`로 변경할 때 기본 버전의 이름을 지정할 수 있습니다. 자세한 내용은 패키지 버전 수명 주기 단원을 참조하십시오.

3. [선택 사항] 이 버전을 관리하거나 디바이스에 정보를 전달하는 데 도움이 되도록 버전 속성에 하나 이상의 이름-값 페어를 입력하세요. 입력한 각 이름-값 페어에 대해 속성 추가를 선택합니다. 자세한 내용은 버전 속성 단원을 참조하십시오.
4. [선택 사항] 태그를 사용하여 이 패키지를 분류하고 관리할 수 있습니다. 태그를 추가하려면 태그를 확장하고 태그 추가를 선택한 다음 키-값 페어를 입력합니다. 최대 50개의 태그를 추가할 수 있습니다. 자세한 내용은 AWS IoT 리소스 태그 지정을 참조하세요.
5. 다음을 선택합니다.

소프트웨어 재료표를 패키지 버전에 연결(선택 사항)

1. SBOMs 구성 창의 3단계: 버전 SBOM(선택 사항)에서 패키지 버전에 연결하기 전에 소프트웨어 재료표를 검증하는 데 사용되는 기본 SBOM 파일 형식 및 검증 모드를 선택합니다.
2. SBOM 파일 추가 창에서 버전이 지정된 Amazon S3 버킷을 나타내는 Amazon 리소스 이름(ARN)과 기본 유형이 작동하지 않는 경우 선호하는 SBOM 파일 형식을 입력합니다.

> **Note**
>
> 패키지 버전에 대한 소프트웨어 자재 명세서가 두 개 이상인 경우 단일 SBOM 파일 또는 여러 SBOM이 포함된 단일 zip 파일을 추가할 수 있습니다.

3. 추가된 SBOM 파일 창에서 패키지 버전에 추가한 SBOM 파일을 볼 수 있습니다.
4. 패키지 및 버전 생성을 선택합니다. 패키지 버전 페이지가 나타나고 추가된 SBOM 파일 창에서 SBOM 파일의 검증 상태를 볼 수 있습니다. SBOM 파일이 검증을 거치면 초기 상태가 `In progress`가 됩니다.

> **Note**
>
> SBOM 파일 검증 상태는 `Invalid file`, `Not started`, `In progress`, `Validated (SPDX)`, `Validated (CycloneDX)` 및 검증 실패 이유입니다.

AWS IoT 작업을 통해 패키지 버전 배포

다음 단계에 따라 AWS Management Console을 통해 패키지 버전을 배포할 수 있습니다.

사전 조건:

시작하기 전에 다음을 수행하십시오.

- 에 AWS IoT 사물을 등록합니다 AWS IoT Core. 디바이스를 추가하는 방법에 대한 지침은 사물 객체 생성을 AWS IoT Core참조하세요.
- [선택 사항] AWS IoT 패키지 버전을 배포할 디바이스를 대상으로 하는 사물 그룹 또는 동적 사물 그룹을 생성합니다. 사물 그룹을 생성하는 방법에 대한 지침은 정적 사물 그룹 생성을 참조하세요. 동적 사물 그룹을 생성하는 방법에 대한 지침은 동적 사물 그룹 생성을 참조하세요.
- 소프트웨어 패키지 및 패키지 버전 생성 자세한 내용은 소프트웨어 패키지 및 패키지 버전 생성 단원을 참조하십시오.
- 작업 문서를 만듭니다. 자세한 내용은 배포를 위한 작업 문서 및 패키지 버전 준비를 참조하세요.

AWS IoT 작업을 배포하려면

1. AWS IoT 콘솔에서 소프트웨어 패키지를 선택합니다.
2. 배포하려는 소프트웨어 패키지를 선택합니다. 소프트웨어 패키지 세부 정보 페이지가 나타납니다.
3. 버전에서 배포하려는 패키지 버전을 선택하고 작업 버전 배포를 선택합니다.
4. 이 포털을 통해 작업을 배포하는 것이 처음인 경우 요구 사항을 설명하는 대화 상자가 나타납니다. 정보를 검토하고 확인을 선택합니다.
5. 배포 이름을 입력하거나 이름 필드에 자동 생성된 이름을 그대로 둡니다.
6. [선택 사항] 설명 필드에 배포의 목적이나 내용을 식별하는 설명을 입력하거나 자동 생성된 정보를 그대로 둡니다.

   참고: 작업 이름 및 설명 필드에 개인 식별 정보를 사용하지 않는 것이 좋습니다.
7. [선택 사항] 이 작업과 연결할 태그를 추가합니다.
8. 다음을 선택합니다.
9. 작업 대상에서 작업을 받아야 하는 사물 또는 사물 그룹을 선택합니다.
10. 작업 파일 필드에 작업 문서 JSON 파일을 지정합니다.
11. 패키지 카탈로그 서비스와 작업 통합을 엽니다.
12. 작업 문서에 지정된 패키지 및 버전을 선택합니다.

> **Note**
>
> 작업 문서에 지정된 것과 동일한 패키지 및 패키지 버전을 선택해야 합니다. 더 많이 포함할 수 있지만 작업은 작업 문서에 포함된 패키지 및 버전에 대한 지침만 발행합니다. 자세한 내용은 배포 시 패키지 및 버전 이름 지정을 참조하세요.

13. 다음을 선택합니다.
14. 작업 구성 페이지의 작업 구성 대화 상자에서 다음 작업 유형 중 하나를 선택합니다.

    - 스냅샷 작업: 스냅샷 작업은 대상 디바이스 및 그룹에서 실행이 완료되면 완료됩니다.
    - 연속 작업: 연속 작업은 사물 그룹에 적용되며 이후에 지정된 대상 그룹에 추가하는 모든 디바이스에서 실행됩니다.
15. 추가 구성 - 선택 사항 대화 상자에서 다음과 같은 선택적 작업 구성을 검토하고 적절히 선택합니다. 자세한 내용은 작업 롤아웃, 예약 및 중단 구성과 작업 실행 제한 시간 및 재시도 구성을 참조하세요.

    - Rollout configuration(롤아웃 구성)
    - Scheduling configuration(예약 구성)
    - Job executions timeout configuration(작업 실행 제한 시간 구성)
    - 작업 실행 재시도 구성
    - Abort configuration(중단 구성)
16. 작업 선택을 검토한 다음 제출을 선택합니다.

작업을 생성하면 콘솔에서는 JSON 서명을 생성해 작업 문서에 배치합니다. AWS IoT 콘솔을 사용하여 작업 상태를 보거나 작업을 취소 또는 삭제할 수 있습니다. 작업을 관리하려면 콘솔의 작업 허브(Job hub of the console)로 이동합니다.

패키지 버전을 AWS IoT 사물에 연결

디바이스에 소프트웨어를 설치한 후 패키지 버전을 예약된 명명된 AWS IoT 사물 섀도에 연결할 수 있습니다. AWS IoT 작업이 배포되고 성공적으로 완료된 후 사물의 예약된 명명된 섀도우를 업데이트하도록 작업이 구성된 경우이 절차를 완료할 필요가 없습니다. 자세한 내용은 명명된 예약 섀도우 단원을 참조하십시오.

사전 조건:

시작하기 전에 다음을 수행하십시오.

- AWS IoT 사물 또는 사물을 생성하고를 통해 원격 측정을 설정합니다 AWS IoT Core. 자세한 내용은 시작하기를 참조하세요 AWS IoT Core.
- 소프트웨어 패키지 및 패키지 버전을 생성합니다. 자세한 내용은 소프트웨어 패키지 및 패키지 버전 생성 단원을 참조하십시오.
- 디바이스에 패키지 버전 소프트웨어를 설치합니다.

> **Note**
>
> 패키지 버전을 AWS IoT 사물에 연결해도 물리적 디바이스에 소프트웨어가 업데이트되거나 설치되지 않습니다. 패키지 버전을 디바이스에 배포해야 합니다.

패키지 버전을 AWS IoT 사물에 연결하려면

1. AWS IoT 콘솔 탐색 창에서 모든 디바이스 메뉴를 펼치고 사물을 선택합니다.
2. 목록에서 업데이트하려는 AWS IoT 사물을 식별하고 사물 이름을 선택하여 세부 정보 페이지를 표시합니다.
3. 세부 정보 섹션에서 패키지 및 버전을 선택합니다.
4. 패키지 및 버전에 추가를 선택합니다.
5. 디바이스 패키지 선택에서 원하는 소프트웨어 패키지를 선택합니다.
6. 버전 선택에서 원하는 소프트웨어 버전을 선택합니다.
7. 디바이스 패키지 추가를 선택합니다.

   패키지와 버전이 선택한 패키지 및 버전 목록에 표시됩니다.
8. 이 사물에 연결할 각 패키지 및 버전에 대해 이 단계를 반복합니다.
9. 완료하면 패키지 및 버전 세부 정보 추가를 선택합니다. 사물 세부 정보 페이지가 열리고 목록에서 새 패키지와 버전을 볼 수 있습니다.

AWS IoT 작업

AWS IoT 작업을 사용하여 연결된 하나 이상의 디바이스로 전송하고 실행할 수 있는 원격 작업 세트를 정의합니다 AWS IoT. 예를 들어 애플리케이션을 다운로드하여 설치하거나, 펌웨어 업데이트를 실행하거나, 재부팅하거나, 인증서를 교체하거나, 원격 문제 해결 작업을 수행하도록 디바이스 집합에 지시하는 작업을 정의할 수 있습니다.

AWS IoT 작업 액세스

콘솔 또는 API를 사용하여 AWS IoT 작업을 시작할 수 있습니다 AWS IoT Core .

콘솔 사용

에 로그인 AWS Management Console하고 AWS IoT 콘솔로 이동합니다. 탐색 창에서 관리를 선택한 다음 작업을 선택합니다. 이 섹션에서 작업을 생성하고 관리할 수 있습니다. 작업 템플릿을 생성하고 관리하려면 탐색 창에서 작업 템플릿(Job templates)을 선택합니다. 자세한 내용은 를 사용하여 작업 생성 및 관리 AWS Management Console 단원을 참조하십시오.

API 또는 CLI 사용

AWS IoT Core API 작업을 사용하여 시작할 수 있습니다. 자세한 내용은 AWS IoT API 참조를 참조하세요. 작업이 구축된 AWS IoT Core AWS IoT API는 AWS SDK에서 지원됩니다. 자세한 내용은 AWS SDK 및 도구 단원을 참조하세요.

AWS CLI 를 사용하여 작업 및 작업 템플릿을 생성하고 관리하기 위한 명령을 실행할 수 있습니다. 자세한 내용은 AWS IoT CLI 참조를 참조하세요.

AWS IoT 작업 리전 및 엔드포인트

AWS IoT 작업은에 고유한 컨트롤 플레인 및 데이터 플레인 API 엔드포인트를 지원합니다 AWS 리전. 데이터 영역 API 엔드포인트는 AWS 계정 및에 고유합니다 AWS 리전. AWS IoT 작업 엔드포인트에 대한 자세한 내용은 일반 참조의 AWS IoT Device Management - 작업 데이터 엔드포인트를 참조하세요. AWS

원격 작업이란 무엇인가요?

> **Note**
>
> 다음 리전에서는 무선 업데이트(OTA) 업데이트를 사용할 수 없습니다.

- 유럽(스페인)
- 아시아 태평양(말레이시아)

원격 작업은 작업자 또는 기술자의 물리적 존재 없이 원격으로 수행할 수 있는 물리적 디바이스, 가상 디바이스 또는 엔드포인트에서 수행할 수 있는 모든 업데이트 또는 작업입니다. 원격 작업은 무선 업데이트(OTA)를 사용하여 수행되므로 디바이스가 물리적으로 존재하지 않아도 됩니다. 에서 디바이스 플릿을 관리 AWS 클라우드 하면 디바이스가에 등록될 때 디바이스에서 원격 작업을 수행할 수 있습니다 AWS IoT Core.

AWS IoT Device Management 작업은에 등록된 디바이스에서 원격 작업을 수행하기 위한 확장 가능한 접근 방식을 제공합니다 AWS IoT Core. 작업은에서 생성 AWS 클라우드 되며 MQTT 또는 HTTP 프로토콜을 통해 OTA 업데이트를 사용하여 모든 대상 디바이스로 푸시됩니다.

AWS IoT Device Management 작업은 안전하고 확장 가능하며 비용 효율적인 방식으로 공장 재설정, 디바이스 재부팅, 소프트웨어 OTA 업데이트와 같은 원격 작업을 수행할 수 있는 기능을 제공합니다.

에 대한 자세한 내용은 단원을 AWS IoT Core참조하십시오AWS IoT란 무엇인가요?.

AWS IoT Device Management 작업에 대한 자세한 내용은 AWS IoT 작업이란 무엇입니까? 섹션을 참조하세요.

원격 작업에 AWS IoT Device Management 작업 사용의 이점

AWS IoT Device Management 작업을 사용하여 원격 작업을 수행하면 디바이스 플릿의 관리가 간소화됩니다. 다음 목록은 AWS IoT Device Management 작업을 사용하여 원격 작업을 수행할 때 얻을 수 있는 몇 가지 주요 이점을 강조합니다.

- 다른 AWS 서비스와 원활하게 통합
  - AWS IoT Device Management 작업은 다음과 같은 부가가치 AWS 서비스 및 기능과 긴밀하게 통합됩니다.
    - Amazon S3: 원격 작업 지침을 해당 콘텐츠에 대한 액세스 권한을 제어하는 안전한 Amazon S3 버킷에 저장합니다. Amazon S3 버킷을 사용하면 AWS IoT Device Management Software Package Catalog와 기본적으로 통합되는 확장 가능하고 내구성이 뛰어난 스토리지 솔루션을 제공하므로 작업이 업데이트 지침에서 참조하고 대체할 수 있습니다 AWS IoT Device Management . 자세한 내용은 What is Amazon S3?를 참조하세요
    - Amazon CloudWatch: 다른 디바이스 활동 외에도 각 디바이스에 대한 작업 실행의 원격 작업 구현 상태를 모니터링하고 기록하여 AWS IoT Device Management 작업의 전체 작업 성능을 추적

하고 분석합니다. 자세한 내용은 What is Amazon CloudWatch?를 참조하세요. 문제 해결을 위해 작업 로그를 모니터링하고 기록 데이터를 캡처합니다. 작업 작동 방식입니다.

  - AWS IoT 디바이스 섀도우 서비스: 디바이스 연결에 관계없이 애플리케이션 및 기타 서비스에서 디바이스 상태를 사용할 수 있도록 AWS IoT Device Management 작업을 사용하여 디바이스 섀도우를 통해 AWS IoT 사물의 디지털 표현을 유지합니다. 자세한 내용은 AWS IoT 디바이스 섀도우 서비스 단원을 참조하십시오.

- 보안 모범 사례
  - 권한 제어: Amazon S3를 사용하여 원격 운영 지침에 대한 액세스 권한을 제어하고 AWS IoT 정책 및 IAM 사용자 역할을 사용하여 디바이스 플릿에 원격 운영 지침을 배포할 수 있는 IAM 사용자를 결정합니다.
    - AWS IoT 정책에 대한 자세한 내용은 섹션을 참조하세요AWS IoT 정책 생성.
    - IAM 사용자 역할에 대한 자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS IoT 섹션을 참조하세요.
- 확장성
  - 대상 작업 배포 : 작업을 생성할 때 작업 문서에 입력된 특정 디바이스 그룹화 기준을 사용하여 대상 작업 배포가 있는 작업에서 작업 문서를 수신하는 디바이스를 제어합니다. 각 디바이스에 대해 AWS IoT 사물을 생성하고 해당 정보를 AWS IoT 레지스트리에 저장하면 플릿 인덱싱을 사용하여 대상 검색을 수행할 수 있습니다. 플릿 인덱싱 검색 결과를 기반으로 사용자 지정 그룹을 생성하여 대상 작업 배포를 지원할 수 있습니다. 자세한 내용은 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하십시오. 작업을 사용하여 스냅샷과 연속 작업을 비교합니다.
  - 작업 상태: 각 디바이스에서 작업 문서의 개별 구현 상태 외에도 디바이스 플릿 수준에서 디바이스 플릿에 대한 작업 문서 롤아웃의 상태 및 전체 작업 상태를 추적합니다. 자세한 내용은 작업 및 작업 실행 상태 단원을 참조하십시오.
  - 새 디바이스 확장성: 연속 작업을 통해 플릿 인덱싱을 사용하여 생성된 기존 사용자 지정 그룹에 작업 문서를 추가하여 새 디바이스에 쉽게 배포합니다. 이렇게 하면 작업 문서를 각 새 디바이스에 별도로 배포해야 하는 시간을 절약할 수 있습니다. 또는 작업 문서를 미리 결정된 디바이스 그룹에 한 번 배포한 다음 작업이 완료되어 스냅샷 샷으로 더 대상화된 접근 방식을 사용할 수 있습니다.
- 유연성
  - 작업 구성: 특정 요구 사항에 맞게 선택적 작업 구성 롤아웃, 예약, 중단, 제한 시간 및 재시도를 사용하여 작업 및 작업 문서를 사용자 지정합니다. 자세한 내용은 작업 구성 단원을 참조하십시오.
- 비용 효율성
  - AWS IoT Device Management 작업을 활용하여 중요한 업데이트를 배포하고 일상적인 유지 관리 작업을 수행하여 디바이스 플릿을 유지 관리하기 위한 보다 효율적인 비용 구조를 소개합니다. 디바이스 플릿을 유지하기 위한 DIY(Do-It-Yourself) 솔루션에는 DIY 솔루션을 호스팅하고 관리하는

데 필요한 인프라, DIY 솔루션을 개발, 유지 및 확장하는 데 드는 인건비, 데이터 전송 비용과 같은 반복적인 가변 비용이 포함됩니다. AWS IoT Device Management 작업의 투명한 고정 비용 구조를 활용하면 디바이스에 대한 작업 문서 롤아웃을 용이하게 하고 각 디바이스의 작업 실행 상태를 추적하는 데 필요한 데이터 전송 비용 외에도 디바이스의 각 작업 실행에 드는 비용을 정확히 알 수 있습니다. 자세한 내용은 AWS IoT Core 요금을 참조하십시오.

AWS IoT 작업이란 무엇입니까?

AWS IoT 작업을 사용하여 연결된 하나 이상의 디바이스로 전송하고 실행할 수 있는 원격 작업 세트를 정의합니다 AWS IoT.

작업을 생성하려면 먼저 디바이스가 원격으로 수행해야 하는 작업을 설명하는 지침 목록이 포함된 작업 문서를 정의합니다. 이러한 작업을 수행하려면 개별 사물, 사물 그룹 또는 둘 다인 대상 목록을 지정합니다. 작업 문서와 대상이 함께 배포를 구성합니다.

배포마다 추가 구성이 있을 수 있습니다.

- 롤아웃(Rollout): 이 구성은 분당 작업 문서를 수신하는 디바이스 수를 정의합니다.
- 중단: 특정 수의 디바이스가 작업 알림을 수신하지 못하는 경우 이 구성을 사용하여 작업을 취소합니다. 이렇게 하면 전체 플릿에 잘못된 업데이트를 보내는 것을 방지할 수 있습니다.
- 시간 제한(Timeout): 일정 기간 내에 작업 대상에서 응답을 수신하지 못하면 작업이 실패할 수 있습니다. 이러한 디바이스에서 실행 중인 작업을 추적할 수 있습니다.
- 재시도: 디바이스가 실패 또는 작업 시간 초과를 보고하는 경우 AWS IoT 작업을 사용하여 디바이스에 작업 문서를 자동으로 재전송할 수 있습니다.
- Scheduling(예약): 이 구성을 사용하면 미래 날짜 및 시간으로 작업을 예약할 수 있습니다. 또한 사전 정의된 트래픽이 적은 기간 동안 디바이스를 업데이트하는 반복 유지 관리 기간을 만들 수 있습니다.

AWS IoT 작업은 대상에 작업을 사용할 수 있음을 알리는 메시지를 보냅니다. 대상은 작업 문서를 다운로드하고, 지정한 작업을 수행하고, 진행 상황을 보고하여 작업 실행을 시작합니다 AWS IoT. 작업에서 제공하는 명령을 실행하여 특정 대상 또는 모든 대상에 대한 AWS IoT 작업 진행 상황을 추적할 수 있습니다. 작업이 시작되면 진행 중(In progress) 상태가 됩니다. 그런 다음 디바이스는 작업이 성공, 실패 또는 시간 초과될 때까지 이 상태를 표시하면서 증분 업데이트를 보고합니다.

다음 주제에서는 작업의 몇 가지 주요 개념과 작업 및 작업 실행의 수명 주기에 대해 설명합니다.

주제

- 작업 관련 주요 개념
- 작업 및 작업 실행 상태

작업 관련 주요 개념

다음 개념은 AWS IoT 작업에 대한 세부 정보와 디바이스에서 원격 작업을 실행하기 위한 작업을 생성하고 배포하는 방법을 제공합니다.

기본 개념

다음은 AWS IoT 작업 사용 시 알아야 할 기본 개념입니다.

작업

작업은 AWS IoT에 연결되는 하나 이상의 디바이스로 전송되어 실행되는 원격 작업입니다. 예를 들어 애플리케이션을 다운로드하여 설치하거나, 펌웨어 업데이트를 실행하거나, 재부팅하거나, 인증서를 교체하거나, 원격 문제 해결 작업을 수행하도록 일련의 디바이스에 지시하는 작업을 정의할 수 있습니다.

작업 문서

작업을 생성하려면 먼저 디바이스에서 수행할 원격 작업을 설명하는 작업 문서를 생성해야 합니다.

작업 문서는 UTF-8 인코딩 JSON 문서이며, 여기에는 디바이스가 작업을 실행하는 데 필요한 정보가 포함됩니다. 작업 문서에는 디바이스가 업데이트 또는 기타 데이터를 다운로드할 수 있는 URL이 1개 이상 포함됩니다. 작업 문서는 Amazon S3 버킷에 저장되거나, 혹은 작업 생성 명령에 인라인으로 포함될 수도 있습니다.

> **Note**
>
> MQTT 응답에서 `jobDocument` 필드는 JSON 객체입니다. HTTP 응답에서 JSON 객체의 문자열 표현입니다.

대상

작업을 생성할 때 작업을 수행해야 하는 디바이스를 포함하는 대상 목록을 지정합니다. 여기서 대상은 사물 또는 사물 그룹 또는 두 가지 모두가 될 수 있습니다. AWS IoT 작업 서비스는 각 대상에 작업을 사용할 수 있음을 알리는 메시지를 보냅니다.

배포

작업 문서를 제공하고 대상 목록을 지정하여 작업을 생성한 후 업데이트를 수행하려는 원격 대상 디바이스에 작업 문서가 배포됩니다. 스냅샷 작업의 경우 작업은 대상 디바이스에 배포된 후 완료됩니다. 연속 작업의 경우 작업은 그룹에 추가될 때 디바이스 그룹에 배포됩니다.

작업 실행

작업 실행은 대상 디바이스의 작업 인스턴스입니다. 대상은 작업 문서를 다운로드하여 작업 실행을 시작합니다. 그런 다음 문서에 지정된 작업을 수행하고 진행 상황을 보고합니다 AWS IoT. 실행 번호는 특정 대상에서 작업 실행의 고유 식별자입니다. AWS IoT 작업 서비스는 대상에 대한 작업 실행 진행 상황과 모든 대상에 대한 작업 진행 상황을 추적하는 명령을 제공합니다.

작업 유형 개념

다음 개념은 작업으로 생성할 수 있는 다양한 유형의 AWS IoT 작업에 대해 더 잘 이해하는 데 도움이 될 수 있습니다.

스냅샷 작업

기본적으로 작업은 생성할 때 지정하는 모든 대상으로 전송됩니다. 이러한 대상들이 작업을 마치면 (또는 작업을 완료할 수 없다고 보고하면) 작업은 완료됩니다.

연속 작업

연속 작업은 생성할 때 지정하는 모든 대상으로 전송됩니다. 이 작업은 계속 실행되고 대상 그룹에 새로 추가되는 모든 디바이스(사물)로 전송됩니다. 예를 들어 연속 작업은 그룹에 추가된 디바이스를 온보딩하거나 업그레이드할 때 사용됩니다. 작업을 생성할 때 옵션으로 제공되는 파라미터를 설정하면 작업을 연속으로 실행할 수 있습니다.

> **Note**
>
> 동적 사물 그룹을 사용하여 IoT 플릿의 대상을 지정할 때는 스냅샷 작업 대신 연속 작업을 사용하는 것이 좋습니다. 연속 작업을 사용하면 그룹에 조인하는 디바이스는 작업이 생성된 후에도 작업 실행을 수신합니다.

미리 서명된 URL

작업 문서에 포함되지 않은 데이터에 대한 안전한 시간 제한 액세스를 위해 미리 서명된 Amazon S3 URL을 사용할 수 있습니다. 먼저 데이터를 Amazon S3 버킷에 저장한 후 자리 표시자 링크를 작업 문

서의 데이터에 추가합니다. AWS IoT Jobs가 작업 문서에 대한 요청을 받으면 자리 표시자 링크를 찾아 작업 문서를 구문 분석한 다음 미리 서명된 Amazon S3 URLs.

자리 표시자 링크는 다음과 같은 형식을 따릅니다.

```
${aws:iot:s3-presigned-url:https://s3.amazonaws.com/bucket/key}
```

여기에서 *bucket*은 버킷 이름이고, *key*는 링크를 연결할 버킷 객체입니다.

베이징 및 닝샤 리전에서 미리 서명된 URL은 리소스 소유자에게 인터넷 콘텐츠 공급자(ICP) 라이선스가 있는 경우에만 작동합니다. 자세한 내용은 중국에서 서비스 시작하기 설명서의 Amazon Simple Storage Service를 참조하세요. AWS

작업 구성 개념

다음 개념은 작업 구성 방법을 이해하는 데 도움이 될 수 있습니다.

롤아웃

대상에게 대기 중인 작업 실행을 얼마나 빨리 알릴지 지정할 수 있습니다. 이로써 단계별 롤아웃을 생성하여 업데이트, 재부팅 및 기타 작업을 더욱 효과적으로 관리할 수 있습니다. 정적 롤아웃 속도 또는 기하급수적인 롤아웃 속도를 사용하여 롤아웃 구성을 생성할 수 있습니다. 분당 알릴 최대 작업 대상 수를 지정하려면 정적 롤아웃 속도를 사용합니다.

롤아웃 속도 설정의 예와 작업 롤아웃 구성에 대한 자세한 내용은 작업 롤아웃, 예약 및 중단 구성 섹션을 참조하세요.

일정 예약

작업 예약 기능을 사용하면 연속 작업과 스냅샷 작업에서 대상 그룹의 모든 디바이스에 대한 작업 문서의 롤아웃 기간을 예약할 수 있습니다. 또한 작업이 대상 그룹의 모든 디바이스에 작업 문서를 롤아웃할 특정 날짜 및 시간을 포함하는 유지 관리 기간을 선택적으로 생성할 수 있습니다. 유지 관리 기간이란 초기 작업 또는 작업 템플릿 생성 중에 매일, 매주, 매월 또는 사용자 정의 날짜 및 시간 간격으로 반복되는 인스턴스입니다. 유지 관리 기간 중에는 연속 작업만 롤아웃을 수행하도록 예약할 수 있습니다.

작업 예약은 해당 작업에만 적용됩니다. 개별 작업 실행은 예약할 수 없습니다. 자세한 내용은 작업 롤아웃, 예약 및 중단 구성 단원을 참조하십시오.

중단

지정된 특정 기준을 충족하지 않으면 롤아웃을 취소할 조건 집합을 생성할 수 있습니다. 자세한 내용은 작업 롤아웃, 예약 및 중단 구성 단원을 참조하십시오.

시간 초과

작업 시간 초과 기능은 작업 배포가 예상치 않게 장시간 동안 IN_PROGRESS 상태에 멈출 때마다 알립니다. 타이머에는 진행 중 타이머와 단계 타이머 두 가지 유형이 있습니다. 작업이 IN_PROGRESS 상태이면 작업 배포의 진행 상황을 모니터링하고 추적할 수 있습니다.

롤아웃 및 중단 구성은 작업에 따라 다르지만 시간 제한 구성은 작업 배포에 따라 다릅니다. 자세한 내용은 작업 실행 제한 시간 및 재시도 구성 단원을 참조하십시오.

재시도

작업 재시도를 사용하면 작업이 실패하거나 시간이 초과되는 경우 또는 둘 모두 발생하는 경우 작업 실행을 재시도할 수 있습니다. 작업을 실행하기 위해 최대 10번의 재시도를 수행할 수 있습니다. 재시도 진행 상황과 작업 실행의 성공 여부를 모니터링하고 추적할 수 있습니다.

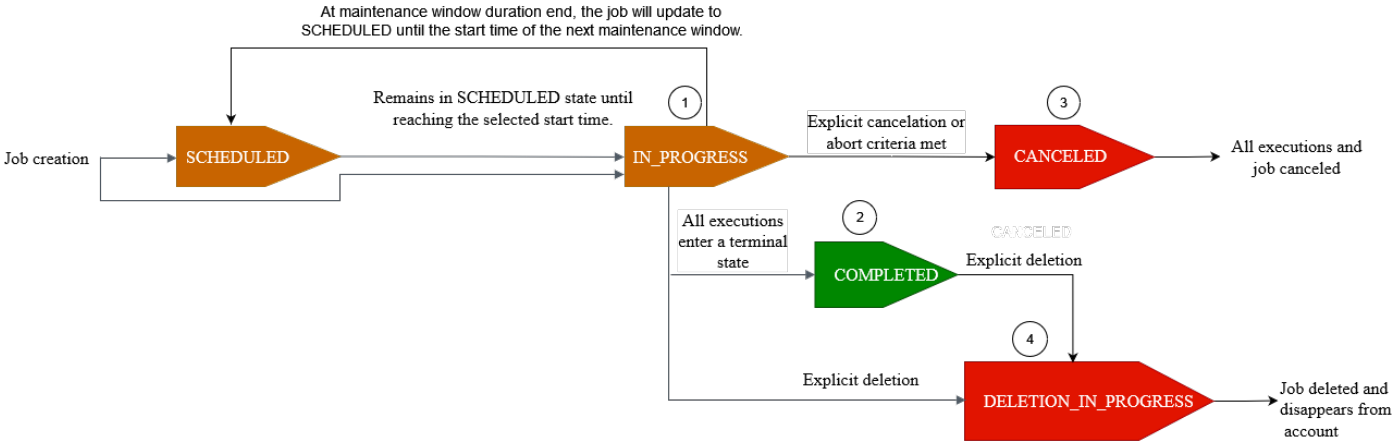
롤아웃 및 중단 구성은 작업에 따라 다르지만 제한 시간 및 재시도 구성은 작업 실행에 따라 다릅니다. 자세한 내용은 작업 실행 제한 시간 및 재시도 구성 단원을 참조하십시오.

작업 및 작업 실행 상태

다음 섹션에서는 AWS IoT 작업의 수명 주기와 작업 실행의 수명 주기에 대해 설명합니다.

작업 상태

다음 다이어그램은 AWS IoT 작업의 다양한 상태를 보여줍니다.



작업을 사용하여 생성하는 AWS IoT 작업은 다음 상태 중 하나일 수 있습니다.

- SCHEDULED

콘솔, CreateJob API 또는 CreateJobTemplate API를 AWS IoT 사용하여 초기 작업 또는 작업 템플릿을 생성하는 동안 AWS IoT 콘솔에서 또는 CreateJob API 또는 CreateJobTemplate APISchedulingConfig에서 선택적 예약 구성을 선택할 수 있습니다. 특정 startTime, endTime 및 endBehavior를 포함하는 예약된 작업을 시작하면 작업 상태가 SCHEDULED로 업데이트됩니다. 작업이 선택한 startTime 또는 다음 유지 관리 기간(유지 관리 기간에 작업 롤아웃을 선택한 경우)의 startTime에 도달하면 상태가 SCHEDULED에서 IN_PROGRESS로 전환되며 대상 그룹의 모든 디바이스에 대한 작업 문서 롤아웃이 시작됩니다.

- IN_PROGRESS

  AWS IoT 콘솔 또는 CreateJob API를 사용하여 작업을 생성하면 작업 상태가 로 업데이트됩니다IN_PROGRESS. 작업 생성 중에 AWS IoT 작업은 대상 그룹의 디바이스에 작업 실행을 롤아웃하기 시작합니다. 모든 작업 실행이 롤아웃된 후 AWS IoT 작업은 디바이스가 원격 작업을 완료할 때까지 기다립니다.

  진행 중인 작업에 적용되는 동시성 및 제한에 대한 자세한 정보는 AWS IoT 작업 제한 섹션을 참조하세요.

  > **Note**
  >
  > IN_PROGRESS 작업이 현재 유지 관리 기간의 끝에 도달하면 작업 문서의 롤아웃이 중지됩니다. 작업은 다음 유지 관리 기간의 startTime까지 SCHEDULED로 업데이트됩니다.

- COMPLETED

  연속 작업은 다음 방법 중 하나로 처리됩니다.

  - 선택적 스케줄링 구성을 선택하지 않은 연속 작업의 경우 작업이 항상 진행 중이며 대상 그룹에 추가된 새 디바이스에 대해 계속 실행됩니다. COMPLETED 상태에 절대 도달하지 않습니다.
  - 선택적 예약 구성을 선택한 연속 작업의 경우 다음 설명이 참입니다.
    - endTime이 제공된 경우 연속 작업은 endTime이 지나고 모든 작업 실행이 최종 상태에 도달하면 COMPLETED 상태가 됩니다.
    - 선택적 예약 구성에서 endTime이 제공되지 않은 경우 연속 작업은 작업 문서 롤아웃을 계속 수행합니다.

  스냅샷 작업의 경우 모든 작업 실행이 SUCCEEDED, FAILED, TIMED_OUT, REMOVED 또는 CANCELED와 같은 최종 상태가 되면 작업 상태가 COMPLETED로 변경됩니다.

- CANCELED

AWS IoT 콘솔, CancelJob API 또는를 사용하여 작업을 취소하면 작업 중단 구성작업 상태가 로 변경됩니다CANCELED. 작업 취소 중에 AWS IoT 작업은 이전에 생성된 작업 실행을 취소하기 시작합니다.

취소 중인 작업에 적용되는 동시성 및 제한에 대한 자세한 정보는 AWS IoT 작업 제한 섹션을 참조하세요.

- DELETION_IN_PROGRESS

  AWS IoT 콘솔 또는 DeleteJob API를 사용하여 작업을 삭제하면 작업 상태가 로 변경됩니다DELETION_IN_PROGRESS. 작업 삭제 중에 AWS IoT 작업은 이전에 생성된 작업 실행을 삭제하기 시작합니다. 모든 작업 실행이 삭제되면 AWS 계정에서 작업이 사라집니다.

작업 실행 상태

다음 표에는 AWS IoT 작업 실행의 다양한 상태와 상태 변경이 디바이스에 의해 시작되는지 아니면 AWS IoT 작업에 의해 시작되는지가 나와 있습니다.

작업 실행 상태 및 소스

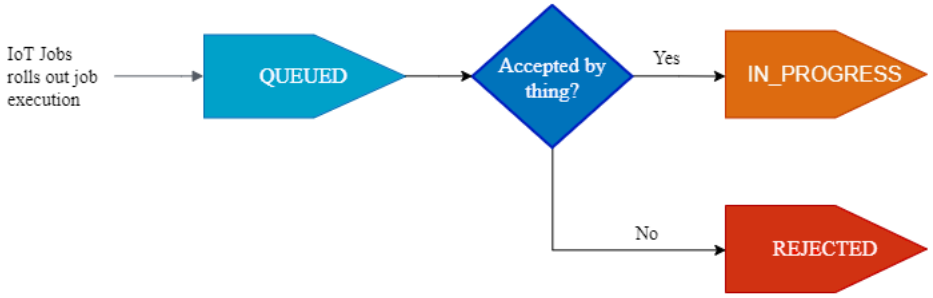
작업 실행 상태	디바이스에 의해 시작	AWS IoT 작업에 의해 시작되었나요?	최종 상태	재시도 가능
QUEUED	아니요	예	아니요	해당 사항 없음
IN_PROGRESS	예	아니요	아니요	해당 사항 없음
SUCCEEDED	예	아니요	예	해당 사항 없음
FAILED	예	아니요	예	예
TIMED_OUT	아니요	예	예	예
REJECTED	예	아니요	예	아니요
REMOVED	아니요	예	예	아니요
CANCELED	아니요	예	예	아니요

다음 섹션에서는 작업으로 작업을 생성할 때 롤아웃되는 작업 실행의 상태에 대해 자세히 설명합니다 AWS IoT .

- 대기됨

  AWS IoT 작업이 대상 디바이스에 대한 작업 실행을 롤아웃하면 작업 실행 상태가 로 설정됩니다QUEUED. 작업 실행은 다음까지 QUEUED 상태로 유지됩니다.

  - 디바이스가 작업 실행을 수신하고 Jobs API 작업을 호출한 후 상태를 IN_PROGRESS로 보고합니다.
  - 사용자가 작업 또는 작업 실행을 취소하거나, 지정한 중단 조건이 충족되거나, 상태가 CANCELED로 변경됩니다.
  - 디바이스가 대상 그룹에서 제거되고 상태가 REMOVED로 변경됩니다.



- IN_PROGRESS

  IoT 디바이스가 예약 작업 주제 $notify 및를 구독$notify-next하고 디바이스가 상태StartNextPendingJobExecution인 API 또는 UpdateJobExecution API를 호출하는 경우 IN_PROGRESS AWS IoT 작업 실행 상태는 로 설정됩니다IN_PROGRESS.

  UpdateJobExecution API는 IN_PROGRESS 상태로 여러 번 호출될 수 있습니다. statusDetails 객체를 사용하여 실행 단계에 대한 추가 세부 정보를 지정할 수 있습니다.

  > **Note**
  >
  > 각 디바이스에 대해 여러 작업을 생성하는 경우 AWS IoT 작업과 MQTT 프로토콜은 전송 순서를 보장하지 않습니다.

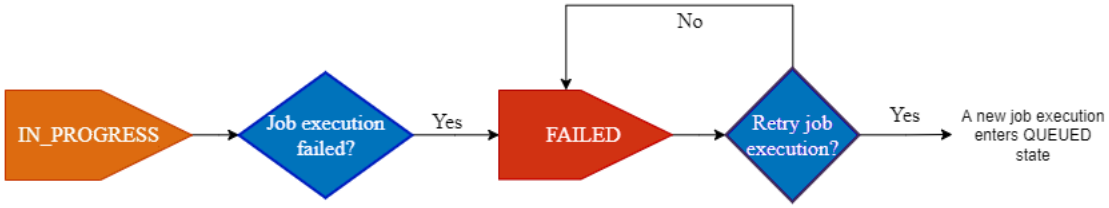
- SUCCEEDED

  디바이스가 원격 작업을 성공적으로 완료하면 디바이스는 작업 실행이 성공했음을 나타내기 SUCCEEDED 위해 상태로 UpdateJobExecution API를 호출해야 합니다. 그러면 AWS IoT 작업은를 업데이트하고 작업 실행 상태를 로 반환합니다SUCCEEDED.



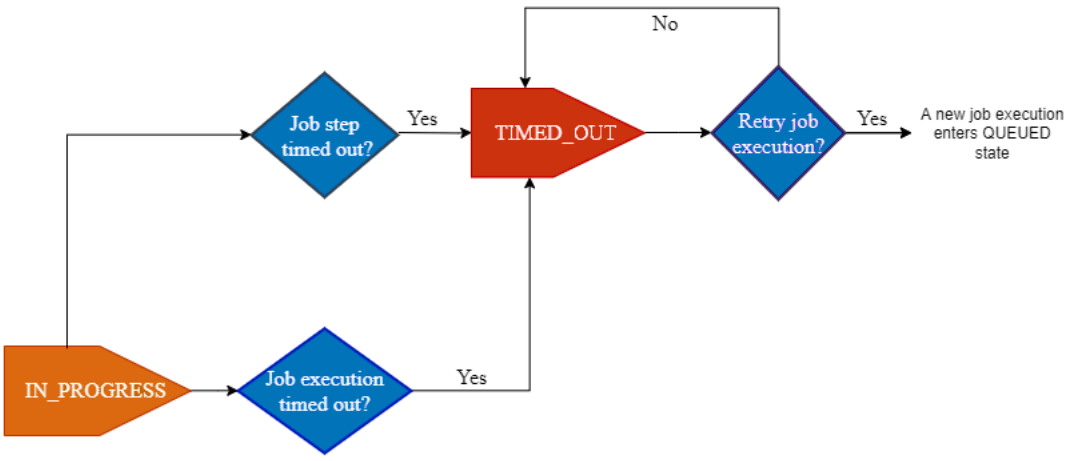
- FAILED

  디바이스가 원격 작업을 완료하지 못하면 디바이스는 작업 실행이 실패했음을 `Failed` 나타내는 상태로 `UpdateJobExecution` API를 호출해야 합니다. 그러면 AWS IoT 작업은를 업데이트하고 작업 실행 상태를 로 반환합니다`Failed`. 작업 실행 재시도 구성을 사용하여 디바이스에 대해 이 작업 실행을 다시 시도할 수 있습니다.



- TIMED_OUT

  상태가 인 경우 디바이스가 작업 단계를 완료하지 못`IN_PROGRESS`하거나 진행 중인 타이머의 제한 시간 내에 원격 작업을 완료하지 못하면 AWS IoT 작업은 작업 실행 상태를 로 설정합니다`TIMED_OUT`. 또한 진행 중인 작업의 각 작업 단계에 대한 단계 타이머가 있으며 작업 실행에만 적용됩니다. 이 진행 타이머 기간은 작업 실행 시간 제한 구성의 `inProgressTimeoutInMinutes` 속성을 사용하여 지정됩니다. 작업 실행 재시도 구성을 사용하여 디바이스에 대해 이 작업 실행을 다시 시도할 수 있습니다.



- REJECTED

디바이스가 유효하지 않거나 호환되지 않는 요청을 수신하면 디바이스는 상태가 인 `UpdateJobExecution` API를 호출해야 합니다`REJECTED`. 그러면 AWS IoT 작업이 업데이트되고 작업 실행 상태가 로 반환됩니다`REJECTED`.

- REMOVED

  디바이스가 동적 사물 그룹에서 분리된 경우와 같이 디바이스가 더 이상 작업 실행의 유효한 대상이 아닌 경우 AWS IoT Jobs는 작업 실행 상태를 `REMOVED`로 설정합니다. 대상 그룹에 사물을 다시 연결하고 디바이스에 대한 작업 실행을 다시 시작할 수 있습니다.

- CANCELED

  콘솔이나 `CancelJob` 또는 `CancelJobExecution` API를 사용하여 작업을 취소하거나 작업 실행을 취소하거나를 사용하여 지정된 중단 기준이 작업 중단 구성 충족되면 AWS IoT 작업은 작업을 취소하고 작업 실행 상태를 로 설정합니다`CANCELED`.

작업 관리

작업을 사용하여 소프트웨어 또는 펌웨어 업데이트를 디바이스에 알립니다. AWS IoT 콘솔, 작업 관리 및 제어 API 작업, AWS Command Line Interface 또는 AWS SDK를 사용하여 작업을 생성하고 관리할 수 있습니다.

작업에 대한 코드 서명

디바이스에 코드를 전송할 때 디바이스에서 코드가 전송 중 수정되었는지 감지하려면 AWS CLI를 사용하여 코드 파일에 서명하는 것이 좋습니다. 지침은 AWS CLI를 사용하여 작업 생성 및 관리를 참조하세요.

자세한 내용은 코드 서명이란 무엇입니까 AWS IoT?를 참조하세요.

작업 문서

작업을 생성하려면 작업 문서를 생성해야 합니다. 에 코드 서명을 사용하는 경우 버전이 지정된 Amazon S3 버킷에 작업 문서를 업로드 AWS IoT해야 합니다. 파일을 만들고 Amazon S3 버킷을 만들고 이 버킷에 파일을 업로드하는 방법에 대한 자세한 내용은 Amazon S3 시작 안내서의 Amazon Simple Storage Service 시작하기를 참조하세요.

> **Tip**
>
> 작업 문서 예제는 JavaScript용 AWS IoT SDK의 job-agent.js 예제를 참조하세요.

미리 서명된 URL

작업 문서에는 코드 파일(또는 기타 파일)을 가리키는 미리 서명된 Amazon S3 URL을 포함할 수 있습니다. 미리 서명된 Amazon S3 URL은 제한된 시간 동안 유효하므로 디바이스가 작업 문서를 요청할 때 생성되지 않습니다. 미리 서명된 URL은 작업 문서를 만들 때에는 생성되지 않았으므로 작업 문서에는 자리 표시자 URL을 대신 사용합니다. 자리 표시자 URL은 다음과 같습니다.

```
${aws:iot:s3-presigned-url-v2:https://
s3.region.amazonaws.com/<bucket>/<code file>}
```

여기서 각 항목은 다음과 같습니다.

- *bucket*은 코드 파일이 들어 있는 Amazon S3 버킷입니다.
- *code file*은 코드 파일의 Amazon S3 키입니다.

디바이스가 작업 문서를 요청하면는 미리 서명된 URL을 AWS IoT 생성하고 자리 표시자 URL을 미리 서명된 URL로 바꿉니다. 그러면 디바이스로 작업 문서가 전송됩니다.

S3에서 파일을 다운로드할 수 있는 권한을 부여하는 IAM 역할

사전 서명된 Amazon S3 URL을 사용하여 작업을 생성할 경우, IAM 역할을 제공해야 합니다. 역할은 데이터 또는 업데이트가 저장되어 있는 Amazon S3 버킷에서 파일을 다운로드할 수 있는 권한을 부여해야 합니다. 또한 이 역할은 AWS IoT 에게 역할을 위임할 수 있는 권한을 부여해야 합니다.

미리 서명된 URL에 제한 시간을 선택적으로 지정할 수도 있습니다. 자세한 내용은 CreateJob을 참조하세요.

역할을 수임할 수 있는 AWS IoT 작업 권한 부여

1. IAM 콘솔의 역할 허브로 이동하고 역할을 선택합니다.
2. 신뢰 관계(Trust Relationships) 탭에서 신뢰 관계 편집(Edit Trust Relationship)을 선택하고 정책 문서를 다음 JSON으로 바꿉니다. 신뢰 정책 업데이트를 선택합니다.

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iot.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 혼동된 대리자 문제를 방지하기 위하여 전역 조건 컨텍스트 키 aws:SourceArn 및 aws:SourceAccount를 정책에 추가합니다.

> **Important**
>
> aws:SourceArn은 arn:aws:iot:*region*:*account-id*:* 형식을 따라야 합니다. *region*이 AWS IoT 리전과 일치하고 *account-id*가 고객 계정 ID와 일치하는지 확인합니다. 자세한 설명은 교차 서비스 혼동된 대리자 예방을 참조하세요.

```
{
  "Effect": "Allow",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service":
          "iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole",
      "Condition": {
         "StringEquals": {
            "aws:SourceAccount": "123456789012"
         },
         "ArnLike": {
              "aws:SourceArn": "arn:aws:iot:*:123456789012:job/*"
         }
```

```
        }
      }
    ]
}
```

4. 해당 작업에서 Amazon S3 객체인 작업 문서를 사용할 경우, 권한 을 선택하고 다음 JSON을 사용합니다. 그러면 Amazon S3 버킷에서 파일을 다운로드할 권한을 부여하는 정책이 추가됩니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your_S3_bucket/*"
        }
    ]
}
```

파일 업로드를 위한 미리 서명된 URL

디바이스가 작업 배포 중에 Amazon S3 버킷에 파일을 업로드해야 하는 경우 작업 문서에 다음과 같은 미리 서명된 URL 자리 표시자를 포함할 수 있습니다.

```
${aws:iot:s3-presigned-url-upload:https://s3.region.amazonaws.com/<bucket>/<key>}
```

작업 문서에 있는 파일 업로드 자리 표시자 URL의 key 속성 내에서 최대 2개의 ${thingName}, ${jobId}, 및 ${executionNumber}를 예약 키워드로 사용할 수 있습니다. key 속성에서 예약된 키워드를 나타내는 로컬 자리 표시자는 구문 분석되고 작업 실행이 생성될 때 대체됩니다. 각 디바이스에 대해 예약된 키워드가 있는 로컬 자리 표시자를 사용하면 디바이스에서 업로드된 각 파일이 해당 디바이스에 고유하고 동일한 작업 배포의 대상이 되는 다른 디바이스에서 유사한 업로드된 파일로 덮어쓰지 않습니다. 작업 배포 중에 파일을 업로드하기 위해 미리 서명된 URL 자리 표시자 내의 로컬 자리 표시자 문제를 해결하는 방법에 대한 자세한 내용은 오류 메시지 일반 문제 해결 섹션을 참조하세요.

> **Note**
>
> Amazon S3 버킷 이름에는 업로드된 파일에 대해 예약된 키워드를 나타내는 로컬 자리 표시자를 포함할 수 없습니다. 로컬 자리 표시자는 key 속성에 있어야 합니다.

디바이스가 이 미리 서명된 URL 자리 표시자를 수신하면 작업 문서에서 미리 서명된 Amazon S3 업로드 URL로 변환됩니다. 디바이스에서 이를 사용하여 대상 Amazon S3 버킷에 파일을 업로드합니다.

> **Note**
>
> 위의 자리 표시자 URL에 Amazon S3 버킷과 키가 제공되지 않은 경우 AWS IoT Jobs는 각 디바이스에 대해 `${thingName}`, `${jobId}` 및 `${executionNumber}`의 최대 2개를 사용하여 키를 자동으로 생성합니다.

Amazon S3 버전 관리를 사용하여 미리 서명된 URL

Amazon S3 버킷에 저장된 파일의 무결성을 보호하는 것은 해당 파일을 사용하여 디바이스 플릿에 안전한 작업 배포를 보장하는 데 중요합니다. Amazon S3 버전 관리를 사용하면 Amazon S3 버킷에 저장된 파일의 각 변형에 대한 버전 식별자를 추가하여 파일의 각 버전을 추적할 수 있습니다. 이를 통해 AWS IoT 작업을 사용하여 디바이스 플릿에 배포되는 파일 버전을 파악할 수 있습니다. Amazon S3 버전 관리에 대한 자세한 내용은 Amazon S3 버킷에서 버전 관리 사용을 참조하세요.

파일이 Amazon S3에 저장되고 작업 문서에 미리 서명된 URL 자리 표시자가 포함된 경우 AWS IoT , 작업에서는 Amazon S3 버킷, 버킷 키 및 Amazon S3 버킷에 저장된 파일 버전을 사용하여 작업 문서에 미리 서명된 URL을 생성합니다. Amazon S3 작업 문서에서 생성된 이 미리 서명된 URL은 작업 문서에서 원래 미리 서명된 URL 자리 표시자를 대체합니다. Amazon S3 버킷에 저장된 파일을 업데이트하면 새 버전의 파일 및 후속 `versionId`가 생성되어 업데이트에 신호를 보내고 향후 작업 배포에서 해당 특정 파일을 대상으로 지정할 수 있는 기능을 제공합니다.

`versionId`를 사용하여 작업 문서에서 미리 서명된 Amazon S3 URL을 보기 전 및 보는 동안 다음 예제를 참조하세요.

Amazon S3 사전 서명된 URL 자리 표시자(작업 배포 전)

```
//Virtual-hosted style URL
${aws:iot:s3-presigned-url-v2:https://bucket-name.s3.region-code.amazonaws.com/key-name%3FversionId%3Dversion-id}

//Path-style URL
${aws:iot:s3-presigned-url-v2:https://s3.region-code.amazonaws.com/bucket-name/key-name%3FversionId%3Dversion-id}
```

Amazon S3 사전 서명된 URL(작업 배포 중)

```
//Virtual-hosted style URL
${aws:iot:s3-presigned-url-v2:https://sample-bucket-name.s3.us-
west-2.amazonaws.com/sample-code-file.png%3FversionId%3Dversion1}

//Path-style
${aws:iot:s3-presigned-url-v2:https://s3.us-west-2.amazonaws.com/sample-bucket-
name/sample-code-file.png%3FversionId%3Dversion1}
```

Amazon S3 가상 호스팅 및 경로 스타일 객체 URLs에 대한 자세한 내용은 가상 호스팅 스타일 요청 및 경로 스타일 요청 섹션을 참조하세요.

> **Note**
>
> Amazon S3 미리 서명된 URL에 `versionId`를 추가하려는 경우 AWS SDK for Java 2.x를 지원하는 URL 인코딩을 준수해야 합니다. 자세한 내용은 Changes in parsing Amazon S3 URIs from version 1 to version 2 섹션을 참조하세요.

Amazon S3 미리 서명된 URL 자리 표시자 버전 차이

다음 목록은 Amazon S3 미리 서명된 URL 자리 표시자 `${aws:iot:s3-presigned-url-v1}`(버전 1)과 `${aws:iot:s3-presigned-url-v2}`(버전 2)의 차이점을 간략하게 설명합니다.

- Amazon S3 미리 서명된 URL 자리 표시자 `${aws:iot:s3-presigned-url-v1}`은 `version-id`를 지원하지 않습니다.
- Amazon S3 미리 서명된 URL 자리 표시자 `${aws:iot:s3-presigned-url-v1}`은 Amazon S3 URL을 인코딩되지 않은 상태로 수신합니다. Amazon S3 미리 서명된 URL 자리 표시자 `${aws:iot:s3-presigned-url-v2}`을 사용하려면 Amazon S3 SDK 표준을 준수하도록 Amazon S3 URL을 인코딩해야 합니다.

주제

- 를 사용하여 작업 생성 및 관리 AWS Management Console
- 를 사용하여 작업 생성 및 관리 AWS CLI

를 사용하여 작업 생성 및 관리 AWS Management Console

이 섹션에서는 AWS IoT 콘솔에서 작업을 생성하고 관리하는 방법을 설명합니다. 작업을 생성한 후 세부 정보 페이지에서 작업에 대한 정보를 보고 작업을 관리할 수 있습니다.

> **Note**
>
> AWS IoT 작업에 대한 코드 서명을 수행하려면를 사용합니다 AWS CLI. 자세한 내용은 Create and manage jobs by using the AWS CLI 섹션을 참조하세요.

주제

- 를 사용하여 관리 작업 생성 AWS Management Console
- 를 사용하여 작업 보기 및 관리 AWS Management Console

를 사용하여 관리 작업 생성 AWS Management Console

작업을 생성하려면 AWS IoT 콘솔에 로그인하고 원격 작업 섹션의 작업 허브로 이동합니다. 그런 다음, 다음 단계를 수행합니다.

1. 작업 페이지의 작업 대화 상자에서 작업 생성을 선택합니다.
2. 사용 중인 디바이스에 따라 사용자 지정 작업, FreeRTOS OTA 업데이트 작업 또는 AWS IoT Greengrass 작업을 생성할 수 있습니다. 이 예에서는 사용자 정의 작업 생성(Create a custom job)을 선택합니다. 다음을 선택합니다.
3. Custom job properties(사용자 지정 작업 속성) 페이지의 Job properties(작업 속성) 대화 상자에서 다음 필드에 정보를 입력합니다.

   - Name(이름): 고유한 영숫자 작업 이름을 입력합니다.
   - Description - optional(설명 - 선택 사항): 작업에 대한 선택적 설명을 입력합니다.
   - Tags - optional(태그 -선택 사항):

   > **Note**
   >
   > 작업 ID 및 설명에 개인 식별 정보를 사용하지 않는 것이 좋습니다.

다음을 선택합니다.

4. 파일 구성 페이지의 작업 대상 대화 상자에서 이 작업을 실행할 사물 또는 사물 그룹 선택합니다.

   Job document(작업 문서) 대화 상자에서 다음 옵션 중 하나를 선택합니다.

   - From file(파일에서): 이전에 Amazon S3 버킷에 업로드한 JSON 작업 파일
     - 코드 서명

       Amazon S3 URL에 있는 작업 문서에서 `${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}`는 코드 서명 프로필을 사용하여 서명된 코드 파일 경로로 대체되기 전까지는 자리 표시자로 필요합니다. 새로 서명된 코드 파일은 처음에 Amazon S3 소스 버킷의 `SignedImages` 폴더에 표시됩니다. `Codesigned_` 접두사가 포함된 새 작업 문서가 생성되며, 서명된 코드 파일 경로가 코드 서명 자리 표시자를 대체하고 새 작업을 생성할 수 있도록 Amazon S3 URL에 배치됩니다.

     - 사전 서명 리소스 URL

       사전 서명 역할 드롭다운에서 미리 서명된 URL에서 생성한 IAM 역할을 선택합니다. Amazon S3에 있는 객체의 URL을 미리 서명하는 데 `${aws:iot:s3-presigned-url:`을 사용하는 것은 Amazon S3에서 객체를 다운로드하는 디바이스의 보안 모범 사례입니다.

       코드 서명 자리 표시자로 사전 서명된 URL을 사용하려면 다음 예시 템플릿을 사용하세요.

       ```
       ${aws:iot:s3-presigned-url:${aws:iot:code-sign-signature:<S3 URL>}
       ```

   - From template(템플릿에서): 작업 문서와 작업 구성이 포함된 작업 템플릿. 작업 템플릿은 생성한 사용자 지정 작업 템플릿 또는 AWS 관리형 템플릿일 수 있습니다.

     디바이스 재부팅과 같이 자주 사용되는 원격 작업을 수행하기 위한 작업을 생성하는 경우 AWS 관리형 템플릿을 사용할 수 있습니다. 이러한 템플릿은 이미 사용하도록 미리 구성되어 있습니다. 자세한 내용은 사용자 정의 작업 템플릿 생성 및 관리형 템플릿에서 사용자 정의 작업 템플릿 생성 단원을 참조하세요.

5. Job configuration(작업 구성) 페이지의 Job configuration(작업 구성) 대화 상자에서 다음 작업 유형 중 하나를 선택합니다.

   - 스냅샷 작업: 스냅샷 작업은 대상 디바이스 및 그룹에서 실행이 완료되면 완료됩니다.
   - 연속 작업: 연속 작업은 사물 그룹에 적용되며 이후에 지정된 대상 그룹에 추가하는 모든 디바이스에서 실행됩니다.

6. Additional configurations - optional(추가 구성 - 선택 사항) 대화 상자에서 다음과 같은 선택적 작업 구성을 검토하고 적절히 선택합니다.

   - Rollout configuration(롤아웃 구성)
   - Scheduling configuration(예약 구성)
   - Job executions timeout configuration(작업 실행 제한 시간 구성)
   - Job executions retry configuration - new(작업 실행 재시도 구성 - 신규)
   - Abort configuration(중단 구성)

   작업 구성에 대한 추가 정보는 다음 섹션을 참조하세요.

   - 작업 롤아웃, 예약 및 중단 구성
   - 작업 실행 제한 시간 및 재시도 구성

   모든 작업 선택 사항을 검토한 다음 Submit(제출)을 선택하여 작업을 생성합니다.

를 사용하여 작업 보기 및 관리 AWS Management Console

작업을 생성하면 콘솔에서는 JSON 서명을 생성해 작업 문서에 배치합니다. AWS IoT 콘솔을 사용하여 상태를 보거나 작업을 취소 또는 삭제할 수 있습니다.

생성한 작업을 선택하면 다음을 찾을 수 있습니다.

- 작업 이름, 설명, 유형, 작업 생성 시간, 마지막 업데이트 시간, 예상 시작 시간과 같은 일반 작업 세부 정보입니다.
- 지정한 모든 작업 구성 및 해당 상태입니다.
- 작업 문서입니다.
- 지정한 작업 실행 및 선택적 태그입니다.

작업을 관리하려면 콘솔의 작업 허브로 이동하여 작업을 편집, 삭제 또는 취소할지 여부를 선택합니다.

를 사용하여 작업 생성 및 관리 AWS CLI

이 단원에서는 작업을 생성하고 관리하는 방법을 설명합니다.

작업 생성

AWS IoT 작업을 생성하려면 CreateJob 명령을 사용합니다. 생성된 작업은 지정한 대상(사물 또는 사물 그룹)에서 실행될 때까지 대기합니다. AWS IoT 작업을 생성하려면 요청 본문에 포함하거나 Amazon S3 문서에 대한 링크로 포함할 수 있는 작업 문서가 필요합니다. 작업에 미리 서명된 Amazon S3 URLs을 사용하여 파일 다운로드가 포함된 경우 파일을 다운로드할 권한이 있고 AWS IoT 작업 서비스에 역할을 수임할 권한을 부여하는 IAM 역할 Amazon 리소스 이름(ARN)이 필요합니다.

API 명령 또는를 사용하여 날짜 및 시간을 입력할 때 구문에 대한 자세한 내용은 타임스탬프를 AWS CLI참조하세요.

작업을 사용한 코드 서명

에 코드 서명을 사용하는 경우 코드 서명 작업을 시작하고 작업 문서에 출력을 포함해야 AWS IoT합니다. 이렇게 하면 코드 서명 프로필을 사용하여 서명된 코드 파일 경로로 대체되기 전까지 자리 표시자로 필요한 작업 문서의 코드 서명 자리 표시자가 대체됩니다. 코드 서명 자리 표시자는 다음과 같이 표시됩니다.

```
    ${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}
```

start-signing-job 명령을 사용하여 코드 서명 작업을 생성합니다. `start-signing-job`은 작업 ID를 반환합니다. 서명이 저장되는 Amazon S3 위치를 가져오려면 describe-signing-job 명령을 사용합니다. 그런 다음 Amazon S3에서 서명을 다운로드할 수 있습니다. 코드 서명 작업에 대한 자세한 내용은 AWS IoT의 코드 서명 단원을 참조하세요.

작업 문서에서는 start-signing-job 명령을 사용하여 Amazon S3 버킷에 배치된 JSON 서명 출력과 코드 파일에 대한 사전 서명된 URL 자리 표시자가 포함되어 있어야 합니다.

```
{
    "presign": "${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/bucket/
image}",
}
```

작업 문서로 작업 생성

다음 명령은 Amazon S3 버킷(*jobBucket*)에 저장된 작업 문서(*job-document.json*)와 Amazon S3에서 파일을 다운로드할 수 있는 권한이 있는 역할(*S3DownloadRole*)을 사용하여 작업을 생성하는 방법을 보여줍니다.

```
aws iot create-job  \
```

```
      --job-id 010  \
      --targets arn:aws:iot:us-east-1:123456789012:thing/thingOne  \
      --document-source https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json
 \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute
\": 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings
\": 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

작업은 *thingOne*에서 실행됩니다.

선택 사항인 `timeout-config` 파라미터는 각 디바이스가 작업 실행을 완료한 시간을 지정합니다. 타이머는 작업 실행 상태가 `IN_PROGRESS`에 설정되면 시작합니다. 시간이 만료되기 전에 작업 실행 상태가 다른 터미널 상태로 설정되어 있지 않으면 `TIMED_OUT`으로 자동으로 설정됩니다.

진행 중 타이머는 업데이트될 수 없으며 이 작업에 대한 모든 작업 실행에 적용됩니다. 작업 실행이이 간격보다 오래 `IN_PROGRESS` 상태로 유지될 때마다 실패하고 터미널 `TIMED_OUT` 상태로 전환됩니다. AWS IoT 또한는 MQTT 알림을 게시합니다.

작업 롤아웃 및 중단에 대한 구성 생성에 대한 자세한 내용은 작업 롤아웃 및 중단 구성을 참조하세요.

> **Note**
>
> Amazon S3 파일로 지정되는 작업 문서는 작업을 생성할 때 가져오게 됩니다. 작업 문서를 생성한 후 작업 문서의 원본으로 사용한 Amazon S3 파일의 내용을 변경해도 작업 대상으로 전송되는 내용은 변경되지 않습니다.

작업 업데이트

작업은 업데이트하려면 UpdateJob 명령을 사용합니다. 작업의 `description`, `presignedUrlConfig`, `jobExecutionsRolloutConfig`, `abortConfig` 및 `timeoutConfig` 필드를 업데이트할 수 있습니다.

```
aws iot update-job  \
```

```
  --job-id 010  \
  --description "updated description" \
  --timeout-config inProgressTimeoutInMinutes=100 \
  --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\": 50,
 \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\": 1000,
 \"numberOfSucceededThings\": 1000}, \"maximumPerMinute\": 1000}}" \
  --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
  --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

자세한 내용은 작업 롤아웃 및 중단 구성을 참조하세요.

작업 취소

작업을 취소하려면 CancelJob 명령을 사용합니다. 작업을 취소하면 작업에 대한 새 작업 실행 AWS IoT 롤아웃이 중지됩니다. 또한 QUEUED 상태인 모든 작업 실행을 취소합니다. 디바이스가 이미 작업을 완료했기 때문에 터미널 상태의 모든 작업 실행을 그대로 AWS IoT 유지합니다. 작업 실행이 IN_PROGRESS 상태인 경우에도 사용자가 선택적 --force 파라미터를 사용하지 않는 한 IoT가 해당 작업 실행을 그대로 둡니다.

다음 명령은 ID가 010인 작업을 취소하는 방법을 보여줍니다.

```
aws iot cancel-job --job-id 010
```

이 명령은 다음 출력을 표시합니다.

```
{
    "jobArn": "string",
    "jobId": "string",
    "description": "string"
}
```

작업을 취소하면 QUEUED 상태인 작업 실행이 취소됩니다. 선택적 --force 파라미터를 지정하는 경우에만 IN_PROGRESS 상태의 작업 실행이 취소됩니다. 종료 상태인 작업 실행은 취소되지 않습니다.

> **Warning**
>
> (`--force` 파라미터를 설정하여) `IN_PROGRESS` 상태의 작업을 취소하면 진행 중인 모든 작업 실행이 취소되고 해당 작업을 실행하고 있는 디바이스가 작업의 실행 상태를 업데이트할 수 없게 됩니다. 주의를 기울여 취소된 작업을 실행하는 각 디바이스가 유효한 상태로 복구될 수 있는지 확인하세요.

취소된 작업 또는 작업 실행 중 하나의 상태는 최종적으로 일관됩니다.는 해당 작업에 대한 새 작업 실행 및 `QUEUED` 작업 실행 예약을 가능한 한 빨리 디바이스로 AWS IoT 중지합니다. 작업 실행 상태가 `CANCELED`로 바뀌려면 디바이스 수와 기타 요인에 따라 시간이 걸릴 수 있습니다.

`AbortConfig` 객체로 정의한 기준을 충족했기 때문에 작업이 취소된 경우 서비스에서는 `comment` 및 `reasonCode` 필드에 대해 자동으로 채워지는 값을 추가합니다. 작업을 사용자가 취소한 경우 `reasonCode`에 대한 값을 직접 생성할 수 있습니다.

작업 실행 취소

디바이스의 작업 실행을 취소하려면 CancelJobExecution 명령을 사용합니다. 이 명령은 `QUEUED` 상태인 작업 실행을 취소합니다. 진행 중인 작업 실행을 취소하려면 `--force` 파라미터를 사용해야 합니다.

다음 명령은 `myThing`에서 실행 중인 작업 `010`에서 작업 실행을 취소하는 방법을 보여줍니다.

```
aws iot cancel-job-execution --job-id 010 --thing-name myThing
```

명령이 아무런 출력도 표시하지 않습니다.

`QUEUED` 상태인 작업 실행이 취소됩니다. 선택적 `--force` 파라미터를 지정하는 경우에만 `IN_PROGRESS` 상태의 작업 실행이 취소됩니다. 최종 상태인 작업 실행은 취소할 수 없습니다.

> **Warning**
>
> `IN_PROGRESS` 상태인 작업 실행을 취소하면 디바이스가 그 작업 실행의 상태를 업데이트할 수 없습니다. 주의를 기울여 디바이스가 유효한 상태로 복구될 수 있는지 확인하세요.

작업 실행이 최종 상태이거나 작업 실행이 `IN_PROGRESS` 상태이고 `--force` 파라미터를 `true`로 설정하지 않은 경우 이 명령에서 `InvalidStateTransitionException`이 발생합니다.

취소된 작업 실행의 상태가 최종적으로 일관됩니다. 작업 실행 상태가 CANCELED로 바뀌려면 기타 요인에 따라 시간이 걸릴 수 있습니다.

작업 삭제

작업과 해당 작업 실행을 삭제하려면 DeleteJob 명령을 사용합니다. 기본적으로 최종 상태(SUCCEEDED 또는 CANCELED)인 작업만 삭제할 수 있습니다. 그렇지 않으면 예외가 발생합니다. force 파라미터가 true로 설정되어 있는 경우에만 IN_PROGRESS 상태인 작업을 삭제할 수 있습니다.

작업을 삭제하려면 다음 명령을 실행합니다.

```
aws iot delete-job --job-id 010 --force|--no-force
```

명령이 아무런 출력도 표시하지 않습니다.

> **Warning**
>
> IN_PROGRESS 상태인 작업을 삭제하면 해당 작업을 배포 중인 디바이스에서 작업 정보에 액세스할 수 없거나 작업 실행 상태를 업데이트할 수 없습니다. 주의를 기울여 삭제된 작업을 배포하는 각 디바이스가 유효한 상태로 복구될 수 있는지 확인하세요.

작업을 삭제하려면 해당 작업에 대해 생성된 작업 실행의 수와 기타 요인에 따라 시간이 걸릴 수 있습니다. 작업이 삭제되는 동안 그 작업의 상태는 DELETION_IN_PROGRESS로 표시됩니다. 이미 DELETION_IN_PROGRESS 상태의 작업을 삭제하거나 취소하려고 하면 오류가 발생합니다.

동시에 10개 작업만 DELETION_IN_PROGRESS 상태일 수 있습니다. 그렇지 않으면 LimitExceededException이 발생합니다.

작업 문서 가져오기

작업에 대한 작업 문서를 검색하려면 GetJobDocument 명령을 사용합니다. 이 작업 문서는 원격 작업의 설명이므로 디바이스에서 실행됩니다.

작업 문서를 가져오려면 다음 명령을 실행합니다.

```
aws iot get-job-document --job-id 010
```

위의 명령은 아래와 같이 지정한 작업의 작업 문서를 반환합니다.

```
{
    "document": "{\n\t\"operation\":\"install\",\n\t\"url\":\"http://amazon.com/
firmWareUpate-01\",\n\t\"data\":\"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
amzn-s3-demo-bucket/datafile}\"\n}"
}
```

> **Note**
>
> 이 명령을 사용하여 작업 문서를 가져올 때는 미리 서명된 Amazon S3 URL이 자리 표시자 URL을 대체하지 않습니다. 디바이스가 GetPendingJobExecutions API 작업을 호출하면 자리 표시자 URL이 작업 문서에서 미리 서명된 Amazon S3 URL로 대체됩니다.

작업 나열

의 모든 작업 목록을 가져오려면 ListJobs 명령을 AWS 계정사용합니다. 작업 데이터 및 작업 실행 데이터는 제한된 시간동안 보존됩니다. 다음 명령을 실행하여의 모든 작업을 나열합니다. AWS 계정

```
aws iot list-jobs
```

위 명령은 아래와 같이 계정의 모든 작업을 작업 상태에 따라 정렬하여 반환합니다.

```
{
    "jobs": [
        {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486687079.743,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/013",
            "createdAt": 1486687079.743,
            "targetSelection": "SNAPSHOT",
            "jobId": "013"
        },
        {
            "status": "SUCCEEDED",
            "lastUpdatedAt": 1486685868.444,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/012",
            "createdAt": 1486685868.444,
            "completedAt": 148668789.690,
            "targetSelection": "SNAPSHOT",
            "jobId": "012"
        },
```

```
        {
            "status": "CANCELED",
            "lastUpdatedAt": 1486678850.575,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/011",
            "createdAt": 1486678850.575,
            "targetSelection": "SNAPSHOT",
            "jobId": "011"
        }
    ]
}
```

작업 설명

작업의 상태를 가져오려면 DescribeJob 명령을 실행합니다. 다음 명령은 작업을 설명하는 방법을 보여줍니다.

```
$ aws iot describe-job --job-id 010
```

위의 명령은 특정 작업의 상태를 반환합니다. 다음 예를 참조하세요.

```
{
    "documentSource": "https://s3.amazonaws.com/amzn-s3-demo-bucket/job-
document.json",
    "job": {
        "status": "IN_PROGRESS",
        "jobArn": "arn:aws:iot:us-east-1:123456789012:job/010",
        "targets": [
            "arn:aws:iot:us-east-1:123456789012:thing/myThing"
        ],
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfFailedThings": 0,
            "numberOfInProgressThings": 0,
            "numberOfQueuedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfTimedOutThings": 0,
            "processingTargets": [
                arn:aws:iot:us-east-1:123456789012:thing/thingOne,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupOne,
                arn:aws:iot:us-east-1:123456789012:thing/thingTwo,
```

```
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupTwo
            ]
        },
        "presignedUrlConfig": {
            "expiresInSec": 60,
            "roleArn": "arn:aws:iam::123456789012:role/S3DownloadRole"
        },
        "jobId": "010",
        "lastUpdatedAt": 1486593195.006,
        "createdAt": 1486593195.006,
        "targetSelection": "SNAPSHOT",
        "jobExecutionsRolloutConfig": {
            "exponentialRate": {
                "baseRatePerMinute": integer,
                "incrementFactor": integer,
                "rateIncreaseCriteria": {
                    "numberOfNotifiedThings": integer, // Set one or the other
                    "numberOfSucceededThings": integer // of these two values.
                },
            "maximumPerMinute": integer
         }
        },
        "abortConfig": {
            "criteriaList": [
                {
                    "action": "string",
                    "failureType": "string",
                    "minNumberOfExecutedThings": integer,
                    "thresholdPercentage": integer
                }
            ]
        },
        "timeoutConfig": {
           "inProgressTimeoutInMinutes": number
          }
    }
}
```

작업 실행 목록 조회

특정 디바이스에서 실행되는 작업이 작업 실행 객체로 표현됩니다. 임의 작업에 대한 모든 작업 실행을 나열할 때는 ListJobExecutionsForJob 명령을 실행합니다. 다음 예에서는 작업에 대한 실행을 나열하는 방법을 보여 줍니다.

```
aws iot list-job-executions-for-job --job-id 010
```

위의 명령은 아래와 같이 작업 실행 목록을 반환합니다.

```
{
    "executionSummaries": [
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 1234567890
        }
    },
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingTwo",
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486593345.659,
            "queuedAt": 1486593196.378,
            "startedAt": 1486593345.659,
            "executionNumber": 4567890123
        }
    }
    ]
}
```

사물에 대한 작업 실행 목록 조회

임의 사물에 대한 모든 작업 실행을 나열할 때는 ListJobExecutionsForThing 명령을 실행합니다. 다음 예에서는 사물에 대한 작업 실행을 나열하는 방법을 보여 줍니다.

```
aws iot list-job-executions-for-thing --thing-name thingOne
```

위의 명령은 아래와 같이 지정된 사물에서 실행 중이거나 실행된 작업 실행 목록을 반환합니다.

```
{
    "executionSummaries": [
    {
        "jobExecutionSummary": {
            "status": "QUEUED",
```

```
            "lastUpdatedAt": 1486687082.071,
            "queuedAt": 1486687082.071,
            "executionNumber": 9876543210
        },
        "jobId": "013"
    },
    {
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "startAt": 1486685870.729,
            "lastUpdatedAt": 1486685870.729,
            "queuedAt": 1486685870.729,
            "executionNumber": 1357924680
        },
        "jobId": "012"
    },
    {
        "jobExecutionSummary": {
            "status": "SUCCEEDED",
            "startAt": 1486678853.415,
            "lastUpdatedAt": 1486678853.415,
            "queuedAt": 1486678853.415,
            "executionNumber": 4357680912
        },
        "jobId": "011"
    },
    {
        "jobExecutionSummary": {
            "status": "CANCELED",
            "startAt": 1486593196.378,
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 2143174250
        },
        "jobId": "010"
    }
    ]
}
```

작업 실행 설명

작업 실행의 상태를 가져올 때는 DescribeJobExecution 명령을 실행합니다. 원하는 작업 실행을 찾으려면 작업 ID와 사물 이름, 그리고 필요한 경우 실행 번호를 지정해야 합니다. 다음은 작업 실행을 설명하는 방법을 보여줍니다.

```
aws iot describe-job-execution --job-id 017 --thing-name thingOne
```

위의 명령은 JobExecution을 반환합니다. 다음 예를 참조하세요.

```
{
    "execution": {
        "jobId": "017",
        "executionNumber": 4516820379,
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "versionNumber": 123,
        "createdAt": 1489084805.285,
        "lastUpdatedAt": 1489086279.937,
        "startedAt": 1489086279.937,
        "status": "IN_PROGRESS",
        "approximateSecondsBeforeTimedOut": 100,
        "statusDetails": {
            "status": "IN_PROGRESS",
            "detailsMap": {
                "percentComplete": "10"
            }
        }
    }
}
```

작업 실행 삭제

작업 실행을 삭제하려면 DeleteJobExecution 명령을 실행합니다. 작업 실행을 찾으려면 작업 ID, 사물 이름 및 실행 번호를 지정해야 합니다. 다음은 작업 실행을 삭제하는 방법을 보여줍니다.

```
aws iot delete-job-execution --job-id 017 --thing-name thingOne --execution-number
 1234567890 --force|--no-force
```

명령이 아무런 출력도 표시하지 않습니다.

기본적으로 작업 실행의 상태는 QUEUED 또는 터미널 상태(SUCCEEDED, FAILED, REJECTED, TIMED_OUT, REMOVED 또는 CANCELED)여야 합니다. 그렇지 않으면 오류가 발생합니다. IN_PROGRESS 상태인 작업을 삭제하기 위해 force 파라미터를 true로 설정할 수 있습니다.

> **Warning**
>
> IN_PROGRESS 상태인 작업 실행을 삭제하면 해당 작업을 실행 중인 디바이스에서 작업 정보에 액세스할 수 없거나 작업 실행 상태를 업데이트할 수 없습니다. 주의를 기울여 디바이스가 유효한 상태로 복구될 수 있는지 확인하세요.

작업 템플릿

작업 템플릿을 사용하여 여러 대상 디바이스 집합에 배포할 수 있는 작업을 미리 구성합니다. 재부팅 또는 애플리케이션 설치와 같이 자주 수행되는 원격 작업을 디바이스에 배포하기 위해 템플릿을 사용하여 표준 구성을 정의할 수 있습니다. 보안 패치 및 버그 수정 배포와 같은 작업을 수행하려는 경우 기존 작업에서 템플릿을 생성할 수 있습니다.

작업 템플릿을 생성할 때 다음 추가 구성과 리소스를 지정합니다.

- 작업 속성
- 작업 문서 및 대상
- 롤아웃, 예약 및 취소 기준
- 시간 초과 및 재시도 기준

사용자 지정 및 AWS 관리형 템플릿

수행하려는 원격 작업에 따라 사용자 지정 작업 템플릿을 생성하거나 AWS 관리형 템플릿을 사용할 수 있습니다. 사용자 지정 작업 템플릿을 사용하여 사용자 지정 작업 문서를 제공하고 디바이스에 배포할 재사용 가능한 작업을 생성합니다. AWS 관리형 템플릿은 일반적으로 수행되는 작업을 위해 AWS IoT 작업에서 제공하는 작업 템플릿입니다. 이러한 템플릿에는 일부 원격 작업에 대해 미리 정의된 작업 문서가 있으므로 고유한 작업 문서를 생성할 필요가 없습니다. 관리형 템플릿을 사용하면 재사용 가능한 작업을 생성하여 디바이스에서 더 빠르게 시작할 수 있습니다.

주제

- AWS 관리형 템플릿을 사용하여 일반적인 원격 작업 배포
- 사용자 정의 작업 템플릿 생성

AWS 관리형 템플릿을 사용하여 일반적인 원격 작업 배포

AWS 관리형 템플릿은에서 제공하는 작업 템플릿입니다 AWS. 재부팅, 파일 다운로드, 디바이스에 애플리케이션 설치와 같이 자주 수행되는 원격 작업에 사용됩니다. 이러한 템플릿에는 각 원격 작업에 대해 미리 정의된 작업 문서가 있으므로 고유한 작업 문서를 생성할 필요가 없습니다.

미리 정의된 구성 집합에서 선택하고 추가 코드를 작성하지 않고도 이러한 템플릿을 사용하여 작업을 생성할 수 있습니다. 관리형 템플릿을 사용하여 플릿에 배포된 작업 문서를 볼 수 있습니다. 이러한 템플릿으로 작업을 생성하고 원격 작업에 다시 사용할 수 있는 사용자 정의 작업 템플릿을 생성할 수 있습니다.

관리형 템플릿에는 무엇이 포함되어 있나요?

각 AWS 관리형 템플릿에는 다음이 포함됩니다.

- 작업 문서에서 명령을 실행할 환경.
- 작업 이름과 해당 파라미터를 지정하는 작업 문서입니다. 예를 들어 파일 다운로드(Download file) 템플릿을 사용하는 경우 작업 이름은 파일 다운로드(Download file)이고 파라미터는 다음과 같을 수 있습니다.
  - 디바이스에 다운로드하려는 파일의 URL입니다. 이 URL은 인터넷 리소스이거나 퍼블릭 또는 사전 서명된 Amazon Simple Storage Service(S3) URL일 수 있습니다.
  - 다운로드한 파일을 저장할 디바이스의 로컬 파일 경로입니다.

  작업 문서와 해당 파라미터에 대한 자세한 내용은 관리형 템플릿 원격 작업 및 작업 문서 섹션을 참조하세요.

사전 조건

디바이스에서 관리형 템플릿 작업 문서에 지정된 원격 작업을 디바이스에서 실행하려면 다음을 수행해야 합니다.

- 디바이스에 특정 소프트웨어 설치

  자체 디바이스 소프트웨어 및 작업 핸들러 또는 AWS IoT Device Client를 사용합니다. 비즈니스 사례에 따라 서로 다른 기능을 수행하도록 둘 다 실행할 수도 있습니다.
  - 자체 디바이스 소프트웨어 및 작업 핸들러 사용

AWS IoT Device SDK 와 원격 작업을 지원하는 핸들러 라이브러리를 사용하여 디바이스에 대한 고유한 코드를 작성할 수 있습니다. 작업을 배포하고 실행하려면 디바이스 에이전트 라이브러리가 올바르게 설치되었고 이러한 디바이스에서 실행 중인지 확인합니다.

원격 작업을 지원하는 자체 핸들러를 사용하도록 선택할 수도 있습니다. 자세한 내용은 AWS IoT Device Client GitHub 리포지토리의 샘플 작업 핸들러를 참조하세요.

- AWS IoT 디바이스 클라이언트 사용

  또는 기본적으로 콘솔에서 직접 모든 관리형 템플릿 사용을 지원하므로 디바이스에 AWS IoT Device Client를 설치하고 실행할 수 있습니다.

  Device Client는 임베디드 Linux 기반 IoT 디바이스에 컴파일하고 설치할 수 있는 C++로 작성된 오픈 소스 소프트웨어입니다. Device Client에는 기본 클라이언트와 개별 클라이언트 측 기능이 있습니다. 기본 클라이언트는 MQTT 프로토콜을 AWS IoT 통해와 연결을 설정하고 다양한 클라이언트 측 기능과 연결할 수 있습니다.

  디바이스에서 원격 작업을 수행하려면 Device Client의 클라이언트 측 작업 기능을 사용합니다. 이 기능에는 작업 문서를 수신하는 파서와 작업 문서에 지정된 원격 작업을 구현하는 작업 핸들러가 포함되어 있습니다. Device Client와 해당 기능에 대한 자세한 내용은 AWS IoT Device Client를 참조하세요.

  디바이스에서 실행할 때 Device Client는 작업 문서를 수신하고 문서에서 명령을 실행하는 데 사용하는 플랫폼별 구현을 갖습니다. Device Client 설정 및 작업 기능에 대한 자세한 내용은 AWS IoT 자습서를 참조하세요.

- 지원되는 환경 사용

  각 관리형 템플릿에 대해 원격 작업을 실행하는 데 사용할 수 있는 환경에 대한 정보를 찾을 수 있습니다. 템플릿에 지정된 대로 지원되는 Linux 환경에서 템플릿을 사용하는 것이 좋습니다. AWS IoT Device Client는 Debian 및 Ubuntu와 같은 일반적인 마이크로프로세서 및 Linux 환경을 지원하므로 관리형 템플릿 원격 작업을 실행하려면 Device Client를 사용합니다.

관리형 템플릿 원격 작업 및 작업 문서

다음 섹션에서는 AWS IoT 작업에 대한 다양한 AWS 관리형 템플릿을 나열하고 디바이스에서 수행할 수 있는 원격 작업을 설명합니다. 아래 섹션에는 작업 문서에 대한 정보와 각 원격 작업의 작업 문서 파라미터에 대한 정보가 있습니다. 디바이스 측 소프트웨어는 템플릿 이름과 파라미터를 사용하여 원격 작업을 수행합니다.

AWS 관리형 템플릿은 템플릿을 사용하여 작업을 생성할 때 값을 지정하는 입력 파라미터를 수락합니다. 모든 관리형 템플릿에는 `runAsUser` 및 `pathToHandler`의 두 가지 선택적 파라미터가 공통적으로 있습니다. `AWS-Reboot` 템플릿을 제외한 나머지 템플릿을 사용하려면 템플릿을 사용하여 작업을 생성할 때 값을 지정해야 하는 추가 입력 파라미터가 필요합니다. 이 필수 입력 파라미터는 사용자가 선택하는 템플릿에 따라 달라집니다. 예를 들어, `AWS-Download-File` 템플릿을 선택하는 경우 설치할 패키지 목록 및 파일을 다운로드할 URL을 지정해야 합니다.

AWS IoT 콘솔 또는 AWS Command Line Interface (AWS CLI)를 사용하여 관리형 템플릿을 사용하는 작업을 생성할 때 입력 파라미터의 값을 지정합니다. CLI를 사용할 때는 `document-parameters` 객체를 사용하여 이러한 값을 제공합니다. 자세한 내용은 documentParameters를 참조하세요.

> **Note**
>
> AWS 관리형 템플릿에서 작업을 생성할 때만 `document-parameters`를 사용합니다. 이 파라미터는 사용자 지정 작업 템플릿과 함께 사용하거나 이러한 템플릿에서 작업을 생성하는 데 사용할 수 없습니다.

다음은 일반적인 선택적 입력 파라미터에 대한 설명을 보여줍니다. 다음 섹션에서 각 관리형 템플릿에 필요한 다른 입력 파라미터에 대한 설명을 확인할 수 있습니다.

`runAsUser`

이 파라미터는 작업 핸들러를 다른 사용자로 실행할지 여부를 지정합니다. 작업 생성 중 지정하지 않으면 작업 핸들러는 Device Client와 동일한 사용자로 실행됩니다. 작업 핸들러를 다른 사용자로 실행할 때 256자 이하의 문자열 값을 지정합니다.

`pathToHandler`

디바이스에서 실행 중인 작업 핸들러의 경로입니다. 작업 생성 중 지정하지 않으면 Device Client는 현재 작업 디렉터리를 사용합니다.

다음은 다양한 원격 작업, 작업 문서 및 수락하는 파라미터를 보여줍니다. 이러한 모든 템플릿은 디바이스에서 원격 작업을 실행하기 위한 Linux 환경을 지원합니다.

AWS–Download–File

템플릿 이름

`AWS-Download-File`

템플릿 설명

파일을 다운로드하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터가 있습니다. 선택적 파라미터 `runAsUser` 및 `pathToHandler`를 지정할 수도 있습니다.

`downloadUrl`

파일을 다운로드할 URL입니다. 이 URL은 인터넷 리소스, 공개적으로 액세스할 수 있는 Amazon S3의 객체 또는 미리 서명된 URL을 사용하여 디바이스에서만 액세스할 수 있는 Amazon S3의 객체일 수 있습니다. 미리 서명된 URL 사용 및 권한 부여에 대한 자세한 내용은 미리 서명된 URL 섹션을 참조하세요.

`filePath`

다운로드한 파일을 저장할 디바이스의 위치를 보여주는 로컬 파일 경로입니다.

디바이스 동작

디바이스는 지정된 위치에서 파일을 다운로드하고 다운로드가 완료되었는지 확인한 후 로컬에 저장합니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 파일을 다운로드하기 위해 실행해야 하는 셸 스크립트(`download-file.sh`)를 보여줍니다. 또한 필수 파라미터 `downloadUrl` 및 `filePath`를 보여줍니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Download-File",
        "type": "runHandler",
        "input": {
          "handler": "download-file.sh",
```

```
          "args": [
            "${aws:iot:parameter:downloadUrl}",
            "${aws:iot:parameter:filePath}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Install–Application

템플릿 이름

AWS-Install-Application

템플릿 설명

하나 이상의 애플리케이션을 설치하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터 packages가 있습니다. 선택적 파라미터 runAsUser 및 pathToHandler를 지정할 수도 있습니다.

packages

설치할 하나 이상의 애플리케이션의 목록(공백으로 구분)입니다.

디바이스 동작

디바이스가 작업 문서에 지정된 대로 애플리케이션을 설치합니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 파일을 다운로드하기 위해 실행해야 하는 셸 스크립트(install-packages.sh)를 보여줍니다. 또한 필수 파라미터 packages를 보여줍니다.

```
{
  "version": "1.0",
```

```
  "steps": [
    {
      "action": {
        "name": "Install-Application",
        "type": "runHandler",
        "input": {
          "handler": "install-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Reboot

템플릿 이름

AWS-Reboot

템플릿 설명

디바이스 재부팅을 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 필수 파라미터가 없습니다. 선택적 파라미터 runAsUser 및 pathToHandler를 지정할 수 있습니다.

디바이스 동작

디바이스가 성공적으로 재부팅됩니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 디바이스를 재부팅하기 위해 실행해야 하는 셸 스크립트(reboot.sh)를 보여줍니다.

```
{
```

```
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Remove–Application

템플릿 이름

AWS-Remove-Application

템플릿 설명

하나 이상의 애플리케이션을 제거하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터 packages가 있습니다. 선택적 파라미터 runAsUser 및 pathToHandler를 지정할 수도 있습니다.

packages

제거할 하나 이상의 애플리케이션의 목록(공백으로 구분)입니다.

디바이스 동작

디바이스가 작업 문서에 지정된 대로 애플리케이션을 제거합니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 파일을 다운로드하기 위해 실행해야 하는 셸 스크립트(remove-packages.sh)를 보여줍니다. 또한 필수 파라미터 packages를 보여줍니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Remove-Application",
        "type": "runHandler",
        "input": {
          "handler": "remove-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Restart–Application

템플릿 이름

AWS-Restart-Application

템플릿 설명

하나 이상의 서비스를 중지하고 다시 시작하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터 services가 있습니다. 선택적 파라미터 runAsUser 및 pathToHandler를 지정할 수도 있습니다.

서비스

다시 시작할 하나 이상의 애플리케이션의 목록(공백으로 구분)입니다.

디바이스 동작

지정된 애플리케이션이 중지되었다가 디바이스에서 다시 시작됩니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 시스템 서비스를 다시 시작하기 위해 실행해야 하는 셸 스크립트(`restart-services.sh`)를 보여줍니다. 또한 필수 파라미터 `services`를 보여줍니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Restart-Application",
        "type": "runHandler",
        "input": {
          "handler": "restart-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Start–Application

템플릿 이름

`AWS-Start-Application`

템플릿 설명

하나 이상의 서비스를 시작하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터 `services`가 있습니다. 선택적 파라미터 `runAsUser` 및 `pathToHandler`를 지정할 수도 있습니다.

`services`

시작할 하나 이상의 애플리케이션의 목록(공백으로 구분)입니다.

디바이스 동작

지정된 애플리케이션이 디바이스에서 실행을 시작합니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 시스템 서비스를 시작하기 위해 실행해야 하는 셸 스크립트(`start-services.sh`)를 보여줍니다. 또한 필수 파라미터 `services`를 보여줍니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Start-Application",
        "type": "runHandler",
        "input": {
          "handler": "start-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Stop–Application

템플릿 이름

`AWS-Stop-Application`

템플릿 설명

하나 이상의 서비스를 중지하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터 services가 있습니다. 선택적 파라미터 runAsUser 및 pathToHandler를 지정할 수도 있습니다.

services

중지할 하나 이상의 애플리케이션의 목록(공백으로 구분)입니다.

디바이스 동작

지정된 애플리케이션이 디바이스에서 실행을 중지합니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 시스템 서비스를 중지하기 위해 실행해야 하는 셸 스크립트(stop-services.sh)를 보여줍니다. 또한 필수 파라미터 services를 보여줍니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Stop-Application",
        "type": "runHandler",
        "input": {
          "handler": "stop-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Run–Command

템플릿 이름

AWS-Run-Command

템플릿 설명

셸 명령을 실행하기 AWS 위해에서 제공하는 관리형 템플릿입니다.

입력 파라미터

이 템플릿에는 다음과 같은 필수 파라미터 command가 있습니다. 선택적 파라미터 runAsUser를 지정할 수도 있습니다.

command

쉼표로 구분된 명령 문자열. 명령 자체에 포함된 모든 쉼표는 이스케이프 처리해야 합니다.

디바이스 동작

디바이스는 작업 문서에 지정된 대로 쉘 명령을 실행합니다.

작업 문서

다음은 작업 문서와 최신 버전을 보여줍니다. 템플릿에는 작업 명령의 경로와 디바이스가 실행할 사용자 제공 명령이 표시됩니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Run-Command",
        "type": "runCommand",
        "input": {
          "command": "${aws:iot:parameter:command}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

주제

- 를 사용하여 AWS 관리형 템플릿에서 작업 생성 AWS Management Console
- 를 사용하여 AWS 관리형 템플릿에서 작업 생성 AWS CLI

를 사용하여 AWS 관리형 템플릿에서 작업 생성 AWS Management Console

AWS Management Console 를 사용하여 AWS 관리형 템플릿에 대한 정보를 가져오고 이러한 템플릿을 사용하여 작업을 생성합니다. 그런 다음 생성한 작업을 사용자 정의 템플릿으로 저장할 수 있습니다.

관리형 템플릿에 대한 세부 정보 보기

AWS IoT 콘솔에서 사용할 수 있는 다양한 관리형 템플릿에 대한 정보를 가져올 수 있습니다.

1. 사용 가능한 관리형 템플릿을 보려면 AWS IoT 콘솔의 작업 템플릿 허브로 이동하여 관리형 템플릿 탭을 선택합니다.
2. 세부 정보를 보려면 관리형 템플릿을 선택합니다.

세부 정보 페이지에는 다음 정보가 포함되어 있습니다.

- 관리형 템플릿의 이름, 설명 및 Amazon 리소스 이름(ARN)입니다.
- Linux와 같이 원격 작업을 수행할 수 있는 환경입니다.
- 작업 핸들러의 경로와 디바이스에서 실행할 명령을 지정하는 JSON 작업 문서입니다. 예를 들어 다음은 AWS-Reboot 템플릿에 대한 예제 작업 문서를 보여줍니다. 템플릿은 작업 핸들러에 대한 경로와 작업 핸들러가 디바이스를 재부팅하기 위해 실행해야 하는 셸 스크립트(`reboot.sh`)를 보여줍니다.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

다양한 원격 작업의 작업 문서와 해당 파라미터에 대한 자세한 내용은 관리형 템플릿 원격 작업 및 작업 문서 섹션을 참조하세요.

- 작업 문서의 최신 버전입니다.

관리형 템플릿을 사용하여 작업 생성

AWS 관리 콘솔을 사용하여 작업을 생성하는 데 사용할 AWS 관리형 템플릿을 선택할 수 있습니다. 이 단원에서 그 방법을 설명합니다.

작업 생성 워크플로를 시작한 다음 작업을 생성하는 동안 사용할 AWS 관리형 템플릿을 선택할 수도 있습니다. 이 워크플로에 대한 자세한 내용은 를 사용하여 작업 생성 및 관리 AWS Management Console 섹션을 참조하세요.

1. AWS 관리형 템플릿 선택

   AWS IoT 콘솔의 작업 템플릿 허브로 이동하여 관리형 템플릿 탭을 선택한 다음 템플릿을 선택합니다.

2. 관리형 템플릿을 사용하여 작업 생성

   1. 템플릿의 세부 정보 페이지에서 작업 생성(Create job)을 선택합니다.

      템플릿 구성이 추가된 작업 생성(Create job) 워크플로의 사용자 정의 작업 속성(Custom job properties) 단계로 콘솔이 전환됩니다.

   2. 고유한 영숫자 작업 이름, 설명(선택 사항) 및 태그를 입력하고 다음(Next)을 선택합니다.

   3. 이 작업에서 실행할 사물 또는 사물 그룹을 작업 대상으로 선택합니다.

   4. 작업 문서(Job document) 섹션에 템플릿이 구성 설정 및 입력 파라미터와 함께 표시됩니다. 선택한 템플릿의 입력 파라미터 값을 입력합니다. 예를 들어, AWS-Download-File 템플릿을 선택한 경우:

      - downloadUrl에 다운로드할 파일의 URL을 입력합니다. 예: `https://example.com/index.html`.
      - filePath에는 다운로드한 파일을 저장할 디바이스의 경로를 입력합니다. 예: *path/to/file*.

      선택적으로 `runAsUser` 및 `pathToHandler` 파라미터에 대한 값을 입력할 수도 있습니다. 각 템플릿의 입력 파라미터에 대한 자세한 내용은 관리형 템플릿 원격 작업 및 작업 문서 섹션을 참조하세요.

5. Job configuration(작업 구성) 페이지에서 작업 유형을 연속 또는 스냅샷 작업으로 선택합니다. 스냅샷 작업은 대상 디바이스 및 그룹에서 실행이 완료되면 완료됩니다. 연속 작업은 사물 그룹에 적용되며 지정된 대상 그룹에 추가하는 모든 디바이스에서 실행됩니다.
6. 작업에 대한 추가 구성을 계속 추가한 다음 작업을 검토하고 생성합니다. 추가 구성에 대한 자세한 내용은 다음 섹션을 참조하세요.
   - 작업 롤아웃, 예약 및 중단 구성
   - 작업 실행 제한 시간 및 재시도 구성

관리형 템플릿에서 사용자 정의 작업 템플릿 생성

AWS 관리형 템플릿과 사용자 지정 작업을 시작점으로 사용하여 자체 사용자 지정 작업 템플릿을 생성할 수 있습니다. 사용자 지정 작업 템플릿을 생성하려면 먼저 이전 섹션에 설명된 대로 AWS 관리형 템플릿에서 작업을 생성합니다.

그런 다음 사용자 지정 작업을 템플릿으로 저장하여 사용자 지정 작업 템플릿을 직접 생성할 수 있습니다. 템플릿으로 저장하려면

1. AWS IoT 콘솔의 작업 허브로 이동하여 관리형 템플릿이 포함된 작업을 선택합니다.
2. 작업 템플릿으로 저장(Save as a job template)을 선택한 다음 사용자 정의 작업 템플릿을 생성합니다. 사용자 정의 작업 템플릿 생성에 대한 자세한 내용은 기존 작업에서 작업 템플릿 생성 섹션을 참조하세요.

를 사용하여 AWS 관리형 템플릿에서 작업 생성 AWS CLI

AWS CLI 를 사용하여 AWS 관리형 템플릿에 대한 정보를 가져오고 이러한 템플릿을 사용하여 작업을 생성합니다. 그런 다음 작업을 템플릿으로 저장한 후 사용자 지정 템플릿을 직접 생성할 수 있습니다.

관리형 템플릿 나열

`list-managed-job-templates` AWS CLI 명령은의 모든 작업 템플릿을 나열합니다 AWS 계정.

```
aws iot list-managed-job-templates
```

기본적으로이 명령을 실행하면 사용 가능한 모든 AWS 관리형 템플릿과 해당 세부 정보가 표시됩니다.

```
{
```

```
    "managedJobTemplates": [
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Reboot:1.0",
            "templateName": "AWS-Reboot",
            "description": "A managed job template for rebooting the device.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Remove-
Application:1.0",
            "templateName": "AWS-Remove-Application",
            "description": "A managed job template for uninstalling one or more
 applications.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Stop-Application:1.0",
            "templateName": "AWS-Stop-Application",
            "description": "A managed job template for stopping one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },

        ...

        {
            "templateArn": "arn:aws:iot:us-east-1::jobtemplate/AWS-Restart-
Application:1.0",
            "templateName": "AWS-Restart-Application",
            "description": "A managed job template for restarting one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
```

```
        }
    ]
}
```

자세한 내용은 ListManagedJobTemplates을 참조하세요.

관리형 템플릿에 대한 세부 정보 가져오기

describe-managed-job-template AWS CLI 명령은 지정된 작업 템플릿에 대한 세부 정보를 가져옵니다. 작업 템플릿 이름과 템플릿 버전(선택 사항)을 지정합니다. 템플릿 버전을 지정하지 않으면 미리 정의된 기본 버전이 반환됩니다. 다음은 명령을 실행하여 AWS-Download-File 템플릿에 대한 세부 정보를 가져오는 예를 보여줍니다.

```
aws iot describe-managed-job-template \
    --template-name AWS-Download-File
```

이 명령은 템플릿 세부 정보, ARN, 해당 작업 문서 및 템플릿의 입력 파라미터의 키-값 페어 목록인 documentParameters 파라미터를 표시합니다. 각 템플릿과 입력 파라미터에 대한 자세한 내용은 관리형 템플릿 원격 작업 및 작업 문서 섹션을 참조하세요.

> **Note**
>
> 이 API를 사용할 때 반환되는 documentParameters 객체는 AWS 관리형 템플릿에서 작업을 생성할 때만 사용해야 합니다. 사용자 지정 작업 템플릿에 객체를 사용해서는 안 됩니다. 이 파라미터를 이용하는 방법의 예는 관리형 템플릿을 사용하여 작업 생성 섹션을 참조하세요.

```
{
    "templateName": "AWS-Download-File",
    "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0",
    "description": "A managed job template for downloading a file.",
    "templateVersion": "1.0",
    "environments": [
        "LINUX"
    ],
    "documentParameters": [
        {
            "key": "downloadUrl",
            "description": "URL of file to download.",
            "regex": "(.*?)",
```

```
            "example": "http://www.example.com/index.html",
            "optional": false
        },
        {
            "key": "filePath",
            "description": "Path on the device where downloaded file is written.",
            "regex": "(.*?)",
            "example": "/path/to/file",
            "optional": false
        },
        {
            "key": "runAsUser",
            "description": "Execute handler as another user. If not specified, then
 handler is executed as the same user as device client.",
            "regex": "(.){0,256}",
            "example": "user1",
            "optional": true
        },
        {
            "key": "pathToHandler",
            "description": "Path to handler on the device. If not specified, then
 device client will use the current working directory.",
            "regex": "(.){0,4096}",
            "example": "/path/to/handler/script",
            "optional": true
        }
    ],
    "document": "{\"version\":\"1.0\",\"steps\":[{\"action\":{\"name
\":\"Download-File\",\"type\":\"runHandler\",\"input\":{\"handler\":
\"download-file.sh\",\"args\":[\"${aws:iot:parameter:downloadUrl}\",
\"${aws:iot:parameter:filePath}\"],\"path\":\"${aws:iot:parameter:pathToHandler}\"},
\"runAsUser\":\"${aws:iot:parameter:runAsUser}\"}}]}"
}
```

자세한 내용은 DescribeManagedJobTemplate을 참조하세요.

관리형 템플릿을 사용하여 작업 생성

create-job AWS CLI 명령을 사용하여 작업 템플릿에서 작업을 생성할 수 있습니다. thing0ne이라는 디바이스를 대상으로 하고, 작업의 기초로 사용할 관리형 템플릿의 Amazon 리소스 이름(ARN)을 지정합니다. create-job 명령의 연결된 파라미터를 전달하여 시간 초과 및 취소 구성과 같은 고급 구성을 재정의할 수 있습니다.

예에서는 AWS-Download-File 템플릿을 사용하는 작업을 생성하는 방법을 보여줍니다. 또한 document-parameters 파라미터를 사용하여 템플릿의 입력 파라미터를 지정하는 방법을 보여줍니다.

> **Note**
>
> AWS 관리형 템플릿에서만 document-parameters 객체를 사용합니다. 사용자 지정 작업 템플릿에 이 객체를 사용해서는 안 됩니다.

```
aws iot create-job \
   --targets arn:aws:iot:region:account-id:thing/thingOne \
   --job-id "new-managed-template-job" \
   --job-template-arn arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0 \
   --document-parameters downloadUrl=https://example.com/index.html,filePath=path/to/
file
```

여기서 각 항목은 다음과 같습니다.

- *region*은 입니다 AWS 리전.
- *account-id*은 고유한 AWS 계정 번호입니다.
- *thingOne*은 작업이 대상으로 지정된 IoT 사물의 이름입니다.
- *AWS-Download-File:1.0*은 관리형 템플릿의 이름입니다.
- https://*example.com/index*.html은 파일을 다운로드할 URL입니다.
- https://*pathto/file/index*는 다운로드한 파일을 저장할 디바이스의 로컬 파일 경로입니다.

다음 명령을 실행하여 *AWS-Download-File* 템플릿에 대한 작업을 생성합니다.

```
{
    "jobArn": "arn:aws:iot:region:account-id:job/new-managed-template-job",
    "jobId": "new-managed-template-job",
    "description": "A managed job template for downloading a file."
}
```

관리형 템플릿에서 사용자 지정 작업 템플릿 생성

1. 이전 섹션의 설명에 따라 관리형 템플릿을 사용하여 작업을 생성합니다.

2. 생성한 작업의 ARN을 사용하여 사용자 정의 작업 템플릿을 생성합니다. 자세한 내용은 기존 작업에서 작업 템플릿 생성 단원을 참조하십시오.

사용자 정의 작업 템플릿 생성

AWS CLI 및 AWS IoT 콘솔을 사용하여 작업 템플릿을 생성할 수 있습니다. , AWS CLI AWS IoT 콘솔 및 Fleet Hub for AWS IoT Device Management 웹 애플리케이션을 사용하여 작업 템플릿에서 작업을 생성할 수도 있습니다. Fleet Hub 애플리케이션에서 작업 템플릿 작업에 대한 자세한 내용은 Fleet Hub for AWS IoT Device Management에서 작업 템플릿 작업을 참조하세요.

> **Note**
>
> 작업 문서의 총 대체 패턴 수는 10개 이하여야 합니다.

주제

- 를 사용하여 사용자 지정 작업 템플릿 생성 AWS Management Console
- 를 사용하여 사용자 지정 작업 템플릿 생성 AWS CLI

를 사용하여 사용자 지정 작업 템플릿 생성 AWS Management Console

이 주제에서는 AWS IoT 콘솔을 사용하여 작업 템플릿에 대한 세부 정보를 생성, 삭제 및 보는 방법을 설명합니다.

사용자 정의 작업 템플릿 생성

원본 사용자 정의 작업 템플릿을 생성하거나 기존 작업에서 작업 템플릿을 생성할 수 있습니다. AWS 관리형 템플릿을 사용하여 생성된 기존 작업에서 사용자 지정 작업 템플릿을 생성할 수도 있습니다. 자세한 내용은 관리형 템플릿에서 사용자 정의 작업 템플릿 생성 단원을 참조하십시오.

원본 작업 템플릿 생성

1. 작업 템플릿 생성 시작

    1. AWS IoT 콘솔의 작업 템플릿 허브로 이동하여 사용자 지정 템플릿 탭을 선택합니다.
    2. 작업 템플릿 생성을 선택합니다.

> **Note**
>
> 또한 플릿 허브 아래 관련 서비스 페이지의 작업 템플릿 페이지로 이동할 수도 있습니다.

2. 작업 템플릿 속성 지정

   작업 템플릿 생성(Create job template) 페이지에서 작업 이름에 대한 영숫자 식별자와 템플릿에 대한 추가 세부 정보를 제공하는 영숫자 설명을 입력합니다.

   > **Note**
   >
   > 작업 ID 또는 설명에 개인 식별 정보를 사용하지 않는 것이 좋습니다.

3. 작업 문서 제공

   S3 버킷에 저장되거나 작업 내에 지정된 인라인 작업 문서로 저장된 JSON 작업 파일을 제공합니다. 이 템플릿을 사용하여 작업을 생성할 때 이 작업 파일이 작업 문서가 됩니다.

   작업 파일이 S3 버킷에 저장되어 있는 경우 S3 URL을 입력하거나S3 찾아보기(Browse S3)를 선택하고 작업 문서로 이동한 다음 해당 작업 파일을 선택합니다.

   > **Note**
   >
   > 현재 리전에서는 S3 버킷만 선택할 수 있습니다.

4. 작업에 대한 추가 구성을 계속 추가한 다음 작업을 검토하고 생성합니다. 추가 옵션 구성에 대한 자세한 내용은 다음 링크를 참조하세요.

   - 작업 롤아웃, 예약 및 중단 구성
   - 작업 실행 제한 시간 및 재시도 구성

기존 작업에서 작업 템플릿 생성

1. 작업 선택

   1. AWS IoT 콘솔의 작업 허브로 이동하여 작업 템플릿의 기반으로 사용할 작업을 선택합니다.
   2. 작업 템플릿으로 저장(Save as a job template)을 선택합니다.

> **Note**
>
> 선택적으로 다른 작업 문서를 선택하거나 원래 작업에서 고급 구성을 편집한 다음 작업 템플릿 생성(Create job template)을 선택할 수 있습니다. 새 작업 템플릿이 작업 템플릿 페이지에 나타납니다.

2. 작업 템플릿 속성 지정

   작업 템플릿 생성(Create job template) 페이지에서 작업 이름에 대한 영숫자 식별자와 템플릿에 대한 추가 세부 정보를 제공하는 영숫자 설명을 입력합니다.

   > **Note**
   >
   > 작업 문서는 템플릿을 생성할 때 지정한 작업 파일입니다. 작업 문서가 S3 위치 대신 작업 내에 지정된 경우 이 작업의 세부 정보 페이지에서 작업 문서를 볼 수 있습니다.

3. 작업에 대한 추가 구성을 계속 추가한 다음 작업을 검토하고 생성합니다. 추가 구성에 대한 자세한 내용은 다음 섹션을 참조하세요.

   - 작업 롤아웃, 예약 및 중단 구성
   - 작업 실행 제한 시간 및 재시도 구성

사용자 정의 작업 템플릿에서 작업 생성

이 주제에 설명된 대로 작업 템플릿의 세부 정보 페이지로 이동하여 사용자 정의 작업 템플릿에서 작업을 생성할 수 있습니다. 작업 생성 워크플로를 실행할 때 사용할 작업 템플릿을 선택하여 작업을 생성할 수도 있습니다. 자세한 내용은 를 사용하여 작업 생성 및 관리 AWS Management Console 단원을 참조하십시오.

이 주제에서는 사용자 정의 작업 템플릿의 세부 정보 페이지에서 작업을 생성하는 방법을 보여줍니다. AWS 관리형 템플릿에서 작업을 생성할 수도 있습니다. 자세한 내용은 관리형 템플릿을 사용하여 작업 생성 단원을 참조하십시오.

1. 사용자 정의 작업 템플릿 선택

   AWS IoT 콘솔의 작업 템플릿 허브로 이동하여 사용자 지정 템플릿 탭을 선택한 다음 템플릿을 선택합니다.

2. 사용자 정의 템플릿을 사용하여 작업 생성

   작업을 생성하려면

   1. 템플릿의 세부 정보 페이지에서 작업 생성(Create job)을 선택합니다.

      템플릿 구성이 추가된 작업 생성(Create job) 워크플로의 사용자 정의 작업 속성(Custom job properties) 단계로 콘솔이 전환됩니다.
   2. 고유한 영숫자 작업 이름, 설명(선택 사항) 및 태그를 입력하고 다음(Next)을 선택합니다.
   3. 이 작업에서 실행할 사물 또는 사물 그룹을 작업 대상으로 선택합니다.

      작업 문서(Job document) 섹션에 템플릿이 구성 설정과 함께 표시됩니다. 다른 작업 문서를 사용하려면 찾아보기를 선택하고 다른 버킷과 문서를 선택합니다. 다음을 선택합니다.
   4. Job configuration(작업 구성) 페이지에서 작업 유형을 연속 또는 스냅샷 작업으로 선택합니다. 스냅샷 작업은 대상 디바이스 및 그룹에서 실행이 완료되면 완료됩니다. 연속 작업은 사물 그룹에 적용되며 지정된 대상 그룹에 추가하는 모든 디바이스에서 실행됩니다.
   5. 작업에 대한 추가 구성을 계속 추가한 다음 작업을 검토하고 생성합니다. 추가 구성에 대한 자세한 내용은 다음 섹션을 참조하세요.
      - 작업 롤아웃, 예약 및 중단 구성
      - 작업 실행 제한 시간 및 재시도 구성

> **Note**
>
> 작업 템플릿에서 생성된 작업이 작업 템플릿에서 제공하는 기존 파라미터를 업데이트하면 업데이트된 파라미터가 해당 작업의 작업 템플릿에서 제공한 기존 파라미터를 재정의합니다.

플릿 허브 웹 애플리케이션을 사용하여 작업 템플릿에서 작업을 생성할 수도 있습니다. Fleet Hub에서 작업을 생성하는 방법에 대한 자세한 내용은 Fleet Hub for AWS IoT Device Management의 작업 템플릿 작업을 참조하세요.

작업 템플릿 삭제

작업 템플릿을 삭제하려면 먼저 AWS IoT 콘솔의 작업 템플릿 허브로 이동하여 사용자 지정 템플릿 탭을 선택합니다. 그런 다음 삭제할 작업 템플릿을 선택하고 다음을 선택합니다.

> **Note**
>
> 삭제는 영구적이며 작업 템플릿은 더 이상 사용자 정의 템플릿(Custom templates) 탭에 표시되지 않습니다.

를 사용하여 사용자 지정 작업 템플릿 생성 AWS CLI

이 주제에서는 AWS CLI를 사용하여 작업 템플릿에 대한 세부 정보를 생성, 삭제 및 검색하는 방법에 대해 설명합니다.

새 작업 템플릿 생성

다음 AWS CLI 명령은 S3 버킷에 저장된 작업 문서(*job-document.json*)와 Amazon S3(*S3DownloadRole*)에서 파일을 다운로드할 수 있는 권한이 있는 역할을 사용하여 작업을 생성하는 방법을 보여줍니다.

```
aws iot create-job-template  \
      --job-template-id 010  \
      --description "My custom job template for updating the device firmware"
      --document-source https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json
 \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\":
 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\":
 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

선택 사항인 `timeout-config` 파라미터는 각 디바이스가 작업 실행을 완료해야 하는 시간을 지정합니다. 타이머는 작업 실행 상태가 `IN_PROGRESS`에 설정되면 시작합니다. 시간이 만료되기 전에 작업 실행 상태가 다른 터미널 상태로 설정되어 있지 않으면 `TIMED_OUT`으로 자동으로 설정됩니다.

진행 중 타이머는 업데이트될 수 없으며 이 작업에 대한 모든 작업 시작에 적용됩니다. 작업 시작이이 간격보다 오래 IN_PROGRESS 상태로 유지될 때마다 작업 시작이 실패하고 터미널 TIMED_OUT 상태로 전환됩니다. AWS IoT 또한는 MQTT 알림을 게시합니다.

작업 롤아웃 및 중단에 대한 구성 생성에 대한 자세한 내용은 작업 롤아웃 및 중단 구성을 참조하세요.

> **Note**
>
> Amazon S3 파일로 지정되는 작업 문서는 작업을 생성할 때 가져오게 됩니다. 작업을 생성한 후 작업 문서의 원본으로 사용한 Amazon S3 파일의 내용을 변경해도 작업 대상으로 전송되는 내용은 변경되지 않습니다.

기존 작업에서 작업 템플릿 생성

다음 AWS CLI 명령은 기존 작업의 Amazon 리소스 이름(ARN)을 지정하여 작업 템플릿을 생성합니다. 새 작업 템플릿은 작업에 지정된 모든 구성을 사용합니다. 필요에 따라 선택적 파라미터를 사용하여 기존 작업의 구성을 변경할 수 있습니다.

```
aws iot create-job-template  \
      --job-arn arn:aws:iot:region:123456789012:job/job-name  \
      --timeout-config inProgressTimeoutInMinutes=100
```

작업 템플릿에 대한 세부 정보 가져오기

다음 AWS CLI 명령은 지정된 작업 템플릿에 대한 세부 정보를 가져옵니다.

```
aws iot describe-job-template \
      --job-template-id template-id
```

이 명령은 다음 출력을 표시합니다.

```
{
   "abortConfig": {
      "criteriaList": [
         {
```

```
            "action": "string",
            "failureType": "string",
            "minNumberOfExecutedThings": number,
            "thresholdPercentage": number
         }
      ]
   },
   "createdAt": number,
   "description": "string",
   "document": "string",
   "documentSource": "string",
   "jobExecutionsRolloutConfig": {
      "exponentialRate": {
         "baseRatePerMinute": number,
         "incrementFactor": number,
         "rateIncreaseCriteria": {
            "numberOfNotifiedThings": number,
            "numberOfSucceededThings": number
         }
      },
      "maximumPerMinute": number
   },
   "jobTemplateArn": "string",
   "jobTemplateId": "string",
   "presignedUrlConfig": {
      "expiresInSec": number,
      "roleArn": "string"
   },
   "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
}
```

작업 템플릿 나열

다음 AWS CLI 명령은의 모든 작업 템플릿을 나열합니다 AWS 계정.

```
aws iot list-job-templates
```

이 명령은 다음 출력을 표시합니다.

```
{
   "jobTemplates": [
      {
         "createdAt": number,
         "description": "string",
         "jobTemplateArn": "string",
         "jobTemplateId": "string"
      }
   ],
   "nextToken": "string"
}
```

추가 결과 페이지를 검색하려면 nextToken 필드의 값을 사용합니다.

작업 템플릿 삭제

다음 AWS CLI 명령은 지정된 작업 템플릿을 삭제합니다.

```
aws iot delete-job-template \
      --job-template-id template-id
```

명령이 아무런 출력도 표시하지 않습니다.

사용자 정의 작업 템플릿에서 작업 생성

다음 AWS CLI 명령은 사용자 지정 작업 템플릿에서 작업을 생성합니다. thing0ne(이)라는 디바이스를 대상으로 하고, 작업의 기초로 사용할 작업 템플릿의 Amazon 리소스 이름(ARN)을 지정합니다. create-job 명령의 연결된 파라미터를 전달하여 시간 초과 및 취소 구성과 같은 고급 구성을 재정의할 수 있습니다.

> **Warning**
>
> document-parameters 객체는 AWS 관리형 템플릿으로 작업을 생성할 때만 create-job 명령과 함께 사용해야 합니다. 사용자 지정 작업 템플릿에 이 객체를 사용해서는 안 됩니다. 이 파라미터를 사용하여 작업을 만드는 방법의 예는 관리형 템플릿을 사용하여 작업 생성 섹션을 참조하세요.

```
aws iot create-job \
      --targets arn:aws:iot:region:123456789012:thing/thingOne  \
      --job-template-arn arn:aws:iot:region:123456789012:jobtemplate/template-id
```

작업 구성

지정된 대상에 배포하는 각 작업에 대해 다음과 같은 추가 구성을 지정할 수 있습니다.

- 롤아웃: 분당 작업 문서를 수신하는 디바이스 수를 정의합니다.
- 예약: 반복 유지 관리 기간을 사용하는 것 외에도 미래 날짜 및 시간에 작업을 예약합니다.
- 중단: 일부 디바이스가 작업 알림을 받지 못하거나 디바이스에서 작업 실행에 대한 실패를 보고하는 경우와 같은 사례에서 작업을 취소합니다.
- 시간 제한(Timeout): 작업 실행이 시작된 후 일정 기간 내에 작업 대상에서 응답이 없는 경우 작업이 실패할 수 있습니다.
- 재시도: 디바이스가 작업 실행을 완료하려고 할 때 실패를 보고하거나 작업 실행 시간이 초과된 경우 작업 실행을 재시도합니다.

이러한 구성을 사용하여 작업 실행 상태를 모니터링하고 잘못된 업데이트가 전체 플릿으로 전송되는 것을 방지할 수 있습니다.

주제

- 작업 구성 작동 방식
- 추가 구성 지정

작업 구성 작동 방식

작업을 배포할 때 롤아웃 및 중단 구성을 사용하고 작업을 실행할 때 제한 시간 및 재시도 구성을 사용합니다. 다음 섹션은 이러한 구성의 작동 방식에 대한 자세한 정보를 보여줍니다.

주제

- 작업 롤아웃, 예약 및 중단 구성
- 작업 실행 제한 시간 및 재시도 구성

작업 롤아웃, 예약 및 중단 구성

작업 롤아웃, 예약, 중단 구성을 사용하여 작업 문서를 수신하는 디바이스 수를 정의하고, 작업 롤아웃을 예약하고, 작업을 취소하는 기준을 결정할 수 있습니다.

작업 롤아웃 구성

대상에게 대기 중인 작업 실행을 얼마나 빨리 알릴지 지정할 수 있습니다. 또한 단계별 롤아웃을 생성하여 업데이트, 재부팅 및 기타 작업을 효과적으로 관리할 수 있습니다. 대상에게 알릴 방법을 지정하려면 작업 롤아웃 속도를 사용합니다.

작업 롤아웃 속도

상수 롤아웃 속도 또는 기하급수적인 롤아웃 속도를 사용하여 롤아웃 구성을 생성할 수 있습니다. 분당 알릴 최대 작업 대상 수를 지정하려면 상수 롤아웃 속도를 사용합니다.

AWS IoT 다양한 기준과 임계값이 충족되면 지수 롤아웃 비율을 사용하여 작업을 배포할 수 있습니다. 실패한 작업 수가 지정한 기준과 일치하면 작업 롤아웃을 취소할 수 있습니다. 작업을 생성할 때 `JobExecutionsRolloutConfig` 객체를 사용하여 작업 롤아웃 속도 기준을 설정합니다. 또한 작업 생성 시 `AbortConfig` 객체를 사용하여 작업 중단 기준을 설정합니다.

다음 예에서는 롤아웃 속도가 작동하는 방식을 보여줍니다. 분당 50개의 기본 속도, 증분 계수 2, 알림을 받고 성공한 분당 디바이스의 수 1,000을 사용하는 작업 롤아웃을 예로 들면, 작업은 분당 50개의 작업을 실행하는 속도로 시작되고 1000개의 사물이 작업 실행 알림을 수신하거나 1,000개의 성공적인 작업 실행이 발생할 때까지 해당 속도로 계속됩니다.

다음 표는 처음 4회의 증가에 걸쳐 롤아웃이 어떻게 진행되는지 보여줍니다.

분당 롤아웃 속도	50	100	200	400
속도 증가를 충족하기 위한 알림을 받는 디바이스 또는 성공한 작업 실행의 수	1,000	2,000	3,000	4,000

> **Note**
>
> 최대 동시 작업 제한인 500개 작업(`isConcurrent = True`)에 도달하면 모든 활성 작업은 `IN-PROGRESS` 상태로 유지되며 동시 작업 수가 499개 이하(`isConcurrent = False`))가 될 때까지 새 작업 실행이 롤아웃되지 않습니다. 이는 스냅샷 작업과 연속 작업에 적용됩니다. `isConcurrent = True`인 경우 현재 작업이 대상 그룹의 모든 디바이스에 작업 실행을 롤아웃하는 중입니다. `isConcurrent = False`인 경우 이 작업은 대상 그룹의 모든 디바이스에

대한 모든 작업 실행 롤아웃을 완료한 것입니다. 대상 그룹의 모든 디바이스가 최종 상태에 도달하거나 대상 그룹의 임계값 백분율에 도달할 때(작업 중단 구성을 선택한 경우) 상태가 업데이트됩니다. `isConcurrent = True` 및 `isConcurrent = False`인 경우 작업 수준 상태는 모두 `IN_PROGRESS`입니다.
활성 및 동시 작업 제한에 대한 자세한 내용은 활성 및 동시 작업 제한 섹션을 참조하세요.

동적 사물 그룹을 사용한 연속 작업의 작업 롤아웃 속도

연속 작업을 사용하여 플릿에서 원격 작업을 롤아웃하면 AWS IoT 작업에서는 대상 사물 그룹의 디바이스에 대한 작업 실행을 롤아웃합니다. 새 디바이스가 동적 사물 그룹에 추가되는 경우 이러한 작업 실행은 작업이 생성된 후에도 새로 추가된 디바이스로 계속 롤아웃됩니다.

롤아웃 구성은 작업을 만들 때까지 그룹에 추가된 디바이스에 대해서만 롤아웃 속도를 제어할 수 있습니다. 작업이 생성된 후 새 디바이스에 대해 디바이스가 대상 그룹에 조인하자마자 거의 실시간으로 작업 실행이 만들어집니다.

작업 예약 구성

미리 결정된 시작 시간, 종료 시간 및 종료 동작(종료 시간에 도달할 때 각 작업 실행에 발생하는 일)을 사용하여 최대 1년 전에 연속 또는 스냅샷 작업을 예약할 수 있습니다. 또한 연속 작업의 빈도, 시작 시간 및 기간을 유연하게 조정할 수 있는 선택적 반복 유지 관리 기간을 만들어 대상 그룹 내의 모든 디바이스에 작업 문서를 롤아웃할 수 있습니다.

작업 예약 구성

시작 시간

예약된 작업의 시작 시간은 작업이 대상 그룹의 모든 디바이스에 대한 작업 문서 롤아웃을 시작할 미래 날짜 및 시간입니다. 예약된 작업의 시작 시간은 연속 작업과 스냅샷 작업에 적용됩니다. 예약된 작업이 처음 생성되면 `SCHEDULED` 상태를 유지합니다. 선택한 `startTime`에 도달하면 `IN_PROGRESS`로 업데이트되고 작업 문서 롤아웃이 시작됩니다. `startTime`은 예약된 작업을 처음 생성한 날짜 및 시간으로부터 1년 이내여야 합니다.

API 명령 또는를 사용할 `startTime` 때의 구문에 대한 자세한 내용은 타임스탬프를 AWS CLI참조하세요.

일광 절약 시간(DST)을 준수하는 위치에서 반복되는 유지 관리 기간 중에 수행되는 선택적 예약 구성을 사용하는 작업의 경우 DST에서 표준 시간으로, 표준 시간에서 DST로 전환하면 시간이 1시간 변경됩니다.

> **Note**
>
> 에 표시되는 시간대는 현재 시스템 시간대 AWS Management Console 입니다. 그러나 이러한 시간대는 시스템에서 UTC로 변환됩니다.

종료 시간

예약된 작업의 종료 시간은 작업이 대상 그룹의 나머지 디바이스에 대한 작업 문서 롤아웃을 중지할 미래 날짜 및 시간입니다. 예약된 작업의 종료 시간은 연속 작업과 스냅샷 작업에 적용됩니다. 예약된 작업이 선택한 endTime에 도달하고 모든 작업 실행이 최종 상태에 도달하면 상태 상태가 IN_PROGRESS에서 COMPLETED로 업데이트됩니다. endTime은 예약된 작업을 처음 생성한 날짜 및 시간으로부터 2년 이내여야 합니다. startTime과 endTime 사이의 최소 기간은 30분입니다. 작업이 endTime에 도달할 때까지 작업 실행 재시도가 수행되며 endBehavior에 따라 이후 진행 방법이 결정됩니다.

API 명령 또는를 사용할 endTime 때의 구문에 대한 자세한 내용은 타임스탬프를 AWS CLI참조하세요.

일광 절약 시간(DST)을 준수하는 위치에서 반복되는 유지 관리 기간 중에 수행되는 선택적 예약 구성을 사용하는 작업의 경우 DST에서 표준 시간으로, 표준 시간에서 DST로 전환하면 시간이 1시간 변경됩니다.

> **Note**
>
> 에 표시되는 시간대는 현재 시스템 시간대 AWS Management Console 입니다. 그러나 이러한 시간대는 시스템에서 UTC로 변환됩니다.

종료 동작

예약된 작업의 종료 동작은 작업이 선택한 endTime에 도달할 때 해당 작업 및 완료되지 않은 모든 작업 실행에 발생하는 일을 결정합니다.

작업 또는 작업 템플릿을 생성할 때 선택할 수 있는 종료 동작은 다음과 같습니다.

- STOP_ROLLOUT
  - STOP_ROLLOUT은 작업 대상 그룹의 나머지 모든 디바이스에 대한 작업 문서 롤아웃을 중지합니다. 또한 모든 QUEUED 및 IN_PROGRESS 작업 실행은 최종 상태에 도달할 때까지 계속됩니다. CANCEL 또는 FORCE_CANCEL을 선택하지 않는 한, 이것이 기본 종료 동작입니다.

- CANCEL
  - CANCEL은 작업 대상 그룹의 나머지 모든 디바이스에 대한 작업 문서 롤아웃을 중지합니다. 또한 모든 QUEUED 작업 실행은 취소되지만 모든 IN_PROGRESS 작업 실행은 최종 상태에 도달할 때까지 계속됩니다.
- FORCE_CANCEL
  - FORCE_CANCEL은 작업 대상 그룹의 나머지 모든 디바이스에 대한 작업 문서 롤아웃을 중지합니다. 또한 모든 QUEUED 및 IN_PROGRESS 작업 실행이 취소됩니다.

> **Note**
>
> endtime을 선택하려면 endbehavior 하나를 선택해야 합니다.

최대 기간

예약된 작업의 최대 기간은 startTime 및 endTime에 관계없이 2년 이하여야 합니다.

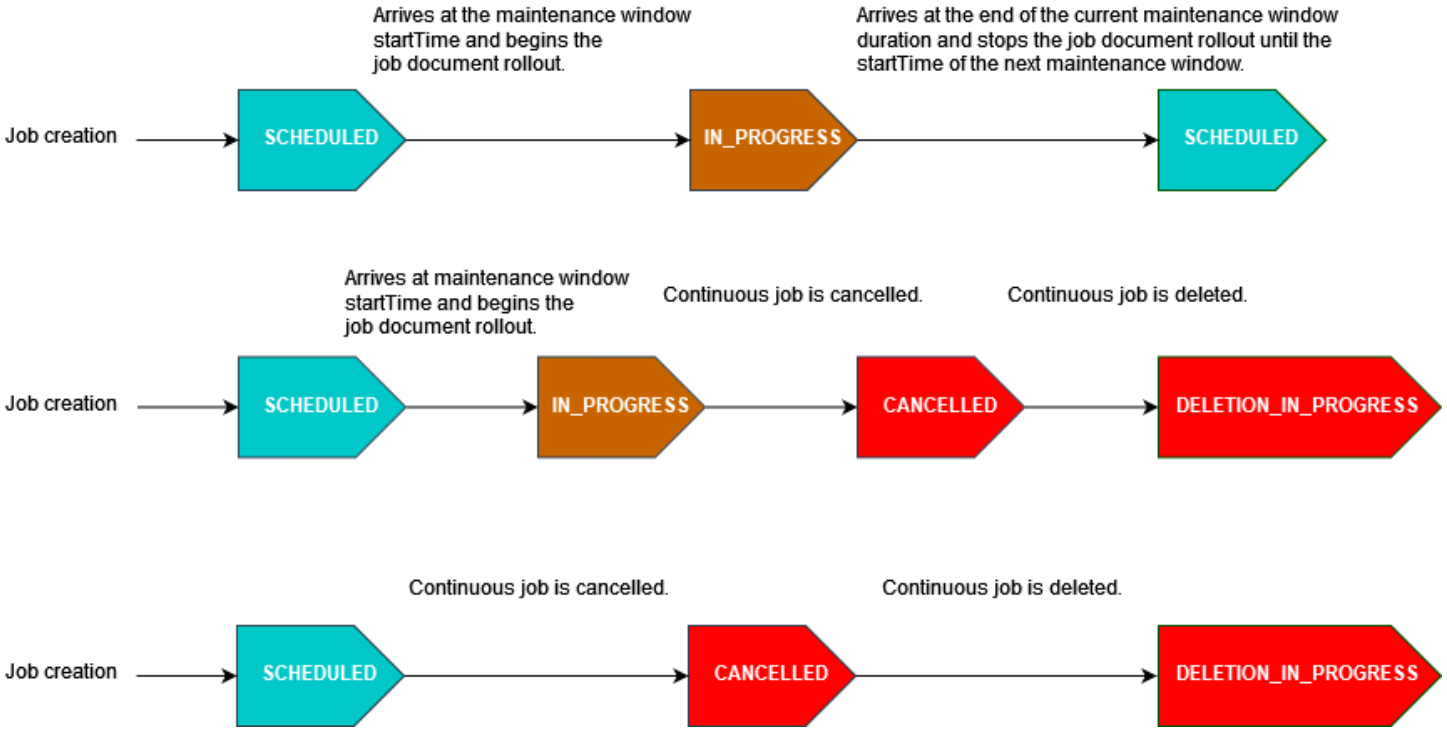
다음 테이블은 예약된 작업의 일반적인 기간 시나리오입니다.

예약된 작업 예시 번호	startTime	endTime	최대 기간
1	최초 작업 생성 직후	최초 작업 생성 1년 후	1년
2	최초 작업 생성 1개월 후	최초 작업 생성 13개월 후	1년
3	최초 작업 생성 1년 후	최초 작업 생성 2년 후	1년
4	최초 작업 생성 직후	최초 작업 생성 2년 후	2년

반복 유지 관리 기간

유지 관리 기간은의 예약 구성 내에서 AWS Management Console , 그리고 CreateJob 및 CreateJobTemplate API SchedulingConfig 내에서 선택적 구성입니다. APIs 미리 정해진 시작 시간, 기간 및 유지 관리 기간이 발생하는 빈도(매일, 매주 또는 매월)로 반복 유지 관리 기간을 설정할 수 있습니다. 유지 관리 기간은 연속 작업에만 적용됩니다. 반복 유지 관리 기간의 최대 기간은 23시간 50분입니다.

다음 다이어그램은 선택적으로 유지 관리 기간이 적용된 다양한 예약된 작업 시나리오의 작업 상태를 보여줍니다.



작업 상태에 대한 자세한 내용은 작업 및 작업 실행 상태 섹션을 참조하세요.

> **Note**
>
> 유지 관리 기간 중에 작업이 `endTime`에 도달하면 `IN_PROGRESS`에서 `COMPLETED`로 업데이트됩니다. 또한 나머지 모든 작업 실행은 작업의 `endBehavior`에 따라 실행됩니다.

Cron 표현식

사용자 지정 빈도를 사용하여 유지 관리 기간 중에 작업 문서를 롤아웃하는 예약된 작업의 경우 사용자 지정 빈도는 cron 표현식을 사용하여 입력됩니다. Cron 표현식에는 각각 공백으로 구분되는 필수 필드 6개가 있습니다.

구문

```
cron(fields)
```

필드	값	와일드카드
Minutes	0~59	, - * /
시간	0~23	, - * /
날짜	1~31	, - * ? / L W
월	1-12 또는 JAN-DEC	, - * /
요일	1-7 또는 SUN-SAT	, - * ? L #
연도	1970~2199	, - * /

와일드카드

- ,(쉼표) 와일드카드는 추가 값을 포함합니다. 예컨대, Month 필드에서 JAN, FEB, MAR은 1월, 2월, 3월을 포함한다는 의미입니다.
- -(대시) 와일드카드는 범위를 지정합니다. 예컨대, Day 필드에서 1-15는 지정된 달의 1일에서 15일까지 포함한다는 의미입니다.
- *(별표) 와일드카드는 필드의 모든 값을 포함합니다. '시간' 필드에서 *는 모든 시간을 포함한다는 의미입니다. '일' 및 '요일' 필드 모두에서 *를 사용할 수 없습니다. 필드 중 하나에 사용할 경우 다른 하나에는 반드시 ?를 사용해야 합니다.
- /(슬래시) 와일드카드로 증분을 지정합니다. 예를 들어, '분' 필드에 1/10을 입력하면 지정한 시간의 1분부터 시작해서 매 10분 간격을 지정할 수 있습니다(즉, 11분, 21분, 31분 등).
- ?(물음표) 와일드카드는 어떤 한 가지나 다른 것을 지정합니다. '일' 필드에 7을 입력하고 '요일' 필드에는 ?을 입력하면 매월 7일이 무슨 요일이든 상관없이 7번째 되는 날을 지정한다는 의미입니다.
- '날짜' 또는 '요일' 필드에서 L 와일드카드로 해당 월 또는 주의 마지막 날을 지정할 수 있습니다.
- '날짜' 필드에서는 **W** 와일드카드로 어떤 한 평일을 지정할 수 있습니다. 예를 들어 '날짜' 필드에 **3W**를 입력하면 해당 월의 세 번째 평일에 가장 가까운 날을 지정할 수 있습니다.
- '요일' 필드의 # 와일드카드는 그 달에 속한 정해진 요일의 특정 인스턴스를 지정합니다. 예를 들어, 3#2는 그 달의 두 번째 화요일입니다. 3은 각 주의 셋째 날이므로 화요일을 나타내고 2는 그 달의 두 번째 해당 요일입니다.

> **Note**
>
> '#' 문자를 사용하는 경우 요일(day-of-week) 필드에 하나의 표현식만 정의할 수 있습니다. 예를 들어 "3#1,6#3"은 두 개의 표현식으로 해석되기 때문에 유효하지 않습니다.

제한 사항

- 같은 cron 표현식에서 '날짜' 및 '요일' 필드를 지정할 수 없습니다. 이들 필드 중 하나에 값(또는 *)을 지정하는 경우에는 다른 필드에서 반드시 ?를 사용해야 합니다.

예시

반복 유지 관리 기간의 `startTime`에 cron 표현식을 사용할 때는 다음 샘플 cron 문자열을 참조하세요.

분	시간	일	월	요일	연도	의미
0	10	*	*	?	*	매일 오전 10시(UTC)에 실행
15	12	*	*	?	*	매일 오후 12시 15분(UTC)에 실행
0	18	?	*	월-금	*	매주 월요일부터 금요일까지 오후 6시(UTC)에 실행
0	8	1	*	?	*	매월 1일 오전 8시

분	시간	일	월	요일	연도	의미
						(UTC)에 실행

반복 유지 관리 기간 종료 로직

유지 관리 기간 중 작업 롤아웃이 현재 유지 관리 기간 종료 시점에 도달하면 다음과 같은 작업이 수행됩니다.

- 작업은 대상 그룹의 나머지 모든 디바이스에 대한 작업 문서 롤아웃을 중지합니다. 다음 유지 관리 기간의 `startTime`에 재개됩니다.
- 상태가 `QUEUED`인 모든 작업 실행은 다음 유지 관리 기간의 `startTime`까지 `QUEUED` 상태로 유지됩니다. 다음 기간에는 디바이스가 작업 문서에 지정된 작업을 수행할 준비가 되었을 때 `IN_PROGRESS`로 전환할 수 있습니다.
- 상태가 `IN_PROGRESS`인 모든 작업 실행은 최종 상태에 도달할 때까지 작업 문서에 지정된 작업을 계속 수행합니다. `JobExecutionsRetryConfig`에 지정된 대로 모든 재시도는 다음 유지 관리 기간의 `startTime`에 이루어집니다.

작업 중단 구성

이 구성을 사용하여 디바이스의 임계값 비율이 해당 기준을 충족할 때 작업을 취소하는 기준을 생성할 수 있습니다. 예를 들어, 이 구성을 사용하여 다음과 같은 경우 작업을 취소할 수 있습니다.

- 임계값 비율의 디바이스가 작업 실행 알림을 받지 못하는 경우(예: 디바이스가 무선 업데이트(OTA)와 호환되지 않는 경우). 이 경우 디바이스가 `REJECTED` 상태를 보고할 수 있습니다.
- 임계값 비율의 디바이스가 작업 실행에 대해 실패를 보고하는 경우(예: Amazon S3 URL에서 작업 문서를 다운로드하려고 할 때 디바이스 연결이 끊어지는 경우). 이와 같은 경우 디바이스는 `FAILURE` 상태를 AWS IoT에 보고하도록 프로그래밍되어야 합니다.
- 작업 실행이 시작된 후 임계값 비율의 디바이스에 대해 작업 실행이 시간 초과되어 `TIMED_OUT` 상태가 보고되는 경우.
- 재시도 실패가 여러 번 있는 경우. 재시도 구성을 추가하는 경우 각 재시도가 AWS 계정에 추가 요금을 발생시킬 수 있습니다. 이와 같은 경우 작업을 취소하면 대기열에 있는 작업 실행을 취소하고 이러한 실행에 대한 재시도를 방지할 수 있습니다. 재시도 구성 및 이 구성을 중단 구성과 함께 사용하는 것에 대한 자세한 내용은 작업 실행 제한 시간 및 재시도 구성 섹션을 참조하세요.

AWS IoT 콘솔 또는 작업 API를 사용하여 AWS IoT 작업 중단 조건을 설정할 수 있습니다.

작업 실행 제한 시간 및 재시도 구성

작업 실행 제한 시간 구성을 사용하면 작업 실행이 설정된 기간보다 오래 진행되는 경우 작업 알림을 전송할 수 있습니다. 작업 실행 재시도 구성을 사용하면 작업이 실패하거나 시간 초과되는 경우 실행을 재시도할 수 있습니다.

작업 실행 시간 제한 구성

작업 실행 시간 제한 구성을 사용하면 작업 실행이 예상치 않게 장시간 동안 IN_PROGRESS 상태에 멈출 때마다 알림을 받을 수 있습니다. 작업이 IN_PROGRESS 상태이면 작업 실행의 진행 상황을 모니터링할 수 있습니다.

작업 시간 제한 타이머

타이머에는 진행 중 타이머와 단계 타이머 두 가지 유형이 있습니다.

진행 중 타이머

작업 또는 작업 템플릿을 생성할 때 진행 중 타이머에 1분에서 7일 사이의 값을 지정할 수 있습니다. 작업 실행 전에는 이 타이머의 값을 업데이트할 수 있습니다. 타이머가 시작된 후에는 업데이트할 수 없으며 타이머 값은 해당 작업의 모든 작업 실행에 적용됩니다. 작업 실행이이 간격보다 오래 IN_PROGRESS 상태를 유지할 때마다 작업 실행이 실패하고 터미널 TIMED_OUT 상태로 전환됩니다. AWS IoT 또한는 MQTT 알림을 게시합니다.

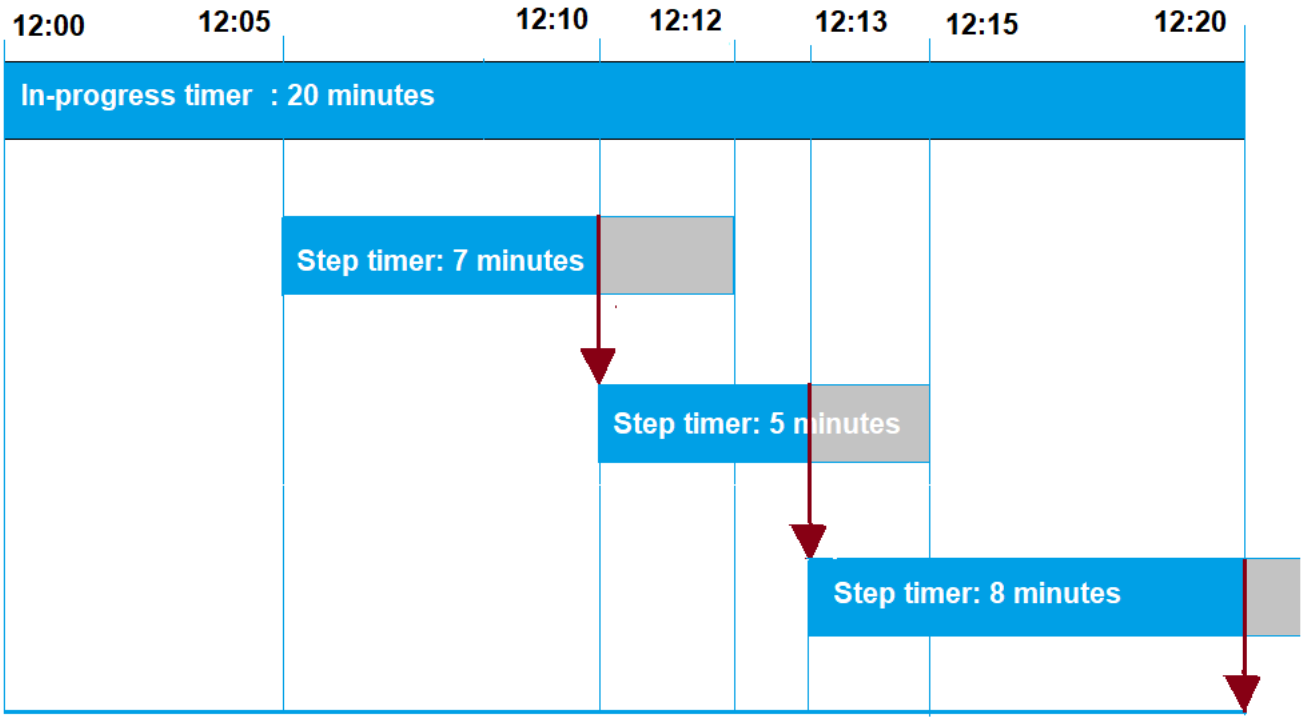
단계 타이머

업데이트하려는 작업 실행에만 적용되는 단계 타이머를 설정할 수도 있습니다. 이 타이머는 진행 중 타이머에는 영향을 주지 않습니다. 작업 실행을 업데이트할 때마다 이 타이머에 대해 새 값을 설정할 수 있습니다. 사물에 대해 대기 중인 다음 작업 실행을 시작할 때 새 단계 타이머를 생성할 수도 있습니다. 작업 실행이 단계 타이머 간격보다 오랫동안 IN_PROGRESS 상태를 유지하는 경우, 해당 작업 실행은 실패하며 터미널 상태인 TIMED_OUT으로 전환됩니다.

> **Note**
>
> AWS IoT 콘솔 또는 AWS IoT 작업 API를 사용하여 진행 중인 타이머를 설정할 수 있습니다. 단계 타이머를 지정하려면 API를 사용합니다.

작업 시간 제한에 대해 타이머가 작동하는 방식

다음은 20분 시간 제한 기간 동안 진행 중 시간 제한과 단계 시간 제한이 서로 상호 작용하는 방식을 설명합니다.



다음은 여러 단계를 보여줍니다.

1. 12:00

   새 작업이 생성되고 작업을 생성할 때 20분 동안 진행 중 타이머가 시작됩니다. 진행 중 타이머가 실행되기 시작하고 작업 실행은 `IN_PROGRESS` 상태로 전환됩니다.

2. 12:05 PM

   7분 값으로 새 단계 타이머가 생성됩니다. 이제 작업 실행이 오후 12시 12분에 시간 초과됩니다.

3. 12:10 PM

   5분 값으로 새 단계 타이머가 생성됩니다. 새 단계 타이머가 생성되면 이전 단계 타이머가 삭제되고 이제 작업 실행은 오후 12시 15분에 시간 초과됩니다.

4. 12:13 PM

9분 값으로 새 단계 타이머가 생성됩니다. 이전 단계 타이머가 삭제되고 이제 작업 실행은 진행 중 타이머가 오후 12시 20분에 시간 초과되므로 오후 12시 20분에 시간 초과됩니다. 단계 타이머는 진행 중 타이머의 절대 한계를 초과할 수 없습니다.

작업 실행 재시도 구성

재시도 구성을 사용하여 특정 조건 집합이 충족될 때 작업 실행을 재시도할 수 있습니다. 작업이 시간 초과되거나 디바이스가 실패할 때 재시도를 시도할 수 있습니다. 시간 제한 실패로 인해 실행을 재시도하려면 시간 제한 구성을 사용하도록 설정해야 합니다.

재시도 구성 사용 방법

다음 단계에 따라 구성을 재시도합니다.

1. FAILED, TIMED_OUT 또는 두 실패 기준 모두에 대해 재시도 구성을 사용할지 여부를 결정합니다. TIMED_OUT 상태의 경우 상태가 보고되면 AWS IoT 작업에서 디바이스에 대한 작업 실행을 자동으로 재시도합니다.
2. FAILED 상태의 경우 작업 실행 실패를 재시도할 수 있는지 확인합니다. 재시도 가능한 경우 디바이스를 FAILURE 상태를 AWS IoT에 보고하도록 프로그래밍합니다. 다음 섹션에서는 재시도 가능한 실패와 재시도 불가능한 실패에 대해 자세히 설명합니다.
3. 앞의 정보를 사용하여 각 실패 유형에 사용할 재시도 횟수를 지정합니다. 단일 디바이스의 경우 두 실패 유형을 모두 합하여 최대 10회의 재시도를 지정할 수 있습니다. 재시도는 실행이 성공하거나 지정된 시도 횟수에 도달하면 자동으로 중지됩니다.
4. 반복적으로 재시도가 실패하는 경우 많은 재시도 횟수로 인해 추가 요금이 발생하지 않도록 작업을 취소하려면 중단 구성을 추가합니다.

> **Note**
>
> 작업이 반복 유지 관리 기간 종료에 도달하면 모든 IN_PROGRESS 작업 실행은 최종 상태에 도달할 때까지 작업 문서에 명시된 작업을 계속 수행합니다. 작업 실행이 최종 상태인 FAILED 또는 유지 관리 기간을 벗어난 TIMED_OUT 상태에 도달하면 시도 횟수가 모두 소진되지 않았으면 다음 기간에 재시도가 이루어집니다. 다음 유지 관리 기간의 startTime에 새 작업 실행이 생성되고 디바이스를 시작할 준비가 될 때까지 QUEUED의 상태가 됩니다.

재시도 및 중단 구성

재시도할 때마다에 추가 요금이 발생합니다 AWS 계정. 반복적인 재시도 실패로 인한 추가 요금이 발생하지 않도록 하려면 중단 구성을 추가하는 것이 좋습니다. 요금에 대한 자세한 내용은 AWS IoT Device Management 요금을 참조하세요.

높은 임계값 비율의 디바이스가 시간 초과되거나 실패를 보고하는 경우 재시도 실패 여러 번 발생할 수 있습니다. 이 경우 중단 구성을 사용하여 작업을 취소하고 대기열에 있는 모든 작업 실행이나 추가적인 재시도를 방지할 수 있습니다.

> **Note**
>
> 작업 실행을 취소하기 위한 중단 기준이 충족되는 경우 QUEUED 작업 실행만 취소됩니다. 대기열에 있는 디바이스에 대한 재시도는 시도되지 않습니다. 그러나 IN_PROGRESS 상태인 현재 작업 실행은 취소되지 않습니다.

또한 실패한 작업 실행을 재시도하기 전에 다음 섹션에 설명된 대로 작업 실행 실패가 재시도 가능한지 여부를 확인하는 것이 좋습니다.

FAILED 실패 유형에 대한 재시도

FAILED 실패 유형에 대한 재시도를 시도하려면 디바이스를 실패한 작업 실행에 대한 FAILURE 상태를 AWS IoT에 보고하도록 프로그래밍해야 합니다. FAILED 작업 실행을 재시도하는 기준으로 재시도 구성을 설정하고 수행할 재시도 횟수를 지정해야 합니다. AWS IoT 작업이 FAILURE 상태를 감지하면 디바이스에 대한 작업 실행을 자동으로 재시도합니다. 재시도는 작업 실행이 성공하거나 최대 재시도 횟수에 도달할 때까지 계속됩니다.

각 재시도 및 이러한 디바이스에서 실행 중인 작업을 추적할 수 있습니다. 실행 상태를 추적하여 지정된 횟수의 재시도가 시도된 후 디바이스를 사용하여 실패를 보고하고 다른 재시도를 초기화할 수 있습니다.

재시도 가능한 실패와 재시도 불가능한 실패

작업 실행 실패는 재시도 가능하거나 재시도 불가능할 수 있습니다. 각 재시도 시 AWS 계정에 요금이 부과될 수 있습니다. 여러 번의 재시도로 인해 추가 요금이 발생하지 않도록 하려면 먼저 작업 실행 실패가 재시도 가능한지 여부를 확인하는 것이 좋습니다. 재시도 가능한 실패의 예로는 Amazon S3 URL에서 작업 문서를 다운로드하려고 시도하는 동안 디바이스에서 발생하는 연결 오류가 있습니다. 작업 실행 실패가 재시도 가능하면 작업 실행이 실패한 경우 FAILURE 상태를 보고하도록 디바이스를 프로그래밍합니다. 그런 다음 FAILED 실행을 재시도하도록 재시도 구성을 설정합니다.

실행을 재시도할 수 없는 경우 재시도하거나 계정에 추가 요금이 부과될 가능성이 없도록 REJECTED 상태를 AWS IoT에 보고하도록 디바이스를 프로그래밍하는 것이 좋습니다. 재시도 불가능한 실패의 예로는 디바이스가 작업 업데이트 수신과 호환되지 않는 경우 또는 작업을 실행하는 동안 메모리 오류가 발생하는 경우를 들 수 있습니다. 이러한 경우 AWS IoT 작업은 FAILED 또는 TIMED_OUT 상태를 감지할 때만 작업 실행을 재시도하기 때문에 작업 실행을 재시도하지 않습니다.

작업 실행 실패가 재시도 가능하다고 확인된 후에도 재시도가 여전히 실패할 경우 디바이스 로그를 확인하는 것이 좋습니다.

> **Note**
>
> 선택적 예약 구성이 있는 작업이 해당 endTime에 도달하면, endBehavior가 대상 그룹의 나머지 모든 디바이스에 대한 작업 문서 롤아웃을 중지하고 나머지 작업 실행을 진행하는 방법을 결정합니다. 재시도 구성을 통해 선택한 경우 재시도됩니다.

TIMEOUT 실패 유형에 대한 재시도

작업을 생성할 때 제한 시간을 활성화하면 상태가에서 로 변경될 때 AWS IoT 작업이 디바이스에 대한 작업 실행을 다시 시도IN_PROGRESS합니다TIMED_OUT. 이 상태 변경은 진행 중 타이머가 시간 초과되거나 지정한 단계 타이머가 IN_PROGRESS 상태이다가 시간 초과될 경우 발생할 수 있습니다. 재시도는 작업 실행이 성공하거나 이 실패 유형에 대한 최대 재시도 횟수에 도달할 때까지 계속됩니다.

연속 작업 및 사물 그룹 멤버십 업데이트

작업 상태가 IN_PROGRESS인 연속 작업의 경우 사물의 그룹 멤버십이 업데이트되면 재시도 횟수가 0으로 재설정됩니다. 예를 들어, 다섯 번의 재시도 횟수를 지정했고 세 번의 재시도가 이미 수행되었다고 가정해 보겠습니다. 이제 동적 사물 그룹의 경우와 같이 사물이 사물 그룹에서 제거되었다가 그룹에 다시 가입하면 재시도 횟수가 0으로 재설정됩니다. 이제 사물 그룹에 대해 남은 두 번의 시도 대신 다섯 번의 재시도를 수행할 수 있습니다. 또한 사물 그룹에서 사물이 제거되면 추가 재시도 시도가 취소됩니다.

추가 구성 지정

작업 또는 작업 템플릿을 생성할 때 이러한 추가 구성을 지정할 수 있습니다. 다음은 이러한 구성을 지정할 수 있는 경우를 보여 줍니다.

- 사용자 정의 작업 템플릿을 생성하는 경우. 템플릿에서 작업을 생성할 때 지정한 추가 구성 설정이 저장됩니다.

- 작업 파일을 사용하여 사용자 정의 작업을 생성하는 경우. 작업 파일은 S3 버킷에 업로드되는 JSON 파일일 수 있습니다.
- 사용자 정의 작업 템플릿을 사용하여 사용자 정의 작업을 생성하는 경우. 이미 템플릿에 이러한 설정이 지정된 경우 설정을 재사용하거나 새 구성 설정을 지정하여 재정의할 수 있습니다.
- AWS 관리형 템플릿을 사용하여 사용자 지정 작업을 생성하는 경우.

주제

- 를 사용하여 작업 구성 지정 AWS Management Console
- 작업 API를 사용하여 AWS IoT 작업 구성 지정

를 사용하여 작업 구성 지정 AWS Management Console

AWS IoT 콘솔을 사용하여 작업에 대한 다양한 구성을 추가할 수 있습니다. 작업을 생성한 후 작업 세부 정보 페이지에서 작업 구성의 상태 세부 정보를 확인할 수 있습니다. 다양한 구성 및 구성 작동 방식에 대한 자세한 내용은 작업 구성 작동 방식 섹션을 참조하세요.

작업 또는 작업 템플릿을 생성할 때 작업 구성을 추가합니다.

사용자 정의 작업 템플릿을 생성하는 경우

사용자 정의 작업 템플릿을 생성할 때 롤아웃 구성을 지정하려면

1. AWS IoT 콘솔의 작업 템플릿 허브로 이동하여 작업 템플릿 생성을 선택합니다.
2. 작업 템플릿 속성을 지정하고, 작업 문서를 제공하고, 추가할 구성을 확장한 다음 구성 파라미터를 지정합니다.

사용자 정의 작업을 생성하는 경우

사용자 정의 작업을 생성할 때 롤아웃 구성을 지정하려면

1. AWS IoT 콘솔의 작업 허브로 이동하여 작업 생성을 선택합니다.
2. 사용자 정의 작업 생성(Create a custom job)을 선택하고 작업 속성, 대상 및 작업 문서에 대해 작업 파일 또는 템플릿을 사용할지 여부를 지정합니다. 사용자 지정 템플릿 또는 AWS 관리형 템플릿을 사용할 수 있습니다.
3. 작업 구성을 선택한 다음 롤아웃 구성(Rollout configuration)을 확장하여 상수 속도(Constant rate)를 사용할지, 아니면 지수 속도(Exponential rate)를 사용할지를 지정합니다. 그런 다음 구성 파라미터를 지정합니다.

다음 섹션에서는 각 구성에 대해 지정할 수 있는 파라미터를 보여줍니다.

롤아웃 구성

상수 롤아웃 속도를 사용할지, 아니면 지수 속도를 사용할지를 지정할 수 있습니다.

- 상수 롤아웃 속도 설정

  작업 실행에 대해 상수 속도를 설정하려면 상수 속도를 선택한 다음 속도의 상한에 대한 분당 최대를 지정합니다. 이 값은 선택 사항이며 범위는 1~1,000입니다. 이 값을 설정하지 않으면 1,000이 기본 값으로 사용됩니다.
- 기하급수적 롤아웃 속도 설정

  기하급수적 속도를 설정하려면 지수 속도(Exponential rate)를 선택하고 다음 파라미터를 지정합니다.
  - 분당 기본 속도

    속도 증가 기준에서 알림을 받은 디바이스 수 또는 성공한 디바이스 수 임계값이 충족될 때까지 작업이 실행되는 속도입니다.
  - 증분 계수

    속도 증가 기준(Rate increase criteria)에서 알림을 받는 디바이스 수(Number of notified devices) 또는 성공한 디바이스 수(Number of succeeded devices) 임계값이 충족된 후 롤아웃 속도 증가에 사용되는 지수 인자입니다.
  - 속도 증가 기준

    알림을 받는 디바이스 수(Number of notified devices) 또는 성공한 디바이스 수(Number of succeeded devices)에 대한 임계값입니다.

중단 구성

새 구성 추가(Add new configuration)를 선택하고 각 구성에 대해 다음 파라미터를 지정합니다.

- 실패 유형

  작업 중단을 시작하는 실패 유형을 지정합니다. 여기에는 FAILED, REJECTED, TIMED_OUT 또는 ALL이 포함됩니다.
- 증분 계수

  작업 중단 기준이 충족되기 전에 발생해야 하는 완료된 작업 실행 수를 지정합니다.

- 임계값 비율

  작업 중단을 시작하는 실행된 총 사물 수를 지정합니다.

예약 구성

각 작업을 처음 생성 시 바로 시작하거나 이후 날짜 및 시간에 시작되도록 예약하거나 반복 유지 관리 기간에 시작하도록 할수 있습니다.

새 구성 추가(Add new configuration)를 선택하고 각 구성에 대해 다음 파라미터를 지정합니다.

- Job start(작업 시작)

  작업이 시작될 날짜와 시간을 지정합니다.

- 반복 유지 관리 기간

  반복 유지 관리 기간은 작업이 작업의 대상 디바이스에 작업 문서를 배포할 수 있는 특정 날짜 및 시간을 정의합니다. 유지 관리 기간은 매일, 매주, 매월 또는 사용자 지정 날짜 및 시간에 반복될 수 있습니다.

- Job end(작업 종료)

  작업이 종료될 날짜와 시간을 지정합니다.

- Job end behavior(작업 종료 동작)

  작업이 끝날 때 완료되지 않은 모든 작업 실행에 대한 종료 동작을 선택합니다.

> **Note**
>
> 선택적 예약 구성과 종료 시간이 있는 작업이 해당 종료 시간에 도달하면 대상 그룹의 나머지 모든 디바이스에 대한 롤아웃을 중지합니다. 또한 나머지 작업 실행을 진행하는 방법에 대해 선택한 종료 동작과 재시도 구성에 따른 재시도를 활용합니다.

제한 시간 구성

기본적으로 제한 시간이 없으며 작업 실행은 취소되거나 삭제됩니다. 제한 시간을 사용하려면 제한 시간 활성화를 선택하고 1분에서 7일 사이의 제한 시간 값을 지정합니다.

재시도 구성

> **Note**
>
> 작업이 생성된 후에는 재시도 횟수를 업데이트할 수 없습니다. 모든 실패 유형에 대한 재시도 구성을 제거할 수만 있습니다. 작업을 생성할 때 구성에 사용할 적절한 재시도 횟수를 고려합니다. 잠재적인 재시도 실패로 인해 과도한 비용이 발생하지 않도록 하려면 중단 구성을 추가합니다.

새 구성 추가(Add new configuration)를 선택하고 각 구성에 대해 다음 파라미터를 지정합니다.

- 실패 유형

  작업 실행 재시도를 트리거해야 하는 실패 유형을 지정합니다. 여기에는 실패(Failed), 제한 시간(Timeout) 및 모두(All)가 포함됩니다.
- 재시도 횟수

  선택한 실패 유형(Failure type)에 대한 재시도 횟수를 지정합니다. 두 실패 유형을 합하여 최대 10번의 재시도를 시도할 수 있습니다.

작업 API를 사용하여 AWS IoT 작업 구성 지정

CreateJob 또는 CreateJobTemplate API를 사용하여 다양한 작업 구성을 지정할 수 있습니다. 다음 섹션에서는 이러한 구성을 추가하는 방법을 설명합니다. 구성을 추가한 후 JobExecutionSummary 및 JobExecutionSummaryForJob을 사용하여 상태를 볼 수 있습니다.

다양한 구성 및 구성 작동 방식에 대한 자세한 내용은 작업 구성 작동 방식 섹션을 참조하세요.

롤아웃 구성

롤아웃 구성에 대해 상수 롤아웃 속도 또는 기하급수적 롤아웃 속도를 지정할 수 있습니다.

- 상수 롤아웃 속도 설정

  상수 롤아웃 속도를 설정하려면 `JobExecutionsRolloutConfig` 객체를 사용하여 `maximumPerMinute` 파라미터를 `CreateJob` 요청에 추가합니다. 이 파라미터는 작업 실행이 발생할 수 있는 속도의 상한을 지정합니다. 이 값은 선택 사항이며 범위는 1~1,000입니다. 이 값을 설정하지 않으면 1,000이 기본값으로 사용됩니다.

```
    "jobExecutionsRolloutConfig": {
        "maximumPerMinute": 1000
    }
```

- 기하급수적 롤아웃 속도 설정

  가변 작업 롤아웃 속도를 설정하려면 JobExecutionsRolloutConfig 객체를 사용합니다. CreateJob API 작업을 실행할 때 ExponentialRolloutRate 속성을 구성할 수 있습니다. 다음 예에서는 exponentialRate 파라미터를 사용하여 기하급수적 롤아웃 속도를 설정합니다. 파라미터에 대한 자세한 내용은 ExponentialRolloutRate 섹션을 참조하세요.

```
{
...
  "jobExecutionsRolloutConfig": {
    "exponentialRate": {
      "baseRatePerMinute": 50,
      "incrementFactor": 2,
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": 1000,
        "numberOfSucceededThings": 1000
      },
      "maximumPerMinute": 1000
    }
  }
...
}
```

여기서 파라미터는 다음과 같습니다.

baseRatePerMinute

numberOfNotifiedThings 또는 numberOfSucceededThings 임계값이 충족될 때까지 작업이 실행되는 속도를 지정합니다.

incrementFactor

numberOfNotifiedThings 또는 numberOfSucceededThings 임계값이 충족된 후 롤아웃 속도 증가의 기준이 되는 지수 인자를 지정합니다.

rateIncreaseCriteria

numberOfNotifiedThings 또는 numberOfSucceededThings 임계값을 지정합니다.

Abort configuration(중단 구성)

API를 사용하여 이 구성을 추가하려면 CreateJob 또는 CreateJobTemplate API 작업을 실행할 때 AbortConfig 파라미터를 지정합니다. 다음 예제에서는 CreateJob API 작업에 지정된 대로 여러 번의 실패한 실행이 발생한 작업 롤아웃에 대한 중단 구성을 보여줍니다.

> **Note**
>
> 작업 실행 삭제는 완료된 전체 실행의 계산 값에 영향을 미칩니다. 작업이 중단되면 서비스는 자동화된 comment 및 reasonCode을 생성해 사용자 주도 취소를 작업 중단 취소와 구분합니다.

```
"abortConfig": {
   "criteriaList": [
      {
         "action": "CANCEL",
         "failureType": "FAILED",
         "minNumberOfExecutedThings": 100,
         "thresholdPercentage": 20
      },
      {
         "action": "CANCEL",
         "failureType": "TIMED_OUT",
         "minNumberOfExecutedThings": 200,
         "thresholdPercentage": 50
      }
   ]
}
```

여기서 파라미터는 다음과 같습니다.

action

중단 기준 충족 시 취할 조치를 지정합니다. 이는 필수 파라미터로, CANCEL이 유일하게 유효한 값입니다.

failureType

작업 중단을 시작해야 하는 실패 유형을 지정합니다. 유효한 값은 FAILED, REJECTED, TIMED_OUT 및 ALL입니다.

minNumberOfExecutedThings

작업 중단 기준이 충족되기 전에 발생해야 하는 완료된 작업 실행 수를 지정합니다. 이 예제에서 AWS IoT 는 100개 이상의 디바이스에서 작업 실행을 완료할 때까지 작업 중단이 발생해야 하는지 확인하지 않습니다.

thresholdPercentage

작업 중단을 시작할 수 있는 작업이 실행된 총 사물 수를 지정합니다. 이 예제에서는 순차적으로 를 AWS IoT 확인하고 임계값 백분율이 충족되면 작업 중단을 시작합니다. 100번의 실행이 완료된 후 전체 실행의 20% 이상이 실패하는 경우 작업 롤아웃이 취소됩니다. 이 기준이 충족되지 않으면 AWS IoT 는 완료된 실행의 50% 이상이 200회 실행이 완료된 후 시간 초과되었는지 확인합니다. 이 경우 작업 롤아웃이 취소됩니다.

Scheduling configuration(예약 구성)

API를 사용하여 이 구성을 추가하려면 CreateJob 또는 CreateJobTemplate API 작업을 실행할 때 선택적 SchedulingConfig를 지정합니다.

```
  "SchedulingConfig": {
    "endBehavior": string
    "endTime": string
    "maintenanceWindows": string
    "startTime": string
}
```

여기서 파라미터는 다음과 같습니다.

startTime

작업이 시작될 날짜와 시간을 지정합니다.

endTime

작업이 종료될 날짜와 시간을 지정합니다.

maintenanceWindows

작업 문서를 대상 그룹의 모든 디바이스에 롤아웃하기 위해 예약된 작업에 대해 선택적 유지 관리 기간을 선택했는지를 지정합니다. maintenanceWindow의 문자열 형식은 날짜의 경우 YYYYY/MM/DD이고 시간의 경우 hh:mm입니다.

endBehavior

endTime에 도달할 때 예약된 작업의 작업 동작을 지정합니다.

> **Note**
>
> 작업에 대한 선택적 SchedulingConfig는 DescribeJob 및 DescribeJobTemplate API에서 볼 수 있습니다.

제한 시간 구성

API를 사용하여 이 구성을 추가하려면 CreateJob 또는 CreateJobTemplate API 작업을 실행할 때 TimeoutConfig 파라미터를 지정합니다.

제한 시간 구성을 사용하려면

1. 작업 또는 작업 템플릿을 생성할 때 진행 중 타이머를 설정하려면 선택 사항인 TimeoutConfig 객체의 inProgressTimeoutInMinutes 속성에 값을 설정합니다.

```
    "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
```

2. 작업 실행을 위한 단계 타이머를 지정하려면 UpdateJobExecution을 호출할 때 stepTimeoutInMinutes에 값을 설정합니다. 단계 타이머는 업데이트하는 작업 실행에만 적용됩니다. 작업 실행을 업데이트할 때마다 이 타이머에 대해 새 값을 설정할 수 있습니다.

> **Note**
>
> UpdateJobExecution은 -1 값으로 새 단계 타이머를 생성함으로써 이미 생성된 단계 타이머를 폐기할 수 없습니다.

```
{
   ...
    "statusDetails": {
      "string" : "string"
   },
```

```
    "stepTimeoutInMinutes": number
}
```

3. 새 단계 타이머를 생성하려면 StartNextPendingJobExecution API 작업을 호출할 수도 있습니다.

재시도 구성

> **Note**
>
> 작업을 생성할 때 구성에 사용할 적절한 재시도 횟수를 고려합니다. 잠재적인 재시도 실패로 인해 과도한 비용이 발생하지 않도록 하려면 중단 구성을 추가합니다. 작업이 생성된 후에는 재시도 횟수를 업데이트할 수 없습니다. UpdateJob API 작업을 사용하여 재시도 횟수를 0으로만 설정할 수 있습니다.

API를 사용하여 이 구성을 추가하려면 `CreateJob` 또는 `CreateJobTemplate` API 작업을 실행할 때 `jobExecutionsRetryConfig` 파라미터를 지정합니다.

```
{
...
  "jobExecutionsRetryConfig": {
      "criteriaList": [
         {
            "failureType": "string",
            "numberOfRetries": number
         }
      ]
  }
...
}
```

여기서 criteriaList는 작업의 각 실패 유형에 대해 허용되는 재시도 횟수를 결정하는 기준 목록을 지정하는 배열입니다.

디바이스와 작업

디바이스는 MQTT, HTTP 서명 버전 4 또는 HTTP TLS를 사용하여 AWS IoT작업과 통신할 수 있습니다. 디바이스가 AWS IoT작업과 통신할 때 사용할 엔드포인트를 확인하려면 DescribeEndpoint 명령을 실행합니다. 예를 들면 다음 명령을 실행하는 경우

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

다음과 같은 결과가 출력됩니다.

```
{
    "endpointAddress": "a1b2c3d4e5f6g7-ats.iot.us-west-2.amazonaws.com"
}
```

MQTT 프로토콜 사용

디바이스는 MQTT 프로토콜을 사용하여 AWS IoT작업과 통신할 수 있습니다. 디바이스는 MQTT 주제를 구독하여 새 작업에 대한 알림을 받고 AWS IoT작업 서비스로부터 응답을 받습니다. 또한 MQTT 주제에 게시하여 작업 시작 상태에 대한 쿼리를 실행하거나 업데이트하기도 합니다. 또한 디바이스마다 자체적으로 일반 MQTT 주제가 있습니다. MQTT 주제 게시 및 구독에 대한 자세한 내용은 디바이스 통신 프로토콜 단원을 참조하세요.

이 통신 방법을 사용하면 디바이스가 디바이스별 인증서와 프라이빗 키를 사용하여 AWS IoT작업으로 인증합니다.

디바이스는 다음 주제를 구독할 수 있습니다. `thing-name`은 디바이스와 연결된 사물의 이름입니다.

- **$aws/things/*thing-name*/jobs/notify**

  보류 중인 작업 시작 목록에서 작업 시작이 추가되거나 제거될 때 알림을 받으려면 이 주제를 구독합니다.

- **$aws/things/*thing-name*/jobs/notify-next**

  보류 중인 다음 작업 실행이 변경될 때 알림을 받으려면 이 주제를 구독합니다.

- **$aws/things/*thing-name*/jobs/*request-name*/accepted**

  AWS IoT작업 서비스는 MQTT 주제에 성공 및 실패 메시지를 게시합니다. 이 주제는 요청에 사용된 주제에 `accepted` 또는 `rejected`를 추가해 구성합니다. 여기서 `request-name`는 `Get`와 같은 요청의 이름이며 주제는 `$aws/things/myThing/jobs/get`.AWS IoTJobs가 `$aws/things/myThing/jobs/get/accepted` 주제에 성공 메시지를 게시할 수 있습니다.

- **$aws/things/*thing-name*/jobs/*request-name*/rejected**

  여기서 `request-name`은 `Get`과 같은 요청의 이름입니다. 요청이 실패하면 AWS IoT작업에서 `$aws/things/myThing/jobs/get/rejected` 주제에 실패 메시지를 게시합니다.

다음 HTTPS API 작업을 사용할 수도 있습니다.

- UpdateJobExecution API를 호출하여 작업 실행 상태를 업데이트할 수 있습니다.
- DescribeJobExecution API를 호출하여 작업 실행 상태에 대한 쿼리를 실행할 수 있습니다.
- GetPendingJobExecutions API를 호출하여 대기 중인 작업 실행 목록을 가져올 수 있습니다.
- jobId를 $next로 사용하여 DescribeJobExecution API를 호출하여 다음 보류 중인 작업 실행을 가져올 수 있습니다.
- StartNextPendingJobExecution API를 호출하여 다음 대기 중인 작업 실행을 가져와 시작할 수 있습니다.

HTTP 서명 버전 4 사용

디바이스는 포트 443에서 HTTP 서명 버전 4를 사용하여 AWS IoT작업과 통신할 수 있습니다. 이는 AWS SDK와 CLI에 의해 사용되는 방법입니다. 이러한 도구에 대한 자세한 내용은 AWS CLI 명령 참조: iot-jobs-data 또는 AWS SDK 및 도구와 기본 언어의 IotJobsDataPlane 단원을 참조하세요.

이 통신 방법을 사용하면 디바이스가 IAM 자격 증명을 사용하여 AWS IoT작업으로 인증합니다.

이 방법을 사용하는 경우 다음 명령을 사용할 수 있습니다.

- DescribeJobExecution

```
aws iot-jobs-data describe-job-execution ...
```

- GetPendingJobExecutions

```
aws iot-jobs-data get-pending-job-executions ...
```

- StartNextPendingJobExecution

```
aws iot-jobs-data start-next-pending-job-execution ...
```

- UpdateJobExecution

```
aws iot-jobs-data update-job-execution ...
```

HTTP TLS 사용

디바이스는이 프로토콜을 지원하는 타사 소프트웨어 클라이언트를 사용하여 포트 8443에서 HTTP TLS를 사용하여 AWS IoT작업과 통신할 수 있습니다.

디바이스는 이 방법으로 X.509 인증서 기반 인증을 사용합니다(예: 디바이스별 인증서 및 프라이빗 키).

이 방법을 사용하는 경우 다음 명령을 사용할 수 있습니다.

- DescribeJobExecution
- GetPendingJobExecutions
- StartNextPendingJobExecution
- UpdateJobExecution

작업 서비스와 작업할 수 있도록 디바이스를 프로그래밍

이 섹션의 예제에서는 MQTT를 사용하여 디바이스가 AWS IoT 작업 서비스와 작업하는 방식을 보여줍니다. 또는 해당 API 또는 CLI 명령을 사용할 수 있습니다. 여기 예제에서는 `MyThing` 디바이스가 다음 MQTT 주제를 구독한다고 가정합니다.

- `$aws/things/MyThing/jobs/notify`(또는 `$aws/things/MyThing/jobs/notify-next`)
- `$aws/things/MyThing/jobs/get/accepted`
- `$aws/things/MyThing/jobs/get/rejected`
- `$aws/things/MyThing/jobs/jobId/get/accepted`
- `$aws/things/MyThing/jobs/jobId/get/rejected`

에 코드 서명을 사용하는 경우 AWS IoT디바이스 코드는 코드 파일의 서명을 확인해야 합니다. 이 서명은 작업 문서의 `codesign` 속성에 있습니다. 코드 파일 서명 확인에 대한 자세한 내용은 디바이스 에이전트 샘플을 참조하세요.

주제

- 디바이스 워크플로우
- 작업 워크플로
- 작업 알림

디바이스 워크플로우

디바이스는 다음 방법 중 하나를 사용하여 실행되는 작업을 처리할 수 있습니다.

- 다음 작업 가져오기

  1. 디바이스가 처음 온라인에 연결되면 해당 디바이스의 `notify-next` 주제를 구독해야 합니다.
  2. jobId `$next`와 함께 DescribeJobExecution MQTT API를 호출하고 `statusDetails`에 저장된 모든 상태를 포함하여 다음 작업, 작업 문서 및 기타 세부 정보를 가져옵니다. 작업 문서에 코드 파일 서명이 있는 경우 작업 요청을 진행하기 전에 서명을 확인해야 합니다.
  3. UpdateJobExecution MQTT API를 호출하여 작업 상태를 업데이트합니다. 혹은 이번 단계와 이전 단계를 단일 호출로 결합하고 싶다면 디바이스에서 StartNextPendingJobExecution을 호출해도 좋습니다.
  4. (선택 사항) UpdateJobExecution 또는 StartNextPendingJobExecution를 호출하면 `stepTimeoutInMinutes`에 대한 값을 설정하여 단계 타이머를 추가할 수 있습니다.
  5. 작업 진행 상황에 대해 보고하려면 UpdateJobExecution MQTT API를 사용하여 작업 문서에서 지정한 작업을 실행합니다.
  6. 이 jobId로 DescribeJobExecution MQTT API를 호출하여 작업 실행을 계속 모니터링합니다. 작업 실행이 삭제되면 DescribeJobExecution에서 `ResourceNotFoundException`을 반환합니다.

     디바이스가 작업을 실행하는 동안 작업 실행이 취소되거나 삭제된 경우 디바이스는 유효한 상태로 복구할 수 있어야 합니다.
  7. 작업이 완료되면 UpdateJobExecution MQTT API를 호출하여 작업 상태를 업데이트하고 성공 또는 실패 여부를 보고합니다.
  8. 이번 작업의 실행 상태는 종료 상태로 바뀌었기 때문에 다음에 실행할 수 있는 작업(있는 경우)도 바뀝니다. 그러고 나서 디바이스에게도 다음 대기 중인 작업 실행이 바뀌었다는 알림 메시지가 수신됩니다. 이때는 디바이스가 2단계에서 설명한 대로 계속 진행되어야 합니다.

  디바이스가 계속해서 온라인 상태이면 보류 중인 다음 작업 실행에 대한 알림 메시지가 계속해서 수신됩니다. 여기에는 작업을 완료할 때 또는 보류 중인 작업 실행이 새롭게 추가될 때 작업 실행 데이터가 포함됩니다. 이때는 디바이스가 2단계에서 설명한 대로 계속 진행됩니다.

- 사용 가능한 작업 중에서 선택

  1. 디바이스가 처음 온라인에 연결되면 해당 사물의 `notify` 주제를 구독해야 합니다.
  2. GetPendingJobExecutions MQTT API를 호출하여 대기 중인 작업 실행 목록을 가져옵니다.
  3. 목록에 작업 실행이 다수인 경우에는 하나를 선택합니다.
  4. DescribeJobExecution MQTT API를 호출하고 `statusDetails`에 저장된 모든 상태를 포함하여 작업 문서와 기타 세부 정보를 가져옵니다.

5. UpdateJobExecution MQTT API를 호출하여 작업 상태를 업데이트합니다. 이 명령에서 `includeJobDocument` 필드가 `true`로 설정되어 있으면 디바이스가 이전 단계를 건너뛰고 이번 단계에서 작업 문서를 가져올 수 있습니다.
6. 선택적으로, UpdateJobExecution를 호출하면 `stepTimeoutInMinutes`에 대한 값을 설정하여 단계 타이머를 추가할 수 있습니다.
7. 작업 진행 상황에 대해 보고하려면 UpdateJobExecution MQTT API를 사용하여 작업 문서에서 지정한 작업을 실행합니다.
8. 이 jobId로 DescribeJobExecution MQTT API를 호출하여 작업 실행을 계속 모니터링합니다. 디바이스가 작업을 실행하는 동안 그 작업 실행이 취소되거나 삭제될 경우, 디바이스가 유효한 상태로 복구할 수 있어야 합니다.
9. 작업이 완료되면 UpdateJobExecution MQTT API를 호출하여 작업 상태를 업데이트하고 성공 또는 실패 여부를 보고합니다.

   디바이스가 계속해서 온라인 상태이면 대기 중인 작업 실행이 새롭게 추가될 때 대기 중인 모든 작업 실행에 대해 알림 메시지가 수신됩니다. 이때는 디바이스가 2단계에서 설명한 대로 계속 진행할 수 있습니다.

디바이스가 작업을 수행할 수 없는 경우에는 UpdateJobExecution MQTT API를 호출하여 작업 상태를 `REJECTED`로 업데이트해야 합니다.

작업 워크플로

다음은 새 작업 시작부터 작업 실행의 완료 상태 보고까지 작업 워크플로의 다양한 단계를 보여줍니다.

새 작업 시작

새 작업이 생성되면AWS IoTJobs는 각 대상 디바이스의 `$aws/things/thing-name/jobs/notify` 주제에 메시지를 게시합니다.

메시지에 포함되는 정보는 다음과 같습니다.

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
```

```
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
    }
}
```

작업 실행이 대기 상태가 되면 디바이스가 '$aws/things/*thingName*/jobs/notify' 주제에서 이 메시지를 수신합니다.

> **Note**
>
> 선택적 SchedulingConfig가 있는 작업의 경우 작업이 초기 상태인 SCHEDULED로 유지됩니다. 작업이 선택한 startTime에 도달하면 다음이 발생합니다.
>
> - 작업 상태가 IN_PROGRESS로 업데이트됩니다.
> - 작업이 대상 그룹의 모든 디바이스에 대한 작업 문서 롤아웃을 시작합니다.

작업 정보 가져오기

작업 실행에 대해 더 많은 정보를 가져오려면 디바이스에서 includeJobDocument 필드를 true(기본값)로 설정하여 DescribeJobExecution MQTT API를 호출합니다.

요청이 성공하면 AWS IoT작업 서비스가 $aws/things/MyThing/jobs/0023/get/accepted 주제에 메시지를 게시합니다.

```
{
    "clientToken" : "client-001",
    "timestamp" : 1489097434407,
    "execution" : {
        "approximateSecondsBeforeTimedOut": number,
        "jobId" : "023",
        "status" : "QUEUED",
        "queuedAt" : 1489097374841,
        "lastUpdatedAt" : 1489097374841,
        "versionNumber" : 1,
        "jobDocument" : {
            < contents of job document >
        }
    }
}
```

요청이 실패하면 AWS IoT작업 서비스가 $aws/things/MyThing/jobs/0023/get/rejected 주제에 메시지를 게시합니다.

이제 디바이스에 작업 문서가 있으므로 원격 작업을 실행하는 데 사용할 수 있습니다. 작업 문서에 미리 서명된 Amazon S3 URL이 포함되어 있으면 디바이스가 해당 URL을 사용하여 작업에 필요한 파일을 다운로드할 수 있습니다.

작업 실행 상태 보고

디바이스가 작업을 실행 중이면 UpdateJobExecution MQTT API를 호출하여 작업 실행 상태를 업데이트할 수 있습니다.

예를 들어 디바이스가 다음 메시지를 $aws/things/MyThing/jobs/0023/update 주제에 게시하여 작업 실행 상태를 IN_PROGRESS로 업데이트할 수 있습니다.

```
{
    "status":"IN_PROGRESS",
    "statusDetails": {
        "progress":"50%"
    },
    "expectedVersion":"1",
    "clientToken":"client001"
}
```

그러면 작업이 아래와 같이 $aws/things/MyThing/jobs/0023/update/accepted 또는 $aws/things/MyThing/jobs/0023/update/rejected 주제에 메시지를 게시하여 응답합니다.

```
{
    "clientToken":"client001",
    "timestamp":1476289222841
}
```

디바이스에서는 StartNextPendingJobExecution을 호출해 앞의 두 요청을 결합할 수 있습니다. 그러면 다음 대기 중인 다음 작업 실행을 가져와서 시작하고 디바이스는 작업 실행 상태를 업데이트할 수 있게 됩니다. 또한 대기 중인 작업 실행이 있을 때는 이 요청으로 작업 문서가 반환됩니다.

작업에 TimeoutConfig가 포함된 경우 진행 중 타이머가 실행을 시작합니다. UpdateJobExecution을 호출할 때 stepTimeoutInMinutes 값을 설정하여 작업 실행을 위한 단계 타이머를 설정할 수도 있습니다. 단계 타이머는 업데이트하는 작업 실행에만 적용됩니다. 작업 실행을 업데이트할 때마다 이 타이머에 대해 새 값을 설정할 수 있습니다. StartNextPendingJobExecution를 호출하는 경우에도 단계 타이머를 생성할 수 있습니다. 작업 실행이 단계 타이머 간격보다 오랫동안 IN_PROGRESS 상태를 유지

하는 경우, 해당 작업 실행은 실패하며 터미널 상태인 TIMED_OUT으로 전환됩니다. 단계 타이머는 작업을 생성할 때 설정한 진행 중 타이머에는 영향을 주지 않습니다.

status 필드는 IN_PROGRESS, SUCCEEDED 또는 FAILED로 설정할 수도 있습니다. 이미 종료 상태인 작업 실행의 상태는 업데이트할 수 없습니다.

보고서 실행 완료

디바이스가 작업 실행을 마치면 UpdateJobExecution MQTT API를 호출합니다. 작업이 성공적으로 완료된 경우에는 status를 SUCCEEDED로 설정하고, 메시지 페이로드의 statusDetails에 작업에 대한 다른 정보를 이름-값 페어로 추가합니다. 진행 중 타이머와 단계 타이머는 작업 실행이 완료되면 끝납니다.

다음 예를 참조하세요.

```
{
    "status":"SUCCEEDED",
    "statusDetails": {
        "progress":"100%"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

작업이 성공적으로 완료되지 않은 경우에는 status를 FAILED로 설정하고, statusDetails에 아래와 같이 발생한 오류에 대한 정보를 추가합니다.

```
{
    "status":"FAILED",
    "statusDetails": {
        "errorCode":"101",
        "errorMsg":"Unable to install update"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

> **Note**
>
> statusDetails 속성에는 이름-값 페어가 몇 개든지 포함될 수 있습니다.

AWS IoT작업 서비스가이 업데이트를 수신하면 $aws/things/MyThing/jobs/notify 주제에 작업 실행이 완료되었음을 나타내는 메시지를 게시합니다.

```
{
    "timestamp":1476290692776,
    "jobs":{}
}
```

추가 작업

디바이스에 대기 중인 작업 실행이 더 있으면 $aws/things/MyThing/jobs/notify에 게시되는 메시지에 해당 작업이 포함됩니다.

다음 예를 참조하세요.

```
{
    "timestamp":1476290692776,
    "jobs":{
        "QUEUED":[{
            "jobId":"0002",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }],
        "IN_PROGRESS":[{
            "jobId":"0003",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }]
    }
}
```

작업 알림

AWS IoT작업 서비스는 작업이 보류 중이거나 목록의 첫 번째 작업 실행이 변경될 때 예약된 주제에 MQTT 메시지를 게시합니다. 디바이스는 이러한 주제를 구독하여 보류 작업을 추적할 수 있습니다.

작업 알림 유형

작업 알림은 JSON 페이로드로서 MQTT 주제로 게시됩니다. 두 가지 종류의 알림이 있습니다.

ListNotification

ListNotification에는 최대 15개의 보류 중인 작업 실행의 목록이 포함되어 있습니다. 이들은 상태(QUEUED 작업 실행 전의 IN_PROGRESS 작업 실행)에 따라 정렬된 후 대기열에 있었던 시간에 따라 정렬됩니다.

ListNotification은(는) 다음 기준 중 하나가 충족될 때마다 게시됩니다.

- 새로운 작업 실행이 대기 중이거나 비단말 상태(IN_PROGRESS 또는 QUEUED)로 변경되었습니다.
- 기존 작업 실행이 터미널 상태(FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED 또는 REMOVED)로 변경됩니다.

예약 구성을 사용하거나 사용하지 않는 제한에 대한 자세한 내용은 작업 실행 한도 섹션을 참조하세요.

NextNotification

- NextNotification에는 대기열에서 다음 순서인 작업 실행 하나에 대한 요약 정보가 포함되어 있습니다.

  NextNotification은(는) 목록 상의 첫 번째 작업 실행에 변경이 있을 때마다 게시됩니다.
  - 새로운 작업 실행은 QUEUED 상태로 목록에 추가되어 목록상의 첫 번째 작업 실행이 됩니다.
  - 목록에서 첫 번째가 아닌 기존 작업 실행의 상태는 QUEUED에서 IN_PROGRESS로 변경되어 목록 상의 첫 번째 작업 실행이 됩니다. (이런 경우는 목록에 다른 IN_PROGRESS작업 실행이 없거나 상태가 QUEUED에서 IN_PROGRESS(으)로 변경된 작업 실행이 목록의 다른 IN_PROGRESS 작업 실행보다 먼저 대기열에 오른 경우에 발생합니다.)
  - 목록에서 첫 번째인 작업 실행의 상태는 단말 상태로 변경되어 목록에서 제거됩니다.

MQTT 주제 게시 및 구독에 대한 자세한 내용은 the section called “디바이스 통신 프로토콜” 단원을 참조하세요.

> **Note**
>
> HTTP 서명 버전 4 또는 HTTP TLS를 사용하여 이 작업과 통신할 경우 알림을 사용할 수 없습니다.

작업 보류 중

AWS IoT작업 서비스는 사물에 대해 보류 중인 작업 실행 목록에 작업이 추가되거나 목록에서 제거되거나 목록의 첫 번째 작업 실행이 변경될 때 MQTT 주제에 메시지를 게시합니다.

- $aws/things/*thingName*/jobs/notify
- $aws/things/*thingName*/jobs/notify-next

메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

$aws/things/*thingName*/jobs/notify:

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : [ {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0
    } ]
  }
}
```

this-job이라는 작업 실행이 선택적 예약 구성을 선택한 작업에서 시작되고 작업 문서 롤아웃이 유지 관리 기간 중에 수행되도록 예약된 경우 반복 유지 관리 기간 중에만 표시됩니다. 다음 예와 같이 유지 관리 기간 외에는 this-job이라는 작업이 보류 중인 작업 실행 목록에서 제외됩니다.

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : []
```

```
  }
}
```

$aws/things/*thingName*/jobs/notify-next:

```
{
  "timestamp" : 10011,
  "execution" : {
    "jobId" : "other-job",
    "status" : "IN_PROGRESS",
    "queuedAt" : 10009,
    "lastUpdatedAt" : 10009,
    "versionNumber" : 1,
    "executionNumber" : 1,
    "jobDocument" : {"c":"d"}
  }
}
```

other-job이라는 작업 실행이 선택적 예약 구성을 선택한 작업에서 시작되고 작업 문서 롤아웃이 유지 관리 기간 중에 수행되도록 예약된 경우 반복 유지 관리 기간 중에만 표시됩니다. 다음 예와 같이 유지 관리 기간 외에는 other-job이라는 작업이 다음 작업 실행으로 나열되지 않습니다.

```
{} //No other pending jobs
```

```
{
  "timestamp" : 10011,
  "execution" : {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0,
      "jobDocument" : {"a":"b"}
  }
} // "this-job" is pending next to "other-job"
```

가능한 작업 실행 상태 값은 QUEUED, IN_PROGRESS, FAILED, SUCCEEDED, CANCELED, TIMED_OUT, REJECTED, REMOVED입니다.

다음의 여러 가지 예들은 작업 실행이 생성되어 한 상태에서 다른 상태로 바뀔 때 각 주제로 게시되는 알림을 보여줍니다.

우선 job1이라는 작업이 하나 생성됩니다. 이 알림은 jobs/notify 주제로 게시됩니다.

```
{
  "timestamp": 1517016948,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

이 알림은 jobs/notify-next 주제로 게시됩니다.

```
{
  "timestamp": 1517016948,
  "execution": {
    "jobId": "job1",
    "status": "QUEUED",
    "queuedAt": 1517016947,
    "lastUpdatedAt": 1517016947,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

다른 작업(job2)을 생성하면, 이 알림이 jobs/notify 주제로 게시됩니다.

```
{
  "timestamp": 1517017192,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
```

```
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

대기열에 있는 다음 작업(job1)이 변경되지 않았으므로 알림이 jobs/notify-next 주제에 게시되지 않습니다. job1이 실행되기 시작하면 그 상태가 IN_PROGRESS로 바뀝니다. 작업 목록과 대기열의 다음 작업이 변경되지 않았으므로 알림이 게시되지 않습니다.

세 번째 작업(job3)이 추가되면, 이 알림이 jobs/notify 주제로 게시됩니다.

```
{
  "timestamp": 1517017906,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517017472,
        "startedAt": 1517017472,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ],
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
```

```
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

대기열에 있는 다음 작업이 아직 job1이므로 알림이 jobs/notify-next 주제에 게시되지 않습니다.

job1이 완료되면 그 상태가 SUCCEEDED로 변경되고 이 알림이 jobs/notify 주제로 게시됩니다.

```
{
  "timestamp": 1517186269,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

이 시점에 job1은 대기열에서 제거되었고 실행할 다음 작업은 job2입니다. 이 알림은 jobs/notify-next 주제로 게시됩니다.

```
{
```

```
  "timestamp": 1517186269,
  "execution": {
    "jobId": "job2",
    "status": "QUEUED",
    "queuedAt": 1517017191,
    "lastUpdatedAt": 1517017191,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

job3가 job2보다 먼저 실행되어야 하는 경우(권장되지 않음), job3의 상태를 IN_PROGRESS로 변경할 수 있습니다. 이렇게 하면, job2는 대기열에서 더 이상 다음 작업이 아니고 이 알림이 jobs/notify-next 주제로 게시됩니다.

```
{
  "timestamp": 1517186779,
  "execution": {
    "jobId": "job3",
    "status": "IN_PROGRESS",
    "queuedAt": 1517017905,
    "startedAt": 1517186779,
    "lastUpdatedAt": 1517186779,
    "versionNumber": 2,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

아무 작업도 추가되거나 제거되지 않았으므로 jobs/notify 주제로 아무 알림도 게시되지 않습니다.

디바이스에서 job2를 거부하고 그 상태를 REJECTED로 변경하면 이 알림이 jobs/notify 주제로 게시됩니다.

```
{
  "timestamp": 1517189392,
  "jobs": {
```

```
    "IN_PROGRESS": [
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517186779,
        "startedAt": 1517186779,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ]
  }
}
```

(아직 진행 중인) job3가 강제로 삭제되면, 이 알림이 jobs/notify 주제로 게시됩니다.

```
{
  "timestamp": 1517189551,
  "jobs": {}
}
```

이 시점에 대기열은 비어 있습니다. 이 알림은 jobs/notify-next 주제로 게시됩니다.

```
{
  "timestamp": 1517189551
}
```

AWS IoT 작업 API 작업

AWS IoT 작업 API는 다음 범주 중 하나에 사용할 수 있습니다.

- 작업 관리 및 제어와 같은 관리 작업. 이것은 제어 영역입니다.
- 이러한 작업을 수행하는 디바이스. 이것은 데이터 영역으로, 데이터를 전송하고 수신하도록 허용합니다.

작업을 관리하고 제어할 때는 HTTPS 프로토콜 API를 사용합니다. 디바이스는 MQTT 또는 HTTPS 프로토콜 API 중에서 하나를 사용할 수 있습니다. 제어 영역 API는 일반적으로 작업을 생성하거나 추적할 때 적은 용량의 호출에 맞게 설계되었습니다. 그래서 단일 요청일 때 연결을 개시하고, 응답이 수신되면 연결을 종료합니다. 데이터 영역 HTTPS 및 MQTT API는 긴 폴링을 허용합니다. 이 API 작업들은 수백만 개의 디바이스까지 확장되는 대용량 트래픽에 맞게 설계되었습니다.

각 AWS IoT 작업 HTTPS API에는 AWS Command Line Interface ()에서 API를 호출할 수 있는 해당 명령이 있습니다AWS CLI. 이러한 명령들은 소문자이며, API 이름을 구성하는 단어들 사이에 하이픈(-)을 사용합니다. 예를 들어 CLI에서 다음과 같이 입력하면 `CreateJob` API를 호출할 수 있습니다.

```
aws iot create-job ...
```

작업 중 오류가 발생하면 오류에 대한 정보가 포함된 오류 응답을 받습니다.

ErrorResponse

AWS IoT 작업 서비스 작업에서 발생한 오류에 대한 정보가 포함됩니다.

다음 예는 이 작업의 구문을 보여줍니다.

```
{
    "code": "ErrorCode",
    "message": "string",
    "clientToken": "string",
    "timestamp": timestamp,
    "executionState": JobExecutionState
}
```

다음은 이 `ErrorResponse`에 대한 설명입니다.

`code`

ErrorCode는 다음과 같이 설정할 수 있습니다.

InvalidTopic

요청이 API 작업에 매핑되지 않는 AWS IoT 작업 네임스페이스의 주제로 전송되었습니다.

InvalidJson

요청 내용이 유효한 UTF-8-인코딩 JSON으로 해석되지 않았습니다.

InvalidRequest

요청 내용이 잘못되었습니다. 예를 들어 `UpdateJobExecution` 요청에 잘못된 상태 세부 정보가 포함되어 있으면 이 코드가 반환됩니다. 메시지에는 오류에 대한 세부 정보가 포함됩니다.

InvalidStateTransition

업데이트가 작업 실행의 현재 상태 때문에 유효하지 않은 상태로 작업 실행을 변경하려고 했습니다. 예를 들어, SUCCEEDED 상태의 요청을 IN_PROGRESS로 변경하려고 합니다. 이때는 오류 메시지 본문에 `executionState` 필드도 포함됩니다.

ResourceNotFound

요청 주제에서 지정한 `JobExecution`이 존재하지 않습니다.

VersionMismatch

요청에 지정된 예상 버전이 작업 AWS IoT 서비스의 작업 실행 버전과 일치하지 않습니다. 이때는 오류 메시지 본문에 `executionState` 필드도 포함됩니다.

InternalError

요청을 처리하는 도중 내부 오류가 발생했습니다.

RequestThrottled

요청에 병목 현상이 발생했습니다.

TerminalStateReached

종료 상태의 작업에서 작업을 설명하는 명령을 실행했을 때 발생합니다.

`message`

오류 메시지 문자열입니다.

`clientToken`

요청과 응답의 연관성을 나타내는 임의 문자열입니다.

`timestamp`

epoch 이후 경과 시간(초)입니다.

`executionState`

`JobExecutionState` 객체입니다. 이 필드는 `code` 필드에 `InvalidStateTransition` 또는 `VersionMismatch` 값이 있을 때만 포함됩니다. 이 두 가지 경우에는 `DescribeJobExecution` 요청을 별도로 실행하여 현재 작업 실행 상태에 대한 데이터를 가져올 필요가 없습니다.

다음은 Jobs API 작업 및 데이터 유형을 나열합니다.

- 작업 관리 및 제어 API 및 데이터 형식
- 작업 디바이스 MQTT API 및 HTTPS API 작업 및 데이터 형식

작업 관리 및 제어 API 및 데이터 형식

다음은 작업 관리 및 제어에서 CLI 및 HTTPS 프로토콜을 통해 사용할 수 있는 명령입니다.

- 작업 관리 및 제어 데이터 형식
- 작업 관리 및 제어 API 작업

CLI 명령에 대한 *endpoint-url* 파라미터를 확인하려면 이 명령을 실행합니다.

```
aws iot describe-endpoint --endpoint-type=iot:Jobs
```

이 명령은 다음 출력을 반환합니다.

```
{
"endpointAddress": "account-specific-prefix.jobs.iot.aws-region.amazonaws.com"
}
```

> **Note**
>
> 작업 엔드포인트는 ALPN x-amzn-http-ca를 지원하지 않습니다.
> 듀얼 스택 엔드포인트(IPv6와 IPv6)를 사용하는 경우 iot:Data-ATS 엔드포인트를 사용합니다. iot:Jobs 엔드포인트는 IPv4만 지원합니다.

작업 관리 및 제어 데이터 형식

다음 데이터 형식은 관리 및 제어 애플리케이션에서 AWS IoT 작업과 통신하는 데 사용됩니다.

작업

Job 객체에는 작업에 대한 세부 정보가 포함됩니다. 다음 예제에서는 구문을 보여줍니다.

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED",
    "forceCanceled": boolean,
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "comment": "string",
```

```
    "targets": ["string"],
    "description": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp,
    "jobProcessDetails": {
        "processingTargets": ["string"],
        "numberOfCanceledThings": long,
        "numberOfSucceededThings": long,
        "numberOfFailedThings": long,
        "numberOfRejectedThings": long,
        "numberOfQueuedThings": long,
        "numberOfInProgressThings": long,
        "numberOfRemovedThings": long,
        "numberOfTimedOutThings": long
    },
    "presignedUrlConfig": {
        "expiresInSec": number,
        "roleArn": "string"
    },
    "jobExecutionsRolloutConfig": {
        "exponentialRate": {
           "baseRatePerMinute": integer,
           "incrementFactor": integer,
           "rateIncreaseCriteria": {
              "numberOfNotifiedThings": integer, // Set one or the other
              "numberOfSucceededThings": integer // of these two values.
           },
           "maximumPerMinute": integer
      }
    },
    "abortConfig": {
       "criteriaList": [
          {
             "action": "string",
             "failureType": "string",
             "minNumberOfExecutedThings": integer,
             "thresholdPercentage": integer
          }
       ]
    },
    "SchedulingConfig": {
      "startTime": string
      "endTime": string
```

```
      "timeZone": string


      "endTimeBehavior": string

   },
    "timeoutConfig": {
        "inProgressTimeoutInMinutes": long
    }
}
```

자세한 내용은 Job 또는 job 섹션을 참조하세요.

JobSummary

JobSummary 객체에는 작업 요약이 포함됩니다. 다음 예제에서는 구문을 보여줍니다.

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED|SCHEDULED",
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "thingGroupId": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp
}
```

자세한 내용은 JobSummary 또는 job-summary 섹션을 참조하세요.

JobExecution

JobExecution 객체는 디바이스의 작업 실행을 나타냅니다. 다음 예제에서는 구문을 보여줍니다.

> **Note**
>
> 컨트롤 플레인 API 작업을 사용하는 경우 JobExecution 데이터 유형에는 필드가 JobDocument 포함되지 않습니다. 이 정보를 얻으려면 GetJobDocument API 작업 또는 get-job-document CLI 명령을 사용할 수 있습니다.

```
{
```

```
    "approximateSecondsBeforeTimedOut": 50,
    "executionNumber": 1234567890,
    "forceCanceled": true|false,
    "jobId": "string",
    "lastUpdatedAt": timestamp,
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "forceCanceled": boolean,
    "statusDetails": {
        "detailsMap": {
            "string": "string" ...
        },
        "status": "string"
    },
    "thingArn": "string",
    "versionNumber": 123
}
```

자세한 내용은 JobExecution 또는 job-execution 섹션을 참조하세요.

JobExecutionSummary

JobExecutionSummary 객체에는 작업 실행 요약 정보가 포함됩니다. 다음 예제에서는 구문을 보여줍니다.

```
{
    "executionNumber": 1234567890,
    "queuedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|REMOVED"
}
```

자세한 내용은 JobExecutionSummary 또는 job-execution-summary 섹션을 참조하세요.

JobExecutionSummaryForJob

JobExecutionSummaryForJob 객체에는 특정 작업에서 작업 실행에 대한 요약 정보가 포함됩니다. 다음 예제에서는 구문을 보여줍니다.

```
{
```

```
    "executionSummaries": [
        {
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyThing",
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            }
        },
        ...
    ]
}
```

자세한 내용은 JobExecutionSummaryForJob 또는 job-execution-summary-for-job 섹션을 참조하세요.

JobExecutionSummaryForThing

JobExecutionSummaryForThing 객체에는 특정 사물의 작업 실행에 대한 요약 정보가 포함됩니다. 다음 예시에서는 구문을 보여줍니다.

```
{
    "executionSummaries": [
        {
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            },
            "jobId": "MyThingJob"
        },
        ...
    ]
}
```

자세한 내용은 JobExecutionSummaryForThing 또는 job-execution-summary-for-thing 섹션을 참조하세요.

작업 관리 및 제어 API 작업

다음 API 작업 또는 CLI 명령을 사용합니다.

AssociateTargetsWithJob

그룹을 연속 작업과 연결합니다. 단, 다음 기준을 만족해야 합니다.

- 생성된 작업에서 `targetSelection` 필드가 `CONTINUOUS`로 설정되어 있어야 합니다.
- 작업 상태가 현재 `IN_PROGRESS`이어야 합니다.
- 작업에 연결되는 대상의 총 수가 100을 넘어서는 안 됩니다.

HTTPS request

```
POST /jobs/jobId/targets

{
"targets": [ "string" ],
"comment": "string"
}
```

자세한 내용은 AssociateTargetsWithJob 단원을 참조하십시오.

CLI syntax

```
aws iot  associate-targets-with-job \
--targets <value> \
--job-id <value> \
[--comment <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"targets": [
"string"
],
"jobId": "string",
"comment": "string"
}
```

자세한 내용은 associate-targets-with-job 단원을 참조하십시오.

CancelJob

작업을 취소합니다.

HTTPS request

```
PUT /jobs/jobId/cancel

{
"force": boolean,
"comment": "string",
"reasonCode": "string"
}
```

자세한 내용은 CancelJob 단원을 참조하십시오.

CLI syntax

```
aws iot cancel-job \
    --job-id <value> \
    [--force <value>]  \
    [--comment <value>]  \
    [--reasonCode <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
    "jobId": "string",
    "force": boolean,
    "comment": "string"
}
```

자세한 내용은 cancel-job 단원을 참조하십시오.

CancelJobExecution

디바이스의 작업 실행을 취소합니다.

HTTPS request

```
PUT /things/thingName/jobs/jobId/cancel

{
"force": boolean,
"expectedVersion": "string",
"statusDetails": {
    "string": "string"
    ...
}
}
```

자세한 내용은 CancelJobExecution 단원을 참조하십시오.

CLI syntax

```
aws iot cancel-job-execution \
--job-id <value> \
--thing-name <value> \
[--force | --no-force] \
[--expected-version <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string",
"thingName": "string",
"force": boolean,
"expectedVersion": long,
"statusDetails": {
"string": "string"
}
}
```

자세한 내용은 cancel-job-execution 단원을 참조하십시오.

CreateJob

작업을 생성합니다. 작업 문서는 Amazon S3 버킷(documentSource 파라미터)에서 또는 요청 본문(document 파라미터)에서 파일에 대한 링크로 제공될 수 있습니다.

선택적 targetSelection 파라미터를 CONTINUOUS(기본값은 SNAPSHOT)로 설정하여 작업을 연속 작업으로 만들 수 있습니다. 연속 작업은 계속 실행되고 새로 추가된 사물에서 시작되므로 그룹에 추가된 디바이스를 온보딩하거나 업그레이드할 때 사용할 수 있습니다. 이는 작업을 생성할 때 그룹에 있던 사물이 작업을 완료한 후에도 발생할 수 있습니다.

작업은 진행 중 타이머의 값을 설정하는 선택 사항인 TimeoutConfig를 포함할 수 있습니다. 진행 중 타이머는 업데이트될 수 없으며 이 작업에 대한 모든 작업 실행에 적용됩니다.

다음은 CreateJob API에 대한 인수에서 실행하는 확인 작업입니다.

- targets 인수는 유효한 사물 또는 사물 그룹 ARN의 목록이어야 합니다. 모든 사물 및 사물 그룹은 에 있어야 합니다 AWS 계정.
- documentSource 인수는 작업 문서에 대해 유효한 Amazon S3 URL이어야 합니다. Amazon S3 URL의 형식은 https://s3.amazonaws.com/*bucketName*/*objectName*과 같습니다.
- documentSource 인수에서 지정하는 URL에 저장된 문서는 UTF-8 인코딩 JSON 문서이어야 합니다.
- 작업 문서의 크기는 MQTT 메시지 크기(128KB) 및 암호화에 대한 제한으로 인해 32KB를 넘을 수 없습니다.
- 는에서 고유해야 jobId 합니다 AWS 계정.

HTTPS request

```
PUT /jobs/jobId

{
"targets": [ "string" ],
"document": "string",
"documentSource": "string",
"description": "string",
"jobTemplateArn": "string",
"presignedUrlConfigData": {
    "roleArn": "string",
    "expiresInSec": "integer"
},
"targetSelection": "CONTINUOUS|SNAPSHOT",
```

```
"jobExecutionsRolloutConfig": {
    "exponentialRate": {
       "baseRatePerMinute": integer,
       "incrementFactor": integer,
       "rateIncreaseCriteria": {
          "numberOfNotifiedThings": integer, // Set one or the other
          "numberOfSucceededThings": integer // of these two values.
       },
       "maximumPerMinute": integer
  }
},
"abortConfig": {
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": integer,
         "thresholdPercentage": integer
      }
   ]
},
"SchedulingConfig": {
    "startTime": string
    "endTime": string
    "timeZone": string


    "endTimeBehavior": string

   }
"timeoutConfig": {
  "inProgressTimeoutInMinutes": long
}
}
```

자세한 내용은 CreateJob 단원을 참조하십시오.

CLI syntax

```
aws iot create-job \
    --job-id <value> \
    --targets <value> \
    [--document-source <value>] \
    [--document <value>] \
```

```
    [--description <value>] \
    [--job-template-arn <value>] \
    [--presigned-url-config <value>] \
    [--target-selection <value>] \
    [--job-executions-rollout-config <value>] \
    [--abort-config <value>] \
    [--timeout-config <value>] \
    [--document-parameters <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

cli-input-json 형식:

```
{
    "jobId": "string",
    "targets": [ "string" ],
    "documentSource": "string",
    "document": "string",
    "description": "string",
    "jobTemplateArn": "string",
    "presignedUrlConfig": {
        "roleArn": "string",
        "expiresInSec": long
     },
    "targetSelection": "string",
    "jobExecutionsRolloutConfig": {
          "exponentialRate": {
              "baseRatePerMinute": integer,
              "incrementFactor": integer,
              "rateIncreaseCriteria": {
                 "numberOfNotifiedThings": integer, // Set one or the other
                 "numberOfSucceededThings": integer // of these two values.
              },
      "maximumPerMinute": integer
      }
    },
    "abortConfig": {
    "criteriaList": [
        {
           "action": "string",
           "failureType": "string",
           "minNumberOfExecutedThings": integer,
           "thresholdPercentage": integer
```

```
        }
      ]
    },
    "timeoutConfig": {
          "inProgressTimeoutInMinutes": long
    },
    "documentParameters": {
    "string": "string"
    }
}
```

자세한 내용은 create-job 단원을 참조하십시오.

DeleteJob

특정 작업과 그와 관련한 작업 실행을 삭제합니다.

작업을 삭제하려면 그 작업에 대해 생성된 작업 실행의 수와 기타 여러 가지 요인에 따라 시간이 걸릴 수 있습니다. 작업이 삭제되는 동안 그 작업의 상태는 "DELETION_IN_PROGRESS"로 표시됩니다. 상태가 이미 "DELETION_IN_PROGRESS"인 작업을 삭제하거나 취소하려고 하면 오류가 발생합니다.

HTTPS request

```
DELETE /jobs/jobId?force=force
```

자세한 내용은 DeleteJob 단원을 참조하십시오.

CLI syntax

```
aws iot  delete-job \
--job-id <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string",
"force": boolean
}
```

자세한 내용은 delete-job 단원을 참조하십시오.

DeleteJobExecution

작업 실행을 삭제합니다.

HTTPS request

```
DELETE /things/thingName/jobs/jobId/executionNumber/executionNumber?force=force
```

자세한 내용은 DeleteJobExecution 단원을 참조하십시오.

CLI syntax

```
aws iot  delete-job-execution \
--job-id <value> \
--thing-name <value> \
--execution-number <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long,
"force": boolean
}
```

자세한 내용은 delete-job-execution 단원을 참조하십시오.

DescribeJob

작업 실행에 대한 세부 정보를 가져옵니다.

HTTPS request

```
GET /jobs/jobId
```

자세한 내용은 DescribeJob 단원을 참조하십시오.

CLI syntax

```
aws iot describe-job \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 형식:

```
{
"jobId": "string"
}
```

자세한 내용은 describe-job 단원을 참조하십시오.

DescribeJobExecution

작업 실행에 대한 세부 정보를 가져옵니다. 작업 실행 상태는 SUCCEEDED 또는 FAILED가 되어야 합니다.

HTTPS request

```
GET /things/thingName/jobs/jobId?executionNumber=executionNumber
```

자세한 내용은 DescribeJobExecution 단원을 참조하십시오.

CLI syntax

```
aws iot  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 형식:

```
{
"jobId": "string",
"thingName": "string",
```

```
"executionNumber": long
}
```

자세한 내용은 describe-job-execution 단원을 참조하십시오.

GetJobDocument

임의 작업에 대한 작업 문서를 가져옵니다.

> **Note**
>
> 반환되는 문서에서는 자리 표시자 URL이 미리 서명된 Amazon S3 URL로 바뀌지 않습니다. 미리 서명된 URLs은 AWS IoT Jobs 서비스가 MQTT를 통해 요청을 수신하는 경우에만 생성됩니다.

HTTPS request

```
GET /jobs/jobId/job-document
```

자세한 내용은 GetJobDocument 단원을 참조하십시오.

CLI syntax

```
aws iot get-job-document \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string"
}
```

자세한 내용은 get-job-document 단원을 참조하십시오.

ListJobExecutionsForJob

임의 작업에 대한 작업 실행 목록을 가져옵니다.

HTTPS request

```
GET /jobs/jobId/things?status=status&maxResults=maxResults&nextToken=nextToken
```

자세한 내용은 ListJobExecutionsForJob 단원을 참조하십시오.

CLI syntax

```
aws iot  list-job-executions-for-job \
--job-id <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

자세한 내용은 list-job-executions-for-job 단원을 참조하십시오.

ListJobExecutionsForThing

사물의 작업 실행 목록을 가져옵니다.

HTTPS request

```
GET /things/thingName/jobs?status=status&maxResults=maxResults&nextToken=nextToken
```

자세한 내용은 ListJobExecutionsForThing 단원을 참조하십시오.

CLI syntax

```
aws iot list-job-executions-for-thing \
--thing-name <value> \
[--status <value>] \
```

```
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"thingName": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

자세한 내용은 `list-job-executions-for-thing` 단원을 참조하십시오.

ListJobs

에서 작업 목록을 가져옵니다 AWS 계정.

HTTPS request

```
GET /jobs?
status=status&targetSelection=targetSelection&thingGroupName=thingGroupName&thingGroupId=thi
```

자세한 내용은 `ListJobs` 단원을 참조하십시오.

CLI syntax

```
aws iot list-jobs \
[--status <value>] \
[--target-selection <value>] \
[--max-results <value>] \
[--next-token <value>] \
[--thing-group-name <value>] \
[--thing-group-id <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
```

```
"status": "string",
"targetSelection": "string",
"maxResults": "integer",
"nextToken": "string",
"thingGroupName": "string",
"thingGroupId": "string"
}
```

자세한 내용은 list-jobs 단원을 참조하십시오.

UpdateJob

지정된 작업의 지원되는 필드를 업데이트합니다. timeoutConfig에 대해 업데이트된 값은 새로 진행 중인 시작 작업에만 적용됩니다. 현재 진행 중인 시작은 이전의 제한 시간 구성으로 계속 실행됩니다.

HTTPS request

```
PATCH /jobs/jobId
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     },
  "maximumPerMinute": number
  },
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
```

```
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

자세한 내용은 UpdateJob 단원을 참조하십시오.

CLI syntax

```
aws iot  update-job \
--job-id <value> \
[--description <value>] \
[--presigned-url-config <value>] \
[--job-executions-rollout-config <value>] \
[--abort-config <value>] \
[--timeout-config <value>] \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     }
  },
  "maximumPerMinute": number
},
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
```

```
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
      }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

자세한 내용은 update-job 단원을 참조하십시오.

작업 디바이스 MQTT API 및 HTTPS API 작업 및 데이터 형식

다음은 MQTT 및 HTTPS 프로토콜을 통해 사용할 수 있는 명령입니다. 작업을 실행하는 디바이스의 데이터 영역에서 이러한 API 작업을 사용합니다.

작업 디바이스 MQTT 및 HTTPS 데이터 형식

다음 데이터 형식은 MQTT 및 HTTPS 프로토콜을 통해 AWS IoT 작업 서비스와 통신하는 데 사용됩니다.

JobExecution

`JobExecution` 객체는 디바이스의 작업 실행을 나타냅니다. 다음 예제에서는 구문을 보여줍니다.

> **Note**
>
> MQTT 및 HTTP 데이터 영역 API 작업을 사용하는 경우 `JobExecution` 데이터 유형에 `JobDocument` 필드가 포함됩니다. 디바이스는 이 정보를 사용하여 작업 실행에서 작업 문서를 검색할 수 있습니다.

```
{
    "jobId" : "string",
    "thingName" : "string",
    "jobDocument" : "string",
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
```

```
        "string": "string"
    },
    "queuedAt" : "timestamp",
    "startedAt" : "timestamp",
    "lastUpdatedAt" : "timestamp",
    "versionNumber" : "number",
    "executionNumber": long
}
```

자세한 내용은 JobExecution 또는 job-execution 섹션을 참조하세요.

JobExecutionState

JobExecutionState에는 작업 실행 상태에 대한 정보가 들어 있습니다. 다음 예제에서는 구문을 보여줍니다.

```
{
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
        ...
    }
    "versionNumber": "number"
}
```

자세한 내용은 JobExecutionState 또는 job-execution-state 섹션을 참조하세요.

JobExecutionSummary

작업 실행에 대한 정보의 하위 집합이 포함됩니다. 다음 예제에서는 구문을 보여줍니다.

```
{
    "jobId": "string",
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "versionNumber": "number",
    "executionNumber": long
}
```

자세한 내용은 JobExecutionSummary 또는 job-execution-summary 섹션을 참조하세요.

다음 섹션에서 MQTT 및 HTTPS API 작업에 대해 자세히 알아보십시오.

- 작업 디바이스 MQTT API 작업
- 작업 디바이스 HTTP API

작업 디바이스 MQTT API 작업

작업 명령에 사용되는 예약된 주제에 MQTT 메시지를 게시하여 작업 디바이스 명령을 발행할 수 있습니다.

디바이스 측 클라이언트는 이러한 명령의 응답 메시지 항목을 구독해야 합니다. AWS IoT Device Client를 사용하는 경우 디바이스는 자동으로 응답 주제를 구독합니다. 즉, 클라이언트가 응답 메시지 주제를 구독했는지 여부에 관계없이 메시지 브로커가 명령 메시지를 게시한 클라이언트에 응답 메시지 주제를 게시합니다. 이러한 응답 메시지는 메시지 브로커를 통과하지 않으며 다른 클라이언트나 규칙에 의해 구독될 수 없습니다.

플릿 모니터링 솔루션에 대한 작업 및 `jobExecution` 이벤트 주제를 구독할 때 먼저 작업 및 작업 실행 이벤트를 사용하여 클라우드 측에서 이벤트를 수신합니다. 메시지 브로커를 통해 처리되고 AWS IoT 규칙에서 사용할 수 있는 작업 진행 메시지는 작업 이벤트로 게시됩니다. 메시지 브로커가 응답 메시지를 게시하기 때문에, 명시적인 구독을 하지 않더라도 수신하는 메시지를 수신하고 식별하도록 클라이언트를 구성해야 합니다. 또한 클라이언트는 클라이언트가 메시지에 대해 작업을 수행하기 전에 수신 메시지 주제의 *`thingName`*이 클라이언트의 사물 이름에 적용되는지 확인해야 합니다.

> **Note**
>
> MQTT Jobs API 명령 메시지에 대한 응답으로가 AWS IoT 전송하는 메시지는 명시적으로 구독했는지 여부에 관계없이 계정에 요금이 부과됩니다.

다음은 MQTT API 작업과 해당 요청 및 응답 구문을 보여 줍니다. 모든 MQTT API 작업에는 다음 파라미터가 있습니다.

clientToken

요청과 응답의 연관성을 나타내는 데 사용하는 선택적 클라이언트 토큰입니다. 여기에 임의 값을 입력하면 응답에 반영됩니다.

`timestamp`

메시지가 전송된 Epoch 이후 경과 시간(초)입니다.

GetPendingJobExecutions

지정된 사물에 대해 종료 상태가 아닌 사물의 모든 작업 목록을 가져옵니다.

이 API를 호출하려면 `$aws/things/thingName/jobs/get` 주제에 메시지를 게시합니다.

요청 페이로드:

```
{ "clientToken": "string" }
```

메시지 브로커는 특정 구독 없이도 `$aws/things/thingName/jobs/get/accepted`와 `$aws/things/thingName/jobs/get/rejected`를 게시합니다. 그러나 클라이언트가 메시지를 수신하려면 메시지를 수신 대기해야 합니다. 자세한 내용은 작업 API 메시지에 대한 참고 사항을 참조하세요.

응답 페이로드:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ],
"timestamp" : 1489096425069,
"clientToken" : "client-001"
}
```

여기서 `inProgressJobs` 및 `queuedJobs`에서는 상태가 `IN_PROGRESS` 또는 `QUEUED`인 JobExecutionSummary 객체 목록을 반환합니다.

StartNextPendingJobExecution

사물의 다음 번 대기 중 작업 실행을 가져오고 시작합니다(`IN_PROGRESS` 또는 `QUEUED` 상태).

- 상태가 `IN_PROGRESS`인 모든 작업 실행이 먼저 반환됩니다.
- 작업 실행이 대기열에 추가된 순서대로 반환됩니다. 작업의 대상 그룹에 사물이 추가 또는 제거되면 새 작업 실행의 롤아웃 순서를 기존 작업 실행과 비교하여 확인하세요.
- 다음 대기 중인 작업 실행이 `QUEUED` 상태이면, 상태가 `IN_PROGRESS`로 바뀌고 작업 실행의 상태 세부 정보가 지정한 대로 설정됩니다.
- 다음 보류 중인 작업 실행이 이미 `IN_PROGRESS` 상태이면 상태 세부 정보가 바뀌지 않습니다.
- 보류 중인 작업 실행이 없으면 응답에 `execution` 필드가 포함되지 않습니다.
- 선택적으로 `stepTimeoutInMinutes` 속성에 대한 값을 설정하여 단계 타이머를 생성할 수 있습니다. `UpdateJobExecution`를 실행하여 이 속성 값을 업데이트하지 않은 경우 단계 타이머가 만료할 때 작업 실행이 시간 초과됩니다.

이 API를 호출하려면 $aws/things/*thingName*/jobs/start-next 주제에 메시지를 게시합니다.

요청 페이로드:

```
{
"statusDetails": {
    "string": "job-execution-state"
    ...
},
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

statusDetails

작업 실행 상태를 설명하는 이름-값 페어의 모음입니다. 지정하지 않으면 statusDetails가 바뀌지 않습니다.

stepTimeOutInMinutes

이 디바이스가 이 작업 실행을 마쳐야 하는 시간을 지정합니다. 이 타이머가 완료하기 전에 또는 타이머가 재설정되면(UpdateJobExecution를 호출하고, 상태를 IN_PROGRESS에 설정하고, 새로운 제한 시간 값을 필드 stepTimeoutInMinutes에 지정하여) 작업 실행 상태는 TIMED_OUT로 설정됩니다. 이 시간 초과를 설정해도 작업이 생성되었을 때(CreateJob 필드를 사용하는timeoutConfig) 지정했을 수도 있는 이 작업 실행 시간 초과에는 영향을 주지 않는다는 점을 유의하세요.

이 파라미터의 유효한 값 범위는 1~10080(1분~7일)입니다. -1 값도 유효하며 (UpdateJobExecutionRequest를 조기 사용하여 생성된) 현재 단계 타이머를 취소합니다.

메시지 브로커는 특정 구독 없이도 $aws/things/*thingName*/jobs/start-next/accepted와 $aws/things/*thingName*/jobs/start-next/rejected를 게시합니다. 그러나 클라이언트가 메시지를 수신하려면 메시지를 수신 대기해야 합니다. 자세한 내용은 작업 API 메시지에 대한 참고 사항을 참조하세요.

응답 페이로드:

```
{
"execution" : JobExecutionData,
"timestamp" : timestamp,
```

```
"clientToken" : "string"
}
```

여기서 execution은 JobExecution 객체입니다. 예:

```
{
"execution" : {
    "jobId" : "022",
    "thingName" : "MyThing",
    "jobDocument" : "< contents of job document >",
    "status" : "IN_PROGRESS",
    "queuedAt" : 1489096123309,
    "lastUpdatedAt" : 1489096123309,
    "versionNumber" : 1,
    "executionNumber" : 1234567890
},
"clientToken" : "client-1",
"timestamp" : 1489088524284,
}
```

DescribeJobExecution

작업 실행에 대한 세부 정보를 가져옵니다.

jobId를 $next로 설정하여 다음 대기 중인(IN_PROGRESS 또는 QUEUED 상태) 사물의 작업 실행을 반환할 수 있습니다.

이 API를 호출하려면 $aws/things/*thingName*/jobs/*jobId*/get 주제에 메시지를 게시합니다.

요청 페이로드:

```
{
"jobId" : "022",
"thingName" : "MyThing",
"executionNumber": long,
"includeJobDocument": boolean,
"clientToken": "string"
}
```

thingName

디바이스와 연결된 사물의 이름입니다.

`jobId`

작업 생성 시 할당한 고유 식별자입니다.

또는 `$next`를 사용하여 다음 대기 중인(`IN_PROGRESS` 또는 `QUEUED` 상태) 사물의 작업 실행을 반환할 수 있습니다. 이 경우에는 상태가 `IN_PROGRESS`인 모든 작업 실행이 먼저 반환됩니다. 작업 실행이 생성된 순서대로 반환됩니다.

`executionNumber`

(선택 사항) 디바이스에서 작업 실행을 식별하기 위한 숫자입니다. 지정하지 않으면 최신 작업 실행이 반환됩니다.

`includeJobDocument`

(선택 사항) `false`로 설정하지 않을 경우 응답에 작업 문서가 포함됩니다. 기본값은 `true`입니다.

메시지 브로커는 특정 구독 없이도 `$aws/things/thingName/jobs/jobId/get/accepted`와 `$aws/things/thingName/jobs/jobId/get/rejected`를 게시합니다. 그러나 클라이언트가 메시지를 수신하려면 메시지를 수신 대기해야 합니다. 자세한 내용은 작업 API 메시지에 대한 참고 사항을 참조하세요.

응답 페이로드:

```
{
"execution" : JobExecutionData,
"timestamp": "timestamp",
"clientToken": "string"
}
```

여기서 `execution`은 JobExecution 객체입니다.

UpdateJobExecution

작업 실행의 상태를 업데이트합니다. `stepTimeoutInMinutes` 속성에 대한 값을 설정하여 단계 타이머를 선택적으로 추가할 수 있습니다. `UpdateJobExecution`를 다시 실행하여 이 속성 값을 업데이트하지 않은 경우 단계 타이머가 만료할 때 작업 실행이 시간 초과됩니다.

이 API를 호출하려면 `$aws/things/thingName/jobs/jobId/update` 주제에 메시지를 게시합니다.

요청 페이로드:

```
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"executionNumber": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`status`

작업 실행의 새 상태(`IN_PROGRESS`, `FAILED`, `SUCCEEDED` 또는 `REJECTED`). 이 명령은 업데이트할 때마다 지정해야 합니다.

`statusDetails`

작업 실행 상태를 설명하는 이름-값 페어의 모음입니다. 지정하지 않으면 `statusDetails`가 바뀌지 않습니다.

`expectedVersion`

예상되는 현재 작업 실행 버전입니다. 작업 실행은 업데이트할 때마다 버전이 일정하게 증가합니다. AWS IoT 작업 서비스에 저장된 작업 실행 버전이 일치하지 않으면 업데이트가 `VersionMismatch` 오류와 함께 거부됩니다. 현재 작업 실행 상태 데이터가 포함된 ErrorResponse도 반환됩니다. (이러한 경우에는 작업 실행 상태에 대한 데이터를 가져오기 위해 `DescribeJobExecution` 요청을 별도로 실행할 필요가 없습니다)

`executionNumber`

(선택 사항) 디바이스에서 작업 실행을 식별하기 위한 숫자입니다. 지정하지 않으면 최신 작업 실행이 사용됩니다.

`includeJobExecutionState`

(선택 사항) 이 명령을 추가하여 `true`로 설정하면 응답에 `JobExecutionState` 필드가 포함됩니다. 기본값은 `false`입니다.

`includeJobDocument`

(선택 사항) 이 명령을 추가하여 `true`로 설정하면 응답에 `JobDocument` 필드가 포함됩니다. 기본값은 `false`입니다.

`stepTimeoutInMinutes`

이 디바이스가 이 작업 실행을 마쳐야 하는 시간을 지정합니다. 이 타이머가 만료되기 전에 또는 타이머가 재설정되기 전에 작업 실행 상태가 최종 상태로 설정되지 않으면 작업 실행 상태가 `TIMED_OUT`이 됩니다. 이 제한 시간을 설정 또는 재설정하더라도 작업을 생성했을 때 지정했을 수 있는 작업 실행 제한 시간은 바뀌지 않습니다.

메시지 브로커는 특정 구독 없이도 `$aws/things/thingName/jobs/jobId/update/accepted`와 `$aws/things/thingName/jobs/jobId/update/rejected`를 게시합니다. 그러나 클라이언트가 메시지를 수신하려면 메시지를 수신 대기해야 합니다. 자세한 내용은 작업 API 메시지에 대한 참고 사항을 참조하세요.

응답 페이로드:

```
{
"executionState": JobExecutionState,
"jobDocument": "string",
"timestamp": timestamp,
"clientToken": "string"
}
```

`executionState`

JobExecutionState 객체입니다.

`jobDocument`

작업 문서 객체.

> **Note**
>
> MQTT 응답에서 `jobDocument` 필드는 JSON 객체입니다. HTTP 응답에서 JSON 객체의 문자열 표현입니다.

`timestamp`

메시지가 전송된 Epoch 이후 경과 시간(초)입니다.

`clientToken`

요청과 응답의 연관성을 나타낼 때 사용하는 클라이언트 토큰입니다.

MQTT 프로토콜을 사용하는 경우 다음 업데이트를 수행할 수도 있습니다.

JobExecutionsChanged

대기 중인 사물의 작업 실행 목록에 작업 실행이 추가되거나 제거될 때마다 전송됩니다.

다음 주제를 사용합니다.

`$aws/things/thingName/jobs/notify`

메시지 페이로드:

```
{
"jobs" : {
    "JobExecutionState": [ JobExecutionSummary ... ]
         },
    "timestamp": timestamp
}
```

NextJobExecutionChanged

`jobId $next`를 사용하는 DescribeJobExecution에서 정의한 바와 같이 대기 중인 사물의 작업 실행 목록에서 다음 작업 실행에 변경 사항이 있을 때마다 전송됩니다. 다음 작업의 실행 세부 정보가 변경될 때는 전송되지 않으며, 오직 `jobId $next`를 사용한 `DescribeJobExecution`에서 반환되는 다음 작업이 변경되었을 때만 전송됩니다. 예를 들어 작업 실행 J1과 J2가 `QUEUED` 상태라고 가정하겠습니다. J1은 대기 중인 작업 실행 목록에서 다음 작업 실행입니다. 이때 J1의 상태는 아무런 변화가 없는데 J2의 상태가 `IN_PROGRESS`로 바뀐다면 이 알림 메시지가 J2의 세부 정보와 함께 전송됩니다.

다음 주제를 사용합니다.

`$aws/things/thingName/jobs/notify-next`

메시지 페이로드:

```
{
"execution" : JobExecution,
"timestamp": timestamp,
}
```

작업 디바이스 HTTP API

디바이스는 포트 443에서 HTTP 서명 버전 4를 사용하여 AWS IoT 작업과 통신할 수 있습니다. 이는 AWS SDKs 및 CLI에서 사용하는 방법입니다. 이 도구에 관한 자세한 내용은 AWS CLI 명령 참조: iot-jobs-data 또는 AWS SDK 및 도구를 참조하세요.

다음 명령은 작업을 실행하는 디바이스에 사용할 수 있습니다. MQTT 프로토콜에서 API 작업을 사용하는 방법에 대한 자세한 내용은 작업 디바이스 MQTT API 작업 섹션을 참조하세요.

GetPendingJobExecutions

지정된 사물에 대해 최종 상태가 아닌 모든 작업 목록을 가져옵니다.

HTTPS request

```
GET /things/thingName/jobs
```

응답:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ]
}
```

자세한 내용은 GetPendingJobExecutions 단원을 참조하십시오.

CLI syntax

```
aws iot-jobs-data get-pending-job-executions \
--thing-name <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"thingName": "string"
}
```

자세한 내용은 get-pending-job-executions 단원을 참조하십시오.

StartNextPendingJobExecution

사물의 다음 번 대기 중 작업 실행을 가져오고 시작합니다(IN_PROGRESS 또는 QUEUED 상태).

- 상태가 IN_PROGRESS인 모든 작업 실행이 먼저 반환됩니다.
- 작업 실행이 생성된 순서대로 반환됩니다.
- 다음 대기 중인 작업 실행이 QUEUED 상태이면, 상태가 IN_PROGRESS로 바뀌고 작업 실행의 상태 세부 정보가 지정한 대로 설정됩니다.
- 다음 보류 중인 작업 실행이 이미 IN_PROGRESS 상태이면 상태 세부 정보가 바뀌지 않습니다.
- 보류 중인 작업 실행이 없으면 응답에 execution 필드가 포함되지 않습니다.
- 선택적으로 stepTimeoutInMinutes 속성에 대한 값을 설정하여 단계 타이머를 생성할 수 있습니다. UpdateJobExecution를 실행하여 이 속성 값을 업데이트하지 않은 경우 단계 타이머가 만료할 때 작업 실행이 시간 초과됩니다.

HTTPS request

다음 예제에서는 요청 구문을 보여줍니다.

```
PUT /things/thingName/jobs/$next
{
"statusDetails": {
    "string": "string"
    ...
},
"stepTimeoutInMinutes": long
}
```

자세한 내용은 StartNextPendingJobExecution 단원을 참조하십시오.

CLI syntax

시놉시스:

```
aws iot-jobs-data  start-next-pending-job-execution \
--thing-name <value> \
{--step-timeout-in-minutes <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 형식:

```
{
"thingName": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": long
}
```

자세한 내용은 start-next-pending-job-execution 단원을 참조하십시오.

DescribeJobExecution

작업 실행에 대한 세부 정보를 가져옵니다.

jobId를 $next로 설정하여 다음 대기 중인 사물의 작업 실행을 반환할 수 있습니다. 작업 실행 상태는 QUEUED 또는 IN_PROGRESS가 되어야 합니다.

HTTPS request

요청:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument
```

응답:

```
{
"execution" : JobExecution,
}
```

자세한 내용은 DescribeJobExecution 단원을 참조하십시오.

CLI syntax

시놉시스:

```
aws iot-jobs-data  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string",
"thingName": "string",
"includeJobDocument": boolean,
"executionNumber": long
}
```

자세한 내용은 describe-job-execution 단원을 참조하십시오.

UpdateJobExecution

작업 실행의 상태를 업데이트합니다. 선택적으로 `stepTimeoutInMinutes` 속성에 대한 값을 설정하여 단계 타이머를 생성할 수 있습니다. `UpdateJobExecution`를 다시 실행하여 이 속성 값을 업데이트하지 않은 경우 단계 타이머가 만료할 때 작업 실행이 시간 초과됩니다.

HTTPS request

요청:

```
POST /things/thingName/jobs/jobId
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
```

```
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"executionNumber": long
}
```

자세한 내용은 UpdateJobExecution 단원을 참조하십시오.

CLI syntax

시놉시스:

```
aws iot-jobs-data  update-job-execution \
--job-id <value> \
--thing-name <value> \
--status <value> \
[--status-details <value>] \
[--expected-version <value>] \
[--include-job-execution-state | --no-include-job-execution-state] \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--step-timeout-in-minutes <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 형식:

```
{
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": number,
"expectedVersion": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"executionNumber": long
}
```

자세한 내용은 update-job-execution 단원을 참조하십시오.

AWS IoT 작업으로 사용자 및 디바이스 보호

사용자에게 디바이스에서 AWS IoT 작업을 사용하도록 권한을 부여하려면 IAM 정책을 사용하여 권한을 부여해야 합니다. 그런 다음 AWS IoT Core 정책을 사용하여 디바이스에 안전하게 연결하고 AWS IoT, 작업 실행을 수신하고, 실행 상태를 업데이트하여 디바이스에 권한을 부여해야 합니다.

AWS IoT 작업에 필요한 정책 유형

다음 표에서는 권한 부여에 사용해야 하는 다양한 유형의 정책을 보여줍니다. 사용해야 하는 필수 정책에 대한 자세한 내용은 권한 부여 섹션을 참조하세요.

필수 정책 유형

사용 사례:	프로토콜	Authentication	컨트롤 플레인/데이터 영역	자격 증명 유형	필수 정책 유형
관리자, 운영자 또는 클라우드 서비스가 작업과 안전하게 작업하도록 권한 부여	HTTPS	AWS 서명 버전 4 인증(포트 443)	제어 영역 및 데이터 영역 모두	Amazon Cognito, IAM 또는 연동 페더레이션 사용자	IAM 정책
IoT 디바이스가 작업과 안전하게 작동하도록 권한 부여	MQTT/HTTPS	TCP 또는 TLS 상호 인증(포트 8883 또는 443)	데이터 영역	X.509 인증서	AWS IoT Core 정책

컨트롤 플레인과 데이터 플레인 모두에서 수행할 수 있는 AWS IoT 작업 작업을 승인하려면 IAM 정책을 사용해야 합니다. 이 작업을 수행하려면 AWS IoT 에 자격 증명이 인증되어 있어야 합니다. Amazon Cognito 자격 증명 또는 IAM 사용자, 그룹 및 역할이어야 합니다. 인증에 대한 자세한 내용은 Authentication 섹션을 참조하세요.

이제 디바이스 게이트웨이에 안전하게 연결하기 위한 AWS IoT Core 정책을 사용하여 데이터 영역에서 디바이스를 승인해야 합니다. 디바이스 게이트웨이를 사용하면 디바이스가와 안전하게 통신하고

AWS IoT, 작업 실행을 수신하고, 작업 실행 상태를 업데이트할 수 있습니다. 보안 MQTT 또는 HTTPS 게시 통신 프로토콜을 사용하여 디바이스 통신이 보호됩니다. 이러한 프로토콜은에서 제공하는 X.509 클라이언트 인증서 AWS IoT 를 사용하여 디바이스 연결을 인증합니다.

다음은 사용자, c 시끄러운 서비스 및 디바이스에 AWS IoT 작업을 사용하도록 권한을 부여하는 방법을 보여줍니다. 제어 영역 및 데이터 영역 API 작업에 대한 자세한 내용은 AWS IoT 작업 API 작업 섹션을 참조하세요.

주제

- 사용자 및 클라우드 서비스에 AWS IoT 작업 사용 권한 부여
- 디바이스가 데이터 영역에서 AWS IoT 작업을 안전하게 사용하도록 권한 부여

사용자 및 클라우드 서비스에 AWS IoT 작업 사용 권한 부여

사용자 및 클라우드 서비스에 권한을 부여하려면 제어 영역과 데이터 영역 모두에서 IAM 정책을 사용해야 합니다. 정책은 HTTPS 프로토콜과 함께 사용해야 하며 사용자를 인증하기 위해 AWS 서명 버전 4 인증(포트 443)을 사용해야 합니다.

> **Note**
>
> AWS IoT Core 정책은 컨트롤 플레인에서 사용해서는 안 됩니다. 사용자 또는 클라우드 서비스 권한 부여에는 IAM 정책만 사용됩니다. 필수 정책 유형에 대한 자세한 내용은 AWS IoT 작업에 필요한 정책 유형 섹션을 참조하세요.

IAM 정책은 정책 문을 포함하는 JSON 문서입니다. 정책 문은 Effect,Action 및 Resource 요소를 사용하여 리소스, 허용되거나 거부되는 작업, 작업이 허용되거나 거부되는 조건을 지정합니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

> **Warning**
>
> `"Action": ["iot:*"]` IAM 정책 또는 AWS IoT Core 정책과 같은 와일드카드 권한을 사용하지 않는 것이 좋습니다. 와일드카드 권한을 사용하는 것은 권장되는 보안 모범 사례가 아닙니다. 자세한 내용은 AWS IoT policies overly permissive를 참조하세요.

제어 영역의 IAM 정책

제어 영역에서 IAM 정책은 작업에 `iot:` 접두사를 사용하여 해당 작업 API 작업을 승인합니다. 예를 들어, `iot:CreateJob` 정책 작업은 사용자에게 CreateJob API를 사용할 권한을 부여합니다.

정책 작업

다음 표는 IAM 정책 작업 및 API 작업을 사용할 수 있는 권한을 보여줍니다. 리소스 유형에 대한 자세한 내용은에서 정의한 리소스 유형을 AWS IoT 참조하세요. 작업에 대한 AWS IoT 자세한 내용은에서 정의한 작업을 AWS IoT 참조하세요.

제어 영역에서의 IAM 정책 작업

정책 작업	API 작업	조건 키	설명
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	• job • thing • thinggroup	그룹을 연속 작업에 연결할 수 있는 권한을 나타냅니다. 대상 연결을 요청할 때마다 iot:AssociateTargetsWithJob 권한이 확인됩니다.
iot:CancelJob	CancelJob	작업	작업을 취소할 수 있는 권한을 나타냅니다. 작업 취소를 요청할 때마다 iot:CancelJob 권한이 확인됩니다.
iot:CancelJobExecution	CancelJobExecution	• job • thing	작업 실행을 취소할 수 있는 권한을 나타냅니다. 작업 실행 취소를 요청할 때마다 iot: CancelJobExecution 권한이 확인됩니다.
iot:CreateJob	CreateJob	• job • thing • thinggroup • jobtemplate • 패키지	작업을 생성할 수 있는 권한을 나타냅니다. 작업 생성을 요청할 때마다 iot: CreateJob 권한이 확인됩니다.

정책 작업	API 작업	조건 키	설명
iot:CreateJobTemplate	CreateJobTemplate	• 작업 • jobtemplate • 패키지	작업 템플릿을 생성할 수 있는 권한을 나타냅니다. 작업 템플릿 생성을 요청할 때마다 iot: CreateJobTemplate 권한이 확인됩니다.
iot:DeleteJob	DeleteJob	작업	작업을 삭제할 수 있는 권한을 나타냅니다. 작업 삭제를 요청할 때마다 iot: DeleteJob 권한이 확인됩니다.
iot:DeleteJobTemplate	DeleteJobTemplate	jobtemplate	작업 템플릿을 삭제할 수 있는 권한을 나타냅니다. 작업 템플릿 삭제를 요청할 때마다 iot: CreateJobTemplate 권한이 확인됩니다.
iot:DeleteJobExecution	DeleteJobTemplate	• job • thing	작업 실행을 삭제할 수 있는 권한을 나타냅니다. 작업 실행 삭제를 요청할 때마다 iot: DeleteJobExecution 권한이 확인됩니다.
iot:DescribeJob	DescribeJob	작업	작업을 설명할 수 있는 권한을 나타냅니다. 작업 설명을 요청할 때마다 iot: DescribeJob 권한이 확인됩니다.
iot:DescribeJobExecution	DescribeJobExecution	• job • thing	작업 실행을 설명할 수 있는 권한을 나타냅니다. 작업 실행 설명을 요청할 때마다 iot: DescribeJobExecution 권한이 확인됩니다.
iot:DescribeJobTemplate	DescribeJobTemplate	jobtemplate	작업 템플릿을 설명할 수 있는 권한을 나타냅니다. 작업 템플릿 설명을 요청할 때마다 iot: DescribeJobTemplate 권한이 확인됩니다.

정책 작업	API 작업	조건 키	설명
iot:DescribeManagedJobTemplate	DescribeManagedJobTemplate	jobtemplate	관리형 작업 템플릿을 설명할 수 있는 권한을 나타냅니다. 관리형 작업 템플릿 설명을 요청할 때마다 iot: DescribeManagedJobTemplate 권한이 확인됩니다.
iot:GetJobDocument	GetJobDocument	작업	작업에 대한 작업 문서를 가져올 수 있는 권한을 나타냅니다. 작업 문서 가져오기를 요청할 때마다 iot:GetJobDocument 권한이 확인됩니다.
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	작업	작업에 대한 작업 실행을 나열할 수 있는 권한을 나타냅니다. 작업에 대한 작업 실행 나열을 요청할 때마다 iot:ListJobExecutionsForJob 권한이 확인됩니다.
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	thing	작업에 대한 작업 실행을 나열할 수 있는 권한을 나타냅니다. 사물에 대한 작업 실행 나열을 요청할 때마다 iot:ListJobExecutionsForThing 권한이 확인됩니다.
iot:ListJobs	ListJobs	없음	작업을 나열할 수 있는 권한을 나타냅니다. 작업 나열을 요청할 때마다 iot:ListJobs 권한이 확인됩니다.
iot:ListJobTemplates	ListJobTemplates	없음	작업 템플릿을 나열할 수 있는 권한을 나타냅니다. 작업 템플릿 나열을 요청할 때마다 iot:ListJobTemplates 권한이 확인됩니다.
iot:ListManagedJobTemplates	ListManagedJobTemplates	없음	관리형 작업 템플릿을 나열할 수 있는 권한을 나타냅니다. 관리형 작업 템플릿 나열을 요청할 때마다 iot:ListManagedJobTemplates 권한이 확인됩니다.

정책 작업	API 작업	조건 키	설명
iot:UpdateJob	UpdateJob	작업	작업을 업데이트할 수 있는 권한을 나타냅니다. 작업 업데이트를 요청할 때마다 iot:UpdateJob 권한이 확인됩니다.
iot:TagResource	TagResource	• 작업 • jobtemplate • thing	특정 리소스에 태그를 지정할 수 있는 권한을 부여합니다.
iot:UntagResource	UntagResource	• 작업 • jobtemplate • thing	특정 리소스의 태그를 해제할 수 있는 권한을 나타냅니다.

기본 IAM 정책 예시

다음 예시는 IoT 사물 및 사물 그룹에 대해 다음 작업을 수행할 수 있는 사용자 권한을 허용하는 IAM 정책을 보여줍니다.

이 예에서 다음과 같이 대체합니다.

- *region* with your AWS 리전, 예: us-east-1.
- *account-id*와 같은 사용자 AWS 계정 번호57EXAMPLE833.
- *thing-group-name*을 작업의 대상인 IoT 사물 그룹의 이름으로 대체합니다. 예: FirmwareUpdateGroup.
- *thing-name*을 작업의 대상인 IoT 사물의 이름으로 대체합니다. 예: MyIoTThing.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
```

```
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thinggroup/thing-group-
name"
        },
        {
            "Action": [
                "iot:DescribeJob",
                "iot:CancelJob",
                "iot:DeleteJob"
            ],
            "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:job/*"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/thing-123",
                "arn:aws:iot:us-east-1:123456789012:job/*"
            ]
        }
    ]
}
```

IP 기반 권한 부여에 대한 IAM 정책 예

보안 주체가 특정 IP 주소에서 컨트롤 플레인 엔드포인트로 API를 호출하지 못하도록 제한할 수 있습니다. 허용할 수 있는 IP 주소를 지정하려면 IAM 정책의 Condition 요소에서 `aws:SourceIp` 전역 조건 키를 사용합니다.

이 조건 키를 사용하면 다른 AWS 서비스에 대한 액세스가와 같이 사용자를 대신하여 이러한 API를 호출하지 못하도록 거부할 수도 있습니다 AWS CloudFormation. 이러한 서비스에 대한 액세스를 허용하려면 aws:SourceIp 키와 함께 `aws:ViaAWSService` 전역 조건 키를 사용합니다. 이렇게 하면 소스 IP 주소 액세스 제한이 보안 주체가 직접 수행한 요청에만 적용됩니다. 자세한 내용은 AWS: 소스 IP를 AWS 기반으로에 대한 액세스 거부를 참조하세요.

다음 예제에서는 특정 IP 주소만 컨트롤 플레인 엔드포인트에 대한 API 호출을 수행할 수 있도록 허용하는 방법을 보여줍니다. `aws:ViaAWSService` 키가 `true`로 설정되어 있으므로 다른 서비스에서 사용자를 대신하여 API를 호출할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                 "IpAddress": {
                     "aws:SourceIp": "123.45.167.89"
                 },
                 "Bool": {
                     "aws:ViaAWSService": "false"
        }
        }
    }]
}
```

데이터 영역의 IAM 정책

데이터 영역의 IAM 정책은 `iotjobsdata:` 접두사를 사용하여 사용자가 수행할 수 있는 작업 API 작업에 권한을 부여합니다. 데이터 플레인에서 `iotjobsdata:DescribeJobExecution` 정책 작업을 사용하여 사용자에게 DescribeJobExecution API 사용 권한을 부여할 수 있습니다.

> **Warning**
>
> AWS IoT 작업의 대상으로 디바이스를 지정할 때는 데이터 영역에서 IAM 정책을 사용하는 것을 권장하지 않습니다. 사용자가 작업을 생성하고 관리할 수 있도록 제어 영역에서 IAM 정책을 사용하는 것이 좋습니다. 데이터 영역에서 디바이스가 작업 실행을 검색하고 실행 상태를 업데

이트하도록 권한을 부여하려면 AWS IoT Core HTTPS 프로토콜에 대한 정책을(를) 사용하세요.

기본 IAM 정책 예시

권한을 부여받아야 하는 API 작업은 일반적으로 CLI 명령을 입력하여 수행합니다. 다음은 `DescribeJobExecution` 작업을 수행하는 예를 보여줍니다.

이 예에서 다음과 같이 대체합니다.

- *region* with your AWS 리전, 예: `us-east-1`.
- *account-id*와 같은 사용자 AWS 계정 번호`57EXAMPLE833`.
- *thing-name*을 작업의 대상인 IoT 사물의 이름으로 대체합니다. 예: `myRegisteredThing`.
- *job-id*는 API를 사용하여 대상이 지정된 작업의 고유 식별자입니다.

```
aws iot-jobs-data describe-job-execution \
    --endpoint-url "https://account-id.jobs.iot.region.amazonaws.com" \
    --job-id jobID --thing-name thing-name
```

다음은 이 작업을 승인하는 샘플 IAM 정책을 보여줍니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iotjobsdata:DescribeJobExecution"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
    }
}
```

IP 기반 권한 부여에 대한 IAM 정책 예

보안 주체가 특정 IP 주소에서 데이터 영역 엔드포인트로 API를 호출하지 못하도록 제한할 수 있습니다. 허용할 수 있는 IP 주소를 지정하려면 IAM 정책의 Condition 요소에서 aws:SourceIp 전역 조건 키를 사용합니다.

이 조건 키를 사용하면 다른 AWS 서비스에 대한 액세스가와 같이 사용자를 대신하여 이러한 API를 호출하지 못하도록 거부할 수도 있습니다 AWS CloudFormation. 이러한 서비스에 대한 액세스를 허용하려면 aws:SourceIp 조건 키와 함께 aws:ViaAWSService 전역 조건 키를 사용합니다. 이렇게 하면 IP 주소 액세스 제한이 보안 주체가 직접 수행한 요청에만 적용됩니다. 자세한 내용은 AWS: 소스 IP를 AWS 기반으로에 대한 액세스 거부를 참조하세요.

다음 예제에서는 특정 IP 주소만 데이터 영역 엔드포인트에 대한 API 호출을 수행할 수 있도록 허용하는 방법을 보여줍니다.

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iotjobsdata:*"
        ],
        "Resource": [
            "*"
        ],
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "123.45.167.89"
        },
            "Bool": {
                "aws:ViaAWSService": "false"
        }
        }
    }]
}
```

다음 예제에서는 특정 IP 주소 또는 주소 범위가 데이터 영역 엔드포인트에 대한 API 호출을 수행하지 못하도록 제한하는 방법을 보여줍니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iotjobsdata:*"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "123.45.167.89",
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            },
            "Resource": [
                "*"
            ]
        }
    ]
}
```

제어 영역과 데이터 영역 모두에 대한 IAM 정책 예시

컨트롤 플레인과 데이터 영역 모두에서 API 작업을 수행하는 경우 컨트롤 플레인 정책 작업은 `iot:` 접두사를, 데이터 영역 정책 작업은 `iotjobsdata:` 접두사를 사용해야 합니다.

예: `DescribeJobExecution` API는 제어 영역과 데이터 영역 모두에서 사용할 수 있습니다. 제어 영역에서 DescribeJobExecution API는 작업 실행을 설명하는 데 사용됩니다. 데이터 영역에서 DescribeJobExecution API는 작업 실행의 세부 정보를 가져오는 데 사용됩니다.

다음 IAM 정책은 사용자에게 제어 영역과 데이터 영역 모두에서 `DescribeJobExecution` API를 사용할 권한을 부여합니다.

이 예에서 다음과 같이 대체합니다.

- *region* with your AWS 리전, 예: `us-east-1`.
- *account-id*와 같은 사용자 AWS 계정 번호57EXAMPLE833.

- *thing-name*을 작업의 대상인 IoT 사물의 이름으로 대체합니다. 예: `MyIoTThing`.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iotjobsdata:DescribeJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/thing-123",
    "arn:aws:iot:us-east-1:123456789012:job/*"
            ]
        }
    ]
}
```

IoT 리소스의 태깅 권한 부여

생성, 수정 또는 사용할 수 있는 작업 및 작업 템플릿을 보다 효과적으로 제어하기 위해 작업 또는 작업 템플릿에 태그를 첨부할 수 있습니다. 태그는 또한 소유권을 식별하고 비용을 청구 그룹에 배치하고 태그를 첨부하여 비용을 배분하고 할당하는 데 도움이 됩니다.

사용자가 또는를 사용하여 생성한 작업 AWS Management Console 또는 작업 템플릿에 태그를 지정하려는 경우 AWS CLI IAM 정책은 사용자에게 태그 지정 권한을 부여해야 합니다. 이 권한을 부여하려면 IAM 정책이 `iot:TagResource` 작업을 사용해야 합니다.

> **Note**
>
> IAM 정책에 해당 `iot:TagResource` 작업이 포함되지 않은 경우, 태그가 있는 `CreateJob` 또는 `CreateJobTemplate`에서 `AccessDeniedException` 오류가 반환됩니다.

AWS Management Console 또는를 사용하여 생성한 작업 또는 작업 템플릿에 태그를 지정하려면 AWS CLI IAM 정책에서 해당 작업에 태그를 지정할 수 있는 권한을 부여해야 합니다. 이 권한을 부여하려면 IAM 정책이 `iot:TagResource` 작업을 사용해야 합니다.

리소스 태깅에 대한 자세한 내용은 AWS IoT 리소스에 태그 지정 섹션을 참조하세요.

IAM 정책 예시

태깅 권한을 부여하는 다음 IAM 정책 예시를 참조하세요.

예제 1.

작업을 만들고 특정 환경에 태그를 지정하기 위해 다음 명령을 사용하는 사용자입니다.

대체 예시:

- *region* with your AWS 리전, 예: `us-east-1`.
- *account-id*와 같은 사용자 AWS 계정 번호`57EXAMPLE833`.
- *thing-name*을 작업의 대상인 IoT 사물의 이름으로 대체합니다. 예: `MyIoTThing`.

```
aws iot create-job
    --job-id test_job
    --targets "arn:aws:iot:region:account-id:thing/thingOne"
    --document-source "https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json"
    --description "test job description"
    --tags Key=environment,Value=beta
```

이 예시에서는 다음 IAM 정책을 사용해야 합니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
```

```
            "iot:CreateJob",
            "iot:CreateJobTemplate",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": [
           "arn:aws:iot:us-east-1:123456789012:job/*",
           "arn:aws:iot:us-east-1:123456789012:jobtemplate/*"
        ]
    }
}
```

디바이스가 데이터 영역에서 AWS IoT 작업을 안전하게 사용하도록 권한 부여

디바이스가 데이터 영역에서 AWS IoT 작업과 안전하게 상호 작용할 수 있도록 권한을 부여하려면 AWS IoT Core policy. AWS IoT Core policies for job은 정책 문이 포함된 JSON 문서입니다. 이러한 정책은 Effect, Action, Resource 요소도 사용하며 IAM 정책과 유사한 규칙을 따릅니다. 요소에 대한 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

정책은 MQTT 및 HTTPS 프로토콜 모두에서 사용할 수 있으며 TCP 또는 TLS 상호 인증을 사용하여 디바이스를 인증해야 합니다. 다음은 서로 다른 통신 프로토콜에서 이러한 정책을 사용하는 방법을 보여줍니다.

> **Warning**
>
> `"Action": ["iot:*"]` IAM 정책 또는 AWS IoT Core 정책과 같은 와일드카드 권한을 사용하지 않는 것이 좋습니다. 와일드카드 권한을 사용하는 것은 권장되는 보안 모범 사례가 아닙니다. 자세한 내용은 AWS IoT policies overly permissive를 참조하세요.

AWS IoT Core MQTT 프로토콜에 대한 정책

AWS IoT Core MQTT 프로토콜에 대한 정책은 작업 디바이스 MQTT API 작업을 사용할 수 있는 권한을 부여합니다. MQTT API 작업은 작업 명령에 예약된 MQTT 주제로 작업하는 데 사용됩니다. 이러한 API 작업에 대한 자세한 내용은 작업 디바이스 MQTT API 작업 섹션을 참조하세요.

MQTT 정책은 `iot:Connect`, `iot:Publish`, `iot:Subscribe`, `iot:Receieve` 등의 정책 작업을 사용하여 작업 주제와 함께 작동합니다. 이러한 정책을 사용하면 메시지 브로커에 연결하고, 작업

MQTT 주제를 구독하고, 디바이스와 클라우드 간에 MQTT 메시지를 송수신할 수 있습니다. 이러한 작업에 대한 자세한 내용은 AWS IoT Core 정책 작업 섹션을 참조하세요.

AWS IoT 작업 주제에 대한 자세한 내용은 섹션을 참조하세요작업 주제.

기본 MQTT 정책 예시

다음 예시에서는 `iot:Publish` 및 `iot:Subscribe`을(를) 사용하여 작업과 작업 실행을 게시하고 구독하는 방법을 보여줍니다.

이 예에서 다음과 같이 대체합니다.

- *`region`* with your AWS 리전, 예: `us-east-1`.
- *`account-id`*와 같은 사용자 AWS 계정 번호`57EXAMPLE833`.
- *`thing-name`*을 작업의 대상인 IoT 사물의 이름으로 대체합니다. 예: `MyIoTThing`.

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/events/job/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/events/jobExecution/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thing-123/jobs/*"
            ]
        }
    ]
}
```

AWS IoT Core HTTPS 프로토콜에 대한 정책

AWS IoT Core 데이터 영역의 정책은 TLS 인증 메커니즘과 함께 HTTPS 프로토콜을 사용하여 디바이스를 승인할 수도 있습니다. 데이터 영역에서 정책은 `iotjobsdata:` 접두사

를 사용하여 디바이스가 수행할 수 있는 작업 API 작업에 권한을 부여합니다. 예를 들어, `iotjobsdata:DescribeJobExecution` 정책 작업은 사용자에게 DescribeJobExecution API를 사용할 권한을 부여합니다.

> **Note**
>
> 데이터 영역 정책 작업은 `iotjobsdata:` 접두사를 사용해야 합니다. 제어 영역에서 작업은 `iot:` 접두사를 사용해야 합니다. 제어 영역과 데이터 영역 정책 작업이 모두 사용되는 경우 IAM 정책 예시는 제어 영역과 데이터 영역 모두에 대한 IAM 정책 예시 섹션을 참조하세요.

정책 작업

다음 표에는 디바이스가 API 작업을 사용하도록 권한을 부여하기 위한 AWS IoT Core 정책 작업 및 권한 목록이 나와 있습니다. 데이터 영역에서 수행할 수 있는 API 작업 목록은 작업 디바이스 HTTP API 섹션을 참조하세요.

> **Note**
>
> 이 작업 실행 정책 작업은 HTTP TLS 엔드포인트에만 적용됩니다. MQTT 엔드포인트를 사용할 경우 앞서 정의한 MQTT 정책 작업을 사용해야 합니다.

AWS IoT Core 데이터 영역의 정책 작업

정책 작업	API 작업	조건 키	설명
`iotjobsdata:DescribeJobExecution`	DescribeJobExecution	• job • thing	작업 실행을 검색할 수 있는 권한을 나타냅니다. 작업 실행 검색을 요청할 때마다 `iotjobsdata:DescribeJobExecution` 권한이 확인됩니다.
`iotjobsdata:GetPendingJobExecutions`	GetPendingJobExecutions	thing	사물에 대해 단말 상태가 아닌 작업의 목록을 검색할 수 있는 권한을 나타냅니다. 목록 검색이 요청될 때마다 `iotjobsdata:GetPen`

정책 작업	API 작업	조건 키	설명
			dingJobExecutions 권한이 확인됩니다.
iotjobsdata:StartNextPendingJobExecution	StartNextPendingJobExecution	thing	사물에 대해 대기 중인 다음 작업 실행을 가져오고 시작할 수 있는 권한을 나타냅니다. (즉, QUEUED에서 IN_PROGRESS 상태로 작업 실행을 업데이트함) 대기 중인 다음 작업의 실행 시작이 요청될 때마다 iotjobsdata:StartNextPendingJobExecution 권한이 확인됩니다.
iotjobsdata:UpdateJobExecution	UpdateJobExecution	thing	작업 실행을 업데이트할 수 있는 권한을 나타냅니다. 작업 실행 상태 업데이트가 요청될 때마다 iotjobsdata:UpdateJobExecution 권한이 확인됩니다.

기본 정책 예시

다음은 모든 리소스에 대한 데이터 영역 API 작업에서 작업을 수행할 수 있는 권한을 부여하는 AWS IoT Core 정책의 예입니다. IoT 사물과 같은 특정 리소스로 정책 범위를 지정할 수 있습니다. 이 예에서 다음과 같이 대체합니다.

- AWS 리전 와 같은가 있는 *##*us-east-1.
- *account-id*와 같은 사용자 AWS 계정 번호57EXAMPLE833.
- *thing-name*을 IoT 사물의 이름으로 대체합니다. 예: MyIoTthing.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
```

```
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
        }
    ]
}
```

이러한 정책을 사용해야 하는 경우는 IoT 디바이스가 AWS IoT Core 정책을 사용하여 이 API 작업 중 하나에 액세스하는 경우입니다. 예를 들면 `DescribeJobExecution` API의 다음 예와 같습니다.

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument&namespaceId=namespaceId
 HTTP/1.1
```

AWS IoT 작업 제한

AWS IoT 작업에는의 최대 서비스 리소스 또는 작업 수에 해당하는 서비스 할당량 또는 제한이 있습니다 AWS 계정.

주제

- 작업 실행 한도
- 활성 및 동시 작업 제한

작업 실행 한도

이 섹션에서는 AWS IoT Device Management의 작업 실행 한도에 대한 정보를 제공합니다.

> **Note**
>
> 이러한 한도는 AWS IoT Device Management Service Quotas 설명서에서 찾을 수 있는 서비스 할당량의 일부가 아닙니다.

보류 중인 작업 실행 수에 대한 정보를 얻으려면 GetPendingJobExecutions API를 사용하거나 AWS IoT 작업에 대한 MQTT 예약 주제를 구독하고를 수신할 수 있습니다작업 알림 유형.

계정에서 보류 중인 작업 실행 수는 예약 구성을 활성화하고 반복 유지 관리 기간을 사용하는지 여부에 따라 달라질 수 있습니다.

대기 중인 작업 실행의 최대 수

API/알림 이름	설명	예약 구성 사용	예약 구성 미사용
ListNotification	이전 작업 실행이 터미널 상태가 되거나 새 작업 실행이 대기열에 추가되거나 비터미널 상태로 변경될 때마다 ListNotification 가 게시됩니다. QUEUED 또는 IN_PROGRESS 인 보류 중인 작업 실행을 최대 15개까지 표시할 수 있습니다.	10	15(유지 관리 기간 동안에 최대 5개의 작업 실행이 ListNotification 에만 표시됨).
GetPendingJobExecutions	GetPendingJobExecutions API를 간접 호출하면 아직 시작되지 않은 작업 실행 목록이 반환되며, API 직접 호출 후에 시작할 수 있습니다. API는 최대 10개의 보류 중인 작업 실행을 반환할 수 있습니다. • 보류 중인 작업 실행 10개 중 IN_PROGRESS 인 실행이 결과에서 필터링됩니다. • 보류 중인 작업 실행 10개 중 작업이 SCHEDULED 상태인 경우, 해당 작업이 결과에서 필터링됩니다.	10	15

활성 및 동시 작업 제한

이 단원에서는 활성 및 동시 작업과 해당 작업에 적용되는 제한에 대해 자세히 알아볼 수 있습니다.

활성 작업 및 활성 작업 제한

AWS IoT 콘솔 또는 CreateJob API를 사용하여 작업을 생성하면 작업 상태가 로 변경됩니다IN_PROGRESS. 진행 중인 모든 작업은 활성 작업이며 활성 작업 제한에 포함됩니다. 여기에는 새 작업 실행을 롤아웃하는 작업이나 디바이스가 작업 실행을 완료할 때까지 대기 중인 작업이 포함됩니다. 이 제한은 연속 작업과 스냅샷 작업 모두에 적용됩니다.

동시 작업 및 작업 동시성 제한

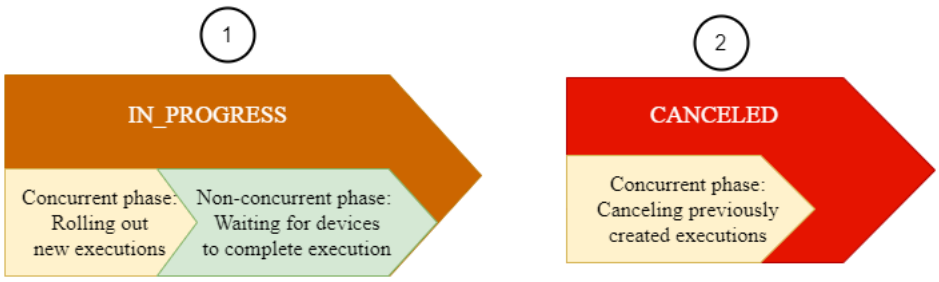
새 작업 실행을 롤아웃하는 진행 중인 작업 또는 이전에 생성된 작업 실행을 취소하는 작업은 동시 작업이며 작업 동시성 한도에 포함됩니다. AWS IoT 작업은 분당 1,000개의 디바이스 속도로 작업 실행을 신속하게 롤아웃하고 취소할 수 있습니다. 각 작업은 concurrent이며 짧은 시간 동안만 작업 동시 실행 한도에 포함됩니다. 작업 실행이 롤아웃되거나 취소된 후에는 작업이 더 이상 동시 작업이 아니며 작업 동시성 제한에 포함되지 않습니다. 디바이스가 작업 실행을 완료할 때까지 기다리는 동안 작업 동시성을 통해 많은 수의 작업을 생성할 수 있습니다.

> **Note**
>
> 선택적 예약 구성이 설정되었고 작업 문서 롤아웃이 유지 관리 기간 중에 수행되도록 예약된 작업이 선택한 startTime에 도달하고 최대 작업 동시 실행 한도에 도달하면 예약된 해당 작업은 CANCELED 상태가 됩니다.

작업이 동시인지 확인하려면 AWS IoT 콘솔에서 또는 DescribeJob 또는 ListJob API를 사용하여 작업의 IsConcurrent 속성을 사용할 수 있습니다. 이 제한은 연속 작업과 스냅샷 작업 모두에 적용됩니다.

에 대한 활성 작업 및 작업 동시성 한도와 기타 AWS IoT 작업 할당량을 보고 한도 증가를 요청 AWS 계정 하려면 AWS IoT 의 디바이스 관리 엔드포인트 및 할당량을 참조하세요 AWS 일반 참조.

다음 다이어그램은 진행 중인 작업과 취소 중인 작업에 작업 동시성이 어떻게 적용되는지 보여줍니다.



> **Note**
>
> 선택적 SchedulingConfig가 있는 새 작업은 초기 상태인 SCHEDULED로 유지된 후 선택된 startTime에 도달하면 IN_PROGRESS로 업데이트됩니다. 선택적 SchedulingConfig가 있는 새 작업이 선택된 startTime에 도달하여 IN_PROGRESS로 업데이트되면 활성 작업 제한 및 작업 동시성 제한 계산에 포함됩니다. 상태가 SCHEDULED인 작업은 활성 작업 제한을 계산하는 데 포함되지만 작업 동시성 제한을 계산하는 데는 포함되지 않습니다.

다음 표에서는 활성 및 동시 작업과 작업 상태의 동시 단계와 비동시 단계에 적용되는 제한을 보여줍니다.

활성 및 동시 작업 제한

작업 상태	Phase(단계)	활성 작업 제한	작업 동시성 제한
SCHEDULED	비동시 단계: AWS IoT 작업은 예약된 startTime 작업이 디바이스에 대한 작업 실행 알림을 시작할 때까지 기다립니다. 이 단계의 작업은 활성 작업 제한에만 포함되며 IsConcurrent 속성이 false로 설정되어 있습니다.	적용됨	적용되지 않음
IN_PROGRESS	동시 단계: AWS IoT 작업은 작업 생성 요청을 수락하고 디바이스에 작업 실행 알림을 롤아웃하기 시작합니다. 이 단계의 작업은 IsConcurrent 속성이 true로 설정된 것을 통해 알 수 있듯이 동시 작업이며 활성 작업과 작업 동시성 제한 모두에 포함됩니다.	적용됨	적용됨
	비동시 단계: AWS IoT 작업은 디바이스가 작업 실행 결과를 보고할 때까지 기다립니다. 이 단계의 작업은 활성 작업 제한에만 포함되며 IsConcurrent 속성이 false로 설정되어 있습니다.	적용됨	적용되지 않음

작업 상태	Phase(단계)	활성 작업 제한	작업 동시성 제한
`Canceled`	동시 단계: AWS IoT 작업은 작업 취소 요청을 수락하고 디바이스에 대해 이전에 생성된 작업 실행 취소를 시작합니다. 이 단계의 작업은 동시 작업이며 `IsConcurrent` 속성이 true로 설정되어 있습니다. 작업 및 작업 실행이 취소된 후에는 작업이 더 이상 동시 작업이 아니며 작업 동시성 제한에 포함되지 않습니다.	적용되지 않음	적용됨

> **Note**
>
> 반복 유지 관리 기간의 최대 기간은 23시간 50분입니다.

AWS IoT Device Management 명령

> **Important**
>
> 이 설명서에서는 AWS IoT Device Management에서 명령 기능을 사용하는 방법을 설명합니다. AWS IoT FleetWise의 이 기능 사용에 대한 자세한 내용은 원격 명령을 참조하세요.
> 안전하고 관련 법률을 준수하는 방식으로 명령을 배포하는 것은 전적으로 사용자의 책임입니다. 책임에 대한 자세한 내용은 AWS IoT 서비스에 대한AWS 서비스 약관을 참조하세요.

AWS IoT Device Management 명령을 사용하여 클라우드에서에 연결된 디바이스로 명령을 보냅니다 AWS IoT. 명령은 한 번에 하나의 디바이스를 대상으로 하며 디바이스 측 로그 검색 또는 디바이스 상태 변경 시작과 같은 지연 시간이 짧은 고처리량 애플리케이션에 사용할 수 있습니다.

명령은에서 관리하는 재사용 가능한 리소스입니다 AWS IoT Device Management. 디바이스에 게시되기 전에 적용되는 구성이 포함되어 있습니다. 전구를 켜거나 차량 도어를 잠금 해제하는 등 특정 사용 사례에 대한 명령 세트를 미리 정의할 수 있습니다.

AWS IoT 명령 기능을 사용하면 다음을 수행할 수 있습니다.

- 정적 또는 동적 페이로드를 사용하여 재사용 가능한 명령 템플릿을 생성한 다음 특정 디바이스에서 실행합니다.
- AWS IoT 사물 또는 등록되지 않은 MQTT 클라이언트로 등록된 디바이스를 대상으로 합니다.
- 동일한 디바이스에서 여러 명령을 동시에 실행합니다.
- 디바이스 프로세스 명령으로 이벤트 알림을 활성화하고 실행 상태를 추적합니다.

다음 주제에서는 명령을 생성하고, 해당 명령을 디바이스로 전송하고, 디바이스에서 보고한 상태를 검색하는 방법을 보여줍니다.

주제

- 빠른 시작
- 명령 개념 및 상태
- 상위 수준 명령 워크플로
- 명령 생성 및 관리
- 명령 실행 시작 및 모니터링

- 명령 리소스 사용 중지

빠른 시작

다음 단계를 완료하여 첫 번째 명령을 전송합니다.

1. 사전 조건 - IoT Core, IoT Core에 연결된 디바이스, Commands API 작업에 대한 IAM 권한, 디바이스에 설치된 MQTT 클라이언트 라이브러리에 온보딩되었는지 확인합니다.
2. 명령 생성 - 정적 페이로드 또는 디바이스 작업을 지정하는 페이로드 템플릿을 사용하여 명령을 정의합니다. 명령 리소스 생성을(를) 참조하세요.
3. 디바이스 식별 - 디바이스를 IoT 사물로 등록하거나 MQTT 클라이언트 ID를 기록해 둡니다. 대상 디바이스 고려 사항을(를) 참조하세요.
4. 권한 구성 - 디바이스가 명령을 수신하고 응답을 게시하도록 허용하는 IAM 정책을 설정합니다.
5. 주제 구독 - 명령 요청 및 응답 주제를 구독하도록 디바이스를 구성합니다. 명령의 대상 디바이스 선택 및 MQTT 주제 구독을(를) 참조하세요.
6. 명령 실행 - 대상 디바이스에서 명령 실행을 시작합니다. 명령 실행 시작을(를) 참조하세요.

일반적인 사용 사례:

- 원격 진단 - 로그 검색, 진단 실행, 디바이스 상태 확인
- 구성 업데이트 - 설정 변경, 파라미터 업데이트, 기능 전환
- 디바이스 제어 - 잠금/잠금 해제, 전원 켜기/끄기, 모드 변경

명령 개념 및 상태

AWS IoT 명령을 사용하여 클라우드에서 연결된 디바이스로 지침을 보냅니다. 이 특성을 사용하는 방법:

1. 디바이스에서 실행하는 데 필요한 구성이 포함된 페이로드로 명령을 생성합니다.
2. 페이로드를 수신하고 작업을 수행할 대상 디바이스를 지정합니다.
3. 대상 디바이스에서 명령을 실행하고 상태 정보를 검색합니다. 문제를 해결하려면 CloudWatch 로그를 참조하세요.

이 워크플로에 대한 자세한 내용은 상위 수준 명령 워크플로 섹션을 참조하세요.

주제

- 명령 주요 개념
- 명령 상태
- 명령 실행 상태

명령 주요 개념

다음 주요 개념은 명령 기능을 이해하는 데 도움이 됩니다. 용어는이 설명서 전체에서 일관되게 사용됩니다.

- 명령 - 디바이스 지침을 정의하는 재사용 가능한 템플릿
- 실행 - 디바이스에서 실행되는 명령의 인스턴스
- 사물 이름 - IoT 레지스트리에 등록된 디바이스의 식별자
- 클라이언트 ID - 등록되지 않은 디바이스의 MQTT 식별자
- 페이로드 - 디바이스로 전송된 명령 데이터
- 주제 - 명령 통신을 위한 MQTT 채널

명령

명령은 클라우드에서 IoT 디바이스로 MQTT 메시지로 전송되는 지침입니다. 페이로드를 수신한 후 디바이스는 지침을 처리하고 구성 설정 수정, 센서 읽기 전송 또는 로그 업로드와 같은 해당 작업을 수행합니다. 그런 다음 디바이스는 결과를 클라우드로 반환하여 원격 모니터링 및 제어를 활성화합니다.

네임스페이스

명령을 생성할 때 해당 네임스페이스를 지정합니다. AWS IoT Device Management 명령의 경우 기본 `AWS-IoT` 네임스페이스를 사용하고 페이로드 또는 payloadTemplate 중 하나를 제공합니다. AWS IoT FleetWise 명령의 경우 `AWS-IoT-FleetWise` 네임스페이스를 사용합니다. 자세한 내용은 AWS IoT FleetWise 개발자 안내서의 원격 명령을 참조하세요.

페이로드

명령을 생성할 때 디바이스가 수행해야 하는 작업을 정의하는 정적 페이로드를 제공합니다. 페이로드는 지원되는 모든 형식을 사용할 수 있습니다. 디바이스가 페이로드를 올바르게 해석하도록 하려면 페이로드 형식 유형을 지정하는 것이 좋습니다. MQTT5 프로토콜을 사용하는 디바이스는 MQTT 표준을 따라 형식을 식별할 수 있습니다. JSON 또는 CBOR에 대한 형식 표시기는 명령 요청 주제에서 사용할 수 있습니다.

페이로드 템플릿

페이로드 템플릿은 사용자가 제공한 파라미터 값에 따라 런타임에 다른 페이로드를 생성하는 자리 표시자가 있는 명령 페이로드를 정의합니다. 예를 들어, 서로 다른 온도 값에 대해 별도의 페이로드를 생성하는 대신 온도 자리 표시자가 있는 하나의 템플릿을 생성하고 실행 중에 값을 지정합니다. 이렇게 하면 여러 개의 유사한 페이로드를 유지 관리할 수 없습니다.

대상 디바이스

명령을 실행하려면 사물 이름(에 등록된 디바이스의 경우 AWS IoT) 또는 MQTT 클라이언트 ID(등록되지 않은 디바이스의 경우)를 사용하여 대상 디바이스를 지정합니다. 클라이언트 ID는 디바이스를 연결하는 데 사용되는 MQTT 프로토콜에 정의된 고유 식별자입니다 AWS IoT. 자세한 내용은 대상 디바이스 고려 사항을 참조하세요.

명령 주제

명령을 실행하기 전에 디바이스는 명령 요청 주제를 구독해야 합니다. 명령을 실행하면 페이로드가 이 주제의 디바이스로 전송됩니다. 실행 후 디바이스는 명령 응답 주제에 결과 및 상태를 게시합니다. 자세한 내용은 명령 주제 단원을 참조하십시오.

명령 실행

실행은 대상 디바이스에서 실행되는 명령의 인스턴스입니다. 실행을 시작하면 페이로드가 디바이스로 전송되고 고유한 실행 ID가 생성됩니다. 디바이스는 명령을 실행하고 진행 상황을 보고합니다 AWS IoT. 디바이스 측 로직은 예약된 주제에 대한 실행 동작 및 상태 보고를 결정합니다.

파라미터 값 조건

페이로드 템플릿으로 명령을 생성할 때 실행 전에 파라미터 값을 검증하는 값 조건을 정의합니다. 값 조건은 파라미터가 요구 사항을 충족하도록 하여 잘못된 실행을 방지합니다.

CommandParameterValue 유형별로 지원되는 연산자

숫자 유형(INTEGER, LONG, DOUBLE, UNSIGNEDLONG)

- EQUALS - 값은 지정된 숫자와 같아야 합니다.
- NOT_EQUALS - 값은 지정된 숫자와 같지 않아야 합니다.
- GREATER_THAN - 값은 지정된 숫자보다 커야 합니다.
- GREATER_THAN_EQUALS - 값은 지정된 숫자보다 크거나 같아야 합니다.
- LESS_THAN - 값은 지정된 숫자보다 작아야 합니다.
- LESS_THAN_EQUALS - 값은 지정된 숫자보다 작거나 같아야 합니다.

- IN_RANGE - 값은 지정된 범위(포함) 내에 있어야 합니다.
- NOT_IN_RANGE - 값이 지정된 범위(포함)를 벗어났어야 합니다.
- IN_SET - 값은 지정된 숫자 중 하나와 일치해야 합니다.
- NOT_IN_SET - 값은 지정된 숫자와 일치하지 않아야 합니다.

문자열 유형(STRING)

- EQUALS - 값은 지정된 문자열과 같아야 합니다.
- NOT_EQUALS - 값은 지정된 문자열과 같지 않아야 합니다.
- IN_SET - 값은 지정된 문자열 중 하나와 일치해야 합니다.
- NOT_IN_SET - 값은 지정된 문자열과 일치하지 않아야 합니다.

부울 유형

- 값 조건은 지원되지 않습니다.

바이너리 유형

- 값 조건은 지원되지 않습니다.

예: 온도 제어 명령

```
{
  "commandId": "SetTemperature",
  "namespace": "AWS-IoT",
  "payloadTemplate": "{\"temperature\":
 \"${aws:iot:commandexecution::parameter:temperature}\"}",
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {
            "numberRange": {
              "min": "60",
              "max": "80"
            }
          }
        }
      ]
    }
```

```
  ]
}
```

이 예제에서 temperature 파라미터는 60~80(포함)이어야 합니다. 이 범위를 벗어나는 값이 있는 실행 요청은 검증에 실패합니다.

> **Note**
>
> 값 조건은 StartCommandExecution API 호출 시 평가됩니다. 실패한 검증은 오류를 반환하고 실행 생성을 방지합니다.

파라미터 값 우선 순위 및 평가

페이로드 템플릿으로 명령 실행을 시작할 때 파라미터 값은 다음 우선 순위를 사용하여 확인됩니다.

1. 실행 요청 파라미터 - StartCommandExecution 요청에 제공된 값이 가장 우선합니다.
2. 명령 기본값 - 실행 요청에 파라미터가 제공되지 않은 경우 파라미터의 defaultValue가 사용됩니다.
3. 값 없음 - 둘 다 제공되지 않으면 실행 요청을 생성하는 데 필요한 파라미터로 실행이 실패합니다.

값 조건은 실행 생성 전에 우선 순위에 대해 위에서 파생된 최종 파라미터 값에 대해 평가됩니다. 검증에 실패하면 실행 요청이 오류를 반환합니다.

예:를 사용한 SetTemperature 명령 **defaultValue**

```
{
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "defaultValue": {"I": 72},
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {"numberRange": {"min": "60", "max": "80"}}
        }
      ]
    }
  ]
```

```
}
```

실행을 시작할 때:

- 요청에 `"temperature": {"I": 75}`를 입력하면 75가 사용됩니다.
- 온도 파라미터를 생략하면 기본값 72가 사용됩니다.
- 두 값 모두 범위 조건에 대해 검증됩니다[60,80]

명령 상태

의 명령은 사용 가능, 사용 중단됨 또는 삭제 보류 중의 세 가지 상태 중 하나일 AWS 계정 수 있습니다.

사용 가능

생성에 성공하면 명령은 사용 가능 상태이며 디바이스에서 실행할 수 있습니다.

Deprecated

더 이상 필요하지 않은 경우 사용 중단 명령을 표시합니다. 더 이상 사용되지 않는 명령은 새 실행을 시작할 수 없지만 보류 중인 실행은 계속 완료됩니다. 새 실행을 활성화하려면 명령을 사용 가능 상태로 복원합니다.

삭제 보류 중

삭제를 위해 명령을 표시하면 최대 제한 시간(기본값: 12시간)보다 오래 사용되지 않으면 자동으로 삭제됩니다. 이 작업은 영구적입니다. 제한 시간 미만 동안 더 이상 사용되지 않거나 더 이상 사용되지 않는 경우 명령은 삭제 보류 중 상태로 전환되고 제한 시간이 만료된 후 제거됩니다.

명령 실행 상태

대상 디바이스에서 실행을 시작하면 `CREATED` 상태가 되고 디바이스 보고서에 따라 다른 상태로 전환할 수 있습니다. 상태 정보를 검색하고 실행을 추적할 수 있습니다.

> **Note**
>
> 디바이스에서 여러 명령을 동시에 실행할 수 있습니다. 동시성 제어를 사용하여 디바이스당 실행을 제한하고 오버로드를 방지합니다. 디바이스당 최대 동시 실행 수는 AWS IoT Device Management 명령 할당량을 참조하세요.

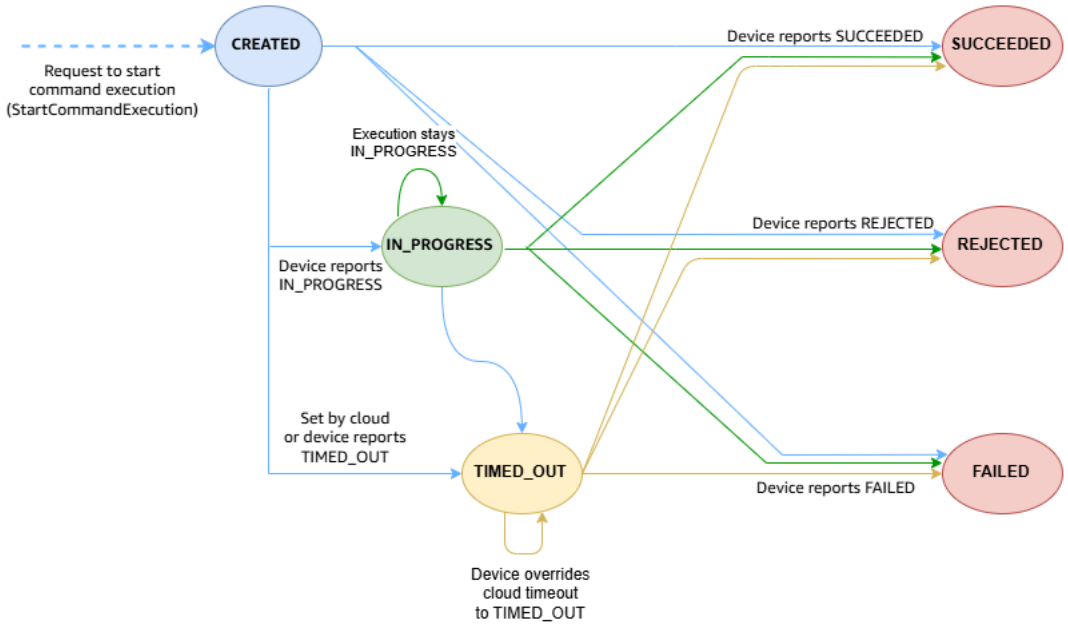
다음 표에는 실행 진행 상황에 따른 실행 상태와 전환이 나와 있습니다.

명령 실행 상태 및 소스

명령 실행 상태	디바이스/클라우드에 의한 시작 여부	터미널 실행 여부	허용된 상태 전환
CREATED	클라우드	아니요	• IN_PROGRESS • SUCCEEDED • FAILED • REJECTED • TIMED_OUT
IN_PROGRESS	장치	아니요	• IN_PROGRESS • SUCCEEDED • FAILED • REJECTED • TIMED_OUT
TIMED_OUT	디바이스 및 클라우드	아니요	• SUCCEEDED • FAILED • REJECTED • TIMED_OUT
SUCCEEDED	장치	예	해당 사항 없음
FAILED	장치	예	해당 사항 없음
REJECTED	장치	예	해당 사항 없음

디바이스는 명령 예약 MQTT 주제를 사용하여 언제든지 상태 및 결과 업데이트를 게시할 수 있습니다. 추가 컨텍스트를 제공하기 위해 디바이스는 `statusReason` 객체에서 `reasonCode` 및 `reasonDescription` 필드를 사용할 수 있습니다.

다음 다이어그램은 실행 상태 전환을 보여줍니다.



> **Note**
>
> 가 제한 시간 내에 디바이스 응답을 AWS IoT 감지하지 못하면 재시도 및 상태 변경을 허용하는 `TIMED_OUT` 임시 상태로 설정됩니다. 디바이스가를 명시적으로 보고하는 경우 `TIMED_OUT`이는 더 이상 전환되지 않는 터미널 상태가 됩니다. 자세한 내용은 비터미널 명령 실행 단원을 참조하십시오.

다음 섹션에서는 터미널 및 비터미널 실행과 해당 상태에 대해 설명합니다.

주제

- 비터미널 명령 실행
- 터미널 명령 실행

비터미널 명령 실행

디바이스의 업데이트를 수락할 수 있는 경우 실행은 비터미널입니다. 비터미널 실행은 활성으로 간주됩니다. 다음 상태는 비터미널 상태입니다.

- CREATED

AWS IoT 콘솔에서 실행을 시작하거나 StartCommandExecution API를 사용하면 요청이 성공하면 상태가 로 변경됩니다CREATED. 이 상태에서 실행은 다른 비터미널 또는 터미널 상태로 전환할 수 있습니다.

- IN_PROGRESS

  페이로드를 수신한 후 디바이스는 지침을 실행하고 지정된 작업을 수행하기 시작할 수 있습니다. 실행 중에 디바이스는 명령 응답 주제에 대한 응답을 게시하고 상태를 로 업데이트할 수 있습니다IN_PROGRESS. 에서 IN_PROGRESS실행은를 제외한 모든 터미널 또는 비터미널 상태로 전환할 수 있습니다CREATED.

  > **Note**
  >
  > UpdateCommandExecution API는 IN_PROGRESS 상태로 여러 번 호출할 수 있습니다. statusReason 객체를 사용하여 추가 실행 세부 정보를 지정합니다.

- TIMED_OUT

  클라우드와 디바이스 모두이 상태를 트리거할 수 있습니다. 다음과 같은 이유로 CREATED 또는 IN_PROGRESS 상태의 실행이 로 변경될 수 TIMED_OUT 있습니다.

  - 명령을 전송한 후 타이머가 시작됩니다. 디바이스가 지정된 기간 내에 응답하지 않으면 클라우드 상태가 로 변경됩니다TIMED_OUT. 이 경우 실행은 비터미널입니다.
  - 디바이스는 상태를 모든 터미널 상태로 재정의하거나 제한 시간을 보고하고 상태를 로 설정할 수 있습니다TIMED_OUT. 이 경우 상태는 로 유지TIMED_OUT되지만 StatusReason 객체 필드는 디바이스 정보에 따라 변경됩니다. 실행은 터미널이 됩니다.

  자세한 내용은 제한 시간 값 및 TIMED_OUT 명령 실행 상태 단원을 참조하십시오.

터미널 명령 실행

디바이스의 업데이트를 더 이상 수락하지 않으면 실행이 터미널이 됩니다. 다음 상태는 터미널 상태입니다. 실행은 CREATED, IN_PROGRESS또는 비터미널 상태에서 터미널 상태로 전환할 수 있습니다TIMED_OUT.

- SUCCEEDED

  디바이스가 실행을 성공적으로 완료하면 명령 응답 주제에 대한 응답을 게시하고 상태를 로 업데이트할 수 있습니다SUCCEEDED.

- FAILED

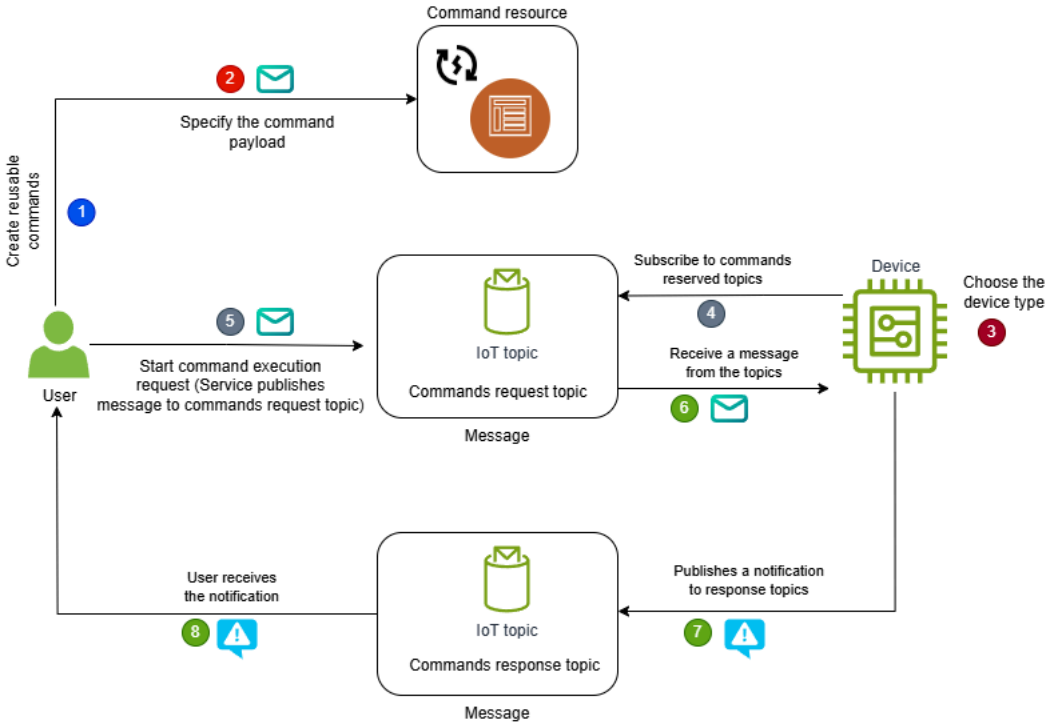
  디바이스가 실행을 완료하지 못하면 명령 응답 주제에 대한 응답을 게시하고 상태를 로 업데이트할 수 있습니다`FAILED`. `statusReason` 객체의 `reasonCode` 및 `reasonDescription` 필드 또는 CloudWatch 로그를 사용하여 실패 문제를 해결합니다.

- REJECTED

  디바이스가 유효하지 않거나 호환되지 않는 요청을 수신하면 상태로 `UpdateCommandExecution` API를 호출할 수 있습니다`REJECTED`. `statusReason` 객체의 `reasonCode` 및 `reasonDescription` 필드 또는 CloudWatch 로그를 사용하여 문제를 해결합니다.

상위 수준 명령 워크플로

이 워크플로는 디바이스가 AWS IoT Device Management 명령과 상호 작용하는 방식을 보여줍니다. 모든 HTTP API 요청은 서명에 Sigv4 자격 증명을 사용합니다.



워크플로우 개요

- 명령 생성 및 관리
- 명령의 대상 디바이스 선택 및 MQTT 주제 구독
- 대상 디바이스에 대한 명령 실행 시작 및 모니터링

- (선택 사항) 명령 이벤트에 대한 알림 활성화

명령 생성 및 관리

디바이스에 대한 명령을 생성하고 관리하려면 다음 단계를 수행합니다.

1. 명령 리소스 생성

   Command Hub에서 또는 `CreateCommand` API를 사용하여 명령을 생성합니다.

2. 페이로드 지정

   모든 형식으로 페이로드를 제공합니다. 올바른 디바이스 해석을 위해 콘텐츠 유형을 지정합니다.

   페이로드 템플릿이 있는 동적 명령의 경우 제공된 파라미터를 사용하여 런타임에 최종 페이로드가 생성됩니다. 템플릿은 JSON 형식만 지원하지만 생성된 페이로드는 JSON 또는 CBOR로 전송할 수 있습니다.

3. (선택 사항) 생성된 명령 관리

   생성 후 표시 이름과 설명을 업데이트합니다. 더 이상 필요하지 않은 경우 명령을 더 이상 사용되지 않는 것으로 표시하거나 영구적으로 삭제합니다. 페이로드 정보를 수정하려면 새 명령을 생성합니다.

명령의 대상 디바이스 선택 및 MQTT 주제 구독

대상 디바이스를 선택하고 명령을 수신하고 응답을 게시하기 위한 MQTT 주제를 구성합니다.

1. 명령의 대상 디바이스 선택

   명령을 수신하고 실행할 대상 디바이스를 선택합니다. 등록된 디바이스에는 사물 이름을 사용하고 등록되지 않은 디바이스에는 클라이언트 ID를 사용합니다. 자세한 내용은 대상 디바이스 고려 사항 단원을 참조하십시오.

2. AWS IoT 디바이스 정책 구성

   실행을 수신하고 업데이트를 게시할 수 있는 권한을 부여하는 IAM 정책을 구성합니다. 정책 예제는 섹션을 참조샘플 IAM 정책하세요.

3. MQTT 연결 설정

   디바이스를 메시지 브로커에 연결하고 요청 및 응답 주제를 구독합니다. 디바이스에는 `iot:Connect` 권한이 필요합니다. `DescribeEndpoint` API 또는 `describe-endpoint` CLI 명령을 사용하여 데이터 영역 엔드포인트를 찾습니다.

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

   이 명령을 실행하면 아래와 같이 계정별 데이터 플레인 엔드포인트가 반환됩니다.

   ```
   account-specific-prefix.iot.region.amazonaws.com
   ```

4. 명령 주제 구독

   명령 요청 주제를 구독합니다. 실행을 시작하면 메시지 브로커가이 주제에 페이로드를 게시합니다. 디바이스는 명령을 수신하고 처리합니다.

   (선택 사항) 응답 주제(`accepted` 또는 `rejected`)를 구독하여 클라우드 서비스가 디바이스 응답을 수락 또는 거부했는지 확인하는 메시지를 수신합니다.

   대체 예시:

   - *<device>* 대신, 대상 디바이스가 IoT 사물로 등록되었는지 아니면 MQTT 클라이언트로 지정되었는지에 따라 `thing` 또는 `client`를 사용합니다.
   - *<DeviceID>* 대신 대상 디바이스의 고유 식별자를 사용합니다. 이 ID는 고유한 MQTT 클라이언트 ID 또는 사물 이름일 수 있습니다.

   > **Note**
   >
   > 페이로드 유형이 JSON 또는 CBOR이 아닌 경우, 명령 요청 주제에 *<PayloadFormat>* 필드가 표시되지 않을 수 있습니다. 페이로드 형식을 가져오려면 MQTT5를 사용하여 MQTT 메시지 헤더에서 형식 정보를 가져오는 것이 좋습니다. 자세한 내용은 명령 주제 단원을 참조하십시오.

   ```
   $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
   $aws/commands/<devices>/<DeviceID>/executions/+/response/accepted/<PayloadFormat>
   $aws/commands/<devices>/<DeviceID>/executions/+/response/rejected/<PayloadFormat>
   ```

대상 디바이스에 대한 명령 실행 시작 및 모니터링

명령을 생성하고 명령의 대상을 지정한 후, 다음 단계를 수행하여 대상 디바이스에서 실행을 시작할 수 있습니다.

1. 대상 디바이스에서 명령 실행 시작

   Command Hub에서 실행을 시작하거나 계정별 엔드포인트와 함께 `StartCommandExecution` API를 사용합니다. `iot:Data-ATS` 듀얼 스택(IPv4/IPv6) 또는 `iot:Jobs` IPv4에만 사용합니다.

   API는 명령 요청 주제에 페이로드를 게시합니다.

   > **Note**
   >
   > 디바이스가 오프라인 상태이고 MQTT 영구 세션을 사용하는 경우 명령은 메시지 브로커에서 대기합니다. 제한 시간 전에 디바이스가 다시 연결되면 명령을 처리하고 결과를 게시할 수 있습니다. 제한 시간이 만료되면 실행 시간이 초과되고 페이로드가 삭제됩니다.

2. 명령 실행 결과 업데이트

   디바이스는 페이로드를 수신하고, 명령을 처리하고, 지정된 작업을 수행하고, `UpdateCommandExecution` MQTT 기반 API를 사용하여 명령 응답 주제에 결과를 게시합니다. 수락 및 거부된 주제를 구독하면 클라우드 서비스가 응답을 수락 또는 거부했는지 확인하는 메시지가 디바이스에 표시됩니다.

   요청 주제에서 지정한 내용에 따라 *<devices>*는 사물 또는 클라이언트일 수 있으며 *<DeviceID>*는 AWS IoT 사물 이름 또는 MQTT 클라이언트 ID일 수 있습니다.

   > **Note**
   >
   > *<PayloadFormat>*은 명령 응답 주제에서 JSON 또는 CBOR만 가능합니다.

   ```
   $aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/
   response/<PayloadFormat>
   ```

3. (선택 사항) 명령 실행 결과 검색

   AWS IoT 콘솔에서 또는를 사용하여 실행 결과를 검색합니다GetCommandExecution. 최신 정보를 보려면 디바이스가 명령 응답 주제에 결과를 게시해야 합니다. 마지막 업데이트 시간, 결과 및 완료 시간을 포함한 추가 세부 정보를 봅니다.

(선택 사항) 명령 이벤트에 대한 알림 활성화

실행 상태가 변경될 때 알림을 받으려면 명령 이벤트를 구독합니다. 이벤트 메시지 형식 및 페이로드 속성을 포함하여 명령 실행 이벤트에 대한 자세한 내용은 섹션을 참조하세요명령 실행 이벤트.

1. 주제 규칙 생성

   상태 변경 알림에 대한 명령 이벤트 주제를 구독합니다. AWS IoT 콘솔 또는를 사용하여 디바이스 데이터를 AWS Lambda Amazon SQS 및 AWS Step Functions와 같은 다른 AWS IoT 서비스로 라우팅하는 주제 규칙을 생성합니다AWS IoT 규칙 생성.

   이 예제에서 *<CommandID>*를 알림을 수신하려는 명령의 식별자로 바꾸고, *<CommandExecutionStatus>*를 명령 실행의 상태로 바꿉니다.

   ```
   $aws/events/commandExecution/<CommandID>/<CommandExecutionStatus>
   ```

   > **Note**
   >
   > 모든 명령 및 명령 실행 상태에 대한 알림을 받으려면, 와일드카드 문자를 사용하고 다음 주제를 구독하면 됩니다.

   ```
   $aws/events/commandExecution/+/#
   ```

2. 명령 이벤트 수신 및 처리

   명령 푸시 알림을 관리하고 구독된 이벤트를 사용하여 애플리케이션을 빌드합니다.

다음 코드는 수신할 명령 이벤트 알림에 대한 샘플 페이로드를 보여줍니다.

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
```

```
    "status":"FAILED",
    "statusReason": {
         "reasonCode": "DEVICE_TOO_BUSY",
         "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn":"arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
    "targetArn":"arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp":1717708862107
}
```

명령 생성 및 관리

AWS IoT Device Management 명령을 사용하여 재사용 가능한 원격 작업을 구성하거나 디바이스에 즉시 지침을 보냅니다. AWS IoT 콘솔에서 또는를 사용하여 명령을 생성하고 관리합니다 AWS CLI.

명령 작업 생성 및 관리

- 명령 리소스 생성
- 명령에 대한 정보 검색
- 에서 명령 나열 AWS 계정
- 명령 리소스 업데이트
- 명령 리소스의 사용 중지 또는 복원
- 명령 리소스 삭제

명령 리소스 생성

명령을 생성할 때 다음 정보를 제공합니다.

- 일반 정보

  대상 디바이스에서 명령을 실행할 때 고유한 명령 ID를 제공하여 명령을 식별합니다. 선택적으로 관리를 위한 표시 이름, 설명 및 태그를 지정합니다.
- 페이로드

  정적 명령의 경우 디바이스 작업을 정의하는 페이로드를 제공합니다. 올바른 디바이스 해석을 위해 페이로드 형식 유형을 지정합니다.

동적 명령은 페이로드 템플릿 속성을 참조하세요.

- 페이로드 템플릿

  동적 명령의 경우 자리 표시자 및 파라미터가 포함된 payloadTemplate을 제공합니다. 선택적으로 `defaultValue` 및 조건을 제공합니다. AWS IoT Device Management 명령은 런타임 시 자리 표시자를 대체합니다. 누락된 파라미터는 defaultValue를 사용합니다. 모든 값은 정의된 조건을 충족해야 합니다.

  다음과 같은 대/소문자를 구분하는 자리 표시자 유형이 지원됩니다.

  - `${aws:iot:commandexecution::parameter:`*`parameter1`*`}` - 이름이 인 파라미터 값의 자리 표시자입니다`parameter1`.
  - `${aws:iot:commandexecution::executionTimeoutSec}` - 런타임에 제공된 명령 실행 제한 시간 파라미터의 자리 표시자입니다.

페이로드 및 명령 주제

명령 예약 주제는 페이로드 형식 유형에 따라 형식을 사용합니다.

- `application/json` 또는 `application/cbor` 콘텐츠 유형의 경우 다음 요청 주제를 사용합니다.

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
  ```

- 다른 콘텐츠 유형 또는 지정되지 않은 형식의 경우이 요청 주제를 사용합니다. 페이로드 형식이 MQTT 메시지 헤더에 나타납니다.

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request
  ```

명령 응답 주제는 페이로드 유형에 관계없이 `json` 또는 `cbor` 형식을 사용합니다. *`<PayloadFormat>`*은 `json` 또는 여야 합니다`cbor`.

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

명령 리소스 생성(콘솔)

다음 섹션에서는 콘솔에서 페이로드 형식 고려 사항 및 명령 생성에 대해 설명합니다.

주제

- 정적 명령 페이로드 형식
- 동적 명령 페이로드 형식
- 명령을 생성하는 방법(콘솔)

정적 명령 페이로드 형식

페이로드는 최대 32KB의 모든 형식을 지원합니다. 안전하고 올바른 디바이스 해석을 위해 페이로드 형식 유형을 지정합니다.

형식을 사용하여 페이로드 `type/subtype` 형식 유형을 지정합니다(예: `application/json` 또는 `application/cbor`). 기본값: `application/octet-stream`. 지원되는 형식은 일반 MIME 유형을 참조하세요.

동적 명령 페이로드 형식

payloadTemplate은 자리 표시자가 하나 이상 있고 최대 32KB인 유효한 JSON이어야 합니다.

`AwsJsonSubstitution` 프리프로세서의 경우 AWS IoT Device Management 명령은 프리프로세서 구성을 기반으로 JSON 또는 CBOR 형식으로 알림을 보냅니다.

명령을 생성하는 방법(콘솔)

콘솔에서 명령을 생성하려면 Command Hub로 이동하여 다음 단계를 따릅니다.

1. 명령 생성을 선택합니다.
2. 고유한 명령 ID를 지정합니다.
3. (선택 사항) 표시 이름, 설명 및 태그를 지정합니다.
4. 디바이스 작업이 포함된 페이로드 파일을 업로드합니다. 올바른 디바이스 해석을 위해 페이로드 형식 유형을 지정합니다.
5. (선택 사항) 자리 표시자가 있는 JSON 페이로드 템플릿의 경우 파라미터는 편집을 위해 인라인 테이블에 미리 채워집니다.
6. (선택 사항) 파라미터 값 유형(필수), 기본값 및 조건을 구성합니다.
7. 명령 생성을 선택합니다.

명령 리소스 생성(CLI)

`CreateCommand` API 또는 `create-command` CLI 명령을 사용하여 명령을 생성합니다.

주제

- 명령 페이로드
- 샘플 IAM 정책
- 정적 명령 생성 예제
- 동적 명령 생성 예제

명령 페이로드

정적 페이로드 또는 페이로드 템플릿을 제공합니다. 정적 페이로드는 base64로 인코딩됩니다. 페이로드 템플릿의 경우 최종 페이로드는 파라미터 값을 사용하여 런타임에를 생성합니다. 디바이스는 페이로드를 처리하고 지정된 작업을 수행합니다. 올바른 디바이스 수신을 위해 페이로드 콘텐츠 유형을 지정합니다.

> **Note**
>
> 명령 생성 후에는 페이로드를 수정할 수 없습니다. 새 명령을 생성하여 페이로드를 수정합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 `CreateCommand` 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *`region`* 대신 AWS 리전(예: *`us-east-1`*)로 대체합니다.
- *`account-id`* 대신 AWS 계정 번호(예: *`123456789012`*)로 대체합니다.
- *`command-id`*와 같이 AWS IoT 명령 ID의 고유 식별자가 있는 *`LockDoor`*. 둘 이상의 명령을 전송하려면 IAM 정책의 리소스 섹션에서 이러한 명령을 지정할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:CreateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
```

```
    }
}
```

정적 명령 생성 예제

다음 예제에서는 정적 명령을 생성하는 방법을 보여줍니다. 애플리케이션에 따라 다음을 대체합니다.

- *<command-id>*를 명령의 고유 식별자로 대체합니다. 예를 들어 집의 문을 잠그려면를 지정할 수 있습니다*LockDoor*. UUID를 사용할 것을 권장합니다. 영숫자, "-" 및 "_"를 사용할 수도 있습니다.
- (선택 사항) *<display-name>*와 *<description>*는 *Lock the doors of my home*와 같이 기억하기 쉬운 이름과 명령에 대한 의미 있는 설명을 제공하는 데 사용할 수 있는 선택적 필드입니다.
- namespace는 명령의 네임스페이스를 지정하는 데 사용할 수 있습니다. 반드시 AWS-IoT이어야 합니다. 에이 기능을 사용하는 방법에 대한 자세한 내용은 원격 명령을 AWS IoT FleetWise참조하세요.
- payload에는 명령을 실행할 때 사용하려는 페이로드에 대한 정보와 콘텐츠 유형이 포함되어 있습니다.

```
aws iot create-command \
    --command-id <command-id> \
    --display-name <display-name> \
    --description <description> \
    --namespace AWS-IoT \
    --payload
 '{"content":"eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=","contentType":"application/json"}'
```

이 명령을 실행하면 명령의 ID 및 ARN(Amazon 리소스 이름)이 포함된 응답이 생성됩니다. 예를 들어, 생성 중에 *LockDoor* 명령을 지정한 경우 다음은 명령 실행의 샘플 출력을 보여줍니다.

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor"
}
```

동적 명령 생성 예제

다음 예제에서는 동적 명령을 생성하는 방법을 보여줍니다. 애플리케이션에 따라 다음을 대체합니다.

- *<command-id>*를 명령의 고유 식별자로 대체합니다. 예를 들어 조명 전원 상태를 설정하려면를 지정할 수 있습니다*Light_Power_Status*. UUID를 사용할 것을 권장합니다. 영숫자, "-" 및 "_"를 사용할 수도 있습니다.
- (선택 사항) *<display-name>*와 *<description>*는 *Turn a light ON or OFF*와 같이 기억하기 쉬운 이름과 명령에 대한 의미 있는 설명을 제공하는 데 사용할 수 있는 선택적 필드입니다.
- namespace는 명령의 네임스페이스를 지정하는 데 사용할 수 있습니다. 반드시 AWS-IoT이어야 합니다. 에서이 기능을 사용하는 방법에 대한 자세한 내용은 원격 명령을 AWS IoT FleetWise참조하세요.
- payloadTemplate 에는 placehodler가 있는 JSON 형식의 플레이오드 템플릿이 포함되어 있습니다.
- preprocessor 에는 payloadTemplate 처리 방법을 결정하는 프리프로세서에 대한 구성이 포함되어 있습니다.
- mandatoryParameter 에는 payloadTemplate의 자리 표시자, 유형, 기본값 및 조건에 해당하는 파라미터가 포함되어 있습니다.

```
aws iot create-command \
    --command-id Light_Power_Status \
    --description "Turn a light ON or OFF" \
    --namespace AWS-IoT \
    --payload-
template '{"powerStatus":"${aws:iot:commandexecution::parameter:powerStatus}"}' \
    --preprocessor awsJsonSubstitution={outputFormat=JSON} \
    --mandatory-parameters "name=powerStatus, defaultValue={S=OFF},
 valueConditions=[{comparisonOperator=IN_SET, operand={strings=['ON','OFF']}}]"
```

이 명령을 실행하면 명령의 ID 및 ARN(Amazon 리소스 이름)이 포함된 응답이 생성됩니다. 예를 들어, 생성 중에 *Light_Power_Status* 명령을 지정한 경우 다음은 명령 실행의 샘플 출력을 보여줍니다.

```
{
    "commandId": "Light_Power_Status",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status"
}
```

명령에 대한 정보 검색

명령을 생성한 후 AWS IoT 콘솔에서 또는 AWS CLI를 사용하여 해당 명령에 대한 정보를 검색할 수 있습니다. 다음 정보를 획득할 수 있습니다.

- 명령 ID, Amazon 리소스 이름(ARN), 명령에 지정한 표시 이름 및 설명
- 대상 디바이스에서 명령을 실행할 수 있는지 또는 명령이 사용 중지되거나 삭제되었는지 여부를 나타내는 명령 상태
- 제공한 페이로드 또는 payloadTemplate입니다.
- 제공한 프리프로세서입니다.
- 제공한 mandatoryParameters입니다.
- 명령이 생성되고 마지막으로 업데이트된 시간

명령 리소스 검색(콘솔)

콘솔에서 명령을 검색하려면 콘솔의 Command Hub로 이동한 AWS IoT 다음 생성한 명령을 선택하여 세부 정보를 확인합니다.

명령 세부 정보 외에도 대상 디바이스에서 명령 실행에 대한 정보를 제공하는 명령 기록을 볼 수 있습니다. 디바이스에서 이 명령을 실행한 후 이 탭에서 실행에 대한 정보를 확인할 수 있습니다.

명령 리소스 검색(CLI)

`GetCommand` HTTP 컨트롤 플레인 API 작업 또는 `get-command` AWS CLI 명령을 사용하여 명령 리소스에 대한 정보를 검색합니다. `CreateCommand` API 요청 또는 `create-command` CLI를 사용하여 명령을 이미 생성했어야 합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 `GetCommand` 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *`region`* 대신 AWS 리전(예: `us-east-1`)로 대체합니다.
- *`account-id`* 대신 AWS 계정 번호(예: *`123456789023`*)로 대체합니다.
- *`command-id`* *`LockDoor`* 또는와 같은 고유한 명령 식별자가 있는 AWS IoT *`Light_Power_Status`*. 둘 이상의 명령을 검색하려면 IAM 정책의 리소스 섹션에서 이러한 명령을 지정할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:GetCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

명령 검색 예제(AWS CLI)

다음 예제에서는를 사용하여 명령에 대한 정보를 검색하는 방법을 보여줍니다get-command AWS CLI. 애플리케이션에 따라 *<command-id>*를 정보를 검색할 명령의 식별자로 바꿉니다. create-command CLI의 응답에서 이 정보를 획득할 수 있습니다.

```
aws iot get-command --command-id <command-id>
```

이 명령을 실행하면 응답이 생성되며, 이 응답에는 명령, 페이로드, 및 응답이 생성되고 마지막으로 업데이트된 시간에 대한 정보가 포함됩니다. 또한 명령이 사용되지 않는지 또는 삭제 중인지 여부를 나타내는 정보도 제공합니다.

예를 들어, 다음 코드는 샘플 응답입니다.

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:<region>:<account>:command/LockDoor",
    "namespace": "AWS-IoT",
    "payload":{
        "content": "eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=",
        "contentType": "application/json"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-03-23T00:50:10.095000-07:00",
    "deprecated": false,
    "pendingDeletion": false
}
```

에서 명령 나열 AWS 계정

명령을 생성한 후 계정에서 생성한 명령을 볼 수 있습니다. 목록에서 다음에 대한 정보를 확인할 수 있습니다.

- 명령 ID 및 명령에 지정한 표시 이름
- 명령의 Amazon 리소스 이름(ARN)
- 대상 디바이스에서 명령을 실행할 수 있는지 또는 명령이 사용 중지되었는지 여부를 나타내는 명령 상태

> **Note**
>
> 계정에서 삭제 중인 명령은 목록에 표시되지 않습니다. 명령이 삭제 보류 중인 경우에도 명령 ID를 사용하여 이러한 명령의 세부 정보를 볼 수 있습니다.

- 명령이 생성되고 마지막으로 업데이트된 시간

계정의 명령 나열(콘솔)

AWS IoT 콘솔에서 Command Hub로 이동하여 생성한 명령 목록과 세부 정보를 찾을 수 있습니다.

계정의 명령 나열(CLI)

생성한 명령을 나열하려면 `ListCommands` API 작업 또는 `list-commands` CLI를 사용합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 `ListCommands` 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *`region`* 대신 AWS 리전(예: `us-east-1`)로 대체합니다.
- *`account-id`* 대신 AWS 계정 번호(예: *`123456789012`*)로 대체합니다.

```
{
```

```
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:ListCommands",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/*"
    }
}
```

계정의 명령 나열 예제

다음 명령은 계정의 명령을 나열하는 방법을 보여줍니다.

```
aws iot list-commands --namespace "AWS-IoT"
```

이 명령을 실행하면 생성한 명령 목록, 명령이 생성된 시간 및 마지막으로 업데이트된 시간이 포함된 응답이 생성됩니다. 여기서는 또한 명령이 사용되지 않는지 또는 대상 디바이스에서 명령을 실행할 수 있는지 여부를 나타내는 명령 상태 정보를 제공합니다. 다양한 상태 및 상태 이유에 대한 자세한 내용은 명령 실행 상태 섹션을 참조하세요.

명령 리소스 업데이트

명령을 생성한 후 명령의 표시 이름과 설명을 업데이트할 수 있습니다.

> **Note**
>
> 명령의 페이로드는 업데이트할 수 없습니다. 이 정보를 업데이트하거나 수정된 페이로드를 사용하려면, 새 명령을 생성해야 합니다.

명령 리소스 업데이트(콘솔)

콘솔에서 명령을 업데이트하려면 콘솔의 Command Hub로 이동하여 다음 단계를 AWS IoT 수행합니다.

1. 기존 명령 리소스를 업데이트하려면 업데이트할 명령을 선택한 다음 작업에서 편집을 선택합니다.
2. 사용하려는 표시 이름 및 설명과 이름-값 페어를 명령의 태그로 지정합니다.
3. 편집을 선택하여 명령을 새 설정과 함께 저장합니다.

명령 리소스 업데이트(CLI)

UpdateCommand 컨트롤 플레인 API 작업 또는 update-command AWS CLI 를 사용하여 명령 리소스를 업데이트합니다. 이 API를 사용하여 다음을 수행할 수 있습니다.

- 생성한 명령의 표시 이름과 설명을 편집합니다.
- 명령 리소스를 사용 하지 않거나, 이미 사용 중지된 명령을 복원합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 UpdateCommand 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *region* 대신 AWS 리전(예: us-east-1)로 대체합니다.
- *account-id* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.
- *command-id* 대신 AWS IoT 고유 명령 식별자(예: *LockDoor*)로 대체합니다. 둘 이상의 명령을 검색하려면 IAM 정책의 리소스 섹션에서 이러한 명령을 지정할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:UpdateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

명령에 대한 정보 업데이트 예제(AWS CLI)

다음 예제에서는 명령을 사용하여 update-command AWS CLI 명령에 대한 정보를 업데이트하는 방법을 보여줍니다. 이 API를 사용하여 명령 리소스를 사용 중지하거나 복원하는 방법에 대한 자세한 내용은 명령 리소스 업데이트(CLI) 섹션을 참조하세요.

이 예제에서는 명령의 표시 이름과 설명을 업데이트하는 방법을 보여줍니다. 애플리케이션에 따라 *<command-id>*를 정보를 검색할 명령의 식별자로 바꿉니다.

```
aws iot update-command \
    --command-id <command-id>
    --displayname <display-name> \
    --description <description>
```

이 명령을 실행하면 명령에 대해 업데이트된 정보와 마지막으로 업데이트된 시간이 포함하여 응답이 생성됩니다. 다음 코드는 AC를 끄는 명령의 표시 이름 및 설명을 업데이트하기 위한 샘플 요청 및 응답을 보여줍니다.

```
aws iot update-command \
    --command-id <LockDoor> \
    --displayname <Secondary lock door> \
    --description <Locks doors to my home>
```

이 명령을 실행하면 다음 응답이 생성됩니다.

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor",
    "displayName": "Secondary lock door",
    "description": "Locks doors to my home",
    "lastUpdatedAt": "2024-05-09T23:15:53.899000-07:00"
}
```

명령 리소스의 사용 중지 또는 복원

명령을 생성한 후, 더 이상 명령을 계속 사용하지 않으려면 사용되지 않음 상태로 표시할 수 있습니다. 명령을 사용 중지하면 대기 중인 모든 명령 실행이 터미널 상태에 도달할 때까지 대상 디바이스에서 계속 실행됩니다. 명령이 사용 중지되면, 대상 디바이스로 새 명령 실행을 보내는 등 해당 명령을 사용하기 위해서는 명령을 복원해야 합니다.

> **Note**
>
> 사용 중지된 명령은 편집하거나 새 실행을 실행할 수 없습니다. 디바이스에서 새 명령을 실행하려면, 명령 상태가 사용 가능으로 변경되도록 디바이스를 복원해야 합니다.

명령 사용 중단 및 복원과 이에 대한 고려 사항에 대한 자세한 내용은 명령 리소스 사용 중지 섹션을 참조하세요.

명령 리소스 삭제

명령을 더 이상 사용하지 않을 경우, 계정에서 영구적으로 제거할 수 있습니다. 삭제 작업이 성공한 경우:

- 명령이 최대 제한 시간인 12시간보다 긴 기간 동안 사용되지 않았던 경우, 해당 명령이 즉시 삭제됩니다.
- 명령이 사용되지 않는 상태거나 최대 제한 시간보다 짧은 기간 동안 사용되지 않았던 경우, 해당 명령은 `pending deletion` 상태가 됩니다. 최대 제한 시간인 12시간이 지나면 계정에서 해당 명령이 자동으로 제거됩니다.

> **Note**
>
> 보류 중인 명령 실행이 있더라도 명령이 삭제될 수 있습니다. 명령은 삭제 보류 중 상태가 되고 계정에서 자동으로 제거됩니다.

명령 리소스 삭제(콘솔)

콘솔에서 명령을 삭제하려면 콘솔의 Command Hub로 이동하여 다음 단계를 AWS IoT 수행합니다.

1. 삭제할 명령을 선택한 다음, 작업에서 삭제를 선택합니다.
2. 명령을 정말로 삭제할 것인지 다시 한 번 묻는 메시지가 나오면 확인 후 삭제를 선택합니다.

명령은 삭제 대상으로 표시되고 12시간 후에 계정에서 영구적으로 제거됩니다.

명령 리소스 삭제(CLI)

`DeleteCommand` HTTP 컨트롤 플레인 API 작업 또는 `delete-command` AWS CLI 명령을 사용하여 명령 리소스를 삭제합니다. 삭제 작업이 성공하면 HTTP `statusCode`가 204 또는 202로 표시되고 최대 제한 시간인 12시간이 지나면 계정에서 명령이 자동으로 삭제됩니다. 204 상태의 경우 명령이 삭제되었음을 나타냅니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 `DeleteCommand` 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *region* 대신 AWS 리전(예: us-east-1)로 대체합니다.
- *account-id* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.
- *command-id* 대신 AWS IoT 고유 명령 식별자(예: *LockDoor*)로 대체합니다. 둘 이상의 명령을 검색하려면 IAM 정책의 리소스 섹션에서 이러한 명령을 지정할 수 있습니다.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:DeleteCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

명령 삭제 예제(AWS CLI)

다음 예제에서는 명령을 사용하여 delete-command AWS CLI 명령을 삭제하는 방법을 보여줍니다. 애플리케이션에 따라 *<command-id>*를 삭제하려는 명령의 식별자로 바꿉니다.

```
aws iot delete-command --command-id <command-id>
```

API 요청이 성공하면 명령은 상태 코드 202 또는 204를 생성합니다. GetCommand API를 사용하여 명령이 계정에 더 이상 존재하지 않는지 확인할 수 있습니다.

명령 실행 시작 및 모니터링

명령을 생성한 후 대상 디바이스에서 실행을 시작합니다. 디바이스는 결과를 업데이트하고 MQTT 예약 주제에 상태를 게시합니다. 계정에서 실행 상태를 검색하고 모니터링합니다.

AWS IoT 콘솔 또는를 사용하여 명령을 시작하고 모니터링합니다 AWS CLI.

명령 작업 시작 및 모니터링

- 명령 실행 시작
- 명령 실행 결과 업데이트

- 명령 실행 검색
- MQTT 테스트 클라이언트를 사용하여 명령 업데이트 보기
- 에서 명령 실행 나열 AWS 계정
- 명령 실행 삭제

명령 실행 시작

> **Important**
>
> 안전하고 관련 법률을 준수하는 방식으로 명령을 배포하는 것은 전적으로 사용자의 책임입니다.

실행을 시작하기 전에 다음을 확인합니다.

- 페이로드 정보로 AWS IoT 네임스페이스에 명령을 생성했습니다. 실행을 시작할 때 디바이스는 페이로드 지침을 처리하고 지정된 작업을 수행합니다. 명령 생성은 섹션을 참조명령 리소스 생성하세요.
- 디바이스가 명령에 대한 MQTT 예약 주제를 구독했습니다. 실행을 시작하면 페이로드 정보가이 예약된 MQTT 요청 주제에 게시됩니다.

  *<devices>*는 사물 또는 MQTT 클라이언트일 수 있습니다. *<DeviceID>*는 사물 이름 또는 클라이언트 ID입니다. 지원되는 *<PayloadFormat>* 값: JSON 및 CBOR. 자세한 내용은 명령 주제 단원을 참조하십시오.

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
  ```

  비 JSON/CBOR *<PayloadFormat>*의 경우 다음 명령 주제 형식을 사용합니다.

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request
  ```

대상 디바이스 고려 사항

명령을 수신하고 실행할 대상 디바이스를 지정합니다. 등록된 디바이스에는 사물 이름을 사용하고 등록되지 않은 디바이스에는 클라이언트 ID를 사용합니다. 페이로드를 수신한 후 디바이스는 명령을 실행하고 지정된 작업을 수행합니다.

AWS IoT 사물

대상 디바이스는 AWS IoT 레지스트리에 등록된 사물일 수 있습니다. 사물은 디바이스 검색 및 관리를 간소화합니다.

Connect 디바이스 페이지에서 또는를 사용하여 디바이스를 사물로 등록합니다`CreateThing`. 를 사용하여 Thing Hub 또는에서 기존 사물을 찾습니다`DescribeThing`. 등록 세부 정보는 레지스트리로 사물 관리를 참조하세요.

클라이언트 ID입니다

등록되지 않은 디바이스의 경우 클라이언트 ID를 사용합니다.

클라이언트 ID는 디바이스에 할당하는 고유 식별자입니다. MQTT 프로토콜에 정의되어 있으며 영숫자 문자, 밑줄 또는 대시를 포함합니다. 에 연결하는 각 디바이스에는 고유한 클라이언트 ID가 AWS IoT 필요합니다.

> **Note**
>
> - 등록된 사물의 경우 클라이언트 ID가 사물 이름과 일치할 수 있습니다.
> - 특정 클라이언트 ID를 대상으로 지정할 때 디바이스는 페이로드를 수신하려면 해당 클라이언트 ID를 AWS IoT 사용하여에 연결해야 합니다.

클라이언트 ID는에 연결할 때 사용하는 MQTT 클라이언트 ID입니다 AWS IoT Core.는이 ID를 AWS IoT 사용하여 디바이스를 식별하고 연결 및 구독을 관리합니다.

명령 실행의 제한 시간 고려 사항

제한 시간은 디바이스가 실행 결과를 제공하는 기간(초)을 지정합니다.

실행을 생성한 후 타이머가 시작됩니다. 디바이스가 오프라인 상태가 되거나 제한 시간 내에 결과를 보고하지 못하면 실행이 상태로 시간 초과됩니다`TIMED_OUT`.

기본값: 10초. 최대: 12시간.

제한 시간 값 및 **`TIMED_OUT`** 명령 실행 상태

클라우드와 디바이스 모두 제한 시간을 보고할 수 있습니다.

명령을 전송한 후 타이머가 시작됩니다. 제한 시간 내에 디바이스 응답이 도착하지 않으면 클라우드는 실행 상태를 사유 코드 `TIMED_OUT`와 함께 로 설정합니다`$NO_RESPONSE_FROM_DEVICE`.

이는 다음과 같은 경우에 발생합니다.

- 실행 중에 디바이스가 오프라인 상태가 되었습니다.
- 디바이스가 제한 시간 내에 실행을 완료하지 못했습니다.
- 디바이스가 제한 시간 내에 상태를 보고하지 못했습니다.

이 경우에 실행 상태가 클라우드에서 `TIMED_OUT`으로 보고되면 명령 실행은 비터미널 상태입니다. 디바이스는 상태를 `SUCCEEDED`, `FAILED`또는 터미널 상태로 재정의하는 응답을 게시할 수 있습니다`REJECTED`. 그러면 명령 실행이 터미널이 되고 추가 업데이트를 수락하지 않습니다.

또한 디바이스는 명령을 실행할 때 제한 시간이 발생했음을 보고하여 클라우드에서 시작된 `TIMED_OUT` 상태를 업데이트할 수 있습니다. 이 경우 명령 실행 상태는 로 유지`TIMED_OUT`되지만 `statusReason` 객체는 디바이스에서 보고한 정보를 기반으로 업데이트됩니다. 그러면 명령 실행이 터미널이 되고 추가 업데이트가 수락되지 않습니다.

MQTT 영구 세션 사용

AWS IoT Device Management 명령 기능과 함께 사용하도록 MQTT 영구 세션을 구성할 수 있습니다. 이 기능은 디바이스가 오프라인 상태가 되고 디바이스가 제한 시간 전에 다시 온라인 상태가 될 때 명령을 계속 수신하고 지정된 지침을 수행하는지 확인하려는 경우에 특히 유용합니다.

기본적으로 MQTT 영구 세션 만료는 60분으로 설정됩니다. 명령 실행 제한 시간이이 기간을 초과하는 값으로 구성된 경우 메시지 브로커가 60분 이상 실행되는 명령 실행을 거부하고 실패할 수 있습니다. 지속 시간이 60분을 초과하는 명령을 실행하려면 영구 세션 만료 시간을 늘리도록 요청할 수 있습니다.

> **Note**
>
> MQTT 영구 세션 기능을 올바르게 사용하고 있는지 확인하려면 Clean Start 플래그를 0으로 설정합니다. 자세한 내용은 MQTT 영구 세션을 참조하세요.

명령 실행 시작(콘솔)

콘솔에서 명령 실행을 시작하려면 콘솔의 Command Hub 페이지로 이동하여 다음 단계를 AWS IoT 수행합니다.

1. 생성한 명령을 실행하려면 명령 실행을 선택합니다.
2. MQTT 예약 주제 및 해당하는 경우 파라미터를 포함하여 생성한 명령에 대한 정보를 검토합니다.

동적 명령의 경우 파라미터 값을 입력하거나 기본값으로 둡니다. 기본값이 없는 파라미터의 경우 이 실행의 일부로 전송할 값을 제공해야 합니다.

3. 명령을 수신하고 실행할 대상 디바이스를 지정합니다. 디바이스가에 등록된 경우 AWS IoT 사물로 지정하거나 디바이스가 아직 등록되지 않은 경우 클라이언트 ID를 AWS IoT사용할 수 있습니다. 자세한 내용은 대상 디바이스 고려 사항 섹션을 참조하세요.
4. (선택 사항) 제한 시간 전에 명령을 실행할 기간을 결정하는 명령의 제한 시간 값을 구성합니다. 명령을 60분 이상 실행해야 하는 경우, MQTT 영구 세션 만료 시간을 늘려야 할 수 있습니다. 자세한 내용은 명령 실행의 제한 시간 고려 사항 단원을 참조하십시오.
5. 명령 실행을 선택합니다.

명령 실행 시작(AWS CLI)

`StartCommandExecution` HTTP 데이터 플레인 API 작업을 사용하여 명령 실행을 시작합니다. API 요청 및 응답은 명령 실행 ID와 상관관계를 가집니다. 디바이스가 명령 실행을 완료한 후, 명령 응답 주제에 메시지를 게시하여 상태 및 실행 결과를 클라우드에 보고할 수 있습니다. 사용자 지정 응답 코드의 경우 소유한 애플리케이션 코드는 응답 메시지를 처리하고 결과를 게시할 수 있습니다 AWS IoT.

디바이스가 명령 요청 주제를 구독한 경우에 `StartCommandExecution` API는 페이로드 메시지를 주제에 게시합니다. 페이로드는 원하는 형식을 사용할 수 있습니다. 자세한 내용은 명령 페이로드 단원을 참조하십시오.

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
```

페이로드 형식이 JSON 또는 CBOR이 아닌 경우 다음은 명령 요청 주제의 형식을 보여줍니다.

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 `StartCommandExecution` 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *region* 대신 AWS 리전(예: `us-east-1`)로 대체합니다.
- *account-id* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.

- *command-id*와 같이 AWS IoT 명령의 고유 식별자가 있는 *LockDoor*. 둘 이상의 명령을 보내려면 IAM 정책에서 이러한 명령을 지정할 수 있습니다.
- *devices* 디바이스가 AWS IoT 사물로 등록되었는지 또는 MQTT 클라이언트로 지정되었는지에 client 따라 thing 또는가 있는 .
- *device-id* thing-name 또는를 사용합니다 AWS IoT client-id.

```
{
  "Effect": "Allow",
  "Action": [
      "iot:StartCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

에 지원되는 조건 키 목록을 보려면 IAM 사용 설명서의에 대한 조건 키를 StartCommandExecution참조하세요. AWS IoT

계정별 데이터 플레인 엔드포인트 획득

API 명령을 실행하기 전에 엔드포인트에 대한 계정별 엔드포인트 URL을 얻어야 합니다. 듀얼 스택 엔드포인트(IPv4 및 IPv6)를 사용하는 경우 iot:Data-ATS를 사용합니다. iot:Jobs 엔드포인트는 IPv4 전용입니다. 예를 들면 다음 명령을 실행하는 경우

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

아래 샘플 응답과 같이 계정별 엔드포인트 URL을 반환합니다.

```
{
    "endpointAddress":
    "<account-specific-prefix>-ats.iot.<region>.api.com"
}
```

명령 실행 시작 예제(AWS CLI)

다음 예제에서는 명령을 사용하여 start-command-execution AWS CLI 명령 실행을 시작하는 방법을 보여줍니다.

대체 예시:

- *<command-arn>*대신 실행하려는 명령의 ARN을 사용합니다. create-command CLI 명령의 응답에서 이 정보를 얻을 수 있습니다. 예를 들어, 조향 모드를 변경하는 명령을 실행하는 경우 arn:aws:iot:*region*:*account-id*:command/*SetComfortSteeringMode*를 사용합니다.
- *<target-arn>*대신 명령을 실행하려는 IoT 사물 또는 MQTT 클라이언트일 수 있는 대상 디바이스의 사물 ARN을 사용합니다. 예를 들어, 대상 디바이스 myRegisteredThing에 대해 명령을 실행하는 경우 arn:aws:iot:*region*:*account-id*:thing/*myRegisteredThing*을 사용합니다.
- *<endpoint-url>* 대신 계정별 데이터 플레인 엔드포인트 획득에서 얻은 계정별 엔드포인트를 사용하며 https://가 접두사로 붙습니다. 예를 들어 https://*123456789012abcd*.jobs.iot.*us-east-1*.amazonaws.com입니다.
- (선택 사항) StartCommandExecution API 작업을 수행할 때 추가 파라미터로 executionTimeoutSeconds를 지정할 수도 있습니다. 이 선택 필드는 디바이스가 명령 실행을 완료해야 하는 시간을 초 단위로 지정합니다. 기본값은 10초입니다. 명령 실행 상태가 CREATED이면 타이머가 시작됩니다. 타이머가 만료되기 전에 명령 실행 결과가 수신되지 않으면 상태가 자동으로 TIMED_OUT으로 변경됩니다.
- ```
  aws iot-jobs-data start-command-execution \
      --command-arn <command-arn>  \
      --target-arn <target-arn> \
      --endpoint <endpoint-url> \
      --execution-timeout-seconds 900
  ```
- (선택 사항) 동적 명령의 경우 대체에 사용할 파라미터와 해당 값을 지정합니다. 명령 생성 시 defaultValue가 설정되지 않은 파라미터의 값을 제공해야 합니다. 파라미터에 defaultValue가 있는 경우 여기에 제공된 파라미터 값이 우선합니다. valueConditions가 설정된 파라미터의 경우 여기에 제공된 파라미터 값이 조건을 충족해야 합니다.

  Light_Power_Status 동적 명령 예제 기반:

- ```
  aws iot-jobs-data start-command-execution \
      --command-arn arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status  \
      --target-arn arn:aws:iot:us-east-1:123456789012:thing/exampleThing \
      --endpoint <endpoint-url> \
      --execution-timeout-seconds 900 \
      --parameters "powerStatus={S=ON}"
  ```

이 명령을 실행하면 명령 실행 ID가 반환됩니다. 이 ID를 사용하여 명령 실행 상태, 세부 정보 및 명령 실행 기록을 쿼리할 수 있습니다.

> **Note**
>
> 명령이 더 이상 사용되지 않는 경우, `StartCommandExecution` API 요청은 검증 예외로 실패합니다. 이 오류를 해결하려면, 먼저 `UpdateCommand` API를 사용하여 명령을 복원한 다음 `StartCommandExecution` 요청을 수행합니다.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542"
}
```

명령 실행 결과 업데이트

`UpdateCommandExecution` MQTT 데이터 플레인 API 작업을 사용하여 명령 실행의 상태 또는 결과를 업데이트합니다.

> **Note**
>
> 이 API를 사용하기 전에 다음 사항을 주의하세요.
>
> - 디바이스가 MQTT 연결을 설정하고 명령 요청 및 응답 주제를 구독해야 합니다. 자세한 내용은 상위 수준 명령 워크플로 단원을 참조하십시오.
> - `StartCommandExecution` API 작업을 사용하여 이 명령을 이미 실행했어야 합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에 이러한 작업을 수행할 수 있는 권한을 부여해야 합니다. 다음은 디바이스에 작업을 수행할 권한을 부여하는 정책의 예입니다. `UpdateCommandExecution` MQTT 작업을 수행할 수 있는 사용자 권한을 허용하는 추가 샘플 IAM 정책은 섹션을 참조하세요연결 및 게시 정책 예제.

대체 예시:

- *Region*와 AWS 리전같은를 사용합니다`us-east-1`.

- *AccountID* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.
- *ThingName*와 같이 명령 실행을 대상으로 하는 AWS IoT 사물의 이름을 사용합니다*myRegisteredThing*.
- *commands-request-topic* AWS IoT 명령 요청 및 응답 주제의 이름과 *commands-response-topic* 함께 및 . 자세한 내용은 상위 수준 명령 워크플로 단원을 참조하십시오.

MQTT 클라이언트 ID에 대한 샘플 IAM 정책

다음 코드는 MQTT 클라이언트 ID를 사용할 때의 샘플 디바이스 정책을 보여줍니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/request",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/request/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/rejected/json"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/request",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/request/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/${iot:ClientId}"
    }
  ]
}
```

IoT 사물에 대한 샘플 IAM 정책

다음 코드는 AWS IoT 사물을 사용할 때의 샘플 디바이스 정책을 보여줍니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
```

```
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/request",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/request/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/request",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/request/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/${iot:ClientId}"
    }
  ]
}
```

UpdateCommandExecution API 사용 방법

요청 주제에서 명령 실행이 수신되면 디바이스는 명령을 처리합니다. 그런 다음 UpdateCommandExecution API를 사용하여 명령 실행의 상태 및 결과를 다음 응답 주제로 업데이트합니다.

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

이 예제에서 *<DeviceID>*는 대상 디바이스의 고유 식별자이고, *<execution-id>*는 대상 디바이스에서 명령 실행의 식별자입니다. *<PayloadFormat>*은 JSON 또는 CBOR일 수 있습니다.

> **Note**
>
> 디바이스를에 등록하지 않은 경우 사물 이름 대신 클라이언트 ID를 식별자로 사용할 AWS IoT 수 있습니다.

```
$aws/commands/clients/<ClientID>/executions/<ExecutionId>/response/<PayloadFormat>
```

디바이스의 실행 상태 업데이트 보고

디바이스는 API를 사용하여 명령 실행에 다음과 같은 상태 업데이트를 보고할 수 있습니다. 이러한 상태에 대한 자세한 내용은 명령 실행 상태 섹션을 참조하세요.

- IN_PROGRESS: 디바이스가 명령 실행을 시작하면 상태를 IN_PROGRESS로 업데이트할 수 있습니다.
- SUCCEEDED: 디바이스가 명령을 성공적으로 처리하고 실행을 완료하면 디바이스는 응답 주제에 SUCCEEDED로 메시지를 게시할 수 있습니다.
- FAILED: 디바이스가 명령을 실행하지 못한 경우 응답 주제에 FAILED로 메시지를 게시할 수 있습니다.
- REJECTED: 디바이스가 명령을 수락하지 못한 경우 응답 주제에 REJECTED로 메시지를 게시할 수 있습니다.
- TIMED_OUT: 다음과 같은 이유로 명령 실행 상태가 TIMED_OUT으로 변경될 수 있습니다.
  - 명령 실행 결과가 수신되지 않았습니다. 이는 지정된 기간 내에 실행이 완료되지 않았거나, 디바이스가 상태 정보를 응답 주제에 게시하지 못했기 때문에 발생할 수 있습니다.
  - 해당 디바이스는 명령을 실행하려고 시도할 때 시간 초과가 발생했음을 보고합니다.

TIMED_OUT 상태에 대한 자세한 내용은 제한 시간 값 및 TIMED_OUT 명령 실행 상태 섹션을 참조하세요.

UpdateCommandExecution API 사용 시 고려 사항

다음은 UpdateCommandExecution API를 사용할 때 고려해야 할 몇 가지 중요한 사항입니다.

- 디바이스는 선택적 statusReason 객체를 사용하여 실행에 대한 추가 정보를 제공할 수 있습니다. 디바이스에서이 객체를 제공하는 경우 객체의 reasonCode 필드는 필수이지만 reasonDescription 필드는 선택 사항입니다.
- 디바이스가 statusReason 객체를 사용하는 경우는 패턴을 사용해야 [A-Z0-9_-]+ 하며 길이가 64자를 초과하지 않아야 reasonCode 합니다. 를 제공하는 경우 길이reasonDescription가 1,024자를 초과하지 않아야 합니다. 줄 바꿈과 같은 제어 문자를 제외한 모든 문자를 사용할 수 있습니다.
- 디바이스는 선택적 result 객체를 사용하여 원격 함수 직접 호출의 반환 값과 같은 명령 실행 결과에 대해 정보를 제공할 수 있습니다. result를 제공하는 경우 하나 이상의 항목이 필요합니다.
- result 필드에서 항목을 키-값 페어로 지정합니다. 각 항목에 대해 데이터 유형 정보를 문자열, 부울 또는 바이너리로 지정해야 합니다. 문자열 데이터 형식은 s 키를 사용하고, 부울 데이터 형식은 b 키를 사용하며, 바이너리 데이터 형식은 bin 키를 사용해야 합니다. 이러한 키가 소문자인지 확인합니다.
- UpdateCommandExecution API를 실행할 때 오류가 발생하면 Amazon CloudWatch의 AWSIoTLogsV2 로그 그룹에서 오류를 볼 수 있습니다. 로깅 활성화 및 로그 보기에 대한 자세한 내용은 AWS IoT 로깅 구성 섹션을 참조하세요.

UpdateCommandExecution API 예제

다음 코드는 자동차 배터리 백분율을 예로 들어서 디바이스가 UpdateCommandExecution API를 사용하여 실행 상태를 보고하고, statusReason 필드를 사용하여 상태에 대한 추가 정보를 제공하고, 결과 필드를 사용하여 실행 결과에 대한 정보를 제공하는 방법을 보여줍니다.

```
{
  "status": "IN_PROGRESS",
  "statusReason": {
    "reasonCode": "200",
    "reasonDescription": "Execution_in_progress"
  },
  "result": {
        "car_battery": {
```

```
            "s": "car battery at 50 percent"
        }
    }
}
```

명령 실행 검색

명령을 실행한 후 AWS IoT 콘솔 및를 사용하여 명령 실행에 대한 정보를 검색할 수 있습니다 AWS CLI. 다음 정보를 획득할 수 있습니다.

> **Note**
>
> 최신 명령 실행 상태를 검색하려면 디바이스가 아래 설명과 같이 `UpdateCommandExecution` MQTT API를 사용하여 상태 정보를 응답 주제에 게시해야 합니다. 디바이스가 이 주제에 게시될 때까지 `GetCommandExecution` API는 상태를 `CREATED` 또는 `TIMED_OUT`으로 보고합니다.

생성하는 각 명령 실행에는 다음이 포함됩니다.

- 명령 실행의 고유 식별자인 실행 ID
- 명령 실행의 상태. 대상 디바이스에서 명령을 실행하면 명령 실행은 `CREATED` 상태가 됩니다. 그런 다음 아래에 설명된 대로 다른 명령 실행 상태로 전환할 수 있습니다.
- 명령 실행의 결과.
- 고유한 명령 ID 및 실행이 생성된 대상 디바이스
- 명령 실행이 생성된 시간을 보여주는 시작 날짜

명령 실행 검색(콘솔)

다음 방법 중 하나를 사용하여 콘솔에서 명령 실행을 검색할 수 있습니다.

- Command Hub 페이지에서

  AWS IoT 콘솔의 Command Hub 페이지로 이동하여 다음 단계를 수행합니다.

  1. 대상 디바이스에서 실행을 생성한 명령을 선택합니다.
  2. 명령 세부 정보 페이지의 명령 내역 탭에 생성한 실행이 표시됩니다. 정보를 검색할 실행을 선택합니다.

3. 디바이스가 UpdateCommandExecution API를 사용하여 결과 정보를 제공한 경우, 이 페이지의 결과 탭에서 이 정보를 찾을 수 있습니다.

- Thing Hub 페이지에서

  명령을 실행할 때 AWS IoT 사물을 대상 디바이스로 선택한 경우 사물 허브 페이지에서 실행 세부 정보를 볼 수 있습니다.

  1. AWS IoT 콘솔의 Thing Hub 페이지로 이동하여 명령 실행을 생성한 사물을 선택합니다.
  2. 사물 세부 정보 페이지의 명령 내역에 생성한 실행이 표시됩니다. 정보를 검색할 실행을 선택합니다.
  3. 디바이스가 UpdateCommandExecution API를 사용하여 결과 정보를 제공한 경우, 이 페이지의 결과 탭에서 이 정보를 찾을 수 있습니다.

명령 실행 검색(CLI)

GetCommandExecution AWS IoT Core 컨트롤 플레인 HTTP API 작업을 사용하여 명령 실행에 대한 정보를 검색합니다. StartCommandExecution API 작업을 사용하여 이 명령을 이미 실행했어야 합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 GetCommandExecution 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *region* 대신 AWS 리전(예: us-east-1)로 대체합니다.
- *account-id* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.
- *command-id*와 같은 고유한 AWS IoT 명령 식별자를 사용합니다*LockDoor*.
- *devices* 디바이스가 AWS IoT 사물로 등록되었는지 또는 MQTT 클라이언트로 지정되었는지에 client 따라 thing 또는가 있는 .
- *device-id* thing-name 또는를 사용합니다 AWS IoT client-id.

```
{
  "Effect": "Allow",
```

```
  "Action": [
      "iot:GetCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

명령 실행 검색 예제

다음 예제에서는 명령을 사용하여 실행된 `start-command-execution` AWS CLI 명령에 대한 정보를 검색하는 방법을 보여줍니다. 다음 예제에서는 조향 모드를 끄기 위해 실행된 명령에 대한 정보를 검색하는 방법을 보여줍니다.

대체 예시:

- *<execution-id>* 대신 정보를 검색하려는 명령 실행의 식별자를 사용합니다.
- *<target-arn>* 대신 실행을 대상으로 하는 디바이스의 Amazon 리소스 번호(ARN)를 사용합니다. `start-command-execution` CLI 명령의 응답에서 이 정보를 얻을 수 있습니다.
- 선택적으로, 디바이스가 `UpdateCommandExection` API를 사용하여 실행 결과를 제공한 경우, `GetCommandExecution` API를 사용하여 `GetCommandExecution` API의 응답에 명령 실행 결과를 포함할지 여부를 지정할 수 있습니다.

```
aws iot get-command-execution
    --execution-id <execution-id> \
    --target-arn <target-arn> \
    --include-result
```

이 명령을 실행하면, 명령 실행의 ARN, 실행 상태, 실행이 시작된 시간 및 완료된 시간에 대한 정보를 포함한 응답이 생성됩니다. 또한, 상태에 대한 추가 정보가 포함된 `statusReason` 객체를 제공합니다. 다양한 상태 및 상태 이유에 대한 자세한 내용은 명령 실행 상태 섹션을 참조하세요.

다음 코드는 API 요청의 샘플 응답을 보여줍니다.

> **Note**
>
> 실행 응답의 `completedAt` 필드는 디바이스가 클라우드에 터미널 상태를 보고하는 시간에 해당합니다. `TIMED_OUT` 상태인 경우, 디바이스가 제한 시간 초과를 보고할 때만 이 필드가 설정

됩니다. 클라우드에서 TIMED_OUT 상태를 설정하면 TIMED_OUT 상태가 업데이트되지 않습니다. 이 제한 시간 초과 동작에 대한 자세한 내용은 명령 실행의 제한 시간 고려 사항 섹션을 참조하세요.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor",
    "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/myRegisteredThing",
    "status": "SUCCEEDED",
    "statusReason": {
        "reasonCode": "DEVICE_SUCCESSFULLY_EXECUTED",
        "reasonDescription": "SUCCESS"
    },
    "result": {
        "sn": { "s": "ABC-001" },
        "digital": { "b": true }
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "completedAt": "2024-03-23T00:50:10.095000-07:00"
}
```

MQTT 테스트 클라이언트를 사용하여 명령 업데이트 보기

MQTT 테스트 클라이언트를 사용하여, 명령 기능을 사용할 때 MQTT를 통한 메시지 교환을 볼 수 있습니다. 디바이스가와 MQTT 연결을 설정한 후 명령을 생성하고 페이로드를 지정한 다음 디바이스에서 실행할 AWS IoT수 있습니다. 명령을 실행할 때 디바이스가 명령에 대한 MQTT 예약 요청 주제를 구독한 경우이 주제에 게시된 페이로드 메시지가 표시됩니다.

그러면 디바이스가 페이로드 지침을 수신하고 AWS IoT 디바이스에서 지정된 작업을 수행합니다. 그런 다음 UpdateCommandExecution API를 사용하여 명령 실행 결과 및 상태 정보를 command. AWS IoT Device Management listens의 MQTT 예약 응답 주제에 게시하고, 업데이트된 정보를 저장하고, 로그를 AWS CloudTrail 및 Amazon CloudWatch에 게시합니다. 그런 다음 콘솔에서 또는 GetCommandExecution API를 사용하여 최신 명령 실행 정보를 검색할 수 있습니다.

다음 단계에서는 MQTT 테스트 클라이언트를 사용하여 메시지를 관찰하는 방법을 보여줍니다.

1. AWS IoT 콘솔에서 MQTT 테스트 클라이언트를 엽니다.
2. 구독 탭에서 다음 주제를 입력한 다음 구독을 선택합니다. 여기서 *<thingId>*는 등록한 디바이스의 사물 이름입니다 AWS IoT.

> **Note**
>
> AWS IoT 콘솔의 Thing Hub 페이지에서 디바이스의 사물 이름을 찾을 수 있습니다. 디바이스를 사물로 등록하지 않은 경우 Connect 디바이스 페이지에서에 연결할 때 디바이스 AWS IoT 를 등록할 수 있습니다.

```
$aws/commands/things/<thingId>/executions/+/request
```

3. (선택 사항) 구독 탭에서 다음 주제를 입력하고 구독을 선택할 수도 있습니다.

```
$aws/commands/things/+/executions/+/response/accepted/json
$aws/commands/things/+/executions/+/response/rejected/json
```

4. 명령 실행을 시작하면, 디바이스가 구독한 요청 주제인 `$aws/commands/things/<thingId>/executions/+/request`를 사용하여 메시지 페이로드가 해당 디바이스로 전송됩니다. MQTT 테스트 클라이언트에는 디바이스가 명령을 처리하는 지침이 포함된 명령 페이로드가 표시됩니다.
5. 디바이스가 명령 실행을 시작하면, 명령에 대한 다음 MQTT 예약 응답 주제에 상태 업데이트를 게시할 수 있습니다.

```
$aws/commands/<devices>/<device-id>/executions/<executionId>/response/json
```

예를 들어, 온도를 원하는 값으로 낮추기 위해 차량의 AC를 켜는 명령을 실행했다고 가정해 보겠습니다. 다음 JSON은 차량이 응답 주제에 게시한 샘플 메시지로 명령을 실행하지 못했음을 보여줍니다.

```
{
  "deviceId": "My_Car",
  "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
  "status": "FAILED",
  "statusReason": {
    "reasonCode": "CAR_LOW_ON_BATTERY",
    "reasonDescription": "Car battery is lower than 5 percent"
  }
}
```

이 경우 차량의 배터리를 충전한 다음, 명령을 다시 실행할 수 있습니다.

에서 명령 실행 나열 AWS 계정

명령을 실행한 후 AWS IoT 콘솔 및를 사용하여 명령 실행에 대한 정보를 검색할 수 있습니다 AWS CLI. 다음 정보를 획득할 수 있습니다.

- 명령 실행의 고유 식별자인 실행 ID
- 명령 실행의 상태. 대상 디바이스에서 명령을 실행하면 명령 실행은 CREATED 상태가 됩니다. 그런 다음 아래에 설명된 대로 다른 명령 실행 상태로 전환할 수 있습니다.
- 고유한 명령 ID 및 실행이 생성된 대상 디바이스
- 명령 실행이 생성된 시간을 보여주는 시작 날짜

계정의 명령 실행 나열(콘솔)

다음 방법 중 하나를 사용하여 콘솔에서 모든 명령 실행을 볼 수 있습니다.

- Command Hub 페이지에서

  AWS IoT 콘솔의 Command Hub 페이지로 이동하여 다음 단계를 수행합니다.

  1. 대상 디바이스에서 실행을 생성한 명령을 선택합니다.
  2. 명령 세부 정보 페이지에서 명령 내역 탭으로 이동하면 생성한 실행 목록이 표시됩니다.
- Thing Hub 페이지에서

  명령을 실행할 때 AWS IoT 사물을 대상 디바이스로 선택하고 단일 디바이스에 대해 여러 명령 실행을 생성한 경우 사물 허브 페이지에서 디바이스의 실행을 볼 수 있습니다.

  1. AWS IoT 콘솔의 Thing Hub 페이지로 이동하여 실행을 생성한 사물을 선택합니다.
  2. 사물 세부 정보 페이지의 명령 내역에 디바이스에 대해 생성한 실행 목록이 표시됩니다.

계정의 명령 실행 나열(CLI)

`ListCommandExecutions` AWS IoT Core 컨트롤 플레인 HTTP API 작업을 사용하여 계정의 모든 명령 실행을 나열합니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에서 이 작업을 수행할 수 있는 권한을 부여하는지 확인합니다. 다음은 사용자가 `ListCommandExecutions` 작업을 수행할 수 있도록 허용하는 IAM 정책 예시입니다.

대체 예시:

- *region* 대신 AWS 리전(예: us-east-1)로 대체합니다.
- *account-id* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.
- *command-id*와 같은 고유한 AWS IoT 명령 식별자를 사용합니다*LockDoor*.

```
{
  "Effect": "Allow",
  "Action": "iot:ListCommandExecutions",
  "Resource": *
}
```

명령 실행 나열 예제

다음 예제에서는 AWS 계정에서 명령 실행을 나열하는 방법을 보여줍니다.

명령을 실행할 때 목록을 필터링할 때, targetArn을 사용하여 특정 디바이스에 대해 생성된 명령 실행만 표시할지 또는 commandArn을 사용하여 지정된 특정 명령에 대한 실행만 표시할지 지정해야 합니다.

대체 예시:

- *<target-arn>* 대신 실행을 대상으로 하는 디바이스의 Amazon 리소스 번호(ARN)(예: arn:aws:iot:*us-east-1*:*123456789012*:thing/*b8e4157c98f332cffb37627f*)를 사용합니다.
- *<target-arn>* 대신 실행을 대상으로 하는 디바이스의 Amazon 리소스 번호(ARN)(예: arn:aws:iot:*us-east-1*:*123456789012*:thing/*b8e4157c98f332cffb37627f*)를 사용합니다.
- *<after>* 대신 생성된 실행을 나열하려는 시간(예: 2024-11-01T03:00)이 표시됩니다.

```
aws iot list-command-executions \
--target-arn <target-arn> \
--started-time-filter '{after=<after>}' \
--sort-order "ASCENDING"
```

이 명령을 실행하면 생성한 명령 실행 목록, 실행이 시작된 시간 및 완료된 시간이 포함된 응답이 생성됩니다. 또한, 상태 정보, 그리고 상태에 대한 추가 정보가 포함된 statusReason 객체도 제공합니다.

```
{
    "commandExecutions": [
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "b2b654ca-1a71-427f-9669-e74ae9d92d24",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "TIMED_OUT",
            "createdAt": "2024-11-24T14:39:25.791000-08:00",
            "startedAt": "2024-11-24T14:39:25.791000-08:00"
        },
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "34bf015f-ef0f-4453-acd0-9cca2d42a48f",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "IN_PROGRESS",
            "createdAt": "2024-11-24T14:05:36.021000-08:00",
            "startedAt": "2024-11-24T14:05:36.021000-08:00"
        }
    ]
}
```

다양한 상태 및 상태 이유에 대한 자세한 내용은 명령 실행 상태 섹션을 참조하세요.

명령 실행 삭제

명령 실행을 더 이상 사용하지 않을 경우, 계정에서 영구적으로 제거할 수 있습니다.

> **Note**
>
> - 명령 실행은 SUCCEEDED, FAILED 또는 REJECTED와 같은 터미널 상태가 된 경우에만 삭제할 수 있습니다.
> - 이 작업은 AWS IoT Core API 또는를 사용해야만 수행할 수 있습니다 AWS CLI. 콘솔에서는 이 작업을 사용할 수 없습니다.

샘플 IAM 정책

이 API 작업을 사용하기 전에 IAM 정책이 디바이스에 이러한 작업을 수행할 수 있는 권한을 부여해야 합니다. 다음은 디바이스에 작업을 수행할 권한을 부여하는 정책의 예입니다.

대체 예시:

- *Region*와 AWS 리전같은를 사용합니다us-east-1.
- *AccountID* 대신 AWS 계정 번호(예: *123456789012*)로 대체합니다.
- *CommandID* 대신 실행을 삭제하려는 명령의 식별자로 대체합니다.
- *devices* 디바이스가 AWS IoT 사물로 등록되었는지 또는 MQTT 클라이언트로 지정되었는지에 client 따라 thing 또는가 있는 .
- *device-id* thing-name 또는를 사용합니다 AWS IoT client-id.

```
{
  "Effect": "Allow",
  "Action": [
      "iot:DeleteCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

명령 실행 삭제 예제

다음 예제에서는 명령을 사용하여 delete-command AWS CLI 명령을 삭제하는 방법을 보여줍니다. 애플리케이션에 따라 *<execution-id>*를 삭제하려는 명령 실행의 식별자로 대체하고 *<target-arn>*를 대상 디바이스의 ARN으로 바꿉니다.

```
aws iot delete-command-execution \
--execution-id <execution-id> \
--target-arn <target-arn>
```

API 요청이 성공하면 명령 실행은 상태 코드 200을 생성합니다. GetCommandExecution API를 사용하여 명령 실행이 계정에 더 이상 존재하지 않는지 확인할 수 있습니다.

명령 리소스 사용 중지

더 이상 사용되지 않으므로 사용해서는 안 되는 명령을 사용 중단합니다. 예를 들어 ID는 같지만 페이로드는 다른 최신 명령을 생성할 때 또는 더 이상 활성 상태로 유지 관리되지 않는 명령을 사용 중지합니다.

주요 고려 사항

명령 사용 중단 시 중요한 고려 사항:

- 명령을 사용 중지해도 삭제되지는 않습니다. ID를 사용하여 명령을 검색하고 재사용을 위해 복원할 수 있습니다.
- 더 이상 사용되지 않는 명령에서 새 실행을 시작하려고 하면 오류가 발생하여 오래된 명령이 사용되지 않습니다.
- 더 이상 사용되지 않는 명령을 실행하려면 먼저 복원합니다. 복원 후 대상 디바이스에서 명령을 정기적으로 사용하고 실행할 수 있게 됩니다.
- 실행이 진행되는 동안 명령을 사용 중지하면 완료 시까지 계속 실행됩니다. 여전히 실행 상태를 검색할 수 있습니다.

명령 리소스 사용 중지(콘솔)

콘솔에서 명령을 사용 중지하려면 콘솔의 Command Hub로 이동하여 다음 단계를 AWS IoT 수행합니다.

1. 사용 중단하려는 명령을 선택한 다음 작업에서 사용 중단을 선택합니다.
2. 명령을 사용 중지할지 확인한 다음 사용 중지를 선택합니다.

명령 리소스 사용 중지(CLI)

update-command CLI를 사용하여 명령을 더 이상 사용되지 않는 것으로 표시합니다. 삭제하기 전에 명령을 사용 중지해야 합니다. 더 이상 사용되지 않는 명령을 사용하려면 먼저 복원합니다.

```
aws iot update-command \
    --command-id <command-id> \
    --deprecated
```

예를 들어, 위 예제에서 업데이트한 *ACSwitch* 명령을 더 이상 사용하지 않는 경우 다음 코드는 명령 실행 시 샘플 출력을 보여줍니다.

```
{
    "commandId": "turnOffAc",
    "deprecated": true,
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
```

```
}
```

사용 중지 시간 및 상태 확인

GetCommand API를 사용하여 명령이 더 이상 사용되지 않는지 여부와 마지막으로 사용 중단된 시기를 확인합니다.

```
aws iot get-command --command-id <turnOffAC>
```

이 명령은 마지막으로 업데이트된 필드의 생성 및 사용 중단 타임스탬프를 포함하여 명령 정보가 포함된 응답을 생성합니다. 이렇게 하면 명령 수명과 삭제 또는 재사용 여부를 결정하는 데 도움이 됩니다. 다음은 *turnOffAc* 명령에 대한 샘플 응답을 보여줍니다.

```
{
    "commandId": "turnOffAC",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/turnOffAC",
    "namespace": "AWS-IoT",
    "payload": {
        "content": "testPayload.json",
        "contentType": "application/json"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00",
    "deprecated": false
}
```

명령 리소스 복원

ACSwitch 명령을 사용하거나 디바이스로 보내려면 먼저 명령을 복원합니다.

콘솔에서 명령을 복원하려면 콘솔의 Command Hub로 이동하여 복원하려는 명령을 AWS IoT 선택한 다음 작업에서 복원을 선택합니다.

AWS IoT Core API 또는를 사용하여 명령을 복원하려면 UpdateCommand API 작업 또는 update-command CLI를 AWS CLI사용합니다. 다음 코드는 샘플 요청 및 응답을 보여줍니다.

```
aws iot update-command \
    --command-id <command-id>
    --no-deprecated
```

다음 코드는 샘플 출력을 보여줍니다.

```
{
    "commandId": "ACSwitch",
    "deprecated": false,
    "lastUpdatedAt": "2024-05-09T23:17:21.954000-07:00"
}
```

AWS IoT 보안 터널링

원격 사이트의 제한된 방화벽 뒤에 디바이스를 배포하는 경우 문제 해결, 구성 업데이트 및 기타 운영 작업을 위해 해당 디바이스에 액세스할 수 있는 방법이 필요합니다. 보안 터널링을 사용하여에서 관리하는 보안 연결을 통해 원격 디바이스와 양방향 통신을 설정합니다 AWS IoT. 보안 터널링은 기존 인바운드 방화벽 규칙을 업데이트할 필요가 없으므로 원격 사이트의 방화벽 규칙에서 제공하는 보안 수준을 동일하게 유지할 수 있습니다.

예를 들어, 몇 백 마일 떨어진 공장에 위치한 센서 디바이스에서 공장 온도를 측정하는 데 문제가 있는 경우 보안 터널링을 사용하여 이 센서 디바이스에 대한 세션을 열고 신속하게 시작할 수 있습니다. 문제를 확인한 후(예: 잘못된 구성 파일) 파일을 재설정하고 동일한 세션을 통해 센서 디바이스를 다시 시작할 수 있습니다. 보다 일반적인 문제 해결(예: 센서 디바이스 조사를 위해 공장에 기술자 파견)에 비해 보안 터널링은 사고 대응 및 복구 시간 및 운영 비용을 줄여줍니다.

보안 터널링이란 무엇입니까?

보안 터널링을 사용하면 원격 사이트의 포트로 제한된 방화벽 뒤에 배포된 디바이스에 액세스할 수 있습니다. AWS 클라우드를 사용하여 랩톱 또는 데스크톱 컴퓨터에서 이를 소스 디바이스로 사용사용하여 대상 디바이스에 연결할 수 있습니다. 소스와 대상은 각 디바이스에서 실행되는 오픈 소스 로컬 프록시를 사용하여 통신합니다. 로컬 프록시는 방화벽 AWS 클라우드 에서 허용하는 개방 포트, 일반적으로 443을 사용하여와 통신합니다. 터널을 통해 전송되는 데이터는 전송 계층 보안(TLS)을 사용하여 암호화됩니다.

주제

- 보안 터널링 개념
- 보안 터널링 작동 방식
- 보안 터널 수명 주기

보안 터널링 개념

다음 용어는 원격 디바이스와의 통신을 설정할 때 보안 터널링에서 사용됩니다. 보안 터널링 작동 방식에 대한 자세한 내용은 보안 터널링 작동 방식 섹션을 참조하세요.

클라이언트 액세스 토큰(CAT)

새 터널이 생성될 때 보안 터널링에 의해 생성된 토큰 페어입니다. CAT는 소스 및 대상 디바이스에서 보안 터널링 서비스에 연결하는 데 사용됩니다. CAT는 한 번만 사용하여 터널에 연결할 수 있

습니다. 터널에 다시 연결하려면 RotateTunnelAccessToken API 작업 또는 rotate-tunnel-access-token CLI 명령을 사용하여 클라이언트 액세스 토큰을 교체하세요.

클라이언트 토큰

클라이언트가 생성한 AWS IoT 보안 터널링 고유 값은 동일한 터널에 대한 모든 후속 재시도 연결에를 사용할 수 있습니다. 이 필드는 선택 사항입니다. 클라이언트 토큰이 제공되지 않으면 클라이언트 액세스 토큰(CAT)은 동일한 터널에 대해 한 번만 사용할 수 있습니다. 동일한 CAT를 사용한 후속 연결 시도는 거부됩니다. 클라이언트 토큰 사용에 대한 자세한 내용은 GitHub의 로컬 프록시 참조 구현을 참조하세요.

대상 애플리케이션

대상 디바이스에서 실행되는 애플리케이션입니다. 예를 들어, 대상 애플리케이션은 보안 터널링을 사용하여 SSH 세션을 설정하기 위한 SSH 데몬일 수 있습니다.

대상 디바이스

액세스하려는 원격 디바이스입니다.

디바이스 에이전트

AWS IoT 디바이스 게이트웨이에 연결하고 MQTT를 통해 새 터널 알림을 수신하는 IoT 애플리케이션입니다. 자세한 내용은 IoT 에이전트 코드 조각 단원을 참조하십시오.

로컬 프록시

소스 및 대상 디바이스에서 실행되고 보안 터널링과 디바이스 애플리케이션 간에 데이터 스트림을 릴레이하는 소프트웨어 프록시입니다. 로컬 프록시는 소스 모드 또는 대상 모드에서 실행할 수 있습니다. 자세한 내용은 로컬 프록시 단원을 참조하세요.

소스 디바이스

운영자가 대상 디바이스(일반적으로 랩톱 또는 데스크톱 컴퓨터)에 대한 세션을 시작하는 데 사용하는 디바이스입니다.

터널

소스 디바이스와 대상 디바이스 간의 양방향 통신을 가능하게 AWS IoT 하는를 통한 논리적 경로입니다.

보안 터널링 작동 방식

다음은 보안 터널링이 소스 디바이스와 대상 디바이스 간의 연결을 설정하는 방법을 보여줍니다. 클라이언트 액세스 토큰(CAT)과 같은 다양한 용어에 대한 자세한 내용은 보안 터널링 개념 섹션을 참조하세요.

1. 터널 열기

   원격 대상 디바이스로 세션을 시작하기 위한 터널을 열려면 AWS Management Console, AWS CLI open-tunnel 명령 또는 OpenTunnel API를 사용할 수 있습니다.

2. 클라이언트 액세스 토큰 페어 다운로드

   터널을 연 후 소스 및 대상에 대한 클라이언트 액세스 토큰(CAT)을 다운로드하여 소스 디바이스에 저장할 수 있습니다. 로컬 프록시를 시작하기 전에 CAT를 검색하고 저장해야 합니다.

3. 대상 모드에서 로컬 프록시 시작

   대상 디바이스에 설치되어 실행되는 IoT 에이전트는 예약된 MQTT 주제 `$aws/things/thing-name/tunnels/notify`를 구독하고 CAT를 받게 됩니다. 여기서 *thing-name*은 대상에 대해 생성한 AWS IoT 사물의 이름입니다. 자세한 내용은 보안 터널링 주제 단원을 참조하십시오.

   이제 IoT 에이전트는 이 CAT를 사용하여 대상 모드에서 로컬 프록시를 시작하고 터널의 대상 측에 연결을 설정합니다. 자세한 내용은 IoT 에이전트 코드 조각 단원을 참조하십시오.

4. 소스 모드에서 로컬 프록시 시작

   터널이 열리면는 소스 디바이스에서 다운로드할 수 있는 소스의 CAT를 AWS IoT Device Management 제공합니다. 이 CAT를 사용하여 소스 모드에서 로컬 프록시를 시작하면 로컬 프록시가 터널의 소스 측에 연결됩니다. 로컬 프로브에 대한 자세한 내용은 로컬 프록시 섹션을 참조하세요.

5. SSH 세션 열기

   터널의 양쪽이 연결되면 소스 측에서 로컬 프록시를 사용하여 SSH 세션을 시작할 수 있습니다.

를 사용하여 터널 AWS Management Console 을 열고 SSH 세션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요터널을 열고 원격 디바이스에 대한 SSH 세션 시작.

다음 동영상에서는 보안 터널링이 작동하는 방식을 설명하며 Raspberry Pi 디바이스에 대한 SSH 세션을 설정하는 프로세스를 안내합니다.

보안 터널 수명 주기

터널은 OPEN 또는 CLOSED 상태일 수 있습니다. 터널에 대한 연결은 CONNECTED 또는 DISCONNECTED 상태일 수 있습니다. 다음은 터널 및 연결의 다양한 상태가 작동하는 방식을 보여줍니다.

1. 터널을 열면 해당 상태가 OPEN입니다. 터널의 소스 및 대상 연결 상태가 DISCONNECTED로 설정됩니다.
2. 디바이스(소스 또는 대상)가 터널에 연결되면 해당 연결 상태가 CONNECTED로 변경됩니다.
3. 디바이스가 터널에서 연결이 끊어졌지만 터널 상태는 OPEN으로 유지되는 경우 해당 연결 상태가 다시 DISCONNECTED로 변경됩니다. 터널이 OPEN으로 남아 있는 동안 디바이스는 반복적으로 터널에 연결되고 터널에서 연결이 끊어질 수 있습니다.

   > **Note**
   >
   > 클라이언트 액세스 토큰(CAT)은 한 번만 사용하여 터널에 연결할 수 있습니다. 터널에 다시 연결하려면 RotateTunnelAccessToken API 작업 또는 rotate-tunnel-access-token CLI 명령을 사용하여 클라이언트 액세스 토큰을 교체하세요. 예시는 클라이언트 액세스 토큰을 교체하여 AWS IoT 보안 터널링 연결 문제 해결 섹션을 참조하세요.

4. CloseTunnel을 호출하거나 터널이 MaxLifetimeTimeout 값보다 오래 OPEN 상태를 유지하는 경우 터널의 상태가 CLOSED가 됩니다. OpenTunnel을 호출할 때 MaxLifetimeTimeout을 구성할 수 있습니다. 값을 지정하지 않으면 MaxLifetimeTimeout 기본값은 12시간입니다.

   > **Note**
   >
   > 터널이 CLOSED이면 다시 열 수 없습니다.

5. 터널이 표시되는 동안 DescribeTunnel 및 ListTunnels를 호출하여 터널 메타데이터를 볼 수 있습니다. 터널은 삭제되기 전에 최소 3시간 동안 AWS IoT 콘솔에서 볼 수 있습니다.

AWS IoT 보안 터널링 자습서

AWS IoT 보안 터널링을 사용하면 고객이에서 관리하는 보안 연결을 통해 방화벽 뒤에 있는 원격 디바이스와 양방향 통신을 설정할 수 있습니다 AWS IoT.

AWS IoT 보안 터널링을 데모하려면 AWS IoT GitHub에서 보안 터널링 데모를 사용하세요.

다음 자습서에서는 보안 터널링을 시작하고 사용하는 방법을 학습하는 데 도움이 되는 내용을 다룹니다. 다음 작업을 수행하는 방법에 대해 알아봅니다.

1. 원격 디바이스에 액세스하기 위한 빠른 설정 및 수동 설정 방법을 사용하여 보안 터널을 생성합니다.
2. 로컬 프록시를 구성하고(수동 설정 방법을 사용하는 경우) 터널에 연결하여 대상 디바이스에 액세스합니다.
3. 로컬 프록시를 구성할 필요 없이 브라우저에서 원격 디바이스에 SSH로 연결합니다.
4. 빠른 설정 방법을 사용하도록 AWS CLI 또는 수동 설정 방법을 사용하여 생성된 터널을 변환합니다.

이 섹션의 자습서

이 섹션의 자습서에서는 AWS Management Console 및 AWS IoT API 참조를 사용하여 터널을 생성하는 방법을 중점적으로 다룹니다. AWS IoT 콘솔에서 터널 허브 페이지 또는 생성한 사물의 세부 정보 페이지에서 터널을 생성할 수 있습니다. 자세한 내용은 AWS IoT 콘솔의 터널 생성 방법 단원을 참조하십시오.

이 섹션의 자습서는 다음과 같습니다.

- 터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스

  이 자습서에서는 빠른 설정 방법을 사용하여 터널 허브 페이지에서 터널을 여는 방법을 보여줍니다. 또한 브라우저 기반 SSH를 사용하여 AWS IoT 콘솔 내의 컨텍스트 내 명령줄 인터페이스를 사용하여 원격 디바이스에 액세스하는 방법을 알아봅니다.

- 수동 설정을 사용하여 터널을 열고 원격 디바이스에 연결

  이 자습서에서는 수동 설정 방법을 사용하여 터널 허브 페이지에서 터널을 여는 방법을 보여줍니다. 또한 소스 디바이스의 터미널에서 로컬 프록시를 구성 및 시작하고 터널에 연결하는 방법도 알아봅니다.

- 원격 디바이스에 대한 터널을 열고 브라우저 기반 SSH 사용

  이 자습서에서는 생성한 사물의 세부 정보 페이지에서 터널을 여는 방법을 보여줍니다. 새 터널을 생성하고 기존 터널을 사용하는 방법을 알아봅니다. 기존 터널은 디바이스에 대해 생성된 최근의 열린 터널에 해당합니다. 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스할 수도 있습니다.

AWS IoT 보안 터널링 자습서

- 터널을 열고 원격 디바이스에 대한 SSH 세션 시작
- 원격 디바이스에 대한 터널을 열고 브라우저 기반 SSH 사용

터널을 열고 원격 디바이스에 대한 SSH 세션 시작

이 자습서에서는 방화벽 뒤에 있는 디바이스에 원격으로 액세스하는 방법을 알아봅니다. 방화벽이 모든 인바운드 트래픽을 차단하기 때문에 디바이스에 대한 직접 SSH 세션을 시작할 수 없습니다. 이 자습서에서는 터널을 연 다음 해당 터널을 사용하여 원격 디바이스에 대한 SSH 세션을 시작하는 방법을 보여줍니다.

자습서의 사전 조건

자습서를 실행하기 위한 사전 조건은 터널을 열고 원격 디바이스에 액세스하는 데 수동 설정 방법을 사용하는지 아니면 빠른 설정 방법을 사용하는지에 따라 달라질 수 있습니다.

> **Note**
>
> 두 설정 방법을 사용할 때 모두 포트 443에서 아웃바운드 트래픽을 허용해야 합니다.

- 빠른 설정 방법 자습서의 사전 조건에 대한 자세한 내용은 빠른 설정 방법의 사전 조건 섹션을 참조하세요.
- 수동 설정 방법 자습서의 사전 조건에 대한 자세한 내용은 수동 설정 방법의 사전 조건 섹션을 참조하세요. 이 설정 방법을 사용하는 경우 소스 디바이스에서 로컬 프록시를 구성해야 합니다. 로컬 프록시 소스 코드를 다운로드하려면 GitHub의 로컬 프록시 참조 구현을 참조하세요.

터널 설정 방법

이 자습서에서는 터널을 열고 원격 디바이스에 연결하는 수동 및 빠른 설정 방법을 알아봅니다. 다음 표에 설정 방법 간의 차이점이 나와 있습니다. 터널을 생성한 후 브라우저 내 명령줄 인터페이스를 사용하여 원격 디바이스에 SSH로 연결할 수 있습니다. 토큰을 분실하거나 터널의 연결이 끊어지는 경우 새 액세스 토큰을 전송하여 터널에 다시 연결할 수 있습니다.

빠른 설정 및 수동 설정 방법

기준	빠른 설정	수동 설정
터널 생성	편집 가능한 기본 구성으로 새 터널을 생성합니다. 원격 디바이스에 액세스하는 데 SSH만 대상 서비스로 사용할 수 있습니다.	터널 구성을 수동으로 지정하여 터널을 생성합니다. 이 방법을 이용하면 SSH 이외의 서비스를 사용하여 원격 디바이스에 연결할 수 있습니다.
액세스 토큰	터널을 생성할 때 사물 이름을 지정한 경우 예약된 MQTT 주제에 대해 대상 액세스 토큰이 디바이스에 자동으로 전달됩니다. 소스 디바이스에서 토큰을 다운로드하거나 관리할 필요가 없습니다.	소스 디바이스에서 토큰을 수동으로 다운로드하고 관리해야 합니다. 터널을 생성할 때 사물 이름을 지정한 경우 예약된 MQTT 주제에 대해 대상 액세스 토큰이 원격 디바이스에 자동으로 전달됩니다.
로컬 프록시	디바이스와 상호 작용할 수 있도록 웹 기반 로컬 프록시가 자동으로 구성됩니다. 로컬 프록시를 수동으로 구성할 필요가 없습니다.	로컬 프록시를 수동으로 구성하고 시작해야 합니다. 로컬 프록시를 구성하려면 AWS IoT Device Client를 사용하거나 GitHub에서 로컬 프록시 참조 구현을 다운로드할 수 있습니다.

AWS IoT 콘솔의 터널 생성 방법

이 섹션의 자습서에서는 AWS Management Console 및 OpenTunnel API를 사용하여 터널을 생성하는 방법을 보여줍니다. 터널을 생성할 때 대상을 구성하는 경우 AWS IoT 보안 터널링은 MQTT 및 예약된 MQTT 주제인)를 통해 대상 클라이언트 액세스 토큰을 원격 디바이스에 전달합니다`$aws/things/RemoteDeviceA/tunnels/notify`. MQTT 메시지를 받으면 원격 디바이스의 IoT 에이전트가 로컬 프록시를 대상 모드로 시작합니다. 자세한 내용은 예약된 주제 단원을 참조하십시오.

> **Note**
>
> 다른 방법을 통해 대상 클라이언트 액세스 토큰을 원격 디바이스에 전달하려는 경우 대상 구성을 생략할 수 있습니다. 자세한 내용은 원격 디바이스 구성 및 IoT 에이전트 사용 단원을 참조하십시오.

AWS IoT 콘솔에서 다음 방법 중 하나를 사용하여 터널을 생성할 수 있습니다. 이러한 방법을 사용하여 터널을 생성하는 데 도움이 되는 자습서에 대한 자세한 내용은 이 섹션의 자습서 섹션을 참조하세요.

- 터널 허브

  터널을 생성할 때는 빠른 설정 방법을 사용할지 아니면 수동 설정 방법을 사용할지를 지정하고 선택적 터널 구성 세부 정보를 제공할 수 있습니다. 구성 세부 정보에는 대상 디바이스의 이름과 디바이스에 연결하는 데 사용할 서비스도 포함됩니다. 터널을 생성한 후 브라우저 내에서 SSH를 사용하거나 AWS IoT 콘솔 외부에서 터미널을 열어 원격 디바이스에 액세스할 수 있습니다.

- Thing details(사물 세부 정보) 페이지

  터널을 생성할 때는 설정 방법을 선택하고 선택적 터널 구성 세부 정보를 제공하는 것 외에도 최근의 열린 터널을 사용할지 아니면 디바이스에 대한 새 터널을 만들지를 지정할 수 있습니다. 기존 터널의 구성 세부 정보는 편집할 수 없습니다. 빠른 설정 방법을 사용하여 액세스 토큰을 교체하고 브라우저 내에서 원격 디바이스에 SSH로 연결할 수 있습니다. 이 방법을 사용하여 터널을 열려면 AWS IoT 레지스트리에 IoT 사물(예: `RemoteDeviceA`)을 생성해야 합니다. 자세한 내용은 AWS IoT 레지스트리에 디바이스 등록을 참조하세요.

이 섹션의 자습서

- 터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스
- 수동 설정을 사용하여 터널을 열고 원격 디바이스에 연결

터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스

빠른 설정 또는 수동 설정 방법을 사용하여 터널을 생성할 수 있습니다. 이 자습서에서는 빠른 설정 방법을 사용하여 터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 연결하는 방법을 보여줍니다. 수동 설정 방법을 사용하여 터널을 여는 방법의 예는 수동 설정을 사용하여 터널을 열고 원격 디바이스에 연결 섹션을 참조하세요.

빠른 설정 방법을 사용하면 편집 가능한 기본 구성으로 새 터널을 생성할 수 있습니다. 웹 기반 로컬 프록시가 구성되며, 액세스 토큰이 MQTT를 사용하여 원격 대상 디바이스에 자동으로 전달됩니다. 터널을 생성한 후 콘솔 내에서 명령줄 인터페이스를 사용하여 원격 디바이스와 상호 작용을 시작할 수 있습니다.

빠른 설정 방법의 경우 SSH를 대상 서비스로 사용하여 원격 디바이스에 액세스해야 합니다. 다양한 설정 방법에 대한 자세한 내용은 터널 설정 방법 섹션을 참조하세요.

빠른 설정 방법의 사전 조건

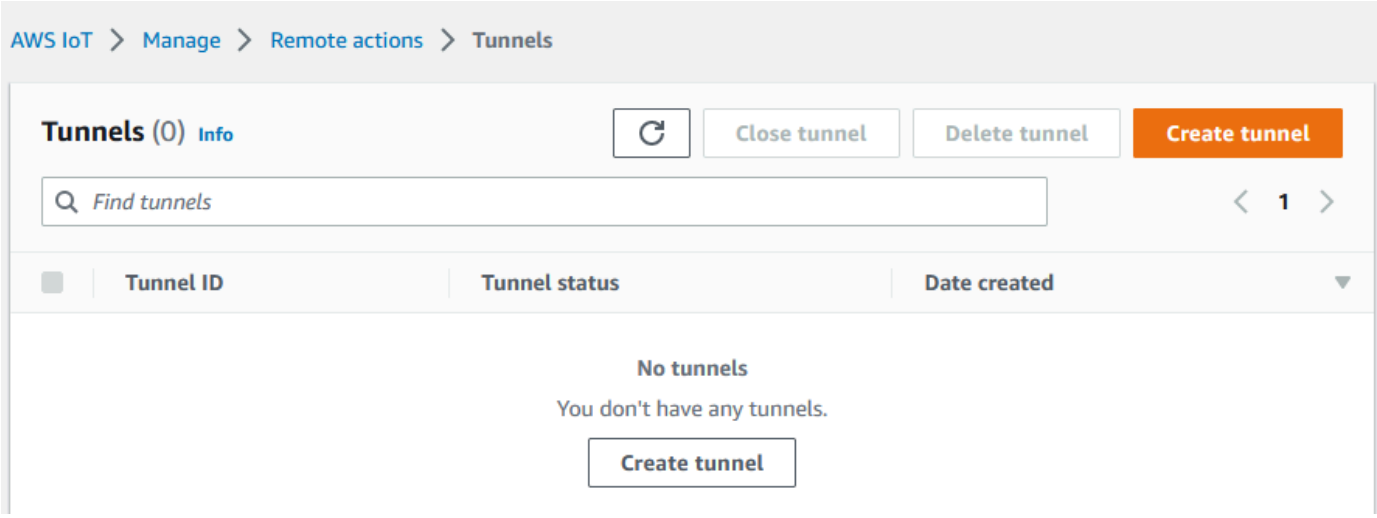
- 원격 디바이스 뒤에 있는 방화벽에서 포트 443의 아웃바운드 트래픽을 허용해야 합니다. 생성하는 터널은 이 포트를 사용하여 원격 디바이스에 연결됩니다.
- 디바이스 게이트웨이에 연결되고 MQTT 주제 구독으로 구성된 원격 디바이스에서 실행되는 IoT AWS IoT 디바이스 에이전트(참조IoT 에이전트 코드 조각)가 있습니다. 자세한 내용은 디바이스를 AWS IoT 디바이스 게이트웨이에 연결을 참조하세요.
- 원격 디바이스에서 실행 중인 SSH 데몬이 있어야 합니다.

터널 열기

AWS Management Console, AWS IoT API 참조 또는를 사용하여 보안 터널을 열 수 있습니다 AWS CLI. 선택적으로 대상 이름을 구성할 수 있지만 이 자습서에서는 필요하지 않습니다. 대상을 구성하면 보안 터널링이 MQTT를 사용하여 원격 디바이스에 액세스 토큰을 자동으로 전달합니다. 자세한 내용은 AWS IoT 콘솔의 터널 생성 방법 단원을 참조하십시오.

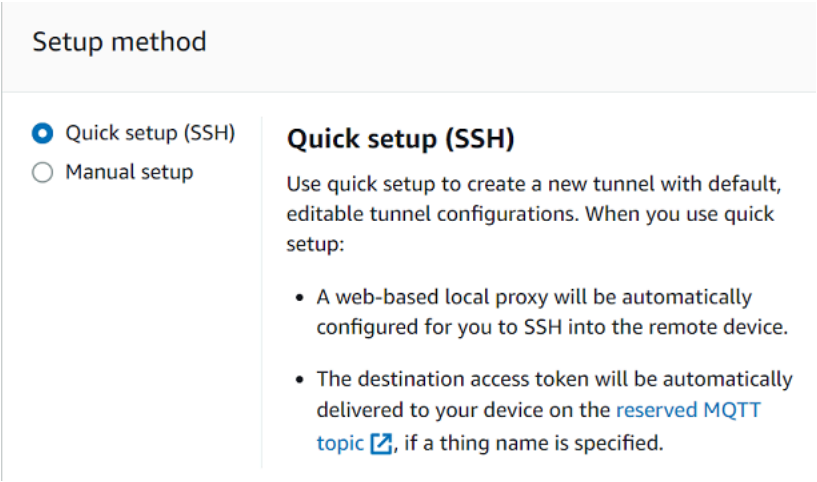
콘솔을 사용하여 터널을 열려면

1. AWS IoT 콘솔의 터널 허브로 이동하고 Create tunnel(터널 생성)을 선택합니다.



2. 이 자습서에서는 터널 생성 방법으로 Quick setup(빠른 설정)을 선택한 후 Next(다음)를 선택합니다.

> **Note**
>
> 생성한 사물의 세부 정보 페이지에서 보안 터널을 만드는 경우 새 터널을 생성할지 아니면 기존 터널을 사용할지를 선택할 수 있습니다. 자세한 내용은 원격 디바이스에 대한 터널을 열고 브라우저 기반 SSH 사용 단원을 참조하십시오.

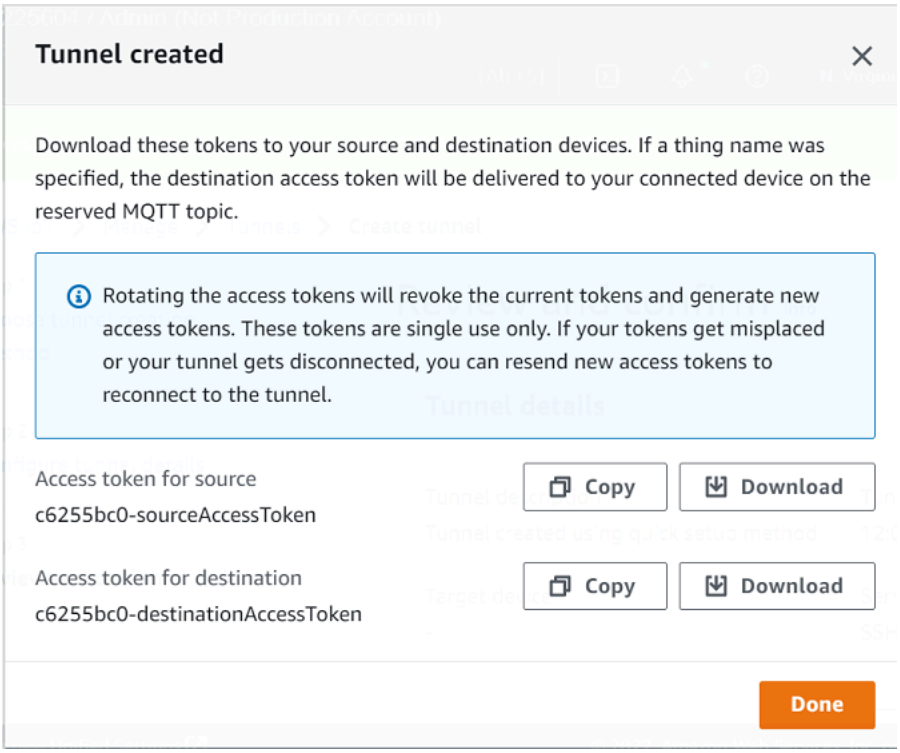


3. 터널 구성 세부 정보를 검토하고 확인합니다. 터널을 생성하려면 Confirm and create(확인 및 생성)를 선택합니다. 이러한 세부 정보를 편집하려면 Previous(이전)를 선택하여 이전 페이지로 돌아간 다음 터널을 확인하고 생성합니다.

> **Note**
>
> 빠른 설정을 사용하는 경우 서비스 이름을 편집할 수 없습니다. SSH를 Service(서비스)로 사용해야 합니다.

4. Done(완료)을 선택하여 터널을 생성합니다.

   이 자습서에서는 소스 또는 대상 액세스 토큰을 다운로드할 필요가 없습니다. 이러한 토큰은 한 번만 사용하여 터널에 연결할 수 있습니다. 터널의 연결이 끊어지면 새 토큰을 생성하고 원격 디바이스로 전송하여 터널에 다시 연결할 수 있습니다. 자세한 내용은 터널 액세스 토큰 재전송 단원을 참조하십시오.



API를 사용하여 터널을 열려면

새 터널을 열기 위해 OpenTunnel API 작업을 사용할 수 있습니다.

> **Note**
>
> AWS IoT 콘솔에서만 빠른 설정 방법을 사용하여 터널을 생성할 수 있습니다. AWS IoT API 참조 API 또는를 사용하는 경우 수동 설정 방법을 AWS CLI사용합니다. 기존에 생성한 터널을 열고 빠른 설정을 사용하도록 터널의 설정 방법을 변경할 수 있습니다. 자세한 내용은 기존 터널을 열고 브라우저 기반 SSH 사용 단원을 참조하십시오.

다음 예제에서는 이 API 작업을 실행하는 방법을 보여줍니다. 선택적으로 사물 이름과 대상 서비스를 지정하려면 `DestinationConfig` 파라미터를 사용합니다. 이 파라미터를 이용하는 방법의 예는 원격 디바이스에 대한 새 터널 열기 섹션을 참조하세요.

```
aws iotsecuretunneling open-tunnel
```

이 명령을 실행하면 새 터널이 생성되고 소스 및 대상 액세스 토큰이 제공됩니다.

```
{
```

```
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

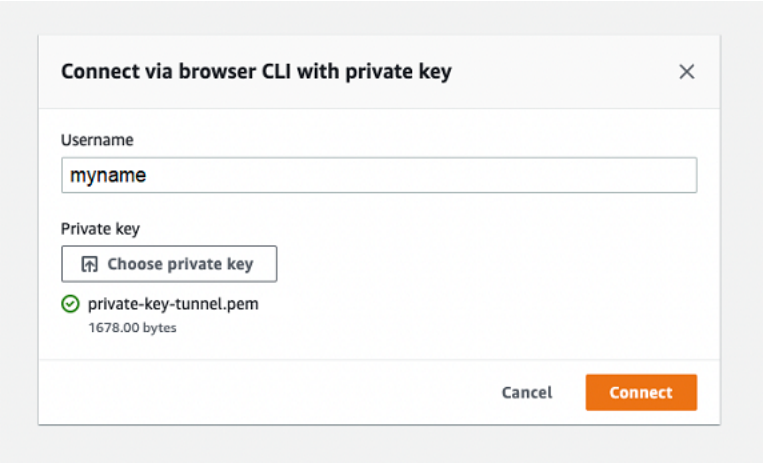
브라우저 기반 SSH 사용

빠른 설정 방법을 사용하여 터널을 생성하고 대상 디바이스가 터널에 연결되면 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스할 수 있습니다. 브라우저 기반 SSH를 사용하면 콘솔 내에서 상황별 명령줄 인터페이스에 명령을 입력하여 원격 디바이스와 직접 통신할 수 있습니다. 이 기능을 사용하는 경우 콘솔 외부에서 터미널을 열거나 로컬 프록시를 구성할 필요가 없기 때문에 원격 디바이스와 더욱 쉽게 상호 작용할 수 있습니다.

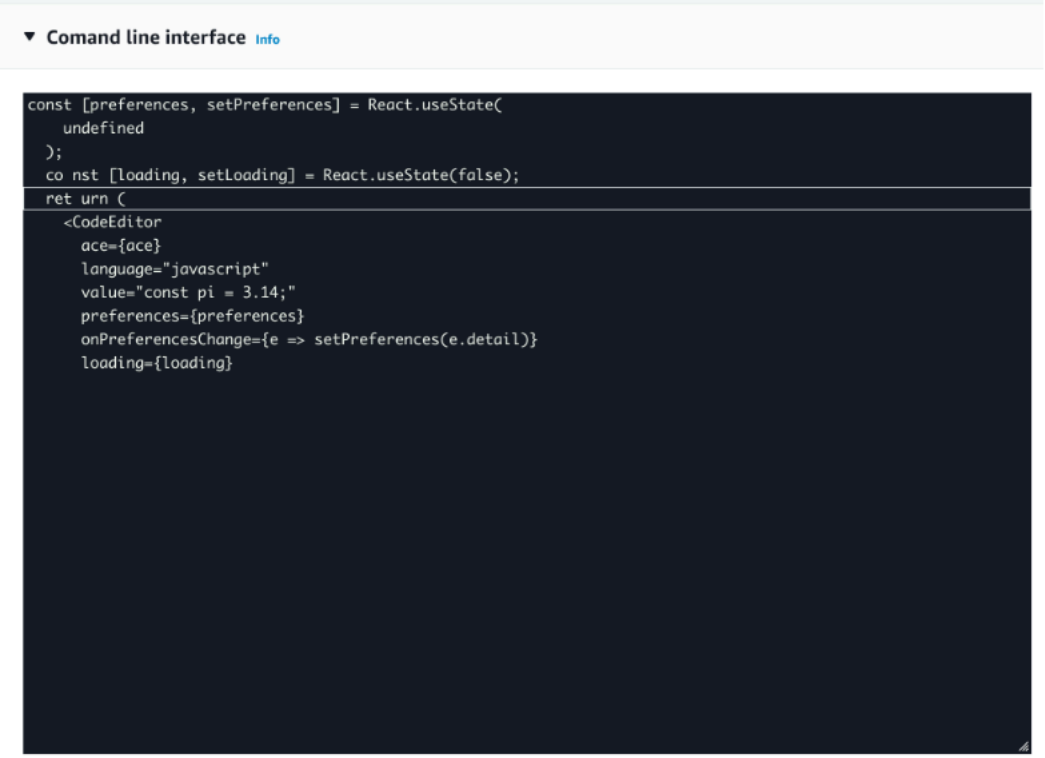
브라우저 기반 SSH를 사용하려면

1. AWS IoT 콘솔의 터널 허브로 이동하고 생성한 터널을 선택하여 세부 정보를 봅니다.
2. Secure Shell(SSH) 섹션을 확장한 다음 Connect(연결)를 선택합니다.
3. 사용자 이름과 암호를 제공하여 SSH 연결에 인증할지 여부를 선택하거나, 보다 안전한 인증을 위해 디바이스의 프라이빗 키를 사용할 수 있습니다. 프라이빗 키를 사용하여 인증하는 경우 PEM 형식(256 및 512) RSA 키만 AWS IoT Secure Tunneling SSH 콘솔에서 작동합니다.

   - 사용자 이름과 암호를 사용하여 연결하려면 Use password(암호 사용)를 선택합니다. 그런 다음 사용자 이름과 암호를 입력하고 브라우저 내 CLI를 사용할 수 있습니다.
   - 대상 디바이스의 프라이빗 키를 사용하여 연결하려면 Use private key(프라이빗 키 사용)를 선택합니다. 사용자 이름을 지정하고 디바이스의 프라이빗 키 파일을 업로드한 다음 Connect(연결)를 선택하여 브라우저 내 CLI 사용을 시작합니다.



SSH 연결에 인증한 후에는 로컬 프록시가 이미 구성되어 있으므로 신속하게 명령을 입력하고 브라우저 CLI를 사용하여 디바이스와 상호 작용할 수 있습니다.



터널 지속 시간 이후에도 브라우저 CLI가 계속 열려 있으면 시간 초과로 인해 명령줄 인터페이스의 연결이 끊어질 수 있습니다. 터널을 복제하고 다른 세션을 시작하여 콘솔 자체 내에서 원격 디바이스와 상호 작용할 수 있습니다.

브라우저 기반 SSH 사용 시 문제 해결

다음은 브라우저 기반 SSH를 사용할 때 발생할 수 있는 몇 가지 문제를 해결하는 방법을 보여줍니다.

- 명령줄 인터페이스 대신 오류가 표시됨

  대상 디바이스의 연결이 끊어졌기 때문에 오류가 표시될 수 있습니다. Generate new access tokens(새 액세스 토큰 생성)를 선택하여 새 액세스 토큰을 생성하고 MQTT를 사용하여 원격 디바이스로 토큰을 전송할 수 있습니다. 새 토큰을 사용하여 터널에 다시 연결할 수 있습니다. 터널에 다시 연결하면 기록이 지워지고 명령줄 세션이 새로 고쳐집니다.
- 프라이빗 키를 사용하여 인증할 때 터널 연결 해제 오류가 표시됨

  프라이빗 키가 대상 디바이스에서 수락되지 않았기 때문에 오류가 표시될 수 있습니다. 이 오류를 해결하려면 인증을 위해 업로드한 프라이빗 키 파일을 확인합니다. 여전히 오류가 표시되면 디바이스

로그를 확인합니다. 원격 디바이스로 새 액세스 토큰을 전송하여 터널에 다시 연결해 볼 수도 있습니다.

- 세션을 사용할 때 터널이 닫힘

  터널이 지정된 시간 이상 열려 있어서 터널이 닫혔다면 명령줄 세션의 연결이 끊어질 수 있습니다. 터널이 닫힌 후에는 터널을 다시 열 수 없습니다. 다시 연결하려면 디바이스에 대한 다른 터널을 열어야 합니다.

  터널을 복제하여 닫힌 터널과 동일한 구성으로 새 터널을 생성할 수 있습니다. AWS IoT 콘솔에서 닫힌 터널을 복제할 수 있습니다. 터널을 복제하려면 닫힌 터널을 선택하여 세부 정보를 확인한 다음 Duplicate tunnel(터널 복제)을 선택합니다. 사용할 터널 지속 시간을 지정한 다음 새 터널을 생성합니다.

정리

- 터널 닫기

  터널을 다 사용한 후에는 터널을 닫는 것이 좋습니다. 터널이 지정된 터널 지속 시간보다 더 오래 열려 있는 경우 터널이 닫힐 수도 있습니다. 터널이 닫힌 후에는 터널을 다시 열 수 없습니다. 닫힌 터널을 선택한 다음 Duplicate tunnel(터널 복제)을 선택하여 여전히 터널을 복제할 수 있습니다. 사용할 터널 지속 시간을 지정한 다음 새 터널을 생성합니다.

  - AWS IoT 콘솔에서 개별 터널 또는 여러 터널을 닫으려면 터널 허브로 이동하여 닫을 터널을 선택한 다음 Close tunnel(터널 닫기)을 선택합니다.
  - AWS IoT API 참조 API를 사용하여 개별 터널 또는 여러 터널을 닫으려면 CloseTunnel API를 사용합니다.

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- 터널 삭제

  에서 터널을 영구적으로 삭제할 수 있습니다 AWS 계정.

  > **Warning**
  > 삭제 작업은 영구적이며 취소할 수 없습니다.

- AWS IoT 콘솔에서 개별 터널 또는 여러 터널을 삭제하려면 터널 허브로 이동하여 삭제할 터널을 선택한 다음 Delete tunnel(터널 삭제)을 선택합니다.
- AWS IoT API 참조 API를 사용하여 개별 터널 또는 여러 터널을 삭제하려면 CloseTunnel API를 사용합니다. API를 사용하는 경우 `delete` 플래그를 `true`로 설정합니다.

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

수동 설정을 사용하여 터널을 열고 원격 디바이스에 연결

터널을 열 때 빠른 설정 또는 수동 설정 방법을 선택하여 원격 디바이스에 연결하도록 터널을 열 수 있습니다. 이 자습서에서는 수동 설정 방법을 사용하여 터널을 열고 로컬 프록시를 구성 및 시작하여 원격 디바이스에 연결하는 방법을 보여줍니다.

수동 설정 방법을 사용하는 경우 터널을 생성할 때 터널 구성을 수동으로 지정해야 합니다. 터널을 생성한 후 브라우저 내에서 SSH를 사용하거나 AWS IoT 콘솔 외부에서 터미널을 열 수 있습니다. 이 자습서에서는 콘솔 외부에서 터미널을 사용하여 원격 디바이스에 액세스하는 방법을 보여줍니다. 또한 로컬 프록시를 구성한 다음 로컬 프록시에 연결하여 원격 디바이스와 상호 작용하는 방법도 알아봅니다. 로컬 프록시에 연결하려면 터널을 생성할 때 소스 액세스 토큰을 다운로드해야 합니다.

이 방법을 이용하면 SSH 이외의 서비스(예: FTP)를 사용하여 원격 디바이스에 연결할 수 있습니다. 다양한 설정 방법에 대한 자세한 내용은 터널 설정 방법 섹션을 참조하세요.

수동 설정 방법의 사전 조건

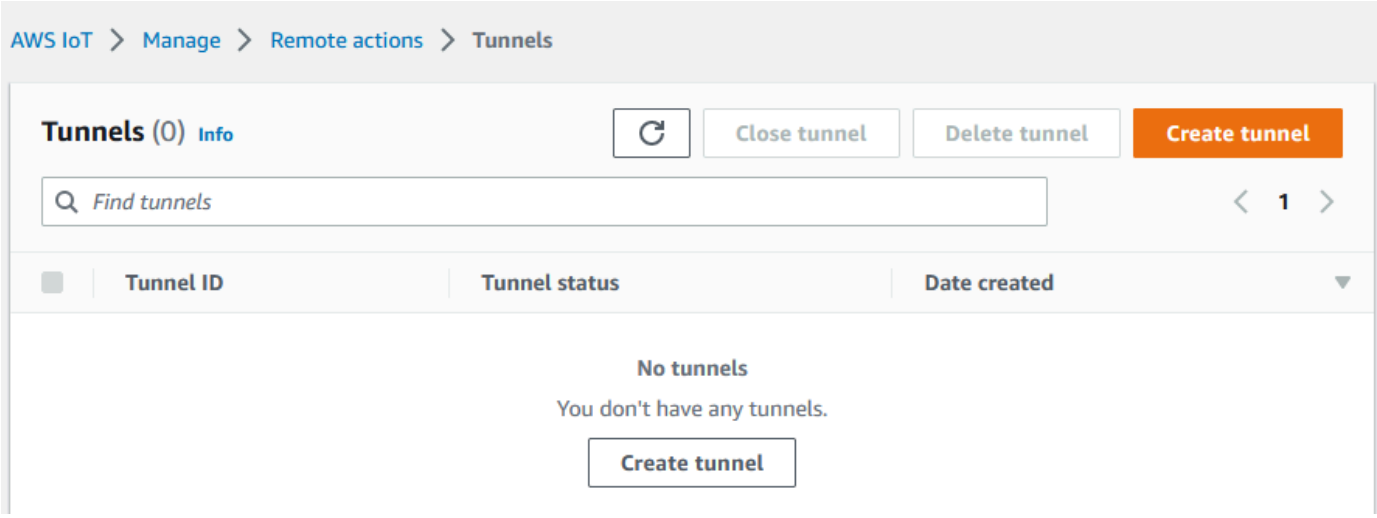
- 원격 디바이스 뒤에 있는 방화벽에서 포트 443의 아웃바운드 트래픽을 허용해야 합니다. 생성하는 터널은 이 포트를 사용하여 원격 디바이스에 연결됩니다.
- 디바이스 게이트웨이에 연결되고 MQTT 주제 구독으로 구성된 원격 디바이스에서 실행되는 IoT AWS IoT 디바이스 에이전트(참조IoT 에이전트 코드 조각)가 있습니다. 자세한 내용은 디바이스를 AWS IoT 디바이스 게이트웨이에 연결을 참조하세요.
- 원격 디바이스에서 실행 중인 SSH 데몬이 있어야 합니다.
- GitHub에서 로컬 프록시 소스 코드를 다운로드하고 선택한 플랫폼에 맞게 빌드했습니다. 이 자습서에서는 이 빌드된 로컬 프록시 실행 파일을 `localproxy`(이)라고 합니다.

터널 열기

AWS Management Console, AWS IoT API 참조 또는를 사용하여 보안 터널을 열 수 있습니다 AWS CLI. 선택적으로 대상 이름을 구성할 수 있지만 이 자습서에서는 필요하지 않습니다. 대상을 구성하면 보안 터널링이 MQTT를 사용하여 원격 디바이스에 액세스 토큰을 자동으로 전달합니다. 자세한 내용은 AWS IoT 콘솔의 터널 생성 방법 단원을 참조하십시오.

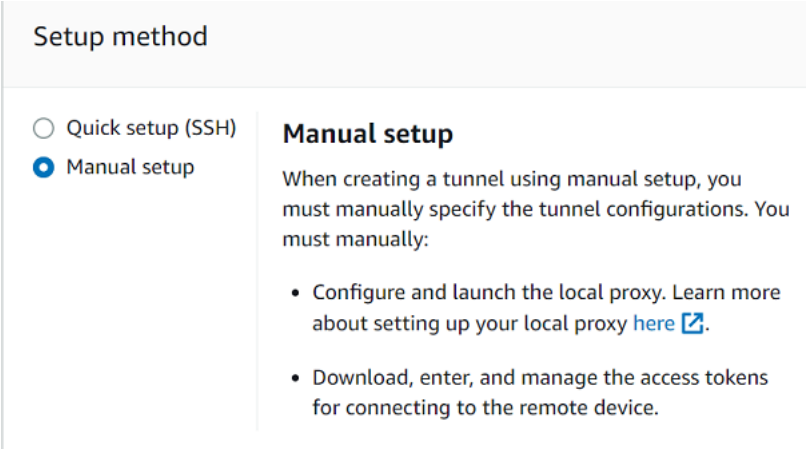
콘솔에서 터널을 열려면

1. AWS IoT 콘솔의 터널 허브로 이동하고 Create tunnel(터널 생성)을 선택합니다.



2. 이 자습서에서는 터널 생성 방법으로 Manual setup(수동 설정)을 선택한 후 Next(다음)를 선택합니다. Quick setup(빠른 설정) 방법을 사용한 터널 생성에 대한 자세한 내용은 터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스 섹션을 참조하세요.

> **Note**
>
> 사물의 세부 정보 페이지에서 보안 터널을 만드는 경우 새 터널을 생성할지 아니면 기존 터널을 사용할지를 선택할 수 있습니다. 자세한 내용은 원격 디바이스에 대한 터널을 열고 브라우저 기반 SSH 사용 단원을 참조하십시오.



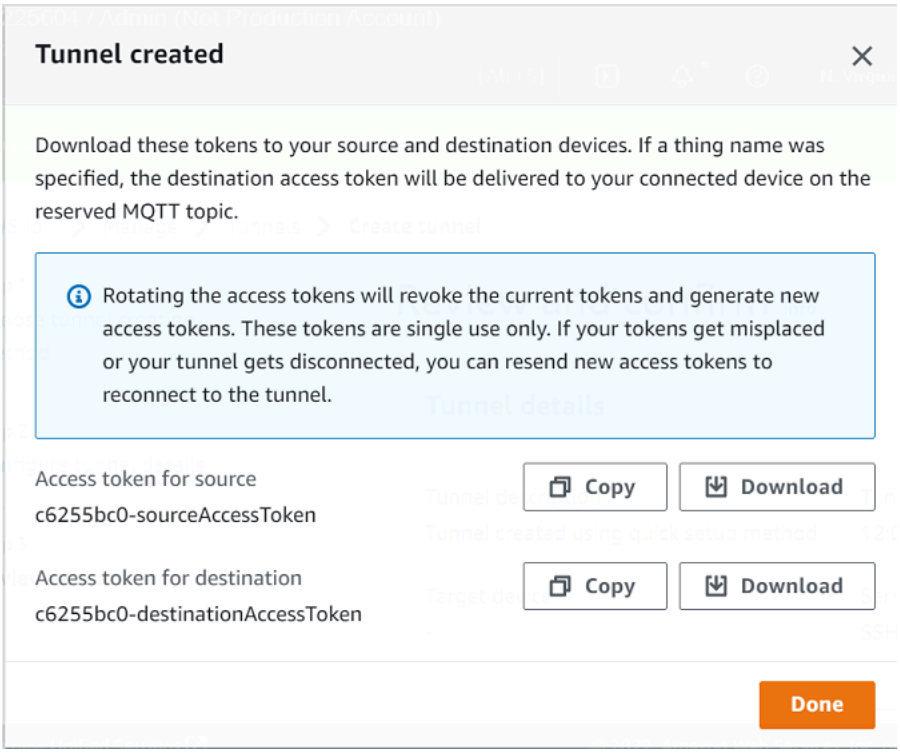
3. (선택 사항) 터널의 구성 설정을 입력합니다. 이 단계를 건너뛰고 다음 단계로 진행하여 터널을 생성할 수도 있습니다.

   리소스를 식별하는 데 도움이 되도록 터널 설명, 터널 제한 시간 및 리소스 태그를 키-값 페어로 입력합니다. 이 자습서에서는 대상 구성을 건너뛸 수 있습니다.

   > **Note**
   >
   > 터널을 열어 둔 기간을 기준으로 요금이 청구되지 않습니다. 새 터널을 생성할 때만 요금이 부과됩니다. 요금 정보는 AWS IoT Device Management 요금의 보안 터널링을 참조하세요.

4. 클라이언트 액세스 토큰을 다운로드한 다음, Done(완료)을 선택합니다. 완료(Done)를 선택한 후에는 토큰을 다운로드할 수 없습니다.

   이러한 토큰은 한 번만 사용하여 터널에 연결할 수 있습니다. 토큰을 분실하거나 터널의 연결이 끊어지면 새 토큰을 생성하고 원격 디바이스로 전송하여 터널에 다시 연결할 수 있습니다.



API를 사용하여 터널을 열려면

새 터널을 열기 위해 OpenTunnel API 작업을 사용할 수 있습니다. API를 사용하여 터널 지속 시간 및 대상 구성과 같은 추가 구성을 지정할 수도 있습니다.

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
```

이 명령을 실행하면 새 터널이 생성되고 소스 및 대상 액세스 토큰이 제공됩니다.

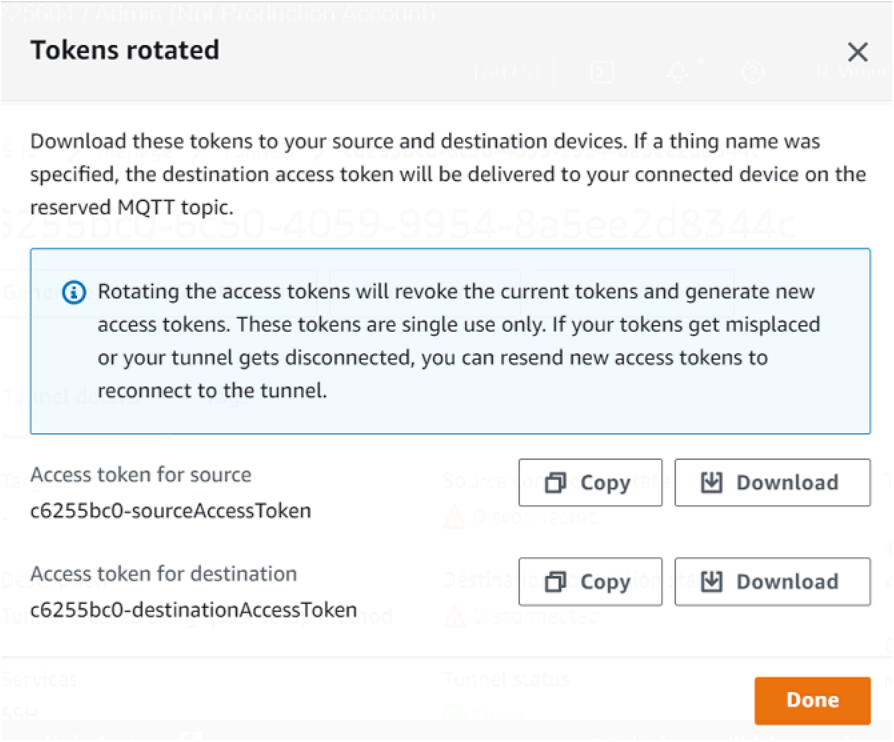
```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

터널 액세스 토큰 재전송

터널을 생성할 때 얻은 토큰은 터널에 연결하는 데 한 번만 사용할 수 있습니다. 액세스 토큰을 잘못 배치하거나 터널 연결이 끊어지면 추가 비용 없이 MQTT를 사용하여 원격 디바이스로 새 액세스 토큰을 재전송할 수 있습니다. AWS IoT 보안 터널링은 현재 토큰을 취소하고 터널에 다시 연결하기 위한 새 액세스 토큰을 반환합니다.

콘솔에서 토큰을 교체하려면

1. AWS IoT 콘솔의 터널 허브로 이동하여 생성한 터널을 선택합니다.
2. Tunnel details(터널 세부 정보) 페이지에서 Generate new access tokens(새 액세스 토큰 생성)를 선택한 후 Next(다음)를 선택합니다.
3. 터널에 사용할 새 액세스 토큰을 다운로드하고 Done(완료)을 선택합니다. 이러한 토큰은 한 번만 사용할 수 있습니다. 토큰을 분실하거나 터널의 연결이 끊어지는 경우 새 액세스 토큰을 다시 전송할 수 있습니다.



API를 사용하여 액세스 토큰을 교체하려면

터널 액세스 토큰을 교체하려는 경우 RotateTunnelAccessToken API 작업을 사용하여 현재 토큰을 취소하고 터널에 다시 연결하기 위한 새 액세스 토큰을 반환할 수 있습니다. 예를 들어, 다음 명령은 대상 디바이스 *RemoteThing1*의 액세스 토큰을 교체합니다.

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

이 명령을 실행하면 다음 예제와 같이 새 액세스 토큰이 생성됩니다. 그런 다음 디바이스 에이전트가 올바르게 설정된 경우 MQTT를 사용하여 토큰이 디바이스에 전달되어 터널에 연결됩니다.

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

액세스 토큰을 교체하는 방법과 시기를 보여주는 예제는 클라이언트 액세스 토큰을 교체하여 AWS IoT 보안 터널링 연결 문제 해결 섹션을 참조하세요.

로컬 프록시 구성 및 시작

원격 디바이스에 연결하려면 노트북에서 터미널을 열어 로컬 프록시를 구성하고 시작합니다. 로컬 프록시는 WebSocket 보안 연결을 통한 보안 터널링을 사용하여 소스 디바이스에서 실행되는 애플리케이션에서 보낸 데이터를 전송합니다. GitHub에서 로컬 프록시 소스를 다운로드할 수 있습니다.

로컬 프록시를 구성한 후 소스 클라이언트 액세스 토큰을 복사한 다음 해당 토큰을 사용하여 로컬 프록시를 소스 모드로 시작합니다. 다음은 로컬 프록시를 시작하는 예제 명령을 보여줍니다. 다음 명령에서 로컬 프록시는 포트 5555에서 새 연결을 수신하도록 구성됩니다. 이 명령에서:

- `-r`는 터널이 생성된 리전과 동일한 리전이어야 AWS 리전하는를 지정합니다.
- `-s`는 프록시가 연결될 포트를 지정합니다.
- `-t`는 클라이언트 토큰 텍스트를 지정합니다.

```
./localproxy -r us-east-1 -s 5555 -t source-client-access-token
```

이 명령을 실행하면 로컬 프록시가 소스 모드로 시작됩니다. 이 명령을 실행한 후 다음 오류가 발생하면 CA 경로를 설정합니다. 자세한 내용은 GitHub의 보안 터널링 로컬 프록시를 참조하세요.

```
Could not perform SSL handshake with proxy server: certificate verify failed
```

다음은 source 모드에서 로컬 프록시를 실행하는 샘플 출력을 보여줍니다.

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

SSH 세션 시작

다른 터미널을 열고 다음 명령을 사용하여 포트 5555의 로컬 프록시에 연결하여 새 SSH 세션을 시작합니다.

```
ssh username@localhost -p 5555
```

SSH 세션에 대한 암호를 묻는 메시지가 나타날 수 있습니다. SSH 세션을 완료했으면 **exit**을(를) 입력하여 세션을 닫습니다.

정리

- 터널 닫기

  터널을 다 사용한 후에는 터널을 닫는 것이 좋습니다. 터널이 지정된 터널 지속 시간보다 더 오래 열려 있는 경우 터널이 닫힐 수도 있습니다. 터널이 닫힌 후에는 터널을 다시 열 수 없습니다. 닫힌 터널을 연 다음 Duplicate tunnel(터널 복제)을 선택하여 여전히 터널을 복제할 수 있습니다. 사용할 터널 지속 시간을 지정한 다음 새 터널을 생성합니다.

  - AWS IoT 콘솔에서 개별 터널 또는 여러 터널을 닫으려면 Tunnels hub(터널 허브)로 이동하여 닫을 터널을 선택한 다음 Close tunnel(터널 닫기)을 선택합니다.
  - AWS IoT API 참조 API를 사용하여 개별 터널 또는 여러 터널을 닫으려면 CloseTunnel API 작업을 사용합니다.

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- 터널 삭제

  에서 터널을 영구적으로 삭제할 수 있습니다 AWS 계정.

  > **Warning**
  > 삭제 작업은 영구적이며 취소할 수 없습니다.

  - AWS IoT 콘솔에서 개별 터널 또는 여러 터널을 삭제하려면 터널 허브로 이동하여 삭제할 터널을 선택한 다음 Delete tunnel(터널 삭제)을 선택합니다.
  - AWS IoT API 참조 API를 사용하여 개별 터널 또는 여러 터널을 삭제하려면 CloseTunnel API 작업을 사용합니다. API를 사용하는 경우 `delete` 플래그를 `true`로 설정합니다.

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
        --delete true
    ```

원격 디바이스에 대한 터널을 열고 브라우저 기반 SSH 사용

AWS IoT 콘솔에서 터널 허브 또는 생성한 IoT 사물의 세부 정보 페이지에서 터널을 생성할 수 있습니다. Tunnels(터널) 허브에서 터널을 생성할 때 빠른 설정을 사용할지 아니면 수동 설정을 사용할지를 지정할 수 있습니다. 자습서 예시는 터널을 열고 원격 디바이스에 대한 SSH 세션 시작 단원을 참조하세요.

AWS IoT 콘솔의 사물 세부 정보 페이지에서 터널을 생성할 때이 자습서에 설명된 대로 새 터널을 생성할지 아니면 해당 사물에 대한 기존 터널을 열지 지정할 수도 있습니다. 기존 터널을 선택하면 이 디바이스에 대해 생성한 최근의 열린 터널에 액세스할 수 있습니다. 그런 다음 터미널 내의 명령줄 인터페이스를 사용하여 디바이스에 SSH로 연결할 수 있습니다.

사전 조건

- 원격 디바이스 뒤에 있는 방화벽에서 포트 443의 아웃바운드 트래픽을 허용해야 합니다. 생성하는 터널은 이 포트를 사용하여 원격 디바이스에 연결됩니다.
- AWS IoT 레지스트리에서 IoT 사물(예: `RemoteDevice1`)을 생성했습니다. 이 사물은 클라우드에 있는 원격 디바이스의 표현에 해당합니다. 자세한 내용은 AWS IoT 레지스트리에 디바이스 등록을 참조하세요.
- 디바이스 게이트웨이에 연결되고 MQTT 주제 구독으로 구성된 원격 디바이스에서 실행되는 IoT AWS IoT 디바이스 에이전트(참조IoT 에이전트 코드 조각)가 있습니다. 자세한 내용은 디바이스 게이트웨이에 AWS IoT 디바이스 연결을 참조하세요.
- 원격 디바이스에서 실행 중인 SSH 데몬이 있어야 합니다.

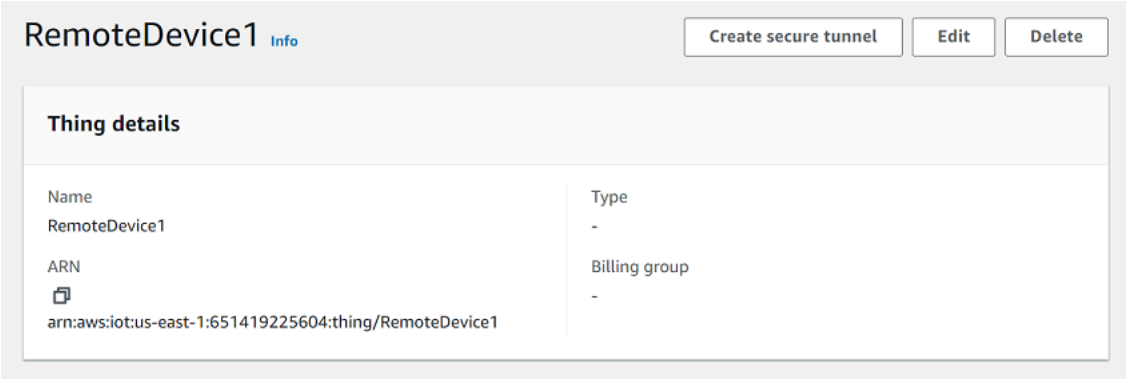
원격 디바이스에 대한 새 터널 열기

원격 디바이스 `RemoteDevice1`에 대한 터널을 연다고 가정합니다. 먼저 AWS IoT 레지스트리에 `RemoteDevice1`이라는 이름으로 IoT 사물을 생성합니다. 그런 다음 AWS Management Console, AWS IoT API 참조 API 또는를 사용하여 터널을 생성할 수 있습니다 AWS CLI.

보안 터널링 서비스는 터널을 생성할 때 대상을 구성함으로써 MQTT 및 예약된 MQTT 주제(`$aws/things/RemoteDeviceA/tunnels/notify`)를 통해 대상 클라이언트 액세스 토큰을 원격 디바이스에 전달합니다. 자세한 내용은 AWS IoT 콘솔의 터널 생성 방법 단원을 참조하십시오.

콘솔에서 원격 디바이스에 대한 터널을 생성하려면

1. `RemoteDevice1` 사물을 선택하고 세부 정보를 확인한 다음 Create secure tunnel(보안 터널 생성)을 선택합니다.



2. 새 터널을 생성할지 아니면 기존 터널을 열지를 선택합니다. 새 터널을 생성하려면 Create new tunnel(새 터널 생성)을 선택합니다. 그런 다음 터널을 생성하는 데 수동 설정 방법을 사용할지 아니면 빠른 설정 방법을 사용할지를 선택할 수 있습니다. 자세한 내용은 수동 설정을 사용하여 터널을 열고 원격 디바이스에 연결 및 터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스 섹션을 참조하세요.

API를 사용하여 원격 디바이스에 대한 터널을 생성하려면

새 터널을 열기 위해 OpenTunnel API 작업을 사용할 수 있습니다. 다음 코드에서는 이 명령을 실행하는 예제를 보여줍니다.

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
    --cli-input-json file://input.json
```

다음은 `input.json` 파일의 내용을 보여줍니다. `destinationConfig` 파라미터를 사용하여 대상 디바이스의 이름(예: *RemoteDevice1*)과 대상 디바이스에 액세스하는 데 사용할 서비스(예: *SSH*)를 지정할 수 있습니다. 선택적으로 터널 설명 및 태그와 같은 추가 파라미터를 지정할 수도 있습니다.

input.json 내용

```
{
   "description": "Tunnel to remote device1",
   "destinationConfig": {
      "services": [ "SSH" ],
      "thingName": "RemoteDevice1"
   }
}
```

이 명령을 실행하면 새 터널이 생성되고 소스 및 대상 액세스 토큰이 제공됩니다.

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

기존 터널을 열고 브라우저 기반 SSH 사용

수동 설정 방법 또는 AWS IoT API 참조 APIRemoteDevice1를 사용하여 원격 디바이스 용 터널을 생성했다고 가정해 보겠습니다. 그런 다음 디바이스에 대한 기존 터널을 열고 Quick setup(빠른 설정)을 선택하여 브라우저 기반 SSH 기능을 사용할 수 있습니다. 기존 터널의 구성은 편집할 수 없으므로 수동 설정 방법을 사용할 수 없습니다.

브라우저 기반 SSH 기능을 사용하려는 경우 소스 액세스 토큰을 다운로드하거나 로컬 프록시를 구성할 필요가 없습니다. 웹 기반 로컬 프록시가 자동으로 구성되므로 원격 디바이스와의 상호 작용을 시작할 수 있습니다.

빠른 설정 방법 및 브라우저 기반 SSH를 사용하려면

1. 생성한 사물인 RemoteDevice1의 세부 정보 페이지로 이동하여 Create secure tunnel(보안 터널 생성)을 선택합니다.
2. Use existing tunnel(기존 터널 사용)을 선택하여 원격 디바이스에 대해 생성한 최근의 열린 터널을 엽니다. 터널 구성은 편집할 수 없으므로 터널에 대한 수동 설정 방법을 사용할 수 없습니다. 빠른 설정 방법을 사용하려면 Quick setup(빠른 설정)을 선택합니다.
3. 터널 구성 세부 정보를 검토하고 확인한 후 터널을 생성합니다. 터널 구성은 편집할 수 없습니다.

   터널을 생성할 때 보안 터널링은 RotateTunnelAccessToken API 작업을 사용하여 원래 액세스 토큰을 취소하고 새 액세스 토큰을 생성합니다. 원격 디바이스에서 MQTT를 사용하는 경우 이러한 토큰은 구독한 MQTT 주제에 대해 원격 디바이스에 자동으로 전달됩니다. 이러한 토큰을 소스 디바이스에 수동으로 다운로드하도록 선택할 수도 있습니다.

터널을 생성한 후에는 브라우저 기반 SSH를 사용하여 상황별 명령줄 인터페이스를 통해 콘솔에서 직접 원격 디바이스와 상호 작용할 수 있습니다. 이 명령줄 인터페이스를 사용하려면 생성한 사물에 대한 터널을 선택하고 세부 정보 페이지에서 Command-line interface(명령줄 인터페이스) 섹션을 확장합니다. 로컬 프록시가 이미 구성되어 있으므로 명령을 입력하여 원격 디바이스인 RemoteDevice1에 대한 액세스 및 상호 작용을 빠르게 시작할 수 있습니다.

빠른 설정 방법 및 브라우저 기반 SSH 사용에 대한 자세한 내용은 터널을 열고 브라우저 기반 SSH를 사용하여 원격 디바이스에 액세스 섹션을 참조하세요.

정리

- 터널 닫기

  터널을 다 사용한 후에는 터널을 닫는 것이 좋습니다. 터널이 지정된 터널 지속 시간보다 더 오래 열려 있는 경우 터널이 닫힐 수도 있습니다. 터널이 닫힌 후에는 터널을 다시 열 수 없습니다. 닫힌 터널을 연 다음 Duplicate tunnel(터널 복제)을 선택하여 여전히 터널을 복제할 수 있습니다. 사용할 터널 지속 시간을 지정한 다음 새 터널을 생성합니다.

  - AWS IoT 콘솔에서 개별 터널 또는 여러 터널을 닫으려면 Tunnels hub(터널 허브)로 이동하여 닫을 터널을 선택한 다음 Close tunnel(터널 닫기)을 선택합니다.
  - AWS IoT API 참조 API를 사용하여 개별 터널 또는 여러 터널을 닫으려면 CloseTunnel API 작업을 사용합니다.

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- 터널 삭제

  에서 터널을 영구적으로 삭제할 수 있습니다 AWS 계정.

  > **Warning**
  >
  > 삭제 작업은 영구적이며 취소할 수 없습니다.

  - AWS IoT 콘솔에서 개별 터널 또는 여러 터널을 삭제하려면 터널 허브로 이동하여 삭제할 터널을 선택한 다음 Delete tunnel(터널 삭제)을 선택합니다.
  - AWS IoT API 참조 API를 사용하여 개별 터널 또는 여러 터널을 삭제하려면 CloseTunnel API 작업을 사용합니다. API를 사용하는 경우 `delete` 플래그를 `true`로 설정합니다.

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
        --delete true
    ```

로컬 프록시

로컬 프록시는 WebSocket 보안 연결을 통한 보안 터널링을 사용하여 소스 디바이스에서 실행되는 애플리케이션에서 보낸 데이터를 전송합니다. GitHub에서 로컬 프록시 소스를 다운로드할 수 있습니다.

로컬 프록시는 두 가지 모드인 `source` 또는 `destination` 모드로 실행할 수 있습니다. 소스 모드에서 로컬 프록시는 TCP 연결을 시작하는 클라이언트 애플리케이션과 동일한 디바이스 또는 네트워크에서 실행됩니다. 대상 모드에서 로컬 프록시는 대상 애플리케이션과 함께 원격 디바이스에서 실행됩니다. 터널 멀티플렉싱을 사용하여 단일 터널에서 한 번에 최대 3개의 데이터 스트림을 지원할 수 있습니다. 각 데이터 스트림에 대해 보안 터널링은 여러 개의 TCP 연결을 사용하므로 시간 초과가 발생할 가능성이 줄어듭니다. 자세한 내용은 보안 터널에서 데이터 스트림 멀티플렉싱 및 동시 TCP 연결 사용 단원을 참조하십시오.

로컬 프록시 사용 방법

소스 및 대상 디바이스에서 로컬 프록시를 실행하여 데이터를 보안 터널링 엔드포인트로 전송할 수 있습니다. 디바이스가 웹 프록시를 사용하는 네트워크에 있는 경우 연결을 인터넷에 전달하기 전에 웹 프록시가 연결을 가로챌 수 있습니다. 이 경우 웹 프록시를 사용하도록 로컬 프록시를 구성해야 합니다. 자세한 내용은 웹 프록시를 사용하는 디바이스에 대한 로컬 프록시 구성 단원을 참조하십시오.

로컬 프록시 워크플로

다음 단계에서는 로컬 프록시가 소스 디바이스와 대상 디바이스에서 실행되는 방식을 보여줍니다.

1. 로컬 프록시를 보안 터널링에 연결

   먼저 로컬 프록시가 보안 터널링에 대한 연결을 설정해야 합니다. 로컬 프록시를 시작할 때 다음 인수를 사용하세요.

   - 터널이 열리는 AWS 리전 를 지정하는 `-r` 인수입니다.
   - `-t` 인수는 `OpenTunnel`에서 반환된 소스 또는 대상 클라이언트 액세스 토큰을 전달합니다.

   > **Note**
   >
   > 동일한 클라이언트 액세스 토큰 값을 사용하는 두 개의 로컬 프록시는 동시에 연결할 수 없습니다.

2. 소스 또는 대상 작업 수행

WebSocket 연결이 설정되면 로컬 프록시는 구성에 따라 소스 모드 또는 대상 모드 작업을 수행합니다.

기본적으로 로컬 프록시는 입력/출력(I/O) 오류가 발생하거나 WebSocket 연결이 예기치 않게 닫히는 경우 보안 터널링에 다시 연결하려고 시도합니다. 이로 인해 TCP 연결이 닫힙니다. TCP 소켓 오류가 발생하면 로컬 프록시는 터널을 통해 메시지를 전송해 상대방에게 해당 TCP 연결을 닫도록 알립니다. 기본적으로 로컬 프록시는 항상 SSL 통신을 사용합니다.

3. 로컬 프록시 중지

   터널을 사용한 후에는 로컬 프록시 프로세스를 중지하는 것이 안전합니다. `CloseTunnel`을 호출하여 터널을 명시적으로 닫는 것이 좋습니다. 활성 터널 클라이언트는 `CloseTunnel` 직접 호출 후 즉시 닫히지 않을 수 있습니다.

를 사용하여 터널 AWS Management Console 을 열고 SSH 세션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요터널을 열고 원격 디바이스에 대한 SSH 세션 시작.

로컬 프록시 모범 사례

로컬 프록시를 실행할 때 다음 모범 사례를 따르십시오.

- 액세스 토큰을 전달하기 위해 `-t` 로컬 프록시 인수를 사용하지 마세요. `AWSIOT_TUNNEL_ACCESS_TOKEN` 환경 변수를 사용하여 로컬 프록시에 대한 액세스 토큰을 설정하는 것이 좋습니다.
- 운영 체제 또는 환경에서 최소 권한으로 로컬 프록시 실행 파일을 실행합니다.
  - Windows에서 관리자로 로컬 프록시를 실행하지 마세요.
  - Linux 및 macOS에서 로컬 프록시를 루트로 실행하지 마세요.
- 별도의 호스트, 컨테이너, 샌드박스, chroot jail 또는 가상화 환경에서 로컬 프록시를 실행하는 것이 좋습니다.
- 도구 체인에 따라 관련 보안 플래그를 사용하여 로컬 프록시를 빌드합니다.
- 네트워크 인터페이스가 여러 개인 디바이스에서 `-b` 인수를 사용하여 TCP 소켓을 대상 애플리케이션과 통신하는 데 사용되는 네트워크 인터페이스에 바인딩합니다.

예제 명령 및 출력

다음은 실행하는 명령 및 해당 출력의 예입니다. 이 예제에서는 `source` 및 `destination` 모드 모두에서 로컬 프록시를 구성하는 방법을 보여 줍니다. 로컬 프록시는 HTTPS 프로토콜을 WebSocket으로

업그레이드하여 장기 연결을 설정한 다음 보안 터널링 디바이스 엔드포인트에 대한 연결을 통해 데이터 전송을 시작합니다.

다음 명령을 실행하기 전에:

터널을 열고 소스 및 대상에 대한 클라이언트 액세스 토큰을 가져와야 합니다. 또한 앞에서 설명한 대로 로컬 프록시를 구축해야 합니다. 로컬 프록시를 구축하려면 GitHub 리포지토리에서 로컬 프록시 소스 코드를 열고 로컬 프록시 빌드 및 설치 지침을 따릅니다.

> **Note**
>
> 예제에 사용된 다음 명령은 `verbosity` 플래그를 사용하여, 로컬 프록시 실행 후 이전에 설명한 몇 가지 단계들을 간단히 보여줍니다. 이 플래그는 테스트 용도로만 사용하는 것이 좋습니다.

소스 모드에서 로컬 프록시 실행

다음 명령은 소스 모드에서 로컬 프록시를 실행하는 방법을 보여줍니다.

Linux/macOS

Linux 또는 macOS의 경우 터미널에서 다음 명령을 실행하여 소스에서 로컬 프록시를 구성하고 시작합니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -s 5555 -v 5 -r us-west-2
```

위치:

- `-s`는 소스 수신 대기 포트이며, 소스 모드에서 로컬 프록시를 시작합니다.
- `-v`는 출력의 세부 사항 수준이며, 0에서 6 사이의 값을 사용할 수 있습니다.
- `-r`은 터널이 열리는 엔드포인트 리전입니다.

파라미터에 대한 자세한 내용은 명령줄 인수를 사용하여 설정되는 옵션을 참조하세요.

Windows

Windows에서는 Linux 또는 macOS의 경우와 유사한 로컬 프록시를 구성하지만 환경 변수를 정의하는 방법은 다른 플랫폼과 다릅니다. cmd 창에서 다음 명령을 실행하여 소스에서 로컬 프록시를 구성하고 시작합니다.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -s 5555 -v 5 -r us-west-2
```

위치:

- `-s`는 소스 수신 대기 포트이며, 소스 모드에서 로컬 프록시를 시작합니다.
- `-v`는 출력의 세부 사항 수준이며, 0에서 6 사이의 값을 사용할 수 있습니다.
- `-r`은 터널이 열리는 엔드포인트 리전입니다.

파라미터에 대한 자세한 내용은 명령줄 인수를 사용하여 설정되는 옵션을 참조하세요.

> **Note**
>
> 소스 모드에서 최신 버전의 로컬 프록시를 사용하는 경우 이전 버전과의 호환성을 위해 소스 디바이스`--destination-client-type V1`에 AWS CLI 파라미터를 포함해야 합니다. 이는 다음 대상 모드 중 하나에 연결할 때 적용됩니다.
>
> - AWS IoT 디바이스 클라이언트
> - AWS IoT 보안 터널링 구성 요소 또는 AWS IoT Greengrass Version 2 보안 터널링 구성 요소
> - 2022년 이전에 작성된 모든 AWS IoT Secure Tunneling 데모 코드
> - 로컬 프록시의 1.X 버전
>
> 이 파라미터는 업데이트된 소스 프록시와 이전 대상 클라이언트 간의 적절한 통신을 보장합니다. 로컬 프록시 버전에 대한 자세한 내용은 GitHub의 AWS IoT Secure Tunneling을 참조하세요.

다음은 `source` 모드에서 로컬 프록시를 실행하는 샘플 출력을 보여줍니다.

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
```

```
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

대상 모드에서 로컬 프록시 실행

다음 명령은 대상 모드에서 로컬 프록시를 실행하는 방법을 보여줍니다.

Linux/macOS

Linux 또는 macOS의 경우 터미널에서 다음 명령을 실행하여 대상에서 로컬 프록시를 구성하고 시작합니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -d 22 -v 5 -r us-west-2
```

위치:

- `-d`는 대상 모드에서 로컬 프록시를 시작하는 대상 애플리케이션입니다.
- `-v`는 출력의 세부 사항 수준이며, 0에서 6 사이의 값을 사용할 수 있습니다.

- `-r`은 터널이 열리는 엔드포인트 리전입니다.

파라미터에 대한 자세한 내용은 명령줄 인수를 사용하여 설정되는 옵션을 참조하세요.

Windows

Windows에서는 Linux 또는 macOS의 경우와 유사한 로컬 프록시를 구성하지만 환경 변수를 정의하는 방법은 다른 플랫폼과 다릅니다. cmd 창에서 다음 명령을 실행하여 대상에서 로컬 프록시를 구성하고 시작합니다.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -d 22 -v 5 -r us-west-2
```

위치:

- `-d`는 대상 모드에서 로컬 프록시를 시작하는 대상 애플리케이션입니다.
- `-v`는 출력의 세부 사항 수준이며, 0에서 6 사이의 값을 사용할 수 있습니다.
- `-r`은 터널이 열리는 엔드포인트 리전입니다.

파라미터에 대한 자세한 내용은 명령줄 인수를 사용하여 설정되는 옵션을 참조하세요.

> **Note**
>
> 대상 모드에서 최신 버전의 로컬 프록시를 사용하는 경우 이전 버전과의 호환성을 위해 대상 디바이스`--destination-client-type V1`에 AWS CLI 파라미터를 포함해야 합니다. 이는 다음 소스 모드 중 하나에 연결할 때 적용됩니다.
>
> - AWS 콘솔의 브라우저 기반 보안 터널링.
> - 로컬 프록시의 1.X 버전
>
> 이 파라미터는 업데이트된 대상 프록시와 이전 소스 클라이언트 간의 적절한 통신을 보장합니다. 로컬 프록시 버전에 대한 자세한 내용은 GitHub의 AWS IoT Secure Tunneling을 참조하세요.

다음은 `destination` 모드에서 로컬 프록시를 실행하는 샘플 출력을 보여줍니다.

```
...
...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

웹 프록시를 사용하는 디바이스에 대한 로컬 프록시 구성

AWS IoT 디바이스에서 로컬 프록시를 사용하여 AWS IoT 보안 터널링 APIs. 로컬 프록시는 WebSocket 보안 연결을 통한 보안 터널링을 사용하여 디바이스 애플리케이션에서 보낸 데이터를 전송합니다. 로컬 프록시는 `source` 또는 `destination` 모드에서 작동할 수 있습니다. `source` 모드에서, 이 프록시는 TCP 연결을 시작하는 디바이스 또는 네트워크에서 실행됩니다. `destination` 모드에서, 로컬 프록시는 대상 애플리케이션과 함께 원격 디바이스에서 실행됩니다. 자세한 내용은 로컬 프록시 단원을 참조하십시오.

AWS IoT 보안 터널링을 사용하려면 로컬 프록시를 인터넷에 직접 연결해야 합니다. 보안 터널링을 사용한 장기 TCP 연결의 경우 로컬 프록시는 HTTPS 요청을 업그레이드하여 보안 터널링 디바이스 연결 엔드포인트 중 하나에 대한 WebSocket 연결을 설정합니다.

디바이스가 웹 프록시를 사용하는 네트워크에 있는 경우 연결을 인터넷에 전달하기 전에 웹 프록시가 연결을 가로챌 수 있습니다. 보안 터널링 디바이스 연결 엔드포인트에 대한 장기 연결을 설정하려면 websocket 사양에 설명된 대로 웹 프록시를 사용하도록 로컬 프록시를 구성하세요.

> **Note**
>
> AWS IoT 디바이스 클라이언트는 웹 프록시를 사용하는 디바이스를 지원하지 않습니다. 웹 프록시를 사용하려면 로컬 프록시를 사용하고 아래에 설명된 대로 웹 프록시에서 작동하도록 구성해야 합니다.

다음 단계는 로컬 프록시가 웹 프록시에서 작동하는 방식을 보여줍니다.

1. 로컬 프록시는 웹 프록시 인증 정보와 함께 보안 터널링 서비스의 원격 주소가 포함된 HTTP CONNECT 요청을 웹 프록시에 전송합니다.
2. 그런 다음 웹 프록시는 원격 보안 터널링 엔드포인트에 대한 장기 연결을 만듭니다.
3. TCP 연결이 설정되고 로컬 프록시는 이제 데이터 전송을 위해 소스 모드와 대상 모드 모두에서 작동합니다.

이 절차를 완료하려면 다음 단계를 수행하세요.

- 로컬 프록시 빌드
- 웹 프록시 구성
- 로컬 프록시 구성 및 시작

로컬 프록시 빌드

GitHub 리포지토리에서 로컬 프록시 소스 코드를 열고 로컬 프록시 빌드 및 설치 지침을 따릅니다.

웹 프록시 구성

로컬 프록시는 HTTP/1.1 사양에 설명된 HTTP 터널링 메커니즘에 의존합니다. 사양을 준수하려면 웹 프록시에서 디바이스가 CONNECT 메서드를 사용하도록 허용해야 합니다.

웹 프록시를 구성하는 방법은 사용 중인 웹 프록시와 웹 프록시 버전에 따라 다릅니다. 웹 프록시를 올바르게 구성하려면 웹 프록시의 설명서를 확인하세요.

웹 프록시를 구성하려면 먼저 웹 프록시 URL을 식별하고 웹 프록시가 HTTP 터널링을 지원하는지 확인합니다. 나중에 로컬 프록시를 구성하고 시작할 때 웹 프록시 URL이 사용됩니다.

1. 웹 프록시 URL 식별

   웹 프록시 URL의 형식은 다음과 같습니다.

   ```
   protocol://web_proxy_host_domain:web_proxy_port
   ```

   AWS IoT 보안 터널링은 웹 프록시에 대한 기본 인증만 지원합니다. 기본 인증을 사용하려면 웹 프록시 URL의 일부로 **username** 및 **password**를 지정해야 합니다. 웹 프록시 URL은 다음 형식입니다.

   ```
   protocol://username:password@web_proxy_host_domain:web_proxy_port
   ```

   - *####*은 http 또는 https가 될 수 있습니다. https을 사용할 것을 권장합니다.
   - *web_proxy_host_domain*은 웹 프록시의 IP 주소 또는 웹 프록시의 IP 주소로 확인되는 DNS 이름입니다.
   - *web_proxy_port*는 웹 프록시가 수신 대기하는 포트입니다.
   - 웹 프록시는 이 **username** 및 **password**를 사용하여 요청을 인증합니다.

2. 웹 프록시 URL 테스트

   웹 프록시가 TCP 터널링을 지원하는지 확인하려면 curl 명령을 사용하고 2xx 또는 3xx 응답을 받는지 확인하세요.

   예를 들어 웹 프록시 URL이 https://server.com:1235인 경우 웹 프록시가 자체 서명된 인증서에 의존할 수 있으므로 curl 명령과 함께 proxy-insecure 플래그를 사용합니다.

   ```
   export HTTPS_PROXY=https://server.com:1235
   curl -I https://aws.amazon.com --proxy-insecure
   ```

   웹 프록시 URL에 http 포트(예: http://server.com:1234)가 있는 경우 proxy-insecure 플래그를 사용할 필요가 없습니다.

   ```
   export HTTPS_PROXY=http://server.com:1234
   ```

```
curl -I https://aws.amazon.com
```

로컬 프록시 구성 및 시작

웹 프록시를 사용하도록 로컬 프록시를 구성하려면 웹 프록시가 사용하는 DNS 도메인 이름 또는 IP 주소 및 포트 번호로 HTTPS_PROXY 환경 변수를 구성해야 합니다.

로컬 프록시를 구성한 후에는 이 README 문서에 설명된 대로 로컬 프록시를 사용할 수 있습니다.

> **Note**
>
> 환경 변수 선언은 대소문자를 구분합니다. 모든 대문자 또는 소문자를 사용하여 각 변수를 한 번 정의하는 것이 좋습니다. 다음 예는 대문자로 선언된 환경 변수를 보여줍니다. 대문자와 소문자를 모두 사용하여 동일한 변수를 지정하면 소문자를 사용하여 지정된 변수가 우선합니다.

다음 명령은 웹 프록시를 사용하고 로컬 프록시를 시작하도록 대상에서 실행 중인 로컬 프록시를 구성하는 방법을 보여줍니다.

- AWSIOT_TUNNEL_ACCESS_TOKEN: 이 변수는 대상에 대한 클라이언트 액세스 토큰(CAT)을 보유합니다.
- HTTPS_PROXY: 이 변수는 로컬 프록시를 구성하기 위한 웹 프록시 URL 또는 IP 주소를 보유합니다.

다음 예제에 표시된 명령은 사용하는 운영 체제 및 웹 프록시가 HTTP 또는 HTTPS 포트에서 수신 대기하는지 여부에 따라 다릅니다.

HTTP 포트에서 수신 대기 중인 웹 프록시

웹 프록시가 HTTP 포트에서 수신 대기 중인 경우 HTTPS_PROXY 변수에 웹 프록시 URL 또는 IP 주소를 제공할 수 있습니다.

Linux/macOS

Linux 또는 macOS의 경우 터미널에서 다음 명령을 실행하여 HTTP 포트를 수신 대기하는 웹 프록시를 사용하도록 대상에서 로컬 프록시를 구성하고 시작합니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
```

```
./localproxy -r us-east-1 -d 22
```

프록시로 인증해야 하는 경우 **username** 및 **password**를 HTTPS_PROXY 변수의 일부로 지정해야 합니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

Windows

Windows에서는 Linux 또는 macOS의 경우와 유사한 로컬 프록시를 구성하지만 환경 변수를 정의하는 방법은 다른 플랫폼과 다릅니다. cmd 창에서 다음 명령을 실행하여 HTTP 포트를 수신 대기하는 웹 프록시를 사용하도록 대상에서 로컬 프록시를 구성하고 시작합니다.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22
```

프록시로 인증해야 하는 경우 **username** 및 **password**를 HTTPS_PROXY 변수의 일부로 지정해야 합니다.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22
```

HTTPS 포트에서 수신 대기 중인 웹 프록시

웹 프록시가 HTTPS 포트에서 수신 대기하는 경우 다음 명령을 실행합니다.

> **Note**
>
> 웹 프록시에 자체 서명된 인증서를 사용하거나 기본 OpenSSL 지원 및 기본 구성이 없는 OS에서 로컬 프록시를 실행하는 경우 GitHub 리포지토리의 인증서 설정 섹션에 설명된 대로 웹 프록시 인증서를 설정해야 합니다.

다음 명령은 앞에서 설명한 대로 설치한 인증서 파일의 경로도 지정하는 경우를 제외하고 HTTP 프록시에 대해 웹 프록시를 구성한 방법과 유사합니다.

Linux/macOS

Linux 또는 macOS의 경우 터미널에서 다음 명령을 실행하여 HTTPS 포트를 수신 대기하는 웹 프록시를 사용하도록 대상에서 실행 중인 로컬 프록시를 구성합니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

프록시로 인증해야 하는 경우 **username** 및 **password**를 HTTPS_PROXY 변수의 일부로 지정해야 합니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

Windows

Windows의 cmd 창에서 다음 명령을 실행하여 HTTP 포트를 수신 대기하는 웹 프록시를 사용하도록 대상에서 로컬 프록시를 구성하고 시작합니다.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

프록시로 인증해야 하는 경우 **username** 및 **password**를 HTTPS_PROXY 변수의 일부로 지정해야 합니다.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

예제 명령 및 출력

다음은 Linux OS에서 실행하는 명령 및 해당 출력의 예입니다. 이 예에서는 HTTP 포트에서 수신 대기하는 웹 프록시를 보여주고 source 및 destination 모드 모두에서 웹 프록시를 사용하도록 로컬 프록시를 구성하는 방법을 보여줍니다. 이러한 명령을 실행하려면 먼저 터널을 열고 소스 및 대상에 대한 클라이언트 액세스 토큰을 가져와야 합니다. 또한 로컬 프록시를 빌드하고 앞에서 설명한 대로 웹 프록시를 구성해야 합니다.

다음은 로컬 프록시를 시작한 후의 단계에 대한 개요입니다. 로컬 프록시:

- URL을 사용하여 프록시 서버에 연결할 수 있도록 웹 프록시 URL을 식별합니다.
- 웹 프록시와의 TCP 연결을 설정합니다.
- 웹 프록시에 HTTP CONNECT 요청을 전송하고 연결이 설정되었음을 나타내는 HTTP/1.1 200 응답을 기다립니다.
- HTTPS 프로토콜을 WebSockets으로 업그레이드하여 장기 연결을 설정합니다.
- 보안 터널링 디바이스 엔드포인트에 대한 연결을 통해 데이터 전송을 시작합니다.

> **Note**
>
> 예제에 사용된 다음 명령은 verbosity 플래그를 사용하여, 로컬 프록시 실행 후 이전에 설명한 몇 가지 단계들을 간단히 보여줍니다. 이 플래그는 테스트 용도로만 사용하는 것이 좋습니다.

소스 모드에서 로컬 프록시 실행

다음 명령은 소스 모드에서 로컬 프록시를 실행하는 방법을 보여줍니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -s 5555 -v 5 -r us-west-2
```

다음은 source 모드에서 로컬 프록시를 실행하는 샘플 출력을 보여줍니다.

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.

...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.11
```

```
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 0a109afffee745f5-00001341-000b8138-cc6c878d80e8adb0-f186064b
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

대상 모드에서 로컬 프록시 실행

다음 명령은 대상 모드에서 로컬 프록시를 실행하는 방법을 보여줍니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -d 22 -v 5 -r us-west-2
```

다음은 destination 모드에서 로컬 프록시를 실행하는 샘플 출력을 보여줍니다.

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.

...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.1
Connected successfully with Web Proxy
```

```
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 06717bfffed3fd05-00001355-000b8315-da3109a85da804dd-24c3d10d
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

보안 터널에서 데이터 스트림 멀티플렉싱 및 동시 TCP 연결 사용

보안 터널링 멀티플렉싱 기능을 사용하여 터널당 여러 데이터 스트림을 사용할 수 있습니다. 멀티플렉싱을 통해 여러 데이터 스트림을 사용하여 디바이스 문제를 해결할 수 있습니다. 또한 여러 로컬 프록시를 구축, 배포 및 시작하거나 동일한 디바이스에서 여러 터널을 열 필요가 없으므로 운영 부하를 줄일 수 있습니다. 예를 들어 여러 HTTP 및 SSH 데이터 스트림을 전송해야 하는 웹 브라우저의 경우 멀티플렉싱을 사용할 수 있습니다.

각 데이터 스트림에 대해 AWS IoT 보안 터널링은 동시 TCP 연결을 지원합니다. 동시 연결을 사용하면 클라이언트에서 요청이 여러 개 있을 때 시간 초과가 발생할 가능성이 줄어듭니다. 예를 들어 대상 디바이스에 로컬인 웹 서버에 원격으로 액세스할 때 로딩 시간을 줄일 수 있습니다.

다음 섹션에서는 멀티플렉싱 및 동시 TCP 연결 사용과 다양한 사용 사례에 대해 자세히 설명합니다.

주제

- 보안 터널에서 다중 데이터 스트림 멀티플렉싱
- 보안 터널에서 동시 TCP 연결 사용

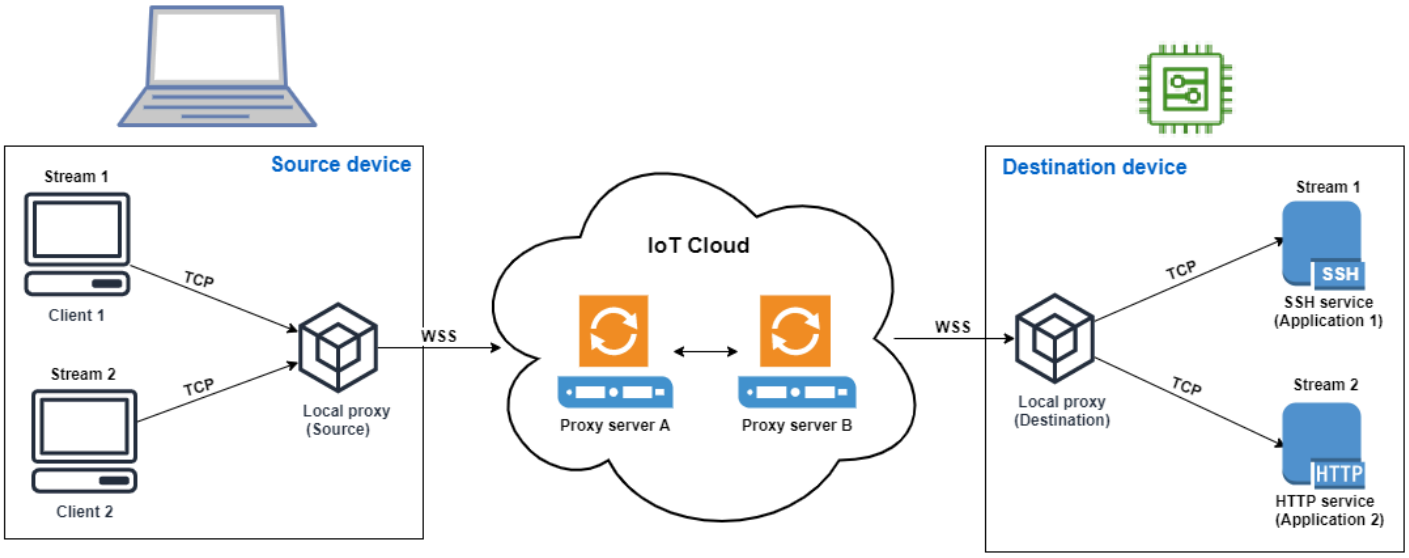
보안 터널에서 다중 데이터 스트림 멀티플렉싱

다중 연결 또는 포트를 사용하는 디바이스에 멀티플렉싱 기능을 사용할 수 있습니다. 문제 해결을 위해 원격 디바이스에 여러 번 연결해야 하는 경우에도 멀티플렉싱을 사용할 수 있습니다. 예를 들어 여러

HTTP 및 SSH 데이터 스트림을 전송해야 하는 웹 브라우저의 경우 멀티플렉싱을 사용할 수 있습니다. 두 스트림의 애플리케이션 데이터는 멀티플렉싱된 터널을 통해 디바이스로 동시에 전송됩니다.

사용 사례

예를 들어 디바이스 내 웹 애플리케이션에 연결하여 일부 네트워킹 파라미터를 변경하고 터미널을 통해 셸 명령을 실행하여 디바이스가 새 네트워킹 파라미터로 제대로 작동하는지 확인해야 할 수 있습니다. 이 시나리오에서는 웹 애플리케이션 및 터미널에 동시에 액세스하기 위해 HTTP 및 SSH 모두를 통해 디바이스에 연결하고 두 개의 병렬 데이터 스트림을 전송해야 할 수 있습니다. 멀티플렉싱 기능을 사용하면 이러한 두 개의 독립 스트림을 동일한 터널을 통해 동시에 전송할 수 있습니다.



멀티플렉싱 터널을 설정하는 방법

다음 절차에서는 여러 포트에 연결해야 하는 애플리케이션을 사용하여 디바이스 문제를 해결하기 위해 멀티플렉싱 터널을 설정하는 방법을 설명합니다. 하나의 HTTP 스트림과 하나의 SSH 스트림이라는 두 개의 멀티플렉싱된 스트림으로 하나의 터널을 설정합니다.

1. (선택 사항) 구성 파일 생성

   구성 파일을 사용하여 소스 및 대상 디바이스를 선택적으로 구성할 수 있습니다. 포트 매핑이 자주 변경될 가능성이 있는 경우 구성 파일을 사용합니다. CLI를 사용하여 포트 매핑을 명시적으로 지정하려는 경우 또는 지정된 수신 대기 포트에서 로컬 프록시를 시작할 필요가 없는 경우 이 단계를 건너뛸 수 있습니다. 구성 파일을 사용하는 방법에 대한 자세한 내용은 GitHub의 Options set via --config(--config를 통해 설정되는 옵션)를 참조하세요.

1. 소스 디바이스의 로컬 프록시가 실행될 폴더에 Config라는 구성 폴더를 생성합니다. 이 폴더 안에 다음 내용으로 SSHSource.ini라는 파일을 만듭니다.

```
HTTP1 = 5555
SSH1 = 3333
```

2. 대상 디바이스의 로컬 프록시가 실행될 폴더에 Config라는 구성 폴더를 생성합니다. 이 폴더 안에 다음 내용으로 SSHDestination.ini라는 파일을 만듭니다.

```
HTTP1 = 80
SSH1 = 22
```

2. 터널 열기

OpenTunnel API 작업 또는 open-tunnel CLI 명령을 사용하여 터널을 엽니다. SSH1 및를 서비스 HTTP1 및 원격 디바이스에 해당하는 AWS IoT 사물 이름으로 지정하여 대상을 구성합니다. SSH 및 HTTP 애플리케이션이 이 원격 디바이스에서 실행되고 있습니다. AWS IoT 레지스트리에 IoT 사물을 이미 생성했어야 합니다. 자세한 내용은 레지스트리로 사물 관리 단원을 참조하십시오.

```
aws iotsecuretunneling open-tunnel \
 --destination-config thingName=RemoteDevice1,services=HTTP1,SSH1
```

이 명령을 실행하면 로컬 프록시를 실행하는 데 사용할 소스 및 대상 액세스 토큰이 생성됩니다.

```
{
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

3. 로컬 프록시 구성 및 시작

로컬 프록시를 실행하려면 먼저 AWS IoT Device Client를 설정하거나 GitHub에서 로컬 프록시 소스 코드를 다운로드하여 원하는 플랫폼에 맞게 빌드합니다. 그런 다음 대상 및 소스 로컬 프록시를 시작하여 보안 터널에 연결할 수 있습니다. 로컬 프록시 구성 및 사용에 대한 자세한 내용은 로컬 프록시 사용 방법 섹션을 참조하세요.

> **Note**
>
> 소스 디바이스에서 구성 파일을 사용하지 않거나 CLI를 사용하여 포트 매핑을 지정하지 않아도 동일한 명령을 사용하여 로컬 프록시를 실행할 수 있습니다. 소스 모드의 로컬 프록시는 사용할 수 있는 포트와 매핑을 자동으로 선택합니다.

Start local proxy using configuration files

구성 파일을 사용하여 소스 및 대상 모드에서 로컬 프록시를 실행하려면 다음 명령을 실행합니다.

```
// ----------------- Start the destination local proxy ----------------------
./localproxy -r us-east-1 -m dst -t destination_client_access_token

// ----------------- Start the source local proxy ---------------------------
// You also run the same command below if you want the local proxy to
// choose the mappings for you instead of using configuration files.
./localproxy -r us-east-1 -m src -t source_client_access_token
```

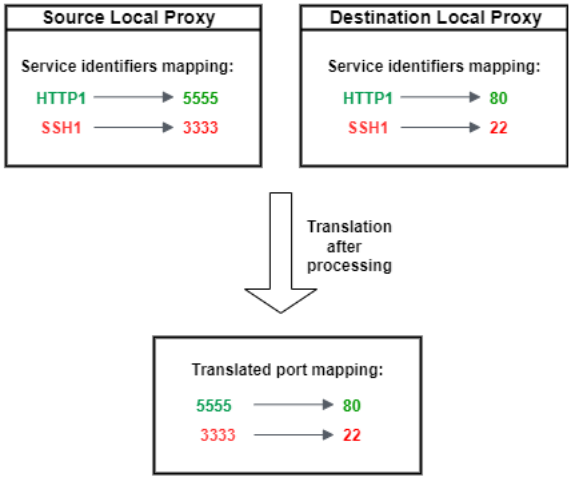
Start local proxy using CLI port mapping

CLI를 사용하여 포트 매핑을 명시적으로 지정해 소스 및 대상 모드에서 로컬 프록시를 실행하려면 다음 명령을 실행합니다.

```
// ----------------- Start the destination local proxy
 ----------------------------------
./localproxy -r us-east-1 -d HTTP1=80,SSH1=22 -t destination_client_access_token

// ----------------- Start the source local proxy
 ---------------------------------------
./localproxy -r us-east-1 -s HTTP1=5555,SSH1=33 -t source_client_access_token
```

이제 SSH 및 HTTP 연결의 애플리케이션 데이터를 멀티플렉싱 터널을 통해 동시에 전송할 수 있습니다. 아래 맵에서 볼 수 있듯이 서비스 식별자는 소스 디바이스와 대상 디바이스 간에 포트 매핑을 변환하는 읽기 쉬운 형식으로 작동합니다. 이 구성을 사용하면, 보안 터널링이 소스 디바이스의 포트 *5555*에서 들어오는 모든 HTTP 트래픽을 대상 디바이스의 포트 *80*으로 전달하고, 포트 *3333*에서 들어오는 모든 SSH 트래픽을 대상 디바이스의 포트 *22*로 전달합니다.



보안 터널에서 동시 TCP 연결 사용

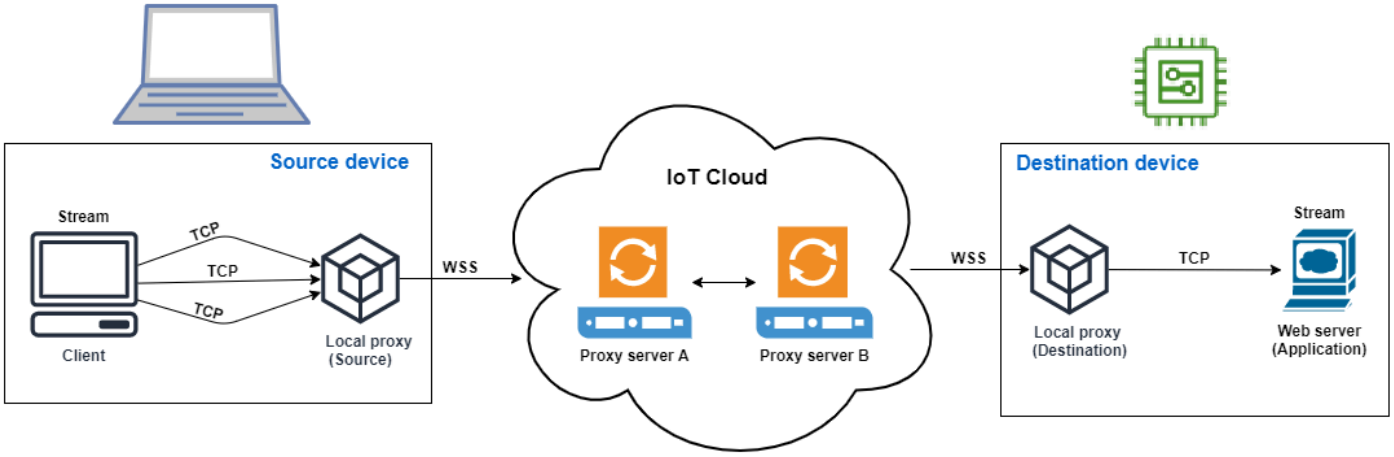
AWS IoT 보안 터널링은 각 데이터 스트림에 대해 둘 이상의 TCP 연결을 동시에 지원합니다. 원격 디바이스에 동시 연결이 필요한 경우 이 기능을 사용할 수 있습니다. 동시 TCP 연결을 사용하면 클라이언트에서 요청이 여러 개 있을 때 시간 초과가 발생할 가능성이 줄어듭니다. 예를 들어 여러 구성 요소가 실행되는 웹 서버에 액세스할 때 동시 TCP 연결을 사용하면 사이트를 로드하는 데 걸리는 시간을 줄일 수 있습니다.

> **Note**
>
> 동시 TCP 연결의 대역폭 제한은 각에 대해 초당 800KB입니다 AWS 계정. AWS IoT 보안 터널링은 수신 요청 수에 따라이 제한을 구성할 수 있습니다.

사용 사례

예를 들어 대상 디바이스에 로컬이고 여러 구성 요소가 실행되는 웹 서버에 원격으로 액세스해야 할 수 있습니다. 단일 TCP 연결을 사용하여 웹 서버에 액세스하려고 할 때 순차적으로 로드하면 사이트에서 리소스를 로드하는 데 걸리는 시간이 늘어날 수 있습니다. 동시 TCP 연결은 사이트의 리소스 요구 사항을 충족하여 로딩 시간을 단축함으로써 액세스 시간을 줄일 수 있습니다. 다음 다이어그램은 원격 디바이스에서 실행되는 웹 서버 애플리케이션에 대한 데이터 스트림에 동시 TCP 연결이 지원되는 방법을 보여줍니다.

> **Note**
>
> 터널을 사용하여 원격 디바이스에서 실행되는 여러 애플리케이션에 액세스하려는 경우 터널 멀티플렉싱을 사용할 수 있습니다. 자세한 내용은 보안 터널에서 다중 데이터 스트림 멀티플렉싱 단원을 참조하십시오.



동시 TCP 연결을 사용하는 방법

다음 절차에서는 동시 TCP 연결을 사용하여 원격 디바이스의 웹 브라우저에 액세스하는 방법을 안내합니다. 클라이언트의 요청이 여러 개 있는 경우 AWS IoT 보안 터널링은 요청을 처리하기 위해 동시 TCP 연결을 자동으로 설정하여 로드 시간을 줄입니다.

1. 터널 열기

   OpenTunnel API 작업 또는 `open-tunnel` CLI 명령을 사용하여 터널을 엽니다. HTTP를 서비스로 지정하고 원격 디바이스에 해당하는 AWS IoT 사물의 이름을 지정하여 대상을 구성합니다. 웹 서버 애플리케이션이 이 원격 디바이스에서 실행되고 있습니다. AWS IoT 레지스트리에서 IoT 사물을 이미 생성했어야 합니다. 자세한 내용은 레지스트리로 사물 관리 단원을 참조하십시오.

   ```
   aws iotsecuretunneling open-tunnel \
    --destination-config thingName=RemoteDevice1,services=HTTP
   ```

   이 명령을 실행하면 로컬 프록시를 실행하는 데 사용할 소스 및 대상 액세스 토큰이 생성됩니다.

   ```
   {
   ```

```
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

2. 로컬 프록시 구성 및 시작

   로컬 프록시를 실행하려면 먼저 GitHub에서 로컬 프록시 소스 코드를 다운로드하여 선택한 플랫폼에 맞게 빌드해야 합니다. 그런 다음 대상 및 소스 로컬 프록시를 시작하여 보안 터널에 연결하고 원격 웹 서버 애플리케이션을 사용할 수 있습니다.

   > **Note**
   >
   > AWS IoT 보안 터널링에서 동시 TCP 연결을 사용하려면 최신 버전의 로컬 프록시로 업그레이드해야 합니다. AWS IoT 디바이스 클라이언트를 사용하여 로컬 프록시를 구성한 경우에는 이 기능을 사용할 수 없습니다.

```
// Start the destination local proxy
./localproxy -r us-east-1 -d HTTP=80 -t destination_client_access_token

// Start the source local proxy
./localproxy -r us-east-1 -s HTTP=5555 -t source_client_access_token
```

   로컬 프록시 구성 및 사용에 대한 자세한 내용은 로컬 프록시 사용 방법 섹션을 참조하세요.

이제 터널을 사용하여 웹 서버 애플리케이션에 액세스할 수 있습니다. 클라이언트에서 여러 요청이 있을 때 AWS IoT 보안 터널링이 자동으로 설정되고 동시 TCP 연결을 처리합니다.

원격 디바이스 구성 및 IoT 에이전트 사용

IoT 에이전트는 클라이언트 액세스 토큰을 포함하는 MQTT 메시지를 받고 원격 디바이스에서 로컬 프록시를 시작하는 데 사용됩니다. 보안 터널링에서 MQTT를 사용하여 클라이언트 액세스 토큰을 전달하려면 원격 디바이스에 IoT 에이전트를 설치하고 실행해야 합니다. IoT 에이전트는 다음과 같은 예약된 IoT MQTT 주제를 구독해야 합니다.

> **Note**
>
> 예약된 MQTT 주제 구독 이외의 방법을 통해 대상 클라이언트 액세스 토큰을 원격 디바이스에 전달하려는 경우 대상 클라이언트 액세스 토큰(CAT) 리스너와 로컬 프록시가 필요할 수 있습니다. CAT 리스너는 선택한 클라이언트 액세스 토큰 전달 메커니즘과 함께 작동하고 대상 모드에서 로컬 프록시를 시작할 수 있어야 합니다.

IoT 에이전트 코드 조각

IoT 에이전트는 다음과 같은 예약된 IoT MQTT 주제를 구독해야 MQTT 메시지를 수신하고 로컬 프록시를 시작할 수 있습니다.

`$aws/things/thing-name/tunnels/notify`

여기서 `thing-name`는 원격 디바이스와 연결된 AWS IoT 사물의 이름입니다.

다음은 MQTT 메시지 페이로드의 예입니다.

```
{
    "clientAccessToken": "destination-client-access-token",
    "clientMode": "destination",
    "region": "aws-region",
    "services": ["destination-service"]
}
```

IoT 에이전트는 MQTT 메시지를 받은 후 적절한 파라미터를 사용하여 원격 디바이스에서 로컬 프록시를 시작해야 합니다.

다음 Java 코드에서는 Java 라이브러리의 AWS IoT 디바이스 SDK 및 ProcessBuilder를 사용하여 보안 터널링을 사용하는 간단한 IoT 에이전트를 빌드하는 방법을 보여줍니다.

```
// Find the IoT device endpoint for your AWS ##
final String endpoint = iotClient.describeEndpoint(new
 DescribeEndpointRequest().withEndpointType("iot:Data-ATS")).getEndpointAddress();

// Instantiate the IoT Agent with your AWS credentials
final String thingName = "RemoteDeviceA";
final String tunnelNotificationTopic = String.format("$aws/things/%s/tunnels/notify",
 thingName);
final AWSIotMqttClient mqttClient = new AWSIotMqttClient(endpoint, thingName,
```

```
                "your_aws_access_key", "your_aws_secret_key");

try {
    mqttClient.connect();
    final TunnelNotificationListener listener = new
 TunnelNotificationListener(tunnelNotificationTopic);
    mqttClient.subscribe(listener, true);
}
finally {
    mqttClient.disconnect();
}

private static class TunnelNotificationListener extends AWSIotTopic {
    public TunnelNotificationListener(String topic) {
        super(topic);
    }

    @Override
    public void onMessage(AWSIotMessage message) {
        try {
            // Deserialize the MQTT message
            final JSONObject json = new JSONObject(message.getStringPayload());

            final String accessToken = json.getString("clientAccessToken");
            final String region = json.getString("region");

            final String clientMode = json.getString("clientMode");
            if (!clientMode.equals("destination")) {
                throw new RuntimeException("Client mode " + clientMode + " in the MQTT
 message is not expected");
            }

            final JSONArray servicesArray = json.getJSONArray("services");
            if (servicesArray.length() > 1) {
                throw new RuntimeException("Services in the MQTT message has more than
 1 service");
            }
            final String service = servicesArray.get(0).toString();
            if (!service.equals("SSH")) {
                throw new RuntimeException("Service " + service + " is not supported");
            }

            // Start the destination local proxy in a separate process to connect to
 the SSH Daemon listening port 22
```

```
            final ProcessBuilder pb = new ProcessBuilder("localproxy",
                        "-t", accessToken,
                        "-r", region,
                        "-d", "localhost:22");
            pb.start();
        }
        catch (Exception e) {
            log.error("Failed to start the local proxy", e);
        }
    }
}
```

터널에 대한 액세스 제어

보안 터널링은 IAM 권한 정책에 사용할 서비스별 작업, 리소스 및 조건 컨텍스트 키를 제공합니다.

터널 액세스 사전 조건

- IAM 정책을 사용하여 AWS 리소스를 보호하는 방법을 알아봅니다.
- IAM 조건을 생성하고 평가하는 방법을 알아봅니다.
- AWS 리소스 태그를 사용하여 리소스를 보호하는 방법을 알아봅니다.

터널 액세스 정책

보안 터널링 API를 사용하려면 권한 부여에 다음 정책을 사용해야 합니다. AWS IoT 보안에 대한 자세한 내용은 단원을 참조하십시오에 대한 자격 증명 및 액세스 관리 AWS IoT.

iot:OpenTunnel

`iot:OpenTunnel` 정책 작업은 보안 주체에게 OpenTunnel을 호출할 수 있는 권한을 부여합니다.

IAM 정책 명령문의 `Resource` 요소에서 다음을 수행합니다.

- 와일드카드 터널 ARN을 지정합니다.

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`
- 특정 IoT 사물에 대한 `OpenTunnel` 권한을 관리할 사물 ARN을 지정합니다.

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:thing/`*`thing-name`*

예를 들어, 다음 정책 설명을 사용하면 TestDevice라는 IoT 사물에 대한 터널을 열 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:OpenTunnel 정책 작업은 다음 조건 키를 지원합니다.

- iot:ThingGroupArn
- iot:TunnelDestinationService
- aws:RequestTag/*tag-key*
- aws:SecureTransport
- aws:TagKeys

다음 정책 설명을 사용하면 사물이 이름이 TestGroup으로 시작하는 사물 그룹에 속하고 터널에서 구성된 대상 서비스가 SSH인 경우 사물에 대한 터널을 열 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ]
        }
    }
```

```
}
```

리소스 태그를 사용하여 터널을 열 수 있는 권한을 제어할 수도 있습니다. 예를 들어, 다음 정책 설명은 태그 키 Owner의 값이 Admin이고 다른 태그가 지정되지 않은 경우 터널을 열 수 있도록 허용합니다. 태그 사용에 대한 일반적인 정보는 AWS IoT 리소스에 태그 지정 섹션을 참조하세요.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/Owner": "Admin"
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "Owner"
        }
    }
}
```

iot:RotateTunnelAccessToken

iot:RotateTunnelAccessToken 정책 작업은 보안 주체에게 RotateTunnelAccessToken을 호출할 수 있는 권한을 부여합니다.

IAM 정책 명령문의 Resource 요소에서 다음을 수행합니다.

- 정규화된 터널 ARN을 지정합니다.

  arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

  와일드카드 터널 ARN을 사용할 수도 있습니다.

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

- 특정 IoT 사물에 대한 RotateTunnelAccessToken 권한을 관리할 사물 ARN을 지정합니다.

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

예를 들어, 다음 정책 문을 사용하면 TestDevice라는 IoT 사물에 대한 터널의 소스 액세스 토큰 또는 클라이언트의 대상 액세스 토큰을 교체할 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:RotateTunnelAccessToken 정책 작업은 다음 조건 키를 지원합니다.

- iot:ThingGroupArn
- iot:TunnelDestinationService
- iot:ClientMode
- aws:SecureTransport

다음 정책 문을 사용하면 사물이 이름이 TestGroup으로 시작하는 사물 그룹에 속하고 터널에서 구성된 대상 서비스가 SSH이며 클라이언트가 DESTINATION 모드인 경우 사물에 대한 대상 액세스 토큰을 교체할 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ],
            "iot:ClientMode": "DESTINATION"
```

```
        }
    }
}
```

iot:DescribeTunnel

`iot:DescribeTunnel` 정책 작업은 보안 주체에게 DescribeTunnel을 호출할 수 있는 권한을 부여합니다.

IAM 정책 문의 `Resource` 요소에서 정규화된 터널 ARN을 지정합니다.

`arn:aws:iot:`*`aws-region`*`: `*`aws-account-id`*`:tunnel/`*`tunnel-id`*

와일드카드 ARN을 사용할 수도 있습니다.

`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`

`iot:DescribeTunnel` 정책 작업은 다음 조건 키를 지원합니다.

- `aws:ResourceTag/`*`tag-key`*
- `aws:SecureTransport`

다음 정책 설명을 사용하면 요청된 터널에 값이 `Admin`인 키 `Owner` 태그가 지정된 경우 `DescribeTunnel`을 호출할 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:DescribeTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/Owner": "Admin"
        }
    }
}
```

iot:ListTunnels

`iot:ListTunnels` 정책 작업은 보안 주체에게 ListTunnels를 호출할 수 있는 권한을 부여합니다.

IAM 정책 명령문의 Resource 요소에서 다음을 수행합니다.

- 와일드카드 터널 ARN을 지정합니다.

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*
- 선택한 IoT 사물에 대한 ListTunnels 권한을 관리할 사물 ARN을 지정합니다.

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

iot:ListTunnels 정책 작업은 조건 키 aws:SecureTransport를 지원합니다.

다음 정책 설명을 사용하면 이름이 TestDevice인 사물에 대한 터널을 나열할 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:ListTunnels",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:ListTagsForResource

iot:ListTagsForResource 정책 작업은 보안 주체에게 ListTagsForResource를 호출할 수 있는 권한을 부여합니다.

IAM 정책 문의 Resource 요소에서 정규화된 터널 ARN을 지정합니다.

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

와일드카드 터널 ARN을 사용할 수도 있습니다.

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:ListTagsForResource 정책 작업은 조건 키 aws:SecureTransport를 지원합니다.

iot:CloseTunnel

iot:CloseTunnel 정책 작업은 보안 주체에게 CloseTunnel을 호출할 수 있는 권한을 부여합니다.

IAM 정책 문의 Resource 요소에서 정규화된 터널 ARN을 지정합니다.

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

와일드카드 터널 ARN을 사용할 수도 있습니다.

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:CloseTunnel 정책 작업은 다음 조건 키를 지원합니다.

- iot:Delete
- aws:ResourceTag/*tag-key*
- aws:SecureTransport

다음 정책 문을 사용하면 요청의 Delete 파라미터가 false이고 요청된 터널에 값이 QATeam인 키 Owner 태그가 지정된 경우 CloseTunnel을 호출할 수 있습니다.

```
{
    "Effect": "Allow",
    "Action": "iot:CloseTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "Bool": {
            "iot:Delete": "false"
        },
        "StringEquals": {
            "aws:ResourceTag/Owner": "QATeam"
        }
    }
}
```

iot:TagResource

iot:TagResource 정책 작업은 보안 주체에게 TagResource를 호출할 수 있는 권한을 부여합니다.

IAM 정책 문의 Resource 요소에서 정규화된 터널 ARN을 지정합니다.

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

와일드카드 터널 ARN을 사용할 수도 있습니다.

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

`iot:TagResource` 정책 작업은 조건 키 `aws:SecureTransport`를 지원합니다.

iot:UntagResource

`iot:UntagResource` 정책 작업은 보안 주체에게 `UntagResource`를 호출할 수 있는 권한을 부여합니다.

IAM 정책 문의 `Resource` 요소에서 정규화된 터널 ARN을 지정합니다.

`arn:aws:iot:`*`aws-region`*`: `*`aws-account-id`*`:tunnel/`*`tunnel-id`*

와일드카드 터널 ARN을 사용할 수도 있습니다.

`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`

`iot:UntagResource` 정책 작업은 조건 키 `aws:SecureTransport`를 지원합니다.

클라이언트 액세스 토큰을 교체하여 AWS IoT 보안 터널링 연결 문제 해결

AWS IoT 보안 터널링을 사용하는 경우 터널이 열려 있더라도 연결 문제가 발생할 수 있습니다. 다음 섹션에서는 발생할 수 있는 문제와 클라이언트 액세스 토큰을 교체하여 문제를 해결하는 방법을 보여줍니다. 클라이언트 액세스 토큰(CAT)을 교체하려면 RotateTunnelAccessToken API 또는 rotate-tunnel-access-token AWS CLI을 사용합니다. 클라이언트를 사용하면서 오류가 발생할 때의 모드가 소스 모드인지 대상 모드인지에 따라 CAT를 소스 또는 대상 모드 또는 둘 다에서 교체할 수 있습니다.

> **Note**
>
> - CAT를 소스 또는 대상 중 어디에서 교체해야 할지 확실하지 않다면 `RotateTunnelAccessToken` API를 사용할 때 `ClientMode`를 전체(ALL)로 설정하여 소스와 대상 모두에서 CAT를 교체할 수 있습니다.
> - CAT를 교체해도 터널 지속 시간은 연장되지 않습니다. 예를 들어, 터널 지속 시간이 12시간이고 터널이 이미 4시간 동안 열려 있었다고 가정합시다. 액세스 토큰을 교체할 때 생성된 새 토큰은 남은 8시간 동안만 사용할 수 있습니다.

주제

- 잘못된 클라이언트 액세스 토큰 오류

- 클라이언트 토큰 불일치 오류
- 원격 디바이스 연결 문제

잘못된 클라이언트 액세스 토큰 오류

AWS IoT 보안 터널링을 사용하는 경우 동일한 클라이언트 액세스 토큰(CAT)을 사용하여 동일한 터널에 다시 연결할 때 연결 오류가 발생할 수 있습니다. 이 경우 로컬 프록시는 보안 터널링 프록시 서버에 연결할 수 없습니다. 소스 모드에서 클라이언트를 사용할 경우 다음 오류 메시지가 표시될 수 있습니다.

```
Invalid access token: The access token was previously used and cannot be used again
```

오류가 발생하는 이유는 클라이언트 액세스 토큰(CAT)이 로컬 프록시에 의해 한 번만 사용될 수 있고 그 후에는 무효화되기 때문입니다. 이 오류를 해결하려면 SOURCE 모드에서 클라이언트 액세스 토큰을 교체하여 소스에 대한 새로운 CAT를 생성하세요. 소스 CAT 교체 방법의 예를 보려면 소스 CAT 교체 예시 섹션을 참조하세요.

클라이언트 토큰 불일치 오류

> **Note**
>
> 클라이언트 토큰을 사용하여 CAT를 재사용하는 것은 권장되지 않습니다. 터널에 다시 연결하기 위해 RotateTunnelAccessToken API를 사용하여 클라이언트 액세스 토큰을 교체하는 것이 좋습니다.

클라이언트 토큰을 사용하는 경우 터널에 다시 연결하기 위해 CAT를 재사용할 수 있습니다. CAT를 재사용하려면 보안 터널링에 처음 연결할 때 CAT와 클라이언트 토큰을 제공해야 합니다. 보안 터널링은 클라이언트 토큰을 저장하므로 동일한 토큰을 사용한 후속 연결을 시도할 경우 클라이언트 토큰도 제공해야 합니다. 클라이언트 토큰 사용에 대한 자세한 내용은 GitHub의 로컬 프록시 참조 구현을 참조하세요.

클라이언트 토큰을 사용할 때 소스 모드에서 클라이언트를 사용할 경우 다음 오류 메시지가 표시될 수 있습니다.

```
Invalid client token: The provided client token does not match the client token
    that was previously set.
```

오류가 발생하는 이유는 제공된 클라이언트 토큰이 터널에 액세스할 때 CAT와 함께 제공된 클라이언트 토큰과 일치하지 않기 때문입니다. 이 오류를 해결하려면 SOURCE 모드에서 CAT를 교체하여 소스에 대한 새로운 CAT를 생성하세요. 다음은 그 한 예입니다.

소스 CAT 교체 예시

다음 예시는 SOURCE 모드에서 RotateTunnelAccessToken API를 실행하여 소스에 대해 새로운 CAT를 생성하는 방법을 보여줍니다.

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --region <region> \
    --tunnel-id <tunnel-id> \
    --client-mode SOURCE
```

이 명령을 실행하면 새 소스 액세스 토큰이 생성되고 터널의 ARN이 반환됩니다.

```
{
    "sourceAccessToken": "<source-access-token>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/<tunnel-id>"
}
```

이제 새 소스 토큰을 사용하여 소스 모드에서 로컬 프록시를 연결할 수 있습니다.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=<source-access-token>
./localproxy -r <region> -s <port>
```

다음은 로컬 프록시를 실행하는 샘플 출력을 보여줍니다.

```
...

[info]    Starting proxy in source mode
...
[info]    Successfully established websocket connection with proxy server ...
[info]    Listening for new connection on port <port>
...
```

원격 디바이스 연결 문제

AWS IoT 보안 터널링을 사용하는 경우 터널이 열려 있더라도 디바이스가 예기치 않게 연결 해제될 수 있습니다. 디바이스가 여전히 터널에 연결되어 있는지 확인하려면 DescribeTunnel API 또는 describe-tunnel AWS CLI을 사용할 수 있습니다.

여러 가지 이유로 디바이스의 연결이 끊어질 수 있습니다. 연결 문제를 해결하려면 다음과 같은 이유로 디바이스의 연결이 끊어진 경우 대상에서 CAT를 교체하면 됩니다.

- 대상의 CAT가 무효화되었습니다.
- 토큰이 보안 터널링 예약 MQTT 주제를 통해 디바이스에 전달되지 않았습니다.

  `$aws/things/<thing-name>/tunnels/notify`

다음 예에서는 이 문제를 해결하는 방법을 보여줍니다.

대상 CAT 교체 예시

원격 디바이스인 *<RemoteThing1>*을 예로 들어보겠습니다. 이 사물에 대해 터널을 열려면 다음 명령을 사용할 수 있습니다.

```
aws iotsecuretunneling open-tunnel \
    --region <region> \
    --destination-config thingName=<RemoteThing1>,services=SSH
```

이 명령을 실행하면 소스 및 대상에 대한 터널 세부 정보와 CAT가 생성됩니다.

```
{
    "sourceAccessToken": "<source-access-token>",
    "destinationAccessToken": "<destination-access-token>",
    "tunnelId": "<tunnel-id>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/tunnel-id"
}
```

그러나 DescribeTunnel API를 사용할 경우 출력은 아래와 같이 디바이스의 연결이 끊어졌음을 나타냅니다.

```
aws iotsecuretunneling describe-tunnel \
    --tunnel-id <tunnel-id> \
    --region <region>
```

이 명령을 실행하면 디바이스가 아직 연결되어 있지 않음을 표시합니다.

```
{
    "tunnel": {
```

```
        ...
        "destinationConnectionState": {
            "status": "DISCONNECTED"
        },
        ...
    }
}
```

이 오류를 해결하려면 클라이언트는 DESTINATION 모드로 설정한 상태에서 RotateTunnelAccessToken API와 대상에 대한 구성을 실행합니다. 이 명령을 실행하면 이전 액세스 토큰이 취소되고 새 토큰이 생성되며 이 토큰을 MQTT 주제로 다시 보냅니다.

$aws/things/*<thing-name>*/tunnels/notify

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

이 명령을 실행하면 아래와 같이 새 액세스 토큰이 생성됩니다. 그런 다음 디바이스 에이전트가 올바르게 설정된 경우 토큰이 디바이스에 전달되어 터널에 연결됩니다.

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

디바이스 프로비저닝

AWS 는 디바이스를 프로비저닝하고 디바이스에 고유한 클라이언트 인증서를 설치하는 여러 가지 방법을 제공합니다. 이 섹션에서는 각각의 방법과 IoT 솔루션에 가장 적합한 방법을 선택하는 방법에 대해 설명합니다. 이러한 옵션은 AWS IoT Core의 X.509 인증서를 사용한 디바이스 제조 및 프로비저닝이라는 백서에 자세히 설명되어 있습니다.

상황에 가장 적합한 옵션을 선택하세요.

- 인증서를 전달하기 전에 IoT 디바이스에 설치할 수 있습니다.

  최종 사용자가 사용할 수 있도록 전달되기 전에 고유한 클라이언트 인증서를 IoT 디바이스에 안전하게 설치할 수 있는 경우 JIT 프로비저닝(JITP) 또는 JIT 등록(JITR)을 사용하려고 합니다.

  JITP 및 JITR을 사용하면 디바이스 인증서에 서명하는 데 사용되는 인증 기관(CA)이에 등록 AWS IoT 되고 디바이스가 처음 연결될 AWS IoT 때에서 인식합니다. 디바이스는 프로비저닝 템플릿의 세부 정보를 사용하여 첫 번째 연결 AWS IoT 에서에 프로비저닝됩니다.

  단일 사물, JITP, JITR 및 고유 인증서가 있는 디바이스의 대량 프로비저닝에 대한 자세한 내용은 the section called “디바이스 인증서가 있는 디바이스 프로비저닝” 단원을 참조하세요.
- 최종 사용자 또는 설치 관리자는 앱을 사용하여 IoT 디바이스에 인증서를 설치할 수 있습니다.

  최종 사용자에게 전달되기 전에 고유한 클라이언트 인증서를 IoT 디바이스에 안전하게 설치할 수 없지만 최종 사용자 또는 설치 관리자가 앱을 사용하여 디바이스를 등록하고 고유한 디바이스 인증서를 설치할 수 있는 경우 신뢰할 수 있는 사용자에 의한 프로비저닝 프로세스를 사용하고자 합니다.

  알려진 계정이 있는 최종 사용자 또는 설치 관리자와 같은 신뢰할 수 있는 사용자를 사용하면 디바이스 제조 프로세스를 간소화할 수 있습니다. 디바이스에는 고유한 클라이언트 인증서 대신 5분 AWS IoT 동안만 디바이스를 연결할 수 있는 임시 인증서가 있습니다. 5분 동안 신뢰할 수 있는 사용자는 수명이 긴 고유한 클라이언트 인증서를 획득하여 디바이스에 설치합니다. 클레임 인증서의 수명이 제한되어 인증서의 손상 위험을 최소화합니다.

  자세한 내용은 the section called “신뢰할 수 있는 사용자에 의한 프로비저닝” 단원을 참조하세요.
- 최종 사용자는 앱을 사용하여 IoT 디바이스에 인증서를 설치할 수 없습니다.

  이전 옵션 중 어느 것도 IoT 솔루션에서 작동하지 않는 경우 클레임에 의한 프로비저닝 프로세스가 옵션입니다. 이 프로세스를 통해 IoT 디바이스에는 플릿의 다른 디바이스에서 공유하는 클레임 인증서가 있습니다. 디바이스가 클레임 인증서에 처음 연결되면는 프로비저닝 템플릿을 사용하여 디바

이스를 AWS IoT 등록하고에 대한 후속 액세스를 위해 디바이스에 고유한 클라이언트 인증서를 발급합니다 AWS IoT.

이 옵션을 사용하면 디바이스를 연결할 때 자동으로 프로비저닝할 수 AWS IoT있지만 클레임 인증서가 손상된 경우 더 큰 위험이 발생할 수 있습니다. 클레임 인증서가 손상되면 인증서를 비활성화할 수 있습니다. 클레임 인증서를 비활성화하면 해당 클레임 인증서가 있는 모든 디바이스가 나중에 등록되지 않습니다. 그러나 클레임 인증서를 비활성화해도 이미 프로비저닝된 디바이스는 차단되지 않습니다.

자세한 내용은 the section called “클레임에 의한 프로비저닝” 단원을 참조하십시오.

에서 디바이스 프로비저닝 AWS IoT

로 디바이스를 프로비저닝 AWS IoT할 때 디바이스와가 안전하게 통신할 AWS IoT 수 있도록 리소스를 생성해야 합니다. 디바이스 플릿을 관리하는 데 도움이 되는 다른 리소스를 생성할 수 있습니다. 프로비저닝 프로세스 중에 다음 리소스를 생성할 수 있습니다.

- IoT 사물.

  IoT 사물은 AWS IoT 디바이스 레지스트리의 항목입니다. 각 사물에는 고유한 이름과 속성 집합이 있으며, 물리적 디바이스와 연결됩니다. 사물 유형을 사용하여 사물을 정의하거나 사물을 사물 그룹으로 그룹화할 수 있습니다. 자세한 내용은 를 사용하여 디바이스 관리 AWS IoT 단원을 참조하세요.

  반드시 생성해야 하는 것은 아니지만 사물을 생성하면 사물 유형, 사물 그룹 및 사물 속성별로 디바이스를 검색하여 디바이스 플릿을 보다 효율적으로 관리할 수 있습니다. 자세한 내용은 플릿 인덱싱 단원을 참조하십시오.

  > **Note**
  >
  > 사물의 연결 상태 데이터를 인덱싱하도록 하려면 사물 이름이 연결 요청에 사용된 클라이언트 ID와 일치하도록 사물을 프로비저닝하고 구성합니다.

- X.509 인증서.

  디바이스는 X.509 인증서를 사용하여 상호 인증을 수행합니다 AWS IoT. 기존 인증서를 등록하거나 새 인증서를 AWS IoT 생성하여 등록할 수 있습니다. 디바이스를 나타내는 사물에 인증서를 연결하여 인증서와 디바이스를 연결합니다. 또한 인증서 및 연결된 프라이빗 키를 디바이스에 복사해야 합니다. 디바이스는에 연결할 때 인증서를 제공합니다 AWS IoT. 자세한 내용은 Authentication 단원을 참조하십시오.

- IoT 정책.

  IoT 정책은 디바이스가 AWS IoT에서 수행할 수 있는 작업을 정의합니다. IoT 정책은 디바이스 인증서에 연결됩니다. 디바이스가 인증서를에 제시하면 정책에 지정된 권한이 부여 AWS IoT됩니다. 자세한 내용은 권한 부여 단원을 참조하십시오. 각 디바이스에는 AWS IoT와 통신하기 위해 인증서가 필요합니다.

AWS IoT 는 프로비저닝 템플릿을 사용하여 자동 플릿 프로비저닝을 지원합니다. 프로비저닝 템플릿은 디바이스를 프로비저닝하는 데 AWS IoT 필요한 리소스를 설명합니다. 템플릿에는 하나의 템플릿을 사용하여 여러 디바이스를 프로비저닝할 수 있는 변수가 포함되어 있습니다. 디바이스를 프로비저닝할 때 사전 또는 맵을 사용하여 디바이스와 관련된 변수의 값을 지정합니다. 다른 디바이스를 프로비저닝하려면 사전에 새 값을 지정합니다.

디바이스에 고유한 인증서(및 연결된 프라이빗 키)가 있는지 여부에 관계없이 자동화된 프로비저닝을 사용할 수 있습니다.

플릿 프로비저닝 API

플릿 프로비저닝에 사용되는 API에는 다음과 같은 몇 가지 범주가 있습니다.

- 이러한 컨트롤 플레인 기능은 플릿 프로비저닝 템플릿을 생성 및 관리하고 신뢰할 수 있는 사용자 정책을 구성합니다.
  - CreateProvisioningTemplate
  - CreateProvisioningTemplateVersion
  - DeleteProvisioningTemplate
  - DeleteProvisioningTemplateVersion
  - DescribeProvisioningTemplate
  - DescribeProvisioningTemplateVersion
  - ListProvisioningTemplates
  - ListProvisioningTemplateVersions
  - UpdateProvisioningTemplate
- 신뢰할 수 있는 사용자는 이 컨트롤 플레인 기능을 사용하여 임시 온보딩 클레임을 생성할 수 있습니다. 이 임시 클레임은 Wi-Fi 구성이나 유사한 방법을 수행하는 중에 디바이스로 전달됩니다.
  - CreateProvisioningClaim

- 프로비저닝 클레임 인증서가 내장되어 있거나 신뢰할 수 있는 사용자가 인증서를 전달하는 디바이스에서 프로비저닝 프로세스 동안 사용되는 MQTT API입니다.
  - the section called “CreateCertificateFromCsr”
  - the section called “CreateKeysAndCertificate”
  - the section called “RegisterThing”

플릿 프로비저닝을 사용하여 디바이스 인증서가 없는 디바이스 프로비저닝

AWS IoT 플릿 프로비저닝을 사용하면 AWS IoT 가 디바이스 인증서와 프라이빗 키를 생성하여 처음에 연결할 때 디바이스 AWS IoT 에 안전하게 전송할 수 있습니다.는 Amazon Root 인증 기관(CA)에서 서명한 클라이언트 인증서를 AWS IoT 제공합니다.

플릿 프로비저닝을 사용하는 방법에는 다음 두 가지가 있습니다.

- 클레임에 의한 프로비저닝
- 신뢰할 수 있는 사용자에 의한 프로비저닝

클레임에 의한 프로비저닝

디바이스는 프로비저닝 클레임 인증서 및 프라이빗 키(특수 용도의 자격 증명)가 내장된 상태로 제조될 수 있습니다. 이러한 인증서가에 등록된 경우 서비스는 이러한 인증서를 디바이스 AWS IoT가 일반 작업에 사용할 수 있는 고유한 디바이스 인증서로 교환할 수 있습니다. 이 프로세스는 다음 단계로 이루어집니다.

디바이스를 전달하기 전에

1. `CreateProvisioningTemplate`을 호출하여 프로비저닝 템플릿을 생성합니다. 이 API는 템플릿 ARN을 반환합니다. 자세한 내용은 디바이스 프로비저닝 MQTT API 단원을 참조하십시오.

   AWS IoT 콘솔에서 플릿 프로비저닝 템플릿을 생성할 수도 있습니다.

   a. 탐색 창에서 여러 디바이스 연결 드롭다운을 선택합니다. 그런 다음 여러 디바이스 연결을 선택합니다.

   b. 프로비저닝 템플릿 생성을 선택합니다.

c. 설치 프로세스에 가장 적합한 프로비저닝 시나리오를 선택합니다. 그리고 다음을 선택합니다.

d. 템플릿 워크플로를 완료합니다.

2. 프로비저닝 클레임 인증서로 사용할 인증서 및 연결된 프라이빗 키를 생성합니다.

3. 이러한 인증서를에 등록 AWS IoT 하고 인증서 사용을 제한하는 IoT 정책을 연결합니다. 다음 IoT 정책의 예는 이 정책과 연결된 인증서의 사용을 프로비저닝 디바이스로 제한합니다.

```
{
    "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/templateName/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/templateName/provision/*"
            ]
        }
    ]
```

```
}
```

4. 디바이스를 프로비저닝할 때 계정의 사물 및 인증서와 같은 IoT 리소스를 생성하거나 업데이트할 수 있는 권한을 AWS IoT 서비스에 부여합니다. 이렇게 하려면 `AWSIoTThingsRegistration` 관리형 정책을 AWS IoT 서비스 보안 주체를 신뢰하는 IAM 역할(프로비저닝 역할이라고 함)에 연결합니다.
5. 프로비저닝 클레임 인증서가 안전하게 내장된 디바이스를 제조합니다.

이제 디바이스를 설치할 위치로 전달할 준비가 되었습니다.

> **Important**
>
> 프로비저닝 클레임 프라이빗 키는 디바이스에서는 물론 어디에서든 항상 보안 유지되어야 합니다. AWS IoT CloudWatch 지표 및 로그를 사용하여 오용 징후를 모니터링하는 것이 좋습니다. 오용이 감지되면 프로비저닝 클레임 인증서를 디바이스 프로비저닝에 사용할 수 없도록 비활성화합니다.

사용할 디바이스를 초기화하려면

1. 디바이스는 AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트를 사용하여 디바이스에 설치된 프로비저닝 클레임 인증서를 AWS IoT 사용하여에 연결하고 인증합니다.

   > **Note**
   >
   > 보안을 위해 `CreateCertificateFromCsr` 및 `CreateKeysAndCertificate`에 의해 반환된 `certificateOwnershipToken`은 1시간 후에 만료됩니다. `RegisterThing`은 `certificateOwnershipToken`이 만료되기 전에 호출되어야 합니다. `CreateCertificateFromCsr` 또는 `CreateKeysAndCertificate`에서 생성된 인증서가 활성화되지 않았으며 토큰이 만료될 때까지 정책 또는 사물에 연결되지 않은 경우 인증서가 삭제됩니다. 토큰이 만료되면 디바이스는 `CreateCertificateFromCsr` 또는 `CreateKeysAndCertificate`를 다시 호출하여 새 인증서를 생성합니다.

2. 디바이스는 이러한 옵션 중 하나를 사용하여 영구 인증서와 프라이빗 키를 가져옵니다. 디바이스는 이후의 모든 인증에 인증서와 키를 사용합니다 AWS IoT.

   a. 를 호출`CreateKeysAndCertificate`하여 인증 기관을 사용하여 새 AWS 인증서와 프라이빗 키를 생성합니다.

또는

b. CreateCertificateFromCsr을 호출하여 프라이빗 키를 안전하게 유지하는 인증서 서명 요청에서 인증서를 생성합니다.

3. 디바이스에서 RegisterThing을 호출하여 디바이스를 AWS IoT 에 등록하고 클라우드 리소스를 생성합니다.

   플릿 프로비저닝 서비스는 프로비저닝 템플릿을 사용하여 IoT 사물과 같은 클라우드 리소스를 정의하고 생성합니다. 템플릿에는 사물이 속한 그룹과 속성이 지정될 수 있습니다. 사물 그룹에 새 사물을 추가하려면 먼저 사물 그룹이 있어야 합니다.

4. 디바이스에 영구 인증서를 저장한 후 디바이스는 프로비저닝 클레임 인증서로 시작한 세션과의 연결을 끊고 영구 인증서를 사용하여 다시 연결해야 합니다.

이제 디바이스가와 정상적으로 통신할 준비가 되었습니다 AWS IoT.

신뢰할 수 있는 사용자에 의한 프로비저닝

대부분의 경우 최종 사용자 또는 설치 기술자와 같은 신뢰할 수 있는 사용자가 모바일 앱을 사용하여 배포된 위치에 디바이스를 구성할 때 디바이스가 AWS IoT 처음으로에 연결됩니다.

> **Important**
>
> 이 절차를 수행하려면 신뢰할 수 있는 사용자의 액세스 및 권한을 관리해야 합니다. 이를 수행하는 한 가지 방법은 신뢰할 수 있는 사용자를 인증하고 이 절차를 수행하는 데 필요한 AWS IoT 기능 및 API 작업에 대한 액세스 권한을 부여하는 신뢰할 수 있는 사용자에 대한 계정을 제공 및 유지 관리하는 것입니다.

디바이스를 전달하기 전에

1. CreateProvisioningTemplate을 호출하여 프로비저닝 템플릿을 생성하고 해당 *templateArn* 및 *templateName*을 반환합니다.
2. 신뢰할 수 있는 사용자가 프로비저닝 프로세스를 시작하는 데 사용하는 IAM 역할을 생성합니다. 프로비저닝 템플릿에서는 해당 사용자만 디바이스를 프로비저닝할 수 있습니다. 예제:

```
{
    "Effect": "Allow",
```

```
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:provisioningtemplate/templateName"
    ]
}
```

3. 디바이스를 프로비저닝할 때 계정의 사물 및 인증서와 같은 IoT 리소스를 생성하거나 업데이트할 수 있는 권한을 AWS IoT 서비스에 부여합니다. 이렇게 하려면 AWSIoTThingsRegistration 관리형 정책을 AWS IoT 서비스 보안 주체를 신뢰하는 IAM 역할(프로비저닝 역할이라고 함)에 연결합니다.
4. 인증할 수 있는 계정을 제공하고 디바이스를 등록하는 데 필요한 AWS API 작업과의 상호 작용을 승인하는 등 신뢰할 수 있는 사용자를 식별할 수 있는 수단을 제공합니다.

사용할 디바이스를 초기화하려면

1. 신뢰할 수 있는 사용자가 프로비저닝 모바일 앱 또는 웹 서비스에 로그인합니다.
2. 모바일 앱 또는 웹 애플리케이션은 IAM 역할을 사용하고 CreateProvisioningClaim을 호출하여 AWS IoT에서 임시 프로비저닝 클레임 인증서를 가져옵니다.

> **Note**
>
> 보안을 위해 CreateProvisioningClaim에서 반환한 임시 프로비저닝 클레임 인증서는 5분 후에 만료됩니다. 다음 단계에서는 임시 프로비저닝 클레임 인증서가 만료되기 전에 유효한 인증서를 반환해야 합니다. 임시 프로비저닝 클레임 인증서는 계정의 인증서 목록에 나타나지 않습니다.

3. 모바일 앱 또는 웹 애플리케이션은 Wi-Fi 자격 증명과 같은 필수 구성 정보와 함께 임시 프로비저닝 클레임 인증서를 디바이스에 제공합니다.
4. 디바이스는 임시 프로비저닝 클레임 인증서를 사용하여 AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트를 통해 AWS IoT 에 연결합니다.
5. 디바이스는 AWS IoT 임시 프로비저닝 클레임 인증서로에 연결한 후 5분 이내에 이러한 옵션 중 하나를 사용하여 영구 인증서와 프라이빗 키를 가져옵니다. 디바이스는 인증서와 이러한 옵션이 향후 모든 인증을 위해 반환하는 키를 사용합니다 AWS IoT.

   a. 를 호출CreateKeysAndCertificate하여 인증 기관을 사용하여 새 AWS 인증서와 프라이빗 키를 생성합니다.

또는

b. CreateCertificateFromCsr을 호출하여 프라이빗 키를 안전하게 유지하는 인증서 서명 요청에서 인증서를 생성합니다.

> **Note**
>
> 임시 프로비저닝 클레임 인증서 AWS IoT 로에 연결한 후 5분 이내에 유효한 인증서를 반환CreateKeysAndCertificate해야 CreateCertificateFromCsr 합니다.

6. 디바이스는를 호출RegisterThing하여에 디바이스를 등록 AWS IoT 하고 클라우드 리소스를 생성합니다.

   플릿 프로비저닝 서비스는 프로비저닝 템플릿을 사용하여 IoT 사물과 같은 클라우드 리소스를 정의하고 생성합니다. 템플릿에는 사물이 속한 그룹과 속성이 지정될 수 있습니다. 사물 그룹에 새 사물을 추가하려면 먼저 사물 그룹이 있어야 합니다.

7. 디바이스에 영구 인증서를 저장한 후 디바이스는 임시 프로비저닝 클레임 인증서로 시작한 세션과의 연결을 끊고 영구 인증서를 사용하여 다시 연결해야 합니다.

이제 디바이스가와 정상적으로 통신할 준비가 되었습니다 AWS IoT.

AWS CLI에서 사전 프로비저닝 후크 사용

다음 절차에서는 사전 프로비저닝 후크를 사용하여 프로비저닝 템플릿을 생성합니다. 여기에 사용된 Lambda 함수는 수정할 수 있는 예제입니다.

사전 프로비저닝 후크를 생성하여 프로비저닝 템플릿에 적용하려면

1. 정의된 입력 및 출력이 있는 Lambda 함수를 만듭니다. Lambda 함수는 고도로 사용자 정의 가능합니다. 사전 프로비저닝 후크를 생성하려면 `allowProvisioning` 및 `parameterOverrides`가 필요합니다. Lambda 함수 생성에 대한 자세한 내용은 AWS 명령줄 인터페이스와 AWS Lambda 함께 사용을 참조하세요.

   다음은 Lambda 함수 출력의 예제입니다.

```
{
  "allowProvisioning": True,
  "parameterOverrides": {
```

```
    "incomingKey0": "incomingValue0",
    "incomingKey1": "incomingValue1"
  }
}
```

2. AWS IoT 는 리소스 기반 정책을 사용하여 Lambda를 호출하므로 Lambda 함수를 호출할 수 있는 권한을 부여 AWS IoT 해야 합니다.

> **Important**
>
> 권한 조작을 방지하기 위해 Lambda 작업에 연결된 정책의 전역 조건 컨텍스트 키에 `source-arn` 또는 `source-account`가 포함되어야 합니다. 자세한 내용은 교차 서비스 혼동된 대리인 방지 단원을 참조하십시오.

다음은 add-permission을 사용하여 Lambda에 대한 권한을 IoT에 부여하는 예제입니다.

```
aws lambda add-permission \
    --function-name myLambdaFunction \
    --statement-id iot-permission \
    --action lambda:InvokeFunction \
    --principal iot.amazonaws.com
```

3. create-provisioning-template 또는 update-provisioning-template 명령을 사용하여 사전 프로비저닝 후크를 템플릿에 추가합니다.

다음 CLI 예제에서는 create-provisioning-template을 사용하여 사전 프로비저닝 후크가 있는 프로비저닝 템플릿을 생성합니다.

```
aws iot create-provisioning-template \
    --template-name myTemplate \
    --provisioning-role-arn arn:aws:iam:us-east-1:1234564789012:role/myRole \
    --template-body file://template.json \
    --pre-provisioning-hook file://hooks.json
```

이 명령의 출력은 다음과 같습니다.

```
{
    "templateArn": "arn:aws:iot:us-east-1:1234564789012:provisioningtemplate/
myTemplate",
    "defaultVersionId": 1,
```

```
    "templateName": myTemplate
}
```

또한 모든 파라미터를 명령줄 파라미터 값으로 입력하는 대신 파일에서 파라미터를 로드하여 시간을 절약할 수 있습니다. 자세한 내용은 파일에서 AWS CLI 파라미터 로드를 참조하세요. 다음은 확장된 JSON 형식의 `template` 파라미터를 보여줍니다.

```
{
    "Parameters" : {
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingTypeName" : {"Fn::Join":["",["ThingTypePrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["widgets", "WA"],
                "BillingGroup": "BillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
```

```
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:504350838278:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
    }
}
```

다음은 확장된 JSON 형식의 `pre-provisioning-hook` 파라미터를 보여줍니다.

```
{
     "targetArn" : "arn:aws:lambda:us-
east-1:765219403047:function:pre_provisioning_test",
     "payloadVersion" : "2020-04-01"
}
```

디바이스 인증서가 있는 디바이스 프로비저닝

AWS IoT 는 디바이스에 이미 디바이스 인증서(및 연결된 프라이빗 키)가 있는 경우 디바이스를 프로비저닝하는 세 가지 방법을 제공합니다.

- 프로비저닝 템플릿을 사용한 단일 사물 프로비저닝. 이 방식은 디바이스를 한 번에 하나씩 프로비저닝해야 하는 경우에 유용한 옵션니다.
- 디바이스가 처음 연결될 때 디바이스를 프로비저닝하는 템플릿을 사용한 Just-in-time 프로비저닝(JITP). AWS IoT이 방식은 다수의 디바이스를 등록해야 하지만 대량 프로비저닝 목록으로 어셈블할 수 있는 디바이스 정보가 없는 경우에 유용한 옵션입니다.
- 대량 등록입니다. 이 옵션을 사용하면 S3 버킷의 파일에 저장된 단일 사물 프로비저닝 템플릿 값의 목록을 지정할 수 있습니다. 이 방식은 알려진 디바이스가 다수 있고 원하는 특성을 목록으로 어셈블할 수 있는 경우 유용합니다.

주제

- 단일 사물 프로비저닝
- JIT 프로비저닝
- 대량 등록

단일 사물 프로비저닝

사물을 프로비저닝할 때는 RegisterThing API 또는 `register-thing` CLI 명령을 사용합니다. `register-thing` CLI 명령에서는 다음 인수가 필요합니다.

--template-body

프로비저닝 템플릿입니다.

--파라미터

프로비저닝 템플릿에서 사용되는 파라미터의 이름-값 페어 목록으로, JSON 형식을 따릅니다(예: `{"ThingName" : "MyProvisionedThing", "CSR" : "csr-text"}`).

프로비저닝 템플릿을(를) 참조하세요.

RegisterThing 또는 `register-thing`은 리소스 ARN과 생성된 인증서의 텍스트를 반환합니다.

```
{
    "certificatePem": "certificate-text",
    "resourceArns": {
    "PolicyLogicalName": "arn:aws:iot:us-
west-2:123456789012:policy/2A6577675B7CD1823E271C7AAD8184F44630FFD7",
```

```
    "certificate": "arn:aws:iot:us-west-2:123456789012:cert/
cd82bb924d4c6ccbb14986dcb4f40f30d892cc6b3ce7ad5008ed6542eea2b049",
    "thing": "arn:aws:iot:us-west-2:123456789012:thing/MyProvisionedThing"
    }
}
```

파라미터가 딕셔너리에서 생략되면 기본값이 사용됩니다. 기본값이 지정되지 않은 경우에는 파라미터가 값으로 치환되지 않습니다.

JIT 프로비저닝

디바이스가 JIT 프로비저닝(JITP)을 통해 디바이스가 AWS IoT연결을 처음 시도할 때 디바이스를 프로비저닝할 수 있습니다. 디바이스를 프로비저닝하려면 자동 등록을 활성화하고, 디바이스 인증서에 서명할 때 사용할 CA 인증서와 프로비저닝 템플릿을 연결해야 합니다. 프로비저닝 성공 및 오류는 Amazon CloudWatch에서 디바이스 프로비저닝 지표로 로그됩니다.

주제

- JITP 개요
- 프로비저닝 템플릿을 사용하여 CA 등록
- 프로비저닝 템플릿 이름을 사용하여 CA 등록

JITP 개요

디바이스가 등록된 CA 인증서 AWS IoT 로 서명된 인증서를 사용하여에 연결하려고 하면는 CA 인증서에서 템플릿을 AWS IoT 로드하고 이를 사용하여 RegisterThing을 호출합니다. JITP 워크플로는 상태 값이 `PENDING_ACTIVATION`인 인증서부터 먼저 등록합니다. 디바이스 프로비저닝 흐름이 완료되면 인증서 상태가 `ACTIVE`로 바뀝니다.

AWS IoT 는 프로비저닝 템플릿에서 선언하고 참조할 수 있는 다음 파라미터를 정의합니다.

- `AWS::IoT::Certificate::Country`
- `AWS::IoT::Certificate::Organization`
- `AWS::IoT::Certificate::OrganizationalUnit`
- `AWS::IoT::Certificate::DistinguishedNameQualifier`
- `AWS::IoT::Certificate::StateName`
- `AWS::IoT::Certificate::CommonName`
- `AWS::IoT::Certificate::SerialNumber`

- `AWS::IoT::Certificate::Id`

위와 같은 프로비저닝 템플릿 파라미터의 값들은 JITP가 프로비저닝할 디바이스의 인증서에서 제목 필드로부터 추출할 수 있는 값으로 제한됩니다. 인증서에는 템플릿 본문의 모든 파라미터에 대한 값이 포함되어야 합니다. `AWS::IoT::Certificate::Id` 파라미터는 인증서에 포함된 ID가 아니라 내부에서 생성된 ID를 참조합니다. AWS IoT 규칙 내의 `principal()` 함수를 사용하여이 ID의 값을 가져올 수 있습니다.

> **Note**
>
> 디바이스의 첫 번째 연결에서 전체 신뢰 체인을 전송할 필요 없이 JITP(AWS IoT Core just-in-time Provisioning) 기능을 사용하여 디바이스를 프로비저닝할 수 있습니다 AWS IoT Core. CA 인증서를 제시하는 것은 선택 사항이지만 디바이스가 AWS IoT Core에 연결할 때 서버 이름 표시(SNI) 확장을 전송해야 합니다.

템플릿 본문 예

다음 JSON 파일은 완전한 JITP 템플릿의 템플릿 본문 예입니다.

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
```

```
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
            },
            "ThingTypeName":"lightBulb-versionA",
            "ThingGroups":[
               "v1-lightbulbs",
               {
                  "Ref":"AWS::IoT::Certificate::Country"
               }
            ]
         },
         "OverrideSettings":{
            "AttributePayload":"MERGE",
            "ThingTypeName":"REPLACE",
            "ThingGroups":"DO_NOTHING"
         }
      },
      "certificate":{
         "Type":"AWS::IoT::Certificate",
         "Properties":{
            "CertificateId":{
               "Ref":"AWS::IoT::Certificate::Id"
            },
            "Status":"ACTIVE"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
            "PolicyDocument":"{ \"Version\": \"2012-10-17\",        \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

이 예제 템플릿은 인증서에서 추출되고 Resources 섹션에서 사용되는 AWS::IoT::Certificate::CommonName, AWS::IoT::Certificate::SerialNumber,

`AWS::IoT::Certificate::Country` 및 `AWS::IoT::Certificate::Id` 프로비저닝 파라미터의 값을 선언합니다. 그러면 JITP 워크플로우가 이 템플릿을 사용하여 다음 작업을 수행합니다.

- 인증서를 등록하고 상태를 PENDING_ACTIVE로 설정
- 사물 리소스 하나를 생성
- 정책 리소스 하나를 생성
- 정책을 인증서에 연결
- 인증서를 사물에 연결합니다.
- 인증서 상태를 ACTIVE로 업데이트

인증서에서 `templateBody`의 `Parameters` 섹션에 언급된 모든 속성이 없으면 디바이스 프로비저닝이 실패합니다. 예를 들어, `AWS::IoT::Certificate::Country`가 템플릿에 포함되어 있지만 인증서에는 `Country` 속성이 없으면 디바이스 프로비저닝이 실패합니다.

또한 CloudTrail을 사용하여 JITP 템플릿과 관련된 문제를 해결할 수도 있습니다. Amazon CloudWatch에 로그된 지표에 대한 자세한 내용은 디바이스 프로비저닝 지표 단원을 참조하세요. 프로비저닝 템플릿에 대한 자세한 내용은 프로비저닝 템플릿을 참조하세요.

> **Note**
>
> 프로비저닝 프로세스 중에 just-in-time 프로비저닝(JITP)은 다른 AWS IoT 컨트롤 플레인 API 작업을 호출합니다. 이러한 호출이 계정에 설정된 AWS IoT 조절 할당량을 초과하면 호출이 제한될 수 있습니다. 필요한 경우 제한 할당량을 높이려면 AWS 고객 지원에 문의하세요.

프로비저닝 템플릿을 사용하여 CA 등록

전체 프로비저닝 템플릿을 사용하여 CA를 등록하려면 다음 단계를 수행합니다.

1. 프로비저닝 템플릿과 역할 ARN 정보를 다음 예와 같이 JSON 파일로 저장합니다.

```
{
     "templateBody" : "{\r\n    \"Parameters\" : {\r\n
 \"AWS::IoT::Certificate::CommonName\": {\r\n            \"Type\": \"String\"\r
\n        },\r\n        \"AWS::IoT::Certificate::SerialNumber\": {\r\n
  \"Type\": \"String\"\r\n        },\r\n        \"AWS::IoT::Certificate::Country
\": {\r\n            \"Type\": \"String\"\r\n        },\r\n
 \"AWS::IoT::Certificate::Id\": {\r\n            \"Type\": \"String\"\r
```

```
\n        }\r\n    },\r\n    \"Resources\": {\r\n        \"thing\": {\r
\n            \"Type\": \"AWS::IoT::Thing\",\r\n            \"Properties
\": {\r\n                \"ThingName\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::CommonName\"\r\n                },\r\n
    \"AttributePayload\": {\r\n                    \"version\": \"v1\",\r\n
                \"serialNumber\": {\r\n                        \"Ref\":
 \"AWS::IoT::Certificate::SerialNumber\"\r\n                    }\r\n
    },\r\n                \"ThingTypeName\": \"lightBulb-versionA\",\r\n
       \"ThingGroups\": [\r\n                    \"v1-lightbulbs\",\r\n
          {\r\n                        \"Ref\": \"AWS::IoT::Certificate::Country
\"\r\n                    }\r\n                ]\r\n            },\r\n
  \"OverrideSettings\": {\r\n                \"AttributePayload\": \"MERGE\",\r\n
               \"ThingTypeName\": \"REPLACE\",\r\n                \"ThingGroups
\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"certificate\": {\r
\n            \"Type\": \"AWS::IoT::Certificate\",\r\n            \"Properties
\": {\r\n                \"CertificateId\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::Id\"\r\n                },\r\n                \"Status\":
 \"ACTIVE\"\r\n            },\r\n            \"OverrideSettings\": {\r\n
     \"Status\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"policy
\": {\r\n            \"Type\": \"AWS::IoT::Policy\",\r\n            \"Properties
\": {\r\n                \"PolicyDocument\": \"{ \\\"Version\\\": \\\"2012-10-17
     \\\", \\\"Statement\\\": [{ \\\"Effect\\\": \\\"Allow\\\", \\\"Action\\\":[\\
\"iot:Publish\\\"], \\\"Resource\\\": [\\\"arn:aws:iot:us-east-1:123456789012:topic
\/foo\/bar\\\"] }] }\"\r\n            }\r\n        }\r\n    }\r\n}",
     "roleArn" : "arn:aws:iam::123456789012:role/JITPRole"
}
```

이 예에서 templateBody 필드의 값은 이스케이프된 문자열로 지정된 JSON 객체여야 하며 이전 목록에 포함된 값만 사용할 수 있습니다. 다양한 도구를 사용하여 json.dumps(Python) 또는 JSON.stringify(노드)와 같은 필수 JSON 출력을 생성할 수 있습니다. roleARN 필드의 값은 AWSIoTThingsRegistration이 연결된 역할의 ARN이어야 합니다. 또한 템플릿이 예제의 인라인 PolicyDocument 대신 기존 PolicyName을 사용할 수 있습니다.

2. RegisterCACertificate API 작업 또는 register-ca-certificate CLI 명령을 사용하여 CA 인증서를 등록합니다. 이전 단계에서 저장한 프로비저닝 템플릿 및 역할 ARN 정보의 디렉터리를 지정합니다.

   다음은 AWS CLI를 사용하여 DEFAULT 모드로 CA 인증서를 등록하는 방법의 예제입니다.

   ```
   aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
   verification-cert file://your-verification-cert
   ```

```
                --set-as-active --allow-auto-registration --registration-config
 file://your-template
```

다음은 AWS CLI를 사용하여 SNI_ONLY 모드로 CA 인증서를 등록하는 방법의 예제입니다.

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --certificate-
mode SNI_ONLY
                 --set-as-active --allow-auto-registration --registration-config
 file://your-template
```

자세한 정보는 CA 인증서 등록을 참조하세요.

3. (선택 사항) UpdateCACertificate API 작업 또는 update-ca-certificate CLI 명령을 사용하여 CA 인증서 설정을 업데이트합니다.

   다음은 AWS CLI를 사용하여 CA 인증서를 업데이트하는 방법의 예입니다.

```
aws iot update-ca-certificate --certificate-id caCertificateId
                --new-auto-registration-status ENABLE --registration-config
 file://your-template
```

프로비저닝 템플릿 이름을 사용하여 CA 등록

프로비저닝 템플릿 이름을 사용하여 CA를 등록하려면 다음 단계를 수행합니다.

1. 프로비저닝 템플릿 본문을 JSON 파일로 저장합니다. 템플릿 본문 예에서 템플릿 본문 예를 찾을 수 있습니다.
2. 프로비저닝 템플릿을 생성하려면 CreateProvisioningTemplate API 또는 create-provisioning-template CLI 명령을 사용합니다.

```
aws iot create-provisioning-template --template-name your-template-name \
        --template-body file://your-template-body.json --type JITP \
        --provisioning-role-arn arn:aws:iam::123456789012:role/test
```

> **Note**
>
> JIT 프로비저닝(JITP)의 경우 프로비저닝 템플릿을 생성할 때 템플릿 유형을 `JITP`로 지정해야 합니다. 템플릿 유형에 대한 자세한 내용은 AWS API 참조의 CreateProvisioningTemplate을 참조하세요.

3. 템플릿 이름으로 CA를 등록하려면 RegisterCACertificate API 또는 `register-ca-certificate` CLI 명령을 사용합니다.

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
verification-cert file://your-verification-cert \
        --set-as-active --allow-auto-registration --registration-config
 templateName=your-template-name
```

대량 등록

`start-thing-registration-task` 명령을 사용하여 대량으로 사물을 등록할 수 있습니다. 이 명령에서는 프로비저닝 템플릿, S3 버킷 이름, 키 이름, 그리고 S3 버킷의 파일에 대한 액세스를 허용하는 역할 ARN이 필요합니다. S3 버킷의 파일은 템플릿의 파라미터를 대체하는 데 사용되는 값을 포함합니다. 파일은 줄 바꿈으로 구분된(newline-delimited) JSON 파일이어야 합니다. 각 줄은 단일 디바이스를 등록하기 위한 모든 파라미터 값을 포함합니다. 예:

```
{"ThingName": "foo", "SerialNumber": "123", "CSR": "csr1"}
{"ThingName": "bar", "SerialNumber": "456", "CSR": "csr2"}
```

다음과 같은 대량 등록 관련 API 작업이 유용할 수 있습니다.

- ListThingRegistrationTasks: 현재 대량 사물 프로비저닝 작업을 나열합니다.
- DescribeThingRegistrationTask: 특정 대량 사물 등록 작업에 대한 정보를 제공합니다.
- StopThingRegistrationTask: 대량 사물 등록 작업을 중지합니다.
- ListThingRegistrationTaskReports: 대량 사물 등록 작업에 대한 결과 및 실패를 확인하는 데 사용됩니다.

> **Note**
>
> - 대량 등록 작업은 계정당 한 번에 하나씩만 실행할 수 있습니다.
> - 대량 등록 작업은 다른 AWS IoT 컨트롤 플레인 API 작업을 호출합니다. 이러한 호출이 계정의 AWS IoT 조절 할당량을 초과해서 조절 오류가 발생할 수 있습니다. 필요한 경우 AWS 고객 지원 센터에 문의하여 AWS IoT 제한 할당량을 늘리세요.

프로비저닝 템플릿

프로비저닝 템플릿은 파라미터를 사용하여 디바이스가 상호 작용에 사용해야 하는 리소스를 설명하는 JSON 문서입니다 AWS IoT. 프로비저닝 템플릿에는 `Parameters`와 `Resources`, 2가지 섹션이 포함됩니다. 에는 두 가지 유형의 프로비저닝 템플릿이 있습니다 AWS IoT. 하나는 JIT 프로비저닝(JITP) 및 대량 등록에 사용되는 템플릿이고, 다른 하나는 폴릿 프로비저닝에 사용되는 템플릿입니다.

주제

- 파라미터 섹션
- 리소스 섹션
- 대량 등록에 대한 템플릿 예
- JIT 프로비저닝(JITP)의 템플릿 예
- 폴릿 프로비저닝

파라미터 섹션

`Parameters` 섹션은 `Resources` 섹션에서 사용할 파라미터를 선언합니다. 각 파라미터는 이름과 형식, 그리고 옵션으로 기본값을 선언합니다. 기본값은 템플릿과 함께 전달되는 딕셔너리에 파라미터 값이 포함되어 있지 않을 때 사용합니다. 템플릿 문서의 `Parameters` 섹션은 다음과 같습니다.

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
```

```
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

위의 템플릿 본문 코드 조각에서는 ThingName, SerialNumber, Location, CSR의 4가지 파라미터를 선언하고 있습니다. 이 파라미터들은 모두 String 형식을 갖습니다. Location 파라미터는 기본값으로 "WA"를 선언하고 있습니다.

리소스 섹션

템플릿 본문의 Resources 섹션에서는 디바이스가 AWS IoT사물, 인증서 및 하나 이상의 IoT 정책과 통신하는 데 필요한 리소스를 선언합니다. 각 리소스는 논리적 이름과 유형, 그리고 여러 가지 속성을 지정합니다.

논리적 이름은 템플릿의 다른 곳에서 리소스를 참조하는 데 사용됩니다.

유형은 선언하는 리소스 종류를 지정합니다. 유효한 유형은 다음과 같습니다.

- AWS::IoT::Thing
- AWS::IoT::Certificate
- AWS::IoT::Policy

지정하는 속성은 선언하는 리소스 유형에 따라 달라집니다.

사물 리소스

사물 리소스는 다음 속성을 사용하여 선언합니다.

- ThingName: 문자열.
- AttributePayload: 선택 사항입니다. 이름-값 페어 목록입니다.
- ThingTypeName: 선택 사항. 사물에 연결되어 있는 사물 유형을 나타내는 문자열
- ThingGroups: 선택 사항. 사물이 속하는 그룹의 목록
- BillingGroup: 선택 사항입니다. 연결된 결제 그룹 이름의 문자열
- PackageVersions: 선택 사항입니다. 연결된 패키지 및 버전 이름의 문자열

인증서 리소스

다음 방법 중 하나로 인증서를 지정할 수 있습니다.

- 인증서 서명 요청(CSR)
- 기존 디바이스 인증서의 ID (플릿 프로비저닝 템플릿에는 인증서 ID만 사용할 수 있습니다.)
- AWS IoT에 등록한 CA 인증서로 생성된 디바이스 인증서. 동일한 제목 필드로 등록한 CA 인증서가 2개 이상인 경우에는 디바이스 인증서 서명에 사용한 CA 인증서도 전달해야 합니다.

> **Note**
>
> 템플릿에서 인증서를 선언할 때는 이들 방법 중에서 한 가지만 사용하세요. 예를 들어 CSR을 사용하는 경우 인증서 ID 또는 디바이스 인증서는 지정할 수 없습니다. 자세한 내용은 X.509 클라이언트 인증서 단원을 참조하세요.

자세한 내용은 X.509 인증서 개요 단원을 참조하세요.

인증서 리소스는 다음 속성을 사용하여 선언합니다.

- `CertificateSigningRequest`: 문자열.
- `CertificateId`: 문자열.
- `CertificatePem`: 문자열.
- `CACertificatePem`: 문자열.
- `Status`: 선택 사항. `ACTIVE` 또는 `INACTIVE`일 수 있는 문자열입니다. 기본값은 ACTIVE입니다.
- `ThingPrincipalType`: 선택 사항입니다. 사물과 위탁자(인증서) 간의 관계 유형을 지정하는 문자열입니다.
  - `EXCLUSIVE_THING`: 독점적 관계를 설정합니다. 위탁자는 이 특정 사물에만 연결할 수 있으며 다른 사물에는 연결할 수 없습니다.
  - `NON_EXCLUSIVE_THING`: 지정한 위탁자를 여러 사물에 연결합니다. 여러 사물을 위탁자에 연결할 수 있습니다. 지정하지 않은 경우의 기본값입니다.

> **Note**
>
> 디바이스 인증서 없이 디바이스를 프로비저닝할 수도 있습니다. 자세한 내용은 플릿 프로비저닝을 사용하여 디바이스 인증서가 없는 디바이스 프로비저닝 단원을 참조하십시오.

예시:

- CSR을 사용하여 지정된 인증서:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateSigningRequest": {"Ref" : "CSR"},
            "Status" : "ACTIVE"
        }
    }
}
```

- 기존 인증서 ID를 사용하여 지정된 인증서:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateId": {"Ref" : "CertificateId"}
        }
    }
}
```

- 기존 인증서 .pem 및 CA 인증서 .pem을 사용하여 지정된 인증서:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CACertificatePem": {"Ref" : "CACertificatePem"},
            "CertificatePem": {"Ref" : "CertificatePem"}
        }
    }
}
```

- 하나의 위탁자에 하나의 사물만 독점적으로 연결합니다.

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
```

```
            "ThingPrincipalType" : "EXCLUSIVE_THING"
        }
    }
}
```

정책 리소스

정책 리소스는 다음 속성 중 한 가지를 사용하여 선언합니다.

- PolicyName: 선택 사항. 문자열. 기본 값은 정책 문서의 해시입니다. PolicyName은 AWS IoT 정책만 참조할 수 있고 IAM 정책은 참조할 수 없습니다. PolicyName 속성에 기존 AWS IoT 정책을 사용하는 경우 정책의 이름을 입력합니다. PolicyDocument 속성은 포함하지 마세요.
- PolicyDocument: 선택 사항. 이스케이프된 문자열로 지정된 JSON 객체. PolicyDocument를 입력하지 않는 경우에는 정책이 이미 생성되어 있어야 합니다.

> **Note**
>
> Policy 섹션이 존재하는 경우에는 PolicyName 또는 PolicyDocument를 지정해야 합니다(두 속성을 모두 지정할 수는 없습니다).

설정 재정의

템플릿이 기존 리소스를 지정하는 경우에는 OverrideSettings 섹션에서 실행할 작업을 지정할 수 있습니다.

`DO_NOTHING`

리소스를 그대로 유지합니다.

`REPLACE`

리소스를 템플릿에서 지정하는 리소스로 변경합니다.

`FAIL`

ResourceConflictsException을 사용해 리소스를 중단합니다.

MERGE

thing의 ThingGroups 및 AttributePayload 속성에만 유효합니다. 사물의 그룹 멤버십이나 기존 속성과 템플릿에 지정된 속성을 병합합니다.

특정 사물을 리소스로 선언할 때 다음 속성에 OverrideSettings를 지정할 수 있습니다.

- ATTRIBUTE_PAYLOAD
- THING_TYPE_NAME
- THING_GROUPS

인증서 리소스를 선언할 경우 Status 속성에 OverrideSettings를 지정할 수 있습니다.

정책 리소스에는 OverrideSettings를 사용할 수 없습니다.

리소스 예제

다음은 사물, 인증서 및 정책을 선언하는 템플릿 조각입니다.

```
{
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
```

```
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
 \"Statement\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

위 예제에서 사물은 다음과 같이 선언되어 있습니다.

- 논리적 이름: `"thing"`
- 유형: `AWS::IoT::Thing`
- 여러 가지 사물 속성

  사물 속성 중에는 사물 이름, 속성 집합, 사물 유형 이름(선택적), 사물이 속하는 사물 그룹 목록(선택적)이 있습니다.

파라미터는 {"Ref":"*parameter-name*"}에서 참조할 수 있습니다. 템플릿을 평가할 때는 파라미터가 템플릿과 함께 전달되는 딕셔터리의 파라미터 값으로 치환됩니다.

위 예제에서 인증서는 다음과 같이 선언되어 있습니다.

- 논리적 이름: `"certificate"`
- 유형: `AWS::IoT::Certificate`
- 속성 집합

  속성 중에는 인증서 CSR과 `ACTIVE` 상태 설정이 있습니다. CSR 텍스트는 템플릿과 함께 전달되는 딕셔너리에서 파라미터로 전달됩니다.

위 예제에서 정책은 다음과 같이 선언되어 있습니다.

- 논리적 이름: `"policy"`
- 유형: `AWS::IoT::Policy`

- 기존 정책의 이름 또는 정책 문서

대량 등록에 대한 템플릿 예

다음 JSON 파일은 CSR을 사용하여 인증서를 지정하는 완전한 프로비저닝 템플릿의 예제입니다.

(PolicyDocument 필드 값은 이스케이프된 문자열로 지정된 JSON 객체여야 합니다.)

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
```

```
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
 \"Statement\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

JIT 프로비저닝(JITP)의 템플릿 예

다음 JSON 파일은 인증서 ID를 사용하여 기존 인증서를 지정하는 완전한 프로비저닝 템플릿의 예제입니다.

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
```

```
            },
            "ThingTypeName":"lightBulb-versionA",
            "ThingGroups":[
               "v1-lightbulbs",
               {
                  "Ref":"AWS::IoT::Certificate::Country"
               }
            ]
         },
         "OverrideSettings":{
            "AttributePayload":"MERGE",
            "ThingTypeName":"REPLACE",
            "ThingGroups":"DO_NOTHING"
         }
      },
      "certificate":{
         "Type":"AWS::IoT::Certificate",
         "Properties":{
            "CertificateId":{
               "Ref":"AWS::IoT::Certificate::Id"
            },
            "Status":"ACTIVE",
            "ThingPrincipalType" : "EXCLUSIVE_THING"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
            "PolicyDocument":"{ \"Version\": \"2012-10-17\",        \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

> **Important**
>
> JIT 프로비저닝에 사용되는 템플릿에서 `CertificateId`를 사용해야 합니다.

프로비저닝 템플릿 유형에 대한 자세한 내용은 AWS API 참조의 섹션을 참조[CreateProvisioningTemplate](#)하세요.

이 템플릿을 사용하여 JIT 프로비저닝을 수행하는 방법에 대한 자세한 내용은 [JIT 프로비저닝](#)을 참조하세요.

플릿 프로비저닝

플릿 프로비저닝 템플릿은에서 클라우드 및 디바이스 구성을 설정하는 AWS IoT 데 사용됩니다. 이러한 템플릿은 JITP 및 대량 등록 템플릿과 동일한 파라미터 및 리소스를 사용합니다. 자세한 내용은 [프로비저닝 템플릿](#) 단원을 참조하세요. 플릿 프로비저닝 템플릿은 `Mapping` 섹션 및 `DeviceConfiguration` 섹션을 포함할 수 있습니다. 플릿 프로비저닝 템플릿 내에서 내장 함수를 사용하여 디바이스별 구성을 생성할 수 있습니다. 플릿 프로비저닝 템플릿은 이름이 지정된 리소스이며 ARN으로 식별됩니다(예: `arn:aws:iot:us-west-2:1234568788:provisioningtemplate/`*`templateName`*).

매핑

`Mappings` 섹션(선택 사항)은 키를 해당하는 명명된 값 세트와 맞춥니다. 예를 들어 리전을 기반으로 AWS 값을 설정하려면 AWS 리전 이름을 키로 사용하고 각 특정 리전에 지정하려는 값을 포함하는 매핑을 생성할 수 있습니다. `Fn::FindInMap` 내장 함수를 사용하여 맵에서 값을 불러올 수 있습니다.

`Mappings` 섹션에는 파라미터, 가상 파라미터 또는 호출 내장 함수를 포함할 수 없습니다.

디바이스 구성

디바이스 구성 섹션은 프로비저닝할 때 디바이스에 전송할 임의의 데이터를 포함합니다. 예제:

```
{
    "DeviceConfiguration": {
        "Foo":"Bar"
    }
}
```

JavaScript Object Notation(JSON) 페이로드 형식을 사용하여 디바이스에 메시지를 보내는 경우는이 데이터를 JSON으로 AWS IoT Core 포맷합니다. CBOR(간결한 이진 객체 표현) 페이로드 형식을 사용하는 경우 AWS IoT Core 는 이 데이터를 CBOR로 포맷합니다. `DeviceConfiguration` 섹션은 중첩 JSON 객체를 지원하지 않습니다.

내장 함수

내장 함수는 Mappings 섹션을 제외한 프로비저닝 템플릿의 모든 섹션에서 사용됩니다.

Fn::Join

지정된 구분 기호로 구분된 값 집합을 단일 값에 추가합니다. 구분 기호가 빈 문자열이면 값은 구분 기호 없이 연결됩니다.

> **Important**
>
> the section called “정책 리소스”에는 Fn::Join이 지원되지 않습니다.

Fn::Select

인덱스별로 객체 목록에서 단일 객체를 반환합니다.

> **Important**
>
> 인덱스가 어레이 범위를 벗어나는 경우 Fn::Select는 null 값을 확인하지 않습니다. 두 조건 모두 프로비저닝 오류가 발생하므로 유효한 인덱스 값을 선택하고 목록에 null이 아닌 값이 포함되어 있는지 확인해야 합니다.

Fn::FindInMap

Mappings 섹션에서 선언된 2수준 맵의 키에 해당하는 값을 반환합니다.

Fn::Split

문자열 목록에서 요소를 선택할 수 있도록 문자열을 문자열 값 목록으로 분할합니다. 문자열이 분할되는 위치를 결정하는 구분 기호(예: 쉼표)를 지정합니다. 문자열을 분할한 후 Fn::Select를 사용하여 요소를 선택합니다.

예를 들어, 쉼표로 구분된 서브넷 ID 문자열을 스택 템플릿으로 가져올 경우 각 쉼표로 문자열을 분할할 수 있습니다. 서브넷 ID 목록에서 Fn::Select를 사용하여 리소스의 서브넷 ID를 지정합니다.

Fn::Sub

입력 문자열의 변수를 사용자가 지정한 값으로 바꿉니다. 이 함수를 사용하여 스택을 생성하거나 업데이트해야만 사용할 수 있는 값이 포함된 명령 또는 출력을 작성할 수 있습니다.

플릿 프로비저닝 템플릿 예

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber": {
            "Type": "String"
        },
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Ref" : "ThingName"},
                "ThingTypeName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["v1-lightbulbs", "WA"],
                "BillingGroup": "LightBulbBillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
```

```
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
        }
}
```

> **Note**
>
> 기존 프로비저닝 템플릿을 업데이트하여 사전 프로비저닝 후크를 추가할 수 있습니다.

사전 프로비저닝 후크

AWS 에서는 프로비저닝 템플릿을 생성할 때 사전 프로비저닝 후크 함수를 사용하여 계정 온보딩 대상 및 디바이스 수를 더 잘 제어할 것을 권장합니다. 사전 프로비저닝 후크는 디바이스 프로비저닝을 허용하기 전에 디바이스에서 전달된 파라미터의 유효성을 검사하는 Lambda 함수입니다. 이 Lambda 함수는 디바이스가 the section called “RegisterThing”을 통해 요청을 전송할 때마다 호출되므로 디바이스를 프로비저닝하기 전에 계정에 있어야 합니다.

> **Important**
>
> 권한 조작을 방지하기 위해 Lambda 작업에 연결된 정책의 전역 조건 컨텍스트 키에 `source-arn` 또는 `source-account`가 포함되어야 합니다. 자세한 내용은 교차 서비스 혼동된 대리인 방지 단원을 참조하십시오.

디바이스를 프로비저닝하려면 Lambda 함수가 입력 객체를 수락하고 이 섹션에 설명된 출력 객체를 반환해야 합니다. Lambda 함수가 `"allowProvisioning": True`의 객체를 반환하는 경우에만 프로비저닝이 진행됩니다.

사전 프로비저닝 후크 입력

AWS IoT 는 디바이스가에 등록할 때이 객체를 Lambda 함수로 전송합니다 AWS IoT.

```
{
    "claimCertificateId" : "string",
    "certificateId" : "string",
    "certificatePem" : "string",
    "templateArn" : "arn:aws:iot:us-east-1:1234567890:provisioningtemplate/MyTemplate",
    "clientId" : "221a6d10-9c7f-42f1-9153-e52e6fc869c1",
    "parameters" : {
        "string" : "string",
        ...
    }
}
```

Lambda 함수에 전달된 `parameters` 객체는 the section called “RegisterThing” 요청 페이로드에서 전달된 `parameters` 인수의 속성을 포함합니다.

사전 프로비저닝 후크 반환 값

Lambda 함수는 프로비저닝 요청을 승인했는지 여부를 나타내는 응답과, 재정의할 속성의 값을 반환해야 합니다.

다음은 사전 프로비저닝 함수에서 성공한 응답의 예입니다.

```
{
    "allowProvisioning": true,
    "parameterOverrides" : {
        "Key": "newCustomValue",
```

```
        ...
    }
}
```

"parameterOverrides" 값이 the section called “RegisterThing” 요청 페이로드의 "parameters" 파라미터에 추가됩니다.

> **Note**
>
> - Lambda 함수가 실패하면 프로비저닝 요청은 ACCESS_DENIED와 함께 실패하며 CloudWatch Logs에 오류가 기록됩니다.
> - Lambda 함수가 응답에서 "allowProvisioning": "true"를 반환하지 않는 경우 프로비저닝 요청은 ACCESS_DENIED와 함께 실패합니다.
> - Lambda 함수는 실행을 완료하고 5초 이내에 반환해야 합니다. 그렇지 않으면 프로비저닝 요청이 실패합니다.

사전 프로비저닝 후크 Lambda 예제

Python

Python에서 사전 프로비저닝 후크 Lambda 예제.

```
import json

def pre_provisioning_hook(event, context):
    print(event)

    return {
        'allowProvisioning': True,
        'parameterOverrides': {
            'DeviceLocation': 'Seattle'
        }
    }
```

Java

Java에서 사전 프로비저닝 후크 Lambda 예제.

핸들러 클래스:

```
package example;

import java.util.Map;
import java.util.HashMap;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class PreProvisioningHook implements
 RequestHandler<PreProvisioningHookRequest, PreProvisioningHookResponse> {

    public PreProvisioningHookResponse handleRequest(PreProvisioningHookRequest
 object, Context context) {
        Map<String, String> parameterOverrides = new HashMap<String, String>();
        parameterOverrides.put("DeviceLocation", "Seattle");

        PreProvisioningHookResponse response = PreProvisioningHookResponse.builder()
                .allowProvisioning(true)
                .parameterOverrides(parameterOverrides)
                .build();

        return response;
    }

}
```

요청 클래스:

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookRequest {
    private String claimCertificateId;
    private String certificateId;
    private String certificatePem;
```

```
    private String templateArn;
    private String clientId;
    private Map<String, String> parameters;
}
```

응답 클래스:

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;


@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookResponse {
    private boolean allowProvisioning;
    private Map<String, String> parameterOverrides;
}
```

JavaScript

JavaScript에서 사전 프로비저닝 후크 Lambda 예시.

```
exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    var reply = {
        allowProvisioning: true,
        parameterOverrides: {
            DeviceLocation: 'Seattle'
        }
     };
     callback(null, reply);
}
```

인증서 공급자를 사용한 자체 관리형 AWS IoT Core 인증서 서명

AWS IoT Core 인증서 공급자를 생성하여 AWS IoT 플릿 프로비저닝에서 인증서 서명 요청(CSRs)에 서명할 수 있습니다. 인증서 공급자는 Lambda 함수와 플릿 프로비저닝을 위한 `CreateCertificateFromCsr` MQTT API를 참조합니다. Lambda 함수는 CSR을 수락하고 서명된 클라이언트 인증서를 반환합니다.

에 인증서 공급자가 없는 경우 AWS 계정CreateCertificateFromCsr MQTT API가 플릿 프로비저닝에서 호출되어 CSR에서 인증서를 생성합니다. 인증서 공급자를 생성하면 CreateCertificateFromCsr MQTT API의 동작이 변경되고 이 MQTT API에 대한 모든 직접 호출이 인증서 공급자를 간접적으로 호출하여 인증서를 발급합니다.

AWS IoT Core 인증서 공급자를 사용하면 , AWS Private CA기타 공개적으로 신뢰할 수 있는 CAs 또는 자체 퍼블릭 키 인프라(PKI)와 같은 프라이빗 인증 기관(CAs 활용하여 CSR에 서명하는 솔루션을 구현할 수 있습니다. 또한 인증서 공급자를 사용하여 유효 기간, 서명 알고리즘, 발급자 및 확장과 같은 클라이언트 인증서의 필드를 사용자 지정할 수 있습니다.

> **Important**
>
> AWS 계정마다 하나의 인증서 공급자만 생성할 수 있습니다. 서명 동작 변경은 AWS 계정에서 인증서 공급자를 삭제할 때까지 CreateCertificateFromCsr MQTT API를 직접적으로 호출하는 전체 플릿에 적용됩니다.

이 주제에서 수행할 작업

- 플릿 프로비저닝에서 자체 관리형 인증서 서명의 작동 방식
- 인증서 공급자 Lambda 함수 입력
- 인증서 공급자 Lambda 함수 반환 값
- Lambda 함수 예제
- 플릿 프로비저닝을 위한 자체 관리형 인증서 서명
- AWS CLI 인증서 공급자에 대한 명령

플릿 프로비저닝에서 자체 관리형 인증서 서명의 작동 방식

주요 개념

다음 개념은 AWS IoT 플릿 프로비저닝에서 자체 관리형 인증서 서명이 작동하는 방식을 이해하는 데 도움이 되는 세부 정보를 제공합니다. 자세한 내용은 플릿 프로비저닝을 사용하여 디바이스 인증서가 없는 디바이스 프로비저닝을 참조하세요.

AWS IoT 플릿 프로비저닝

AWS IoT 플릿 프로비저닝(플릿 프로비저닝의 약자)을 사용하면는 디바이스 인증서를 AWS IoT Core 생성하고 디바이스가 처음에 연결할 때 디바이스 AWS IoT Core 에 안전하게 전송합니다. 플릿 프로비저닝을 사용하여 디바이스 인증서가 없는 디바이스를 AWS IoT Core에 연결할 수 있습니다.

인증서 서명 요청(CSR)

플릿 프로비저닝 프로세스에서 디바이스는 플릿 프로비저닝 MQTT API를 통해 AWS IoT Core 에 요청합니다. 이 요청에는 클라이언트 인증서를 생성하기 위해 서명할 인증서 서명 요청(CSR)이 포함됩니다.

AWS 플릿 프로비저닝에서 관리형 인증서 서명

AWS 관리형는 플릿 프로비저닝에서 인증서 서명을 위한 기본 설정입니다. AWS 관리형 인증서 서명을 통해 AWS IoT Core 는 자체 CACSRs에 서명합니다. CAs

플릿 프로비저닝에서 서명하는 자체 관리형 인증서

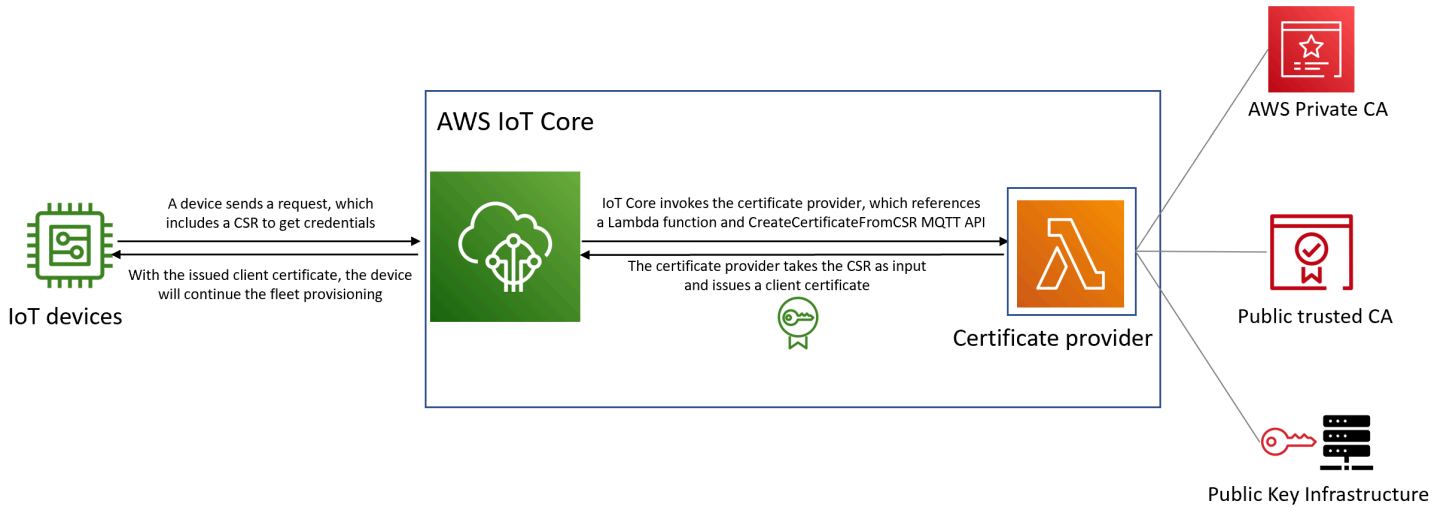
자체 관리형은 플릿 프로비저닝에서 인증서 서명을 위한 또 다른 옵션입니다. 자체 관리형 인증서 서명을 사용하면 CSR에 서명할 AWS IoT Core 인증서 공급자를 생성합니다. 자체 관리형 인증서 서명을 사용하여 AWS 프라이빗 CA, 기타 공개적으로 신뢰할 수 있는 CA 또는 자체 퍼블릭 키 인프라(PKI)에서 생성된 CA를 사용하여 CSR에 서명할 수 있습니다.

AWS IoT Core 인증서 공급자

AWS IoT Core 인증서 공급자(인증서 공급자의 약자)는 플릿 프로비저닝에서 자체 관리형 인증서 서명에 사용되는 고객 관리형 리소스입니다.

다이어그램

다음 다이어그램은 AWS IoT 플릿 프로비저닝에서 자체 인증서 서명이 작동하는 방식을 간략히 보여줍니다.



- 새 IoT 디바이스가 제조되거나 플릿에 도입되면 자체 인증을 위해 클라이언트 인증서가 필요합니다 AWS IoT Core.
- 플릿 프로비저닝 프로세스의 일환으로 디바이스는 플릿 프로비저닝 MQTT APIs를 통해에 AWS IoT Core 클라이언트 인증서를 요청합니다. 이 요청에는 인증서 서명 요청(CSR)이 포함됩니다.
- AWS IoT Core 는 인증서 공급자를 호출하고 CSR을 공급자에게 입력으로 전달합니다.
- 인증서 공급자는 CSR을 입력으로 받아 클라이언트 인증서를 발급합니다.

  AWS 관리형 인증서 서명의 경우는 자체 CA를 사용하여 CSR에 AWS IoT Core 서명하고 클라이언트 인증서를 발급합니다.
- 발급된 클라이언트 인증서를 사용하면 디바이스가 플릿 프로비저닝을 계속하고 AWS IoT Core와 보안 연결을 설정합니다.

인증서 공급자 Lambda 함수 입력

AWS IoT Core 는 디바이스가 Lambda 함수에 등록할 때 다음 객체를 Lambda 함수로 전송합니다. `certificateSigningRequest`의 값은 `CreateCertificateFromCsr` 요청에 제공된 프라이버시 강화 메일(PEM) 형식의 CSR입니다. `principalId`는 `CreateCertificateFromCsr` 요청 AWS IoT Core 시에 연결하는 데 사용되는 보안 주체의 ID입니다. `clientId`는 MQTT 연결에 설정된 클라이언트 ID입니다.

```
{
 "certificateSigningRequest": "string",
 "principalId": "string",
 "clientId": "string"
```

```
}
```

인증서 공급자 Lambda 함수 반환 값

Lambda 함수는 `certificatePem` 값이 포함된 응답을 반환해야 합니다. 다음은 성공적인 응답의 예입니다. AWS IoT Core 는 반환 값(`certificatePem`)을 사용하여 인증서를 생성합니다.

```
{
 "certificatePem": "string"
}
```

등록에 성공하면 `CreateCertificateFromCsr`는 `CreateCertificateFromCsr` 응답에 동일한 `certificatePem`을 반환합니다. 자세한 내용은 CreateCertificateFromCsr 페이로드 예제를 참조하세요.

Lambda 함수 예제

인증서 공급자를 생성하기 전에 Lambda 함수를 생성하여 CSR에 서명해야 합니다. 다음은 Python의 Lambda 함수에 대한 예제입니다. 이 함수는 프라이빗 CA와 `SHA256WITHRSA` 서명 알고리즘을 사용하여 입력 CSR에 서명하도록 AWS Private CA 를 직접 호출합니다. 반환된 클라이언트 인증서는 1년 동안 유효합니다. AWS Private CA 및 프라이빗 CA 생성 방법에 대한 자세한 내용은 AWS 프라이빗 CA란 무엇입니까? 및 프라이빗 CA 생성을 참조하세요.

```
import os
import time
import uuid
import boto3

def lambda_handler(event, context):
    ca_arn = os.environ['CA_ARN']
    csr = (event['certificateSigningRequest']).encode('utf-8')

    acmpca = boto3.client('acm-pca')
    cert_arn = acmpca.issue_certificate(
        CertificateAuthorityArn=ca_arn,
        Csr=csr,
        Validity={"Type": "DAYS", "Value": 365},
        SigningAlgorithm='SHA256WITHRSA',
        IdempotencyToken=str(uuid.uuid4())
    )['CertificateArn']
```

```
    # Wait for certificate to be issued
    time.sleep(1)
    cert_pem = acmpca.get_certificate(
        CertificateAuthorityArn=ca_arn,
        CertificateArn=cert_arn
    )['Certificate']

    return {
        'certificatePem': cert_pem
    }
```

Important

- Lambda 함수에서 반환하는 인증서는 인증서 서명 요청(CSR)과 동일한 주제 이름과 퍼블릭 키를 가져야 합니다.
- Lambda 함수는 5초 후에 실행을 완료해야 합니다.
- Lambda 함수는 인증서 공급자 리소스와 동일한 AWS 계정 및 리전에 있어야 합니다.
- AWS IoT 서비스 보안 주체에게 Lambda 함수에 대한 호출 권한을 부여해야 합니다. 혼동된 대리자 문제를 방지하려면 호출 권한에 대해 `sourceArn` 및 `sourceAccount`를 설정하는 것이 좋습니다. 자세한 설명은 교차 서비스 혼동된 대리자 예방을 참조하세요.

Lambda에 대한 다음 리소스 기반 정책 예제는 AWS IoT 에 Lambda 함수를 간접 호출할 수 있는 권한을 부여합니다.

```
{
 "Version":"2012-10-17",
 "Id": "InvokePermission",
 "Statement": [
  {
   "Sid": "LambdaAllowIotProvider",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "lambda:InvokeFunction",
   "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
   "Condition": {
```

```
    "StringEquals": {
     "AWS:SourceAccount": "123456789012"
    },
    "ArnLike": {
    "AWS:SourceArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider/my-
certificate-provider"
    }
   }
  }
 ]
}
```

플릿 프로비저닝을 위한 자체 관리형 인증서 서명

AWS CLI 또는 AWS Management Console을 사용하여 플릿 프로비저닝에 자체 관리형 인증서 서명을 선택할 수 있습니다.

AWS CLI

자체 관리형 인증서 서명을 선택하려면 인증서 공급자를 AWS IoT Core 생성하여 플릿 프로비저닝에서 CSRs에 서명해야 합니다. AWS IoT Core 는 인증서 공급자를 호출하여 CSR을 입력으로 받아 클라이언트 인증서를 반환합니다. 인증서 공급자를 생성하려면 `CreateCertificateProvider` API 작업 또는 `create-certificate-provider` CLI 명령을 사용합니다.

> **Note**
>
> 인증서 공급자를 생성하면 플릿 프로비저닝을 위한 `CreateCertificateFromCsr` API 동작이 변경되어 `CreateCertificateFromCsr`에 대한 모든 직접 호출이 인증서 공급자를 간접 호출하여 인증서를 생성합니다. 인증서 공급자가 생성된 후 이 동작이 변경되는 데 몇 분 정도 걸릴 수 있습니다.

```
aws iot create-certificate-provider \
               --certificateProviderName my-certificate-provider \
               --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-1 \
               --accountDefaultForOperations CreateCertificateFromCsr
```

다음은 이 명령의 예시 출력을 보여줍니다.

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

자세한 내용은 AWS IoT API Reference의 `CreateCertificateProvider` 섹션을 참조하세요.

AWS Management Console

를 사용하여 자체 관리형 인증서 서명을 선택하려면 다음 단계를 AWS Management Console따릅니다.

1. AWS IoT 콘솔로 이동합니다.
2. 왼쪽 탐색의 보안 아래에서 인증서 서명을 선택합니다.
3. 인증서 서명 페이지의 인증서 서명 세부 정보에서 인증서 서명 방법 편집을 선택합니다.
4. 인증서 서명 방법 편집 페이지의 인증서 서명 방법에서 자체 관리형을 선택합니다.
5. 자체 관리형 설정 섹션에서 인증서 공급자의 이름을 입력한 다음 Lambda 함수를 생성하거나 선택합니다.
6. 인증서 서명 업데이트를 선택합니다.

AWS CLI 인증서 공급자에 대한 명령

인증서 공급자 생성

인증서 공급자를 생성하려면 `CreateCertificateProvider` API 작업 또는 `create-certificate-provider` CLI 명령을 사용합니다.

> **Note**
>
> 인증서 공급자를 생성하면 플릿 프로비저닝을 위한 `CreateCertificateFromCsr` API 동작이 변경되어 `CreateCertificateFromCsr`에 대한 모든 직접 호출이 인증서 공급자를 간접 호출하여 인증서를 생성합니다. 인증서 공급자가 생성된 후 이 동작이 변경되는 데 몇 분 정도 걸릴 수 있습니다.

```
aws iot create-certificate-provider \
```

```
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

다음은 이 명령의 예시 출력을 보여줍니다.

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

자세한 내용은 AWS IoT API Reference의 CreateCertificateProvider 섹션을 참조하세요.

인증서 공급자 업데이트

인증서 공급자를 업데이트하려면 UpdateCertificateProvider API 작업 또는 update-certificate-provider CLI 명령을 사용합니다.

```
aws iot update-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-2 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

다음은 이 명령의 예시 출력을 보여줍니다.

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

자세한 내용을 알아보려면 AWS IoT API Reference의 UpdateCertificateProvider 섹션을 참조하세요.

인증서 공급자 설명

인증서 공급자를 설명하려면 DescribeCertificateProvider API 작업 또는 describe-certificate-provider CLI 명령을 사용합니다.

```
aws iot describe-certificate-provider --certificateProviderName my-certificate-provider
```

다음은 이 명령의 예시 출력을 보여줍니다.

```
{
 "certificateProviderName": "my-certificate-provider",
 "lambdaFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
 "accountDefaultForOperations": [
  "CreateCertificateFromCsr"
 ],
 "creationDate": "2022-11-03T00:15",
 "lastModifiedDate": "2022-11-18T00:15"
}
```

자세한 내용은 AWS IoT API Reference의 DescribeCertificateProvider 섹션을 참조하세요.

인증서 공급자 삭제

인증서 공급자를 삭제하려면 DeleteCertificateProvider API 작업 또는 delete-certificate-provider CLI 명령을 사용합니다. 인증서 공급자 리소스를 삭제하면의 동작CreateCertificateFromCsr이 재개 AWS IoT 되고 CSR AWS IoT 에서가 서명한 인증서가 생성됩니다.

```
aws iot delete-certificate-provider --certificateProviderName my-certificate-provider
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT API Reference의 DeleteCertificateProvider 섹션을 참조하세요.

인증서 공급자 나열

내의 인증서 공급자를 나열하려면 ListCertificateProviders API 작업 또는 list-certificate-providers CLI 명령을 AWS 계정사용합니다.

```
aws iot list-certificate-providers
```

다음은 이 명령의 예시 출력을 보여줍니다.

```
{
 "certificateProviders": [
  {
```

```
   "certificateProviderName": "my-certificate-provider",
   "certificateProviderArn": "arn:aws:iot:us-
east-1:123456789012:certificateprovider:my-certificate-provider"
  }
 ]
}
```

자세한 내용은 AWS IoT API Reference의 `ListCertificateProvider` 섹션을 참조하세요.

디바이스를 설치하는 사용자를 위한 IAM 정책 및 역할 생성

> **Note**
>
> 이 절차는 AWS IoT 콘솔에서 지시한 경우에만 사용해야 합니다.
> 콘솔에서 이 페이지로 이동하려면 새 프로비저닝 템플릿 생성(create a new provisioning template)을 엽니다.

AWS IoT 콘솔에서이 작업을 수행할 수 없는 이유는 무엇입니까?

가장 안전한 환경을 위해 IAM 작업은 IAM 콘솔에서 수행됩니다. 이 섹션의 절차는 프로비저닝 템플릿 사용에 필요한 IAM 역할과 정책을 생성하는 단계를 안내합니다.

디바이스를 설치할 사용자에 대한 IAM 정책 생성

이 절차에서는 사용자가 프로비저닝 템플릿을 사용하여 디바이스를 설치할 수 있도록 권한을 부여하는 IAM 정책을 생성하는 방법을 설명합니다.

이 절차를 수행하는 동안 IAM 콘솔과 AWS IoT 콘솔 간에 전환하게 됩니다. 이 절차를 수행하는 동안 두 콘솔을 동시에 여는 것이 좋습니다.

디바이스를 설치할 사용자에 대한 IAM 정책을 생성하려면

1. IAM 콘솔의 정책 허브를 엽니다.
2. 정책 생성을 선택하세요.
3. 정책 생성 페이지에서 JSON 탭을 선택합니다.
4. AWS IoT 콘솔에서 사용자 정책 및 역할 구성을 선택한 페이지로 전환합니다.
5. 샘플 프로비저닝 정책(Sample provisioning policy)에서 복사(Copy)를 선택합니다.
6. IAM 콘솔로 다시 전환합니다.

7. JSON 편집기에서 AWS IoT 콘솔에서 복사한 정책을 붙여 넣습니다. 이 정책은 AWS IoT 콘솔에서 생성하는 템플릿에만 적용됩니다.
8. 계속하려면 다음: 태그를 선택합니다.
9. 태그 추가(선택 사항)(Add tags(Optional)) 페이지에서 이 정책에 추가하려는 각 태그에 대해 태그 추가(Add tag)를 선택합니다. 추가할 태그가 없는 경우 이 단계를 건너뛸 수 있습니다.
10. 계속하려면 다음: 검토를 선택합니다.
11. 정책 검토 페이지에서 다음을 수행합니다.

    a. 이름(Name)*에 정책 용도를 기억하는 데 도움이 되는 정책 이름을 입력합니다.

       다음 절차에서 사용할 것이므로 이 정책에 지정한 이름을 기록해 둡니다.

    b. 생성 중인 정책에 대한 선택적인 설명을 입력할 수 있습니다.

    c. 이 정책의 나머지 부분과 태그를 검토합니다.
12. 정책 생성(Create policy)을 선택하여 새 정책의 생성을 마칩니다.

새 정책을 생성한 후 the section called “디바이스를 설치할 사용자에 대한 IAM 역할 생성”로 계속 진행하여 이 정책을 연결할 사용자의 역할 항목을 만듭니다.

디바이스를 설치할 사용자에 대한 IAM 역할 생성

이 단계에서는 프로비저닝 템플릿을 사용하여 디바이스를 설치할 사용자를 인증하는 IAM 역할을 생성하는 방법을 설명합니다.

디바이스를 설치할 사용자에 대한 IAM 정책을 생성하려면

1. IAM 콘솔에서 역할 허브를 엽니다.
2. 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 선택(Select trusted entity)에서 생성 중인 템플릿에 대한 액세스 권한을 부여할 신뢰할 수 있는 엔터티 유형을 선택합니다.
4. 액세스 권한을 부여하려는 신뢰할 수 있는 엔터티 ID를 선택하거나 입력한 후 다음(Next)을 선택합니다.
5. 권한 추가(Add permissions) 페이지에서 권한 정책(Permission policies)의 검색 상자에 이전 절차에서 만든 정책의 이름을 입력합니다.
6. 정책 목록에서, 이전 절차에서 생성한 정책을 선택한 후 다음(Next)을 선택합니다.
7. 이름, 검토 및 생성(Name, review, and create) 섹션에서 다음을 수행합니다.

a. 역할 이름(Role name)에 이 역할의 목적을 기억하는 데 도움이 되는 역할 이름을 입력합니다.
b. 설명(Description)에 역할의 선택적인 설명을 입력할 수 있습니다. 이 항목을 입력하지 않아도 계속 진행할 수 있습니다.
c. 1단계와 2단계의 값을 검토합니다.
d. 태그 추가(선택 사항)(Add tags (Optional))에서 이 역할에 태그를 추가할 수 있습니다. 이 항목을 입력하지 않아도 계속 진행할 수 있습니다.
e. 이 페이지의 정보가 완전하고 정확한지 확인한 다음 역할 생성(Create role)을 선택합니다.

새 역할을 생성한 후 AWS IoT 콘솔로 돌아가서 템플릿을 계속 생성합니다.

기존 정책을 업데이트하여 새 템플릿 승인

다음 단계에서는 프로비저닝 템플릿을 사용하여 디바이스를 설치할 수 있도록 사용자에게 권한을 부여하는 IAM 정책에 새 템플릿을 추가하는 방법을 설명합니다.

기존 IAM 정책에 새 템플릿을 추가하려면

1. IAM 콘솔의 정책 허브를 엽니다.
2. 검색 상자에 업데이트할 정책의 이름을 입력합니다.
3. 검색 상자 아래 목록에서 업데이트하려는 정책을 찾아 정책 이름을 선택합니다.
4. 정책 요약(Policy summary)에서 JSON 탭(이 패널이 아직 표시되지 않은 경우)을 선택합니다.
5. 정책 문서를 수정하려면 정책 편집(Edit policy)을 선택합니다.
6. 편집기에서 JSON 탭(이 패널이 아직 표시되지 않은 경우)을 선택합니다.
7. 정책 문서에서 `iot:CreateProvisioningClaim` 작업이 포함된 정책 문을 찾습니다.

   정책 문서에 `iot:CreateProvisioningClaim` 작업이 포함된 정책 문이 없는 경우 다음 명령문 코드 조각을 복사하여 정책 문서의 `Statement` 배열에 추가 항목으로 붙여 넣습니다.

> **Note**
>
> 이 코드 조각은 `Statement` 배열에서 닫는 `]` 문자 앞에 위치해야 합니다. 구문 오류를 해결하기 위해 이 코드 조각 앞이나 뒤에 쉼표를 추가해야 할 수 있습니다.

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "--PUT YOUR NEW TEMPLATE ARN HERE--"
    ]
}
```

8. AWS IoT 콘솔에서 사용자 역할 권한 수정을 선택한 페이지로 전환합니다.
9. 템플릿의 리소스 ARN(Resource ARN)을 찾고 복사(Copy)를 선택합니다.
10. IAM 콘솔로 다시 전환합니다.
11. 복사한 Amazon 리소스 이름(ARN)을 `Statement` 배열의 템플릿 ARN 목록 맨 위에 붙여 넣어 첫 번째 항목으로 만듭니다.

    이 ARN이 배열에서 유일한 ARN인 경우 방금 붙여넣은 값 끝에 있는 쉼표를 제거합니다.
12. 업데이트된 정책 문을 검토하고 편집기에서 나타낸 오류를 해결합니다.
13. 업데이트된 정책 문서를 저장하려면 정책 검토(Review policy)를 선택합니다.
14. 정책을 검토하고 변경 사항 저장(Save changes)을 선택합니다.
15. AWS IoT 콘솔로 돌아갑니다.

디바이스 프로비저닝 MQTT API

플릿 프로비저닝 서비스는 다음과 같은 MQTT API 작업을 지원합니다.

- the section called “CreateCertificateFromCsr”
- the section called “CreateKeysAndCertificate”
- the section called “RegisterThing”

이 API는 주제의 *payload-format*에 따라 Concise Binary Object Representation(CBOR) 형식 및 JavaScript Object Notation(JSON) 형식의 응답 버퍼를 지원합니다. 명확성을 위해 이 섹션의 응답 및 요청 예제는 JSON 형식으로 표시됩니다.

payload-format	응답 형식 데이터 유형
cbor	Concise Binary Object Representation(CBOR)

payload-format	응답 형식 데이터 유형
json	JavaScript Object Notation(JSON)

> **Important**
>
> 요청 메시지 주제를 게시하기 전에 응답 주제를 구독하여 응답을 수신합니다. 이 API에서 사용되는 메시지는 MQTT의 게시/구독 프로토콜을 사용하여 요청 및 응답 상호 작용을 제공합니다. 요청을 게시하기 전에 응답 주제를 구독하지 않으면 해당 요청의 결과가 수신되지 않을 수 있습니다.
>
> IoT Core 플릿 프로비저닝은 API 요청을 게시하는 데 사용된 것과 동일한 MQTT 연결을 통해 디바이스 프로비저닝 MQTT API 결과를 반환합니다.

CreateCertificateFromCsr

인증서 서명 요청(CSR)에서 인증서를 생성합니다.는 Amazon Root 인증 기관(CA)에서 서명한 클라이언트 인증서를 AWS IoT 제공합니다. 새 인증서의 상태는 `PENDING_ACTIVATION`입니다. 이 인증서로 사물을 프로비저닝하도록 `RegisterThing`를 호출하면 템플릿에 설명된 대로 인증서 상태가 `ACTIVE` 또는 `INACTIVE`로 변경됩니다.

인증 기관 인증서 및 인증서 서명 요청을 사용하여 클라이언트 인증서를 생성하는 방법에 대한 자세한 내용은 CA 인증서를 사용하여 클라이언트 인증서 생성 섹션을 참조하세요.

> **Note**
>
> 보안을 위해 `CreateCertificateFromCsr`에 의해 반환된 `certificateOwnershipToken`은 1시간 후에 만료됩니다. `RegisterThing`은 `certificateOwnershipToken`이 만료되기 전에 호출되어야 합니다. `CreateCertificateFromCsr`에서 생성된 인증서가 활성화되지 않았으며 토큰이 만료될 때까지 정책 또는 사물에 연결되지 않은 경우 인증서가 삭제됩니다. 토큰이 만료되면 디바이스는 `CreateCertificateFromCsr`을 호출하여 새 인증서를 생성할 수 있습니다.

CreateCertificateFromCsr 요청

`$aws/certificates/create-from-csr/`*`payload-format`* 주제와 함께 메시지를 게시합니다.

`payload-format`

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

CreateCertificateFromCsr 요청 페이로드

```
{
    "certificateSigningRequest": "string"
}
```

`certificateSigningRequest`

PEM 형식의 CSR입니다.

CreateCertificateFromCsr 응답

`$aws/certificates/create-from-csr/`*`payload-format`*`/accepted`을 구독합니다.

`payload-format`

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

CreateCertificateFromCsr 응답 페이로드

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

`certificateOwnershipToken`

프로비저닝하는 동안 인증서의 소유권을 증명하는 토큰입니다.

`certificateId`

인증서 ID입니다. 인증서 관리 작업에서는 certificateId만 필요합니다.

`certificatePem`

PEM 형식의 인증서 데이터입니다.

CreateCertificateFromCsr 오류

오류 응답을 수신하려면 `$aws/certificates/create-from-csr/payload-format/rejected`를 구독합니다.

`payload-format`

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

CreateCertificateFromCsr 오류 페이로드

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

상태 코드

`errorCode`

오류 코드입니다.

`errorMessage`

오류 메시지입니다.

CreateKeysAndCertificate

새 키와 certificate를 생성합니다.는 Amazon Root 인증 기관(CA)에서 서명한 클라이언트 인증서를 AWS IoT 제공합니다. 새 인증서의 상태는 `PENDING_ACTIVATION`입니다. 이 인증서로 사물을 프로비저닝하도록 `RegisterThing`를 호출하면 템플릿에 설명된 대로 인증서 상태가 `ACTIVE` 또는 `INACTIVE`로 변경됩니다.

> **Note**
>
> 보안을 위해 `CreateKeysAndCertificate`에 의해 반환된 `certificateOwnershipToken`은 1시간 후에 만료됩니다. `RegisterThing`은

certificateOwnershipToken이 만료되기 전에 호출되어야 합니다. CreateKeysAndCertificate에서 생성된 인증서가 활성화되지 않았으며 토큰이 만료될 때까지 정책 또는 사물에 연결되지 않은 경우 인증서가 삭제됩니다. 토큰이 만료되면 디바이스는 CreateKeysAndCertificate을 호출하여 새 인증서를 생성할 수 있습니다.

CreateKeysAndCertificate 요청

빈 메시지 페이로드와 함께 $aws/certificates/create/*payload-format*에 메시지를 게시합니다.

payload-format

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

CreateKeysAndCertificate 응답

$aws/certificates/create/*payload-format*/accepted을 구독합니다.

payload-format

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

CreateKeysAndCertificate 응답

```
{
    "certificateId": "string",
    "certificatePem": "string",
    "privateKey": "string",
    "certificateOwnershipToken": "string"
}
```

certificateId

인증서 ID입니다.

certificatePem

PEM 형식의 인증서 데이터입니다.

`privateKey`

프라이빗 키입니다.

`certificateOwnershipToken`

프로비저닝하는 동안 인증서의 소유권을 증명하는 토큰입니다.

CreateKeysAndCertificate 오류

오류 응답을 수신하려면 `$aws/certificates/create/`*`payload-format`*`/rejected`를 구독합니다.

`payload-format`

`cbor` 또는 `json`와 같은 메시지 페이로드 형식입니다.

CreateKeysAndCertificate 오류 페이로드

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

상태 코드

`errorCode`

오류 코드입니다.

`errorMessage`

오류 메시지입니다.

RegisterThing

미리 정의된 템플릿을 사용하여 사물을 프로비저닝합니다.

RegisterThing 요청

`$aws/provisioning-templates/templateName/provision/payload-format`에 메시지를 게시합니다.

`payload-format`

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

`templateName`

프로비저닝 템플릿 이름입니다.

RegisterThing request 페이로드

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
        "string": "string",
        ...
    }
}
```

`certificateOwnershipToken`

인증서 소유권을 증명하는 토큰입니다. AWS IoT 는 MQTT를 통해 인증서를 생성할 때 토큰을 생성합니다.

`parameters`

선택 사항. 등록 요청을 평가하기 위해 사전 프로비저닝 후크에서 사용되는 디바이스의 키-값 페어입니다.

RegisterThing 응답

`$aws/provisioning-templates/templateName/provision/payload-format/accepted`을 구독합니다.

`payload-format`

cbor 또는 json와 같은 메시지 페이로드 형식입니다.

`templateName`

프로비저닝 템플릿 이름입니다.

RegisterThing 응답 페이로드

```
{
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

`deviceConfiguration`

템플릿에 정의된 디바이스 구성입니다.

`thingName`

프로비저닝 중에 생성된 IoT 사물의 이름입니다.

RegisterThing 오류 응답

오류 응답을 수신하려면 `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/rejected`를 구독합니다.

`payload-format`

`cbor` 또는 `json`와 같은 메시지 페이로드 형식입니다.

`templateName`

프로비저닝 템플릿 이름입니다.

RegisterThing 오류 응답 페이로드

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
```

```
}
```

`statusCode`

상태 코드

`errorCode`

오류 코드입니다.

`errorMessage`

오류 메시지입니다.

플릿 인덱싱

플릿 인덱싱을 사용하여 AWS IoT 레지스트리, AWS IoT 디바이스 섀도우, AWS IoT 연결, AWS IoT Device Management Software Package Catalog 및 AWS IoT Device Defender 위반과 같은 소스에서 디바이스의 데이터를 인덱싱, 검색 및 집계할 수 있습니다. 디바이스 그룹을 쿼리하고 상태, 연결 및 디바이스 위반을 포함하여 디바이스 속성의 다양한 조합을 기반으로 하는 디바이스 레코드에 대한 통계를 집계할 수 있습니다. 플릿 인덱싱을 사용하면 기기 플릿을 구성 및 조사하고 문제를 해결할 수 있습니다.

플릿 인덱싱은 다음 기능을 제공합니다.

인덱스 업데이트 관리

사물 그룹, 사물 레지스트리, 디바이스 섀도우, 디바이스 연결 및 디바이스 위반에 대한 업데이트를 인덱싱하도록 플릿 인덱스를 설정할 수 있습니다. 플릿 인덱싱을 활성화하면 AWS IoT 가 사물 또는 사물 그룹에 대한 인덱스를 생성합니다. `AWS_Things`는 모든 사물을 위해 생성된 인덱스이고, `AWS_ThingGroups`는 모든 사물 그룹을 포함하는 인덱스입니다. 플릿 인덱싱이 활성화되면 인덱스에서 쿼리를 실행할 수 있습니다. 예를 들어 휴대용이고 배터리 수명이 70% 이상인 모든 디바이스를 찾을 수 있습니다.는 인덱스를 최신 데이터로 계속 AWS IoT 업데이트합니다. 자세한 내용은 플릿 인덱싱 관리를 참조하세요.

특정 디바이스의 연결 상태 쿼리

이 API는 최신 디바이스별 연결 정보에 대해 지연 시간이 짧고 처리량이 많은 액세스를 제공합니다. 자세한 내용은 디바이스 연결 상태를 참조하세요.

데이터 원본 전체에서 검색

쿼리 언어를 기반으로 쿼리 문자열을 생성하고 이를 사용하여 데이터 소스 전체에서 검색할 수 있습니다. 또한 플릿 인덱싱 설정에서 데이터 소스를 구성하여 인덱싱 구성에 검색하려는 데이터 소스가 포함되도록 해야 합니다. 쿼리 문자열은 찾으려는 항목을 설명합니다. AWS 관리형 필드, 사용자 지정 필드 및 인덱싱된 데이터 소스의 속성을 사용하여 쿼리를 생성할 수 있습니다. 플릿 인덱싱을 지원하는 데이터 원본에 대한 자세한 내용은 사물 인덱싱 관리를 참조하세요.

집계 데이터 쿼리

디바이스에서 집계 데이터를 검색하고 통계, 백분위수, 카디널리티 또는 특정 필드와 관련된 검색 쿼리가 있는 사물 목록을 반환할 수 있습니다. AWS 관리형 필드 또는 플릿 인덱싱 설정 내에서 사용자 지정 필드로 구성하는 모든 속성에 대해 집계를 실행할 수 있습니다. 집계 데이터에 대한 자세한 내용은 집계 데이터 쿼리를 참조하세요.

플릿 지표를 사용하여 집계 데이터 모니터링 및 경보 생성

플릿 지표를 사용하여 집계 데이터를 CloudWatch에 자동으로 전송하여 추세를 분석하고 사전 정의된 임계값을 기준으로 플릿의 집계 상태를 모니터링하는 경보를 생성할 수 있습니다. 플릿 지표에 대한 자세한 내용은 플릿 지표를 참조하세요.

플릿 인덱싱 관리

플릿 인덱싱은 사물 인덱싱과 사물 그룹 인덱싱이라는 두 가지 유형의 인덱스를 관리합니다.

사물 인덱싱

모든 사물에 대해 생성된 인덱스는 `AWS_Things`입니다. 사물 인덱싱은 AWS IoT 레지스트리 데이터, AWS IoT 디바이스 섀도우 데이터, AWS IoT 연결 데이터 및 AWS IoT Device Defender 위반 데이터와 같은 데이터 소스를 지원합니다. 이러한 데이터 소스를 플릿 인덱싱 구성에 추가하면 사물을 검색하고, 집계 데이터를 쿼리하고, 검색 쿼리를 기반으로 동적 사물 그룹 및 플릿 지표를 만들 수 있습니다.

레지스트리 - 사물 관리에 도움이 되는 레지스트리를AWS IoT 제공합니다. 플릿 인덱싱 구성에 레지스트리 데이터를 추가하여 사물 이름, 설명 및 기타 레지스트리 속성을 기반으로 디바이스를 검색할 수 있습니다. 레지스트리에 대한 자세한 내용은 레지스트리를 사용하여 사물을 관리하는 방법을 참조하세요.

섀도우 - AWS IoT 디바이스 섀도우 서비스는 디바이스의 상태 데이터를 저장하는 데 도움이 되는 섀도우를 제공합니다. 사물 인덱싱은 명명되지 않은 클래식 섀도우와 명명된 섀도우를 모두 지원합니다. 명명된 섀도우를 인덱싱하려면 명명된 섀도우 설정을 활성화하고 사물 인덱싱 구성에서 섀도우 이름을 지정합니다. 기본적으로 당 최대 10개의 섀도우 이름을 추가할 수 있습니다 AWS 계정. 섀도우 이름 수 제한을 늘리는 방법을 보려면 AWS 일반 참조의 AWS IoT Device Management 할당량을 참조하세요.

인덱싱하려는 명명된 섀도우 추가

- AWS IoT 콘솔을 사용하는 경우 사물 인덱싱(Thing indexing)을 켜고 명명된 섀도우 추가(Add named shadows)를 선택한 다음 명명된 섀도우 선택(Named shadow selection)을 통해 섀도우 이름을 추가합니다.
- AWS Command Line Interface (AWS CLI)를 사용하는 경우를 `namedShadowIndexingMode`로 설정하고에서 섀도우 이름을 `ON`지정합니다`IndexingFilter`. CLI 명령 예제를 보려면 사물 인덱싱 관리를 참조하세요.

> **Important**
>
> 2022년 7월 20일은 AWS IoT 디바이스 관리 플릿 인덱싱 통합과 AWS IoT Core 명명된 섀도우 및 AWS IoT Device Defender 탐지 위반의 일반 가용성(GA) 릴리스입니다. 이번 정식 출시 릴리스를 통해 섀도우 이름을 지정하여 특정 명명된 섀도우를 인덱싱할 수 있는 기능이 지원됩니다. 2021년 11월 30일부터 2022년 7월 19일까지인 이 기능의 공개 프리뷰 기간 중에 인덱싱을 위해 명명된 섀도우를 추가한 경우, 플릿 인덱싱 설정을 재구성하고 특정 섀도우 이름을 선택하여 인덱싱 비용을 줄이고 성능을 최적화하는 것이 좋습니다.

섀도우에 대한 자세한 내용은 AWS IoT 디바이스 섀도우 서비스를 참조하세요.

연결 - 디바이스 연결 데이터를 사용하면 디바이스의 연결 상태를 식별하는 데 도움이 됩니다. 이 연결 데이터는 수명 주기 이벤트에서 제공합니다. 클라이언트가 연결 또는 연결 해제되면는 메시지와 함께 수명 주기 이벤트를 MQTT 주제에 AWS IoT 게시합니다. 연결 또는 연결 해제 메시지는 연결 상태에 대한 세부 정보를 제공하는 JSON 요소 목록일 수 있습니다. 디바이스 연결에 대한 자세한 내용은 수명 주기 이벤트를 참조하세요.

Device Defender 위반-AWS IoT Device Defender 위반 데이터는 Security Profile에서 정의한 정상 동작과 비교하여 비정상적인 디바이스 동작을 식별하는 데 도움이 됩니다. 보안 프로파일에는 예상되는 디바이스 동작 집합이 포함되어 있습니다. 각 동작은 디바이스의 정상적인 동작을 지정하는 지표표를 사용합니다. Device Defender 위반에 대한 자세한 내용은 AWS IoT Device Defender Detect를 참조하세요.

자세한 내용은 사물 인덱싱 관리를 참조하세요.

사물 그룹 인덱싱

`AWS_ThingGroups`는 모든 사물 그룹을 포함하는 인덱스입니다. 이 인덱스를 사용하여 그룹 이름, 설명, 속성 및 모든 상위 그룹 이름을 기반으로 그룹을 검색할 수 있습니다.

자세한 내용은 사물 그룹 인덱싱 관리를 참조하세요.

관리형 필드

관리형 필드에는 사물, 사물 그룹, 디바이스 섀도우, 디바이스 연결 및 Device Defender 위반과 관련된 데이터가 포함됩니다.는 관리형 필드의 데이터 유형을 AWS IoT 정의합니다. AWS IoT 사물을 생성할 때 각 관리형 필드의 값을 지정합니다. 예를 들어, 사물 이름, 사물 그룹 및 사물 설명은 모두 관리형 필드입니다. 플릿 인덱싱은 지정한 인덱싱 모드를 기반으로 관리형 필드를 인덱싱합니다. 관리형 필드는 `customFields`에서 변경하거나 표시할 수 없습니다. 자세한 내용은 사용자 정의 필드를 참조하세요.

다음은 사물 인덱싱을 위한 관리형 필드 목록입니다.

- 레지스트리에 대한 관리형 필드

```
"managedFields" : [
  {name:thingId, type:String},
  {name:thingName, type:String},
  {name:registry.version, type:Number},
  {name:registry.thingTypeName, type:String},
  {name:registry.thingGroupNames, type:String},
]
```

- 명명되지 않은 클래식 섀도우에 대한 관리형 필드

```
"managedFields" : [
  {name:shadow.version, type:Number},
  {name:shadow.hasDelta, type:Boolean}
]
```

- 명명된 섀도우에 대한 관리형 필드

```
"managedFields" : [
  {name:shadow.name.shadowName.version, type:Number},
  {name:shadow.name.shadowName.hasDelta, type:Boolean}
]
```

- 사물 연결에 대한 관리형 필드

```
"managedFields" : [
  {name:connectivity.timestamp, type:Number},
  {name:connectivity.version, type:Number},
  {name:connectivity.connected, type:Boolean},
```

```
  {name:connectivity.disconnectReason, type:String}
]
```

- Device Defender에 대한 관리형 필드

```
"managedFields" : [
  {name:deviceDefender.violationCount, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.metricName, type:String},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationTime, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationValue, type:String},
  {name:deviceDefender.securityprofile.behaviorname.inViolation, type:Boolean}
]
```

- 사물 그룹에 대한 관리형 필드

```
"managedFields" : [
  {name:description, type:String},
  {name:parentGroupNames, type:String},
  {name:thingGroupId, type:String},
  {name:thingGroupName, type:String},
  {name:version, type:Number},
]
```

다음 표에는 검색할 수 없는 관리형 필드가 나열되어 있습니다.

데이터 소스	검색할 수 없는 관리형 필드
레지스트리	`registry.version`
명명되지 않은 섀도우	`shadow.version`
명명된 섀도우	`shadow.name.*.version`
Device Defender	`deviceDefender.version`
사물 그룹	`version`

사용자 정의 필드

사물 속성, 디바이스 섀도우 데이터 및 Device Defender 위반 데이터를 인덱싱할 사용자 정의 필드를 생성하여 집계할 수 있습니다. `customFields` 속성은 필드 이름과 데이터 형식 페어의 목록입니다. 데이터 유형에 따라 집계 쿼리를 수행할 수 있습니다. 선택한 인덱싱 모드는 `customFields`에서 지정할 수 있는 필드에 영향을 줍니다. 예를 들어, `REGISTRY` 인덱싱 모드를 지정하면 사물 섀도우에서 사용자 지정 필드를 지정할 수 없습니다. update-indexing-configuration CLI 명령을 사용하여 사용자 정의 필드를 생성하거나 업데이트할 수 있습니다(인덱싱 구성 예 업데이트의 예제 명령 참조).

- 사용자 정의 필드 이름

사물 및 사물 그룹 속성에 대한 사용자 정의 필드 이름은 `attributes.`로 시작하고 그 뒤에 속성 이름이 옵니다. 명명되지 않은 섀도우 인덱싱이 켜져 있으면 사물의 사용자 정의 필드 이름이 `shadow.desired` 또는 `shadow.reported`로 시작하고 그 뒤에 명명되지 않은 섀도우 데이터 값 이름이 올 수 있습니다. 명명된 섀도우 인덱싱이 켜져 있으면 사물의 사용자 정의 필드 이름이 `shadow.name.*.desired.` 또는 `shadow.name.*.reported.`로 시작하고 그 뒤에 명명된 섀도우 데이터 값이 올 수 있습니다. Device Defender 위반 인덱싱이 켜져 있으면 사물의 사용자 정의 필드 이름이 `deviceDefender.`로 시작하고 그 뒤에 Device Defender 위반 데이터 값이 올 수 있습니다.

접두사 뒤에 오는 속성 또는 데이터 값 이름에는 영숫자, -(하이픈) 및 _(밑줄) 문자만 사용할 수 있습니다. 공백이 없어야 합니다.

구성의 사용자 정의 필드와 인덱싱되는 값 간에 유형 불일치가 있는 경우 플릿 인덱싱은 집계 쿼리에 대해 이러한 불일치 값을 무시합니다. CloudWatch Logs는 집계 쿼리 문제를 해결할 때 유용합니다. 자세한 내용은 플릿 인덱싱 서비스에 대한 집계 쿼리 문제 해결 단원을 참조하십시오.

- 사용자 지정 필드 유형

사용자 정의 필드 유형에는 `Number`, `String` 및 `Boolean` 값이 지원됩니다.

사물 인덱싱 관리

모든 사물에 대해 생성된 인덱스는 `AWS_Things`입니다. AWS IoT 레지스트리 데이터, AWS IoT 디바이스 섀도우 데이터, AWS IoT 연결 데이터 및 AWS IoT Device Defender 위반 데이터와 같은 데이터 소스에서 인덱싱할 대상을 제어할 수 있습니다.

이 주제에서 수행할 작업

- 사물 인덱싱 활성화

- 사물 인덱스 설명
- 사물 인덱스에 대한 쿼리
- 규제 및 제한
- 권한 부여

사물 인덱싱 활성화

update-indexing-configuration CLI 명령 또는 UpdateIndexingConfiguration API 작업을 사용하여 AWS_Things 인덱스를 생성하고 해당 구성을 제어합니다. --thing-indexing-configuration(thingIndexingConfiguration) 파라미터를 사용하여 인덱싱할 데이터 종류(예: 레지스트리, 섀도우, 디바이스 연결 데이터 및 Device Defender 위반 데이터)를 제어합니다.

--thing-indexing-configuration 파라미터는 다음 구조의 문자열을 사용합니다.

```
{
  "thingIndexingMode": "OFF"|"REGISTRY"|"REGISTRY_AND_SHADOW",
  "thingConnectivityIndexingMode": "OFF"|"STATUS",
  "deviceDefenderIndexingMode": "OFF"|"VIOLATIONS",
  "namedShadowIndexingMode": "OFF"|"ON",
  "managedFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "customFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "filter": {
     "namedShadowNames": [ "string" ],
     "geoLocations": [
        {
            "name": "String",
            "order": "LonLat|LatLon"
        }
    ]
```

```
  }
}
```

사물 인덱싱 모드

인덱싱 구성에서 디바이스를 인덱싱하고 검색하려는 데이터 소스에 따라 다양한 사물 인덱싱 모드를 지정할 수 있습니다.

- `thingIndexingMode`: 레지스트리 또는 섀도가 인덱싱되는지 여부를 제어합니다. `thingIndexingMode` 이 `OFF`로 설정되면 사물 인덱싱이 비활성화됩니다.

- `thingConnectivityIndexingMode`: 사물 연결 데이터가 인덱싱되는지를 지정합니다.

- `deviceDefenderIndexingMode`: Device Defender 위반 데이터가 인덱싱되는지 여부를 지정합니다.
- `namedShadowIndexingMode`: 명명된 섀도 데이터가 인덱싱되는지 여부를 지정합니다. 플릿 인덱싱 구성에 추가할 명명된 섀도우를 선택하려면 `namedShadowIndexingMode`를 `ON`으로 설정하고 `filter`에서 명명된 섀도우 이름을 지정합니다.

아래 표에는 각 인덱싱 모드의 유효한 값과 각 값에 대해 인덱싱된 데이터 소스가 나와 있습니다.

속성	유효값	레지스트리	섀도우	연결	DD 위반	명명된 섀도우
`thingIndexingMode`	OFF					
	REGISTRY	✓				
	REGISTRY_AND_SHADOW	✓	✓			
`thingConnectivityIndexingMode`	지정되지 않음.					
	OFF					
	STATUS			✓		

속성	유효값	레지스트리	섀도우	연결	DD 위반	명명된 섀도우
deviceDefenderIndexingMode	지정되지 않음.					
	OFF					
	VIOLATIONS				✓	
namedShadowIndexingMode	지정되지 않음.					
	OFF					
	ON					✓

관리형 필드 및 사용자 정의 필드

관리형 필드

관리형 필드에는 사물, 사물 그룹, 디바이스 섀도우, 디바이스 연결 및 Device Defender 위반과 관련된 데이터가 포함됩니다.는 관리형 필드의 데이터 유형을 AWS IoT 정의합니다. AWS IoT 사물을 생성할 때 각 관리형 필드의 값을 지정합니다. 예를 들어, 사물 이름, 사물 그룹 및 사물 설명은 모두 관리형 필드입니다. 플릿 인덱싱은 지정한 인덱싱 모드를 기반으로 관리형 필드를 인덱싱합니다. 관리형 필드는 `customFields`에서 변경하거나 표시할 수 없습니다.

사용자 지정 필드

속성, 디바이스 섀도우 데이터 및 Device Defender 위반 데이터를 인덱싱할 사용자 지정 필드를 만들어 집계할 수 있습니다. `customFields` 속성은 필드 이름과 데이터 형식 페어의 목록입니다. 데이터 유형에 따라 집계 쿼리를 수행할 수 있습니다. 선택한 인덱싱 모드는 `customFields`에서 지정할 수 있는 필드에 영향을 줍니다. 예를 들어, `REGISTRY` 인덱싱 모드를 지정하면 사물 섀도우에서 사용자 지정 필드를 지정할 수 없습니다. update-indexing-configuration CLI 명령을 사용하여 사용자 정의 필드를 생성하거나 업데이트할 수 있습니다(인덱싱 구성 예 업데이트의 예제 명령 참조). 자세한 내용은 사용자 정의 필드를 참조하세요.

인덱싱 필터

인덱싱 필터는 명명된 섀도 및 지리적 위치 데이터에 대한 추가 선택을 제공합니다.

namedShadowNames

플릿 인덱싱 구성에 추가할 명명된 섀도를 선택하려면 namedShadowIndexingMode를 ON으로 설정하고 namedShadowNames에서 명명된 섀도 이름을 지정합니다.

예제

```
"filter": {
     "namedShadowNames": [ "namedShadow1", "namedShadow2" ]
}
```

geoLocations

플릿 인덱싱 구성에 지리적 위치 데이터를 추가하려면:

- 지리적 위치 데이터가 클래식(이름 없음) 섀도에 저장되는 경우 thingIndexingMode를 REGISTRY_AND_SHADOW로 설정하고 geoLocations 필터에서 지리적 위치 데이터를 지정합니다.

  아래 예제 필터는 클래식(이름 없음) 섀도에서 geoLocation 객체를 지정합니다.

  ```
  "filter": {
       "geoLocations": [
          {
              "name": "shadow.reported.location",
              "order": "LonLat"
          }
       ]
    }
  ```

- 지리적 위치 데이터가 명명된 섀도우에 저장되어 있는 경우 namedShadowIndexingMode를 ON으로 설정하고 namedShadowNames 필터에 섀도 이름을 추가한 다음 geoLocations 필터에 지리 위치 데이터를 지정합니다.

  아래 예제 필터는 명명된 섀도(nameShadow1)에서 geoLocation 객체를 지정합니다.

  ```
  "filter": {
       "namedShadowNames": [ "namedShadow1" ],
       "geoLocations": [
          {
              "name": "shadow.name.namedShadow1.reported.location",
  ```

```
            "order": "LonLat"
        }
    ]
}
```

자세한 내용은 AWS IoT API 참조의 IndexingFilter를 참조하세요.

인덱싱 구성 예 업데이트

인덱싱 구성을 업데이트하려면 update-indexing-configuration CLI 명령를 AWS IoT 사용합니다. 다음 예제에서는 update-indexing-configuration을 사용하는 방법을 보여줍니다.

짧은 구문:

```
aws iot update-indexing-configuration --thing-indexing-configuration \
'thingIndexingMode=REGISTRY_AND_SHADOW, deviceDefenderIndexingMode=VIOLATIONS,
namedShadowIndexingMode=ON,filter={namedShadowNames=[thing1shadow]},
 thingConnectivityIndexingMode=STATUS,
customFields=[{name=attributes.version,type=Number},
{name=shadow.name.thing1shadow.desired.DefaultDesired, type=String},
 {name=shadow.desired.power, type=Boolean},
{name=deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number,
 type=Number}]'
```

JSON 구문:

```
aws iot update-indexing-configuration --cli-input-json \ '{
          "thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY_AND_SHADOW",
          "thingConnectivityIndexingMode": "STATUS",
          "deviceDefenderIndexingMode": "VIOLATIONS",
          "namedShadowIndexingMode": "ON",
          "filter": { "namedShadowNames": ["thing1shadow"]},
          "customFields": [ { "name": "shadow.desired.power", "type": "Boolean" },
          {"name": "attributes.version", "type": "Number"},
          {"name": "shadow.name.thing1shadow.desired.DefaultDesired", "type":
 "String"},
          {"name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
 "type": Number} ] } }'
```

이 명령은 출력을 생성하지 않습니다.

사물 인덱스 상태를 확인하려면 `describe-index` CLI 명령을 실행합니다.

```
aws iot describe-index --index-name "AWS_Things"
```

`describe-index` 명령의 출력은 다음과 같습니다.

```
{
    "indexName": "AWS_Things",
    "indexStatus": "ACTIVE",
    "schema": "MULTI_INDEXING_MODE"
}
```

> **Note**
>
> 플릿 인덱싱이 플릿 인덱스를 업데이트하는 데 잠시 시간이 걸릴 수 있습니다. `indexStatus`가 ACTIVE로 표시될 때까지 기다렸다가 사용하는 것이 좋습니다. 구성한 데이터 원본에 따라 스키마 필드에 다른 값이 있을 수 있습니다. 자세한 내용은 사물 인덱스 설명을 참조하세요.

사물 인덱싱 구성 세부 정보를 보려면 `get-indexing-configuration` CLI 명령을 실행합니다.

```
aws iot get-indexing-configuration
```

`get-indexing-configuration` 명령의 출력은 다음과 같습니다.

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY_AND_SHADOW",
        "thingConnectivityIndexingMode": "STATUS",
        "deviceDefenderIndexingMode": "VIOLATIONS",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "connectivity.disconnectReason",
                "type": "String"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
```

```
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "deviceDefender.violationCount",
                "type": "Number"
            },
            {
                "name": "shadow.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "shadow.name.*.version",
                "type": "Number"
            },
            {
                "name": "shadow.version",
                "type": "Number"
            },
            {
                "name": "connectivity.version",
                "type": "Number"
            },
            {
                "name": "connectivity.timestamp",
                "type": "Number"
            },
            {
                "name": "shadow.name.*.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
                "name": "thingId",
                "type": "String"
            },
            {
                "name": "connectivity.connected",
                "type": "Boolean"
            },
```

```
            {
                "name": "registry.thingGroupNames",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "shadow.name.thing1shadow.desired.DefaultDesired",
                "type": "String"
            },

            {
                "name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
                "type": "Number"
            },
            {
                "name": "shadow.desired.power",
                "type": "Boolean"
            },
            {
                "name": "attributes.version",
                "type": "Number"
            }
        ],
        "filter": {
              "namedShadowNames": [
                  "thing1shadow"
              ]
          }
      },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

사용자 지정 필드를 업데이트하려면 update-indexing-configuration 명령을 실행하면 됩니다. 예제는 다음과 같습니다.

```
aws iot update-indexing-configuration --thing-indexing-configuration

 'thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.version,type=Number},
```

```
{name=attributes.color,type=String},{name=shadow.desired.power,type=Boolean},
{name=shadow.desired.intensity,type=Number}]'
```

이 명령은 인덱싱 구성에 shadow.desired.intensity를 추가했습니다.

> **Note**
>
> 인덱싱 구성에서 사용자 정의 필드를 업데이트하면 모든 기존 사용자 정의 필드를 덮어씁니다. update-indexing-configuration을 호출할 때 모든 사용자 지정 필드를 지정해야 합니다.

인덱스가 다시 작성되면 새로 추가된 필드, 검색 레지스트리 데이터, 섀도우 데이터 및 사물 연결 상태 데이터에 대해 집계 쿼리를 사용할 수 있습니다.

인덱싱 모드를 변경할 때 새 인덱싱 모드를 사용하여 모든 사용자 정의 필드가 유효한지 확인합니다. 예를 들어, shadow.desired.temperature라는 사용자 정의 필드에서 REGISTRY_AND_SHADOW 모드를 사용하여 시작하는 경우 인덱싱 모드를 REGISTRY로 변경하기 전에 shadow.desired.temperature 사용자 정의 필드를 삭제해야 합니다. 인덱싱 구성에 해당 인덱싱 모드로 인덱싱되지 않은 사용자 정의 필드가 포함되어 있으면 업데이트가 실패합니다.

사물 인덱스 설명

다음은 describe-index CLI 명령을 사용하여 현재 사물 인덱스 상태를 가져오는 방법을 나타낸 명령입니다.

```
aws iot describe-index --index-name "AWS_Things"
```

명령의 응답은 다음과 같을 수 있습니다.

```
{
    "indexName": "AWS_Things",
    "indexStatus": "BUILDING",
    "schema": "REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS"
}
```

인덱싱을 처음 플릿하면가 인덱스를 AWS IoT 빌드합니다. indexStatus가 BUILDING 상태일 경우 인덱스를 쿼리할 수 없습니다. 사물 인덱스에 대한 schema는 인덱싱할 데이터 형식(REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS)을 가리킵니다.

인덱스 구성을 변경하면 인덱스를 다시 작성해야 합니다. 이 프로세스 중의 `indexStatus`는 `REBUILDING`입니다. 다시 작성되는 도중에 사물 인덱스의 데이터에 대한 쿼리는 실행할 수 있습니다. 예를 들어, 인덱스가 다시 작성되는 동안 인덱스 구성을 `REGISTRY`에서 `REGISTRY_AND_SHADOW`로 변경하는 경우 최신 업데이트를 포함하여 레지스트리 데이터를 쿼리할 수 있습니다. 하지만 리빌드가 완료될 때까지 섀도우 데이터에 대해서는 쿼리를 실행할 수 없습니다. 인덱스를 빌드 또는 리빌드하는 데 걸리는 시간은 데이터 크기에 따라 달라집니다.

구성한 데이터 원본에 따라 스키마 필드에 다른 값이 표시될 수 있습니다. 아래 테이블에는 다양한 스키마 값과 해당 설명이 나와 있습니다.

스키마	설명
OFF	데이터 원본이 구성되거나 인덱싱되지 않습니다.
REGISTRY	레지스트리 데이터는 인덱싱됩니다.
REGISTRY_AND_SHADOW	레지스트리 데이터와 명명되지 않은(클래식) 섀도우 데이터가 인덱싱됩니다.
REGISTRY_AND_CONNECTIVITY	레지스트리 데이터와 연결 데이터가 인덱싱됩니다.
REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS	레지스트리 데이터, 명명되지 않은 (클래식) 섀도우 데이터 및 연결 데이터가 인덱싱됩니다.
MULTI_INDEXING_MODE	레지스트리, 명명되지 않은 (클래식) 섀도우 또는 연결 데이터 외에 명명된 섀도우 또는 Device Defender 위반 데이터가 인덱싱됩니다.

사물 인덱스에 대한 쿼리

search-index CLI 명령을 사용하여 인덱스의 데이터에 대한 쿼리를 실행합니다.

```
aws iot search-index --index-name "AWS_Things" --query-string
          "thingName:mything*"
```

```
{
    "things":[{
```

```
          "thingName":"mything1",
          "thingGroupNames":[
             "mygroup1"
          ],
          "thingId":"a4b9f759-b0f2-4857-8a4b-967745ed9f4e",
          "attributes":{
             "attribute1":"abc"
          },
          "connectivity": {
             "connected":false,
             "timestamp":1556649874716,
             "disconnectReason": "CONNECTION_LOST"
          }
      },
      {
          "thingName":"mything2",
          "thingTypeName":"MyThingType",
          "thingGroupNames":[
             "mygroup1",
             "mygroup2"
          ],
          "thingId":"01014ef9-e97e-44c6-985a-d0b06924f2af",
          "attributes":{
             "model":"1.2",
             "country":"usa"
          },
          "shadow":{
             "desired":{
                "location":"new york",
                "myvalues":[3, 4, 5]
             },
             "reported":{
                "location":"new york",
                "myvalues":[1, 2, 3],
                "stats":{
                   "battery":78
                }
             },
             "metadata":{
                  "desired":{
                        "location":{
                             "timestamp":123456789
                         },
                        "myvalues":{
```

```
                        "timestamp":123456789
                    }
                },
                "reported":{
                    "location":{
                        "timestamp":34535454
                     },
                    "myvalues":{
                        "timestamp":34535454
                    },
                    "stats":{
                        "battery":{
                            "timestamp":34535454
                        }
                    }
                }
            },
            "version":10,
            "timestamp":34535454
        },
        "connectivity": {
            "connected":true,
            "timestamp":1556649855046
        }
    }],
    "nextToken":"AQFCuvk7zZ3D9pOYMbFCeHbdZ+h=G"
}
```

JSON 응답에서 "connectivity"(thingConnectivityIndexingMode=STATUS 설정으로 사용 설정됨)는 부울 값, 타임스탬프 및 디바이스가 AWS IoT Core에 연결되었는지를 나타내는 disconnectReason을 제공합니다. CONNECTION_LOST로 인해 POSIX 시간 1556649874716에 연결 해제된(false) 디바이스 "mything1"입니다. 연결 해제 이유에 대한 자세한 내용은 수명 주기 이벤트를 참조하세요.

```
"connectivity": {
    "connected":false,
    "timestamp":1556649874716,
    "disconnectReason": "CONNECTION_LOST"
}
```

POSIX 시간 1556649855046에 연결된(true) 디바이스 "mything2":

```
"connectivity": {
    "connected":true,
    "timestamp":1556649855046
}
```

타임스탬프는 Epoch 이후 밀리초 단위로 제공되므로 1556649855046은 2019년 4월 30일 화요일 오후 6:44:15.046(UTC)을 나타냅니다.

> **Important**
>
> 디바이스가 약 한 시간 동안 연결이 끊어져 있으면 연결 상태 값 "timestamp" 및 "disconnectReason"이 누락되었을 수 있습니다.

규제 및 제한

AWS_Things에 대한 규제 및 제한입니다.

복잡한 유형의 섀도우 필드

섀도우 필드는 필드 값이 배열을 포함하지 않는 JSON 객체 또는 단순 유형으로만 구성된 배열과 같은 단순 유형인 경우에만 인덱싱됩니다. 단순한 유형이란 문자열, 숫자 또는 문자의 한 부분이 true 또는 false라는 의미입니다. 예를 들어, 다음의 섀도우 상태에서 "palette" 필드의 값은 유형이 복잡한 항목이 포함된 어레이이므로 인덱싱되지 않습니다. "colors" 필드의 값은 어레이의 각 값이 문자열이므로 인덱스 처리됩니다.

```
{
    "state": {
        "reported": {
            "switched": "ON",
            "colors": [ "RED", "GREEN", "BLUE" ],
            "palette": [
                {
                    "name": "RED",
                    "intensity": 124
                },
                {
                    "name": "GREEN",
                    "intensity": 68
                },
```

```
                {
                    "name": "BLUE",
                    "intensity": 201
                }
            ]
        }
    }
}
```

중첩 섀도우 필드 이름

중첩 섀도우 필드의 이름은 마침표(.)로 구분된 문자열로 저장됩니다. 예를 들어, 섀도우 문서가 다음과 같을 경우

```
{
  "state": {
    "desired": {
      "one": {
        "two": {
          "three": "v2"
        }
      }
    }
  }
}
```

필드 three의 이름은 desired.one.two.three로 저장됩니다. 섀도우 문서가 다음과 같이 저장된 경우에도

```
{
  "state": {
    "desired": {
      "one.two.three": "v2"
    }
  }
}
```

둘 다 shadow.desired.one.two.three:v2에 대한 쿼리와 일치합니다. 섀도우 필드 이름에 마침표를 사용하지 마세요.

섀도우 메타데이터

섀도우 메타데이터 섹션의 필드는 인덱스 처리가 되지만, 섀도우의 "state" 섹션에서 해당 필드가 인덱스 처리된 경우에만 처리됩니다. (앞의 예에서 섀도우의 메타데이터 섹션의 "palette" 필드도 인덱싱되지 않습니다.)

등록되지 않은 디바이스

플릿 인덱싱은 레지스트리에 등록된 사물의 thingName과 연결 clientId가 동일한 디바이스의 연결 상태를 인덱싱합니다.

미등록 섀도우

UpdateThingShadow를 사용하여 AWS IoT 계정에 등록되지 않은 사물 이름을 사용하여 섀도우를 생성하는 경우이 섀도우의 필드는 인덱싱되지 않습니다. 이는 명명되지 않은 클래식 섀도우와 명명된 섀도우 모두에 적용됩니다.

숫자 값

레지스트리 혹은 섀도우 데이터가 서비스에서 숫자 값으로 인식되는 경우, 숫자 값으로 인덱스 처리합니다. 범위와 비교 연산자를 포함한 쿼리를 숫자 값에 형성할 수 있습니다(예: "attribute.foo<5" 또는 "shadow.reported.foo:[75 TO 80]"). 숫자로 인식되려면 데이터 값이 유효한 리터럴 유형 JSON 숫자여야 합니다. 값은 -2^53~2^53-1 범위의 정수, 선택적 지수 표기법이 있는 배정밀도 부동 소수점 또는 이러한 값만 포함하는 배열의 일부일 수 있습니다.

Null 값

Null 값은 인덱싱되지 않습니다.

최대값

집계 쿼리의 최대 사용자 정의 필드 수는 5개입니다.

집계 쿼리에 대해 요청된 최대 백분위수는 100입니다.

권한 부여

다음과 같이 AWS IoT 정책 작업에서 사물 인덱스를 Amazon 리소스 이름(ARN)으로 지정할 수 있습니다.

작업	Resource
iot:SearchIndex	인덱스 ARN(예: arn:aws:iot: *your-aws-region your-aws-account* :index/AWS_Things)
iot:DescribeIndex	인덱스 ARN(예: arn:aws:iot: *your-aws-region* :index/AWS_Things)

> **Note**
>
> 플릿 인덱스를 쿼리할 권한이 있는 경우 전체 플릿에서 사물 데이터에 액세스할 수 있습니다.

사물 그룹 인덱싱 관리

AWS_ThingGroups는 모든 사물 그룹을 포함하는 인덱스입니다. 이 인덱스를 사용하여 그룹 이름, 설명, 속성 및 모든 상위 그룹 이름을 기반으로 그룹을 검색할 수 있습니다.

사물 그룹 인덱싱 활성화

UpdateIndexingConfiguration API의 thing-group-indexing-configuration 설정을 사용하여 AWS_ThingGroups 인덱스를 생성하고 해당 구성을 제어할 수 있습니다. GetIndexingConfiguration API를 사용하여 현재 인덱싱 구성을 가져올 수 있습니다.

사물 그룹 인덱싱 구성을 업데이트하려면 update-indexing-configuration CLI 명령을 실행합니다.

```
aws iot update-indexing-configuration --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

또한 다음과 같이 단일 명령에서 사물 및 사물 그룹 인덱싱 모두에 대한 구성을 업데이트할 수 있습니다.

```
aws iot update-indexing-configuration --thing-indexing-configuration
 thingIndexingMode=REGISTRY --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

thingGroupIndexingMode의 유효한 값은 다음과 같습니다.

OFF

인덱싱이 없거나 인덱스를 삭제합니다.

ON

AWS_ThingGroups 인덱스를 생성하거나 구성합니다.

현재 사물 및 사물 그룹 인덱싱 구성을 검색하려면 get-indexing-configuration CLI 명령을 사용합니다.

```
aws iot get-indexing-configuration
```

명령의 응답은 다음과 같습니다.

```
{
   "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "ON"
    }
}
```

그룹 인덱스 설명

하여 현재 describe-index 인덱스 상태를 검색하려면 AWS_ThingGroups CLI 명령을 사용합니다.

```
aws iot describe-index --index-name "AWS_ThingGroups"
```

명령의 응답은 다음과 같습니다.

```
{
   "indexStatus": "ACTIVE",
   "indexName": "AWS_ThingGroups",
   "schema": "THING_GROUPS"
}
```

AWS IoT 는 인덱싱을 처음 수행할 때 인덱스를 빌드합니다. indexStatus가 BUILDING 상태일 경우에는 인덱스에 대한 쿼리를 실행할 수 없습니다.

사물 그룹 인덱스에 대한 쿼리

인덱스의 데이터를 쿼리하려면 search-index CLI 명령을 사용합니다.

```
aws iot search-index --index-name "AWS_ThingGroups" --query-string
 "thingGroupName:mythinggroup*"
```

권한 부여

다음과 같이 AWS IoT 정책 작업에서 사물 그룹 인덱스를 리소스 ARN으로 지정할 수 있습니다.

작업	Resource
iot:SearchIndex	인덱스 ARN(예: arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups)
iot:DescribeIndex	인덱스 ARN(예: arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups)

디바이스 연결 상태 쿼리

AWS IoT 플릿 인덱싱은 개별 디바이스 연결 쿼리를 지원하므로 특정 디바이스의 연결 상태 및 관련 메타데이터를 효율적으로 검색할 수 있습니다. 이 기능은 기존 플릿 전체 인덱싱 및 쿼리 기능을 보완합니다.

작동 방식

디바이스 연결 쿼리 지원은 최적화된 단일 디바이스 연결 상태 검색에 사용할 수 있습니다. 이 API는 최신 디바이스별 연결 정보에 대해 지연 시간이 짧고 처리량이 많은 액세스를 제공합니다. 연결 인덱싱을 활성화하면 표준 쿼리로 요금이 청구되는 이 쿼리 API에 액세스할 수 있습니다. 자세한 내용은 AWS IoT Device Management 요금을 참조하세요.

특성

디바이스 연결 쿼리 지원을 통해 다음을 수행할 수 있습니다.

1. thingName을 사용하여 지정된 디바이스에 대한 현재 연결 상태(연결 또는 연결 해제됨)를 쿼리합니다.
2. 다음을 포함한 추가 연결 메타데이터를 검색합니다.
   a. 연결 해제 이유

b. 가장 최근 연결 또는 연결 해제 이벤트의 타임 스탬프입니다.

> **Note**
>
> 플릿 인덱싱은 레지스트리에 등록된 사물의 `thingName`과 연결 `clientId`가 동일한 디바이스의 연결 상태를 인덱싱합니다.

이점

1. 짧은 지연 시간: 최신 디바이스 연결 상태를 반영하고 짧은 지연 시간을 제공하여 IoT Core의 연결 상태 변경을 반영합니다. IoT Core는 디바이스로부터 연결 해제 요청을 받는 즉시 또는 연결 해제 요청을 보내지 않고 디바이스가 연결 해제되는 경우, 디바이스가 연결 해제된 것으로 판단합니다. IoT 코어는 클라이언트가 연결 해제되었다고 확인되기 전에 구성된 연결 유지 시간의 1.5배를 기다립니다. 연결 상태 API는 일반적으로 IoT Core가 디바이스의 연결 상태를 확인한 후 1초 이내에 이러한 변경 사항을 반영합니다.
2. 높은 처리량: 기본적으로 350개의 초당 트랜잭션(TPS)을 지원하며 요청 시 더 높게 조정할 수 있습니다.
3. 데이터 보존: 플릿 인덱싱(FI) ConnectivityIndexing 모드가 활성화되어 있고 사물이 삭제되지 않은 경우, 이벤트 데이터를 무기한 저장합니다. 연결 인덱싱을 비활성화하면 레코드가 보존되지 않습니다.

> **Note**
>
> 이 API를 시작하기 전에 연결 상태 인덱싱이 활성화된 경우, 플릿 인덱싱은 API 시작 후에 발생한 연결 상태 변경을 추적하기 시작하고 이러한 변경 사항에 따라 업데이트된 상태를 반영합니다.

사전 조건

디바이스 연결 쿼리 지원을 사용하려면 다음을 수행하세요.

1. AWS 계정 설정
2. 원하는 리전 AWS IoT Core 의에 디바이스 온보딩 및 등록
3. 연결 인덱싱을 사용하여 플릿 인덱싱 활성화

> **Note**
>
> 연결 인덱싱이 이미 활성화된 경우 추가 설정이 필요하지 않습니다.

자세한 설정 지침은 AWS IoT 개발자 안내서를 참조하세요.

예제

```
aws iot get-thing-connectivity-data --thing-name myThingName
```

```
{
   "connected": true,
   "disconnectReason": "NONE",
   "thingName": "myThingName",
   "timestamp": "2024-12-19T10:00:00.000000-08:00"
}
```

- `thingName`: 요청에 표시된 디바이스의 이름입니다. 이는 AWS IoT Core에 연결하는 데 사용하는 clientId와도 일치합니다.
- `disconnectReason`: 연결 해제 이유입니다. 연결된 디바이스의 경우 NONE으로 나타납니다.
- `connected`: 이 디바이스가 현재 연결되어 있음을 나타내는 부울 값 true입니다.
- `timestamp`: 디바이스의 가장 최근에 있었던 연결 해제를 밀리초로 나타내는 타임 스탬프입니다.

```
aws iot get-thing-connectivity-data --thing-name myThingName
```

```
{
   "connected": false,
   "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
   "thingName": "myThingName",
   "timestamp": "2024-12-19T10:30:00.000000-08:00"
}
```

- `thingName`: 요청에 표시된 디바이스의 이름입니다. 이는 AWS IoT Core에 연결하는 데 사용하는 clientId와도 일치합니다.

- `disconnectReason`: 연결 해제 이유는 클라이언트가 연결 해제 AWS IoT Core 될 것이라고 표시했음을 나타내는 CLIENT_INITIATED_DISCONNECT입니다.
- `connected`: 이 디바이스가 현재 연결 해제되었음을 나타내는 부울 값 false입니다.
- `timestamp`: 디바이스의 가장 최근에 있었던 연결 해제를 밀리초로 나타내는 타임 스탬프입니다.

```
aws iot get-thing-connectivity-data --thing-name neverConnectedThing
```

```
{
   "connected": false,
   "disconnectReason": "UNKNOWN",
   "thingName": "neverConnectedThing"
}
```

- `thingName`: 요청에 표시된 디바이스의 이름입니다. 이는 AWS IoT Core에 연결하는 데 사용하는 clientId와도 일치합니다.
- `disconnectReason`: 연결 해제 이유입니다. 연결된 적이 없거나 플릿 인덱싱에 마지막 연결 해제 이유가 저장되지 않은 디바이스의 경우 "UNKNOWN"이 됩니다.
- `connected`: 이 디바이스가 현재 연결 해제되었음을 나타내는 부울 값 false입니다.
- `timestamp`: 연결된 적이 없거나 플릿 인덱싱에 마지막 타임 스탬프가 저장되지 않은 디바이스에 대해서는 타임 스탬프가 반환되지 않습니다.

집계 데이터 쿼리

AWS IoT 는 디바이스 플릿에서 집계 데이터를 검색할 수 있는 4APIs(`GetStatistics`, `GetCardinalityGetPercentiles`, 및 `GetBucketsAggregation`)를 제공합니다.

> **Note**
>
> 집계 API에 대한 누락되거나 예상치 못한 값과 관련된 문제의 경우 플릿 인덱싱 문제 해결 안내서를 참조하세요.

GetStatistics

GetStatistics API 및 get-statistics CLI 명령은 지정된 집계 필드에 대한 개수, 평균, 합계, 최소값, 최대값, 제곱합, 분산 및 표준 편차를 반환합니다.

get-statistics CLI 명령은 다음 파라미터를 사용합니다.

`index-name`

검색할 인덱스의 이름입니다. 기본값은 `AWS_Things`입니다.

`query-string`

인덱스 검색에 사용되는 쿼리입니다. 에서 인덱싱된 모든 사물의 수를 가져오`"*"`도록를 지정할 수 있습니다 AWS 계정.

`aggregationField`

(선택 사항) 집계할 필드입니다. 이 필드는 update-indexing-configuration을 호출할 때 정의된 관리형 필드 또는 사용자 지정 필드여야 합니다. 집계 필드를 지정하지 않은 경우 `registry.version`이 집계 필드로 사용됩니다.

`query-version`

사용할 쿼리 버전입니다. 기본값은 `2017-09-30`입니다.

집계 필드의 유형은 반환된 통계에 영향을 줄 수 있습니다.

문자열 값을 사용하는 GetStatistics

문자열 필드에서 집계하는 경우 `GetStatistics`를 호출하면 쿼리와 일치하는 속성이 있는 디바이스 수가 반환됩니다. 다음 예를 참조하세요.

```
aws iot get-statistics --aggregation-field 'attributes.stringAttribute'
            --query-string '*'
```

이 명령은 이름이 `stringAttribute`인 속성이 포함된 디바이스 수를 반환합니다.

```
{
  "statistics": {
    "count": 3
```

```
  }
}
```

부울 값을 사용하는 GetStatistics

부울 집계 필드로 `GetStatistics`를 호출하는 경우:

- AVERAGE는 쿼리와 일치하는 디바이스의 백분율입니다.
- MINIMUM은 다음 규칙에 따라 0 또는 1입니다.
  - 집계 필드의 모든 값이 `false`인 경우 MINIMUM은 0입니다.
  - 집계 필드의 모든 값이 `true`인 경우 MINIMUM은 1입니다.
  - 집계 필드의 값이 `false`와 `true`의 혼합인 경우 MINIMUM은 0입니다.
- MAXIMUM은 다음 규칙에 따라 0 또는 1입니다.
  - 집계 필드의 모든 값이 `false`인 경우 MAXIMUM은 0입니다.
  - 집계 필드의 모든 값이 `true`인 경우 MAXIMUM은 1입니다.
  - 집계 필드의 값이 `false`와 `true`의 혼합인 경우 MAXIMUM은 1입니다.
- SUM은 부울 값에 해당하는 정수의 합계입니다.
- COUNT는 쿼리 문자열 기준과 일치하고 유효한 집계 필드 값을 포함하는 항목의 개수입니다.

숫자 값을 사용하는 GetStatistics

`GetStatistics`를 호출하고 `Number` 유형의 집계 필드를 지정하면 `GetStatistics`에서 다음 값을 반환합니다.

count

쿼리 문자열 기준과 일치하고 유효한 집계 필드 값을 포함하는 항목의 개수입니다.

평균

쿼리와 일치하는 숫자 값의 평균입니다.

sum

쿼리와 일치하는 숫자 값의 합계입니다.

minimum

쿼리와 일치하는 숫자 값 중 가장 작은 값입니다.

maximum

쿼리와 일치하는 숫자 값 중 가장 큰 값입니다.

sumOfSquares

쿼리와 일치하는 숫자 값의 제곱합입니다.

variance

쿼리와 일치하는 숫자 값의 분산입니다. 값 집합의 분산은 해당 집합의 평균 값과 각 값의 차이에 대한 제곱의 평균입니다.

stdDeviation

쿼리와 일치하는 숫자 값의 표준 편차입니다. 값 집합의 표준 편차는 해당 값이 얼마나 분산되어 있는지를 측정한 것입니다.

다음 예제에서는 숫자 사용자 지정 필드로 get-statistics를 호출하는 방법을 보여줍니다.

```
aws iot get-statistics --aggregation-field 'attributes.numericAttribute2'
            --query-string '*'
```

```
{
  "statistics": {
    "count": 3,
    "average": 33.333333333333336,
    "sum": 100.0,
    "minimum": -125.0,
    "maximum": 150.0,
    "sumOfSquares": 43750.0,
    "variance": 13472.22222222222,
    "stdDeviation": 116.06990230986766
  }
}
```

숫자 집계 필드의 경우 필드 값이 최대 실수 값을 초과하면 통계 값이 비어 있습니다.

GetCardinality

GetCardinality API 및 get-cardinality CLI 명령은 쿼리와 일치하는 고유한 값의 대략적인 개수를 반환합니다. 예를 들어, 배터리 잔량이 50% 미만인 디바이스 수를 찾으려는 경우

```
aws iot get-cardinality --index-name AWS_Things --query-string "batterylevel
          > 50" --aggregation-field "shadow.reported.batterylevel"
```

이 명령은 배터리 잔량이 50% 이상인 사물 수를 반환합니다.

```
{
    "cardinality": 100
}
```

`cardinality`는 일치하는 필드가 없어도 항상 get-cardinality에 의해 반환됩니다. 다음 예를 참조하세요.

```
aws iot get-cardinality --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_STR"
```

```
{
    "cardinality": 0
}
```

get-cardinality CLI 명령은 다음 파라미터를 사용합니다.

`index-name`

검색할 인덱스의 이름입니다. 기본값은 `AWS_Things`입니다.

`query-string`

인덱스 검색에 사용되는 쿼리입니다. 에서 인덱싱된 모든 사물의 수를 가져오`"*"`도록를 지정할 수 있습니다 AWS 계정.

`aggregationField`

집계할 필드입니다.

`query-version`

사용할 쿼리 버전입니다. 기본값은 `2017-09-30`입니다.

GetPercentiles

GetPercentiles API 및 get-percentiles CLI 명령은 쿼리와 일치하는 집계된 값을 백분위수 그룹으로 그룹화합니다. 기본 백분위수 그룹은 1, 5, 25, 50, 75, 95, 99입니다. 단, `GetPercentiles`를 호출할 때

직접 백분위수를 지정할 수 있습니다. 이 함수는 지정된 각 백분위수 그룹(또는 기본 백분위수 그룹)에 대한 값을 반환합니다. 백분위수 그룹 “1"에는 쿼리와 일치하는 값의 약 1%에서 발생하는 집계된 필드 값이 포함됩니다. 백분위수 그룹 “5"에는 쿼리와 일치하는 값의 약 5%에서 발생하는 집계된 필드 값이 포함됩니다. 결과는 근사치이며 쿼리와 일치하는 값이 많을수록 백분위수 값이 더 정확합니다.

다음 예제에서는 get-percentiles CLI 명령을 호출하는 방법을 보여줍니다.

```
aws iot get-percentiles --query-string "thingName:*" --aggregation-field
          "attributes.customField_NUM" --percents 10 20 30 40 50 60 70 80 90 99
```

```
{
    "percentiles": [
        {
            "value": 3.0,
            "percent": 80.0
        },
        {
            "value": 2.5999999999999996,
            "percent": 70.0
        },
        {
            "value": 3.0,
            "percent": 90.0
        },
        {
            "value": 2.0,
            "percent": 50.0
        },
        {
            "value": 2.0,
            "percent": 60.0
        },
        {
            "value": 1.0,
            "percent": 10.0
        },
        {
            "value": 2.0,
            "percent": 40.0
        },
        {
            "value": 1.0,
            "percent": 20.0
```

```
        },
        {
            "value": 1.4,
            "percent": 30.0
        },
        {
            "value": 3.0,
            "percent": 99.0
        }
    ]
}
```

다음 명령은 일치하는 문서가 없을 때 get-percentiles에서 반환된 출력을 보여줍니다.

```
aws iot get-percentiles --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_NUM"
```

```
{
    "percentiles": []
}
```

get-percentile CLI 명령은 다음 파라미터를 사용합니다.

`index-name`

검색할 인덱스의 이름입니다. 기본값은 `AWS_Things`입니다.

`query-string`

인덱스 검색에 사용되는 쿼리입니다. 에서 인덱싱된 모든 사물의 수를 가져오`"*"`도록를 지정할 수 있습니다 AWS 계정.

`aggregationField`

집계할 필드로, `Number` 유형이어야 합니다.

`query-version`

사용할 쿼리 버전입니다. 기본값은 `2017-09-30`입니다.

`percents`

(선택 사항) 이 파라미터를 사용하여 사용자 지정 백분위수 그룹을 지정할 수 있습니다.

GetBucketsAggregation

GetBucketsAggregation API 및 get-buckets-aggregation CLI 명령은 쿼리 문자열 기준에 맞는 버킷 목록과 총 사물 수를 반환합니다.

다음 예제에서는 get-buckets-aggregation CLI 명령을 호출하는 방법을 보여줍니다.

```
aws iot get-buckets-aggregation --query-string '*' --index-name AWS_Things --
aggregation-field 'shadow.reported.batterylevelpercent' --buckets-aggregation-type
 'termsAggregation={maxBuckets=5}'
```

이 명령은 다음을 반환합니다.

```
{
    "totalCount": 20,
    "buckets": [
        {
            "keyValue": "100",
            "count": 12
        },
        {
            "keyValue": "90",
            "count": 5
        },
        {
            "keyValue": "75",
            "count": 3
        }
    ]
}
```

get-buckets-aggregation CLI 명령은 다음 파라미터를 사용합니다.

`index-name`

검색할 인덱스의 이름입니다. 기본값은 `AWS_Things`입니다.

`query-string`

인덱스 검색에 사용되는 쿼리입니다. 에서 인덱싱된 모든 사물의 수를 가져오`"*"`도록를 지정할 수 있습니다 AWS 계정.

`aggregation-field`

집계할 필드입니다.

`buckets-aggregation-type`

수행할 응답 형태 및 버킷 집계 유형의 기본 제어입니다.

권한 부여

다음과 같이 AWS IoT 정책 작업에서 사물 그룹 인덱스를 리소스 ARN으로 지정할 수 있습니다.

작업	Resource
`iot:GetStatistics`	인덱스 ARN(예: `arn:aws:iot:` *`your-aws-region`* `:index/AWS_Things` 또는 `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`)

쿼리 구문

플릿 인덱싱에서는 쿼리 구문을 사용하여 쿼리를 지정합니다.

지원되는 기능

쿼리 구문은 다음 기능을 지원합니다.

- 용어 및 구
- 필드 검색
- 접두사 검색
- 범위 검색
- 부울 연산자 `AND`, `OR`, `NOT` 및 `-`. 하이픈은 검색 결과에서 일부를 제외하는데 사용됩니다(예: `thingName:(tv* AND -plasma)`).
- 그룹화
- 필드 그룹화
- 특수 문자의 이스케이프 처리(예: \ 사용)

- 선행 와일드카드 사용은 쿼리당 쿼리 용어 1개로 제한됩니다. 예를 들어 `thingName:*my` 및 `thingGroupNames:*echo` 모두를 동일한 쿼리에서 검색할 수는 없습니다. 선행 와일드카드가 포함된 쿼리의 최대 쿼리 길이는 100자입니다.

> **Note**
>
> 플릿 인덱싱 쿼리 API 제한이 초당 요청 30개를 초과하는 계정은 선행 와일드카드 쿼리 용어를 사용할 때 제한이 적용됩니다. 이 requests-per-second 30개 한도는 모든 플릿 인덱싱 쿼리 APIs의 계정 수준에서 적용됩니다.

지원되지 않는 기능

쿼리 구문은 다음 기능을 지원하지 않습니다.

- 정규식
- 부스팅
- 순위 결정
- 퍼지 검색
- 근접 검색
- 정렬
- 집계
- 특수 문자: `` ` ``, `@`, `#`, `%`, `\`, `/`, `'`, `;` 및 `,`. `,`는 지오쿼리에서만 지원됩니다.

참고

쿼리 언어에 대해서 알아야 할 몇 가지가 있습니다.

- 기본 연산자는 AND입니다. `"thingName:abc thingType:xyz"`에 대한 쿼리는 `"thingName:abc AND thingType:xyz"`와 같습니다.
- 필드가 지정되지 않은 경우는 모든 레지스트리, 디바이스 섀도우 및 디바이스 디펜더 필드의 용어를 AWS IoT 검색합니다.
- 모든 필드 이름은 대/소문자를 구분합니다.
- 검색은 대/소문자를 구분하지 않습니다. 단어는 Java의 `Character.isWhitespace(int)`에 의해 정의된 대로 공백 문자로 구분됩니다.

- 디바이스 섀도우 데이터(이름 없는 섀도우 및 명명된 섀도우)의 인덱싱에는 보고된 섹션, 원하는 섹션, 델타 섹션 및 메타데이터 섹션이 포함됩니다.
- 디바이스 섀도우 및 레지스트리 버전은 검색할 수 없지만 응답에 표시됩니다.
- 쿼리 1개의 최대 용어 수는 12개입니다.
- 특수 문자 `,`는 지오쿼리에서만 지원됩니다.

사물 쿼리 예

쿼리 구문을 사용하여 쿼리 문자열에서 쿼리를 지정합니다. 쿼리가 `SearchIndex` API에 전달됩니다. 다음은 쿼리 문자열의 몇 가지 예를 나열한 표입니다.

쿼리 문자열	결과
`abc`	레지스트리, 섀도우(명명되지 않은 클래식 섀도우와 명명된 섀도우) 또는 Device Defender 위반 필드에서 "abc"를 쿼리합니다.
`thingName:myThingName`	이름이 "myThingName"인 사물에 대한 쿼리를 실행합니다.
`thingName:my*`	이름이 "my"로 시작하는 사물에 대한 쿼리를 실행합니다.
`thingName:*my`	이름이 "my"로 끝나는 사물에 대한 쿼리를 실행합니다. 선행 와일드카드 사용은 쿼리당 쿼리 용어 1개로 제한됩니다. 선행 와일드카드가 포함된 쿼리의 최대 쿼리 길이는 100자입니다.
`thingName:*my*`	하위 문자열 "my"가 포함된 사물에 대한 쿼리입니다. 선행 와일드카드 사용은 쿼리당 쿼리 용어 1개로 제한됩니다. 선행 와일드카드가 포함된 쿼리의 최대 쿼리 길이는 100자입니다.
`thingName:ab?`	'ab'에 문자 1개가 추가된 이름(예: 'aba', 'abb', 'abc' 등)의 사물에 대한 쿼리를 실행합니다.
`thingTypeName:aa`	유형 "aa"와 연결된 사물에 대한 쿼리를 실행합니다.

쿼리 문자열	결과
`thingGroupNames:a`	상위 사물 그룹 또는 청구 그룹 이름이 ‘a’인 사물에 대한 쿼리입니다.
`thingGroupNames:a*`	상위 사물 그룹 또는 청구 그룹 이름이 패턴 ‘a’와 일치하는 사물에 대한 쿼리입니다.
`attributes.myAttribute:75`	"myAttribute" 속성 값이 75인 사물에 대한 쿼리를 실행합니다.
`attributes.myAttribute: [75 TO 80]`	"myAttribute" 속성 값이 숫자 범위(75~80, 두 값 모두 포함)에 해당하는 사물에 대한 쿼리를 실행합니다.
`attributes.myAttribute: {75 TO 80]`	"myAttribute" 속성 값이 숫자 범위(>75 및 <=80)에 해당하는 사물에 대한 쿼리를 실행합니다.
`attributes.serialNumber: ["abcd" TO "abcf"]`	"serialNumber" 속성 값이 영숫자 문자열 범위에 속하는 사물에 대한 쿼리를 실행합니다. 이 쿼리는 "serialNumber" 속성 값이 "abcd", "abce" 또는 "abcf"인 사물을 반환합니다.
`attributes.myAttribute:i*t`	"myAttribute" 속성 값이 'i'부터 시작하여 개수에 제한 없이 문자가 중간에 오고 마지막에 't'로 끝나는 사물에 대한 쿼리를 실행합니다.
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	부울 표현식을 사용하여 용어가 결합되어 있는 사물에 대한 쿼리를 실행합니다. 이 쿼리는 속성 값이 'abc'이고 속성 이름이 'attr1'인 사물, 5보다 작으며 속성 이름이 'attr2'인 사물, 10 이하이며 속성 이름이 'attr3'인 사물을 반환합니다.
`shadow.hasDelta:true`	델타 요소가 있는 명명되지 않은 섀도우가 있는 항목을 쿼리합니다.
`NOT attributes.model:legacy`	속성 "모델"이 "legacy"가 아닌 사물에 대한 쿼리를 실행합니다.

쿼리 문자열	결과
`shadow.reported.stats.battery:{70 TO 100} (v2 OR v3) NOT attributes.model:legacy`	다음과 같은 사물에 대한 쿼리를 실행합니다. • 사물의 섀도우 `stats.battery` 속성 값이 70~100인 경우 • 텍스트 "v2" 또는 "v3"이 사물 이름, 유형 이름 또는 속성 값에 포함되는 경우 • 사물의 `model` 속성이 "legacy"로 설정되지 않은 경우
`shadow.reported.myvalues:2`	섀도우의 보고 섹션에 있는 `myvalues` 어레이에 2의 값이 포함된 사물에 대한 쿼리를 실행합니다.
`shadow.reported.location:* NOT shadow.desired.stats.battery:*`	다음과 같은 사물에 대한 쿼리를 실행합니다. • `location` 속성이 섀도우의 `reported` 섹션에 있는 경우 • `stats.battery` 속성이 섀도우의 `desired` 섹션에 없는 경우
`shadow.name.<shadowName>.hasDelta:true`	주어진 이름의 섀도우와 델타 요소가 있는 사물에 대한 쿼리입니다.
`shadow.name.<shadowName>.desired.filament:*`	주어진 이름의 섀도우와 원하는 필라멘트 속성이 있는 사물에 대한 쿼리입니다.
`shadow.name.<shadowName>.reported.location:*`	지정된 이름의 섀도우가 있고 명명된 섀도우의 보고된 섹션에 `location` 속성이 존재하는 사물에 대한 쿼리입니다.
`connectivity.connected:true`	연결된 모든 디바이스를 쿼리합니다.
`connectivity.connected:false`	연결이 끊긴 모든 디바이스를 쿼리합니다.

쿼리 문자열	결과
`connectivity.connected:true AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	연결 타임스탬프가 >= 1557651600000이고, <= 1557867600000인 연결된 디바이스를 모두 쿼리합니다. 타임스탬프는 Epoch 이후 밀리초 단위로 제공됩니다.
`connectivity.connected:false AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	연결 해제 타임스탬프가 >= 1557651600000이고, <= 1557867600000인 연결이 끊긴 디바이스를 모두 쿼리합니다. 타임스탬프는 Epoch 이후 밀리초 단위로 제공됩니다.
`connectivity.connected:true AND connectivity.timestamp > 1557651600000`	연결 타임스탬프가 > 1508972224인 연결된 디바이스를 모두 쿼리합니다. 타임스탬프는 Epoch 이후 밀리초 단위로 제공됩니다.
`connectivity.connected:*`	연결 정보가 있는 모든 디바이스를 쿼리합니다.
`connectivity.disconnectReason:*`	연결 disconnectReason이 있는 모든 디바이스 쿼리에 대한 쿼리를 실행합니다.
`connectivity.disconnectReason:CLIENT_INITIATED_DISCONNECT`	CLIENT_INITIATED_DISCONNECT로 인해 연결이 해제된 모든 디바이스에 대한 쿼리입니다.
`deviceDefender.violationCount:[0 TO 100]`	Device Defender 위반 개수 값이 숫자 범위(0~100, 두 값 모두 포함)에 속하는 사물에 대한 쿼리입니다.
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.inViolation:true`	보안 프로파일 `device-SecurityProfile` 에 정의된 동작 `disconnectBehavior` 를 위반하는 사물에 대한 쿼리입니다. inViolation:false는 유효한 쿼리가 아닙니다.
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationValue.number>2`	마지막 위반 이벤트 값이 2보다 큰 보안 프로파일 device-SecurityProfile에 정의된 동작 `disconnectBehavior` 를 위반하는 사물에 대한 쿼리입니다.

쿼리 문자열	결과
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationTime>1634227200000`	마지막 위반 이벤트가 지정된 epoch 시간 이후인 보안 프로필 device-SecurityProfile에 정의된 동작 `disconnectBehavior` 를 위반하는 사물에 대한 쿼리입니다.
`shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km`	좌표 47.6204,-122.3491에서 방사상 거리 15.5km 내에 있는 사물에 대한 쿼리입니다. 이 쿼리 문자열은 위치 데이터가 명명된 섀도에 저장될 때 적용됩니다.
`shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km`	좌표 47.6204,-122.3491에서 방사상 거리 15.5km 내에 있는 사물에 대한 쿼리입니다. 이 쿼리 문자열은 위치 데이터가 클래식 섀도에 저장될 때 적용됩니다.

사물 그룹 쿼리 예

쿼리는 쿼리 구문을 사용해 쿼리 문자열로 지정된 후 `SearchIndex` API로 전달됩니다. 다음은 쿼리 문자열의 몇 가지 예를 나열한 표입니다.

쿼리 문자열	결과
`abc`	필드에서 "abc"를 쿼리합니다.
`thingGroupName:myGroupThingName`	이름이 "myGroupThingName"인 사물 그룹에 대한 쿼리를 실행합니다.
`thingGroupName:my*`	이름이 "my"로 시작하는 사물 그룹에 대한 쿼리를 실행합니다.
`thingGroupName:ab?`	'ab'에 문자 1개가 추가된 이름(예: 'aba', 'abb', 'abc' 등)의 사물 그룹에 대한 쿼리를 실행합니다.

쿼리 문자열	결과
`attributes.myAttribute:75`	"myAttribute" 속성 값이 75인 사물 그룹에 대한 쿼리를 실행합니다.
`attributes.myAttribute:[75 TO 80]`	"myAttribute" 속성 값이 숫자 범위(75~80, 80 포함)에 해당하는 사물 그룹에 대한 쿼리를 실행합니다.
`attributes.myAttribute:[75 TO 80]`	"myAttribute" 속성 값이 숫자 범위(>75 and <=80)에 해당하는 사물 그룹에 대한 쿼리를 실행합니다.
`attributes.myAttribute: ["abcd" TO "abcf"]`	"myAttribute" 속성 값이 영숫자 문자열 범위에 속하는 사물 그룹에 대한 쿼리를 실행합니다. 이 쿼리는 "serialNumber" 속성 값이 "abcd", "abce" 또는 "abcf"인 사물 그룹을 반환합니다.
`attributes.myAttribute:i*t`	"myAttribute" 속성 값이 'i'부터 시작하여 개수에 제한 없이 문자가 중간에 오고 마지막에 't'로 끝나는 사물 그룹에 대한 쿼리를 실행합니다.
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	부울 표현식을 사용하여 용어가 결합되어 있는 사물 그룹에 대한 쿼리를 실행합니다. 이 쿼리는 속성 값이 'abc'이고 속성 이름이 'attr1'인 사물 그룹, 5보다 작으며 속성 이름이 'attr2'인 사물 그룹, 10 이하이며 속성 이름이 'attr3'인 사물 그룹을 반환합니다.
`NOT attributes.myAttribute:cde`	"myAttribute" 속성 값이 "cde"가 아닌 사물 그룹에 대한 쿼리를 실행합니다.
`parentGroupNames:(myParentThingGroupName)`	상위 그룹 이름이 "myParentThingGroupName"과 일치하는 사물 그룹에 대한 쿼리
`parentGroupNames:(myParentThingGroupName OR myRootThingGroupName)`	상위 그룹 이름이 "myParentThingGroupName" 또는 "myRootThingGroupName"과 일치하는 사물 그룹에 대한 쿼리
`parentGroupNames:(myParentThingGroupNa*)`	상위 그룹 이름이 "myParentThingGroupNa"로 시작하는 사물 그룹에 대한 쿼리

위치 데이터 인덱싱

AWS IoT 플릿 인덱싱을 사용하여 디바이스의 마지막 전송 위치 데이터를 인덱싱하고 지오쿼리를 사용하여 디바이스를 검색할 수 있습니다. 이 기능은 위치 추적 및 근접 검색과 같은 디바이스 모니터링 및 관리 사용 사례를 해결합니다. 위치 인덱싱은 다른 플릿 인덱싱 기능과 비슷하게 작동하며 사물 인덱싱에 지정할 추가 구성과 함께 작동합니다.

일반적인 사용 사례는 다음과 같습니다. 원하는 지리적 경계 내에 위치한 디바이스를 검색하고 집계합니다. 인덱싱된 데이터 소스에서 디바이스 메타데이터 및 상태와 관련된 쿼리 용어를 사용하여 위치별 인사이트 가져오기, 플릿 모니터링 맵 내에서 렌더링 지연을 줄이고 마지막으로 보고된 디바이스 위치를 추적하기 위해 특정 지리적 영역에 결과를 필터링하는 등의 세분화된 보기를 제공하고, 원하는 경계 한도를 벗어나는 디바이스를 식별하고 플릿 지표를 사용하여 경보를 생성합니다. 위치 인덱싱 및 지오쿼리를 시작하려면 ??? 섹션을 참조하세요.

지원되는 데이터 형식

AWS IoT 플릿 인덱싱은 다음 위치 데이터 형식을 지원합니다.

1. 좌표 참조 시스템의 잘 알려진 텍스트 표현

   지리적 정보 - 좌표 참조 시스템 형식의 잘 알려진 텍스트 표현 뒤에 오는 문자열입니다. 예시는 `"POINT(long lat)"`가 될 수 있습니다.

2. 좌표를 나타내는 문자열

   `"latitude, longitude"` 또는 `"longitude, latitude"` 형식의 문자열입니다. `"longitude, latitude"`을 사용하면 `geoLocations`의 `order`도 함께 지정해야 합니다. 예시는 `"41.12,-71.34"`가 될 수 있습니다.

3. lat(위도), lon(경도) 키의 객체

   이 형식은 클래식 섀도와 명명된 섀도 모두에 적용됩니다. 지원되는 키: `lat`, `latitude`, `lon`, `long`, `longitude`. 예시는 `{"lat": 41.12, "lon": -71.34}`가 될 수 있습니다.

4. 좌표를 나타내는 배열

   `[lat,lon]` 또는 `[lon,lat]` 형식의 배열입니다. GeoJSON의 좌표와 동일한 `[lon,lat]` 형식(클래식 섀도우 및 명명된 섀도에 적용 가능)을 사용하는 경우 `geoLocations`에서도 `order`를 지정해야 합니다.

   예시는 다음과 같습니다.

```
{
  "location": {
    "coordinates": [
      **Longitude**,
      **Latitude**
    ],
    "type": "Point",
    "properties": {
      "country": "United States",
      "city": "New York",
      "postalCode": "*****",
      "horizontalAccuracy": 20,
      "horizontalConfidenceLevel": 0.67,
      "state": "New York",
      "timestamp": "2023-01-04T20:59:13.024Z"
    }
  }
}
```

위치 데이터를 인덱싱하는 방법

다음 단계에서는 위치 데이터에 대한 인덱싱 구성을 업데이트하고 지오쿼리를 사용하여 디바이스를 검색하는 방법을 보여줍니다.

1. 위치 데이터가 저장되는 위치 파악

   플릿 인덱싱은 현재 클래식 섀도 또는 명명된 섀도우에 저장된 위치 데이터 인덱싱을 지원합니다.

2. 지원되는 위치 데이터 형식 사용

   위치 데이터 형식이 지원되는 데이터 형식 중 하나를 따르는지 확인합니다.

3. 인덱싱 구성 업데이트

   최소한 사물(등록) 인덱싱 구성을 활성화해야 합니다. 또한 위치 데이터가 포함된 클래식 섀도 또는 명명된 섀도에서 인덱싱을 활성화해야 합니다. 사물 인덱싱을 업데이트할 때는 인덱싱 구성에 위치 데이터를 포함해야 합니다.

4. 지오쿼리 생성 및 실행

   사용 사례에 따라 지오쿼리를 생성하고 실행하여 디바이스를 검색합니다. 구성한 지오쿼리는 쿼리 구문을 따라야 합니다. ???에서 몇 가지 예제를 찾아볼 수 있습니다.

사물 인덱싱 구성 업데이트

위치 데이터를 인덱싱하려면 인덱싱 구성을 업데이트하고 위치 데이터를 포함해야 합니다. 위치 데이터가 저장되는 위치에 따라 다음 단계에 따라 인덱싱 구성을 업데이트합니다.

클래식 섀도에 저장된 위치 데이터

위치 데이터가 클래식 섀도에 저장되는 경우 `thingIndexingMode`를 `REGISTRY_AND_SHADOW`로 설정하고 `filter`의 `geoLocations` 필드(`name` 및 `order`)에 위치 데이터를 지정해야 합니다.

다음 사물 인덱싱 구성 예제에서는 위치 데이터 경로`shadow.reported.coordinates`를 `name` 및 `LonLat`를 `order`로 지정합니다.

```
{
 "thingIndexingMode": "REGISTRY_AND_SHADOW",
 "filter": {
  "geoLocations": [
   {
    "name": "shadow.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- `thingIndexingMode`

  인덱싱 모드는 레지스트리 또는 섀도가 인덱싱되는지 여부를 제어합니다. `thingIndexingMode` 이 `OFF`로 설정되면 사물 인덱싱이 비활성화됩니다.

  클래식 섀도에 저장된 위치 데이터를 인덱싱하려면 `thingIndexingMode`를 `REGISTRY_AND_SHADOW`로 설정해야 합니다. 자세한 내용은 ??? 단원을 참조하십시오.
- `filter`

  인덱싱 필터는 명명된 섀도 및 지리적 위치 데이터에 대한 추가 선택을 제공합니다. 자세한 내용은 ??? 단원을 참조하십시오.
- `geoLocations`

  인덱싱할 지리적 위치 대상 목록입니다. 인덱싱을 위한 기본 최대 지리적 위치 대상 수는 1입니다. 제한 증가를 요청하려면 AWS IoT Device Management 할당량 섹션을 참조하세요.

- name

  지리적 위치 대상 필드의 이름입니다. name의 예제 값은 섀도의 위치 데이터 경로인 shadow.reported.coordinates가 될 수 있습니다.

- order

  지리적 위치 대상 필드의 순서입니다. 유효한 값: LatLon 및 LonLat. LatLon은 위도 및 경도를 의미하고, LonLat는 경도 및 위도를 의미합니다. 이 필드는 선택 사항입니다. 기본값은 LatLon입니다.

명명된 섀도에 저장된 위치 데이터

위치 데이터가 명명된 섀도에 저장되는 경우 namedShadowIndexingMode를 ON로 설정하고 filter의 namedShadowNames 필드에 명명된 섀도 이름을 추가하고 filter의 geoLocations 필드에 위치 데이터 경로를 지정합니다.

다음 사물 인덱싱 구성 예제에서는 위치 데이터 경로shadow.name.namedShadow1.reported.coordinates를 name 및 LonLat를 order로 지정합니다.

```
{
 "thingIndexingMode": "REGISTRY",
 "namedShadowIndexingMode": "ON",
 "filter": {
  "namedShadowNames": [
   "namedShadow1"
  ],
  "geoLocations": [
   {
    "name": "shadow.name.namedShadow1.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- thingIndexingMode

  인덱싱 모드는 레지스트리 또는 섀도가 인덱싱되는지 여부를 제어합니다. thingIndexingMode 이 OFF로 설정되면 사물 인덱싱이 비활성화됩니다.

명명된 섀도에 저장된 위치 데이터를 인덱싱하려면 thingIndexingMode를 REGISTRY(또는 REGISTRY_AND_SHADOW)로 설정해야 합니다. 자세한 내용은 ??? 단원을 참조하십시오.

- filter

  인덱싱 필터는 명명된 섀도 및 지리적 위치 데이터에 대한 추가 선택을 제공합니다. 자세한 내용은 ??? 단원을 참조하십시오.

- geoLocations

  인덱싱할 지리적 위치 대상 목록입니다. 인덱싱을 위한 기본 최대 지리적 위치 대상 수는 1입니다. 제한 증가를 요청하려면 AWS IoT Device Management 할당량 섹션을 참조하세요.

- name

  지리적 위치 대상 필드의 이름입니다. name의 예제 값은 섀도의 위치 데이터 경로인 shadow.name.namedShadow1.reported.coordinates가 될 수 있습니다.

- order

  지리적 위치 대상 필드의 순서입니다. 유효한 값: LatLon 및 LonLat. LatLon은 위도 및 경도를 의미하고, LonLat는 경도 및 위도를 의미합니다. 이 필드는 선택 사항입니다. 기본값은 LatLon입니다.

지오쿼리 예시

위치 데이터에 대한 인덱싱 구성을 완료한 후 지오쿼리를 실행하여 디바이스를 검색합니다. 지오쿼리를 다른 쿼리 문자열과 결합할 수도 있습니다. 자세한 내용은 ??? 및 ??? 섹션을 참조하세요.

쿼리 예시 1

이 예시에서는 위치 데이터가 명명된 섀도 gps-tracker에 저장된다고 가정합니다. 이 명령의 출력은 좌표(47.6204,-122.3491)가 있는 중심점에서 15.5km의 반경 거리 내에 있는 디바이스 목록입니다.

```
aws iot search-index --query-string \
"shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

쿼리 예시 2

이 예제에서는 위치 데이터가 클래식 섀도에 저장된다고 가정합니다. 이 명령의 출력은 좌표(47.6204,-122.3491)가 있는 중심점에서 15.5km의 반경 거리 내에 있는 디바이스 목록입니다.

```
aws iot search-index --query-string \
"shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

쿼리 예시 3

이 예제에서는 위치 데이터가 클래식 섀도에 저장된다고 가정합니다. 이 명령의 출력은 연결되지 않고 좌표(47.6204,-122.3491)가 있는 중심점에서 15.5km의 방사상 거리를 벗어나는 디바이스 목록입니다.

```
aws iot search-index --query-string \
"connectivity.connected:false AND (NOT
 shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km)"
```

시작하기 자습서

이 자습서에서는 플릿 인덱싱을 사용하여 위치 데이터를 인덱싱하는 방법을 보여줍니다. 간소화를 위해 디바이스를 나타내는 사물을 생성하고 위치 데이터를 명명된 섀도에 저장하고, 위치 인덱싱을 위한 사물 인덱싱 구성을 업데이트하고, 예시 지오쿼리를 실행하여 방사형 경계 내의 디바이스를 검색합니다.

이 자습서는 완료하는 데 약 15분이 소요됩니다.

이 주제에서 수행할 작업

- 사전 조건
- 사물 및 섀도 생성
- 사물 인덱싱 구성 업데이트
- 지오쿼리 실행

사전 조건

- AWS CLI의 최신 버전을 설치합니다.
- 위치 인덱싱 및 지오쿼리 , 사물 인덱싱 관리 및 쿼리 구문을 숙지합니다.

사물 및 섀도 생성

디바이스를 나타내는 사물과 위치 데이터를 저장할 명명된 섀도를 생성합니다(협정 47.61564, -122.33584).

1. 다음 명령을 실행하여 Bike-1이라는 자전거를 나타내는 사물을 생성합니다. 를 사용하여 사물을 생성하는 방법에 대한 자세한 AWS CLI내용은 AWS CLI 참조의 create-thing을 참조하세요.

```
aws iot create-thing --thing-name "Bike-1" \
--attribute-payload '{"attributes": {"model":"OEM-2302-12", "battery":"35",
 "acqDate":"06/09/23"}}'
```

이 명령의 출력은 다음과 같을 수 있습니다.

```
{
    "thingName": "Bike-1",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/Bike-1",
    "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df"
}
```

2. 다음 명령을 실행하여 Bike-1의 위치 데이터를 저장할 명명된 섀도를 생성합니다(협정 47.61564, -122.33584). 를 사용하여 명명된 섀도우를 생성하는 방법에 대한 자세한 AWS CLI내용은 AWS CLI 참조의 update-thing-shadow를 참조하세요.

```
aws iot-data update-thing-shadow \
--thing-name Bike-1 \
--shadow-name Bike1-shadow \
--cli-binary-format raw-in-base64-out \
--payload '{"state":{"reported":{"coordinates":{"lat": 47.6153, "lon": -122.3333}}}}'
 \
"output.txt" \
```

이 명령은 출력을 생성하지 않습니다. 생성한 명명된 섀도를 보려면 list-named-shadows-for-thing CLI 명령을 실행할 수 있습니다.

```
aws iot-data list-named-shadows-for-thing --thing-name Bike-1
```

이 명령의 출력은 다음과 같을 수 있습니다.

```
{
    "results": [
        "Bike1-shadow"
    ],
    "timestamp": 1699574309
}
```

사물 인덱싱 구성 업데이트

위치 데이터를 인덱싱하려면 사물 인덱싱 구성을 업데이트하여 위치 데이터를 포함해야 합니다. 위치 데이터는 이 자습서의 명명된 섀도에 저장되므로 `thingIndexingMode`를 `REGISTRY`(최소 요구 사항에서)로 설정하고 `namedShadowIndexingMode`를 `ON`으로 설정하고 위치 데이터를 구성에 추가합니다. 이 예시에서는 명명된 섀도의 이름과 섀도의 위치 데이터 경로를 `filter`에 추가해야 합니다.

1. 명령을 실행하여 위치 인덱싱에 대한 인덱싱 구성을 업데이트합니다.

```
aws iot update-indexing-configuration --cli-input-json '{
"thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY",
"thingConnectivityIndexingMode": "OFF",
"deviceDefenderIndexingMode": "OFF",
"namedShadowIndexingMode": "ON",
"filter": {
    "namedShadowNames": ["Bike1-shadow"],
    "geoLocations":[{
        "name":"shadow.name.Bike1-shadow.reported.coordinates"
    }]
},
"customFields": [
{ "name":"attributes.battery",
"type":"Number"}] } }'
```

   명령은 출력을 생성하지 않습니다. 업데이트가 완료될 때까지 잠시 기다려야 할 수 있습니다. 사물 인덱스 상태를 확인하려면 describe-index CLI 명령을 실행합니다. `indexStatus` 가 `ACTIVE`로 표시되면 사물 인덱싱 업데이트가 완료된 것입니다.

2. 명령을 실행하여 인덱싱 구성을 확인합니다. 이 단계는 선택 사항입니다.

```
aws iot get-indexing-configuration
```

   출력은 다음과 같습니다.

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY",
        "thingConnectivityIndexingMode": "OFF",
        "deviceDefenderIndexingMode": "OFF",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
```

```
            {
                "name": "shadow.name.*.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
                "name": "registry.thingGroupNames",
                "type": "String"
            },
            {
                "name": "shadow.name.*.version",
                "type": "Number"
            },
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "thingId",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "attributes.battery",
                "type": "Number"
            }
        ],
        "filter": {
            "namedShadowNames": [
                "Bike1-shadow"
            ],
            "geoLocations": [
                {
                    "name": "shadow.name.Bike1-shadow.reported.coordinates",
                    "order": "LatLon"
                }
```

```
            ]
        }
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

지오쿼리 실행

이제 위치 데이터를 포함하도록 사물 인덱싱 구성을 업데이트했습니다. 몇 가지 지오쿼리를 생성하고 실행하여 원하는 검색 결과를 얻을 수 있는지 확인합니다. 지오쿼리는 쿼리 구문을 따라야 합니다. ???에서 몇 가지 유용한 예제 지오쿼리를 찾을 수 있습니다.

다음 예제 명령에서는 지오쿼리 `shadow.name.Bike1-shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km`를 사용하여 좌표(47.6204,-122.3491)를 사용하여 중심점에서 15.5km의 반경 거리 내에 있는 디바이스를 검색합니다.

```
aws iot search-index --query-string "shadow.name.Bike1-
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

좌표 'lat': 47.6153, 'lon': -122.3333에 중심점에서 15.5km 거리 내에 있는 디바이스가 있으므로 출력에서 이 디바이스(Bike-1)를 볼 수 있습니다. 출력은 다음과 같습니다.

```
{
    "things": [
        {
            "thingName": "Bike-1",
            "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df",
            "attributes": {
                "acqDate": "06/09/23",
                "battery": "35",
                "model": "OEM-2302-12"
            },
            "shadow": "{\"reported\":{\"coordinates\":{\"lat\":47.6153,\"lon
\":-122.3333}},\"metadata\":{\"reported\":{\"coordinates\":{\"lat\":{\"timestamp
\":1699572906},\"lon\":{\"timestamp\":1699572906}}}},\"hasDelta\":false,\"version\":1}"
        }
    ]
```

```
}
```

자세한 내용은 ??? 단원을 참조하십시오.

플릿 지표

플릿 지표는 디바이스의 데이터를 인덱싱, 검색 및 집계할 수 있는 관리형 서비스인 플릿 인덱싱의 기능입니다 AWS IoT. 플릿 지표를 사용하면 플릿 디바이스의 연결 해제율 또는 지정된 기간의 평균 배터리 수준 변경 검토를 포함하여 시간 경과에 따라 CloudWatch에서 플릿 디바이스의 집계 상태를 모니터링할 수 있습니다.

플릿 지표를 사용하면 추세 분석 및 경보 생성을 위한 지표로 CloudWatch에 지속적으로 결과를 내보내는 집계 쿼리를 작성할 수 있습니다. 모니터링 태스크의 경우 다양한 집계 유형(통계(Statistics), 카디널리티(Cardinality) 및 백분위수(Percentile))의 집계 쿼리를 지정할 수 있습니다. 모든 집계 쿼리를 저장하여 나중에 재사용할 플릿 지표를 생성할 수 있습니다.

시작하기 자습서

이 튜토리얼에서는 잠재적인 이상을 감지하기 위해 센서의 온도를 모니터링하는 플릿 지표를 생성합니다. 플릿 지표를 생성할 때 온도가 화씨 80도를 초과하는 센서의 수를 감지하는 집계 쿼리를 정의합니다. 쿼리가 60초마다 실행되도록 지정하면 쿼리 결과가 CloudWatch로 내보내지며, 여기에서 잠재적인 고온 위험이 있는 센서의 수를 보고 경보를 설정할 수 있습니다. 이 자습서를 완료하려면 AWS CLI를 사용합니다.

이 자습서에서는 다음을 수행하는 방법을 알아봅니다.

- 설정
- 플릿 지표 생성
- CloudWatch에서 지표 보기
- 리소스 정리

이 자습서는 완료하는 데 약 15분이 소요됩니다.

사전 조건

- AWS CLI의 최신 버전을 설치하려면
- 집계 데이터 쿼리 익히기
- Amazon CloudWatch 지표 사용 익히기

설정

플릿 지표를 사용하려면 플릿 인덱싱을 사용하도록 설정합니다. 지정된 데이터 원본과 관련 구성을 사용하여 사물 또는 사물 그룹에 대한 플릿 인덱싱을 사용하도록 설정하려면 사물 인덱싱 관리 및 사물 그룹 인덱싱 관리의 지침을 따르세요.

설정

1. 다음 명령을 실행하여 플릿 인덱싱을 사용하도록 설정하고 검색할 데이터 원본을 지정합니다.

```
aws iot update-indexing-configuration \
--thing-indexing-configuration
 "thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.temperature,type=Num
{name=attributes.rackId,type=String},
{name=attributes.stateNormal,type=Boolean}],thingConnectivityIndexingMode=STATUS" \
```

   위의 CLI 명령 예시는 AWS_Things 인덱스를 사용하여 레지스트리 데이터, 섀도우 데이터 및 사물 연결 상태 검색을 지원하도록 플릿 인덱싱을 사용하도록 설정합니다.

   구성 변경을 완료하는 데 몇 분 정도 걸릴 수 있습니다. 플릿 지표를 생성하기 전에 플릿 인덱싱이 사용하도록 설정되어 있는지 확인합니다.

   플릿 인덱싱이 사용하도록 설정되었는지 확인하려면 다음 CLI 명령을 실행합니다.

```
aws --region us-east-1 iot describe-index --index-name "AWS_Things"
```

   자세한 내용은 사물 인덱싱 사용을 참조하세요.

2. 다음 bash 스크립트를 실행하여 10개의 사물을 생성하고 설명합니다.

```
# Bash script. Type `bash` before running in other shells.

Temperatures=(70 71 72 73 74 75 47 97 98 99)
Racks=(Rack1 Rack1 Rack2 Rack2 Rack3 Rack4 Rack5 Rack6 Rack6 Rack6)
IsNormal=(true true true true true true false false false false)

for ((i=0; i < 10; i++))
do
  thing=$(aws iot create-thing --thing-name "TempSensor$i" --attribute-
payload attributes="{temperature=${Temperatures[@]:$i:1},rackId=${Racks[@]:
$i:1},stateNormal=${IsNormal[@]:$i:1}}")
  aws iot describe-thing --thing-name "TempSensor$i"
```

```
done
```

이 스크립트는 10개의 센서를 나타내는 10개의 사물을 생성합니다. 각 사물에는 다음 표에 설명된 대로 `temperature`, `rackId` 및 `stateNormal` 속성이 있습니다.

속성	데이터 유형	설명
`temperature`	숫자	화씨의 온도 값
`rackId`	문자열	센서가 포함된 서버 랙의 ID
`stateNormal`	부울	센서의 온도 값이 정상인지 여부

이 스크립트의 출력에는 10개의 JSON 파일이 포함됩니다. JSON 파일 중 하나가 다음과 같습니다.

```
{
    "version": 1,
    "thingName": "TempSensor0",
    "defaultClientId": "TempSensor0",
    "attributes": {
        "rackId": "Rack1",
        "stateNormal": "true",
        "temperature": "70"
    },
    "thingArn": "arn:aws:iot:region:account:thing/TempSensor0",
    "thingId": "example-thing-id"
}
```

자세한 내용은 사물 생성을 참조하세요.

플릿 지표 생성

플릿 지표를 생성하는 방법

1. 다음 명령을 실행하여 *high_temp_FM*이라는 플릿 지표를 생성합니다. CloudWatch에서 온도가 화씨 80도를 초과하는 센서 수를 모니터링하는 플릿 지표를 생성합니다.

```
aws iot create-fleet-metric --metric-name "high_temp_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature >80" --period 60 --aggregation-
field "attributes.temperature" --aggregation-type name=Statistics,values=count
```

--metric-name

데이터 형식: 문자열. `--metric-name` 파라미터는 플릿 지표 이름을 지정합니다. 이 예에서는 high_temp_FM이라는 플릿 지표를 생성하고 있습니다.

--query-string

데이터 형식: 문자열. `--query-string` 파라미터는 쿼리 문자열을 지정합니다. 이 예에서 쿼리 문자열은 이름이 TempSensor로 시작하고 온도가 화씨 80도 이상인 모든 사물을 쿼리하는 것을 의미합니다. 자세한 내용은 쿼리 구문을 참조하세요.

--period

데이터 형식: 정수. `--period` 파라미터는 집계된 데이터를 검색하는 시간을 초 단위로 지정합니다. 이 예에서는 생성 중인 플릿 지표가 60초마다 집계된 데이터를 검색하도록 지정합니다.

--aggregation-field

데이터 형식: 문자열. `--aggregation-field` 파라미터는 평가할 속성을 지정합니다. 이 예에서는 온도 속성을 평가해야 합니다.

--aggregation-type

`--aggregation-type` 파라미터는 플릿 지표에 표시할 통계 요약을 지정합니다. 모니터링 태스크의 경우 다양한 집계 유형(통계(Statistics), 카디널리티(Cardinality) 및 백분위 수(Percentile))의 집계 쿼리 속성을 사용자 지정할 수 있습니다. 이 예에서 집계 유형과 통계에 대해 개수를 지정하여 쿼리와 일치하는 속성이 있는 디바이스 수를 반환합니다. 즉, 이름이 TempSensor로 시작하고 온도가 화씨 80도 이상인 디바이스 수를 반환합니다. 자세한 내용은 집계 데이터 쿼리를 참조하세요.

이 명령의 출력은 다음과 같습니다.

```
{
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "metricName": "high_temp_FM"
```

```
}
```

> **Note**
>
> CloudWatch에 데이터 포인트가 표시되는 데 시간이 걸릴 수 있습니다.

플릿 지표를 생성하는 방법에 대해 자세히 알아보려면 플릿 지표 관리를 참조하세요.

플릿 지표를 생성할 수 없는 경우 플릿 지표 문제 해결을 참조하세요.

2. (선택 사항) 다음 명령을 실행하여 high_temp_FM이라는 플릿 지표를 설명합니다.

```
aws iot describe-fleet-metric --metric-name "high_temp_FM"
```

이 명령의 출력은 다음과 같습니다.

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625249775.834,
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625249775.834,
    "metricName": "high_temp_FM"
}
```

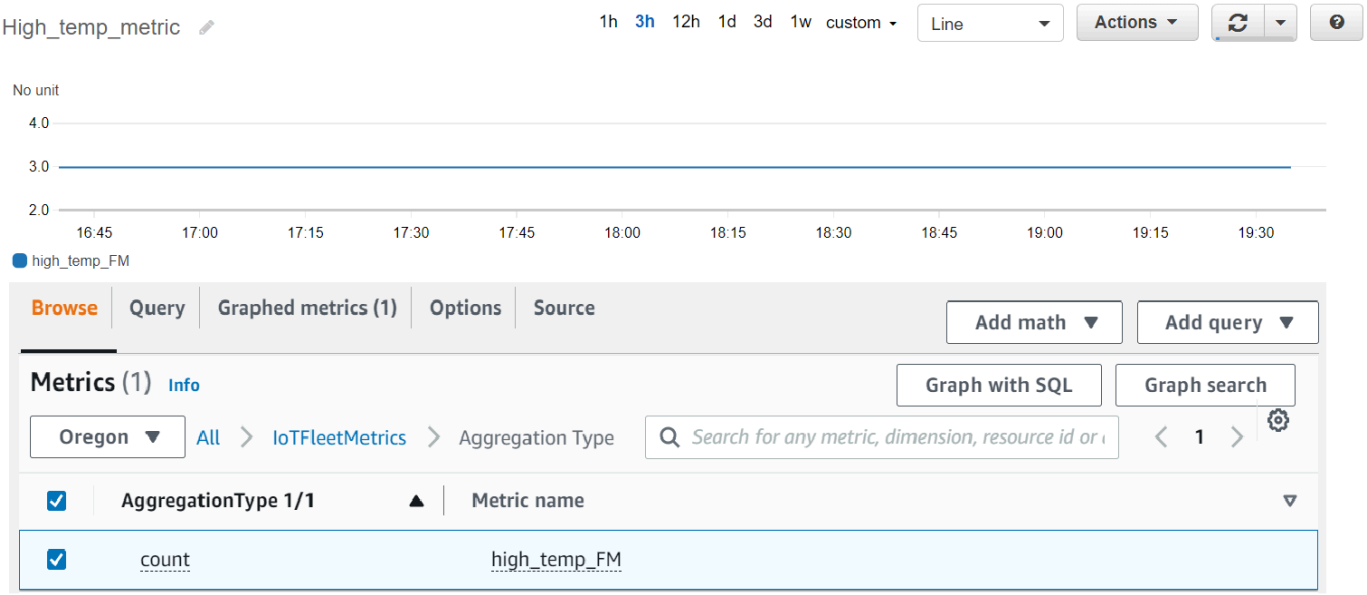
CloudWatch에서 플릿 지표 보기

플릿 지표를 생성한 후 CloudWatch에서 지표 데이터를 볼 수 있습니다. 이 튜토리얼에서는 이름이 TempSensor로 시작하고 온도가 화씨 80도 이상인 센서의 수를 보여주는 지표를 볼 수 있습니다.

CloudWatch에서 데이터 포인트를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 왼쪽 패널의 CloudWatch 메뉴에서 지표를 선택하여 하위 메뉴를 확장한 다음 모든 지표를 선택합니다. 이렇게 하면 위쪽 절반에 그래프가 표시되고 아래쪽 절반에 4개의 탭 섹션이 있는 페이지가 열립니다.
3. 첫 번째 탭 섹션인 모든 지표에는 그룹에서 볼 수 있는 모든 지표가 나열됩니다. IoTFleetMetrics를 선택합니다. 여기에는 모든 플릿 지표가 포함됩니다.
4. 모든 지표(All metrics) 탭의 집계 유형(Aggregation type) 섹션에서 집계 유형(Aggregation type)을 선택하여 생성한 모든 플릿 지표를 확인합니다.
5. 플릿 지표를 선택하여 집계 유형(Aggregation type) 섹션 왼쪽에 그래프를 표시합니다. 지표 이름 왼쪽에 값 ## 값이 표시되며 이는 이 자습서의 플릿 지표 생성 섹션에서 지정한 집계 유형의 값입니다.
6. 모든 지표(All metrics) 탭의 오른쪽에 있는 그래프로 표시된 지표(Graphed metrics)라는 두 번째 탭을 선택하여 이전 단계에서 선택한 플릿 지표를 확인합니다.

   다음과 같이 화씨 80도보다 높은 온도의 센서 수를 표시하는 그래프를 볼 수 있어야 합니다.



> **Note**
>
> CloudWatch의 기간(Period) 속성은 기본적으로 5분으로 설정됩니다. CloudWatch에 표시되는 데이터 포인트 사이의 시간 간격입니다. 필요에 따라 기간(Period) 설정을 변경할 수 있습니다.

7. (선택 사항) 지표 경보를 설정할 수 있습니다.

    1. 왼쪽 패널의 CloudWatch 메뉴에서 경보를 선택하여 하위 메뉴를 확장한 다음 모든 경보를 선택합니다.
    2. 경보(Alarms) 페이지의 오른쪽 상단 모서리에서 경보 생성(Create alarm)을 선택합니다. 콘솔에서 경보 생성(Create alarm) 지침에 따라 경보를 생성합니다. 자세한 내용은 Amazon CloudWatch 경보 사용을 참조하세요.

자세한 내용은 Amazon CloudWatch 지표 사용을 참조하세요.

CloudWatch에서 데이터 포인트를 볼 수 없는 경우 플릿 지표 문제 해결을 참조하세요.

정리

플릿 지표를 삭제하려면

delete-fleet-metric CLI 명령을 사용하여 플릿 지표를 삭제합니다.

high_temp_FM이라는 플릿 지표를 삭제하려면 다음 명령을 실행합니다.

```
aws iot delete-fleet-metric --metric-name "high_temp_FM"
```

사물을 정리하려면

delete-thing CLI 명령을 사용하여 사물을 삭제합니다.

생성한 10개의 사물을 삭제하려면 다음 스크립트를 실행합니다.

```
# Bash script. Type `bash` before running in other shells.

for ((i=0; i < 10; i++))
do
  thing=$(aws iot delete-thing --thing-name "TempSensor$i")
done
```

CloudWatch에서 지표를 정리하려면

CloudWatch는 지표 삭제를 지원하지 않습니다. 지표는 보존 일정에 따라 만료됩니다. 자세한 내용은 Amazon CloudWatch 지표 사용을 참조하세요.

플릿 지표 관리

이 주제에서는 AWS IoT 콘솔을 사용하고 플릿 지표 AWS CLI 를 관리하는 방법을 보여줍니다.

주제

- 플릿 지표 관리(콘솔)
- 플릿 지표 관리(CLI)
- IoT 리소스의 태깅 권한 부여

플릿 지표 관리(콘솔)

다음 섹션에서는 AWS IoT 콘솔을 사용하여 플릿 지표를 관리하는 방법을 보여줍니다. 플릿 지표를 생성하기 전에 연결된 데이터 원본 및 구성으로 플릿 인덱싱을 사용하도록 설정했는지 확인합니다.

플릿 인덱싱 사용 설정

플릿 인덱싱을 이미 사용하도록 설정한 경우 이 섹션을 건너뜁니다.

플릿 인덱싱을 사용하도록 설정하지 않은 경우 다음 지침을 따릅니다.

1. https://console.aws.amazon.com/iot/ AWS IoT 콘솔을 엽니다.
2. AWS IoT 메뉴에서 설정을 선택합니다.
3. 자세한 설정을 보려면 설정(Settings) 페이지에서 플릿 인덱싱(Fleet indexing) 섹션까지 아래로 스크롤합니다.
4. 플릿 인덱싱 설정을 업데이트하려면 플릿 인덱싱(Fleet indexing) 섹션 오른쪽에서 인덱싱 관리(Manage indexing)를 선택합니다.
5. 플릿 인덱싱 관리(Manage fleet indexing)페이지에서 필요에 따라 플릿 인덱싱 설정을 업데이트합니다.
   - 구성

     사물 인덱싱을 설정하려면 사물 인덱싱(Thing indexing)을 켜고 인덱싱할 데이터 원본을 선택합니다.

     사물 그룹 인덱싱을 설정하려면 사물 그룹 인덱싱(Thing group indexing)을 켭니다.
   - 집계를 위한 사용자 지정 필드(Custom fields for aggregation) - 선택 사항

     사용자 지정 필드는 필드 이름 및 필드 유형 페어의 목록입니다.

사용자 정의 필드 페어를 추가하려면 새 필드 추가(Add new field)를 선택합니다. `attributes.temperature`와 같은 사용자 정의 필드 이름을 입력한 다음 필드 유형(Field type) 메뉴에서 필드 유형을 선택합니다. 사용자 정의 필드 이름은 `attributes.`로 시작하고 사물 집계 쿼리를 실행하기 위한 속성으로 저장됩니다.

설정을 업데이트하고 저장하려면 업데이트(Update)를 선택합니다.

플릿 지표 생성

1. https://console.aws.amazon.com/iot/ AWS IoT 콘솔을 엽니다.
2. AWS IoT 메뉴에서 관리를 선택한 다음 플릿 지표를 선택합니다.
3. 플릿 지표(Fleet metrics) 페이지에서 플릿 지표 생성(Create fleet metric)을 선택하고 생성 단계를 완료합니다.
4. 1단계 플릿 지표 구성(Configure fleet metrics)
   - 쿼리(Query) 섹션에서 쿼리 문자열을 입력하여 집계 검색을 수행할 사물 또는 사물 그룹을 지정합니다. 쿼리 문자열은 속성과 값으로 구성됩니다. 속성(Properties)에서 원하는 속성을 선택합니다. 원하는 속성이 목록에 없는 경우 필드에 속성을 입력합니다. `:` 다음에 값을 입력합니다. 예제 쿼리 문자열은 `thingName:TempSensor*`와 같습니다. 입력한 각 쿼리 문자열에 대해 Enter 키를 누릅니다. 여러 쿼리 문자열을 입력하는 경우 and, or, and not 또는 or not을 선택하여 관계를 지정합니다.
   - 보고서 속성의 해당 목록에서 인덱스 이름, 집계 유형 및 집계 필드를 선택합니다. 그런 다음 데이터 선택(Select data)에서 집계할 데이터를 선택합니다. 여기에서 여러 데이터 값을 선택할 수 있습니다.
   - 다음을 선택합니다.
5. 2단계 플릿 지표 속성 지정
   - 플릿 지표 이름(Fleet metric name) 필드에 생성 중인 플릿 지표의 이름을 입력합니다.
   - 설명 - 선택 사항(Description - optional) 필드에 생성 중인 플릿 지표에 대한 설명을 입력합니다. 이 필드는 선택 사항입니다.
   - 시간(Hours) 및 분(Minutes) 필드에 플릿 지표에서 CloudWatch로 데이터를 내보낼 시간(빈도)을 입력합니다.
   - 다음을 선택합니다.
6. 3단계에서 검토 및 생성(Review and create)
   - 1단계 및 2단계의 설정을 검토합니다. 설정을 편집하려면 편집(Edit)을 선택합니다.

- 플릿 지표 생성(Create fleet metric)을 선택합니다.

성공적으로 생성되면 플릿 지표가 플릿 지표(Fleet metric) 페이지에 나열됩니다.

플릿 지표 업데이트

1. 플릿 지표 페이지에서 업데이트할 플릿 지표를 선택합니다.
2. 플릿 지표 세부 정보(Details) 탭에서 편집(Edit)을 선택합니다. 그러면 세 단계 중 하나에서 플릿 지표를 업데이트할 수 있는 생성 단계가 열립니다.
3. 플릿 지표 업데이트를 완료한 후 플릿 지표 업데이트(Update fleet metric)를 선택합니다.

플릿 지표 삭제

1. 플릿 지표 페이지에서 삭제할 플릿 지표를 선택합니다.
2. 플릿 지표에 대한 세부 정보를 보여 주는 다음 페이지에서 삭제(Delete)를 선택합니다.
3. 대화 상자에 플릿 지표의 이름을 입력하여 삭제를 확인합니다.
4. 삭제를 선택합니다. 플릿 지표가 영구적으로 삭제됩니다.

플릿 지표 관리(CLI)

다음 섹션에서는를 사용하여 플릿 지표 AWS CLI 를 관리하는 방법을 보여줍니다. 플릿 지표를 생성하기 전에 연결된 데이터 원본 및 구성으로 플릿 인덱싱을 사용하도록 설정했는지 확인합니다. 사물 또는 사물 그룹에 대한 집합 인덱싱을 사용하도록 설정하려면 사물 인덱싱 관리 및 사물 그룹 인덱싱 관리의 지침을 따르세요.

플릿 지표 생성

create-fleet-metric CLI 명령을 사용하여 플릿 지표를 생성할 수 있습니다.

```
aws iot create-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 60 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum
```

이 명령의 출력에는 플릿 지표의 이름과 Amazon 리소스 이름(ARN)이 포함됩니다. 출력은 다음과 같습니다.

```
{
```

```
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "metricName": "YourFleetMetricName"
}
```

플릿 지표 나열

list-fleet-metric CLI 명령을 사용하여 계정의 모든 플릿 지표를 나열할 수 있습니다.

```
aws iot list-fleet-metrics
```

이 명령의 출력에는 모든 플릿 지표가 포함됩니다. 출력은 다음과 같습니다.

```
{
    "fleetMetrics": [
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric1",
            "metricName": "YourFleetMetric1"
        },
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric2",
            "metricName": "YourFleetMetric2"
        }
    ]
}
```

플릿 지표 설명

describe-fleet-metric CLI 명령을 사용하여 플릿 지표에 대한 자세한 정보를 표시할 수 있습니다.

```
aws iot describe-fleet-metric --metric-name "YourFleetMetricName"
```

명령의 출력에는 지정된 플릿 지표에 대한 자세한 정보가 포함됩니다. 출력은 다음과 같습니다.

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625790642.355,
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
```

```
    "version": 1,
    "aggregationType": {
        "values": [
            "sum"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625790642.355,
    "metricName": "YourFleetMetricName"
}
```

플릿 지표 업데이트

update-fleet-metric CLI 명령을 사용하여 플릿 지표를 업데이트할 수 있습니다.

```
aws iot update-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 120 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum,count --index-name AWS_Things
```

update-fleet-metric 명령은 출력을 생성하지 않습니다. describe-fleet-metric CLI 명령을 사용하여 결과를 확인할 수 있습니다.

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625792300.881,
    "queryString": "*",
    "period": 120,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 2,
    "aggregationType": {
        "values": [
            "sum",
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625792300.881,
    "metricName": "YourFleetMetricName"
}
```

플릿 지표 삭제

delete-fleet-metric CLI 명령을 사용하여 플릿 지표를 삭제합니다.

```
aws iot delete-fleet-metric --metric-name "YourFleetMetricName"
```

이 명령은 삭제에 성공하거나 존재하지 않는 플릿 지표를 지정하는 경우 출력을 생성하지 않습니다.

자세한 내용은 플릿 지표 문제 해결을 참조하세요.

IoT 리소스의 태깅 권한 부여

생성, 수정 또는 사용할 수 있는 플릿 지표를 더 잘 제어하려면 플릿 지표에 태그를 연결할 수 있습니다.

AWS Management Console 또는를 사용하여 생성한 플릿 지표에 태그를 AWS CLI지정하려면 IAM 정책에 `iot:TagResource` 작업을 포함하여 사용자에게 권한을 부여해야 합니다. IAM 정책에 `iot:TagResource`가 포함되지 않은 경우 태그가 있는 플릿 지표를 생성하는 모든 작업에 `AccessDeniedException` 오류가 반환됩니다.

리소스 태그 지정에 대한 추가적인 일반 정보는 AWS IoT 리소스 태그 지정을 참조하세요.

IAM 정책 예시

플릿 지표를 생성할 때 태그 지정 권한을 부여하는 다음 IAM 정책 예제를 참조하세요.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Action": [
    "iot:TagResource"
   ],
   "Effect": "Allow",
   "Resource": [
    "arn:aws:iot:*:*:fleetmetric/*"
   ]
  },
  {
   "Action": [
    "iot:CreateFleetMetric"
   ],
   "Effect": "Allow",
```

```
    "Resource": [
     "arn:aws:iot:*:*:index/*",
     "arn:aws:iot:*:*:fleetmetric/*"
    ]
   }
  ]
}
```

자세한 내용은 Actions, resources, and condition keys for AWS IoT 섹션을 참조하세요.

MQTT 기반 파일 전송

파일을 관리하고 플릿의 AWS IoT 디바이스로 전송하는 데 사용할 수 있는 한 가지 옵션은 MQTT 기반 파일 전송입니다. AWS 클라우드에서이 기능을 사용하면 여러 파일이 포함된 스트림을 생성하고, 스트림 데이터(파일 목록 및 설명)를 업데이트하고, 스트림 데이터를 가져오는 등의 작업을 수행할 수 있습니다. AWS IoT MQTT 기반 파일 전송은 JSON 또는 CBOR의 요청 및 응답 메시지를 지원하는 MQTT 프로토콜을 사용하여 IoT 디바이스로 작은 블록으로 데이터를 전송할 수 있습니다.

를 사용하여 IoT 디바이스와 데이터를 주고받는 방법에 대한 자세한 내용은 섹션을 AWS IoT참조하세요디바이스를에 연결 AWS IoT.

주제

- 스트림이란 무엇입니까?
- AWS 클라우드에서 스트림 관리
- 디바이스에서 AWS IoT MQTT 기반 파일 전송 사용
- FreeRTOS OTA의 예제 사용 사례

스트림이란 무엇입니까?

에서 AWS IoT스트림은 IoT 디바이스로 전송할 수 있는 파일 목록을 추상화하는 공개적으로 주소를 지정할 수 있는 리소스입니다. 일반적인 스트림에는 다음 정보가 포함됩니다.

- 주어진 시간에 스트림을 고유하게 식별하는 Amazon 리소스 이름(ARN)입니다. 이 ARN 에는 `arn:`*`partition`*`:iot:`*`region`*`:`*`account-ID`*`:stream/`*`stream ID` 패턴이 있습니다.
- 스트림을 식별하고 AWS Command Line Interface () 또는 SDK 명령에서 사용되는(일반적으로 필요한AWS CLI) 스트림 ID입니다.
- 스트림 리소스에 대한 설명을 제공하는 스트림 설명입니다.
- 스트림의 특정 버전을 식별하는 스트림 버전입니다. 디바이스에서 데이터 전송을 시작하기 직전에 스트림 데이터를 수정할 수 있으므로 디바이스에서 스트림 버전을 사용하여 일관성 검사를 시행할 수 있습니다.
- 디바이스에 전송할 수 있는 파일 목록입니다. 목록의 각 파일에 대해 스트림은 파일 ID, 파일 크기 및 파일 주소 정보(예: Amazon S3 버킷 이름, 객체 키 및 객체 버전)를 기록합니다.
- MQTT 기반 파일 전송에 데이터 스토리지에 저장된 스트림 파일을 읽을 수 있는 권한을 부여하는 AWS IoT AWS Identity and Access Management (IAM) 역할입니다.

AWS IoT MQTT 기반 파일 전송은 디바이스가 클라우드에서 데이터를 전송할 수 있도록 다음과 같은 기능을 제공합니다. AWS

- MQTT 프로토콜을 사용한 데이터 전송
- JSON 또는 CBOR 형식을 지원합니다.
- 스트림을 설명하는 기능(`DescribeStream` API)을 사용하여 스트림 파일 목록, 스트림 버전 및 관련 정보를 얻을 수 있습니다.
- 작은 블록으로 데이터를 전송할 수 있는 기능(`GetStream` API)을 사용하여 하드웨어 제약 조건이 있는 디바이스가 블록을 수신할 수 있도록 합니다.
- 요청당 동적 블록 크기를 지원하여 메모리 용량이 다른 디바이스를 지원합니다.
- 여러 디바이스가 동일한 스트림 파일에서 데이터 블록을 요청할 때 동시 스트리밍 요청을 최적화합니다.
- 스트림 파일을 위한 데이터 스토리지로서 Amazon S3를 사용합니다.
- AWS IoT MQTT 기반 파일 전송에서 CloudWatch로의 데이터 전송 로그 게시를 지원합니다.

MQTT 기반 파일 전송 할당량에 대한 자세한 내용은 AWS 일반 참조의 AWS IoT Core 서비스 할당량을 참조하세요.

AWS 클라우드에서 스트림 관리

AWS IoT 는 AWS 클라우드에서 스트림을 관리하는 데 사용할 수 있는 AWS SDK 및 AWS CLI 명령을 제공합니다. 이러한 명령을 사용하여 다음을 수행할 수 있습니다.

- 스트림을 생성합니다. CLI/SDK
- 해당 정보를 가져오기 위한 스트림을 설명합니다. CLI/SDK
- 에서 스트림을 나열합니다 AWS 계정. CLI/SDK
- 스트림의 파일 목록 또는 스트림 설명을 업데이트합니다. CLI/SDK
- 스트림을 삭제합니다. CLI/SDK

> **Note**
>
> 현재, 스트림을 AWS Management Console에서 볼 수 없습니다. 스트림을 관리하려면 AWS CLI 또는 AWS SDK를 사용해야 합니다 AWS IoT. 또한 Embedded C SDK는 MQTT 기반 파일 전송을 지원하는 유일한 SDK입니다.

디바이스에서 AWS IoT MQTT 기반 파일 전송을 사용하기 전에 다음 섹션에 표시된 대로 디바이스에 대해 다음 조건이 충족되는지 확인해야 합니다.

- MQTT를 통해 데이터를 전송하는 데 필요한 올바른 권한을 반영한 정책입니다.
- 디바이스는 AWS IoT Device Gateway에 연결할 수 있습니다.
- 리소스에 태그를 지정할 수 있다는 설명이 명시된 정책입니다. 태그를 사용하여 `CreateStream`을 호출하는 경우 `iot:TagResource`는 필수 항목입니다.

디바이스에서 AWS IoT MQTT 기반 파일 전송을 사용하기 전에 다음 섹션의 단계에 따라 디바이스가 제대로 인증되고 Device Gateway에 AWS IoT 연결할 수 있는지 확인해야 합니다.

디바이스에 권한 부여

AWS IoT 정책 생성의 단계를 따라 디바이스 정책을 생성하거나 기존 디바이스 정책을 사용할 수 있습니다. 디바이스와 연결된 인증서에 정책을 연결하고 디바이스 정책에 다음 권한을 추가합니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        }
    ]
}
```

디바이스를에 연결 AWS IoT

를 연결하려면 AWS IoT MQTT 기반 파일 전송을 사용하는 디바이스가 필요합니다 AWS IoT. AWS IoT MQTT 기반 파일 전송은 AWS 클라우드 AWS IoT 의와 통합되므로 디바이스는 AWS IoT 데이터 영역의 엔드포인트에 직접 연결해야 합니다.

> **Note**
>
> AWS IoT 데이터 영역의 엔드포인트는 AWS 계정 및 리전에 따라 다릅니다. 및 디바이스가 등록된 리전 AWS 계정 에 엔드포인트를 사용해야 합니다 AWS IoT.

자세한 정보는 에 연결 AWS IoT Core을 참조하세요.

TagResource 사용

`CreateStream` API 작업은 1개 이상의 대용량 파일을 MQTT를 통해 청크 단위로 전송하는 스트림을 생성합니다.

`CreateStream` API 호출을 위해 다음 권한이 필요합니다.

- `iot:CreateStream`
- `iot:TagResource`(`CreateStream`에 태그가 있는 경우)

이 두 권한을 지원하는 정책은 다음과 같습니다.

```
{
```

```
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iot:CreateStream",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:stream/streamId"
    }
}
```

`iot:TagResource` 정책 설명 작업은 사용자가 적절한 권한 없이 리소스에서 태그를 만들거나 업데이트할 수 없도록 하는 데 필요합니다. `iot:TagResource`의 구체적인 정책 설명 작업이 없고 요청에 태그가 포함된 경우 `CreateStream` API 호출이 `AccessDeniedException`을 반환합니다.

자세한 내용은 다음 링크를 참조하세요.

- CreateStream
- TagResource
- 태그

디바이스에서 AWS IoT MQTT 기반 파일 전송 사용

데이터 전송 프로세스를 시작하려면 디바이스가 최소한 스트림 ID를 포함하는 초기 데이터 세트를 수신해야 합니다. AWS IoT 작업을(를) 사용하여 작업 문서에 초기 데이터 세트를 포함하여 디바이스에 대한 데이터 전송 작업을 예약할 수 있습니다. 디바이스가 초기 데이터 세트를 수신하면 AWS IoT MQTT 기반 파일 전송과의 상호 작용을 시작해야 합니다. AWS IoT MQTT 기반 파일 전송과 데이터를 교환하려면 디바이스는 다음을 수행해야 합니다.

- MQTT 프로토콜을 사용하여 MQTT 기반 파일 전송 주제을(를) 구독합니다.
- 요청을 전송한 다음 MQTT 메시지를 사용하여 응답을 받을 때까지 기다립니다.

선택적으로 초기 데이터 세트에 스트림 파일 ID와 스트림 버전을 포함할 수 있습니다. ID를 얻기 위해 디바이스에서 `DescribeStream` 요청을 할 필요가 없기 때문에 스트림 파일 ID를 디바이스에 전송하면 디바이스의 펌웨어/소프트웨어 프로그래밍을 단순화할 수 있습니다. 디바이스는 스트림이 예기치 않게 업데이트된 경우 일관성 검사를 시행하기 위해 `GetStream` 요청에서 스트림 버전을 지정할 수 있습니다.

DescripbeStream을 사용하여 스트림 데이터 가져오기

AWS IoT MQTT 기반 파일 전송은 디바이스로 스트림 데이터를 전송하는 DescribeStream API를 제공합니다. 이 API에서 반환되는 스트림 데이터에는 스트림 ID, 스트림 버전, 스트림 설명 및 스트림 파일 목록이 포함되며 각 스트림 파일 목록에는 파일 ID와 파일 크기(바이트)가 있습니다. 이 정보를 사용하여 디바이스는 임의의 파일을 선택하여 데이터 전송 프로세스를 시작할 수 있습니다.

> **Note**
>
> 디바이스가 초기 데이터 세트에서 필요한 모든 스트림 파일 ID를 수신하는 경우 DescribeStream API를 사용할 필요가 없습니다.

DescribeStream 요청을 하려면 다음 단계를 따르세요.

1. "수락됨" 주제 필터 $aws/things/*ThingName*/streams/*StreamId*/description/json을 구독합니다.
2. "거부됨" 주제 필터 $aws/things/*ThingName*/streams/*StreamId*/rejected/json을 구독합니다.
3. 메시지를 $aws/things/*ThingName*/streams/*StreamId*/describe/json에 전송해 DescribeStream 요청을 게시합니다.
4. 요청이 수락된 경우 디바이스가 "수락됨" 주제 필터에 대한 DescribeStream 응답을 수신합니다.
5. 요청이 거부된 경우 디바이스는 "거부됨" 주제 필터에 대한 오류 응답을 수신합니다.

> **Note**
>
> 표시된 주제 및 주제 필터에서 json을 cbor로 바꾸면 디바이스가 JSON보다 더 간결한 CBOR 형식의 메시지를 수신합니다.

DescribeStream 요청

JSON 형식의 일반적인 DescribeStream 요청은 다음 예제와 같습니다.

```
{
```

```
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039"
}
```

- (선택 사항) “c“은 클라이언트 토큰 필드입니다.

  클라이언트 토큰은 64바이트보다 길면 안 됩니다. 클라이언트 토큰이 64바이트보다 길면 오류 응답과 InvalidRequest 오류 메시지가 발생합니다.

DescribeStream 응답

JSON 형식의 DescribeStream 응답은 다음 예제와 유사합니다.

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039",
    "s": 1,
    "d": "This is the description of stream ABC.",
    "r": [
        {
            "f": 0,
            "z": 131072
        },
        {
            "f": 1,
            "z": 51200
        }
    ]
}
```

- “c“은(는) 클라이언트 토큰 필드입니다. DescribeStream 요청에 제공된 경우에 반환됩니다. 클라이언트 토큰을 사용하여 응답을 요청과 연결합니다.
- “s“은(는) 정수로된 스트림 버전입니다. 이 버전을 사용하여 GetStream 요청에 대한 일관성 검사를 수행할 수 있습니다.
- “r“에는 스트림의 파일 목록이 포함되어 있습니다.
  - “f“은(는) 정수로 된 스트림 파일 ID입니다.
  - “z“은(는) 스트림 파일 크기(바이트 수)입니다.
- "d"에는 스트림에 대한 설명이 포함되어 있습니다.

스트림 파일에서 데이터 블록 가져오기

GetStream API를 사용하면 디바이스가 작은 데이터 블록에서 스트림 파일을 수신할 수 있으므로 대규모 블록 크기를 처리하는 데 제약이 있는 디바이스에서 사용할 수 있습니다. 전체 데이터 파일을 수신하려면 디바이스에서 모든 데이터 블록이 수신되고 처리될 때까지 여러 요청 및 응답을 전송하거나 수신해야 할 수 있습니다.

GetStream 요청

GetStream 요청을 하려면 다음 단계를 따르세요.

1. "수락된" 주제 필터 $aws/things/*ThingName*/streams/*StreamId*/data/json을 구독합니다.
2. "거부된" 주제 필터 $aws/things/*ThingName*/streams/*StreamId*/rejected/json을 구독합니다.
3. 주제 $aws/things/*ThingName*/streams/*StreamId*/get/json에 GetStream 요청을 게시합니다.
4. 요청이 수락되면 디바이스에 하나 이상의 "수락된" 주제 필터에 대한 GetStream 응답을 수신합니다. 각 응답 메시지에는 단일 블록에 대한 기본 정보와 데이터 페이로드가 포함됩니다.
5. 3 및 4단계를 반복하여 모든 데이터 블록을 수신합니다. 요청된 데이터 양이 128KB보다 큰 경우 이러한 단계를 반복해야 합니다. 여러 GetStream 요청을 통해 요청된 모든 데이터를 수신하도록 디바이스를 프로그래밍해야 합니다.
6. 요청이 거부되면 디바이스에서 "거부된" 주제 필터에 대한 오류 응답을 수신하게 됩니다.

> **Note**
>
> - 표시된 주제 및 주제 필터에서 "json"을 "cbor"로 바꾸면 디바이스에서 JSON보다 더 컴팩트한 CBOR 형식으로 메시지를 수신합니다.
> - AWS IoT MQTT 기반 파일 전송은 블록 크기를 128KB로 제한합니다. 128KB를 초과하는 블록에 대한 요청을 하면 요청이 실패합니다.
> - 총 크기가 128KB보다 큰 여러 블록에 대해 요청할 수 있습니다(예: 총 160KB의 데이터에 대해 각각 32KB의 5개 블록을 요청하는 경우). 이 경우 요청은 실패하지 않지만 요청된 모든 데이터를 수신하려면 디바이스에서 여러 요청을 해야 합니다. 이 서비스는 디바이스가 추가 요청을 할 때 추가 블록을 전송합니다. 이전 응답을 올바르게 받고 처리한 후에만 새 요청을 계속하는 것이 좋습니다.

- 요청된 데이터의 전체 크기에 관계없이, 블록이 수신되지 않거나 올바르게 수신되지 않을 때 재시도를 시작하도록 디바이스를 프로그래밍해야 합니다.

JSON 형식의 일반적인 GetStream 요청은 다음 예제와 같습니다.

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "s": 1,
    "f": 0,
    "l": 4096,
    "o": 2,
    "n": 100,
    "b": "..."
}
```

- [선택 사항] "c"은(는) 클라이언트 토큰 필드입니다.

  클라이언트 토큰은 64바이트보다 길면 안 됩니다. 클라이언트 토큰이 64바이트보다 길면 오류 응답과 InvalidRequest 오류 메시지가 발생합니다.
- [선택 사항] "s"은(는) 스트림 버전 필드(정수)입니다.

  MQTT 기반 파일 전송은 요청된 버전 및 클라우드의 최신 스트림 버전을 기반으로 일관성 검사를 적용합니다. 스트림 버전이 디바이스에서 전송된 경우 GetStream 요청이 클라우드의 최신 스트림 버전과 일치하지 않는 경우 서비스에서 오류 응답과 VersionMismatch 오류 메시지를 전송합니다. 일반적으로 디바이스는 초기 데이터 세트 또는 DescribeStream에 대한 응답에서 예상되는 (최신) 스트림 버전을 수신합니다.
- "f"은(는) 스트림 파일 ID(0~255 사이의 정수)입니다.

  스트림 파일 ID는 AWS CLI 또는 SDK를 사용하여 스트림을 생성하거나 업데이트할 때 필요합니다. 디바이스가 존재하지 않는 ID를 가진 스트림 파일을 요청하면 서비스에서 오류 응답과 ResourceNotFound 오류 메시지를 전송합니다.
- "l"은(는) 바이트 단위의 데이터 블록 크기(256~131,072 사이의 정수)입니다.

  비트맵 필드를 사용하여 GetStream 응답에서 반환될 스트림 파일 부분을 지정하는 방법에 대한 지침은 GetStream 요청을 위한 비트맵 빌드을(를) 참조하세요. 디바이스가 범위를 벗어나는 블록 크기를 지정하는 경우 서비스는 오류 응답과 BlockSizeOutOfBounds 오류 메시지를 전송합니다.
- [선택 사항] "o"은(는) 스트림 파일에 있는 블록의 오프셋(0~98,304 사이의 정수)입니다.

비트맵 필드를 사용하여 GetStream 응답에서 반환될 스트림 파일 부분을 지정하는 방법에 대한 지침은 GetStream 요청을 위한 비트맵 빌드을(를) 참조하세요. 최대 값 98,304는 24MB 스트림 파일 크기 제한과 최소 블록 크기의 256바이트를 기준으로 합니다. 지정하지 않은 경우 기본값은 0입니다.

- [선택 사항] "n"은(는) 요청된 블록 수(0~98,304 사이의 정수)입니다.

  "n" 필드는 (1) 요청된 블록 수 또는 (2) 비트맵 필드("b")가 사용되는 경우 비트맵 요청에 의해 반환될 블록 수에 대한 제한을 지정합니다. 이 두 번째 사용은 선택 사항입니다. 정의되지 않은 경우 기본값은 131,072/*DataBlockSize*입니다.

- [선택 사항] "b"은(는) 요청되는 블록을 나타내는 비트맵입니다.

  비트맵을 사용하면 디바이스에서 비연속 블록을 요청할 수 있으므로 오류 발생 후 재시도를 보다 편리하게 처리할 수 있습니다. 비트맵 필드를 사용하여 GetStream 응답에서 반환될 스트림 파일 부분을 지정하는 방법에 대한 지침은 GetStream 요청을 위한 비트맵 빌드을(를) 참조하세요. 이 필드의 경우 비트맵을, 16진수 표기법으로 비트맵의 값을 나타내는 문자열로 변환합니다. 비트맵은 12,288바이트 미만이어야 합니다.

> **Important**
>
> "n" 또는 "b"를 지정해야 합니다. 둘 다 지정하지 않으면 파일 크기가 131,072바이트(128KB) 미만인 경우 GetStream 요청이 유효하지 않을 수 있습니다.

GetStream 응답

JSON 형식의 GetStream 응답은 요청된 각 데이터 블록에 대해 이 예제와 같습니다.

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "f": 0,
    "l": 4096,
    "i": 2,
    "p": "..."
}
```

- "c"은(는) 클라이언트 토큰 필드입니다. GetStream 요청에 제공된 경우에 반환됩니다. 클라이언트 토큰을 사용하여 응답을 요청과 연결합니다.

- “f“은(는) 현재 데이터 블록 페이로드가 속한 스트림 파일의 ID입니다.
- “l“은(는) 데이터 블록 페이로드의 크기(바이트)입니다.
- “i“은(는) 페이로드에 포함된 데이터 블록의 ID입니다. 데이터 블록은 0부터 시작해서 번호가 매겨집니다.
- “p“에는 데이터 블록 페이로드가 포함됩니다. 이 필드는 Base64 인코딩으로 데이터 블록의 값을 나타내는 문자열입니다.

GetStream 요청을 위한 비트맵 빌드

GetStream 요청에서 비트맵 필드(b)를 사용하여 스트림 파일에서 비연속 블록을 가져올 수 있습니다. 이렇게 하면 RAM 용량이 제한된 디바이스에서 네트워크 전송 문제를 처리할 수 있습니다. 디바이스는 수신되지 않았거나 올바르게 수신되지 않은 블록만 요청할 수 있습니다. 비트맵은 반환될 스트림 파일의 블록을 결정합니다. 비트맵에서 1로 설정된 각 비트에 대해 스트림 파일의 해당 블록이 반환됩니다.

다음은 GetStream 요청에서 비트맵 및 해당 지원 필드를 지정하는 방법의 예입니다. 예를 들어, 256바이트(블록 크기)의 청크로 스트림 파일을 수신하려고 합니다. 256바이트의 각 블록은 0에서 시작하여 파일에서 위치를 지정하는 숫자를 갖는 것으로 생각합니다. 따라서 블록 0이 파일의 256바이트의 첫 번째 블록이고 블록 1이 두 번째 블록이 되는 방식입니다. 파일에서 블록 20, 21, 24 및 43을 요청하려고 합니다.

블록 오프셋

첫 번째 블록이 숫자 20이기 때문에 오프셋(o 필드)을 20으로 지정하여 비트맵의 공간을 절약할 수 있습니다.

블록 수

디바이스가 제한된 메모리 리소스로 처리할 수 있는 것보다 많은 블록을 수신하지 않도록 하려면 MQTT 기반 파일 전송에서 전송한 각 메시지에 반환해야 하는 최대 블록 수를 지정할 수 있습니다. 비트맵 자체가 이 블록 수보다 작게 지정하거나 MQTT 기반 파일 전송에서 전송한 응답 메시지의 총 크기가 GetStream 요청당 128KB의 서비스 제한보다 크게 되는 경우 이 값은 무시됩니다.

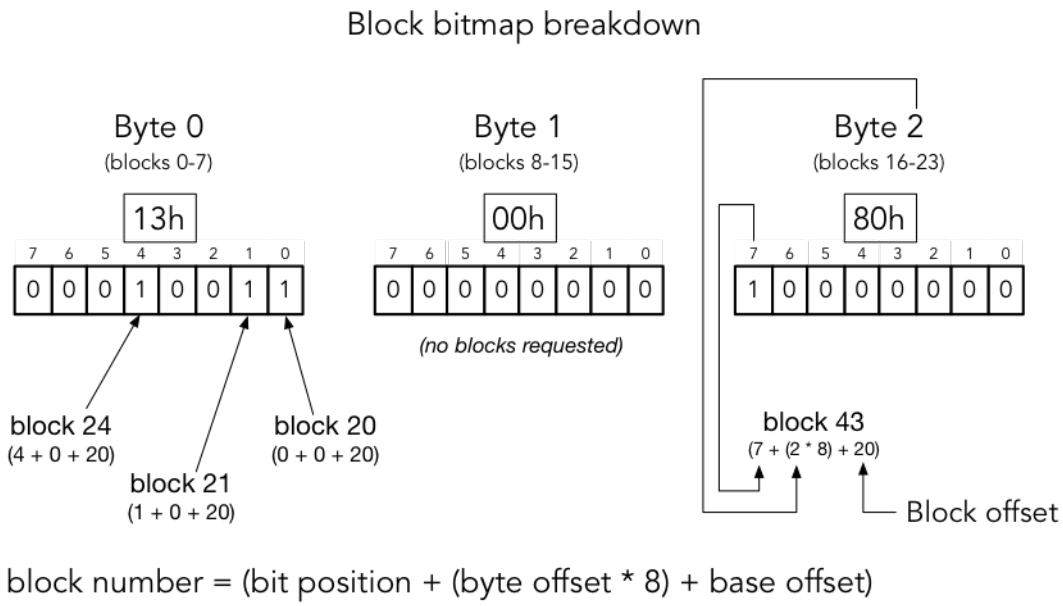
블록 비트맵

비트맵 자체는 16진법으로 표현된 부호 없는 바이트 배열이며 GetStream 요청에 숫자의 문자열 표현으로 포함됩니다. 그러나 이 문자열을 구성하기 위해 비트맵을 긴 비트 시퀀스(이진수)로 생각해봅시다. 이 시퀀스의 비트가 1로 설정되면 스트림 파일의 해당 블록이 디바이스로 다시 전송됩니다. 예를 들어 블록 20, 21, 24 및 43을 받기를 원하므로 비트 20, 21, 24 및 43을 비트맵에 설정해야 합니다. 블록 오프셋을 사용하여 공간을 절약할 수 있으므로 각 블록 번호에서 오프셋을 뺀 후 다음 예제와 같이 0, 1, 4 및 23비트를 설정하려고 합니다.

```
1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

한 번에 1바이트(8비트)를 취하면 일반적으로 "0b00010011", "0b00000000" 및 "0b10000000"으로 작성됩니다. 0비트는 첫 번째 바이트의 끝에 이진 표현으로 표시되고 마지막 바이트의 시작 부분에 23비트가 표시됩니다. 규칙을 알지 못하면 혼란스러울 수 있습니다. 첫 번째 바이트는 비트 7-0(순서대로), 두 번째 바이트는 15-8비트를 포함하고, 세 번째 바이트는 23-16비트 등을 포함합니다. 16진수 표기법에서는 "0x130080"으로 변환됩니다.

> **Tip**
>
> 표준 바이너리를 16진수 표기법으로 변환할 수 있습니다. 한 번에 네 개의 2진수를 가져와서 16진수로 변환하세요. 예를 들어 “0001"은 “1"이 되고 “0011"은 “3"이 됩니다.



이 모든 것이 포함된 `GetStream` 요청의 JSON 형식은 다음과 같습니다.

```
{
    "c" : "1",
    "s" : 1,
    "l" : 256,
    "f" : 1,
    "o" : 20,
    "n" : 32,
    "b" : "130080"
```

```
}
```

- “c“은(는) 클라이언트 토큰 필드입니다.
- 's'는 예상 스트림 버전입니다.
- “l“은(는) 데이터 블록 페이로드의 크기(바이트)입니다.
- 'f'는 소스 파일 인덱스의 ID입니다.
- 'o'는 블록 오프셋입니다.
- 'n'은 블록 수입니다.
- 'b'는 오프셋에서 시작하는 누락된 blockId 비트맵입니다. 이 값은 base64로 인코딩해야 합니다.

AWS IoT MQTT 기반 파일 전송으로 인한 오류 처리

`DescribeStream` 및 `GetStream` API 모두에 대해 디바이스로 전송되는 오류 응답에는 클라이언트 토큰, 오류 코드 및 오류 메시지가 포함됩니다. 일반적인 오류 응답은 다음 예제와 비슷합니다.

```
{
    "o": "BlockSizeOutOfBounds",
    "m": "The block size is out of bounds",
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380"
}
```

- "o"은(는) 오류가 발생한 이유를 나타내는 오류 코드입니다. 자세한 내용은 이 단원의 뒷부분에 나오는 오류 코드를 참조하세요.
- “m“은(는) 오류에 대한 세부 정보가 포함된 오류 메시지입니다.
- “c“은(는) 클라이언트 토큰 필드입니다. `DescribeStream` 요청에 제공된 경우에 반환될 수 있습니다. 클라이언트 토큰을 사용하여 응답을 요청과 연결할 수 있습니다.

  클라이언트 토큰 필드가 항상 오류 응답에 포함되는 것은 아닙니다. 요청에 주어진 클라이언트 토큰이 유효하지 않거나 형식이 잘못된 경우 오류 응답으로 반환되지 않습니다.

> **Note**
>
> 이전 버전과의 호환성을 위해 오류 응답의 필드는 약어가 아닌 형식일 수 있습니다. 예를 들어 오류 코드는 “code” 또는 “o” 필드에 의해 지정될 수 있으며 클라이언트 토큰 필드는

"clientToken" 또는 "c" 필드에 의해 지정될 수 있습니다. 위에 표시된 약어 양식을 사용하는 것이 좋습니다.

InvalidTopic

스트림 메시지의 MQTT 주제가 유효하지 않습니다.

InvalidJson

스트림 요청이 유효한 JSON 문서가 아닙니다.

InvalidCbor

스트림 요청이 유효한 CBOR 문서가 아닙니다.

InvalidRequest

요청은 일반적으로 잘못된 형식으로 식별됩니다. 자세한 내용은 오류 메시지를 참조하세요.

권한이 없음

요청에는 Amazon S3와 같은 스토리지 미디어의 스트림 데이터 파일에 액세스할 수 있는 권한이 없습니다. 자세한 내용은 오류 메시지를 참조하세요.

BlockSizeOutOfBounds

블록 크기가 범위를 벗어났습니다. AWS IoT Core Service Quotas의 "MQTT 기반 파일 전송" 섹션을 참조하세요.

OffsetOutOfBounds

오프셋이 범위를 벗어났습니다. AWS IoT Core Service Quotas의 "MQTT 기반 파일 전송" 섹션을 참조하세요.

BlockCountLimitExceeded

요청 블록 수가 범위를 벗어났습니다. AWS IoT Core Service Quotas의 "MQTT 기반 파일 전송" 섹션을 참조하세요.

BlockBitmapLimitExceeded

요청 비트맵의 크기가 범위를 벗어났습니다. AWS IoT Core Service Quotas의 "MQTT 기반 파일 전송" 섹션을 참조하세요.

ResourceNotFound

요청된 스트림, 파일, 파일 버전 또는 블록을 찾을 수 없습니다. 자세한 내용은 오류 메시지를 참조하세요.

VersionMismatch

요청의 스트림 버전이 MQTT 기반 파일 전송 기능의 스트림 버전과 일치하지 않습니다. 이는 스트림 버전이 디바이스에서 처음 수신된 이후 스트림 데이터가 수정되었음을 나타냅니다.

ETagMismatch

스트림의 S3 ETag가 최신 S3 객체 버전의 ETag와 일치하지 않습니다.

InternalError

MQTT 기반 파일 전송에서 내부 오류가 발생했습니다.

FreeRTOS OTA의 예제 사용 사례

FreeRTOS OTA(over-the-air) 에이전트는 AWS IoT MQTT 기반 파일 전송을 사용하여 FreeRTOS 펌웨어 이미지를 FreeRTOS 디바이스로 전송합니다. 초기 데이터 세트를 디바이스로 전송하기 위해 AWS IoT 작업 서비스를 사용하여 FreeRTOS 디바이스에 대한 OTA 업데이트 작업을 예약합니다.

MQTT 기반 파일 전송 클라이언트의 참조 구현에 대해서는 FreeRTOS 설명서의 FreeRTOS OTA 에이전트 코드를 참조하세요.

디바이스 어드바이저

Device Advisor는 디바이스 소프트웨어 개발 중에 IoT 디바이스의 유효성을 검사하기 위한 클라우드 기반의 완전 관리형 테스트 기능입니다. Device Advisor는 프로덕션에 디바이스를 배포하기 AWS IoT Core전에 IoT 디바이스의 안정적이고 안전한 연결을 검증하는 데 사용할 수 있는 사전 구축된 테스트를 제공합니다. Device Advisor의 미리 빌드된 테스트는 TLS, MQTT, 디바이스 섀도우 및 IoT 작업 사용에 대한 모범 사례에 대해 디바이스 소프트웨어를 검증하는 데 도움이 됩니다. 서명된 자격 보고서를 다운로드하고 AWS 파트너 네트워크에 제출함으로써 디바이스를 보내고 테스트를 기다릴 필요 없이 AWS 파트너 디바이스 카탈로그를 위한 디바이스 검증을 받을 수 있습니다.

> **Note**
>
> Device Advisor는 us-east-1, us-west-2, ap-northeast-1 및 eu-west-1 리전에서 지원됩니다.
> Device Advisor는 MQTT 및 MQTT over WebSocket Secure(WSS) 프로토콜을 사용하여 메시지를 게시하고 구독하는 디바이스와 클라이언트를 지원합니다. 모든 프로토콜은 IPv4 및 IPv6을 지원합니다.
> Device Advisor는 현재 RSA 서버 인증서를 지원합니다.

에 연결하도록 구축된 모든 디바이스는 Device Advisor를 활용할 AWS IoT Core 수 있습니다. AWS IoT 콘솔에서 또는 AWS CLI 또는 SDK를 사용하여 Device Advisor에 액세스할 수 있습니다. 디바이스를 테스트할 준비가 되면에 등록 AWS IoT Core 하고 Device Advisor 엔드포인트로 디바이스 소프트웨어를 구성합니다. 그런 다음 미리 빌드된 테스트를 선택하고, 구성하고, 디바이스에서 테스트를 실행하고, 자세한 로그 또는 검증 보고서와 함께 테스트 결과를 가져옵니다.

Device Advisor는 AWS 클라우드의 테스트 엔드포인트입니다. Device Advisor가 제공하는 테스트 엔드포인트에 연결하도록 디바이스를 구성하여 테스트할 수 있습니다. 테스트 엔드포인트에 연결하도록 디바이스를 구성한 후 Device Advisor 콘솔을 방문하거나 AWS SDK를 사용하여 디바이스에서 실행할 테스트를 선택할 수 있습니다. 그런 다음 Device Advisor는 리소스 프로비저닝, 테스트 프로세스 예약, 상태 시스템 관리, 디바이스 동작 기록, 결과 로깅, 테스트 보고서 형식으로 최종 결과 제공 등 테스트의 전체 수명 주기를 관리합니다.

TLS 프로토콜

전송 계층 보안(TLS) 프로토콜은 인터넷과 같은 안전하지 않은 네트워크를 통해 기밀 데이터를 암호화하는 데 사용됩니다. TLS 프로토콜은 보안 소켓 계층(SSL) 프로토콜의 후속 프로토콜입니다.

Device Advisor는 다음 TLS 프로토콜을 지원합니다.

- TLS 1.3(권장)
- TLS 1.2

프로토콜, 포트 매핑 및 인증

디바이스 통신 프로토콜은 디바이스 또는 클라이언트가 디바이스 엔드포인트를 사용하여 메시지 브로커에 연결하는 데 사용됩니다. 다음 테이블에는 Device Advisor 엔드포인트가 지원하는 프로토콜과 사용되는 인증 방법 및 포트가 나열되어 있습니다.

프로토콜, 인증 및 포트 매핑

프로토콜	지원되는 작업	Authentication	포트	ALPN 프로토콜 이름
MQTT over WebSocket	게시, 구독	서명 버전 4	443	해당 사항 없음
MQTT	게시, 구독	X.509 클라이언트 인증서	8883	`x-amzn-mqtt-ca`
MQTT	게시, 구독	X.509 클라이언트 인증서	443	해당 사항 없음

이번 장은 다음과 같은 단원들로 구성되어 있습니다.

- 설정
- 콘솔에서 Device Advisor 시작하기
- Device Advisor 워크플로
- Device Advisor 세부 콘솔 워크플로
- 장기간 테스트 콘솔 워크플로
- Device Advisor VPC 엔드포인트(AWS PrivateLink)
- Device Advisor 테스트 케이스

설정

Device Advisor를 처음 사용하는 경우, 먼저 다음 작업을 완료해야 합니다.

IoT 사물 생성

먼저 IoT 사물을 생성하고 이 사물에 인증서를 첨부합니다. 사물을 생성하는 방법에 대한 자습서는 사물 객체 생성을 참조하세요.

디바이스 역할로 사용할 IAM 역할 생성

> **Note**
>
> Device Advisor 콘솔을 사용하여 디바이스 역할을 빠르게 생성할 수 있습니다. Device Advisor 콘솔을 사용하여 디바이스 역할을 설정하는 방법은 콘솔에서 Device Advisor 시작하기를 참조하세요.

1. AWS Identity and Access Management 콘솔로 이동하여 Device Advisor 테스트에 사용하는에 로그인 AWS 계정 합니다.
2. 왼쪽 탐색 창에서 정책(Policies)을 선택합니다.
3. 정책 생성을 선택합니다.
4. 정책 생성(Create policy)에서 다음을 수행합니다.

   a. 서비스(Service)에서 IoT를 선택합니다.

   b. 작업에서 다음 작업 중 하나를 수행합니다.

      - (권장) 이전 섹션에서 생성한 IoT 사물 또는 인증서에 첨부된 정책을 기반으로 작업을 선택합니다.
      - 필터 작업 상자에서 다음 작업을 검색하고 선택합니다.

        - `Connect`
        - `Publish`
        - `Subscribe`
        - `Receive`
        - `RetainPublish`

   c. 리소스에서 클라이언트, 주제 및 주제 리소스를 제한합니다. 이러한 리소스를 제한하는 것은 보안 모범 사례입니다. 리소스를 제한하려면 다음 작업을 수행하세요.

      i. CONNECT 작업에 대한 클라이언트 리소스 ARN 지정(Specify client resource ARN for the Connect action)을 선택합니다.

ii. ARN 추가를 선택하고 다음 중 하나를 수행합니다.

> **Note**
>
> clientId는 디바이스가 Device Advisor와 상호 작용하기 위해 사용하는 MQTT 클라이언트 ID입니다.

- 시각적 ARN 편집기에서 리전, accountID, clientID를 지정합니다.
- 테스트 케이스를 실행하려는 IoT 주제의 Amazon 리소스 이름(ARN)을 수동으로 입력합니다.

iii. 추가를 선택합니다.

iv. 수신 및 1개 추가 작업에 대한 주제 리소스 ARN 지정을 선택합니다.

v. ARN 추가를 선택하고 다음 중 하나를 수행합니다.

> **Note**
>
> 주제 이름은 디바이스가 메시지를 게시하는 대상 MQTT 주제입니다.

- 시각적 ARN 편집기에서 리전, accountID, 주제 이름을 지정합니다.
- 테스트 케이스를 실행하려는 IoT 주제의 ARN을 수동으로 입력합니다.

vi. 추가(Add)를 선택합니다.

vii. 구독 작업에 대한 topicFilter 리소스 ARN 지정을 선택합니다.

viii. ARN 추가를 선택하고 다음 중 하나를 수행합니다.

> **Note**
>
> 주제 이름은 디바이스가 구독하는 MQTT 주제입니다.

- 시각적 ARN 편집기에서 리전, accountID, 주제 이름을 지정합니다.
- 테스트 케이스를 실행하려는 IoT 주제의 ARN을 수동으로 입력합니다.

ix. 추가를 선택합니다.

5. 다음: 태그를 선택합니다.

6. 다음: 검토를 선택합니다.
7. 정책 검토에 정책의 이름을 입력합니다.
8. 정책 생성을 선택합니다.
9. 왼쪽 탐색 창에서 역할을 선택합니다.
10. 역할 생성을 선택합니다.
11. 신뢰할 수 있는 엔터티 선택에서 사용자 지정 신뢰 정책을 선택합니다.
12. 사용자 지정 신뢰 정책 상자에 다음 신뢰 정책을 입력합니다. 혼동된 대리자 문제를 방지하기 위하여 전역 조건 컨텍스트 키 aws:SourceArn 및 aws:SourceAccount를 정책에 추가합니다.

> **Important**
>
> aws:SourceArn이 format: arn:aws:iotdeviceadvisor:*region:account-id*:*.를 준수해야 합니다. *region*이 AWS IoT 리전과 일치하고 *account-id*가 고객 계정 ID와 일치해야 합니다. 자세한 설명은 교차 서비스 혼동된 대리자 예방을 참조하세요.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAwsIoTCoreDeviceAdvisor",
            "Effect": "Allow",
            "Principal": {
                "Service": "iotdeviceadvisor.amazonaws.com"
        },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
            },
                "ArnLike": {
                    "aws:SourceArn":
 "arn:aws:iotdeviceadvisor:*:123456789012:suitedefinition/*"
            }
        }
        }
    ]
}
```

13. 다음을 선택합니다.
14. 4단계에서 생성한 정책을 선택합니다.
15. (선택 사항) 권한 경계 설정에서 권한 경계를 사용하여 최대 역할 권한 제어를 선택한 후 방금 생성한 정책을 선택합니다.
16. 다음을 선택합니다.
17. 역할 이름 및 역할 설명을 입력합니다.
18. 역할 생성을 선택합니다.

IAM 사용자가 Device Advisor를 사용할 수 있도록 사용자 정의 관리형 정책 생성

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔로 이동합니다. 메시지가 표시되면 AWS 자격 증명을 입력하여 로그인합니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 선택한 후 JSON 탭을 선택합니다.
4. Device Advisor를 사용하는 데 필요한 권한을 추가합니다. 정책 문서는 보안 모범 사례 주제에서 찾을 수 있습니다.
5. 정책 검토를 선택합니다.
6. 이름 및 설명을 입력합니다.
7. 정책 생성을 선택하세요.

Device Advisor를 사용할 IAM 사용자 생성

> **Note**
>
> Device Advisor 테스트를 실행할 때 사용할 IAM 사용자를 생성하는 것이 좋습니다. 관리자 사용자로부터 Device Advisor 테스트를 실행하면 보안 위험이 발생할 수 있으므로 권장하지 않습니다.

1. IAM 콘솔(https://console.aws.amazon.com/iam/)로 이동합니다. 메시지가 표시되면 AWS 자격 증명을 입력하여 로그인합니다.
2. 왼쪽 탐색 창에서 사용자를 선택합니다.

3. 사용자 추가를 선택합니다.
4. 사용자 이름을 입력합니다.
5. 사용자는 AWS 외부에서와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다 AWS Management Console. 프로그래밍 방식 액세스를 부여하는 방법은에 액세스하는 사용자 유형에 따라 다릅니다 AWS.

   사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자	목적	방법
IAM	(권장) 콘솔 자격 증명을 임시 자격 증명으로 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 Login for AWS local development를 참조하세요. • AWS SDKs 경우 SDK 및 도구 참조 안내서의 AWS 로컬 개발을 위한 로그인을 참조하세요. AWS SDKs
작업 인력 ID (IAM Identity Center에서 관리되는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 를 사용하도록 구성을 AWS IAM Identity Center 참조하세요. • AWS SDKs, 도구 및 AWS APIs의 경우 SDK 및 도구 참조 안내서의 IAM Identity

프로그래밍 방식 액세스가 필요한 사용자	목적	방법
		Center 인증을 참조하세요. AWS SDKs
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	IAM 사용 설명서의 AWS 리소스에서 임시 자격 증명 사용의 지침을 따릅니다.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하세요. • AWS SDKs 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용하여 인증을 참조하세요. AWS SDKs • AWS APIs 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요.

6. 다음: 권한을 선택합니다.
7. 액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요.

   • 의 사용자 및 그룹 AWS IAM Identity Center:

     권한 세트를 생성합니다. AWS IAM Identity Center 사용자 안내서에서 권한 세트 생성의 지침을 따릅니다.

   • ID 제공업체를 통해 IAM에서 관리되는 사용자:

ID 페더레이션을 위한 역할을 생성합니다. IAM 사용자 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용자 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따릅니다.

8. 생성한 사용자 지정 관리형 정책의 이름을 검색 상자에 입력합니다. 그런 다음 정책 이름 확인란을 선택합니다.
9. 다음: 태그를 선택합니다.
10. 다음: 검토를 선택합니다.
11. 사용자 생성을 선택합니다.
12. 닫기를 선택합니다.

Device Advisor를 사용하려면 사용자를 대신하여 AWS 리소스(사물, 인증서 및 엔드포인트)에 액세스해야 합니다. IAM 사용자에게 필요한 권한이 있어야 합니다. 또한 IAM 사용자에게 필요한 권한 정책을 연결하는 경우 Device Advisor는 Amazon CloudWatch에 로그를 게시합니다.

디바이스 구성

Device Advisor에서는 SNI(서버 이름 표시) TLS 확장을 사용하여 TLS 구성을 적용합니다. 디바이스는 Device Advisor 테스트 엔드포인트와 동일한 서버 이름을 연결하고 전달할 때 이 확장을 사용해야 합니다.

Device Advisor는 테스트가 `Running` 상태일 때 TLS 연결을 허용합니다. 각 테스트 실행 전후에는 TLS 연결을 거부합니다. 이러한 이유로 Device Advisor에서 완전히 자동화된 테스트 경험을 위해 디바이스 연결 재시도 메커니즘을 사용할 것을 권장합니다. TLS 연결, MQTT 연결 및 MQTT 게시와 같이 둘 이상의 테스트 케이스가 포함된 테스트 제품군을 실행할 수 있습니다. 테스트 케이스 여러 개를 실행하는 경우 디바이스가 5초마다 테스트 엔드포인트에 연결을 시도하도록 하는 것이 좋습니다. 그런 다음 여러 테스트 케이스를 순서대로 실행하도록 자동화할 수 있습니다.

> **Note**
>
> 디바이스 소프트웨어를 테스트할 수 있도록 준비하려면 AWS IoT Core에 연결할 수 있는 SDK를 사용하는 것이 좋습니다. 그런 다음 AWS 계정에 제공된 Device Advisor 테스트 엔드포인트로 SDK를 업데이트해야 합니다.

Device Advisor는 계정 수준 엔드포인트와 디바이스 수준 엔드포인트라는 두 가지 유형의 엔드포인트를 지원합니다. 사용 사례에 가장 적합한 엔드포인트를 선택합니다. 서로 다른 디바이스에 대해 여러 테스트 제품군을 동시에 실행하려면 디바이스 수준 엔드포인트를 사용합니다.

다음 명령을 실행하여 디바이스 수준 엔드포인트를 가져옵니다.

X.509 클라이언트 인증서를 사용하는 MQTT 고객의 경우:

```
aws iotdeviceadvisor get-endpoint --thing-arn your-thing-arn
```

또는

```
aws iotdeviceadvisor get-endpoint --certificate-arn your-certificate-arn
```

서명 버전 4를 사용하는 MQTT over WebSocket 고객의 경우:

```
aws iotdeviceadvisor get-endpoint --device-role-arn your-device-role-arn --
authentication-method SignatureVersion4
```

한 번에 하나의 테스트 제품군을 실행하려면 계정 수준 엔드포인트를 선택합니다. 다음 명령을 실행하여 계정 수준 엔드포인트를 가져옵니다.

```
aws iotdeviceadvisor get-endpoint
```

콘솔에서 Device Advisor 시작하기

이 자습서는 콘솔 AWS IoT Core Device Advisor 에서를 시작하는 데 도움이 됩니다. Device Advisor는 필수 테스트 및 서명된 자격 보고서와 같은 기능을 제공합니다. 이 테스트 및 보고서를 사용하여 AWS IoT Core 자격 프로그램에 자세히 설명된 대로 AWS 파트너 디바이스 카탈로그에서 디바이스에 자격을 부여하고 나열할 수 있습니다.

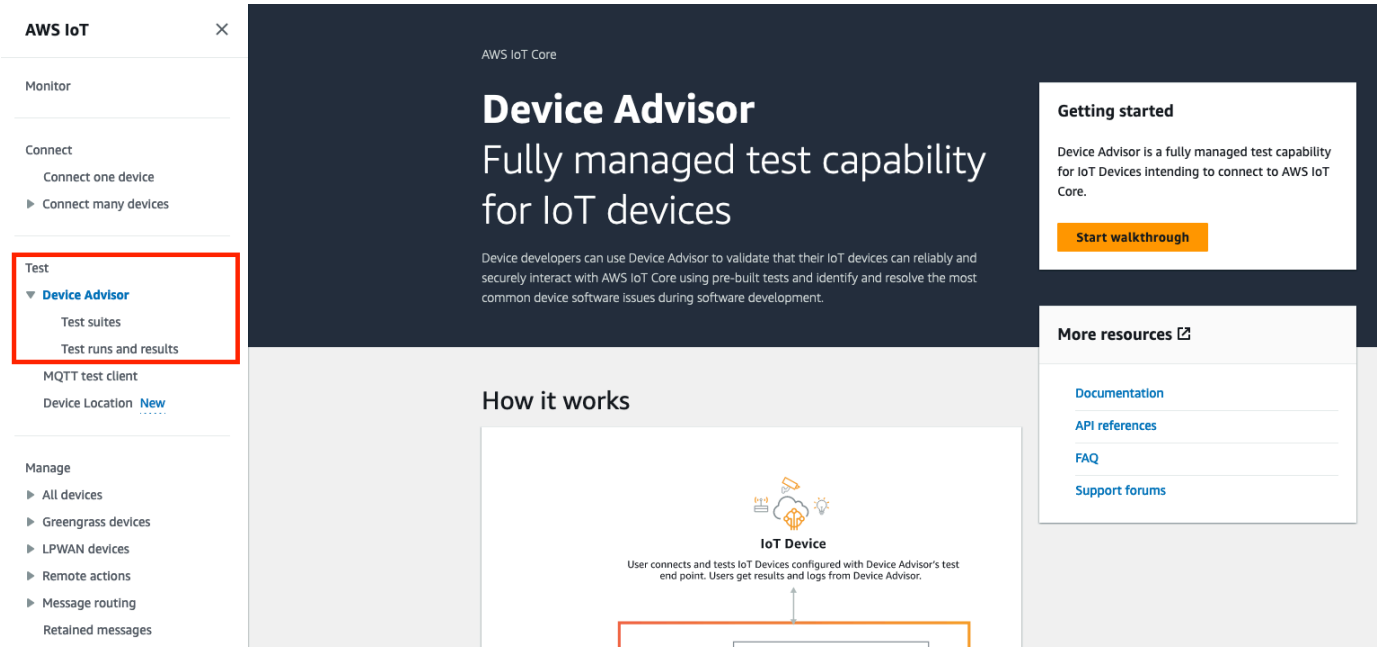
Device Advisor 사용에 대한 자세한 내용은 Device Advisor 워크플로 및 Device Advisor 세부 콘솔 워크플로 섹션을 참조하세요.

이 자습서를 완료하려면 설정에 설명된 단계를 따릅니다.

> **Note**
>
> Device Advisor는 AWS 리전다음에서 지원됩니다.
>
> - 미국 동부(버지니아 북부)
> - 미국 서부(오레곤)
> - 아시아 태평양(도쿄)
> - 유럽(아일랜드)

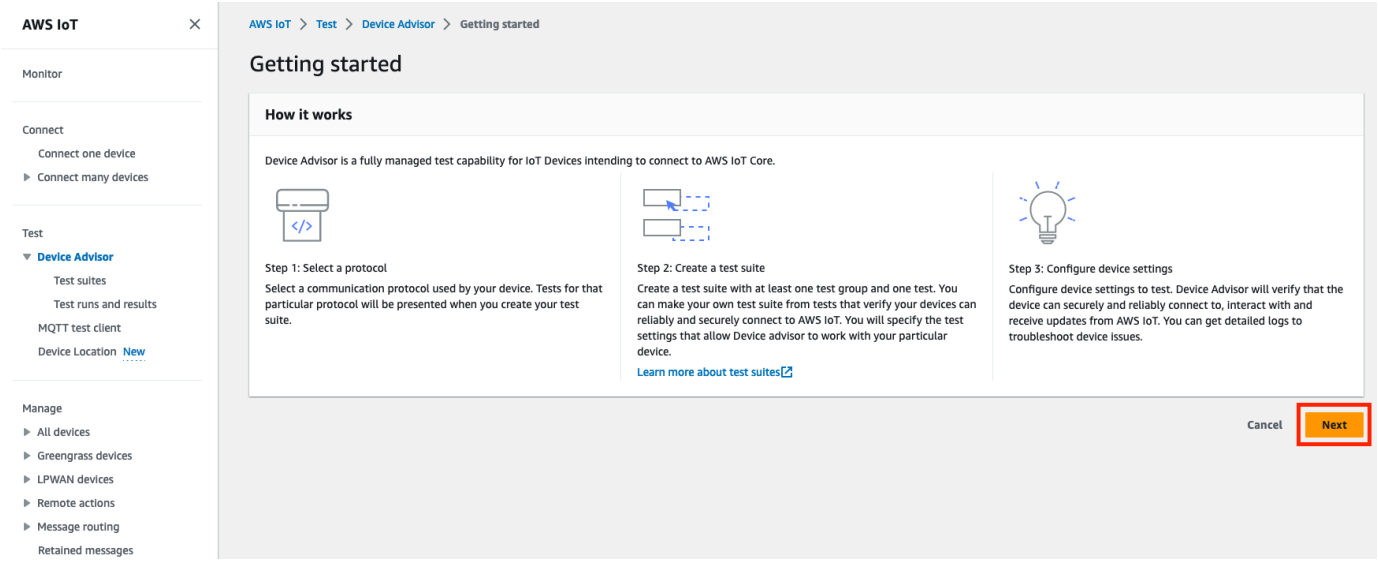
시작하기

1. AWS IoT 콘솔 탐색 창의 테스트에서 Device Advisor를 선택합니다. 그런 다음 콘솔에서 설명 시작 버튼을 선택합니다.

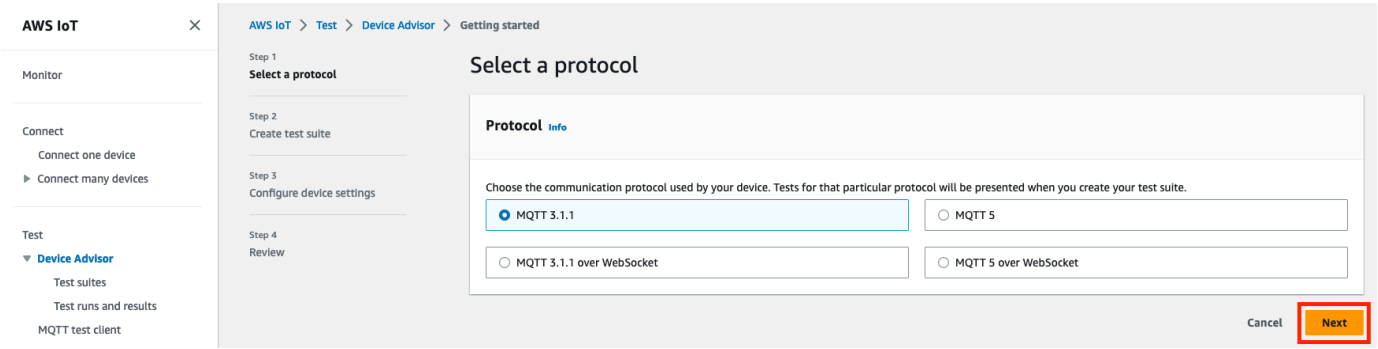


2. Device Advisor 시작하기 페이지에서는 디바이스에 대해 테스트 제품군을 만들고 테스트를 실행하는 데 필요한 단계에 대한 개요를 제공합니다. 여기에서 계정의 Device Advisor 테스트 엔드포인트를 찾을 수도 있습니다. 이 테스트 엔드포인트에 연결하기 위해 테스트에 사용되는 디바이스의 펌웨어 또는 소프트웨어를 구성해야 합니다.

이 자습서를 완료하려면 먼저 사물 및 인증서를 생성해야 합니다. 작동 방식에 있는 정보를 검토한 후 다음을 선택합니다.

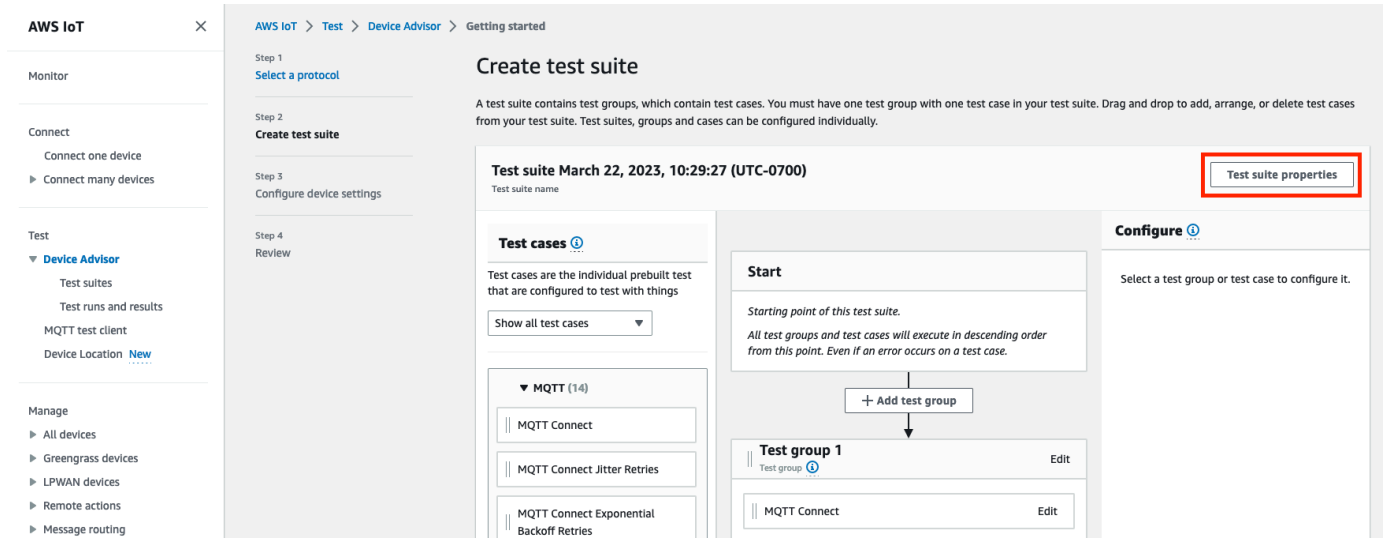


3. 1단계: 프로토콜 선택에서 나열된 옵션 중에서 프로토콜을 선택합니다. 그리고 다음을 선택합니다.



4. 2단계(Step 2)에서 사용자 정의 테스트 스위트를 만들고 구성합니다. 사용자 지정 테스트 스위트에는 하나 이상의 테스트 그룹이 있어야 하며 각 테스트 그룹에는 하나 이상의 테스트 케이스가 있어야 합니다. 사용자가 시작할 수 있도록 MQTT 연결 테스트 케이스를 추가했습니다.

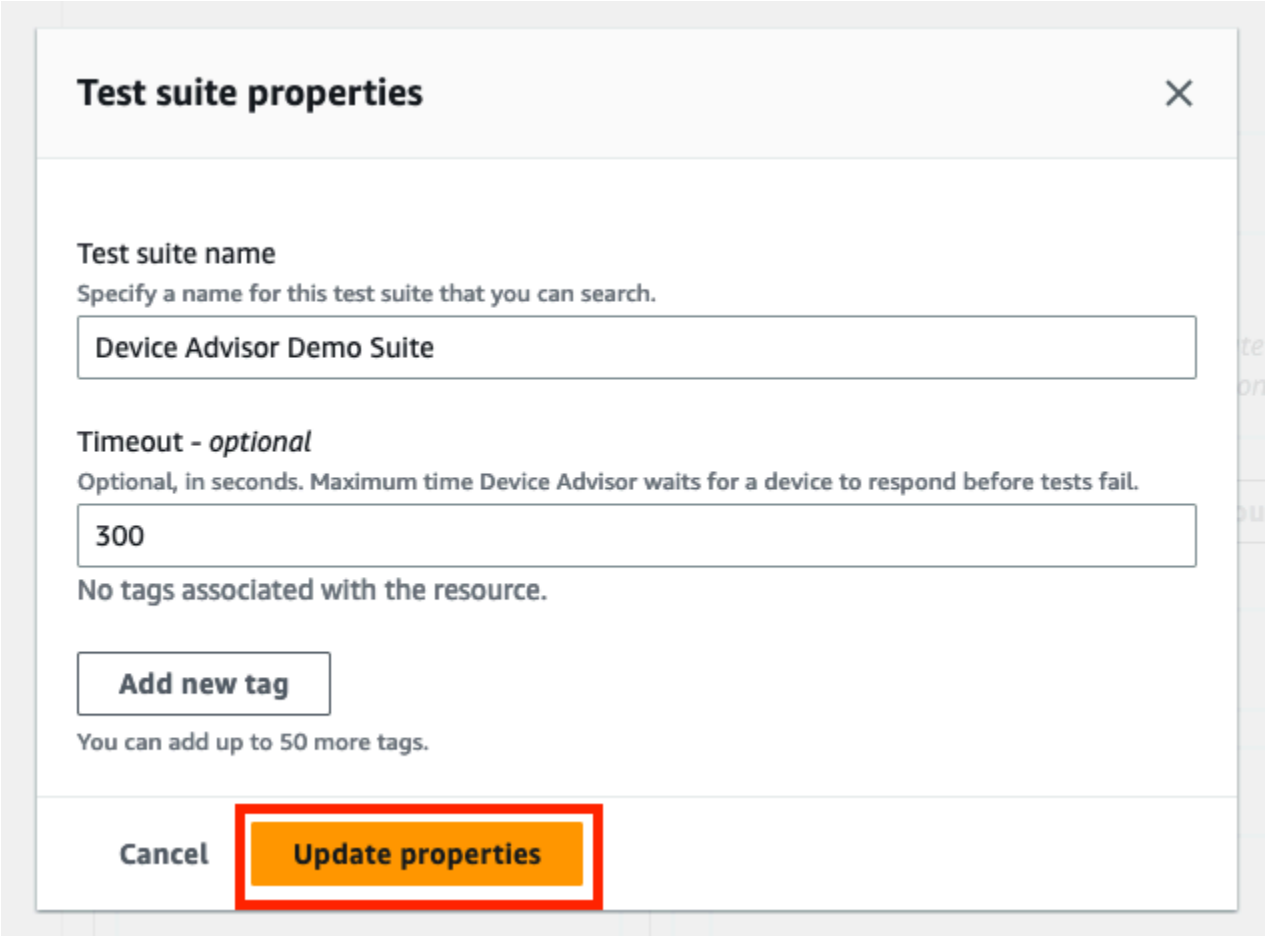
   테스트 스위트 속성(Test suite properties)을 선택합니다.



테스트 제품군을 만들 때 테스트 제품군 속성을 제공합니다. 다음 스위트 수준 속성을 구성할 수 있습니다.

- 테스트 제품군 이름(Test suite name): 테스트 제품군 이름을 입력합니다.
- 제한 시간(선택 사항): 현재 테스트 제품군의 각 테스트 케이스에 대한 제한 시간(초)입니다. 제한 시간 값을 지정하지 않으면 기본값이 사용됩니다.
- 태그(Tags)(선택 사항): 테스트 제품군에 태그를 추가합니다.

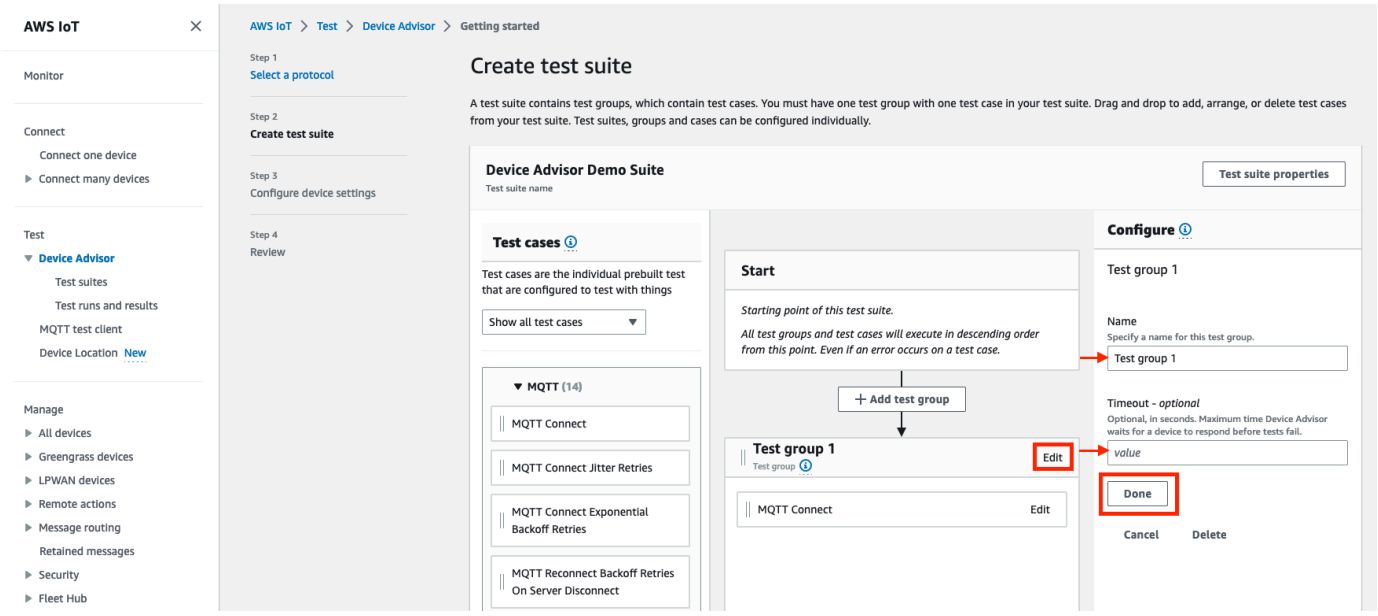
완료했으면 Update properties(속성 업데이트)를 선택합니다.



5. (선택 사항) 테스트 그룹 이름 옆에 있는 편집 버튼을 선택하여 테스트 제품군 그룹 구성을 업데이트할 수 있습니다.

   - 이름(Name): 테스트 스위트 그룹의 사용자 정의 이름을 입력합니다.
   - 제한 시간(선택 사항): 현재 테스트 제품군의 각 테스트 케이스에 대한 제한 시간(초)입니다. 제한 시간 값을 지정하지 않으면 기본값이 사용됩니다.

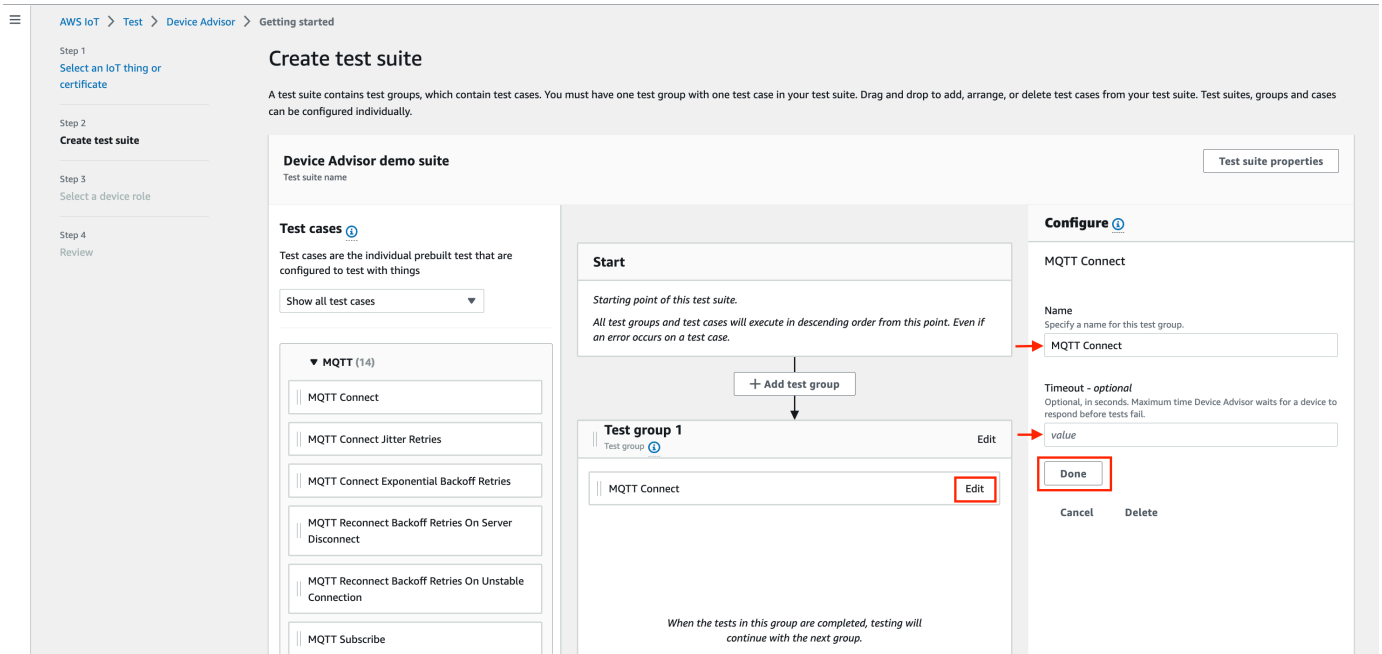
   마쳤으면 완료를 선택하여 계속합니다.



6. (선택 사항) 테스트 케이스 이름 옆에 있는 편집 버튼을 선택하여 테스트 케이스의 테스트 케이스 구성을 업데이트할 수 있습니다.

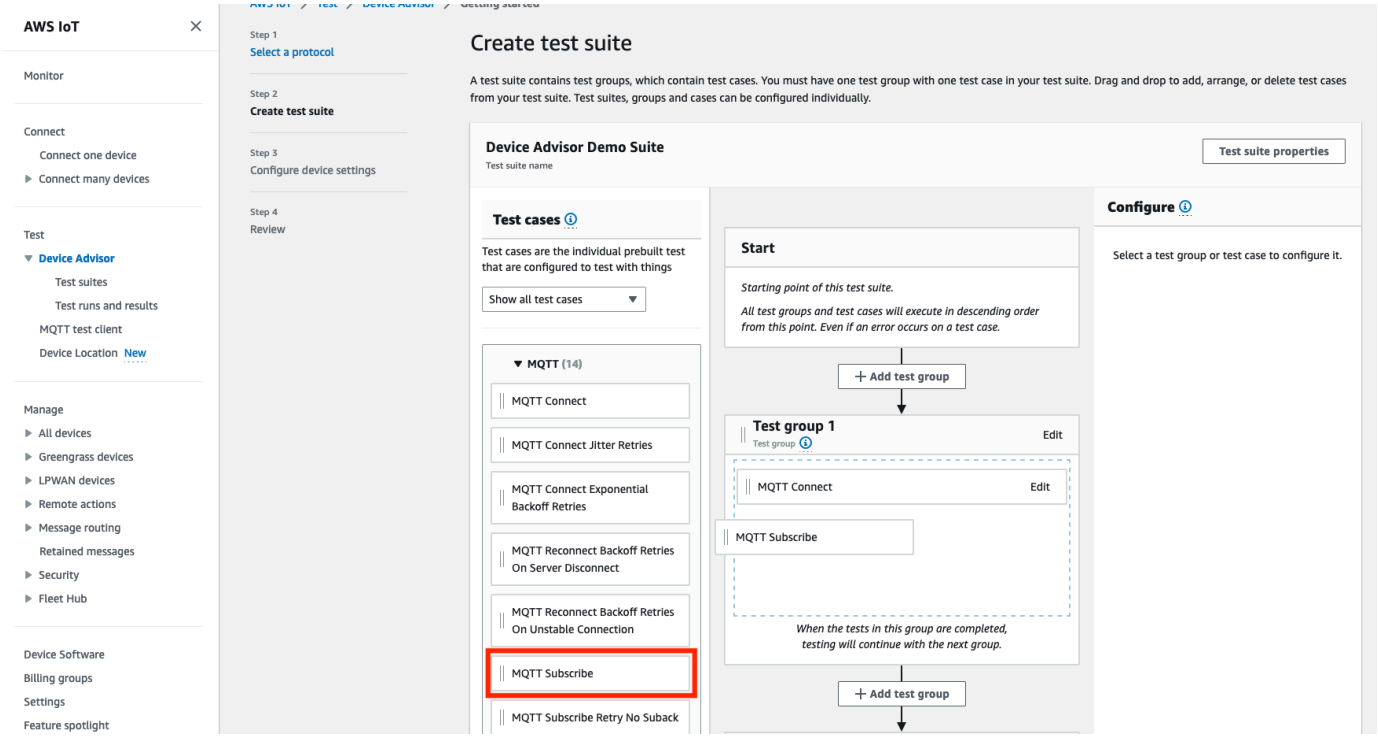
   - 이름(Name): 테스트 스위트 그룹의 사용자 정의 이름을 입력합니다.
   - 제한 시간(선택 사항): 선택한 테스트 케이스의 제한 시간(초)입니다. 제한 시간 값을 지정하지 않으면 기본값이 사용됩니다.

   마쳤으면 완료를 선택하여 계속합니다.



7. (선택 사항) 테스트 제품군에 테스트 그룹을 더 추가하려면 테스트 그룹 추가를 선택하고 5단계의 지침을 따릅니다.

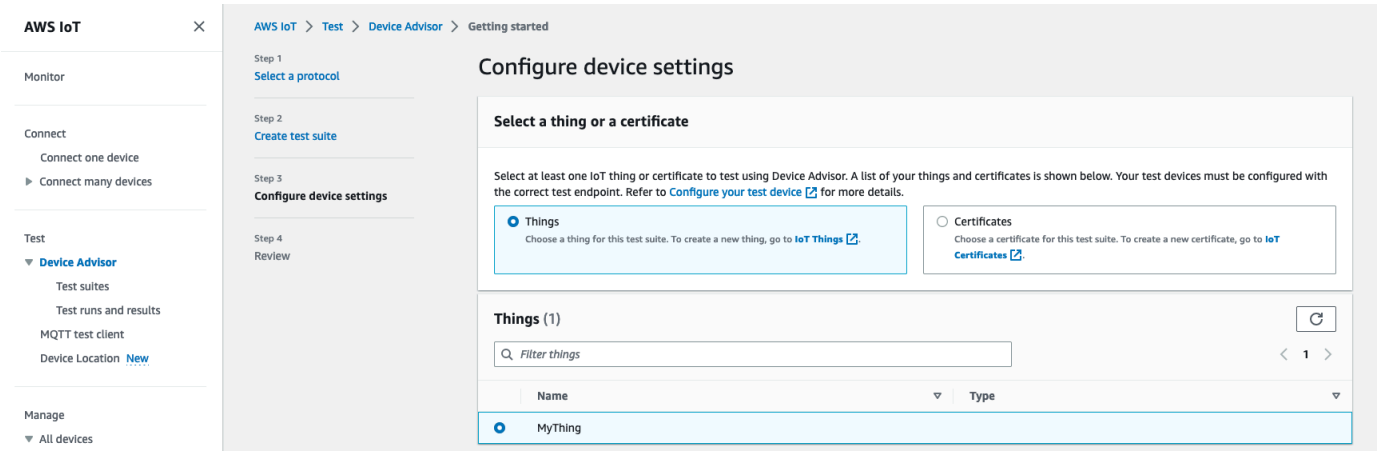
8. (선택 사항) 더 많은 테스트 케이스를 추가하려면 테스트 케이스 섹션의 테스트 케이스를 테스트 그룹 중 하나로 끌어옵니다.



9. 테스트 그룹 및 테스트 케이스의 순서를 변경할 수 있습니다. 변경하려면 나열된 테스트 케이스를 목록 위나 아래로 드래그합니다. Device Advisor는 테스트를 나열된 순서대로 실행합니다.

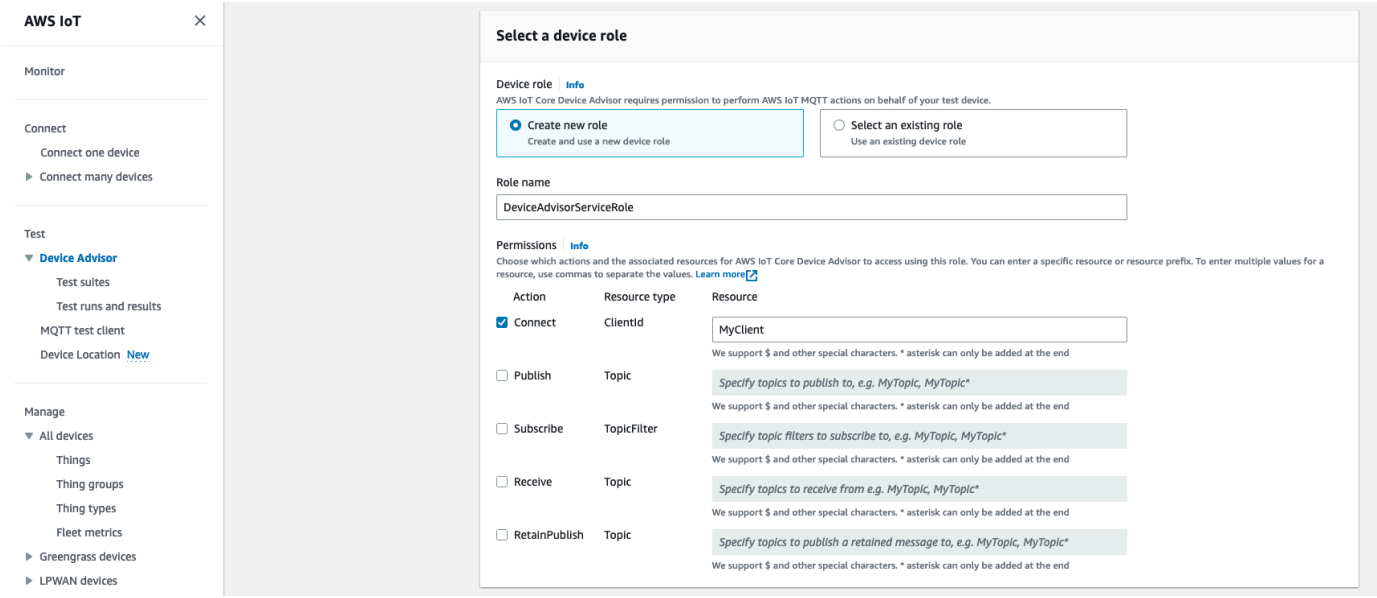
   테스트 스위트를 구성한 후 다음(Next)을 선택합니다.

10. 3단계에서 Device Advisor를 사용하여 테스트할 AWS IoT 사물 또는 인증서를 선택합니다. 기존 사물이나 인증서가 없는 경우 설정을 참조하세요.

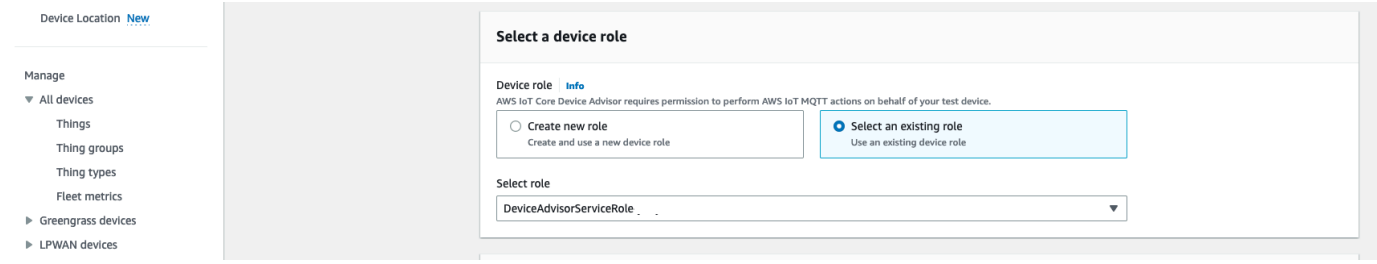


11. Device Advisor가 테스트 디바이스를 대신하여 AWS IoT MQTT 작업을 수행하는 데 사용하는 디바이스 역할을 구성할 수 있습니다. MQTT 연결 테스트 케이스의 경우에만 연결 작업이 자동으로 선택됩니다. 테스트 제품군을 실행하려면 디바이스 역할에 이 권한이 필요하기 때문입니다. 다른 테스트 케이스의 경우 해당하는 작업이 선택됩니다.

    선택한 각 작업에 대한 리소스 값을 제공합니다. 예를 들어 연결 작업의 경우 디바이스가 Device Advisor 엔드포인트에 연결하기 위해 사용하는 클라이언트 ID를 제공합니다. 여러 값을 쉼표로 구분하여 제공하고 값 앞에 와일드카드(*) 문자를 사용할 수 있습니다. 예를 들어, `MyTopic`으로 시작하는 주제에 대한 게시 권한을 제공하려면 리소스 값으로 **`MyTopic*`**을 입력합니다.

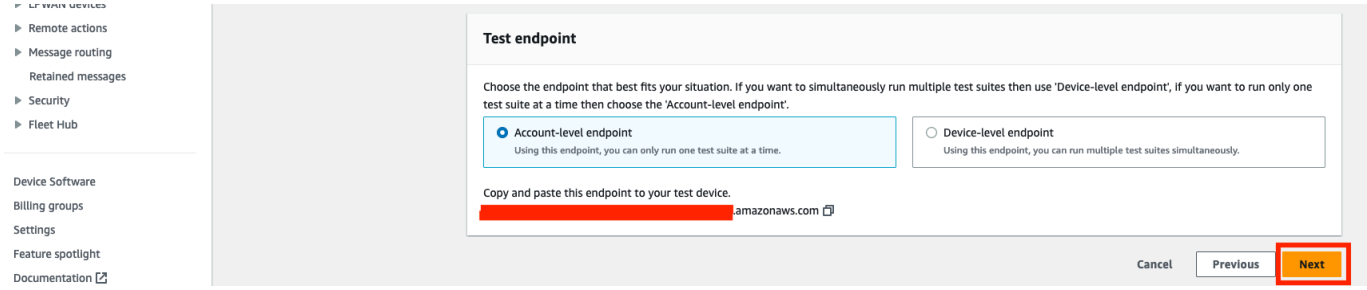


    설정에서 이전에 만든 디바이스 역할을 사용하려면 기존 역할 선택을 선택합니다. 그런 다음 역할 선택에서 디바이스 역할을 선택합니다.



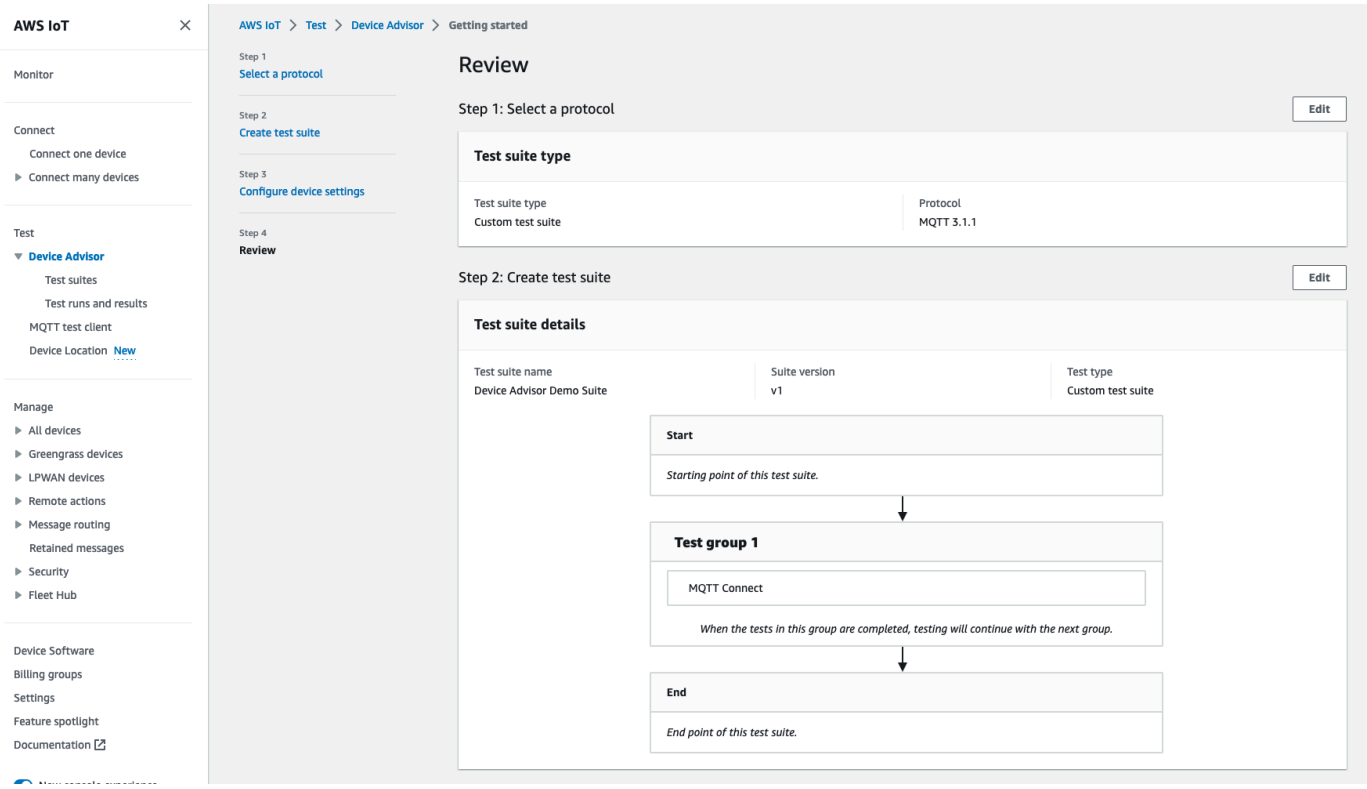
    제공된 두 옵션 중 하나를 사용하여 디바이스 역할을 구성하고 다음을 선택합니다.

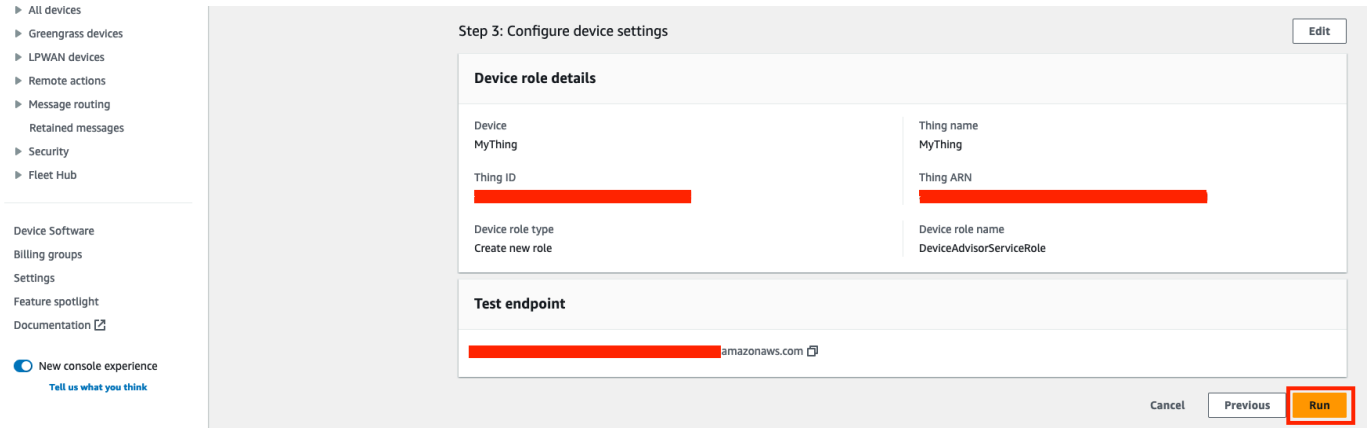
12. 테스트 엔드포인트 섹션에서 사용 사례에 가장 적합한 엔드포인트를 선택합니다. 여러 테스트 제품군을 동일한와 동시에 실행하려면 디바이스 수준 엔드포인트를 AWS 계정선택합니다. 한 번에 하나의 테스트 제품군을 실행하려면 계정 수준 엔드포인트를 선택합니다.



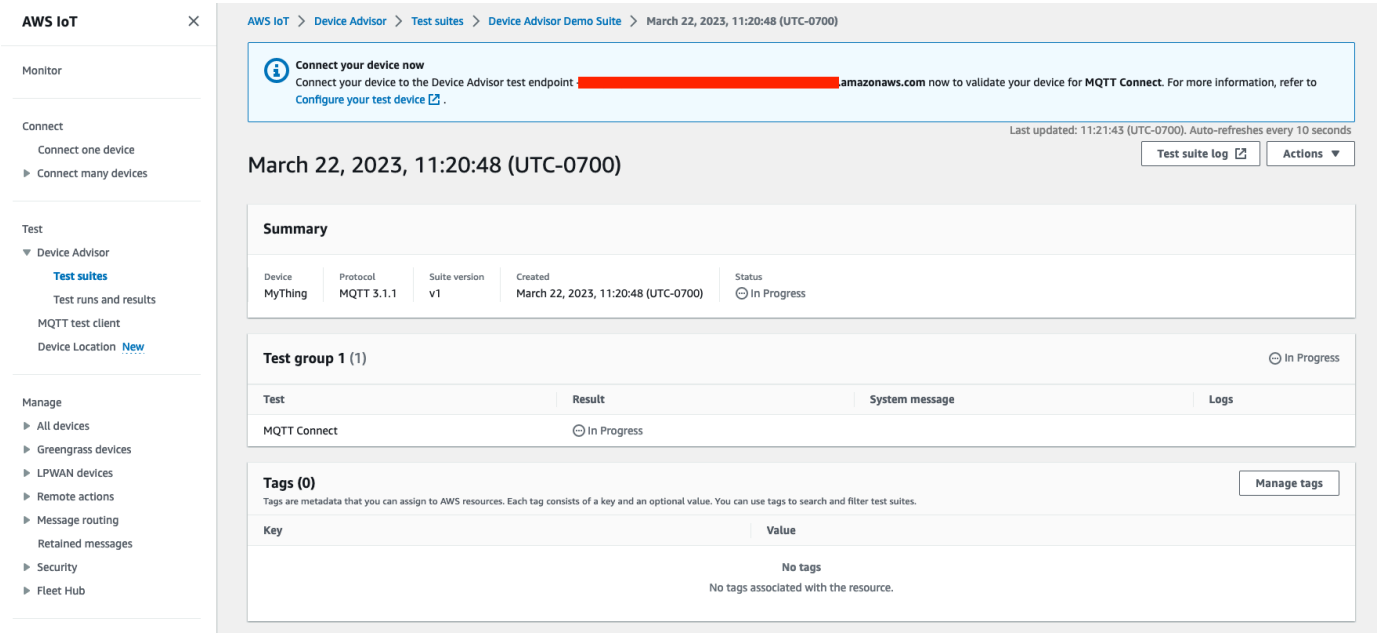
13. 4단계에서는 선택한 테스트 디바이스, 테스트 엔드포인트, 테스트 제품군 및 구성된 테스트 디바이스 역할에 대한 개요를 제공합니다. 섹션을 변경하려면 편집하려는 섹션 위에 있는 편집 버튼을 선택합니다. 테스트 구성을 확인했으면 실행을 선택하여 테스트 제품군을 만들고 테스트를 실행합니다.

> **Note**
>
> 최상의 결과를 위해 테스트 제품군 실행을 시작하기 전에 선택한 테스트 디바이스를 Device Advisor 테스트 엔드포인트에 연결할 수 있습니다. 디바이스에서 테스트 엔드포인트에 최대 1~2분 동안 5초마다 연결을 시도할 수 있도록 구성된 메커니즘을 사용하는 것이 좋습니다.





14. 테스트의 탐색 창에서 Device Advisor를 선택하고 테스트 실행 및 결과를 선택합니다. 실행 세부 정보 및 로그를 보려면 테스트 제품군 실행을 선택합니다.



15. 제품군 실행에 대한 Amazon CloudWatch 로그에 액세스하는 방법:

    - 테스트 스위트 로그(Test suite log)를 선택하여 실행 중인 테스트 스위트에 대한 CloudWatch 로그를 봅니다.
    - 테스트 케이스에 대한 테스트 케이스 로그(Test suite log)를 선택하여 테스트 케이스 관련 CloudWatch 로그를 봅니다.

16. 테스트 결과에 따라 모든 테스트가 통과될 때까지 디바이스의 문제를 해결합니다.

Device Advisor 워크플로

이 자습서에서는 사용자 지정 테스트 제품군을 만들고 콘솔에서 테스트할 디바이스에 대해 테스트를 실행하는 방법을 설명합니다. 테스트가 완료되면 테스트 결과와 세부 로그를 볼 수 있습니다.

사전 조건

이 자습서를 시작하기 전에 설정에 설명된 단계를 완료하세요.

테스트 스위트 정의 만들기

먼저 AWS SDK를 설치합니다.

rootGroup 구문

루트 그룹은 테스트 제품군에 포함할 테스트 케이스를 지정하는 JSON 문자열입니다. 또한 해당 테스트 케이스에 필요한 모든 구성을 지정합니다. 루트 그룹을 사용하여 필요에 따라 테스트 제품군을 구조화하고 정렬하세요. 테스트 스위트의 계층 구조는 다음과 같습니다.

```
test suite # test group(s) # test case(s)
```

테스트 스위트에는 하나 이상의 테스트 그룹이 있어야 하며 각 테스트 그룹에는 하나 이상의 테스트 케이스가 있어야 합니다. Device Advisor는 테스트 그룹 및 테스트 케이스를 정의하는 순서대로 테스트를 실행합니다.

각 루트 그룹에는 다음과 같은 기본 구조를 따릅니다.

```
{
    "configuration": {  // for all tests in the test suite
        "": ""
    }
    "tests": [{
        "name": ""
        "configuration": {  // for all sub-groups in this test group
            "": ""
        },
        "tests": [{
            "name": ""
            "configuration": {  // for all test cases in this test group
                "": ""
            },
            "test": {
```

```
                "id": ""
                "version": ""
            }
        }]
    }]
}
```

루트 그룹에서 그룹에 포함된 name, configuration, tests를 사용하여 테스트 제품군을 정의합니다. tests 그룹에는 개별 테스트의 정의가 포함됩니다. 각 테스트는 해당 테스트의 테스트 케이스를 정의하는 name, configuration, test 블록으로 정의합니다. 마지막으로, 각 테스트 케이스는 id 및 version으로 정의됩니다.

각 테스트 케이스에 대한 "id" 및 "version" 필드(test 블록)의 사용 방법에 대한 자세한 내용은 Device Advisor 테스트 케이스 섹션을 참조하세요. 이 섹션에는 사용 가능한 configuration 설정에 대한 정보가 포함되어 있습니다.

다음 블록은 루트 그룹 구성의 예입니다. 이 구성은 구성 필드에 대한 MQTT Connect Happy Case 및 MQTT Connect Exponential Backoff Retries 테스트 케이스를 지정합니다.

```
{
    "configuration": {},  // Suite-level configuration
    "tests": [            // Group definitions should be provided here
      {
        "name": "My_MQTT_Connect_Group",  // Group definition name
        "configuration": {}               // Group definition-level configuration,
        "tests": [                        // Test case definitions should be provided
 here
        {
            "name": "My_MQTT_Connect_Happy_Case",  // Test case definition name
            "configuration": {
                "EXECUTION_TIMEOUT": 300        // Test case definition-level
 configuration, in seconds
            },
            "test": {
                "id": "MQTT_Connect",              // test case id
                "version": "0.0.0"                 // test case version
            }
        },
        {
            "name": "My_MQTT_Connect_Jitter_Backoff_Retries",  // Test case definition
 name
            "configuration": {
```

```
                "EXECUTION_TIMEOUT": 600                // Test case definition-level
 configuration,  in seconds
            },
            "test": {
                "id": "MQTT_Connect_Jitter_Backoff_Retries",  // test case id
                "version": "0.0.0"                              // test case version
            }
        }]
    }]
}
```

테스트 스위트 정의를 만들 때 루트 그룹 구성을 제공해야 합니다. 응답 객체에 반환된 `suiteDefinitionId`를 저장합니다. 이 ID를 사용하여 테스트 제품군 정의 정보를 검색하고 테스트 제품군을 실행할 수 있습니다.

다음은 Java SDK 예제입니다.

```
response = iotDeviceAdvisorClient.createSuiteDefinition(
        CreateSuiteDefinitionRequest.builder()
            .suiteDefinitionConfiguration(SuiteDefinitionConfiguration.builder()
                .suiteDefinitionName("your-suite-definition-name")
                .devices(
                    DeviceUnderTest.builder()
                        .thingArn("your-test-device-thing-arn")
                        .certificateArn("your-test-device-certificate-arn")
                        .deviceRoleArn("your-device-role-arn") //if using SigV4 for
 MQTT over WebSocket
                        .build()
                )
                .rootGroup("your-root-group-configuration")
                .devicePermissionRoleArn("your-device-permission-role-arn")
                .protocol("MqttV3_1_1 || MqttV5 || MqttV3_1_1_OverWebSocket ||
 MqttV5_OverWebSocket")
                .build()
            )
            .build()
)
```

테스트 스위트 정의 가져오기

테스트 제품군 정의를 만든 후 `CreateSuiteDefinition` API 작업의 응답 객체에서 `suiteDefinitionId`를 수신합니다.

작업이 suiteDefinitionId를 반환하면 각 그룹 내에 새로운 id 필드가 표시되고 루트 그룹 내에 테스트 케이스 정의가 표시될 수 있습니다. 이러한 ID를 사용하여 테스트 제품군 정의의 하위 집합을 실행할 수 있습니다.

Java SDK 예제:

```
response = iotDeviceAdvisorClient.GetSuiteDefinition(
    GetSuiteDefinitionRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .build()
)
```

테스트 엔드포인트 가져오기

GetEndpoint API 작업을 사용하여 디바이스에서 사용하는 테스트 엔드포인트를 가져옵니다. 테스트에 가장 적합한 엔드포인트를 선택합니다. 여러 테스트 제품군을 동시에 실행하려면 thing ARN, certificate ARN 또는 device role ARN을 제공하여 디바이스 수준 엔드포인트를 사용합니다. 단일 테스트 제품군을 실행하려면 GetEndpoint 작업에 인수를 제공하지 않고 계정 수준 엔드포인트를 선택합니다.

SDK 예제:

```
response = iotDeviceAdvisorClient.getEndpoint(GetEndpointRequest.builder()
.certificateArn("your-test-device-certificate-arn")
.thingArn("your-test-device-thing-arn")
.deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

.build())
```

테스트 스위트 실행 시작하기

테스트 제품군 정의를 만들고 테스트 디바이스를 Device Advisor 테스트 엔드포인트에 연결하도록 구성한 후 StartSuiteRun API로 테스트 제품군을 실행합니다.

MQTT 고객의 경우 certificateArn 또는 thingArn을 사용하여 테스트 제품군을 실행하세요. 둘 다 구성된 경우에 인증서가 해당 사물에 속하면 인증서가 사용됩니다.

MQTT over WebSocket 고객의 경우 deviceRoleArn을 사용하여 테스트 제품군을 실행합니다. 지정된 역할이 테스트 제품군 정의에 지정된 역할과 다른 경우 지정된 역할이 정의된 역할을 재정의합니다.

디바이스 수준 엔드포인트를 사용하여 하나의 AWS 계정으로 여러 테스트 제품군을 병렬로 실행하는 경우 .parallelRun()에 true를 사용합니다.

SDK 예제:

```
response = iotDeviceAdvisorClient.startSuiteRun(StartSuiteRunRequest.builder()
.suiteDefinitionId("your-suite-definition-id")
.suiteRunConfiguration(SuiteRunConfiguration.builder()
    .primaryDevice(DeviceUnderTest.builder()
        .certificateArn("your-test-device-certificate-arn")
        .thingArn("your-test-device-thing-arn")
        .deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

        .build())
    .parallelRun(true | false)
    .build())
.build())
```

응답에서 suiteRunId를 저장합니다. 이 테스트 제품군 실행 결과를 검색하는 데 사용됩니다.

테스트 스위트 실행 가져오기

테스트 스위트 실행을 시작한 후에는 GetSuiteRun API로 진행 상황과 결과를 확인할 수 있습니다.

SDK 예제:

```
// Using the SDK, call the GetSuiteRun API.

response = iotDeviceAdvisorClient.GetSuiteRun(
GetSuiteRunRequest.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

테스트 스위트 실행 중지

아직 진행 중인 테스트 제품군 실행을 중지하려면 StopSuiteRun API 작업을 호출할 수 있습니다. StopSuiteRun 작업을 호출하고 나면 서비스가 정리 프로세스를 시작합니다. 서비스가 정리 프로세스를 실행하는 동안 테스트 제품군 실행 상태는 Stopping으로 업데이트됩니다. 정리 프로세스에는 몇 분이 걸릴 수 있습니다. 프로세스가 완료되면 테스트 제품군 실행 상태가 Stopped로 업데이트됩니다. 테스트 실행이 완전히 중지된 후에 다른 테스트 제품군 실행을 시작할 수 있습니다. 이전 섹션에 나

타난 바와 같이 GetSuiteRun API 작업을 사용하여 주기적으로 제품군 실행 상태를 확인할 수 있습니다.

SDK 예제:

```
// Using the SDK, call the StopSuiteRun API.

response = iotDeviceAdvisorClient.StopSuiteRun(
StopSuiteRun.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

성공적인 자격 테스트 스위트 실행을 위한 자격 보고서 받기

성공적으로 완료되는 자격 테스트 제품군을 실행하는 경우 GetSuiteRunReport API 작업을 사용하여 자격 보고서를 검색할 수 있습니다. 이 자격 보고서를 사용하여 AWS IoT Core 자격 프로그램으로 디바이스에 자격을 부여합니다. 테스트 스위트가 자격 테스트 스위트인지 여부를 확인하려면 intendedForQualification 파라미터가 true로 설정되어 있는지 확인하세요. GetSuiteRunReport API 작업을 호출한 후 반환된 URL에서 최대 90초 동안 보고서를 다운로드할 수 있습니다. 이전 GetSuiteRunReport API 작업을 호출한 때부터 90초 이상 경과했으면 작업을 다시 호출하여 유효한 URL을 새로 검색합니다.

SDK 예제:

```
// Using the SDK, call the getSuiteRunReport API.

response = iotDeviceAdvisorClient.getSuiteRunReport(
    GetSuiteRunReportRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .suiteRunId("your-suite-run-id")
        .build()
)
```

Device Advisor 세부 콘솔 워크플로

이 자습서에서는 사용자 지정 테스트 스위트를 만들고 콘솔에서 테스트할 디바이스에 대해 테스트를 실행합니다. 테스트가 완료되면 테스트 결과와 세부 로그를 볼 수 있습니다.

자습서

- 사전 조건
- 테스트 스위트 정의 만들기
- 테스트 스위트 실행 시작하기
- 테스트 스위트 실행 중지(선택 사항)
- 테스트 스위트 실행 세부 정보 및 로그 보기
- AWS IoT 검증 보고서 다운로드

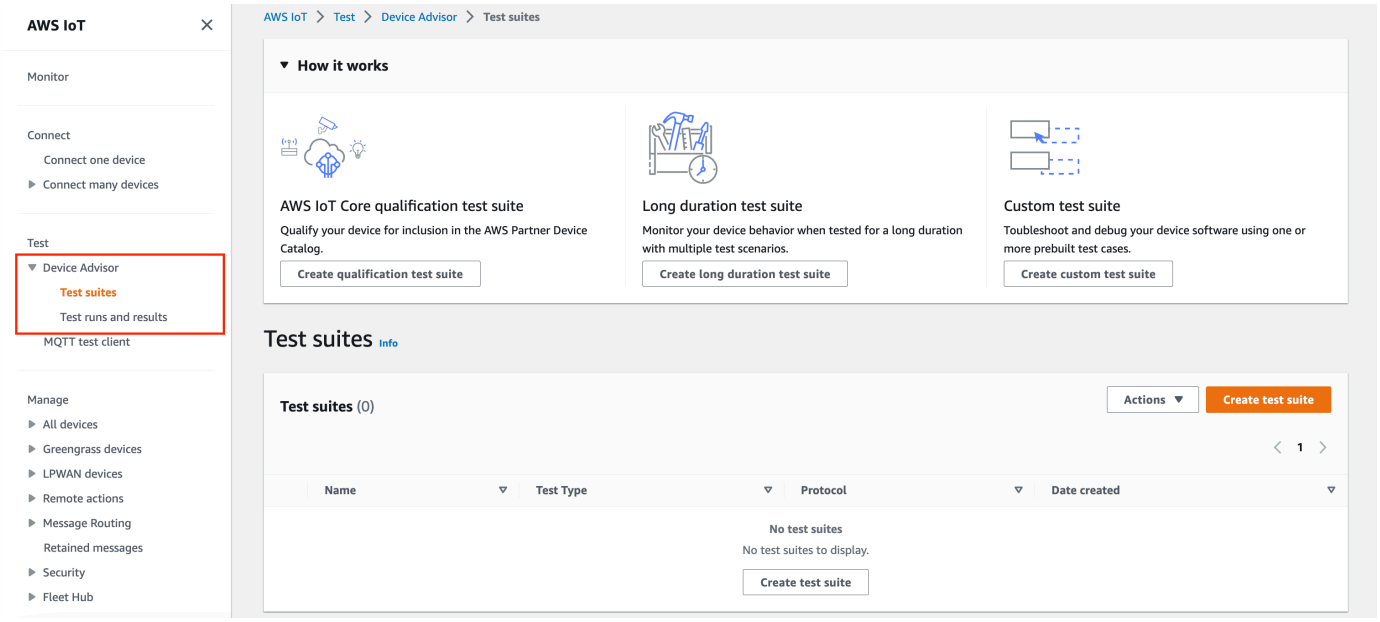
사전 조건

이 튜토리얼을 완료하려면 사물 및 인증서를 생성해야 합니다.

테스트 스위트 정의 만들기

테스트 제품군을 생성하여 디바이스에 대해 실행하고 확인을 수행할 수 있습니다.

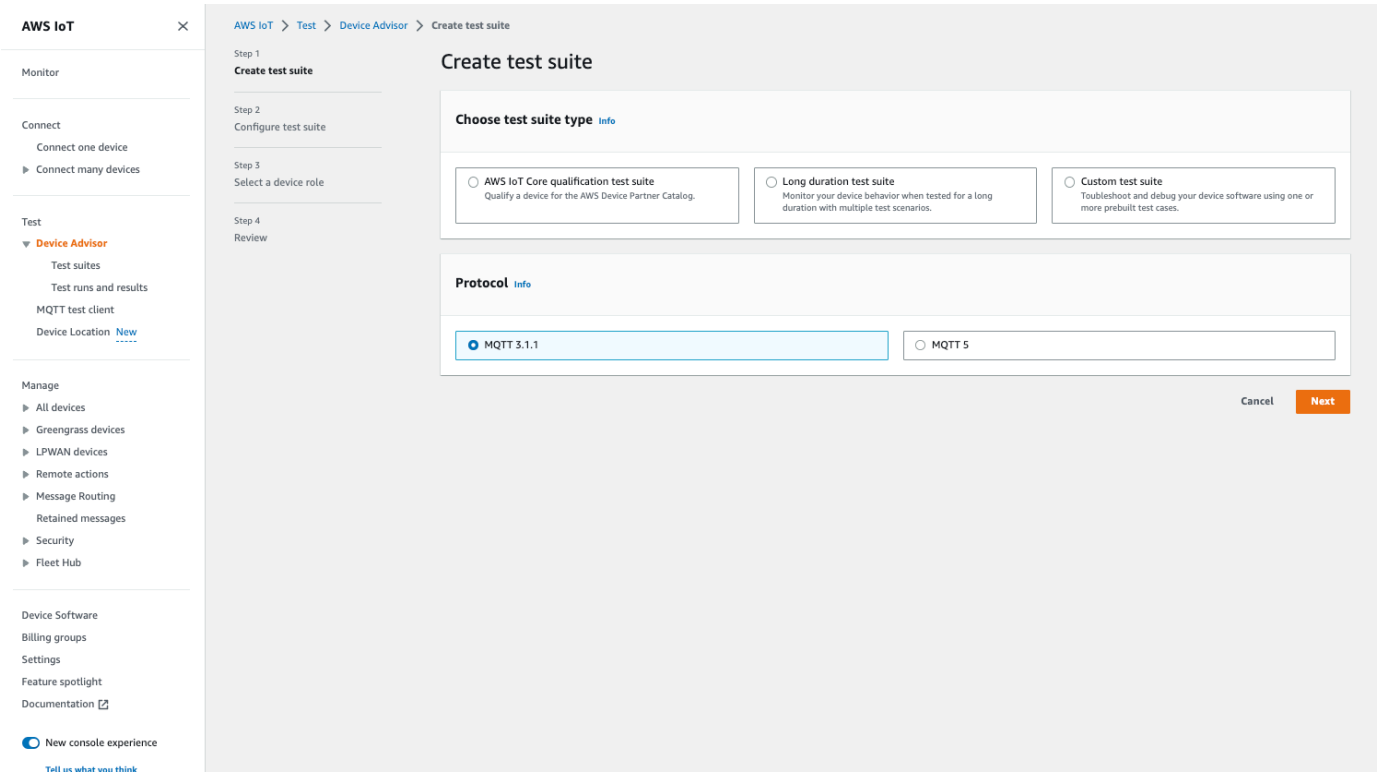
1. AWS IoT 콘솔의 탐색 창에서 테스트(Test)를 확장하고,Device Advisor를 확장한 다음 테스트 스위트(Test suites)를 선택합니다.



   테스트 스위트 생성(Create test suite)을 선택합니다.

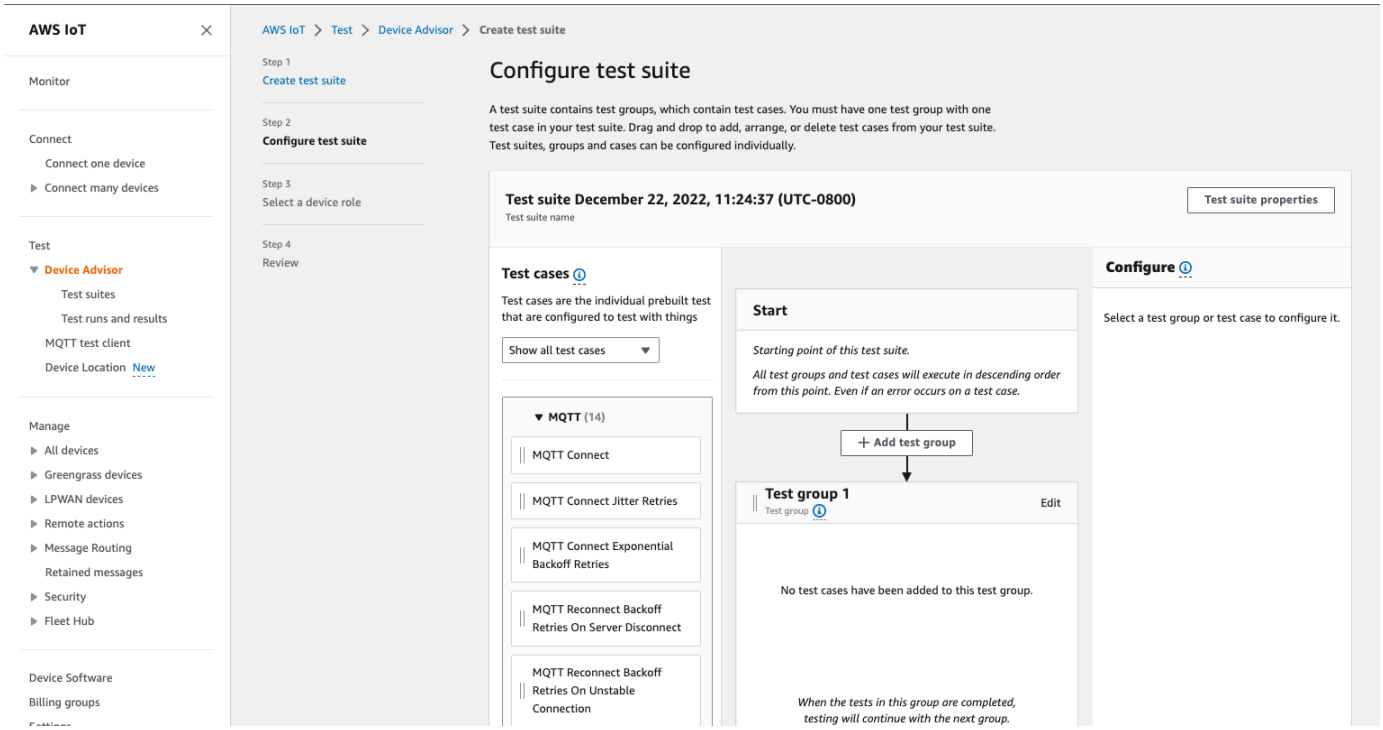
2. `Use the AWS Qualification test suite` 또는 `Create a new test suite`을 선택합니다.

   프로토콜의 경우 MQTT 3.1.1 또는 MQTT 5를 선택합니다.

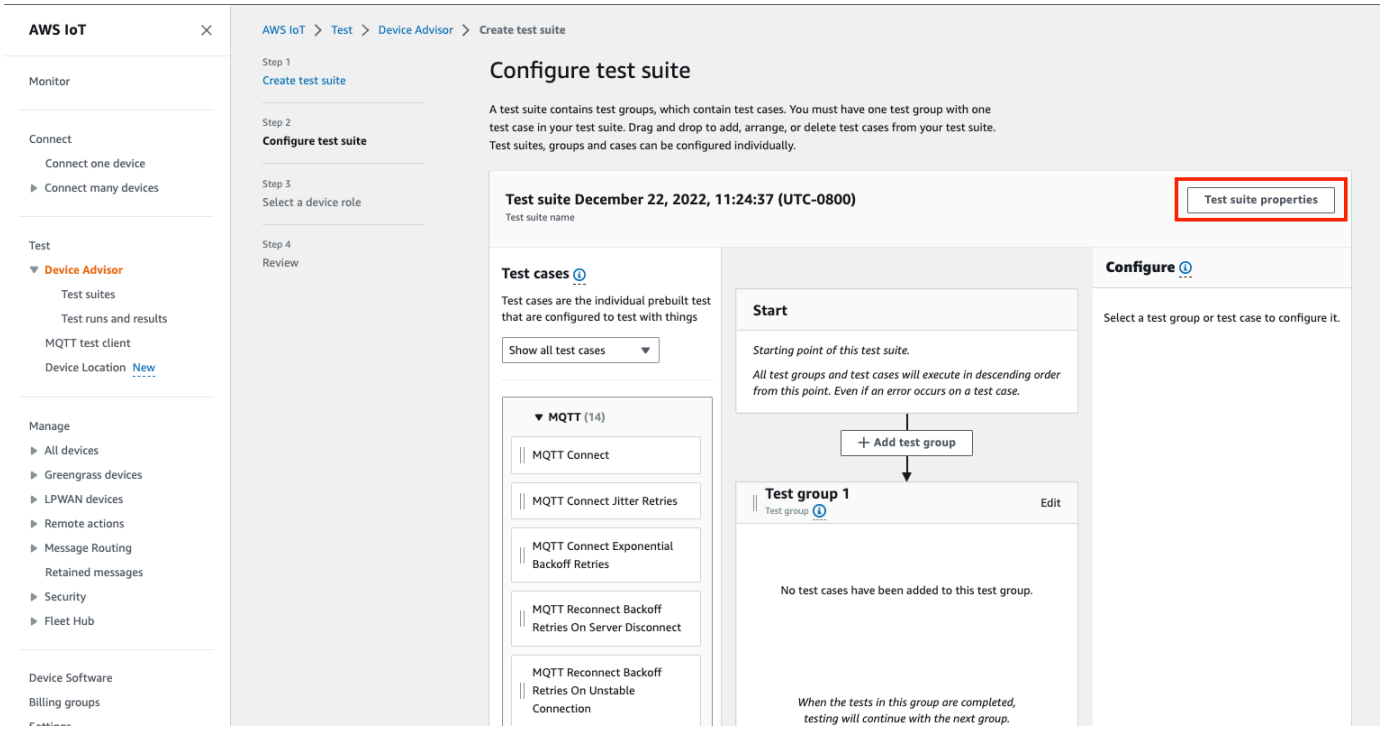


`Use the AWS Qualification test suite`를 선택하여 디바이스를 검증하고 AWS 파트너 디바이스 카탈로그에 나열합니다. 이 옵션을 선택하면 AWS IoT Core 자격 프로그램에 디바이스의 자격을 검증하는 데 필요한 테스트 케이스가 미리 선택됩니다. 테스트 그룹 및 테스트 케이스는 추가하거나 제거할 수 없습니다. 여전히 테스트 스위트 속성을 구성해야 합니다.

`Create a new test suite` 옵션을 선택하여 사용자 정의 테스트 스위트를 생성하고 구성합니다. 초기 테스트 및 문제 해결을 위해 이 옵션으로 시작하는 것이 좋습니다. 사용자 지정 테스트 스위트에는 하나 이상의 테스트 그룹이 있어야 하며 각 테스트 그룹에는 하나 이상의 테스트 케이스가 있어야 합니다. 이 자습서에서의 목적에 맞게 이 옵션을 선택하고 다음(Next)을 선택합니다.



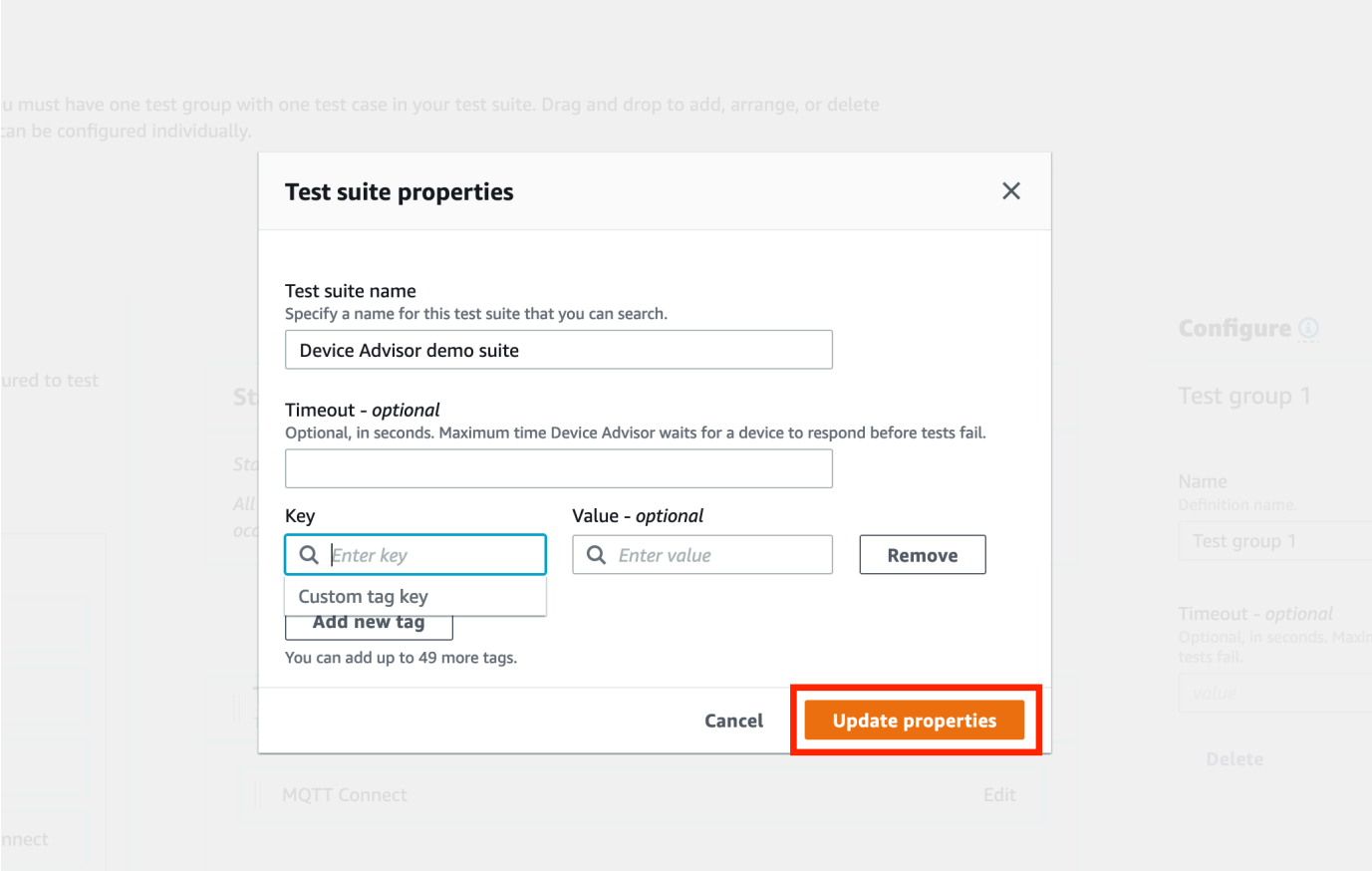
3. 테스트 스위트 속성(Test suite properties)을 선택합니다. 테스트 스위트를 생성할 때 테스트 스위트 속성을 생성해야 합니다.



테스트 스위트 속성(Test suite properties)에서 다음을 작성합니다.

- 테스트 스위트 이름: 사용자 지정 이름으로 스위트를 생성할 수 있습니다.

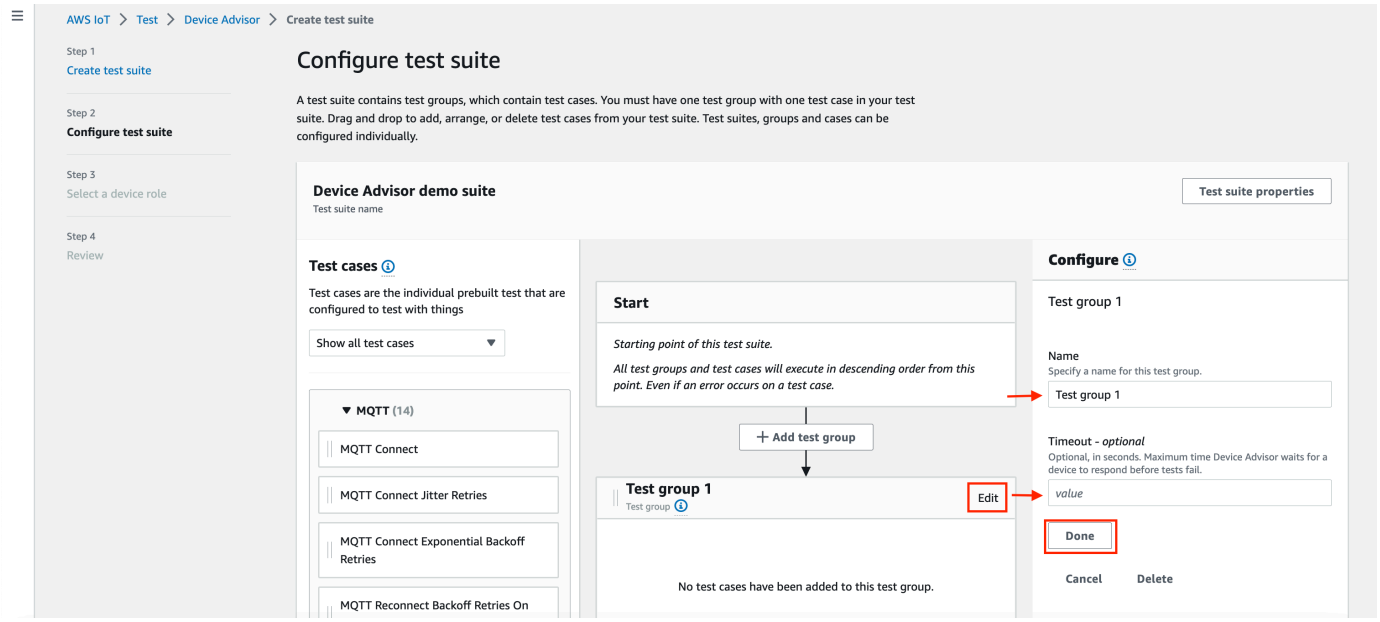
- 제한 시간(Timeout)(선택 사항): 현재 테스트 스위트의 각 테스트 케이스에 대한 제한 시간(초)입니다. 제한 시간 값을 지정하지 않으면 기본값이 사용됩니다.
- 태그(Tags)(선택 사항): 테스트 제품군에 태그를 추가합니다.



완료했으면 속성 업데이트(Update properties)를 선택합니다.

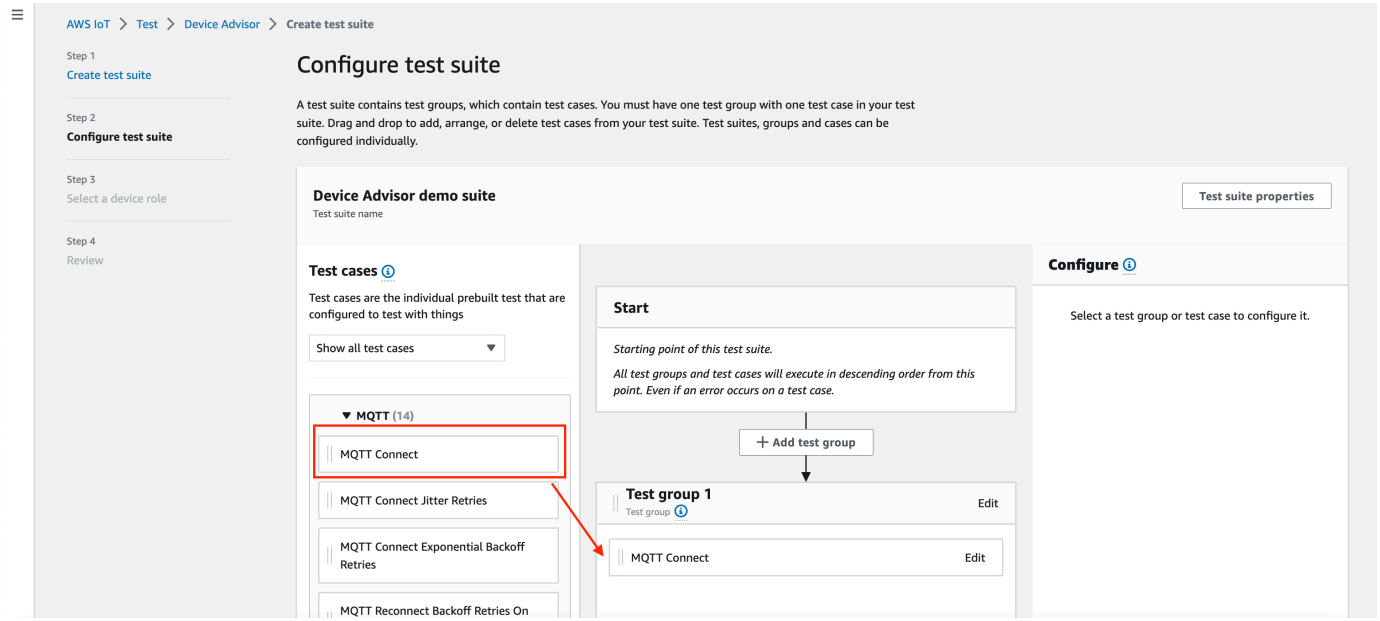
4. 그룹 수준 구성을 수정하려면 `Test group 1`에서 편집을 선택합니다. 그런 다음 이름을 입력하여 그룹에 사용자 지정 이름을 부여합니다.

   선택 사항으로 선택한 테스트 그룹 아래에서 제한 시간(Timeout)(초 단위)을 입력할 수도 있습니다. 제한 시간 값을 지정하지 않으면 기본값이 사용됩니다.



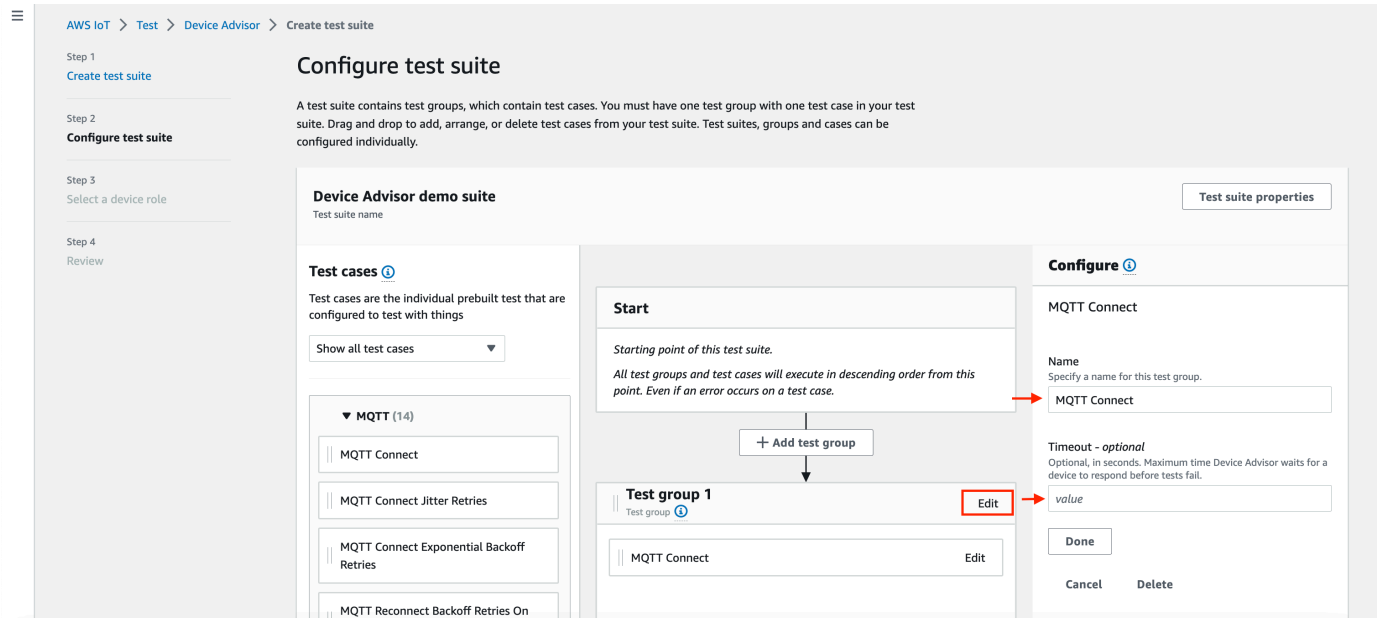
완료를 선택합니다.

5. 사용 가능한 테스트 케이스 중 하나를 테스트 케이스(Test cases)에서 테스트 그룹으로 끌어옵니다.



6. 테스트 그룹에 추가한 테스트 케이스의 테스트 케이스 수준 구성을 수정하려면 편집(Edit)을 선택합니다. 그런 다음 이름을 입력하여 그룹에 사용자 지정 이름을 부여합니다.

   선택 사항으로 선택한 테스트 그룹 아래에서 제한 시간(Timeout)(초 단위)을 입력할 수도 있습니다. 제한 시간 값을 지정하지 않으면 기본값이 사용됩니다.

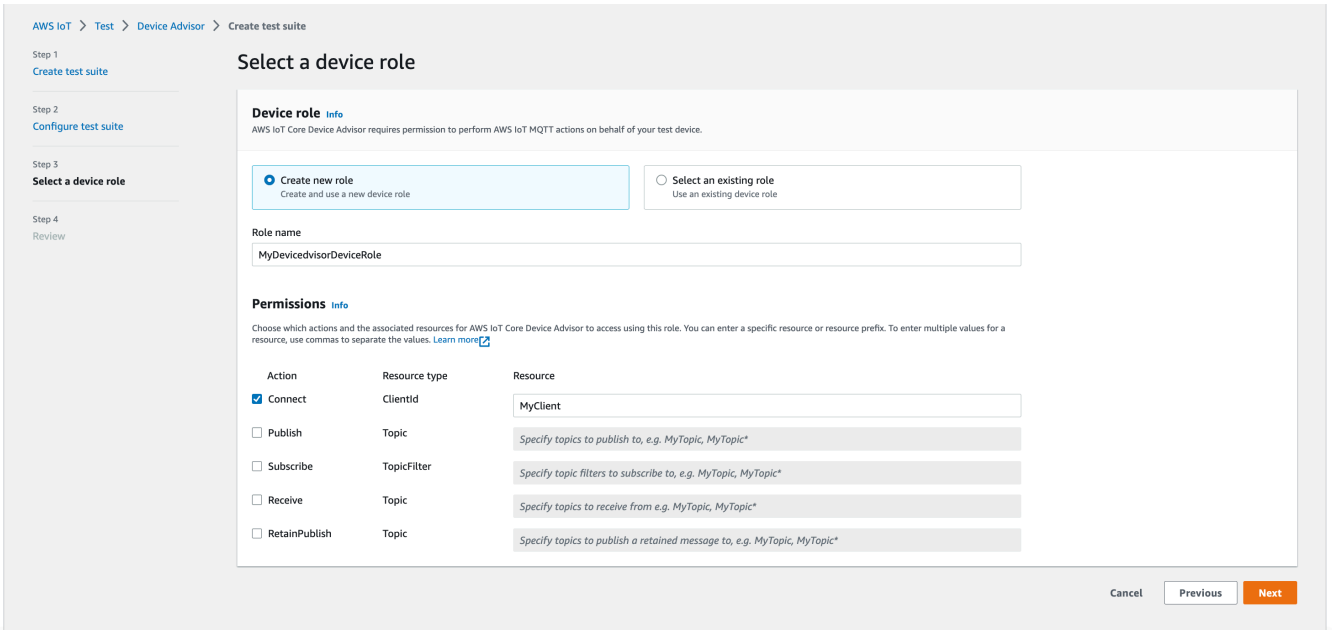


완료를 선택합니다.

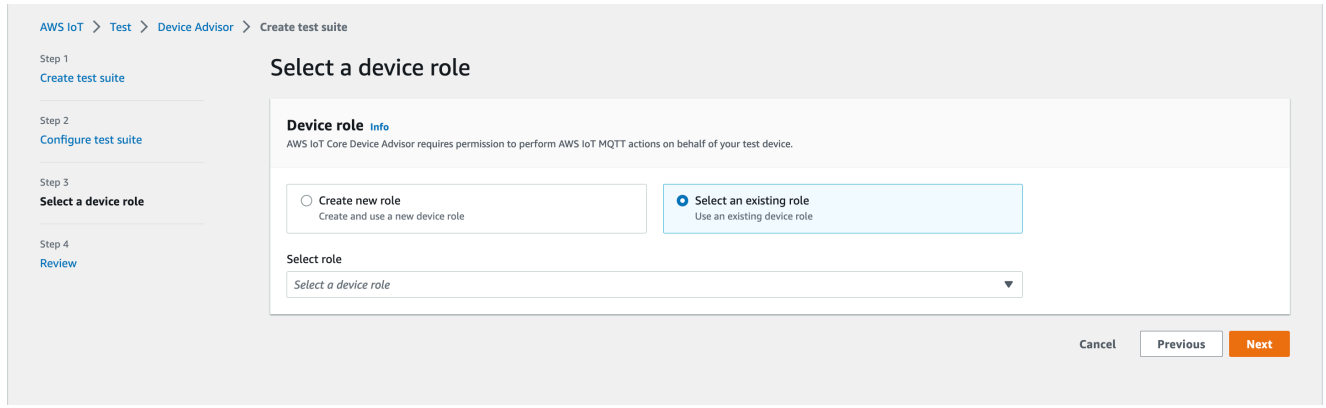
> **Note**
>
> 테스트 스위트에 더 많은 테스트 그룹을 추가하려면 테스트 그룹 추가(Add test group)를 선택합니다. 위의 단계에 따라 더 많은 테스트 그룹을 생성 및 구성하거나 하나 이상의 테스트 그룹에 더 많은 테스트 케이스를 추가합니다. 테스트 케이스를 선택하고 원하는 위치로 끌어서 테스트 그룹 및 테스트 케이스를 재정렬할 수 있습니다. Device Advisor는 테스트 그룹 및 테스트 케이스를 정의하는 순서대로 테스트를 실행합니다.

7. 다음을 선택합니다.
8. 3단계에서 Device Advisor가 테스트 디바이스를 대신하여 AWS IoT MQTT 작업을 수행하는 데 사용할 디바이스 역할을 구성합니다.

   2단계(Step 2)에서 MQTT Connect 테스트 케이스만 선택한 경우 이 테스트 스위트를 실행하기 위한 장치 역할에 해당 권한이 필요하므로 Connect 작업이 자동으로 확인됩니다. 다른 테스트 케이스를 선택한 경우 해당 필수 작업이 확인됩니다. 각 작업에 대한 리소스 값이 제공되는지 확인합니다. 예를 들어 Connect 작업의 경우 장치가 Device Advisor 엔드포인트와 연결할 클라이언트 ID를 제공합니다. 쉼표로 값을 구분하여 여러 값을 제공할 수 있으며 와일드카드(*) 문자를 사용하여 접두사 값을 제공할 수도 있습니다. 예를 들어, `MyTopic`으로 시작하는 주제에 대한 게시 권한을 제공하려면 리소스 값으로 “`MyTopic*`”을 제공합니다.



이전에 장치 역할을 이미 생성했고 해당 역할을 사용하려면 기존 역할 선택(Select an existing role)을 선택하고 역할 선택(Select role)에서 장치 역할을 선택합니다.



제공된 두 옵션 중 하나를 사용하여 디바이스 역할을 구성하고 다음(Next)을 선택합니다.

9. 4단계(Step 4)에서 각 단계에서 제공되는 구성이 정확한지 확인합니다. 특정 단계에 대해 제공된 구성을 편집하려면 해당 단계에 대한 편집(Edit)을 선택합니다.

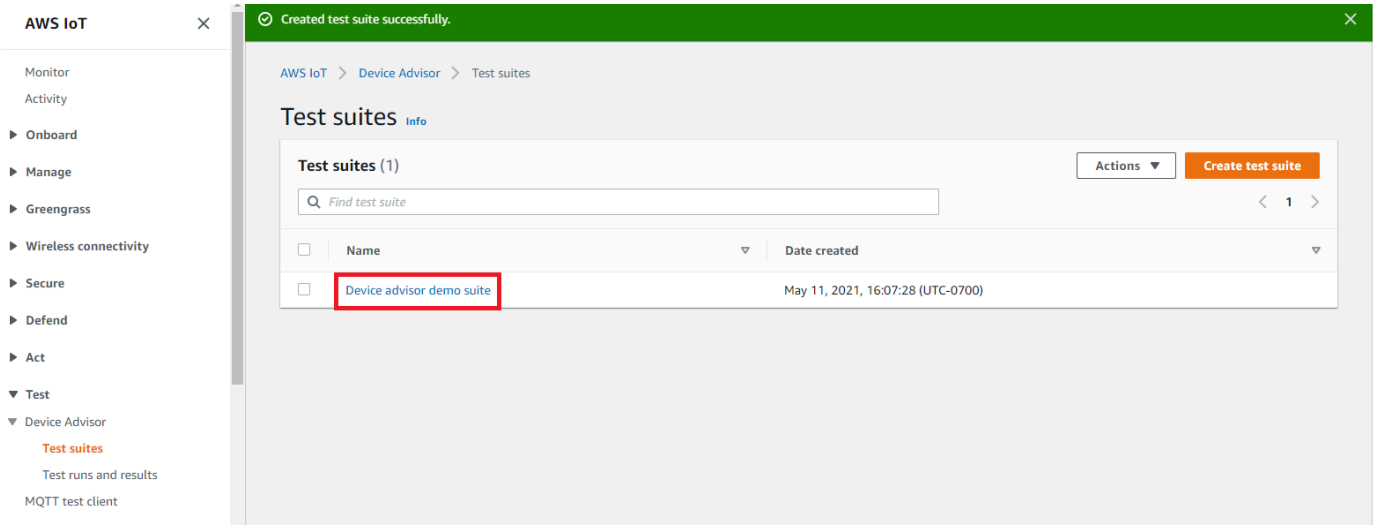
   구성을 확인한 후 테스트 제품군 생성(Create test suite)을 선택합니다.

   테스트 스위트가 성공적으로 생성되면 생성된 모든 테스트 스위트를 볼 수 있는 테스트 스위트(Test suites) 페이지로 리디렉션됩니다.

   테스트 스위트 생성이 실패한 경우 이전 지침에 따라 테스트 스위트, 테스트 그룹, 테스트 케이스 및 디바이스 역할이 구성되었는지 확인합니다.

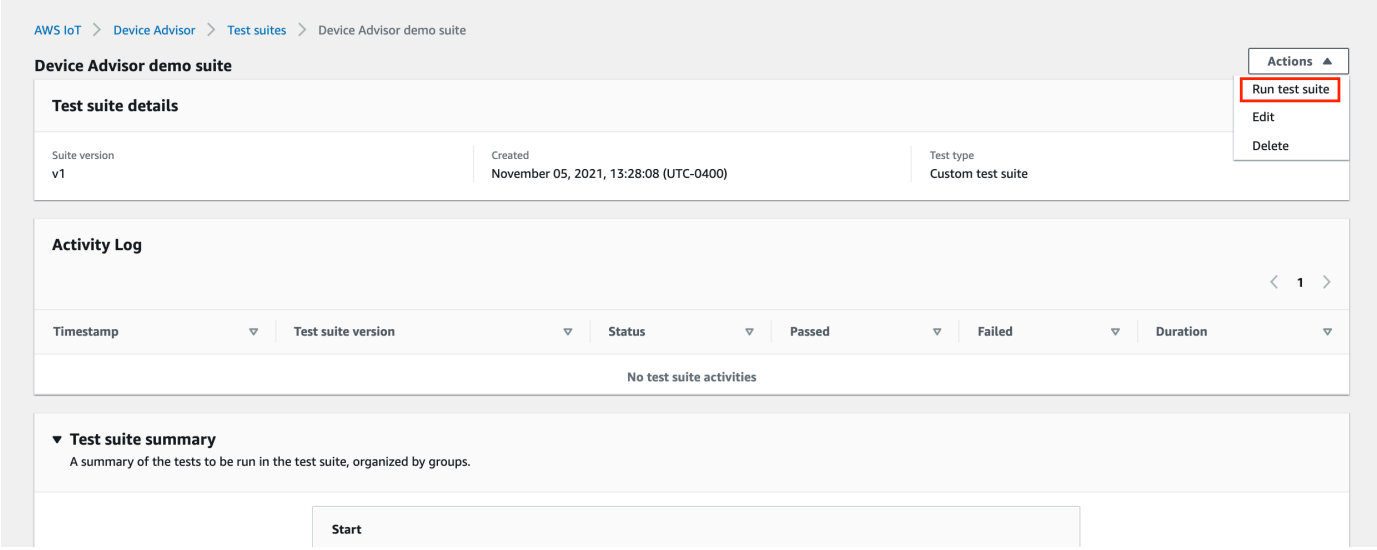
테스트 스위트 실행 시작하기

1. AWS IoT 콘솔의 탐색 창에서 테스트, Device Advisor를 차례로 확장한 다음 테스트 스위트를 선택합니다.
2. 테스트 스위트 세부 정보를 보려는 테스트 스위트를 선택합니다.



테스트 스위트 세부 정보 페이지에는 테스트 스위트와 관련된 모든 정보가 표시됩니다.

3. 작업(Actions)을 선택한 다음테스트 스위트 실행(Run test suite)을 선택합니다.



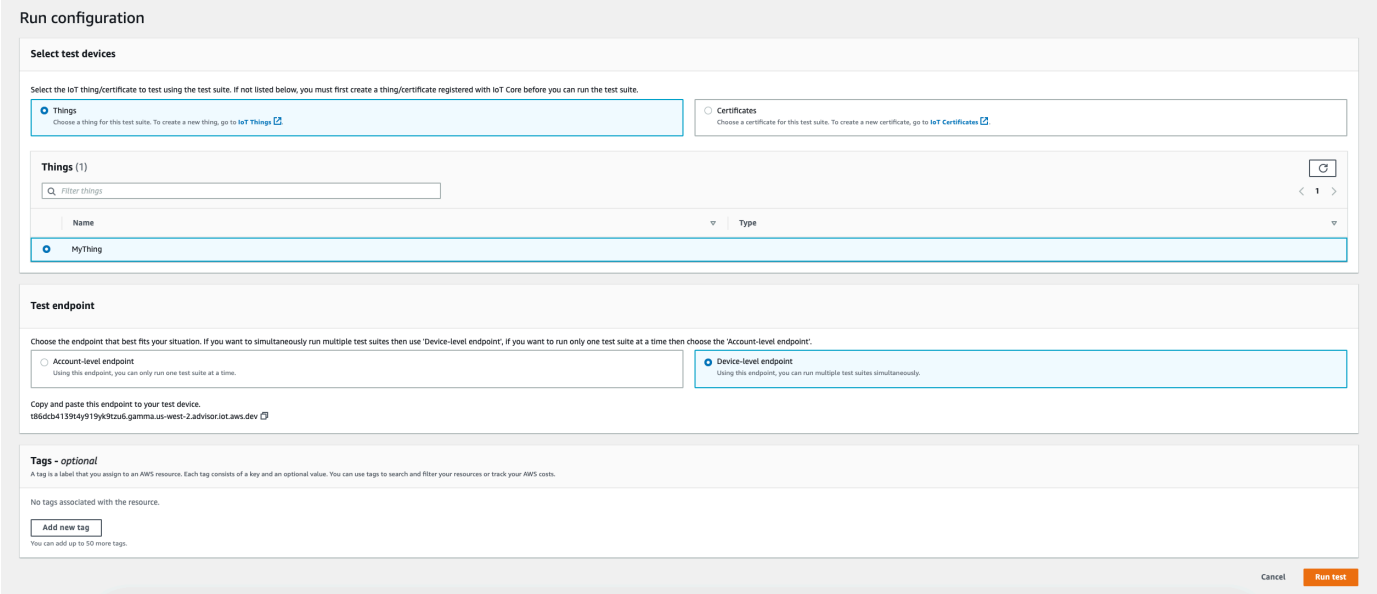
4. 구성 실행에서 Device Advisor를 사용하여 테스트할 AWS IoT 사물 또는 인증서를 선택해야 합니다. 기존 사물이나 인증서가 없는 경우 먼저 AWS IoT Core 리소스를 생성합니다.

   테스트 엔드포인트(Test endpoint) 섹션에서 케이스에 가장 적합한 엔드포인트를 선택합니다. 나중에 동일한 AWS 계정을 사용하여 여러 테스트 제품군을 동시에 실행하려는 경우 디바이스 수준

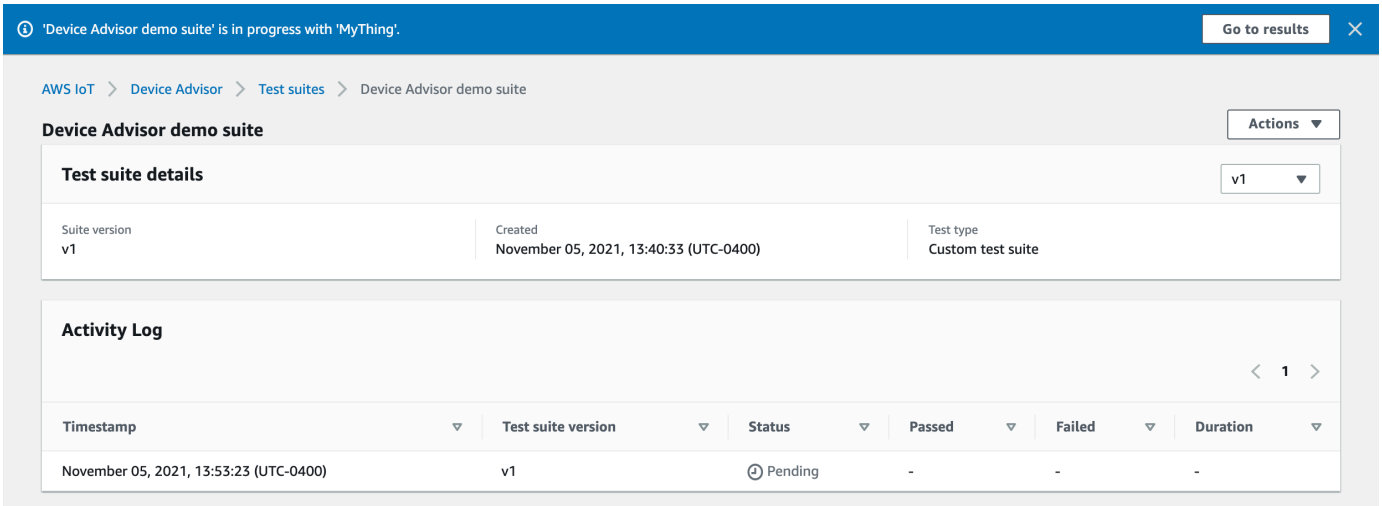
엔드포인트를 선택합니다. 한 번에 하나의 테스트 스위트만 실행할 계획인 경우 계정 수준 엔드포인트(Account-level endpoint)를 선택합니다.

선택한 Device Advisor의 테스트 엔드포인트로 테스트 디바이스를 구성합니다.

사물 또는 인증서를 선택하고 Device Advisor 엔드포인트를 선택한 후 테스트 실행(Run test)을 선택합니다.



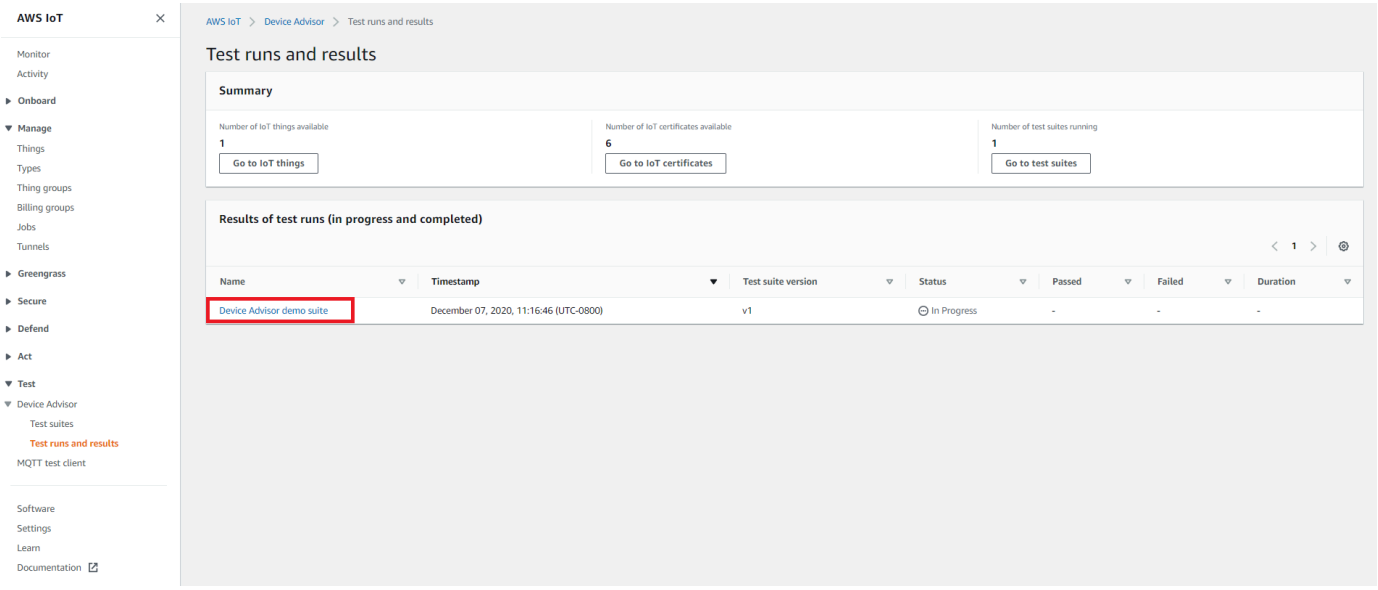
5. 테스트 실행 세부 정보를 보려면 상단 배너의 결과로 이동(Go to results)을 선택합니다.



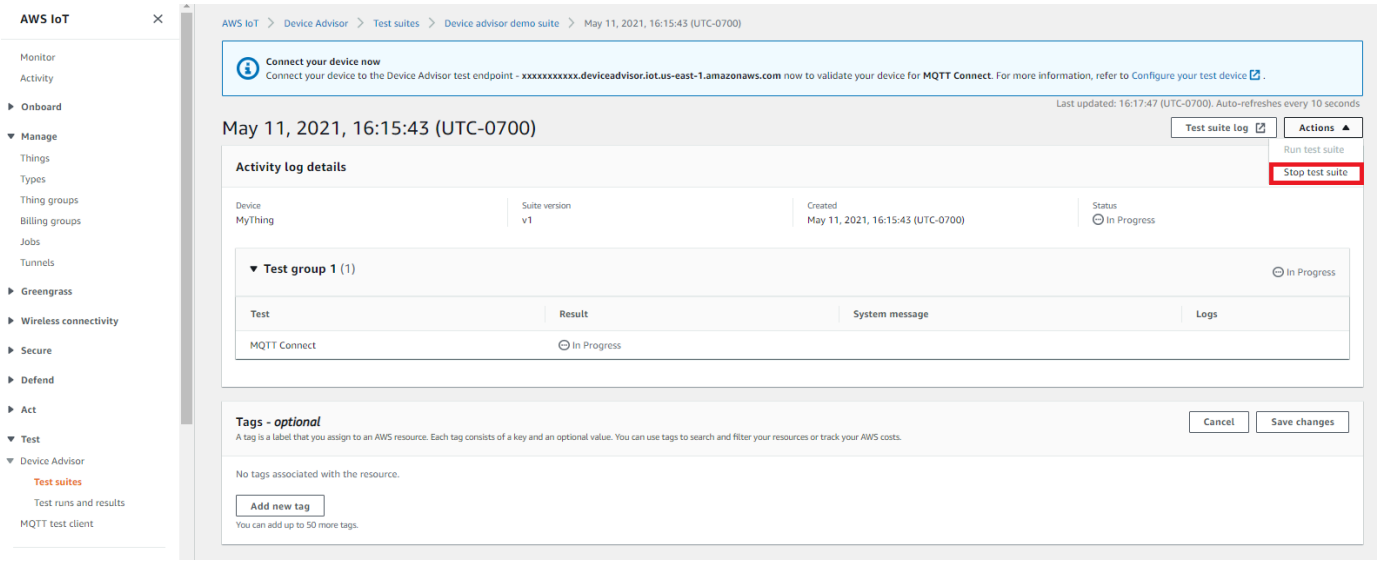
테스트 스위트 실행 중지(선택 사항)

1. AWS IoT 콘솔의 탐색 창에서 테스트, Device Advisor를 차례로 확장한 다음 테스트 실행 및 결과를 선택합니다.

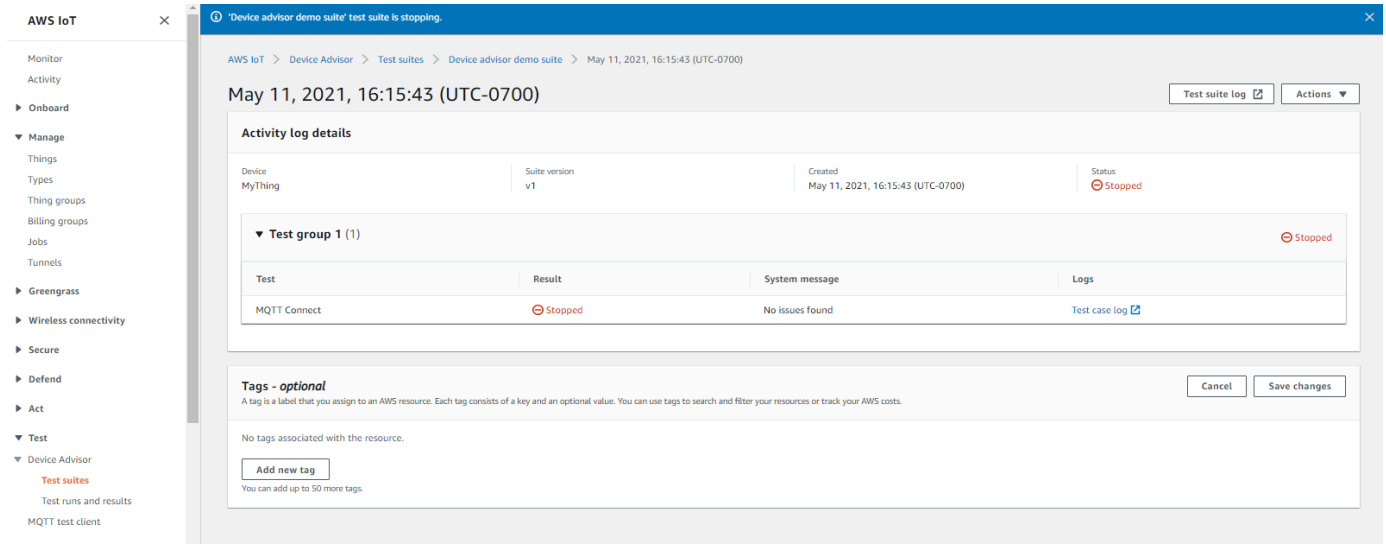
2. 중지하려는 진행 중인 테스트 스위트를 선택합니다.



3. 작업(Actions)을 선택한 다음 테스트 스위트 중지(Stop test suite)를 선택합니다.



4. 이 정리 프로세스는 완료하는 데 몇 분 정도 걸립니다. 정리 프로세스를 실행하는 동안 테스트 실행 상태는 `STOPPING`이 됩니다. 새 스위트 실행을 시작하기 전에 정리 프로세스가 완료되고 테스트 스위트 상태가 `STOPPED` 상태로 변경될 때까지 기다립니다.

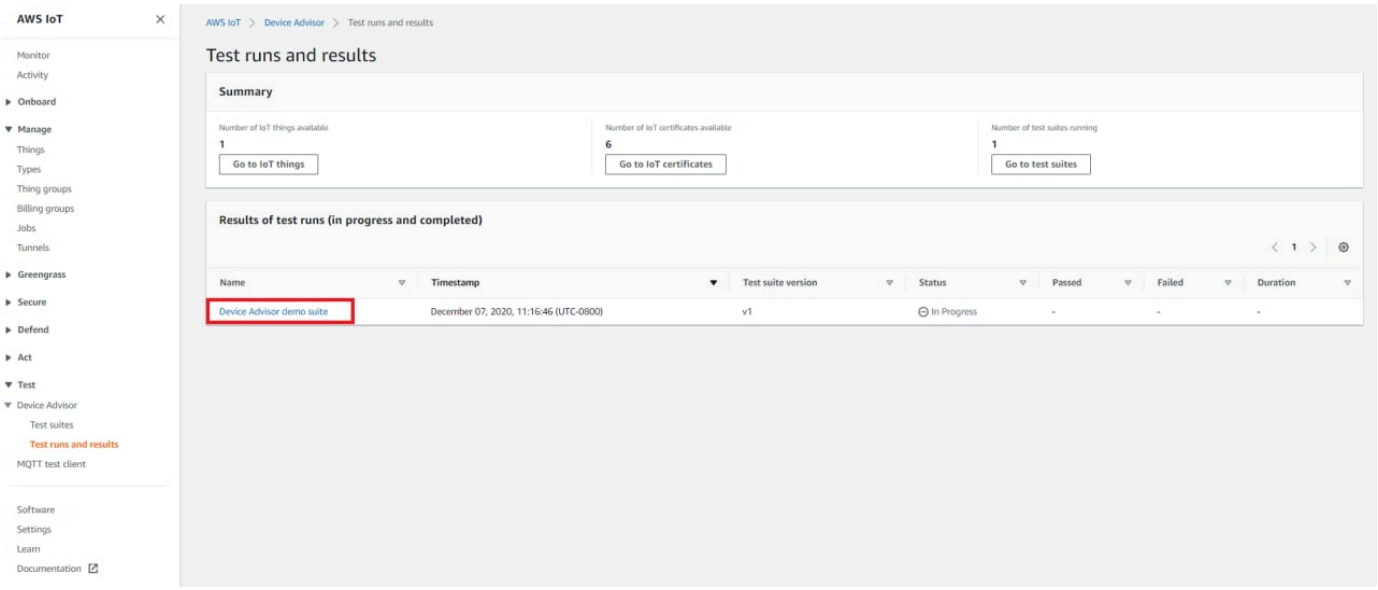


테스트 스위트 실행 세부 정보 및 로그 보기

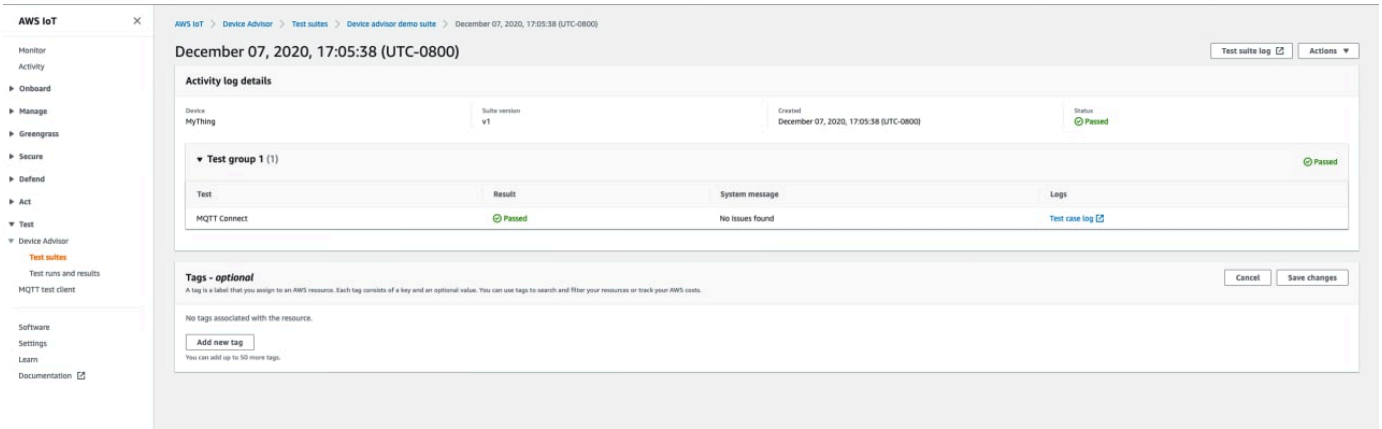
1. AWS IoT 콘솔의 탐색 창에서 테스트, Device Advisor를 차례로 확장한 다음 테스트 실행 및 결과를 선택합니다.

   이 페이지에는 다음이 표시됩니다.

   - IoT 사물의 수
   - IoT 인증서 수
   - 현재 실행 중인 테스트 스위트 수
   - 생성된 모든 테스트 스위트 실행
2. 실행 세부 정보와 로그를 보려는 테스트 스위트를 선택합니다.



실행 요약 페이지에는 현재 테스트 스위트 실행의 상태가 표시됩니다. 이 페이지는 10초마다 자동으로 새로 고쳐집니다. 디바이스에서 테스트 엔드포인트에 1~2분 동안 5초마다 연결을 시도할 수 있도록 구성된 메커니즘을 사용하는 것이 좋습니다. 그런 다음 자동화된 방식으로 여러 테스트 케이스를 순서대로 실행할 수 있습니다.



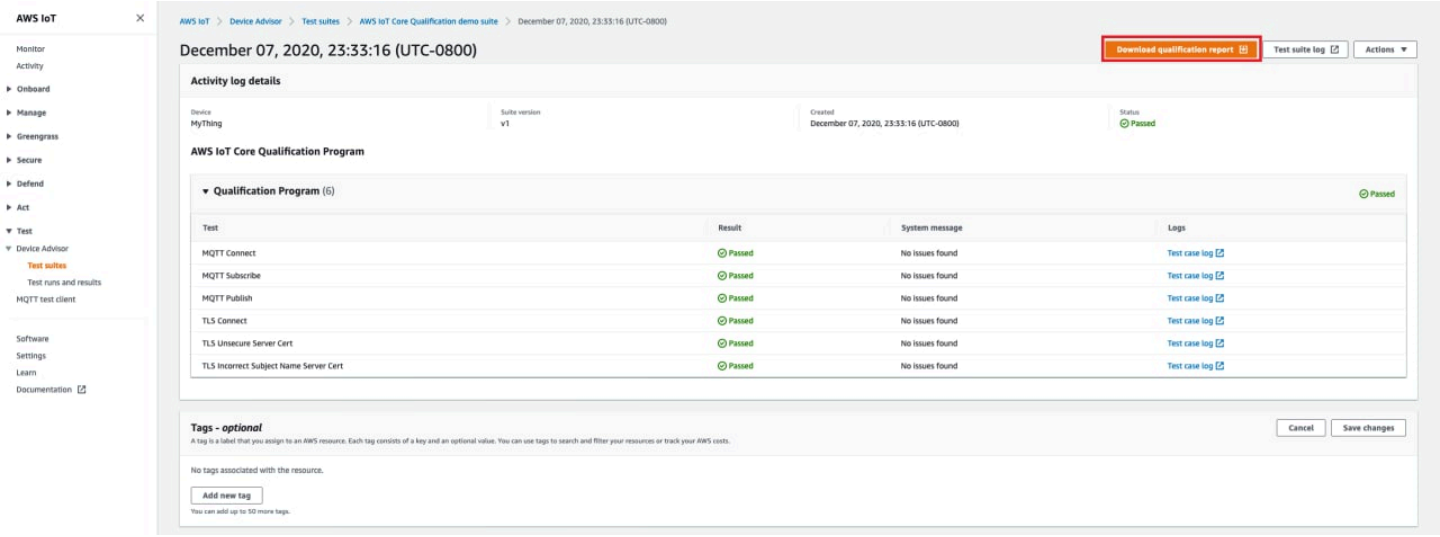
3. 테스트 스위트 실행에 대한 CloudWatch 로그에 액세스하려면 테스트 스위트 로그(Test suite log)를 선택하세요.

   모든 테스트 케이스에 대한 CloudWatch 로그에 액세스하려면 테스트 케이스 로그(Test case log)를 선택하세요.

4. 테스트 결과에 따라 모든 테스트가 통과될 때까지 디바이스의 문제를 해결합니다.

AWS IoT 검증 보고서 다운로드

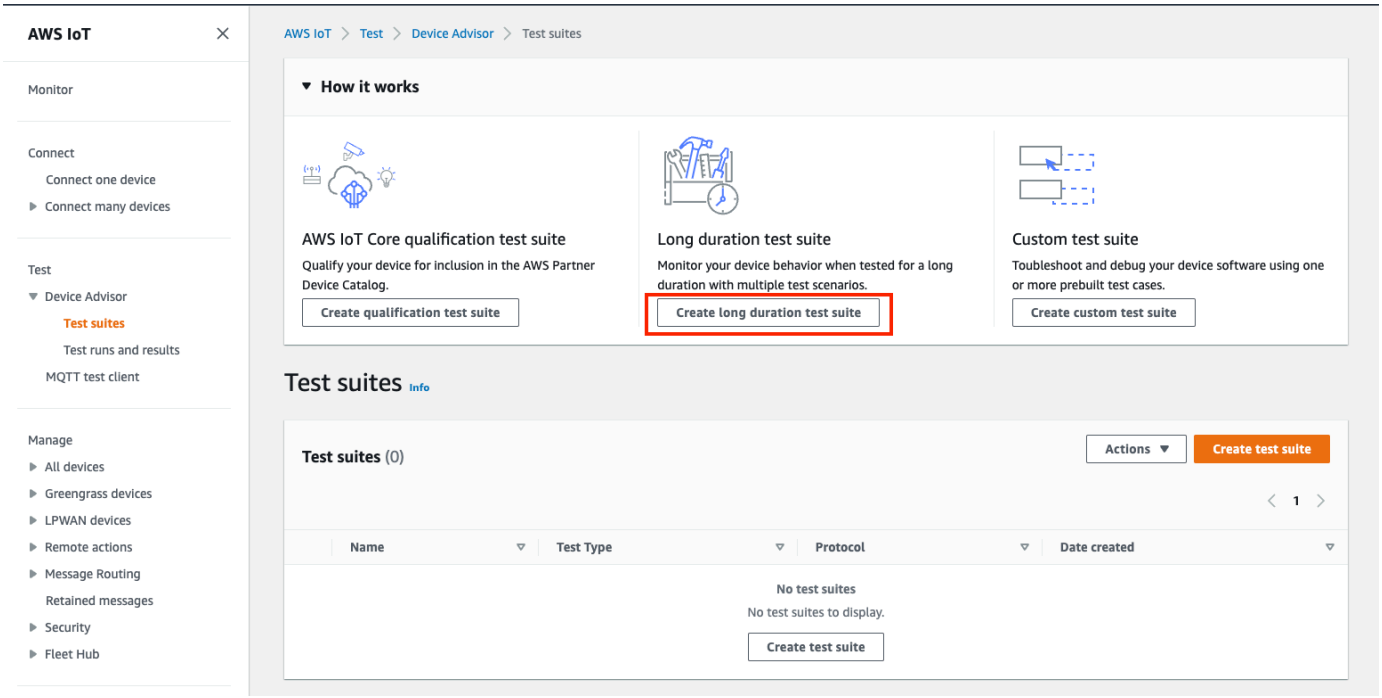
테스트 제품군을 생성하는 동안 AWS IoT 검증 테스트 제품군 사용 옵션을 선택하고 검증 테스트 제품군을 실행할 수 있었던 경우 테스트 실행 요약 페이지에서 검증 보고서 다운로드를 선택하여 검증 보고서를 다운로드할 수 있습니다.



장기간 테스트 콘솔 워크플로

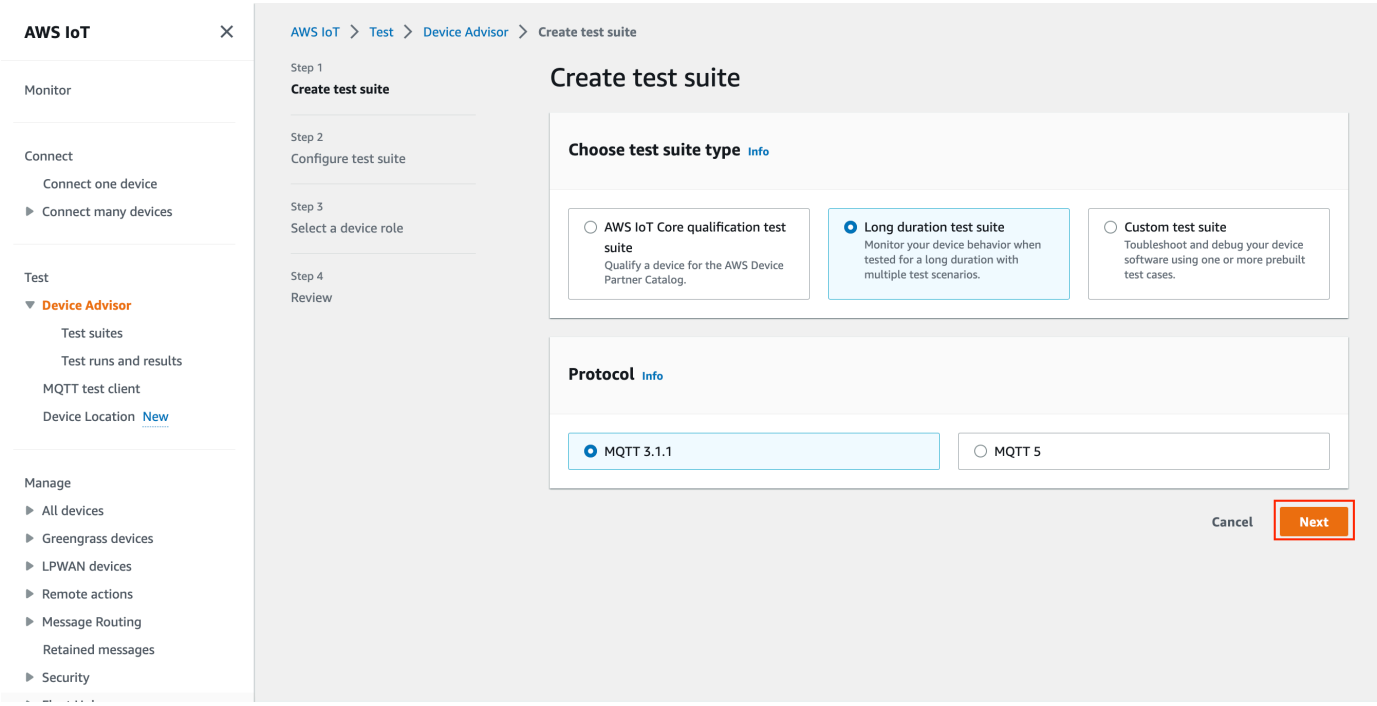
이 자습서는 콘솔을 사용하여 Device Advisor에서 장기간 테스트를 시작하는 데 도움이 됩니다. 자습서를 완료하려면 설정의 단계를 따릅니다.

1. AWS IoT 콘솔 탐색 창에서 Test(테스트)를 확장하고, Device Advisor를 확장한 다음, Test suites(테스트 제품군)를 선택합니다. 페이지에서 Create long duration test suite(장기간 테스트 제품군 생성)를 선택합니다.



2. Create test suite(테스트 제품군 생성) 페이지에서 Long duration test suite(장기간 테스트 제품군)를 선택하고 Next(다음)를 선택합니다.

   프로토콜의 경우 MQTT 3.1.1 또는 MQTT 5를 선택합니다.



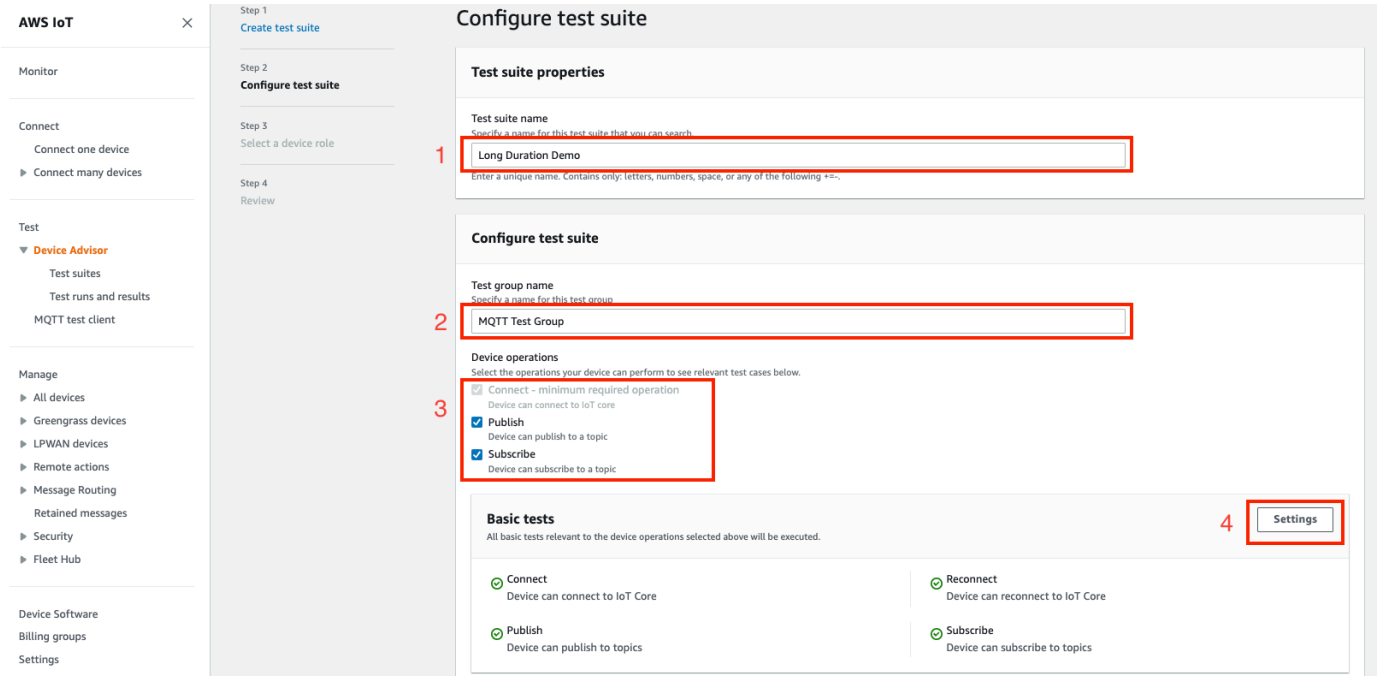
3. Configure test suite(테스트 제품군 구성) 페이지에서 다음 작업을 수행합니다.

   a. Test suite name(테스트 제품군 이름) 필드를 업데이트합니다.

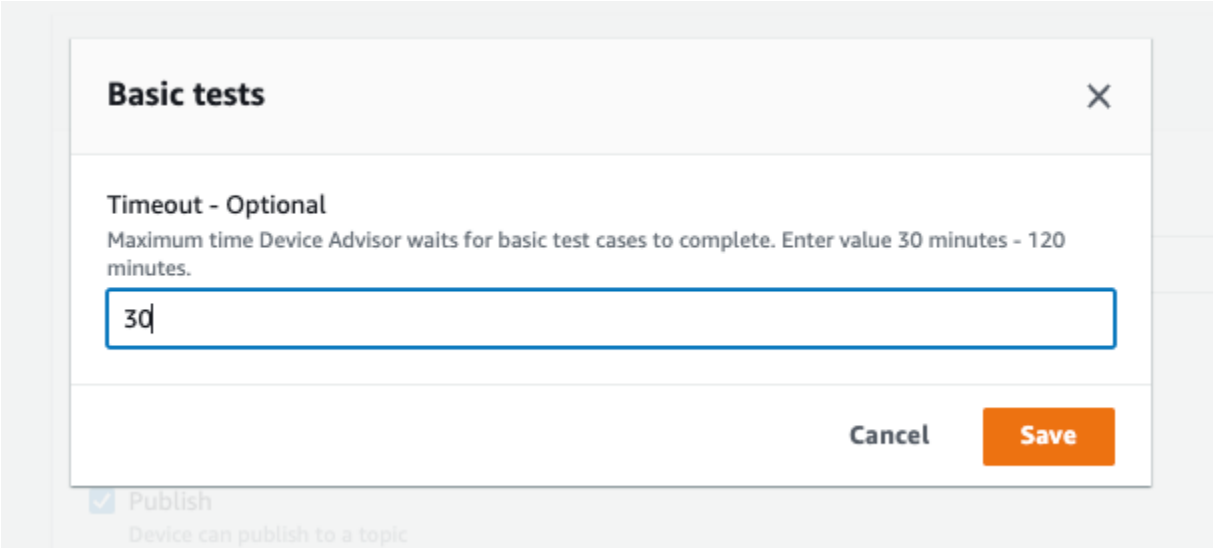
b. Test group name(테스트 그룹 이름) 필드를 업데이트합니다.

c. 디바이스가 수행할 수 있는 Device operations(디바이스 작업)를 선택합니다. 그러면 실행할 테스트가 선택됩니다.

d. Settings(설정) 옵션을 선택합니다.



4. (선택 사항) 기본 테스트가 완료될 때까지 Device Advisor가 기다려야 하는 최대 시간을 입력합니다. 저장을 선택합니다.



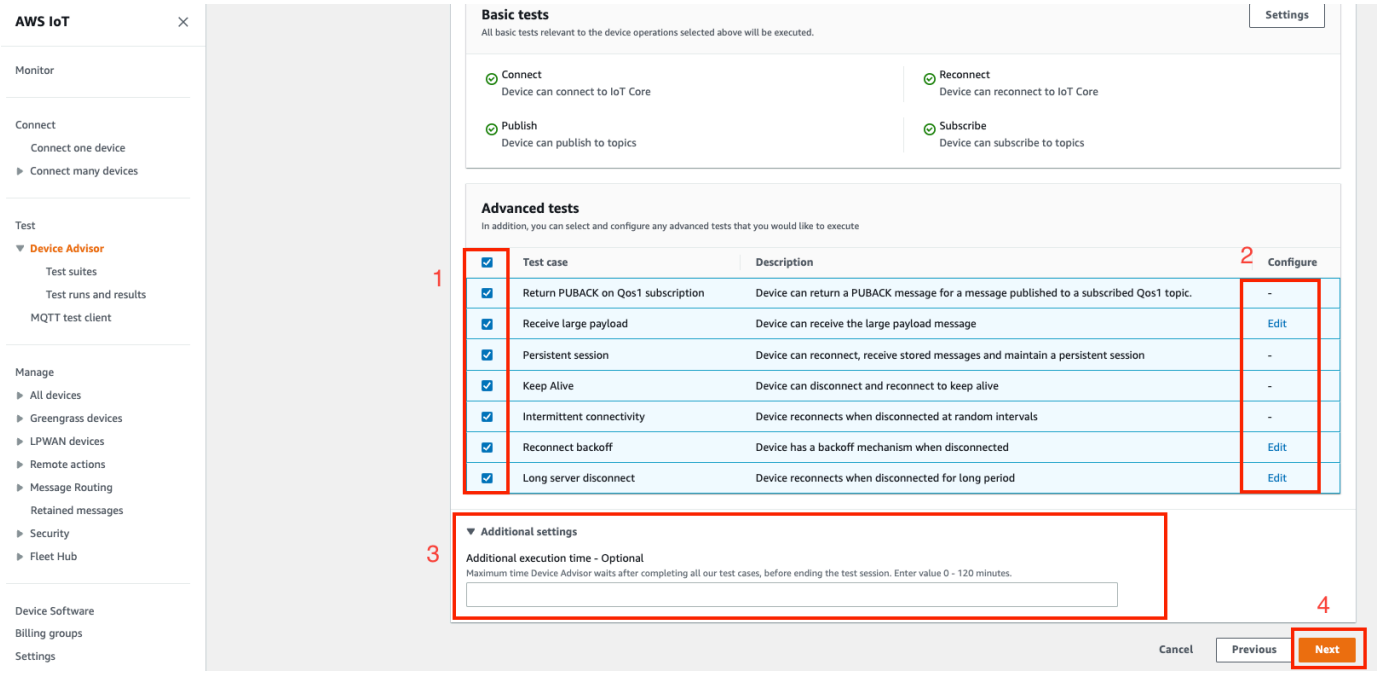
5. Advanced tests(고급 테스트) 및 Additional settings(추가 설정) 섹션에서 다음을 수행합니다.

a. 이 테스트의 일부로 실행할 고급 테스트를 선택하거나 선택 취소합니다.

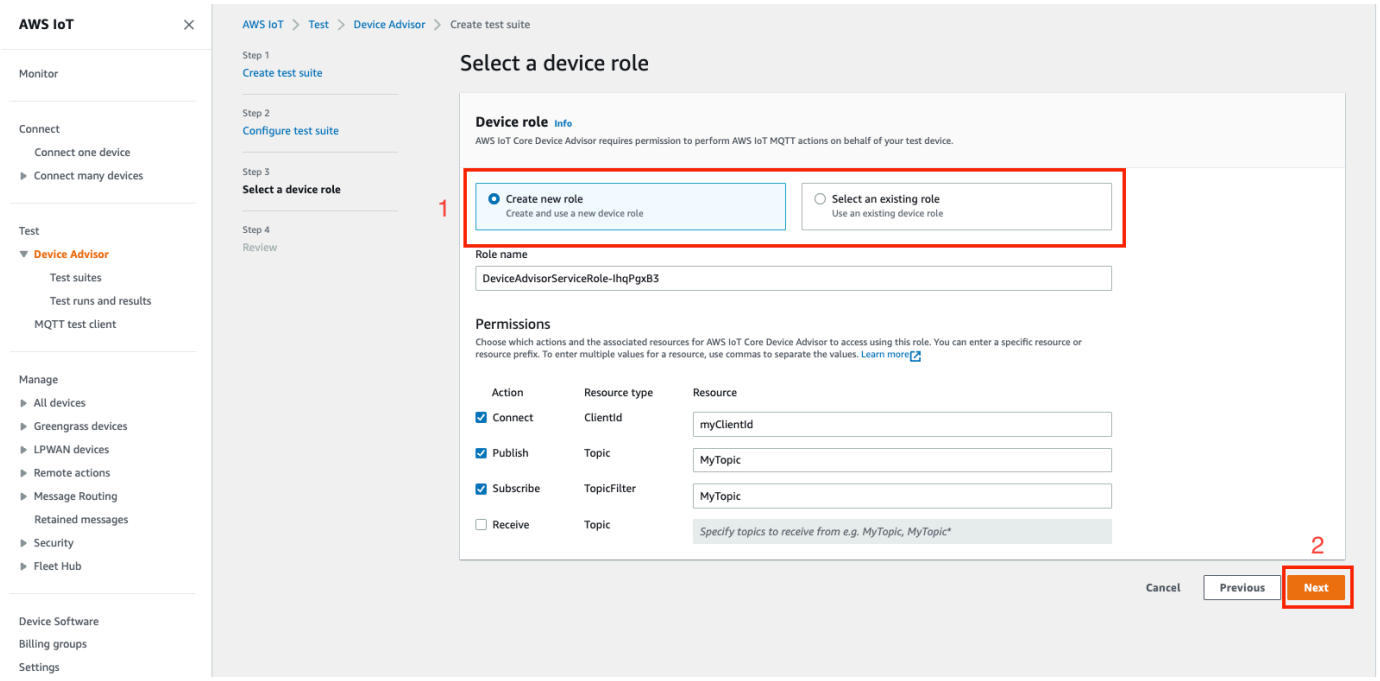
b. 해당하는 경우 테스트에 대한 구성을 편집합니다.

c. Additional settngs(추가 설정) 섹션에서 Additional execution time(추가 실행 시간)을 구성합니다.

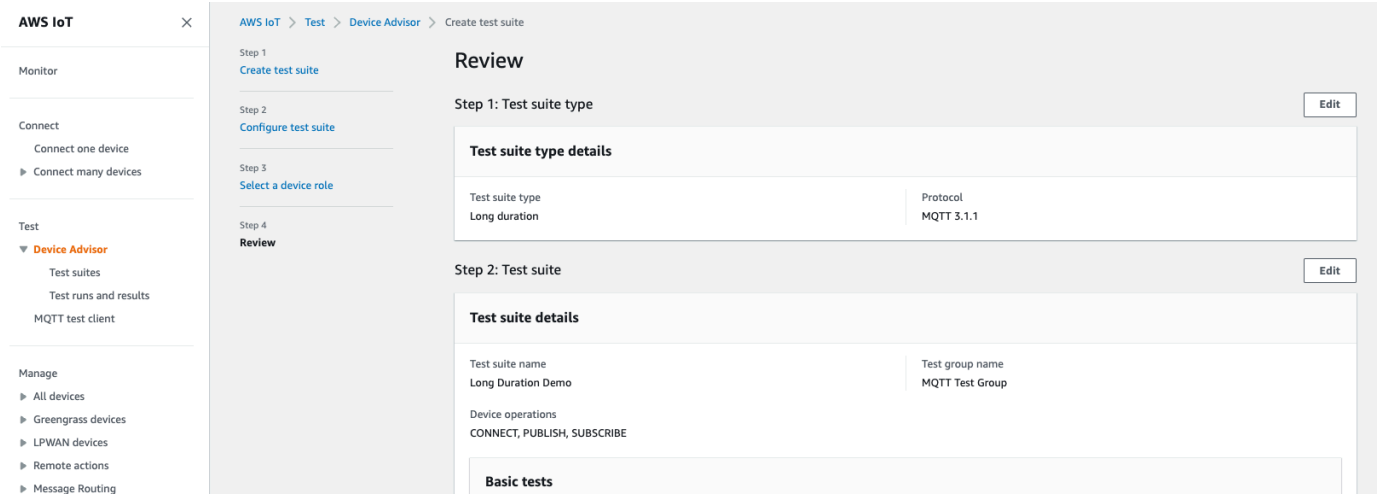
d. Next(다음)를 선택하여 다음 단계로 진행합니다.

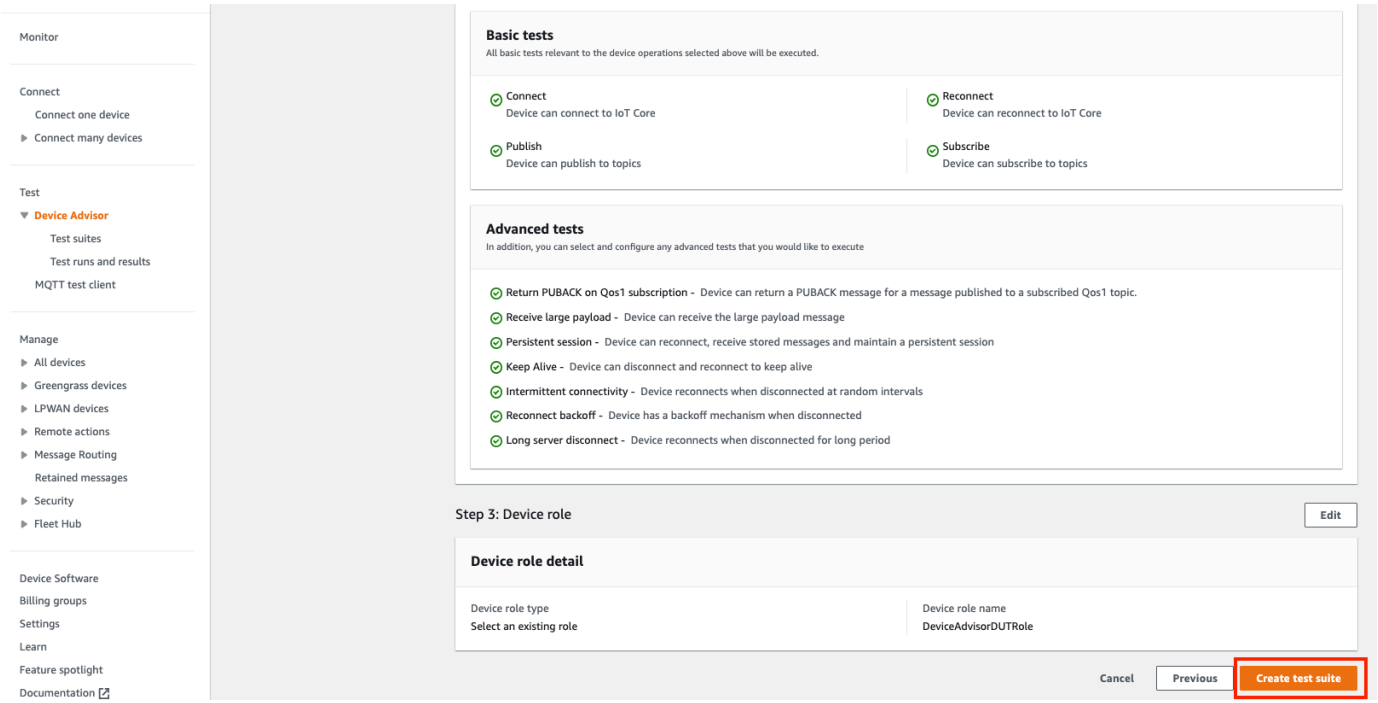


6. 이 단계에서는 새 역할을 생성하거나 기존 역할을 선택합니다. 세부 정보는 [디바이스 역할로 사용할 IAM 역할 생성]을 참조하세요.

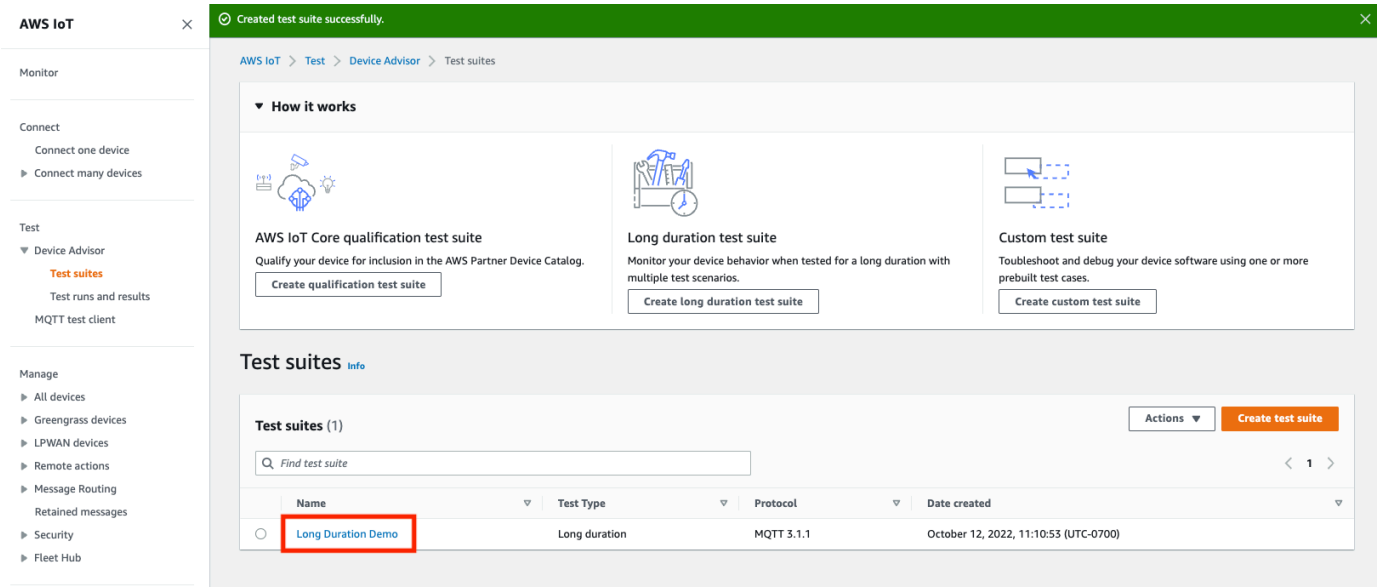


7. 이 단계까지 생성된 모든 구성을 검토하고 Create test suite(테스트 제품군 생성)를 선택합니다.

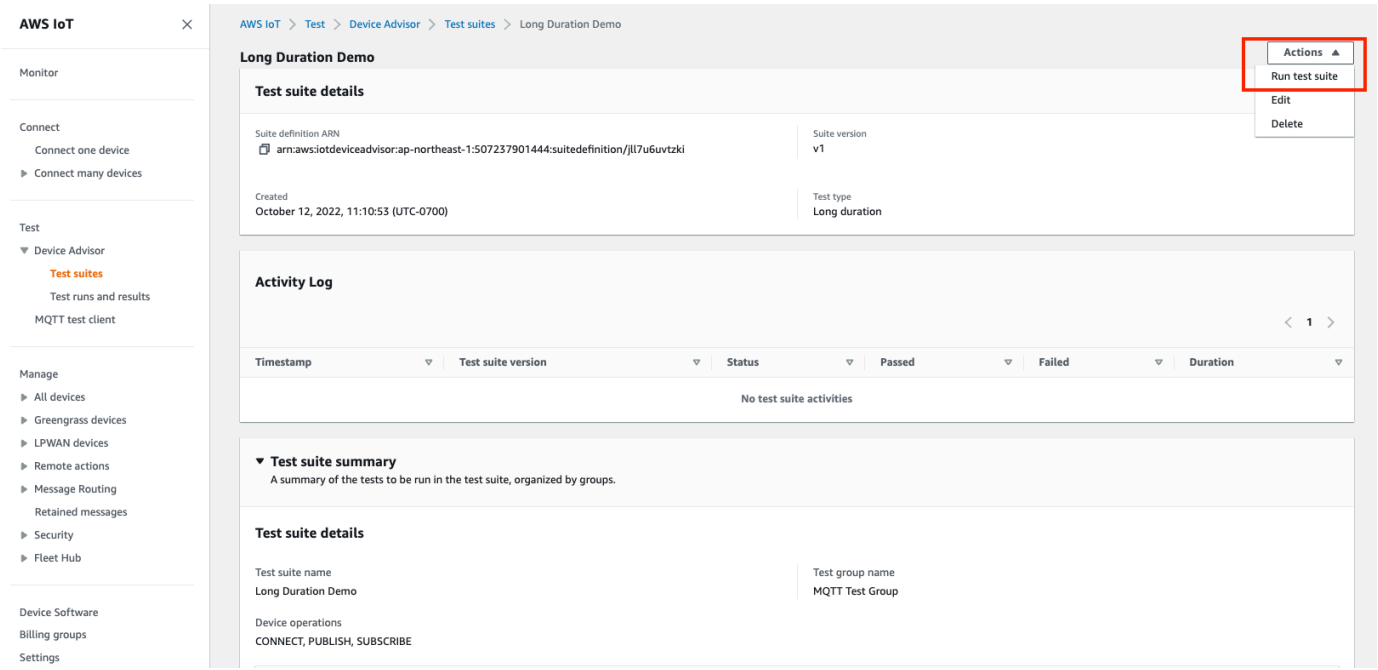




8. 생성된 테스트 제품군이 Test suites(테스트 제품군) 섹션 아래에 표시됩니다. 세부 정보를 보려면 제품군을 선택합니다.



9. 생성된 테스트 제품군을 실행하려면 Actions(작업), Run test suite(테스트 제품군 실행)를 차례로 선택합니다.

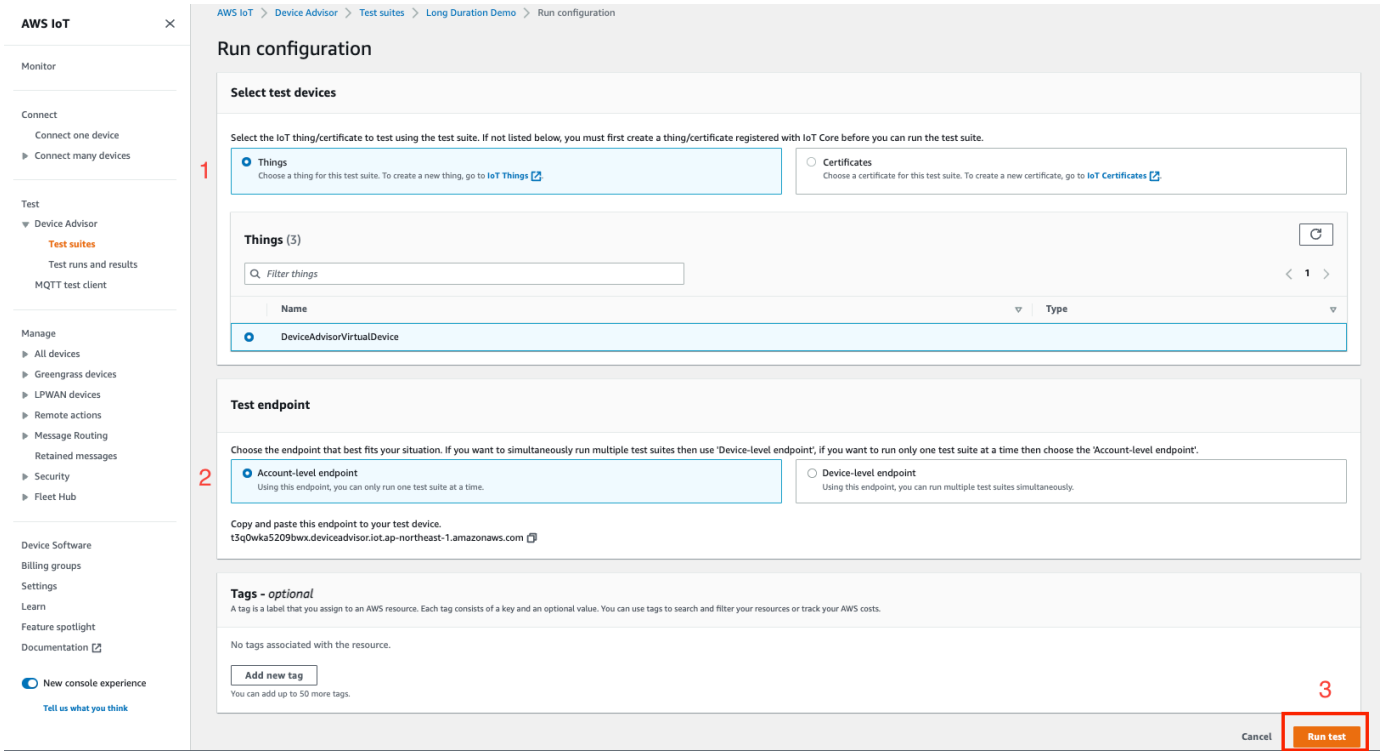


10. Run configuration(구성 실행) 페이지에서 구성 옵션을 선택합니다.

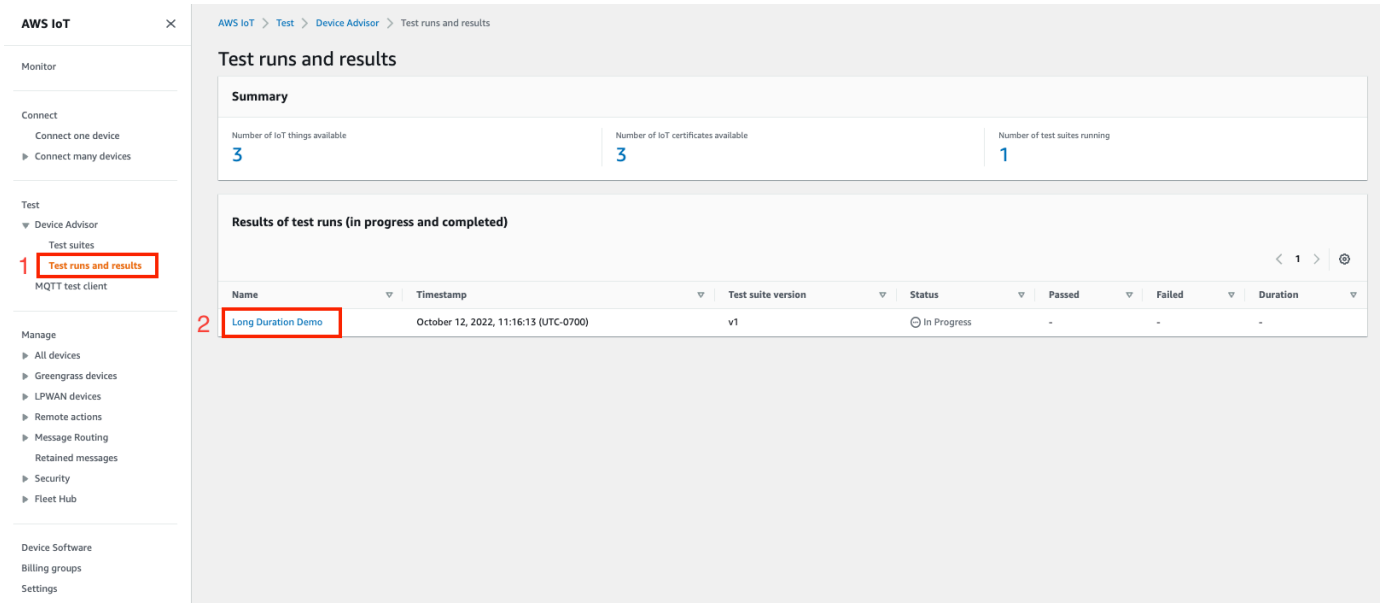
    a. 테스트를 실행할 사물 또는 인증서를 선택합니다.

    b. Account-level endpoint(계정 수준 엔드포인트) 또는 Device-level endpoint(디바이스 수준 엔드포인트)를 선택합니다.

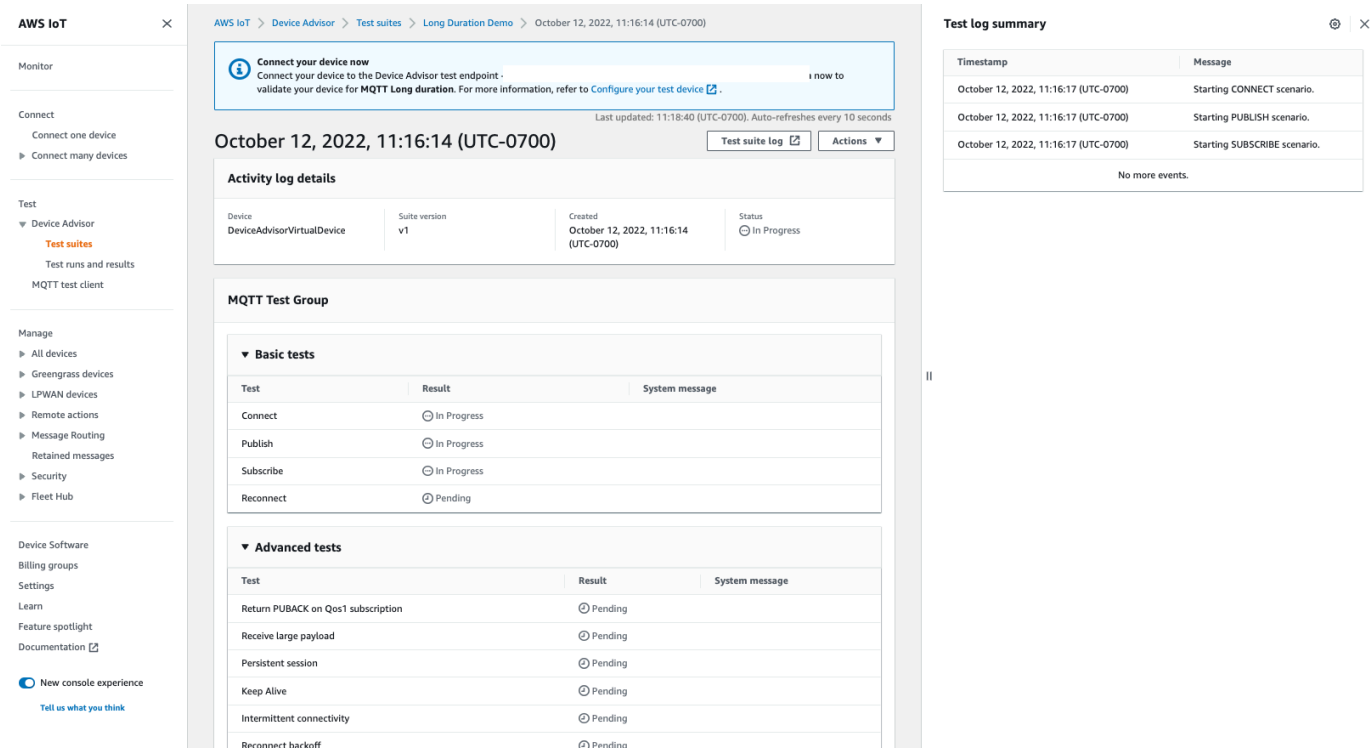
    c. Run test(테스트 실행)를 선택하여 테스트를 실행합니다.



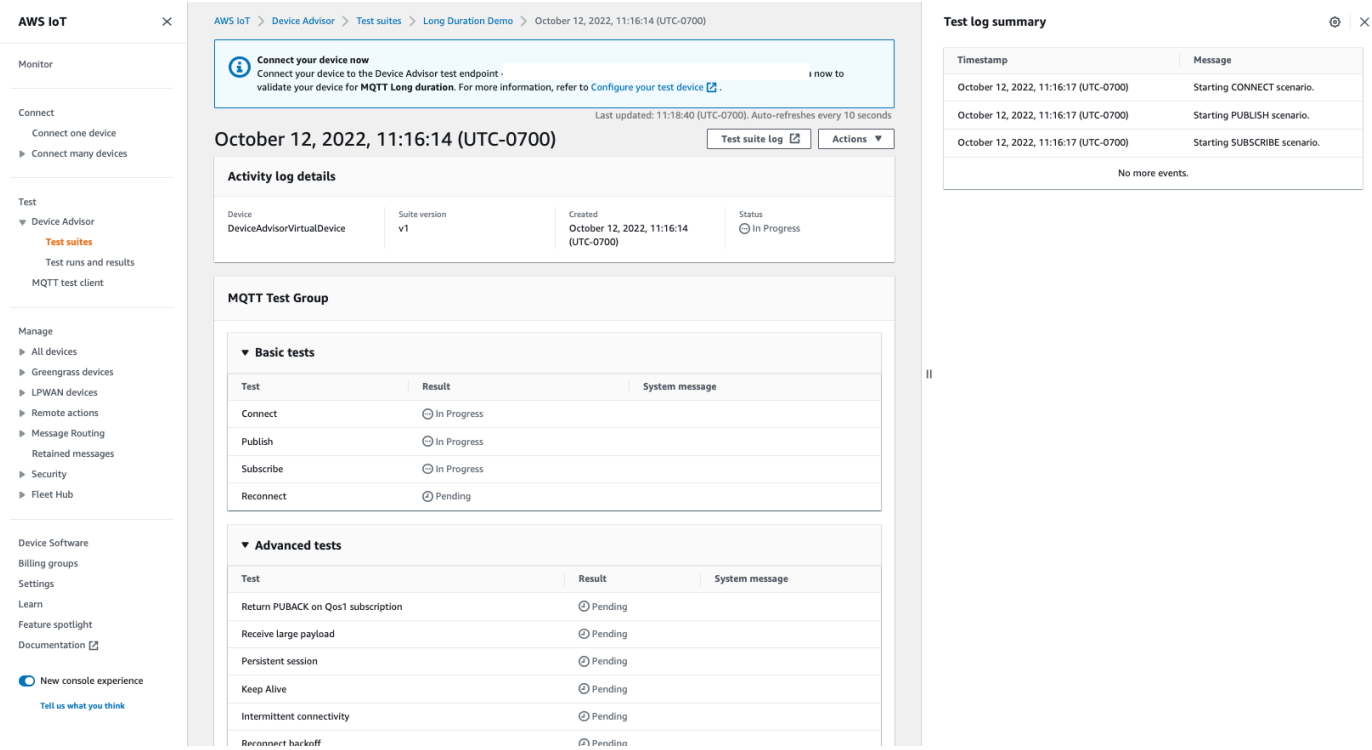
11. 테스트 제품군의 실행 결과를 보려면 왼쪽 탐색 창에서 Test runs and results(테스트 실행 및 결과)를 선택합니다. 결과의 세부 정보를 보려면 실행한 테스트 제품군을 선택합니다.



12. 앞의 단계를 수행하면 테스트 요약 페이지가 나타납니다. 테스트 실행의 모든 세부 정보가 이 페이지에 표시됩니다. 콘솔에 디바이스 연결을 시작하라는 메시지가 표시되면 제공된 엔드포인트에 디바이스를 연결합니다. 이 페이지에서 테스트 진행 상황을 확인할 수 있습니다.



13. 장기간 테스트에서는 사이드 패널의 추가 Test log summary(테스트 로그 요약)에 디바이스와 브로커 간에 발생하는 모든 중요한 이벤트가 거의 실시간으로 표시됩니다. 자세한 로그를 보려면 테스트 케이스 로그를 클릭합니다.



Device Advisor VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 생성하여 VPC와 AWS IoT Core Device Advisor 테스트 엔드포인트(데이터 영역) 간에 프라이빗 연결을 설정할 수 있습니다. 이 엔드포인트를 사용하여 프로덕션에 AWS IoT 디바이스를 배포하기 AWS IoT Core 전에 와의 안정적이고 안전한 연결을 위해 디바이스를 검증할 수 있습니다. Device Advisor의 미리 빌드된 테스트는 디바이스 소프트웨어가 TLS, MQTT, 디바이스 섀도우 및 AWS IoT 작업 사용에 대한 모범 사례를 따르는지 검증하는 데 도움이 됩니다.

AWS PrivateLink는 IoT 디바이스에 사용되는 인터페이스 엔드포인트에 전원을 공급합니다. 이 서비스를 사용하면 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 Direct Connect 연결 없이 AWS IoT Core Device Advisor 테스트 엔드포인트에 비공개로 액세스하는 데 도움이 됩니다. TCP 및 MQTT 패킷을 전송하는 VPC의 인스턴스는 AWS IoT Core Device Advisor 테스트 엔드포인트와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다. VPC와 AWS IoT Core Device Advisor 간의 트래픽은 나가지 않습니다 AWS 클라우드. IoT 디바이스와 Device Advisor 테스트 케이스 간의 모든 TLS 및 MQTT 통신은 AWS 계정의 리소스 내에서 유지됩니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다.

VPC 엔드포인트 인터페이스 사용에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

AWS IoT Core Device Advisor VPC 엔드포인트에 대한 고려 사항

인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한 사항을 검토하세요. 계속하기 전에 다음 사항을 고려하세요.

- AWS IoT Core Device Advisor 는 현재 VPC에서 Device Advisor 테스트 엔드포인트(데이터 영역)에 대한 호출을 지원합니다. 메시지 브로커는 데이터 영역 통신을 사용하여 데이터를 전송하고 수신합니다. 이 작업은 TLS 및 MQTT 패킷을 사용하여 수행됩니다. AWS IoT 디바이스를 Device Advisor 테스트 엔드포인트에 AWS IoT Core Device Advisor 연결하기 위한 VPC 엔드포인트입니다. 컨트롤 플레인 API 작업은 이 VPC 엔드포인트에서 사용되지 않습니다. 테스트 제품군 또는 기타 컨트롤 플레인 API를 만들거나 실행하려면 퍼블릭 인터넷을 통해 콘솔, AWS SDK 또는 AWS 명령줄 인터페이스를 사용하세요.
- 다음은에 대한 VPC 엔드포인트를 AWS 리전 지원합니다 AWS IoT Core Device Advisor.
  - 미국 동부(버지니아 북부)
  - 미국 서부(오레곤)
  - 아시아 태평양(도쿄)

  - 유럽(아일랜드)
- Device Advisor는 X.509 클라이언트 인증서 및 RSA 서버 인증서와 함께 MQT를 지원합니다.
- 현재 VPC 엔드포인트 정책은 지원하지 않습니다.
- VPC 엔드포인트를 연결하는 리소스를 생성하는 방법에 대한 지침은 VPC 엔드포인트 사전 조건을 확인하세요. VPC 엔드포인트를 사용하려면 AWS IoT Core Device Advisor VPC 및 프라이빗 서브넷을 생성해야 합니다.
- AWS PrivateLink 리소스에는 할당량이 있습니다. 자세한 내용은 AWS PrivateLink 할당량을 참조하세요.
- VPC 엔드포인트는 IPv4 트래픽만 지원합니다.

에 대한 인터페이스 VPC 엔드포인트 생성 AWS IoT Core Device Advisor

VPC 엔드포인트를 시작하려면 인터페이스 VPC 엔드포인트를 생성합니다. 그런 다음를 AWS IoT Core Device Advisor 로 선택합니다 AWS 서비스. 를 사용하는 경우 describe-vpc-endpoint-services를 AWS CLI호출하여의 가용 영역에 AWS IoT Core Device Advisor 가 있는지 확인합니다 AWS 리전. 엔드포인트에 연결된 보안 그룹이 MQTT 및 TLS 트래픽에 대한 TCP 프로토콜 통신을 허용하는지 확인합니다. 예를 들어, 미국 동부(버지니아 북부) 리전에서는 다음 명령을 사용합니다.

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.deviceadvisor.iot
```

다음 서비스 이름을 AWS IoT Core 사용하여 용 VPC 엔드포인트를 생성할 수 있습니다.

- com.amazonaws.region.deviceadvisor.iot

기본적으로 엔드포인트에 대해 프라이빗 DNS가 켜져 있습니다. 이렇게 하면 기본 테스트 엔드포인트를 프라이빗 서브넷 내에서 계속 사용할 수 있습니다. 계정 또는 디바이스 수준 엔드포인트를 가져오려면 콘솔 AWS CLI 또는 AWS SDK를 사용합니다. 예를 들어 퍼블릭 서브넷이나 퍼블릭 인터넷에서 get-endpoint를 실행하는 경우 엔드포인트를 가져와서 Device Advisor에 연결하는 데 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하세요.

MQTT 클라이언트를 VPC 엔드포인트 인터페이스에 연결하기 위해 AWS PrivateLink 서비스는 VPC에 연결된 프라이빗 호스팅 영역에 DNS 레코드를 생성합니다. 이러한 DNS 레코드는 AWS IoT 디바이스의 요청을 VPC 엔드포인트로 전달합니다.

VPC 엔드포인트를 AWS IoT Core Device Advisor 통해에 대한 액세스 제어

VPC 조건 컨텍스트 키를 사용하여에 대한 디바이스 액세스를 제한 AWS IoT Core Device Advisor 하고 VPC 엔드포인트를 통해서만 액세스를 허용할 수 있습니다.는 다음 VPC 관련 컨텍스트 키를 AWS IoT Core 지원합니다.

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core Device Advisor 는 현재 VPC 엔드포인트 정책을 지원하지 않습니다.

다음 정책은 사물 이름과 일치하는 클라이언트 ID를 AWS IoT Core Device Advisor 사용하여에 연결할 수 있는 권한을 부여합니다. 또한 사물 이름 접두사가 붙은 모든 주제에 게시됩니다. 정책은 특정 VPC 엔드포인트 ID를 사용하여 VPC 엔드포인트에 연결하는 디바이스를 조건으로 합니다. 이 정책은 퍼블릭 AWS IoT Core Device Advisor 테스트 엔드포인트에 대한 연결 시도를 거부합니다.

```
{
"Version":"2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
"StringEquals": {
"aws:SourceVpce": "vpce-1a2b3c4d"
            }
        }

        },
```

```
        {
"Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

Device Advisor 테스트 케이스

Device Advisor는 6가지 범주로 미리 빌드된 테스트를 제공합니다.

- TLS
- MQTT
- 섀도우
- 작업 실행
- 권한 및 정책
- 장기간 테스트

Device Advisor 테스트 사례는 AWS Device Qualification Program에 적합합니다.

AWS 디바이스 자격 프로그램에 따라 자격을 얻으려면 디바이스가 다음 테스트를 통과해야 합니다.

> **Note**
>
> 다음은 자격 테스트의 수정된 목록입니다.

- TLS 연결(“TLS 연결”)

- TLS 잘못된 주체 이름 서버 인증서(“잘못된 주체 일반 이름(CN)/주체 대체 이름(SAN)”)
- TLS 비보안 서버 인증서(“공인 CA가 서명하지 않음”)
- AWS IoT 암호화 제품군에 대한 TLS 디바이스 지원(" AWS IoT 권장 암호화 제품군에 대한 TLS 디바이스 지원")
- TLS 최대 크기 조각 수신('TLS 수신 최대 크기 조각')
- TLS 만료된 서버 인증서('만료된 서버 인증서')
- TLS 대형 서버 인증서('TLS 대형 서버 인증서')
- MQTT Connect("Device send CONNECT to AWS IoT Core (Happy case)")
- MQTT 구독(“구독 가능(해피 케이스)”)
- MQTT 게시(“QoS0(해피 케이스)”)
- MQTT 연결 지터 재시도('지터 백오프를 사용하는 디바이스 연결 재시도 - CONNACK 응답 없음')

TLS

이러한 테스트를 사용하여 디바이스와 간의 전송 계층 보안 프로토콜(TLS) AWS IoT 이 안전한지 확인합니다.

> **Note**
>
> 이제 Device Advisor에서 TLS 1.3을 지원합니다.

해피 패스

TLS 연결

테스트 중인 디바이스가 TLS 핸드셰이크를 완료할 수 있는지 확인합니다 AWS IoT. 이 테스트는 클라이언트 디바이스의 MQTT 구현의 유효성을 검사하지 않습니다.

Example API 테스트 케이스 정의:

> **Note**
>
> `EXECUTION_TIMEOUT`의 기본값은 5분입니다. 최상의 결과를 얻으려면 시간 제한 값을 2분으로 설정하는 것이 좋습니다.

```
"tests":[
   {
      "name":"my_tls_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Connect",
         "version":"0.0.0"
      }
   }
]
```

Example테스트 케이스 출력:

- 통과 - 테스트 중인 디바이스가 TLS 핸드셰이크를 완료했습니다 AWS IoT.
- 경고와 함께 전달 - 테스트 중인 디바이스가 TLS 핸드셰이크를 완료 AWS IoT했지만 디바이스 또는의 TLS 경고 메시지가 있었습니다 AWS IoT.
- 실패 - 핸드셰이크 오류 AWS IoT 로 인해 테스트 중인 디바이스가에서 TLS 핸드셰이크를 완료하지 못했습니다.

TLS 최대 크기 조각 수신

이 테스트 케이스는 디바이스가 TLS 최대 크기 조각을 수신하고 처리할 수 있다는 것을 검증하는 데 도움이 됩니다. 대규모 페이로드를 수신하려면 테스트 디바이스가 QoS 1로 미리 구성된 주제를 구독해야 합니다. `${payload}` 구성을 사용하여 페이로드를 사용자 정의할 수 있습니다.

Example API 테스트 케이스 정의:

> **Note**
>
> `EXECUTION_TIMEOUT`의 기본값은 5분입니다. 최상의 결과를 얻으려면 시간 제한 값을 2분으로 설정하는 것이 좋습니다.

```
"tests":[
   {
      "name":"TLS Receive Maximum Size Fragments",
      "configuration": {
```

```
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
         "PAYLOAD_FORMAT":"{"message":"${payload}"}", // A string with a placeholder
 ${payload}, or leave it empty to receive a plain string.
         "TRIGGER_TOPIC": "test_1" // A topic to which a device will subscribe, and
 to which a test case will publish a large payload.
      },
      "test":{
         "id":"TLS_Receive_Maximum_Size_Fragments",
         "version":"0.0.0"
      }
   }
]
```

암호 그룹

AWS IoT 권장 암호 제품군에 대한 TLS 디바이스 지원

테스트 중인 디바이스의 TLS 클라이언트 Hello 메시지에 있는 암호 제품군에 권장 AWS IoT 암호 제품군이 포함되어 있다는 것을 검증합니다. 디바이스에서 지원하는 암호 제품군에 대한 다른 인사이트를 제공합니다.

Example API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_tls_support_aws_iot_cipher_suites_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Support_AWS_IoT_Cipher_Suites",
         "version":"0.0.0"
      }
```

```
    }
]
```

Example테스트 케이스 출력:

- 통과 - 테스트 중인 디바이스 암호 제품군에는 권장 AWS IoT 암호 제품군 중 하나 이상이 포함되어 있으며 지원되지 않는 암호 제품군은 포함되어 있지 않습니다.
- 경고가 있는 통과 - 디바이스 암호 제품군에 하나 이상의 AWS IoT 암호 제품군이 포함되지만 다음과 같은 문제가 있습니다.
  1. 권장 암호 제품군이 포함되어 있지 않습니다.
  2. 여기에는에서 지원하지 않는 암호 제품군이 포함되어 있습니다 AWS IoT.

  지원되지 않는 암호 제품군이 안전한지 확인하는 것이 좋습니다.
- 실패 - 테스트 중인 디바이스에 AWS IoT 지원되는 암호 제품군이 포함되어 있지 않습니다.

더 큰 크기의 서버 인증서

TLS 대형 서버 인증서

디바이스에서 더 큰 크기의 서버 인증서를 수신하고 처리할 때 AWS IoT 로 TLS 핸드셰이크를 완료할 수 있다는 것을 검증합니다. 이 테스트에서 사용하는 서버 인증서의 크기(바이트)는 TLS Connect 테스트 사례 및 IoT Core에서 현재 사용되는 크기보다 20만큼 큽니다.이 테스트 사례에서는가 TLS에 대한 디바이스의 버퍼 공간을 AWS IoT 테스트합니다. 버퍼 공간이 충분히 크면 TLS 핸드셰이크가 오류 없이 모두 적용됩니다. 이 테스트는 디바이스의 MQTT 구현을 검증하지 않습니다. TLS 핸드셰이크 프로세스가 완료된 후 테스트 케이스가 종료됩니다.

Example API 테스트 케이스 정의:

> **Note**
>
> `EXECUTION_TIMEOUT`의 기본값은 5분입니다. 최상의 결과를 얻으려면 시간 제한 값을 2분으로 설정하는 것이 좋습니다. 이 테스트 케이스는 실패하지만 TLS 연결 테스트 케이스는 통과하는 경우 TLS에 대한 디바이스의 버퍼 공간 제한을 늘리는 것이 좋습니다. 버퍼 공간 제한을 늘리면 크기가 늘어나는 경우 디바이스가 더 큰 크기의 서버 인증서를 처리할 수 있습니다.

```
"tests":[
```

```
   {
      "name":"my_tls_large_size_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Large_Size_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example테스트 케이스 출력:

- 통과 - 테스트 중인 디바이스가 AWS IoT와 TLS 핸드셰이크를 완료했습니다.
- 경고와 함께 전달 - 테스트 중인 디바이스가 TLS 핸드셰이크를 완료 AWS IoT했지만 디바이스 또는의 TLS 경고 메시지가 있습니다 AWS IoT.
- 실패 - 핸드셰이크 프로세스 중 오류로 AWS IoT 인해 테스트 중인 디바이스가 로 TLS 핸드셰이크를 완료하지 못했습니다.

TLS 비보안 서버 인증서

공인 CA에서 서명하지 않음

테스트 중인 디바이스가 ATS CA의 유효한 서명이 없는 서버 인증서를 받는 경우 연결을 종료한다는 것을 검증합니다. 디바이스는 유효한 인증서를 제공하는 엔드포인트에만 연결해야 합니다.

Example API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_tls_unsecure_server_cert_test",
      "configuration": {
```

```
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Unsecure_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example테스트 케이스 출력:

- 통과 - 테스트 중인 디바이스가 연결을 종료했습니다.
- 실패 - 테스트 중인 디바이스가 TLS 핸드셰이크를 완료했습니다 AWS IoT.

TLS 잘못된 주체 이름 서버 인증서/잘못된 주체 일반 이름(CN)/주체 대체 이름(SAN)

테스트 중인 디바이스가 요청한 도메인 이름과 다른 도메인 이름에 대한 서버 인증서를 제공하는 경우 연결을 종료하는지 확인합니다.

Example API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_tls_incorrect_subject_name_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"TLS_Incorrect_Subject_Name_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example테스트 케이스 출력:

- 통과 - 테스트 중인 디바이스가 연결을 종료했습니다.
- 실패 - 테스트 중인 디바이스가 TLS 핸드셰이크를 완료했습니다 AWS IoT.

TLS 만료된 서버 인증서

만료된 서버 인증서

만료된 서버 인증서가 제공되는 경우 테스트 중인 디바이스가 연결을 종료하는지 확인합니다.

Example API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_tls_expired_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Expired_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example테스트 케이스 출력:

- 통과 - 테스트 중인 디바이스가 TLS 핸드셰이크 완료를 거부합니다 AWS IoT. 디바이스는 연결을 끊기 전에 TLS 알림 메시지를 보냅니다.
- 경고가 있는 통과 - 테스트 중인 디바이스가 AWS IoT와 TLS 핸드셰이크를 완료하기를 거부합니다. 하지만 연결을 끊기 전에 TLS 알림 메시지를 보내지 않습니다.
- 실패 - 테스트 중인 디바이스가 TLS 핸드셰이크를 완료합니다 AWS IoT.

MQTT

연결, 연결 끊기, 다시 연결

"디바이스 전송 CONNECT 대상 AWS IoT Core (해피 케이스)"

테스트 중인 디바이스가 CONNECT 요청을 전송하는지 확인합니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_mqtt_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"MQTT_Connect",
         "version":"0.0.0"
      }
   }
]
```

“디바이스는 QoS1을 위한 임의의 주제로 PUBACK를 반환 할 수 있습니다.”

이 테스트 케이스는 디바이스(클라이언트)가 QoS1을 사용하여 주제를 구독한 후 브로커로부터 게시 메시지를 받은 경우 PUBACK 메시지를 반환할 수 있는지 확인합니다.

페이로드 콘텐츠 및 페이로드 크기는 이 테스트 케이스에 대해 구성할 수 있습니다. 페이로드 크기가 구성된 경우 Device Advisor는 페이로드 콘텐츠 값을 덮어쓰고 미리 정의된 페이로드를 원하는 크기로 디바이스에 보냅니다. 페이로드 크기는 0에서 128 사이의 값이며 128KB를 초과할 수 없습니다. AWS IoT Core 는 AWS IoT Core 메시지 브로커 및 프로토콜 제한 및 할당량 페이지에 표시된 대로 128KB보다 큰 게시 및 연결 요청을 거부합니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값은 2분을 권장합니다. PAYLOAD_SIZE는 0에서 128KB 사이의 값으로 구성할 수 있습니다. Device Advisor는 지정된 크기의 사전 정의된 페이로드를 디바이스에 다시 전송하므로 페이로드 크기를 정의하면 페이로드 콘텐츠가 무시됩니다.

```
"tests":[
{
        "name":"my_mqtt_client_puback_qos1",
        "configuration": {
            // optional:"TRIGGER_TOPIC": "myTopic",
            "EXECUTION_TIMEOUT":"300", // in seconds
            "PAYLOAD_FOR_PUBLISH_VALIDATION":"custom payload",
            "PAYLOAD_SIZE":"100" // in kilobytes
        },
        "test": {
            "id": "MQTT_Client_Puback_QoS1",
            "version": "0.0.0"
        }
    }
]
```

“지터 백오프로 디바이스 연결 재시도 - CONNACK 응답 없음”

테스트 중인 디바이스가 브로커와 5회 이상 다시 연결할 때 적절한 지터 백오프를 사용하는지 확인합니다. 브로커는 테스트의 CONNNECT 요청에 따라 디바이스의 타임스탬프를 기록하고, 패킷 유효성 검사를 수행하고, 테스트 중인 디바이스에 CONNACK를 보내지 않고 일시 중지하고, 테스트 중인 디바이스가 요청을 다시 보낼 때까지 기다립니다. 여섯 번째 연결 시도는 통과할 수 있으며 CONNACK는 테스트 중인 디바이스로 다시 흐를 수 있습니다.

위의 프로세스가 다시 수행됩니다. 전체적으로 이 테스트 케이스는 디바이스가 총 12번 이상 연결되어야 합니다. 수집된 타임스탬프는 테스트 중인 디바이스에서 지터 백오프가 사용되는지 확인하는 데 사용됩니다. 테스트 중인 디바이스에 엄격하게 지수 백오프 지연이 있는 경우 이 테스트 케이스가 경고와 함께 통과됩니다.

이 테스트 케이스를 통과하려면 테스트 중인 디바이스에서 지수 백오프 및 지터 메커니즘을 구현하는 것이 좋습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 4분으로 설정하는 것이 좋습니다.

```
"tests":[
   {
      "name":"my_mqtt_jitter_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",    // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Jitter_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

“지수 백오프로 디바이스 연결 재시도 - CONNACK 응답 없음”

테스트 중인 디바이스가 브로커와 5회 이상 다시 연결할 때 적절한 지수 백오프를 사용하는지 확인합니다. 브로커는 테스트의 CONNNECT 요청에 따라 디바이스의 타임스탬프를 기록하고, 패킷 유효성 검사를 수행하고, 클라이언트 디바이스에 CONNACK를 보내지 않고 일시 중지하고, 테스트 중인 디바이스가 요청을 다시 보낼 때까지 기다립니다. 수집된 타임스탬프는 테스트 중인 디바이스에서 지수 백오프가 사용되는지 확인하는 데 사용됩니다.

이 테스트 케이스를 통과하려면 테스트 중인 디바이스에서 지수 백오프 및 지터 메커니즘을 구현하는 것이 좋습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 4분으로 설정하는 것이 좋습니다.

```
"tests":[
   {
      "name":"my_mqtt_exponential_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"600",  // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Exponential_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

“지터 백오프로 디바이스 재연결 - 서버 연결 해제 후”

테스트 중인 디바이스가 서버에서 연결 해제된 후 다시 연결하는 동안 필요한 지터 및 백오프를 사용하는지 검증합니다. Device Advisor는 최소 5회 동안 서버에서 디바이스 연결을 해제하고 MQTT 재연결을 위한 디바이스 동작을 관찰합니다. Device Advisor는 테스트 중인 디바이스에 대한 CONNECT 요청의 타임스탬프를 기록하고, 패킷 유효성 검사를 수행하고, 클라이언트 디바이스에 CONNACK를 보내지 않고 일시 중지하고, 테스트 중인 디바이스가 요청을 다시 보낼 때까지 기다립니다. 수집된 타임스탬프는 테스트 중인 디바이스가 다시 연결하는 동안 지터와 백오프가 사용되는지 확인하는 데 사용됩니다. 테스트 중인 디바이스에 엄격한 지수 백오프가 있거나 적절한 지터 백오프 메커니즘을 구현하지 않는 경우 이 테스트 케이스는 경고와 함께 통과합니다. 테스트 중인 디바이스가 선형 백오프 또는 상수 백오프 메커니즘을 구현한 경우 테스트가 실패합니다.

이 테스트 케이스를 통과하려면 테스트 중인 장치에서 지수 백오프 및 지터 메커니즘을 구현하는 것이 좋습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 4분으로 설정하는 것이 좋습니다.
> 백오프에 대한 검증을 위한 재연결 시도 횟수는 RECONNECTION_ATTEMPTS를 지정하여 변경할 수 있습니다. 번호는 5~10 사이여야 합니다. 기본값은 5입니다.

```
"tests":[
```

```
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_server_disconnect",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Server_Disconnect",
         "version":"0.0.0"
      }
   }
]
```

"지터 백오프로 디바이스 재연결 - 연결 상태가 불안정한 경우"

테스트 중인 디바이스가 불안정한 연결에서 다시 연결하는 동안 필요한 지터 및 백오프를 사용하는지 확인합니다. Device Advisor는 5번의 성공적인 연결 후 서버에서 디바이스의 연결을 끊고 MQTT 재연결에 대한 디바이스의 동작을 관찰합니다. Device Advisor는 테스트 중인 디바이스에 대한 CONNECT 요청의 타임스탬프를 기록하고, 패킷 유효성 검사를 수행하고, CONNACK를 다시 보내고, 연결을 끊고, 연결 해제의 타임스탬프를 기록하고, 테스트 중인 디바이스가 요청을 다시 보낼 때까지 기다립니다. 수집된 타임스탬프는 성공적이지만 불안정한 연결 후 다시 연결하는 동안 테스트 중인 디바이스가 지터와 백오프를 사용하는지 확인하는 데 사용됩니다. 테스트 중인 디바이스에 엄격한 지수 백오프가 있거나 적절한 지터 백오프 메커니즘을 구현하지 않는 경우 이 테스트 케이스는 경고와 함께 통과합니다. 테스트 중인 디바이스가 선형 백오프 또는 상수 백오프 메커니즘을 구현한 경우 테스트가 실패합니다.

이 테스트 케이스를 통과하려면 테스트 중인 장치에서 지수 백오프 및 지터 메커니즘을 구현하는 것이 좋습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 4분으로 설정하는 것이 좋습니다.
> 백오프에 대한 검증을 위한 재연결 시도 횟수는 RECONNECTION_ATTEMPTS를 지정하여 변경할 수 있습니다. 번호는 5~10 사이여야 합니다. 기본값은 5입니다.

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_unstable_connection",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Unstable_Connection",
         "version":"0.0.0"
      }
   }
]
```

게시

“QoS0(해피 케이스)”

테스트 중인 디바이스가 QoS0 또는 QoS1이 포함된 메시지를 게시하는지 확인합니다. 테스트 설정에서 이 주제 값 및 페이로드를 지정하여 메시지 주제 및 페이로드의 유효성을 검사할 수도 있습니다.

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_mqtt_publish_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish",
```

```
         "version":"0.0.0"
      }
   }
]
```

“QoS1 게시 재시도 - PUBACK 없음”

브로커가 PUBACK을 보내지 않는 경우 테스트 중인 디바이스가 QoS1과 함께 전송된 메시지를 다시 게시하는지 확인합니다. 테스트 설정에서 이 주제를 지정하여 메시지 주제의 유효성을 검사할 수도 있습니다. 메시지를 다시 게시하기 전에 클라이언트 디바이스의 연결을 끊으면 안 됩니다. 또한 이 테스트는 다시 게시된 메시지의 패킷 식별자가 원본과 같은지 확인합니다. 테스트 실행 중에 디바이스의 연결이 끊겼다가 다시 연결되면 테스트 케이스가 오류 없이 재설정되며 디바이스에서 테스트 케이스 단계를 다시 수행해야 합니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 적어도 4분이 권장됩니다.

```
"tests":[
   {
      "name":"my_mqtt_publish_retry_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retry_No_Puback",
         "version":"0.0.0"
      }
   }
]
```

'보관된 메시지 게시'

테스트 중인 디바이스가 retainFlag가 true로 설정된 메시지를 게시하는지 확인합니다. 테스트 설정에서 이 주제 값 및 페이로드를 지정하여 메시지 주제 및 페이로드의 유효성을 검사할 수 있습

니다. PUBLISH 패킷 내에서 전송된 `retainFlag`가 true로 설정되지 않은 경우 테스트 케이스가 실패합니다.

API 테스트 케이스 정의:

> **Note**
>
> `EXECUTION_TIMEOUT`의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다. 이 테스트 케이스를 실행하려면 디바이스 역할에 `iot:RetainPublish` 작업을 추가합니다.

```
"tests":[
   {
      "name":"my_mqtt_publish_retained_messages_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds

 "TOPIC_FOR_PUBLISH_RETAINED_VALIDATION": "my_TOPIC_FOR_PUBLISH_RETAINED_VALIDATION",

 "PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retained_Messages",
         "version":"0.0.0"
      }
   }
]
```

'사용자 속성을 포함한 게시'

테스트 중인 디바이스가 올바른 사용자 속성이 포함된 메시지를 게시하는지 확인합니다. 테스트 설정에서 이름-값 페어를 설정하여 사용자 속성의 유효성을 검사할 수 있습니다. 사용자 속성이 제공되지 않거나 일치하지 않는 경우 테스트 케이스가 실패합니다.

API 테스트 케이스 정의:

> **Note**
>
> MQTT5 전용 테스트 케이스입니다.

EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_mqtt_user_property_test",
      "test":{
        "USER_PROPERTIES": [
            {"name": "name1", "value":"value1"},
            {"name": "name2", "value":"value2"}
        ],
        "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"MQTT_Publish_User_Property",
         "version":"0.0.0"
      }
   }
]
```

Subscribe

“구독 가능(해피 케이스)”

테스트 중인 디바이스가 MQTT 주제를 구독하는지 확인합니다. 테스트 설정에서 이 주제를 지정하여 테스트 중인 디바이스가 구독하는 주제의 유효성을 검사할 수도 있습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_mqtt_subscribe_test",
```

```
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["my_TOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe",
         "version":"0.0.0"
      }
   }
]
```

“구독 재시도 - SUBACK 없음”

테스트 중인 디바이스가 실패한 MQTT 주제를 구독을 재시도하는지 확인합니다. 그런 다음 서버는 대기하고 SUBACK를 보내지 않습니다. 클라이언트 디바이스가 구독을 다시 시도하지 않으면 테스트가 실패합니다. 클라이언트 디바이스는 동일한 패킷 ID로 실패한 구독을 다시 시도해야 합니다. 테스트 설정에서 이 주제를 지정하여 테스트 중인 디바이스가 구독하는 주제의 유효성을 검사할 수도 있습니다. 테스트 실행 중에 디바이스의 연결이 끊겼다가 다시 연결되면 테스트 케이스가 오류 없이 재설정되며 디바이스에서 테스트 케이스 단계를 다시 수행해야 합니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 4분으로 설정하는 것이 좋습니다.

```
"tests":[
   {
      "name":"my_mqtt_subscribe_retry_test",
      "configuration":{
         "EXECUTION_TIMEOUT":"300",  // in seconds
         // optional:
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["myTOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe_Retry_No_Suback",
```

```
         "version":"0.0.0"
      }
   }
]
```

Keep-Alive

"Mqtt No Ack PingResp"

이 테스트 케이스는 테스트 중인 디바이스가 ping 응답을 받지 못했을 때 연결이 해제되었는지 확인합니다. 이 테스트 사례의 일환으로 Device Advisor는 게시, 구독 및 ping 요청을 AWS IoT Core 위해에서 전송된 응답을 차단합니다. 또한 테스트 중인 디바이스가 MQTT 연결을 해제하는지 검증합니다.

API 테스트 케이스 정의:

> **Note**
>
> `EXECUTION_TIMEOUT`의 기본값은 5분입니다. 시간 제한을 `keepAliveTime` 값의 1.5배보다 크게 설정하는 것이 좋습니다.
> 이 테스트의 최대 `keepAliveTime` 시간은 230초를 초과하지 않아야 합니다.

```
"tests":[
    {
       "name":"Mqtt No Ack PingResp",
       "configuration":
          //optional:
          "EXECUTION_TIMEOUT":"306",   // in seconds
       },
       "test":{
          "id":"MQTT_No_Ack_PingResp",
          "version":"0.0.0"
       }
    }
]
```

영구 세션

‘영구 세션(해피 케이스)’

이 테스트 케이스는 영구 세션에서 연결이 끊어질 때 디바이스 동작을 검증합니다. 테스트 케이스는 디바이스가 다시 연결되고, 명시적으로 재구독하지 않은 채로 트리거 주제에 대한 구독을 다시 시작하고, 주제에 저장된 메시지를 수신하고, 영구 세션에서 예상대로 작동할 수 있는지 확인합니다. 이 테스트 사례가 통과하면 클라이언트 디바이스가 예상대로 AWS IoT Core 브로커와의 영구 세션을 유지할 수 있음을 나타냅니다. AWS IoT 영구 세션에 대한 자세한 내용은 MQTT 영구 세션 사용을 참조하세요.

이 테스트 케이스에서는 클라이언트 디바이스가 클린 세션 플래그가 거짓으로 설정된 AWS IoT Core 와 CONNECT된 후 트리거 주제를 구독할 것으로 예상됩니다. 구독에 성공하면 AWS IoT Core Device Advisor에서 디바이스의 연결을 해제합니다. 디바이스가 연결이 끊긴 상태인 동안 QoS 1 메시지 페이로드가 해당 주제에 저장됩니다. 그런 다음 Device Advisor는 클라이언트 디바이스가 테스트 엔드포인트에 다시 연결할 수 있도록 허용합니다. 이 시점에서 영구 세션이 있기 때문에 클라이언트 디바이스는 추가 SUBSCRIBE 패킷을 보내지 않고 주제 구독을 재개하고 브로커로부터 QoS 1 메시지를 수신할 것으로 기대됩니다. 다시 연결한 후 클라이언트 디바이스가 추가 SUBSCRIBE 패킷을 전송하여 트리거 주제를 다시 구독하거나 클라이언트가 트리거 주제에서 저장된 메시지를 수신하지 못하면 테스트 케이스가 실패합니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 최소한 4분으로 설정하는 것이 좋습니다. 첫 번째 연결에서 클라이언트 디바이스는 이전에 구독한 적이 없는 TRIGGER_TOPIC을 명시적으로 구독해야 합니다. 테스트 케이스를 통과하려면 클라이언트 디바이스가 QoS 1을 사용하여 TRIGGER_TOPIC을 성공적으로 구독해야 합니다. 다시 연결한 후 클라이언트 디바이스는 활성 영구 세션이 있다는 것을 이해하도록 기대됩니다. 따라서 트리거 주제가 보낸 저장된 메시지를 수락하고 해당 특정 메시지에 대해 PUBACK을 반환해야 합니다.

```
"tests":[
   {
      "name":"my_mqtt_persistent_session_happy_case",
      "configuration":{
```

```
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         // if Payload not provided, a string will be stored in the trigger topic to
 be sent back to the client device
         "PAYLOAD": "The message which should be received from AWS IoT Broker after
 re-connecting to a persistent session from the specified trigger topic.",

         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Persistent_Session_Happy_Case",
         "version":"0.0.0"
      }
   }
]
```

“영구 세션 - 세션 만료”

이 테스트 케이스는 연결이 해제된 디바이스가 만료된 영구 세션에 다시 연결할 때 디바이스 동작을 검증하는 데 도움을 줍니다. 세션이 만료되면 새 SUBSCRIBE 패킷을 명시적으로 전송하여 디바이스가 이전에 구독한 주제를 다시 구독할 것으로 기대됩니다.

첫 번째 연결 중에 영구 세션을 시작하기 위해 `CleanSession` 플래그가 false로 설정되므로 테스트 디바이스가 AWS IoT 브로커와 연결될 것으로 예상됩니다. 그러면 디바이스가 트리거 주제를 구독할 것입니다. 그런 다음 구독이 성공하고 영구 세션을 시작한 후 AWS IoT Core Device Advisor에서 디바이스의 연결을 해제합니다. 연결 해제 후 AWS IoT Core Device Advisor를 사용하면 테스트 디바이스가 테스트 엔드포인트와 다시 연결할 수 있습니다. 이 시점에서 테스트 디바이스가 다른 CONNECT 패킷을 전송할 때 AWS IoT Core Device Advisor는 영구 세션이 만료되었음을 나타내는 CONNACK 패킷을 다시 보냅니다. 테스트 디바이스는 이 패킷을 올바르게 해석해야 하며 영구 세션이 종료될 때 동일한 트리거 주제를 다시 구독할 것으로 기대됩니다. 테스트 디바이스가 주제 트리거를 다시 구독하지 않으면 테스트 케이스가 실패합니다. 테스트를 통과하려면 디바이스가 영구 세션이 끝났음을 이해하고 두 번째 연결에서 동일한 트리거 주제에 대해 새 SUBSCRIBE 패킷을 다시 보내야 합니다.

테스트 디바이스가 이 테스트 케이스를 통과하면 디바이스가 영구 세션 만료 시 기대되는 방식으로 재연결을 처리할 수 있다는 뜻입니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 시간 초과 값을 최소한 4분으로 설정하는 것이 좋습니다. 테스트 디바이스는 이전에 구독한 적이 없는 TRIGGER_TOPIC을 명시적으로 구독해야 합니다. 테스트 케이스를 통과하려면 테스트 디바이스가 CleanSession 플래그가 거짓으로 설정된 CONNECT 패킷을 보내야 하며 QoS 1을 사용하여 트리거 주제에 성공적으로 구독해야 합니다. 연결에 성공하면 AWS IoT Core Device Advisor가 디바이스의 연결을 해제합니다. 연결이 끊긴 후 AWS IoT Core Device Advisor는 디바이스가 다시 연결되도록 허용하고 Device Advisor가 영구 세션을 종료했을 TRIGGER_TOPIC 것이므로 AWS IoT Core 디바이스는 동일한를 다시 구독해야 합니다.

```
"tests":[
   {
      "name":"my_expired_persistent_session_test",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Expired_Persistent_Session",
         "version":"0.0.0"
      }
   }
]
```

섀도우

이러한 테스트를 사용하여 테스트 중인 디바이스가 AWS IoT 디바이스 섀도우 서비스를 올바르게 사용하는지 확인합니다. 자세한 내용은 AWS IoT 디바이스 섀도우 서비스를 참조하세요. 이러한 테스트 케이스가 테스트 스위트에 구성된 경우 스위트 실행을 시작할 때 사물을 제공하는 것이 필요합니다.

MQTT over WebSocket은 현재 지원되지 않습니다.

게시

'디바이스가 연결 후 상태를 게시합니다(해피 케이스)'

디바이스가에 연결한 후 상태를 게시할 수 있는지 확인 AWS IoT Core

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_shadow_publish_reported_state",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME",
         "REPORTED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         }
      },
      "test":{
         "id":"Shadow_Publish_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

REPORTED_STATE는 연결 후 디바이스의 정확한 섀도우 상태에 대한 추가 검증을 위해 제공될 수 있습니다. 기본적으로 이 테스트 케이스는 디바이스 게시 상태의 유효성을 검사합니다.

*SHADOW_NAME*이 제공되지 않으면 테스트 케이스는 기본적으로 명명되지 않은(클래식) 섀도우 유형의 주제 접두사에 게시된 메시지를 찾습니다. 디바이스에서 명명된 섀도우 유형을 사용하는 경우 섀도우 이름을 제공합니다. 자세한 내용은 디바이스에서 섀도우 사용 섹션을 참조하세요.

업데이트

'디바이스가 보고된 상태를 원하는 상태로 업데이트합니다(해피 케이스)'

디바이스가 수신된 모든 업데이트 메시지를 읽고 디바이스의 상태를 원하는 상태 속성과 일치하도록 동기화하는지 확인합니다. 동기화 후 디바이스에서 최신 보고된 상태를 게시해야 합니다. 테스트를 실행하기 전에 디바이스에 이미 기존 섀도우가 있는 경우 테스트 케이스에 대해 구성된 원하는 상태와 기존 보고된 상태가 아직 일치하지 않는지 확인합니다. 섀도우 문서에서 ClientToken 필드가 DeviceAdvisorShadowTestCaseSetup일 것이므로 이 필드를 확인하여 Device Advisor가 보낸 섀도우 업데이트 메시지를 식별할 수 있습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 2분으로 설정하는 것을 권장합니다.

```
"tests":[
   {
      "name":"my_shadow_update_reported_state",
      "configuration": {
         "DESIRED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         },
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME"
      },
      "test":{
         "id":"Shadow_Update_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

DESIRED_STATE에는 적어도 속성 하나와 관련 값이 있어야 합니다.

SHADOW_NAME이 제공되지 않으면 테스트 케이스는 기본적으로 명명되지 않은(클래식) 섀도우 유형의 주제 접두사에 게시된 메시지를 찾습니다. 디바이스에서 명명된 섀도우 유형을 사용하는 경우 섀도우 이름을 제공합니다. 자세한 내용은 디바이스에서 섀도우 사용 섹션을 참조하세요.

작업 실행

"디바이스가 작업 실행을 완료할 수 있습니다."

이 테스트 사례는 디바이스가 AWS IoT 작업을 사용하여 업데이트를 수신할 수 있는지 확인하고 성공적인 업데이트 상태를 게시하는 데 도움이 됩니다. AWS IoT 작업에 대한 자세한 내용은 작업을 참조하세요.

이 테스트 케이스를 성공적으로 실행하려면 디바이스 역할을 부여해야 하는 예약된 AWS 주제 두 개가 있습니다. 작업 활동 관련 메시지를 구독하려면 notify 및 notify-next 주제를 사용합니다. 디바이스 역할은 다음 주제에 대해 PUBLISH 작업을 허용해야 합니다.

- $aws/things/thingName/jobs/jobId/get
- $aws/things/thingName/jobs/jobId/update

다음 주제에 대해 SUBSCRIBE 및 RECEIVE 작업을 허용하는 것이 좋습니다.

- $aws/things/thingName/jobs/get/accepted
- $aws/things/thingName/jobs/jobId/get/rejected
- $aws/things/thingName/jobs/jobId/update/accepted
- $aws/things/thingName/jobs/jobId/update/rejected

다음 주제에 대해 SUBSCRIBE 작업을 허용하는 것이 좋습니다.

- $aws/things/thingName/jobs/notify-next

이러한 예약된 주제에 대한 자세한 내용은 AWS IoT 작업의 예약된 주제를 참조하세요.

MQTT over WebSocket은 현재 지원되지 않습니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 5분입니다. 제한 시간 값을 3분으로 설정하는 것을 권장합니다. 제공된 AWS IoT 작업 문서 또는 소스에 따라 제한 시간 값을 조정합니다(예: 작업을 실행하는 데 시간이 오래 걸리는 경우 테스트 사례에 더 긴 제한 시간 값을 정의). 테

스트를 실행하려면 유효한 AWS IoT 작업 문서 또는 이미 존재하는 작업 ID가 필요합니다. AWS IoT 작업 문서는 JSON 문서 또는 S3 링크로 제공될 수 있습니다. 작업 문서가 제공된 경우 작업 ID를 제공하는 것은 선택 사항입니다. 작업 ID가 제공되면 Device Advisor는 사용자를 대신하여 AWS IoT 작업을 생성하는 동안 해당 ID를 사용합니다. 작업 문서가 제공되지 않은 경우 테스트 케이스를 실행하는 리전과 동일한 리전에 있는 기존 ID를 제공할 수 있습니다. 이 경우 Device Advisor는 테스트 사례를 실행하는 동안 해당 AWS IoT 작업을 사용합니다.

```
"tests": [
   {
      "name":"my_job_execution",
      "configuration": {
         // optional:
         // Test case will create a job task by using either JOB_DOCUMENT or
 JOB_DOCUMENT_SOURCE.
         // If you manage the job task on your own, leave it empty and provide the
 JOB_JOBID (self-managed job task).
         // JOB_DOCUMENT is a JSON formatted string
         "JOB_DOCUMENT": "{
            \"operation\":\"reboot\",
            \"files\" : {
               \"fileName\" : \"install.py\",
               \"url\" : \"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
bucket-name/key}\"
            }
         }",
         // JOB_DOCUMENT_SOURCE is an S3 link to the job document. It will be used
 only if JOB_DOCUMENT is not provided.
         "JOB_DOCUMENT_SOURCE": "https://s3.amazonaws.com/bucket-name/key",
         // JOB_JOBID is mandatory, only if neither document nor document source is
 provided. (Test case needs to know the self-managed job task id).
         "JOB_JOBID": "String",
         // JOB_PRESIGN_ROLE_ARN is used for the presign Url, which will replace the
 placeholder in the JOB_DOCUMENT field
         "JOB_PRESIGN_ROLE_ARN": "String",
         // Presigned Url expiration time. It must be between 60 and 3600 seconds,
 with the default value being 3600.
         "JOB_PRESIGN_EXPIRES_IN_SEC": "Long"
         "EXECUTION_TIMEOUT": "300", // in seconds
      },
      "test": {
```

```
        "id": "Job_Execution",
        "version": "0.0.0"
      }
   }
]
```

작업 문서 생성 및 사용에 관한 자세한 내용은 작업 문서를 참조하세요.

권한 및 정책

다음 테스트를 사용하여 디바이스 인증서에 연결된 정책이 표준 모범 사례를 따르는지 확인할 수 있습니다.

MQTT over WebSocket은 현재 지원되지 않습니다.

“디바이스 인증서 연결 정책에 와일드카드가 포함되어 있지 않습니다.”

디바이스와 연결된 권한 정책이 모범 사례를 따르고 디바이스에 필요한 것보다 더 많은 권한을 부여하지 않는지 확인합니다.

API 테스트 케이스 정의:

> **Note**
>
> EXECUTION_TIMEOUT의 기본값은 1분입니다. 시간 제한을 30초 이상으로 설정하는 것이 좋습니다.

```
"tests":[
   {
        "name":"my_security_device_policies",
        "configuration": {
            // optional:
            "EXECUTION_TIMEOUT":"60"    // in seconds
        },
        "test": {
            "id": "Security_Device_Policies",
            "version": "0.0.0"
        }
    }
```

```
]
```

장기간 테스트

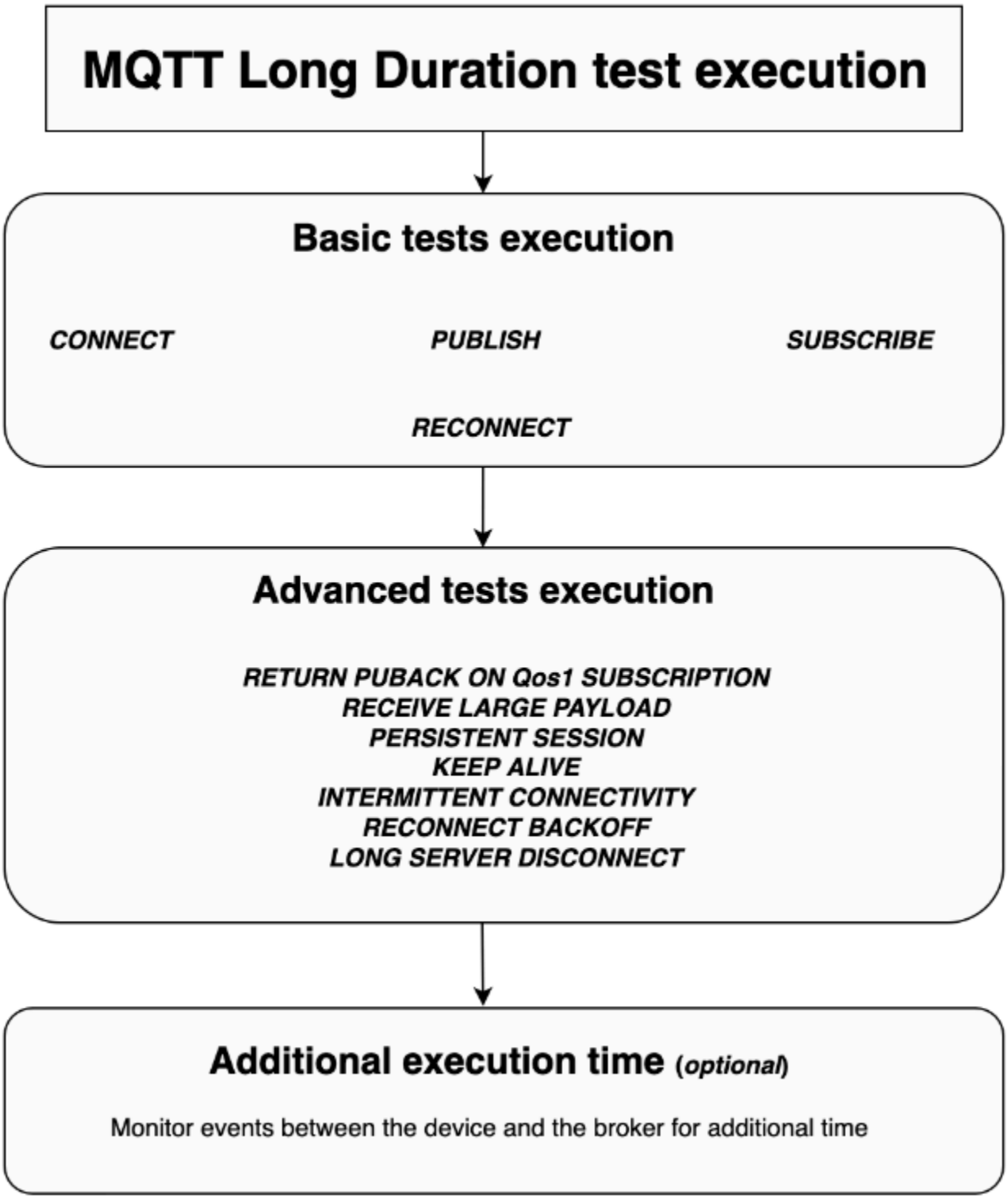
장기간 테스트는 디바이스가 장기간 작동할 때 디바이스의 동작을 모니터링하는 새로운 테스트 제품군입니다. 디바이스의 특정 동작에 초점을 맞춘 개별 테스트를 실행하는 것과 달리, 장기간 테스트는 디바이스 수명 동안 다양한 실제 시나리오에서 디바이스의 동작을 검사합니다. Device Advisor는 최대한 효율적인 순서로 테스트를 오케스트레이션합니다. 이 테스트에서는 테스트 중인 디바이스의 성능에 대한 유용한 지표가 포함된 요약 로그를 비롯한 결과 및 로그가 생성됩니다.

MQTT 장기간 테스트 케이스

MQTT 장기 테스트 케이스에서 디바이스의 동작은 MQTT 연결, 구독, 게시 및 재연결과 같은 해피 케이스 시나리오에서 처음에 관찰됩니다. 그런 다음 MQTT 재연결 백오프, 긴 서버 연결 해제 및 간헐적 연결과 같은 여러 복잡한 실패 시나리오에서 디바이스가 관찰됩니다.

MQTT 장기간 테스트 케이스 실행 흐름

MQTT 장기긴 테스트 케이스 실행에는 세 단계가 있습니다.


MQTT Long Duration test execution
Basic tests execution
CONNECT
PUBLISH
SUBSCRIBE
RECONNECT
Advanced tests execution
RETURN PUBACK ON Qos1 SUBSCRIPTION
RECEIVE LARGE PAYLOAD
PERSISTENT SESSION
KEEP ALIVE
INTERMITTENT CONNECTIVITY
RECONNECT BACKOFF
LONG SERVER DISCONNECT
Additional execution time (optional)
Monitor events between the device and the broker for additional time

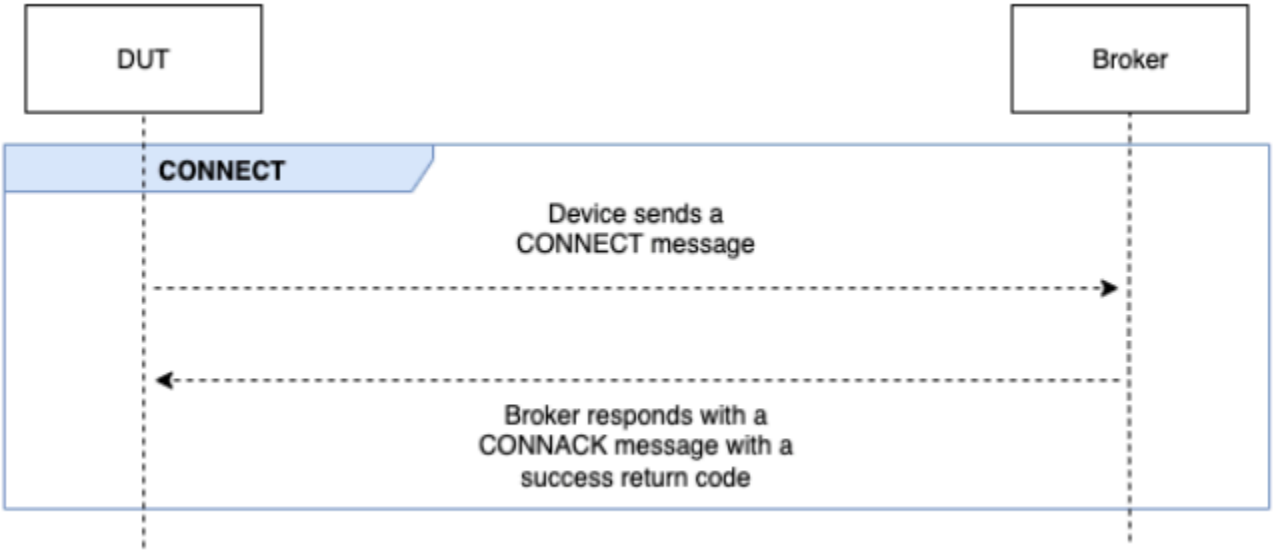
기본 테스트 실행

이 단계에서 테스트 케이스는 간단한 테스트를 병렬로 실행합니다. 테스트에서는 디바이스에 구성에서 선택된 작업이 있는지 확인합니다.

기본 테스트 세트에는 선택한 작업에 따라 다음이 포함될 수 있습니다.

CONNECT

이 시나리오는 디바이스가 브로커와 성공적으로 연결할 수 있는지 확인합니다.

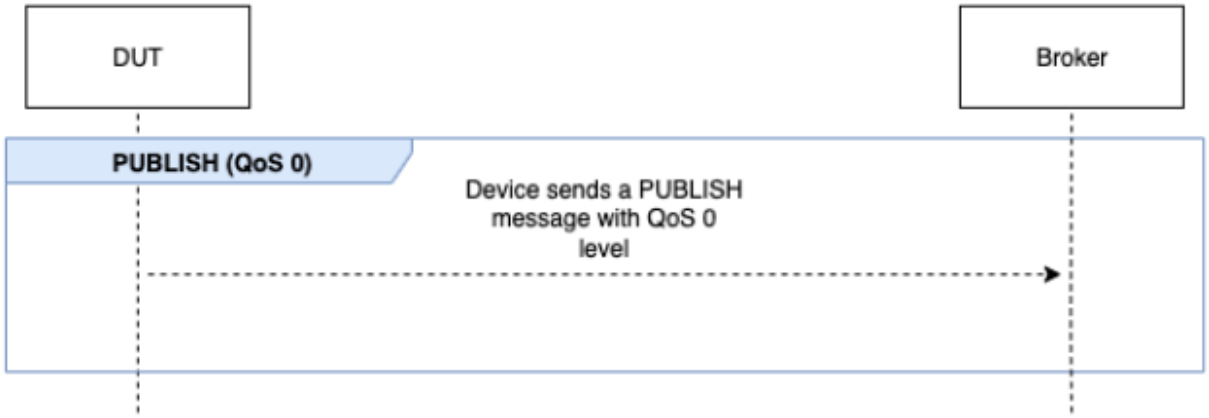


PUBLISH

이 시나리오는 디바이스가 브로커에 대해 성공적으로 게시하는지 확인합니다.

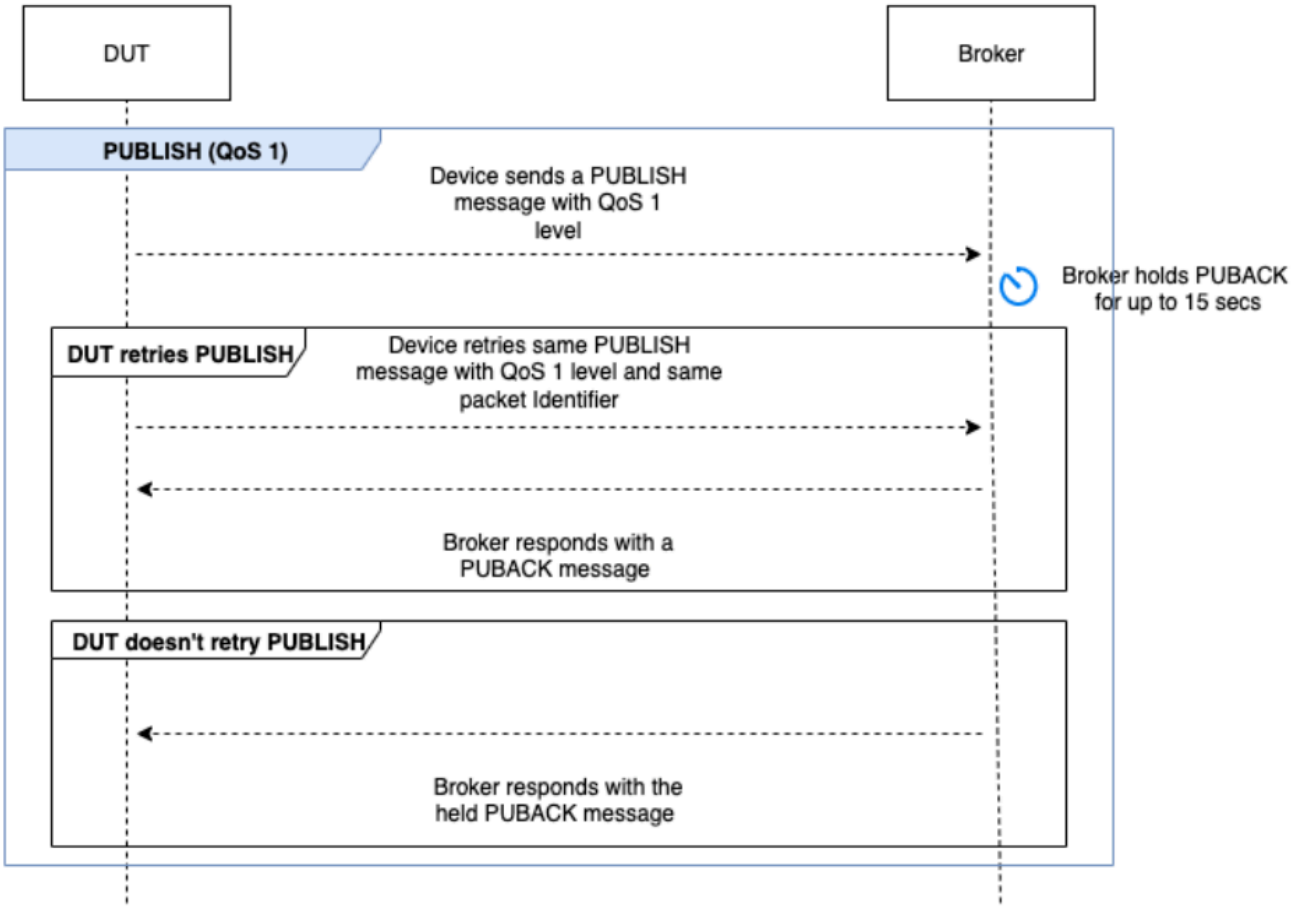
QoS 0

이 테스트 케이스는 QoS 0으로 게시하는 동안 디바이스가 브로커에 PUBLISH 메시지를 성공적으로 전송하는지 확인합니다. 이 테스트는 디바이스에서 PUBACK 메시지를 수신할 때까지 기다리지 않습니다.



QoS 1

이 테스트 케이스에서는 디바이스가 QoS 1을 사용하여 브로커에 두 개의PUBLISH 메시지를 보낼 것으로 예상합니다. 첫 번째 PUBLISH 메시지 이후 브로커는 응답하기 전에 최대 15초 동안 기다립니다. 디바이스는 15초 이내에 동일한 패킷 식별자로 원본 PUBLISH 메시지를 재시도해야 합니다. 이렇게 하면 브로커가 PUBACK 메시지로 응답하고 테스트에서 검증합니다. 디바이스가 PUBLISH를 재시도하지 않으면 원본 PUBACK이 디바이스로 전송되고 테스트는 시스템 메시지와 더불어 경고와 함께 통과로 표시됩니다 테스트 실행 중에 디바이스의 연결이 끊겼다가 다시 연결되면 테스트 시나리오가 오류 없이 재설정되며 디바이스에서 테스트 시나리오 단계를 다시 수행해야 합니다.

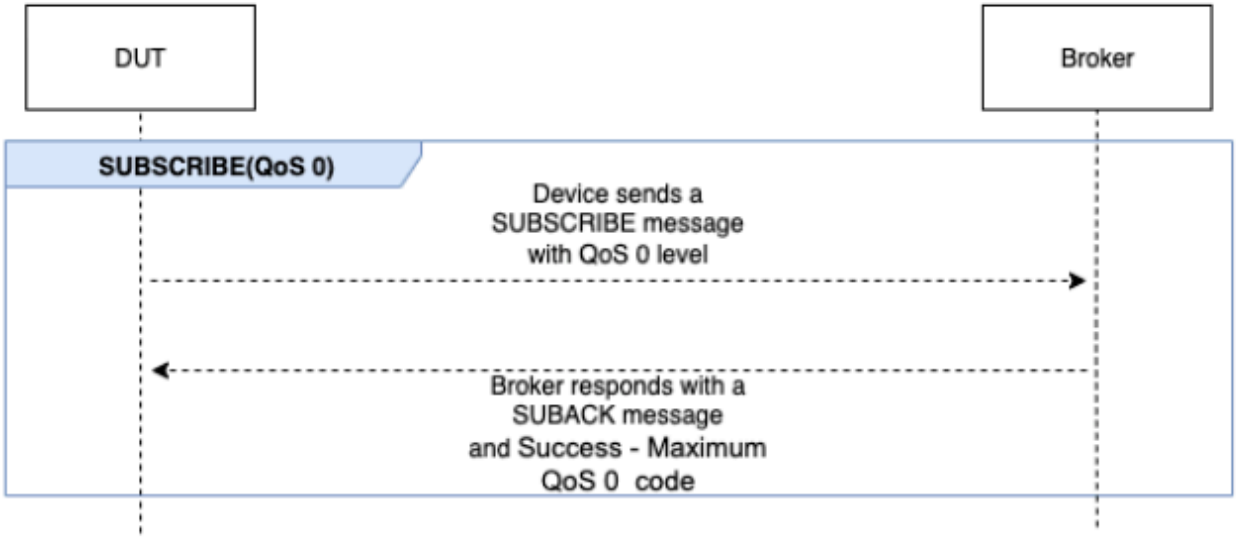


SUBSCRIBE

이 시나리오는 디바이스가 브로커에 대해 성공적으로 구독하는지 확인합니다.

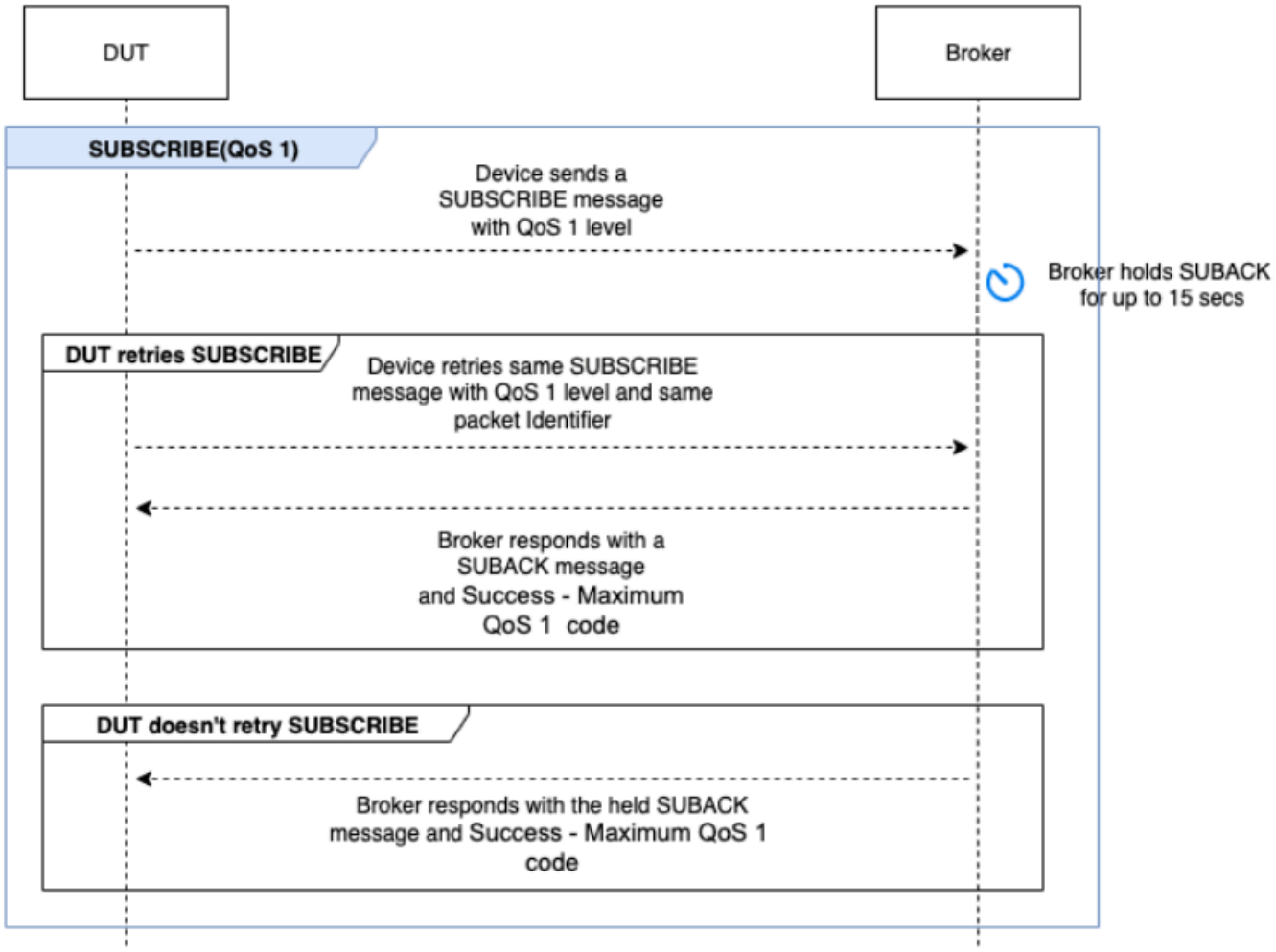
QoS 0

이 테스트 케이스는 QoS 0으로 구독하는 동안 디바이스가 브로커에 SUBSCRIBE 메시지를 성공적으로 전송하는지 확인합니다. 테스트에서 디바이스가 SUBACK 메시지를 수신할 때까지 기다리지 않습니다.



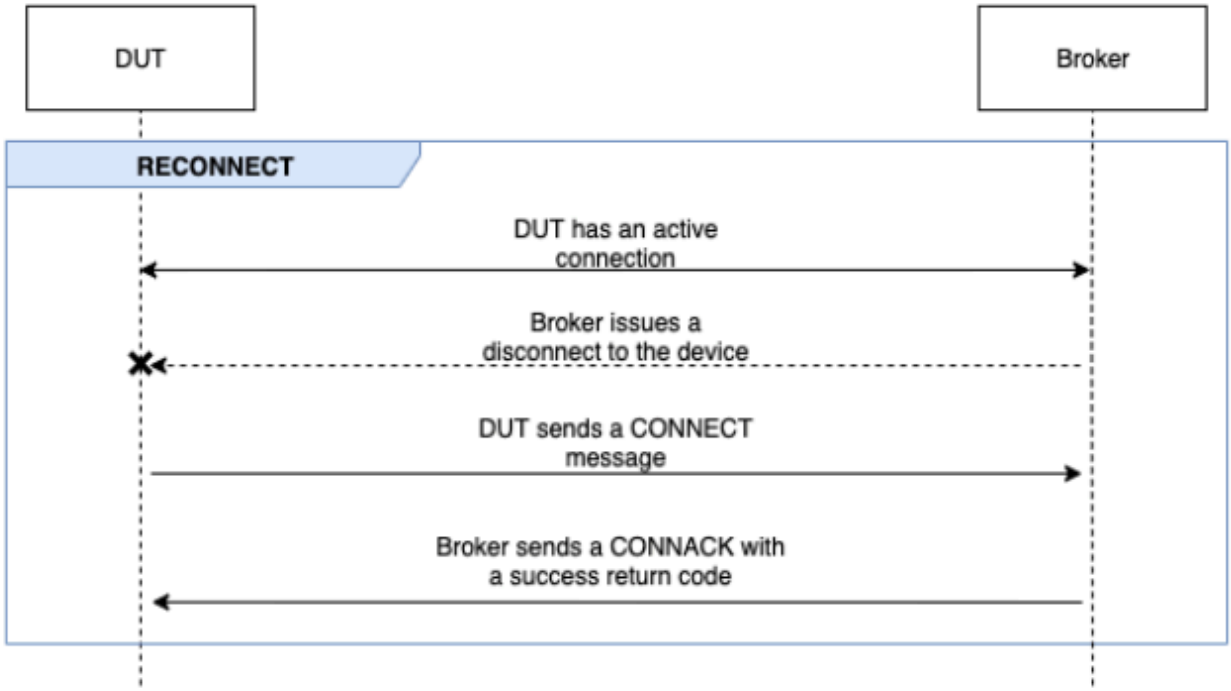
QoS 1

이 테스트 케이스에서는 디바이스가 QoS 1을 사용하여 브로커에 두 개의SUBSCRIBE 메시지를 보낼 것으로 예상합니다. 첫 번째 SUBSCRIBE 메시지 이후 브로커는 응답하기 전에 최대 15초 동안 기다립니다. 디바이스는 15초 이내에 동일한 패킷 식별자로 원본 SUBSCRIBE 메시지를 재시도해야 합니다. 이렇게 하면 브로커가 SUBACK 메시지로 응답하고 테스트에서 검증합니다. 디바이스가 SUBSCRIBE를 재시도하지 않으면 원본 SUBACK이 디바이스로 전송되고 테스트는 시스템 메시지와 더불어 경고와 함께 통과로 표시됩니다 테스트 실행 중에 디바이스의 연결이 끊겼다가 다시 연결되면 테스트 시나리오가 오류 없이 재설정되며 디바이스에서 테스트 시나리오 단계를 다시 수행해야 합니다.



RECONNECT

이 시나리오는 성공적인 연결에서 디바이스의 연결이 끊어진 후 디바이스가 브로커와 성공적으로 다시 연결되는지 확인합니다. Device Advisor는 테스트 제품군을 실행하는 동안 이전에 디바이스를 두 번 이상 연결한 경우 디바이스의 연결을 끊지 않습니다. 대신 테스트가 통과로 표시됩니다.



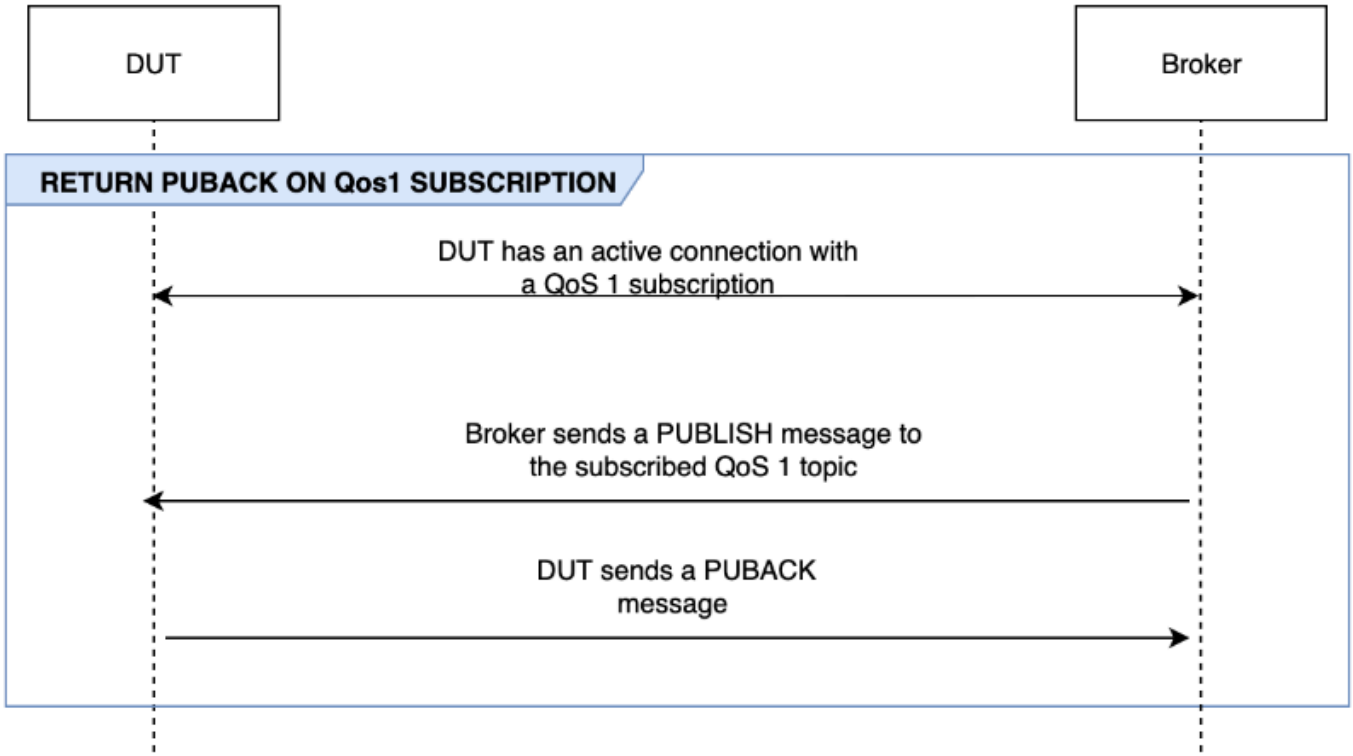
고급 테스트 실행

이 단계에서 테스트 케이스는 디바이스가 모범 사례를 준수하는지 확인하기 위해 더 복잡한 테스트를 순차적으로 실행합니다. 이러한 고급 테스트는 선택할 수 있으며 필요하지 않은 경우 선택 취소할 수 있습니다. 각 고급 테스트에는 시나리오에서 요구하는 내용에 따라 고유한 제한 시간 값이 있습니다.

RETURN PUBACK ON QoS 1 SUBSCRIPTION

> **Note**
>
> 디바이스가 QoS 1 구독을 수행할 수 있는 경우에만 이 시나리오를 선택합니다.

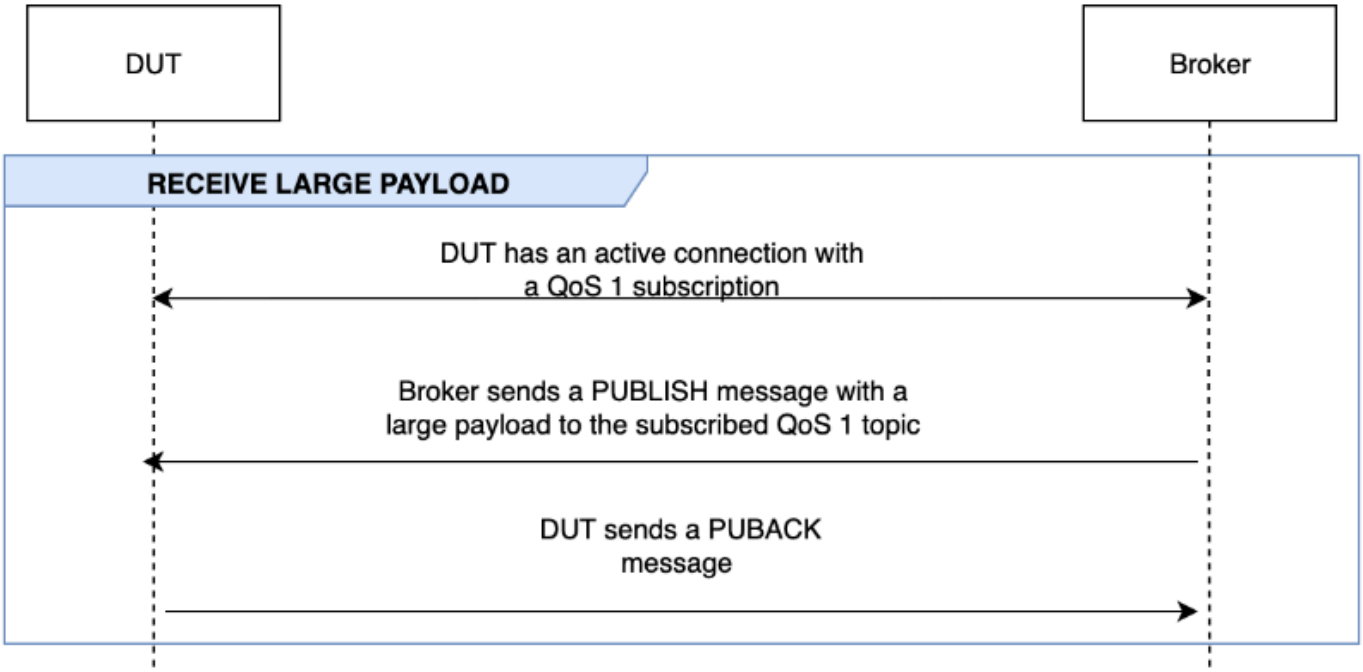
이 시나리오는 디바이스가 주제를 구독하고 브로커에서 PUBLISH 메시지를 수신한 후 PUBACK 메시지를 반환하는지 확인합니다.



RECEIVE LARGE PAYLOAD

> **Note**
>
> 디바이스가 QoS 1 구독을 수행할 수 있는 경우에만 이 시나리오를 선택합니다.

이 시나리오는 페이로드가 큰 QoS 1 주제에 대해 브로커에서 `PUBLISH` 메시지를 수신한 후 디바이스가 `PUBACK` 메시지로 응답하는지 확인합니다. 예상 페이로드의 형식은 `LONG_PAYLOAD_FORMAT` 옵션을 사용하여 구성할 수 있습니다.



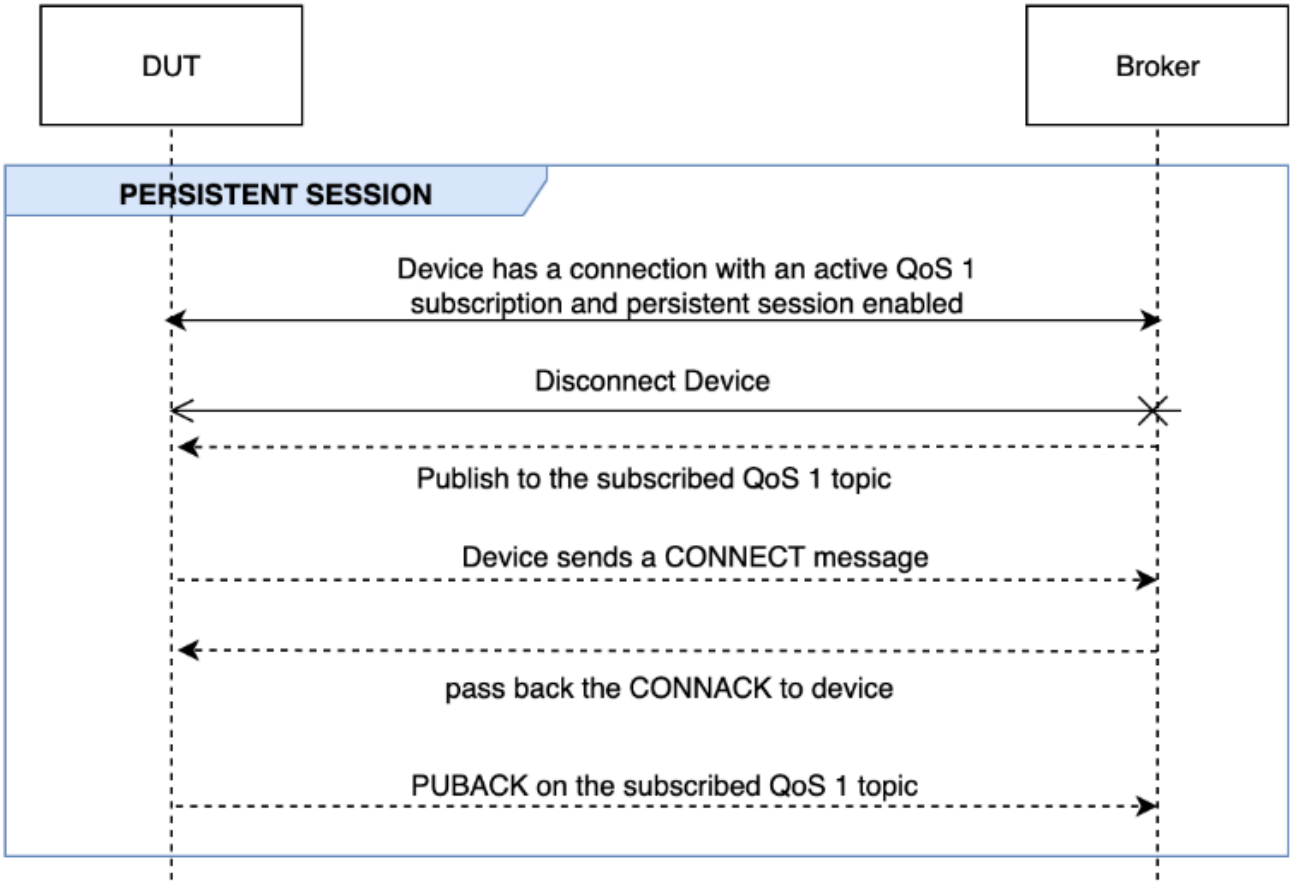
PERSISTENT SESSION

> **Note**
>
> 디바이스가 QoS 1 구독을 수행할 수 있고 영구 세션을 유지할 수 있는 경우에만 이 시나리오를 선택합니다.

이 시나리오에서는 영구 세션을 유지하는 디바이스 동작을 확인합니다. 테스트는 다음 조건이 충족될 때 유효성을 검증합니다.

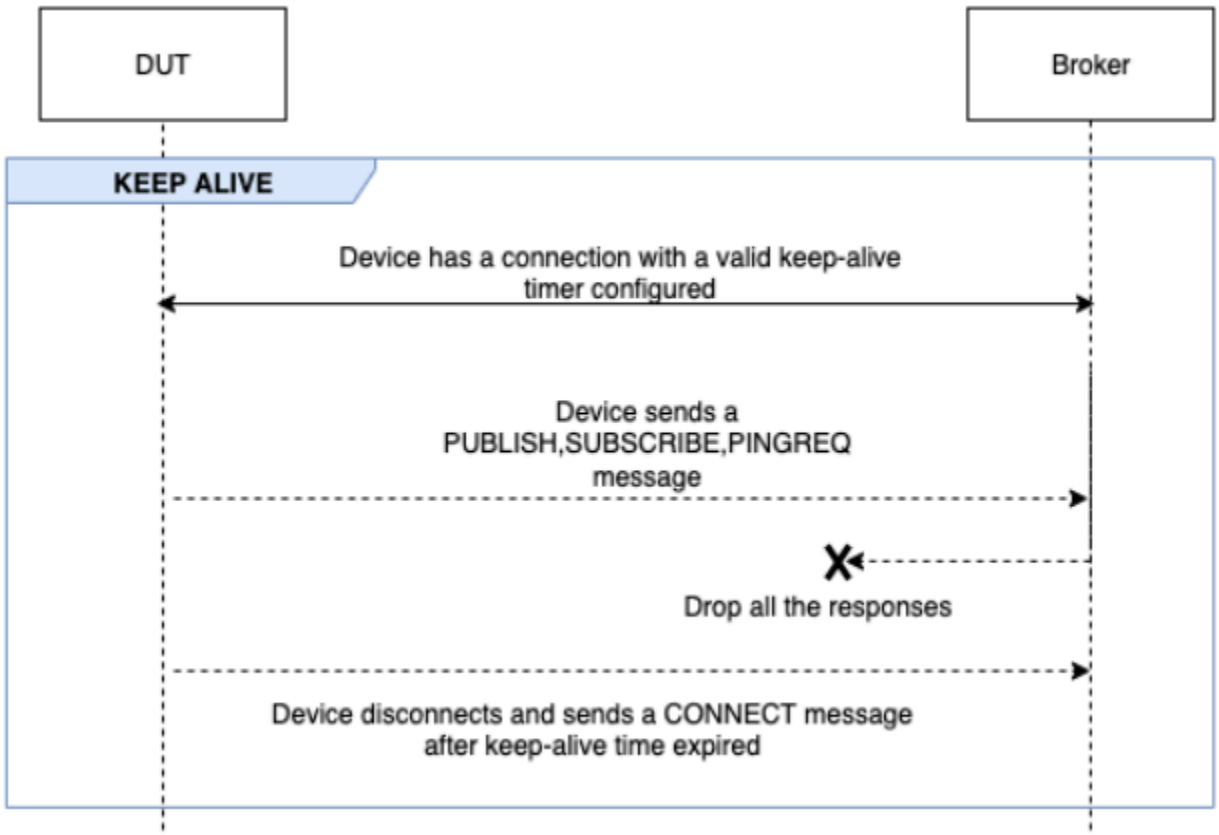
- 디바이스가 활성 QoS 1 구독 및 영구 세션이 활성화된 상태로 브로커에 연결됩니다.
- 세션 중에 디바이스에서 브로커의 연결이 성공적으로 해제됩니다.
- 디바이스가 브로커에 다시 연결되고 해당 주제를 명시적으로 다시 구독하지 않고도 트리거 주제에 대한 구독을 재개합니다.
- 디바이스가 구독한 주제에 대해 브로커가 저장한 메시지를 성공적으로 수신하고 예상대로 실행됩니다.

AWS IoT 영구 세션에 대한 자세한 내용은 MQTT 영구 세션 사용을 참조하세요.



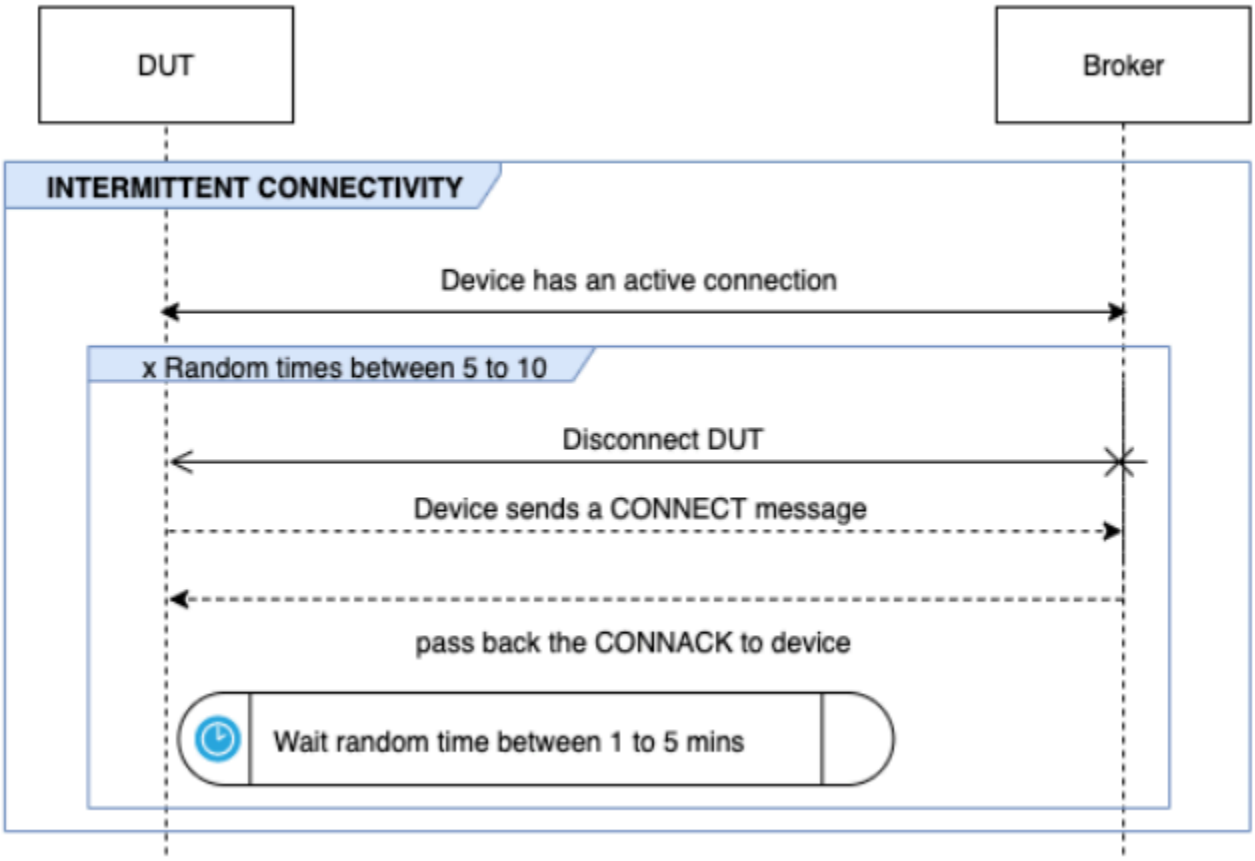
KEEP ALIVE

이 시나리오는 디바이스가 브로커로부터 ping 응답을 받지 못한 후 연결이 성공적으로 해제되는지 확인합니다. 연결에는 유효한 연결 유지 타이머가 구성되어 있어야 합니다. 이 테스트의 일환으로 브로커는 PUBLISH, SUBSCRIBE 및 PINGREQ 메시지에 대해 전송된 모든 응답을 차단합니다. 또한 테스트 중인 디바이스가 MQTT 연결을 해제하는지 검증합니다.



INTERMITTENT CONNECTIVITY

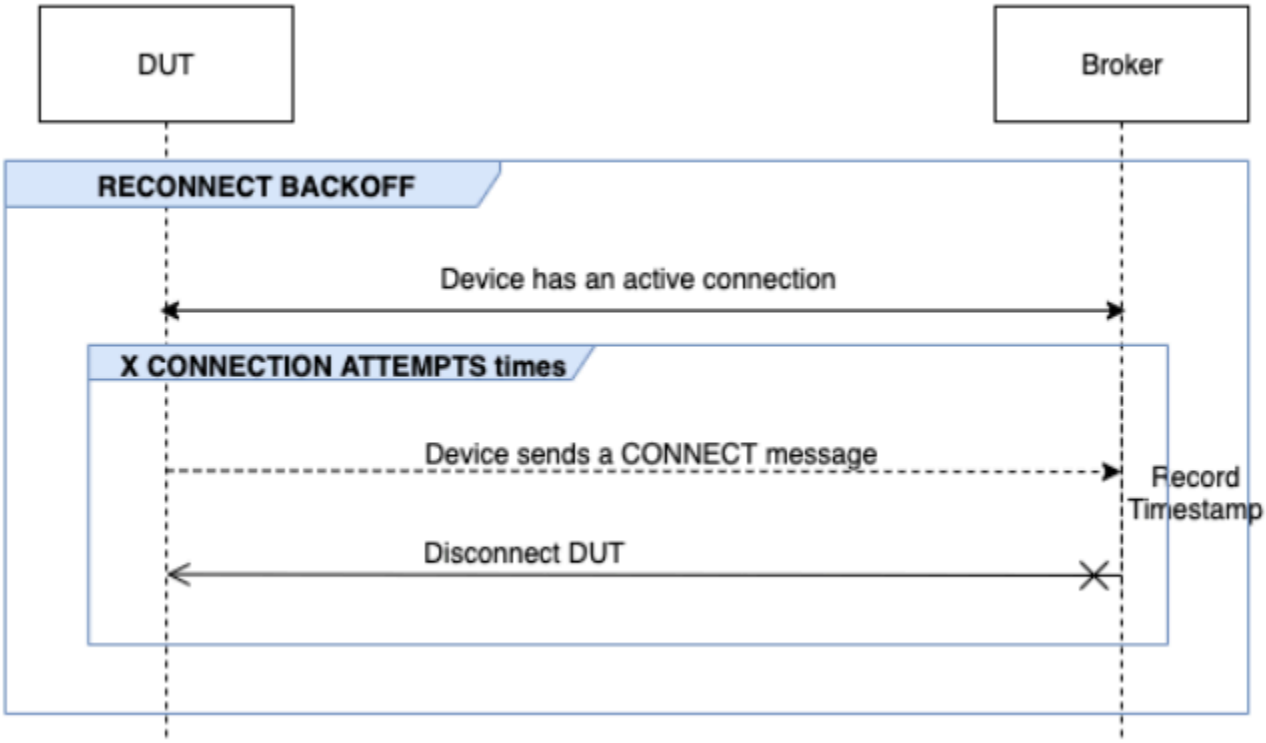
이 시나리오는 브로커가 임의의 시간 동안 임의의 간격으로 디바이스 연결을 해제한 후 디바이스가 브로커에 다시 연결될 수 있는지 확인합니다.



RECONNECT BACKOFF

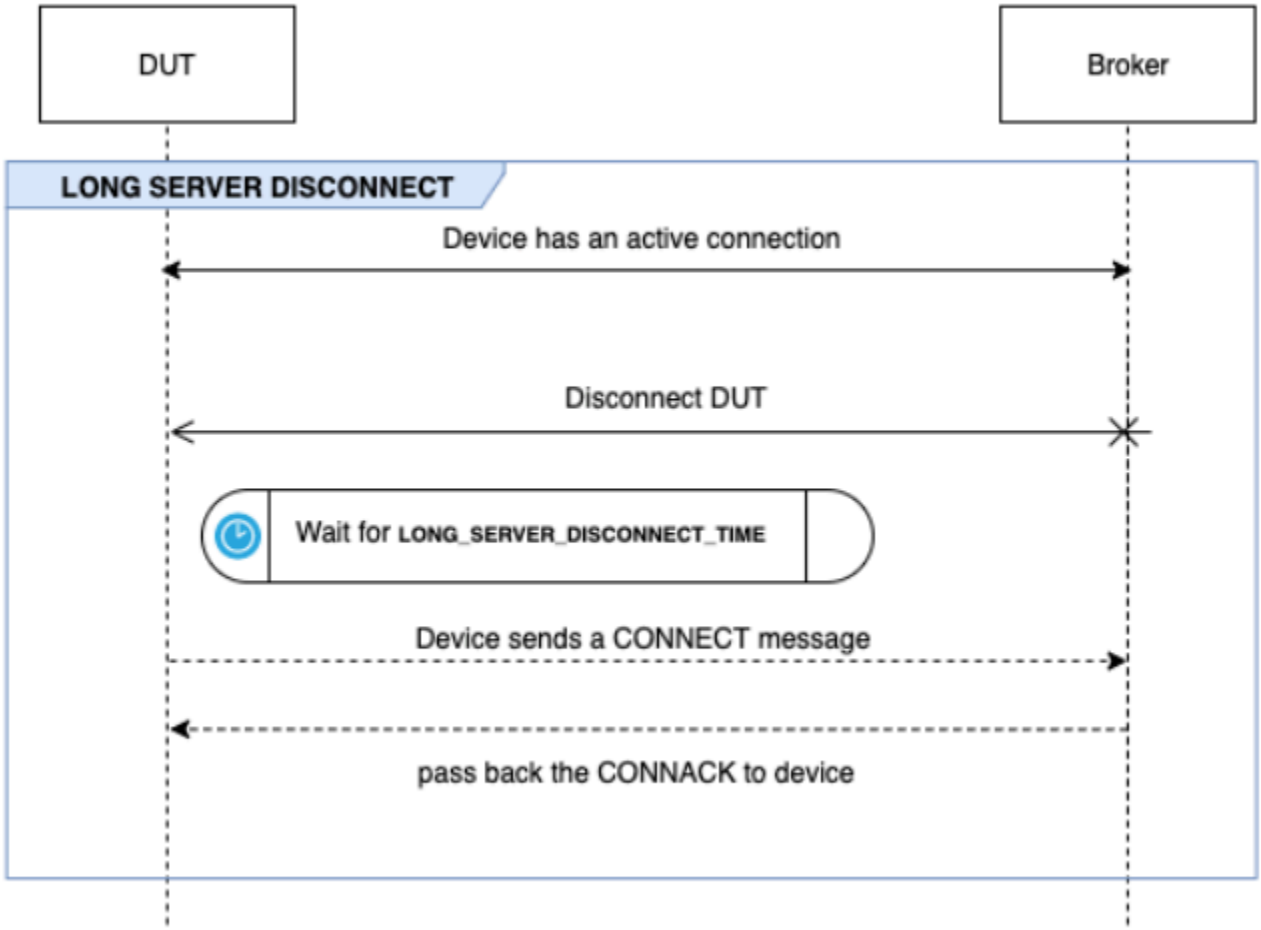
이 시나리오는 브로커가 여러 번 연결을 해제할 때 디바이스에 백오프 메커니즘이 구현되어 있는지 확인합니다. Device Advisor는 백오프 유형을 지수, 지터, 선형 또는 상수로 보고합니다. 백오프 시도 횟수는 `BACKOFF_CONNECTION_ATTEMPTS` 옵션을 사용하여 구성할 수 있습니다. 기본값은 5입니다. 값은 5~10으로 구성 가능합니다.

이 테스트를 통과하려면 테스트 중인 디바이스에서 지수 백오프 및 지터 메커니즘을 구현하는 것이 좋습니다.



LONG SERVER DISCONNECT

이 시나리오는 브로커가 오랜 시간(최대 120분) 동안 디바이스 연결을 해제한 후 디바이스가 성공적으로 다시 연결될 수 있는지 확인합니다. 서버 연결 해제 시간은 `LONG_SERVER_DISCONNECT_TIME` 옵션을 사용하여 구성할 수 있습니다. 기본값은 120분입니다. 이 값은 30분~120분으로 구성할 수 있습니다.



추가 실행 시간

추가 실행 시간은 위의 모든 테스트를 완료한 후 테스트 케이스를 종료하기 전에 테스트가 대기하는 시간입니다. 고객은 이 추가 기간을 사용하여 디바이스와 브로커 간의 모든 통신을 모니터링하고 기록합니다. ADDITIONAL_EXECUTION_TIME 옵션을 사용하여 추가 실행 시간을 구성할 수 있습니다. 기본적으로 이 옵션은 0분으로 설정되며 0분~120분이 될 수 있습니다.

MQTT 장기간 테스트 구성 옵션

MQTT 장기간 테스트를 위해 제공되는 모든 구성 옵션은 선택 사항입니다. 다음 옵션을 사용할 수 있습니다.

OPERATIONS

디바이스에서 수행하는 작업 목록(예: CONNECT, PUBLISH, SUBSCRIBE)입니다. 테스트 케이스는 지정된 작업을 기반으로 시나리오를 실행합니다. 지정되지 않은 작업은 유효한 것으로 간주됩니다.

```
{
"OPERATIONS": ["PUBLISH", "SUBSCRIBE"]
//by default the test assumes device can CONNECT
}
```

SCENARIOS

선택한 작업에 따라 테스트 케이스는 시나리오를 실행하여 디바이스의 동작을 검증합니다. 다음 두 가지 유형의 시나리오가 있습니다.

- 기본 시나리오는 디바이스가 구성의 일부로 위에서 선택한 작업을 수행할 수 있는지 확인하는 간단한 테스트입니다. 이는 구성에 지정된 작업을 기반으로 미리 선택됩니다. 구성에 다른 입력이 필요하지 않습니다.
- 고급 시나리오는 실제 조건이 충족되었을 때 디바이스가 모범 사례를 따르는지 확인하기 위해 디바이스에 대해 수행되는 보다 복잡한 시나리오입니다. 이는 선택 사항이며 테스트 제품군의 구성 입력에 시나리오 배열로 전달할 수 있습니다.

```
{
    "SCENARIOS": [      // list of advanced scenarios
                "PUBACK_QOS_1",
                "RECEIVE_LARGE_PAYLOAD",
                "PERSISTENT_SESSION",
                "KEEP_ALIVE",
                "INTERMITTENT_CONNECTIVITY",
                "RECONNECT_BACK_OFF",
                "LONG_SERVER_DISCONNECT"
    ]
}
```

BASIC_TESTS_EXECUTION_TIME_OUT:

테스트 케이스에서 모든 기본 테스트가 완료될 때까지 기다리는 최대 시간입니다. 기본값은 60분입니다. 이 값은 30분~120분으로 구성할 수 있습니다.

LONG_SERVER_DISCONNECT_TIME:

Long Server Disconnect 테스트 중에 테스트 케이스가 디바이스의 연결을 해제했다가 다시 연결하는 데 걸리는 시간입니다. 기본값은 60분입니다. 이 값은 30분~120분으로 구성할 수 있습니다.

ADDITIONAL_EXECUTION_TIME:

이 옵션을 구성하면 모든 테스트가 완료된 후 디바이스와 브로커 간의 이벤트를 모니터링할 수 있는 시간이 제공됩니다. 기본값은 0분입니다. 이 값은 0분~120분으로 구성할 수 있습니다.

BACKOFF_CONNECTION_ATTEMPTS:

이 옵션은 테스트 케이스에서 디바이스의 연결을 해제하는 횟수를 구성합니다. 이는 Reconnect Backoff 테스트에서 사용됩니다. 기본값은 5회입니다. 이 값은 5~10으로 구성할 수 있습니다.

LONG_PAYLOAD_FORMAT:

테스트 케이스가 디바이스에서 구독하는 QoS 1 주제에 게시할 때 디바이스에서 예상하는 메시지 페이로드의 형식입니다.

API 테스트 케이스 정의:

```
{
"tests":[
   {
      "name":"my_mqtt_long_duration_test",
      "configuration": {
         // optional
         "OPERATIONS": ["PUBLISH", "SUBSCRIBE"],
         "SCENARIOS": [
            "LONG_SERVER_DISCONNECT",
            "RECONNECT_BACK_OFF",
            "KEEP_ALIVE",
            "RECEIVE_LARGE_PAYLOAD",
            "INTERMITTENT_CONNECTIVITY",
            "PERSISTENT_SESSION",
         ],
         "BASIC_TESTS_EXECUTION_TIMEOUT": 60, // in minutes (60 minutes by default)
         "LONG_SERVER_DISCONNECT_TIME": 60,   // in minutes (120 minutes by default)
         "ADDITIONAL_EXECUTION_TIME": 60,     // in minutes (0 minutes by default)
         "BACKOFF_CONNECTION_ATTEMPTS": "5",
         "LONG_PAYLOAD_FORMAT":"{"message":"${payload}"}"
      },
      "test":{
         "id":"MQTT_Long_Duration",
         "version":"0.0.0"
      }
   }
 ]
```

```
}
```

MQTT 장기간 테스트 케이스 요약 로그

MQTT 장기간 테스트 케이스는 일반 테스트 케이스보다 더 오래 실행됩니다. 실행 중 디바이스 연결, 게시 및 구독과 같은 중요한 이벤트를 나열하는 별도의 요약 로그가 제공됩니다. 세부 정보에는 테스트된 항목, 테스트되지 않은 항목 및 실패한 항목이 포함됩니다. 로그 끝부분에는 테스트 케이스 실행 중에 발생한 모든 이벤트의 요약이 포함됩니다. 여기에는 다음이 포함됩니다.

- 디바이스에 구성된 연결 유지 타이머
- 디바이스에 구성된 영구 세션 플래그
- 테스트 실행 중 디바이스 연결 수
- 디바이스 재연결 백오프 유형(재연결 백오프 테스트에 대해 검증된 경우)
- 테스트 케이스 실행 중에 디바이스가 게시한 주제
- 테스트 케이스 실행 중에 디바이스가 구독한 주제

AWS IoT Core 디바이스 위치

AWS IoT Core 디바이스 위치 기능을 사용하기 전에이 기능의 이용 약관을 검토하세요. AWS 는 검색을 실행하는 데 사용되는 위치 데이터 및 기타 정보와 같은 지리적 위치 검색 요청 파라미터를 현재 사용 중인 외부에 있을 수 있는 선택한 타사 데이터 공급자에게 전송할 수 AWS 리전 있습니다. 사용할 서드파티 공급자와 솔버는 수신된 입력 페이로드를 기반으로 선택됩니다. 자세한 내용은 AWS 서비스 약관을 참조하세요.

AWS IoT Core 디바이스 위치를 사용하여 타사 솔버를 사용하여 IoT 디바이스의 위치를 테스트합니다. 솔버는 측정 데이터를 확인하고 디바이스의 위치를 추정하는 서드 파티 공급업체에서 제공하는 알고리즘입니다. 디바이스의 위치를 식별하면 현장에서 디바이스를 추적하고 디버깅하여 문제를 해결할 수 있습니다.

다양한 소스에서 수집된 측정 데이터가 확인되고 지리적 위치 정보가 GeoJSON 페이로드로 보고됩니다. GeoJSON 형식은 지리적 데이터 구조를 인코딩하는 데 사용되는 형식입니다. 페이로드에는 세계 측지 시스템 좌표계(WGS84)를 기반으로 하는 디바이스 위치의 위도 및 경도 좌표가 포함됩니다.

주제

- 측정 유형 및 솔버
- AWS IoT Core 디바이스 위치 작동 방식
- AWS IoT Core 디바이스 위치를 사용하는 방법
- IoT 디바이스의 위치 확인
- 디바이스 위치 MQTT 주제를 사용하여 AWS IoT Core 디바이스 위치 확인
- 위치 솔버 및 디바이스 페이로드

측정 유형 및 솔버

AWS IoT Core Device Location은 타사 공급업체와 협력하여 측정 데이터를 해결하고 예상 디바이스 위치를 제공합니다. 다음 표에는 측정 유형과 서드 파티 위치 솔버, 지원되는 디바이스에 대한 정보가 나와 있습니다. LoRaWAN 디바이스 및 해당 디바이스의 위치 구성에 대한 자세한 내용은 LoRaWAN 리소스의 위치 구성 섹션을 참조하세요.

> **Note**
>
> 일반 IoT 디바이스 및 Sidewalk 디바이스는 디바이스 위치 MQTT 주제를 사용하여 위치 정보를 얻을 수 있습니다. Wi-Fi, 셀룰러 및 IP 주소 측정 유형의 경우 디바이스가 정의된 GeoJSON 형식으로 예약된 주제에 측정 데이터를 게시하는 경우 AWS IoT Core 디바이스 위치는 디바이스의 위치를 확인할 수 있습니다. GNSS 측정 유형의 경우 GNSS 솔버를 사용하여 확인된 위치 정보를 얻기 위해 측정 데이터를 스캔하려면 디바이스에 LR11xx 칩이 있어야 합니다. LoRaWAN 디바이스의 위치 정보를 가져오는 방법에 대한 자세한 내용은 AWS IoT Wireless 설명서의 Configure position for LoRaWAN resources를 참조하세요.

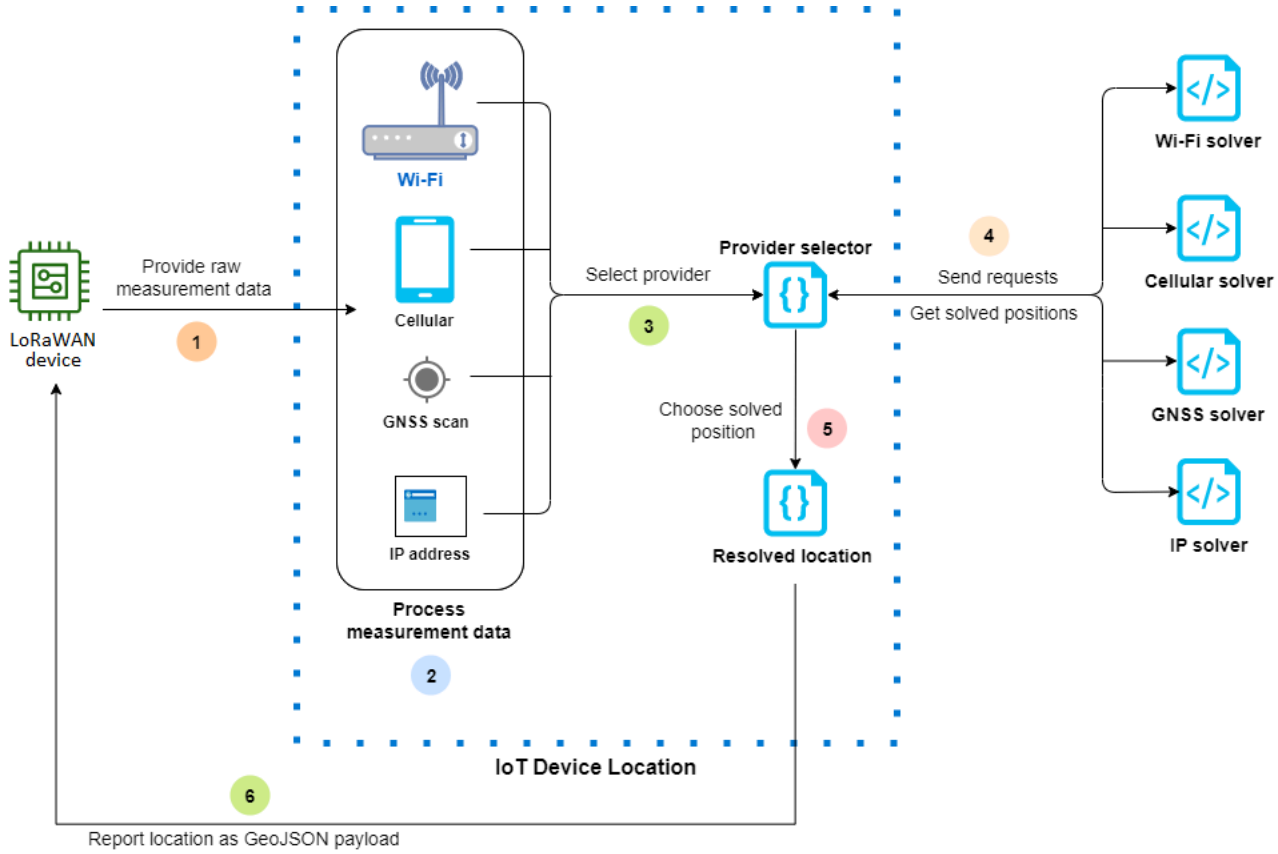
측정 유형 및 솔버

측정 유형	서드 파티 솔버	지원되는 디바이스
Wi-Fi 액세스 포인트	Wi-Fi 기반 솔버	일반 IoT 디바이스, LoRaWAN 및 Amazon Sidewalk 디바이스
셀룰러 라디오 타워: GSM, LTE, CDMA, SCDMA, WCMDA 및 TD-SCDMA 데이터	셀룰러 기반 솔버	일반 IoT 디바이스, LoRaWAN 및 Amazon Sidewalk 디바이스
IP 주소	IP 역방향 조회 솔버	TCP/IP를 통해 연결하는 모든 IoT 디바이스
GNSS 스캔 데이터(NAV 메시지)	GNSS 솔버	일반 IoT 디바이스, LoRaWAN 및 Amazon Sidewalk 디바이스
Bluetooth Low Energy(BLE)	BLE 기반 솔버	Amazon Sidewalk 디바이스

위치 솔버에 대한 자세한 내용과 다양한 측정 유형의 디바이스 페이로드를 보여주는 예는 위치 솔버 및 디바이스 페이로드 섹션을 참조하세요.

AWS IoT Core 디바이스 위치 작동 방식

다음 다이어그램은 AWS IoT Core Device Location이 측정 데이터를 수집하고 디바이스의 위치 정보를 확인하는 방법을 보여줍니다.



다음 단계에서는 AWS IoT Core Device Location의 작동 방식을 보여줍니다.

1. 측정 데이터 수신

   먼저 디바이스 위치와 관련된 원시 측정 데이터가 디바이스에서 전송됩니다. 측정 데이터는 JSON 페이로드로 지정됩니다.

2. 측정 데이터 처리

   측정 데이터가 처리되고 AWS IoT Core 디바이스 위치는 사용할 측정 데이터를 선택합니다.이 데이터는 Wi-Fi, 셀룰러, GNSS 스캔 또는 IP 주소 정보일 수 있습니다.

3. 솔버 선택

   측정 데이터를 기반으로 서드 파티 솔버가 선택됩니다. 예를 들어, 측정 데이터에 Wi-Fi 및 IP 주소 정보가 포함된 경우 Wi-Fi 솔버와 IP 역방향 조회 솔버가 선택됩니다.

4. 확인된 위치 가져오기

   API 요청은 위치를 확인하도록 요청하는 솔버 공급자에게 전송됩니다. 그러면 AWS IoT Core 디바이스 위치는 솔버에서 예상 지리적 위치 정보를 가져옵니다.

5. 확인된 위치 선택

   확인된 위치 정보와 정확도를 비교하고 AWS IoT Core 디바이스 위치는 정확도가 가장 높은 지리적 위치 결과를 선택합니다.

6. 위치 정보 출력

   지리적 위치 정보가 GeoJSON 페이로드로 전송됩니다. 페이로드에는 WGS84 지리 좌표, 정확도 정보, 신뢰 수준, 확인된 위치를 가져온 타임스탬프가 포함됩니다.

AWS IoT Core 디바이스 위치를 사용하는 방법

다음 단계에서는 AWS IoT Core 디바이스 위치를 사용하는 방법을 보여줍니다.

1. 측정 데이터 제공

   디바이스 위치와 관련된 원시 측정 데이터를 JSON 페이로드로 지정합니다. 페이로드 측정 데이터를 검색하려면 디바이스 로그로 이동하거나 CloudWatch Logs를 사용하여 페이로드 데이터 정보를 복사합니다. JSON 페이로드에는 하나 이상의 데이터 측정 유형이 포함되어야 합니다. 다양한 솔버의 페이로드 형식을 보여주는 예는 위치 솔버 및 디바이스 페이로드 섹션을 참조하세요.

2. 위치 정보 확인

   AWS IoT 콘솔의 디바이스 위치 페이지 또는 GetPositionEstimate API 작업을 사용하여 페이로드 측정 데이터를 전달하고 디바이스 위치를 확인합니다. 그런 다음 AWS IoT Core 디바이스 위치는 정확도가 가장 높은 솔버를 선택하고 디바이스 위치를 보고합니다. 자세한 내용은 IoT 디바이스의 위치 확인 단원을 참조하십시오.

3. 위치 정보 복사

   AWS IoT Core 디바이스 위치에서 확인되고 GeoJSON 페이로드로 보고된 지리적 위치 정보를 확인합니다. 애플리케이션 및 기타 AWS 서비스에 사용할 페이로드를 복사할 수 있습니다. 예를 들어 Location AWS IoT 규칙 작업을 사용하여 지리적 위치 데이터를 Amazon Location Service로 전송할 수 있습니다.

다음 주제에서는 AWS IoT Core 디바이스 위치를 사용하는 방법과 디바이스 위치 페이로드의 예를 보여줍니다.

- IoT 디바이스의 위치 확인
- 위치 솔버 및 디바이스 페이로드

IoT 디바이스의 위치 확인

AWS IoT Core 디바이스 위치를 사용하여 디바이스에서 측정 데이터를 디코딩하고 타사 솔버를 사용하여 디바이스 위치를 확인합니다. 확인된 위치는 지리 좌표 및 정확도 정보가 포함된 GeoJSON 페이로드로 생성됩니다. AWS IoT 콘솔, AWS IoT 무선 API 또는에서 디바이스의 위치를 확인할 수 있습니다 AWS CLI.

주제

- 디바이스 위치 확인(콘솔)
- 디바이스 위치 확인(API)
- 위치를 확인할 때 발생하는 오류 해결

디바이스 위치 확인(콘솔)

디바이스 위치를 확인하려면(콘솔)

1. AWS IoT 콘솔의 디바이스 위치 페이지로 이동합니다.
2. 디바이스 로그 또는 CloudWatch Logs에서 페이로드 측정 데이터를 가져와서 페이로드를 통한 위치 해석 섹션에 입력합니다.

   다음 코드는 샘플 JSON 페이로드를 보여줍니다. 페이로드에는 셀룰러 및 Wi-Fi 측정 데이터가 포함됩니다. 페이로드에 추가 유형의 측정 데이터가 포함된 경우 정확도가 가장 높은 솔버가 사용됩니다. 자세한 정보와 페이로드의 예는 the section called “위치 솔버 및 디바이스 페이로드” 섹션을 참조하세요.

   > **Note**
   >
   > JSON 페이로드에는 하나 이상의 측정 데이터 유형이 포함되어야 합니다.

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress": "54.240.198.35"
    },
    "WiFiAccessPoints": [{
        "MacAddress": "A0:EC:F9:1E:32:C1",
        "Rss": -77
```

```
    }],
    "CellTowers": {
        "Gsm": [{
            "Mcc": 262,
            "Mnc": 1,
            "Lac": 5126,
            "GeranCid": 16504,
            "GsmLocalId": {
                "Bsic": 6,
                "Bcch": 82
            },
            "GsmTimingAdvance": 1,
            "RxLevel": -110,
            "GsmNmr": [{
                "Bsic": 7,
                "Bcch": 85,
                "RxLevel": -100,
                "GlobalIdentity": {
                    "Lac": 1,
                    "GeranCid": 1
                }
            }]
        }],
        "Wcdma": [{
            "Mcc": 262,
            "Mnc": 7,
            "Lac": 65535,
            "UtranCid": 14674663,
            "WcdmaNmr": [{
                    "Uarfcndl": 10786,
                    "UtranCid": 14674663,
                    "Psc": 149
                },
                {
                    "Uarfcndl": 10762,
                    "UtranCid": 14674663,
                    "Psc": 211
                }
            ]
        }],
        "Lte": [{
            "Mcc": 262,
            "Mnc": 2,
            "EutranCid": 2898945,
```

```
            "Rsrp": -50,
            "Rsrq": -5,
            "LteNmr": [{
                    "Earfcn": 6300,
                    "Pci": 237,
                    "Rsrp": -60,
                    "Rsrq": -6,
                    "EutranCid": 2898945
                },
                {
                    "Earfcn": 6300,
                    "Pci": 442,
                    "Rsrp": -70,
                    "Rsrq": -7,
                    "EutranCid": 2898945
                }
            ]
        }]
    }
}
```

3. 위치 정보를 확인하려면 Resolve(확인)를 선택합니다.

   위치 정보는 blob 유형이며 지리적 데이터 구조를 인코딩하는 데 사용되는 GeoJSON 형식의 페이로드로 반환됩니다. 페이로드에는 다음이 포함됩니다.

   - 위도 및 경도 정보를 포함하는 WGS84 지리 좌표. 고도 정보도 포함될 수 있습니다.
   - 보고된 위치 정보의 유형(예: 포인트). 포인트 위치 유형은 GeoJSON 포인트로 인코딩된 WGS84 위도 및 경도로 위치를 나타냅니다.
   - 해석기에서 추정된 위치 정보와 실제 디바이스 위치 간의 차이를 미터로 나타내는 수평 및 수직 정확도 정보.
   - 위치 추정치 응답의 불확실성을 나타내는 신뢰 수준. 기본값은 0.68이며, 이는 실제 디바이스 위치가 추정된 위치의 불확실성 반경 내에 있을 확률이 68%임을 나타냅니다.
   - 디바이스가 위치한 도시, 주, 국가 및 우편번호. 이 정보는 IP 역방향 조회 해석기를 사용하는 경우에만 보고됩니다.
   - 위치가 확인된 날짜와 시간에 해당하는 타임스탬프 정보. Unix 타임스탬프 형식이 사용됩니다.

   다음 코드는 위치를 확인하여 반환된 샘플 GeoJSON 페이로드를 보여줍니다.

> **Note**
>
> AWS IoT Core Device Location이 위치 확인을 시도할 때 오류를 보고하는 경우 오류를 해결하고 위치를 해결할 수 있습니다. 자세한 내용은 위치를 확인할 때 발생하는 오류 해결 단원을 참조하십시오.


```
{
    "coordinates": [
        13.376076698303223,
        52.51823043823242
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 45,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 303,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T12:23:58.189Z"
    }
}
```


4. 리소스 위치 섹션으로 이동하여 AWS IoT Core 디바이스 위치에서 보고하는 지리적 위치 정보를 확인합니다. 다른 애플리케이션 및에 사용할 AWS 서비스페이로드를 복사할 수 있습니다. 예를 들어, Location를 사용하여 지리적 위치 데이터를 Amazon Location Service로 전송할 수 있습니다.

디바이스 위치 확인(API)

AWS IoT 무선 API를 사용하여 디바이스 위치를 확인하려면 GetPositionEstimate API 작업 또는 get-position-estimate CLI 명령을 사용합니다. 페이로드 측정 데이터를 입력으로 지정하고 API 작업을 실행하여 디바이스 위치를 확인합니다.

> **Note**
>
> `GetPositionEstimate` API 작업은 디바이스 또는 상태 정보를 저장하지 않으며 과거 위치 데이터를 검색하는 데 사용할 수 없습니다. 측정 데이터를 확인하고 추정된 위치를 산출하는 일회성 작업을 수행합니다. 위치 정보를 검색하려면 이 API 작업을 수행할 때마다 페이로드 정보를 지정해야 합니다.

다음 명령은 이 API 작업을 사용하여 위치를 확인하는 방법의 예를 보여줍니다.

> **Note**
>
> `get-position-estimate` CLI 명령을 실행할 때 출력 JSON 파일을 첫 번째 입력으로 지정해야 합니다. 이 JSON 파일은 CLI에서 응답으로 얻은 예상 위치 정보를 GeoJSON 형식으로 저장합니다. 예를 들어 다음 명령은 위치 정보를 *`locationout.json`* 파일에 저장합니다.

```
aws iotwireless get-position-estimate locationout.json \
    --ip IpAddress=""54.240.198.35"" \
    --wi-fi-access-points \
        MacAddress="A0:EC:F9:1E:32:C1",Rss=-75 \
        MacAddress="A0:EC:F9:15:72:5E",Rss=-67
```

이 예제에는 측정 유형으로 Wi-Fi 액세스 포인트와 IP 주소가 모두 포함됩니다. AWS IoT Core 디바이스 위치는 Wi-Fi 솔버와 IP 역방향 조회 솔버 중에서 선택하고 정확도가 더 높은 솔버를 선택합니다.

확인돈 위치는 지리적 데이터 구조를 인코딩하는 데 사용되는 GeoJSON 형식의 페이로드로 반환됩니다. 그런 다음 이 정보는 *`locationout.json`* 파일에 저장됩니다. 페이로드에는 WGS84 위도 및 경도 좌표, 정확도 및 신뢰 수준 정보, 위치 데이터 유형, 위치가 확인된 타임스탬프가 포함됩니다.

```
{
    "coordinates": [
        13.37704086303711,
        52.51865005493164
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 707,
        "verticalConfidenceLevel": 0.68,
```

```
        "horizontalAccuracy": 389,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T14:03:57.391Z"
    }
}
```

위치를 확인할 때 발생하는 오류 해결

위치를 확인하려고 하면 다음 오류 코드 중 하나가 표시될 수 있습니다. AWS IoT Core 디바이스 위치는 `GetPositionEstimate` API 작업을 사용할 때 오류를 생성하거나 AWS IoT 콘솔의 오류에 해당하는 행 번호를 참조할 수 있습니다.

- 400 오류

  이 오류는 디바이스 페이로드 JSON의 형식을 AWS IoT Core 디바이스 위치에서 확인할 수 없음을 나타냅니다. 다음과 같은 이유로 오류가 발생할 수 있습니다.

  - JSON 측정 데이터의 형식이 잘못되었습니다.
  - 페이로드에 타임스탬프 정보만 포함되어 있습니다.
  - IP 주소와 같은 측정 데이터 파라미터가 유효하지 않습니다.

  이 오류를 해결하려면 JSON 형식이 올바른지, 하나 이상의 측정 유형의 데이터가 입력으로 포함되어 있는지 확인합니다. IP 주소가 잘못된 경우 오류를 해결하기 위해 유효한 IP 주소를 제공하는 방법에 대한 자세한 내용은 IP 역방향 조회 솔버 섹션을 참조하세요.

- 403 오류

  이 오류는 API 작업을 수행하거나 AWS IoT 콘솔을 사용하여 디바이스 위치를 검색할 수 있는 권한이 없음을 나타냅니다. 이 오류를 해결하려면 이 작업을 수행하는 데 필요한 권한이 있는지 확인합니다. 세션 또는 AWS Management Console 세션 AWS CLI 토큰이 만료된 경우이 오류가 발생할 수 있습니다. 이 오류를 해결하려면를 사용하도록 세션 토큰을 새로 고치거나 AWS CLI에서 로그아웃 AWS Management Console 한 다음 자격 증명을 사용하여 로그인합니다.

- 404 오류

  이 오류는 AWS IoT Core 디바이스 위치에서 위치 정보를 찾거나 해결하지 못했음을 나타냅니다. 측정 데이터 입력에 데이터가 충분하지 않은 경우 등으로 인해 오류가 발생할 수 있습니다. 예제:

  - MAC 주소 또는 셀룰러 타워 정보가 충분하지 않습니다.

- 위치 조회 및 검색에 IP 주소를 사용할 수 없습니다.
- GNSS 페이로드가 충분하지 않습니다.

  이러한 경우 오류를 해결하려면 측정 데이터에 디바이스 위치 확인에 필요한 충분한 정보가 포함되어 있는지 확인합니다.

- 500 오류

  이 오류는 AWS IoT Core Device Location에서 위치 확인을 시도할 때 내부 서버 예외가 발생했음을 나타냅니다. 이 오류를 해결하려면 세션을 새로 고치고 확인할 측정 데이터를 다시 전송해 봅니다.

디바이스 위치 MQTT 주제를 사용하여 AWS IoT Core 디바이스 위치 확인

예약된 MQTT 주제를 사용하여 디바이스 위치 기능을 통해 AWS IoT Core 디바이스의 최신 위치 정보를 가져올 수 있습니다.

디바이스 위치 MQTT 주제의 형식

AWS IoT Core 디바이스 위치에 대한 예약 주제는 다음 접두사를 사용합니다.

`$aws/device_location/{customer_device_id}/`

전체 주제를 만들려면 먼저 디바이스를 식별하는 데 사용하는 고유 ID로 *`customer_device_id`*를 바꾸세요. 디바이스가 AWS IoT 사물로 등록된 *`thingName`*경우 for LoRaWAN 및 Sidewalk 디바이스와 `WirelessDeviceId`같은와를 지정하는 것이 좋습니다. 그런 다음, 다음 섹션에 표시된 것과 같이 주제에 `get_position_estimate` 및 `get_position_estimate/accepted` 등의 주제 스텁을 추가합니다.

> **Note**
>
> *`{customer_device_id}`*에는 문자, 숫자 및 대시만 포함할 수 있습니다. 디바이스 위치 주제를 구독할 때는 더하기 기호(+)만 와일드카드 문자로 사용할 수 있습니다. 예를 들어, *`{customer_device_id}`*에 + 와일드카드를 사용하여 디바이스의 위치 정보를 얻을 수 있습니다. `$aws/device_location/+/get_position_estimate/accepted` 주제를 구독하는 경우 성공적으로 해석되면 임의의 디바이스 ID와 일치하는 디바이스의 위치 정보가 포함된 메시지가 게시됩니다.

다음은 AWS IoT Core 디바이스 위치와 상호 작용하는 데 사용되는 예약된 주제입니다.

디바이스 위치 MQTT 주제

Topic	허용된 작업	설명
$aws/device_location/*customer_device_id*/get_position_estimate	게시	디바이스는이 주제에 게시하여 디바이스 AWS IoT Core 위치에서 스캔한 원시 측정 데이터를 확인할 수 있습니다.
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	Subscribe	AWS IoT Core Device Location은 디바이스 위치를 성공적으로 확인하면이 주제에 위치 정보를 게시합니다.
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	Subscribe	AWS IoT Core 디바이스 위치가 디바이스 위치를 확인하지 못하면이 주제에 오류 정보를 게시합니다.

디바이스 위치 MQTT 주제 정책

디바이스 위치 주제에서 메시지를 수신하려면 디바이스가 AWS IoT 디바이스 게이트웨이에 연결하고 MQTT 주제를 구독하도록 허용하는 정책을 사용해야 합니다.

다음은 다양한 주제에 대한 메시지를 수신하는 데 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/rejected"
            ]
        }
    ]
}
```

디바이스 위치 주제 및 페이로드

다음은 AWS IoT Core 디바이스 위치 주제, 메시지 페이로드의 형식 및 각 주제에 대한 예제 정책을 보여줍니다.

주제

- /get_position_estimate
- /get_position_estimate/accepted
- /get_position_estimate/rejected

/get_position_estimate

이 주제에 메시지를 게시하여 디바이스 위치에서 확인할 AWS IoT Core 디바이스의 원시 측정 데이터를 가져옵니다.

```
$aws/device_location/customer_device_id/get_position_estimate
```

AWS IoT Core Device Location은 /get_position_estimate/accepted 또는에 게시하여 응답합니다/get_position_estimate/rejected.

> **Note**
>
> 이 주제에 게시된 메시지는 유효한 JSON 페이로드여야 합니다. 입력 메시지가 유효한 JSON 형식이 아닌 경우 응답을 받을 수 없습니다. 자세한 내용은 메시지 페이로드를 참조하세요.

메시지 페이로드

메시지 페이로드 형식은 AWS IoT Wireless API 작업 요청 본문인와 유사한 구조를 따릅니다GetPositionEstimate. OTA 업데이트는 다음을 포함합니다.

- 위치가 해석된 날짜와 시간에 해당하는 선택적 Timestamp 문자열. Timestamp 문자열의 최소 길이는 1자이고 최대 길이는 10자입니다.
- 요청을 응답에 매핑하는 데 사용할 수 있는 선택적 MessageId 문자열. 이 문자열을 지정하면 get_position_estimate/accepted 또는 get_position_estimate/rejected 주제에 게시된 메시지에 이 MessageId가 포함됩니다. MessageID 문자열의 최소 길이는 1자이고 최대 길이는 256자입니다.
- 다음 측정 유형 중 하나 이상을 포함하는 디바이스의 측정 데이터:
  - WiFiAccessPoint
  - CellTowers
  - IpAddress
  - Gnss

다음은 샘플 메시지 페이로드를 보여줍니다.

```
{
```

```
    "Timestamp": "1664313161",
    "MessageId": "ABCD1",
    "WiFiAccessPoints":  [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",
            "Rss": -66
        }
    ],
    "Ip":{
        "IpAddress": "54.192.168.0"
    },
    "Gnss":{
      "Payload":"8295A614A2029517F4F77C0A7823B161A6FC57E25183D96535E3689783F6CA48",
      "CaptureTime":1354393948
    }
}
```

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate"
            ]
        }
    ]
}
```

/get_position_estimate/accepted

AWS IoT Core Device Location은 디바이스에 대해 확인된 위치 정보를 반환할 때이 주제에 대한 응답을 게시합니다. 위치 정보는 GeoJSON 형식으로 반환됩니다.

```
$aws/device_location/customer_device_id/get_position_estimate/accepted
```

다음은 메시지 페이로드와 예시 정책을 보여줍니다.

메시지 페이로드

다음은 GeoJSON 형식의 메시지 페이로드 예시입니다. 원시 측정 데이터에 MessageId를 지정하고 AWS IoT Core 디바이스 위치가 위치 정보를 성공적으로 해결한 경우 메시지 페이로드는 동일한 MessageId 정보를 반환합니다.

```
{
    "coordinates": [
        13.37704086303711,
        52.51865005493164
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 707,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 389,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T14:03:57.391Z",
        "messageId": "ABCD1"
    }
}
```

예제 정책

다음은 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/accepted"
            ]
        }
    ]
}
```

/get_position_estimate/rejected

AWS IoT Core 디바이스 위치가 디바이스 위치를 확인하지 못하면이 주제에 대한 오류 응답을 게시합니다.

```
$aws/device_location/customer_device_id/get_position_estimate/rejected
```

다음은 메시지 페이로드와 예시 정책을 보여줍니다. 이러한 오류에 대한 자세한 내용은 위치를 확인할 때 발생하는 오류 해결 섹션을 참조하세요.

메시지 페이로드

다음은 AWS IoT Core 디바이스 위치가 위치 정보를 확인하지 못한 이유를 나타내는 오류 코드와 메시지를 제공하는 메시지 페이로드의 예입니다. 원시 측정 데이터를 제공할 `MessageId` 때를 지정했고 AWS IoT Core 디바이스 위치가 위치 정보를 확인하지 못한 경우 메시지 페이로드에 동일한 `MessageId` 정보가 반환됩니다.

```
{
    "errorCode": 500,
    "errorMessage":"Internal server error",
    "messageId": "ABCD1"
```

```
}
```

예제 정책

다음은 필요한 정책의 예입니다.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/device_location/
device-123/get_position_estimate/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/device-123/
get_position_estimate/rejected"
      ]
    }
  ]
}
```

위치 솔버 및 디바이스 페이로드

Location 솔버는 IoT 디바이스의 위치를 확인하는 데 사용할 수 있는 알고리즘입니다. AWS IoT Core Device Location은 다음 위치 솔버를 지원합니다. 여기서는 이러한 측정 유형에 대한 JSON 페이로드 형식의 예, 솔버에서 지원하는 디바이스 및 위치 확인 방법을 보여줍니다.

디바이스 위치를 확인하려면 이러한 측정 데이터 유형 중 하나 이상을 지정합니다. 결합된 모든 측정 데이터에 대해 확인된 위치 하나가 반환됩니다.

주제

- Wi-Fi 기반 솔버
- 셀룰러 기반 솔버
- IP 역방향 조회 솔버
- GNSS 솔버
- Amazon Sidewalk 지원 디바이스용 BLE 솔버

Wi-Fi 기반 솔버

Wi-Fi 액세스 포인트의 스캔 정보를 사용하여 위치를 확인하려면 Wi-Fi 기반 솔버를 사용합니다. 솔버는 무선랜 기술을 지원하며 TCP/IP, LoRaWAN 무선 디바이스 및 Amazon Sidewalk 지원 디바이스를 통해 연결하는 일반 IoT 디바이스의 디바이스 위치를 계산하는 데 사용할 수 있습니다.

LoRaWAN 디바이스에는 들어오는 Wi-Fi 스캔 정보를 디코딩할 수 있는 LoRa Edge 칩셋이 있어야 합니다. LoRa Edge는 장거리 LoRa 송수신장치, 다중 위성군 GNSS 스캐너 및 패시브 Wi-Fi MAC 스캐너를 통합한 초저전력 플랫폼으로, 지리적 위치 애플리케이션을 대상으로 합니다. 디바이스에서 업링크 메시지를 수신하면 Wi-Fi 스캔 데이터가 AWS IoT Core 디바이스 위치로 전송되고 위치는 Wi-Fi 스캔 결과를 기반으로 추정됩니다. 그런 다음 디코딩된 정보가 Wi-Fi 기반 솔버로 전달되어 위치 정보가 검색됩니다.

Amazon Sidewalk 지원 디바이스의 위치 데이터 해결에 대한 자세한 내용은 AWS IoT Core for Amazon Sidewalk 개발자 안내서를 참조하세요.

Wi-Fi 기반 솔버 페이로드 예시

다음 코드는 측정 데이터가 포함된 디바이스의 JSON 페이로드에 대한 예를 보여줍니다. AWS IoT Core Device Location은이 데이터를 입력으로 수신하면 솔버 공급자에게 HTTP 요청을 보내 위치 정보를 확인합니다. 정보를 검색하려면 MAC 주소와 수신 신호 강도(RSS) 값을 지정합니다. 이렇게 하려면 이 형식을 사용하여 JSON 페이로드를 제공하거나 GetPositionEstimate API 작업의 WiFiAccessPoints 객체 파라미터를 사용합니다.

```
{
    "Timestamp": 1664313161,    // optional
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",  // required
            "Rss": -75                          // required
        }
    ]
```

```
}
```

셀룰러 기반 솔버

셀룰러 기반 솔버를 사용하면 셀룰러 라디오 타워에서 얻은 측정 데이터를 통해 위치를 확인할 수 있습니다. 이 솔버는 다음 기술을 지원합니다. 이러한 기술 중 일부 또는 전부의 측정 데이터를 포함하더라도 하나의 확인된 위치 정보를 얻을 수 있습니다.

- GSM
- CDMA
- WCDMA
- TD-SCDMA
- LTE

셀룰러 기반 솔버 페이로드 예시

다음 코드는 셀룰러 측정 데이터가 포함된 디바이스의 JSON 페이로드에 대한 예를 보여줍니다. AWS IoT Core Device Location은이 데이터를 입력으로 수신하면 솔버 공급자에게 HTTP 요청을 보내 위치 정보를 확인합니다. 정보를 검색하려면 콘솔에서 이 형식을 사용하여 JSON 페이로드를 제공하거나 GetPositionEstimate API 작업의 CellTowers 파라미터 값을 지정합니다. 이러한 셀룰러 기술 중 일부 또는 전부를 사용해 파라미터 값을 지정하여 측정 데이터를 제공할 수 있습니다.

LTE(Long-Term Evolution)

이 측정 데이터를 사용할 때는 모바일 네트워크의 네트워크 및 국가 코드, 선택적 추가 파라미터(로컬 ID 관련 정보 등)와 같은 정보를 지정해야 합니다. 다음 코드는 페이로드 형식의 예를 보여줍니다. 이러한 파라미터에 대한 자세한 내용은 LTE 객체를 참조하세요.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Lte": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "EutranCid": int,             // required. Make sure that you use int for
 EutranCid.
          "Tac": int,                   // optional
          "LteLocalId": {               // optional
```

```
            "Pci": int,               // required
            "Earfcn": int,            // required
        },
        "LteTimingAdvance": int,      // optional
        "Rsrp": int,                  // optional
        "Rsrq": float,                // optional
        "NrCapable": boolean,         // optional
        "LteNmr": [                   // optional
              {
                  "Pci": int,         // required
                  "Earfcn": int,      // required
                  "EutranCid": int,   // required
                  "Rsrp": int,        // optional
                  "Rsrq": float       // optional
              }
          ]
      }
    ]
  }
}
```

GSM(Global System for Mobile Communications)

이 측정 데이터를 사용할 때는 모바일 네트워크의 네트워크 및 국가 코드, 기지국 정보, 선택적 추가 파라미터와 같은 정보를 지정해야 합니다. 다음 코드는 페이로드 형식의 예를 보여줍니다. 이러한 파라미터에 대한 자세한 내용은 GSM 객체를 참조하세요.

```
{
    "Timestamp": 1664313161,          // optional
    "CellTowers": {
        "Gsm": [
        {
          "Mcc": int,                 // required
          "Mnc": int,                 // required
          "Lac": int,                 // required
          "GeranCid": int,            // required
          "GsmLocalId": {             // optional
              "Bsic": int,            // required
              "Bcch": int,            // required
          },
          "GsmTimingAdvance": int,    // optional
          "RxLevel": int,             // optional
          "GsmNmr": [                 // optional
```

```
            {
                "Bsic": int,            // required
                "Bcch": int,            // required
                "RxLevel": int,         // optional
                "GlobalIdentity": {
                    "Lac": int,         // required
                    "GeranCid": int     // required
                }
             }
          ]
       }
    ]
}
```

CDMA(Code-Division Multiple Access)

이 측정 데이터를 사용할 때는 신호 강도 및 식별 정보, 기지국 정보, 선택적 추가 파라미터와 같은 정보를 지정해야 합니다. 다음 코드는 페이로드 형식의 예를 보여줍니다. 이러한 파라미터에 대한 자세한 내용은 CDMA 객체를 참조하세요.

```
{
    "Timestamp": 1664313161,              // optional
    "CellTowers": {
        "Cdma": [
        {
            "SystemId": int,               // required
            "NetworkId": int,              // required
            "BaseStationId": int,          // required
            "RegistrationZone": int,       // optional
            "CdmaLocalId": {               // optional
              "PnOffset": int,             // required
              "CdmaChannel": int,          // required
            },
            "PilotPower": int,             // optional
            "BaseLat": float,              // optional
            "BaseLng": float,              // optional
            "CdmaNmr": [                   // optional
                {
                    "PnOffset": int,       // required
                    "CdmaChannel": int,    // required
                    "PilotPower": int,     // optional
                    "BaseStationId": int   // optional
                }
```

```
            ]
        }
    ]
  }
}
```

WCDMA(Wideband Code-Division Multiple Access)

이 측정 데이터를 사용할 때는 네트워크 및 국가 코드, 신호 강도 및 식별 정보, 기지국 정보, 선택적 추가 파라미터와 같은 정보를 지정해야 합니다. 다음 코드는 페이로드 형식의 예를 보여줍니다. 이러한 파라미터에 대한 자세한 내용은 CDMA 객체를 참조하세요.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Wcdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
          "WcdmaLocalId": {             // optional
              "Uarfcndl": int,          // required
              "Psc": int,               // required
          },
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "WcdmaNmr": [                 // optional
                {
                  "Uarfcndl": int,      // required
                  "Psc": int,           // required
                  "UtranCid": int,      // required
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
            ]
        }
    ]
  }
}
```

TD-SCDMA(Time Division Synchronous Code-Division Multiple Access)

이 측정 데이터를 사용할 때는 네트워크 및 국가 코드, 신호 강도 및 식별 정보, 기지국 정보, 선택적 추가 파라미터와 같은 정보를 지정해야 합니다. 다음 코드는 페이로드 형식의 예를 보여줍니다. 이러한 파라미터에 대한 자세한 내용은 CDMA 객체를 참조하세요.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Tdscdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
          "TdscdmaLocalId": {           // optional
              "Uarfcn": int,            // required
              "CellParams": int,        // required
          },
          "TdscdmaTimingAdvance": int,  // optional
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "TdscdmaNmr": [               // optional
                {
                  "Uarfcn": int,        // required
                  "CellParams": int,    // required
                  "UtranCid": int,      // optional
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
            ]
         }
      ]
   }
}
```

IP 역방향 조회 솔버

IP 역방향 조회 솔버를 통해 IP 주소를 입력으로 사용하여 위치를 확인할 수 있습니다. 솔버는 프로비저닝된 디바이스에서 위치 정보를 가져올 수 있습니다 AWS IoT. IPv4 또는 IPv6 표준 패턴이나 IPv6 16진수 압축 패턴의 형식을 사용하여 IP 주소 정보를 지정합니다. 그런 다음 디바이스가 위치한 도시 및 국가와 같은 추가 정보를 포함하여 확인된 위치 추정치를 얻습니다.

> **Note**
>
> IP 역방향 조회를 사용할 경우 특정 집 주소 또는 도로명 주소를 식별하거나 찾기 위한 목적으로 이를 사용하지 않는다는 데 동의하는 것으로 간주됩니다.

IP 역방향 조회 솔버 페이로드 예시

다음 코드는 측정 데이터가 포함된 디바이스의 JSON 페이로드에 대한 예를 보여줍니다. AWS IoT Core Device Location은 측정 데이터에서 IP 주소 정보를 수신하면 솔버 공급자의 데이터베이스에서이 정보를 조회한 다음 위치 정보를 확인하는 데 사용됩니다. 정보를 검색하려면 콘솔에서 이 형식을 사용하여 JSON 페이로드를 제공하거나 GetPositionEstimate API 작업의 Ip 파라미터 값을 지정합니다.

> **Note**
>
> 이 해석기를 사용하면 디바이스가 위치한 도시, 주, 국가 및 우편번호가 좌표와 함께 보고됩니다. 예제는 디바이스 위치 확인(콘솔) 섹션을 참조하세요.

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress":"54.240.198.35"
      }
}
```

GNSS 솔버

GNSS 스캔 결과 메시지 또는 NAV 메시지에 포함된 정보를 사용하여 디바이스 위치를 검색하려면 GNSS(Global Navigation Satellite System) 솔버를 사용합니다. 선택적으로 추가 GNSS 지원 정보를 제공하여 솔버가 신호를 검색하는 데 사용해야 하는 변수 수를 줄일 수 있습니다. 위치, 고도, 캡처 시간 및 정확도 정보가 포함된 이 지원 정보를 제공하면 솔버가 시야 내 위성을 쉽게 식별하고 디바이스 위치를 계산할 수 있습니다.

이 솔버는 LoRaWAN 디바이스 및 Amazon Sidewalk 지원 디바이스뿐만 아니라 프로비저닝된 AWS IoT 디바이스로 프로비저닝된 다른 디바이스에도 사용할 수 있습니다 AWS IoT. 일반 IoT 디바이스의 경우, 디바이스가 GNSS를 이용한 위치 추정을 지원하면 해당 디바이스에서 GNSS 스캔 정보가 수신될 때 송수신장치가 위치 정보를 확인합니다. LoRaWAN 디바이스의 경우 디바이스에 LoRa Edge

칩셋이 있어야 합니다. 디바이스에서 업링크 메시지를 수신하면 GNSS 스캔 데이터가 AWS IoT for LoRaWAN 또는 for Amazon Sidewalk AWS IoT 로 전송되고 위치는 트랜시버의 스캔 결과를 기반으로 추정됩니다.

GNSS 솔버 페이로드 예시

다음 코드는 측정 데이터가 포함된 디바이스의 JSON 페이로드에 대한 예를 보여줍니다. AWS IoT Core Device Location은 측정 데이터에 페이로드가 포함된 GNSS 스캔 정보를 수신하면 트랜시버와 포함된 추가 지원 정보를 사용하여 신호를 검색하고 위치 정보를 확인합니다. 정보를 검색하려면 콘솔에서 이 형식을 사용하여 JSON 페이로드를 제공하거나 GetPositionEstimate API 작업의 Gnss 파라미터 값을 지정합니다.

> **Note**
>
> AWS IoT Core 디바이스 위치가 디바이스 위치를 확인하려면 먼저 페이로드에서 대상 바이트를 제거해야 합니다.

```
{
    "Timestamp": 1664313161,                    // optional
    "Gnss": {
        "AssistAltitude": number,               // optional
        "AssistPosition": [ number ],           // optional
        "CaptureTime": number,                  // optional
        "CaptureTimeAccuracy": number,          // optional
        "Payload": "string",                    // required
        "Use2DSolver": boolean                  // optional
   }
}
```

Amazon Sidewalk 지원 디바이스용 BLE 솔버

> **Note**
>
> Bluetooth Low Energy 기반 위치의 경우 Amazon Sidewalk에 연결되어 있고 커뮤니티 조사 결과 기능이 활성화된 근처 Sidewalk 게이트웨이의 대략적인 위치를 기반으로 위치 좌표를 AWS IoT 반환합니다. 게이트웨이 위치 데이터는 AWS 콘텐츠이며 Amazon Sidewalk에 연결된 디바이스를 찾는 데 도움을 주기 위한 목적으로만 제공되며 해당 용도로만 데이터를 사용해야 합니다. 일반적으로 제공되는 인터페이스 및 기능을 통해서만 위치 데이터를 사용하고 액세스해야

하며, 당사가 제공하는 게이트웨이 위치 데이터를 재식별, 리버스 엔지니어링 또는 다시 매핑하려고 시도해서는 안 됩니다.

를 통해 연결하는 Amazon Sidewalk 엔드 디바이스는 Amazon Sidewalk 엔드 디바이스에서 게시한 BLE, Wi-Fi 또는 GNSS 업링크 메시지를 사용하여 위치 데이터를 확인할 AWS IoT Core for Amazon Sidewalk 수 있습니다. 자세한 내용은 AWS IoT Core for Amazon Sidewalk 개발자 안내서를 참조하세요.

이벤트 메시지

이 섹션에는 사물 또는 작업이 업데이트되거나 변경될 AWS IoT 때에서 게시한 메시지에 대한 정보가 포함되어 있습니다. 디바이스의 장애 또는 작동 변경을 모니터링하고 디바이스가 발생할 때 작업을 트리거할 수 있는 감지기를 생성할 수 있는 AWS IoT Events 서비스에 대한 자세한 내용은 섹션을 참조하세요AWS IoT Events.

이벤트 메시지 생성 방법

AWS IoT 는 특정 이벤트가 발생할 때 이벤트 메시지를 게시합니다. 예를 들어 이벤트는 사물이 추가되거나, 업데이트되거나, 삭제될 때 레지스트리에 의해 생성됩니다. 이러한 이벤트가 발생할 때마다 단일 이벤트 메시지가 전송됩니다. 전송된 이벤트 메시지는 MQTT를 통해 JSON 페이로드와 함께 게시됩니다. 페이로드 내용은 이벤트 유형에 따라 달라집니다.

> **Note**
>
> 이벤트 메시지의 게시는 1회까지 보장됩니다. 하지만 1회 이상 게시하는 것도 가능합니다. 이벤트 메시지의 순서는 보장되지 않습니다.

이벤트 메시지 수신 정책

이벤트 메시지를 수신하려면 디바이스가 AWS IoT 디바이스 게이트웨이에 연결하고 MQTT 이벤트 주제를 구독하도록 허용하는 적절한 정책을 사용해야 합니다. 또한 적합한 주제 필터도 구독해야 합니다.

다음은 수명 주기 이벤트를 수신하는 데 필요한 정책의 예입니다.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:/$aws/events/*"
            ]
        }
    ]
}
```

에 대한 이벤트 활성화 AWS IoT

예약된 주제의 구독자가 메시지를 수신하려면 먼저에서 또는 API AWS Management Console 또는 CLI를 사용하여 이벤트 메시지를 활성화해야 합니다. 다양한 옵션에서 관리하는 이벤트 메시지에 대한 자세한 내용은 AWS IoT 이벤트 구성 설정 테이블을 참조하세요.

- 이벤트 메시지를 활성화하려면 AWS IoT 콘솔의 설정 탭으로 이동한 다음 이벤트 기반 메시지 섹션에서 이벤트 관리를 선택합니다. 관리할 이벤트를 지정할 수 있습니다.
- API 또는 CLI를 사용하여 게시할 이벤트 유형을 제어하려면 UpdateEventConfigurations API를 호출하거나 update-event-configurations CLI 명령을 사용합니다. 예:

```
aws iot update-event-configurations --event-configurations "{\"THING\":{\"Enabled\":
 true}}"
```

> **Note**
>
> 모든 큰따옴표(")는 백슬래시(\)로 이스케이프됩니다.

DescribeEventConfigurations API를 호출하거나 describe-event-configurations CLI 명령을 사용하여 현재 이벤트 구성을 가져올 수 있습니다. 예:.

```
aws iot describe-event-configurations
```

AWS IoT 이벤트 구성 설정 표

이벤트 범주 (AWS IoT 콘솔: 설정: 이벤트 기반 메시지)	**eventConfigurations** 키 값 (AWS CLI/API)	이벤트 메시지 주제
(AWS CLI/API를 사용해서만 구성할 수 있음)	`CA_CERTIFICATE`	`$aws/events/certificates/registered/` *caCertificateId*
(AWS CLI/API를 사용해서만 구성할 수 있음)	`CERTIFICATE`	`$aws/events/presence/connected/` *clientId*
(AWS CLI/API를 사용해서만 구성할 수 있음)	`CERTIFICATE`	`$aws/events/presence/disconnected/` *clientId*
(AWS CLI/API를 사용해서만 구성할 수 있음)	`CERTIFICATE`	`$aws/events/subscriptions/subscribed/` *clientId*
(AWS CLI/API를 사용해서만 구성할 수 있음)	`CERTIFICATE`	`$aws/events/subscriptions/unsubscribed/` *clientId*
작업 완료됨, 취소됨	`JOB`	`$aws/events/job/`*jobID*`/canceled`
작업 완료됨, 취소됨	`JOB`	`$aws/events/job/`*jobID*`/cancellation_in_progress`
작업 완료됨, 취소됨	`JOB`	`$aws/events/job/`*jobID*`/completed`
작업 완료됨, 취소됨	`JOB`	`$aws/events/job/`*jobID*`/deleted`

이벤트 범주 (AWS IoT 콘솔: 설정: 이벤트 기반 메시지)	**eventConfigurations** 키 값 (AWS CLI/API)	이벤트 메시지 주제
작업 완료됨, 취소됨	JOB	$aws/events/job/*jobID*/deletion_in_progress
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/canceled
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/deleted
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/failed
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/rejected
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/removed
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/succeeded
작업 실행: 성공, 실패, 거부됨, 취소됨, 제거됨	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/timed_out
사물: 생성됨, 업데이트됨, 삭제됨	THING	$aws/events/thing/ *thingName* /created
사물: 생성됨, 업데이트됨, 삭제됨	THING	$aws/events/thing/ *thingName* /updated

이벤트 범주 (AWS IoT 콘솔: 설정: 이벤트 기반 메시지)	**eventConfigurations** 키 값 (AWS CLI/API)	이벤트 메시지 주제
사물: 생성됨, 업데이트됨, 삭제됨	THING	$aws/events/thing/ *thingName* /deleted
사물 그룹: 추가됨, 제거됨	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /created
사물 그룹: 추가됨, 제거됨	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /updated
사물 그룹: 추가됨, 제거됨	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /deleted
사물 그룹 계층: 추가됨, 제거됨	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /added
사물 그룹 계층: 추가됨, 제거됨	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /removed

이벤트 범주 (AWS IoT 콘솔: 설정: 이벤트 기반 메시지)	**eventConfigurations** 키 값 (AWS CLI/API)	이벤트 메시지 주제
사물 그룹 멤버십: 추가됨, 제거됨	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /added
사물 그룹 멤버십: 추가됨, 제거됨	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /removed
사물 유형: 생성됨, 업데이트됨, 삭제됨	THING_TYPE	$aws/events/thingType/ *thingTypeName* /created
사물 유형: 생성됨, 업데이트됨, 삭제됨	THING_TYPE	$aws/events/thingType/ *thingTypeName* /updated
사물 유형: 생성됨, 업데이트됨, 삭제됨	THING_TYPE	$aws/events/thingType/ *thingTypeName* /deleted

이벤트 범주 (AWS IoT 콘솔: 설정: 이벤트 기반 메시지)	**eventConfigurations** 키 값 (AWS CLI/API)	이벤트 메시지 주제
사물 유형 연결: 추가됨, 제거됨	THING_TYPE_ASSOCIATION	$aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /added $aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /removed

레지스트리 이벤트

레지스트리는 사물, 사물 유형 및 사물 그룹이 생성되거나, 업데이트되거나, 삭제될 때 이벤트 메시지를 게시할 수 있습니다. 그러나 이러한 이벤트는 기본적으로 사용할 수 없습니다. 이러한 이벤트를 설정하는 방법에 대한 자세한 내용은 에 대한 이벤트 활성화 AWS IoT 섹션을 참조하세요.

레지스트리에서 제공하는 이벤트 유형은 다음과 같습니다.

- 사물 이벤트
- 사물 유형 이벤트
- 사물 그룹 이벤트

사물 이벤트

사물 생성/업데이트/삭제됨

레지스트리는 사물이 생성, 업데이트 또는 삭제될 때 다음과 같은 이벤트 메시지를 게시합니다.

- $aws/events/thing/*thingName*/created
- $aws/events/thing/*thingName*/updated

- $aws/events/thing/*thingName*/deleted

메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
    "eventType" : "THING_EVENT",
    "eventId" : "f5ae9b94-8b8e-4d8e-8c8f-b3266dd89853",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName" : "MyThing",
    "versionNumber" : 1,
    "thingTypeName" : null,
    "attributes": {
                "attribute3": "value3",
                "attribute1": "value1",
                "attribute2": "value2"
    }
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

eventType

"THING_EVENT"로 설정합니다.

eventId

고유한 이벤트 ID(문자열)입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

작업

이벤트를 트리거한 작업입니다. 유효값은 다음과 같습니다.

- CREATED
- UPDATED
- DELETED

accountId

AWS 계정 ID.

thingId

생성되거나, 업데이트되거나, 삭제된 사물의 ID입니다.

thingName

생성되거나, 업데이트되거나, 삭제된 사물의 이름입니다.

versionNumber

생성되거나, 업데이트되거나, 삭제된 사물의 버전입니다. 사물이 생성될 때는 이 값이 1로 설정됩니다. 이후 사물이 업데이트될 때마다 1씩 증가합니다.

thingTypeName

사물과 연결된 사물(있는 경우)의 유형입니다. 그렇지 않을 경우 `null`입니다.

attributes

사물과 연결된 이름-값 페어의 모음입니다.

사물 유형 이벤트

사물 유형 관련 이벤트:

- 사물 유형 생성/업데이트/사용 중지/사용 중지 해제/삭제
- 사물 유형을 사물과 연결/연결 해제

사물 유형 생성/업데이트/사용 중지/사용 중지 해제/삭제

레지스트리는 사물 유형이 생성, 업데이트, 사용 중지, 사용 중지 해제 또는 삭제될 때 다음과 같은 이벤트 메시지를 게시합니다.

- `$aws/events/thingType/`*`thingTypeName`*`/created`
- `$aws/events/thingType/`*`thingTypeName`*`/updated`
- `$aws/events/thingType/`*`thingTypeName`*`/deleted`

메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
    "eventType" : "THING_TYPE_EVENT",
    "eventId" : "8827376c-4b05-49a3-9b3b-733729df7ed5",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingTypeId" : "c530ae83-32aa-4592-94d3-da29879d1aac",
    "thingTypeName" : "MyThingType",
    "isDeprecated" : false|true,
    "deprecationDate" : null,
    "searchableAttributes" : [ "attribute1", "attribute2", "attribute3" ],
    "propagatingAttributes": [
        {
     "userPropertyKey": "key",
     "thingAttribute": "model"
        },
        {
     "userPropertyKey": "key",
     "connectionAttribute": "iot:ClientId"
        }
    ],
    "description" : "My thing type"
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

eventType

"THING_TYPE_EVENT"로 설정합니다.

eventId

고유한 이벤트 ID(문자열)입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

작업

이벤트를 트리거한 작업입니다. 유효값은 다음과 같습니다.

- CREATED
- UPDATED
- DELETED

accountId

AWS 계정 ID.

thingTypeId

생성되거나, 업데이트되거나, 사용 중지되거나, 삭제된 사물 유형의 ID입니다.

thingTypeName

생성되거나, 업데이트되거나, 사용 중지되거나, 삭제된 사물 유형의 이름입니다.

isDeprecated

사물 유형의 사용이 중단된 경우에는 `true`입니다. 그렇지 않을 경우 `false`입니다.

deprecationDate

사물 유형의 사용이 중단된 시점의 UNIX 타임스탬프입니다.

searchableAttributes

검색에 사용할 수 있는 사물 유형과 연결된 이름-값 페어의 모음입니다.

propagatingAttributes

전파 속성 목록입니다. 전파 속성에는 사물 속성, 연결 속성 및 사용자 속성 키가 포함될 수 있습니다. 자세한 내용은 메시지 보강을 위한 전파 속성 추가를 참조하세요.

description

사물 유형에 대한 설명입니다.

사물 유형을 사물과 연결/연결 해제

레지스트리는 사물 유형이 사물과 연결 또는 연결 해제될 때 다음과 같은 이벤트 메시지를 게시합니다.

- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/thingType/`*`typeName`*`/added`
- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/thingType/`*`typeName`*`/removed`

다음은 `added` 페이로드의 예입니다. `removed` 메시지에 대한 페이로드는 유사합니다.

```
{
```

```
    "eventId" : "87f8e095-531c-47b3-aab5-5171364d138d",
    "eventType" : "THING_TYPE_ASSOCIATION_EVENT",
    "operation" : "ADDED",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName": "myThing",
    "thingTypeName" : "MyThingType",
    "timestamp" : 1234567890123,
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

eventId

고유한 이벤트 ID(문자열)입니다.

eventType

"THING_TYPE_ASSOCIATION_EVENT"로 설정합니다.

작업

이벤트를 트리거한 작업입니다. 유효값은 다음과 같습니다.

- ADDED
- REMOVED

thingId

유형 연결이 변경된 사물의 ID입니다.

thingName

유형 연결이 변경된 사물의 이름입니다.

thingTypeName

사물과 연결되었거나 더 이상 연결되지 않은 사물 유형입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

사물 그룹 이벤트

사물 그룹 관련 이벤트:

- 사물 그룹 생성/업데이트/삭제
- 사물 그룹에(서) 사물 추가/제거
- 사물 그룹에(서) 사물 그룹 추가/삭제

사물 그룹 생성/업데이트/삭제

레지스트리는 사물 그룹이 생성, 업데이트 또는 삭제될 때 다음과 같은 이벤트 메시지를 게시합니다.

- `$aws/events/thingGroup/`*`groupName`*`/created`
- `$aws/events/thingGroup/`*`groupName`*`/updated`
- `$aws/events/thingGroup/`*`groupName`*`/deleted`

다음은 `updated` 페이로드의 예입니다. `created` 및 `deleted` 메시지에 대한 페이로드는 유사합니다.

```
{
  "eventType": "THING_GROUP_EVENT",
  "eventId": "8b9ea8626aeaa1e42100f3f32b975899",
  "timestamp": 1603995417409,
  "operation": "UPDATED",
  "accountId": "571EXAMPLE833",
  "thingGroupId": "8757eec8-bb37-4cca-a6fa-403b003d139f",
  "thingGroupName": "Tg_level5",
  "versionNumber": 3,
  "parentGroupName": "Tg_level4",
  "parentGroupId": "5fce366a-7875-4c0e-870b-79d8d1dce119",
  "description": "New description for Tg_level5",
  "rootToParentThingGroups": [
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/TgTopLevel",
      "groupId": "36aa0482-f80d-4e13-9bff-1c0a75c055f6"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level1",
      "groupId": "bc1643e1-5a85-4eac-b45a-92509cbe2a77"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level2",
      "groupId": "0476f3d2-9beb-48bb-ae2c-ea8bd6458158"
    },
    {
```

```
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level3",
      "groupId": "1d9d4ffe-a6b0-48d6-9de6-2e54d1eae78f"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level4",
      "groupId": "5fce366a-7875-4c0e-870b-79d8d1dce119"
    }
  ],
  "attributes": {
    "attribute1": "value1",
    "attribute3": "value3",
    "attribute2": "value2"
  },
  "dynamicGroupMappingId": null
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

eventType

"THING_GROUP_EVENT"로 설정합니다.

eventId

고유한 이벤트 ID(문자열)입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

작업

이벤트를 트리거한 작업입니다. 유효값은 다음과 같습니다.

- CREATED
- UPDATED
- DELETED

accountId

AWS 계정 ID.

thingGroupId

생성되거나, 업데이트되거나, 삭제된 사물 그룹의 ID입니다.

thingGroupName

생성되거나, 업데이트되거나, 삭제된 사물 그룹의 이름입니다.

versionNumber

사물 그룹 버전입니다. 사물 그룹이 생성될 때는 이 값이 1로 설정됩니다. 이후 사물 그룹이 업데이트될 때마다 1씩 증가합니다.

parentGroupName

상위 사물 그룹(있는 경우)의 이름입니다.

parentGroupId

상위 사물 그룹(있는 경우)의 ID입니다.

description

사물 그룹에 대한 설명입니다.

rootToParentThingGroups

상위 사물 그룹에 대한 정보의 배열입니다. 각 상위 사물 그룹에 대한 요소가 한 가지 있으며, 이는 루트 사물 그룹에서 시작하여 사물 그룹의 상위까지 계속됩니다. 각 항목에는 사물 그룹의 `groupArn` 및 `groupId`이(가) 포함됩니다.

attributes

사물 그룹과 연결된 이름-값 페어의 모음입니다.

사물 그룹에(서) 사물 추가/제거

레지스트리는 사물이 사물 그룹에 추가되거나 사물 그룹에서 제거될 때 다음과 같은 이벤트 메시지를 게시합니다.

- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/added`
- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/removed`

메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
```

```
    "eventType" : "THING_GROUP_MEMBERSHIP_EVENT",
    "eventId" : "d684bd5f-6f6e-48e1-950c-766ac7f02fd1",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "groupArn" : "arn:aws:iot:ap-northeast-2:123456789012:thinggroup/
MyChildThingGroup",
    "groupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "thingArn" : "arn:aws:iot:ap-northeast-2:123456789012:thing/MyThing",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "membershipId" : "8505ebf8-4d32-4286-80e9-c23a4a16bbd8"
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

eventType

"THING_GROUP_MEMBERSHIP_EVENT"로 설정합니다.

eventId

이벤트 ID입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

작업

사물이 사물 그룹에 추가된 경우에는 ADDED을(를) 게시합니다. 사물이 사물 그룹에서 제거된 경우에는 REMOVED을(를) 게시합니다.

accountId

AWS 계정 ID.

groupArn

사물 그룹의 ARN입니다.

groupId

그룹의 ID입니다.

thingArn

사물 그룹에 추가되거나, 혹은 사물 그룹에서 제거된 사물의 ARN입니다.

thingId

사물 그룹에 추가되거나, 혹은 사물 그룹에서 제거된 사물의 ID입니다.

membershipId

사물과 사물 그룹의 관계를 나타내는 ID입니다. 이 값은 사물을 사물 그룹에 추가할 때 생성됩니다.

사물 그룹에(서) 사물 그룹 추가/삭제

레지스트리는 사물 그룹이 다른 사물 그룹에 추가되거나 사물 그룹에서 제거될 때 다음과 같은 이벤트 메시지를 게시합니다.

- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/added`
- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/removed`

메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
    "eventType" : "THING_GROUP_HIERARCHY_EVENT",
    "eventId" : "264192c7-b573-46ef-ab7b-489fcd47da41",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "thingGroupId" : "8f82a106-6b1d-4331-8984-a84db5f6f8cb",
    "thingGroupName" : "MyRootThingGroup",
    "childGroupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "childGroupName" : "MyChildThingGroup"
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

eventType

"THING_GROUP_HIERARCHY_EVENT"로 설정합니다.

eventId

이벤트 ID입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

작업

사물이 사물 그룹에 추가된 경우에는 ADDED을(를) 게시합니다. 사물이 사물 그룹에서 제거된 경우에는 REMOVED을(를) 게시합니다.

accountId

AWS 계정 ID.

thingGroupId

상위 사물 그룹의 ID입니다.

thingGroupName

상위 사물 그룹의 이름입니다.

childGroupId

하위 사물 그룹의 ID입니다.

childGroupName

하위 사물 그룹의 이름입니다.

작업 이벤트

AWS IoT 작업 서비스는 작업이 보류, 완료 또는 취소될 때와 디바이스가 작업 실행 시 성공 또는 실패를 보고할 때 MQTT 프로토콜의 예약된 주제에 게시합니다. 디바이스 또는 관리 및 모니터링 애플리케이션은 이러한 주제를 구독하여 작업 상태를 추적할 수 있습니다.

작업 이벤트 사용 방법

AWS IoT 작업 서비스의 응답 메시지는 메시지 브로커를 통과하지 않으며 다른 클라이언트 또는 규칙에서 구독할 수 없습니다. 작업 활동 관련 메시지를 구독하려면 notify 및 notify-next 주제를 사용합니다. 작업 주제에 대한 자세한 내용은 작업 주제 섹션을 참조하세요.

작업 업데이트에 대한 알림을 받으려면를 사용하거나 API AWS Management Console또는 CLI를 사용하여 이러한 작업 이벤트를 활성화합니다. 자세한 내용은 에 대한 이벤트 활성화 AWS IoT 단원을 참조하십시오.

작업 이벤트 작동 방식

작업 취소 및 삭제에는 다소 시간이 걸릴 수 있기 때문에 요청의 시작과 종료를 알리기 위해 2 개의 메시지가 전송됩니다. 예를 들어, 취소 요청이 시작되면 `$aws/events/job/jobID/cancellation_in_progress` 주제로 메시지가 전송됩니다. 취소 요청이 완료되면 `$aws/events/job/jobID/canceled` 주제로 메시지가 전송됩니다.

작업 삭제 요청에 대해서도 비슷한 프로세스가 수행됩니다. 관리 및 모니터링 애플리케이션은 이러한 주제를 구독하여 작업 상태를 추적할 수 있습니다. MQTT 주제 게시 및 구독에 대한 자세한 내용은 the section called “디바이스 통신 프로토콜” 단원을 참조하세요.

작업 이벤트 유형

다음 예제는 다양한 작업 이벤트 유형을 보여줍니다.

작업 완료/취소/삭제됨

AWS IoT 작업 서비스는 작업이 완료, 취소, 삭제되거나 취소 또는 삭제가 진행 중일 때 MQTT 주제에 메시지를 게시합니다.

- `$aws/events/job/jobID/completed`
- `$aws/events/job/jobID/canceled`
- `$aws/events/job/jobID/deleted`
- `$aws/events/job/jobID/cancellation_in_progress`
- `$aws/events/job/jobID/deletion_in_progress`

`completed` 메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
  "eventType": "JOB",
  "eventId": "7364ffd1-8b65-4824-85d5-6c14686c97c6",
  "timestamp": 1234567890,
  "operation": "completed",
  "jobId": "27450507-bf6f-4012-92af-bb8a1c8c4484",
  "status": "COMPLETED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/a39f6f91-70cf-4bd2-a381-9c66df1a80d0",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/2fc4c0a4-6e45-4525-
a238-0fe8d3dd21bb"
  ],
  "description": "My Job Description",
```

```
  "completedAt": 1234567890123,
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123,
  "jobProcessDetails": {
    "numberOfCanceledThings": 0,
    "numberOfRejectedThings": 0,
    "numberOfFailedThings": 0,
    "numberOfRemovedThings": 0,
    "numberOfSucceededThings": 3
  }
}
```

canceled 메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
  "eventType": "JOB",
  "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
  "timestamp": 1234567890,
  "operation": "canceled",
  "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
  "status": "CANCELED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
  ],
  "description": "My job description",
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123
}
```

deleted 메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deleted",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETED",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
```

```
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

cancellation_in_progress 메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "cancellation_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "CANCELLATION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

deletion_in_progress 메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deletion_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETION_IN_PROGRESS",
```

```
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

작업 실행 종료 상태

AWS IoT 디바이스가 작업 실행을 터미널 상태로 업데이트하면 작업 서비스가 메시지를 게시합니다.

- $aws/events/jobExecution/*jobID*/succeeded
- $aws/events/jobExecution/*jobID*/failed
- $aws/events/jobExecution/*jobID*/rejected
- $aws/events/jobExecution/*jobID*/canceled
- $aws/events/jobExecution/*jobID*/timed_out
- $aws/events/jobExecution/*jobID*/removed
- $aws/events/jobExecution/*jobID*/deleted

메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
  "eventType": "JOB_EXECUTION",
  "eventId": "cca89fa5-8a7f-4ced-8c20-5e653afb3572",
  "timestamp": 1234567890,
  "operation": "succeeded|failed|rejected|canceled|removed|timed_out",
  "jobId": "154b39e5-60b0-48a4-9b73-f6f8dd032d27",
  "thingArn": "arn:aws:iot:us-east-1:123456789012:myThing/6d639fbc-8f85-4a90-924d-
a2867f8366a7",
  "status": "SUCCEEDED|FAILED|REJECTED|CANCELED|REMOVED|TIMED_OUT",
  "statusDetails": {
    "key": "value"
  }
}
```

명령 실행 이벤트

AWS IoT 는 명령 실행 상태가 변경될 때 이벤트 메시지를 MQTT 주제에 게시합니다. 이러한 이벤트를 사용하여 명령 실행 진행 상황을 모니터링하고 상태 변경에 응답하는 애플리케이션을 빌드할 수 있습니다.

명령 실행 이벤트 주제

명령 실행 이벤트는 다음 MQTT 주제에 게시됩니다.

```
$aws/events/commandExecution/commandId/status
```

위치:

- *commandId*는 명령의 식별자입니다.
- *status*는 명령 실행의 상태입니다. 유효한 값은 CREATED, IN_PROGRESS, SUCCEEDED, FAILED, REJECTED, TIMED_OUT입니다.

모든 명령과 모든 상태에 대한 알림을 받으려면 와일드카드 문자를 사용하여 다음 주제를 구독합니다.

```
$aws/events/commandExecution/+/#
```

명령 실행 이벤트 메시지

명령 실행 상태가 변경되면는 해당 MQTT 주제에 이벤트 메시지를 AWS IoT 게시합니다. 메시지에는 아래와 같은 페이로드 예제가 포함됩니다.

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "status": "FAILED",
    "statusReason": {
        "reasonCode": "DEVICE_TOO_BUSY",
        "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
    "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp": 1717708862107
```

```
}
```

페이로드에는 다음과 같은 속성이 포함됩니다.

executionId

명령 실행의 고유 식별자(문자열).

상태

명령 실행의 상태입니다. 유효한 값은 `CREATED`, `IN_PROGRESS`, `SUCCEEDED`, `FAILED`, `REJECTED`, `TIMED_OUT`입니다.

statusReason

사용 가능한 경우 상태에 대한 추가 정보가 포함된 객체입니다. `reasonCode` 및 `reasonDescription` 필드를 포함합니다.

eventType

"COMMAND_EXECUTION"으로 설정합니다.

commandArn

명령의 Amazon 리소스 이름(ARN)입니다.

targetArn

명령 실행을 위한 대상 디바이스(사물 또는 클라이언트)의 ARN입니다.

timestamp

이벤트가 발생한 시점의 UNIX 타임스탬프입니다.

수명 주기 이벤트

AWS IoT 는 MQTT 주제에 수명 주기 이벤트를 게시할 수 있습니다. 이러한 이벤트는 기본적으로 사용할 수 있으며 비활성화할 수 없습니다.

> **Note**
>
> 수명 주기 메시지는 틀린 순서로 전송될 수 있습니다. 중복 메시지를 수신할 수도 있습니다. `thingName`은 클라이언트가 독점 사물 기능을 사용하여 연결하는 경우에만 포함됩니다.

이 주제에서 수행할 작업

- 연결/연결 해제 이벤트
- 연결 시도 실패 이벤트
- 구독/구독 취소 이벤트

연결/연결 해제 이벤트

> **Note**
>
> AWS IoT Device Management 플릿 인덱싱을 사용하면 사물을 검색하고, 집계 쿼리를 실행하고, 사물 연결/연결 해제 이벤트를 기반으로 동적 그룹을 생성할 수 있습니다. 자세한 내용은 플릿 인덱싱을 참조하세요.

AWS IoT 는 클라이언트가 연결 또는 연결 해제될 때 다음 MQTT 주제에 메시지를 게시합니다.

- `$aws/events/presence/connected/`*`clientId`* – 클라이언트가 메시지 브로커에 연결됩니다.
- `$aws/events/presence/disconnected/`*`clientId`* – 클라이언트가 메시지 브로커에서 연결 해제됩니다.

다음은 `$aws/events/presence/connected/`*`clientId`* 주제에 게시되는 연결/연결 해제 메시지에 포함되는 JSON 요소의 목록입니다.

clientId

연결 또는 연결 해제하는 클라이언트의 클라이언트 ID입니다.

> **Note**
>
> # 또는 +가 포함된 클라이언트 ID는 수명 주기 이벤트를 수신하지 않습니다.

thingName

IoT 사물의 이름입니다. `thingName`은 클라이언트가 독점 사물 기능을 사용하여 연결하는 경우에만 포함됩니다.

clientInitiatedDisconnect

클라이언트가 연결 해제를 시작했으면 True, 그렇지 않으면 false입니다. 연결 해제 메시지에서만 찾을 수 있습니다.

disconnectReason

클라이언트가 연결을 해제하는 이유입니다. 연결 해제 메시지에서만 찾을 수 있습니다. 다음 테이블에는 유효한 값과 연결이 끊겼을 때 브로커가 마지막 유언 및 증거(LWT) 메시지를 보낼지가 나와 있습니다.

연결 해제 이유	설명	브로커가 LWT 메시지를 보냄
AUTH_ERROR	클라이언트가 인증에 실패했거나 권한 부여가 실패했습니다.	예[1]
CLIENT_INITIATED_DISCONNECT	클라이언트는 연결을 해제할 것을 나타냅니다. 클라이언트는 MQTT DISCONNECT 제어 패킷 또는 Close frame(클라이언트가 WebSocket 연결을 사용하는 경우)를 전송하여 이 작업을 수행할 수 있습니다.	아니요
CLIENT_ERROR	클라이언트가 연결 해제를 초래하는 잘못된 동작을 수행했습니다. 예를 들어 동일한 연결에서 둘 이상의 MQTT CONNECT 패킷을 전송하는 경우 또는 클라이언트가 페이로드 제한을 초과하는 페이로드로 게시하려고 시도하는 경우 클라이언트가 연결 해제됩니다.	예
CONNECTION_LOST	클라이언트-서버 연결이 차단되었습니다. 이는 네트워크 지연 시간이 길거나 인터넷 연결이 끊긴 경우에 발생할 수 있습니다.	예
DUPLICATE_CLIENTID	클라이언트가 이미 사용 중인 클라이언트 ID를 사용하고 있습니다. 이 경우 이미 연결된 클라이언트는 이 연결 해제 이유로 연결이 해제됩니다.	예

연결 해제 이유	설명	브로커가 LWT 메시지를 보냄
FORBIDDEN_ACCESS	클라이언트가 연결이 금지되어 있습니다. 예를 들어 IP 주소가 거부된 클라이언트는 연결에 실패합니다.	예[1]
MQTT_KEEP_ALIVE_TIMEOUT	클라이언트의 연결 유지 시간의 1.5배 동안 클라이언트-서버 통신이 없으면 클라이언트의 연결이 해제됩니다.	예
SERVER_ERROR	예기치 않은 서버 문제로 인해 연결이 해제되었습니다.	예
SERVER_INITIATED_DISCONNECT	서버가 운영상 이유로 의도적으로 클라이언트의 연결을 해제합니다.	예
API_INITIATED_DISCONNECT	DeleteConnection API를 사용하여 클라이언트의 연결이 끊어졌습니다.	예[2]
THROTTLED	클라이언트가 조절 제한을 초과하여 연결이 해제되었습니다.	예
WEBSOCKET_TTL_EXPIRATION	WebSocket이 TTL(Time-to-Live) 값보다 오래 연결되었기 때문에 클라이언트의 연결이 해제되었습니다.	예
CUSTOMAUTH_TTL_EXPIRATION	해당 사용자 지정 승인자의 TTL(Time-to-Live) 값보다 오래 연결되었기 때문에 클라이언트의 연결이 해제되었습니다.	예

[1]이 오류가 발생하기 전에 디바이스의 연결이 활성 상태였던 경우

[2]Last Will and Testament(LWT) 메시지를 방지하려면 DeleteConnection API의 기본 LWT 전송 동작을 재정의하도록 preventWillMessage=true를 설정합니다.

eventType

이벤트의 유형입니다. 유효한 값은 `connected` 또는 `disconnected`입니다.

ipAddress

연결 클라이언트의 IP 주소입니다. 이 주소는 IPv4 또는 IPv6 형식일 수 있습니다. 연결 메시지에서만 찾을 수 있습니다.

principalIdentifier

인증에 사용되는 자격 증명입니다. TLS 상호 인증 인증서의 경우 이 항목은 인증서 ID입니다. 다른 연결에서는 IAM 자격 증명입니다.

sessionIdentifier

세션 수명 동안 AWS IoT 존재하는의 전역 고유 식별자입니다.

timestamp

이벤트 발생 시점의 근사치

versionNumber

수명 주기 이벤트의 버전 번호입니다. 이것은 각 클라이언트 ID 연결에 대해 단조 증가하는 긴 정수 값입니다. 버전 번호는 구독자가 수명 주기 이벤트의 순서를 추론하는 데 사용할 수 있습니다.

> **Note**
>
> 클라이언트 연결의 연결 및 연결 해제 메시지는 버전 번호가 동일합니다.
> 버전 번호는 값을 건너뛸 수 있으며 각 이벤트마다 항상 1씩 증가한다는 보장이 없습니다.
> 클라이언트가 약 1시간 동안 연결되지 않으면 버전 번호가 0으로 재설정됩니다. 영구 세션의 경우 클라이언트가 해당 세션에 대해 구성된 TTL(time-to-live)보다 오래 연결 해제되면 버전 번호가 0으로 재설정됩니다.

연결 메시지의 구조는 다음과 같습니다.

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
```

```
    "ipAddress": "192.0.2.0",
    "versionNumber": 0
}
```

연결 해제 메시지의 구조는 다음과 같습니다.

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002340451,
    "eventType": "disconnected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "clientInitiatedDisconnect": true,
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
    "versionNumber": 0
}
```

클라이언트 연결 해제 처리

마지막 유언 및 증거(LWT) 메시지를 포함하는 수명 주기 이벤트에 대해 항상 대기 상태가 구현되도록 하는 것이 모범 사례입니다. 연결 해제 메시지가 수신되면 코드가 조치를 취하기 전에 일정 기간을 기다리고 디바이스가 여전히 오프라인 상태인지 확인해야 합니다. 이를 수행하는 한 가지 방법은 SQS 지연 대기열을 사용하는 것입니다. 클라이언트가 LWT 또는 수명 주기 이벤트를 수신하면, 메시지 내용을 대기열에 넣을 수 있습니다(예: 5초간). 해당 메시지를 사용할 수 있게 되고 (Lambda 또는 다른 서비스에서) 처리하면, 추가 조치를 취하기 전에 디바이스가 여전히 오프라인 상태인지 여부를 먼저 확인할 수 있습니다.

연결 시도 실패 이벤트

AWS IoT 는 클라이언트가 연결할 권한이 없거나 마지막 의지 및 증명이 구성되고 클라이언트가 해당 마지막 의지 주제에 게시할 권한이 없는 경우 다음 MQTT 주제에 메시지를 게시합니다.

```
$aws/events/presence/connect_failed/clientId
```

다음은 $aws/events/presence/connect_failed/*clientId* 주제에 게시되는 연결 인증 메시지에 포함되는 JSON 요소의 목록입니다.

clientId

연결을 시도하여 실패한 클라이언트의 클라이언트 ID입니다.

> **Note**
>
> # 또는 +가 포함된 클라이언트 ID는 수명 주기 이벤트를 수신하지 않습니다.

thingName

IoT 사물의 이름입니다. `thingName`은 클라이언트가 독점 사물 기능을 사용하여 연결하는 경우에만 포함됩니다.

timestamp

이벤트 발생 시점의 근사치

eventType

이벤트의 유형입니다. 유효한 값: `connect_failed`

connectFailureReason

연결이 실패하는 이유입니다. 유효한 값: `AUTHORIZATION_FAILED`

principalIdentifier

인증에 사용되는 자격 증명입니다. TLS 상호 인증 인증서의 경우 이 항목은 인증서 ID입니다. 다른 연결에서는 IAM 자격 증명입니다.

sessionIdentifier

세션 수명 동안 AWS IoT 존재하는의 전역 고유 식별자입니다.

ipAddress

연결 클라이언트의 IP 주소입니다. 이 주소는 IPv4 또는 IPv6 형식일 수 있습니다. 연결 메시지에서만 찾을 수 있습니다.

연결 실패 메시지의 구조는 다음과 같습니다.

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "connect_failed",
    "connectFailureReason": "AUTHORIZATION_FAILED",
    "principalIdentifier": "12345678901234567890123456789012",
```

```
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "ipAddress" : "192.0.2.0"
}
```

구독/구독 취소 이벤트

AWS IoT 는 클라이언트가 MQTT 주제를 구독하거나 구독 취소할 때 다음 MQTT 주제에 메시지를 게시합니다.

```
$aws/events/subscriptions/subscribed/clientId
```

또는

```
$aws/events/subscriptions/unsubscribed/clientId
```

여기서 `clientId`은(는) AWS IoT 메시지 브로커에 연결하는 MQTT 클라이언트 ID입니다.

이 주제에 게시된 메시지는 구조가 다음과 같습니다.

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "subscribed" | "unsubscribed",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "topics" : ["foo/bar","device/data","dog/cat"]
}
```

다음은 `$aws/events/subscriptions/subscribed/clientId` 및 `$aws/events/subscriptions/unsubscribed/clientId` 주제에 게시되는 구독/구독 취소 메시지에 포함되는 JSON 요소의 목록입니다.

clientId

구독 또는 구독 취소하는 클라이언트의 클라이언트 ID입니다.

> **Note**
>
> # 또는 +가 포함된 클라이언트 ID는 수명 주기 이벤트를 수신하지 않습니다.

thingName

IoT 사물의 이름입니다. `thingName`은 클라이언트가 독점 사물 기능을 사용하여 연결하는 경우에만 포함됩니다.

eventType

이벤트의 유형입니다. 유효한 값은 `subscribed` 또는 `unsubscribed`입니다.

principalIdentifier

인증에 사용되는 자격 증명입니다. TLS 상호 인증 인증서의 경우 이 항목은 인증서 ID입니다. 다른 연결에서는 IAM 자격 증명입니다.

sessionIdentifier

세션 수명 동안 AWS IoT 존재하는의 전역 고유 식별자입니다.

timestamp

이벤트 발생 시점의 근사치

topics

클라이언트가 구독한 MQTT 주제의 어레이입니다.

> **Note**
>
> 수명 주기 메시지는 틀린 순서로 전송될 수 있습니다. 중복 메시지를 수신할 수도 있습니다.

문제 해결 AWS IoT

이 주제를 개선하도록 도와주세요.
이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

다음은 AWS IoT에서 일반적으로 발생하는 문제를 해결하는 데 유용한 정보입니다.

작업

- AWS IoT Core 문제 해결 가이드
- AWS IoT Device Management 문제 해결 가이드
- AWS IoT Device Advisor 문제 해결 가이드
- AWS IoT 오류

AWS IoT Core 문제 해결 가이드

이 주제를 개선하도록 도와주세요.
이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

이 섹션은에 대한 문제 해결 섹션입니다 AWS IoT Core.

주제

- 연결 문제 진단
- 진단 규칙 문제
- Shadows 문제 진단
- Salesforce IoT 입력 스트림 작업 문제 진단
- 스트림 제한 진단
- 디바이스 플릿 연결 해제 문제 해결

연결 문제 진단

> 이 주제를 개선하도록 도와주세요.
> 이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

에 성공적으로 연결하려면 다음이 AWS IoT 필요합니다.

- 유효한 연결
- 유효한 활성 인증서
- 원하는 연결 및 작업을 허용하는 정책

Connection

올바른 엔드포인트를 찾으려면 어떻게 해야 합니까?

- `aws iot describe-endpoint --endpoint-type iot:Data-ATS`에서 반환한 `endpointAddress`

  또는

- `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"`에서 반환한 `domainName`

올바른 서버 이름 표시(SNI) 값을 찾으려면 어떻게 합니까?

올바른 SNI 값은 describe-endpoint에서 반환된 `endpointAddress` 또는 describe-domain-configuration 명령에서 반환된 `domainName`입니다. 이전 단계에서 엔드포인트와 동일한 주소입니다. 디바이스를에 연결할 때 AWS IoT Core클라이언트는 서버 이름 표시(SNI) 확장을 전송할 수 있습니다.이 확장은 필수는 아니지만 적극 권장됩니다. 다중 계정 등록, 사용자 지정 도메인, VPC 엔드포인트와 같은 기능을 사용하려면 SNI 확장을 사용해야 합니다. 자세한 내용은 의 전송 보안을 AWS IoT 참조하세요.

지속되는 연결 문제를 해결하려면 어떻게 해야 하나요?

AWS Device Advisor를 사용하여 문제를 해결할 수 있습니다. Device Advisor의 미리 빌드된 테스트는 TLS, MQTT, AWS IoT 디바이스 섀도우 및 AWS IoT 작업 사용에 대한 모범 사례에 대해 디바이스 소프트웨어를 검증하는 데 도움이 됩니다.

다음은 기존 Device Advisor 콘텐츠 링크입니다.

Authentication

AWS IoT 엔드포인트에 연결하려면 디바이스를 인증해야 합니다. 인증X.509 클라이언트 인증서에를 사용하는 디바이스의 경우 인증서를에 등록 AWS IoT 하고 활성화해야 합니다.

디바이스가 AWS IoT 엔드포인트를 인증하는 방법

클라이언트의 트러스트 스토어에 AWS IoT CA 인증서를 추가합니다. AWS IoT Core에서 서버 인증에 대한 설명서를 참조한 다음 링크를 따라 적절한 CA 인증서를 다운로드하세요.

디바이스가에 연결되면 무엇이 확인되나요 AWS IoT?

디바이스가 AWS IoT에 연결을 시도할 때:

1. AWS IoT 는 유효한 인증서와 서버 이름 표시(SNI) 값을 확인합니다.
2. AWS IoT 는 사용된 인증서가 계정에 등록되어 AWS IoT 있고 활성화되었는지 확인합니다.
3. 디바이스가 메시지 구독 또는 게시 AWS IoT와 같은 작업을 수행하려고 하면 연결하는 데 사용한 인증서에 연결된 정책을 확인하여 디바이스가 해당 작업을 수행할 권한이 있는지 확인합니다.

올바로 구성된 인증서를 어떻게 확인할 수 있습니까?

OpenSSL `s_client` 명령을 사용하여 AWS IoT 엔드포인트와 연결을 테스트하세요.

```
openssl s_client -connect custom_endpoint.iot.aws-region.amazonaws.com:8443 -
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

`openssl s_client` 사용에 관한 자세한 내용은 OpenSSL s_client 설명서를 참조하세요.

인증서 상태를 확인하려면 어떻게 합니까?

- 인증서 나열

  인증서 ID를 모르는 경우 `aws iot list-certificates` 명령을 사용하여 모든 인증서의 상태를 볼 수 있습니다.

- 인증서 세부 정보 표시

  인증서의 ID를 알고 있는 경우 이 명령은 인증서에 대한 자세한 정보를 표시합니다.

  ```
  aws iot describe-certificate --certificate-id "certificateId"
  ```

- AWS IoT 콘솔에서 인증서 검토

  AWS IoT 콘솔의 왼쪽 메뉴에서 보안(Secure)을 선택한 다음 인증서(Certificates)를 선택합니다.

목록에서 연결하는 데 사용할 인증서를 선택하여 세부 정보 페이지를 엽니다.

인증서의 세부 정보 페이지에서 현재 상태를 볼 수 있습니다.

인증서 상태는 세부 정보 페이지의 오른쪽 상단 모서리에 있는 작업(Actions) 메뉴를 사용하여 변경할 수 있습니다.

권한 부여

AWS IoT 리소스는 AWS IoT Core 정책를 사용하여 해당 리소스가 작업을 수행하도록 승인합니다. 작업을 승인하려면 지정된 AWS IoT 리소스에 해당 작업을 수행할 수 있는 권한을 부여하는 정책 문서가 연결되어 있어야 합니다.

브로커로부터 PUBNACK 또는 SUBNACK 응답을 받았습니다. 어떻게 해야 합니까?

호출하는 데 사용하는 인증서에 연결된 정책이 있는지 확인합니다 AWS IoT. 모든 게시/구독 작업은 기본적으로 거부됩니다.

연결된 정책이 수행하려는 작업을 승인하는지 확인하세요.

연결된 정책이 승인된 작업을 수행하려는 리소스를 승인하는지 확인하세요.

내 로그에 AUTHORIZATION_FAILURE 항목이 있습니다.

호출하는 데 사용하는 인증서에 연결된 정책이 있는지 확인합니다 AWS IoT. 모든 게시/구독 작업은 기본적으로 거부됩니다.

연결된 정책이 수행하려는 작업을 승인하는지 확인하세요.

연결된 정책이 승인된 작업을 수행하려는 리소스를 승인하는지 확인하세요.

정책에서 권한 부여한 내용을 어떻게 확인합니까?

AWS IoT 콘솔의 왼쪽 메뉴에서 보안을 선택한 다음 인증서를 선택합니다.

목록에서 연결하는 데 사용할 인증서를 선택하여 세부 정보 페이지를 엽니다.

인증서의 세부 정보 페이지에서 현재 상태를 볼 수 있습니다.

인증서 세부 정보 페이지의 왼쪽 메뉴에서 정책(Policies)을 선택하고 인증서에 연결된 정책을 봅니다.

원하는 정책을 선택하여 세부 정보 페이지를 봅니다.

정책의 세부 정보 페이지에서 정책의 정책 문서를 검토하여 어떤 권한을 부여하는지 확인합니다.

정책 문서 편집(Edit policy document)을 선택하여 정책 문서를 변경합니다.

보안 및 자격 증명

AWS IoT 사용자 지정 도메인 구성을 위한 서버 인증서를 제공하면 인증서의 도메인 이름은 최대 4개입니다.

자세한 내용은 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.

진단 규칙 문제

> 이 주제를 개선하도록 도와주세요.
> 이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

이 섹션에서는 규칙에 문제가 발생할 때 확인해야 할 몇 가지 사항에 대해 설명합니다.

문제 해결을 위한 CloudWatch Logs 구성

규칙 관련 문제를 디버깅하는 가장 좋은 방법은 CloudWatch Logs을 사용하는 것입니다. 에 대해 CloudWatch Logs를 활성화하면 트리거되는 규칙과 성공 또는 실패를 확인할 AWS IoT수 있습니다. 또한 WHERE 절 조건이 일치하는지 여부에 대한 정보도 얻을 수 있습니다. 자세한 내용은 CloudWatch Logs AWS IoT 를 사용하여 모니터링 단원을 참조하세요.

가장 일반적인 규칙 문제는 권한 부여입니다. 로그에는 리소스에서 AssumeRole 실행 권한이 역할에게 부여되지 않았는지 표시됩니다. 아래는 세부 로깅에서 작성된 로그 예제입니다.


```
{
    "timestamp": "2017-12-09 22:49:17.954",
    "logLevel": "ERROR",
    "traceId": "ff563525-6469-506a-e141-78d40375fc4e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecution",
    "clientId": "iotconsole-123456789012-3",
    "topicName": "test-topic",
    "ruleName": "rule1",
    "ruleAction": "DynamoAction",
```

```
    "resources": {
        "ItemHashKeyField": "id",
        "Table": "trashbin",
        "Operation": "Insert",
        "ItemHashKeyValue": "id",
        "IsPayloadJSON": "true"
    },
    "principalId": "ABCDEFG1234567ABCD890:outis",
    "details": "User: arn:aws:sts::123456789012:assumed-role/dynamo-
testbin/5aUMInJH is not authorized to perform: dynamodb:PutItem on
 resource: arn:aws:dynamodb:us-east-1:123456789012:table/testbin (Service:
 AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException; Request ID:
 AKQJ987654321AKQJ123456789AKQJ987654321AKQJ987654321)"
}
```

아래는 글로벌 로깅에서 비슷하게 작성된 로그 예제입니다.

```
2017-12-09 22:49:17.954 TRACEID:ff562535-6964-506a-e141-78d40375fc4e
PRINCIPALID:ABCDEFG1234567ABCD890:outis [ERROR] EVENT:DynamoActionFailure
TOPICNAME:test-topic CLIENTID:iotconsole-123456789012-3
MESSAGE:Dynamo Insert record failed. The error received was User:
 arn:aws:sts::123456789012:assumed-role/dynamo-testbin/5aUMInJI is not authorized to
 perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
testbin
(Service: AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException;
 Request ID: AKQJ987654321AKQJ987654321AKQJ987654321AKQJ987654321).
Message arrived on: test-topic, Action: dynamo, Table: trashbin, HashKeyField: id,
 HashKeyValue: id, RangeKeyField: None, RangeKeyValue: 123456789012
No newer events found at the moment. Retry.
```

자세한 내용은 the section called “CloudWatch 콘솔에서 AWS IoT 로그 보기” 단원을 참조하세요.

외부 서비스 진단

외부 서비스는 최종 사용자에 의해 제어됩니다. 규칙을 실행하기 전에 규칙에 연결된 외부 서비스가 설정되어 있고 애플리케이션에 충분한 처리량 및 용량 단위가 있는지 확인하세요.

SQL 문제 진단

SQL 쿼리가 예상한 데이터를 반환하지 않는 경우:

- 로그에서 오류 메시지를 검토합니다.

- SQL 구문이 메시지의 JSON 문서와 일치하는지 확인합니다.

  주제 메시지 페이로드의 JSON 문서에 사용된 객체 및 속성 이름과 함께 쿼리에 사용된 객체 및 속성 이름을 검토합니다. SQL 쿼리의 JSON 형식에 대한 자세한 내용은 JSON 확장 단원을 참조하세요.
- JSON 객체 또는 속성 이름에 예약된 문자 또는 숫자가 포함되어 있는지 확인합니다.

  SQL 쿼리의 JSON 개체 참조의 예약 문자에 대한 자세한 내용은 JSON 확장 단원을 참조하세요.

Shadows 문제 진단

이 주제를 개선하도록 도와주세요.
이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

섀도우 진단

문제	문제 해결 지침
디바이스 섀도우 문서가 `Invalid JSON document`로 거부됩니다.	JSON을 잘 알지 못할 경우 본 설명서에서 제공하는 예제를 자체 용도에 맞춰 수정하세요. 자세한 내용은 섀도우 문서 예제 단원을 참조하세요.
올바른 JSON을 제출했지만 디바이스 섀도우 문서에 저장되지 않거나 일부만 저장됩니다.	JSON 형식 지정 지침을 따르는지 확인하세요. `desired` 및 `reported` 섹션의 JSON 필드만 저장됩니다. 이들 섹션을 벗어나는 JSON 내용은 (형식적으로 정확하더라도) 무시됩니다.
디바이스 섀도우가 허용 크기를 초과한다는 오류가 표시되었습니다.	디바이스 섀도우는 8KB의 데이터만 지원합니다. JSON 문서 내부에서 필드 이름을 줄여보거나 단순히 추가 사물을 생성하여 더 많은 섀도우를 생성하세요. 디바이스 하나에 개수와 상관 없이 사물/섀도우를 무제한으로 연결할 수 있습니다. 단, 각 사물 이름이 계정에서 고유해야 합니다.
디바이스 섀도우를 받을 때 섀도우의 크기가 8KB보다 큽니다. 어떻게 이럴 수 있나요?	수신 시 AWS IoT 서비스는 메타데이터를 디바이스의 섀도우에 추가합니다. 서비스가 이 데이

문제	문제 해결 지침
	터를 응답에 포함시키지만 8KB 제한에는 포함되지 않습니다. 디바이스 섀도우로 전송된 상태 문서 내 `desired` 상태 및 `reported` 상태에 대한 데이터만 제한에 포함됩니다.
잘못된 버전 때문에 요청이 거부되었습니다. 어떻게 해야 합니까?	GET 작업을 수행하여 최신 상태 문서 버전과 동기화하세요. MQTT를 사용하는 경우, /update/accepted 주제를 구독하세요. 그러면 상태 변경이 통보되며 최신 버전의 JSON 문서를 받을 수 있습니다.
타임스탬프는 몇 초 정도 늦습니다.	개별 필드 및 전체 JSON 문서의 타임스탬프는 AWS IoT 서비스가 문서를 수신하거나 상태 문서가 ./update/accepted 및 ./update/delta 메시지에 게시될 때 업데이트됩니다. 메시지가 네트워크에서 지연될 수 있으며, 이로 인해 타임스탬프가 몇 초 정도 늦을 수 있습니다.
내 디바이스가 해당 섀도우 주제에 대해 게시 및 구독할 수 있지만, HTTP REST API를 통해 섀도우 문서를 업데이트하려 하면 HTTP 403이 발생합니다.	이러한 주제에 대한 액세스를 허용하고 사용 중인 자격 증명에 해당 작업(UPDATE/GET/DELETE)을 허용하는 정책을 IAM에 생성했는지 확인하세요. IAM 정책 및 인증서 정책은 독립적입니다.
기타 문제	디바이스 섀도우 서비스는 CloudWatch Logs에 오류를 로그합니다. 디바이스 및 구성 문제를 식별하려면 CloudWatch Logs를 활성화하여 로그에서 디버깅 정보를 확인하세요.

Salesforce IoT 입력 스트림 작업 문제 진단

이 주제를 개선하도록 도와주세요.

이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

실행 추적

Salesforce 작업의 실행 추적을 어떻게 확인하는가?

CloudWatch Logs AWS IoT 를 사용하여 모니터링 단원을 참조하세요. 로그를 활성화하면 Salesforce 작업의 실행 추적을 확인할 수 있습니다.

작업 성공 및 실패

메시지가 Salesforce IoT 입력 스트림으로 성공적으로 전송되었는지 어떻게 확인할 수 있는가?

CloudWatch Logs의 Salesforce 작업 실행에 의해 생성되는 로그를 확인합니다. 가 표시되면 AWS IoT 규칙 엔진이 Salesforce IoT로부터 메시지가 대상 입력 스트림으로 성공적으로 푸시되었다는 확인을 받았음을 `Action executed successfully`의미합니다.

Salesforce IoT 플랫폼에 문제가 발생한 경우 Salesforce IoT 지원 센터에 문의하세요.

메시지가 Salesforce IoT 입력 스트림으로 성공적으로 전송되지 않았을 경우 어떻게 해야 하는가?

CloudWatch Logs의 Salesforce 작업 실행에 의해 생성되는 로그를 확인합니다. 로그 항목에 따라 다음과 같이 해 볼 수 있습니다.

`Failed to locate the host`

작업의 `url` 파라미터가 올바르고 해당 Salesforce IoT 입력 스트림이 존재하는지 확인하세요.

`Received Internal Server Error from Salesforce`

다시 해 보세요. 문제가 계속되면 Salesforce IoT 지원 센터에 문의하세요.

`Received Bad Request Exception from Salesforce`

전송하려는 페이로드에 오류가 있는지 확인하세요.

`Received Unsupported Media Type Exception from Salesforce`

Salesforce IoT는 현재 이진 페이지로드를 지원하지 않습니다. JSON 페이로드를 전송하고 있는지 확인하세요.

`Received Unauthorized Exception from Salesforce`

작업의 `token` 파라미터가 올바르고 토큰이 유효한지 확인하세요.

`Received Not Found Exception from Salesforce`

작업의 `url` 파라미터가 올바르고 해당 Salesforce IoT 입력 스트림이 존재하는지 확인하세요.

여기에 나열되지 않은 오류가 발생하면 AWS IoT Support에 문의하십시오.

스트림 제한 진단

" AWS 계정에 대한 스트림 제한 초과" 문제 해결

"Error: You have exceeded the limit for the number of streams in your AWS account."가 표시되는 경우 제한 증가를 요청하는 대신 계정에서 사용되지 않는 스트림을 정리할 수 있습니다.

AWS CLI 또는 SDK를 사용하여 생성한 미사용 스트림을 정리하려면:

```
aws iot delete-stream -stream-id value
```

자세한 내용은 스트림 삭제를 참조하세요.

> **Note**
>
> list-streams 명령을 사용하여 스트림 ID를 찾을 수 있습니다.

디바이스 플릿 연결 해제 문제 해결

> **이 주제를 개선하도록 도와주세요.**
>
> 이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

AWS IoT 디바이스 플릿 연결 해제는 여러 가지 이유로 발생할 수 있습니다. 이 문서에서는 연결 해제 이유를 진단하는 방법과 정기적인 AWS IoT 서비스 유지 관리 또는 제한 제한으로 인한 연결 해제를 처리하는 방법을 설명합니다.

연결 해제 원인을 진단하려면

CloudWatch에서 AWSIotLogsV2 로그 그룹을 확인하여 로그 항목의 disconnectReason 필드에서 연결 해제 이유를 식별할 수 있습니다.

AWS IoT의 수명 주기 이벤트 기능을 사용하여 연결 해제 이유를 식별할 수도 있습니다. 수명 주기의 연결 해제 이벤트($aws/events/presence/disconnected/*clientId*)를 구독한 경우 연결 해제

가 발생할 AWS IoT 때에서 알림을 받게 됩니다. 알림의 `disconnectReason` 필드에서 연결 해제 이유를 확인할 수 있습니다.

자세한 내용은 CloudWatch AWS IoT 로그 항목 및 수명 주기 이벤트를 참조하세요.

AWS IoT 서비스 유지 관리로 인한 연결 해제 문제를 해결하려면

AWS IoT의 서비스 유지 관리로 인한 연결 해제는 AWS IoT의 수명 주기 이벤트 및 CloudWatch`SERVER_INITIATED_DISCONNECT`로 로깅됩니다. 이러한 연결 해제를 처리하려면 클라이언트 측 설정을 조정하여 디바이스를 AWS IoT 플랫폼에 자동으로 다시 연결할 수 있도록 합니다.

조절 제한으로 인한 연결 끊김 문제를 해결하려면

제한 제한으로 인한 연결 해제는 AWS IoT의 수명 주기 이벤트 및 CloudWatch`THROTTLED`로 로깅됩니다. 이러한 연결 끊김을 처리하기 위해 디바이스 수가 증가함에 따라 메시지 브로커 제한 증가를 요청할 수 있습니다.

자세한 내용은 AWS IoT 코어 메시지 브로커를 참조하세요.

AWS IoT Device Management 문제 해결 가이드

> 이 주제를 개선하도록 도와주세요.
> 이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

이 섹션은에 대한 문제 해결 섹션입니다 AWS IoT Device Management.

주제

- AWS IoT 작업 문제 해결
- 플릿 인덱싱 문제 해결
- AWS IoT 디바이스 관리 소프트웨어 패키지 카탈로그 문제 해결

AWS IoT 작업 문제 해결

다음은 AWS IoT 작업에 대한 문제 해결 섹션입니다.

AWS IoT 작업 엔드포인트를 찾으려면 어떻게 해야 합니까?

AWS IoT 작업 컨트롤 플레인 엔드포인트를 찾으려면 어떻게 해야 합니까?

AWS IoT 작업은 HTTPS 프로토콜을 사용하여 컨트롤 플레인 API 작업을 지원합니다. HTTPS 프로토콜을 사용하여 올바른 컨트롤 플레인 엔드포인트에 연결했는지 확인하세요.

AWS 리전별 엔드포인트 목록은 AWS IoT 코어 - 컨트롤 플레인 엔드포인트를 참조하세요.

FIPS 준수 AWS IoT 작업 컨트롤 플레인 엔드포인트 목록은 서비스별 FIPS 엔드포인트를 참조하세요.

> **Note**
>
> AWS IoT 작업 및는 동일한 AWS 리전별 엔드포인트를 AWS IoT Core 공유합니다.

AWS IoT 작업 데이터 영역 엔드포인트를 찾으려면 어떻게 해야 합니까?

AWS IoT 작업은 HTTPS 및 MQTT 프로토콜을 사용하여 데이터 영역 API 작업을 지원합니다. HTTPS 프로토콜을 사용하여 올바른 데이터터 플레인 엔드포인트에 연결했는지 확인하세요.

- HTTPS 프로토콜
  - 아래에 표시된 describe-endpoint CLI 명령 또는 `DescribeEndpoint` REST API를 사용합니다. 엔드포인트 유형에는 `iot:Jobs`를 사용합니다.

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

- MQTT 프로토콜
  - 아래에 표시된 describe-endpoint CLI 명령 또는 `DescribeEndpoint` REST API를 사용합니다. 엔드포인트 유형에는 `iot:Data-ATS`를 사용합니다.

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

FIPS 준수 AWS IoT 작업 데이터 플레인 엔드포인트는 서비스별 FIPS 엔드포인트를 참조하세요.

AWS IoT 작업 활동을 모니터링하고 지표를 제공하려면 어떻게 해야 합니까?

Amazon CloudWatch를 사용한 AWS IoT 작업 모니터링 활동은 진행 중인 AWS IoT 작업 작업에 대한 실시간 가시성을 제공하고 AWS IoT 규칙을 통해 CloudWatch 경보로 비용을 제어하는 데 도움이 됩

니다. AWS IoT 작업 활동을 모니터링하고 CloudWatch 경보를 설정하려면 먼저 로깅을 구성해야 합니다. 로깅 설정에 대한 자세한 내용은 AWS IoT 로깅 구성 섹션을 참조하세요.

Amazon CloudWatch에 대한 자세한 내용과 IAM 사용자 역할을 통해 CloudWatch 리소스 사용 권한을 설정하는 방법에 대한 자세한 내용은 Amazon CloudWatch의 자격 증명 및 액세스 관리를 참조하세요.

Amazon CloudWatch를 사용하여 AWS IoT 작업 지표 및 모니터링을 설정하려면 어떻게 해야 합니까?

AWS IoT 로깅을 설정하려면 AWS IoT 로깅 구성에 설명된 단계를 따릅니다. AWS IoT 로깅 설정은 에서 수행할 수 있습니다. AWS Management Console AWS CLI또는 특정 사물 그룹에 대한 API. AWS IoT logging 설정은 AWS CLI 또는 API에서만 수행해야 합니다.

AWS IoT 작업 지표 섹션에는 AWS IoT 작업 활동을 모니터링하는 데 사용되는 AWS IoT 작업 지표가 포함되어 있습니다. 및에서 지표를 보는 방법을 설명합니다 AWS Management Console AWS CLI.

또한 면밀히 모니터링하려는 특정 지표에 대해 경고하도록 CloudWatch 경보를 설정할 수 있습니다. 경보 설정에 대한 지침은 Amazon CloudWatch 경보 사용을 참조하세요.

디바이스 플릿 및 단일 디바이스 문제 해결

작업 실행 상태는 무기한 **QUEUED**로 유지됩니다.

상태가 QUEUED인 작업 실행이 다음 논리적 상태(예: IN_PROGRESS, FAILED 또는 TIMED_OUT) 로 진행되지 않는 경우 다음 시나리오 중 하나가 원인일 수 있습니다.

- CloudWatch 콘솔에 있는 CloudWatch 로그에서 디바이스 활동을 검토하세요. 자세한 내용은 CloudWatch Logs를 AWS IoT 사용한 모니터링을 참조하세요.
- 작업 및 후속 작업 실행과 관련된 IAM 역할에는 해당 IAM 역할에 연결된 IAM 정책의 정책 설명 중 하나에 나열된 올바른 권한이 없을 수 있습니다. `describe-job` API를 사용하여 해당 작업 및 후속 작업 실행에 연결된 IAM 역할을 식별하고 올바른 권한이 있는지 IAM 정책을 검토하세요. 정책 권한 설명이 업데이트되면 리소스에서 `AssumeRole` API 명령을 수행할 수 있어야 합니다.

내 사물 또는 사물 그룹에 대한 작업 실행이 생성되지 않았습니다.

작업의 상태가 IN_PROGRESS로 업데이트되면 대상 그룹의 모든 디바이스에 작업 문서가 롤아웃되기 시작합니다. 이 상태 업데이트는 각 대상 디바이스에 대한 작업 실행을 생성합니다. 대상 디바이스 중 하나에 대한 작업 실행이 생성되지 않은 경우 다음 지침을 참조하세요.

- 작업이 직접 thing을 대상으로 지정하고, 작업이 IN_PROGRESS 상태이며 작업이 동시 실행 중이어야 합니다. 세 가지 조건을 모두 충족한다면 작업이 아직 대상 그룹의 모든 디바이스에 작업 실행을 전송 중이며 해당 thing이 아직 작업 실행을 받지 못한 것입니다.
  - AWS Management Console에서 작업에 대한 대상 그룹의 디바이스와 작업 상태를 검토하거나 describe-job API 명령을 사용합니다.
  - describe-job API 명령을 사용하여 작업의 IsConcurrent 속성이 true 또는 false로 설정되어 있는지 검토하세요. 자세한 내용은 작업 제한을 참조하세요.
- 작업이 직접 *thing*을 대상으로 지정하지 않았습니다.
  - Thing이 ThingGroup에 추가되었고 작업이 ThingGroup을 대상으로 지정했다면 Thing이 ThingGroup에 속하는지 확인해야 합니다.
  - 작업이 IN_PROGRESS 상태의 스냅샷 작업이며 동시 실행 중이라면, 아직 대상 그룹의 모든 디바이스에 작업 실행을 전송 중이며 해당 이 아직 작업 실행을 받지 못한 것입니다.
  - 작업이 IN_PROGRESS 상태의 연속 작업이며 동시 실행 중이라면, 아직 대상 그룹의 모든 디바이스에 작업 실행을 전송 중이며 해당 이 아직 작업 실행을 받지 못한 것입니다. 연속 작업의 경우에 한해, ThingGroup에서 Thing을 제거한 후 ThingGroup에 Thing을 다시 추가할 수 있습니다.
  - 작업이 상태인 스냅샷 작업IN_PROGRESS이고 동시 작업이 아닌 경우 Thing 또는 ThingGroup 멤버십 관계가 AWS IoT 작업에서 승인되지 않을 수 있습니다. Job을 생성하기 전에 AddThingToThingGroup 통화 후 몇 초의 대기 시간을 추가하는 것이 좋습니다. 또는 대상 선택을 Continuous로 전환하여 서비스가 지연 Thing 및 ThingGroup 멤버십 연결 이벤트를 채우도록 할 수 있습니다.

LimitedExceededException 오류로 인한 새 작업 실패

LimitedExceededException 오류 응답과 함께 작업 생성이 실패하는 경우, list-jobs API를 호출하고 isConcurrent=true를 사용하여 모든 작업을 검토하여 작업 동시 실행 한도에 도달하지 않았는지 확인하세요. 동시 작업에 대한 추가 정보는 작업 제한을 참조하세요. 작업 동시 실행 제한과 한도 증가를 요청하려면 AWS IoT Device Management 작업 제한과 할당량을 참조하세요.

작업 문서 크기 제한

작업 문서 크기는 MQTT 페이로드 크기로 제한됩니다. 32kB(킬로바이트), 32,000B(바이트)보다 큰 작업 문서가 필요한 경우, Amazon S3에 작업 문서를 생성 및 저장하고 CreateJob API 또는 AWS CLI를 사용할 수 있도록 documentSource 필드에 Amazon S3 객체 URL을 추가하세요. 의 경우

작업을 생성할 때 Amazon S3 URL 텍스트 상자에 Amazon S3 객체 URL을 AWS Management Console추가합니다.

- AWS Management Console 작업 설명서 생성: 를 사용하여 작업 생성 및 관리 AWS Management Console
- AWS CLI 작업 설명서 생성: 를 사용하여 작업 생성 및 관리 AWS CLI
- `CreateJob` API 설명서: CreateJob

디바이스 측 MQTT 메시지 요청 스로틀 제한

오류 코드 400(`ThrottlingException`)을 수신한 경우 동시 디바이스 측 요청 한도에 도달하여 디바이스 측 MQTT 메시지가 실패한 것입니다. 스로틀 한도 및 조정 가능 여부에 대한 자세한 내용은 AWS IoT Device Management 작업 한도 및 할당량을 참조하세요.

연결 제한 시간 오류

오류 코드 400(`RequestExpired`)은 높은 지연 시간 또는 낮은 클라이언트 측 제한 시간 값으로 인한 연결 실패를 나타냅니다.

- 클라이언트 측과 서버 측 간의 연결 테스트에 대한 자세한 내용은 디바이스 데이터 엔드포인트와의 연결 테스트를 참조하세요.

잘못된 API 명령어

API 명령이 유효하지 않다는 오류 메시지가 표시되지 않도록 올바른 API 명령을 입력했는지 확인하세요. AWS IoT API 명령의 전체 목록은 AWS IoT API 참조를 참조하세요.

서비스 측 연결 오류

오류 코드 503(`ServiceUnavailable`)은 오류가 서버 측에서 발생했음을 나타냅니다.

- 모든 AWS Health DashboardAWS 서비스의 현재 상태는 (모든 서비스)를 참조하세요. AWS
- AWS Health Dashboard 현재 개인 상태는 (개인 AWS 계정)을 참조하세요 AWS 계정.

플릿 인덱싱 문제 해결

플릿 인덱싱 서비스에 대한 집계 쿼리 문제 해결

형식 불일치 오류가 있는 경우 CloudWatch Logs를 사용하여 문제를 해결할 수 있습니다. CloudWatch Logs를 활성화해야 플릿 인덱싱 서비스에서 로그를 작성할 수 있습니다. 자세한 내용은 CloudWatch Logs AWS IoT 를 사용하여 모니터링 단원을 참조하십시오.

비관리형 필드에 대해 집계 쿼리를 수행하려면 `UpdateIndexingConfiguration` 또는 update-indexing-configuration에 전달된 `customFields` 인수에 정의한 필드만 지정해야 합니다. 필드 값이 구성된 필드 데이터 형식과 일치하지 않는 경우 집계 쿼리를 수행할 때 이 값이 무시됩니다.

유형 불일치로 인해 필드를 인덱싱할 수 없는 경우 플릿 인덱싱 서비스는 CloudWatch Logs로 오류 로그를 보냅니다. 이 오류 로그에는 필드 이름, 변환할 수 없는 값 및 디바이스에 대한 사물 이름이 포함됩니다. 다음은 오류 로그의 예입니다.

```
{
  "timestamp": "2017-02-20 20:31:22.932",
  "logLevel": "ERROR",
  "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
  "accountId": "000000000000",
  "status": "SucceededWithIssues",
  "eventType": "IndexingCustomFieldFailed",
  "thingName": "thing0",
  "failedCustomFields": [
    {
      "Name": "attributeName1",
      "Value": "apple",
      "ExpectedType": "String"
    },
    {
      "Name": "attributeName2",
      "Value": "2",
      "ExpectedType": "Boolean"
    }
  ]
}
```

디바이스가 약 한 시간 동안 연결이 끊어져 있으면 연결 상태 `timestamp` 값이 누락되었을 수 있습니다. 영구 세션의 경우 클라이언트가 해당 세션에 대해 구성된 TTL(Time-to-Live)보다 오래 연결이 끊어진 후에 값이 누락되었을 수 있습니다. 연결 상태 데이터는 클라이언트 ID에 일치하는 사물 이름이 있

는 연결에 대해서만 인덱싱됩니다. (클라이언트 ID는 디바이스를 연결하는 데 사용되는 값입니다 AWS IoT Core.)

플릿 인덱싱 구성 문제 해결

플릿 인덱싱 구성을 다운그레이드할 수 없음

플릿 지표 또는 동적 그룹과 연결된 데이터 소스를 제거하려는 경우에는 플릿 인덱싱 구성을 다운그레이드할 수 없습니다.

예를 들어 인덱싱 구성에 레지스트리 데이터, 섀도우 데이터 및 연결 데이터가 있고 플릿 지표가 쿼리 `thingName:TempSensor* AND shadow.desired.temperature>80`에 있는 경우 레지스트리 데이터만 포함하도록 인덱싱 구성을 업데이트하면 오류가 발생합니다.

기존 플릿 지표에서 사용하는 사용자 정의 필드 수정은 지원되지 않습니다.

플릿 지표나 동적 그룹이 호환되지 않아 인덱싱 구성을 업데이트할 수 없음

호환되지 않는 플릿 지표 또는 동적 그룹으로 인해 인덱싱 구성을 업데이트할 수 없는 경우, 인덱싱 구성을 업데이트하기 전에 호환되지 않는 플릿 지표 또는 동적 그룹을 삭제하세요.

위치 인덱싱 및 지오쿼리 문제 해결

위치 인덱싱 및 지오쿼리에서 일치하지 않는 유형 오류를 해결하려면 CloudWatch 로그를 활성화하면 됩니다. CloudWatch를 AWS IoT 사용하여 모니터링하는 방법에 대한 자세한 내용은 step-by-step 가이드를 따르세요.

지오쿼리를 사용하여 위치 데이터를 인덱싱할 때 `UpdateIndexingConfiguration`에서 지정하는 위치 필드는 `geoLocations`에 전달하는 위치 필드와 일치해야 합니다. 불일치가 있는 경우 플릿 인덱싱은 일치하지 않는 유형 오류를 CloudWatch로 전송합니다. 이 오류 로그에는 필드 이름, 변환할 수 없는 값 및 디바이스에 대한 사물 이름이 포함됩니다.

다음은 오류 로그의 예입니다.

```
{
"timestamp": "2023-11-09 01:39:43.466",
    "logLevel": "ERROR",
    "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "IndexingGeoLocationFieldFailed",
    "thingName": "thing0",
```

```
    "failedGeolocationFields": [
        {
"Name": "attributeName1",
            "Value": "apple",
            "ExpectedType": "Geopoint"
        }
    ],
    "reason": "failed to index the field because it could not be converted to one of
 the expected geoLocation formats."
}
```

자세한 내용은 위치 데이터 인덱싱 단원을 참조하십시오.

플릿 지표 문제 해결

CloudWatch에서 데이터 포인트를 볼 수 없음

플릿 지표를 생성할 수 있지만 CloudWatch에서 데이터 포인트를 볼 수 없는 경우 쿼리 문자열 기준을 충족하는 항목이 없을 수 있습니다.

플릿 지표를 생성하는 방법에 대한 이 예제 명령을 참조하세요.

```
aws iot create-fleet-metric --metric-name "example_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature>80" --period 60 --aggregation-field
 "attributes.temperature" --aggregation-type name=Statistics,values=count
```

쿼리 문자열 기준 `--query-string "thingName:TempSensor* AND attributes.temperature>80"`을 충족하는 사물이 없는 경우

- `values=count`를 사용하면 플릿 지표를 생성할 수 있고 CloudWatch에 표시할 데이터 포인트가 있습니다. 값 `count`의 데이터 포인트는 항상 0입니다.
- `count` 이외의 `values`를 사용하면 플릿 지표를 생성할 수 있지만 CloudWatch에 집합 지표가 표시되지 않고 CloudWatch에 표시할 데이터 포인트가 없습니다.

AWS IoT 디바이스 관리 소프트웨어 패키지 카탈로그 문제 해결

디바이스 AWS IoT 관리 소프트웨어 패키지 카탈로그의 문제 해결 섹션입니다.

오류 메시지 일반 문제 해결

이 섹션에서는 소프트웨어 패키지 버전 수명 주기 전반에 걸쳐 나타나는 일반적인 오류를 나열합니다.

`HeadBucket` 오류

작업 배포 중에 파일 업로드에 사용되는 Amazon S3 버킷을 검증하기 위해 HeadBucket API 작업 또는 head-bucket CLI 명령을 직접적으로 호출할 때 다음 오류 메시지가 나타납니다.

Amazon S3 버킷을 사용하여 작업 배포 중에 파일을 업로드하는 방법에 대한 자세한 내용은 파일 업로드를 위한 미리 서명된 URL 섹션을 참조하세요.

```
InvalidRoleException
    "Permission denied when attempting to use role %s to access bucket %s."
```

```
InvalidRequestException
    "Cross region S3 bucket is not supported for presigned url upload placeholder"
```

```
InvalidRequestException
    "S3 bucket in job document presigned url upload placeholder not found"
```

```
InvalidRequestException
    "Given S3 bucket name is invalid."
```

```
InvalidRequestException
    "Provided S3 bucket is not valid: %s. Error: %s"
```

Amazon S3 GetObject

잘못된 인수가 제공되면 다음 오류 메시지가 발생하여 Amazon S3 `GetObject` API 작업이 실패합니다.

```
InvalidRequestException
    "Provided argument for presigned url is invalid"
```

Amazon S3 버전 ID 지원

버전 관리 제어를 사용하여 Amazon S3 버킷에 대한 액세스를 요청할 때 `versionId`를 포함시켜야 합니다. 그렇지 않으면 아래 오류가 채워질 수 있습니다.

버전 관리 제어를 사용하는 Amazon S3 버킷에 대한 자세한 내용은 Amazon S3 버킷에서 버전 관리를 참조하세요.

```
InvalidRequestException
```

```
    "VersionId not found when attempting to access s3 url"
```

파일 업로드를 위한 미리 서명된 URL 내부의 자리 표시자

작업 배포 중에 대상 Amazon S3 버킷에 파일을 업로드하는 데 사용되는 미리 서명된 URL 내에 자리 표시자에 문제가 발생하면 다음 오류 메시지가 나타납니다. Amazon S3 버킷을 사용하여 작업 배포 중에 파일을 업로드하는 방법과 로컬 자리 표시자에 대한 자세한 내용은 파일 업로드를 위한 미리 서명된 URL 섹션을 참조하세요.

로컬 자리 표시자가 인식되지 않으면 아래 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
    "Undefined placeholder, ${...}, inside of presign url upload parameter"
```

파일 업로드가 아닌 미리 서명된 URL에서 로컬 자리 표시자를 사용하려고 하면 아래 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
    "Local placeholder, ${...}, is only valid inside of presign url upload"
```

Amazon S3 URL 중첩 잘못됨

Amazon S3 URL이 다른 자리 표시자 내부에 잘못 중첩되면 다음 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
    "${aws:%s[...]} should not be the second layer pattern."
```

패키지 버전 아티팩트 중첩

패키지 버전 아티팩트 미리 서명된 URL이 다른 자리 표시자 내부에 잘못 중첩되면 다음 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
    "${aws:iot:package:[...]:artifact:s3-presigned-url} cannot be nested inside another
 placeholder."
```

패키지 버전 아티팩트 누락

참조된 패키지 버전 아티팩트를 찾을 수 없는 경우 다음 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
```

```
    "Package %s version %s does not have an associated artifact to generate an S3
presigned url."
```

소프트웨어 패키지 및 패키지 버전 자리 표시자

다음 오류 메시지는 destinationPackageVersions 파라미터 또는 패키지 버전 세부 정보 페이지의 버전 ARN 탭에 참조된 여러 소프트웨어 패키지 및 패키지 버전으로 인해 소프트웨어 패키지 및 패키지 버전의 작업 문서 자리 표시자가 작업 배포에 대해 원하는 유효한 값으로 확인되지 않는 경우 나타납니다.

```
InvalidJobDocumentException
    "Cannot resolve empty package name and version name given multiple elements in
destination package versions."
```

빈 소프트웨어 패키지 및 패키지 버전 사용

작업 문서에서 빈 패키지 또는 패키지 버전을 다른 버전 없이 사용하려고 시도하면 다음 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
    "Empty package name and version name have to be used in pair."
```

작업 문서에서 Null 사용

작업 문서에서 패키지 버전으로 $null을 지정하려고 하면 다음 오류 메시지가 나타납니다. $null은 CreateJob API 작업을 사용할 때 destinationPackageVersions 파라미터 내에서만 사용할 수 있습니다.

```
InvalidJobDocumentException
    "$null is not allowed to be referenced as a package version in job documents."
```

패키지 버전의 모든 속성

패키지 버전의 모든 속성을 사용하고 추가 텍스트 또는 자리 표시자로 둘러싸려고 하면 다음 오류 메시지가 나타납니다.

소프트웨어 패키지 버전의 모든 속성 사용에 대한 자세한 내용은 AWS IoT 작업의 대체 파라미터 섹션을 참조하세요.

```
InvalidJobDocumentException
```

```
    "The package version attribute placeholder for all attributes has to be a
json value by itself and not appended with other strings or nested with other
placeholders."
```

파일 업로드를 위한 미리 서명된 URL의 로컬 자리 표시자 제한

작업 배포 중에 파일 업로드를 위해 미리 서명된 URL에 사용되는 로컬 자리 표시자 수 제한을 초과하면 다음 오류 메시지가 나타납니다.

작업 배포 중에 파일 업로드에 미리 서명된 URL을 사용하는 방법에 대한 자세한 내용은 파일 업로드를 위한 미리 서명된 URL 섹션을 참조하세요.

```
InvalidJobDocumentException
    "The occurrence of local placeholder %s within S3 presigned url upload placeholder
exceeds limit of %d."
```

Amazon S3 버킷의 로컬 자리 표시자

작업 배포 중에 파일 업로드에 사용되는 미리 서명된 URL 자리 표시자의 Amazon S3 버킷 이름에 로컬 자리 표시자 URL을 배치하려고 하면 다음 오류 메시지가 나타납니다.

작업 배포 중에 파일 업로드에 미리 서명된 URL을 사용하는 방법에 대한 자세한 내용은 파일 업로드를 위한 미리 서명된 URL 섹션을 참조하세요.

```
InvalidJobDocumentException
    "S3 bucket name in presigned url upload is not allowed to contain any placeholders"
```

여는 괄호 및 닫는 괄호

닫는 괄호 “}” 없이 작업 문서에 파라미터 또는 자리 표시자를 추가하면 다음 오류 메시지가 나타납니다.

```
InvalidJobDocumentException
    "One or more parameters or placeholders are not terminated."
```

Amazon S3 사전 서명된 URL을 사용하는 IAM 역할

IAM 역할 없이 작업 문서에서 Amazon S3 미리 서명된 URL을 사용하려고 하면 다음 오류 메시지가 나타납니다.

Amazon S3 미리 서명된 URL 대한 자세한 내용은 미리 서명된 URL 섹션을 참조하세요.

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document."
```

패키지 버전 아티팩트에 대한 Amazon S3 사전 서명된 URL을 사용하는 IAM 역할

IAM 역할이 없는 작업 문서에서 패키지 버전 아티팩트를 나타내는 Amazon S3 미리 서명된 URL을 사용하려고 하면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document for package %s version %s artifact."
```

소프트웨어 재료표 오류 메시지

이 섹션에서는 패키지 버전에 연결된 소프트웨어 재료표(SBOM)와 관련된 일반적인 오류를 나열합니다.

SBOM 연결 요청에 대한 입력 검증

`AssociateSbomWithPackageVersion` API 작업을 사용하고 `s3Location` 파라미터가 null인 경우 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "Associate request needs to include SBOM reference"
```

`AssociateSbomWithPackageVersion` API 작업에 대한 자세한 내용은 AssociateSbomWithPackageVersion을 참조하세요.

SBOM 유효성 검사 오류

이 섹션에서는 소프트웨어 패키지 버전과 연결된 경우 소프트웨어 재료표(SBOM)의 초기 유효 기간 동안 나타나는 일반적인 오류를 나열합니다.

`AssociateSbomWithPackageVersion` API 작업을 사용하고 파라미터가 null인 경우 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 bucket name for SBOM cannot be null"
```

`AssociateSbomWithPackageVersion` API 작업에 대해 `s3Location` 파라미터에 있는 `bucket`의 문자열이 너무 길면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 bucket name for SBOM is illegal. String length exceeds
 limit"
```

key 파라미터가 null이면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 key name for SBOM cannot be null"
```

AssociateSbomWithPackageVersion API 작업에 대해 s3Location 파라미터에 있는 key의 문자열이 너무 길면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 key name for SBOM is illegal. String length exceeds limit"
```

AssociateSbomWithPackageVersion API 작업에 대해 s3Location 파라미터에 있는 version의 문자열이 null이면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 object version for SBOM cannot be null"
```

AssociateSbomWithPackageVersion API 작업에 대해 s3Location 파라미터에 있는 version의 문자열이 너무 길면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 object version for SBOM is illegal. String length exceeds
 limit"
```

Amazon S3 버킷에 저장된 SBOM zip 아카이브 파일의 크기가 너무 크면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "S3 object file size exceeds limit"
```

AssociateSbomWithPackageVersion API 작업을 사용하고 현재 진행 중인 SBOM 검증 수가 이미 최대 한도에 도달하면 다음 오류 메시지가 나타납니다.

```
LimitExceededException "Too many ongoing SBOM validation workflows. Please wait and
 retry"
```

Amazon S3 버킷의 SBOM 파일 액세스 문제

Amazon S3 버킷이 존재하지 않거나 Amazon S3 버킷에 액세스할 수 있는 적절한 권한이 부여되지 않아 다른 엔터티가 Amazon S3 Amazon S3 버킷에 액세스하지 못하는 경우 다음 오류 메시지가 나타납니다.

Amazon S3 버킷에 액세스하는 데 필요한 권한에 대한 자세한 정보는 소프트웨어 재료표 스토리지 섹션을 참조하세요.

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket exists and S3 permission is granted."
```

Amazon S3 버킷이 존재하지 않거나 Amazon S3 버킷에 저장된 콘텐츠에 액세스할 수 있는 적절한 권한이 부여되지 않아 다른 엔터티가 key 파라미터의 SBOM zip 아카이브 파일에 액세스하지 못하는 경우 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the key
 exists and S3 permission is granted."
```

버킷, 키 및 버전 ID가 존재하지 않아 다른 엔터티가 Amazon S3 버킷에 액세스하지 못하거나 Amazon S3 버킷에 액세스할 수 있는 적절한 권한이 부여되지 않은 경우 다음 오류 메시지가 나타납니다. 또한 부여된 권한이 Amazon S3 버킷의 SBOM zip 아카이브 파일에 액세스하기에 충분하지 않은 경우 이 오류 메시지가 표시될 수 있습니다.

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket/key/version exists and S3 permission is granted."
```

버킷이 다른 리전에 위치하여 다른 엔터티가 Amazon S3 버킷에 액세스하지 못하면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "Cross-region S3 bucket for %s is not supported."
```

AssociateSbomWithPackageVersion API 작업을 사용할 때 bucket, key 또는 version 파라미터의 철자가 잘못 지정되어 다른 엔터티가 Amazon S3 버킷에 액세스하지 못하면 다음 오류 메시지가 나타납니다.

```
InvalidRequestException "Please make sure SBOM S3 bucket name/key length/version is
 valid"
```

AWS IoT Device Advisor 문제 해결 가이드

이 주제를 개선하도록 도와주세요.

이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

일반

Q: 여러 테스트 스위트를 병렬로 실행할 수 있습니까?

A: 예. Device Advisor는 디바이스 수준 엔드포인트를 사용하여 서로 다른 디바이스에서 여러 테스트 스위트를 실행할 수 있습니다. 계정 수준 엔드포인트를 사용하는 경우 계정당 하나의 Device Advisor 엔드포인트를 사용할 수 있으므로 한 번에 하나의 스위트를 실행할 수 있습니다. 디바이스 구성에 대한 자세한 내용은 디바이스 구성을 참조하세요.

Q: 디바이스에서 Device Advisor에 의해 TLS 연결이 거부되었다는 것을 확인했습니다. 예상된 동작입니까?

A: 예. Device Advisor는 각 테스트 실행 전후에 TLS 연결을 거부합니다. 사용자는 Device Advisor를 통한 테스트 완전 자동화를 위해 디바이스 재시도 메커니즘을 구현하는 것이 좋습니다. TLS 연결, MQTT 연결 및 MQTT 게시와 같이 둘 이상의 테스트 사례로 테스트 스위트를 실행하는 경우 디바이스에 빌드된 메커니즘을 사용하는 것이 좋습니다. 이 메커니즘은 매 5초마다 1분~2분간 테스트 엔드포인트에 연결을 시도할 수 있습니다. 이렇게 하면 자동화된 방식으로 여러 테스트 케이스를 순서대로 실행할 수 있습니다.

보안 분석 및 운영 문제 해결 목적으로 내 계정에서 이루어진 모든 Device Advisor API 호출 기록을 얻을 수 있습니까?

A: 예. 계정에서 수행한 Device Advisor API 호출 기록을 수신하려면 AWS IoT 관리 콘솔에서 CloudTrail을 켜고 이벤트 소스를 로 필터링하면 됩니다`iotdeviceadvisor.amazonaws.com`.

Q: CloudWatch에서 Device Advisor 로그를 보려면 어떻게 해야 합니까?

A: 서비스 역할에 필요한 정책(예: CloudWatchFullAccess)을 추가하면 테스트 스위트 실행 중에 생성된 로그가 CloudWatch에 업로드됩니다(설정 참조). 테스트 스위트에 테스트 사례가 하나 이상 있는 경우 로그 스트림이 두 개인 ‘aws/iot/deviceadvisor/$testSuiteId’ 로그 그룹이 생성됩니다. 스트림 하나는 ‘$testRunId’라는 이름이며 설치 및 정리 단계 등 테스트 스위트에서 테스트 사례 실행 전후에 수행된 작업의 로그를 포함합니다. 또 다른 스트림은 테스트 스위트 실행과 관련된 ‘$suiteRunId_$testRunId’입니다. 디바이스 및에서 전송된 이벤트 AWS IoT Core 는이 로그 스트림에 기록됩니다.

Q: 디바이스 권한 역할의 용도는 무엇입니까?

A: Device Advisor는 테스트 디바이스와 사이에 위치하여 테스트 시나리오 AWS IoT Core 를 시뮬레이션합니다. 디바이스 권한 역할을 가정하고 사용자를 대신하여 연결을 시작하여 테스트 디바이스의 연결 및 메시지를 수락하고 AWS IoT Core 로 전달합니다. 디바이스 역할 권한이 테스트를 실행하는 데 사용하는 인증서의 권한과 동일한지 확인하는 것이 중요합니다. Device Advisor가 디바

이스 권한 역할을 사용하여 사용자를 대신하여에 AWS IoT Core 대한 연결을 시작할 때 AWS IoT 인증서 정책이 적용되지 않습니다. 그러나 설정한 디바이스 권한 역할의 권한은 적용됩니다.

Q: Device Advisor는 어떤 리전에서 지원되나요?

A: Device Advisor는 us-east-1, us-west-2, ap-northeast-1, and eu-west-1 리전에서 지원됩니다.

Q: 일관성이 없는 결과가 표시되면 어떻게 해야 하나요?

A: 일관성 없는 결과의 주요 원인 중 하나는 테스트의 `EXECUTION_TIMEOUT`을 너무 낮은 값으로 설정하는 것입니다. 권장 및 기본 `EXECUTION_TIMEOUT` 값에 대한 자세한 내용은 Device Advisor 테스트 사례를 참조하세요.

Q: Device Advisor는 어떤 MQTT 프로토콜을 지원합니까?

A: Device Advisor는 X509 클라이언트 인증서와 함께 MQTT 3.1.1 버전을 지원합니다.

Q: 디바이스를 테스트 엔드포인트에 연결하려고 해도 테스트 케이스가 실행 시간 초과 메시지와 함께 실패하면 어떻게 하나요?

A: 디바이스 역할로 사용할 IAM 역할 생성에 나와 있는 모든 단계를 검증합니다. 테스트가 여전히 실패하면 장치가 올바른 서버 이름 표시(SNI) 확장을 전송하지 않을 수 있습니다. 이 확장은 Device Advisor가 작동하는 데 필요합니다. 올바른 SNI 값은 디바이스 구성 섹션을 따를 때 반환되는 엔드포인트 주소입니다. AWS IoT 또한 디바이스가 전송 계층 보안(TLS) 프로토콜로 SNI(Server Name Indication) 확장을 전송해야 합니다. 자세한 내용은 전송 보안을 참조하세요 AWS IoT.

Q: "libaws-c-mqtt: AWS_ERROR_MQTT_UNEXPECTED_HANGUP" 오류가 표시되면서 MQTT 연결이 실패하거나 디바이스의 MQTT 연결이 Device Advisor 엔드포인트에서 자동으로 끊기고 있습니다. 이 오류를 어떻게 해결할 수 있나요?

A: 이 특정 오류 코드 및 예기치 않은 연결 끊김은 여러 가지 원인으로 인해 발생할 수 있지만, 디바이스에 연결된 디바이스 역할과 관련이 있을 가능성이 큽니다. 아래 체크포인트(우선순위 순)를 점검하여 이 문제를 해결할 수 있습니다.

- 디바이스에 연결된 디바이스 역할에는 테스트를 실행하는 데 필요한 최소한의 IAM 권한이 있어야 합니다. Device Advisor는 연결된 디바이스 역할을 사용해서 테스트 디바이스를 대신하여 AWS IoT MQTT 작업을 수행합니다. 필요한 권한이 없는 경우, 디바이스가 Device Advisor 엔드포인트에 연결을 시도하는 중에 `AWS_ERROR_MQTT_UNEXPECTED_HANGUP` 오류가 표시되거나 예기치 않은 연결 끊김이 발생합니다. 예를 들어 MQTT 게시 테스트 사례를 실행하도록 선택한 경우, 연결 작업과 게시 작업이 모두 해당 ClientId 및 Topic이 있는 역할에 포함되어야 합니다. 쉼표로 값을 구분하여 여러 값을 제공할 수 있으며, 와일드카드(*) 문자를 사용하여 접두사 값을 제공할 수 있습니다. 예: `TestTopic`으로 시작하는 주제에 대한 게시 권한을 제공하려면 리소스 값으로 "`TestTopic*`"을 제공할 수 있습니다. 다음은 몇 가지 정책 예제입니다.

- 리소스 유형에 대한 디바이스 역할에 정의된 값과 코드에 사용된 실제 값이 일치하지 않습니다. 예를 들어 역할에 정의된 ClientID와 디바이스 코드에 사용된 실제 ClientID가 불일치하는 경우입니다. ClientId, Topic 및 TopicFilter와 같은 값은 디바이스 역할 및 코드에서 동일해야 합니다.
- 디바이스에 연결된 디바이스 인증서는 활성 상태여야 하며, 리소스에 대한 필수 작업 권한과 함께 연결된 정책이 있어야 합니다. 디바이스 인증서 정책은 AWS IoT 리소스 및 AWS IoT Core 데이터 영역 작업에 대한 액세스 권한을 부여하거나 거부합니다. Device Advisor를 사용하려면 테스트 사례 중에 사용되는 작업 권한을 부여하는 활성 디바이스 인증서가 디바이스에 연결되어 있어야 합니다.

AWS IoT 오류

> 이 주제를 개선하도록 도와주세요.
> 이 주제를 개선할 수 있는 의견이 있으시다면 알려주세요.

이 섹션에서는에서 전송한 오류 코드를 나열합니다 AWS IoT.

메시지 브로커 오류 코드

오류 코드	오류 설명
400	잘못된 요청.
401	권한이 없음.
403	금지됨.
426	업그레이드가 필요합니다.
503	서비스 사용 불가.

자격 증명 및 보안 오류 코드

오류 코드	오류 설명
401	권한이 없음.

디바이스 섀도우 오류 코드

오류 코드	오류 설명
400	잘못된 요청.
401	권한이 없음.
403	금지됨.
404	찾을 수 없음.
409	충돌.
413	요청이 너무 큼.
422	요청을 처리하지 못함.
429	요청이 너무 많음.
500	내부 오류.
503	서비스 사용 불가.

AWS IoT 디바이스 SDK, 모바일 SDK 및 AWS IoT 디바이스 클라이언트

이 페이지에는 AWS IoT와 선택한 하드웨어 플랫폼을 사용하여 혁신적인 IoT 솔루션을 구축하는 데 도움이 되는 AWS IoT 디바이스 SDK, 오픈 소스 라이브러리, 개발자 가이드, 샘플 앱 및 포팅 가이드가 요약되어 있습니다.

이러한 SDK는 IoT 디바이스에서 사용할 수 있습니다. 모바일 디바이스에서 사용할 IoT 앱을 개발하는 경우 AWS Mobile SDK 단원을 참조하세요. IoT 앱 또는 서버 측 프로그램을 개발하는 경우 AWS SDKs 단원을 참조하세요.

AWS IoT 디바이스 SDK

AWS IoT 디바이스 SDK는 오픈 소스 라이브러리, 샘플 포함 개발자 설명서, 포팅 안내서를 포함하고 있어 사용자는 선택한 하드웨어 플랫폼에 따라 혁신적인 IoT 제품 또는 솔루션을 구축할 수 있습니다.

> **Note**
>
> AWS IoT Device SDK는 MQTT 5 클라이언트를 릴리스했습니다. AWS IoT 디바이스 SDK는 macOS에서 TLS 1.3을 사용하는 것을 지원하지 않습니다.

이러한 SDK는 IoT 디바이스를 MQTT 및 WSS 프로토콜을 사용하여 AWS IoT에 연결합니다.

C++

AWS IoT C++ Device SDK

AWS IoT C++ 디바이스 SDK는 개발자가 AWS 및 AWS IoT API를 사용하여 커넥티드 애플리케이션을 개발할 수 있는 키트입니다. 특히 이 SDK는 리소스의 제한을 받지 않으면서 메시지 대기열, 멀티-스레딩 지원, 최신 언어 같은 고급 기능이 필요한 디바이스를 위해 설계되었습니다. 자세한 내용은 다음 자료를 참조하세요.

- GitHub의 AWS IoT 디바이스 SDK C++ v2
- AWS IoT 디바이스 SDK C++ v2 Readme
- AWS IoT 디바이스 SDK C++ v2 샘플
- AWS IoT 디바이스 SDK C++ v2 API 설명서

Python

Python용 AWS IoT 디바이스 SDK

AWS IoT Device SDK for Python은 개발자가 디바이스를 사용하여 MQTT 또는 MQTT over WebSocket 프로토콜을 통해 AWS IoT 플랫폼에 액세스하는 Python 스크립트를 작성할 수 있게 해줍니다. 디바이스를 AWS IoT에 연결함으로써 사용자는 AWS IoT가 제공하는 메시지 브로커, 규칙 및 섀도우, 그리고 AWS Lambda, Kinesis, Amazon S3 등의 다른 AWS 서비스를 안전하게 사용할 수 있습니다.

- AWS IoT Device SDK for Python v2 on GitHub
- AWS IoT Device SDK for Python v2 Readme
- Python v2용 AWS IoT 디바이스 SDK 샘플
- Python v2용 AWS IoT 디바이스 SDK API 설명서

JavaScript

JavaScript용 AWS IoT 디바이스 SDK

aws-iot-device-sdk.js 패키지는 개발자가 MQTT 또는 MQTT over WebSocket 프로토콜을 사용하여 AWS IoT에 액세스하는 JavaScript 애플리케이션을 작성할 수 있게 해줍니다. 이 패키지는 Node.js 환경 및 브라우저 애플리케이션에서 사용할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- AWS IoT Device SDK for JavaScript v2 on GitHub
- AWS IoT Device SDK for JavaScript v2 Readme
- JavaScript v2용 AWS IoT 디바이스 SDK 샘플
- JavaScript v2용 AWS IoT 디바이스 SDK API 설명서

Java

Java용 AWS IoT 디바이스 SDK

AWS IoT Device SDK for Java는 Java 개발자가 MQTT 또는 MQTT over WebSocket 프로토콜을 통해 AWS IoT에 액세스할 수 있게 해줍니다. 이 SDK에는 섀도우 지원이 기본 제공됩니다. 섀도우는 GET, UPDATE, DELETE 등의 HTTP 메서드를 사용하여 액세스할 수 있습니다. 또한 이 SDK는 간소화된 섀도우 액세스 모델을 지원합니다. 이 모델에서는 개발자가 JSON 문서를 직렬화 또는 역직렬화할 필요 없이 단지 getter 및 setter 메서드를 사용하여 섀도우와 데이터를 교환할 수 있습니다.

> **Note**
>
> Java v2용 AWS IoT Device SDK는 이제 Android 개발을 지원합니다. 자세한 내용은 Android용 AWS IoT Device SDK를 참조하세요.

자세한 내용은 다음 자료를 참조하세요.

- AWS IoT Device SDK for Java v2 on GitHub
- AWS IoT Device SDK for Java v2 Readme
- Java v2용 AWS IoT 디바이스 SDK 샘플
- Java v2용 AWS IoT 디바이스 SDK API 설명서

Swift

AWS IoT Device SDK for Swift

AWS IoT Device SDK for Swift를 사용하면 Swift 개발자가 MQTT 5 프로토콜을 사용하여 Linux 및 Apple macOS, iOS 및 tvOS 플랫폼용 AWS IoT 애플리케이션을 생성할 수 있습니다.

자세한 내용은 다음 자료를 참조하세요.

- GitHub의AWS IoT Device SDK for Java
- AWS IoT Device SDK for Java Readme
- AWS IoT Device SDK for Swift 샘플

AWS IoT Device SDK for Embedded C

> **Note**
>
> 이 SDK는 숙련된 임베디드 소프트웨어 개발자가 사용하기 위한 것입니다.

AWS IoT Device SDK for Embedded C(C-SDK)는 MIT 오픈 소스 라이선스에 따라 제공되는 C 소스 파일의 모음으로, IoT 장치를 AWS IoT Core에 안전하게 연결하기 위해 임베디드 애플리케이션에서 사용할 수 있습니다. 여기에는 MQTT 클라이언트, JSON 파서 및 AWS IoT 디바이스 섀도우, AWS IoT 작

업, AWS IoT 플릿 프로비저닝 및 AWS IoT Device Defender 라이브러리가 포함됩니다. 이 SDK는 소스 형식으로 배포되며 애플리케이션 코드, 기타 라이브러리 및 선택한 운영 체제(OS)와 함께 고객 펌웨어에 구축될 수 있습니다.

AWS IoT Device SDK for Embedded C는 일반적으로 최적화된 C 언어 런타임을 필요로 하는 리소스 제약 디바이스를 대상으로 합니다. 모든 운영 체제에서 SDK를 사용하고, 모든 프로세서 유형(예: MCU 및 MPU)에서 호스팅할 수 있습니다.

자세한 내용은 다음 자료를 참조하세요.

- AWS IoT Device SDK for Embedded C on GitHub
- AWS IoT Device SDK for Embedded C Readme
- Embedded C용 AWS IoT 디바이스 SDK 샘플

AWS Mobile SDK

AWS Mobile SDK는 모바일 앱 개발자에게 AWS IoT Core 서비스의 API, MQTT를 사용한 IoT 디바이스 통신 및 기타 AWS 서비스의 API에 대한 플랫폼별 지원을 제공합니다.

Android

AWS Mobile SDK for Android

AWS Mobile SDK for Android에는 개발자가 AWS을(를) 사용하여 커넥티드 모바일 애플리케이션을 구축할 수 있도록 라이브러리, 샘플 및 설명서가 포함되어 있습니다. 이 SDK는 MQTT 디바이스 통신에 대한 지원과 AWS IoT Core 서비스의 API 호출도 포함합니다. 자세한 내용은 다음 자료를 참조하세요.

- AWS Mobile SDK for AndroidGitHub의
- AWS Mobile SDK for Android Readme
- AWS Mobile SDK for Android 샘플
- AWS Mobile SDK for Android API 참조
- AWSIoTClient Class 참조 문서

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS은 Apache 오픈 소스 라이선스 아래 배포되는 오픈 소스 소프트웨어 개발 키트입니다. AWS Mobile SDK for iOS는 개발자가 AWS를 사용하여 연결된 모바일 애플리케이션을 구축할 수 있도록 라이브러리, 코드 샘플 및 설명서를 제공합니다. 이 SDK는 MQTT 디바이스 통신에 대한 지원과 AWS IoT Core 서비스의 API 호출도 포함합니다. 자세한 내용은 다음 자료를 참조하세요.

- AWS Mobile SDK for iOSGitHub의
- AWS Mobile SDK for iOS Readme
- AWS Mobile SDK for iOS 샘플
- 의 AWSIoT Class 참조 문서AWS Mobile SDK for iOS

AWS IoT 디바이스 클라이언트

AWS IoT 디바이스 클라이언트는 디바이스를 AWS IoT에 연결하고, 플릿 프로비저닝 작업을 수행하고, 디바이스 보안 정책을 지원하고, 보안 터널링을 사용하여 연결하고, 디바이스에서 작업을 처리하는 데 도움이 되는 코드를 제공합니다. 디바이스에 이 소프트웨어를 설치하여 이러한 일상적인 디바이스 작업을 처리할 수 있으므로 특정 솔루션에 집중할 수 있습니다.

> **Note**
>
> AWS IoT 디바이스 클라이언트는 x86_64 또는 ARM 프로세서 및 일반 Linux 운영 체제가 있는 마이크로프로세서 기반 IoT 디바이스와 함께 작동합니다.

C++

AWS IoT 디바이스 클라이언트

AWS IoT Device Client in C++에 대한 자세한 내용은 다음을 참조하세요.

- GitHub에 있는 AWS IoT Device Client in C++ 소스 코드
- AWS IoT Device Client in C++ Readme

이전 AWS IoT 디바이스 SDK 버전

위에 나열된 최신 버전으로 대체된 이전 버전의 AWS IoT Device SDK입니다. 이러한 SDK는 유지 관리 및 보안 업데이트만 받습니다. 새 기능을 포함하도록 업데이트되지 않으며 새 프로젝트에서 사용해서는 안 됩니다.

- AWS IoT C++ Device SDK on GitHub
- AWS IoT C++ Device SDK Readme
- AWS IoT Device SDK for Python v1 on GitHub
- AWS IoT Device SDK for Python v1 Readme
- AWS IoT Device SDK for Java on GitHub
- AWS IoT Device SDK for Java Readme
- AWS IoT Device SDK for JavaScript on GitHub
- AWS IoT Device SDK for JavaScript Readme
- Arduino Yún SDK on GitHub
- Arduino Yún SDK Readme

AWS SDKs AWS IoT 사용에 대한 코드 예제

다음 코드 예제에서는 AWS 소프트웨어 개발 키트(SDK)와 AWS IoT 함께를 사용하는 방법을 보여줍니다.

기본 사항은 서비스 내에서 필수 작업을 수행하는 방법을 보여주는 코드 예제입니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 직접 호출하는 방법을 보여주며, 관련 시나리오의 컨텍스트에 맞는 작업을 볼 수 있습니다.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

코드 예시

- AWS SDKs AWS IoT 사용에 대한 기본 예제
  - 안녕하세요 AWS IoT
  - AWS SDK를 AWS IoT 사용하여의 기본 사항 알아보기
  - AWS SDKs AWS IoT 를 사용하기 위한 작업
    - AWS SDK 또는 CLI와 AttachThingPrincipal 함께 사용
    - AWS SDK 또는 CLI와 CreateKeysAndCertificate 함께 사용
    - AWS SDK 또는 CLI와 CreateThing 함께 사용
    - AWS SDK 또는 CLI와 CreateTopicRule 함께 사용
    - AWS SDK 또는 CLI와 DeleteCertificate 함께 사용
    - AWS SDK 또는 CLI와 DeleteThing 함께 사용
    - AWS SDK 또는 CLI와 DeleteTopicRule 함께 사용
    - AWS SDK 또는 CLI와 DescribeEndpoint 함께 사용
    - AWS SDK 또는 CLI와 DescribeThing 함께 사용
    - AWS SDK 또는 CLI와 DetachThingPrincipal 함께 사용
    - AWS SDK 또는 CLI와 ListCertificates 함께 사용
    - AWS SDK 또는 CLI와 ListThings 함께 사용
    - AWS SDK 또는 CLI와 SearchIndex 함께 사용
    - AWS SDK 또는 CLI와 UpdateIndexingConfiguration 함께 사용
    - AWS SDK 또는 CLI와 UpdateThing 함께 사용

AWS SDKs AWS IoT 사용에 대한 기본 예제

다음 코드 예제에서는 AWS IoT SDKs에서의 기본 사항을 AWS 사용하는 방법을 보여줍니다.

예제

- 안녕하세요 AWS IoT
- AWS SDK를 AWS IoT 사용하여의 기본 사항 알아보기
- AWS SDKs AWS IoT 를 사용하기 위한 작업
  - AWS SDK 또는 CLI와 AttachThingPrincipal 함께 사용
  - AWS SDK 또는 CLI와 CreateKeysAndCertificate 함께 사용
  - AWS SDK 또는 CLI와 CreateThing 함께 사용
  - AWS SDK 또는 CLI와 CreateTopicRule 함께 사용
  - AWS SDK 또는 CLI와 DeleteCertificate 함께 사용
  - AWS SDK 또는 CLI와 DeleteThing 함께 사용
  - AWS SDK 또는 CLI와 DeleteTopicRule 함께 사용
  - AWS SDK 또는 CLI와 DescribeEndpoint 함께 사용
  - AWS SDK 또는 CLI와 DescribeThing 함께 사용
  - AWS SDK 또는 CLI와 DetachThingPrincipal 함께 사용
  - AWS SDK 또는 CLI와 ListCertificates 함께 사용
  - AWS SDK 또는 CLI와 ListThings 함께 사용
  - AWS SDK 또는 CLI와 SearchIndex 함께 사용
  - AWS SDK 또는 CLI와 UpdateIndexingConfiguration 함께 사용
  - AWS SDK 또는 CLI와 UpdateThing 함께 사용

안녕하세요 AWS IoT

다음 코드 예제에서는 AWS IoT의 사용을 시작하는 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
```

```
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
                Console.WriteLine("You can create IoT Things using the IoT Basics
scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
```

```
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 listThings를 참조하세요.

C++

SDK for C++

CMakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS iot)

# Set this project's name.
project("hello_iot")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
```

```
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_iot.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_iot.cpp 소스 파일의 코드입니다.

```
#include <aws/core/Aws.h>
#include <aws/iot/IoTClient.h>
#include <aws/iot/model/ListThingsRequest.h>
#include <iostream>

/*
 *  A "Hello IoT" starter application which initializes an AWS IoT client and
 *  lists the AWS IoT topics in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_iot'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
```

```
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::IoT::IoTClient iotClient(clientConfig);
        // List the things in the current account.
        Aws::IoT::Model::ListThingsRequest listThingsRequest;

        Aws::String nextToken; // Used for pagination.
        Aws::Vector<Aws::IoT::Model::ThingAttribute> allThings;

        do {
            if (!nextToken.empty()) {
                listThingsRequest.SetNextToken(nextToken);
            }

            Aws::IoT::Model::ListThingsOutcome listThingsOutcome =
iotClient.ListThings(
                    listThingsRequest);
            if (listThingsOutcome.IsSuccess()) {
                const Aws::Vector<Aws::IoT::Model::ThingAttribute> &things =
listThingsOutcome.GetResult().GetThings();
                allThings.insert(allThings.end(), things.begin(), things.end());
                nextToken = listThingsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "List things failed"
                          << listThingsOutcome.GetError().GetMessage() <<
std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << allThings.size() << " thing(s) found." << std::endl;
        for (auto const &thing: allThings) {
            std::cout << thing.GetThingName() << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
```

```
    return 0;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 listThings를 참조하세요.

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotClient;
import software.amazon.awssdk.services.iot.model.ListThingsRequest;
import software.amazon.awssdk.services.iot.model.ListThingsResponse;
import software.amazon.awssdk.services.iot.model.ThingAttribute;
import software.amazon.awssdk.services.iot.paginators.ListThingsIterable;

import java.util.List;

public class HelloIoT {
    public static void main(String[] args) {
        System.out.println("Hello AWS IoT. Here is a listing of your AWS IoT
 Things:");
        IotClient iotClient = IotClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllThings(iotClient);
    }
```

```
    public static void listAllThings(IotClient iotClient) {
        iotClient.listThingsPaginator(ListThingsRequest.builder()
                .maxResults(10)
                .build())
            .stream()
            .flatMap(response -> response.things().stream())
            .forEach(attribute -> {
                System.out.println("Thing name: " + attribute.thingName());
                System.out.println("Thing ARN: " + attribute.thingArn());
            });
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 listThings를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.ListThingsRequest

suspend fun main() {
    println("A listing of your AWS IoT Things:")
    listAllThings()
}

suspend fun listAllThings() {
    val thingsRequest =
        ListThingsRequest {
            maxResults = 10
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
```

```
        val response = iotClient.listThings(thingsRequest)
        val thingList = response.things
        if (thingList != null) {
            for (attribute in thingList) {
                println("Thing name ${attribute.thingName}")
                println("Thing ARN: ${attribute.thingArn}")
            }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 listThings를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
def hello_iot():
    """
    Use the AWS SDK for Python (Boto3) to create an AWS IoT client and list
    up to 10 things in your AWS IoT account.
    This example uses the default settings specified in your shared credentials
    and config files.
    """
    try:
        iot_client = boto3.client("iot")
        response = iot_client.list_things(maxResults=10)
        things = response.get("things", [])

        print("Hello, AWS IoT! Here are your things:")
        if things:
            for i, thing in enumerate(things, 1):
                print(f"{i}. {thing['thingName']}")
        else:
            print("No things found in your AWS IoT account.")
```

```
    except ClientError as e:
        if e.response["Error"]["Code"] == "UnauthorizedException":
            print("You don't have permission to access AWS IoT.")
        else:
            print(f"Couldn't access AWS IoT. Error: {e}")
    except NoCredentialsError:
        print("No AWS credentials found. Please configure your credentials.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 listThings를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK를 AWS IoT 사용하여의 기본 사항 알아보기

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- AWS IoT 사물을 생성합니다.
- 디바이스 인증서를 생성합니다.
- 속성을 사용하여 AWS IoT 사물을 업데이트합니다.
- 고유한 엔드포인트를 반환합니다.
- AWS IoT 인증서를 나열합니다.
- 섀도우를 업데이트 AWS IoT 합니다.
- 상태 정보를 씁니다.
- 규칙을 생성합니다.
- 규칙을 나열합니다.
- 사물 이름을 사용하여 사물을 검색합니다.
- AWS IoT 사물을 삭제합니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS IoT 기능을 보여주는 대화형 시나리오를 실행합니다.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/
basics/iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
```

```
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected
 dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";
```

```
        try
        {
            // Step 1: Create an AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in
the AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN
value is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
```

```
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing
the communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for
{thingName}? (y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
```

```
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
                        Console.WriteLine(lines[i]);
                    }
                    if (lines.Length > 5)
                    {
                        Console.WriteLine("...");
                    }

                    Console.WriteLine($"\nCertificate ARN:");
                    Console.WriteLine(certificateArn);

                    // Step 3: Attach the Certificate to the AWS IoT Thing
                    Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                    var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                    if (attachResult)
                    {
                        Console.WriteLine("Certificate attached to Thing
successfully.");
                    }
                    else
                    {
                        Console.WriteLine("Failed to attach certificate to
Thing.");
                    }

                    Console.WriteLine("Thing Details:");
                    Console.WriteLine($"Thing Name: {thingName}");
                    Console.WriteLine($"Thing ARN: {thingArn}");
                }
                else
                {
                    Console.WriteLine("Failed to create certificate.");
                }
            }
            Console.WriteLine(new string('-', 80));

            // Step 4: Update an AWS IoT Thing with Attributes
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
```

```
            Console.WriteLine("IoT Thing attributes, represented as key-value
pairs, offer a pivotal advantage in facilitating efficient data");
            Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var attributes = new Dictionary<string, string>
            {
                { "Location", "Seattle" },
                { "DeviceType", "Sensor" },
                { "Firmware", "1.2.3" }
            };

            await iotWrapper.UpdateThingAsync(thingName, attributes);
            Console.WriteLine("Thing attributes updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 5: Return a unique endpoint specific to the Amazon Web
Services account
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
```

```
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5
certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });
```

```
            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a
rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like
to deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for
default '{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();
```

```
                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS
topic: {snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM
role: {roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for
default '{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation
outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
```

```
                    }
                }
                else
                {
                    Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
            }
        }
        else
        {
            Console.WriteLine("Skipping CloudFormation stack deployment and
rule creation.");
        }
        Console.WriteLine(new string('-', 80));

        // Step 10: List your rules
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("10. List your rules.");
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var rules = await iotWrapper.ListTopicRulesAsync();
        Console.WriteLine("List of IoT Rules:");
        foreach (var rule in rules.Take(5)) // Show first 5 rules
        {
            Console.WriteLine($"Rule Name: {rule.RuleName}");
            Console.WriteLine($"Rule ARN: {rule.RuleArn}");
            Console.WriteLine("--------------");
        }
        Console.WriteLine();
        Console.WriteLine(new string('-', 80));

        // Step 11: Search things using the Thing name
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("11. Search things using the Thing name.");
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
 {searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
 {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the
 certificate for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
 {thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
 certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
```

```
                    Console.ReadLine();
                }

                await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");

                await iotWrapper.DeleteCertificateAsync(certificateId);
                Console.WriteLine($"{certificateArn} was successfully
deleted.");
            }
            Console.WriteLine(new string('-', 80));
        }

        // Step 13: Delete the AWS IoT Thing
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("13. Delete the AWS IoT Thing.");
        var deleteThing = "y";
        if (IsInteractive)
        {
            Console.Write($"Do you want to delete the IoT Thing? (y/n)");
            deleteThing = Console.ReadLine();
        }
        else
        {
            Console.WriteLine($"Deleting IoT Thing {thingName}...");
        }

        if (deleteThing?.ToLower() == "y")
        {
            await iotWrapper.DeleteThingAsync(thingName);
            Console.WriteLine($"Deleted Thing {thingName}");
        }
        Console.WriteLine(new string('-', 80));

        // Step 14: Clean up CloudFormation stack
        if (!string.IsNullOrEmpty(snsTopicArn))
        {
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("14. Clean up CloudFormation stack.");
            Console.WriteLine("Deleting the CloudFormation stack and all
resources...");
```

```
                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
 delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
 iotWrapper.DeleteTopicRuleAsync(ruleName);

                    var stackCleanupSuccess = await
 DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
 stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check
 the logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
 {_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
```

```
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during
CloudFormation stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
```

```
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string>
{ Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await
WaitForStackCompletion(response.StackId, cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created
successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation
failed: {stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
```

```
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}'
already exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
```

```
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack
creation to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };
```

```
            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being
deleted. This may take a few minutes.");
```

```
            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };
```

```
            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to
be deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
```

```
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press
 Enter for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-
a-zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

AWS IoT SDK 메서드의 래퍼 클래스입니다.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
```

```
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</
returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
```

```
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or
 null if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
 CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await
 _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
 {response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
 response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
```

```
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
```

```
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
```

```
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing
failed.</returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
```

```
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
 shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
 MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
 {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</
returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
```

```
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</
param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
 snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
```

```
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
```

```
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT
rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
```

```
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count}
Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting
for the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready
(attempt {attempt}/{maxRetries})...");
```

```
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if
it's ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if
(configResponse.ThingIndexingConfiguration?.ThingIndexingMode ==
ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await
_amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready
after configuration update");
            return new List<ThingDocument>();
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
 why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's
 why: {ex.Message}");
            return false;
        }
    }
```

```
    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
```

```
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);
```

```
            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
 next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS IoT 사물을 생성합니다.

```
    Aws::String thingName = askQuestion("Enter a thing name: ");

    if (!createThing(thingName, clientConfiguration)) {
        std::cerr << "Exiting because createThing failed." << std::endl;
        cleanup("", "", "", "", "", false, clientConfiguration);
        return false;
    }
```

```
//! Create an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
```

```
    }

    return outcome.IsSuccess();
}
```

디바이스 인증서를 생성합니다.

```
    Aws::String certificateARN;
    Aws::String certificateID;
    if (askYesNoQuestion("Would you like to create a certificate for your thing?
(y/n) ")) {
        Aws::String outputFolder;
        if (askYesNoQuestion(
                "Would you like to save the certificate and keys to file? (y/n)
")) {
            outputFolder = std::filesystem::current_path();
            outputFolder += "/device_keys_and_certificates";

            std::filesystem::create_directories(outputFolder);

            std::cout << "The certificate and keys will be saved to the folder: "
                      << outputFolder << std::endl;
        }

        if (!createKeysAndCertificate(outputFolder, certificateARN,
certificateID,
                                      clientConfiguration)) {
            std::cerr << "Exiting because createKeysAndCertificate failed."
                      << std::endl;
            cleanup(thingName, "", "", "", "", false, clientConfiguration);
            return false;
        }

        std::cout << "\nNext, the certificate will be attached to the thing.\n"
                  << std::endl;
        if (!attachThingPrincipal(certificateARN, thingName,
clientConfiguration)) {
            std::cerr << "Exiting because attachThingPrincipal failed." <<
std::endl;
            cleanup(thingName, certificateARN, certificateID, "", "",
                    false,
                    clientConfiguration);
```

```
            return false;
        }
    }
```

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;
```

```
            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }
```

```
    return outcome.IsSuccess();
}

//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
    else {
        std::cerr << "Failed to attach principal to thing." <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

AWS IoT 사물에 대해 다양한 작업을 수행합니다.

```
    if (!updateThing(thingName, { {"location", "Office"}, {"firmwareVersion",
 "v2.0"} }, clientConfiguration)) {
        std::cerr << "Exiting because updateThing failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }
```

```
   printAsterisksLine();

   std::cout << "Now an endpoint will be retrieved for your account.\n" <<
std::endl;
   std::cout << "An IoT Endpoint refers to a specific URL or Uniform Resource
Locator that serves as the entry point\n"
   << "for communication between IoT devices and the AWS IoT service." <<
std::endl;

   askQuestion("Press Enter to continue:", alwaysTrueTest);

   Aws::String endpoint;
   if (!describeEndpoint(endpoint, clientConfiguration)) {
       std::cerr << "Exiting because getEndpoint failed." << std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }
   std::cout <<"Your endpoint is " << endpoint << "." << std::endl;
   printAsterisksLine();

   std::cout << "Now the certificates in your account will be listed." <<
std::endl;
   askQuestion("Press Enter to continue:", alwaysTrueTest);

   if (!listCertificates(clientConfiguration)) {
       std::cerr << "Exiting because listCertificates failed." << std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }

   printAsterisksLine();

   std::cout << "Now the shadow for the thing will be updated.\n" << std::endl;
   std::cout << "A thing shadow refers to a feature that enables you to create a
virtual representation, or \"shadow,\"\n"
   << "of a physical device or thing. The thing shadow allows you to synchronize
and control the state of a device between\n"
   << "the cloud and the device itself. and the AWS IoT service. For example,
you can write and retrieve JSON data from a thing shadow." << std::endl;
   askQuestion("Press Enter to continue:", alwaysTrueTest);
```

```
    if (!updateThingShadow(thingName, R"({"state":{"reported":
{"temperature":25,"humidity":50}}})", clientConfiguration)) {
        std::cerr << "Exiting because updateThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now, the state information for the shadow will be retrieved.\n"
 << std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    Aws::String shadowState;
    if (!getThingShadow(thingName, shadowState, clientConfiguration)) {
        std::cerr << "Exiting because getThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }
    std::cout << "The retrieved shadow state is: " << shadowState << std::endl;

    printAsterisksLine();

    std::cout << "A rule with now be added to to the thing.\n" << std::endl;
    std::cout << "Any user who has permission to create rules will be able to
 access data processed by the rule." << std::endl;
    std::cout << "In this case, the rule will use an Simple Notification Service
(SNS) topic and an IAM rule." << std::endl;
    std::cout << "These resources will be created using a CloudFormation
 template." << std::endl;
    std::cout << "Stack creation may take a few minutes." << std::endl;

    askQuestion("Press Enter to continue: ", alwaysTrueTest);
    Aws::Map<Aws::String, Aws::String> outputs
 =createCloudFormationStack(STACK_NAME,clientConfiguration);
    if (outputs.empty()) {
        std::cerr << "Exiting because createCloudFormationStack failed." <<
 std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }
```

```
    // Retrieve the topic ARN and role ARN from the CloudFormation stack outputs.
    auto topicArnIter = outputs.find(SNS_TOPIC_ARN_OUTPUT);
    auto roleArnIter = outputs.find(ROLE_ARN_OUTPUT);
    if ((topicArnIter == outputs.end()) || (roleArnIter == outputs.end())) {
        std::cerr << "Exiting because output '" << SNS_TOPIC_ARN_OUTPUT <<
        "' or '" << ROLE_ARN_OUTPUT << "'not found in the CloudFormation stack."
 << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
                false,
                clientConfiguration);
        return false;
    }

    Aws::String topicArn = topicArnIter->second;
    Aws::String roleArn = roleArnIter->second;
    Aws::String sqlStatement = "SELECT * FROM '";
    sqlStatement += MQTT_MESSAGE_TOPIC_FILTER;
    sqlStatement += "'";

    printAsterisksLine();

    std::cout << "Now a rule will be created.\n" << std::endl;
    std::cout << "Rules are an administrator-level action. Any user who has
 permission\n"
                 << "to create rules will be able to access data processed by the
 rule." << std::endl;
    std::cout << "In this case, the rule will use an SNS topic" << std::endl;
    std::cout << "and the following SQL statement '" << sqlStatement << "'." <<
 std::endl;
    std::cout << "For more information on IoT SQL, see https://
docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html" <<
 std::endl;
    Aws::String ruleName = askQuestion("Enter a rule name: ");
    if (!createTopicRule(ruleName, topicArn, sqlStatement, roleArn,
 clientConfiguration)) {
        std::cerr << "Exiting because createRule failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();
```

```
    std::cout << "Now your rules will be listed.\n" << std::endl;
    askQuestion("Press Enter to continue: ", alwaysTrueTest);
    if (!listTopicRules(clientConfiguration)) {
        std::cerr << "Exiting because listRules failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();
    Aws::String queryString = "thingName:" + thingName;
    std::cout << "Now the AWS IoT fleet index will be queried with the query\n'"
    << queryString << "'.\n" << std::endl;
    std::cout << "For query information, see https://docs.aws.amazon.com/iot/
latest/developerguide/query-syntax.html" << std::endl;

    std::cout << "For this query to work, thing indexing must be enabled in your
 account.\n"
    << "This can be done with the awscli command line by calling 'aws iot update-
indexing-configuration'\n"
       << "or it can be done programmatically." << std::endl;
    std::cout << "For more information, see https://docs.aws.amazon.com/iot/
latest/developerguide/managing-index.html" << std::endl;
    if (askYesNoQuestion("Do you want to enable thing indexing in your account?
 (y/n) "))
    {
        Aws::IoT::Model::ThingIndexingConfiguration thingIndexingConfiguration;

 thingIndexingConfiguration.SetThingIndexingMode(Aws::IoT::Model::ThingIndexingMode::REGI

 thingIndexingConfiguration.SetThingConnectivityIndexingMode(Aws::IoT::Model::ThingConnec
        // The ThingGroupIndexingConfiguration object is ignored if not set.
        Aws::IoT::Model::ThingGroupIndexingConfiguration
 thingGroupIndexingConfiguration;
        if (!updateIndexingConfiguration(thingIndexingConfiguration,
 thingGroupIndexingConfiguration, clientConfiguration)) {
            std::cerr << "Exiting because updateIndexingConfiguration failed." <<
 std::endl;
            cleanup(thingName, certificateARN, certificateID, STACK_NAME,
                    ruleName, false,
                    clientConfiguration);
            return false;
```

```
        }
    }

    if (!searchIndex(queryString, clientConfiguration)) {

        std::cerr << "Exiting because searchIndex failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
    }
```

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }
```

```
    return outcome.IsSuccess();
}

//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}

//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;
```

```
    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}

//! Update the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param document: The state information, in JSON format.
```

```
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThingShadow(const Aws::String &thingName,
                                    const Aws::String &document,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient
 iotDataPlaneClient(clientConfiguration);
    Aws::IoTDataPlane::Model::UpdateThingShadowRequest updateThingShadowRequest;
    updateThingShadowRequest.SetThingName(thingName);
    std::shared_ptr<std::stringstream> streamBuf =
 std::make_shared<std::stringstream>(
            document);
    updateThingShadowRequest.SetBody(streamBuf);
    Aws::IoTDataPlane::Model::UpdateThingShadowOutcome outcome =
 iotDataPlaneClient.UpdateThingShadow(
            updateThingShadowRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing shadow." << std::endl;
    }
    else {
        std::cerr << "Error while updating thing shadow."
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Get the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param documentResult: String to receive the state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::getThingShadow(const Aws::String &thingName,
                                 Aws::String &documentResult,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient iotClient(clientConfiguration);
    Aws::IoTDataPlane::Model::GetThingShadowRequest request;
    request.SetThingName(thingName);
    auto outcome = iotClient.GetThingShadow(request);
```

```
    if (outcome.IsSuccess()) {
        std::stringstream ss;
        ss << outcome.GetResult().GetPayload().rdbuf();
        documentResult = ss.str();
    }
    else {
        std::cerr << "Error getting thing shadow: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
                             const Aws::String &snsTopicARN, const Aws::String
 &sql,
                             const Aws::String &roleARN,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});
```

```
    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}

//! Lists the AWS IoT topic rules.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listTopicRules(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListTopicRulesRequest request;

    Aws::Vector<Aws::IoT::Model::TopicRuleListItem> allRules;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::ListTopicRulesOutcome outcome =
 iotClient.ListTopicRules(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListTopicRulesResult &result =
 outcome.GetResult();
            allRules.insert(allRules.end(),
                            result.GetRules().cbegin(),
                            result.GetRules().cend());

            nextToken = result.GetNextToken();
        }
```

```
        else {
            std::cerr << "ListTopicRules error: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());

    std::cout << "ListTopicRules: " << allRules.size() << " rule(s) found."
              << std::endl;
    for (auto &rule: allRules) {
        std::cout << "  Rule name: " << rule.GetRuleName() << ", rule ARN: "
                  << rule.GetRuleArn() << "." << std::endl;
    }

    return true;
}

//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);
```

```
        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
}
```

리소스를 정리합니다.

```
bool
AwsDoc::IoT::cleanup(const Aws::String &thingName, const Aws::String
 &certificateARN,
                     const Aws::String &certificateID, const Aws::String
 &stackName,
                     const Aws::String &ruleName, bool askForConfirmation,
                     const Aws::Client::ClientConfiguration &clientConfiguration)
 {
    bool result = true;

    if (!ruleName.empty() && (!askForConfirmation ||
                              askYesNoQuestion("Delete the rule '" + ruleName +
                                               "'? (y/n) "))) {
        result &= deleteTopicRule(ruleName, clientConfiguration);
```

```
    }

    Aws::CloudFormation::CloudFormationClient
 cloudFormationClient(clientConfiguration);

    if (!stackName.empty() && (!askForConfirmation ||
                               askYesNoQuestion(
                                       "Delete the CloudFormation stack '" +
 stackName +
                                       "'? (y/n) "))) {
        result &= deleteStack(stackName, clientConfiguration);
    }

    if (!certificateARN.empty() && (!askForConfirmation ||
                                    askYesNoQuestion("Delete the certificate '" +
                                                     certificateARN + "'? (y/n)
 "))) {
        result &= detachThingPrincipal(certificateARN, thingName,
 clientConfiguration);
        result &= deleteCertificate(certificateID, clientConfiguration);
    }

    if (!thingName.empty() && (!askForConfirmation ||
                               askYesNoQuestion("Delete the thing '" + thingName
 +
                                                "'? (y/n) "))) {
        result &= deleteThing(thingName, clientConfiguration);
    }

    return result;
}
```

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
```

```
                                        const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);
```

```
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
```

```
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

AWS IoT 기능을 보여주는 대화형 시나리오를 실행합니다.

```
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
```

```
 * This Java example performs these tasks:
 *
 * 1. Creates an AWS IoT Thing.
 * 2. Generate and attach a device certificate.
 * 3. Update an AWS IoT Thing with Attributes.
 * 4. Get an AWS IoT Endpoint.
 * 5. List your certificates.
 * 6. Updates the shadow for the specified thing..
 * 7. Write out the state information, in JSON format
 * 8. Creates a rule
 * 9. List rules
 * 10. Search things
 * 11. Detach amd delete the certificate.
 * 12. Delete Thing.
 */
public class IotScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <roleARN> <snsAction>

                Where:
                    roleARN - The ARN of an IAM role that has permission to work
 with AWS IOT.
                    snsAction  - An ARN of an SNS topic.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        IotActions iotActions = new IotActions();
        String thingName;
        String ruleName;
        String roleARN = args[0];
        String snsAction = args[1];
        Scanner scanner = new Scanner(System.in);

        System.out.println(DASHES);
```

```
        System.out.println("Welcome to the AWS IoT basics scenario.");
        System.out.println("""
            This example program demonstrates various interactions with the AWS
Internet of Things (IoT) Core service. The program guides you through a series
of steps,
            including creating an IoT Thing, generating a device certificate,
updating the Thing with attributes, and so on.
            It utilizes the AWS SDK for Java V2 and incorporates functionality
for creating and managing IoT Things, certificates, rules,
            shadows, and performing searches. The program aims to showcase AWS
IoT capabilities and provides a comprehensive example for
            developers working with AWS IoT in a Java environment.

            Let's get started...

            """);
        System.out.println(DASHES);

        System.out.println("1. Create an AWS IoT Thing.");
        System.out.println("""
            An AWS IoT Thing represents a virtual entity in the AWS IoT service
that can be associated with
            a physical device.
            """);
        // Prompt the user for input.
        System.out.print("Enter Thing name: ");
        thingName = scanner.nextLine();
        iotActions.createIoTThing(thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Generate a device certificate.");
        System.out.println("""
            A device certificate performs a role in securing the communication
between devices (Things)
            and the AWS IoT platform.
            """);

        System.out.print("Do you want to create a certificate for " +thingName
+"? (y/n)");
        String certAns = scanner.nextLine();
        String certificateArn="" ;
        if (certAns != null && certAns.trim().equalsIgnoreCase("y")) {
            certificateArn = iotActions.createCertificate();
```

```
            System.out.println("Attach the certificate to the AWS IoT Thing.");
            iotActions.attachCertificateToThing(thingName, certificateArn);
        } else {
            System.out.println("A device certificate was not created.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Update an AWS IoT Thing with Attributes.");
        System.out.println("""
             IoT Thing attributes, represented as key-value pairs, offer a
pivotal advantage in facilitating efficient data
             management and retrieval within the AWS IoT ecosystem.
            """);
        waitForInputToContinue(scanner);
        iotActions.updateShadowThing(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Return a unique endpoint specific to the Amazon
Web Services account.");
        System.out.println("""
            An IoT Endpoint refers to a specific URL or Uniform Resource Locator
that serves as the entry point for communication between IoT devices and the AWS
IoT service.
           """);
        waitForInputToContinue(scanner);
        String endpointUrl = iotActions.describeEndpoint();
        System.out.println("The endpoint is "+endpointUrl);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. List your AWS IoT certificates");
        waitForInputToContinue(scanner);
        if (certificateArn.length() > 0) {
            iotActions.listCertificates();
        } else {
            System.out.println("You did not create a certificates. Skipping this
step.");
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("6. Create an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
        System.out.println("""
            A Thing Shadow refers to a feature that enables you to create a
virtual representation, or "shadow,"
            of a physical device or thing. The Thing Shadow allows you to
synchronize and control the state of a device between
            the cloud and the device itself. and the AWS IoT service. For
example, you can write and retrieve JSON data from a Thing Shadow.
           """);
        waitForInputToContinue(scanner);
        iotActions.updateShadowThing(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Write out the state information, in JSON
format.");
        waitForInputToContinue(scanner);
        iotActions.getPayload(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Creates a rule");
        System.out.println("""
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """);
        System.out.print("Enter Rule name: ");
        ruleName = scanner.nextLine();
        iotActions.createIoTRule(roleARN, ruleName, snsAction);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. List your rules.");
        waitForInputToContinue(scanner);
        iotActions.listIoTRules();
        waitForInputToContinue(scanner);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("10. Search things using the Thing name.");
        waitForInputToContinue(scanner);
        String queryString = "thingName:"+thingName ;
        iotActions.searchThings(queryString);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        if (certificateArn.length() > 0) {
            System.out.print("Do you want to detach and delete the certificate
for " +thingName +"? (y/n)");
            String delAns = scanner.nextLine();
            if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
                System.out.println("11. You selected to detach amd delete the
certificate.");
                waitForInputToContinue(scanner);
                iotActions.detachThingPrincipal(thingName, certificateArn);
                iotActions.deleteCertificate(certificateArn);
                waitForInputToContinue(scanner);
            } else {
                System.out.println("11. You selected not to delete the
certificate.");
            }
        } else {
            System.out.println("11. You did not create a certificate so there is
nothing to delete.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Delete the AWS IoT Thing.");
        System.out.print("Do you want to delete the IoT Thing? (y/n)");
        String delAns = scanner.nextLine();
        if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
            iotActions.deleteIoTThing(thingName);
        } else {
            System.out.println("The IoT Thing was not deleted.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS IoT workflow has successfully completed.");
```

```
        System.out.println(DASHES);
    }


    private static void waitForInputToContinue(Scanner scanner) {
        while (true) {
            System.out.println("");
            System.out.println("Enter 'c' followed by <ENTER> to continue:");
            String input = scanner.nextLine();

            if (input.trim().equalsIgnoreCase("c")) {
                System.out.println("Continuing with the program...");
                System.out.println("");
                break;
            } else {
                // Handle invalid input.
                System.out.println("Invalid input. Please try again.");
            }
        }
    }
}
```

AWS IoT SDK 메서드의 래퍼 클래스입니다.

```
import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.core.retry.RetryPolicy;
import software.amazon.awssdk.http.async.SdkAsyncHttpClient;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotAsyncClient;
import software.amazon.awssdk.services.iot.model.Action;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.Certificate;
import
 software.amazon.awssdk.services.iot.model.CreateKeysAndCertificateResponse;
import software.amazon.awssdk.services.iot.model.CreateThingRequest;
import software.amazon.awssdk.services.iot.model.CreateThingResponse;
import software.amazon.awssdk.services.iot.model.CreateTopicRuleRequest;
```

```
import software.amazon.awssdk.services.iot.model.CreateTopicRuleResponse;
import software.amazon.awssdk.services.iot.model.DeleteCertificateRequest;
import software.amazon.awssdk.services.iot.model.DeleteCertificateResponse;
import software.amazon.awssdk.services.iot.model.DeleteThingRequest;
import software.amazon.awssdk.services.iot.model.DeleteThingResponse;
import software.amazon.awssdk.services.iot.model.DescribeEndpointRequest;
import software.amazon.awssdk.services.iot.model.DescribeEndpointResponse;
import software.amazon.awssdk.services.iot.model.DescribeThingRequest;
import software.amazon.awssdk.services.iot.model.DescribeThingResponse;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.IotException;
import software.amazon.awssdk.services.iot.model.ListCertificatesResponse;
import software.amazon.awssdk.services.iot.model.ListTopicRulesRequest;
import software.amazon.awssdk.services.iot.model.ListTopicRulesResponse;
import software.amazon.awssdk.services.iot.model.SearchIndexRequest;
import software.amazon.awssdk.services.iot.model.SearchIndexResponse;
import software.amazon.awssdk.services.iot.model.TopicRuleListItem;
import software.amazon.awssdk.services.iot.model.SnsAction;
import software.amazon.awssdk.services.iot.model.TopicRulePayload;
import software.amazon.awssdk.services.iotdataplane.IotDataPlaneAsyncClient;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowRequest;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowResponse;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowRequest;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowResponse;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.util.List;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class IotActions {

    private static IotAsyncClient iotAsyncClient;

    private static IotDataPlaneAsyncClient iotAsyncDataPlaneClient;

    private static final String TOPIC = "your-iot-topic";

    private static IotDataPlaneAsyncClient getAsyncDataPlaneClient() {
```

```
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(100)
            .connectionTimeout(Duration.ofSeconds(60))
            .readTimeout(Duration.ofSeconds(60))
            .writeTimeout(Duration.ofSeconds(60))
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))
            .apiCallAttemptTimeout(Duration.ofSeconds(90))
            .retryPolicy(RetryPolicy.builder()
                .numRetries(3)
                .build())
            .build();

        if (iotAsyncDataPlaneClient == null) {
            iotAsyncDataPlaneClient = IotDataPlaneAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return iotAsyncDataPlaneClient;
    }


    private static IotAsyncClient getAsyncClient() {
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(100)
            .connectionTimeout(Duration.ofSeconds(60))
            .readTimeout(Duration.ofSeconds(60))
            .writeTimeout(Duration.ofSeconds(60))
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))
            .apiCallAttemptTimeout(Duration.ofSeconds(90))
            .retryPolicy(RetryPolicy.builder()
                .numRetries(3)
                .build())
            .build();
```

```
        if (iotAsyncClient == null) {
            iotAsyncClient = IotAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return iotAsyncClient;
    }

    /**
     * Creates an IoT certificate asynchronously.
     *
     * @return The ARN of the created certificate.
     * <p>
     * This method initiates an asynchronous request to create an IoT
certificate.
     * If the request is successful, it prints the certificate details and
returns the certificate ARN.
     * If an exception occurs, it prints the error message.
     */
    public String createCertificate() {
        CompletableFuture<CreateKeysAndCertificateResponse> future =
getAsyncClient().createKeysAndCertificate();
        final String[] certificateArn = {null};
        future.whenComplete((response, ex) -> {
            if (response != null) {
                String certificatePem = response.certificatePem();
                certificateArn[0] = response.certificateArn();

                // Print the details.
                System.out.println("\nCertificate:");
                System.out.println(certificatePem);
                System.out.println("\nCertificate ARN:");
                System.out.println(certificateArn[0]);

            } else {
                Throwable cause = (ex instanceof CompletionException) ?
ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
```

```
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
        return certificateArn[0];
    }

    /**
     * Creates an IoT Thing with the specified name asynchronously.
     *
     * @param thingName The name of the IoT Thing to create.
     *
     * This method initiates an asynchronous request to create an IoT Thing with
the specified name.
     * If the request is successful, it prints the name of the thing and its ARN
value.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTThing(String thingName) {
        CreateThingRequest createThingRequest = CreateThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<CreateThingResponse> future =
getAsyncClient().createThing(createThingRequest);
        future.whenComplete((createThingResponse, ex) -> {
            if (createThingResponse != null) {
                System.out.println(thingName + " was successfully created. The
ARN value is " + createThingResponse.thingArn());
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });
```

```
        future.join();
    }

    /**
     * Attaches a certificate to an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to attach.
     *
     * This method initiates an asynchronous request to attach a certificate to
an IoT Thing.
     * If the request is successful, it prints a confirmation message and
additional information about the Thing.
     * If an exception occurs, it prints the error message.
     */
    public void attachCertificateToThing(String thingName, String certificateArn)
{
        AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
            .thingName(thingName)
            .principal(certificateArn)
            .build();

        CompletableFuture<AttachThingPrincipalResponse> future =
getAsyncClient().attachThingPrincipal(principalRequest);
        future.whenComplete((attachResponse, ex) -> {
            if (attachResponse != null &&
attachResponse.sdkHttpResponse().isSuccessful()) {
                System.out.println("Certificate attached to Thing
successfully.");

                // Print additional information about the Thing.
                describeThing(thingName);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to attach certificate to Thing.
HTTP Status Code: " +
```

```
                        attachResponse.sdkHttpResponse().statusCode());
                }
            }
        });

        future.join();
    }

    /**
     * Describes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to describe an IoT Thing.
     * If the request is successful, it prints the Thing details.
     * If an exception occurs, it prints the error message.
     */
    private void describeThing(String thingName) {
        DescribeThingRequest thingRequest = DescribeThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DescribeThingResponse> future =
getAsyncClient().describeThing(thingRequest);
        future.whenComplete((describeResponse, ex) -> {
            if (describeResponse != null) {
                System.out.println("Thing Details:");
                System.out.println("Thing Name: " +
describeResponse.thingName());
                System.out.println("Thing ARN: " + describeResponse.thingArn());
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to describe Thing.");
                }
            }
        });
```

```
        future.join();
    }

    /**
     * Updates the shadow of an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to update the shadow of an
IoT Thing.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void updateShadowThing(String thingName) {
        // Create Thing Shadow State Document.
        String stateDocument = "{\"state\":{\"reported\":{\"temperature\":25,
\"humidity\":50}}}";
        SdkBytes data = SdkBytes.fromString(stateDocument,
StandardCharsets.UTF_8);
        UpdateThingShadowRequest updateThingShadowRequest =
UpdateThingShadowRequest.builder()
            .thingName(thingName)
            .payload(data)
            .build();

        CompletableFuture<UpdateThingShadowResponse> future =
getAsyncDataPlaneClient().updateThingShadow(updateThingShadowRequest);
        future.whenComplete((updateResponse, ex) -> {
            if (updateResponse != null) {
                System.out.println("Thing Shadow updated successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to update Thing Shadow.");
                }
            }
        });
```

```
        future.join();
    }

    /**
     * Describes the endpoint of the IoT service asynchronously.
     *
     * @return A CompletableFuture containing the full endpoint URL.
     *
     * This method initiates an asynchronous request to describe the endpoint of
 the IoT service.
     * If the request is successful, it prints and returns the full endpoint URL.
     * If an exception occurs, it prints the error message.
     */
    public String describeEndpoint() {
        CompletableFuture<DescribeEndpointResponse> future =
 getAsyncClient().describeEndpoint(DescribeEndpointRequest.builder().endpointType("iot:Da
ATS").build());
        final String[] result = {null};

        future.whenComplete((endpointResponse, ex) -> {
            if (endpointResponse != null) {
                String endpointUrl = endpointResponse.endpointAddress();
                String exString = getValue(endpointUrl);
                String fullEndpoint = "https://" + exString + "-ats.iot.us-
east-1.amazonaws.com";

                System.out.println("Full Endpoint URL: " + fullEndpoint);
                result[0] = fullEndpoint;
            } else {
                Throwable cause = (ex instanceof CompletionException) ?
 ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return result[0];
    }
```

```
    /**
     * Extracts a specific value from the endpoint URL.
     *
     * @param input The endpoint URL to process.
     * @return The extracted value from the endpoint URL.
     */
    private static String getValue(String input) {
        // Define a regular expression pattern for extracting the subdomain.
        Pattern pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\
\.com");

        // Match the pattern against the input string.
        Matcher matcher = pattern.matcher(input);

        // Check if a match is found.
        if (matcher.find()) {
            // Extract the subdomain from the first capturing group.
            String subdomain = matcher.group(1);
            System.out.println("Extracted subdomain: " + subdomain);
            return subdomain ;
        } else {
            System.out.println("No match found");
        }
        return "" ;
    }

    /**
     * Lists all certificates asynchronously.
     *
     * This method initiates an asynchronous request to list all certificates.
     * If the request is successful, it prints the certificate IDs and ARNs.
     * If an exception occurs, it prints the error message.
     */
    public void listCertificates() {
        CompletableFuture<ListCertificatesResponse> future =
 getAsyncClient().listCertificates();
        future.whenComplete((response, ex) -> {
            if (response != null) {
                List<Certificate> certList = response.certificates();
                for (Certificate cert : certList) {
                    System.out.println("Cert id: " + cert.certificateId());
                    System.out.println("Cert Arn: " + cert.certificateArn());
                }
```

```
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list certificates.");
                }
            }
        });

        future.join();
    }

    /**
     * Retrieves the payload of a Thing's shadow asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to get the payload of a
Thing's shadow.
     * If the request is successful, it prints the shadow data.
     * If an exception occurs, it prints the error message.
     */
    public void getPayload(String thingName) {
        GetThingShadowRequest getThingShadowRequest =
GetThingShadowRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<GetThingShadowResponse> future =
getAsyncDataPlaneClient().getThingShadow(getThingShadowRequest);
        future.whenComplete((getThingShadowResponse, ex) -> {
            if (getThingShadowResponse != null) {
                // Extracting payload from response.
                SdkBytes payload = getThingShadowResponse.payload();
                String payloadString = payload.asUtf8String();
                System.out.println("Received Shadow Data: " + payloadString);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
```

```
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to get Thing Shadow payload.");
                }
            }
        });

        future.join();
    }

    /**
     * Creates an IoT rule asynchronously.
     *
     * @param roleARN The ARN of the IAM role that grants access to the rule's
actions.
     * @param ruleName The name of the IoT rule.
     * @param action The ARN of the action to perform when the rule is triggered.
     *
     * This method initiates an asynchronous request to create an IoT rule.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTRule(String roleARN, String ruleName, String action) {
        String sql = "SELECT * FROM '" + TOPIC + "'";
        SnsAction action1 = SnsAction.builder()
            .targetArn(action)
            .roleArn(roleARN)
            .build();

        // Create the action.
        Action myAction = Action.builder()
            .sns(action1)
            .build();

        // Create the topic rule payload.
        TopicRulePayload topicRulePayload = TopicRulePayload.builder()
            .sql(sql)
            .actions(myAction)
            .build();
```

```
        // Create the topic rule request.
        CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
            .ruleName(ruleName)
            .topicRulePayload(topicRulePayload)
            .build();

        CompletableFuture<CreateTopicRuleResponse> future =
getAsyncClient().createTopicRule(topicRuleRequest);
        future.whenComplete((response, ex) -> {
            if (response != null) {
                System.out.println("IoT Rule created successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to create IoT Rule.");
                }
            }
        });

        future.join();
    }

    /**
     * Lists IoT rules asynchronously.
     *
     * This method initiates an asynchronous request to list IoT rules.
     * If the request is successful, it prints the names and ARNs of the rules.
     * If an exception occurs, it prints the error message.
     */
    public void listIoTRules() {
        ListTopicRulesRequest listTopicRulesRequest =
ListTopicRulesRequest.builder().build();
        CompletableFuture<ListTopicRulesResponse> future =
getAsyncClient().listTopicRules(listTopicRulesRequest);
        future.whenComplete((listTopicRulesResponse, ex) -> {
            if (listTopicRulesResponse != null) {
                System.out.println("List of IoT Rules:");
```

```
                List<TopicRuleListItem> ruleList =
listTopicRulesResponse.rules();
                for (TopicRuleListItem rule : ruleList) {
                    System.out.println("Rule Name: " + rule.ruleName());
                    System.out.println("Rule ARN: " + rule.ruleArn());
                    System.out.println("--------------");
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list IoT Rules.");
                }
            }
        });

        future.join();
    }

    /**
     * Searches for IoT Things asynchronously based on a query string.
     *
     * @param queryString The query string to search for Things.
     *
     * This method initiates an asynchronous request to search for IoT Things.
     * If the request is successful and Things are found, it prints their IDs.
     * If no Things are found, it prints a message indicating so.
     * If an exception occurs, it prints the error message.
     */
    public void searchThings(String queryString) {
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();

        CompletableFuture<SearchIndexResponse> future =
getAsyncClient().searchIndex(searchIndexRequest);
        future.whenComplete((searchIndexResponse, ex) -> {
            if (searchIndexResponse != null) {
                // Process the result.
```

```
                if (searchIndexResponse.things().isEmpty()) {
                    System.out.println("No things found.");
                } else {
                    searchIndexResponse.things().forEach(thing ->
System.out.println("Thing id found using search is " + thing.thingId()));
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to search for IoT Things.");
                }
            }
        });

        future.join();
    }

    /**
     * Detaches a principal (certificate) from an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to detach.
     *
     * This method initiates an asynchronous request to detach a certificate from
an IoT Thing.
     * If the detachment is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void detachThingPrincipal(String thingName, String certificateArn) {
        DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
            .principal(certificateArn)
            .thingName(thingName)
            .build();

        CompletableFuture<DetachThingPrincipalResponse> future =
getAsyncClient().detachThingPrincipal(thingPrincipalRequest);
        future.whenComplete((voidResult, ex) -> {
```

```
            if (ex == null) {
                System.out.println(certificateArn + " was successfully removed
from " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Deletes a certificate asynchronously.
     *
     * @param certificateArn The ARN of the certificate to delete.
     *
     * This method initiates an asynchronous request to delete a certificate.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteCertificate(String certificateArn) {
        DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
            .build();

        CompletableFuture<DeleteCertificateResponse> future =
getAsyncClient().deleteCertificate(certificateProviderRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully
deleted.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
```

```
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Deletes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing to delete.
     *
     * This method initiates an asynchronous request to delete an IoT Thing.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteIoTThing(String thingName) {
        DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DeleteThingResponse> future =
getAsyncClient().deleteThing(deleteThingRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println("Deleted Thing " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    // Get the cert Id  from the Cert ARN value.
    private String extractCertificateId(String certificateArn) {
        // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
```

```
        String[] arnParts = certificateArn.split(":");
        String certificateIdPart = arnParts[arnParts.length - 1];
        return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") +
 1);
    }
}
```

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.Action
import aws.sdk.kotlin.services.iot.model.AttachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.AttributePayload
import aws.sdk.kotlin.services.iot.model.CreateThingRequest
import aws.sdk.kotlin.services.iot.model.CreateTopicRuleRequest
import aws.sdk.kotlin.services.iot.model.DeleteCertificateRequest
import aws.sdk.kotlin.services.iot.model.DeleteThingRequest
import aws.sdk.kotlin.services.iot.model.DescribeEndpointRequest
import aws.sdk.kotlin.services.iot.model.DescribeThingRequest
import aws.sdk.kotlin.services.iot.model.DetachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.ListTopicRulesRequest
import aws.sdk.kotlin.services.iot.model.SearchIndexRequest
import aws.sdk.kotlin.services.iot.model.SnsAction
import aws.sdk.kotlin.services.iot.model.TopicRulePayload
import aws.sdk.kotlin.services.iot.model.UpdateThingRequest
import aws.sdk.kotlin.services.iotdataplane.IotDataPlaneClient
import aws.sdk.kotlin.services.iotdataplane.model.GetThingShadowRequest
import aws.sdk.kotlin.services.iotdataplane.model.UpdateThingShadowRequest
import aws.smithy.kotlin.runtime.content.ByteStream
import aws.smithy.kotlin.runtime.content.toByteArray
import java.util.Scanner
import java.util.regex.Pattern
```

```
import kotlin.system.exitProcess

/**
 * Before running this Kotlin code example, ensure that your development
 environment
 * is set up, including configuring your credentials.
 *
 * For detailed instructions, refer to the following documentation topic:
 * [Setting Up Your Development Environment](https://docs.aws.amazon.com/sdk-for-
kotlin/latest/developer-guide/setup.html)
 *
 * This code example requires an SNS topic and an IAM Role.
 * Follow the steps in the documentation to set up these resources:
 *
 * - [Creating an SNS Topic](https://docs.aws.amazon.com/sns/latest/dg/sns-
getting-started.html#step-create-topic)
 * - [Creating an IAM Role](https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html)
 */

val DASHES = String(CharArray(80)).replace("\u0000", "-")
val TOPIC = "your-iot-topic"

suspend fun main(args: Array<String>) {
    val usage =
        """
        Usage:
            <roleARN> <snsAction>

        Where:
            roleARN - The ARN of an IAM role that has permission to work with AWS
 IOT.
            snsAction  - An ARN of an SNS topic.

        """.trimIndent()

    if (args.size != 2) {
        println(usage)
        exitProcess(1)
    }

    var thingName: String
    val roleARN = args[0]
    val snsAction = args[1]
```

```
    val scanner = Scanner(System.`in`)

    println(DASHES)
    println("Welcome to the AWS IoT example scenario.")
    println(
        """
        This example program demonstrates various interactions with the AWS
Internet of Things (IoT) Core service.
        The program guides you through a series of steps, including creating an
IoT thing, generating a device certificate,
        updating the thing with attributes, and so on.

        It utilizes the AWS SDK for Kotlin and incorporates functionality for
creating and managing IoT things, certificates, rules,
        shadows, and performing searches. The program aims to showcase AWS IoT
capabilities and provides a comprehensive example for
        developers working with AWS IoT in a Kotlin environment.
        """.trimIndent(),
    )

    print("Press Enter to continue...")
    scanner.nextLine()
    println(DASHES)

    println(DASHES)
    println("1. Create an AWS IoT thing.")
    println(
        """
        An AWS IoT thing represents a virtual entity in the AWS IoT service that
can be associated with a physical device.
        """.trimIndent(),
    )
    // Prompt the user for input.
    print("Enter thing name: ")
    thingName = scanner.nextLine()
    createIoTThing(thingName)
    describeThing(thingName)
    println(DASHES)

    println(DASHES)
    println("2. Generate a device certificate.")
    println(
        """
```

```
        A device certificate performs a role in securing the communication
between devices (things) and the AWS IoT platform.
        """.trimIndent(),
    )

    print("Do you want to create a certificate for $thingName? (y/n)")
    val certAns = scanner.nextLine()
    var certificateArn: String? = ""
    if (certAns != null && certAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
        certificateArn = createCertificate()
        println("Attach the certificate to the AWS IoT thing.")
        attachCertificateToThing(thingName, certificateArn)
    } else {
        println("A device certificate was not created.")
    }
    println(DASHES)

    println(DASHES)
    println("3. Update an AWS IoT thing with Attributes.")
    println(
        """
        IoT thing attributes, represented as key-value pairs, offer a pivotal
advantage in facilitating efficient data
        management and retrieval within the AWS IoT ecosystem.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateThing(thingName)
    println(DASHES)

    println(DASHES)
    println("4. Return a unique endpoint specific to the Amazon Web Services
account.")
    println(
        """
        An IoT Endpoint refers to a specific URL or Uniform Resource Locator that
serves as the entry point for communication between IoT devices and the AWS IoT
service.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
```

```
    val endpointUrl = describeEndpoint()
    println(DASHES)

    println(DASHES)
    println("5. List your AWS IoT certificates")
    print("Press Enter to continue...")
    scanner.nextLine()
    if (certificateArn!!.isNotEmpty()) {
        listCertificates()
    } else {
        println("You did not create a certificates. Skipping this step.")
    }
    println(DASHES)

    println(DASHES)
    println("6. Create an IoT shadow that refers to a digital representation or
virtual twin of a physical IoT device")
    println(
        """
        A thing shadow refers to a feature that enables you to create a virtual
representation, or "shadow,"
        of a physical device or thing. The thing shadow allows you to synchronize
and control the state of a device between
        the cloud and the device itself. and the AWS IoT service. For example,
you can write and retrieve JSON data from a thing shadow.

        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateShawdowThing(thingName)
    println(DASHES)

    println(DASHES)
    println("7. Write out the state information, in JSON format.")
    print("Press Enter to continue...")
    scanner.nextLine()
    getPayload(thingName)
    println(DASHES)

    println(DASHES)
    println("8. Creates a rule")
    println(
        """
```

```
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """.trimIndent(),
    )
    print("Enter Rule name: ")
    val ruleName = scanner.nextLine()
    createIoTRule(roleARN, ruleName, snsAction)
    println(DASHES)

    println(DASHES)
    println("9. List your rules.")
    print("Press Enter to continue...")
    scanner.nextLine()
    listIoTRules()
    println(DASHES)

    println(DASHES)
    println("10. Search things using the name.")
    print("Press Enter to continue...")
    scanner.nextLine()
    val queryString = "thingName:$thingName"
    searchThings(queryString)
    println(DASHES)

    println(DASHES)
    if (certificateArn.length > 0) {
        print("Do you want to detach and delete the certificate for $thingName?
(y/n)")
        val delAns = scanner.nextLine()
        if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
            println("11. You selected to detach amd delete the certificate.")
            print("Press Enter to continue...")
            scanner.nextLine()
            detachThingPrincipal(thingName, certificateArn)
            deleteCertificate(certificateArn)
        } else {
            println("11. You selected not to delete the certificate.")
        }
    } else {
        println("11. You did not create a certificate so there is nothing to
delete.")
    }
```

```
    println(DASHES)

    println(DASHES)
    println("12. Delete the AWS IoT thing.")
    print("Do you want to delete the IoT thing? (y/n)")
    val delAns = scanner.nextLine()
    if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
 true)) {
        deleteIoTThing(thingName)
    } else {
        println("The IoT thing was not deleted.")
    }
    println(DASHES)

    println(DASHES)
    println("The AWS IoT workflow has successfully completed.")
    println(DASHES)
}

suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}

suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}

private fun extractCertificateId(certificateArn: String): String? {
    // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
```

```
    val arnParts = certificateArn.split(":".toRegex()).dropLastWhile
 { it.isEmpty() }.toTypedArray()
    val certificateIdPart = arnParts[arnParts.size - 1]
    return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") + 1)
}

suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}

suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}

suspend fun listIoTRules() {
    val listTopicRulesRequest = ListTopicRulesRequest {}

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
```

```
        val listTopicRulesResponse =
 iotClient.listTopicRules(listTopicRulesRequest)
        println("List of IoT rules:")
        val ruleList = listTopicRulesResponse.rules
        ruleList?.forEach { rule ->
            println("Rule name: ${rule.ruleName}")
            println("Rule ARN: ${rule.ruleArn}")
            println("--------------")
        }
    }
}

suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
```

```
    }
}

suspend fun getPayload(thingNameVal: String?) {
    val getThingShadowRequest =
        GetThingShadowRequest {
            thingName = thingNameVal
        }

    IotDataPlaneClient.fromEnvironment { region = "us-east-1" }.use
 { iotPlaneClient ->
        val getThingShadowResponse =
 iotPlaneClient.getThingShadow(getThingShadowRequest)
        val payload = getThingShadowResponse.payload
        val payloadString = payload?.let { java.lang.String(it, Charsets.UTF_8) }
        println("Received shadow data: $payloadString")
    }
}

suspend fun listCertificates() {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}

suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}

private fun getValue(input: String?): String {
    // Define a regular expression pattern for extracting the subdomain.
```

```
    val pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\\.com")

    // Match the pattern against the input string.
    val matcher = pattern.matcher(input)

    // Check if a match is found.
    if (matcher.find()) {
        val subdomain = matcher.group(1)
        println("Extracted subdomain: $subdomain")
        return subdomain
    } else {
        println("No match found")
    }
    return ""
}

suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}

suspend fun updateShawdowThing(thingNameVal: String?) {
    // Create the thing shadow state document.
```

```
    val stateDocument = "{\"state\":{\"reported\":{\"temperature\":25, \"humidity
\":50}}}"
    val byteStream: ByteStream = ByteStream.fromString(stateDocument)
    val byteArray: ByteArray = byteStream.toByteArray()

    val updateThingShadowRequest =
        UpdateThingShadowRequest {
            thingName = thingNameVal
            payload = byteArray
        }

    IotDataPlaneClient.fromEnvironment { region = "us-east-1" }.use
 { iotPlaneClient ->
        iotPlaneClient.updateThingShadow(updateThingShadowRequest)
        println("The thing shadow was updated successfully.")
    }
}

suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}

suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
```

```
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}

suspend fun createCertificate(): String? {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}

suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

IoT 래퍼 클래스를 생성하여 작업을 관리합니다.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)


    def create_thing(self, thing_name):
        """
        Creates an AWS IoT thing.

        :param thing_name: The name of the thing to create.
        :return: The name and ARN of the created thing.
        """
        try:
            response = self.iot_client.create_thing(thingName=thing_name)
            logger.info("Created thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Thing %s already exists. Skipping creation.",
 thing_name)
                return None
            logger.error(
                "Couldn't create thing %s. Here's why: %s: %s",
                thing_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

```
    def list_things(self):
        """
        Lists AWS IoT things.

        :return: The list of things.
        """
        try:
            things = []
            paginator = self.iot_client.get_paginator("list_things")
            for page in paginator.paginate():
                things.extend(page["things"])
            logger.info("Retrieved %s things.", len(things))
            return things
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list things. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise



    def create_keys_and_certificate(self):
        """
        Creates keys and a certificate for an AWS IoT thing.

        :return: The certificate ID, ARN, and PEM.
        """
        try:
            response =
self.iot_client.create_keys_and_certificate(setAsActive=True)
            logger.info("Created certificate %s.", response["certificateId"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't create keys and certificate. Here's why: %s: %s",
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response


    def attach_thing_principal(self, thing_name, principal):
        """
        Attaches a certificate to an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.attach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Attached principal %s to thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot attach principal. Resource not found.")
                return
            logger.error(
                "Couldn't attach principal to thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def describe_endpoint(self, endpoint_type="iot:Data-ATS"):
        """
        Gets the AWS IoT endpoint.

        :param endpoint_type: The endpoint type.
        :return: The endpoint.
        """
        try:
            response =
self.iot_client.describe_endpoint(endpointType=endpoint_type)
            logger.info("Retrieved endpoint %s.", response["endpointAddress"])
```

```
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't describe endpoint. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["endpointAddress"]


    def list_certificates(self):
        """
        Lists AWS IoT certificates.

        :return: The list of certificates.
        """
        try:
            certificates = []
            paginator = self.iot_client.get_paginator("list_certificates")
            for page in paginator.paginate():
                certificates.extend(page["certificates"])
            logger.info("Retrieved %s certificates.", len(certificates))
            return certificates
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list certificates. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def detach_thing_principal(self, thing_name, principal):
        """
        Detaches a certificate from an AWS IoT thing.

        :param thing_name: The name of the thing.
```

```
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.detach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Detached principal %s from thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot detach principal. Resource not found.")
                return
            logger.error(
                "Couldn't detach principal from thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_certificate(self, certificate_id):
        """
        Deletes an AWS IoT certificate.

        :param certificate_id: The ID of the certificate to delete.
        """
        try:
            self.iot_client.update_certificate(
                certificateId=certificate_id, newStatus="INACTIVE"
            )
            self.iot_client.delete_certificate(certificateId=certificate_id)
            logger.info("Deleted certificate %s.", certificate_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete certificate. Resource not found.")
                return
            logger.error(
                "Couldn't delete certificate. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def create_topic_rule(self, rule_name, topic, sns_action_arn, role_arn):
        """
        Creates an AWS IoT topic rule.

        :param rule_name: The name of the rule.
        :param topic: The MQTT topic to subscribe to.
        :param sns_action_arn: The ARN of the SNS topic to publish to.
        :param role_arn: The ARN of the IAM role.
        """
        try:
            self.iot_client.create_topic_rule(
                ruleName=rule_name,
                topicRulePayload={
                    "sql": f"SELECT * FROM '{topic}'",
                    "actions": [
                        {"sns": {"targetArn": sns_action_arn, "roleArn":
role_arn}}
                    ],
                },
            )
            logger.info("Created topic rule %s.", rule_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Topic rule %s already exists. Skipping creation.",
rule_name)
                return
            logger.error(
                "Couldn't create topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def list_topic_rules(self):
        """
        Lists AWS IoT topic rules.

        :return: The list of topic rules.
        """
        try:
            rules = []
            paginator = self.iot_client.get_paginator("list_topic_rules")
            for page in paginator.paginate():
```

```
                rules.extend(page["rules"])
            logger.info("Retrieved %s topic rules.", len(rules))
            return rules
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list topic rules. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise



    def search_index(self, query):
        """
        Searches the AWS IoT index.

        :param query: The search query.
        :return: The list of things found.
        """
        try:
            response = self.iot_client.search_index(queryString=query)
            logger.info("Found %s things.", len(response.get("things", [])))
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't search index. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response.get("things", [])


    def update_indexing_configuration(self):
        """
        Updates the AWS IoT indexing configuration to enable thing indexing.
        """
```

```
        try:
            self.iot_client.update_indexing_configuration(
                thingIndexingConfiguration={"thingIndexingMode": "REGISTRY"}
            )
            logger.info("Updated indexing configuration.")
        except ClientError as err:
            logger.error(
                "Couldn't update indexing configuration. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_thing(self, thing_name):
        """
        Deletes an AWS IoT thing.

        :param thing_name: The name of the thing to delete.
        """
        try:
            self.iot_client.delete_thing(thingName=thing_name)
            logger.info("Deleted thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete thing. Resource not found.")
                return
            logger.error(
                "Couldn't delete thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_topic_rule(self, rule_name):
        """
        Deletes an AWS IoT topic rule.

        :param rule_name: The name of the rule to delete.
        """
        try:
            self.iot_client.delete_topic_rule(ruleName=rule_name)
            logger.info("Deleted topic rule %s.", rule_name)
```

```
        except ClientError as err:
            logger.error(
                "Couldn't delete topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def update_thing_shadow(self, thing_name, shadow_state):
        """
        Updates the shadow for an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param shadow_state: The shadow state as a dictionary.
        """
        import json
        try:
            self.iot_data_client.update_thing_shadow(
                thingName=thing_name, payload=json.dumps(shadow_state)
            )
            logger.info("Updated shadow for thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot update thing shadow. Resource not found.")
                return
            logger.error(
                "Couldn't update thing shadow. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_thing_shadow(self, thing_name):
        """
        Gets the shadow for an AWS IoT thing.

        :param thing_name: The name of the thing.
        :return: The shadow state as a dictionary.
        """
        import json
        try:
```

```
            response =
 self.iot_data_client.get_thing_shadow(thingName=thing_name)
            shadow = json.loads(response["payload"].read())
            logger.info("Retrieved shadow for thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot get thing shadow. Resource not found.")
                return None
            logger.error(
                "Couldn't get thing shadow. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return shadow
```

IoT 기본 사항을 보여주는 대화형 시나리오를 실행합니다.

```
class IoTScenario:
    """Runs an interactive scenario that shows how to use AWS IoT."""

    is_interactive = True

    def __init__(self, iot_wrapper, iot_data_client, cfn_client,
 stack_name="IoTBasicsStack", template_path=None):
        """
        :param iot_wrapper: An instance of the IoTWrapper class.
        :param iot_data_client: A Boto3 IoT Data Plane client.
        :param cfn_client: A Boto3 CloudFormation client.
        :param stack_name: Name for the CloudFormation stack.
        :param template_path: Path to the CloudFormation template file.
        """
        self.iot_wrapper = iot_wrapper
        self.iot_data_client = iot_data_client
        self.cfn_client = cfn_client
        self.thing_name = None
        self.certificate_arn = None
        self.certificate_id = None
        self.rule_name = None
        self.stack_name = stack_name
```

```
        self.template_path = template_path or "../../../scenarios/basics/iot/
iot_usecase/resources/cfn_template.yaml"

    def _deploy_stack(self):
        """Deploy CloudFormation stack and return outputs."""
        with open(self.template_path, "r") as f:
            template_body = f.read()

        try:
            self.cfn_client.create_stack(
                StackName=self.stack_name,
                TemplateBody=template_body,
                Capabilities=["CAPABILITY_NAMED_IAM"]
            )

            waiter = self.cfn_client.get_waiter("stack_create_complete")
            waiter.wait(StackName=self.stack_name)

            response = self.cfn_client.describe_stacks(StackName=self.stack_name)
            outputs = {output["OutputKey"]: output["OutputValue"]
                       for output in response["Stacks"][0]["Outputs"]}
            return outputs["SNSTopicArn"], outputs["RoleArn"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "AlreadyExistsException":
                response =
 self.cfn_client.describe_stacks(StackName=self.stack_name)
                outputs = {output["OutputKey"]: output["OutputValue"]
                           for output in response["Stacks"][0]["Outputs"]}
                return outputs["SNSTopicArn"], outputs["RoleArn"]
            raise

    def _cleanup_stack(self):
        """Delete CloudFormation stack."""
        try:
            self.cfn_client.delete_stack(StackName=self.stack_name)
            waiter = self.cfn_client.get_waiter("stack_delete_complete")
            waiter.wait(StackName=self.stack_name)
            print("CloudFormation stack deleted successfully.")
        except ClientError as err:
            logger.error(f"Failed to delete stack: {err}")

    def run_scenario(self, thing_name, rule_name):
        """
        Runs the IoT basics scenario.
```

```
        :param thing_name: The name of the thing to create.
        :param rule_name: The name of the topic rule to create.
        """
        print("-" * 88)
        print("Welcome to the AWS IoT basics scenario!")
        print("-" * 88)
        print(
            "This scenario demonstrates how to interact with AWS IoT using the
AWS SDK for Python (Boto3).\n"
            "AWS IoT provides secure, bi-directional communication between
Internet-connected devices\n"
            "and the AWS cloud. You can manage device connections, process device
data, and build IoT applications.\n"
        )

        self.thing_name = thing_name
        self.rule_name = rule_name

        try:
            print("\nDeploying CloudFormation stack...")
            sns_topic_arn, role_arn = self._deploy_stack()
            print(f"Stack deployed. SNS Topic: {sns_topic_arn}")

            input("\nNext, we'll create an AWS IoT thing. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("1. Create an AWS IoT thing")
            print("-" * 88)
            response = self.iot_wrapper.create_thing(thing_name)
            print(f"Created thing: {response['thingName']}")
            print(f"Thing ARN: {response['thingArn']}")

            input("\nNext, we'll list things. Press Enter to continue...") if
self.is_interactive else None
            print("\n" + "-" * 88)
            print("2. List things")
            print("-" * 88)
            things = self.iot_wrapper.list_things()
            print(f"Found {len(things)} thing(s) in your account")
            for thing in things[:5]:  # Show first 5
                print(f"  Thing name: {thing['thingName']}")
```

```
            input("\nNext, we'll generate a device certificate. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("3. Generate a device certificate")
            print("-" * 88)
            cert_response = self.iot_wrapper.create_keys_and_certificate()
            self.certificate_arn = cert_response["certificateArn"]
            self.certificate_id = cert_response["certificateId"]
            print(f"Created certificate: {self.certificate_id}")

            input("\nNext, we'll attach the certificate to the thing. Press Enter
to continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("4. Attach the certificate to the thing")
            print("-" * 88)
            self.iot_wrapper.attach_thing_principal(thing_name,
self.certificate_arn)
            print(f"Attached certificate to thing: {thing_name}")

            input("\nNext, we'll update the thing shadow. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("5. Update the thing shadow")
            print("-" * 88)
            shadow_state = {"state": {"reported": {"temperature": 25, "humidity":
50}}}
            self.iot_wrapper.update_thing_shadow(thing_name, shadow_state)
            print(f"Updated shadow for thing: {thing_name}")

            input("\nNext, we'll get the thing shadow. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("6. Get the thing shadow")
            print("-" * 88)
            shadow = self.iot_wrapper.get_thing_shadow(thing_name)
            print(f"Shadow state: {json.dumps(shadow['state'], indent=2)}")

            input("\nNext, we'll get the AWS IoT endpoint. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("7. Get the AWS IoT endpoint")
            print("-" * 88)
            endpoint = self.iot_wrapper.describe_endpoint()
            print(f"IoT endpoint: {endpoint}")
```

```
            input("\nNext, we'll list certificates. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("8. List certificates")
            print("-" * 88)
            certificates = self.iot_wrapper.list_certificates()
            print(f"Found {len(certificates)} certificate(s)")
            for cert in certificates:
                print(f"  Certificate ID: {cert['certificateId']}")
                print(f"  Certificate ARN: {cert['certificateArn']}")
                print(f"  Status: {cert['status']}")

            input("\nNext, we'll create a topic rule. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("9. Create a topic rule")
            print("-" * 88)
            self.iot_wrapper.create_topic_rule(
                rule_name, f"device/{thing_name}/data", sns_topic_arn, role_arn
            )
            print(f"Created topic rule: {rule_name}")

            input("\nNext, we'll list topic rules. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("10. List topic rules")
            print("-" * 88)
            rules = self.iot_wrapper.list_topic_rules()
            print(f"Found {len(rules)} topic rule(s)")
            for rule in rules:
                print(f"  Rule name: {rule['ruleName']}")
                print(f"  Rule ARN: {rule['ruleArn']}")

            input("\nNext, we'll configure thing indexing. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("11. Configure thing indexing")
            print("-" * 88)
            self.iot_wrapper.update_indexing_configuration()
            print("Enabled thing indexing")
            print("Waiting for indexing to be ready...")
            time.sleep(10)
```

```
            input("\nNext, we'll search for things. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("12. Search for things")
            print("-" * 88)
            try:
                things = self.iot_wrapper.search_index(f"thingName:{thing_name}")
                if things:
                    print(f"Found {len(things)} thing(s) matching the query")
                    for thing in things:
                        print(f"  Thing name: {thing.get('thingName', 'N/A')}")
                        print(f"  Thing ID: {thing.get('thingId', 'N/A')}")
                else:
                    print("No things found. Indexing may take a few minutes.")
            except ClientError as err:
                if err.response["Error"]["Code"] in [
                    "IndexNotReadyException",
                    "InvalidRequestException",
                ]:
                    print("Search index not ready yet. This is expected.")
                else:
                    raise

        except ClientError as err:
            logger.error(
                "Scenario failed: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        finally:
            self._cleanup()

    def _cleanup(self):
        """Cleans up resources created during the scenario."""
        if not self.thing_name:
            return

        print("\n" + "-" * 88)
        print("Cleanup")
        print("-" * 88)

        if q.ask("Do you want to delete the resources? (y/n) ", q.is_yesno):
            try:
```

```
                if self.certificate_arn:
                    print(f"Detaching certificate from thing: {self.thing_name}")
                    self.iot_wrapper.detach_thing_principal(
                        self.thing_name, self.certificate_arn
                    )

                if self.certificate_id:
                    print(f"Deleting certificate: {self.certificate_id}")
                    self.iot_wrapper.delete_certificate(self.certificate_id)

                if self.thing_name:
                    print(f"Deleting thing: {self.thing_name}")
                    self.iot_wrapper.delete_thing(self.thing_name)

                if self.rule_name:
                    print(f"Deleting topic rule: {self.rule_name}")
                    self.iot_wrapper.delete_topic_rule(self.rule_name)

                self._cleanup_stack()
                print("Resources deleted successfully.")
            except ClientError as err:
                logger.error(
                    "Cleanup failed: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
        else:
            print("Resources will remain in your account.")

        print("\n" + "-" * 88)
        print("Thanks for using AWS IoT!")
        print("-" * 88)
```

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDKs AWS IoT 를 사용하기 위한 작업

다음 코드 예제에서는 AWS SDKs를 사용하여 개별 AWS IoT 작업을 수행하는 방법을 보여줍니다. 각 예시에는 GitHub에 대한 링크가 포함되어 있습니다. 여기에서 코드 설정 및 실행에 대한 지침을 찾을 수 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 AWS IoT API 참조를 참조하세요.

예제

- AWS SDK 또는 CLI와 AttachThingPrincipal 함께 사용
- AWS SDK 또는 CLI와 CreateKeysAndCertificate 함께 사용
- AWS SDK 또는 CLI와 CreateThing 함께 사용
- AWS SDK 또는 CLI와 CreateTopicRule 함께 사용
- AWS SDK 또는 CLI와 DeleteCertificate 함께 사용
- AWS SDK 또는 CLI와 DeleteThing 함께 사용
- AWS SDK 또는 CLI와 DeleteTopicRule 함께 사용
- AWS SDK 또는 CLI와 DescribeEndpoint 함께 사용
- AWS SDK 또는 CLI와 DescribeThing 함께 사용
- AWS SDK 또는 CLI와 DetachThingPrincipal 함께 사용
- AWS SDK 또는 CLI와 ListCertificates 함께 사용
- AWS SDK 또는 CLI와 ListThings 함께 사용
- AWS SDK 또는 CLI와 SearchIndex 함께 사용
- AWS SDK 또는 CLI와 UpdateIndexingConfiguration 함께 사용
- AWS SDK 또는 CLI와 UpdateThing 함께 사용

AWS SDK 또는 CLI와 **`AttachThingPrincipal`** 함께 사용

다음 코드 예시는 `AttachThingPrincipal`의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 AttachThingPrincipal을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
    else {
        std::cerr << "Failed to attach principal to thing." <<
```

```
                    outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 AttachThingPrincipal을 참조하세요.

CLI

AWS CLI

사물에 인증서 연결

다음 `attach-thing-principal` 예시에서는 MyTemperatureSensor 사물에 인증서를 연결합니다. 인증서는 ARN으로 식별됩니다. AWS IoT 콘솔에서 인증서의 ARN을 찾을 수 있습니다.

```
aws iot attach-thing-principal \
    --thing-name MyTemperatureSensor \
    --principal arn:aws:iot:us-
west-2:123456789012:cert/2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT 개발자 안내서의 레지스트리를 사용하여 사물을 관리하는 방법을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 AttachThingPrincipal을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
```

```
    * Attaches a certificate to an IoT Thing asynchronously.
    *
    * @param thingName The name of the IoT Thing.
    * @param certificateArn The ARN of the certificate to attach.
    *
    * This method initiates an asynchronous request to attach a certificate to
an IoT Thing.
    * If the request is successful, it prints a confirmation message and
additional information about the Thing.
    * If an exception occurs, it prints the error message.
    */
   public void attachCertificateToThing(String thingName, String certificateArn)
{
       AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
           .thingName(thingName)
           .principal(certificateArn)
           .build();

       CompletableFuture<AttachThingPrincipalResponse> future =
getAsyncClient().attachThingPrincipal(principalRequest);
       future.whenComplete((attachResponse, ex) -> {
           if (attachResponse != null &&
attachResponse.sdkHttpResponse().isSuccessful()) {
               System.out.println("Certificate attached to Thing
successfully.");

               // Print additional information about the Thing.
               describeThing(thingName);
           } else {
               Throwable cause = ex != null ? ex.getCause() : null;
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else if (cause != null) {
                   System.err.println("Unexpected error: " +
cause.getMessage());
               } else {
                   System.err.println("Failed to attach certificate to Thing.
HTTP Status Code: " +
                           attachResponse.sdkHttpResponse().statusCode());
               }
           }
       });
```

```
        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 AttachThingPrincipal을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 AttachThingPrincipal을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def attach_thing_principal(self, thing_name, principal):
        """
        Attaches a certificate to an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.attach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Attached principal %s to thing %s.", principal,
 thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
```

```
                logger.error("Cannot attach principal. Resource not found.")
                return
            logger.error(
                "Couldn't attach principal to thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 AttachThingPrincipal을 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **CreateKeysAndCertificate** 함께 사용

다음 코드 예시는 CreateKeysAndCertificate의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   /// <summary>
   /// Creates a device certificate for AWS IoT.
   /// </summary>
   /// <returns>The certificate details including ARN and certificate PEM, or
null if creation failed.</returns>
   public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
   {
       try
       {
```

```
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await
_amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조에서 CreateKeysAndCertificate를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
```

```
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
```

```
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조에서 CreateKeysAndCertificate를 참조하세요.

CLI

AWS CLI

RSA 키 쌍 생성 및 X.509 인증서 발급

다음 `create-keys-and-certificate`는 2048비트 RSA 키 쌍을 생성한 후 발급된 퍼블릭 키를 사용해 X.509 인증서를 발급합니다. AWS IoT가이 인증서에 대한 프라이빗 키를 제공하는 유일한 시간이기 때문에 안전한 위치에 보관해야 합니다.

```
aws iot create-keys-and-certificate \
    --certificate-pem-outfile "myTest.cert.pem" \
    --public-key-outfile "myTest.public.key" \
    --private-key-outfile "myTest.private.key"
```

출력:

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\nMMEXAMPLEuuN/
dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y
```

```
+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQ
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB
\n-----END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omittted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

자세한 내용은 AWS IoT 개발자 안내서의 IoT 디바이스 인증서 생성 및 등록을 참조하세요.

AWS IoT

- API 세부 정보는 AWS CLI 명령 참조의 CreateKeysAndCertificate를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Creates an IoT certificate asynchronously.
     *
     * @return The ARN of the created certificate.
     * <p>
     * This method initiates an asynchronous request to create an IoT
certificate.
     * If the request is successful, it prints the certificate details and
returns the certificate ARN.
     * If an exception occurs, it prints the error message.
     */
    public String createCertificate() {
        CompletableFuture<CreateKeysAndCertificateResponse> future =
getAsyncClient().createKeysAndCertificate();
        final String[] certificateArn = {null};
        future.whenComplete((response, ex) -> {
            if (response != null) {
```

```
                String certificatePem = response.certificatePem();
                certificateArn[0] = response.certificateArn();

                // Print the details.
                System.out.println("\nCertificate:");
                System.out.println(certificatePem);
                System.out.println("\nCertificate ARN:");
                System.out.println(certificateArn[0]);

            } else {
                Throwable cause = (ex instanceof CompletionException) ?
ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
        return certificateArn[0];
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조에서 CreateKeysAndCertificate를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createCertificate(): String? {
```

```
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateKeysAndCertificate를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
```

```
        return cls(iot_client, iot_data_client)

    def create_keys_and_certificate(self):
        """
        Creates keys and a certificate for an AWS IoT thing.

        :return: The certificate ID, ARN, and PEM.
        """
        try:
            response =
self.iot_client.create_keys_and_certificate(setAsActive=True)
            logger.info("Created certificate %s.", response["certificateId"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't create keys and certificate. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 CreateKeysAndCertificate를 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`CreateThing`** 함께 사용

다음 코드 예시는 `CreateThing`의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</
returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
 {ex.Message}");
            return null;
        }
```

```
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateThing을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Create an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateThing을 참조하세요.

CLI

AWS CLI

예제 1: 레지스트리에서 사물 레코드 생성

다음 create-thing 예시에서는 AWS IoT 사물 레지스트리에서 디바이스에 대한 항목을 생성합니다.

```
aws iot create-thing \
    --thing-name SampleIoTThing
```

출력:

```
{
    "thingName": "SampleIoTThing",
    "thingArn": "arn:aws:iot:us-west-2: 123456789012:thing/SampleIoTThing",
    "thingId": " EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE "
}
```

예시 2: 사물 유형과 연결된 사물 정의

다음 create-thing 예시에서는 지정된 사물 유형과 속성이 있는 사물을 생성합니다.

```
aws iot create-thing \
    --thing-name "MyLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

출력:

```
{
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797"
}
```

자세한 내용은 AWS IoT 개발자 안내서의 레지스트리를 사용하여 사물을 관리하는 방법 및 사물 유형을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 CreateThing을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   /**
    * Creates an IoT Thing with the specified name asynchronously.
    *
    * @param thingName The name of the IoT Thing to create.
    *
    * This method initiates an asynchronous request to create an IoT Thing with
the specified name.
    * If the request is successful, it prints the name of the thing and its ARN
value.
    * If an exception occurs, it prints the error message.
    */
   public void createIoTThing(String thingName) {
       CreateThingRequest createThingRequest = CreateThingRequest.builder()
           .thingName(thingName)
           .build();

       CompletableFuture<CreateThingResponse> future =
getAsyncClient().createThing(createThingRequest);
       future.whenComplete((createThingResponse, ex) -> {
           if (createThingResponse != null) {
               System.out.println(thingName + " was successfully created. The
ARN value is " + createThingResponse.thingArn());
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else {
                   System.err.println("Unexpected error: " +
cause.getMessage());
```

```
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateThing을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateThing을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_thing(self, thing_name):
        """
        Creates an AWS IoT thing.

        :param thing_name: The name of the thing to create.
        :return: The name and ARN of the created thing.
        """
        try:
            response = self.iot_client.create_thing(thingName=thing_name)
            logger.info("Created thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Thing %s already exists. Skipping creation.",
 thing_name)
                return None
```

```
            logger.error(
                "Couldn't create thing %s. Here's why: %s: %s",
                thing_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 CreateThing을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **CreateTopicRule** 함께 사용

다음 코드 예시는 CreateTopicRule의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</
param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
 snsTopicArn, string roleArn)
```

```
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                        RuleDisabled = false
                    }
                };

                await _amazonIoT.CreateTopicRuleAsync(request);
                _logger.LogInformation($"Created IoT rule {ruleName}");
                return true;
            }
            catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
            {
                _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
                return false;
            }
            catch (Exception ex)
            {
                _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
                return false;
            }
        }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 CreateTopicRule을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
                             const Aws::String &snsTopicARN, const Aws::String
 &sql,
                             const Aws::String &roleARN,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});
```

```
    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 CreateTopicRule을 참조하세요.

CLI

AWS CLI

Amazon SNS 알림을 보내는 규칙을 생성하려면

다음 `create-topic-rule` 예시에서는 디바이스 섀도우에서 찾을 수 있는 토양 수분 수준 판독값이 낮을 때 Amazon SNS 메시지를 보내는 규칙을 생성합니다.

```
aws iot create-topic-rule \
    --rule-name "LowMoistureRule" \
    --topic-rule-payload file://plant-rule.json
```

이 예시에서는 다음 JSON 코드를 `plant-rule.json`이라는 파일에 저장해야 합니다.

```
{
    "sql": "SELECT * FROM '$aws/things/MyRPi/shadow/update/accepted' WHERE
 state.reported.moisture = 'low'\n",
    "description": "Sends an alert whenever soil moisture level readings are too
 low.",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
            "sns": {
                "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyRPiLowMoistureTopic",
```

```
                "roleArn": "arn:aws:iam::123456789012:role/service-role/
MyRPiLowMoistureTopicRole",
                "messageFormat": "RAW"
            }
    }]
}
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT 개발자 안내서의 IoT 규칙 생성을 참조하세요 AWS IoT.

- API 세부 정보는 AWS CLI 명령 참조의 CreateTopicRule을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Creates an IoT rule asynchronously.
     *
     * @param roleARN The ARN of the IAM role that grants access to the rule's
actions.
     * @param ruleName The name of the IoT rule.
     * @param action The ARN of the action to perform when the rule is triggered.
     *
     * This method initiates an asynchronous request to create an IoT rule.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTRule(String roleARN, String ruleName, String action) {
        String sql = "SELECT * FROM '" + TOPIC + "'";
        SnsAction action1 = SnsAction.builder()
            .targetArn(action)
            .roleArn(roleARN)
            .build();
```

```
        // Create the action.
        Action myAction = Action.builder()
            .sns(action1)
            .build();

        // Create the topic rule payload.
        TopicRulePayload topicRulePayload = TopicRulePayload.builder()
            .sql(sql)
            .actions(myAction)
            .build();

        // Create the topic rule request.
        CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
            .ruleName(ruleName)
            .topicRulePayload(topicRulePayload)
            .build();

        CompletableFuture<CreateTopicRuleResponse> future =
getAsyncClient().createTopicRule(topicRuleRequest);
        future.whenComplete((response, ex) -> {
            if (response != null) {
                System.out.println("IoT Rule created successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to create IoT Rule.");
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 CreateTopicRule을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
```

```
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 CreateTopicRule을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_topic_rule(self, rule_name, topic, sns_action_arn, role_arn):
        """
        Creates an AWS IoT topic rule.

        :param rule_name: The name of the rule.
        :param topic: The MQTT topic to subscribe to.
        :param sns_action_arn: The ARN of the SNS topic to publish to.
        :param role_arn: The ARN of the IAM role.
        """
```

```
        try:
            self.iot_client.create_topic_rule(
                ruleName=rule_name,
                topicRulePayload={
                    "sql": f"SELECT * FROM '{topic}'",
                    "actions": [
                        {"sns": {"targetArn": sns_action_arn, "roleArn":
role_arn}}
                    ],
                },
            )
            logger.info("Created topic rule %s.", rule_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Topic rule %s already exists. Skipping creation.",
rule_name)
                return
            logger.error(
                "Couldn't create topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 CreateTopicRule을 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteCertificate** 함께 사용

다음 코드 예시는 DeleteCertificate의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
```

```
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteCertificate를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
```

```
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteCertificate를 참조하세요.

CLI

AWS CLI

디바이스 인증서를 삭제하려면

다음 `delete-certificate` 예시에서는 지정된 ID로 디바이스 인증서를 삭제합니다.

```
aws iot delete-certificate \
    --certificate-
id c0c57bbc8baaf4631a9a0345c957657f5e710473e3ddbee1428d216d54d53ac9
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT API 참조의 DeleteCertificate를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteCertificate를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Deletes a certificate asynchronously.
     *
     * @param certificateArn The ARN of the certificate to delete.
     *
     * This method initiates an asynchronous request to delete a certificate.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteCertificate(String certificateArn) {
        DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
            .build();

        CompletableFuture<DeleteCertificateResponse> future =
getAsyncClient().deleteCertificate(certificateProviderRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully
deleted.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteCertificate를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteCertificate를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
```

```
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_certificate(self, certificate_id):
        """
        Deletes an AWS IoT certificate.

        :param certificate_id: The ID of the certificate to delete.
        """
        try:
            self.iot_client.update_certificate(
                certificateId=certificate_id, newStatus="INACTIVE"
            )
            self.iot_client.delete_certificate(certificateId=certificate_id)
            logger.info("Deleted certificate %s.", certificate_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete certificate. Resource not found.")
                return
            logger.error(
                "Couldn't delete certificate. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteCertificate를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteThing** 함께 사용

다음 코드 예시는 DeleteThing의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
```

```
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DeleteThing을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteThing을 참조하세요.

CLI

AWS CLI

사물의 세부 정보를 표시하려면

다음 `delete-thing` 예시에서는 AWS 계정의 AWS IoT 레지스트리에서 사물을 삭제합니다.

aws iot delete-thing --thing-name "FourthBulb"

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT 개발자 안내서의 레지스트리를 사용하여 사물을 관리하는 방법을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteThing을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * Deletes an IoT Thing asynchronously.
 *
 * @param thingName The name of the IoT Thing to delete.
 *
 * This method initiates an asynchronous request to delete an IoT Thing.
 * If the deletion is successful, it prints a confirmation message.
 * If an exception occurs, it prints the error message.
 */
public void deleteIoTThing(String thingName) {
    DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
        .thingName(thingName)
        .build();
```

```
        CompletableFuture<DeleteThingResponse> future =
getAsyncClient().deleteThing(deleteThingRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println("Deleted Thing " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DeleteThing을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
```

```
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DeleteThing을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_thing(self, thing_name):
        """
        Deletes an AWS IoT thing.

        :param thing_name: The name of the thing to delete.
        """
        try:
            self.iot_client.delete_thing(thingName=thing_name)
            logger.info("Deleted thing %s.", thing_name)
```

```
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete thing. Resource not found.")
                return
            logger.error(
                "Couldn't delete thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 DeleteThing을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DeleteTopicRule** 함께 사용

다음 코드 예시는 DeleteTopicRule의 사용 방법을 보여 줍니다.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DeleteTopicRule를 참조하세요.

CLI

AWS CLI

규칙 삭제

다음 `delete-topic-rule` 예시에서는 지정된 규칙을 삭제합니다.

```
aws iot delete-topic-rule \
    --rule-name "LowMoistureRule"
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT 개발자 안내서의 규칙 삭제를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DeleteTopicRule을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_topic_rule(self, rule_name):
        """
        Deletes an AWS IoT topic rule.

        :param rule_name: The name of the rule to delete.
        """
        try:
            self.iot_client.delete_topic_rule(ruleName=rule_name)
            logger.info("Deleted topic rule %s.", rule_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 DeleteTopicRule을 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DescribeEndpoint** 함께 사용

다음 코드 예시는 DescribeEndpoint의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
```

```
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
 {ex.Message}");
            return null;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DescribeEndpoint를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.
```

```
    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeEndpoint를 참조하세요.

CLI

AWS CLI

예제 1: 현재 AWS 엔드포인트 가져오기

다음 `describe-endpoint` 예시에서는 모든 명령이 적용되는 기본 AWS 엔드포인트를 검색합니다.

```
aws iot describe-endpoint
```

출력:

```
{
    "endpointAddress": "abc123defghijk.iot.us-west-2.amazonaws.com"
}
```

자세한 내용은 AWS IoT 개발자 안내서의 DescribeEndpoint를 참조하세요.

예시 2: ATS 엔드포인트 가져오기

다음 describe-endpoint 예시에서는 Amazon Trust Services(ATS) 엔드포인트를 가져옵니다.

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

출력:

```
{
    "endpointAddress": "abc123defghijk-ats.iot.us-west-2.amazonaws.com"
}
```

자세한 내용은 AWS IoT 개발자 안내서의 X.509 인증서 및 AWS IoT를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DescribeEndpoint를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Describes the endpoint of the IoT service asynchronously.
     *
     * @return A CompletableFuture containing the full endpoint URL.
     *
     * This method initiates an asynchronous request to describe the endpoint of
 the IoT service.
     * If the request is successful, it prints and returns the full endpoint URL.
     * If an exception occurs, it prints the error message.
     */
    public String describeEndpoint() {
        CompletableFuture<DescribeEndpointResponse> future =
 getAsyncClient().describeEndpoint(DescribeEndpointRequest.builder().endpointType("iot:Da
ATS").build());
        final String[] result = {null};
```

```
        future.whenComplete((endpointResponse, ex) -> {
            if (endpointResponse != null) {
                String endpointUrl = endpointResponse.endpointAddress();
                String exString = getValue(endpointUrl);
                String fullEndpoint = "https://" + exString + "-ats.iot.us-
east-1.amazonaws.com";

                System.out.println("Full Endpoint URL: " + fullEndpoint);
                result[0] = fullEndpoint;
            } else {
                Throwable cause = (ex instanceof CompletionException) ?
 ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return result[0];
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeEndpoint를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
```

```
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeEndpoint를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def describe_endpoint(self, endpoint_type="iot:Data-ATS"):
        """
```

```
        Gets the AWS IoT endpoint.

        :param endpoint_type: The endpoint type.
        :return: The endpoint.
        """
        try:
            response =
self.iot_client.describe_endpoint(endpointType=endpoint_type)
            logger.info("Retrieved endpoint %s.", response["endpointAddress"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't describe endpoint. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["endpointAddress"]
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 DescribeEndpoint를 참조하세요. AWS

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
async fn show_address(client: &Client, endpoint_type: &str) -> Result<(), Error>
 {
    let resp = client
        .describe_endpoint()
        .endpoint_type(endpoint_type)
        .send()
```

```
        .await?;

    println!("Endpoint address: {}", resp.endpoint_address.unwrap());

    println!();

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 DescribeEndpoint를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **DescribeThing** 함께 사용

다음 코드 예시는 DescribeThing의 사용 방법을 보여 줍니다.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Describe an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeThing(const Aws::String &thingName,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DescribeThingRequest request;
```

```
    request.SetThingName(thingName);

    Aws::IoT::Model::DescribeThingOutcome outcome =
 iotClient.DescribeThing(request);

    if (outcome.IsSuccess()) {
        const Aws::IoT::Model::DescribeThingResult &result = outcome.GetResult();
        std::cout << "Retrieved thing '" << result.GetThingName() << "'" <<
 std::endl;
        std::cout << "thingArn: " << result.GetThingArn() << std::endl;
        std::cout << result.GetAttributes().size() << " attribute(s) retrieved"
                  << std::endl;
        for (const auto &attribute: result.GetAttributes()) {
            std::cout << "  attribute: " << attribute.first << "=" <<
 attribute.second
                      << std::endl;
        }
    }
    else {
        std::cerr << "Error describing thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DescribeThing을 참조하세요.

CLI

AWS CLI

사물의 세부 정보를 표시하려면

다음 `describe-thing` 예제에서는 AWS 계정의 AWS IoT 레지스트리에 정의된 사물(디바이스)에 대한 정보를 표시합니다.

aws iot describe-thing --thing-name "MyLightBulb"

출력:

```
{
```

```
    "defaultClientId": "MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    },
    "version": 1
}
```

자세한 내용은 AWS IoT 개발자 안내서의 레지스트리를 사용하여 사물을 관리하는 방법을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DescribeThing 섹션을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Describes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to describe an IoT Thing.
     * If the request is successful, it prints the Thing details.
     * If an exception occurs, it prints the error message.
     */
    private void describeThing(String thingName) {
        DescribeThingRequest thingRequest = DescribeThingRequest.builder()
            .thingName(thingName)
            .build();
```

```
        CompletableFuture<DescribeThingResponse> future =
getAsyncClient().describeThing(thingRequest);
        future.whenComplete((describeResponse, ex) -> {
            if (describeResponse != null) {
                System.out.println("Thing Details:");
                System.out.println("Thing Name: " +
describeResponse.thingName());
                System.out.println("Thing ARN: " + describeResponse.thingArn());
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to describe Thing.");
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DescribeThing을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
```

```
        }

    // Print Thing details.
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DescribeThing을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 `DetachThingPrincipal` 함께 사용

다음 코드 예시는 `DetachThingPrincipal`의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
```

```
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's
why: {ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 DetachThingPrincipal을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Detach a principal from an AWS IoT thing.
/*!
```

```
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 DetachThingPrincipal을 참조하세요.

CLI

AWS CLI

사물에서 인증서 및 위탁자를 분리하려면

다음 detach-thing-principal 예시에서는 지정된 사물에서 위탁자를 나타내는 인증서를 제거합니다.

```
aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT 개발자 안내서의 레지스트리를 사용하여 사물을 관리하는 방법을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 DetachThingPrincipal을 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Detaches a principal (certificate) from an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to detach.
     *
     * This method initiates an asynchronous request to detach a certificate from
an IoT Thing.
     * If the detachment is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void detachThingPrincipal(String thingName, String certificateArn) {
        DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
            .principal(certificateArn)
            .thingName(thingName)
```

```
            .build();

        CompletableFuture<DetachThingPrincipalResponse> future =
 getAsyncClient().detachThingPrincipal(thingPrincipalRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully removed
 from " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 DetachThingPrincipal을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
```

```
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 DetachThingPrincipal을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def detach_thing_principal(self, thing_name, principal):
        """
        Detaches a certificate from an AWS IoT thing.
```

```
        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.detach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Detached principal %s from thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot detach principal. Resource not found.")
                return
            logger.error(
                "Couldn't detach principal from thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 DetachThingPrincipal을 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **`ListCertificates`** 함께 사용

다음 코드 예시는 `ListCertificates`의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing
failed.</returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 ListCertificates를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
```

```
        std::cout << std::endl;
    }

    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 ListCertificates를 참조하세요.

CLI

AWS CLI

예제 1: AWS 계정에 등록된 인증서 나열

다음 `list-certificates` 예시에서는 계정에 등록된 모든 인증서를 나열합니다. 페이징 수가 기본 제한인 25보다 큰 경우, 이 명령의 `nextMarker` 응답 값을 사용하여 다음 명령에 제공하여 다음 결과 배치를 가져올 수 있습니다. 값 없이 `nextMarker`가 반환될 때까지 반복합니다.

```
aws iot list-certificates
```

출력:

```
{
    "certificates": [
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
            "certificateId":
 "604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36",
            "status": "ACTIVE",
            "creationDate": 1556810537.617
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e
            "certificateId":
 "262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e",
            "status": "ACTIVE",
            "creationDate": 1546447050.885
        },
```

```
        {
            "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "certificateId":
 "b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "status": "ACTIVE",
            "creationDate": 1546292258.322
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42
            "certificateId":
 "7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42",
            "status": "ACTIVE",
            "creationDate": 1541457693.453
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3
            "certificateId":
 "54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3",
            "status": "ACTIVE",
            "creationDate": 1541113568.611
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e
            "certificateId":
 "4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e",
            "status": "ACTIVE",
            "creationDate": 1541022751.983
        }
    ]
}
```

- API 세부 정보는 AWS CLI 명령 참조의 ListCertificates를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Lists all certificates asynchronously.
     *
     * This method initiates an asynchronous request to list all certificates.
     * If the request is successful, it prints the certificate IDs and ARNs.
     * If an exception occurs, it prints the error message.
     */
    public void listCertificates() {
        CompletableFuture<ListCertificatesResponse> future =
getAsyncClient().listCertificates();
        future.whenComplete((response, ex) -> {
            if (response != null) {
                List<Certificate> certList = response.certificates();
                for (Certificate cert : certList) {
                    System.out.println("Cert id: " + cert.certificateId());
                    System.out.println("Cert Arn: " + cert.certificateArn());
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list certificates.");
                }
            }
        });

        future.join();
```

```
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 ListCertificates를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listCertificates() {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 ListCertificates를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def list_certificates(self):
        """
        Lists AWS IoT certificates.

        :return: The list of certificates.
        """
        try:
            certificates = []
            paginator = self.iot_client.get_paginator("list_certificates")
            for page in paginator.paginate():
                certificates.extend(page["certificates"])
            logger.info("Retrieved %s certificates.", len(certificates))
            return certificates
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list certificates. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListCertificates를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **ListThings** 함께 사용

다음 코드 예시는 ListThings의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
```

```
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- API 세부 정보는 API 참조의 ListThingsAWS SDK for .NET 를 참조하세요.

CLI

AWS CLI

예시 1: 레지스트리의 모든 사물 나열

다음 `list-things` 예시에서는 AWS 계정의 AWS IoT 레지스트리에 정의된 사물(디바이스)을 나열합니다.

```
aws iot list-things
```

출력:


```
{
    "things": [
        {
            "thingName": "ThirdBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/ThirdBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 2
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        },
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
        "thingName": "SampleIoTThing",
        "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/SampleIoTThing",
        "attributes": {},
        "version": 1
        }
    ]
}
```

예시 2: 특정 속성을 가진 정의된 사물 나열

다음 `list-things` 예시에서는 이름이 `wattage`인 속성을 가진 사물의 목록을 표시합니다.

```
aws iot list-things \
    --attribute-name wattage
```

출력:

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        }
    ]
}
```

자세한 내용은 AWS IoT 개발자 안내서의 레지스트리를 사용하여 사물을 관리하는 방법을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 ListThings를 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def list_things(self):
        """
        Lists AWS IoT things.

        :return: The list of things.
        """
        try:
            things = []
            paginator = self.iot_client.get_paginator("list_things")
            for page in paginator.paginate():
                things.extend(page["things"])
            logger.info("Retrieved %s things.", len(things))
            return things
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
```

```
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list things. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 ListThings를 참조하세요.

Rust

SDK for Rust

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
async fn show_things(client: &Client) -> Result<(), Error> {
    let resp = client.list_things().send().await?;

    println!("Things:");

    for thing in resp.things.unwrap() {
        println!(
            "  Name:  {}",
            thing.thing_name.as_deref().unwrap_or_default()
        );
        println!(
            "  Type:  {}",
            thing.thing_type_name.as_deref().unwrap_or_default()
        );
        println!(
            "  ARN:   {}",
            thing.thing_arn.as_deref().unwrap_or_default()
        );
```

```
        println!();
    }

    println!();

    Ok(())
}
```

- API 세부 정보는 AWS SDK for Rust API 참조의 ListThings을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **SearchIndex** 함께 사용

다음 코드 예시는 SearchIndex의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
 search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
```

```
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count}
Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting
for the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
```

```
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready
(attempt {attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if
it's ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if
(configResponse.ThingIndexingConfiguration?.ThingIndexingMode ==
ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };
```

```
                        var response = await
_amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready
after configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 SearchIndex를 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
```

```
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 SearchIndex를 참조하세요.

CLI

AWS CLI

사물 인덱스를 쿼리하려면

다음 `search-index` 예시에서는 `LightBulb`의 유형이 있는 사물의 `AWS_Things` 인덱스를 쿼리합니다.

```
aws iot search-index \
    --index-name "AWS_Things" \
    --query-string "thingTypeName:LightBulb"
```

출력:

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
```

```
            "thingId": "40da2e73-c6af-406e-b415-15acae538797",
            "thingTypeName": "LightBulb",
            "thingGroupNames": [
                "LightBulbs",
                "DeadBulbs"
            ],
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "ThirdBulb",
            "thingId": "615c8455-33d5-40e8-95fd-3ee8b24490af",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingId": "6dae0d3f-40c1-476a-80c4-1ed24ba6aa11",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        }
    ]
}
```

자세한 내용은 AWS IoT 개발자 안내서의 사물 인덱싱 관리를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 SearchIndex를 참조하세요.

Java

SDK for Java 2.x

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /**
     * Searches for IoT Things asynchronously based on a query string.
     *
     * @param queryString The query string to search for Things.
     *
     * This method initiates an asynchronous request to search for IoT Things.
     * If the request is successful and Things are found, it prints their IDs.
     * If no Things are found, it prints a message indicating so.
     * If an exception occurs, it prints the error message.
     */
    public void searchThings(String queryString) {
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();

        CompletableFuture<SearchIndexResponse> future =
getAsyncClient().searchIndex(searchIndexRequest);
        future.whenComplete((searchIndexResponse, ex) -> {
            if (searchIndexResponse != null) {
                // Process the result.
                if (searchIndexResponse.things().isEmpty()) {
                    System.out.println("No things found.");
                } else {
                    searchIndexResponse.things().forEach(thing ->
System.out.println("Thing id found using search is " + thing.thingId()));
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
```

```
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                } else {
                    System.err.println("Failed to search for IoT Things.");
                }
            }
        });

        future.join();
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 SearchIndex를 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 SearchIndex을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def search_index(self, query):
        """
        Searches the AWS IoT index.

        :param query: The search query.
        :return: The list of things found.
        """
        try:
            response = self.iot_client.search_index(queryString=query)
            logger.info("Found %s things.", len(response.get("things", [])))
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't search index. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response.get("things", [])
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 SearchIndex를 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **UpdateIndexingConfiguration** 함께 사용

다음 코드 예시는 UpdateIndexingConfiguration의 사용 방법을 보여 줍니다.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Update the indexing configuration.
/*!
  \param thingIndexingConfiguration: A ThingIndexingConfiguration object which is
 ignored if not set.
  \param thingGroupIndexingConfiguration: A ThingGroupIndexingConfiguration
 object which is ignored if not set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
```

```
 */
bool AwsDoc::IoT::updateIndexingConfiguration(
        const Aws::IoT::Model::ThingIndexingConfiguration
 &thingIndexingConfiguration,
        const Aws::IoT::Model::ThingGroupIndexingConfiguration
 &thingGroupIndexingConfiguration,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::UpdateIndexingConfigurationRequest request;

    if (thingIndexingConfiguration.ThingIndexingModeHasBeenSet()) {
        request.SetThingIndexingConfiguration(thingIndexingConfiguration);
    }

    if (thingGroupIndexingConfiguration.ThingGroupIndexingModeHasBeenSet()) {

 request.SetThingGroupIndexingConfiguration(thingGroupIndexingConfiguration);
    }

    Aws::IoT::Model::UpdateIndexingConfigurationOutcome outcome =
 iotClient.UpdateIndexingConfiguration(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "UpdateIndexingConfiguration succeeded." << std::endl;
    }
    else {
        std::cerr << "UpdateIndexingConfiguration failed."
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 UpdateIndexingConfiguration을 참조하세요.

CLI

AWS CLI

사물 인덱싱을 활성화하려면

다음 update-indexing-configuration 예제에서는 사물 인덱싱이 AWS_Things 인덱스를 사용하여 레지스트리 데이터, 섀도우 데이터 및 사물 연결 상태 검색을 지원하도록 합니다.

```
aws iot update-indexing-configuration
    --thing-indexing-
configuration thingIndexingMode=REGISTRY_AND_SHADOW,thingConnectivityIndexingMode=STATUS
```

이 명령은 출력을 생성하지 않습니다.

자세한 내용은 AWS IoT 개발자 안내서의 사물 인덱싱 관리를 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateIndexingConfiguration을 참조하세요.

Python

SDK for Python(Boto3)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def update_indexing_configuration(self):
```

```
        """
        Updates the AWS IoT indexing configuration to enable thing indexing.
        """
        try:
            self.iot_client.update_indexing_configuration(
                thingIndexingConfiguration={"thingIndexingMode": "REGISTRY"}
            )
            logger.info("Updated indexing configuration.")
        except ClientError as err:
            logger.error(
                "Couldn't update indexing configuration. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API 세부 정보는 SDK for Python (Boto3) API 참조의 UpdateIndexingConfiguration을 참조하세요. AWS

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 CLI와 **UpdateThing** 함께 사용

다음 코드 예시는 UpdateThing의 사용 방법을 보여 줍니다.

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
```

```
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API 참조의 UpdateThing을 참조하세요.

C++

SDK for C++

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 UpdateThing을 참조하세요.

CLI

AWS CLI

사물에 사물 유형을 연결하려면

다음 update-thing 예시에서는 AWS IoT 레지스트리의 사물을 사물 유형과 연결합니다. 연결하면 사물 유형에 의해 정의된 속성의 값을 입력합니다.

```
aws iot update-thing \
    --thing-name "MyOtherLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

이 명령은 출력을 생성하지 않습니다. 결과를 표시하려면 describe-thing 명령을 사용합니다.

자세한 내용은 AWS IoT 개발자 안내서의 사물 유형을 참조하세요.

- API 세부 정보는 AWS CLI 명령 참조의 UpdateThing을 참조하세요.

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub에 더 많은 내용이 있습니다. AWS 코드 예 리포지토리에서 전체 예를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
```

```
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API 참조의 UpdateThing을 참조하세요.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 섹션을 참조하세요AWS SDK AWS IoT 에서 사용. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS IoT 할당량

AWS 일반 참조에서 AWS IoT 할당량에 대한 정보를 찾을 수 있습니다.

- AWS IoT Core 할당량 정보는 AWS IoT Core 엔드포인트 및 할당량을 참조하세요.
- AWS IoT Device Management 할당량 정보는 AWS IoT Device Management 엔드포인트 및 할당량을 참조하세요.
- AWS IoT Device Defender 할당량 정보는 AWS IoT Device Defender 엔드포인트 및 할당량을 참조하세요.

AWS IoT Core 요금

AWS 마케팅 페이지 및 AWS 요금 계산기에서 AWS IoT Core 요금 정보를 찾을 수 있습니다.

- AWS IoT Core 요금 정보를 확인하려면 AWS IoT Core 요금을 참조하세요.
- 아키텍트 솔루션의 비용을 추정하려면 AWS 요금 계산기를 참조하세요.

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