デベロッパーガイド

AWS IoT Core



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

AWS IoT Core: デベロッパーガイド

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

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

とは AWS IoT 1
デバイスとアプリのアクセス方法 AWS IoT 2
AWS IoT できること 3
産業における IoT 3
ホームオートメーションの IoT 3
の AWS IoT 仕組み 4
IoT ユニバース 4
AWS IoT サービスの概要 7
AWS IoT Core サービス 12
の詳細 AWS IoT 15
のトレーニングリソース AWS IoT 16
AWS IoT リソースとガイド 16
AWS IoT ソーシャルメディアで 17
AWS AWS IoT Core ルールエンジンで使用される サービス 17
でサポートされている通信プロトコル AWS IoT Core 19
AWS IoT コンソールの新機能 19
凡例 24
AWS SDKs の使用 24
開始方法のチュートリアル 26
最初のデバイスを に接続する AWS IoT Core 26
セットアップ AWS アカウント 28
にサインアップする AWS アカウント 28
管理アクセスを持つユーザーを作成する 29
AWS IoT コンソールを開く 30
インタラクティブチュートリアル 31
IoT デバイスの接続 32
オフラインデバイスの状態の保存 33
デバイスデータのサービスへのルーティング 34
クイックコネクトチュートリアル 35
ステップ 1. チュートリアルを開始する 36
ステップ 2. モノのオブジェクトを作成する 37
ステップ 3. デバイスにファイルをダウンロードする 40
ステップ 4. サンプルを実行する 43
ステップ 5. さらに詳しく 46

接続テスト 47
Advanced Connect のチュートリアル 53
どのデバイスオプションが最適ですか? 54
AWS IoT リソースの作成 55
デバイスを設定する 59
MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する 99
MQTT クライアントで MQTT メッセージを表示する 100
MQTT クライアントから MQTT メッセージを発行する 103
MQTT クライアントで共有サブスクリプションをテストする 105
AWS IoT チュートリアル 107
AWS IoT Device Client を使用したデモの構築 107
AWS IoT Device Client でデモを構築するための前提条件 108
IoT Device Client を使用する準備 111
IoT Device Client のインストールと設定 125
MQTT を使用してデバイスクライアントと通信する 139
Device Client で IoT ジョブを実行する 159
クリーンアップ 174
AWS IoT Device SDKsを使用したソリューションの構築 184
AWS IoT Device SDKs を使用してソリューションの構築を開始する 185
AWS IoT Device SDK AWS IoT Core を使用してデバイスを に接続する 185
デバイスデータを他の サービスにルーティングする AWS IoT ルールの作成 210
デバイスがオフラインになっている間にデバイスの状態をデバイスシャドウで保持する 256
のカスタムオーソライザーの作成 AWS IoT Core 287
AWS IoT と Raspberry Pi による土壌湿度のモニタリング 305
に接続する AWS IoT Core 319
AWS IoT Core- コントロールプレーンエンドポイント 319
AWS IoT デバイスエンドポイント 321
AWS IoT Core for LoRaWAN ゲートウェイとデバイス 322
AWS IoT Core サービスエンドポイントに接続する 323
AWS CLI の AWS IoT Core 324
AWS SDKs 325
AWS モバイル SDKs 330
AWS IoT Core サービスの REST APIs 331
デバイスを に接続する AWS IoT 332
AWS IoT デバイスデータとサービスエンドポイント 333
AWS IoT デバイス SDKs 335

デバイス通信プロトコル 338
MQTT トピック 387
ドメイン設定 415
FIPS AWS IoT エンドポイントに接続する 443
AWS IoT Core- コントロールプレーンエンドポイント 443
AWS IoT Core– データプレーンエンドポイント 443
AWS IoT Core- 認証情報プロバイダーエンドポイント 444
AWS IoT Device Management– ジョブデータエンドポイント 444
AWS IoT Device Management– Fleet Hub エンドポイント 445
AWS IoT Device Management- Secure Tunneling API エンドポイント 445
AWS IoT Device Management - マネージド統合エンドポイント 446
デバイスを管理する 447
レジストリ 448
モノの作成 448
モノのリスト表示 449
モノを記述する 451
モノの更新 452
モノの削除 452
モノにプリンシパルをアタッチする 453
プリンシパルに関連付けられているモノを一覧表示する 454
モノに関連付けられているプリンシパルを一覧表示する 454
プリンシパルに関連付けられているモノを一覧表示する (V2) 455
モノに関連付けられているプリンシパルを一覧表示する (V2) 456
モノからプリンシパルをデタッチする 456
モノのタイプ 457
モノのタイプを作成する 458
モノのタイプをリスト表示する 458
モノのタイプを記述する 459
モノのタイプをモノに関連付ける 459
モノのタイプを更新する 460
モノのタイプを非推奨にする 460
モノのタイプを削除する 462
モノの静的グループ 462
モノの静的グループの作成 464
モノのグループの説明 465
モノの静的グループにモノを追加する 466

モノの静的グループからモノを削除する 466
モノのグループ内のモノを一覧表示する 467
モノのグループの一覧表示 467
モノが属するグループを一覧表示する 469
モノの静的グループの更新 470
モノのグループを削除する 471
モノの静的グループにポリシーをアタッチする 471
モノの静的グループからポリシーをデタッチする 472
モノの静的グループにアタッチされているポリシーを一覧表示する 472
ポリシーのグループを一覧表示する 473
モノの有効なポリシーを取得する 474
MQTT アクションテストの認可 474
モノの動的グループ 476
モノの動的グループのユースケース 477
モノの動的グループを作成する 478
モノの動的グループを記述する 479
モノの動的グループを更新する 481
モノの動的グループを削除する 481
モノの動的および静的グループの制限 481
モノの動的グループの制限 482
モノを接続に関連付ける 484
ユースケース 485
モノを接続に関連付ける方法 485
伝播属性を追加する 488
AWS マネジメントコンソール 489
AWS CLI 490
リソースにタグを付ける 492
タグの基本 492
タグの制約と制限 493
IAM ポリシーでタグ付けする 494
請求グループ 496
コスト配分と使用状況データの表示 497
セキュリティ 499
のセキュリティ AWS IoT 500
認証 501
X.509 証明書の概要 501

サーバー認証 501
クライアント認証 505
カスタム認証と認可 545
Authorization 574
AWS トレーニングと認定 577
AWS IoT Core ポリシー 577
AWS IoT Core 認証情報プロバイダーを使用した AWS サービスへの直接呼び出しの許可 .. 658
IAM を使用したクロスアカウントアクセス 664
データ保護 667
でのデータの暗号化 AWS IoT 668
のトランスポートセキュリティ AWS IoT Core 668
データ暗号化 674
アイデンティティ/アクセス管理 689
オーディエンス 690
IAM アイデンティティを使用した認証 690
ポリシーを使用したアクセスの管理 691
が IAM と AWS IoT 連携する方法 693
アイデンティティベースのポリシーの例 726
AWS マネージドポリシー 730
トラブルシューティング 745
ログ記録とモニタリング 748
モニタリングツール 748
コンプライアンス検証 750
耐障害性 750
VPC エンドポイント AWS IoT Core での の使用 751
AWS IoT Core コントロールプレーン用の VPC エンドポイントの作成 752
AWS IoT Core データプレーンの VPC エンドポイントの作成 752
AWS IoT Core 認証情報プロバイダーの VPC エンドポイントの作成 753
Amazon VPC インターフェイスエンドポイントの作成 754
プライベートホストゾーンを設定 756
VPC エンドポイント AWS IoT Core を介した へのアクセスの制御 758
制限事項 760
を使用した VPC エンドポイントのスケーリング AWS IoT Core 761
VPC エンドポイントでのカスタムドメインの使用 761
の VPC エンドポイントの可用性 AWS IoT Core 761
VPC エンドポイント AWS IoT Device Management での の使用 761

インフラストラクチャセキュリティ 767
セキュリティモニタリング 768
セキュリティのベストプラクティス 768
での MQTT 接続の保護 AWS IoT 768
デバイスのクロックを同期させる 771
サーバー証明書の検証 772
デバイスごとの単一の ID を使用する 772
バックアップ AWS リージョン として 2 番目の を使用する 773
ジャストインタイムプロビジョニングの使用 773
AWS IoT Device Advisor テストを実行するアクセス許可 773
デバイスアドバイザーのクロスサービスでの混乱した代理の防止 775
AWS トレーニングと認定 776
モニタリング AWS IoT 777
AWS IoT ログ記録の設定 778
V2 ログインの設定 AWS IoT 779
ログ記録ロールとポリシーの構成 779
でログ記録を設定する AWS IoT (コンソール) 782
アカウントとイベントレベルのログインを設定する AWS IoT (CLI) 784
AWS IoT (CLI) でリソース固有のオーバーライドを設定する 787
ログレベル 789
Amazon CloudWatch を使用して AWS IoT アラームとメトリクスをモニタリングする 790
AWS IoT メトリクスの使用 790
CloudWatch アラームの作成 791
メトリクスとディメンション 795
CloudWatch Logs AWS IoT を使用したモニタリング 819
CloudWatch コンソールでの AWS IoT ログの表示 820
CloudWatch Logs AWS IoT ログエントリ 821
Amazon CloudWatch にデバイス側のログをアップロード 860
仕組み 861
AWS IoT ルールを使用したデバイス側のログのアップロード 862
AWS IoT API コールのログ記録 872
AWS IoT CloudTrail の情報 872
AWS IoT ログファイルエントリについて 873
Rules 876
アクセス許可の付与 877
ルールエンジンアクセスを取り消す 880

ロールのアクセス権限の適用 880
ルールの作成 881
ルールを作成する (コンソール) 882
ルール (CLI) を作成する 884
ルールを管理する 888
ルールのタグ付け 888
ルールの表示 890
ルールの削除 890
AWS IoT ルールアクション 890
Apache Kafka 893
CloudWatch アラーム 906
CloudWatch Logs 908
CloudWatch メトリクス 910
DynamoDB 913
DynamoDBv2 916
Elasticsearch 918
HTTP 921
AWS IoT Events 970
AWS IoT SiteWise 972
Firehose 978
Kinesis Data Streams 980
Lambda 983
ロケーション 986
OpenSearch 990
再発行 993
S3 996
Salesforce IoT 998
SNS 999
SQS 1002
ステップ関数 1004
Timestream 1006
ルールのトラブルシューティング 1013
クロスアカウントリソースにアクセスする 1014
前提条件 1014
Amazon SQS のクロスアカウント設定 1014
Amazon SNS のクロスアカウント設定 1016

Amazon S3 のクロスアカウント設定 1018
のクロスアカウント設定 AWS Lambda 1021
エラー処理 (エラーアクション) 1023
エラーアクションメッセージ形式 1023
エラーアクションの例 1025
基本的な取り込み 1026
基本的な取り込みの使用 1027
AWS IoT SQL リファレンス 1028
SELECT 句 1029
FROM 句 1031
SET 句 1032
WHERE 句 1036
データ型 1037
オペレータ 1042
関数 1052
リテラル 1124
Case ステートメント 1125
JSON 拡張 1126
置換テンプレート 1128
ネストされたオブジェクトのクエリ 1130
バイナリペイロード 1132
SQL バージョン 1139
Shadows 1141
シャドウの使用 1141
名前付きのシャドウまたは名前のないシャドウの使用の選択 1142
シャドウにアクセスする 1142
デバイス、アプリ、その他のクラウドサービスでのシャドウの使用 1143
メッセージの順序 1144
シャドウメッセージのトリム 1146
デバイスでのシャドウの使用 1146
への最初の接続時にデバイスを初期化する AWS IoT 1148
デバイスが に接続されている間のメッセージの処理 AWS IoT 1150
デバイスが に再接続したときのメッセージの処理 AWS IoT 1151
アプリとサービスでのシャドウの使用 1151
への接続時にアプリまたはサービスを初期化する AWS IoT 1153
アプリケーションまたはサービスが に接続されている間の処理状態の変更 AWS IoT 1153

デバイスが接続されているかどうかの検出 1153
デバイスシャドウサービス通信のシミュレーション 1155
シミュレーションの設定 1156
デバイスの初期化 1156
アプリからアップデートを送信する 1160
デバイスでの更新に応答 1163
アプリで更新を確認する 1167
シミュレーションを超える 1169
シャドウとの相互作用 1169
プロトコルサポート 1170
状態の要求と報告 1170
シャドウの更新 1171
シャドウキュメントの取得 1175
シャドウデータの削除 1176
Device Shadow の REST API 1179
GetThingShadow 1181
UpdateThingShadow 1182
DeleteThingShadow 1183
ListNamedShadowsForThing 1184
Device Shadow MQTT トピック 1186
/get 1187
/get/accepted 1188
/get/rejected 1189
/update 1190
/update/delta 1191
/update/accepted 1192
/update/documents 1193
/update/rejected 1194
/delete 1195
/delete/accepted 1196
/delete/rejected 1197
Device Shadow サービスドキュメント 1198
シャドウドキュメントの例 1198
ドキュメントのプロパティ 1205
delta 状態 1206
シャドウドキュメントのバージョニング 1208

シャドウドキュメント内のクライアントトークン 1208
空のシャドウドキュメントのプロパティ 1209
シャドウドキュメント内の配列値 1210
Device Shadow エラーメッセージ 1210
Software Package Catalog 1213
Software Package Catalog の使用準備 1214
パッケージバージョンライフサイクル 1214
パッケージバージョンの命名規則 1216
デフォルトバージョン 1216
バージョン属性 1216
ソフトウェア部品表 1217
AWS IoT フリートインデックス作成の有効化 1220
予約済みの名前付きシャドウ 1221
ソフトウェアパッケージの削除 1223
セキュリティの準備 1223
リソースベースの認証 1223
AWS IoT パッケージバージョンをデプロイするためのジョブ権限 1225
AWS IoT 予約済み名前付きシャドウを更新するジョブ権限 1226
AWS IoT Amazon S3 からダウンロードするジョブのアクセス許可 1229
パッケージバージョンのソフトウェア部品表を更新するアクセス許可 1229
フリートインデックス作成の準備 1232
$package シャドウをデータソースとして設定する 1232
コンソールに表示されるメトリクス 1233
クエリパターン 1234
getBucketsAggregation によるパッケージバージョン配布の収集 1236
AWS IoT ジョブの準備 1237
AWS IoT ジョブの置換パラメータ 1237
デプロイ用のジョブドキュメントとパッケージバージョンの準備 1241
デプロイ時のパッケージとバージョンの命名 1245
モノの AWS IoT 動的グループによるジョブのターゲット設定 1245
予約済みの名前付きシャドウとパッケージバージョン 1246
ソフトウェアパッケージのアンインストール 1247
開始方法 1247
パッケージとバージョンの作成 1248
パッケージバージョンのデプロイ 1250
パッケージバージョンの関連付け 1253

ジョブ 1255
AWS IoT ジョブへのアクセス 1255
AWS IoT ジョブのリージョンとエンドポイント 1255
リモートオペレーションとは 1256
リモートオペレーションに AWS IoT Device Management ジョブを使用する利点 1256
AWS IoT ジョブとは 1258
ジョブの主要な概念 1259
Jobs とジョブ実行の状態 1263
ジョブの管理 1269
ジョブのコード署名 1269
ジョブドキュメント 1269
署名付き URL 1270
ファイルアップロード用の署名付き URL 1272
Amazon S3 バージョニングを使用した署名付き URL 1273
コンソールを使用してジョブを作成および管理します。 1275
CLI を使用してジョブを作成および管理します。 1278
ジョブテンプレート 1290
カスタムテンプレートと AWS マネージドテンプレート 1290
AWS マネージドテンプレートを使用する 1291
カスタムジョブテンプレートを作成する 1311
ジョブの 設定 1319
ジョブ設定の仕組み 1320
追加の設定を指定する 1334
デバイスとジョブ 1343
ジョブを処理するデバイスのプログラミング 1346
デバイスのワークフロー 1346
ジョブワークフロー 1348
ジョブの通知 1352
AWS IoT ジョブ API オペレーション 1361
ジョブ管理と制御のデータ型 1363
ジョブデバイス MQTT および HTTPS API オペレーションとデータ型 1383
ジョブのユーザーおよびデバイスの保護 1398
AWS IoT ジョブに必要なポリシータイプ 1398
ジョブユーザーおよびクラウドサービスを承認する 1400
デバイスにジョブの使用を承認する 1413
AWS IoT ジョブの制限 1417

ジョブ実行の制限 1418
アクティブなジョブおよび同時ジョブの制限 1419
コマンド 1423
クイックスタート 1424
コマンドの概念とステータス 1424
コマンドキーの概念 1425
コマンドの状態 1429
コマンド実行ステータス 1430
コマンドワークフロー 1434
コマンドの作成と管理 1434
ターゲットを選択してトピックをサブスクライブする 1435
コマンド実行を開始してモニタリングする 1437
(オプション) コマンドイベントの通知を有効にする 1438
コマンドの作成と管理 1439
コマンドリソースを作成する 1439
コマンドに関する情報を取得する 1445
でコマンドを一覧表示する AWS アカウント 1447
コマンドリソースを更新する 1448
コマンドリソースを非推奨にする、または復元する 1451
コマンドリソースを削除する 1451
コマンド実行を開始してモニタリングする 1453
コマンド実行を開始する 1453
コマンド実行の結果を更新する 1460
コマンド実行を取得する 1466
MQTT テストクライアントを使用したコマンドの更新の表示 1470
でコマンド実行を一覧表示する AWS アカウント 1471
コマンド実行を削除する 1474
コマンドリソースを非推奨にする 1475
主な考慮事項 1476
コマンドリソースを非推奨にする (コンソール) 1476
コマンドリソースを非推奨にする(CLI) 1476
非推奨の時間とステータスを確認する 1477
コマンドリソースを復元する 1477
セキュアトンネリング 1479
セキュアトンネリングとは? 1479
セキュアトンネリングの概念 1479

セキュアトンネリングの仕組み 1481
安全なトンネルのライフサイクル 1482
セキュアトンネリングのチュートリアル 1483
このセクションのチュートリアル 1483
トンネルを開き、リモートデバイスへの SSH セッションを開始します 1484
リモートデバイス用のトンネルを開き、ブラウザベースの SSH を使用する 1502
ローカルプロキシ 1507
ローカルプロキシの使用方法 1507
ウェブプロキシを使用するデバイスのローカルプロキシを設定します 1514
多重化と同時 TCP 接続 1522
複数のデータストリームの多重化 1523
同時 TCP 接続の使用 1527
リモートデバイスの設定と IoT エージェントの使用 1530
IoT エージェントスニペット 1530
トンネルへのアクセスの制御 1532
トンネルアクセスの前提条件 1532
トンネルアクセスポリシー 1532
セキュアトンネリングの接続性の問題を解決する 1540
無効なクライアントアクセストークンのエラー 1540
クライアントトークンの不一致エラー 1541
リモートデバイスの接続性に関する問題 1542
デバイスプロビジョニング 1545
でのデバイスのプロビジョニング AWS IoT 1546
フリートプロビジョニング API 1547
フリートプロビジョニングを使用したデバイス証明書がないデバイスのプロビジョニング ... 1548
クレームによるプロビジョニング 1549
信頼されたユーザーによるプロビジョニング 1552
CLI AWS での事前プロビジョニングフックの使用 1554
デバイス証明書があるデバイスのプロビジョニング 1558
単一のモノプロビジョニング 1558
ジャストインタイムプロビジョニング 1559
一括登録 1565
プロビジョニングテンプレート 1566
Parameters セクション 1567
Resources セクション 1567
一括登録のテンプレート例 1573

ジャストインタイムプロビジョニング (JITP) のテンプレート例 1575
フリートプロビジョニング 1576
事前プロビジョニングフック 1580
事前プロビジョニングフックの入力 1581
事前プロビジョニングフックの戻り値 1581
事前プロビジョニングフック Lambda の例 1582
証明書プロバイダーを使用したセルフマネージド AWS IoT Core 証明書署名 1585
フリートプロビジョニングでのセルフマネージド証明書署名の仕組み 1586
証明書プロバイダーの Lambda 関数入力 1587
証明書プロバイダーの Lambda 関数の戻り値 1588
Lambda 関数の例 1588
フリートプロビジョニング用のセルフマネージド証明書署名 1590
AWS CLI 証明書プロバイダーの コマンド 1591
デバイスをインストールするユーザーの IAM ポリシーとロールの作成 1594
デバイスをインストールするユーザーの IAM ポリシーの作成 1595
デバイスをインストールするユーザーの IAM ロールの作成 1596
新しいテンプレートを許可するように既存のポリシーを更新する 1597
デバイスプロビジョニング MQTT API 1598
CreateCertificateFromCsr 1599
CreateKeysAndCertificate 1601
RegisterThing 1603
フリートインデックス作成 1607
インデックスの更新の管理 1607
特定のデバイスの接続ステータスのクエリ 1607
データソース全体での検索 1607
集計データのクエリ 1608
フリートメトリクスを使用した集計データのモニタリングとアラームの作成 1608
フリートのインデックス作成の管理 1608
モノのインデックス作成 1608
モノのグループのインデックス作成 1610
管理対象フィールド 1610
カスタムフィールド 1612
モノのインデックス作成の管理 1613
モノのグループのインデックス作成の管理 1629
デバイスの接続ステータス 1631
仕組み 1631

機能 1631
利点 1632
前提条件 1632
例 1633
集計データのクエリ 1634
GetStatistics 1635
GetCardinality 1637
GetPercentiles 1638
GetBucketsAggregation 1641
Authorization 1642
クエリ構文 1642
サポートされている機能 1642
サポートされていない 機能 1643
注意事項 1643
モノのクエリの例 1644
モノのグループのクエリの例 1648
位置データのインデックス作成 1650
サポートされているデータ形式 1650
位置データのインデックス作成方法 1651
モノのインデックス作成を設定する 1652
ジオクエリの例 1655
入門チュートリアル 1656
フリートメトリクス 1661
入門チュートリアル 1661
フリートのメトリクスの管理 1668
MQTT ベースのファイル配信 1676
ストリーミングとは 1676
ストリームを管理する 1677
デバイスにアクセス許可を付与する 1678
デバイスを に接続する AWS IoT 1679
TagResource の使用 1679
デバイスで AWS IoT MQTT ベースのファイル配信を使用する 1680
DescribeStream を使用してストリーミングデータを取得する 1681
ストリーミングファイルからデータブロックを取得する 1683
AWS IoT MQTT ベースのファイル配信によるエラーの処理 1689
FreeRTOS OTA のユースケースの例 1691

Device Advisor 1692
設定 1694
IoT モノの作成 1694
デバイスロールとして使用する IAM ロールを作成する 1694
IAM ユーザーが Device Advisor を使用するためのカスタムマネージドポリシーを作成する 1697
Device Advisor のテストの実行に使用する IAM ユーザーを作成する 1698
デバイスを設定する 1701
コンソールでの Device Advisor の開始方法 1702
Device Advisor ワークフロー 1712
前提条件 1712
テストスイート定義を作成する 1712
テストスイート定義を取得する 1715
テストエンドポイントを取得する 1716
テストスイートの実行を開始する 1716
テストスイートの実行を取得する 1717
テストスイートの実行を停止する 1717
成功した認定テストスイートの実行の認定レポートを取得する 1718
Device Advisor の詳細コンソールワークフロー 1718
前提条件 1719
テストスイート定義を作成する 1719
テストスイートの実行を開始する 1726
テストスイートの実行を停止する (オプション) 1728
テストスイートの実行の詳細とログを表示する 1729
AWS IoT 認定レポートをダウンロードする 1731
長期テストコンソールのワークフロー 1731
デバイスアドバイザー VPC エンドポイント (AWS PrivateLink) 1740
AWS IoT Core Device Advisor VPC エンドポイントに関する考慮事項 1740
のインターフェイス VPC エンドポイントを作成する AWS IoT Core Device Advisor 1741
VPC エンドポイント AWS IoT Core Device Advisor を介した へのアクセスの制御 1742
Device Advisor テストケース 1743
Device Qualification Program の対象となる AWS Device Advisor テストケース。 1744
TLS 1744
MQTT 1751
シャドウ 1766
ジョブの実行 1768

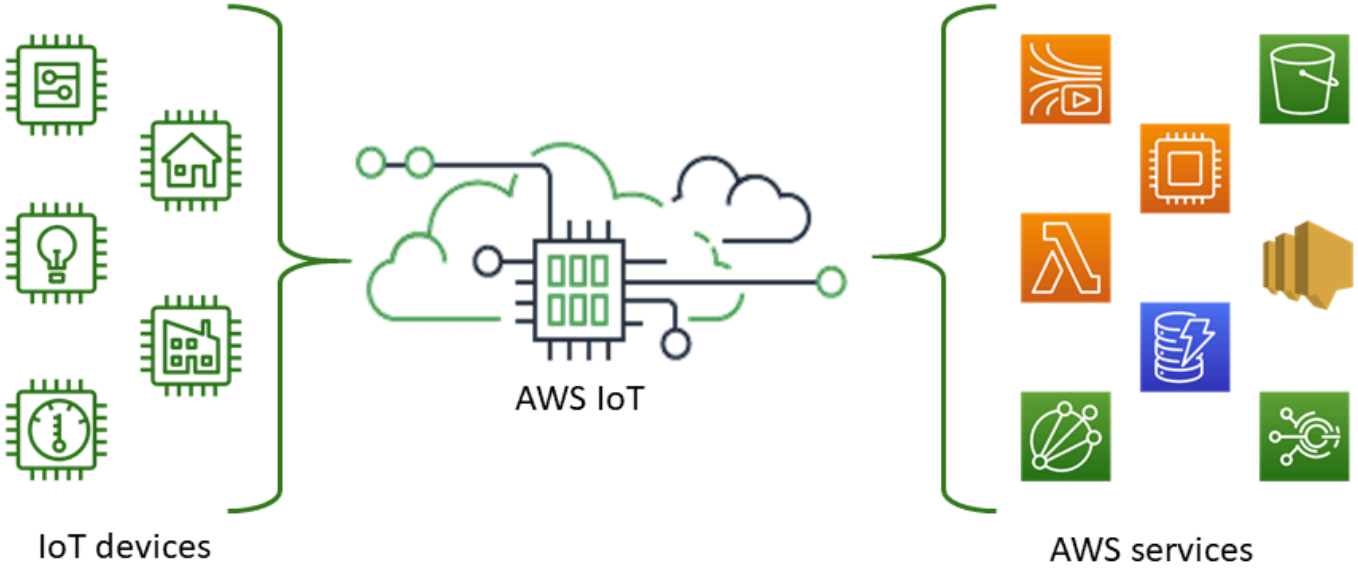
アクセス許可とポリシー 1770
長期テスト 1771
Device Location 1789
測定タイプとソルバー 1789
AWS IoT Core Device Location の仕組み 1791
AWS IoT Core Device Location の使用方法 1792
IoT デバイスの位置を解決する 1793
デバイスの位置を解決する (コンソール) 1793
デバイス位置の解決 (API) 1797
位置の解決時のトラブルシューティング 1798
MQTT トピックを使用したデバイス位置の解決 1800
デバイスの位置情報 MQTT トピックの形式 1800
デバイスの位置情報 MQTT トピックのポリシー 1801
デバイスの位置情報トピックとペイロード 1802
位置ソルバーとデバイスペイロード 1807
Wi-Fi ベースのソルバー 1808
セルラーベースのソルバー 1809
IP リバースルックアップソルバー 1813
GNSS ソルバー 1814
Amazon Sidewalk 対応デバイスの BLE ソルバー 1816
イベントメッセージ 1817
イベントメッセージが生成される方法 1817
イベントメッセージを受信するためのポリシー 1817
のイベントを有効にする AWS IoT 1818
登録イベント 1823
モノのイベント 1823
モノのタイプのイベント 1825
モノのグループのイベント 1828
ジョブイベント 1834
ライフサイクルイベント 1839
接続/切断イベント 1839
接続試行失敗イベント 1844
サブスクライブ/サブスクライブ解除イベント 1845
トラブルシューティング 1848
AWS IoT Core トラブルシューティングガイド 1848
接続関連の問題の診断 1849

ルール関連の問題の診断 1852
シャドウ関連の問題の診断 1855
Salesforce アクションの問題の診断 1857
ストリーム制限の診断 1858
デバイス群切断のトラブルシューティング 1859
AWS IoT Device Management トラブルシューティングガイド 1860
AWS IoT ジョブのトラブルシューティング 1860
フリートインデックス作成のトラブルシューティング 1865
AWS IoT デバイス管理ソフトウェアパッケージカタログのトラブルシューティング 1868
AWS IoT Device Advisor トラブルシューティングガイド 1875
AWS IoT エラー 1878
AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Client 1880
AWS IoTDevice SDKs 1880
AWS IoT Device SDK for Embedded C 1882
AWS Mobile SDK 1883
AWS IoT Device Client 1884
AWS IoT Device SDK の以前のバージョン 1885
コードの例 1886
基本 1887
こんにち AWS IoTは 1887
基本を学ぶ 1896
アクション 2007
AWS IoT のクォータ 2105
AWS IoT Core の料金 2106
..... mmcvii

とは AWS IoT

AWS IoT は、IoT デバイスを他のデバイスや AWS クラウドサービスに接続するクラウドサービスを提供します。 は、IoT デバイスを AWS IoTベースのソリューションに統合するのに役立つデバイスソフトウェア AWS IoT を提供します。デバイスが に接続できる場合 AWS IoT、 は AWS が提供するクラウドサービスに接続 AWS IoT できます。

の実践的な概要については AWS IoT、「」を参照してください開始方法のチュートリアル。



を使用すると AWS IoT、ソリューションに最適なup-to-dateテクノロジーを選択できます。フィールドで IoT デバイスを管理およびサポートしやすくするために、 は次のプロトコル AWS IoT Core をサポートしています。

- MQTT (Message Queuing and Telemetry Transport)
- MQTT over WSS (Websockets Secure)
- HTTPS (Hypertext Transfer Protocol - Secure)
- LoRaWAN (Long Range Wide Area Network)

AWS IoT Core メッセージブローカーは、MQTT および MQTT over WSS プロトコルを使用してメッセージを発行およびサブスクライブするデバイスとクライアントをサポートします。また、HTTPS プロトコルを使用してメッセージを発行するデバイスとクライアントもサポートしています。

AWS IoT Core for LoRaWAN は、ワイヤレス LoRaWAN (低電力長距離広域ネットワーク) デバイスの接続と管理に役立ちます。 AWS IoT Core for LoRaWAN は、LoRaWAN ネットワークサーバー (LNS) の開発と運用の必要性を置き換えます。

デバイス通信、ルール、ジョブなどの AWS IoT 機能が必要ない場合は、AWS 「メッセージング」を参照して、要件により適した他の AWS IoT メッセージングサービスに関する情報を確認してください。

デバイスとアプリのアクセス方法 AWS IoT

AWS IoT は、 に次のインターフェイスを提供しますAWS IoT チュートリアル。

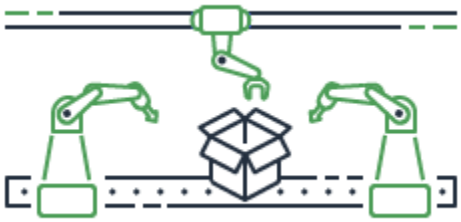
- AWS IoT Device SDKs- との間でメッセージを送受信するアプリケーションをデバイスに構築します AWS IoT。詳細については、「AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Client」を参照してください。
- AWS IoT Core for LoRaWAN — for LoRaWAN を使用して、長距離 WAN (LoRaWAN) デバイスとゲートウェイを接続して管理します。 AWS IoT Core LoRaWAN
- AWS Command Line Interface (AWS CLI) — Windows、macOS、Linux AWS IoT で のコマンドを実行します。これらのコマンドで、モノのオブジェクト、証明書、ルール、ジョブ、およびポリシーを作成し、管理することができます。開始するには、「AWS Command Line Interface ユーザーガイド」を参照してください。のコマンドの詳細については AWS IoT、「 コマンドリファレンス」の「iot」を参照してください。 AWS CLI
- AWS IoT API — HTTP または HTTPS リクエストを使用して IoT アプリケーションを構築します。これらの API アクションで、モノのオブジェクト、証明書、ルール、およびポリシーをプログラムにより作成し、管理することができます。の API アクションの詳細については AWS IoT、AWS IoT 「 API リファレンス」の「アクション」を参照してください。
- AWS SDKs- 言語固有の APIs を使用して IoT アプリケーションを構築します。これらの SDK は HTTP/HTTPS API をラップし、サポートされているいずれの言語でもプログラミングできます。詳細については、AWS の SDK およびツールを参照してください。

AWS IoT コンソール AWS IoT から にアクセスすることもできます。これにより、モノのオブジェクト、証明書、ルール、ジョブ、ポリシー、および IoT ソリューションのその他の要素を設定および管理できるグラフィカルユーザーインターフェイス (GUI) が提供されます。

AWS IoT できること

このトピックでは、 AWS IoT でサポートされており、かつ、お客様が必要とする可能性のあるソリューションの一部について説明します。

産業における IoT



以下は、産業プロセスのパフォーマンスと生産性を向上させるために IoT テクノロジーを適用する産業ユースケースの AWS IoT ソリューションの例です。

産業ユースケース向けソリューション

- AWS IoT を使用して産業オペレーションで予測品質モデルを構築する

  AWS IoT が産業運用からデータを収集して分析し、予測品質モデルを構築する方法について説明します。詳細はこちら
- を使用して AWS IoT 、産業オペレーションの予測メンテナンスをサポート

  AWS IoT が計画外のダウンタイムを削減するための予防メンテナンスの計画にどのように役立つかをご覧ください。詳細はこちら

ホームオートメーションの IoT



以下は、IoT テクノロジーを適用して、コネクテッドホームデバイスを使用して家庭のアクティビティを自動化するスケーラブルな IoT アプリケーションを構築する、ホームオートメーションのユースケース向けの AWS IoT ソリューションの例です。 IoT

ホームオートメーション向けソリューション

- 接続されたホーム AWS IoT で を使用する

  AWS IoT が統合されたホームオートメーションソリューションを提供する方法をご覧ください。

- AWS IoT を使用してホームセキュリティとモニタリングを提供する

  AWS IoT が機械学習とエッジコンピューティングをホームオートメーションソリューションに適用する方法をご覧ください。

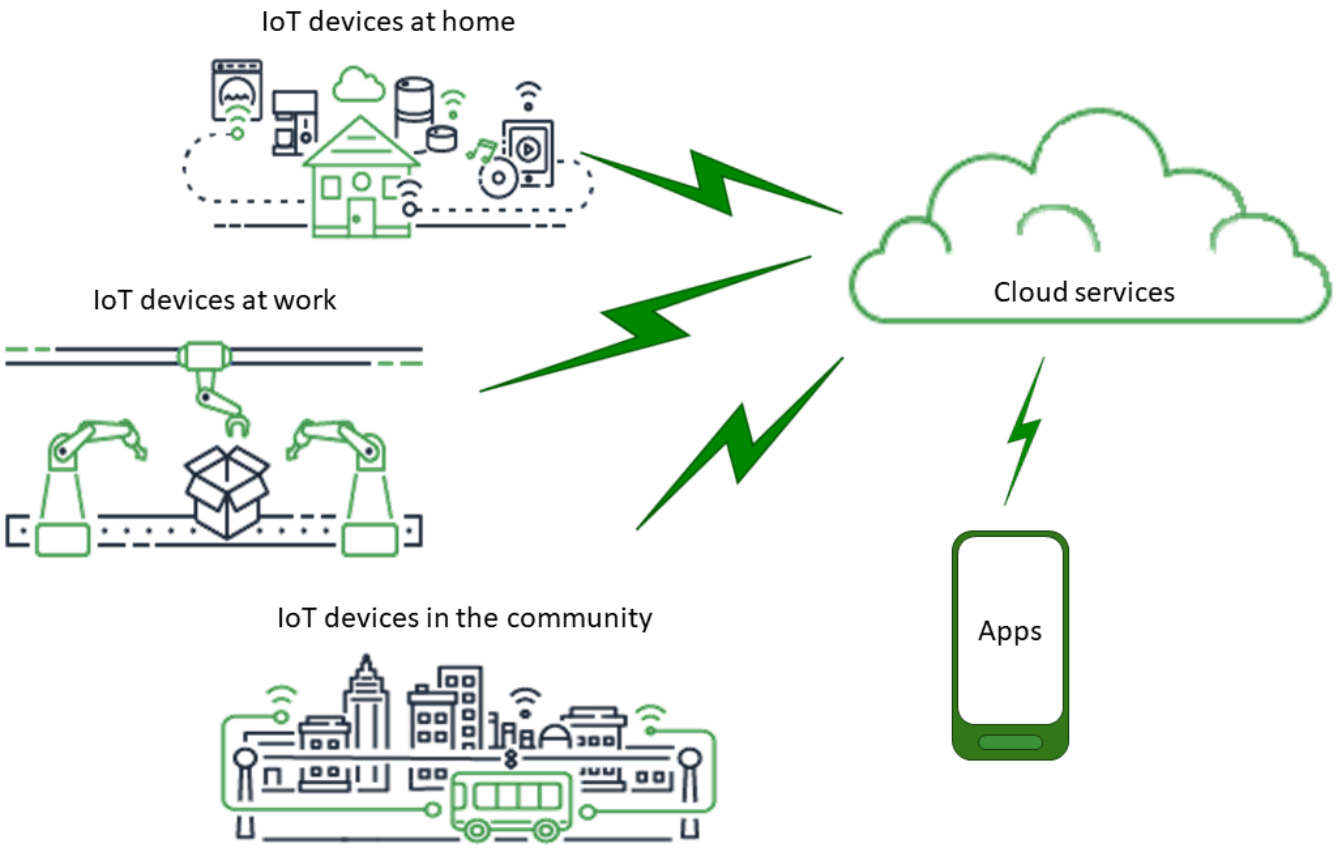
産業、消費者向け、および商用のユースケースのソリューションのリストについては、AWS IoT ソリューションリポジトリを参照してください。

の AWS IoT 仕組み

AWS IoT は、IoT ソリューションの実装に使用できるクラウドサービスとデバイスサポートを提供します。 は、IoT ベースのアプリケーションをサポートするために多くのクラウドサービス AWS を提供します。どこから始めればよいかを理解できるように、このセクションでは、IoT ユニバースの概要を知るための重要な概念の図と定義を示します。

IoT ユニバース

一般に、モノのインターネット (IoT) は、この図に示す主要なコンポーネントで構成されています。



アプリケーション

アプリケーションを使用すると、エンドユーザーは IoT デバイスと、それらのデバイスが接続されているクラウドサービスによって提供される機能にアクセスできます。

クラウドサービス

クラウドサービスは、インターネットに接続されている分散型の大規模データストレージおよび処理サービスです。以下に例を示します。

- IoT 接続および管理サービス

  AWS IoT は、IoT 接続および管理サービスの例です。
- Amazon Elastic Compute Cloud や などのコンピューティングサービス AWS Lambda
- Amazon DynamoDB などのデータベースサービス

通信

デバイスは、さまざまな技術やプロトコルを使用してクラウドサービスと通信します。その例を以下に示します。

- Wi-Fi/ブロードバンドインターネット
- ブロードバンドセルラーデータ
- 狭帯域セルラーデータ
- Long-range Wide Area Network (LoRaWAN)
- 独自の RF 通信

デバイス

デバイスとは、インターフェイスと通信を管理するハードウェアの一種です。デバイスは、通常、監視および制御対象の実際のインターフェイスの近くに配置されます。デバイスには、マイクロコントローラ、CPU、メモリなどのコンピューティングおよびストレージリソースを含めることができます。その例を以下に示します。

- Raspberry Pi
- Arduino
- 音声インターフェイスアシスタント
- LoRaWAN およびデバイス
- Amazon Sidewalk デバイス
- カスタム IoT デバイス

インターフェイス

インターフェイスは、デバイスを物理世界に接続するコンポーネントです。

- ユーザーインターフェイス

  デバイスとユーザーが相互に通信することを可能にするコンポーネント。

  - 入力インターフェイス

    ユーザーがデバイスと通信できるようにする

例: キーパッド、ボタン

- 出力インターフェイス

  デバイスがユーザーと通信できるようにする

  例：英数字表示、グラフィック表示、インジケーターライト、アラームベル

- センサー

  デバイスが理解できる方法で外部で何かを測定または感知する入力コンポーネント。以下に例を示します。

  - 温度センサー (温度をアナログ信号またはデジタル信号に変換)
  - 湿度センサー (相対湿度をアナログ信号またはデジタル信号に変換)
  - アナログ-デジタル変換器 (アナログ電圧を数値に変換)
  - 超音波距離測定単位 (距離を数値に変換)
  - 光学センサー (光レベルを数値に変換)
  - カメラ (画像データをデジタルデータに変換)

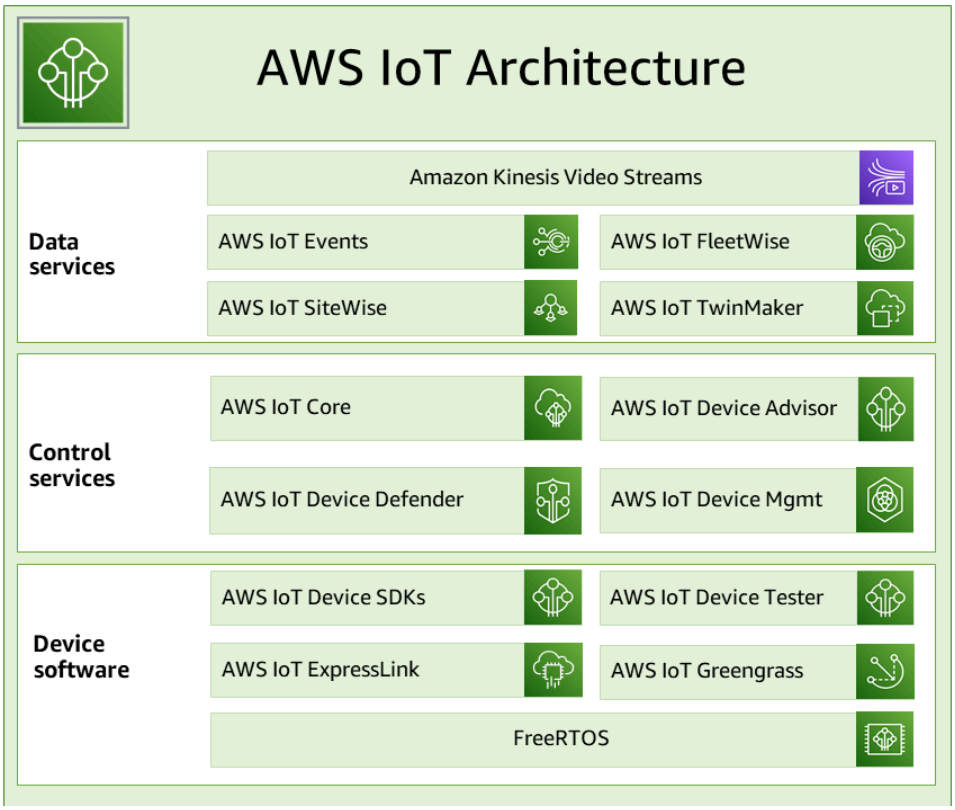
- アクチュエータ

  デバイスが外部で何かを制御するために使用できる出力コンポーネント。その例を以下に示します。

  - ステッピングモーター (電気信号を動きに変換)
  - リレー (高電圧と電流を制御)

AWS IoT サービスの概要

IoT ユニバースでは、 は、世界とやり取りするデバイスをサポートするサービスと、それらと の間を通過するデータ AWS IoT を提供します AWS IoT。 AWS IoT はIoT ソリューションをサポートするためにこの図に示すサービスで構成されます。



AWS IoT デバイスソフトウェア

AWS IoT は、 IoT デバイスをサポートするためにこのソフトウェアを提供します。

AWS IoT デバイス SDKs

AWS IoT Device SDK と Mobile SDKsは、デバイスを効率的に接続するのに役立ちます AWS IoT。 AWS IoT Device SDK と Mobile SDKs には、オープンソースライブラリ、サンプルを含む開発者ガイド、移植ガイドが含まれており、選択したハードウェアプラットフォームで革新的な IoT 製品またはソリューションを構築できます。

AWS IoT Device Tester

AWS IoT Device Tester for FreeRTOS および AWS IoT Greengrass は、マイクロコントローラー用のテスト自動化ツールです。 は、FreeRTOS または を実行し AWS IoT Greengrass 、 AWS IoT サービスと相互運用するかどうかを判断するためにデバイスを AWS IoT Device Tester テストします。

AWS IoT ExpressLink

AWS IoT ExpressLink は、 AWS パートナーによって開発および提供されるさまざまなハードウェアモジュールを強化します。接続モジュールには AWS検証済みソフトウェアが含まれて

いるため、デバイスをクラウドに安全に接続し、さまざまな AWS サービスとシームレスに統合できます。詳細については、AWS IoT Express Link 概要ページにアクセスするか、AWS IoT ExpressLink プログラマーガイドを参照してください。

AWS IoT Greengrass

AWS IoT Greengrass はエッジデバイスに AWS IoT を拡張し、生成したデータに対してローカルに動作し、機械学習モデルに基づいて予測を実行し、デバイスデータをフィルタリングおよび集計できるようにします。 AWS IoT Greengrass は、デバイスがそのデータが生成される場所に近いデータを収集および分析し、ローカルイベントに自律的に反応し、ローカルネットワーク上の他のデバイスと安全に通信できるようにします。を使用して AWS IoT Greengrass 、エッジデバイスを AWS サービスまたはサードパーティーサービスに接続できるコンポーネントと呼ばれる構築済みのソフトウェアモジュールを使用してエッジアプリケーションを構築できます。

FreeRTOS

FreeRTOS は、マイクロコントローラー用のオープンソースのリアルタイムオペレーティングシステムであり、IoT ソリューションに小型で低電力のエッジデバイスを含めることを可能にします。FreeRTOS には、カーネルと、多くのアプリケーションをサポートする一連のソフトウェアライブラリが含まれています。FreeRTOS システムは、小型で低電力のデバイスを AWS IoT に安全に接続し、AWS IoT Greengrass を実行するより強力なエッジデバイスをサポートできます。

AWS IoT コントロールサービス

次の AWS IoT サービスに接続して、IoT ソリューションのデバイスを管理します。

AWS IoT Core

AWS IoT Core は、接続されたデバイスがクラウドアプリケーションやその他のデバイスと安全にやり取りできるようにするマネージドクラウドサービスです。 は多くのデバイスやメッセージをサポート AWS IoT Core でき、それらのメッセージを AWS IoT エンドポイントやその他のデバイスに処理してルーティングできます。を使用すると AWS IoT Core、アプリケーションは接続されていない場合でもすべてのデバイスを操作できます。

AWS IoT Core Device Advisor

AWS IoT Core Device Advisor は、デバイスソフトウェア開発中に IoT デバイスを検証するためのクラウドベースのフルマネージドテスト機能です。Device Advisor は、デバイスを本稼働環境にデプロイする前に AWS IoT Core、IoT デバイスとの信頼性と安全な接続を検証するために使用できる構築済みのテストを提供します。

AWS IoT Device Defender

AWS IoT Device Defender は、IoT デバイスのフリートを保護するのに役立ちます。 AWS IoT Device Defender はIoT 設定を継続的に監査して、セキュリティのベストプラクティスから逸脱していないことを確認します。 AWS IoT Device Defender は、複数のデバイス間で共有されている ID 証明書や、 に接続しようとしている ID 証明書が取り消されたデバイスなど、セキュリティリスクを引き起こす可能性のある IoT 設定のギャップを検出すると、アラートを送信しますAWS IoT Core。

AWS IoT デバイス管理

AWS IoT デバイス管理サービスは、デバイスフリートを構成する多数の接続デバイスの追跡、モニタリング、管理に役立ちます。 AWS IoT デバイス管理サービスは、IoT デバイスがデプロイされた後に適切かつ安全に動作することを確実にするのに役立ちます。また、デバイスへのアクセスのためのセキュアなトンネリングを提供し、デバイスの正常性を監視し、問題を検出してリモートでトラブルシューティングし、デバイスのソフトウェアとファームウェアの更新を管理するサービスも提供します。

AWS IoT データサービス

IoT ソリューション内のデバイスからのデータを分析し、次の AWS IoT サービスを使用して適切なアクションを実行します。

Amazon Kinesis Video Streams

Amazon Kinesis Video Streams を使用すると、デバイスから AWS クラウドにライブビデオをストリーミングできます。このビデオは永続的に保存、暗号化、インデックス作成され、easy-to-useを介してデータにアクセスできます。 APIs Amazon Kinesis Video Streams を使用すると、スマートフォン、セキュリティカメラ、ウェブカメラ、車、ドローンやその他のソースに設置されるカメラのような何百万ものソースからライブ動画データの膨大な量を取得できます。Amazon Kinesis Video Streams を使用すると、ライブおよびオンデマンド視聴のために動画を再生し、Amazon Rekognition Video と ML フレームワーク用のライブラリとの統合を通じて、コンピュータービジョンと動画分析を活用するアプリケーションをすばやく構築できます。また、オーディオデータ、熱画像、深度データ、RADAR データなどの多くの時刻シリアル化された非ビデオデータも送信できます。

Amazon Kinesis Video Streams with WebRTC

Amazon Kinesis Video Streams with WebRTC は、標準に準拠した WebRTC をフルマネージド機能として実装しています。Amazon Kinesis Video Streams with WebRTC を使用して、メディア

を安全にライブストリーミングしたり、カメラ IoT デバイスと WebRTC 準拠のモバイルプレーヤーやウェブプレーヤー間で、双方向のオーディオ対話やビデオ対話を実行したりできます。フルマネージド機能を実装しているため、アプリケーションとデバイス間でメディアを安全にストリーミングするために、シグナリングサーバーやメディアリレーサーバーなど、WebRTC 関連のクラウドインフラストラクチャを構築、運用、または拡張する必要はありません。Amazon Kinesis Video Streams with WebRTC を使用すると、ライブピアツーピアメディアストリーミング、またはカメラ IoT デバイス、Web ブラウザ、およびモバイルデバイス間のリアルタイムのオーディオ対話やビデオ対話のためのアプリケーションをさまざまなユースケースで簡単に構築できます。

AWS IoT イベント

AWS IoT イベントは、IoT センサーとアプリケーションからのイベントを検出して応答します。イベントは、照明やセキュリティカメラをアクティブ化するために移動信号を使用するモーションディテクターなど、予想よりも複雑な状況を識別するデータのパターンです。 AWS IoT イベントは、複数の IoT センサーやアプリケーションからのデータを継続的にモニタリングし、IoT SiteWise、DynamoDB などの他のサービスと統合して AWS IoT Core、早期検出や独自のインサイトを可能にします。

AWS IoT FleetWise

AWS IoT FleetWise は、車両データを収集してほぼリアルタイムでクラウドに転送するために使用できるマネージドサービスです。 AWS IoT FleetWise を使用すると、さまざまなプロトコルとデータ形式を使用する車両からデータを簡単に収集して整理できます。 AWS IoT FleetWise は、低レベルのメッセージを人間が読める値に変換し、データ分析のためにクラウドのデータ形式を標準化するのに役立ちます。また、データ収集スキームを定義して、車両で収集するデータとクラウドに転送するタイミングを制御することもできます。

AWS IoT SiteWise

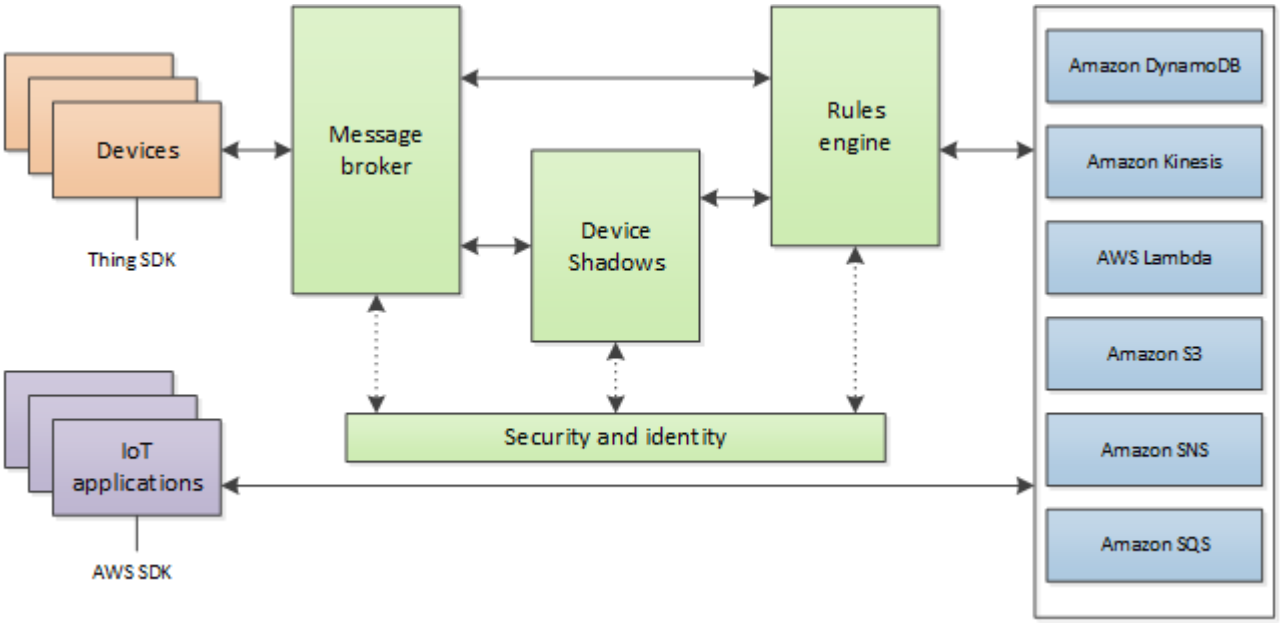
AWS IoT SiteWise は、施設内のゲートウェイ上で動作するソフトウェアを提供することにより、MQTT メッセージまたは API によって産業機器から渡されたデータを大規模に収集、保存、整理、監視します。ゲートウェイはオンプレミスのデータサーバーに安全に接続し、データを収集して整理し、 AWS クラウドに送信するプロセスを自動化します。

AWS IoT TwinMaker

AWS IoT TwinMaker は、物理システムとデジタルシステムの運用デジタルツインを構築します。AWS IoT TwinMaker は、さまざまな実世界のセンサー、カメラ、エンタープライズアプリケーションの測定値と分析を使用してデジタル視覚化を作成し、物理的な工場、建物、または産業プラントを追跡するのに役立ちます。実際のデータを使用して、オペレーションのモニタリング、エラーの診断と修正、およびオペレーションの最適化を行うことができます。

AWS IoT Core サービス

AWS IoT Core は、IoT デバイスを AWS クラウドに接続するサービスを提供し、他のクラウドサービスやアプリケーションがインターネットに接続されたデバイスとやり取りできるようにします。



次のセクションでは、図に示す各 AWS IoT Core サービスについて説明します。

AWS IoT Core メッセージングサービス

AWS IoT Core 接続サービスは、IoT デバイスとの安全な通信を提供し、デバイスとデバイス間を通過するメッセージを管理します AWS IoT。

デバイスゲートウェイ

デバイスから AWS IoTへのセキュアかつ効率的な通信を可能にします。デバイス通信は X.509 証明書を使用するセキュアなプロトコルによって保護されます。

メッセージブローカー

デバイスと AWS IoT アプリケーションが相互にメッセージを発行および受信するための安全なメカニズムを提供します。パブリッシュとサブスクライブには、MQTT を直接または WebSocket を介して使用できます。 AWS IoT がサポートするプロトコルの詳細については、the section called “デバイス通信プロトコル” を参照してください。デバイスおよびクライアントは、HTTP REST インターフェイスを使用して、メッセージをブローカーにデータを発行することもできます。

メッセージブローカーは、デバイスデータをサブスクライブしているデバイスと、Device Shadow AWS IoT Core サービスやルールエンジンなどの他のサービスに配信します。

AWS IoT Core for LoRaWAN

AWS IoT Core for LoRaWAN を使用すると、LoRaWAN ネットワークサーバー (LNS) を開発および運用することなくLoRaWAN デバイスとゲートウェイを AWS に接続することで、プライベート LoRaWAN ネットワークを設定できます。LoRaWAN デバイスから受信したメッセージは、ルールエンジンに送信され、そこでフォーマットされて、他の AWS IoT サービスに送信されます。

ルールエンジン

ルールエンジンは、ストレージと追加処理のために、メッセージブローカーから他の AWS IoT サービスにデータを接続します。例えば、DynamoDB テーブルを挿入、更新、または照会したり、Rules エンジンで定義した式に基づいて Lambda 関数を呼び出したりできます。SQL ベースの言語を使用して、メッセージペイロードからデータを選択し、データを処理して、他のサービス (Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB、 AWS Lambdaなど) にデータを送信できます。メッセージブローカーや他のサブスクライバーにメッセージを再発行するルールを作成することもできます。詳細については、「のルール AWS IoT」を参照してください。

AWS IoT Core コントロールサービス

AWS IoT Core コントロールサービスは、デバイスのセキュリティ、管理、登録機能を提供します。

カスタム認証サービス

カスタム認証サービスと Lambda 関数を使用して、独自の認証および認可戦略を管理できるカスタムオーソライザーを定義できます。カスタムオーソライザーにより、 AWS IoT はデバイスを認証し、ベアラトークンの認証と認可戦略を使用してオペレーションを認可することができます。

カスタムオーソライザーは、JSON Web Token の検証や OAuth プロバイダーのコールアウトなど、さまざまな認証戦略を実装できます。MQTT オペレーションを認可するためにデバイスゲートウェイが使用するポリシー文書を戻す必要があります。詳細については、「カスタム認証と認可」を参照してください。

デバイスプロビジョニングサービス

デバイスに必要なリソース (モノのオブジェクト、証明書、1 つ以上のポリシー) を記述したテンプレートを使用して、デバイスをプロビジョニングできます。モノのオブジェクトは、デバイ

スを記述する属性を含むレジストリのエントリです。デバイスは証明書を使用して認証します AWS IoT。ポリシーは、デバイスが AWS IoTで実行できるオペレーションを決定します。

テンプレートには、ディクショナリ (マップ) の値で置き換えられる変数が含まれています。同じテンプレートを使用して、ディクショナリのテンプレート変数に異なる値を渡すだけで、複数のデバイスをプロビジョニングすることができます。詳細については、「デバイスプロビジョニング」を参照してください。

グループレジストリ

グループはこれらをグループに分類することで、複数のデバイスを一度に管理できるようにします。グループ内にもグループが含まれているので、グループ階層を構築することもできます。親グループで行った操作は、子グループでも適用されます。親グループ内のすべてのデバイスおよび子グループ内のすべてのデバイスにも同じアクションが適用されます。グループに付与されたアクセス許可は、グループ内のすべてのデバイスとそのすべての子グループに適用されます。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。

ジョブサービス

AWS IoTに接続された 1 つ以上のデバイスに送信され実行される一連のリモート操作を定義できます。たとえば、一連のデバイスに対して、アプリケーションやファームウェア更新のダウンロードとインストール、再起動、証明書のローテーション、またはリモートトラブルシューティングオペレーションの実行を指示するジョブを定義できます。

ジョブを作成するには、実行するリモートオペレーションの説明と、それを実行するターゲットのリストを指定します。ターゲットは個々のデバイス、グループ、またはその両方にすることができます。詳細については、「AWS IoT ジョブ」を参照してください。

[Registry]

AWS クラウドで各デバイスに関連付けられたリソースの整理に使用されます。デバイスを登録し、各デバイスに最大 3 つのカスタム属性を関連付けることができます。また、各デバイスに証明書と MQTT クライアント ID を関連付けて、デバイスの管理とトラブルシューティングの機能を強化することもできます。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。

セキュリティとアイデンティティサービス

AWS クラウドのセキュリティに関する責任を共有します。デバイスは、メッセージブローカーにデータをセキュアに送信するために、認証情報を安全な場所に保管する必要があります。メッセージブローカーとルールエンジンは、 AWS のセキュリティ機能を使用して、デバイスまたは

他の AWS のサービスにデータをセキュアに送信します。詳細については、「認証」を参照してください。

AWS IoT Core データサービス

AWS IoT Core データサービスは、必ずしも接続されていないデバイスでも、IoT ソリューションが信頼性の高いアプリケーションエクスペリエンスを提供するのに役立ちます。

デバイスシャドウ

デバイスの現在の状態情報の保存と取得に使用される JSON ドキュメントです。

Device Shadow サービス

Device Shadow サービスはデバイスの状態を維持し、デバイスがオンラインであるかどうかにかかわらず、アプリケーションがデバイスと通信できるようにします。デバイスがオフラインの場合、Device Shadow サービスは接続されたアプリケーションのためにデータを管理します。デバイスが再接続すると、Device Shadow サービス内のシャドウの状態と同期します。デバイスは、常に接続されていない可能性のあるアプリケーションやその他のデバイスで使用できるように、現在の状態をシャドウに発行することもできます。詳細については、「AWS IoT Device Shadow サービス」を参照してください。

AWS IoT Core サポートサービス

の Amazon Sidewalk 統合 AWS IoT Core

Amazon Sidewalk は、接続オプションを改善してデバイスの連携を改善する共有ネットワークです。Amazon Sidewalk は、ペットや貴重品を探すデバイス、スマートホームセキュリティと照明制御を提供するデバイス、家電製品やツールのリモート診断を提供するデバイスなど、お客様のさまざまなデバイスをサポートしています。の Amazon Sidewalk 統合 AWS IoT Core により、デバイスメーカーは Sidewalk デバイスフリートを AWS IoT クラウドに追加できます。

詳細については、「AWS IoT Core for Amazon Sidewalk」を参照してください。

の詳細 AWS IoT

このトピックは、 の世界を理解するのに役立ちます AWS IoT。IoT ソリューションがさまざまなユースケースにどのように適用されるかに関する一般的な情報、トレーニングリソース、 AWS IoT

およびその他すべての AWS サービスのソーシャルメディアへのリンク、および が AWS IoT 使用するサービスと通信プロトコルのリストを取得できます。

のトレーニングリソース AWS IoT

これらのトレーニングコースは、 AWS IoT とそのソリューション設計への適用方法を学ぶのに役立ちます。

- の概要 AWS IoT

  AWS IoT とそのコアサービスの動画概要。
- AWS IoT の認証と認可について詳しく知る事

  AWS IoT の認証と認可の概念を深く学ぶ上級コース。クライアントが AWS IoT コントロールプレーン API とデータプレーン API にアクセスすることを認証および認可する方法を学びます。
- モノのインターネット (IoT) 基礎シリーズ

  さまざまな IoT テクノロジーと機能に関する IoT eLearning モジュールのラーニングパス。

AWS IoT リソースとガイド

これらは、 の特定の側面に関する詳細な技術リソースです AWS IoT。

- IoT レンズ – AWS IoT Well-Architected フレームワーク

  IoT アプリケーションを設計するためのベストプラクティスを説明するドキュメント AWS。
- の MQTT トピックの設計 AWS IoT Core

  で MQTT トピックを設計 AWS IoT Core し、MQTT で AWS IoT Core 機能を活用するためのベストプラクティスを説明するホワイトペーパー。
- 要約と概要

  が大規模なデバイスのフリートをプロビジョニング AWS IoT するためのさまざまな方法を説明した PDF ドキュメント。
- AWS IoT Core Device Advisor

  AWS IoT Core Device Advisor は、デバイスを本番環境にデプロイする前に AWS IoT Core、信頼性が高く安全な接続のベストプラクティスについて IoT デバイスを検証するために使用できる事前構築されたテストを提供します。

- AWS IoT リソース

  検索可能なインデックスで表示で表示される、技術ガイド、リファレンスアーキテクチャ、eBooks、厳選されたブログ投稿など、IoT 固有のリソース。

- IoT Atlas

  一般的な IoT の設計の問題を解決する方法の概要。IoT Atlasは、IoT ソリューションの開発中に直面する可能性のある設計上の課題を詳細に調査します。

- AWS ホワイトペーパーとガイド

  およびその他の AWS テクノロジーに関するホワイトペーパー AWS IoT とガイドの現在のコレクション。

AWS IoT ソーシャルメディアで

これらのソーシャルメディアチャネルは、 AWS IoT および AWS関連トピックに関する情報を提供します。

- のモノのインターネット AWS IoT — 公式ブログ
- AWS IoT YouTube の Amazon Web Services チャンネルの ビデオ

これらのソーシャルメディアアカウントは、以下を含むすべての AWS サービスを対象としています。 AWS IoT

- YouTube のアマゾン ウェブ サービスのチャンネル
- Twitter でのアマゾン ウェブ サービス
- Facebook でのアマゾン ウェブ サービス
- Instagram でのアマゾン ウェブ サービス
- LinkedIn でのアマゾン ウェブ サービス

AWS AWS IoT Core ルールエンジンで使用される サービス

AWS IoT Core ルールエンジンは、これらの AWS サービスに接続できます。

- Amazon DynamoDB

Amazon DynamoDB は、スケーラブルな NoSQL データベースサービスであり、高速で予測可能なデータベースパフォーマンスが特長です。

- Amazon Kinesis

  Amazon Kinesis では、リアルタイムのストリーミングデータを簡単に収集、処理、分析できるため、タイムリーな洞察を得て、新しい情報に迅速に対応できます。Amazon Kinesis は、動画、音声、アプリケーションログ、ウェブサイトのクリックストリーミング、機械学習、分析、その他のアプリケーション用の IoT テレメトリデータなどのリアルタイムデータを取り込むことができます。

- AWS Lambda

  AWS Lambda では、サーバーのプロビジョニングや管理を行わずにコードを実行できます。データ AWS IoT やイベントから自動的にトリガーしたり、ウェブやモバイルアプリから直接呼び出すようにコードを設定できます。

- Amazon Simple Storage Service

  Amazon Simple Storage Service (Amazon S3) は、ウェブ上のどこからでも、いつでも任意の量のデータを保存および取得できます。 AWS IoT ルールは、データを Amazon S3 に送信して保存できます。

- Amazon Simple Notification Service

  Amazon Simple Notification Service (Amazon SNS) は、アプリケーション、エンドユーザー、およびデバイスでクラウドから通知を送受信できるようにするウェブサービスです。

- Amazon Simple Queue Service

  Amazon Simple Queue Service (Amazon SQS) は、メッセージキューイングサービスであり、マイクロサービス、分散システム、およびサーバーレスアプリケーションの疎結合化とスケールを行います。

- Amazon OpenSearch Service

  Amazon OpenSearch Service (OpenSearch Service) は、一般的なオープンソースの検索および分析エンジンである OpenSearch のデプロイ、運用、およびスケーリングを容易にするマネージドサービスです。

- Amazon SageMaker AI

Amazon SageMaker AI は、IoT データのパターンを検出して、機械学習 (ML) モデルを作成できます。このサービスは、これらのモデルを使用して新しいデータを処理し、アプリケーションのために予測を生成します。

- Amazon CloudWatch

  Amazon CloudWatch は、独自のモニタリングシステムおよびインフラストラクチャのセットアップ、管理、およびスケールを支援する、信頼性、拡張性、および柔軟性を備えたモニタリングソリューションを提供します。

でサポートされている通信プロトコル AWS IoT Core

これらのトピックでは、 AWS IoTで使用される通信プロトコルについての詳細情報を示します。で使用されるプロトコル AWS IoT 、およびデバイスとサービスの接続の詳細については AWS IoT、「」を参照してくださいに接続する AWS IoT Core。

- MQTT (Message Queuing Telemetry Transport)

  MQTT プロトコル仕様を閲覧できる MQTT.org サイトのホームページ。が MQTT AWS IoT をサポートする方法の詳細については、「」を参照してくださいMQTT。

- HTTPS (Hypertext Transfer Protocol - Secure)

  デバイスとアプリケーションは、HTTPS を使用して AWS IoT サービスにアクセスできます。

- LoRaWAN (Long Range Wide Area Network)

  LoRaWAN デバイスとゲートウェイは、 AWS IoT Core for LoRaWAN AWS IoT Core を使用してに接続できます。

- TLS (Transport Layer Security) v1.3

  TLS v1.3 (RFC 5246) の仕様。 は TLS v1.3 AWS IoT を使用して、デバイスと 間の安全な接続を確立します AWS IoT。

AWS IoT コンソールの新機能

AWS IoT コンソールのユーザーインターフェイスを新しいエクスペリエンスに更新中です。ユーザーインターフェイスを段階的に更新しているため、コンソールの一部のページには新しいエクスペ

リエンスがあり、一部のページには元のエクスペリエンスと新しいエクスペリエンスの両方があり、一部のページには元のエクスペリエンスしかない場合があります。

この表は、2022 年 1 月 27 日時点の AWS IoT コンソールのユーザーインターフェイスの個々の領域の状態を示しています。

AWS IoT コンソールのユーザーインターフェイスの状態

コンソールページ	元のエクスペリエンス	新しいエクスペリエンス	コメント
モニタリング	利用不可	利用可能	
アクティビティ	利用不可	使用可能	
Onboard (オンボード) - 使用を開始する	利用不可	利用可能	CN リージョンでは使用できません
Onboard (オンボード) - フリートプロビジョニングテンプレート	利用可能	利用可能	
管理 - モノ	使用可能	使用可能	
Manage (管理) - タイプ	使用可能	使用可能	
Manage (管理) - モノのグループ	使用可能	使用可能	
Manage (管理) - 請求グループ	使用可能	使用可能	
管理 - ジョブ	使用可能	使用可能	
Manage (管理) - ジョブテンプレート	利用不可	利用可能	
Manage (管理) - トンネル	利用不可	使用可能	

コンソールページ	元のエクスペリエンス	新しいエクスペリエンス	コメント
Fleet Hub -使用を開始します。	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Fleet Hub -アプリケーション	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Greengrass - 使用を開始する	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Greengrass - コアデバイス	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Greengrass - コンポーネント	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Greengrass - デプロイ	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Greengrass - Classic (V1)	利用可能	利用可能	

コンソールページ	元のエクスペリエンス	新しいエクスペリエンス	コメント
Wireless connectivity (ワイヤレス接続) - はじめに	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Wireless connectivity (ワイヤレス接続) - ゲートウェイ	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Wireless connectivity (ワイヤレス接続) - デバイス	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Wireless connectivity (ワイヤレス接続) - プロファイル	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Wireless connectivity (ワイヤレス接続) - 送信先	利用不可	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Secure (安全性) - 証明書	利用可能	利用可能	
Secure (安全性) - ポリシー	利用可能	利用可能	
Secure (安全性) - CA	利用可能	利用可能	
Secure (安全性) - ロールのエイリアス	利用可能	利用可能	

コンソールページ	元のエクスペリエンス	新しいエクスペリエンス	コメント
Secure (安全性) - オーソライザー	利用可能	利用可能	
Defend (防御) - はじめに	利用不可	利用可能	
Defend (防御) - 監査	利用不可	利用可能	
Defend (防御) - 検出	利用不可	利用可能	
Defend (防御) - 緩和アクション	利用不可	利用可能	
Defend (防御) - 設定	利用不可	利用可能	
Act (実行) - ルール	利用可能	利用可能	
Act (実行) - 送信先	利用可能	利用可能	
Test (テスト) - Device Advisor	利用可能	利用可能	すべての AWS リージョン で利用可能なわけではありません。
Test (テスト) - MQTT テストクライアント	使用可能	使用可能	
ソフトウェア	利用可能	利用可能	
設定：	利用不可	利用可能	
学習	使用可能	まだ利用できません。	

凡例

ステータス値

- 利用可能

  このユーザーインターフェイスエクスペリエンスを使用できます。
- 利用不可

  このユーザーインターフェイスエクスペリエンスは使用できません。
- まだ利用できません。

  新しいユーザーインターフェイスエクスペリエンスは準備中ですが、まだ準備は完了していません。
- 進行中

  新しいユーザーインターフェイスエクスペリエンスは、更新中です。ただし、ページによっては元のユーザーエクスペリエンスが残っている場合があります。

AWS SDK AWS IoT での の使用

AWS Software Development Kit (SDKsは、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようになる API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例

SDK ドキュメント	コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback] リンクから、コードの例をリクエストしてください。

AWS IoT Core チュートリアルの開始方法

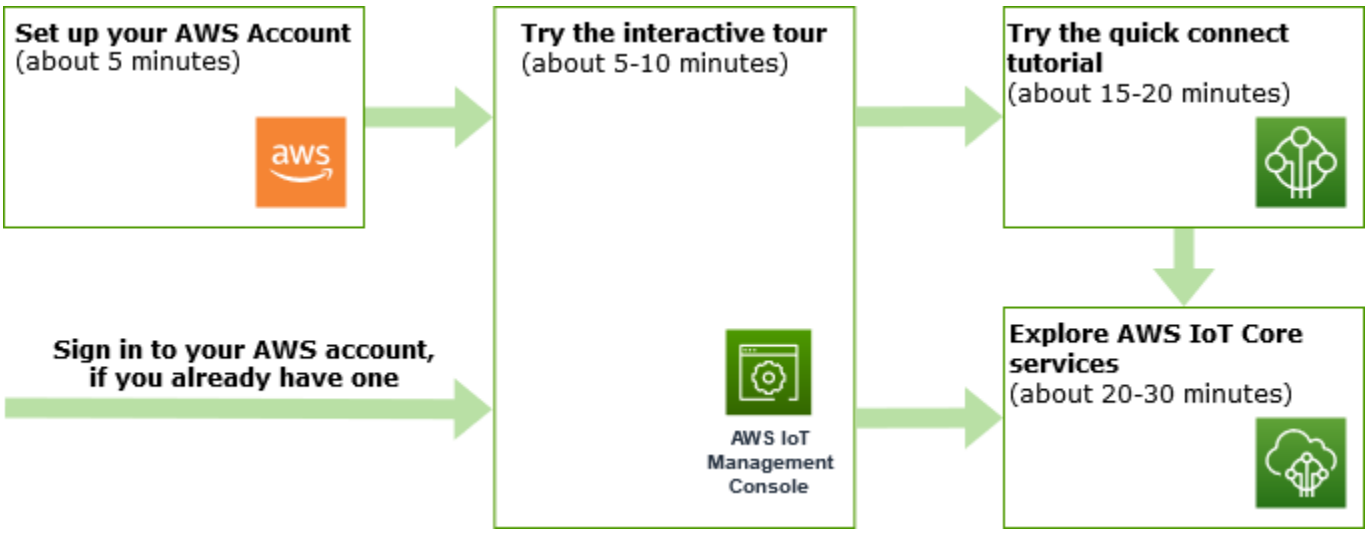
IoT を初めて使用する場合でも、長年の経験がある場合でも、これらのリソースには、 の使用を開始するのに役立つ AWS IoT 概念と用語が記載されています AWS IoT。

- AWS IoT とそのコンポーネントを で確認しますの AWS IoT 仕組み。
- トレーニング資料と動画のコレクションから、AWS IoTの詳細をご覧ください。このトピックには、 AWS IoT が接続できるサービスのリスト、ソーシャルメディアのリンク、通信プロトコル仕様へのリンクも含まれています。
- the section called “最初のデバイスを に接続する AWS IoT Core”.
- に接続する AWS IoT Core および AWS IoT チュートリアルを詳しく調べて、IoT ソリューションを開発します。
- Device Advisor を使用して、安全で信頼性の高い通信について IoT デバイスをテストおよび検証します。
- フリートインデックス作成、AWS IoT ジョブ、AWS IoT Device Defender などの AWS IoT Core 管理サービスを使用してソリューションを管理します。
- AWS IoT データサービスを使用して、デバイスからのデータを分析します。

最初のデバイスを に接続する AWS IoT Core

AWS IoT Core サービスは IoT デバイスを AWS IoT サービスや他の AWS サービスに接続します。AWS IoT Core には、IoT デバイスとクラウド間のメッセージを接続して処理する IoT デバイスゲートウェイとメッセージブローカーが含まれています。

AWS IoT Core と の使用を開始する方法は次のとおりです AWS IoT。



このセクションでは、 の主要サービスを紹介 AWS IoT Core する のツアーと、デバイスを に接続 AWS IoT Core してそれらの間でメッセージを渡す方法のいくつかの例を示します。デバイスとクラウド間でメッセージを渡すことは、すべての IoT ソリューションの基本であり、デバイスが他の AWS サービスとやり取りする方法でもあります。

- セットアップ AWS アカウント

  AWS IoT サービスを使用する前に、 を設定する必要があります AWS アカウント。と AWS アカウント IAM ユーザーがすでにある場合は、それらを使用してこのステップをスキップできます。
- クイックコネクトチュートリアルを試す

  このチュートリアルは、 をすばやく開始 AWS IoT し、限られたシナリオでどのように機能するかを確認する場合に最適です。このチュートリアルでは、デバイスが必要で、そのデバイスに AWS IoT ソフトウェアをインストールします。IoT デバイスをお持ちでない場合は、このチュートリアルのデバイスとして Windows、Linux、または macOS のパーソナルコンピュータを使用できます。試したい AWS IoTがデバイスがない場合は、次のオプションを試してください。
- インタラクティブチュートリアルを試す

  このデモは、デバイスを接続したりソフトウェアをダウンロードしたりすることなく、基本的な AWS IoT ソリューションで何ができるかを確認したい場合に最適です。インタラクティブチュートリアルでは、 AWS IoT Core サービス上に構築されたシミュレートされたソリューションを紹介します。このソリューションは、サービスがどのようにやり取りするかを示します。
- 実践的なチュートリアルで AWS IoT Core のサービスを調べる

  このチュートリアルは、ルールエンジンやシャドウなどの他の AWS IoT Core 機能を引き続き探索 AWS IoT できるように、 の使用を開始する開発者に最適です。このチュートリアルは、クイック

コネクトチュートリアルと同様の手順に従いますが、各ステップをさらに詳しく説明し、より高度なチュートリアルへのよりスムーズな移行を可能にします。

- MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する

  MQTT テストクライアントの使用方法を学び、最初のデバイスが MQTT メッセージを AWS IoTに発行するのを観察します。MQTT テストクライアントは、デバイス接続の監視とトラブルシューティングに役立つツールです。

> **Note**
>
> これらの開始方法のチュートリアルを複数試したり、同じチュートリアルを繰り返したりする場合は、前のチュートリアルで作成したモノのオブジェクトを削除してから、別のチュートリアルを開始してください。前のチュートリアルからモノのオブジェクトを削除しない場合は、以降のチュートリアルで別のモノの名前を使用する必要があります。これは、モノの名前は、アカウントおよび AWS リージョン内で一意でなければならないためです。

詳細については AWS IoT Core、「とは AWS IoT Core」を参照してください。

セットアップ AWS アカウント

AWS IoT Core を初めて使用する場合は、事前に以下のタスクを完了してください。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- AWS IoT コンソールを開く

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS マネジメントコンソールとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、「AWS サインイン ユーザーガイド」の「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、AWS IAM Identity Center 「 ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン 「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セットを作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループを追加する」を参照してください。

- AWS IoT コンソールを開く

と AWS アカウント ユーザーがすでにある場合は、それらを使用して にスキップできますthe section called “AWS IoT コンソールを開く”。

AWS IoT コンソールを開く

このセクションのコンソール指向トピックのほとんどは、 AWS IoT コンソールから開始されます。にまだサインインしていない場合は AWS アカウント、サインインし、AWS IoT コンソールを開いて次のセクションに進み、 の使用を続行します AWS IoT。

インタラクティブチュートリアル

インタラクティブチュートリアルでは、 AWS IoT上に構築されたシンプルな IoT ソリューションのコンポーネントを示します。このチュートリアルでは、IoT デバイスが AWS IoT Core サービスとやり取りする方法を示します。このトピックでは、 AWS IoT Core インタラクティブチュートリアルのプレビューを提供します。

> **Note**
>
> コンソールの画像には、このチュートリアルの画像には表示されないアニメーションが含まれています。

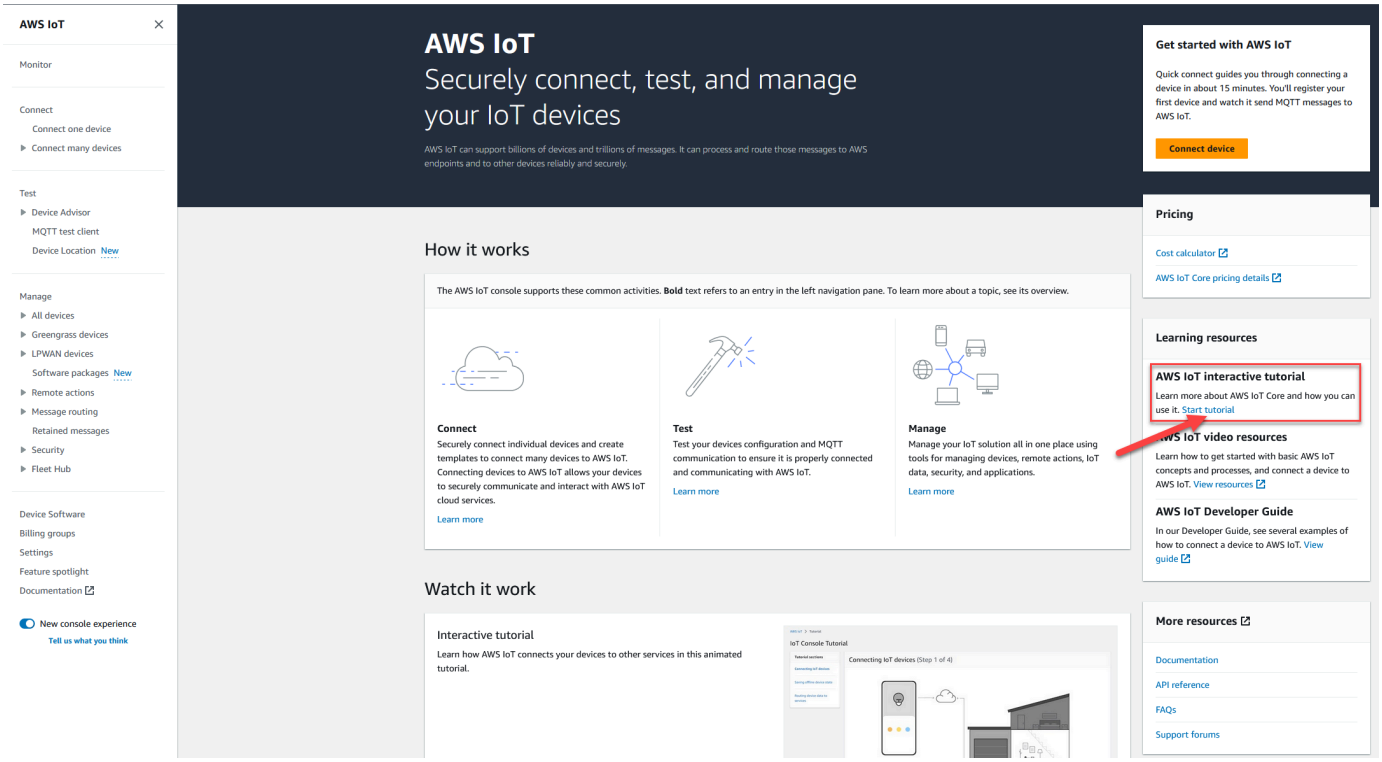
デモを実行するには、まずthe section called “セットアップ AWS アカウント”を実行する必要があります。ただし、このチュートリアルでは、 AWS IoT リソース、追加のソフトウェア、またはコーディングは必要ありません。

このデモにかかる時間は、約 5 ~ 10 分です。10 分かけることで、各ステップについて理解する時間が長くなります。

AWS IoT Core インタラクティブチュートリアルを実行するには

1. AWS IoT コンソールでAWS IoT ホームページを開きます。

   AWS IoT ホームページの [学習リソース]] ウィンドウペインで、[チュートリアルを開始する] を選択します。



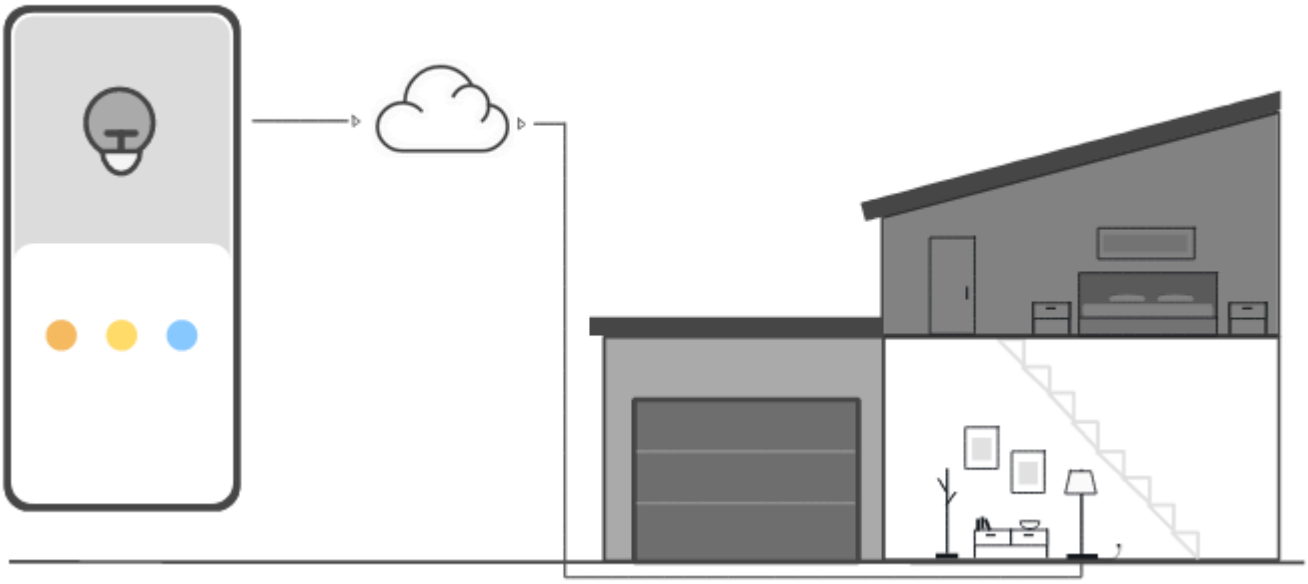
2. [AWS IoT コンソールのチュートリアル]ページで、チュートリアルのセクションを確認し、準備ができたら [開始] セクションを選択します。

以下のセクションでは、 AWS IoT コンソールチュートリアルでこれらの AWS IoT Core 機能がどのように表示されるかについて説明します。

- IoT デバイスの接続
- オフラインデバイスの状態の保存
- デバイスデータのサービスへのルーティング

IoT デバイスの接続

IoT デバイスが と通信する方法について説明します AWS IoT Core。

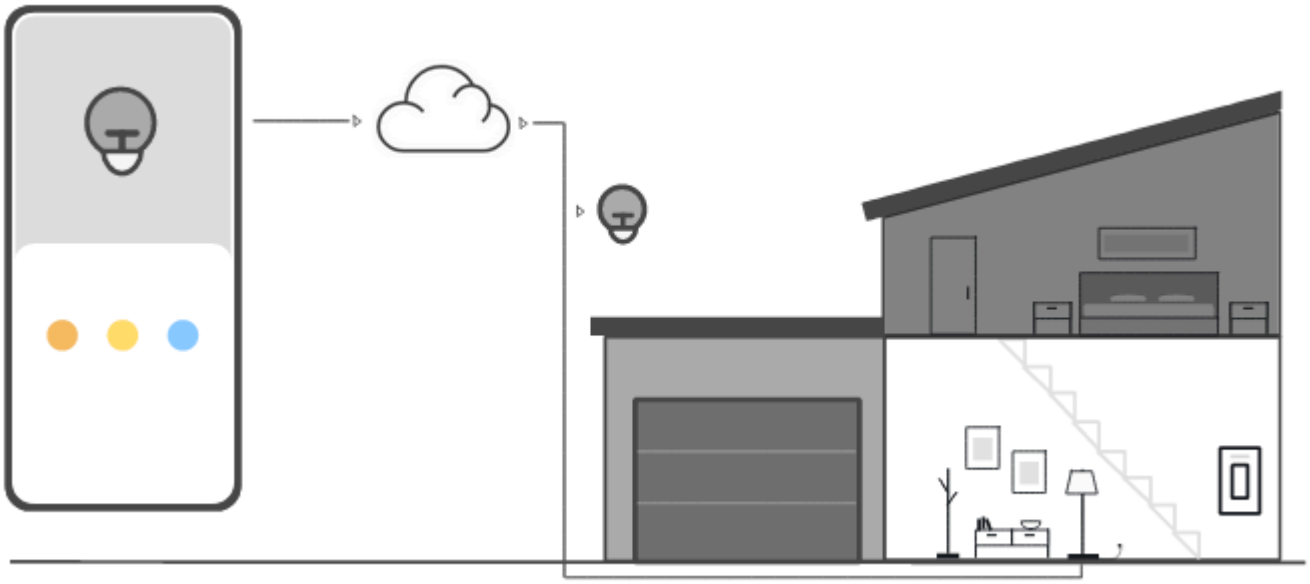


このステップのアニメーションは、左の制御デバイスと右の家庭用スマートランプという 2 つのデバイスが、クラウド内の AWS IoT Core とどのように接続して通信するかを示します。アニメーションは、デバイスが と通信 AWS IoT Core し、受信したメッセージに反応する様子を示しています。

デバイスを に接続する方法の詳細については AWS IoT Core、「」を参照してください[に接続する AWS IoT Core](#)。

オフラインデバイスの状態の保存

デバイスまたはアプリがオフラインのときに がデバイスの状態 AWS IoT Core を保存する方法について説明します。



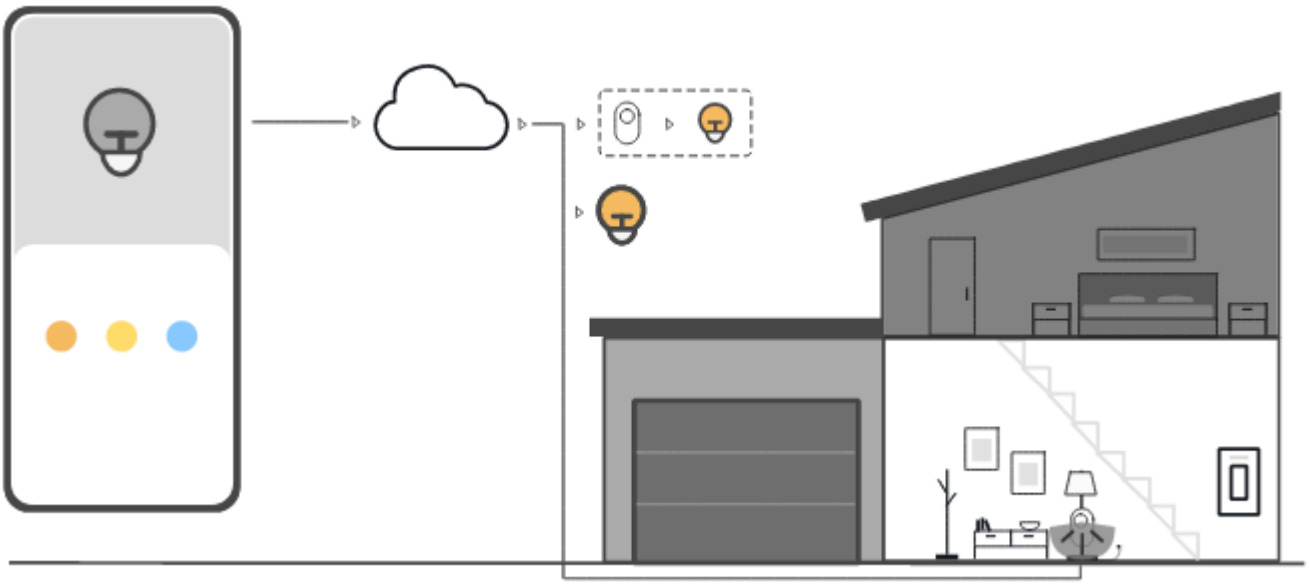
このステップのアニメーションは、 の Device Shadow サービスがコントロールデバイスとスマートランプのデバイス状態情報 AWS IoT Core を保存する方法を示しています。スマートランプがオフラインの場合、デバイスシャドウは制御デバイスからのコマンドを保存します。

スマートランプが再接続されると AWS IoT Core、これらのコマンドを取得します。制御デバイスがオフラインの場合、デバイスシャドウはスマートランプから状態情報を保存します。制御デバイスが再接続されると、スマートランプの現在の状態を取得して表示を更新します。

デバイスシャドウの詳細については、「AWS IoT Device Shadow サービス」を参照してください。

デバイスデータのサービスへのルーティング

AWS IoT Core がデバイスの状態を他の AWS サービスに送信する方法について説明します。



このステップのアニメーションは、 が AWS IoT rules. AWS IoT rules を使用してデバイスから他の AWS サービスにデータ AWS IoT Core を送信する方法を示しています。 はデバイスからの特定のメッセージをサブスクライブし、それらのメッセージ内のデータを解釈し、解釈されたデータを他のサービスにルーティングします。この例では、 AWS IoT ルールはモーションセンサーからデータを解釈し、コマンドを Device Shadow に送信して、スマート電球に送信します。前の例と同様に、デバイスシャドウは制御デバイスのデバイス状態情報を保存します。

AWS IoT ルールの詳細については、「」を参照してくださいのルール AWS IoT。

AWS IoT Core クイック接続チュートリアルを試す

このチュートリアルでは、最初のモノのオブジェクトを作成し、そのオブジェクトにデバイスを接続して、MQTT メッセージを送信する様子を観察します。

このチュートリアルは 15～20 分を要します。

このチュートリアルは、 の使用をすばやく開始 AWS IoT して、限られたシナリオでの動作を確認するユーザーに最適です。より多くの機能やサービスに興味を持つきっかけとなる例を探している場合は、実践的なチュートリアル AWS IoT Core で を試す を試してください。

このチュートリアルでは、非常に小さな IoT ソリューション AWS IoT Core の一部として のモノのリソースに接続するデバイスでソフトウェアをダウンロードして実行します。デバイスは、Raspberry Pi などの IoT デバイスにすることも、Linux、OS と OSX、または Windows を実行しているコンピュータにすることもできます。Long Range WAN (LoRaWAN) デバイスを に接続す

る場合は AWS IoT、「> デバイスとゲートウェイを AWS IoT Core for LoRaWAN に接続する」のチュートリアルを参照してください。

デバイスが AWS IoT コンソールを実行できるブラウザをサポートしている場合は、そのデバイスでこのチュートリアルを完了することをお勧めします。

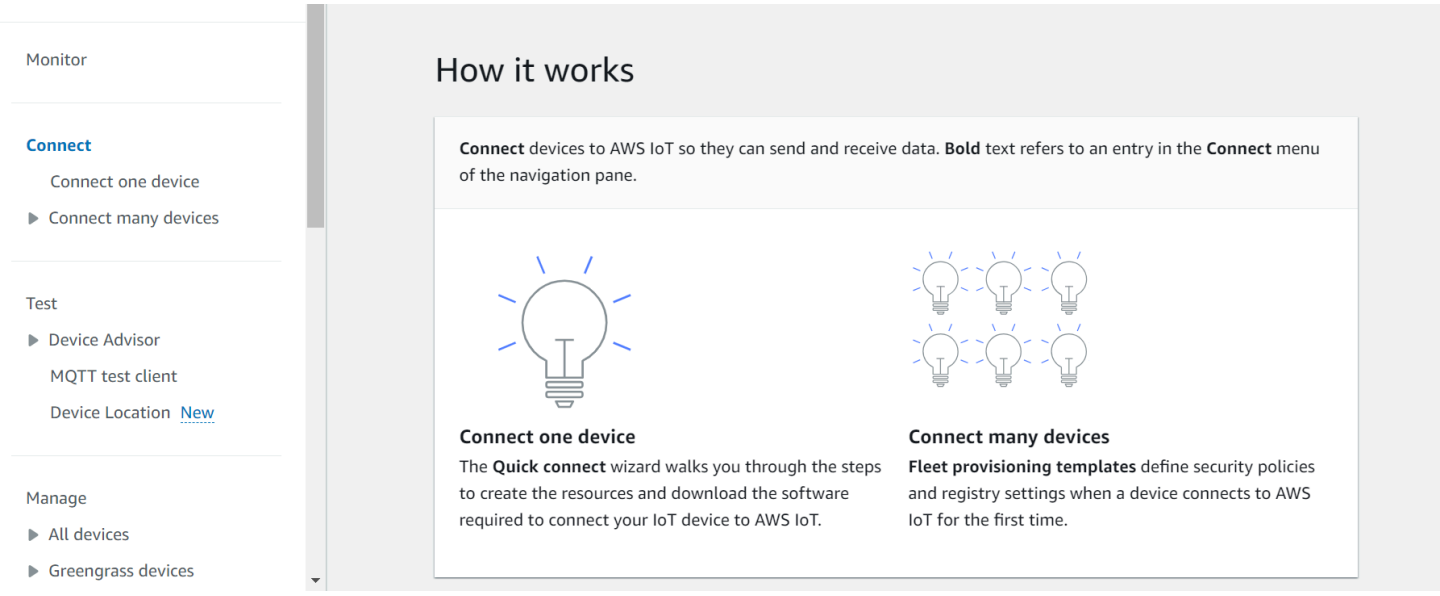
> **Note**
>
> デバイスに互換性のあるブラウザがない場合は、コンピュータでこのチュートリアルに従ってください。手順でファイルをダウンロードするように求められたら、それをコンピュータにダウンロードしてから、ダウンロードしたファイルを Secure Copy (SCP) または同様のプロセスを使用してデバイスに転送します。

このチュートリアルでは、IoT デバイスが AWS アカウントのデバイスデータエンドポイントのポート 8443 と通信する必要があります。そのポートにアクセスできるかどうかをテストするには、「デバイスデータエンドポイントとの接続をテストする」の手順を試してください。

ステップ 1. チュートリアルを開始する

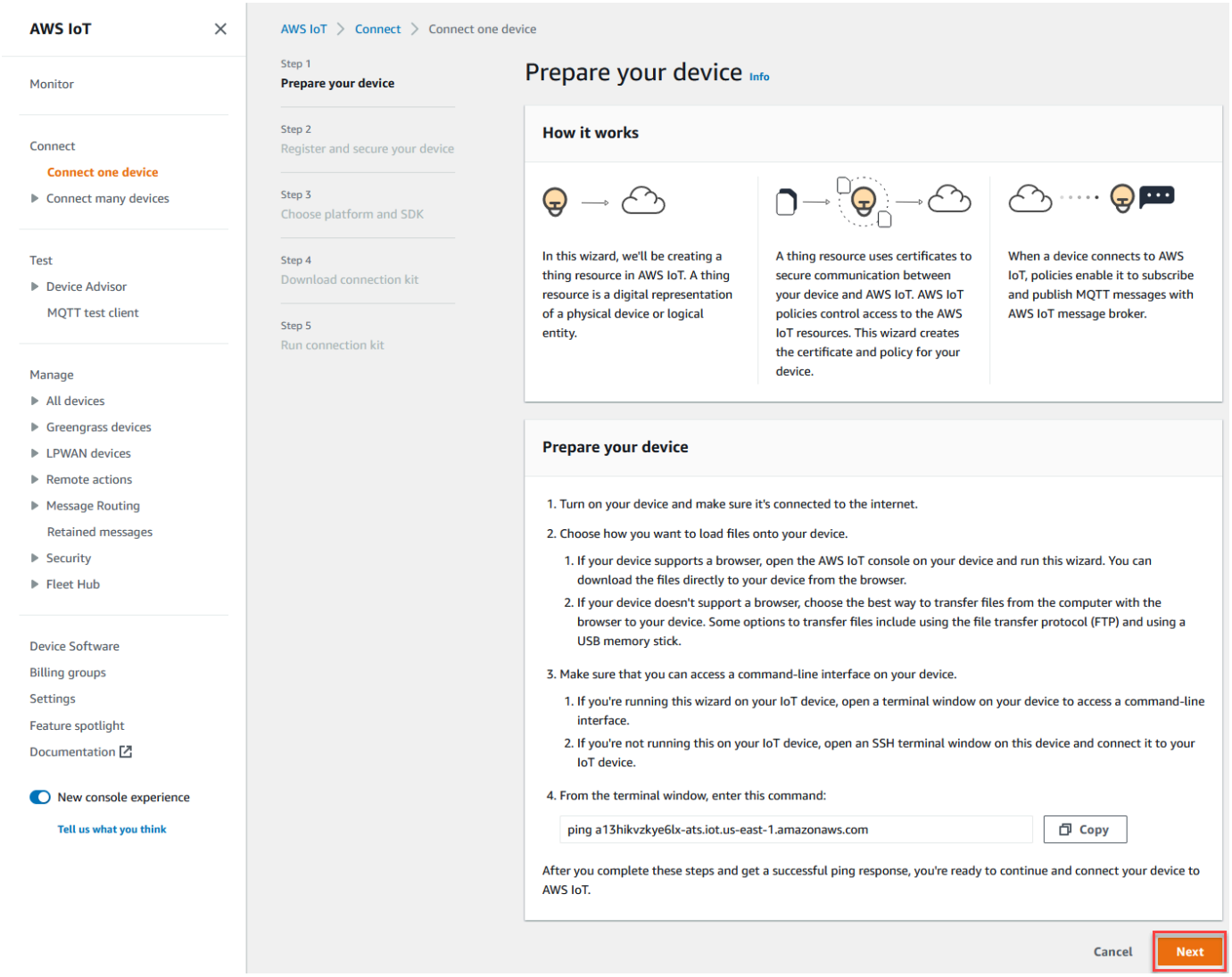
可能であれば、デバイスでこの手順を完了してください。それ以外の場合は、この手順の後半でファイルをデバイスに転送する準備をしてください。

チュートリアルを開始するには、AWS IoT コンソールにログインします。 AWS IoT コンソールのホームページの左側で、Connect を選択し、Connect one device を選択します。



ステップ 2. モノのオブジェクトを作成する

1. [Prepare your device] (デバイスを準備する) セクションで、画面の指示に従ってデバイスを AWS IoTに接続する準備をします。



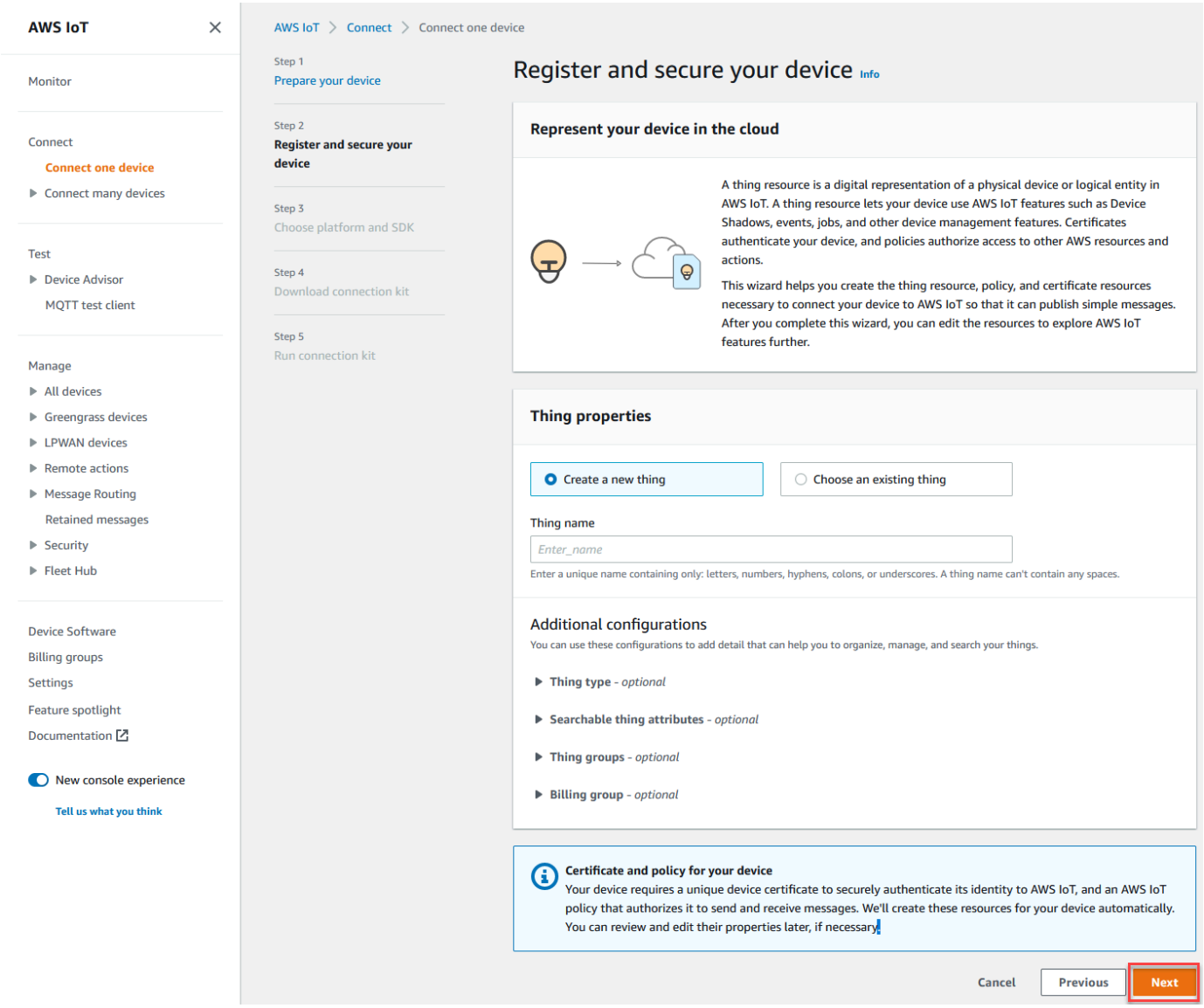
2. [Register and secure your device] (デバイスを登録して保護する) セクションで、[Create a new thing] (新しいモノの作成) または [Choose an existing thing] (既存のモノを選択) を選択してください。[Thing name] (モノの名前) フィールドで、モノのオブジェクトの名前を入力します。この例で使用されているモノの名前は **`TutorialTestThing`** です。

> **Important**
>
> 続行する前に、モノの名前をもう一度確認します。

> モノのオブジェクトの作成後にモノの名前を変更することはできません。モノの名前を変更するには、正しいモノの名前の新しいモノのオブジェクトを作成し、間違った名前のモノのオブジェクトを削除する必要があります。

[Additional configurations] (追加の設定) セクションで、リストされているオプション設定を使用して、モノのリソースをさらにカスタマイズします。

モノのオブジェクトに名前を付けて、追加の設定を選択したら、[Next] (次へ) を選択します。

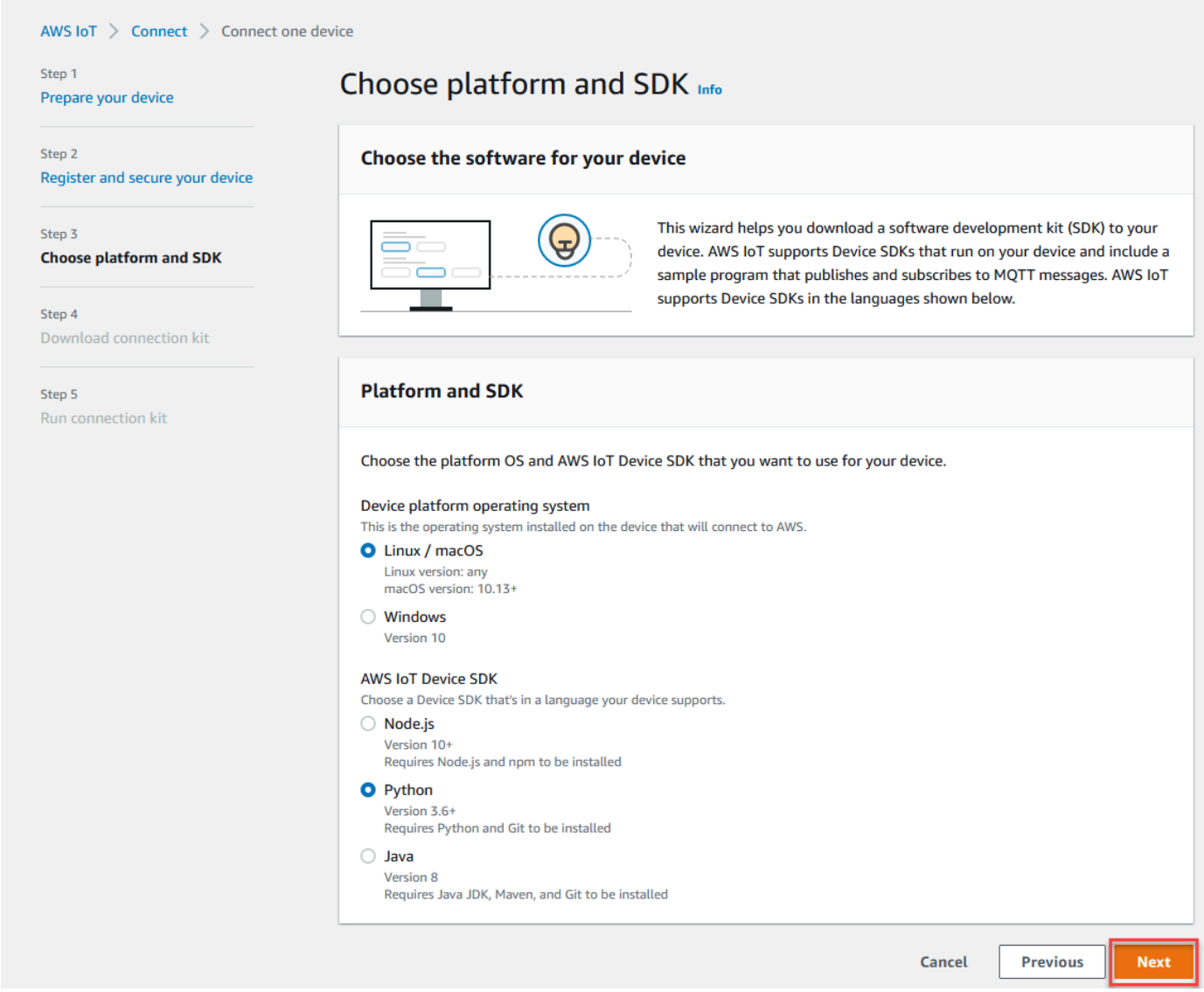


3. プラットフォームと SDK の選択セクションで、使用する AWS IoT Device SDK のプラットフォームと言語を選択します。この例では、Linux/OSX プラットフォームと Python SDK を使

用しています。次のステップに進む前に python3 と pip3 がターゲットデバイスにインストールされていることを確認してください。

> **Note**
>
> コンソールのページの下部で、選択した SDK に必要な前提条件であるソフトウェアの一覧を確認してください。
> 次のステップに進む前に、必要なソフトウェアをターゲットコンピュータにインストールしておく必要があります。

プラットフォームとデバイスの SDK 言語を選択したら、[Next] (次へ) を選択します。

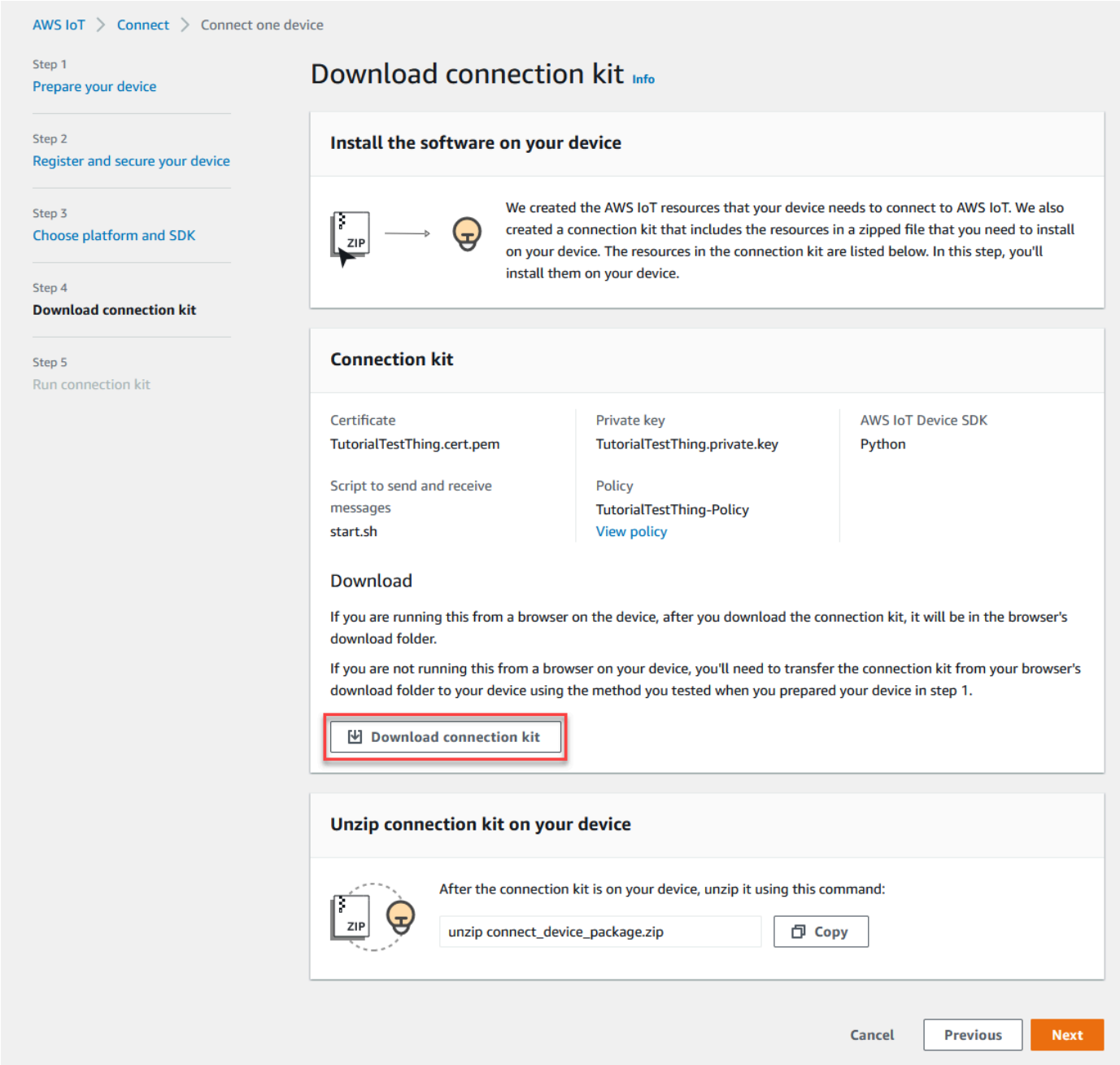


ステップ 3. デバイスにファイルをダウンロードする

このページは、 AWS IoT が接続キットを作成した後に表示されます。このキットには、デバイスが必要とする以下のファイルとリソースが含まれています。

- デバイスの認証に使用するモノの証明書ファイル
- モノのオブジェクトが AWS IoTとインタラクションすることを承認するポリシーリソース
- AWS Device SDK をダウンロードし、デバイスでサンプルプログラムを実行するスクリプト

1. 続行する準備ができたら、[Download connection kit for] (接続キットのダウンロード) ボタンを選択して、前に選択したプラットフォーム用の接続キットをダウンロードします。



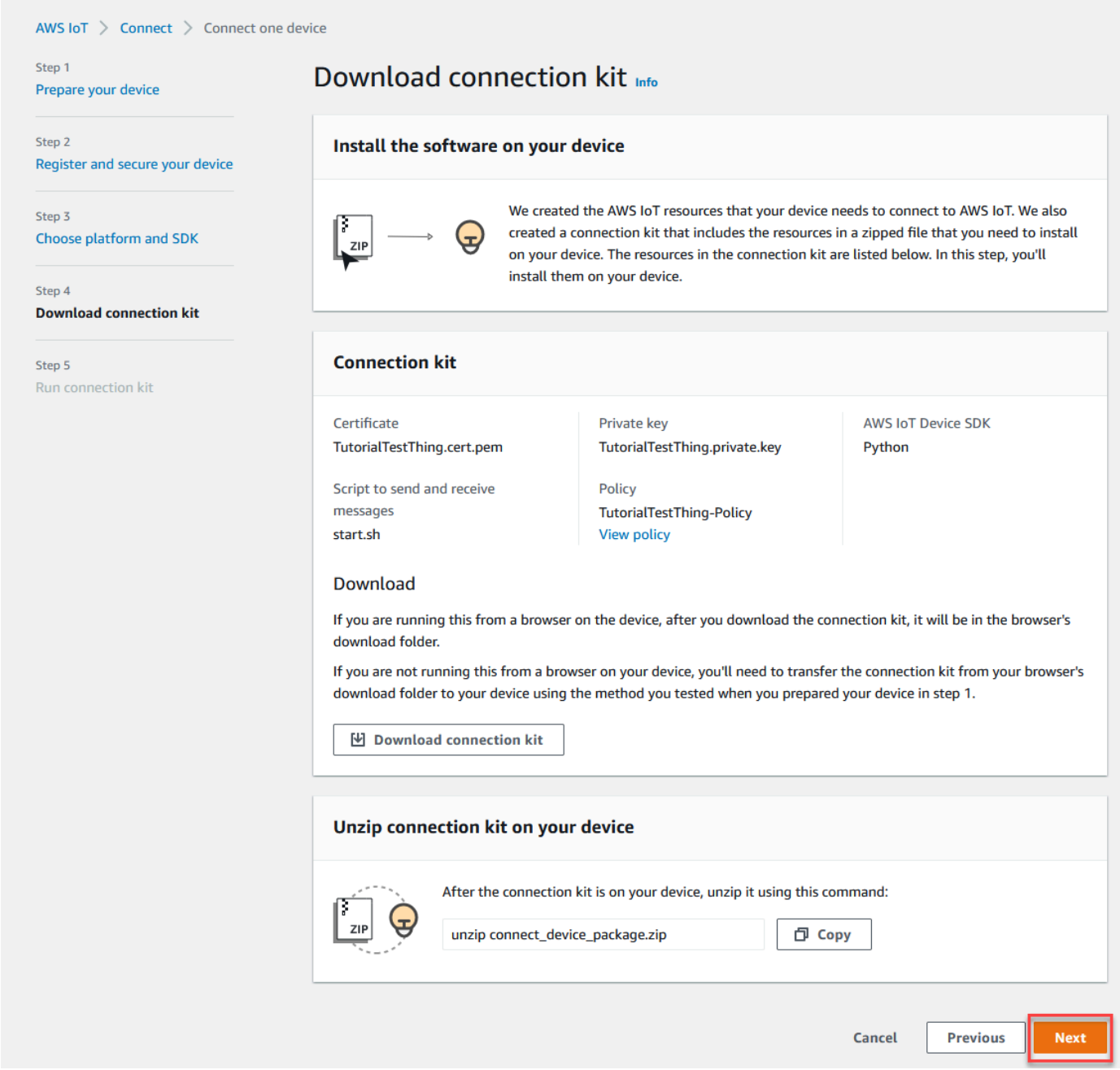
2. この手順をデバイスで実行している場合は、コマンドラインコマンドを実行できるディレクトリに接続キットファイルを保存します。

   この手順をデバイスで実行していない場合は、接続キットファイルをローカルディレクトリに保存し、そのファイルをデバイスに転送します。

3. [Unzip connection kit on your device] (デバイスで接続キットを解凍) セクションで、接続キットファイルが置かれているディレクトリに unzip connect_device_package.zip と入力します。

Windows PowerShell コマンドウィンドウを使用していて、unzip コマンドが機能しない場合は、unzip を expand-archive に置き換えて、コマンドラインをもう一度試してください。

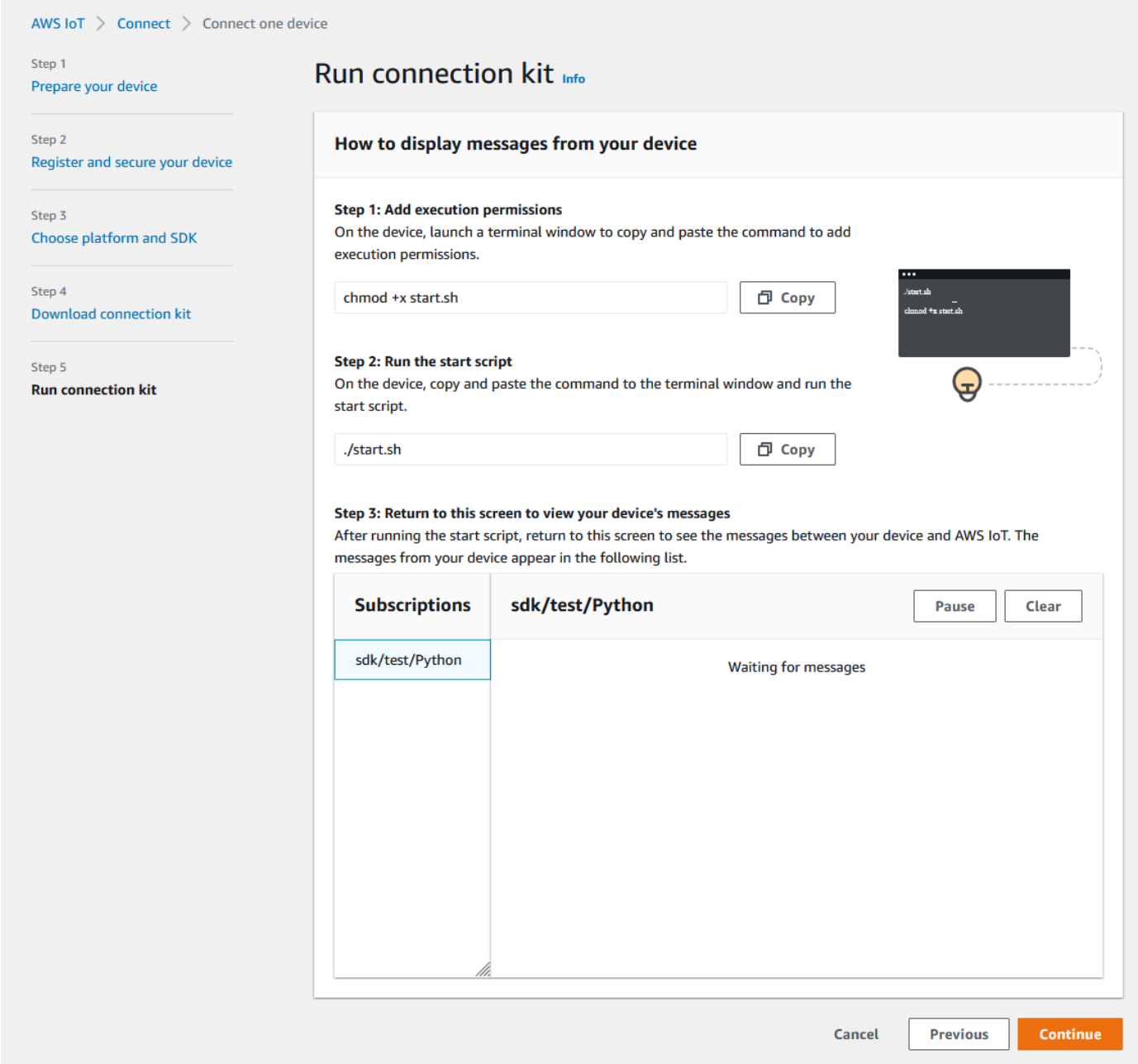
4. デバイスに接続キットファイルを作成したら、[Next] (次へ) を選択してチュートリアルを続行します。



ステップ 4. サンプルを実行する

この手順は、コンソールに表示される指示に従って、デバイスのターミナルまたはコマンドウィンドウで実行します。コンソールに表示されるコマンドは、the section called “ステップ 2. モノのオブジェクトを作成する” で選択したオペレーティングシステム用のコマンドです。ここに示されているものは、Linux/OSX オペレーティングシステム用です。

1. デバイスのターミナルまたはコマンドウィンドウで、接続キットファイルを含む ディレクトリで、 AWS IoT コンソールに表示されるステップを実行します。



2. コンソールで[Step 2] (ステップ 2) のコマンドを入力すると、デバイスのターミナルまたはコマンドウィンドウで、次のような出力が表示されます。この出力は、プログラムが AWS IoT Core との間で送受信しているメッセージからのものです。

```
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]"'
```

   サンプルプログラムの実行中は、テストメッセージ `Hello World!` も表示されます。テストメッセージは、デバイスのターミナルまたはコマンドウィンドウに表示されます。

   > **Note**
   >
   > トピックのサブスクリプションと発行の詳細については、選択した SDK のサンプルコードを参照してください。

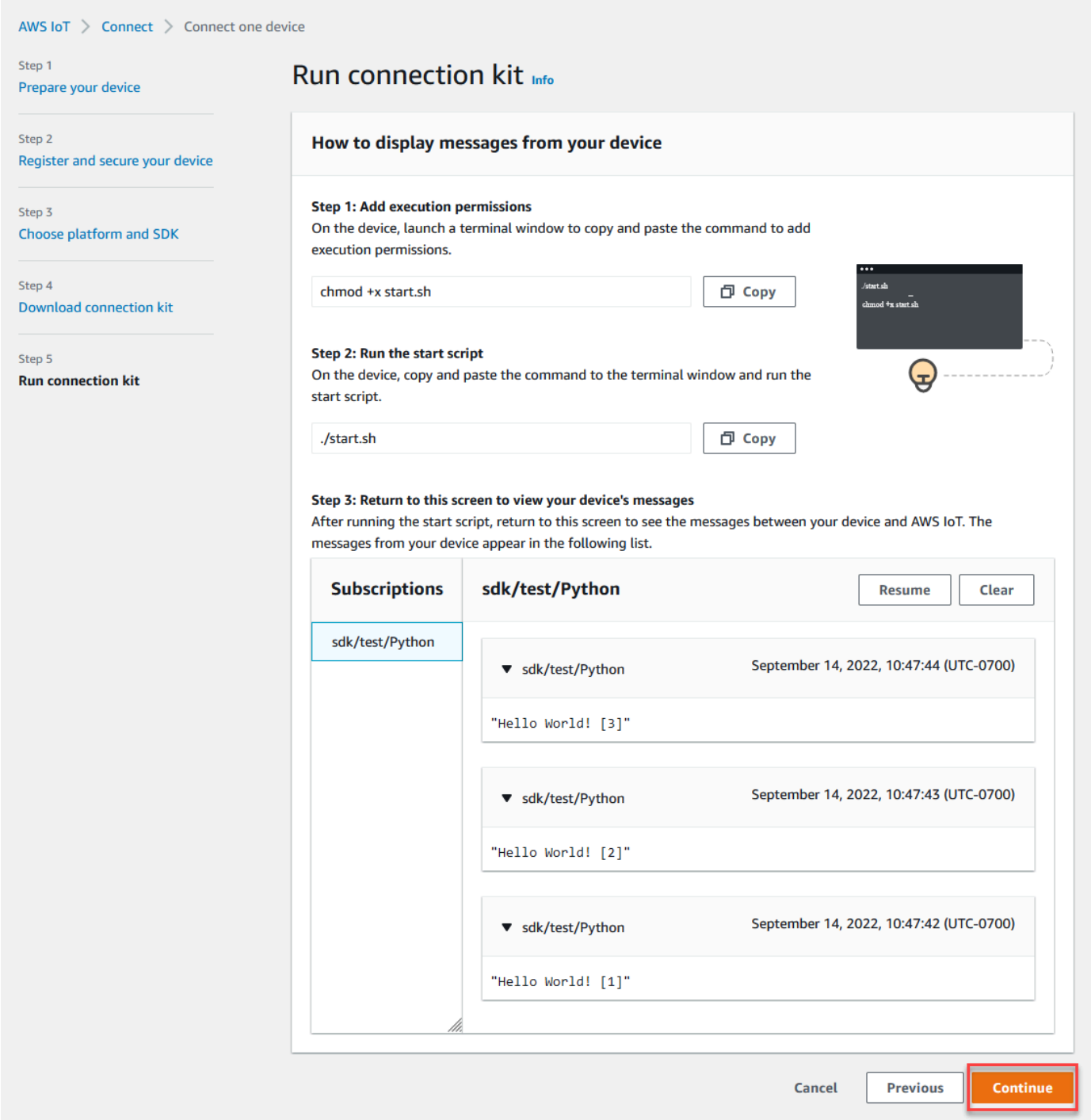
3. この手順のコンソールで [Step 2] (ステップ 2) からのコマンドを繰り返して、サンプルプログラムを再度実行できます。
4. (オプション) AWS IoT コンソールで IoT クライアントからのメッセージを表示するには、 AWS IoT コンソールのテストページで MQTT テストクライアントを開きます。 Python SDK を選択した場合、[MQTT test client] (MQTT テストクライアント) の [Topic filter] (トピックフィルター) に **`sdk/test/python`** などのトピックを入力して、デバイスからのメッセージをサブスクライブします。トピックフィルターは、大文字と小文字を識別し、[Step 1] (ステップ 1) で選択した SDK のプログラミング言語によって異なります。トピックのサブスクリプションと発行の詳細については、選択した SDK のコード例を参照してください。
5. テストトピックを購読後、デバイスで ./start.sh を実行します。詳細については、「the section called “MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する”」を参照してください。

   ./start.sh の実行後、次のようなメッセージが MQTT クライアントに表示されます。

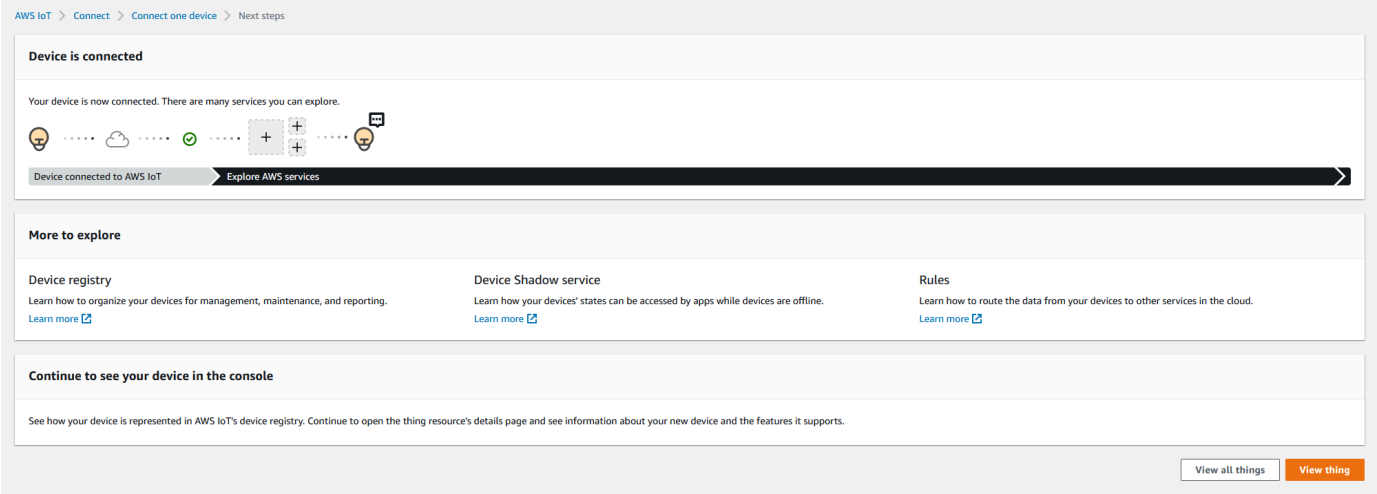
```
{
  "message": "Hello World!" [1]
}
```

[] で囲われている sequence 番号は、新しい Hello World! メッセージを受信するたびに 1 ずつ増加し、プログラムを終了すると停止します。

6. チュートリアルを終了して概要を表示するには、 AWS IoT コンソールで「続行」を選択します。



7. AWS IoT クイック接続チュートリアルの概要が表示されます。



ステップ 5. さらに詳しく

クイックスタートを完了した後、 AWS IoT さらに詳しく調べるアイデアをいくつか紹介します。

- MQTT テストクライアントで MQTT メッセージを表示する

  AWS IoT コンソールから、 AWS IoT コンソールの [テスト] ページで [MQTT クライアント] を開くことができます。MQTT テストクライアントで # にサブスクライブし、デバイスで前の手順で説明したようにプログラム ./start.sh を実行します。詳細については、「the section called “MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する”」を参照してください。

- Device Advisor を使用してデバイスでテストを実行する

  Device Advisor を使用して、デバイスが安全かつ確実に に接続し、操作できるかどうかをテストします AWS IoT。

- the section called “インタラクティブチュートリアル”

  インタラクティブチュートリアルを開始するには、 AWS IoT コンソールの学習ページから、「 の AWS IoT 仕組み」タイルで、チュートリアルの開始を選択します。

- より多くのチュートリアルを見る

  このクイックスタートでは、 のサンプルのみを提供します AWS IoT。さらに AWS IoT 詳しく調べ、強力な IoT ソリューションプラットフォームにする機能については、 で開発プラットフォームの準備を開始します実践的なチュートリアル AWS IoT Core で を試す。

デバイスデータエンドポイントとの接続をテストする

このトピックでは、アカウントのデバイスデータエンドポイント (AWS IoTに接続するために IoT デバイスが使用するエンドポイント) に対するデバイスの接続をテストする方法について説明します。

この手順は、テストするデバイス上で実行するか、テストするデバイスに接続された SSH ターミナルセッションを使用して実行します。

デバイスデータエンドポイントとのデバイスの接続をテストするには

- デバイスデータエンドポイントを検索する
- 接続をすばやくテストする
- アプリを入手してデバイスのデータエンドポイントとポートへの接続をテストする
- デバイスデータエンドポイントとポートへの接続をテストする

デバイスデータエンドポイントを検索する

この手順では、AWS IoT コンソールでデバイスデータエンドポイントを検索して IoT デバイスへの接続をテストする方法について説明します。

デバイスデータエンドポイントを検索するには

1. AWS IoT コンソールの [接続] で、[ドメイン設定] に移動します。
2. [ドメイン設定] ページで、[ドメイン設定] コンテナに移動し、ドメイン名をコピーします。エンドポイント値は に固有 AWS アカウント であり、 の例に似ていますa3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com。
3. デバイスデータポイントを保存します。このデバイスデータエンドポイントは、次の手順で使用します。

接続をすばやくテストする

この手順では、デバイスデータエンドポイントとの一般的な接続をテストしますが、デバイスが使用する特定のポートはテストされません。このテストでは一般的なプログラムを使用します。通常、これは、デバイスが AWS IoTに接続できるかどうかを確認するには十分です。

デバイスが使用する特定のポートとの接続をテストする場合は、この手順をスキップして、「アプリを入手してデバイスのデータエンドポイントとポートへの接続をテストする」に進みます。

デバイスデータエンドポイントをすばやくテストするには

1. ターミナルまたはデバイスのコマンドラインウィンドウで、サンプルデバイスデータエンドポイント (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) をアカウントのデバイスデータエンドポイントで置き換え、次のコマンドを入力します。

   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. 次のような出力が ping に表示された場合、デバイスデータエンドポイントに正常に接続されています。と AWS IoT 直接通信しませんでしたが、サーバーが見つかったため、このエンドポイントを介して AWS IoT が使用可能である可能性があります。

   ```
   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
   ```

   この結果に満足したら、ここでテストを終了できます。

   AWS IoTが使用する特定のポートとの接続をテストする場合、「アプリを入手してデバイスのデータエンドポイントとポートへの接続をテストする」に進んでください。

3. ping で正常な出力が返されなかった場合、エンドポイントの値を参照して、正しいエンドポイントが入力されていること、およびデバイスがインターネットに接続されていることを確認してください。

アプリを入手してデバイスのデータエンドポイントとポートへの接続をテストする

nmap を使用して、より詳細な接続テストを実行できます。この手順では、nmap がデバイスにインストールされているかどうかをテストします。

デバイスに **nmap** がインストールされていることを確認するには

1. ターミナルまたはテストするデバイスのコマンドラインウィンドウで、次のコマンドを入力し、nmap がインストールされていることを確認します。

   ```
   nmap --version
   ```

2. 次のような出力が表示された場合、nmap がインストールされていて、the section called “デバイスデータエンドポイントとポートへの接続をテストする” に接続できます。

   ```
   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. 前のステップに示すような応答がない表示されない場合、デバイスに nmap をインストールする必要があります。デバイスのオペレーティングシステムの手順を選択します。

Linux

この手順を実行するには、コンピュータにソフトウェアをインストールするためのアクセス許可が必要です。

Linux コンピュータに nmap をインストールするには

1. ターミナルまたはデバイスのコマンドラインウィンドウで、それが実行している Linux のバージョンに対応するコマンドを入力します。

   a. Debian または Ubuntu の場合:

      ```
      sudo apt install nmap
      ```

   b. CentOS または RHEL の場合:

```
sudo yum install nmap
```

2. 次のコマンドを使用してインストールをテストします。

```
nmap --version
```

3. 次のような出力が表示された場合、nmap がインストールされていて、the section called “デバイスデータエンドポイントとポートへの接続をテストする” に接続できます。

```
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

この手順を実行するには、コンピュータにソフトウェアをインストールするためのアクセス許可が必要です。

macOS コンピュータに nmap をインストールするには

1. ブラウザで https://nmap.org/download#macosx を開き、Latest stable release (最新の安定したリリース) を開きます。

   プロンプトが表示されたら、[Open with DiskImageInstaller] (DiskImageInstaller で開く) を選択します。
2. インストールウィンドウで、パッケージを Applications フォルダに移動します。
3. [検索] で Applications フォルダ内の nmap-xxxx-mpkg パッケージを見つけます。パッケージを Ctrl-click し、[開く] を選択してパッケージを開きます。
4. セキュリティダイアログボックスを確認します。nmap をインストールする準備ができたら、[開く] を選択して nmap をインストールします。
5. Terminal で、次のコマンドを使用してインストールをテストします。

```
nmap --version
```

6. 次のような出力が表示された場合、nmap がインストールされていて、the section called “デバイスデータエンドポイントとポートへの接続をテストする” に接続できます。

```
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

この手順を実行するには、コンピュータにソフトウェアをインストールするためのアクセス許可が必要です。

Windows コンピュータに nmap をインストールするには

1. ブラウザで https://nmap.org/download#windows を開き、セットアッププログラムの Latest stable release (最新の安定したリリース) をダウンロードします。

   プロンプトが表示されたら、[ファイルを保存] を選択します。ファイルをダウンロードしたら、ダウンロードフォルダからファイルを開きます。
2. セットアップファイルのダウンロードが完了したら、ダウンロードした nmap-xxxx-setup.exe 開いてアプリをインストールします。
3. プログラムのインストール時に、デフォルト設定を受け入れます。

   このテストには Npcap アプリは必要ありません。このアプリをインストールしない場合は、このオプションの選択を解除できます。
4. Command で、次のコマンドを使用してインストールをテストします。

```
nmap --version
```

5. 次のような出力が表示された場合、nmap がインストールされていて、the section called “デバイスデータエンドポイントとポートへの接続をテストする” に接続できます。

```
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
```

デバイスデータエンドポイントとポートへの接続をテストする

この手順では、選択したポートを使用して IoT デバイスのデバイスデータエンドポイントへの接続をテストします。

デバイスデータエンドポイントとポートへの接続をテストするには

1. ターミナルまたはデバイスのコマンドラインウィンドウで、サンプルデバイスデータエンドポイント (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) をアカウントのデバイスデータエンドポイントで置き換え、次のコマンドを入力します。

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

2. 次のような出力が nmap に表示された場合、nmap は、選択したポートのデバイスデータエンドポイントに正常に接続できました。

   ```
   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
   ```

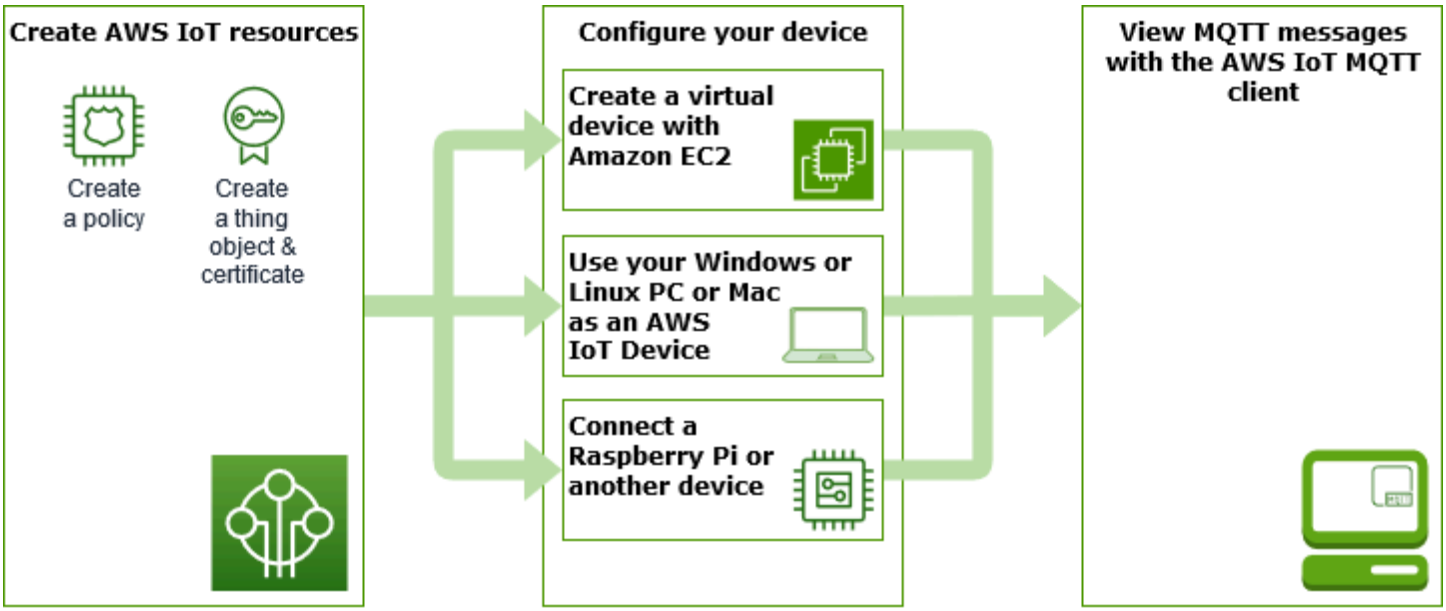
3. nmap で正常な出力が返されなかった場合、エンドポイントの値を参照して、正しいエンドポイントが入力されていること、およびデバイスがインターネットに接続されていることを確認してください。

ステップ 1 で使用したポート (**8443**) を置き換えることによって、デバイスデータエンドポイントのその他のポート (プライマリ HTTPS ポートのポート443 など) をテストできます。

実践的なチュートリアル AWS IoT Core で を試す

このチュートリアルでは、 ソフトウェアをインストールし、デバイスを に接続するために必要な AWS IoT リソースを作成して AWS IoT Core 、 で MQTT メッセージを送受信できるようにします AWS IoT Core。 AWS IoT コンソールの MQTT クライアントにメッセージが表示されます。

このチュートリアルは 20～30 分を要します。IoT デバイスまたは Raspberry Pi を使用している場合、例えばオペレーティングシステムをインストールしてデバイスを設定する必要があるときは、このチュートリアルには時間がかかることがあります。



このチュートリアルは、 の使用を開始して、ルールエンジンやシャドウなどのより高度な機能を引き続き探索 AWS IoT Core できるようにする開発者に最適です。このチュートリアルでは、クイックスタートチュートリアルよりも詳細な手順を説明することで、 とそれが他の AWS サービスとやり取りする方法を AWS IoT Core 学習し続ける準備をします。簡単な Hello World 体験を探している場合は、AWS IoT Core クイック接続チュートリアルを試す をお試しください。

AWS アカウント と AWS IoT コンソールを設定したら、以下の手順に従ってデバイスを接続し、メッセージを送信します AWS IoT Core。

次の手順

- 最適なデバイスオプションを選択する
- the section called “AWS IoT リソースの作成” (Amazon EC2 で仮想デバイスを作成しない場合)
- the section called “デバイスを設定する”
- the section called “MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する”

詳細については AWS IoT Core、「とは AWS IoT Core」を参照してください。

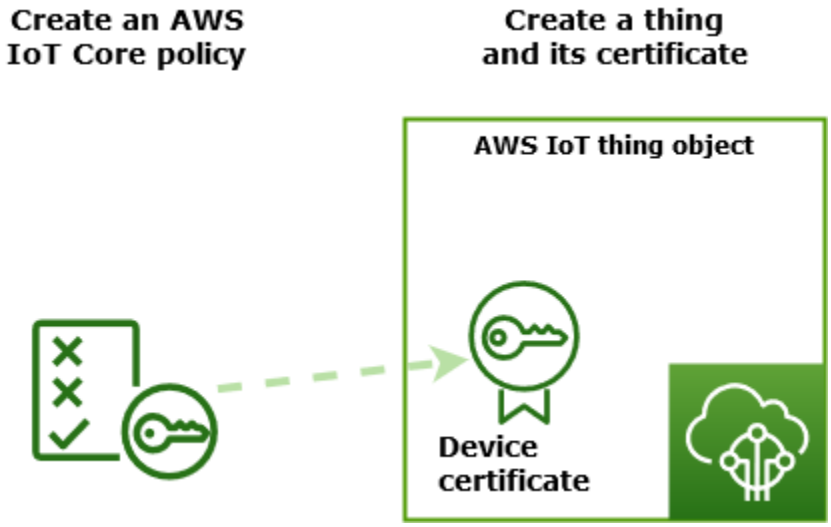
どのデバイスオプションが最適ですか?

どのオプションを選択すればよいかわからない場合は、どれが最適かを判断するために、次の各オプションのメリットとデメリットのリストを役立ててください。

オプション	これは、次の場合には、良い選択肢である場合があります。	これは、次の場合には、良い選択肢ではない場合があります。
the section called “Amazon EC2 を使用して仮想デバイスを作成する”	• テストする独自のデバイスがない場合。 • 自分のシステムにソフトウェアをインストールしたくない場合。 • Linux OS でテストしたい場合。	• コマンドラインのコマンドの使用に慣れていません。 • 追加の AWS 料金を発生させたくない場合。 • Linux OS ではテストしたくない場合。
the section called “Windows または Linux の PC または Mac を AWS IoT デバイスとして使用する”	• 追加の AWS 料金を発生させたくない場合。 • 追加のデバイスを設定したくない場合。	• パーソナルコンピュータにソフトウェアをインストールしたくない場合。 • より代表的なテストプラットフォームを必要としている場合。
the section called “Raspberry Pi または他のデバイスを接続する”	• AWS IoT 実際のデバイスでテストしたい。 • テストするデバイスが既にある場合。 • ハードウェアをシステムに統合した経験がある場合。	• 試してみるためだけにデバイスを購入したり設定したりしたくない場合。 • 現時点では、できるだけ AWS IoT 簡単にテストしたいと考えています。

AWS IoT リソースの作成

このチュートリアルでは、デバイスがメッセージに接続 AWS IoT Core して交換するために必要な AWS IoT リソースを作成します。



1. デバイスが AWS IoT サービスとやり取りすることを許可する AWS IoT ポリシードキュメントを作成します。
2. AWS IoT とその X.509 デバイス証明書にモノのオブジェクトを作成し、ポリシードキュメントをアタッチします。Thing オブジェクトは、 AWS IoT レジストリ内のデバイスの仮想表現です。証明書はデバイスを に認証し AWS IoT Core、ポリシードキュメントはデバイスが とやり取りすることを許可します AWS IoT。

> **Note**
>
> the section called “Amazon EC2 を使用して仮想デバイスを作成する” を予定している場合は、このページをスキップして the section called “デバイスを設定する” に進むことができます。これらのリソースは、仮想のモノを作成するときに作成します。

このチュートリアルでは、 AWS IoT コンソールを使用して AWS IoT リソースを作成します。デバイスがウェブブラウザをサポートしている場合は、証明書ファイルをデバイスに直接ダウンロードできるため、デバイスのウェブブラウザでこの手順を実行する方が簡単な場合があります。この手順を別のコンピュータで実行する場合は、サンプルアプリケーションで証明書ファイルを使用する前に、デバイスに証明書ファイルをコピーする必要があります。

AWS IoT ポリシーを作成する

デバイスは X.509 証明書を使用して認証します AWS IoT Core。証明書には AWS IoT ポリシーがアタッチされています。これらのポリシーは、デバイスで実行できる AWS IoT オペレーション (MQTT トピックへのサブスクライブや公開など) を決定します。デバイスは、接続してメッセージを送信するときに証明書を提示します AWS IoT Core。

サンプルプログラムを実行するために必要な AWS IoT オペレーションの実行をデバイスに許可するポリシーを作成します。後で作成するデバイス証明書にアタッチできるように、最初に AWS IoT ポリシーを作成する必要があります。

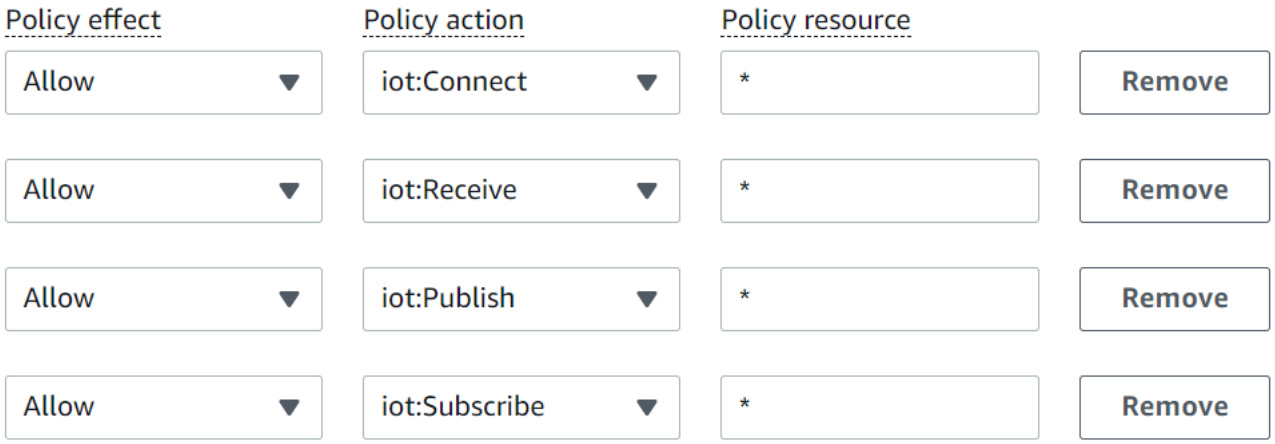
AWS IoT ポリシーを作成するには

1. 左のメニューの AWS IoT コンソールで、[セキュリティ]、[ポリシー] の順に選択します。
2. [You don't have a policy yet] (ポリシーを作成していません) ページで、[ポリシーの作成] を選択します。

   アカウントに既存のポリシーがある場合は、[ポリシーを作成] を選択します。
3. [ポリシーの作成] ページで、以下のステップを実行します。

   1. [ポリシーのプロパティ] セクションの [プロパティ名] フィールドにポリシーの名前 (**My_Iot_Policy** など) を入力します。ポリシー名には個人を特定できる情報を使用しないでください。
   2. [ポリシードキュメント] セクションで、リソースアクセスを AWS IoT Core オペレーションに対して付与または拒否するポリシーステートメントを作成します。**iot:Connect** を実行するアクセス許可をすべてのクライアントに付与するポリシーステートメントを作成するには、次のステップに従います。
      - [ポリシーの効果] フィールドで [許可] を選択します。このポリシーが証明書にアタッチされているすべてのクライアントは、[ポリシーアクション] フィールドにリストされているアクションを実行できます。
      - [ポリシーアクション] フィールドで、**iot:Connect** などのポリシーアクションを選択します。ポリシーアクションは、デバイスが Device SDK からサンプルプログラムを実行するときに実行するためのアクセス許可が必要なアクションです。
      - [ポリシーリソース フィールドにリソース、Amazon リソースネーム (ARN)、または * を入力します。任意のクライアント (デバイス) を選択する場合は、* を入力します。

      iot:Receive、**iot:Publish**、および **iot:Subscribe** のポリシーステートメントを作成するには、[新しいステートメントを追加] を選択して、上記のステップを繰り返します。



> **Note**
>
> このクイックスタートでは、単純化のためにワイルドカード (*) 文字が使用されます。セキュリティを強化するには、リソースとしてワイルドカード文字の代わりにクライアント ARN を指定して、メッセージを接続して発行できるクライアント (デバイス) を制限する必要があります。クライアント ARN は、`arn:aws:iot:`*`your-region`*`:`*`your-aws-account`*`:client/`*`my-client-id`* の形式に従います。ただし、その ARN をポリシーに割り当てるには、まずリソース (クライアントデバイス、モノのシャドウなど) を作成する必要があります。詳細については、「AWS IoT Core アクションリソース」を参照してください。

4. ポリシーの情報を入力した後、[作成] を選択します。

詳細については、「が IAM と AWS IoT 連携する方法」を参照してください。

モノのオブジェクトを作成する

に接続されたデバイスは AWS IoT Core 、 AWS IoT レジストリ内のモノのオブジェクトによって表されます。モノのオブジェクトは、特定のデバイスまたは論理エンティティを表します。物理的なデバイスやセンサー (電球、または電気をつけるための壁にあるスイッチなど) は、モノとして扱うことができます。また、アプリケーションのインスタンスや、接続しないが AWS IoT接続する他のデバイス (エンジンセンサーやコントロールパネルがある車など) に関連する物理エンティティなどの論理エンティティでもかまいません。

AWS IoT コンソールでモノを作成するには

1. 左のメニューの AWS IoT コンソールで、[すべてのデバイス]、[モノ] の順に選択します。

2. [モノ] ページで [モノを作成する] を選択します。
3. [Creating things] (モノを作成する) ページで、[Create a single thing] (単一のモノを作成する) を選択し、[Next] (次へ) を選択します。
4. [モノのプロパティを指定する] ページで、[モノの名前]に、モノの名前 (**`MyIotThing`** など) を入力します。

   モノ名は後で変更できないため、モノ名は慎重に選択してください。

   モノの名前を変更するには、新しいモノを作成して、新しい名前を付け、古いモノを削除する必要があります。

   > **Note**
   >
   > モノの名前で個人を特定できる情報を使用しないでください。モノの名前は、暗号化されていない通信やレポートに表示されることがあります。

5. このページの残りのフィールドは空のままにしておきます。[Next] を選びます。
6. [デバイス証明書を構成する-optional] ページで [新しい証明書を自動生成する (推奨)] を選択します。[Next] を選択します。
7. [証明書へのポリシーのアタッチ - optional] ページで、前のセクションで作成したポリシーを選択します。そのセクションでは、ポリシーの名前は **`My_Iot_Policy`** です。[モノを作成する] を選択します。
8. [証明書とキーのダウンロード] ページで:

   1. 各証明書およびキーファイルをダウンロードし、後で使用できるように保存します。これらのファイルをデバイスにインストールする必要があります。

      証明書ファイルを保存するときは、次の表に名前を付けます。これらは、後の例で使用されるファイル名です。

      証明書ファイル名

ファイル	ファイルパス
プライベートキー	`private.pem.key`
パブリックキー	(これらの例では使用されません)

ファイル	ファイルパス
デバイス証明書	`device.pem.crt`
ルート CA 証明書	`Amazon-root-CA-1.pem`

2. これらのファイルのルート CA ファイルをダウンロードするには、ルート CA 証明書ファイルの [Download] (ダウンロード) リンクをクリックします。このリンクは、使用しているデータエンドポイントと暗号スイートのタイプに対応します。このチュートリアルで、RSA 2048 ビットキー: Amazon ルート CA 1 の右側にある [ダウンロード] を選択し、RSA 2048 ビットキー: Amazon ルート CA 1 証明書ファイルをダウンロードします。

   > **Important**
   >
   > このページから移動する前に、証明書ファイルを保存する必要があります。コンソールでこのページから移動すると、証明書ファイルにはアクセスできなくなります。このステップで作成した証明書ファイルをダウンロードし忘れた場合は、このコンソール画面を終了し、コンソールのモノのリストに移動して、作成したモノのオブジェクトを削除してから、この手順を最初からやり直す必要があります。

3. [Done] を選択します。

この手順を完了すると、新しいモノのオブジェクトがモノのリストに表示されます。

デバイスを設定する

このセクションでは、デバイスを設定して AWS IoT Coreに接続する方法について説明します。の使用を開始する AWS IoT Core が、まだデバイスがない場合は、Amazon EC2 を使用して仮想デバイスを作成するか、Windows PC または Mac を IoT デバイスとして使用できます。

試す最適なデバイスオプションを選択します AWS IoT Core。もちろん、すべてを試すこともできますが、一度に試すのは 1 つのみにしてください。どのデバイスオプションが最適かわからない場合は、最適なデバイスオプションを選択する方法を読んでから、このページに戻ってください。

デバイスオプション

- Amazon EC2 を使用して仮想デバイスを作成する
- Windows または Linux の PC または Mac を AWS IoT デバイスとして使用する
- Raspberry Pi または他のデバイスを接続する

Amazon EC2 を使用して仮想デバイスを作成する

このチュートリアルでは、クラウドで仮想デバイスとして機能する Amazon EC2 インスタンスを作成します。

このチュートリアルを完了するには、 が必要です AWS アカウント。アカウントをお持ちではない場合、続行する前に、セットアップ AWS アカウント に記載されている手順を完了してください。

このチュートリアルでは、次の作業を行います。

- Amazon EC2 インスタンスをセットアップする
- Git、Node.js をインストールし、 を設定する AWS CLI
- 仮想デバイスの AWS IoT リソースを作成する
- AWS IoT Device SDK for JavaScript をインストールする
- サンプルアプリケーションを実行する
- AWS IoT コンソールでサンプルアプリからのメッセージを表示する

Amazon EC2 インスタンスをセットアップする

次の手順は、物理デバイスの代わりに仮想デバイスとして機能する Amazon EC2 インスタンスを作成する方法を示しています。

これが作成する最初の Amazon EC2 インスタンスの場合、「Amazon EC2 Linux インスタンスの開始方法」を参照してください。

インスタンスを起動するには

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. 左側のコンソールメニューから [Instances] (インスタンス) セクションを展開し、[Instances] (インスタンス) を選択します。[Instances] (インスタンス) ダッシュボードから、右側の [Launch instances] (インスタンスの起動) を選択すると、基本設定のリストが表示されます。
3. [Name and tags] (名前とタグ) セクションで、インスタンスの名前を入力し、オプションでタグを追加します。
4. [Application and OS Images (Amazon Machine Image)] (アプリケーションと OS イメージ (Amazon マシンイメージ)) セクションで、Amazon Linux 2 AMI (HVM) など、インスタンス用の AMI テンプレートを選択します。この AMI が「Free tier eligible」(無料利用枠対象) としてマークされていることに注意してください。

5. [Instance type] (インスタンスタイプ) セクションで、インスタンスのハードウェア設定を選択できます。デフォルトで選択されている `t2.micro` タイプを選択します。このインスタンスタイプは無料利用枠の対象であることに注意してください。
6. [Key pair (login)] (キーペア (ログイン)) セクションでドロップダウンリストからキーペア名を選択するか、[Create a new key pair] (新しいキーペアの作成) を選択して新しいキーペアを作成します。新しいキーペアを作成するときは、プライベートキーファイルをダウンロードして安全な場所に保存してください。これは、ダウンロードして保存する唯一の機会だからです。インスタンスを起動する際はキーペアの名前を指定する必要があり、インスタンスに接続する際は毎回対応するプライベートキーを指定する必要があります。

   > **Warning**
   >
   > [Proceed without a key pair] (キーペアオプションなしで続行) を選択しないでください。キーペアなしでインスタンスを起動すると、インスタンスに接続できません。

7. [Network settings] (ネットワーク設定) セクションと [Configure storage] (ストレージの設定) セクションでは、デフォルト設定のままでかまいません。準備ができたら、[Launch Instances] (インスタンスの起動) を選択します。
8. インスタンスを起動することを知らせる確認ページが表示されます。インスタンスの表示を選択して確認ページを閉じ、コンソールに戻ります。
9. インスタンス画面で、起動のステータスを確認できます。インスタンスの起動には短時間かかります。インスタンスを起動すると、その初期状態は `pending` です。インスタンスがスタートすると、その状態は `running` に変わり、公開 DNS 名を受け取ります。(公開 DNS (IPv4) 列が非表示の場合は、ページの右上隅にある 列の表示 / 非表示 (歯車のシェープをしたアイコン) を選択してから、公開 DNS (IPv4) を選択します。)
10. インスタンスが接続できるようになるまで、インスタンスの準備が整うまでに数分かかる場合があります。インスタンスのステータスチェックが正常に終了したことを確認してください。この情報は [Status Checks] 列で確認できます。

    新しいインスタンスがステータスチェックに合格したら、次の手順に進み、それに接続します。

インスタンスに接続するには

Amazon EC2 コンソールからインスタンスを選択し、Amazon EC2 Instance Connect を使用して接続することを選択することにより、ブラウザベースのクライアントを使用してインスタンスに接続できます。Instance Connect はアクセス許可を処理し、正常な接続を提供します。

1. Amazon EC2 コンソールの https://console.aws.amazon.com/ec2/ を開いてください。
2. 左のメニューで、[Instances] (インスタンス) を選択します。
3. インスタンスを選択し、[接続] を選択してください。
4. [Amazon EC2 Instance Connect]、[Connect] (接続) を選択します。

これで、新しい Amazon EC2 インスタンスにログインする Amazon EC2 Instance Connect ウィンドウができたはずです。

Git、Node.js をインストールし、 を設定する AWS CLI

このセクションでは、Git と Node.js を Linux インスタンスにインストールします。

Git をインストールするには

1. Amazon EC2 Instance Connect ウィンドウで、次のコマンドを使用してインスタンスを更新します。

   ```
   sudo yum update -y
   ```

2. Amazon EC2 Instance Connectウィンドウで、次のコマンドを使用して Git をインストールします。

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

3. Git がインストールされているかどうか、および Git の現在のバージョンを確認するには、次のコマンドを実行します。

   ```
   git --version
   ```

Node.js をインストールするには

1. Amazon EC2 Instance Connect ウィンドウで、次のコマンドを使用してノードバージョンマネージャー (nvm) をインストールします。

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

   nvm では Node.js の複数のバージョンをインストールすることができ、またそれらの切り替えもできるため、nvm を使用して Node.js をインストールします。

2. Amazon EC2 Instance Connect ウィンドウで、このコマンドを使用して nvm を有効にします。

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

3. Amazon EC2 Instance Connect ウィンドウで、このコマンドを使用して、nvm を使用し、最新バージョンの Node.js をインストールします。

```
nvm install 16
```

> **Note**
>
> Node.js の最新の LTS リリースがインストールされます。

Node.js をインストールすると、Node Package Manager (npm) もインストールされるため、必要に応じて追加のモジュールをインストールできます。

4. Amazon EC2 Instance Connect ウィンドウで、このコマンドを使用して、その Node.js がインストールされ、正しく実行されていることをテストします。

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

このチュートリアルでは Node v10.0 以降が必要です。詳細については、「チュートリアル: Amazon EC2 インスタンスでの Node.js のセットアップ」を参照してください。

を設定するには AWS CLI

Amazon EC2 インスタンスには、 AWS CLIがプリロードされています。ただし、 AWS CLI プロファイルを完了する必要があります。CLI の設定方法の詳細については、「AWS CLIの設定」を参照してください。

1. 次の例は、サンプル値を示しています。それらを自分の値に置き換えます。これらの値は、AWS コンソールの [My Security Credentials] (セキュリティ認証情報) の下のアカウント情報で確認できます。

   Amazon EC2 Instance Connect ウィンドウで、次のコマンドを入力します。

```
aws configure
```

その後、表示されるプロンプトでアカウントの値を入力します。

```
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. 次のコマンドを使用して AWS CLI 設定をテストできます。

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

が正しく設定されている場合、 コマンド AWS CLI は からエンドポイントアドレスを返す必要があります AWS アカウント。

仮想デバイスの AWS IoT リソースを作成する

このセクションでは、 を使用してモノのオブジェクトとその証明書ファイルを仮想デバイスに直接 AWS CLI 作成する方法について説明します。これは、別のコンピュータからデバイスにコピーすることで生じる可能性のある複雑さを避けるために、デバイス上で直接行われます。このセクションでは、仮想デバイス用に次のリソースを作成します。

- 仮想デバイスを表すモノのオブジェクト AWS IoT。
- 仮想デバイスを認証するための証明書。
- 仮想デバイスが AWS IoTに接続し、メッセージを発行、受信、およびサブスクライブすることを許可するポリシードキュメント。

Linux インスタンスで AWS IoT モノのオブジェクトを作成するには

に接続されたデバイスは AWS IoT 、 AWS IoT レジストリ内のモノのオブジェクトによって表されます。モノのオブジェクトは、特定のデバイスまたは論理エンティティを表します。この場合、モノのオブジェクトは仮想デバイス、つまりこの Amazon EC2 インスタンスを表します。

1. Amazon EC2 Instance Connect ウィンドウで、次のコマンドを実行してモノのオブジェクトを作成します。

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

2. JSON レスポンスは以下のようになります。

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

Linux インスタンスで AWS IoT キーと証明書を作成してアタッチするには

create-keys-and-certificate コマンドを実行すると、Amazon ルート認証局によって署名されたクライアント証明書が作成されます。この証明書は、仮想デバイスの ID を認証するために使用されます。

1. Amazon EC2 Instance Connectウィンドウで、証明書とキーファイルを保存するディレクトリを作成します。

   ```
   mkdir ~/certs
   ```

2. Amazon EC2 Instance Connect ウィンドウで、このコマンドを使用して、Amazon 認証機関 (CA) 証明書のコピーをダウンロードします。

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

3. Amazon EC2 Instance Connect ウィンドウで、次のコマンドを実行して、プライベートキー、パブリックキー、および X.509 証明書ファイルを作成します。このコマンドは、証明書を に登録してアクティブ化します 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"
   ```

   レスポンスは次のようになります。certificateArn を保存して、後続のコマンドで使用できるようにします。証明書をモノにアタッチし、後の手順で証明書にポリシーをアタッチするには、この証明書が必要になります。

   ```
   {
       "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
+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 ウィンドウで、次のコマンドと、前のコマンドからの応答に含まれている *certificateArn* を使用して、作成したばかりの証明書にモノのオブジェクトをアタッチします。

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

成功した場合、このコマンドは出力を表示しません。

ポリシーを作成してアタッチするには

1. Amazon EC2 Instance Connectウィンドウで、このポリシードキュメントをコピーして **~/policy.json**という名前のファイルに貼り付け、ポリシーファイルを作成します。

   お気に入りの Linux エディタがなければ、このコマンドを使用して nano を開くことができます。

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

   その中に policy.json のポリシードキュメントを貼り付けます。ctrl-x で nano エディタを終了し、ファイルを保存します。

   policy.json のポリシードキュメントの内容。

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

2. Amazon EC2 Instance Connect ウィンドウで、次のコマンドを使用してポリシーを作成します。

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

出力:

```
{
    "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ウィンドウで、次のコマンドを使用して、仮想デバイスの証明書にポリシーをアタッチします。

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

成功した場合、このコマンドは出力を表示しません。

AWS IoT Device SDK for JavaScript をインストールする

このセクションでは、 AWS IoT Device SDK for JavaScript をインストールします。これには、アプリケーションが AWS IoT およびサンプルプログラムとの通信に使用できるコードが含まれてい

ます。詳細については、「AWS IoT Device SDK for JavaScript GitHub repository」(Device SDK for JavaScript GitHub リポジトリ) を参照してください。

Linux インスタンスに AWS IoT Device SDK for JavaScript をインストールするには

1. Amazon EC2 Instance Connect ウィンドウで、このコマンドを使用して AWS IoT Device SDK for JavaScript リポジトリをホームaws-iot-device-sdk-js-v2ディレクトリの ディレクトリにクローンします。

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

2. 前のステップで作成した aws-iot-device-sdk-js-v2 ディレクトリに移動します。

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

3. npm を使用して SDK をインストールします。

```
npm install
```

サンプルアプリケーションを実行する

次のセクションのコマンドは、次の表に示すように、キーおよび証明書ファイルがご利用の仮想デバイスに保存されていることを前提としています。

証明書ファイル名

ファイル	ファイルパス
プライベートキー	~/certs/private.pem.key
デバイス証明書	~/certs/device.pem.crt
ルート CA 証明書	~/certs/Amazon-root-CA-1.pem

このセクションでは、 AWS IoT Device SDK for JavaScript の aws-iot-device-sdk-js-v2/samples/node ディレクトリにあるpub-sub.jsサンプルアプリケーションをインストールして実行します。このアプリケーションは、デバイス (Amazon EC2 インスタンス) が MQTT ライブラリを使用して MQTT メッセージを発行およびサブスクライブする方法を示します。pub-sub.js サンプ

ルアプリケーションは、トピック、topic_1 をサブスクライブし、そのトピックに対して 10 個のメッセージを発行し、メッセージブローカーから受信したメッセージを表示します。

サンプルアプリケーションをインストールして実行するには

1. Amazon EC2 Instance Connect ウィンドウで、SDK が作成した aws-iot-device-sdk-js-v2/samples/node/pub_sub ディレクトリに移動し、これらのコマンドを使用してサンプルアプリケーションをインストールします。

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

2. Amazon EC2 Instance Connect ウィンドウで、このコマンド AWS IoT を使用して から *your-iot-endpoint* を取得します。

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

3. Amazon EC2 Instance Connect ウィンドウで、示されているように *your-iot-endpoint* を挿入し、このコマンドを実行します。

   ```
   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
   ```

サンプルアプリケーション:

1. AWS IoT Core アカウントの に接続します。
2. メッセージトピック topic_1 をサブスクライブし、そのトピックで受信したメッセージを表示します。
3. 10 個のメッセージをトピック、topic_1 に発行します。
4. 次のような出力を表示します。

```
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}
```

サンプルアプリケーションの実行に問題がある場合は、the section called “サンプルアプリケーションの問題をトラブルシューティングする” を確認してください。

コマンドラインに `--verbosity debug` パラメータを追加して、サンプルアプリケーションが実行内容に関する詳細なメッセージを表示するようにすることもできます。この情報は、問題の修正に役立つ場合があります。

AWS IoT コンソールでサンプルアプリからのメッセージを表示する

AWS IoT コンソールのMQTT テストクライアントを使用することで、サンプルアプリケーションメッセージがメッセージブローカーを通過するときにそれらを見ることができます。

サンプルアプリケーションによって発行された MQTT メッセージを表示するには

1. 確認MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する。これは、AWS IoT コンソールで MQTT テストクライアントを使用して、メッセージブローカーを通過する MQTT メッセージを表示する方法を学ぶのに役立ちます。
2. AWS IoT コンソールで MQTT テストクライアントを開きます。
3. 「トピックへのサブスクライブ」で、「topic_1」というトピックをサブスクライブします。
4. Amazon EC2 Instance Connect ウィンドウで、サンプルアプリケーションを再度実行し、AWS IoT コンソールの MQTT テストクライアントのメッセージを確認します。

```
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 と がプロトコル AWS IoT Core をサポートする方法の詳細については、「MQTT」を参照してください。

Windows または Linux の PC または Mac を AWS IoT デバイスとして使用する

このチュートリアルでは、 で使用するパーソナルコンピュータを設定します AWS IoT。これらの手順は、Windows および Linux の PC および Mac をサポートしています。これを行うには、コンピュータにソフトウェアをインストールする必要があります。コンピュータにソフトウェアをインストールしたくない場合は、すべてのソフトウェアを仮想マシンにインストールする Amazon EC2 を使用して仮想デバイスを作成する を試すことができます。

このチュートリアルでは、次の作業を行います。

- パーソナルコンピュータを設定する
- Git、Python、および AWS IoT Device SDK for Python のインストール
- ポリシーを設定し、サンプルアプリケーションを実行する
- AWS IoT コンソールでサンプルアプリからのメッセージを表示する
- Python で共有サブスクリプションのサンプルを実行する

パーソナルコンピュータを設定する

このチュートリアルを完了するには、インターネットに接続した Windows もしくは Linux PC または Mac が必要です。

次のステップに進む前に、コンピュータでコマンドラインウィンドウを開くことができることを確認してください。Windows PC で cmd.exe を使用します。Linux PC または Mac では、Terminal を使用します。

Git、Python、および AWS IoT Device SDK for Python のインストール

このセクションでは、Python と AWS IoT Device SDK for Python をコンピュータにインストールします。

Git と Python の最新バージョンをインストールする

この手順では、最新バージョンの Git と Python をパーソナルコンピュータにインストールする方法について説明します。

Git と Python をダウンロードしてコンピュータにインストールするには

1. Git がコンピュータにインストールされているかどうかを確認します。このコマンドをコマンドラインに入力します。

   ```
   git --version
   ```

   コマンドが Git バージョンを表示する場合は、Git がインストールされており、次のステップに進むことができます。

   コマンドがエラーを表示する場合は、https://git-scm.com/downloadを開いてコンピュータに Git をインストールします。
2. Python が既にインストールされているかどうかを確認します。このコマンドをコマンドラインに入力します。

   ```
   python -V
   ```

   > **Note**
   >
   > このコマンドがエラー `Python was not found` を表示する場合、オペレーティングシステムが Python v3.x 実行可能ファイルを `Python3` として呼び出していることが原因である可能性があります。その場合は、`python` のすべてのインスタンスを `python3` に置き換えて、このチュートリアルの残りの部分を続行してください。

   コマンドが Python のバージョンを表示する場合、Python は既にインストールされています。このチュートリアルには、Python v3.7 以降が必要です。
3. Python がインストールされている場合は、このセクションの残りの手順を省略できます。インストールされていない場合は、続行します。
4. https://www.python.org/downloads/ を開き、コンピュータ用のインストーラをダウンロードします。
5. ダウンロードが自動的にインストールを開始しなかった場合は、ダウンロードしたプログラムを実行して Python をインストールします。
6. Python のインストールを確認します。

   ```
   python -V
   ```

コマンドが Python バージョンを表示することを確認します。Python のバージョンが表示されない場合は、再度 Python をダウンロードしてインストールしてください。

AWS IoT Device SDK for Python のインストール

Device AWS IoT SDK for Python をコンピュータにインストールするには

1. AWS IoT Device SDK for Python の v2 をインストールします。

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

2. AWS IoT Device SDK for Python リポジトリをホームディレクトリの aws-iot-device-sdk-python-v2 ディレクトリにクローンします。この手順は、###としてインストールするファイルのベースディレクトリを参照しています。

   ###ディレクトリの実際の場所は、オペレーティングシステムによって異なります。

   Linux/macOS

   macOS および Linux では、###ディレクトリは ~ です。

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

   Windows

   Windows では、 cmdウィンドウでこのコマンドを実行すると、###ディレクトリパスを見つけることができます。

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

   > **Note**
   >
   > cmd.exe ではなく Windows PowerShell を使用している場合は、次のコマンドを使用します。

```
echo $home
```

詳細については、「AWS IoT Device SDK for JavaScript GitHub repository」(Device SDK for JavaScript GitHub リポジトリ) を参照してください。

サンプルアプリケーションの実行を準備する

サンプルアプリケーションを実行するためにシステムを準備するには

- certs ディレクトリを作成します。certs でモノのオブジェクトを作成および登録したときに保存したプライベートキー、デバイス証明書、およびルート CA 証明書ファイルを the section called “AWS IoT リソースの作成” ディレクトリにコピーします。送信先ディレクトリ内の各ファイルのファイル名は、テーブル内のファイル名と一致する必要があります。

  次のセクションのコマンドは、次の表に示すように、キーおよび証明書ファイルがデバイスに保存されていることを前提としています。

  Linux/macOS

  このコマンドを実行して、certs サブディレクトリを作成します。このサブディレクトリは、サンプルアプリケーションの実行に使用します。

  ```
  mkdir ~/certs
  ```

  新しいサブディレクトリの、次の表に示す送信先ファイルのパスにファイルをコピーします。

  証明書ファイル名

ファイル	ファイルパス
プライベートキー	~/certs/private.pem.key
デバイス証明書	~/certs/device.pem.crt
ルート CA 証明書	~/certs/Amazon-root-CA-1.pem

このコマンドを実行して、certs ディレクトリ内のファイルを一覧表示し、それらを表に一覧表示されているファイルと比較します。

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

Windows

このコマンドを実行して、certs サブディレクトリを作成します。このサブディレクトリは、サンプルアプリケーションの実行に使用します。

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

新しいサブディレクトリの、次の表に示す送信先ファイルのパスにファイルをコピーします。

証明書ファイル名

ファイル	ファイルパス
プライベートキー	%USERPROFILE%\certs\private.pem.key
デバイス証明書	%USERPROFILE%\certs\device.pem.crt
ルート CA 証明書	%USERPROFILE%\certs\Amazon-root-CA-1.pem

このコマンドを実行して、certs ディレクトリ内のファイルを一覧表示し、それらを表に一覧表示されているファイルと比較します。

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

ポリシーを設定し、サンプルアプリケーションを実行する

このセクションでは、ポリシーを設定し、 AWS IoT Device SDK for Pythonの `aws-iot-device-sdk-python-v2/samples` ディレクトリにある `pubsub.py` サンプルスクリプトを実行します。このスクリプトは、デバイスが MQTT ライブラリを使用して MQTT メッセージを発行およびサブスクライブする方法を示します。

`pubsub.py`サンプルアプリケーションは、トピック、`test/topic` をサブスクライブし、そのトピックに対して10 個のメッセージを発行し、メッセージブローカーから受信したメッセージを表示します。

`pubsub.py` サンプルスクリプトを実行するには、次の情報が必要です。

アプリケーションパラメータ値

Parameter	値がある場所
`your-iot-endpoint`	1. AWS IoT コンソールの左のメニューで、[Settings] (設定) を選択します。 2. [Setting] (設定) ページで、エンドポイントがデバイスデータエンドポイントセクションに表示されます。

`your-iot-endpoint` 値の形式は *`endpoint_id`*`-ats.iot.`*`region`*`.amazonaws.com` です (例: `a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com`)。

スクリプトを実行する前に、モノのポリシーが、接続、サブスクライブ、発行、および受信するためのアクセス許可をサンプルスクリプトに提供していることを確認してください。

モノのリソースのポリシードキュメントを検索して確認するには

1. AWS IoT コンソールの [Things] (モノ) リストで、デバイスを表すモノのリソースを検索します。
2. デバイスを表すモノのリソースの [Name] (名前) リンクを選択して、[Thing details] (モノの詳細) ページを開きます。
3. [Thing details] (モノの詳細) ページの [Certificates] (証明書) タブで、モノのリソースにアタッチされている証明書を選択します。リストに含まれる証明書は 1 つだけにする必要があります。複数の証明書がある場合は、デバイスにインストールされているファイルを含み、そのファイルが AWS IoT Coreへの接続に使用される証明書を選択します。

[Certificate] (証明書) 詳細ページの [Policies] (ポリシー) タブで、証明書にアタッチされているポリシーを選択します。1 つだけにする必要があります。複数のポリシーがある場合は、それぞれについて次の手順を繰り返して、少なくとも 1 つのポリシーで必要なアクセスが許可されていることを確認します。

4. [Policy] (ポリシー) 概要ページで、JSON エディタを検索し、[Edit policy document] (ポリシードキュメントの編集) を選択して、必要に応じてポリシードキュメントを確認および編集します。

5. 次の例では、ポリシー JSON が表示されます。"Resource" 要素で、 を各Resource値の AWS アカウント AWS リージョン と *region:account*に置き換えます。

```
{
    "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/macOS でサンプルスクリプトを実行するには

1. コマンドラインウィンドウで、SDK がこれらのコマンドを使用して作成した ~/aws-iot-device-sdk-python-v2/samples/node/pub_sub ディレクトリに移動します。

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

2. コマンドラインウィンドウに示されているように、*your-iot-endpoint* を置き換えて、このコマンドを実行します。

```
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 PC でサンプルアプリケーションを実行するには

1. コマンドラインウィンドウで、SDK が作成した %USERPROFILE%\aws-iot-device-sdk-python-v2\samples ディレクトリに移動し、これらのコマンドを使用してサンプルアプリケーションをインストールします。

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

2. コマンドラインウィンドウに示されているように、*your-iot-endpoint* を置き換えて、このコマンドを実行します。

```
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
```

サンプルスクリプト:

1. AWS IoT Core アカウントの に接続します。

2. メッセージトピックの test/topic をサブスクライブし、そのトピックで受信したメッセージを表示します。
3. トピック test/topic に 10 個のメッセージを発行します。
4. 次のような出力を表示します。

```
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!
```

サンプルアプリケーションの実行に問題がある場合は、the section called “サンプルアプリケーションの問題をトラブルシューティングする” を確認してください。

コマンドラインに `--verbosity Debug` パラメータを追加して、サンプルアプリケーションが実行内容に関する詳細なメッセージを表示するようにすることもできます。この情報は、問題の修正に役立ちます。

AWS IoT コンソールでサンプルアプリからのメッセージを表示する

AWS IoT コンソールのMQTT テストクライアントを使用することで、サンプルアプリケーションメッセージがメッセージブローカーを通過するときにそれらを見ることができます。

サンプルアプリケーションによって発行された MQTT メッセージを表示するには

1. 確認MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する。これは、AWS IoT コンソールで MQTT テストクライアントを使用して、メッセージブローカーを通過する MQTT メッセージを表示する方法を学ぶのに役立ちます。
2. AWS IoT コンソールで MQTT テストクライアントを開きます。
3. 「トピックへのサブスクライブ」で、トピック、「テスト/トピック」をサブスクライブします。
4. コマンドラインウィンドウで、サンプルアプリケーションを再度実行し、AWS IoT コンソールの MQTT クライアントでメッセージを確認します。

   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 と がプロトコル AWS IoT Core をサポートする方法の詳細については、「MQTT」を参照してください。

Python で共有サブスクリプションのサンプルを実行する

AWS IoT Core は、MQTT 3 と MQTT 5 の両方の共有サブスクリプションをサポートしています。共有サブスクリプションを使用すると、1 つのトピックへのサブスクリプションを複数のクライアントで共有できますが、そのトピックに公開されたメッセージをランダム配信を使って受信できるのは 1 つのクライアントのみです。共有サブスクリプションを使用するには、クライアントで共有サブスク

リプションのトピックフィルター: $share/{ShareName}/{TopicFilter} をサブスクライブします。

ポリシーを設定し、共有サブスクリプションのサンプルを実行するには

1. 共有サブスクリプションのサンプルを実行するには、「MQTT 5 共有サブスクリプション」に記載されているようにモノのポリシーを設定する必要があります。
2. 共有サブスクリプションのサンプルを実行するには、次のコマンドを実行します。

   Linux/macOS

   Linux/macOS でサンプルスクリプトを実行するには

   1. コマンドラインウィンドウで、SDK がこれらのコマンドを使用して作成した ~/aws-iot-device-sdk-python-v2/samples ディレクトリに移動します。

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

   2. コマンドラインウィンドウに示されているように、*your-iot-endpoint* を置き換えて、このコマンドを実行します。

      ```
      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 PC でサンプルアプリケーションを実行するには

   1. コマンドラインウィンドウで、SDK が作成した %USERPROFILE%\aws-iot-device-sdk-python-v2\samples ディレクトリに移動し、これらのコマンドを使用してサンプルアプリケーションをインストールします。

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

   2. コマンドラインウィンドウに示されているように、*your-iot-endpoint* を置き換えて、このコマンドを実行します。

      ```
      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**
>
> オプションとして、必要に応じ、サンプルを実行する際にグループ識別子を指定できます (例: `--group_identifier consumer`)。グループ識別子を指定しなかった場合、`python-sample` がデフォルトのグループ識別子です。

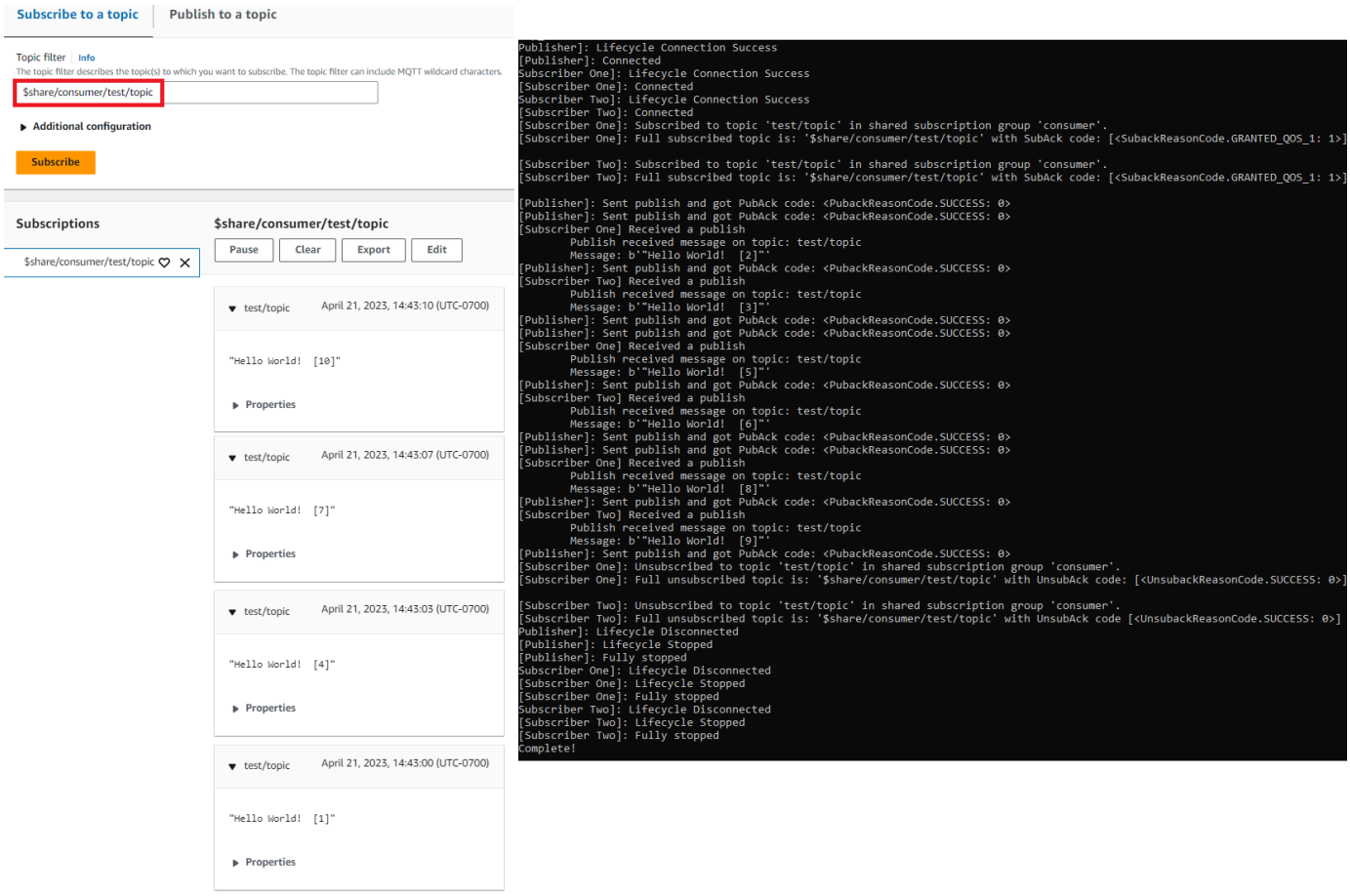
3. コマンドラインの出力は次のようになります。

```
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 コンソールで [MQTT テストクライアント] を開きます。[トピックをサブスクライブする] で、次のような共有サブスクリプションのトピックをサブスクライブします: `$share/consumer/test/topic` 必要に応じて、サンプルを実行する際にグループ識別子を指定できます (例: `--group_identifier consumer`)。グループ識別子を指定しなかった場合、デフォルト値は `python-sample` です。詳細については、「MQTT 5 共有サブスクリプション Python サンプル」と「AWS IoT Core 開発者ガイド」の「共有サブスクリプション」を参照してください。

   コマンドラインウィンドウで、サンプルアプリケーションを再度実行し、AWS IoT コンソールの [MQTT テストクライアント] でメッセージの配信を確認します。



Raspberry Pi または他のデバイスを接続する

このセクションでは、 で使用する Raspberry Pi を設定します AWS IoT。接続したい別のデバイスがある場合、Raspberry Pi の手順には、これらの指示をデバイスに合わせて適用するのに役立つ参照先が含まれています。

通常、この処理には約 20 分かかりますが、システムソフトウェアのアップグレードが多数ある場合は、インストールにより長い時間を要する場合があります。

このチュートリアルでは、次の作業を行います。

- デバイスをセットアップする
- AWS IoT Device SDK に必要なツールとライブラリをインストールする
- AWS IoT Device SDK のインストール
- サンプルアプリケーションをインストールして実行する
- AWS IoT コンソールでサンプルアプリからのメッセージを表示する

> **Important**
>
> これらの指示を他のデバイスやオペレーティングシステムに合わせて適用するのが難しい場合があります。これらの指示を解釈してご利用のデバイスに適用するには、そのデバイスを十分に理解する必要があります。
> デバイスの設定中に問題が発生した場合は AWS IoT、 Amazon EC2 を使用して仮想デバイスを作成するや などの他のデバイスオプションのいずれかを代わりに試すことができますWindows または Linux の PC または Mac を AWS IoT デバイスとして使用する。

デバイスをセットアップする

このステップの目的は、オペレーティングシステム (OS) を起動し、インターネットに接続し、コマンドラインインターフェイスでデバイスを操作できるようにデバイスを設定するために必要なものを用意することです。

このチュートリアルを完了するには、以下が必要です。

- AWS アカウント。アカウントをお持ちではない場合、続行する前に、セットアップ AWS アカウント に記載されている手順を完了してください。
- Raspberry Pi 3 モデル B 以降の最新のモデル。これは、Raspberry Pi の以前のバージョンで動作する可能性がありますが、テストされていません。
- Raspberry Pi OS (32 ビット)または、それ以降。Raspberry Pi OS の最新バージョンを使用することをお勧めします。以前のバージョンの OS は動作する可能性がありますが、テストされていません。

この例を実行するために、グラフィカルユーザーインターフェイス (GUI) でデスクトップをインストールする必要はありません。ただし、Raspberry Pi を初めて使用し、Raspberry Pi ハードウェアがサポートしている場合は、デスクトップを GUI で使用する方が簡単です。

- イーサネットまたは WiFi 接続。
- キーボード、マウス、モニタ、ケーブル、電源装置、およびデバイスに必要なその他のハードウェア。

> **Important**
>
> 次のステップに進む前に、デバイスにオペレーティングシステムがインストール、設定、および実行されている必要があります。デバイスはインターネットに接続されていなければなりません。また、コマンドラインインターフェイスを使用してデバイスにアクセスできる必要があります。コマンドラインアクセスは、直接接続されたキーボード、マウス、モニタを介して、または SSH ターミナルリモートインターフェイスを使用して行うことができます。

グラフィカルユーザーインターフェイス (GUI) を備えた Raspberry Pi 上でオペレーティングシステムを実行している場合は、デバイス上でターミナルウィンドウを開き、そのウィンドウで以下の手順を実行します。それ以外の場合であって、PuTTY などのリモートターミナルを使用してデバイスに接続するときは、デバイスへのリモートターミナルを開いてそれを使用します。

AWS IoT Device SDK に必要なツールとライブラリをインストールする

AWS IoT Device SDK とサンプルコードをインストールする前に、システムが最新であり、SDKs をインストールするために必要なツールとライブラリがあることを確認してください。

1. オペレーティングシステムを更新し、必要なライブラリをインストールします。

   AWS IoT Device SDK をインストールする前に、デバイスのターミナルウィンドウでこれらのコマンドを実行してオペレーティングシステムを更新し、必要なライブラリをインストールします。

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```

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

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

2. Git をインストールする

   デバイスのオペレーティングシステムに Git がインストールされていない場合は、インストールして AWS IoT Device SDK for JavaScript をインストールする必要があります。

   a. Git が既にインストールされているかどうかを確認するために、このコマンドを実行してテストします。

      ```
      git --version
      ```

   b. 前のコマンドで Git バージョンが返された場合、Git は既にインストールされており、ステップ 3 に進むことができます。

   c. git コマンドを実行するとエラーが表示される場合は、このコマンドを実行して Git をインストールします。

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

   d. このコマンドを実行して、Git がインストールされているかどうかをもう一度テストします。

      ```
      git --version
      ```

   e. Git がインストール済みである場合は、次のセクションに進みます。インストールされていない場合は、トラブルシューティングを行い、エラーを修正してから続行します。 AWS IoT Device SDK for JavaScript をインストールするには、Git が必要です。

AWS IoT Device SDK のインストール

AWS IoT Device SDK をインストールします。

Python

このセクションでは、Python、その開発ツール、および AWS IoT Device SDK for Python をデバイスにインストールします。これらの手順は、最新の Raspberry Pi OS を実行する Raspberry Pi

を対象としています。別のデバイスをお持ちの場合、または別のオペレーティングシステムを使用している場合は、これらの手順をデバイスに合わせて調整する必要があります。

1. Python およびその開発ツールをインストールする

   AWS IoT Device SDK for Python では、Python v3.5 以降を Raspberry Pi にインストールする必要があります。

   デバイスのターミナルウィンドウで、次のコマンドを実行します。

   1. このコマンドを実行して、デバイスにインストールされている Python のバージョンを確認します。

      ```
      python3 --version
      ```

      Python がインストールされている場合は、そのバージョンが表示されます。
   2. 表示されているバージョンが Python 3.5 以降の場合は、ステップ 2 にスキップできます。
   3. 表示されたバージョンが Python 3.5 よりも前の場合は、このコマンドを実行して正しいバージョンをインストールできます。

      ```
      sudo apt install python3
      ```
   4. このコマンドを実行して、正しいバージョンの Python がインストールされていることを確認します。

      ```
      python3 --version
      ```

2. pip3 をテストする

   デバイスのターミナルウィンドウで、次のコマンドを実行します。

   1. このコマンドを実行して、pip3 がインストールされているかどうかを確認します。

      ```
      pip3 --version
      ```
   2. コマンドがバージョン番号を返す場合、pip3 はインストールされており、ステップ 3 にスキップできます。

3. 前のコマンドがエラーを返す場合は、このコマンドを実行して pip3 をインストールします。

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

4. このコマンドを実行して、pip3 がインストールされているかどうかを確認します。

```
pip3 --version
```

3. 現在の AWS IoT Device SDK for Python をインストールする

AWS IoT Device SDK for Python をインストールし、サンプルアプリケーションをデバイスにダウンロードします。

デバイスで、次のコマンドを実行します。

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

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

JavaScript

このセクションでは、Node.js、npm パッケージマネージャー、 AWS IoT Device SDK for JavaScript をデバイスにインストールします。これらの手順は、Raspberry Pi OS を実行している Raspberry Pi を対象としています。別のデバイスをお持ちの場合、または別のオペレーティングシステムを使用している場合は、これらの手順をデバイスに合わせて調整する必要があります。

1. Node.js の最新バージョンをインストールする

AWS IoT Device SDK for JavaScript では、Node.js と npm パッケージマネージャーを Raspberry Pi にインストールする必要があります。

a. このコマンドを入力して、ノードリポジトリの最新バージョンをダウンロードします。

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

b. Node と npm をインストールします。

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

c. Node のインストールを確認します。

```
node -v
```

コマンドが Node のバージョンを表示することを確認します。このチュートリアルでは Node v10.0 以降が必要です。Node のバージョンが表示されない場合は、Node リポジトリを再度ダウンロードしてみてください。

d. npm のインストールを確認します。

```
npm -v
```

コマンドが npm バージョンを表示することを確認します。npm のバージョンが表示されない場合は、Node と npm をもう一度インストールしてみてください。

e. デバイスを再起動します。

```
sudo shutdown -r 0
```

デバイスの再起動後に続行します。

2. AWS IoT Device SDK for JavaScript をインストールする

Raspberry Pi に AWS IoT Device SDK for JavaScript をインストールします。

a. AWS IoT Device SDK for JavaScript リポジトリを###aws-iot-device-sdk-js-v2ディレクトリの ディレクトリにクローンします。Raspberry Pi では、###ディレクトリは ~/ であり、次のコマンドで###ディレクトリとして使用されます。デバイスが###ディレクトリに別のパスを使用している場合は、次のコマンドで ~/ をデバイスの正しいパスに置き換える必要があります。

これらのコマンドは、~/aws-iot-device-sdk-js-v2 ディレクトリを作成し、SDK コードをそのディレクトリにコピーします。

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

b. 前の手順で作成した aws-iot-device-sdk-js-v2 ディレクトリに変更し、npm install を実行して SDK をインストールします。コマンド npm install が aws-crt ライブラリビルドを呼び出し、これは完了するまで数分かかることがあります。

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

サンプルアプリケーションをインストールして実行する

このセクションでは、 AWS IoT Device SDK にあるpubsubサンプルアプリケーションをインストールして実行します。このアプリケーションは、デバイスが MQTT ライブラリを使用して MQTT メッセージを発行およびサブスクライブする方法を示します。サンプルアプリケーションは トピック topic_1 にサブスクライブし、そのトピックに 10 個のメッセージを発行し、メッセージブローカーから受信したメッセージを表示します。

証明書ファイルをインストールする

サンプルアプリケーションでは、デバイスを認証する証明書ファイルをそのデバイスにインストールする必要があります。

サンプルアプリケーションのデバイス証明書ファイルをインストールするには

1. これらのコマンドを実行して、###ディレクトリにcertsサブディレクトリを作成します。

```
cd ~
mkdir certs
```

2. ~/certs ディレクトリで、前に the section called “AWS IoT リソースの作成” で作成したプライベートキー、デバイス証明書、およびルート CA 証明書をコピーします。

   証明書ファイルをデバイスにコピーする方法は、デバイスおよびオペレーティングシステムによって異なりますが、ここでは説明を割愛します。ただし、デバイスがグラフィカルユーザーインターフェイス (GUI) をサポートしており、ウェブブラウザがある場合は、デバイスのウェブブラウザから the section called “AWS IoT リソースの作成” で説明されている手順を実行して、結果のファイルをデバイスに直接ダウンロードできます。

   次のセクションのコマンドは、次の表に示すように、キーおよび証明書ファイルがデバイスに保存されていることを前提としています。

証明書ファイル名

ファイル	ファイルパス
ルート CA 証明書	~/certs/Amazon-root-CA-1.pem
デバイス証明書	~/certs/device.pem.crt
プライベートキー	~/certs/private.pem.key

サンプルアプリケーションを実行するには、次の情報が必要です。

アプリケーションパラメータ値

Parameter	値がある場所
your-iot-endpoint	AWS IoT コンソールで、[All devices] (すべてのデバイス)、[Things] (モノ) の順に選択します。 AWS IoT メニューの [設定] ページ。エンドポイントが [Device data endpoint] (デバイスデータエンドポイント) セクションに表示されます。

your-iot-endpoint 値の形式は*endpoint_id*-ats.iot.*region*.amazonaws.com です (例: a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com)。

Python

サンプルアプリケーションをインストールして実行するには

1. サンプルアプリディレクトリに移動します。

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

2. コマンドラインウィンドウで、示されているように *your-iot-endpoint* を置き換えて、このコマンドを実行します。

```
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. サンプルアプリケーションが次のようになっていることを観察します。

   1. アカウントの AWS IoT サービスに接続します。
   2. メッセージトピック topic_1 をサブスクライブし、そのトピックで受信したメッセージを表示します。
   3. 10 個のメッセージをトピック、topic_1 に発行します。
   4. 次のような出力を表示します。

```
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!
```

サンプルアプリケーションの実行に問題がある場合は、the section called “サンプルアプリケーションの問題をトラブルシューティングする” を確認してください。

コマンドラインに `--verbosity Debug` パラメータを追加して、サンプルアプリケーションが実行内容に関する詳細なメッセージを表示するようにすることもできます。この情報は、問題の修正に役立つ場合があります。

JavaScript

サンプルアプリケーションをインストールして実行するには

1. コマンドラインウィンドウで、SDK が作成した `~/aws-iot-device-sdk-js-v2/samples/node/pub_sub` ディレクトリに移動し、これらのコマンドを使用してサンプルアプリケーションをインストールします。コマンド `npm install` が `aws-crt` ライブラリビルドを呼び出し、これは完了するまで数分かかることがあります。

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

2. コマンドラインウィンドウで、示されているように *`your-iot-endpoint`* を置き換えて、このコマンドを実行します。

   ```
   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. サンプルアプリケーションが次のようになっていることを観察します。

   1. アカウントの AWS IoT サービスに接続します。
   2. メッセージトピック topic_1 をサブスクライブし、そのトピックで受信したメッセージを表示します。
   3. 10 個のメッセージをトピック、topic_1 に発行します。
   4. 次のような出力を表示します。

   ```
   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}
```

サンプルアプリケーションの実行に問題がある場合は、the section called “サンプルアプリケーションの問題をトラブルシューティングする” を確認してください。

コマンドラインに `--verbosity Debug` パラメータを追加して、サンプルアプリケーションが実行内容に関する詳細なメッセージを表示するようにすることもできます。この情報は、問題の修正に役立つ場合があります。

AWS IoT コンソールでサンプルアプリからのメッセージを表示する

AWS IoT コンソールのMQTT テストクライアントを使用することで、サンプルアプリケーションメッセージがメッセージブローカーを通過するときにそれらを見ることができます。

サンプルアプリケーションによって発行された MQTT メッセージを表示するには

1. 確認MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する。これは、AWS IoT コンソールで MQTT テストクライアントを使用して、メッセージブローカーを通過する MQTT メッセージを表示する方法を学ぶのに役立ちます。
2. AWS IoT コンソールで MQTT テストクライアントを開きます。
3. トピック topic_1 にサブスクライブします。
4. コマンドラインウィンドウで、サンプルアプリケーションを再度実行し、AWS IoT コンソールの MQTT クライアントでメッセージを確認します。

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
```

サンプルアプリケーションの問題をトラブルシューティングする

サンプルアプリケーションを実行しようとしたときにエラーが発生した場合は、次の点を確認してください。

証明書を確認する

証明書がアクティブでない場合、 AWS IoT はその証明書を認可に使用する接続試行を受け入れません。証明書を作成するときに、[Activate] (有効化) ボタンを見落としがちです。幸いなことに、AWS IoT コンソールから証明書を有効化できます。

証明書の有効化を確認するには

1. AWS IoT コンソールの左側のメニューで、[Secure] (安全性) を選択し、[Certificates] (証明書) を選択します。
2. 証明書のリストで、演習用に作成した証明書を見つけ、[Status] (ステータス) 列でそのステータスを確認します。

   証明書の名前を覚えていない場合は、[Inactive] (無効) の証明書がないかを確認し、使用している証明書かどうかを確認してください。

   リストから証明書を選択し、詳細のページを開きます。詳細ページには、証明書の特定に役立つ [Create date] (作成日) が表示されます。

3. 有効化されていない証明書を有効化するには、証明書の詳細ページから [Actions] (アクション) を選択してから、[Activate] (有効化) を選択します。

正しい証明書を見つけ、アクティブであることを確認できたにもかかわらず、サンプルアプリケーションの実行にまだ問題がある場合は、次のステップで説明するように、そのポリシーを確認します。

the section called “モノのオブジェクトを作成する” の手順に従って、新しいモノと新しい証明書の作成を試みることもできます。新しいモノを作成する場合は、新しいモノの名前を付けて、新しい証明書ファイルをデバイスにダウンロードする必要があります。

証明書にアタッチされているポリシーを確認する

ポリシーは でのアクションを承認します AWS IoT。 AWS IoT への接続に使用される証明書にポリシーがない場合、または接続を許可するポリシーがない場合、証明書がアクティブであっても、接続は拒否されます。

証明書にアタッチされたポリシーを確認するには

1. 前の項目の説明に従って証明書を見つけ、その詳細のページを開きます。
2. 証明書の詳細ページの左側のメニューで、[Policies] (ポリシー) を選択して、証明書にアタッチされているポリシーを表示します。
3. 証明書にポリシーが添付されていない場合は、[Actions] (アクション) メニューを選択し、[Attach policy] (ポリシーの添付) を選択してポリシーを追加します。

   先ほど the section called “AWS IoT リソースの作成” で作成したポリシーを選択します。
4. ポリシーがアタッチされている場合は、ポリシータイルを選択してその詳細ページを開きます。

   詳細ページで、ポリシードキュメントを調べて、the section called “AWS IoT ポリシーを作成する”で作成したものと同じ情報が含まれていることを確認します。

コマンドラインをチェックする

ご利用のシステム用に正しいコマンドラインを使用していることを確認してください。Linux および macOS システムで使用されるコマンドは、Windows システムで使用されるコマンドとは異なることがよくあります。

エンドポイントアドレスを確認する

入力したコマンドを確認し、コマンドのエンドポイントアドレスを AWS IoT コンソールのアドレスと照らし合わせて再確認します。

証明書ファイルのファイル名を確認する

入力したコマンドのファイル名を、`certs` ディレクトリ内の証明書ファイルのファイル名と比較します。

一部のシステムでは、正しく機能するためにファイル名を引用符で囲む必要がある場合があります。

SDK のインストールを確認する

SDK のインストールが完了しており、正しいことを確認します。

懸念がある場合は、デバイスに SDK を再インストールします。ほとんどの場合、これは、 AWS IoT「Device SDK for ***SDK ##***をインストール」というタイトルのチュートリアルの セクションを見つけて、もう一度手順に従うことが問題です。

AWS IoT Device SDK for JavaScript を使用している場合は、サンプルアプリケーションをインストールしてから実行してください。SDK をインストールしても、サンプルアプリケーションは自動的にはインストールされません。サンプルアプリケーションは、SDK のインストール後に手動でインストールする必要があります。

MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する

このセクションでは、 AWS IoT コンソールで AWS IoT MQTT テストクライアントを使用して、 によって送受信される MQTT メッセージを監視する方法について説明します AWS IoT。このセクションで使用される例は、AWS IoT Core チュートリアルの開始方法 で使用される例に関連しています。ただし、例で使用されている *`topicName`* を、ご利用の IoT ソリューションで使用されている任意のトピック名またはトピックフィルターに置き換えることができます。

デバイスは、トピックによって識別される MQTT メッセージを発行して状態を通信し AWS IoT、MQTT メッセージ AWS IoT を発行してデバイスとアプリケーションに変更とイベントを通知します。MQTT クライアントを使用すると、これらのトピックをサブスクライブして、メッセージを発生時に確認できます。MQTT テストクライアントを使用して、 のサブスクライブしているデバイスやサービスに MQTT メッセージを発行することもできます AWS アカウント。

内容

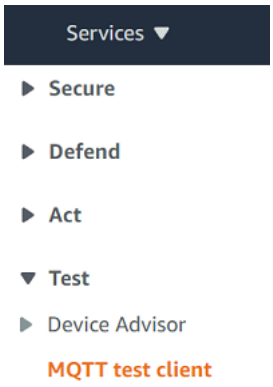
- MQTT クライアントで MQTT メッセージを表示する
- MQTT クライアントから MQTT メッセージを発行する
- MQTT クライアントで共有サブスクリプションをテストする

MQTT クライアントで MQTT メッセージを表示する

次の手順では、デバイスがメッセージを発行する特定の MQTT トピックをサブスクライブし、それらのメッセージを AWS IoT コンソール に表示する方法について説明します。

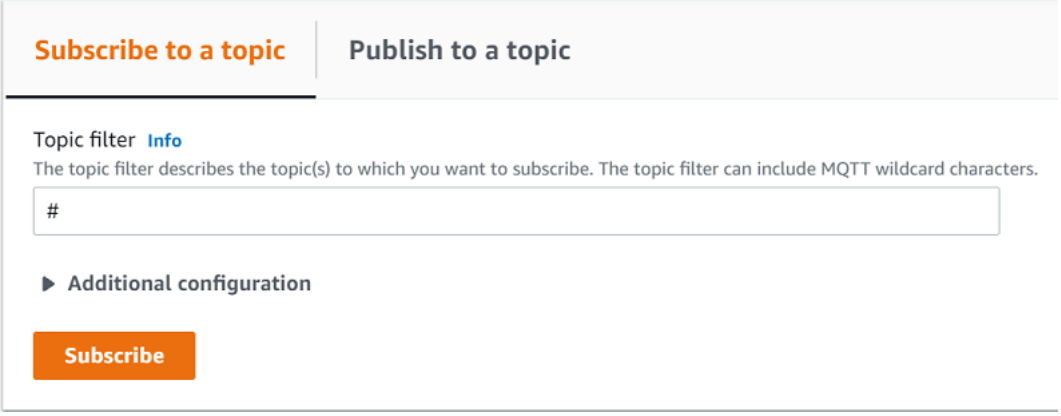
MQTT テストクライアントで MQTT メッセージを表示するには

1. AWS IoT コンソールの左側のメニューで、[Test] (テスト)、[MQTT test client] (MQTT テストクライアント) の順に選択します。

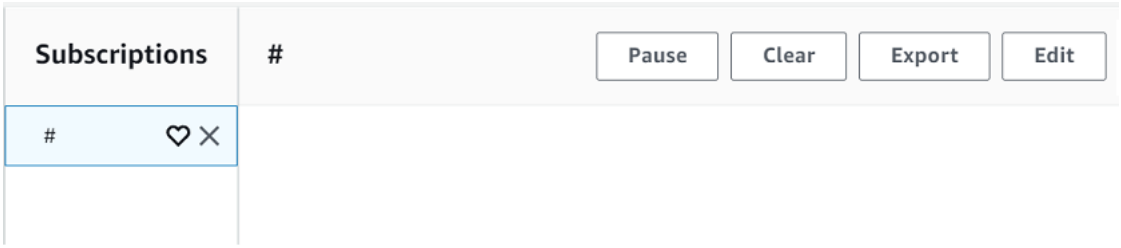


2. [Subscribe to a topic] (トピックにサブスクライブする) タブで、*topicName* を入力して、デバイスが発行する対象のトピックにサブスクライブします。開始方法のサンプルアプリケーションについては、# にサブスクライブします。これにより、すべてのメッセージトピックにサブスクライブします。

   開始方法の例を使用して続行し、[Subscribe to a topic] (トピックをサブスクライブする) タブの [Topic filter] (トピックフィルター) フィールドで# と入力し、[Subscribe (サブスクライブ)]を選択します。



トピックメッセージログページ # が開き、# が [Subscriptions] (サブスクリプション) リストに表示されます。で設定したデバイスがサンプルプログラムthe section called “デバイスを設定する”を実行している場合、# メッセージログ AWS IoT に に送信するメッセージが表示されます。メッセージログエントリは、サブスクライブされたトピックを含むメッセージが によって受信されると、発行セクションの下に表示されます AWS IoT。



3. # メッセージログページでは、トピックにメッセージを発行することもできますが、トピック名を指定する必要があります。# トピックに発行することはできません。

   サブスクライブしたトピックに発行されたメッセージは、受信されるとメッセージログに表示されます。最初に表示されるのは、最新のメッセージです。

MQTT メッセージのトラブルシューティング

ワイルドカードトピックフィルターを使用する

メッセージが期待どおりにメッセージログに表示されない場合は、トピック名フィルター の説明に従って、ワイルドカードトピックフィルターをサブスクライブしてみてください。MQTT マルチレベルワイルドカードトピックフィルターはハッシュまたはポンド記号 (#) であり、[Subscription topic] (サブスクリプショントピック) フィールドでトピックフィルターとして使用できます。

トピックフィルターにサブスクライブすると、メッセージブローカーによって受信されるすべてのトピックがサブスクライブされます。トピックフィルターパスの要素を # 複数レベルのワイルド

カード文字または「+」単一レベルのワイルドカード文字に置き換えることで、絞り込むことができます。

トピックフィルターでワイルドカードを使用する場合

- マルチレベルのワイルドカード文字は、トピックフィルターの最後の文字にする必要があります。
- トピックフィルターパスには、トピックレベルごとに 1 つの単一レベルのワイルドカード文字のみを使用できます。

以下に例を示します。

トピックのフィルター	次でメッセージを表示します
#	任意のトピック名
topic_1/#	topic_1/ で始まるトピック名
topic_1/level_2/#	topic_1/level_2/ で始まるトピック名
topic_1/+/level_3	topic_1/ で始まり、/level_3で終わり、その間に任意の値の 1 つの要素があるトピック名。

トピックフィルターの詳細については、トピック名フィルター を参照してください。

トピック名のエラーをチェックする

MQTT トピック名とトピックフィルターでは、大文字と小文字が区別されます。例えば、サブスクライブしたトピックである topic_1 ではなく Topic_1 (大文字の T) にデバイスがメッセージを発行している場合、そのメッセージは MQTT テストクライアントに表示されません。ただし、ワイルドカードトピックフィルターをサブスクライブすると、デバイスがメッセージを発行していることが表示され、想定したものとは異なるトピック名を使用していることがわかります。

MQTT クライアントから MQTT メッセージを発行する

MQTT トピックにメッセージを発行するには

1. MQTT テストクライアントページの [Publish to a topic] (トピックへの発行) タブの [Topic name] (トピック名) フィールドに、メッセージの *topicName* を入力します。この例では **my/topic** を使用します。

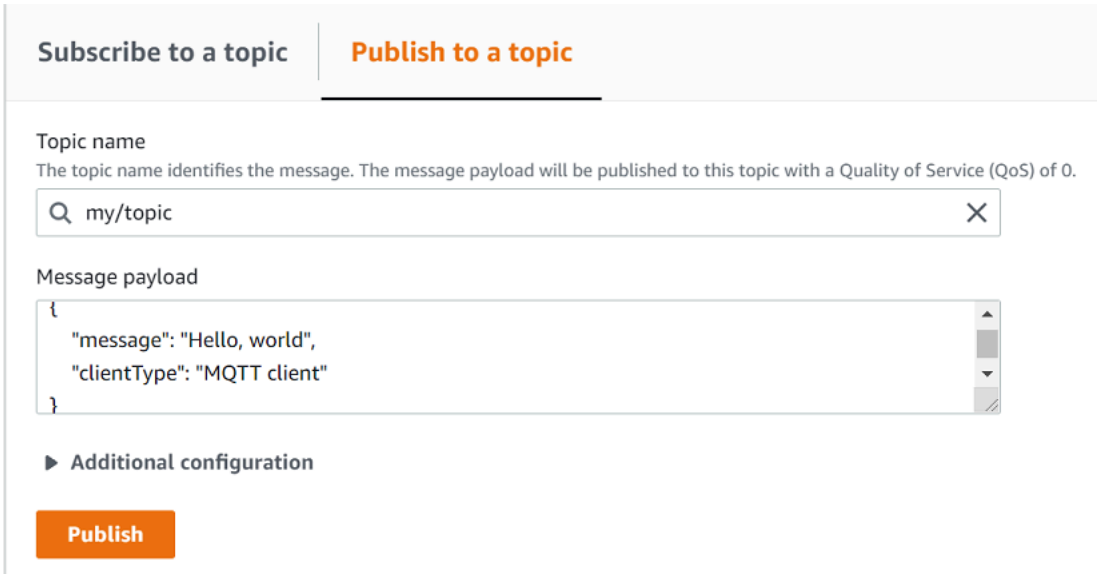
   > **Note**
   >
   > MQTT テストクライアントで使用する場合でも、システム実装で使用する場合でも、トピック名に個人を特定できる情報を使用しないでください。トピック名は、暗号化されていない通信およびレポートに表示できます。

2. メッセージペイロードウィンドウで、次の JSON を入力します。

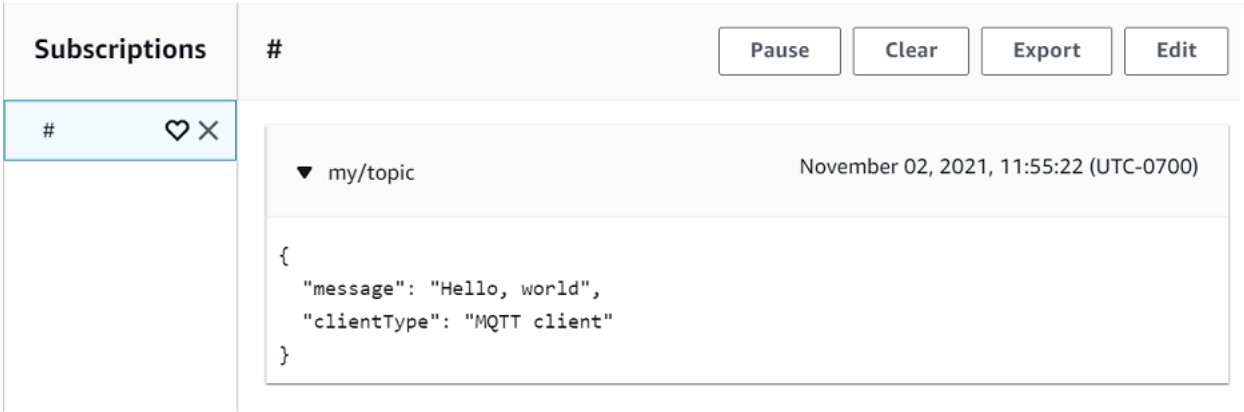
   ```
   {
       "message": "Hello, world",
       "clientType": "MQTT test client"
   }
   ```

3. [Publish] (発行) を選択して、メッセージを AWS IoTに発行します。

   > **Note**
   >
   > メッセージを発行する前に、my/topic トピックにサブスクライブしていることを確認してください。



4. [Subscriptions] (サブスクリプション) のリストで、[my/topic] を選びます。発行メッセージペイロードウィンドウの下の MQTT テストクライアントにメッセージが表示されます。



MQTT メッセージを他のトピックに発行するには、[Topic name] (トピック名) フィールドの *topicName* を変更し、[Publish] (発行) ボタンを選びます。

> **Important**
>
> 重複するトピック (例: probe1/temperature および probe1/#) で複数のサブスクリプションを作成する場合、両方のサブスクリプションに一致するトピックに発行された 1 つのメッセージが、重複するサブスクリプションごとに 1 回ずつ、複数回にわたって配信される可能性があります。

MQTT クライアントで共有サブスクリプションをテストする

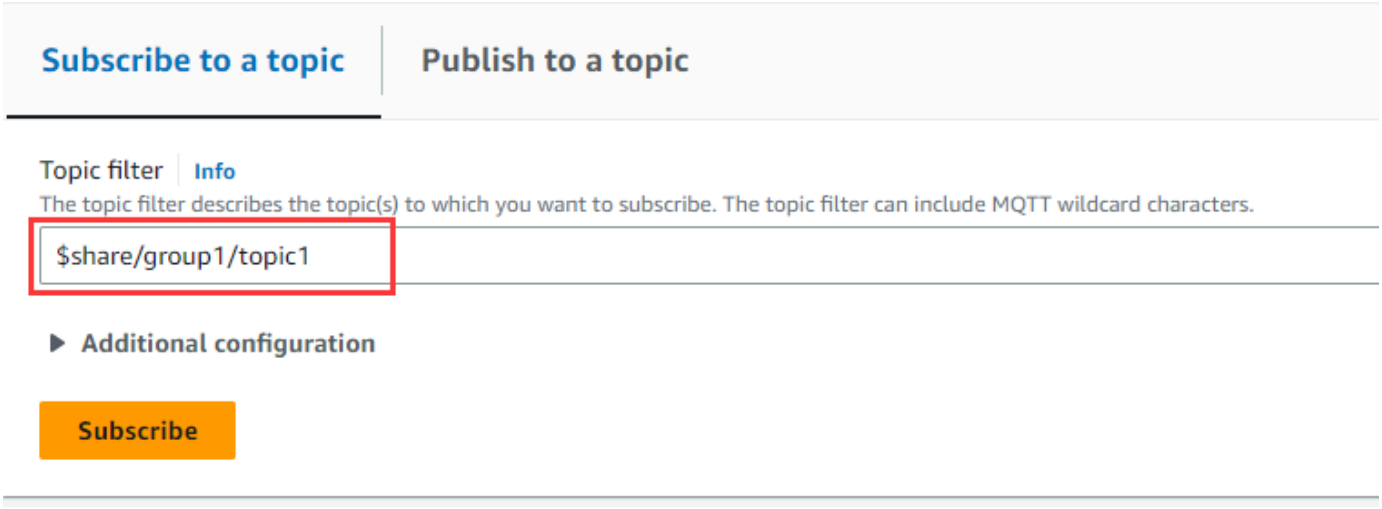
このセクションでは、 AWS IoT コンソールで AWS IoT MQTT クライアントを使用して、共有サブスクリプション AWS IoT を使用して送受信された MQTT メッセージを監視する方法について説明します。 では、複数のクライアントが、ランダムディストリビューションを使用してそのトピックに発行されたメッセージを受信するクライアントが 1 つだけであるトピックへのサブスクリプションを共有???できます。同じサブスクリプションを共有する複数の MQTT クライアント (この例では 2 つの MQTT クライアント) をシミュレートするには、複数のウェブブラウザからAWS IoT コンソールで AWS IoT MQTT クライアントを開きます。このセクションで使用されている例は、「AWS IoT Core チュートリアルの開始方法」で使用されている例とは関連がありません。詳細については、「共有サブスクリプション」を参照してください。

MQTT トピックのサブスクリプションを共有するには

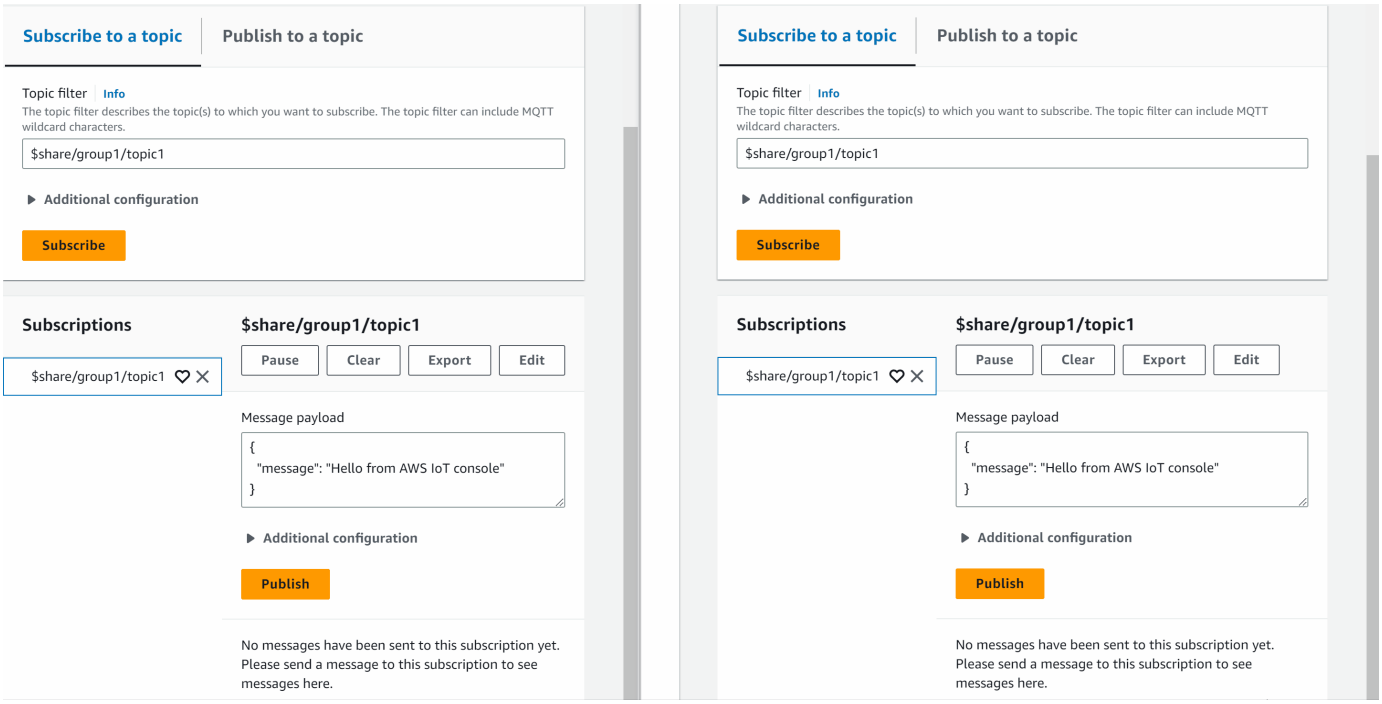
1. AWS IoT コンソールのナビゲーションペインで、[テスト] を選択し、次に [MQTT テストクライアント] を選択します。
2. [Subscribe to a topic] (トピックにサブスクライブする) タブで、*topicName* を入力して、デバイスが発行する対象のトピックにサブスクライブします。共有サブスクリプションを使用するには、以下のように共有サブスクリプションのトピックフィルターにサブスクライブします。

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

   サンプルのトピックフィルターは、メッセージトピック **topic1** にサブスクライブしている **$share/group1/topic1** にすることができます。



3. 別のウェブブラウザを開き、手順 1 と手順 2 を繰り返します。この方法では、同じサブスクリプション **$share/group1/topic1** を共有する 2 つの異なる MQTT クライアントをシミュレートしています。
4. 1 つの MQTT クライアントを選択し、[トピックに公開する] タブの [トピック名] フィールドに、メッセージの *topicName* を入力します。この例では **topic1** を使用します。メッセージを数回公開してみてください。両方の MQTT クライアントのサブスクリプションリストで、クライアントがランダム配信を使用してメッセージを受信していることを確認できるようになります。この例では、同じメッセージ「Hello from AWS IoT console」を 3 回公開します。左側の MQTT クライアントはメッセージを 2 回受信し、右側の MQTT クライアントはメッセージを 1 回受信しました。



AWS IoT チュートリアル

AWS IoT チュートリアルは、2 つの異なる目標をサポートするために 2 つの学習パスに分かれています。目標に最適なラーニングパスを選択してください。

- AWS IoT ソリューションのアイデアをテストまたは実証するためのproof-of-conceptを構築したい

  デバイスで AWS IoT Device Client を使用して一般的な IoT タスクとアプリケーションを実証するには、 the section called “AWS IoT Device Client を使用したデモの構築”ラーニングパスに従います。 AWS IoT Device Client には、最小限の開発でend-to-endのソリューションを実証するために独自のクラウドリソースを適用できるデバイスソフトウェアが用意されています。

  AWS IoT Device Client の詳細については、AWS IoT 「 Device Client」を参照してください。
- ソリューションをデプロイするための本番ソフトウェアを構築する方法を学びたい場合

  AWS IoT Device SDK を使用して特定の要件を満たす独自のソリューションソフトウェアを作成するには、 the section called “AWS IoT Device SDKsを使用したソリューションの構築”ラーニングパスに従います。

  使用可能な AWS IoT Device SDKs「」を参照してください???。 AWS SDKs「構築するツール AWS」を参照してください。

AWS IoT チュートリアルのラーニングパスオプション

- AWS IoT Device Client を使用したデモの構築
- AWS IoT Device SDKsを使用したソリューションの構築

AWS IoT Device Client を使用したデモの構築

このラーニングパスのチュートリアルでは、 AWS IoT Device Client を使用してデモンストレーションソフトウェアを開発する手順について説明します。 AWS IoT Device Client は、 IoT デバイスで実行されるソフトウェアを提供し、 が構築されている IoT ソリューションの側面をテストしてデモンストレーションします AWS IoT。

これらのチュートリアルの目的は、デバイスソフトウェアを開発する前に がソリューション AWS IoT をサポートしていることを確信できるように、探索と実験を容易にすることです。

これらのチュートリアルでは、次の内容を学習します。

- で IoT デバイスとして使用する Raspberry Pi を準備する方法 AWS IoT
- デバイスで AWS IoT Device Client を使用して AWS IoT 機能を示す方法

このラーニングパスでは、独自の Raspberry Pi に AWS IoT Device Client をインストールし、クラウドに AWS IoT リソースを作成して IoT ソリューションのアイデアを示します。このラーニングパスのチュートリアルでは、Raspberry Pi を使用して機能を実証しますが、他のデバイスへの適応に役立つ目標と手順について説明します。

AWS IoT Device Client でデモを構築するための前提条件

このセクションでは、このラーニングパスのチュートリアルを開始する前に、必要な内容について説明します。

このラーニングパスのチュートリアルを完了するには、以下が必要です。

- AWS アカウント

  これらのチュートリアルで使用する機能を使用するには AWS アカウント AWS IoT 、既存の を使用できますが、追加のロールまたはアクセス許可を追加する必要がある場合があります。

  新しい を作成する必要がある場合は AWS アカウント、「」を参照してくださいthe section called “セットアップ AWS アカウント”。
- Raspberry Pi または互換性のある IoT デバイス

  チュートリアルでは、Raspberry Pi を使用します。さまざまなフォームファクタで入手でき、どこにでもあり、比較的安価なデモンストレーションデバイスだからです。チュートリアルは、Raspberry Pi 3 Model B+、Raspberry Pi 4 Model B、および Ubuntu Server 20.04 LTS (HVM) で実行されている Amazon EC2 インスタンスでテストされています。を使用してコマンド AWS CLI を実行するには、最新バージョンの Raspberry Pi OS (Raspberry Pi OS (64 ビット) または OS Lite) を使用することをお勧めします。以前のバージョンの OS でも動作する可能性がありますが、テストはしていません。

> **Note**
>
> チュートリアルでは、各ステップの目標を説明して、これまで当社で試していない IoT ハードウェアへの適応に役立つようにしていますが、他のデバイスに適応させる方法を具体的に説明しているわけではありません。

- IoT デバイスのオペレーティングシステムに慣れていること

  これらのチュートリアルの手順では、Raspberry Pi でサポートされているコマンドラインインターフェイスからの基本的な Linux コマンドとオペレーションの使用に慣れていることを前提としています。これらのオペレーションに慣れていない場合は、チュートリアルを完了するのに時間がかかるかもしれません。

  これらのチュートリアルを完了するために望ましいのは、以下の操作方法を事前に理解しておくことです。

  - コンポーネントの組み立てと接続、必要な電源へのデバイスの接続、メモリカードの取り付けと取り外しなど、基本的なデバイス操作を安全に実行する。
  - システムソフトウェアとファイルをデバイスにアップロードおよびダウンロードする。お使いのデバイスが microSD カードなどのリムーバブルストレージデバイスを使用していない場合は、デバイスに接続し、システムソフトウェアとファイルをデバイスにアップロードおよびダウンロードする方法を知っておく必要があります。
  - デバイスを使用する予定のネットワークにデバイスを接続する。
  - SSH ターミナルまたは同様のプログラムを使用して、別のコンピュータからデバイスに接続する。
  - コマンドラインインターフェイスを使用して、デバイス上のファイルとディレクトリの作成、コピー、移動、名前の変更、およびアクセス許可の設定を行う。
  - デバイスに新しいプログラムをインストールする。
  - FTP や SCP などのツールを使用して、デバイスとの間でファイルを転送する。
- IoT ソリューションの開発およびテスト環境

  チュートリアルでは、必要なソフトウェアとハードウェアについて説明します。ただし、チュートリアルでは、オペレーションが明示的に説明されていない場合がありますが、実行できると想定しています。このようなハードウェアとオペレーションの例は次のとおりです。

  - ファイルをダウンロードして保存するローカルホストコンピュータ

Raspberry Pi の場合、これは通常、microSD メモリカードへの読み書きが可能なパーソナルコンピュータまたはラップトップです。ローカルホストコンピュータでは、次のことが必要です。

- インターネットに接続されている。
- AWS CLI がインストールされ、設定されている。
- AWS コンソールをサポートするウェブブラウザがある。

- ローカルホストコンピュータをデバイスに接続して通信し、コマンドを入力し、ファイルを転送する手段

  Raspberry Pi では、これはローカルホストコンピュータからの SSH と SCP を使用して行われることがよくあります。

- IoT デバイスに接続するモニターとキーボード

  これらは役立ちますが、チュートリアルを完了するために必須ではありません。

- ローカルホストコンピュータと IoT デバイスがインターネットに接続する手段

  これは、インターネットに接続されているルーターまたはゲートウェイへのケーブル接続またはワイヤレスネットワーク接続です。ローカルホストは Raspberry Pi にも接続できる必要があります。これには、同じローカルエリアネットワーク上に存在することが必要になる場合があります。チュートリアルでは、特定のデバイスまたはデバイス構成に対してこれを設定する方法を説明することはできませんが、この接続をテストする方法を説明します。

- ローカルエリアネットワークのルーターにアクセスして、接続されたデバイスを表示する

  このラーニングパスのチュートリアルを完了するには、IoT デバイスの IP アドレスを見つける必要があります。

  ローカルエリアネットワークでは、デバイスが接続しているネットワークルーターの admin インターフェイスにアクセスすることでこれを行うことができます。ルーター内のデバイスに固定 IP アドレスを割り当てることができれば、デバイスが再起動するたびに行う再接続が簡単になります。

  キーボードとモニターがデバイスに接続されている場合、ifconfig でデバイスの IP アドレスを表示できます。

  これらの方法が使えない場合は、再起動するたびにデバイスの IP アドレスを特定する方法を見つける必要があります。

必要な要素をすべて用意したら、the section called “IoT Device Client を使用する準備” に進みます。

このラーニングパスのチュートリアル

- チュートリアル: Device Client 用の AWS IoT デバイスの準備
- チュートリアル: AWS IoT Device Client のインストールと設定
- チュートリアル: AWS IoT Device Client との MQTT メッセージ通信のデモンストレーション
- チュートリアル: AWS IoT Device Client を使用したリモートアクション (ジョブ) のデモンストレーション
- チュートリアル: AWS IoT Device Client チュートリアルの実行後のクリーンアップ

チュートリアル: Device Client 用の AWS IoT デバイスの準備

このチュートリアルでは、このラーニングパスの後続のチュートリアルに備えるための Raspberry Pi の初期化を順を追って説明します。

このチュートリアルの目標は、デバイスのオペレーティングシステムの現在のバージョンをインストールし、開発環境のコンテキストでデバイスと通信できるようにすることです。

前提条件

このチュートリアルを開始する前に、「the section called “AWS IoT Device Client でデモを構築するための前提条件”」にリストされている項目が使用可能で、使用する準備が整っていることを確認してください。

このチュートリアルの完了には 90 分ほどかかります。

このチュートリアルでは、次の作業を行います。

- デバイスのオペレーティングシステムをインストールおよび更新します。
- チュートリアルの実行に必要な追加のソフトウェアをインストールして検証します。
- デバイスの接続をテストし、必要な証明書をインストールします。

このチュートリアルを完了すると、次のチュートリアルでは Device Client を使用するデモ用に AWS IoT デバイスを準備します。

このチュートリアルの手順

- デバイスのオペレーティングシステムをインストールおよび更新する
- デバイスに必要なソフトウェアをインストールして確認する
- デバイスをテストして Amazon CA 証明書を保存する

デバイスのオペレーティングシステムをインストールおよび更新する

このセクションの手順では、Raspberry Pi がシステムドライブに使用する microSD カードを初期化する方法について説明します。Raspberry Pi の microSD カードには、オペレーティングシステム (OS) ソフトウェアとそのアプリケーションファイルストレージ用のスペースが含まれています。Raspberry Pi を使用していない場合は、デバイスの指示に従ってデバイスのオペレーティングシステムソフトウェアのインストールおよび更新をします。

このセクションを完了すると、IoT デバイスを起動し、ローカルホストコンピュータのターミナルプログラムからそれに接続できるようになります。

必要な機器:

- ローカルでの開発およびテスト環境
- インターネットに接続できる Raspberry Pi またはお使いの IoT デバイス
- 少なくとも 8 GB の容量または OS および必要なソフトウェアに十分なストレージを備えた microSD メモリカード。

> **Note**
>
> これらの演習での microSD カードの選択では、必要な大きさでできるだけ小さいものを選択してください。
> 小さな SD カードの方が、バックアップと更新が高速になります。Raspberry Pi では、これらのチュートリアルに 8 GB を超える microSD カードは必要ありません。特定のアプリケーションにより多くのスペースが必要な場合は、このチュートリアルで保存するイメージファイルのサイズが小さくなると、選択したカードのサポートされているすべての領域を使用するように、大きなカードのファイルシステムのサイズを変更できます。

オプションの機器:

- Raspberry Pi に接続された USB キーボード
- モニターを Raspberry Pi に接続するための HDMI モニターとケーブル

このセクションの手順は次のとおりです。

- デバイスのオペレーティングシステムを microSD カードにロードする
- 新しいオペレーティングシステムで IoT デバイスを起動する
- ローカルホストコンピュータをデバイスに接続する

デバイスのオペレーティングシステムを microSD カードにロードする

この手順では、ローカルホストコンピュータを使用して、デバイスのオペレーティングシステムを microSD カードにロードします。

> **Note**
>
> デバイスがオペレーティングシステムにリムーバブルストレージメディアを使用していない場合は、そのデバイスの手順に従ってオペレーティングシステムをインストールし、the section called “IoT デバイスを起動する” に進みます。

オペレーティングシステムを Raspberry Pi にインストールするには

1. ローカルホストコンピュータで、使用する Raspberry Pi オペレーティングシステムイメージをダウンロードして解凍します。最新バージョンは以下から入手できます。https://www.raspberrypi.com/software/operating-systems/

   Raspberry Pi OS のバージョンを選択する

   このチュートリアルでは Raspberry Pi OS Lite バージョンを使用します。これは、このラーニングパスのこれらのチュートリアルをサポートする最小のバージョンだからです。このバージョンの Raspberry Pi OS はコマンドラインインターフェイスのみを備えており、グラフィカルユーザーインターフェイスはありません。グラフィカルユーザーインターフェイスを備えた最新の Raspberry Pi OS のバージョンも、これらのチュートリアルで動作します。ただし、このラーニングパスで説明されている手順では、コマンドラインインターフェイスのみを使用して Raspberry Pi 上でオペレーションを実行します。
2. microSD カードをローカルホストコンピュータに挿入します。
3. SD カードイメージングツールを使用して、解凍した OS イメージファイルを microSD カードに書き込みます。
4. Raspberry Pi OS イメージを microSD に書き込んだら、以下を行います。

a. コマンドラインウィンドウまたはファイルエクスプローラーウィンドウで microSD カードの BOOT パーティションを開きます。

b. microSD カードの BOOT パーティションのルートディレクトリに、ファイル拡張子もコンテンツもない `ssh` という名前の空のファイルを作成します。これにより、初めて起動したときに SSH 通信を有効にするように Raspberry Pi に指示します。

5. microSD カードを取り出して、ローカルホストコンピュータから安全に取り外します。

microSD カードは the section called “IoT デバイスを起動する” の準備ができました。

新しいオペレーティングシステムで IoT デバイスを起動する

この手順では、microSD カードをインストールし、ダウンロードしたオペレーティングシステムを使用して初めて Raspberry Pi を起動します。

新しいオペレーティングシステムで IoT デバイスを起動するには

1. デバイスの電源を切った状態で、前のステップ the section called “OS をロードする” の microSD カードを Raspberry Pi に挿入します。
2. デバイスを有線ネットワークに接続します。
3. これらのチュートリアルでは、SSH ターミナルを使用して、ローカルホストコンピュータから Raspberry Pi とやり取りします。

   デバイスと直接やり取りする場合は、次の方法も使用できます。

   a. ローカルホストコンピュータのターミナルウィンドウを Raspberry Pi に接続する前に、HDMI モニターを接続して Raspberry Pi のコンソールメッセージを確認します。

   b. Raspberry Pi と直接やりとりする場合は、USB キーボード接続します。
4. 電源を Raspberry Pi に接続し、初期化されるまで約 1 分待ちます。

   Raspberry Pi にモニターが接続されている場合は、そのモニターで起動プロセスを確認できます。
5. デバイスの IP アドレスを調べます。

   - HDMI モニターを Raspberry Pi に接続した場合、モニターに表示されるメッセージに IP アドレスが表示されます。
   - Raspberry Pi が接続されているルーターにアクセスできる場合は、ルーターの admin インターフェイスでそのアドレスを確認できます。

Raspberry Pi の IP アドレスを取得したら、the section called “ホストコンピュータを接続する” の準備が整います。

ローカルホストコンピュータをデバイスに接続する

この手順では、ローカルホストコンピュータ上のターミナルプログラムを使用して Raspberry Pi に接続し、デフォルトのパスワードを変更します。

ローカルホストコンピュータをデバイスに接続するには

1. ローカルホストコンピュータで、次の SSH ターミナルプログラムを開きます。

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

   > **Note**
   >
   > PuTTY は Windows に自動的にインストールされません。コンピュータにない場合は、ダウンロードおよびインストールが必要になる場合があります。

2. ターミナルプログラムを Raspberry Pi の IP アドレスに接続し、デフォルトの認証情報を使用してログインします。

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

3. Raspberry Pi にログインしたら、`pi` ユーザーのパスワードを変更します。

   ```
   passwd
   ```

   プロンプトに従って、パスワードを変更します。

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

ターミナルウィンドウで Raspberry Pi のコマンドラインプロンプトが表示され、パスワードを変更したら、the section called “必要なソフトウェアをインストールして検証する” に進む準備が整います。

デバイスに必要なソフトウェアをインストールして確認する

このセクションの手順は、前のセクションから続き、Raspberry Pi のオペレーティングシステムを最新の状態にし、次のセクションで AWS IoT Device Client の構築とインストールに使用するソフトウェアを Raspberry Pi にインストールします。

このセクションを完了すると、Raspberry Pi には、このラーニングパスのチュートリアルで必要なソフトウェアである最新のオペレーティングシステムが搭載され、お客様の場所に合わせて設定されます。

必要な機器:

- 前のセクションで準備したローカルでの開発およびテスト環境
- 前のセクションで使用した Raspberry Pi
- 前のセクションで準備した microSD メモリカード

> **Note**
>
> Raspberry Pi Model 3+ および Raspberry Pi Model 4 は、このラーニングパスで説明されているすべてのコマンドを実行できます。IoT デバイスがソフトウェアをコンパイルしたり、を実行できない場合は AWS Command Line Interface、必要なコンパイラをローカルホストコンピュータにインストールしてソフトウェアを構築し、IoT デバイスに転送する必要があります。デバイスにソフトウェアをインストールおよびビルドする方法の詳細については、デバイスのソフトウェアのドキュメントを参照してください。

このセクションの手順は次のとおりです。

- オペレーティングシステムソフトウェアを更新する
- 必要なアプリケーションおよびライブラリをインストールする
- (オプション) microSD カードイメージを保存する

オペレーティングシステムソフトウェアを更新する

この手順では、オペレーティングシステムソフトウェアを更新します。

Raspberry Pi でオペレーティングシステムソフトウェアをアップデートするには

ローカルホストコンピュータのターミナルウィンドウでこれらの手順を実行します。

1. Raspberry Pi のシステムソフトウェアを更新するには、次のコマンドを入力します。

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

2. Raspberry Pi のロケールとタイムゾーンの設定を更新します (オプション)。

   次のコマンドを入力して、デバイスのロケールとタイムゾーンの設定を更新します。

   ```
   sudo raspi-config
   ```

   a. デバイスのロケールを設定するには、次の手順を実行します。

      i. [Raspberry Pi Software Configuration Tool (raspi-config)] (Raspberry Pi ソフトウェア設定ツール (raspi-config)) 画面で、オプション [5] を選択します。

         5 Localisation Options Configure language and regional settings

         Tab キーを使用して [<Select>] (<選択>) に移動してから、space bar を押します。

      ii. ローカリゼーションオプションメニューで、オプション [L1] を選択します。

         L1 Locale Configure language and regional settings

         Tab キーを使用して [<Select>] (<選択>) に移動してから、space bar を押します。

      iii. ロケールのオプションの一覧で、矢印キーを使用してスクロールし、space bar を使用してインストールするものをマークすることで、Raspberry Pi にインストールするロケールを選択します。

         米国では、**en_US.UTF-8** を選択するのが適切です。

      iv. デバイスのロケールを選択したら、Tab キーを使用して [<OK>] を選択してから space bar を押して、[Configuring locales] (ロケールの設定) 確認ページを表示します。

b. デバイスのタイムゾーンを設定するには、次の手順を実行します。

i. [raspi-config] (raspi-config) 画面で、オプション [5] を選択します。

5 Localisation Options Configure language and regional settings

Tab キーを使用して [<Select>] (<選択>) に移動してから、space bar を押します。

ii. ローカリゼーションオプションメニューで、矢印キーを使用してオプション [L2] を次のように選択します。

L2 time zone Configure time zone

Tab キーを使用して [<Select>] (<選択>) に移動してから、space bar を押します。

iii. [Configuring tzdata] (tzdata の設定) メニューで、リストから自身の地域を選択します。

Tab キーを使用して [<OK>] に移動し、space bar を押します。

iv. 都市のリストで、矢印キーを使用して、自身のタイムゾーン内の都市を選択します。

タイムゾーンを設定するには、Tab キー を使用して [<OK>] に移動してから、space bar を押します。

c. 設定の更新が完了したら、Tab キーを使用して [<Finish>] (<完了>) に移動してから、space bar を押して [raspi-config] アプリケーションを閉じます。

3. Raspberry Pi を再起動するには、次のコマンドを入力します。

```
sudo shutdown -r 0
```

4. Raspberry Pi が再起動するのを待ちます。

5. Raspberry Pi が再起動したら、ローカルホストコンピュータのターミナルウィンドウを Raspberry Pi に再接続します。

Raspberry Pi システムソフトウェアが設定され、the section called “アプリケーションおよびライブラリをインストールする” に進む準備が整いました。

必要なアプリケーションおよびライブラリをインストールする

この手順では、以降のチュートリアルで使用するアプリケーションソフトウェアとライブラリをインストールします。

Raspberry Pi を使用している場合、または IoT デバイスで必要なソフトウェアをコンパイルできる場合は、ローカルホストコンピュータのターミナルウィンドウで次の手順を実行します。ローカルホストコンピュータ上で IoT デバイス用のソフトウェアをコンパイルする必要がある場合は、デバイス上でこれらの手順を実行する方法について、IoT デバイスのソフトウェアドキュメントを確認してください。

アプリケーションソフトウェアとライブラリを Raspberry Pi にインストールするには

1. 次のコマンドを入力して、アプリケーションソフトウェアとライブラリをインストールします。

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

2. 次のコマンドを入力して、正しいバージョンのソフトウェアがインストールされていることを確認します。

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

3. 次のバージョンのアプリケーションソフトウェアがインストールされていることを確認します。

   - gcc 9.3.0 以降
   - cmake 3.10.x 以降
   - OpenSSL 1.1.1 以降
   - git 2.20.1 以降

Raspberry Pi に必要なアプリケーションソフトウェアの許容バージョンがある場合、the section called “(オプション) microSD イメージを保存する” に進む準備が整っています。

(オプション) microSD カードイメージを保存する

このラーニングパスのチュートリアルの各所で、Raspberry Pi の microSD カードイメージのコピーをローカルホストコンピュータ上のファイルに保存する手順が言及されています。この手順は奨励されていますが、必須タスクではありません。提案されているところで microSD カードのイメージを保存することで、このラーニングパスのセーブポイントの前にある手順をスキップできます。これにより、何かを再試行する必要がある場合に時間を節約できます。microSD カードのイメージを定期的に保存しないと、その結果として、microSD カードが破損した場合、またはアプリケーションや

その設定を誤って構成した場合は、ラーニングパスのチュートリアルを最初からやり直す必要がある可能性があります。

この時点で、Raspberry Pi の microSD カードに更新された OS と基本的なアプリケーションソフトウェアがロードされています。この時点で、microSD カードの内容をファイルに保存することで、これまでの手順を完了するのにかかる時間を節約できます。デバイスの microSD イメージの現在のイメージがあれば、最初からソフトウェアをインストールして更新することなく、この時点からチュートリアルや手順を続行または再試行できます。

microSD カードのイメージをファイルに保存するには

1. 次のコマンドを入力して Raspberry Pi をシャットダウンします。

   ```
   sudo shutdown -h 0
   ```

2. Raspberry Pi が完全にシャットダウンしたら、電源を切ります。
3. Raspberry Pi から microSD カードを取り外します。
4. ローカルホストコンピュータで、次の操作を行います。

   a. microSD カードを挿入します。

   b. SD カードイメージングツールを使用して、microSD カードのイメージをファイルに保存します。

   c. microSD カードのイメージを保存したら、ローカルホストコンピュータからカードを取り出します。

5. Raspberry Pi の電源を切断した状態で、microSD カードを Raspberry Pi に挿入します。
6. Raspberry Pi の電源を入れます。
7. 約 1 分待ってからローカルホストコンピュータで、Raspberry Pi に接続されていたローカルホストコンピュータのターミナルウィンドウを再接続し、Raspberry Pi にログインします。

デバイスをテストして Amazon CA 証明書を保存する

このセクションの手順は、前のセクションから続き、 AWS Command Line Interface と接続の認証に使用される認証機関証明書をインストールします AWS IoT Core。

このセクションを完了すると、Raspberry Pi に AWS IoT Device Client をインストールするために必要なシステムソフトウェアがあり、インターネットへの接続が機能していることがわかります。

必要な機器:

- 前のセクションで準備したローカルでの開発およびテスト環境
- 前のセクションで使用した Raspberry Pi
- 前のセクションで準備した microSD メモリカード

このセクションの手順は次のとおりです。

- のインストール AWS Command Line Interface
- AWS アカウント 認証情報を設定する
- Amazon ルート CA 証明書をダウンロードする
- (オプション) microSD カードイメージを保存する

のインストール AWS Command Line Interface

この手順では、 を Raspberry Pi AWS CLI にインストールします。

Raspberry Pi を使用している場合、または IoT デバイスでソフトウェアをコンパイルできる場合は、ローカルホストコンピュータのターミナルウィンドウで次の手順を実行します。ローカルホストコンピュータで IoT デバイス用のソフトウェアをコンパイルする必要がある場合は、IoT デバイスのソフトウェアドキュメントで、必要なライブラリに関する情報を確認してください。

Raspberry Pi AWS CLI に をインストールするには

1. AWS CLIをダウンロードしてインストールするには、次のコマンドを実行します。

```
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. このコマンドを実行して をインストールします AWS CLI。このコマンドは完了までに、最大で 15 分かかります。

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

3. このコマンドを実行して、 の正しいバージョン AWS CLI がインストールされていることを確認します。

   ```
   aws --version
   ```

   のバージョンは 2.2 以降 AWS CLI である必要があります。

が現在のバージョン AWS CLI を表示した場合、 に進む準備が整いますthe section called “アカウントの認証情報を設定する”。

AWS アカウント 認証情報を設定する

この手順では、 AWS アカウント 認証情報を取得し、Raspberry Pi で使用するための認証情報を追加します。

デバイスに AWS アカウント 認証情報を追加するには

1. からアクセスキー ID とシークレットアクセスキーを取得 AWS アカウント して、デバイスで AWS CLI を認証します。

   AWS IAM を初めて使用する場合、https://aws.amazon.com/premiumsupport/knowledge-center/create-access-key/ では、 AWS コンソールで を実行し、デバイスで使用する IAM 認証情報を作成する AWS プロセスについて説明します。
2. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、デバイスの [Access Key ID] (アクセスキー ID) と [Secret Access Key] (シークレットアクセスキー) 認証情報を使用して、次の操作を行います。

   a. 次のコマンドを使用して、 AWS アプリの設定を実行します。

      ```
      aws configure
      ```

   b. プロンプトが表示されたら、認証情報と設定情報を入力します。

      ```
      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. このコマンドを実行して、 AWS アカウント および AWS IoT Core エンドポイントへのデバイスのアクセスをテストします。

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

この例のように、 AWS アカウント特定の AWS IoT データエンドポイントを返す必要があります。

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

AWS アカウント特定の AWS IoT データエンドポイントが表示された場合、Raspberry Pi には に進むための接続とアクセス許可がありますthe section called “Amazon ルート CA 証明書をダウンロードする”。

> **Important**
>
> これで AWS アカウント 、認証情報が Raspberry Pi の microSD カードに保存されます。これにより、 との今後のやり取りや、これらのチュートリアルで作成するソフトウェア AWS が容易になりますが、デフォルトでこのステップ後に作成した microSD カードイメージにも保存され、複製されます。
> AWS アカウント 認証情報のセキュリティを保護するために、microSD カードイメージを保存する前に、`aws configure`再度 を実行し、アクセスキー ID とシークレットアクセスキーにランダムな文字を入力して認証情報を消去して、 AWS アカウント 認証情報が侵害されないようにすることを検討してください。
> AWS アカウント 誤って認証情報を保存した場合は、IAM AWS コンソールで非アクティブ化できます。

Amazon ルート CA 証明書をダウンロードする

この手順では、Amazon ルート認証局 (CA) の証明書のコピーをダウンロードして保存します。この証明書をダウンロードすると、以降のチュートリアルで使用するために保存され、デバイスと AWS のサービスとの接続もテストされます。

Amazon ルート CA 証明書をダウンロードして保存するには

1. 次のコマンドを実行して、証明書用のディレクトリを作成します。

```
mkdir ~/certs
```

2. 次のコマンドを実行して、Amazon ルート CA 証明書をダウンロードします。

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

3. 次のコマンドを実行して、証明書ディレクトリとそのファイルへのアクセス権を設定します。

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

4. 次のコマンドを実行して、新しいディレクトリ内の CA 証明書ファイルを確認します。

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

次のようなエントリが表示されます。日付と時刻は異なりますが、ファイルサイズおよびその他の情報はここに示すものと同じである必要があります。

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

ファイルサイズが 1188 でない場合、curl コマンドパラメータを確認します。間違ったファイルをダウンロードした可能性があります。

(オプション) microSD カードイメージを保存する

この時点で、Raspberry Pi の microSD カードに更新された OS と基本的なアプリケーションソフトウェアがロードされています。

microSD カードのイメージをファイルに保存するには

1. ローカルホストコンピュータのターミナルウィンドウで、 AWS 認証情報をクリアします。

   a. 次のコマンドを使用して、 AWS アプリの設定を実行します。

   ```
   aws configure
   ```

b. プロンプトが表示されたら、認証情報を置き換えます。Enter を押すことによって、[Default region name] (デフォルトリージョン名) と [Default output format] (デフォルト出力形式) をそのままにしておくことができます。

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

2. 次のコマンドを入力して Raspberry Pi をシャットダウンします。

```
sudo shutdown -h 0
```

3. Raspberry Pi が完全にシャットダウンしたら、電源コネクタを取り外します。
4. デバイスから microSD カードを取り外します。
5. ローカルホストコンピュータで、次の操作を行います。

   a. microSD カードを挿入します。

   b. SD カードイメージングツールを使用して、microSD カードのイメージをファイルに保存します。

   c. microSD カードのイメージを保存したら、ローカルホストコンピュータからカードを取り出します。

6. Raspberry Pi の電源を切断した状態で、microSD カードを Raspberry Pi に挿入します。
7. デバイスに電源を入れます。
8. 約 1 分後、ローカルホストコンピュータで、ターミナルウィンドウセッションを再起動し、デバイスにログインします。

   まだ AWS アカウント 認証情報を再入力しないでください。

Raspberry Pi を再起動してログインしたら、the section called “IoT Device Client のインストールと設定” に進む準備が整います。

チュートリアル: AWS IoT Device Client のインストールと設定

このチュートリアルでは、 AWS IoT Device Client のインストールと設定、およびこのデモやその他のデモで使用する AWS IoT リソースの作成について説明します。

このチュートリアルを開始するには、以下を行います。

- 前のチュートリアルでローカルホストコンピュータと Raspberry Pi が使用できるようにしておく。

このチュートリアルの完了には最大 90 分かかる場合があります。

このトピックが終了したら、次の状態になります。

- IoT デバイスは、他の AWS IoT Device Client デモで使用できるようになります。
- IoT デバイスをプロビジョニングします AWS IoT Core。
- デバイスに AWS IoT Device Client をダウンロードしてインストールします。
- デバイスの microSD カードのイメージが保存され、以降のチュートリアルで使用できるようになっています。

必要な機器:

- 前のセクションで準備したローカルでの開発およびテスト環境
- 前のセクションで使用した Raspberry Pi
- 前のセクションで使用した Raspberry Pi の microSD メモリカード

このチュートリアルの手順

- AWS IoT Device Client をダウンロードして保存する
- で Raspberry Pi をプロビジョニングする AWS IoT
- 接続をテストするように AWS IoT Device Client を設定する

AWS IoT Device Client をダウンロードして保存する

このセクションの手順では、 AWS IoT Device Client をダウンロードしてコンパイルし、Raspberry Pi にインストールします。インストールをテストした後、Raspberry Pi の microSD カードのイメージを保存して、後でチュートリアルをもう一度試すときに使用できます。

このセクションの手順は次のとおりです。

- AWS IoT Device Client をダウンロードして構築する
- チュートリアルで使用するディレクトリを作成する

- (オプション) microSD カードイメージを保存する

AWS IoT Device Client をダウンロードして構築する

この手順では、Raspberry Pi に AWS IoT Device Client をインストールします。

Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次のコマンドを実行します。

Raspberry Pi に AWS IoT Device Client をインストールするには

1. これらのコマンドを入力して、Raspberry Pi で AWS IoT Device Client をダウンロードして構築します。

```
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. このコマンドを実行して AWS IoT Device Client を構築します。このコマンドは完了までに、最大で 15 分かかります。

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

   AWS IoT Device Client のコンパイルとして表示される警告メッセージは無視できます。

   これらのチュートリアルはgcc、2021 年 10 月 30 日のバージョンの Raspberry Pi OS (ブルズアイ) の 、バージョン (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110、2021 年 5 月 7 日のバージョンの Raspberry Pi OS (バスター) の gcc、バージョン (Raspbian 8.3.0-6+rpi1) 8.3.0 上に構築された AWS IoT Device Client でテストされています。

3. AWS IoT Device Client の構築が完了したら、このコマンドを実行してテストします。

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

AWS IoT Device Client のコマンドラインヘルプが表示された場合は、 AWS IoT Device Client が正常に構築され、使用できるようになります。

チュートリアルで使用するディレクトリを作成する

この手順では、このラーニングパスのチュートリアルで使用されるファイルを保存するために使用されるディレクトリを Raspberry Pi 上に作成します。

このラーニングパスのチュートリアルで使用されるディレクトリを作成するには、次の手順を実行します。

1. 次のコマンドを実行して、必要なディレクトリを作成します。

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

2. 次のコマンドを実行して、新しいディレクトリに対するアクセス許可を設定します。

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

これらのディレクトリを作成してアクセス許可を設定したら、the section called “(オプション) microSD カードイメージを保存する” に進みます。

(オプション) microSD カードイメージを保存する

この時点で、Raspberry Pi の microSD カードには、更新された OS、基本的なアプリケーションソフトウェア、および AWS IoT Device Client があります。

これらの演習とチュートリアルをもう一度試してみたい場合は、この手順で保存した microSD カードのイメージを新しい microSD カードに書き込み、前の手順をスキップして、the section called “Raspberry Pi をプロビジョニングする” からチュートリアルを続けることができます。

microSD カードのイメージをファイルに保存するには、次の手順を実行します。

Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次の操作を行います。

1. AWS アカウント 認証情報が保存されていないことを確認します。

   a. 次のコマンドを使用して、 AWS アプリの設定を実行します。

   ```
   aws configure
   ```

   b. 認証情報が保存されている場合 (プロンプトに表示されている場合)、以下に示すようにプロンプトが表示されたら、文字列 **XYXYXYXYX** を入力します。[Default region name] (デフォルトリージョン名) と [Default output format] (デフォルト出力形式) はブランクのままにしておきます。

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

2. 次のコマンドを入力して、Raspberry Pi をシャットダウンします。

   ```
   sudo shutdown -h 0
   ```

3. Raspberry Pi が完全にシャットダウンしたら、電源コネクタを取り外します。
4. デバイスから microSD カードを取り外します。
5. ローカルホストコンピュータで、次の操作を行います。

   a. microSD カードを挿入します。

   b. SD カードイメージングツールを使用して、microSD カードのイメージをファイルに保存します。

   c. microSD カードのイメージを保存したら、ローカルホストコンピュータからカードを取り出します。

この microSD は the section called “Raspberry Pi をプロビジョニングする” で引き続き使用できます。

で Raspberry Pi をプロビジョニングする AWS IoT

このセクションの手順は、 と AWS IoT Device Client AWS CLI がインストールされている保存された microSD イメージから開始し、Raspberry Pi をプロビジョニングする AWS IoT リソースとデバイス証明書を作成します AWS IoT。

Raspberry Pi に microSD カードをインストールします。

この手順では、必要なソフトウェアを Raspberry Pi にロードして設定した microSD カードをインストールし、このラーニングパスのチュートリアルを続行 AWS アカウント できるように を設定します。

このラーニングパスの演習とチュートリアルに必要なソフトウェアが格納されている the section called “(オプション) microSD カードイメージを保存する” で作成した microSD カードを使用します。

Raspberry Pi に microSD カードをインストールするには

1. Raspberry Pi の電源を切断した状態で、microSD カードを Raspberry Pi に挿入します。
2. Raspberry Pi の電源を入れます。
3. 約 1 分後、ローカルホストコンピュータで、ターミナルウィンドウセッションを再起動し、Raspberry Pi にログインします。
4. ローカルホストコンピュータのターミナルウィンドウで、[Access Key ID] (アクセスキー ID) と [Secret Access Key] (シークレットアクセスキー) を使用して、Raspberry Pi の認証情報に次の操作を行います。

   a. 次のコマンドを使用して、 AWS アプリの設定を実行します。

   ```
   aws configure
   ```

   b. プロンプトが表示されたら、 AWS アカウント 認証情報と設定情報を入力します。

   ```
   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
   ```

AWS アカウント 認証情報を復元したら、 に進む準備が整いますthe section called “でデバイスをプロビジョニングする AWS IoT Core”。

でデバイスをプロビジョニングする AWS IoT Core

このセクションの手順では、Raspberry Pi をプロビジョニングする AWS IoT リソースを作成します AWS IoT。これらのリソースを作成する際には、さまざまな情報を記録するように求められます。この情報は、次の手順の AWS IoT Device Client 設定で使用されます。

Raspberry Pi を操作するには AWS IoT、プロビジョニングする必要があります。プロビジョニングは、Raspberry Pi を IoT デバイスとしてサポートするために必要な AWS IoT リソースを作成および設定するプロセスです。

Raspberry Pi の電源を入れて再起動したら、ローカルホストコンピュータのターミナルウィンドウを Raspberry Pi に接続し、以下の手順を完了します。

このセクションの手順は次のとおりです。

- デバイス証明書ファイルを作成およびダウンロードする
- AWS IoT リソースの作成

デバイス証明書ファイルを作成およびダウンロードする

この手順では、このデモのデバイス証明書ファイルが作成されます。

Raspberry Pi のデバイス証明書ファイルを作成してダウンロードするには

1. ローカルホストコンピュータのターミナルウィンドウで、次のコマンドを入力して、デバイスのデバイス証明書ファイルを作成します。

```
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"
```

このコマンドでは次のようなレスポンスが返されます。後で使用できるように、*certificateArn* の値を記録しておきます。

```
{
    "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. 次のコマンドを入力して、証明書ディレクトリとそのファイルに対するアクセス許可を設定します。

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

3. 次のコマンドを実行して、証明書のディレクトリおよびファイルに対するアクセス許可を確認します。

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

コマンドの出力は、ファイルの日付と時刻が異なることを除いて、ここで表示されるものと同じである必要があります。

```
-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
```

この時点で、デバイス証明書ファイルが Raspberry Pi にインストールされており、the section called “AWS IoT リソースの作成” に進むことができます。

AWS IoT リソースの作成

この手順では、デバイスが AWS IoT の機能やサービスにアクセスするために必要なリソース AWS IoT を作成して、 でデバイスをプロビジョニングします。

でデバイスをプロビジョニングするには AWS IoT

1. ローカルホストコンピュータのターミナルウィンドウで、次のコマンドを入力して、 AWS アカウントのデバイスデータエンドポイントのアドレスを取得します。

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

これまでの手順で入力したコマンドでは次のようなレスポンスが返されます。後で使用できるように、*endpointAddress* の値を記録しておきます。

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

2. Raspberry Pi の AWS IoT モノのリソースを作成するには、このコマンドを入力します。

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

AWS IoT モノのリソースが作成された場合、コマンドはこのようなレスポンスを返します。

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

3. ターミナルウィンドウで、次の操作を行います。

   a. nano などのテキストエディタを開きます。

   b. この JSON ポリシードキュメントをコピーして、開いているテキストエディタに貼り付けます。

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

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

> **Note**
>
> このポリシードキュメントでは、すべてのリソースに対する接続、受信、発行、およびサブスクリプションを許可するという広範なアクセス許可が付与されます。通常、ポリシーは、特定のアクションを実行するアクセス許可を特定のリソースのみに付与します。ただし、最初のデバイス接続テストでは、このテスト中のアクセス問題の可能性を最小限に抑えるために、この過度に一般的で広範なアクセス許可を付与するポリシーが使用されます。以降のチュートリアルでは、より狭い範囲のポリシードキュメントを使用して、ポリシー設計のより良いプラクティスを示します。

c. テキストエディタのファイルを **~/policies/dev_cli_test_thing_policy.json** として保存します。

4. 前のステップのポリシードキュメントを使用して AWS IoT ポリシーを作成するには、このコマンドを実行します。

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

ポリシーが作成されると、コマンドは次のようなレスポンスを返します。

```
{
    "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. 次のコマンドを実行して、ポリシーをデバイス証明書にアタッチします。*certificateArn* を以前に保存した certificateArn の値に置き換えます。

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

成功した場合、このコマンドは何も返しません。

6. このコマンドを実行して、デバイス証明書を AWS IoT モノのリソースにアタッチします。*certificateArn* を以前に保存した certificateArn の値に置き換えます。

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

成功した場合、このコマンドは何も返しません。

でデバイスを正常にプロビジョニングしたら AWS IoT、 に進む準備が整いますthe section called “Device Client を設定して接続をテストする”。

接続をテストするように AWS IoT Device Client を設定する

このセクションの手順では、Raspberry Pi から MQTT メッセージを発行するように AWS IoT Device Client を設定します。

このセクションの手順は次のとおりです。

- 設定ファイルを作成する
- MQTT テストクライアントを開く
- AWS IoT Device Client の実行

設定ファイルを作成する

この手順では、 AWS IoT Device Client をテストするための設定ファイルを作成します。

AWS IoT Device Client をテストする設定ファイルを作成するには

- Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次の操作を行います。

a. 次のコマンドを入力して、設定ファイルのディレクトリを作成し、ディレクトリに対するアクセス許可を設定します。

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

b. nano などのテキストエディタを開きます。

c. この JSON ドキュメントをコピーして、開いているテキストエディタに貼り付けます。

```
{
  "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. ########の値を、「」で見つけ AWS アカウント た のデバイスデータエンドポイントに置き換えますthe section called “でデバイスをプロビジョニングする AWS IoT Core”。

e. テキストエディタのファイルを **~/dc-configs/dc-testconn-config.json** として保存します。

f. 次のコマンドを実行して、新しい設定ファイルでアクセス許可を設定します。

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

ファイルを保存すると、the section called “MQTT テストクライアントを開く” に進む準備が整います。

MQTT テストクライアントを開く

この手順では、 AWS IoT デバイスクライアントが実行時に発行する MQTT メッセージにサブスクライブするように、コンソールで MQTT テストクライアントを準備します。 AWS IoT

[MQTT test client] (MQTT テストクライアント) を準備してすべての MQTT メッセージをサブスクライブするには

1. ローカルホストコンピュータの AWS IoT コンソールで、[MQTT test client] (MQTT テストクライアント) を選択します。
2. [トピックへサブスクライブする] タブの [トピックのフィルター] に # (ハッシュ記号) を入力し、[サブスクライブ] をクリックして すべての MQTT トピックをサブスクライブします。

3. [Subscriptions] (サブスクリプション) ラベルの下に、「**#**」(単一の # 記号) が表示されることを確認します。

ウィンドウで [MQTT test client] (MQTT テストクライアント) を開いたままにして、the section called “AWS IoT Device Client の実行” に進みます。

AWS IoT Device Client の実行

この手順では、 AWS IoT Device Client を実行して、MQTT テストクライアントが受信して表示する単一の MQTT メッセージを発行します。

AWS IoT Device Client から MQTT メッセージを送信するには

1. この手順を実行する際に、Raspberry Pi に接続されているターミナルウィンドウと、[MQTT test client] (MQTT テストクライアント) のウィンドウが両方とも表示できることを確認します。
2. ターミナルウィンドウで、これらのコマンドを入力して、 で作成された設定ファイルを使用して AWS IoT Device Client を実行しますthe section called “設定ファイルを作成する”。

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

   ターミナルウィンドウに、 AWS IoT Device Client は情報メッセージと実行時に発生するエラーを表示します。

   ターミナルウィンドウにエラーが表示されない場合は、[MQTT test client] (MQTT テストクライアント) を確認します。
3. [MQTT test client] (MQTT テストクライアント) の [Subscriptions] (サブスクリプション) ウィンドウで、`test/dc/pubtopic` メッセージトピックに送信された「Hello World!」メッセージを確認します。
4. AWS IoT Device Client にエラーが表示されず、MQTT テストクライアントの`test/dc/pubtopic`メッセージに Hello World! が送信されたことがわかった場合、接続は成功しています。
5. ターミナルウィンドウで、 **^C** (Ctrl-C) と入力して AWS IoT Device Client を停止します。

AWS IoT Device Client が Raspberry Pi で正しく実行され、 と通信できることを実証したら AWS IoT、 に進むことができますthe section called “MQTT を使用してデバイスクライアントと通信する”。

チュートリアル: AWS IoT Device Client との MQTT メッセージ通信のデモンストレーション

このチュートリアルでは、 AWS IoT Device Client が IoT ソリューションで一般的に使用される MQTT メッセージをサブスクライブして発行する方法を示します。

このチュートリアルを開始するには、以下を行います。

- ローカルホストコンピュータと Raspberry Pi は、前のセクションで使用したのと同じ設定にします。

  AWS IoT Device Client のインストール後に microSD カードイメージを保存した場合は、Raspberry Pi でそのイメージに microSD カードを使用できます。
- このデモを以前に実行したことがある場合は、リソースエラー???が重複しないように、「」を参照して、以前の実行で AWS IoT 作成したすべてのリソースを削除します。

このチュートリアルの完了には 45 分ほどかかります。

このトピックが終了したら、次の状態になります。

- IoT デバイスが から MQTT メッセージをサブスクライブ AWS IoT し、MQTT メッセージをパブリッシュする方法はさまざまです AWS IoT。

必要な機器:

- 前のセクションで準備したローカルでの開発およびテスト環境
- 前のセクションで使用した Raspberry Pi
- 前のセクションで使用した Raspberry Pi の microSD メモリカード

このチュートリアルの手順

- Raspberry Pi を準備して MQTT メッセージ通信のデモンストレーションをする
- AWS IoT Device Client を使用したメッセージの発行のデモンストレーション
- AWS IoT Device Client を使用したメッセージのサブスクライブのデモンストレーション

Raspberry Pi を準備して MQTT メッセージ通信のデモンストレーションをする

この手順では、 AWS IoT と Raspberry Pi にリソースを作成し、 AWS IoT Device Client を使用した MQTT メッセージ通信を示します。

このセクションの手順は次のとおりです。

- MQTT 通信をデモンストレーションするための証明書ファイルを作成する
- MQTT 通信をデモンストレーションするためにデバイスをプロビジョニングする
- MQTT 通信を示すように AWS IoT Device Client 設定ファイルと MQTT テストクライアントを設定する

MQTT 通信をデモンストレーションするための証明書ファイルを作成する

この手順では、このデモのデバイス証明書ファイルが作成されます。

Raspberry Pi のデバイス証明書ファイルを作成してダウンロードするには

1. ローカルホストコンピュータのターミナルウィンドウで、次のコマンドを入力して、デバイスのデバイス証明書ファイルを作成します。

   ```
   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"
   ```

   このコマンドでは次のようなレスポンスが返されます。後で使用するために *certificateArn* の値を保存します。

   ```
   {
   "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. 次のコマンドを入力して、証明書ディレクトリとそのファイルに対するアクセス許可を設定します。

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

3. 次のコマンドを実行して、証明書のディレクトリおよびファイルに対するアクセス許可を確認します。

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

コマンドの出力は、ファイルの日付と時刻が異なることを除いて、ここで表示されるものと同じである必要があります。

```
-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. 次のコマンドを入力して、ログファイルのディレクトリを作成します。

```
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 通信をデモンストレーションするためにデバイスをプロビジョニングする

このセクションでは、 で Raspberry Pi をプロビジョニングする AWS IoT リソースを作成します AWS IoT。

AWS IoTでデバイスをプロビジョニングするには:

1. ローカルホストコンピュータのターミナルウィンドウで、次のコマンドを入力して、 AWS アカウントのデバイスデータエンドポイントのアドレスを取得します。

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

   エンドポイントの値は、前のチュートリアルでこのコマンドを実行してから変更されていません。ここでコマンドを再度実行すると、このチュートリアルで使用する設定ファイルにデータエンドポイントの値を簡単に検索して貼り付けることができるようになります。

   これまでの手順で入力したコマンドでは次のようなレスポンスが返されます。後で使用できるように、*endpointAddress* の値を記録しておきます。

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

2. このコマンドを入力して、Raspberry Pi の新しい AWS IoT モノのリソースを作成します。

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

   AWS IoT モノのリソースはクラウド内のデバイスの仮想表現であるため、 で複数のモノのリソースを作成して AWS IoT 、さまざまな目的に使用できます。これらすべてを同じ物理 IoT デバイスで使用して、デバイスのさまざまな側面を表すことができます。

   このチュートリアルでは、Raspberry Pi を表すために一度に 1 つのモノのリソースのみを使用します。このように、これらのチュートリアルでは、デモの AWS IoT リソースを作成した後、それぞれに特別に作成したリソースを使用してデモに戻って繰り返すことができるように、さまざまなデモを表します。

   AWS IoT モノのリソースが作成された場合、コマンドはこのようなレスポンスを返します。

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

3. ターミナルウィンドウで、次の操作を行います。

a. nano などのテキストエディタを開きます。

b. この JSON ドキュメントをコピーして、開いているテキストエディタに貼り付けます。

```
{
    "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. エディタで、ポリシードキュメントの各Resourceセクションで、*us-west-2:57EXAMPLE833* を 、コロン文字 (:) AWS リージョン、および 12 桁の AWS アカウント 数字に置き換えます。

d. テキストエディタのファイルを **~/policies/pubsub_test_thing_policy.json** として保存します。

4. 前のステップのポリシードキュメントを使用して AWS IoT ポリシーを作成するには、このコマンドを実行します。

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

ポリシーが作成されると、コマンドは次のようなレスポンスを返します。

```
{
                                    "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. 次のコマンドを実行して、ポリシーをデバイス証明書にアタッチします。*certificateArn* を、このセクション内で前に保存した certificateArn 値に置き換えます。

```
aws iot attach-policy \
--policy-name "PubSubTestThingPolicy" \
```

```
--target "certificateArn"
```

成功した場合、このコマンドは何も返しません。

6. 次のコマンドを実行して、デバイス証明書を AWS IoT のモノのリソースにアタッチします。*certificateArn* を、このセクション内で前に保存した certificateArn 値に置き換えます。

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

成功した場合、このコマンドは何も返しません。

でデバイスを正常にプロビジョニングしたら AWS IoT、 に進む準備が整いますthe section called “Device Client 設定ファイルと MQTT クライアントを設定する”。

MQTT 通信を示すように AWS IoT Device Client 設定ファイルと MQTT テストクライアントを設定する

この手順では、 AWS IoT Device Client をテストするための設定ファイルを作成します。

AWS IoT Device Client をテストする設定ファイルを作成するには

1. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次の操作を行います。

   a. nano などのテキストエディタを開きます。

   b. この JSON ドキュメントをコピーして、開いているテキストエディタに貼り付けます。

```
{
  "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. *#######*値を、「」で見つけ AWS アカウント た のデバイスデータエンドポイントに置き換えますthe section called “でデバイスをプロビジョニングする AWS IoT Core”。

d. テキストエディタのファイルを **~/dc-configs/dc-pubsub-config.json** として保存します。

e. 次のコマンドを実行して、新しい設定ファイルでアクセス許可を設定します。

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

2. [MQTT test client] (MQTT テストクライアント) を準備して、すべての MQTT メッセージをサブスクライブするには、次の手順を行います。

a. ローカルホストコンピュータの AWS IoT コンソールで、[MQTT test client] (MQTT テストクライアント) を選択します。

b. [Subscribe to a topic] (トピックをサブスクライブする) タブの [Topic filter] (トピックのフィルター) に「**#**」(単一の # 記号) を入力し、[Subscribe] (サブスクライブ) をクリックします。

c. [Subscriptions] (サブスクリプション) ラベルの下に、「**#**」(単一の # 記号) が表示されることを確認します。

このチュートリアルを進めている間は、ウィンドウで [MQTT test client] (MQTT テストクライアント) を開いたままにします。

ファイルを保存し、[MQTT test client] (MQTT テストクライアント) を設定したら、the section called “IoT Device Client でメッセージを発行する” に進む準備が整いました。

AWS IoT Device Client を使用したメッセージの発行のデモンストレーション

このセクションの手順では、 AWS IoT Device Client がデフォルトおよびカスタム MQTT メッセージを送信する方法を示します。

前のステップでこれらの演習用に作成したポリシー内のポリシーステートメントでは、以下のアクションを実行するアクセス許可が Raspberry Pi に付与されます。

- **`iot:Connect`**

  Device Client を実行PubSubTestThingしている Raspberry Pi である という名前の AWS IoT クライアントに接続を許可します。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Connect"
  ],
```

```
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing"
      ]
    }
```

- **`iot:Publish`**

  Raspberry Pi に MQTT トピック `test/dc/pubtopic` でメッセージを発行するアクセス許可を付与します。

  ```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic"
      ]
    }
  ```

  `iot:Publish` アクションで、リソース配列にリストされている MQTT トピックに発行するアクセス許可が付与されます。これらのメッセージの内容は、ポリシーステートメントによって制御されません。

AWS IoT Device Client を使用してデフォルトのメッセージを発行する

この手順では、 AWS IoT Device Client を実行して、MQTT テストクライアントが受信して表示する単一のデフォルトの MQTT メッセージを発行します。

AWS IoT Device Client からデフォルトの MQTT メッセージを送信するには

1. この手順を実行する際に、Raspberry Pi に接続されている、ローカルホストコンピュータ上のターミナルウィンドウと、[MQTT test client] (MQTT テストクライアント) のウィンドウが両方とも表示できることを確認します。
2. ターミナルウィンドウで、これらのコマンドを入力して、 で作成された設定ファイルを使用して AWS IoT Device Client を実行しますthe section called “設定ファイルを作成する”。

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

ターミナルウィンドウに、 AWS IoT Device Client は情報メッセージと実行時に発生するエラーを表示します。

ターミナルウィンドウにエラーが表示されない場合は、[MQTT test client] (MQTT テストクライアント) を確認します。

3. [MQTT テストクライアント] の [サブスクリプション] ウィンドウで、`test/dc/pubtopic` メッセージトピックに送信された「Hello World!」メッセージを確認します。
4. AWS IoT Device Client にエラーが表示されず、MQTT テストクライアントの`test/dc/pubtopic`メッセージに Hello World! が送信されたことがわかった場合、接続は成功しています。
5. ターミナルウィンドウで、 **^C** (Ctrl-C) と入力して AWS IoT Device Client を停止します。

AWS IoT Device Client がデフォルトの MQTT メッセージを発行したことを証明したら、 に進むことができますthe section called “カスタム MQTT メッセージを発行する”。

AWS IoT Device Client を使用してカスタムメッセージを発行する

このセクションの手順では、カスタム MQTT メッセージを作成し、 AWS IoT Device Client を実行して、カスタム MQTT メッセージを 1 回発行させて、[MQTT test client] (MQTT テストクライアント) が受信して表示するようにします。

AWS IoT Device Client のカスタム MQTT メッセージを作成する

Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次の手順を実行します。

AWS IoT Device Client が発行するカスタムメッセージを作成するには

1. ターミナルウィンドウで、nano などのテキストエディタを開きます。
2. テキストエディタに次の JSON ドキュメントをコピーして貼り付けます。これは、 AWS IoT Device Client が発行する MQTT メッセージペイロードです。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
```

```
    "bearing": 255
  }
}
```

3. テキストエディタの内容を **~/messages/sample-ws-message.json** として保存します。
4. 次のコマンドを入力して、作成したメッセージファイルのアクセス許可を設定します。

```
chmod 600 ~/messages/*
```

Device AWS IoT Client がカスタムメッセージを送信するために使用する設定ファイルを作成するには

1. ターミナルウィンドウで、 などのテキストエディタでnano、既存の AWS IoT Device Client 設定ファイル を開きます**~/dc-configs/dc-pubsub-config.json**。
2. samples オブジェクトが次のようになるように編集します。このファイルの他の部分を変更する必要はありません。

```
  "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. テキストエディタの内容を **~/dc-configs/dc-pubsub-custom-config.json** として保存します。
4. 次のコマンドを実行して、新しい設定ファイルでアクセス許可を設定します。

```
chmod 644 ~/dc-configs/dc-pubsub-custom-config.json
```

AWS IoT Device Client を使用してカスタム MQTT メッセージを発行する

この変更は、MQTT メッセージペイロードの内容のみに影響するので、現在のポリシーは引き続き機能します。ただし、MQTT トピック (~/dc-configs/dc-pubsub-custom-config.json の publish-topic 値で定義されたもの) が変更されている場合、Raspberry Pi が新しい MQTT トピックに発行できるように iot::Publish ポリシーステートメントも変更する必要があります。

AWS IoT Device Client から MQTT メッセージを送信するには

1. この手順を実行する際に、ターミナルウィンドウと、[MQTT test client] (MQTT テストクライアント) のウィンドウが両方とも表示できることを確認します。また、[MQTT test client] (MQTT テストクライアント) のサブスクリプションが、引き続き[#] トピックフィルターであることを確認してください。そうでない場合は、[#] トピックフィルターのサブスクリプションに戻します。
2. ターミナルウィンドウで次のコマンドを入力して、the section called “設定ファイルを作成する” で作成した設定ファイルを使用する AWS IoT Device Client を実行します。

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

   ターミナルウィンドウに、 AWS IoT Device Client は情報メッセージと実行時に発生するエラーを表示します。

   ターミナルウィンドウにエラーが表示されない場合は、[MQTT test client] (MQTT テストクライアント) を確認します。
3. [MQTT test client] (MQTT テストクライアント) の [Subscriptions] (サブスクリプション) ウィンドウで、`test/dc/pubtopic` メッセージトピックに送信されたカスタムメッセージペイロードを確認します。
4. AWS IoT Device Client にエラーが表示されず、MQTT テストクライアントの`test/dc/pubtopic`メッセージに発行したカスタムメッセージペイロードが表示された場合は、カスタムメッセージが正常に発行されました。
5. ターミナルウィンドウで、 **^C** (Ctrl-C) と入力して AWS IoT Device Client を停止します。

AWS IoT Device Client がカスタムメッセージペイロードを発行したことを証明したら、 に進むことができますthe section called “IoT Device Client でメッセージをサブスクライブする”。

AWS IoT Device Client を使用したメッセージのサブスクライブのデモンストレーション

このセクションでは、次の 2 種類のメッセージサブスクリプションについて説明します。

- 単一トピックのサブスクリプション
- ワイルドカードトピックのサブスクリプション

これらの演習用に作成したポリシー内のポリシーステートメントでは、以下のアクションを実行するアクセス許可が Raspberry Pi に付与されます。

- **`iot:Receive`**

  `Resource` オブジェクト内の名前と一致する MQTT トピックを受信するアクセス許可を AWS IoT Device Client に付与します。

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Receive"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic"
    ]
  }
  ```

- **`iot:Subscribe`**

  `Resource` オブジェクト内の という名前のものに一致する MQTT トピックフィルターをサブスクライブするアクセス許可を AWS IoT Device Client に付与します。

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Subscribe"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic"
    ]
  }
  ```

単一の MQTT メッセージトピックをサブスクライブする

この手順では、 AWS IoT Device Client が MQTT メッセージをサブスクライブしてログに記録する方法を示します。

Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、**`~/dc-configs/dc-pubsub-custom-config.json`** の内容を一覧表示するか、ファイルをテキスト工

ディタで開き、その内容を確認します。samples オブジェクトを見つけます。これは次のように表示されています。

```
"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 の値は AWS IoT Device Client が実行時にサブスクライブする MQTT トピックであることに注意します。 AWS IoT Device Client は、このサブスクリプションから受信したメッセージペイロードを subscribe-file値内の という名前のファイルに書き込みます。

AWS IoT Device Client から MQTT メッセージトピックをサブスクライブするには

1. この手順を実行する際に、ターミナルウィンドウと、[MQTT test client] (MQTT テストクライアント) のウィンドウが両方とも表示できることを確認します。また、[MQTT test client] (MQTT テストクライアント) のサブスクリプションが、引き続き[#] トピックフィルターであることを確認してください。そうでない場合は、[#] トピックフィルターのサブスクリプションに戻します。
2. ターミナルウィンドウで、これらのコマンドを入力して、 で作成された設定ファイルを使用して AWS IoT Device Client を実行しますthe section called “設定ファイルを作成する”。

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

   ターミナルウィンドウに、 AWS IoT Device Client は情報メッセージと実行時に発生するエラーを表示します。

   ターミナルウィンドウにエラーが表示されない場合は、 AWS IoT で操作を続行します。
3. AWS IoT コンソールの MQTT テストクライアントで、トピックへの発行タブを選択します。
4. [Topic name] (トピック名) に **test/dc/subtopic** と入力します。
5. [Message payload] (メッセージペイロード) で、メッセージの内容を確認します。
6. MQTT メッセージを発行するには、[Publish] (発行) をクリックします。
7. ターミナルウィンドウで、 AWS IoT Device Client から受信した次のようなエントリを確認します。

```
2021-11-10T16:02:20.890Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 45 bytes
```

8. メッセージが受信されたことを示すメッセージ受信エントリが表示されたら、 **^C** (Ctrl-C) を入力して AWS IoT Device Client を停止します。
9. 次のコマンドを入力して、メッセージログファイルの末尾を表示し、[MQTT test client] (MQTT テストクライアント) から発行したメッセージを表示します。

```
tail ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

ログファイル内のメッセージを表示することで、MQTT テストクライアントから公開したメッセージを AWS IoT Device Client が受信したことを実証しました。

ワイルドカード文字を使用して複数の MQTT メッセージトピックをサブスクライブする

これらの手順は、 AWS IoT Device Client がワイルドカード文字を使用して MQTT メッセージをサブスクライブおよびログ記録する方法を示しています。これを行うには、次の操作を行います。

1. AWS IoT Device Client が MQTT トピックのサブスクライブに使用するトピックフィルターを更新します。
2. デバイスが使用するポリシーを更新して、新しいサブスクリプションができるようにします。
3. AWS IoT Device Client を実行し、MQTT テストコンソールからメッセージを発行します。

ワイルドカード MQTT トピックフィルターを使用して複数の MQTT メッセージトピックをサブスクライブする設定ファイルを作成するには

1. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、**~/dc-configs/dc-pubsub-custom-config.json** を開いて編集し、`samples` オブジェクトを見つけます。
2. テキストエディタで、`samples` オブジェクトを見つけ、`subscribe-topic` の値を次のように更新します。

```
  "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"
```

新しい subscribe-topic の値は、最後に MQTT ワイルドカード文字が付いた MQTT トピックフィルターです。これは、test/dc/ で始まるすべての MQTT トピックへのサブスクリプション示しています。 AWS IoT Device Client は、このサブスクリプションから受信したメッセージペイロードを の という名前のファイルに書き込みますsubscribe-file。

3. 変更された設定ファイルを **~/dc-configs/dc-pubsub-wild-config.json** として保存し、エディタを終了します。

複数の MQTT メッセージトピックをサブスクライブして受信できるように Raspberry Pi が使用するポリシーを変更するには

1. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、任意のテキストエディタで **~/policies/pubsub_test_thing_policy.json** を編集するために開き、ファイル内のポリシーステートメント iot::Subscribe と iot::Receive を見つけます。
2. iot::Subscribe ポリシーステートメントで、Resource オブジェクトの文字列を更新して、次のように subtopic を * に置き換えます。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Subscribe"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*"
  ]
}
```

> **Note**
>
> MQTT トピックフィルターワイルドカード文字「+」(プラス記号) と「#」(# 記号) です。末尾に # が付いたサブスクリプションリクエストは、# 文字の前にある文字列 (例えば、この場合「test/dc/」) で始まるすべてのトピックをサブスクライブします。ただし、このサブスクリプションを承認するポリシーステートメントのリソース値には、トピックフィルター ARN では # (# 記号) の代わりに * (アスタリスク) を使用する

必要があります。これは、ポリシープロセッサが MQTT が使用するとは別のワイルドカード文字を使用するためです。
ポリシーでトピックおよびトピックフィルターにワイルドカード文字を使用する方法の詳細については、「MQTT および AWS IoT Core ポリシーでのワイルドカード文字の使用」を参照してください。

3. iot::Receive ポリシーステートメントで、Resource オブジェクトの文字列を更新して、次のように subtopic を * に置き換えます。

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*"
      ]
    }
```

4. 更新されたポリシードキュメントを **~/policies/pubsub_wild_test_thing_policy.json** として保存し、エディタを終了します。
5. 次のコマンドを入力して、このチュートリアルのポリシーを更新して、新しいリソース定義を使用します。

```
aws iot create-policy-version \
--set-as-default \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_wild_test_thing_policy.json"
```

コマンドが成功すると、次のようなレスポンスが返されます。policyVersionId が 2 になったことに注意してください。これがこのポリシーの 2 番目のバージョンであることを示しています。

ポリシーを正常に更新した場合は、次の手順に進むことができます。

```
{
    "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
}
```

ポリシーバージョンが多すぎて新しいバージョンを保存できないというエラーが表示された場合は、次のコマンドを入力して、ポリシーの現在のバージョンを一覧表示します。このコマンドが返すリストを確認して、削除できるポリシーバージョンを探します。

```
aws iot list-policy-versions --policy-name "PubSubTestThingPolicy"
```

不要になったバージョンを削除するには、次のコマンドを入力します。デフォルトのポリシーバージョンを削除することはできません。デフォルトのポリシーバージョンは、`isDefaultVersion` の値が `true` であるものです。

```
aws iot delete-policy-version \
--policy-name "PubSubTestThingPolicy" \
--policy-version-id policyId
```

ポリシーバージョンを削除したら、このステップを再試行してください。

更新された設定ファイルとポリシーを使用すると、 AWS IoT Device Client でワイルドカードサブスクリプションをデモンストレーションする準備が整います。

AWS IoT Device Client が複数の MQTT メッセージトピックをサブスクライブおよび受信する方法を示すには

1. [MQTT test client] (MQTT テストクライアント) で、サブスクリプションを確認します。[MQTT test client] (MQTT テストクライアント) で「**#**」トピックフィルターがサブスクライブされていれば、次のステップに進みます。そうでない場合は、[MQTT テストクライアント] の [トピック

へのサブスクライブ] タブで、[トピックフィルター] に #(ハッシュ記号文字) を入力し、[サブスクライブ] を選択してサブスクライブします。

2. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次のコマンドを入力して AWS IoT Device Client を起動します。

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

3. ローカルホストコンピュータのターミナルウィンドウで AWS IoT Device Client の出力を見ながら、MQTT テストクライアントに戻ります。[Publish to a topic] (トピックに公開する) タブで、[Topic name] (トピック名) に「**test/dc/subtopic**」と入力し、[Publish] (発行) をクリックします。

4. ターミナルウィンドウで、次のようなメッセージを検索して、メッセージが受信されたことを確認します。

```
2021-11-10T16:34:20.101Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 76 bytes
```

5. ローカルホストコンピュータのターミナルウィンドウで AWS IoT Device Client の出力を見ながら、MQTT テストクライアントに戻ります。[Publish to a topic] (トピックに公開する) タブで、[Topic name] (トピック名) に「**test/dc/subtopic2**」と入力し、[Publish] (発行) をクリックします。

6. ターミナルウィンドウで、次のようなメッセージを検索して、メッセージが受信されたことを確認します。

```
2021-11-10T16:34:32.078Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 77 bytes
```

7. 両方のメッセージが受信されたことを確認するメッセージが表示されたら、 **^C** (Ctrl-C) と入力して AWS IoT Device Client を停止します。

8. 次のコマンドを入力して、メッセージログファイルの末尾を表示し、[MQTT test client] (MQTT テストクライアント) から発行したメッセージを表示します。

```
tail -n 20 ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

> **Note**
>
> ログファイルには、メッセージペイロードのみが含まれます。メッセージトピックは、受信したメッセージログファイルに記録されません。
> 受信したログに AWS IoT Device Client によって発行されたメッセージが表示される場合もあります。これは、ワイルドカードトピックフィルターにそのメッセージトピックが含まれており、発行されたメッセージがサブスクライバーに送信される前にメッセージブローカーがサブスクリプションリクエストを処理することがあるためです。

ログファイルのエントリは、メッセージが受信されたことを示しています。この手順は、他のトピック名を使用して繰り返すことができます。`test/dc/` で始まるトピック名を持つメッセージはすべて受信されてログ記録されます。他の文字列で始まるトピック名のメッセージは無視されます。

AWS IoT Device Client が MQTT メッセージを発行およびサブスクライブする方法を示したら、「」に進みますチュートリアル: AWS IoT Device Client を使用したリモートアクション (ジョブ) のデモンストレーション。

チュートリアル: AWS IoT Device Client を使用したリモートアクション (ジョブ) のデモンストレーション

これらのチュートリアルでは、ジョブを設定して Raspberry Pi にデプロイして、IoT デバイスにリモートオペレーションを送信する方法をデモンストレーションします。

このチュートリアルを開始するには、以下を行います。

- ローカルホストコンピュータ Raspberry Pi を前のセクションで使用したのと同様の設定してください。
- 前のセクションのチュートリアルを完了していない場合は、 に AWS IoT Device Client をインストールした後に保存したイメージを持つ microSD カードで Raspberry Pi を使用して、このチュートリアルを試すことができます(オプション) microSD カードイメージを保存する。
- このデモを以前に実行したことがある場合は、リソースエラー???が重複しないように、「」を参照して、以前の実行で AWS IoT 作成したすべてのリソースを削除します。

このチュートリアルの完了には 45 分ほどかかります。

このトピックが終了したら、次の状態になります。

- IoT デバイスが AWS IoT Core を使用して、 によって管理されるリモートオペレーションを実行するさまざまな方法を説明します AWS IoT 。

必要な機器:

- 前のセクションでテストしたローカルの開発およびテスト環境
- 前のセクションでテストした Raspberry Pi
- 前のセクションでテストした Raspberry Pi の microSD メモリカード

このチュートリアルの手順

- ジョブを実行するために Raspberry Pi を準備する
- AWS IoT Device Client AWS IoT を使用して でジョブを作成して実行する

ジョブを実行するために Raspberry Pi を準備する

このセクションの手順では、 AWS IoT Device Client を使用して Raspberry Pi がジョブを実行するように準備する方法について説明します。

> **Note**
>
> これらの手順はデバイス固有です。このセクションの手順を複数のデバイスで同時に実行する場合、各デバイスには独自のポリシーと、一意のデバイス固有の証明書およびモノ名が必要です。各デバイスに固有のリソースを割り当てるには、手順の説明に従ってデバイス固有の要素を変更しながら、デバイスごとに 1 回ずつこの手順を実行します。

このチュートリアルの手順

- Raspberry Pi をプロビジョニングしてジョブをデモンストレーションする
- ジョブエージェントを実行するように AWS IoT Device Client を設定する

Raspberry Pi をプロビジョニングしてジョブをデモンストレーションする

このセクションの手順では、 AWS IoT リソースとデバイス証明書を作成して AWS IoT 、 で Raspberry Pi をプロビジョニングします。

トピック

- デバイス証明書ファイルを作成してダウンロードし、 AWS IoT ジョブをデモンストレーションする
- AWS IoT ジョブをデモンストレーションする AWS IoT リソースを作成する

デバイス証明書ファイルを作成してダウンロードし、 AWS IoT ジョブをデモンストレーションする

この手順では、このデモのデバイス証明書ファイルが作成されます。

複数のデバイスを準備する場合は、この手順を各デバイスで実行する必要があります。

Raspberry Pi のデバイス証明書ファイルを作成してダウンロードするには、次の手順を実行します。

Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次のコマンドを入力します。

1. 次のコマンドを入力して、デバイスのデバイス証明書ファイルを作成します。

```
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"
```

このコマンドでは次のようなレスポンスが返されます。後で使用するために *certificateArn* の値を保存します。

```
{
"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. 次のコマンドを入力して、証明書ディレクトリとそのファイルに対するアクセス許可を設定します。

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

3. 次のコマンドを実行して、証明書のディレクトリおよびファイルに対するアクセス許可を確認します。

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

コマンドの出力は、ファイルの日付と時刻が異なることを除いて、ここで表示されるものと同じである必要があります。

```
-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
```

デバイス証明書ファイルを Raspberry Pi にダウンロードしたら、the section called “ジョブに Raspberry Pi をプロビジョニングする” に進む準備が整います。

AWS IoT ジョブをデモンストレーションする AWS IoT リソースを作成する

このデバイスの AWS IoT リソースを作成します。

複数のデバイスを準備する場合は、この手順をデバイスごとに実行する必要があります。

AWS IoTでデバイスをプロビジョニングするには:

Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次の操作を行います。

1. 次のコマンドを入力して、 AWS アカウントのデバイスデータエンドポイントのアドレスを取得します。

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

最後にこのコマンドを実行してからエンドポイントの値は変更されていません。ここでコマンドをここで再度実行すると、このチュートリアルで使用する設定ファイルにデータエンドポイントの値を簡単に検索して貼り付けることができます。

describe-endpoint コマンドでは次のようなレスポンスが返されます。後で使用できるように、*endpointAddress* の値を記録しておきます。

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

2. *uniqueThingName* をデバイスの一意の名前に置き換えます。このチュートリアルを複数のデバイスで実行する場合は、各デバイスに独自の名前を付けます。例えば、**TestDevice01**、**TestDevice02** などです。

   このコマンドを入力して、Raspberry Pi の新しい AWS IoT モノのリソースを作成します。

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

   AWS IoT モノのリソースはクラウド内のデバイスの仮想表現であるため、 で複数のモノのリソースを作成して AWS IoT 、さまざまな目的に使用できます。これらすべてを同じ物理 IoT デバイスで使用して、デバイスのさまざまな側面を表すことができます。

   > **Note**
   >
   > 複数のデバイスに対してポリシーを確保する場合は、静的なモノの名前である *uniqueThingName* の代わりに `${iot:Thing.ThingName}` を使用できます。

   これらのチュートリアルでは、デバイスごとに一度に 1 つのモノのリソースしか使用しません。このように、これらのチュートリアルでは、デモの AWS IoT リソースを作成した後、それぞれに特別に作成したリソースを使用してデモに戻って繰り返すことができるように、さまざまなデモを表します。

AWS IoT モノのリソースが作成された場合、コマンドはこのようなレスポンスを返します。このデバイスで実行するジョブを作成するときに、後で使用するために *thingArn* の値を記録します。

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

3. ターミナルウィンドウで、次の操作を行います。

   a. nano などのテキストエディタを開きます。

   b. この JSON ドキュメントをコピーして、開いているテキストエディタに貼り付けます。

```
{
    "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. エディタでは、すべてのポリシーステートメントのResourceセクションで、*us-west-2:57EXAMPLE833* を、自身の AWS リージョン、コロン文字 (:)、12 桁の AWS アカウント 番号に置き換えます。

d. エディタで、すべてのポリシーステートメントの *uniqueThingName* を、このリソースに付けたモノ名と置き換えます。

e. テキストエディタのファイルを **~/policies/jobs_test_thing_policy.json** として保存します。

複数のデバイスに対してこの手順を実行する場合は、各デバイスでこのファイル名にファイルを保存します。

4. *uniqueThingName* をデバイスのモノの名前に置き換え、このコマンドを実行して、そのデバイスに合わせた AWS IoT ポリシーを作成します。

```
aws iot create-policy \
--policy-name "JobTestPolicyFor uniqueThingName" \
--policy-document "file://~/policies/jobs_test_thing_policy.json"
```

ポリシーが作成されると、コマンドは次のようなレスポンスを返します。

```
{
    "policyName": "JobTestPolicyFor uniqueThingName",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
JobTestPolicyFor uniqueThingName",
    "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* をデバイスのモノ名と置き換え、*certificateArn* をこのセクションで前に保存したこのデバイスの certificateArn の値に置き換えてから、次のコマンドを実行して、ポリシーをデバイス証明書にアタッチします。

```
aws iot attach-policy \
--policy-name "JobTestPolicyFor<uniqueThingName>" \
--target "<certificateArn>"
```

成功した場合、このコマンドは何も返しません。

6. *uniqueThingName* をデバイスのモノの名前に置き換え、 をこのセクションの前半で保存したcertificateArn値*certificateArn*に置き換え、このコマンドを実行してデバイス証明書を AWS IoT モノリソースにアタッチします。

```
aws iot attach-thing-principal \
--thing-name "<uniqueThingName>" \
--principal "<certificateArn>"
```

成功した場合、このコマンドは何も返しません。

Raspberry Pi を正常にプロビジョニングしたら、引き続きテストに含まれる別の Raspberry Pi についてこのセクションを繰り返します。すべてのデバイスがプロビジョニングされている場合は、the section called “Device Client を設定してジョブを実行する” に進みます。

ジョブエージェントを実行するように AWS IoT Device Client を設定する

この手順では、 AWS IoT Device Client がジョブエージェントを実行するための設定ファイルを作成します。

注: 複数のデバイスを準備する場合は、この手順を各デバイスで実行する必要があります。

AWS IoT Device Client をテストする設定ファイルを作成するには:

1. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、次の操作を行います。

   a. nano などのテキストエディタを開きます。

   b. この JSON ドキュメントをコピーして、開いているテキストエディタに貼り付けます。

```
{
  "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. *#######*値を、「」で見つけ AWS アカウント た のデバイスデータエンドポイント値に置き換えますthe section called “でデバイスをプロビジョニングする AWS IoT Core”。

d. *uniqueThingName* をこのデバイスに使用したモノ名に置き換えます。

e. テキストエディタのファイルを **~/dc-configs/dc-jobs-config.json** として保存します。

2. 次のコマンドを実行して、新しい設定ファイルのファイルアクセス許可を設定します。

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

[MQTT test client] (MQTT テストクライアント) はこのテストには使用しません。デバイスはジョブ関連の MQTT メッセージを と交換しますが AWS IoT、ジョブの進行状況メッセージはジョブを実行しているデバイスとのみ交換されます。ジョブの進行状況メッセージはジョブを実行しているデバイスとのみ交換されるため、 AWS IoT コンソールなどの別のデバイスからサブスクライブすることはできません。

設定ファイルを保存すると、the section called “ジョブを作成して実行する” に進む準備が整います。

AWS IoT Device Client AWS IoT を使用して でジョブを作成して実行する

このセクションの手順では、ジョブドキュメントと AWS IoT ジョブリソースを作成します。ジョブリソースを作成すると、 AWS IoT はジョブドキュメントを、ジョブエージェントがデバイスまたはクライアントにジョブドキュメントを適用する指定されたジョブターゲットに送信します。

このセクションの手順

- IoT ジョブのジョブドキュメントを作成して保存する
- 1 つの IoT デバイスの AWS IoT でジョブを実行する

IoT ジョブのジョブドキュメントを作成して保存する

この手順では、 ジョブリソースに含めるシンプルな AWS IoT ジョブドキュメントを作成します。このジョブドキュメントでは「Hello world!」と ジョブターゲットに表示されます。

ジョブドキュメントを作成して保存するには、次の手順に従います。

1. ジョブドキュメントを保存する Amazon S3 バケットを選択します。これに使用する既存の Amazon S3 バケットがない場合は、バケットを作成する必要があります。Amazon S3 バケットを作成する方法については、「Amazon S3 の開始方法」のトピックを参照してください。
2. このジョブのジョブドキュメントを作成して保存します。

   a. ローカルホストコンピュータで、テキストエディタを開きます。

   b. 次のテキストをコピーしてエディタに貼り付けます。

   ```
   {
       "operation": "echo",
       "args": ["Hello world!"]
   }
   ```

   c. ローカルホストコンピュータで、エディタの内容を **hello-world-job.json** という名前のファイルに保存します。

   d. ファイルが正しく保存されたことを確認します。一部のテキストエディタはテキストファイルを保存するときに自動的に `.txt` をファイル名に追加します。エディタが `.txt` をファイル名に追加した場合、先に進む前にファイル名を修正してください。

3. *path_to_file* を、**hello-world-job.json** が現在のディレクトリにない場合はそこへのパスに置き換え、*s3_bucket_name* を、選択したバケットへの Amazon S3 バケットパスに置き換えてから、次のコマンドを実行して、ジョブドキュメントを Amazon S3 バケットに配置します。

   ```
   aws s3api put-object \
   --key hello-world-job.json \
   --body path_to_file/hello-world-job.json --bucket s3_bucket_name
   ```

   Amazon S3 に保存したジョブドキュメントを特定するジョブドキュメント URL は、次の URL の *s3_bucket_name* と *AWS_region* を置き換えたものになります。後で *document_path* として使用するために、結果の URL を記録します。

   ```
   https://s3_bucket_name.s3.AWS_Region.amazonaws.com/hello-world-job.json
   ```

> **Note**
>
> AWS セキュリティにより、ブラウザを使用するなどして AWS アカウント、 の外部でこの URL を開くことができなくなります。URL は、デフォルトで ファイルにアクセスできる AWS IoT ジョブエンジンによって使用されます。本番稼働環境では、 AWS IoT サービスに、Amazon S3 に保存されているジョブドキュメントにアクセスするためのアクセス許可があることを確認する必要があります。

ジョブドキュメントの URL を保存したら、the section called “単一のデバイスのジョブを実行する”に進みます。

1 つの IoT デバイスの AWS IoT でジョブを実行する

このセクションの手順では、Raspberry Pi で AWS IoT Device Client を起動して、デバイスでジョブエージェントを実行してジョブの実行を待ちます。また、 にジョブリソースを作成し AWS IoT、ジョブを に送信して IoT デバイス上で実行します。

> **Note**
>
> この手順では、1 つのデバイスでのみジョブを実行します。

Raspberry Pi でジョブエージェントを開始するには、次の手順に従います。

1. Raspberry Pi に接続されているローカルホストコンピュータのターミナルウィンドウで、このコマンドを実行して AWS IoT Device Client を起動します。

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

2. ターミナルウィンドウで、 AWS IoT Device Client と がこれらのメッセージを表示することを確認します。

```
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. ターミナルウィンドウで、次のメッセージが表示されたら、次の手順に進み、ジョブリソースを作成します。これは、リストの最後のエントリではない可能性があることに注意してください。

```
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

AWS IoT ジョブリソースを作成するには

1. ローカルホストコンピュータで、次の操作を行います。

   a. *job_document_url* を the section called “ジョブドキュメントを作成して保存する” のジョブドキュメント URL に置き換えます。

b. *thing_arn* をデバイス用に作成したモノのリソースの ARN に置き換えてから、次のコマンドを実行します。

```
aws iot create-job \
--job-id hello-world-job-1 \
--document-source "job_document_url" \
--targets "thing_arn" \
--target-selection SNAPSHOT
```

成功すると、コマンドは次のような結果を返します。

```
{
  "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-job-1",
  "jobId": "hello-world-job-1"
}
```

2. ターミナルウィンドウでは、このような AWS IoT Device Client からの出力が表示されます。

```
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 Client が実行中でジョブを待っている間、`job-id`値を変更してステップ 1 create-jobから を再実行することで、別のジョブを送信できます。

ジョブの実行が完了したら、ターミナルウィンドウで ^C (control-C) と入力して AWS IoT Device Client を停止します。

チュートリアル: AWS IoT Device Client チュートリアルの実行後のクリーンアップ

このチュートリアルの手順では、このラーニングパスのチュートリアルを完了するまでに作成したファイルとリソースを削除する手順を説明します。

このチュートリアルの手順

- ステップ 1: Device Client でデモを構築した後で AWS IoT デバイスをクリーンアップする
- ステップ 2: AWS IoT Device Client でデモを構築 AWS アカウント した後で をクリーンアップする

ステップ 1: Device Client でデモを構築した後で AWS IoT デバイスをクリーンアップする

このチュートリアルでは、このラーニングパスでデモを作成した後に microSD カードをクリーンアップする方法の 2 つのオプションについて説明します。必要なセキュリティレベルを提供するオプションを選択します。

デバイスの microSD カードをクリーニングしても、作成した AWS IoT リソースは削除されないことに注意してください。デバイスの microSD カードをクリーンアップした後に AWS IoT リソースをクリーンアップするには、「」のチュートリアルを確認してくださいthe section called “ AWS IoT Device Client でデモを構築した後のクリーンアップ”。

オプション 1: microSD カードを書き換えてクリーンアップする

このラーニングパスのチュートリアルを完了した後、microSD カードをクリーニングする最も簡単で徹底的な方法は、デバイスの最初の準備時に作成して保存しておいたイメージファイルで microSD カードを上書きすることです。

この手順では、ローカルホストコンピュータを使用して、保存された microSD カードイメージを microSD カードに書き込みます。

> **Note**
>
> デバイスがオペレーティングシステムにリムーバブルストレージメディアを使用していない場合は、そのデバイスの手順を参照してください。

microSD カードに新しいイメージを書き込むには

1. ローカルホストコンピュータで、microSD カードに書き込む保存した microSD カードイメージを見つけます。
2. microSD カードをローカルホストコンピュータに挿入します。
3. SD カードイメージングツールを使用して、選択したイメージファイルを microSD カードに書き込みます。

4. Raspberry Pi OS イメージを microSD カードに書き込んだ後、microSD カードを取り出し、ローカルホストコンピュータから安全に取り外します。

microSD カードは使用可能になっています。

オプション 2: ユーザーディレクトリを削除してクリーンアップする

チュートリアルを完了した後、microSD カードのイメージを書き換えずに microSD カードをクリーニングするには、ユーザーディレクトリを個別に削除します。この方法では、システムファイルがインストールされている場合、そのファイルは削除されないため、保存されたイメージで microSD カードを書き換える方法ほど徹底的ではありません。

必要性に対して、ユーザーディレクトリが十分に削除されるならば、この手順に従うことができます。

このラーニングパスのユーザーディレクトリをデバイスから削除するには

1. 次のコマンドを実行して、デバイスに接続されたターミナルウィンドウで、このラーニングパスで作成されたユーザーディレクトリ、サブディレクトリ、およびそのすべてのファイルを削除します。

> **Note**
>
> これらのディレクトリとファイルを削除すると、チュートリアルを再度完了しないとデモを実行できなくなります。

```
rm -Rf ~/dc-configs
rm -Rf ~/policies
rm -Rf ~/messages
rm -Rf ~/certs
rm -Rf ~/.aws-iot-device-client
```

2. 次のコマンドを実行して、デバイスに接続されているターミナルウィンドウで、アプリケーションのソースディレクトリとファイルを削除します。

> **Note**
>
> これらのコマンドはプログラムをアンインストールしません。プログラムのビルドとインストールに使用されたソースファイルのみを削除します。これらのファイルを削除すると、 AWS CLI と AWS IoT Device Client が機能しない場合があります。

```
rm -Rf ~/aws-cli
rm -Rf ~/aws
rm -Rf ~/aws-iot-device-client
```

ステップ 2: AWS IoT Device Client でデモを構築 AWS アカウント した後で をクリーンアップする

これらの手順は、このラーニングパスのチュートリアルを完了する際に作成した AWS リソースを特定して削除するのに役立ちます。

AWS IoT リソースをクリーンアップする

この手順は、このラーニングパスのチュートリアルを完了する際に作成した AWS IoT リソースを特定して削除するのに役立ちます。

AWS IoT このラーニングパスで作成された リソース

チュートリアル	モノのリソース	ポリシーリソース
the section called “IoT Device Client のインストールと設定”	DevCliTestThing	DevCliTestThingPolicy
the section called “MQTT を使用してデバイスクライアントと通信する”	PubSubTestThing	PubSubTestThingPolicy
the section called “Device Client で IoT ジョブを実行する”	ユーザー定義 (複数ある可能性があります)	ユーザー定義 (複数ある可能性があります)

AWS IoT リソースを削除するには、作成したモノのリソースごとに以下の手順に従います。

1. *thing_name* を削除するモノのリソースの名前に置き換えてから、次のコマンドを実行して、ローカルホストコンピュータからモノのリソースにアタッチされた証明書を一覧表示します。

   ```
   aws iot list-thing-principals --thing-name thing_name
   ```

   このコマンドは、*thing_name* にアタッチされている証明書を一覧表示するこのようなレスポンスを返します。ほとんどの場合、リストに含まれる証明書は 1 つだけです。

   ```
   {
       "principals": [
           "arn:aws:iot:us-
   west-2:57EXAMPLE833:cert/23853eea3cf0edc7f8a69c74abeafa27b2b52823cab5b3e156295e94b26ae8ac"
       ]
   }
   ```

2. 前のコマンドでリストされた各証明書について、次の手順を実行します。

   a. *certificate_ID* を前のコマンドの証明書 ID に置き換えます。証明書 ID は、前のコマンドで返された ARN で cert/ の後に続く英数字です。次に、次のコマンドを実行して、証明書を無効にします。

      ```
      aws iot update-certificate --new-status INACTIVE --certificate-
      id certificate_ID
      ```

      成功すると、このコマンドは何も返しません。

   b. *certificate_ARN* を以前に返された証明書のリストの証明書 ARN に置き換えてから、次のコマンドを実行して、この証明書にアタッチされたポリシーを一覧表示します。

      ```
      aws iot list-attached-policies --target certificate_ARN
      ```

      このコマンドは、証明書にアタッチされたポリシーを一覧表示するこのようなレスポンスを返します。ほとんどの場合、リストにはポリシーが 1 つしかありません。

      ```
      {
          "policies": [
              {
                  "policyName": "DevCliTestThingPolicy",
      ```

```
            "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
DevCliTestThingPolicy"
        }
    ]
}
```

c. 証明書にアタッチされた各ポリシーについて、次の手順を実行します。

i. *policy_name* を 前のコマンドの policyName の値に置き換え、*certificate_ARN* を証明書の ARN に置き換えてから、次のコマンドを実行して、証明書からポリシーをデタッチします。

```
aws iot detach-policy --policy-name policy_name --target certificate_ARN
```

成功すると、このコマンドは何も返しません。

ii. *policy_name* を policyName の値に置き換えてから、次のコマンドを実行して、ポリシーがその他の証明書にアタッチされているかどうかを確認します。

```
aws iot list-targets-for-policy --policy-name policy_name
```

コマンドがこのような空のリストを返す場合、ポリシーはどの証明書にもアタッチされていません。その場合、ポリシーのバージョンを一覧表示します。ポリシーにまだ証明書が添付されている場合は、detach-thing-principal ステップに進みます。

```
{
    "targets": []
}
```

iii. *policy_name* を policyName の値に置き換えてから、次のコマンドを実行してポリシーのバージョンを確認します。ポリシーを削除するには、ポリシーのバージョンが 1 つだけであることが必要です。

```
aws iot list-policy-versions --policy-name policy_name
```

次の例のように、ポリシーにバージョンが 1 つしかない場合は、スキップして delete-policy ステップに進み、直ちにポリシーを削除できます。

```
{
    "policyVersions": [
```

```
        {
            "versionId": "1",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:02:46.778000+00:00"
        }
    ]
}
```

次の例のように、ポリシーに複数のバージョンがある場合、ポリシーを削除する前に、isDefaultVersion の値が false であるポリシーバージョンを削除する必要があります。

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

ポリシーバージョンを削除する必要がある場合は、*policy_name* を policyName の値に置き換え、*version_ID* を前のコマンドの versionId の値に置き換えてから、次のコマンドを実行してポリシーバージョンを削除します。

```
aws iot delete-policy-version --policy-name policy_name --policy-version-
id version_ID
```

成功すると、このコマンドは何も返しません。

ポリシーバージョンを 1 つ削除した後、ポリシーのポリシーバージョンが 1 つになるまで、このステップを繰り返します。

iv. *policy_name* を policyName の値に置き換えてから、次のコマンドを実行してポリシーを削除します。

```
aws iot delete-policy --policy-name policy_name
```

d. *thing_name* をモノの名前に置き換え、*certificate_ARN* を証明書の ARN に置き換えてから、次のコマンドを実行してモノのリソースから証明書をデタッチします。

```
aws iot detach-thing-principal --thing-name thing_name --
principal certificate_ARN
```

成功すると、このコマンドは何も返しません。

e. *certificate_ID* を前のコマンドの証明書 ID に置き換えます。証明書 ID は、前のコマンドで返された ARN で cert/ の後に続く英数字です。次に、次のコマンドを実行して、証明書リソースを削除します。

```
aws iot delete-certificate --certificate-id certificate_ID
```

成功すると、このコマンドは何も返しません。

3. *thing_name* をモノの名前に置き換えて、次のコマンドを実行してモノを削除します。

```
aws iot delete-thing --thing-name thing_name
```

成功すると、このコマンドは何も返しません。

AWS リソースをクリーンアップする

この手順は、このラーニングパスのチュートリアルを完了する際に作成した他の AWS リソースを特定して削除するのに役立ちます。

このラーニングパスで作成されたその他の AWS リソース

チュートリアル	リソースタイプ	リソース名または ID
the section called “Device Client で IoT ジョブを実行する”	Amazon S3 オブジェクト	hello-world-job.json

チュートリアル	リソースタイプ	リソース名または ID
the section called “Device Client で IoT ジョブを実行する”	AWS IoT ジョブリソース	ユーザー定義

このラーニングパスで作成された AWS リソースを削除するには

1. このラーニングパスで作成されたジョブを削除するには

   a. このコマンドを実行して、 のジョブを一覧表示します AWS アカウント。

   ```
   aws iot list-jobs
   ```

   コマンドは、 AWS アカウント および 内の AWS IoT ジョブのリストを返します AWS リージョン 。

   ```
   {
       "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. このラーニングパスで作成したジョブとしてリストから認識したジョブごとに、 を削除するジョブjobIdの値*jobId*に置き換え、このコマンドを実行して AWS IoT ジョブを削除します。

```
aws iot delete-job --job-id jobId
```

コマンドが正常に終了すると、何も返しません。

2. このラーニングパスの Amazon S3 バケットに保存したジョブドキュメントを削除するには

a. *bucket* を使用したバケットの名前に置き換えてから、次のコマンドを実行して、使用した Amazon S3 バケット内のオブジェクトを一覧表示します。

```
aws s3api list-objects --bucket bucket
```

このコマンドは、次のようにバケット内の Amazon S3 オブジェクトのリストを返します。

```
{
    "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. このラーニングパスで作成したオブジェクトとしてリストから認識したオブジェクトごとに、*bucket* をバケット名に置き換え、*key* を削除するオブジェクトのキー値に置き換えてから、次のコマンドを実行して Amazon S3 オブジェクトを削除します。

```
aws s3api delete-object --bucket bucket --key key
```

コマンドが正常に終了すると、何も返しません。

このラーニングパスの完了時に作成したすべての AWS リソースとオブジェクトを削除したら、チュートリアルを最初からやり直すことができます。

AWS IoT Device SDKsを使用したソリューションの構築

このセクションのチュートリアルでは、 を使用して本番環境にデプロイできる IoT ソリューションを開発する手順について説明します AWS IoT。

これらのチュートリアルは、 AWS IoT Device SDKs を使用し、安全で信頼性の高いソリューションを作成するために適用されている概念をより詳細に説明the section called “AWS IoT Device Client を使用したデモの構築”するため、 のセクションよりも完了までに時間がかかる場合があります。

AWS IoT Device SDKs を使用してソリューションの構築を開始する

これらのチュートリアルでは、さまざまな AWS IoT シナリオについて説明します。必要に応じて、チュートリアルでは AWS IoT Device SDKsを使用します。

トピック

- チュートリアル: Device SDK AWS IoT Core を使用して AWS IoT デバイスを に接続する
- デバイスデータを他の サービスにルーティングする AWS IoT ルールの作成
- デバイスがオフラインになっている間にデバイスの状態をデバイスシャドウで保持する
- チュートリアル: のカスタムオーソライザーの作成 AWS IoT Core
- チュートリアル: AWS IoT と Raspberry Pi による土壌湿度のモニタリング

チュートリアル: Device SDK AWS IoT Core を使用して AWS IoT デバイスを に接続する

このチュートリアルでは、デバイスを に接続 AWS IoT Core して、データを送受信する方法を示します AWS IoT。このチュートリアルを完了する AWS IoT Core と、デバイスが に接続するように設定され、デバイスが と通信する方法を理解できます AWS IoT。

トピック

- 前提条件
- のデバイスを準備します。 AWS IoT
- MQTT プロトコルを確認する
- pubsub.py Device SDK サンプルアプリケーションを確認する
- デバイスを接続して と通信する AWS IoT Core
- 結果を確認する
- チュートリアル: AWS IoT Device SDK for Embedded C の使用

前提条件

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- AWS IoT Core チュートリアルの開始方法 を完了していること

the section called “デバイスを設定する” する必要があるチュートリアルのセクションで、デバイスの the section called “Raspberry Pi または他のデバイスを接続する” オプションを選択し、Python 言語オプションを使用してデバイスを設定します。

> **Note**
>
> このチュートリアルでも使用するため、そのチュートリアルで使用するターミナルウィンドウは開いたままにします。

- Device SDK v2 for Python を実行できる AWS IoT デバイス。

  このチュートリアルでは、比較的強力なデバイスを必要とする Python コード例 AWS IoT Core を使用して、デバイスを に接続する方法を示します。リソースに制約のあるデバイスを使用している場合は、これらのコード例が機能しない可能性があります。その場合は、「the section called “AWS IoT Device SDK for Embedded C の使用”」チュートリアルの方が、接続に成功する可能性がより高くなります。

- デバイスに接続するために必要な情報を取得済みであること

  デバイスを に接続するには AWS IoT、モノの名前、ホスト名、ポート番号に関する情報が必要です。

  > **Note**
  >
  > カスタム認証を使用してデバイスを AWS IoT Coreに接続することもできます。オーソライザーの Lambda 関数に渡す接続データは、使用するプロトコルによって異なります。

  - モノの名前: 接続する AWS IoT モノの名前。デバイスとして を AWS IoT モノとして登録している必要があります。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。
  - ホスト名: アカウント固有の IoT エンドポイントのホスト名。
  - ポート番号: 接続先のポート番号。

  Python SDK の AWS IoT `configureEndpoint`メソッドを使用して、ホスト名とポート番号を設定できます。

```
myAWSIoTMQTTClient.configureEndpoint("random.iot.region.amazonaws.com", 8883)
```

のデバイスを準備します。 AWS IoT

AWS IoT Core チュートリアルの開始方法 では、デバイスと AWS アカウントが通信できるように準備しました。このセクションでは、 とのデバイス接続に適用される準備の側面を確認します AWS IoT Core。

デバイスの AWS IoT Core接続に際して

1. AWS アカウント が必要です。

   の手順セットアップ AWS アカウントでは、 をまだ作成 AWS アカウント していない場合に を作成する方法について説明します。
2. そのアカウントでは、 AWS アカウント および リージョンのデバイスに次のAWS IoT リソースを定義する必要があります。

   AWS IoT リソースの作成 の手順では、 AWS アカウント とリージョンのデバイス用にこれらのリソースを作成する方法が示されています。
   - AWS IoT に登録され、デバイスを認証するために有効化されたデバイス証明書。

     多くの場合、証明書は AWS IoT モノのオブジェクトを使用して作成され、そのオブジェクトにアタッチされます。モノのオブジェクトはデバイスが に接続するためには必要ありませんが AWS IoT、追加の AWS IoT 機能がデバイスで利用可能になります。
   - デバイス証明書にアタッチされたポリシーで、接続 AWS IoT Core して必要なすべてのアクションを実行することを許可します。
3. AWS アカウントのデバイスエンドポイントにアクセスできるインターネット接続。

   デバイスエンドポイントは で説明AWS IoT デバイスデータとサービスエンドポイントされており、AWS IoT コンソールの設定ページに表示されます。
4. 通信ソフトウェア (AWS IoT Device SDK が提供するものなど)。このチュートリアルでは、AWS IoT Device SDK v2 for Python を使用します。

MQTT プロトコルを確認する

サンプルアプリケーションについて説明する前に、MQTT プロトコルを理解しておくと役立ちます。MQTT プロトコルには、HTTP などの他のネットワーク通信プロトコルに比べていくつかの利点があり、IoT デバイスで一般的な選択肢となっています。このセクションでは、このチュートリアルに適用される MQTT の主要な側面を確認します。MQTT と HTTP を比較する方法の詳細については、「デバイス通信用のアプリケーションプロトコルの選択」を参照してください。

MQTT は、発行/サブスクライブ通信モデルを使用します

MQTT プロトコルは、ホストとの発行/サブスクライブ通信モデルを使用します。このモデルは、HTTP が使用するリクエスト/応答モデルとは異なります。MQTT では、デバイスは一意のクライアント ID によって識別されるホストとのセッションを確立します。データを送信するために、デバイスはトピックによって識別されたメッセージをホストのメッセージブローカーに発行します。メッセージブローカーからメッセージを受信するために、デバイスは、サブスクリプションリクエストでトピックフィルターをメッセージブローカーに送信することにより、トピックをサブスクライブします。

MQTT サポート永続セッション

メッセージブローカーは、デバイスからメッセージを受信し、サブスクライブしているデバイスにメッセージを発行します。永続セッション (開始デバイスが切断されている場合でもアクティブなセッション) では、デバイスは、切断中に発行されたメッセージを取得できます。デバイス側では、MQTT は、デバイスによって送信されたメッセージをホストが確実に受信できるようにするサービス品質レベル (QoS) をサポートします。

pubsub.py Device SDK サンプルアプリケーションを確認する

このセクションでは、このチュートリアルで使用されている AWS IoT Device SDK v2 for Python の `pubsub.py` サンプルアプリケーションを確認します。ここでは、 に接続 AWS IoT Core して MQTT メッセージを発行およびサブスクライブする方法を確認します。次のセクションでは、デバイスが接続および通信する方法を調べるのに役立ついくつかの演習を示します AWS IoT Core。

pubsub.py サンプルアプリケーションは、 との MQTT 接続のこれらの側面を示しています AWS IoT Core。

- 通信プロトコル
- 永続セッション
- サービスの品質
- メッセージの発行
- メッセージのサブスクリプション
- デバイスの切断と再接続

通信プロトコル

pubsub.py サンプルは、MQTT および MQTT over WSS プロトコルを使用した MQTT 接続を示しています。AWS 共通ランタイム (AWS CRT) ライブラリは、低レベルの通信プロトコルサポートを提供し、 AWS IoT Device SDK v2 for Python に含まれています。

MQTT

このpubsub.pyサンプルではmtls_from_path、MQTT プロトコル AWS IoT Core を使用してとの接続を確立mqtt_connection_builderするために、 で (ここに示す) を呼び出します。mtls_from_pathは X.509 証明書と TLS v1.2 を使用してデバイスを認証します。CRT AWS ライブラリは、その接続の下位レベルの詳細を処理します。

```
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 アカウントの IoT デバイスエンドポイント

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

cert_filepath

デバイスの証明書ファイルへのパス

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

pri_key_filepath

証明書ファイルで作成されたデバイスのプライベートキーファイルへのパス

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`ca_filepath`

Root CA ファイルへのパス。MQTT サーバーがまだトラストストアにない証明書を使用する場合にのみ必要です。

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`client_bootstrap`

ソケット通信アクティビティを処理する共通ランタイムオブジェクト

サンプルアプリケーションでは、このオブジェクトは `mqtt_connection_builder.mtls_from_path` の呼び出しの前にインスタンス化されます。

`on_connection_interrupted`, `on_connection_resumed`

デバイスの接続が中断され、再開されたときに呼び出すコールバック関数

`client_id`

AWS リージョンでこのデバイスを一意に識別する ID

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`clean_session`

新しい永続セッションを開始するか、既存のセッションに再接続するか (存在する場合)

`keep_alive_secs`

CONNECT リクエストで送信するキープアライブ値 (秒単位)。この間隔で ping が自動的に送信されます。サーバーは、この値の 1.5 倍の時間が経過しても ping を受信しなかった場合、接続が失われたとみなします。

MQTT over WSS

pubsub.py サンプルでは、[mqtt_connection_builder] の `websockets_with_default_aws_signing` (以下に示されています) を呼び出して、WSS 経由で MQTT プロトコルを使用して AWS IoT Core との接続を確立します。`websockets_with_default_aws_signing` は、署名 V4 を使用して WSS 経由で MQTT 接続を作成し、デバイスを認証します。

```
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 アカウントの IoT デバイスエンドポイント

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`client_bootstrap`

ソケット通信アクティビティを処理する共通ランタイムオブジェクト

サンプルアプリケーションでは、このオブジェクトは `mqtt_connection_builder.websockets_with_default_aws_signing` の呼び出しの前にインスタンス化されます。

`region`

AWS 署名 V4 認証で使用される署名リージョン。pubsub.py では、コマンドラインに入力されたパラメータを渡します。

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`credentials_provider`

認証に使用するために提供される AWS 認証情報

サンプルアプリケーションでは、このオブジェクトは `mqtt_connection_builder.websockets_with_default_aws_signing` の呼び出しの前にインスタンス化されます。

`websocket_proxy_options`

HTTP プロキシオプション (プロキシホストを使用している場合)

サンプルアプリケーションでは、この値は `mqtt_connection_builder.websockets_with_default_aws_signing` の呼び出しの前に初期化されます。

`ca_filepath`

Root CA ファイルへのパス。MQTT サーバーがまだトラストストアにない証明書を使用する場合にのみ必要です。

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`on_connection_interrupted`, `on_connection_resumed`

デバイスの接続が中断され、再開されたときに呼び出すコールバック関数

`client_id`

AWS リージョンでこのデバイスを一意に識別する ID。

サンプルアプリケーションでは、この値はコマンドラインから渡されます。

`clean_session`

新しい永続セッションを開始するか、既存のセッションに再接続するか (存在する場合)

`keep_alive_secs`

`CONNECT` リクエストで送信するキープアライブ値 (秒単位)。この間隔で ping が自動的に送信されます。サーバーは、この値の 1.5 倍の時間が経過しても ping を受信しなかった場合、接続が失われたとみなします。

HTTPS

HTTPS とは? では、HTTPS リクエストを発行するデバイス AWS IoT Core をサポートしています。プログラミングの観点からは、デバイスは他のアプリケーションと同様に HTTPS リクエストを AWS IoT Core に送信します。デバイスから HTTP メッセージを送信する Python プログラムの例については、Python の `requests` ライブラリを使用した HTTPS コード例を参照してください。この例では、 が AWS IoT Core MQTT メッセージとして解釈するように、HTTPS AWS IoT Core を使用して にメッセージを送信します。

はデバイスからの HTTPS リクエスト AWS IoT Core をサポートしますが、デバイスの通信に使用するプロトコルについて情報に基づいた決定デバイス通信用のアプリケーションプロトコルの選択ができるように、 に関する情報を必ず確認してください。

永続セッション

サンプルアプリケーションでは、clean_session パラメータを False に設定して、接続を永続化する必要があることを示します。実際には、これは、この呼び出しによって開かれた接続が、既存の永続セッション (存在する場合) に再接続することを意味します。そうでなければ、新しい永続セッションを作成して接続します。

永続セッションでは、デバイスに送信されたメッセージは、デバイスが接続されていない間、メッセージブローカーによって保存されます。デバイスが永続セッションに再接続すると、メッセージブローカーは、サブスクライブしている保存済みメッセージをデバイスに送信します。

永続セッションがない場合、デバイスは、デバイスが接続されていないときに送信されたメッセージを受信しません。どのオプションを使用するかは、アプリケーションと、デバイスが接続されていないときに発生するメッセージを通信する必要があるかどうかによって異なります。詳細については、「」を参照してくださいMQTT 永続的セッション

サービスの品質

デバイスがメッセージを発行およびサブスクライブするときに、優先するサービス品質 (QoS) を設定できます。 は、発行およびサブスクライブオペレーションの QoS レベル 0 と 1 AWS IoT をサポートします。の QoS レベルの詳細については AWS IoT、「」を参照してくださいMQTT Quality of Service (QoS) オプション。

Python の AWS CRT ランタイムは、サポートされる QoS レベルにこれらの定数を定義します。

Python サービス品質レベル

MQTT QoS レベル	SDK で使用される Python シンボリック値	説明
QoS レベル 0	mqtt.QoS.AT_MOST_ONCE	メッセージの送信は、受信されたかどうかにかかわらず、1 回だけ行われます。例えば、デバイスが接続されていない場合やネットワークエラーがある場合など、メッセージがまったく送信されない場合があります。

MQTT QoS レベル	SDK で使用される Python シンボリック値	説明
QoS レベル 1	`mqtt.QoS.AT_LEAST_ONCE`	メッセージは、PUBACK 確認応答を受信するまで繰り返し送信されます。

サンプルアプリケーションでは、発行およびサブスクライブのリクエストは QoS レベル 1 (`mqtt.QoS.AT_LEAST_ONCE`) で行われます。

- 発行での QoS

  デバイスが QoS レベル 1 のメッセージを発行すると、メッセージブローカーからの PUBACK 応答を受信するまでメッセージが繰り返し送信されます。デバイスが接続されていない場合、メッセージは再接続後に送信されるようにキューに入れられます。

- サブスクライブでの QoS

  デバイスが QoS レベル 1 のメッセージをサブスクライブすると、メッセージブローカーは、デバイスに送信できるようになるまで、デバイスがサブスクライブしているメッセージを保存します。メッセージブローカーは、デバイスから PUBACK 応答を受信するまでメッセージを再送信します。

メッセージの発行

への接続が正常に確立されると AWS IoT Core、デバイスはメッセージを発行できます。pubsub.py サンプルでは、`mqtt_connection` オブジェクトの `publish` オペレーションを呼び出してこれを行います。

```
mqtt_connection.publish(
    topic=args.topic,
    payload=message,
    qos=mqtt.QoS.AT_LEAST_ONCE
)
```

`topic`

メッセージを識別するメッセージのトピック名

サンプルアプリケーションでは、これはコマンドラインから渡されます。

`payload`

文字列としてフォーマットされたメッセージペイロード (例: JSON ドキュメント)

サンプルアプリケーションでは、これはコマンドラインから渡されます。

JSON ドキュメントは一般的なペイロード形式であり、他の AWS IoT のサービスによって認識されます。ただし、メッセージペイロードのデータ形式は、パブリッシャーとサブスクライバーが同意する任意の形式にすることができます。ただし、他の AWS IoT サービスでは、いくつかのケースにおいて、ほとんどのオペレーションで JSON と CBOR のみが認識されます。

`qos`

このメッセージの QoS レベル

メッセージのサブスクリプション

AWS IoT およびその他の のサービスやデバイスからメッセージを受信するには、デバイスはそれらのメッセージをトピック名でサブスクライブします。デバイスは、トピック名を指定して個々のメッセージをサブスクライブし、ワイルドカード文字を含めることができるトピックフィルターを指定してメッセージのグループをサブスクライブできます。この pubsub.py サンプルでは、ここに示すコードを使用してメッセージをサブスクライブし、受信後にメッセージを処理するためのコールバック関数を登録します。

```
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`

サブスクライブするトピック。これは、トピック名またはトピックフィルターにすることができます。

サンプルアプリケーションでは、これはコマンドラインから渡されます。

qos

デバイスが切断されている間、メッセージブローカーがこれらのメッセージを保存する必要があるかどうか。

mqtt.QoS.AT_LEAST_ONCE (QoS レベル 1) の値では、接続の作成時に永続セッションを指定する必要があります (clean_session=False)。

callback

サブスクライブされたメッセージを処理するために呼び出す関数。

mqtt_connection.subscribe 関数は、future とパケット ID を返します。サブスクリプションリクエストが正常に開始された場合、返されるパケット ID は 0 より大きくなります。サブスクリプションがメッセージブローカーによって受信され、登録されたことを確認するには、コード例に示すように、非同期オペレーションの結果が返されるまで待機する必要があります。

コールバック関数

pubsub.py サンプルのコールバックは、デバイスがサブスクライブされたメッセージを受信したときに処理します。

```
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

メッセージのトピック

これは、トピックフィルターにサブスクライブしている場合でも、受信したメッセージの特定のトピック名です。

payload

メッセージペイロード

このフォーマットはアプリケーション固有です。

kwargs

mqtt.Connection.subscribe で説明されている可能な追加の引数。

pubsub.py サンプルでは、on_message_received はトピックとそのペイロードのみを表示します。また、制限に達した後、プログラムを終了するために受信したメッセージもカウントします。

アプリケーションはトピックとペイロードを評価して、実行するアクションを決定します。

デバイスの切断と再接続

pubsub.py サンプルには、デバイスが切断されたときと接続が再確立されたときに呼び出されるコールバック関数が含まれています。これらのイベントに対してデバイスが実行するアクションは、アプリケーション固有です。

デバイスが初めて接続するとき、受信するトピックをサブスクライブする必要があります。再接続時にデバイスのセッションが存在する場合、そのサブスクリプションが復元され、それらのサブスクリプションから保存されたメッセージは再接続後にデバイスに送信されます。

再接続時にデバイスのセッションが存在しない場合は、サブスクリプションを再サブスクライブする必要があります。永続セッションには有効期限があり、デバイスが切断されている時間が長すぎると期限切れになる可能性があります。

デバイスを接続して と通信する AWS IoT Core

このセクションでは、デバイスの AWS IoT Coreへの接続のさまざまな側面を詳しく知るのに役立ついくつかの演習を紹介します。これらの演習では、 AWS IoT コンソールで MQTT テストクライアントを使用して、デバイスが発行する内容を確認し、デバイスにメッセージを発行します。これらの演習では、「AWS IoT Device SDK v2 for Python」の pubsub.py サンプルを使用し、「開始方法のチュートリアル」のチュートリアルで得た経験を基に構築します。

このセクションでは、以下を行います。

- ワイルドカードトピックフィルターをサブスクライブする
- トピックフィルターのサブスクリプションを処理する
- デバイスからメッセージを発行する

これらの演習では、pubsub.py サンプルプログラムから始めます。

> **Note**
>
> これらの演習では、開始方法のチュートリアル のチュートリアルを完了しており、そのチュートリアルのデバイスのターミナルウィンドウを使用していることを前提としています。

ワイルドカードトピックフィルターをサブスクライブする

この演習では、pubsub.py を呼び出してワイルドカードトピックフィルターをサブスクライブするために使用するコマンドラインを変更し、メッセージのトピックに基づいて受信したメッセージを処理します。

演習手順

この演習では、デバイスに温度制御と照明制御が含まれていると想像してください。これらのトピック名を使用して、トピックに関するメッセージを識別します。

1. 演習を開始する前に、開始方法のチュートリアル のチュートリアルのこのコマンドをデバイスで実行して、演習の準備がすべて整っていることを確認してください。

   ```
   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
   ```

   開始方法のチュートリアルで見たのと同じ出力が表示されるはずです。
2. この演習では、これらのコマンドラインパラメータを変更します。

アクション	コマンドラインパラメータ	Effect
追加	--message ""	リッスンのみを実行するように pubsub.py を設定する
追加	--count 2	2 通のメッセージを受信した後にプログラムを終了する
変更	--topic device/+/details	にサブスクライブするトピックフィルターを定義する

これらの変更を最初のコマンドラインに加えると、このコマンドラインになります。デバイスのターミナルウィンドウにこのコマンドを入力します。

```
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
```

プログラムは次のように表示されます。

```
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...
```

ターミナルにこのように表示された場合、デバイスは準備ができており、トピック名がdeviceで始まり /detailで終わるメッセージをリッスンしています。それでは、それをテストしてみましょう。

3. デバイスが受信する可能性のあるメッセージをいくつか示します。

トピック名	メッセージペイロード
device/temp/details	{ "desiredTemp": 20, "currentTemp": 15 }
device/light/details	{ "desiredLight": 100, "currentLight": 50 }

4. AWS IoT コンソールで MQTT テストクライアントを使用して、前のステップで説明したメッセージをデバイスに送信します。

   a. AWS IoT コンソールで MQTT テストクライアントを開きます。

   b. [Subscribe to topic] (トピックへのサブスクライブ) の [Subscription topic] (トピックのサブスクリプション) フィールドで、トピックフィルター **device/+/details** を入力して、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。

c. MQTT テストクライアントの [Subscriptions] (サブスクリプション) 列で、[device/+/details] を選択します。

d. 上記の表の各トピックについて、MQTT テストクライアントで次の操作を行います。

1. [Publish] (発行) で、テーブルの [Topic name] (トピック名) 列の値を入力します。
2. トピック名の下のメッセージペイロードフィールドに、表の [Message payload] (メッセージペイロード) 列の値を入力します。
3. pubsub.py が実行されているターミナルウィンドウを確認し、MQTT テストクライアントで [Publish to topic] (トピックに発行) を選択します。

ターミナルウィンドウで pubsub.py によってメッセージが受信されたことがわかります。

演習結果

これにより、pubsub.py は、ワイルドカードトピックフィルターを使用してメッセージをサブスクライブし、それらを受信し、ターミナルウィンドウに表示しました。単一のトピックフィルターをサブスクライブし、2 つの異なるトピックを持つメッセージを処理するためにコールバック関数が呼び出されたことに注意してください。

トピックフィルターのサブスクリプションを処理する

前の演習に基づいて、pubsub.py サンプルアプリケーションを変更してメッセージトピックを評価し、トピックに基づいてサブスクライブされたメッセージを処理します。

演習手順

メッセージトピックを評価するには

1. pubsub.py を pubsub2.py にコピーします。
2. お好きなテキストエディタまたは IDE で pubsub2.py を開きます。
3. pubsub2.py で、on_message_received 関数を見つけます。
4. on_message_received で、次のコードを print("Received message で始まる行の後、および global received_count で始まる行の前に挿入します。

```
    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. このコマンドラインを使用して、変更を保存し、変更したプログラムを実行します。

```
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 コンソールで、MQTT テストクライアントを開きます。
7. [Subscribe to topic] (トピックへのサブスクライブ) の [Subscription topic] (トピックのサブスクリプション) フィールドで、トピックフィルター **device/+/details** を入力して、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。
8. MQTT テストクライアントの [Subscriptions] (サブスクリプション) 列で、[device/+/details] を選択します。
9. この表の各トピックについて、MQTT テストクライアントで次の操作を行います。

トピック名	メッセージペイロード
device/temp/details	{ "desiredTemp": 20, "currentTemp": 15 }
device/light/details	{ "desiredLight": 100, "currentLight": 50 }

1. [Publish] (発行) で、テーブルの [Topic name] (トピック名) 列の値を入力します。
2. トピック名の下のメッセージペイロードフィールドに、表の [Message payload] (メッセージペイロード) 列の値を入力します。

3. pubsub.py が実行されているターミナルウィンドウを確認し、MQTT テストクライアントで [Publish to topic] (トピックに発行) を選択します。

ターミナルウィンドウで pubsub.py によってメッセージが受信されたことがわかります。

ターミナルウィンドウに、このような内容が表示されます。

```
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!
```

演習結果

この演習では、サンプルアプリケーションがコールバック関数で複数のメッセージを認識して処理するようにコードを追加しました。これにより、デバイスはメッセージを受信し、それに基づいてアクションを実行できます。

デバイスが複数のメッセージを受信して処理する別の方法は、異なるメッセージを個別にサブスクライブし、各サブスクリプションを独自のコールバック関数に割り当てることです。

デバイスからメッセージを発行する

pubsub.py サンプルアプリケーションを使用して、デバイスからメッセージを発行できます。メッセージをそのまま発行しますが、メッセージを JSON ドキュメントとして読み取ることはできません。この演習では、読み取り可能なメッセージペイロードに JSON ドキュメントを発行できるようにサンプルアプリを変更します AWS IoT Core。

演習手順

この演習では、次のメッセージが device/dataトピックとともに送信されます。

```
{
    "timestamp": 1601048303,
    "sensorId": 28,
    "sensorData": [
        {
        "sensorName": "Wind speed",
        "sensorValue": 34.2211224
        }
    ]
}
```

この演習のメッセージを監視するように MQTT テストクライアントを準備するには

1. [Subscribe to topic] (トピックへのサブスクライブ)の[Subscription topic field] (トピックのサブスクリプションフィールド)で、topic filter(トピックフィルター) **device/data** を入力して、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。
2. MQTT テストクライアントの [Subscriptions] (サブスクリプション) 列で、[device/data] (デバイス/データ) を選択します。
3. MQTT テストクライアントウィンドウを開いたままにして、デバイスからのメッセージを待ちます。

pubsub.py サンプルアプリケーションで JSON ドキュメントを送信するには

1. デバイスで、pubsub.py を pubsub3.py にコピーします。
2. pubsub3.py を編集して、発行するメッセージのフォーマット方法を変更します。
   a. テキストエディタで pubsub3.py を開きます。
   b. 次のコード行を見つけます。

      ```
      message = "{} [{}]".format(message_string, publish_count)
      ```

   c. 次のように変更します。

      ```
      message = "{}".format(message_string)
      ```

   d. 次のコード行を見つけます。

```
    message_json = json.dumps(message)
```

e. 次のように変更します。

```
    message = "{}".json.dumps(json.loads(message))
```

f. 変更を保存します。

3. デバイス上でこのコマンドを実行して、メッセージを 2 回送信します。

```
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 テストクライアントで、次のようにメッセージペイロード内の JSON ドキュメントを解釈してフォーマットしたことを確認します。



デフォルトでは、pubsub3.py は送信するメッセージもサブスクライブします。アプリの出力でメッセージを受信したことがわかります。ターミナルウィンドウは次のようになります。

```
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!
```

演習結果

これにより、デバイスは に送信するメッセージを生成 AWS IoT Core して基本的な接続をテストし、 が処理 AWS IoT Core するデバイスメッセージを提供できます。たとえば、このアプリを使用して、デバイスからテスト AWS IoT ルールアクションにテストデータを送信できます。

結果を確認する

このチュートリアルの例では、デバイスが と通信する方法の基本 AWS IoT Core、つまり AWS IoT ソリューションの基本的な部分について実践的な経験をしました。デバイスが と通信できる場合 AWS IoT Core、対応できるサービスやその他のデバイスにメッセージを AWS 渡すことができます。同様に、 AWS サービスやその他のデバイスは、デバイスに返送されるメッセージにつながる情報を処理できます。

さらに詳しく調べる準備ができたら AWS IoT Core 、以下のチュートリアルを試してください。

- the section called “Amazon SNS 通知の送信”
- the section called “デバイスデータを DynamoDB テーブルに保存する”
- the section called “ AWS Lambda 関数を使用した通知のフォーマット”

チュートリアル: AWS IoT Device SDK for Embedded C の使用

このセクションでは、AWS IoT Device SDK for Embedded C を実行する方法について説明します。

このセクションの手順

- ステップ 1: AWS IoT Device SDK for Embedded C をインストールする

- ステップ 2: サンプルアプリケーションを設定する
- ステップ 3: サンプルアプリケーションをビルドして実行する

ステップ 1: AWS IoT Device SDK for Embedded C をインストールする

AWS IoT Device SDK for Embedded C は通常、最適化された C 言語ランタイムを必要とするリソース制約のあるデバイスを対象としています。この SDK は、任意のオペレーティングシステムで使用でき、任意のプロセッサタイプ (MCU や MPU など) でホストできます。使用可能なメモリと処理リソースが多い場合は、上位の AWS IoT Device および Mobile SDK (C++、Java、JavaScript、Python など) のいずれかを使用することをお勧めします。

一般的に、AWS IoT Device SDK for Embedded C は、組み込みオペレーティングシステムを実行する MCU またはローエンド MPU を使用するシステムを対象としています。このセクションのプログラミング例では、ご利用のデバイスが Linux を使用していると仮定しています。

Example

1. GitHub から AWS IoT Device SDK for Embedded C をデバイスにダウンロードします。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-embedded-c.git --recurse-
   submodules
   ```

   これにより、現在のディレクトリに aws-iot-device-sdk-embedded-c という名前のディレクトリが作成されます。

2. そのディレクトリに移動し、最新のリリースを確認します。最新のリリースタグについては、github.com/aws/aws-iot-device-sdk-embedded-C/tags を参照してください。

   ```
   cd aws-iot-device-sdk-embedded-c
   git checkout latest-release-tag
   ```

3. OpenSSL バージョン 1.1.0 以降をインストールします。OpenSSL 開発ライブラリは、通常、パッケージマネージャーを介してインストールされた場合、「libssl-dev」または「openssl-devel」と呼ばれます。

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

ステップ 2: サンプルアプリケーションを設定する

AWS IoT Device SDK for Embedded C には、サンプルアプリケーションが含まれています。分かりやすいように、このチュートリアルでは `mqtt_demo_mutual_auth` アプリケーションを使用して、AWS IoT Core メッセージブローカーに接続し MQTT トピックにサブスクライブして発行する方法を示します。

1. AWS IoT Core チュートリアルの開始方法 で作成した証明書およびプライベートキーを `build/bin/certificates` ディレクトリにコピーします。

   > **Note**
   >
   > デバイスおよびルート CA 証明書の有効期限切れや失効の対象となります。これらの証明書の有効期限が切れたり、失効した場合は、新しい CA 証明書またはプライベートキーおよびデバイス証明書をデバイスにコピーする必要があります。

2. サンプルは、パーソナル AWS IoT Core エンドポイント、プライベートキー、証明書、およびルート CA 証明書で設定する必要があります。`aws-iot-device-sdk-embedded-c/demos/mqtt/mqtt_demo_mutual_auth` ディレクトリに移動します。

   AWS CLIがインストールされている場合は、コマンドを使用してアカウントのエンドポイント URL を見つける事ができます。

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

   AWS CLI がインストールされていない場合は、[AWS IoTconsole](コンソール)を開きます。ナビゲーションペインで、[Manage (管理)]、[Things (モノ)] の順に選択します。デバイスの IoT のモノを選択し、[Interact] (操作) を選択します。モノ の詳細ページの [HTTPS] セクションにエンドポイントが表示されます。

3. `demo_config.h` ファイルを開いて、以下の値を更新します。

   AWS_IOT_ENDPOINT

   パーソナルエンドポイント。

   CLIENT_CERT_PATH

   証明書ファイルのパス (例　`certificates/device.pem.crt`")。

CLIENT_PRIVATE_KEY_PATH

プライベートキーのファイル名 (例　certificates/private.pem.key)。

以下に例を示します。

```
// 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. 次のコマンドを使用して、デバイスに CMake がインストールされているかどうかを確認します。

```
cmake --version
```

コンパイラのバージョン情報が表示された場合は、次のセクションに進みます。

エラーが発生する、または情報が表示されない場合は、次のコマンドを使用して cmake パッケージをインストールする必要があります。

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

cmake --version コマンドを再度実行し、CMake がインストールされ、続行する準備ができていることを確認します。

5. 次のコマンドを使用して、デバイスに開発ツールがインストールされているかどうかを確認します。

```
gcc --version
```

コンパイラのバージョン情報が表示された場合は、次のセクションに進みます。

エラーが発生したり、コンパイラ情報が表示されない場合は、次のコマンドを使用して build-essential パッケージをインストールする必要があります。

```
sudo apt-get install build-essential
```

gcc --version コマンドを再度実行し、ビルドツールがインストールされ、続行する準備ができていることを確認します。

ステップ 3: サンプルアプリケーションをビルドして実行する

この手順では、デバイスで mqtt_demo_mutual_auth アプリケーションを生成し、AWS IoT Device SDK for Embedded C を使用して AWS IoT コンソールに接続する方法について説明します。

AWS IoT Device SDK for Embedded C サンプルアプリケーションを実行するには

1. aws-iot-device-sdk-embedded-c に移動し、ディレクトリを作成します。

   ```
   mkdir build && cd build
   ```

2. 次の CMake コマンドを入力して、ビルドに必要な Makefiles を生成します。

   ```
   cmake ..
   ```

3. 次のコマンドを入力して、実行可能アプリケーションファイルをビルドします。

   ```
   make
   ```

4. 次のコマンドで mqtt_demo_mutual_auth アプリを実行します。

   ```
   cd bin
   ./mqtt_demo_mutual_auth
   ```

   次のような出力が表示されます:

```
[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.
```

これで、デバイスは AWS IoT Device SDK for Embedded C を使用して AWS IoT に接続されました。

AWS IoT コンソールを使用して、サンプルアプリケーションが発行している MQTT メッセージを表示することもできます。AWS IoT コンソールで MQTT クライアントを使用する方法については、「the section called “MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する”」を参照してください。

デバイスデータを他の サービスにルーティングする AWS IoT ルールの作成

これらのチュートリアルでは、より一般的な AWS IoT ルールアクションを使用してルールを作成およびテストする方法を示します。

AWS IoT ルールは、デバイスから他の AWS サービスにデータを送信します。特定の MQTT メッセージをリッスンし、メッセージペイロード内のデータをフォーマットし、結果を他の AWS のサービスに送信します。

Lambda 関数やそれ以上に複雑なものを使用するルールを作成することが目標であっても、ここに示されている順序でこれらを試すことをお勧めします。チュートリアルは、基本的なものから複雑なものへと順に表示されます。新しい概念を段階的に提示することで、特定のチュートリアルを持たないルールアクションの作成に使用できる概念を学習するのに役立ちます。

> **Note**
>
> AWS IoT ルールは、IoT デバイスから他の AWS サービスにデータを送信するのに役立ちます。ただし、これを正常に実行するには、データを送信する他のサービスに関する実用的な知識が必要です。これらのチュートリアルでは、タスクを完了するために必要な情報が提供されていますが、ソリューションで使用する前に、データの送信先のサービスについて詳しく知っておくと便利です。他のサービスの詳細な説明 AWS は、これらのチュートリアルの範囲外です。

チュートリアルのシナリオの概要

これらのチュートリアルのシナリオは、定期的にデータを発行する気象センサーデバイスのシナリオです。この架空のシステムには、このようなセンサー装置がたくさんあります。ただし、このセクションのチュートリアルでは、1 つのデバイスに焦点を当てつつ、複数のセンサーに対応する方法を示します。

このセクションのチュートリアルでは、 AWS IoT ルールを使用して、気象センサーデバイスのこの架空のシステムで次のタスクを実行する方法を示します。

- チュートリアル: MQTT メッセージの再発行

  このチュートリアルでは、気象センサーから受信した MQTT メッセージを、センサー ID と温度値のみを含むメッセージとして再発行する方法を示します。 AWS IoT Core サービスのみを使用し、シンプルな SQL クエリと MQTT クライアントを使用してルールをテストする方法をデモンストレーションします。

- チュートリアル: Amazon SNS 通知の送信

  このチュートリアルでは、気象センサーデバイスの値が特定の値を超えた場合に SNS メッセージを送信する方法を示します。前のチュートリアルで説明した概念に基づいており、別の AWS サービスである Amazon Simple Notification Service (Amazon SNS) を操作する方法を追加しています。

Amazon SNS を初めて使用する場合は、このチュートリアルを開始する前に、開始方法の演習を行います。

- チュートリアル: デバイスデータの DynamoDB テーブルへの保存

  このチュートリアルでは、気象センサーデバイスのデータをデータベーステーブルに保存する方法を示します。ルールクエリステートメントと置換テンプレートを使用して、送信先のサービスである Amazon DynamoDB のメッセージデータをフォーマットします。

  DynamoDB を初めて使用する場合は、このチュートリアルを開始する前に、開始方法の演習を行います。

- チュートリアル: AWS Lambda 関数を使用して通知をフォーマットする

  このチュートリアルでは、Lambda 関数を呼び出してデバイスデータを再フォーマットし、それをテキストメッセージとして送信する方法を示します。AWS Lambda 関数に Python スクリプトと AWS SDK 関数を追加して、気象センサーデバイスからのメッセージペイロードデータでフォーマットし、テキストメッセージを送信します。

  Lambda を初めて使用する場合は、このチュートリアルを開始する前に、開始方法の演習を行います。

AWS IoT ルールの概要

これらのチュートリアルはすべて AWS IoT ルールを作成します。

デバイスから別の AWS サービスにデータを送信する AWS IoT ルールには、以下を使用します。

- ルールクエリステートメントは、次のもので構成されます。
  - メッセージペイロードからデータを選択してフォーマットする SQL SELECT 句
  - 使用するメッセージを識別するトピックフィルター (ルールクエリステートメントの FROM オブジェクト)
  - アクションを実行する特定の条件を指定するオプションの条件ステートメント (SQL WHERE 句)
- 少なくとも 1 つのルールアクション

デバイスは、MQTT トピックにメッセージを発行します。SQL SELECT ステートメントのトピックフィルターは、ルールを適用する MQTT トピックを識別します。SQL SELECT ステートメントで指

定されたフィールドは、ルールのアクションで使用するために、着信 MQTT メッセージペイロードからのデータをフォーマットします。ルールのすべてのアクションのリストについては、「AWS IoT ルールのアクション」を参照してください。

このセクションのチュートリアル

- チュートリアル: MQTT メッセージの再発行
- チュートリアル: Amazon SNS 通知の送信
- チュートリアル: デバイスデータの DynamoDB テーブルへの保存
- チュートリアル: AWS Lambda 関数を使用して通知をフォーマットする

チュートリアル: MQTT メッセージの再発行

このチュートリアルでは、指定された MQTT メッセージが受信されたときに MQTT メッセージを発行する AWS IoT ルールを作成する方法を示します。受信メッセージペイロードは、発行前にルールによって変更できます。これにより、デバイスやそのファームウェアを変更することなく、特定のアプリケーションに合わせたメッセージを作成できます。また、ルールのフィルタリング機能を使用して、特定の条件が満たされた場合にのみメッセージを発行することもできます。

ルールによって再発行されるメッセージは、他の AWS IoT デバイスまたはクライアントによって送信されるメッセージのように動作します。デバイスは、他の MQTT メッセージトピックにサブスクライブできるのと同様に、再発行されたメッセージにサブスクライブできます。

このチュートリアルでは、次の内容を学習します。

- ルールクエリステートメントでシンプルな SQL クエリと関数を使用する方法
- MQTT クライアントを使用して AWS IoT ルールをテストする方法

このチュートリアルの完了には 30 分ほどかかります。

このチュートリアルでは、次の作業を行います。

- MQTT トピックと AWS IoT ルールを確認する
- ステップ 1: MQTT メッセージを再発行する AWS IoT ルールを作成する
- ステップ 2: 新しいルールをテストする
- ステップ 3: 結果と次のステップを確認する

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- セットアップ AWS アカウント

  このチュートリアルを完了するには、 AWS アカウント と AWS IoT コンソールが必要です。
- MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する を確認したこと

  トピックにサブスクライブおよび発行するために MQTT クライアントを使用できることを確認してください。この手順では、MQTT クライアントを使用して新しいルールをテストします。

MQTT トピックと AWS IoT ルールを確認する

AWS IoT ルールについて話す前に、MQTT プロトコルを理解するのに役立ちます。IoT ソリューションでは、MQTT プロトコルは HTTP などの他のネットワーク通信プロトコルよりも優れた機能を提供しているため、IoT デバイスでの使用のための選択肢として好まれています。このセクションでは、このチュートリアルに適用される MQTT の主な側面を見ていきます。MQTT と HTTP を比較する方法の詳細については、「デバイス通信用のアプリケーションプロトコルの選択」を参照してください。

MQTT プロトコル

MQTT プロトコルは、ホストとの発行/サブスクライブ通信モデルを使用します。データを送信するために、デバイスはトピックによって識別される AWS IoT メッセージをメッセージブローカーに発行します。メッセージブローカーからメッセージを受信するために、デバイスは、サブスクリプションリクエストでトピックフィルターをメッセージブローカーに送信することにより、受信するトピックにサブスクライブします。 AWS IoT ルールエンジンは、メッセージブローカーから MQTT メッセージを受信します。

AWS IoT ルール

AWS IoT ルールは、ルールクエリステートメントと 1 つ以上のルールアクションで構成されます。AWS IoT ルールエンジンが MQTT メッセージを受信すると、これらの要素は次のようにメッセージに対するアクションを実行します。

- ルールクエリステートメント

  ルールのクエリステートメントは、使用する MQTT トピックを記述し、メッセージペイロードからのデータを解釈し、一般的な SQL データベースで使用されるステートメントに似た SQL ステートメントによって記述されるようにデータをフォーマットします。クエリステートメントの結果は、ルールのアクションに送信されるデータです。

- ルールアクション

  ルールの各ルールアクションは、ルールのクエリステートメントから生じるデータに作用します。は多くのルールアクション AWS IoT をサポートします。ただし、このチュートリアルでは、再発行 ルールアクションに焦点を当てます。このアクションは、クエリステートメントの結果を特定のトピックを持つ MQTT メッセージとして発行します。

ステップ 1: MQTT メッセージを再発行する AWS IoT ルールを作成する

このチュートリアルで作成する AWS IoT ルールは、*device_id* がメッセージを送信したデバイスの ID である device/*device_id*/data MQTT トピックにサブスクライブします。これらのトピックは、device/+/data として トピックフィルターによって記述されます。ここで、+ は、2 つのスラッシュ文字の間の任意の文字列に一致するワイルドカードです。

一致するトピックからルールがメッセージを受信すると、device_id および temperature の値を device/data/temp トピックを含む新しい MQTT メッセージとして再発行します。

例えば、device/22/dataトピックを含む MQTT メッセージのペイロードは、以下のようになります。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

このルールは、メッセージペイロードから temperature 値を、トピックから device_id を取得し、device/data/temp トピックを含む MQTT メッセージおよびメッセージペイロードとして次のように再発行します。

```
{
  "device_id": "22",
  "temperature": 28
}
```

このルールでは、デバイスの ID と温度データのみを必要とするデバイスは、`device/data/temp` トピックにサブスクライブして、その情報のみを受信します。

MQTT メッセージを再発行するルールを作成するには

1. AWS IoT コンソールのルールハブを開きます。
2. [Rules] (ルール) で、[Create] (作成) を選択して、新しいルールの作成を開始します。
3. [Create a rule] (ルールを作成) の上部で、次のように操作します。

   a. [Name] (名前) で、ルールの名前を入力します。このチュートリアルでは、**`republish_temp`** という名前を付けます。

      ルール名は、アカウントとリージョン内で一意である必要があります。また、スペースを含めることはできません。この名前にアンダースコア文字を使用して、ルールの名前の 2 つの単語を区切りました。

   b. [Description] (説明) で、ルールの説明を入力します。

      わかりやすい説明を使用すると、このルールの動作と作成した理由を思い出すのに役立ちます。説明は必要なだけ長くすることができるので、できるだけ詳述してください。

4. [Create a rule] (ルールを作成) の [Rule query statement] (ルールクエリステートメント) で、以下を実行します。

   a. [Using SQL version] (SQL バージョンの使用) で、**`2016-03-23`** を選択します。

   b. [Rule query statement] (ルールクエリステートメント) 編集ボックスで、ステートメントを入力します。

      ```
      SELECT topic(2) as device_id, temperature FROM 'device/+/data'
      ```

      このステートメント:

      - `device/+/data` トピックフィルターに一致するトピックを持つ MQTT メッセージをリッスンします。
      - トピック文字列から 2 番目の要素を選択し、`device_id` フィールドに割り当てます。
      - メッセージペイロードから値 `temperature` フィールドを選択し、`temperature` フィールドに割り当てます。

5. [Set one or more actions] (1 つ以上のアクションを設定) で、以下を実行します。

a. このルールのルールアクションのリストを開くには、[Add action] (アクションの追加) を選択します。

b. 「アクションの選択」で、「 AWS IoT トピックへのメッセージの再発行」を選択します。

c. アクションリストの一番下で、[Configure action] (アクションを設定) を選択すると、選択したアクションの設定ページが開きます。

6. [Configure action] (アクションの設定) で、次のとおり実行します。

a. [Topic] (トピック) で、**`device/data/temp`** と入力します。これは、このルールが発行するメッセージの MQTT トピックです。

b. [Quality of Service] (サービスの品質) で、[0 - The message is delivered zero or more times] (0 - メッセージは 0 回以上配信されます。) を選択します。

c. 「ロールを選択または作成して、このアクションを実行するための AWS IoT アクセス権を付与する」を参照してください。

i. [ロールの作成] を選択します。[Create a new role] (新しいロールを作成) ダイアログボックスが開きます。

ii. 新しいロールを説明する名前を入力します。このチュートリアルでは、**`republish_role`** を使用します。

新しいロールを作成すると、ルールアクションを実行するための正しいポリシーが作成され、新しいロールにアタッチされます。このルールアクションのトピックを変更するか、別のルールアクションでこのロールを使用する場合は、そのロールのポリシーを更新して、新しいトピックまたはアクションを承認する必要があります。既存のロールを更新するには、このセクションで [Update role] (ロールの更新) を選択します。

iii. [Create Role] (ロールの作成) を選択してロールを作成し、ダイアログボックスを閉じます。

d. [Add action] (アクションの追加) を選択してアクションをルールに追加し、[Create a rule] (ルールの作成) ページに戻ります。

7. AWS IoT トピックアクションへのメッセージの再発行が「1 つ以上のアクションを設定する」にリストされるようになりました。

新しいアクションのタイルの [Republish a message to an AWS IoT topic] (IoT トピックにメッセージを再発行する) をクリックすると、再発行アクションが発行されるトピックが表示されます。

このルールに追加するルールアクションはこれだけです。

8. [Create a rule] (ルールの作成) で、一番下までスクロールし、[Create rule] (ルールの作成) を選択してルールを作成し、この手順を完了します。

ステップ 2: 新しいルールをテストする

新しいルールをテストするには、MQTT クライアントを使用して、このルールで使用される MQTT メッセージを発行して、これにサブスクライブします。

新しいウィンドウの AWS IoT コンソールで MQTT クライアントを開きます。これにより、MQTT クライアントの設定を失うことなくルールを編集できます。MQTT クライアントは、サブスクリプションやメッセージログをコンソール内の別のページに移動するために残しておいても、それらを保持しません。

MQTT クライアントを使用してルールをテストするには

1. AWS IoT コンソールの MQTT クライアントで、入力トピック (この場合は `device/+/data`) をサブスクライブします。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。

   b. [Subscription topic] (サブスクリプショントピック) で、入力トピックフィルター **`device/+/data`** のトピックを入力します。

   c. 残りのフィールドはデフォルト設定のままにします。

   d. [Subscribe to topic] を選択します。

      [Subscriptions] (サブスクリプション) 列の [Publish to a topic] (トピックへの発行) の下に **`device/+/data`** が表示されます。

2. ルールが発行するトピックをサブスクライブします: `device/data/temp`。

   a. [Subscriptions] (サブスクリプション) で、[Subscribe to a topic] (トピックへサブスクライブする) を再度選択し、[Subscription topic] (サブスクリプショントピック) で、再発行されたメッセージのトピック **`device/data/temp`** を入力します。

   b. 残りのフィールドはデフォルト設定のままにします。

   c. [Subscribe to topic] を選択します。

      [Subscriptions] (サブスクリプション) 列の [device/+/data] の下に **`device/data/temp`** が表示されます。

3. 特定のデバイス ID **device/22/data** を使用して入力トピックにメッセージを発行します。ワイルドカード文字を含む MQTT トピックには発行できません。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Publish to topic] (トピックに発行) を選択します。

   b. [Publish] (発行) フィールドに、入力トピック名 **device/22/data** を入力します。

   c. ここに表示されているサンプルデータをコピーし、トピック名の下にある編集ボックスにサンプルデータを貼り付けます。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. MQTT メッセージを送信するには、[Publish to topic] (トピックに発行) を選択します。

4. 送信されたメッセージを確認します。

   a. MQTT クライアントの [Subscriptions] (サブスクリプション) で、以前にサブスクライブした 2 つのトピックの横に緑色のドットが表示されます。

      緑のドットは、最後に表示した後に 1 つ以上の新しいメッセージを受信したことを示します。

   b. [Subscriptions] (サブスクリプション) で [device/+/data] を選択して、メッセージペイロードが今発行したものと一致し、次のようになっていることを確認します。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. [Subscriptions] (サブスクリプション) で、[device/data/temp] を選択して、再発行されたメッセージペイロードが次のようになっていることを確認します。

```
{
  "device_id": "22",
  "temperature": 28
}
```

device_id 値は引用符で囲まれた文字列で、temperature 値は数値であることに注意してください。これは、topic() 関数が入力メッセージのトピック名から文字列を抽出している一方で、temperature 値は入力メッセージのペイロードの数値を使用することによるものです。

device_id 値を数値にしたい場合は、ルールクエリステートメントで topic(2) を次のように置き換えます。

```
cast(topic(2) AS DECIMAL)
```

topic(2) 値を数値にキャストすることは、トピックのその部分に数字のみが含まれている場合にのみ機能することに注意してください。

5. 正しいメッセージが device/data/temp トピックに発行されたことが確認できれば、ルールは機能しています。再発行ルールアクションの詳細については、次のセクションを参照してください。

   device/+/data または device/data/temp トピックのいずれかに正しいメッセージが発行されたことが確認できない場合は、トラブルシューティングのヒントを確認してください。

再発行メッセージルールのトラブルシューティング

想定する結果が表示されない場合に備えて、確認すべき事項をいくつかご紹介します。

- エラーバナーが表示された

  入力メッセージの発行時にエラーが発生した場合は、まずそのエラーを修正してください。次の手順は、このエラーを修正するのに役立つ場合があります。

- MQTT クライアントで入力メッセージが表示されない

入力メッセージを device/22/data トピックにサブスクライブすると、そのメッセージは MQTT クライアントに表示されます。device/+/dataトピックフィルターを選択します。

確認すべき事項

- サブスクライブしたトピックフィルターを確認する

  手順の説明に従って入力メッセージのトピックをサブスクライブした場合は、発行するたびに入力メッセージのコピーが表示されます。

  メッセージが表示されない場合は、サブスクライブしたトピック名を確認し、発行したトピックと比較します。トピック名は大文字と小文字が区別されます。サブスクライブしたトピックは、メッセージペイロードを発行したトピックと同一である必要があります。

- メッセージ発行機能を確認する

  MQTT クライアントの [Subscriptions] (サブスクリプション) で、[device/+/data] を選択し、パブリッシュ メッセージのトピックを確認してから、[Publish to topic] (トピックに発行) を選択します。トピックの下にある編集ボックスからメッセージペイロードがメッセージリストに表示されるのを確認できるはずです。

- MQTT クライアントに再発行されたメッセージが表示されない

  ルールが機能するには、メッセージを受信および再発行することを許可する正しいポリシーを有しており、メッセージを受信する必要があります。

  確認すべき事項

  - AWS リージョン MQTT クライアントの と作成したルールを確認する

    MQTT クライアントを実行しているコンソールは、作成したルールと同じ AWS リージョンにある必要があります。

  - ルールクエリステートメントの入力メッセージのトピックを確認する

    ルールが機能するためには、ルールクエリステートメントの FROM 句のトピックフィルターに一致するトピック名を持つメッセージを受信する必要があります。

    ルールクエリステートメントのトピックフィルターの綴りを、MQTT クライアントのトピックフィルターの綴りと照らし合わせて確認します。トピック名では大文字と小文字が区別され、メッセージのトピックはルールクエリステートメントのトピックフィルターと一致する必要があります。

- 入力メッセージペイロードの内容を確認する

  ルールが機能するためには、SELECT ステートメントで宣言されているメッセージペイロード内のデータフィールドを見つける必要があります。

  ルールクエリステートメントの `temperature` フィールドの綴りを、MQTT クライアントのメッセージペイロードの綴りと照らし合わせて確認します。フィールド名では大文字と小文字が区別され、ルールクエリステートメントの `temperature` フィールドはメッセージペイロードの `temperature` フィールドと同じである必要があります。

  メッセージペイロード内の JSON ドキュメントが正しくフォーマットされていることを確認します。JSON にコンマがないなどのエラーがある場合、ルールはそれを読み取ることができません。

- ルールアクションで再発行されたメッセージトピックを確認する

  再発行ルールアクションが新しいメッセージを発行するトピックは、MQTT クライアントでサブスクライブしたトピックと一致する必要があります。

  コンソールで作成したルールを開き、ルールアクションがメッセージを再発行するトピックを確認します。

- ルールによって使用されているロールを確認する

  ルールのアクションには、元のトピックを受け取り、新しいトピックを発行するためのアクセス許可が必要です。

  ルールがメッセージデータを受信して再発行することを許可するポリシーは、使用されるトピックに固有です。メッセージデータの再発行に使用するトピックを変更する場合は、ルールアクションのロールを更新して、現在のトピックに一致するようにポリシーを更新する必要があります。

  これが問題であると思われる場合は、再発行ルールアクションを編集して、新しいロールを作成します。ルールアクションによって作成された新しいロールは、これらのアクションを実行するために必要な権限を受け取ります。

ステップ 3: 結果と次のステップを確認する

このチュートリアルでは、次の作業を行いました。

- 新しい MQTT メッセージを生成するために、ルールクエリステートメントでシンプルな SQL クエリといくつかの関数を使用しました。
- 新しいメッセージを再発行するルールを作成しました。
- MQTT クライアントを使用して AWS IoT ルールをテストしました。

次の手順

このルールでいくつかのメッセージを再発行した後、チュートリアルのいくつかの側面を変更すると、再発行メッセージにどのように影響するかを試してみてください。手始めにいくつかアイデアをご紹介します。

- 入力メッセージのトピックで *device_id* を変更し、再発行されたメッセージ ペイロードの影響を観察します。
- ルールクエリステートメントで選択したフィールドを変更し、再発行されたメッセージペイロードに生じる影響を確認します。
- このシリーズの次のチュートリアルを試して、チュートリアル: Amazon SNS 通知の送信 の方法を学びましょう。

このチュートリアルで使用する [Republish] (再発行) ルールアクションも、ルールクエリステートメントのデバッグに役立ちます。例えば、このアクションをルールに追加して、ルールクエリステートメントがルールアクションで使用されるデータをどのようにフォーマットしているかを確認できます。

チュートリアル: Amazon SNS 通知の送信

このチュートリアルでは、MQTT メッセージデータを Amazon SNS トピックに送信して SMS テキストメッセージとして送信できるようにする AWS IoT ルールを作成する方法を示します。

このチュートリアルでは、温度がルールで設定された値を超えるたびに、気象センサーから Amazon SNS トピックのすべてのサブスクライバーにメッセージデータを送信するルールを作成します。ルールは、報告された温度がルールで設定された値を超えたことを検出し、そのとき、デバイス ID、報告された温度、および超過した温度制限のみを含む新しいメッセージペイロードを作成します。ルールは、新しいメッセージペイロードを JSON ドキュメントとして SNS トピックに送信し、このトピックが SNS トピックのすべてのサブスクライバーに通知します。

このチュートリアルでは、次の内容を学習します。

- Amazon SNS 通知を作成してテストする方法
- AWS IoT ルールから Amazon SNS 通知を呼び出す方法
- ルールクエリステートメントでシンプルな SQL クエリと関数を使用する方法
- MQTT クライアントを使用して AWS IoT ルールをテストする方法

このチュートリアルの完了には 30 分ほどかかります。

このチュートリアルでは、次の作業を行います。

- ステップ 1: SMS テキストメッセージを送信する Amazon SNS トピックを作成する
- ステップ 2: テキストメッセージを送信する AWS IoT ルールを作成する
- ステップ 3: AWS IoT ルールと Amazon SNS 通知をテストする
- ステップ 4: 結果と次のステップを確認する

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- セットアップ AWS アカウント

  このチュートリアルを完了するには、 AWS アカウント と AWS IoT コンソールが必要です。
- MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する を確認したこと

  トピックにサブスクライブおよび発行するために MQTT クライアントを使用できることを確認してください。この手順では、MQTT クライアントを使用して新しいルールをテストします。
- Amazon Simple Notification Service を確認しました

  これまでに Amazon SNS を使用したことがない場合は、Amazon SNS へのアクセスの設定を確認してください。他の AWS IoT チュートリアルをすでに完了している場合は、 AWS アカウント が正しく設定されている必要があります。

ステップ 1: SMS テキストメッセージを送信する Amazon SNS トピックを作成する

この手順では、気象センサーがメッセージデータを送信できる Amazon SNS トピックを作成する方法について説明します。その後、Amazon SNS トピックは、すべてのサブスクライバーに、温度制限を超えたことを SMS テキストメッセージで通知します。

SMS テキストメッセージを送信する Amazon SNS トピックを作成するには

1. Amazon SNS トピックを作成します。

   a. Amazon SNS コンソールにサインインします。

   b. 左のナビゲーションペインで、[トピック] を選択します。

   c. [トピック] ページで、[トピックの作成] を選択します。

   d. [Details] (詳細) で、[Standard] (標準) タイプを選択します。デフォルトでは、コンソールは FIFO トピックを作成します。

   e. [Name] (名前) で、SNS トピック名を入力します。このチュートリアルでは、**`high_temp_notice`** と入力します。

   f. ページの最下部にスクロールし、[Create topic] (トピックの作成) を選択します。

      コンソールに新しいトピックの [詳細] ページが表示されます。

2. Amazon SNS サブスクリプションを作成します。

   > **Note**
   >
   > このサブスクリプションで使用する電話番号では、このチュートリアルで送信するメッセージのテキストメッセージ料金が発生する可能性があります。

   a. high_temp_notice トピックの詳細ページで、[Create subscription] (サブスクリプションの作成) を選択します。

   b. [Create subscription] (サブスクリプションの作成) の [Details] (詳細) セクションの [Protocol] (プロトコル) リストで、[SMS] を選択します。

   c. [Endpoint] (エンドポイント) で、テキストメッセージを受信できる電話の番号を入力します。+ で始まり、国コードと市外局番が含まれ、他の句読文字が含まれないように入力してください。

   d. [Create subscription] を選択します。

3. Amazon SNS 通知をテストします。

   a. Amazon SNS コンソールの左のナビゲーションペインで、[Topics] (トピック) を選択します。

   b. トピックの詳細ページを開くには、[Topics] (トピック) のトピックのリストで、[high_temp_notice] を選択します。

c. [Publish message to topic] (トピックへのメッセージの発行) ページを開くには、[high_temp_notice] の詳細ページで [Publish message] (メッセージの発行) を選択します。

d. [Publish message to topic] (トピックへのメッセージの発行) の [Message body] (メッセージ本文) セクションの [Message body to send to the endpoint] (エンドポイントに送信するメッセージ本文) で、短いメッセージを入力します。

e. ページの下部まで下方向にスクロールし、[Publish message] (メッセージの発行) を選択します。

f. サブスクリプションを作成するときに以前に使用した番号の電話で、メッセージが受信されたことを確認します。

テストメッセージが受信されない場合は、電話番号と電話の設定を再度確認してください。

チュートリアルを続行する前に、Amazon SNS コンソールからテストメッセージを発行できることを確認してください。

ステップ 2: テキストメッセージを送信する AWS IoT ルールを作成する

このチュートリアルで作成する AWS IoT ルールは、 *device_id*がメッセージを送信したデバイスの ID である device/*device_id*/data MQTT トピックにサブスクライブします。これらのトピックは、device/+/data としてトピックフィルターで記述されます。ここで、+ は、2 つのスラッシュ文字の間の任意の文字列に一致するワイルドカードです。このルールは、メッセージペイロードの temperature フィールドの値もテストします。

ルールは、一致するトピックからメッセージを受信すると、トピック名から *device_id* を、メッセージペイロードから temperature 値を取得し、テストする制限に定数値を追加し、これらの値を JSON ドキュメントとして Amazon SNS 通知トピックに送信します。

例えば、気象センサーデバイス番号 32 からの MQTT メッセージは device/32/data トピックを使用し、次のようなメッセージペイロードを持っています。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
```

```
  }
}
```

ルールのルールクエリステートメントは、メッセージペイロードから `temperature` 値を、トピック名から *`device_id`* 値を取得し、定数 `max_temperature` 値を追加して、次のようなメッセージペイロードを Amazon SNS トピックに送信します。

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30
}
```

制限を超えた温度値を検出する AWS IoT ルールを作成し、Amazon SNS トピックに送信するデータを作成するには

1. AWS IoT コンソールのルールハブを開きます。
2. これが最初のルールである場合は、[Create] (作成) または [Create a rule] (ルールの作成) を選択します。
3. [Create a rule] (ルールの作成) で以下のとおり操作します。

   a. [名前] に「**`temp_limit_notify`**」と入力します。

      ルール名は AWS アカウント および リージョン内で一意である必要があり、スペースを含めることはできません。この名前にアンダースコア文字を使用して、ルールの名前の単語を区切りました。

   b. [Description] (説明) で、ルールの説明を入力します。

      わかりやすい説明を使用すると、このルールの動作と作成した理由を簡単に思い出すことができます。説明は必要なだけ長くすることができるので、できるだけ詳述してください。
4. [Create a rule] (ルールを作成) の [Rule query statement] (ルールクエリステートメント) で、以下を実行します。

   a. [Using SQL version] (SQL バージョンの使用) で、2016-03-23 を選択します。

   b. [Rule query statement] (ルールクエリステートメント) 編集ボックスで、ステートメントを入力します。

      ```
      SELECT topic(2) as device_id,
          temperature as reported_temperature,
      ```

```
  30 as max_temperature
FROM 'device/+/data'
WHERE temperature > 30
```

このステートメント:

- device/+/data トピックフィルターに一致し、temperature 値が 30 より大きいトピックを含む MQTT メッセージをリッスンします。
- トピック文字列から 2 番目の要素を選択し、device_id フィールドに割り当てます。
- メッセージペイロードから値 temperature フィールドを選択し、reported_temperature フィールドに割り当てます。
- 制限値を表す定数値 30 を作成し、それを max_temperature フィールドに割り当てます。

5. このルールのルールアクションのリストを開くには、[Set one or more actions] (1 つ以上のアクションを設定する) で [Add action] (アクションの追加) を選択します。
6. [Select an action] (アクションを選択してください) で、[Send a message as an SNS push notification] (SNS プッシュ通知としてメッセージを送信する) を選択します。
7. 選択したアクションの設定ページを開くには、アクションリストの下部にある [Configure action] (アクションの設定) を選択します。
8. [Configure action] (アクションの設定) で、次のとおり実行します。

   a. SNS ターゲットで、[Select] (選択) を選択し、high_temp_notice という名前の SNS トピックを見つけて、[Select] (選択) を選択します。

   b. [Message format] (メッセージ形式) で、[RAW] を選択します。

   c. 「ロールを選択または作成して、このアクションを実行するための AWS IoT アクセス権を付与する」で、「ロールの作成」を選択します。

   d. [Create a new role] (新しいロールの作成) の [Name] (名前) で、新しいロールの一意の名前を入力します。このチュートリアルでは、**sns_rule_role** を使用します。

   e. [ロールの作成] を選択します。

   このチュートリアルを繰り返す場合、または既存のロールを再利用する場合は、続行する前に [Update role] (ロールの更新) を選択してください。これにより、ロールのポリシードキュメントが更新され、SNS ターゲットで動作します。

9. [Add action] (アクションの追加) を選択して、[Create a rule] (ルールの作成) ページに戻ります。

新しいアクションのタイルの [Send a message as an SNS push notification] (SNS プッシュ通知としてメッセージを送信する) の下に、ルールが呼び出す SNS トピックが表示されます。

このルールに追加するルールアクションはこれだけです。

10. ルールを作成してこの手順を完了するには、[Create a rule] (ルールの作成) で一番下までスクロールし、[Create rule] (ルールの作成) を選択します。

ステップ 3: AWS IoT ルールと Amazon SNS 通知をテストする

新しいルールをテストするには、MQTT クライアントを使用して、このルールで使用される MQTT メッセージを発行して、これにサブスクライブします。

新しいウィンドウの AWS IoT コンソールで MQTT クライアントを開きます。これにより、MQTT クライアントの設定を失うことなくルールを編集できます。コンソールの別のページに移動するために MQTT クライアントを残しても、サブスクリプションやメッセージログは保持されません。

MQTT クライアントを使用してルールをテストするには

1. AWS IoT コンソールの MQTT クライアントで、入力トピック (この場合は `device/+/data`) をサブスクライブします。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。

   b. [Subscription topic] (サブスクリプショントピック) で、入力トピックフィルター **`device/+/data`** のトピックを入力します。

   c. 残りのフィールドはデフォルト設定のままにします。

   d. [Subscribe to topic] を選択します。

      [Subscriptions] (サブスクリプション) 列の [Publish to a topic] (トピックへの発行) の下に **`device/+/data`** が表示されます。

2. 特定のデバイス ID **`device/32/data`** を使用して入力トピックにメッセージを発行します。ワイルドカード文字を含む MQTT トピックには発行できません。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Publish to topic] (トピックに発行) を選択します。

   b. [Publish] (発行) フィールドに、入力トピック名 **`device/32/data`** を入力します。

c. ここに表示されているサンプルデータをコピーし、トピック名の下にある編集ボックスにサンプルデータを貼り付けます。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. [Publish to topic] (トピックに発行) を選択して、MQTT メッセージを発行します。

3. テキストメッセージが送信されたことを確認します。

a. MQTT クライアントの [Subscriptions] (サブスクリプション) の下に、以前にサブスクライブしたトピックの隣に緑色のドットが表示されます。

緑色のドットは、最後にメッセージを表示してから 1 つ以上の新しいメッセージが受信されたことを示します。

b. [Subscriptions] (サブスクリプション) で [device/+/data] を選択して、メッセージペイロードが今発行したものと一致し、次のようになっていることを確認します。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. SNS トピックのサブスクライブに使用した電話を確認し、メッセージペイロードの内容が次のようになっていることを確認します。

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

device_id 値は引用符で囲まれた文字列で、temperature 値は数値であることに注意してください。これは、topic() 関数が入力メッセージのトピック名から文字列を抽出している一方で、temperature 値は入力メッセージのペイロードの数値を使用することによるものです。

device_id 値を数値にしたい場合は、ルールクエリステートメントで topic(2) を次のように置き換えます。

```
cast(topic(2) AS DECIMAL)
```

topic(2) 値を数値にキャストすると、トピックのその部分に数字のみが含まれている場合にのみ DECIMAL 値が機能することに注意してください。

4. 温度が制限を超えていない MQTT メッセージの送信を試みます。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Publish to topic] (トピックに発行) を選択します。

   b. [Publish] (発行) フィールドに、入力トピック名 **device/33/data** を入力します。

   c. ここに表示されているサンプルデータをコピーし、トピック名の下にある編集ボックスにサンプルデータを貼り付けます。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. MQTT メッセージを送信するには、[Publish to topic] (トピックに発行) を選択します。

device/+/data サブスクリプションで送信したメッセージが表示されます。ただし、温度値がルールクエリステートメントの最大温度を下回っているため、テキストメッセージは受信されません。

正しい動作が確認できない場合は、トラブルシューティングのヒントを確認してください。

SNS メッセージルールのトラブルシューティング

想定する結果が得られない場合に備えて、確認すべき事項をいくつか示します。

- エラーバナーが表示された

  入力メッセージの発行時にエラーが発生した場合は、まずそのエラーを修正してください。次の手順は、このエラーを修正するのに役立つ場合があります。

- MQTT クライアントで入力メッセージが表示されない

  手順で説明されているように `device/+/data` トピックフィルターをサブスクライブした場合、入力メッセージを `device/22/data` トピックに発行するたびに、そのメッセージが MQTT クライアントに表示されます。

  確認すべき事項

  - サブスクライブしたトピックフィルターを確認する

    手順の説明に従って入力メッセージのトピックをサブスクライブした場合は、発行するたびに入力メッセージのコピーが表示されます。

    メッセージが表示されない場合は、サブスクライブしたトピック名を確認し、発行したトピックと比較します。トピック名は大文字と小文字が区別されます。サブスクライブしたトピックは、メッセージペイロードを発行したトピックと同一である必要があります。

  - メッセージ発行機能を確認する

    MQTT クライアントの [Subscriptions] (サブスクリプション) で、[device/+/data] を選択し、パブリッシュ メッセージのトピックを確認してから、[Publish to topic] (トピックに発行) を選択します。トピックの下にある編集ボックスからメッセージペイロードがメッセージリストに表示されるのを確認できるはずです。

- SMS メッセージが届かない

  ルールが機能するには、メッセージの受信と SNS 通知の送信を許可する正しいポリシーを有しており、メッセージを受信する必要があります。

  確認すべき事項

  - AWS リージョン MQTT クライアントの と作成したルールを確認する

    MQTT クライアントを実行しているコンソールは、作成したルールと同じ AWS リージョンにある必要があります。

- メッセージペイロードの温度値がテストしきい値を超えていることを確認します。

  ルールクエリステートメントで定義されている温度値が 30 以下の場合、ルールはそのアクションを実行しません。

- ルールクエリステートメントの入力メッセージのトピックを確認する

  ルールが機能するためには、ルールクエリステートメントの FROM 句のトピックフィルターに一致するトピック名を持つメッセージを受信する必要があります。

  ルールクエリステートメントのトピックフィルターの綴りを、MQTT クライアントのトピックフィルターの綴りと照らし合わせて確認します。トピック名では大文字と小文字が区別され、メッセージのトピックはルールクエリステートメントのトピックフィルターと一致する必要があります。

- 入力メッセージペイロードの内容を確認する

  ルールが機能するためには、SELECT ステートメントで宣言されているメッセージペイロード内のデータフィールドを見つける必要があります。

  ルールクエリステートメントの `temperature` フィールドの綴りを、MQTT クライアントのメッセージペイロードの綴りと照らし合わせて確認します。フィールド名では大文字と小文字が区別され、ルールクエリステートメントの `temperature` フィールドはメッセージペイロードの `temperature` フィールドと同じである必要があります。

  メッセージペイロード内の JSON ドキュメントが正しくフォーマットされていることを確認します。JSON にコンマがないなどのエラーがある場合、ルールはそれを読み取ることができません。

- ルールアクションで再発行されたメッセージトピックを確認する

  再発行ルールアクションが新しいメッセージを発行するトピックは、MQTT クライアントでサブスクライブしたトピックと一致する必要があります。

  コンソールで作成したルールを開き、ルールアクションがメッセージを再発行するトピックを確認します。

- ルールによって使用されているロールを確認する

  ルールのアクションには、元のトピックを受け取り、新しいトピックを発行するためのアクセス許可が必要です。

ルールがメッセージデータを受信して再発行することを許可するポリシーは、使用されるトピックに固有です。メッセージデータの再発行に使用するトピックを変更する場合は、ルールアクションのロールを更新して、現在のトピックに一致するようにポリシーを更新する必要があります。

これが問題であると思われる場合は、再発行ルールアクションを編集して、新しいロールを作成します。ルールアクションによって作成された新しいロールは、これらのアクションを実行するために必要な権限を受け取ります。

ステップ 4: 結果と次のステップを確認する

このチュートリアルでは、次の作業を行いました。

- Amazon SNS 通知トピックとサブスクリプションを作成し、テストしました。
- ルールクエリステートメントでシンプルな SQL クエリと関数を使用して、通知用の新しいメッセージを作成しました。
- カスタマイズされたメッセージペイロードを使用した Amazon SNS 通知を送信する AWS IoT ルールを作成しました。
- MQTT クライアントを使用して AWS IoT ルールをテストしました。

次の手順

このルールを使用していくつかのテキストメッセージを送信した後、チュートリアルの一部を変更すると、メッセージと送信される場合にどのような影響があるかを試してみてください。手始めにいくつかアイデアをご紹介します。

- 入力メッセージのトピックの *device_id* を変更し、テキストメッセージの内容に生じる影響を確認します。
- ルールクエリステートメントで選択したフィールドを変更し、テキストメッセージの内容に生じる影響を確認します。
- ルールクエリステートメントのテストを変更して、最高温度ではなく最低温度をテストします。`max_temperature` の名前を変更することを忘れないでください!
- SNS 通知の送信時に MQTT メッセージを送信する再発行ルールアクションを追加します。
- このシリーズの次のチュートリアルを試して、チュートリアル: デバイスデータの DynamoDB テーブルへの保存 の方法を学びましょう。

チュートリアル: デバイスデータの DynamoDB テーブルへの保存

このチュートリアルでは、DynamoDB テーブルにメッセージデータを送信する AWS IoT ルールを作成する方法を示します。

このチュートリアルでは、架空の気象センサーデバイスから DynamoDB テーブルにメッセージデータを送信するルールを作成します。このルールは、多くの気象センサーからのデータをフォーマットして、単一のデータベーステーブルに追加できるようにします。

このチュートリアルで学習する内容

- DynamoDB テーブルの作成方法
- AWS IoT ルールから DynamoDB テーブルにメッセージデータを送信する方法
- AWS IoT ルールで置換テンプレートを使用する方法
- ルールクエリステートメントでシンプルな SQL クエリと関数を使用する方法
- MQTT クライアントを使用して AWS IoT ルールをテストする方法

このチュートリアルの完了には 30 分ほどかかります。

このチュートリアルでは、次の作業を行います。

- ステップ 1: このチュートリアルの DynamoDB テーブルを作成する
- ステップ 2: DynamoDB テーブルにデータを送信する AWS IoT ルールを作成する
- ステップ 3: AWS IoT ルールと DynamoDB テーブルをテストする
- ステップ 4: 結果と次のステップを確認する

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- セットアップ AWS アカウント

  このチュートリアルを完了するには、 AWS アカウント と AWS IoT コンソールが必要です。
- MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する を確認したこと

  トピックにサブスクライブおよび発行するために MQTT クライアントを使用できることを確認してください。この手順では、MQTT クライアントを使用して新しいルールをテストします。
- Amazon DynamoDB の概要を確認しました

これまで DynamoDB を使用したことがない場合は、DynamoDB の開始方法を確認して、DynamoDB の基本的な概念と操作に慣れてください。

ステップ 1: このチュートリアルの DynamoDB テーブルを作成する

このチュートリアルでは、これらの属性を持つ DynamoDB テーブルを作成し、架空の気象センサーデバイスのデータを記録します。

- `sample_time` はプライマリキーで、サンプルが記録された時間を記述します。
- `device_id` はソートキーで、サンプルを提供したデバイスを記述します
- `device_data` は、デバイスから受信され、ルールクエリステートメントによってフォーマットされたデータです

このチュートリアルの DynamoDB テーブルを作成するには

1. DynamoDB コンソールを開き、[Create table] (テーブルの作成) を選択します。
2. Create table (テーブルの作成) の
   a. [Table name] (テーブル名) で、テーブル名 **`wx_data`** を入力します。
   b. [Partition key] (パーティションキー) で **`sample_time`** と入力し、フィールドの横にあるオプションリストで **`Number`** を選択します。
   c. [Sort key] (ソートキー) で **`device_id`** と入力します。フィールドの横にあるオプションリストで **`Number`** を選択します。
   d. ページの下部で、[Create] (作成) を選択します。

後で DynamoDB ルールアクションを設定するときに `device_data` を定義します。

ステップ 2: DynamoDB テーブルにデータを送信する AWS IoT ルールを作成する

このステップでは、ルールクエリステートメントを使用して、架空の気象センサーデバイスからのデータをフォーマットし、データベーステーブルに書き込みます。

気象センサーデバイスから受信したサンプルメッセージペイロードは、次のようになります。

```
{
  "temperature": 28,
```

```
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

データベースエントリでは、ルールクエリステートメントを使用して、メッセージペイロードの構造を次のようにフラット化します。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind_velocity": 22,
  "wind_bearing": 255
}
```

このルールでは、いくつかの 置換テンプレート も使用します。置換テンプレートは、関数およびメッセージデータから動的値を挿入することを可能にする式です。

DynamoDB テーブルにデータを送信する AWS IoT ルールを作成するには

1. AWS IoT コンソールの [Rules] (ルール) ハブを開きます。または、 AWS マネジメントコンソール 内で AWS IoT ホームページを開き、メッセージルーティング > ルールに移動できます。
2. [Rules] (ルール) で新しいルールの作成を開始するには、[Create rule] (ルールの作成) を選択します。
3. [Rule properties] (ルールのプロパティ) の

   a. [Rule name] (ルール名) で **`wx_data_ddb`** と入力します。

      ルール名は AWS アカウント および リージョン内で一意である必要があり、スペースを含めることはできません。この名前にアンダースコア文字を使用して、ルールの名前の 2 つの単語を区切りました。

   b. [Rule description] (ルールの説明) で、ルールを説明します。

      わかりやすい説明を使用すると、このルールの動作と作成した理由を簡単に思い出すことができます。説明は必要なだけ長くすることができるので、できるだけ詳述してください。
4. [次へ] を選択して続行します。

5. [SQL statement] (SQL ステートメント) の

   a. [SQL version] (SQL バージョン) で、**2016-03-23** を選択します。

   b. [SQL statement] (SQL ステートメント) 編集ボックスで、ステートメントを入力します。

```
SELECT temperature, humidity, barometer,
  wind.velocity as wind_velocity,
  wind.bearing as wind_bearing,
FROM 'device/+/data'
```

   このステートメント:

   - `device/+/data` トピックフィルターに一致するトピックを持つ MQTT メッセージをリッスンします。
   - `wind` 属性の要素を個々の属性としてフォーマットします。
   - `temperature`、`humidity`、および `barometer` 属性を変更せずに渡します。

6. [次へ] を選択して続行します。

7. [Rule actions] (ルールのアクション) で

   a. このルールのルールアクションのリストを開くには、[Action 1] (アクション 1) で **DynamoDB** を選択します。

> **Note**
>
> ルールアクションとして DynamoDBv2 ではなく DynamoDB を選択してください。

   b. [Table name] (テーブル名) で、前の手順で作成した DynamoDB テーブルの名前 **wx_data** を選択します。

   [Partition key type] (パーティションキータイプ) および [Sort key type] (ソートキータイプ) のフィールドには、DynamoDB テーブルの値が入力されます。

   c. [パーティションキー] に「**sample_time**」と入力します。

   d. [パーティションキーの値] に「**${timestamp()}**」と入力します。

   これは、このルールで使用する 置換テンプレート の最初のものです。メッセージペイロードの値を使用する代わりに、timestamp 関数から返された値を使用します。詳細については、AWS IoT Core デベロッパーガイドの 「タイムスタンプ」を参照してください。

e. [Sort key] (ソートキー) に「**device_id**」と入力します。

f. [ソートキー値] に「**${cast(topic(2) AS DECIMAL)}**」と入力します。

これは、このルールで使用する 置換テンプレート の 2 番目のものです。キーの数値形式と一致するように DECIMAL 値にキャストした後、デバイスの ID である [topic] (トピック) 名の 2 番目の要素の値を挿入します。トピックの詳細については、AWS IoT Core デベロッパーガイドの「トピック」を参照してください。または、キャスティングの詳細については、AWS IoT Core デベロッパーガイドの「キャスト」を参照してください。

g. [この列にメッセージデータを書き込む] に **device_data** と入力します。

これにより、DynamoDB テーブルに `device_data` 列が作成されます。

h. [オペレーション] は空白のままにします。

i. [IAM role] (IAM ロール) で、[Create new role] (新しいロールの作成) を選択します。

j. [Create role] (ロールの作成) ダイアログボックスの [Role name] (ロール名) に [wx_ddb_role] と入力します。この新しいロールには、作成した **wx_data** DynamoDB テーブルにデータを送信する **wx_data_ddb** ルールを許可する「aws-iot-rule」というプレフィックスが付いたポリシーが自動的に含まれます。

k. IAM role (IAM ロール) で **wx_ddb_role** を選択します。

l. ページの最下部にある [Next] (次へ) を選択します。

8. [Review and create] (確認と作成) ページの最下部で、[Create] (作成) を選択して、ルールを作成します。

ステップ 3: AWS IoT ルールと DynamoDB テーブルをテストする

新しいルールをテストするには、MQTT クライアントを使用して、このテストで使用した MQTT メッセージを発行し、これにサブスクライブします。

新しいウィンドウの AWS IoT コンソールで MQTT クライアントを開きます。これにより、MQTT クライアントの設定を失うことなくルールを編集できます。MQTT クライアントは、サブスクリプションやメッセージログをコンソール内の別のページに移動するために残しておいても、それらを保持しません。また、別のコンソールウィンドウをAWS IoT コンソールの DynamoDB Tables ハブに開いて、ルールが送信する新しいエントリを表示することもできます。

MQTT クライアントを使用してルールをテストするには

1. AWS IoT コンソールの MQTT クライアントで、入力トピック `device/+/data` をサブスクライブします。

a. MQTT クライアントで、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。

b. [Topic filter] (トピックフィルター) に、入力トピックフィルター **device/+/data** のトピックを入力します。

c. [Subscribe] (サブスクライブ) を選択します。

2. 特定のデバイス ID **device/22/data** を使用して入力トピックにメッセージを発行します。ワイルドカード文字を含む MQTT トピックには発行できません。

a. MQTT クライアントで、[Publish to a topic] (トピックへの発行) を選択します。

b. [Topic name] (トピック名) に、入力トピックの名前 **device/22/data** を入力します。

c. [Message payload] (メッセージペイロード) に、次のサンプルデータを入力します。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. MQTT メッセージを発行するには、[Publish] (発行) を選択します。

e. MQTT クライアントで、[Subscribe to a topic] (トピックへのサブスクライブ) を選択します。[Subscribe] (サブスクライブ) 列で、**device/+/data** サブスクリプションを選択します。前のステップのサンプルデータがそこに表示されていることを確認します。

3. ルールが作成した DynamoDB テーブルの行を確認します。

a. AWS IoT コンソールの DynamoDB Tables ハブで、wx_data を選択し、Items タブを選択します。

[Items] (アイテム) タブを既に開いている場合は、テーブルのヘッダーの右上にある更新アイコンを選択して、表示を更新する必要がある場合があります。

b. テーブルの sample_time 値はリンクであり、オープンであることに注意してください。最初のメッセージを送ったばかりの場合は、そのメッセージだけがリストに表示されます。

このリンクには、テーブルのその行のすべてのデータが表示されます。

c. device_data エントリを展開して、ルールクエリステートメントの結果のデータを表示します。

d. この表示で使用できるデータのさまざまな表現を詳しく確認します。このディスプレイでデータを編集することもできます。

e. このデータ行の確認が終了したら、加えた変更を保存するには [Save] (保存) を選択し、変更を保存せずに終了するには [Cancel] (キャンセル) を選択します。

正しい動作が確認できない場合は、トラブルシューティングのヒントを確認してください。

DynamoDB ルールのトラブルシューティング

想定する結果が表示されない場合に備えて、確認すべき事項をいくつかご紹介します。

- エラーバナーが表示された

  入力メッセージの発行時にエラーが発生した場合は、まずそのエラーを修正してください。次の手順は、このエラーを修正するのに役立つ場合があります。

- MQTT クライアントで入力メッセージが表示されない

  入力メッセージを `device/22/data` トピックにサブスクライブすると、そのメッセージは MQTT クライアントに表示されます。`device/+/data`トピックフィルターを選択します。

  確認すべき事項

  - サブスクライブしたトピックフィルターを確認する

    手順の説明に従って入力メッセージのトピックをサブスクライブした場合は、発行するたびに入力メッセージのコピーが表示されます。

    メッセージが表示されない場合は、サブスクライブしたトピック名を確認し、発行したトピックと比較します。トピック名は大文字と小文字が区別されます。サブスクライブしたトピックは、メッセージペイロードを発行したトピックと同一である必要があります。

  - メッセージ発行機能を確認する

    MQTT クライアントの [Subscriptions] (サブスクリプション) で、[device/+/data] を選択し、パブリッシュ メッセージのトピックを確認してから、[Publish to topic] (トピックに発行) を選択します。トピックの下にある編集ボックスからメッセージペイロードがメッセージリストに表示されるのを確認できるはずです。

- DynamoDB テーブルにデータが表示されない

まず、テーブルのヘッダーの右上にある更新アイコンを選択して、表示を更新します。探しているデータが表示されない場合は、以下を確認してください。

確認すべき事項

- AWS リージョン MQTT クライアントの と作成したルールを確認する

  MQTT クライアントを実行しているコンソールは、作成したルールと同じ AWS リージョンにある必要があります。

- ルールクエリステートメントの入力メッセージのトピックを確認する

  ルールが機能するためには、ルールクエリステートメントの FROM 句のトピックフィルターに一致するトピック名を持つメッセージを受信する必要があります。

  ルールクエリステートメントのトピックフィルターの綴りを、MQTT クライアントのトピックフィルターの綴りと照らし合わせて確認します。トピック名では大文字と小文字が区別され、メッセージのトピックはルールクエリステートメントのトピックフィルターと一致する必要があります。

- 入力メッセージペイロードの内容を確認する

  ルールが機能するためには、SELECT ステートメントで宣言されているメッセージペイロード内のデータフィールドを見つける必要があります。

  ルールクエリステートメントの `temperature` フィールドの綴りを、MQTT クライアントのメッセージペイロードの綴りと照らし合わせて確認します。フィールド名では大文字と小文字が区別され、ルールクエリステートメントの `temperature` フィールドはメッセージペイロードの `temperature` フィールドと同じである必要があります。

  メッセージペイロード内の JSON ドキュメントが正しくフォーマットされていることを確認します。JSON にコンマがないなどのエラーがある場合、ルールはそれを読み取ることができません。

- ルールアクションで使用されるキー名とフィールド名を確認する

  トピックルールで使用されるフィールド名は、発行メッセージの JSON メッセージペイロードにあるフィールド名と一致する必要があります。

  コンソールで作成したルールを開き、そのルールを持つ MQTT クライアントで使用されているルールアクション設定のフィールド名を確認します。

- ルールによって使用されているロールを確認する

ルールのアクションには、元のトピックを受け取り、新しいトピックを発行するためのアクセス許可が必要です。

ルールにメッセージデータの受信と DynamoDB テーブルの更新を許可するポリシーは、使用されるトピックに固有のものです。ルールで使用されるトピックまたは DynamoDB テーブル名を変更した場合は、ルールアクションのロールを更新して、ポリシーが一致するように更新する必要があります。

これが問題であると疑われる場合は、ルールのアクションを編集し、新しいロールを作成します。ルールアクションによって作成された新しいロールは、これらのアクションを実行するために必要な権限を受け取ります。

ステップ 4: 結果と次のステップを確認する

このルールを使用して DynamoDB テーブルにいくつかのメッセージを送信した後、チュートリアルからいくつかの側面を変更すると、テーブルに書き込まれるデータにどのように影響するかを試してみてください。手始めにいくつかアイデアをご紹介します。

- 入力メッセージのトピックの *device_id* を変更し、データへの影響を観察します。これを使用して、複数の気象センサーからのデータ受信をシミュレートできます。
- ルールクエリステートメントで選択したフィールドを変更し、データへの影響を確認します。これを使用すると、テーブルに保存されたデータをフィルタリングできます。
- 再発行ルールアクションを追加して、テーブルに追加された各行について MQTT メッセージを送信します。これをデバッグに使用することができます。

このチュートリアルを完了したら、the section called “ AWS Lambda 関数を使用した通知のフォーマット” を確認してください。

チュートリアル: AWS Lambda 関数を使用して通知をフォーマットする

このチュートリアルでは、MQTT メッセージデータを AWS Lambda アクションに送信してフォーマットし、別の AWS サービスに送信する方法を示します。このチュートリアルでは、 AWS Lambda アクションは AWS SDK を使用して、 の使用方法に関するチュートリアルで作成した Amazon SNS トピックにフォーマットされたメッセージを送信しますthe section called “Amazon SNS 通知の送信”。

the section called “Amazon SNS 通知の送信” の方法に関するチュートリアルでは、ルールのクエリステートメントから生成された JSON ドキュメントがテキストメッセージの本文として送信されました。その結果、次の例のようなテキストメッセージが表示されました。

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

このチュートリアルでは、 AWS Lambda ルールアクションを使用して、ルールクエリステートメントのデータを次の例のようにわかりやすい形式にフォーマットする AWS Lambda 関数を呼び出します。

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

このチュートリアルで作成する AWS Lambda 関数は、ルールクエリステートメントのデータを使用してメッセージ文字列をフォーマットし、 AWS SDK の SNS 発行関数を呼び出して通知を作成します。

このチュートリアルで学習する内容

- AWS Lambda 関数を作成してテストする方法
- AWS Lambda 関数で AWS SDK を使用して Amazon SNS 通知を発行する方法
- ルールクエリステートメントでシンプルな SQL クエリと関数を使用する方法
- MQTT クライアントを使用して AWS IoT ルールをテストする方法

このチュートリアルの完了には 45 分ほどかかります。

このチュートリアルでは、次の作業を行います。

- ステップ 1: テキストメッセージを送信する AWS Lambda 関数を作成する
- ステップ 2: AWS IoT ルールアクションを使用して AWS Lambda ルールを作成する
- ステップ 3: AWS IoT ルールと AWS Lambda ルールアクションをテストする
- ステップ 4: 結果と次のステップを確認する

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- セットアップ AWS アカウント

  このチュートリアルを完了するには、 AWS アカウント と AWS IoT コンソールが必要です。
- MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する を確認したこと

トピックにサブスクライブおよび発行するために MQTT クライアントを使用できることを確認してください。この手順では、MQTT クライアントを使用して新しいルールをテストします。

- このセクションのその他のルールチュートリアルを完了しました

  このチュートリアルでは、the section called “Amazon SNS 通知の送信” の方法に関するチュートリアルで作成した SNS 通知トピックが必要です。また、このセクションの他のルール関連のチュートリアルを完了していることも前提としています。

- AWS Lambda の概要を確認する

  AWS Lambda を使用したことがない場合は、AWS Lambda「」と「Lambda の開始方法」を確認して、その用語と概念を確認してください。

ステップ 1: テキストメッセージを送信する AWS Lambda 関数を作成する

このチュートリアルの AWS Lambda 関数は、ルールクエリステートメントの結果を受け取り、要素をテキスト文字列に挿入し、その結果の文字列を通知のメッセージとして Amazon SNS に送信します。

通知の送信にルールアクションthe section called “Amazon SNS 通知の送信”を使用した AWS IoT のチュートリアルとは異なり、このチュートリアルは AWS SDK の 関数を使用して Lambda 関数から通知を送信します。ただし、このチュートリアルで使用されている実際の Amazon SNS 通知トピックは、the section called “Amazon SNS 通知の送信” の方法に関するチュートリアルで使用したものと同じです。

テキストメッセージを送信する AWS Lambda 関数を作成するには

1. 新しい AWS Lambda 関数を作成します。

   a. AWS Lambda コンソールで、[関数の作成] を選択します。

   b. [Create function] (関数の作成) で、[Use a blueprint] (ブループリントを使用) を選択します。

      `hello-world-python` ブループリントを検索して選択し、[Configure] (設定) を選択します。

   c. [Basic information] (基本的な情報) で、次の操作を実行します。

      i. [Function name] (関数名) で、関数の名前 **`format-high-temp-notification`** を入力します。

ii. 実行ロールで、 AWS ポリシーテンプレートから新しいロールを作成するを選択します。

iii. [Role name] (ロール名) で、新しいロールの名前 **format-high-temp-notification-role** を入力します。

iv. [Policy templates - optional] (ポリシーテンプレート - オプション) で、[Amazon SNS publish policy] (Amazon SNS 発行ポリシー) を検索して選択します。

v. [Create function] を選択します。

2. ブループリントコードを変更して、Amazon SNS 通知をフォーマットして送信します。

a. 関数を作成すると、format-high-temp-notification の詳細ページが表示されます。表示されない場合は、[Lambda Functions] (Lambda 関数) ページから開きます。

b. format-high-temp-notification の詳細ページで、[Configuration] (設定) タブを選択し、[Function code] (関数コード) パネルまでスクロールします。

c. [Function code] (関数コード) ウィンドウの [Environment] (環境) ペインで、Python ファイル `lambda_function.py` を選択します。

d. [Function code] (関数コード) ウィンドウで、ブループリントから元のプログラムコードをすべて削除し、このコードに置き換えます。

```
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. [デプロイ] を選択します。

3. 新しいウィンドウで、the section called “Amazon SNS 通知の送信” の方法に関するチュートリアルから、Amazon SNS トピックの Amazon リソースネーム (ARN) を検索します。

   a. 新しいウィンドウで、Amazon SNS コンソールの [Topics] (トピック) ページを開きます。

   b. [Topics] (トピック) ページで、Amazon SNS トピックのリストから high_temp_notice 通知トピックを見つけます。

   c. 次のステップで使用する high_temp_notice 通知トピックの ARN を見つけます。

4. Lambda 関数のテストケースを作成します。

   a. コンソールの [Lambda Functions] (Lambda 関数) ページの format-high-temp-notification ページで、ページの右上にある (無効に見える) [Select a test event] (テストイベントの選択) を選択し、[Configure test events] (テストイベントの設定) を選択します。

   b. [Configure test event] (テストイベントの設定) で、[Create new test event] (新しいテストイベントの作成) を選択します。

   c. [Event name] (イベント名) で、**SampleRuleOutput** と入力します。

   d. [Event name] (イベント名) の下の JSON エディタで、このサンプル JSON ドキュメントを貼り付けます。これは、 AWS IoT ルールが Lambda 関数に送信する内容の例です。

```
{
  "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 通知トピックの ARN があるウィンドウを参照し、ARN 値をコピーします。

f. JSON エディターの `notify_topic_arn` 値を通知トピックの ARN に置き換えます。

AWS IoT ルールを作成するときにこの ARN 値を再度使用できるように、このウィンドウを開いたままにします。

g. [Create] を選択します。

5. サンプルデータを使用して関数をテストします。

a. format-high-temp-notification の詳細ページの右上で、SampleRuleOutput が [Test] (テスト) ボタンの横に表示されていることを確認します。表示されない場合は、利用可能なテストイベントのリストから選択します。

b. サンプルルール出力メッセージを関数に送信するには、[Test] (テスト) を選択します。

関数と通知の両方が機能した場合は、通知にサブスクライブした電話でテキストメッセージが受信されます。

電話でテキストメッセージが受信されない場合は、操作の結果を確認してください。[Function code] (関数コード) パネルの [Execution result] (実行結果) タブで、応答を確認して、発生したエラーを見つけます。関数が電話に通知を送信できるようになるまで、次のステップに進まないでください。

ステップ 2: AWS IoT ルールアクションを使用して AWS Lambda ルールを作成する

このステップでは、ルールクエリステートメントを使用して、架空の気象センサーデバイスからのデータをフォーマットして Lambda 関数に送信します。この関数は、テキストメッセージをフォーマットして送信します。

気象装置から受信したサンプルメッセージペイロードは、次のようになります。

```
{
  "temperature": 28,
```

```
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

このルールでは、ルールクエリステートメントを使用して、次のような Lambda 関数のメッセージペイロードを作成します。

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

これには、Lambda 関数が正しいテキストメッセージをフォーマットして送信するために必要なすべての情報が含まれます。

Lambda 関数を呼び出す AWS IoT ルールを作成するには

1. AWS IoT コンソールのルールハブを開きます。
2. [Rules] (ルール) で新しいルールの作成を開始するには、[Create] (作成) を選択します。
3. [Create a rule] (ルールを作成) の上部で、次のように操作します。

   a. [Name] (名前) で、ルールの名前 **wx_friendly_text** を入力します。

      ルール名は AWS アカウント および リージョン内で一意である必要があり、スペースを含めることはできません。この名前にアンダースコア文字を使用して、ルールの名前の 2 つの単語を区切りました。

   b. [Description] (説明) で、ルールの説明を入力します。

      わかりやすい説明を使用すると、このルールの動作と作成した理由を簡単に思い出すことができます。説明は必要なだけ長くすることができるので、できるだけ詳述してください。
4. [Create a rule] (ルールを作成) の [Rule query statement] (ルールクエリステートメント) で、以下を実行します。

   a. [Using SQL version] (SQL バージョンの使用) で、**2016-03-23** を選択します。

b. [Rule query statement] (ルールクエリステートメント) 編集ボックスで、ステートメントを入力します。

```
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
```

このステートメント:

- device/+/data トピックフィルターに一致し、temperature 値が 30 より大きいトピックを含む MQTT メッセージをリッスンします。
- トピック文字列から 2 番目の要素を選択し、それを 10 進数に変換してから、device_id フィールドに割り当てます。
- メッセージペイロードから temperature フィールドの値を選択し、それを reported_temperature フィールドに割り当てます。
- 制限値を表す定数値 30 を作成し、それを max_temperature フィールドに割り当てます。
- notify_topic_arn フィールドの定数値を作成します。

c. high_temp_notice 通知トピックの ARN があるウィンドウを参照し、ARN 値をコピーします。

d. ルールクエリステートメントエディタの ARN 値 (*arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice*) を通知トピックの ARN に置き換えます。

5. [Set one or more actions] (1 つ以上のアクションを設定) で、以下を実行します。

a. このルールのルールアクションのリストを開くには、[Add action] (アクションの追加) を選択します。

b. [Select an action] (アクションを選択してください) で、[Send a message to a Lambda function] (Lambda 関数にメッセージを送信する) を選択します。

c. 選択したアクションの設定ページを開くには、アクションリストの下部にある [Configure action] (アクションの設定) を選択します。

6. [Configure action] (アクションの設定) で、次のとおり実行します。

a. [Function name] (関数名) で、[Select] (選択) を選択します。

b. format-high-temp-notification を選択します。

c. [Configure action] (アクションの設定) の下部で、[Add action] (アクションの追加) を選択します。

d. ルールを作成するには、[Create a rule] (ルールの作成) の下部にある [Create rule] (ルールの作成) を選択します。

ステップ 3: AWS IoT ルールと AWS Lambda ルールアクションをテストする

新しいルールをテストするには、MQTT クライアントを使用して、このルールで使用される MQTT メッセージを発行して、これにサブスクライブします。

新しいウィンドウの AWS IoT コンソールで MQTT クライアントを開きます。これで、MQTT クライアントの設定を失うことなくルールを編集できるようになりました。コンソールの別のページに移動するために MQTT クライアントを残すと、サブスクリプションやメッセージログは失われます。

MQTT クライアントを使用してルールをテストするには

1. AWS IoT コンソールの MQTT クライアントで、入力トピック (この場合は `device/+/data`) をサブスクライブします。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Subscribe to topic] (トピックへのサブスクライブ) を選択します。

   b. [Subscription topic] (サブスクリプショントピック) で、入力トピックフィルター **`device/+/data`** のトピックを入力します。

   c. 残りのフィールドはデフォルト設定のままにします。

   d. [Subscribe to topic] を選択します。

      [Subscriptions] (サブスクリプション) 列の [Publish to a topic] (トピックへの発行) の下に **`device/+/data`** が表示されます。

2. 特定のデバイス ID **`device/32/data`** を使用して入力トピックにメッセージを発行します。ワイルドカード文字を含む MQTT トピックには発行できません。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Publish to topic] (トピックに発行) を選択します。

   b. [Publish] (発行) フィールドに、入力トピック名 **`device/32/data`** を入力します。

   c. ここに表示されているサンプルデータをコピーし、トピック名の下にある編集ボックスにサンプルデータを貼り付けます。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. MQTT メッセージを発行するには、[Publish to topic] (トピックに発行) を選択します。

3. テキストメッセージが送信されたことを確認します。

a. MQTT クライアントの [Subscriptions] (サブスクリプション) の下に、以前にサブスクライブしたトピックの隣に緑色のドットが表示されます。

緑色のドットは、最後にメッセージを表示してから 1 つ以上の新しいメッセージが受信されたことを示します。

b. [Subscriptions] (サブスクリプション) で [device/+/data] を選択して、メッセージペイロードが今発行したものと一致し、次のようになっていることを確認します。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. SNS トピックのサブスクライブに使用した電話を確認し、メッセージペイロードの内容が次のようになっていることを確認します。

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

メッセージトピックのトピック ID 要素を変更する場合、`topic(2)` 値を数値にキャストすることは、メッセージトピックのその要素に数字のみが含まれている場合にのみ機能することに注意してください。

4. 温度が制限を超えていない MQTT メッセージの送信を試みます。

   a. MQTT クライアントの [Subscription] (サブスクリプション) で、[Publish to topic] (トピックに発行) を選択します。

   b. [Publish] (発行) フィールドに、入力トピック名 **device/33/data** を入力します。

   c. ここに表示されているサンプルデータをコピーし、トピック名の下にある編集ボックスにサンプルデータを貼り付けます。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. MQTT メッセージを送信するには、[Publish to topic] (トピックに発行) を選択します。

   device/+/data サブスクリプションで送信したメッセージが表示されますが、温度値がルールクエリステートメントの最大温度を下回っているため、テキストメッセージは受信されません。

   正しい動作が確認できない場合は、トラブルシューティングのヒントを確認してください。

AWS Lambda ルールと通知のトラブルシューティング

想定する結果が得られない場合に備えて、確認すべき事項をいくつか示します。

- エラーバナーが表示された

  入力メッセージの発行時にエラーが発生した場合は、まずそのエラーを修正してください。次の手順は、このエラーを修正するのに役立つ場合があります。

- MQTT クライアントで入力メッセージが表示されない

  手順で説明されているように `device/+/data` トピックフィルターをサブスクライブした場合、入力メッセージを `device/32/data` トピックに発行するたびに、そのメッセージが MQTT クライアントに表示されます。

確認すべき事項

- サブスクライブしたトピックフィルターを確認する

  手順の説明に従って入力メッセージのトピックをサブスクライブした場合は、発行するたびに入力メッセージのコピーが表示されます。

  メッセージが表示されない場合は、サブスクライブしたトピック名を確認し、発行したトピックと比較します。トピック名は大文字と小文字が区別されます。サブスクライブしたトピックは、メッセージペイロードを発行したトピックと同一である必要があります。

- メッセージ発行機能を確認する

  MQTT クライアントの [Subscriptions] (サブスクリプション) で、[device/+/data] を選択し、パブリッシュ メッセージのトピックを確認してから、[Publish to topic] (トピックに発行) を選択します。トピックの下にある編集ボックスからメッセージペイロードがメッセージリストに表示されるのを確認できるはずです。

- SMS メッセージが届かない

  ルールが機能するには、メッセージの受信と SNS 通知の送信を許可する正しいポリシーを有しており、メッセージを受信する必要があります。

  確認すべき事項

  - AWS リージョン MQTT クライアントの と作成したルールを確認する

    MQTT クライアントを実行しているコンソールは、作成したルールと同じ AWS リージョンにある必要があります。

  - メッセージペイロードの温度値がテストしきい値を超えていることを確認します。

    ルールクエリステートメントで定義されている温度値が 30 以下の場合、ルールはそのアクションを実行しません。

  - ルールクエリステートメントの入力メッセージのトピックを確認する

    ルールが機能するためには、ルールクエリステートメントの FROM 句のトピックフィルターに一致するトピック名を持つメッセージを受信する必要があります。

    ルールクエリステートメントのトピックフィルターの綴りを、MQTT クライアントのトピックフィルターの綴りと照らし合わせて確認します。トピック名では大文字と小文字が区別され、

メッセージのトピックはルールクエリステートメントのトピックフィルターと一致する必要があります。

- 入力メッセージペイロードの内容を確認する

  ルールが機能するためには、SELECT ステートメントで宣言されているメッセージペイロード内のデータフィールドを見つける必要があります。

  ルールクエリステートメントの `temperature` フィールドの綴りを、MQTT クライアントのメッセージペイロードの綴りと照らし合わせて確認します。フィールド名では大文字と小文字が区別され、ルールクエリステートメントの `temperature` フィールドはメッセージペイロードの `temperature` フィールドと同じである必要があります。

  メッセージペイロード内の JSON ドキュメントが正しくフォーマットされていることを確認します。JSON にコンマがないなどのエラーがある場合、ルールはそれを読み取ることができません。

- Amazon SNS 通知を確認する

  ステップ 1: SMS テキストメッセージを送信する Amazon SNS トピックを作成する では、Amazon SNS 通知をテストする方法を説明するステップ 3 を参照し、通知をテストして通知が機能することを確認します。

- Lambda 関数を確認する

  ステップ 1: テキストメッセージを送信する AWS Lambda 関数を作成する では、テストデータを使用して Lambda 関数をテストする方法を説明するステップ 5 を参照し、Lambda 関数をテストします。

- ルールによって使用されているロールを確認する

  ルールのアクションには、元のトピックを受け取り、新しいトピックを発行するためのアクセス許可が必要です。

  ルールがメッセージデータを受信して再発行することを許可するポリシーは、使用されるトピックに固有です。メッセージデータの再発行に使用するトピックを変更する場合は、ルールアクションのロールを更新して、現在のトピックに一致するようにポリシーを更新する必要があります。

  これが問題であると思われる場合は、再発行ルールアクションを編集して、新しいロールを作成します。ルールアクションによって作成された新しいロールは、これらのアクションを実行するために必要な権限を受け取ります。

ステップ 4: 結果と次のステップを確認する

このチュートリアルでは、次の作業を行いました。

- カスタマイズされたメッセージペイロードを使用した Amazon SNS 通知を送信する Lambda 関数を呼び出す AWS IoT ルールを作成しました。
- ルールクエリステートメントでシンプルな SQL クエリと関数を使用して、Lambda 関数の新しいメッセージペイロードを作成しました。
- MQTT クライアントを使用して AWS IoT ルールをテストしました。

次の手順

このルールを使用していくつかのテキストメッセージを送信した後、チュートリアルの一部を変更すると、メッセージと送信される場合にどのような影響があるかを試してみてください。手始めにいくつかアイデアをご紹介します。

- 入力メッセージのトピックの *device_id* を変更し、テキストメッセージの内容に生じる影響を確認します。
- ルールクエリステートメントで選択したフィールドを変更し、Lambda 関数を更新して新しいメッセージで使用し、テキストメッセージの内容に生じる影響を確認します。
- ルールクエリステートメントのテストを変更して、最高温度ではなく最低温度をテストします。Lambda 関数を更新して新しいメッセージをフォーマットし、`max_temperature` の名前を変更することを忘れないでください。
- AWS IoT ルールの開発中および使用中に発生する可能性のあるエラーを見つける方法の詳細については、「」を参照してくださいモニタリング AWS IoT。

デバイスがオフラインになっている間にデバイスの状態をデバイスシャドウで保持する

これらのチュートリアルでは、 AWS IoT Device Shadow サービスを使用してデバイスの状態情報を保存および更新する方法を示します。JSON ドキュメントである Shadow ドキュメントは、デバイス、ローカルアプリケーション、またはサービスによって発行されたメッセージに基づいて、デバイスの状態の変化を示します。このチュートリアルでは、Shadow ドキュメントが電球の色の変化を示します。これらのチュートリアルでは、デバイスがインターネットから切断されている場合でもシャドウがこの情報を保存し、オンラインに戻ってこの情報をリクエストしたときに最新の状態情報をデバイスに返す方法も示しています。

ここに示されている順序でこれらのチュートリアルを試すことをお勧めします。この順序は、作成が必要な AWS IoT リソースと必要なハードウェアのセットアップから始まります。これは、概念を段階的に学ぶのにも役立ちます。これらのチュートリアルでは、 で使用する Raspberry Pi デバイスを設定して接続する方法を示します AWS IoT。必要なハードウェアがない場合は、選択したデバイスに適応させるか、Amazon EC2 で仮想デバイスを作成して、これらのチュートリアルに従うことができます。

チュートリアルのシナリオの概要

これらのチュートリアルのシナリオは、電球の色を変更し、そのデータを予約済みのシャドウトピックに発行するローカルアプリケーションまたはサービスです。これらのチュートリアルは、インタラクティブな開始方法のチュートリアルで説明されている Device Shadow 機能に似ており、Raspberry Pi デバイスに実装されています。このセクションのチュートリアルでは、名前付きのシャドウまたは複数のデバイスに対応する方法を示しつつ、単一のクラシックシャドウに焦点を当てます。

以下のチュートリアルは、 AWS IoT Device Shadow サービスの使用方法を学ぶのに役立ちます。

- チュートリアル: シャドウアプリケーションを実行するための Raspberry Pi の準備

  このチュートリアルでは、 に接続するための Raspberry Pi デバイスを設定する方法を示します AWS IoT。また、 AWS IoT ポリシードキュメントとモノのリソースを作成し、証明書をダウンロードして、そのモノのリソースにポリシーをアタッチします。このチュートリアルの完了には 30 分ほどかかります。

- チュートリアル: Device SDK のインストールと Device Shadows のサンプルアプリケーションの実行

  このチュートリアルでは、必要なツール、ソフトウェア、 AWS IoT Device SDK for Python をインストールし、サンプルシャドウアプリケーションを実行する方法を示します。このチュートリアルは、Raspberry Pi または他のデバイスを接続する に示されている概念に基づいており、完了までに 20 分かかります。

- チュートリアル: サンプルアプリケーションと MQTT テストクライアントを使用した Device Shadow とのやり取り

  このチュートリアルでは、shadow.pyサンプルアプリケーションとAWS IoT コンソールを使用して、 AWS IoT Device Shadows と電球の状態の変化の相互作用を観察する方法を示します。このチュートリアルでは、MQTT メッセージを Device Shadow の予約済みトピックに送信する方法も示しています。このチュートリアルの完了には 45 分間を要する場合があります。

AWS IoT Device Shadow の概要

Device Shadow は、 AWS IoT レジストリで作成したモノのリソースによって管理されるデバイスの永続的な仮想表現です。Shadow ドキュメントは、デバイスの現在の状態の情報を保存および取得するために使用される JSON または JavaScript 表記のドキュメントです。シャドウを使用すれば、デバイスがインターネットに接続されているかどうかにかかわらず、MQTT トピックまたは HTTP REST API を介してデバイスの状態を取得および設定できます。

シャドウのドキュメントには、デバイスの状態の次の側面を説明する `state` プロパティが含まれています。

- `desired`: アプリケーションは、`desired` オブジェクトを更新することによって、デバイスプロパティの必要な状態を指定します。
- `reported`: デバイスは、`reported` オブジェクト内の現在の状態を報告します。
- `delta`: `delta` オブジェクト内の目的の状態と報告された状態 AWS IoT の違いを報告します。

これは、Shadow 状態ドキュメントの例です。

```
{
  "state": {
    "desired": {
      "color": "green"
      },
    "reported": {
      "color": "blue"
      },
    "delta": {
      "color": "green"
      }
   }
}
```

デバイスの Shadow ドキュメントを更新するには、予約済みの MQTT トピック、HTTP で `GET`、`UPDATE`、および `DELETE` オペレーションをサポートする Device Shadow REST API、ならびに AWS IoT CLI を使用できます。

前の例で、`desired` 色を `yellow` に変更したいとします。これを行うには、UpdateThingShadow API にリクエストを送信するか、Update トピック `$aws/things/THING_NAME/shadow/update` にメッセージを発行します。

```
{
  "state": {
    "desired": {
      "color": yellow
    }
  }
}
```

更新は、リクエストで指定したフィールドにのみ反映されます。Device Shadow が正常に更新されると、 はdeltaトピック に新しいdesired状態 AWS IoT を発行します$aws/things/THING_NAME/shadow/delta。この場合の Shadow ドキュメントは次のようになります。

```
{
  "state": {
    "desired": {
      "color": yellow
    },
    "reported": {
      "color": green
    },
    "delta": {
      "color": yellow
      }
  }
}
```

新しい状態は、次の JSON メッセージ$aws/things/THING_NAME/shadow/updateを含むUpdateトピックを使用して AWS IoT Device Shadow に報告されます。

```
{
  "state": {
    "reported": {
      "color": yellow
    }
  }
}
```

現在の状態情報を取得する場合は、GetThingShadow API にリクエストを送信するか、MQTT メッセージを Get トピック $aws/things/THING_NAME/shadow/get に発行します。

Device Shadow サービスの使用の詳細については、「AWS IoT Device Shadow サービス」を参照してください。

デバイス、アプリケーション、およびサービスでの Device Shadow の使用の詳細については、デバイスでのシャドウの使用 および アプリとサービスでのシャドウの使用 を参照してください。

AWS IoT シャドウの操作については、「」を参照してくださいシャドウとの相互作用。

MQTT の予約済みトピックおよび HTTP REST API については、Device Shadow MQTT トピック および Device Shadow の REST API を参照してください。

チュートリアル: シャドウアプリケーションを実行するための Raspberry Pi の準備

このチュートリアルでは、Raspberry Pi デバイスをセットアップして設定し、デバイスが MQTT メッセージを接続および交換するために必要な AWS IoT リソースを作成する方法を示します。

> **Note**
>
> the section called “Amazon EC2 を使用して仮想デバイスを作成する” する予定がある場合は、このページをスキップして the section called “デバイスを設定する” に進むことができます。仮想のモノを作成するときに、これらのリソースを作成します。Raspberry Pi の代わりに別のデバイスを使用したい場合は、選択したデバイスに合わせてこれらのチュートリアルに従ってみてください。

このチュートリアルの学習内容は次のとおりです。

- Raspberry Pi デバイスを設定し、 で使用するように設定します AWS IoT。
- デバイスが サービスとやり取り AWS IoT することを許可する AWS IoT ポリシードキュメントを作成します。
- X.509 デバイス証明書にモノ AWS IoT のリソースを作成し、ポリシードキュメントをアタッチします。

  モノは、 AWS IoT レジストリ内のデバイスの仮想表現です。証明書はデバイスを AWS IoT Core に認証し、ポリシードキュメントはデバイスを操作することを許可します AWS IoT。

このチュートリアルを実行する方法

Device Shadows の `shadow.py` サンプルアプリケーションを実行するには、 AWS IoTに接続する Raspberry Pi デバイスが必要です。ここに示されている順序でこのチュートリアルに従うことをお

勧めします。この順序では、最初に Raspberry Pi とその付属内容を設定してから、ポリシーを作成し、作成したモノのリソースにポリシーをアタッチします。その後、Raspberry Pi でサポートされているグラフィカルユーザーインターフェイス (GUI) を使用してデバイスのウェブブラウザで AWS IoT コンソールを開くことで、このチュートリアルに従うことができます。これにより、接続のために証明書を Raspberry Pi に直接ダウンロードしやすくなります AWS IoT。

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- AWS アカウント。アカウントをお持ちではない場合、続行する前に、セットアップ AWS アカウント に記載されている手順を完了してください。このチュートリアルを完了するには、 AWS アカウント と AWS IoT コンソールが必要です。
- Raspberry Pi とその必要な付属内容。次が必要になります。
  - Raspberry Pi 3 モデル B 以降の最新のモデル。このチュートリアルは、Raspberry Pi の以前のバージョンでも機能する可能性がありますが、テストはしていません。
  - Raspberry Pi OS (32 ビット)または、それ以降。Raspberry Pi OS の最新バージョンを使用することをお勧めします。以前のバージョンの OS でも動作する可能性がありますが、テストはしていません。
  - イーサネットまたは Wi-Fi 接続。
  - キーボード、マウス、モニタ、ケーブル、および電源。

このチュートリアルの完了には 30 分ほどかかります。

ステップ 1: Raspberry Pi デバイスをセットアップおよび設定する

このセクションでは、 で使用する Raspberry Pi デバイスを設定します AWS IoT。

> **Important**
>
> これらの指示を他のデバイスやオペレーティングシステムに合わせて適用するのが難しい場合があります。これらの指示を解釈してご利用のデバイスに適用するには、そのデバイスを十分に理解する必要があります。問題が発生した場合は、Amazon EC2 を使用して仮想デバイスを作成する または Windows または Linux の PC または Mac を AWS IoT デバイスとして使用する など、他のデバイスオプションのいずれかを代替策として試してください。

Raspberry Pi を設定して、オペレーティングシステム (OS) を起動し、インターネットに接続し、コマンドラインインターフェイスで Raspberry Pi とインタラクションできるようにする必要がありま

す。Raspberry Pi でサポートされているグラフィカルユーザーインターフェイス (GUI) を使用して AWS IoT コンソールを開き、このチュートリアルの残りの部分を実行することもできます。

Raspberry Pi をセットアップするには

1. SD カードを Raspberry Pi の MicroSD カードスロットに挿入します。一部の SD カードには、ボードの起動後に OS をインストールするためのメニューを表示するインストールマネージャーがプリロードされています。Raspberry Pi イメージャーを使用して、カードに OS をインストールすることもできます。
2. Raspberry Pi の HDMI ポートに接続する HDMI ケーブルに、HDMI TV またはモニターを接続します。
3. キーボードとマウスを Raspberry Pi の USB ポートに接続し、電源アダプタをつないでボードを起動します。

Raspberry Pi の起動後、SD カードにインストールマネージャーがプリロードされている場合は、オペレーティングシステムをインストールするためのメニューが表示されます。OS のインストールに問題がある場合は、次の手順を試すことができます。Raspberry Pi のセットアップの詳細については、「Raspberry Pi のセットアップ」を参照してください。

Raspberry Pi のセットアップで問題が発生している場合は、以下の操作を実行します。

- ボードを起動する前に、SD カードを挿入したかどうかを確認してください。ボードを起動した後に SD カードを差し込むと、インストールメニューが表示されない場合があります。
- テレビまたはモニタの電源が入っていて、正しい入力が選択されていることを確認してください。
- Raspberry Pi と互換性のあるソフトウェアを使用していることを確認します。

Raspberry Pi OS をインストールして設定したら、Raspberry Pi のウェブブラウザを開き、 AWS IoT Core コンソールに移動して、このチュートリアルの残りのステップを続行します。

AWS IoT Core コンソールを開くことができる場合は、Raspberry Pi の準備が整い、 に進むことができますthe section called “でのデバイスのプロビジョニング AWS IoT ”。

問題がある場合や、さらにサポートが必要な場合は、「Getting help for your Raspberry Pi」を参照してください。

チュートリアル: でのデバイスのプロビジョニング AWS IoT

このセクションでは、チュートリアルで使用する AWS IoT Core リソースを作成します。

でデバイスをプロビジョニングする手順 AWS IoT

- ステップ 1: Device Shadow の AWS IoT ポリシーを作成する
- ステップ 2: モノのリソースを作成し、ポリシーをモノにアタッチする
- ステップ 3: 結果と次のステップを確認する

ステップ 1: Device Shadow の AWS IoT ポリシーを作成する

X.509 証明書 AWS IoT Core AWS IoT は、デバイスが Device Shadow サービスで使用される MQTT 予約トピックへのサブスクライブや発行などの AWS IoT オペレーションを実行できるようにする証明書にアタッチされます。デバイスは、接続してメッセージを送信するときに証明書を提示します AWS IoT Core。

この手順では、サンプルプログラムの実行に必要な AWS IoT オペレーションをデバイスが実行できるようにするポリシーを作成します。タスクの実行に必要なアクセス許可のみを付与するポリシーを作成することをお勧めします。最初に AWS IoT ポリシーを作成し、後で作成するデバイス証明書にアタッチします。

AWS IoT ポリシーを作成するには

1. 左のメニューで、[Secure] (安全性)、[Policies] (ポリシー) の順に選択します。アカウントに既存のポリシーがある場合は、[作成] を選択します。それ以外の場合は、[ポリシーがまだありません] ページで [ポリシーの作成] を選択します。
2. [ポリシーの作成] ページで、以下の手順を実行します。

   a. [名前] フィールドに、ポリシーの名前 (**My_Device_Shadow_policy** など) を入力します。ポリシー名で個人を特定できる情報を使用しないでください。

   b. ポリシードキュメントでは、MQTT 予約トピックを発行およびサブスクライブするアクセス許可をデバイスに付与する接続、サブスクライブ、受信、および発行アクションについて説明します。

      以下のサンプルポリシーをコピーして、ポリシードキュメントに貼り付けます。`thingname` を、作成するモノの名前 (などMy_light_bulb)、 をサービスを使用しているregion AWS IoT リージョン、 をお客様の AWS アカウント 番号accountに置き換えます。 AWS IoT ポリシーの詳細については、「」を参照してくださいAWS IoT Core ポリシー。

```
{
```

```
    "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-*"
        }
    ]
}
```

ステップ 2: モノのリソースを作成し、ポリシーをモノにアタッチする

に接続されたデバイスは、 AWS IoT レジストリ内のモノのリソースで表す AWS IoT ことができます。モノのリソースは、このチュートリアルの電球など、特定のデバイスまたは論理エンティティを表します。

でモノを作成する方法については AWS IoT、「」で説明されている手順に従ってくださいモノのオブジェクトを作成する。そのチュートリアルの手順に従うときに注意すべき重要な点がいくつかあります。

1. [単一のモノを作成する] を選択し、[名前] フィールドに、以前にポリシーを作成したときに指定した thingname と同じモノの名前 (例: My_light_bulb) を入力します。

   モノを作成した後に名前を変更することはできません。thingname 以外の名前を付けた場合は、thingname という名前の新しいモノを作成し、古いモノを削除します。

   > **Note**
   >
   > モノの名前で個人を特定できる情報を使用しないでください。モノの名前は、暗号化されていない通信やレポートに表示されることがあります。

2. [証明書が作成されました!] ページの各証明書ファイルを見つけやすい場所にダウンロードすることをお勧めします。サンプルアプリケーションを実行するには、これらのファイルをインストールする必要があります。

Raspberry Pi の home ディレクトリ内の `certs` サブディレクトリにファイルをダウンロードし、次の表に示されているように、各ファイルに簡素な名前を付けることをお勧めします。

証明書ファイル名

ファイル	ファイルパス
ルート CA 証明書	`~/certs/Amazon-root-CA-1.pem`
デバイス証明書	`~/certs/device.pem.crt`
プライベートキー	`~/certs/private.pem.key`

3. 証明書を有効にして接続を有効にしたら AWS IoT、ポリシーをアタッチを選択し、以前に作成したポリシー (など**`My_Device_Shadow_policy`**) をモノにアタッチしていることを確認します。

   モノを作成したら、モノのリソースが AWS IoT コンソールのモノのリストに表示されます。

ステップ 3: 結果と次のステップを確認する

このチュートリアルで学習した内容は次のとおりです。

- Raspberry Pi デバイスをセットアップおよび設定します。
- デバイスが サービスとやり取り AWS IoT することを許可する AWS IoT ポリシードキュメントを作成します。
- モノのリソースと関連する X.509 デバイス証明書を作成し、それにポリシードキュメントをアタッチします。

次の手順

AWS IoT デバイス SDK for Python をインストールし、`shadow.py`サンプルアプリケーションを実行し、Device Shadows を使用して状態を制御できるようになりました。このチュートリアルの実行方法の詳細については、チュートリアル: Device SDK のインストールと Device Shadows のサンプルアプリケーションの実行 を参照してください。

チュートリアル: Device SDK のインストールと Device Shadows のサンプルアプリケーションの実行

このセクションでは、必要なソフトウェアと AWS IoT Device SDK for Python をインストールし、`shadow.py`サンプルアプリケーションを実行して Shadow ドキュメントを編集し、シャドウの状態を制御する方法について説明します。

このチュートリアルの学習内容は次のとおりです。

- インストールされているソフトウェアと AWS IoT Device SDK for Python を使用して、サンプルアプリケーションを実行します。
- サンプルアプリケーションを使用して値を入力すると、どのように AWS IoT コンソールで目的の値が発行されるかについて説明します。
- `shadow.py` サンプルアプリケーションと、MQTT プロトコルを使用してシャドウの状態を更新する方法を確認してください。

このチュートリアルを実行する前に:

をセットアップし AWS アカウント、Raspberry Pi デバイスを設定し、Device Shadow サービスの MQTT 予約トピックを発行およびサブスクライブするアクセス許可をデバイスに付与する AWS IoT モノとポリシーを作成しておく必要があります。詳細については、「チュートリアル: シャドウアプリケーションを実行するための Raspberry Pi の準備」を参照してください。

Git、Python、および AWS IoT Device SDK for Python もインストールされている必要があります。このチュートリアルは、チュートリアル Raspberry Pi または他のデバイスを接続する で提示された概念に基づいています。そのチュートリアルをまだ試していない場合は、そのチュートリアルで説明されている手順に従って証明書ファイルと Device SDK をインストールしてから、このチュートリアルに戻って `shadow.py` サンプルアプリケーションを実行することをお勧めします。

このチュートリアルでは、次の作業を行います。

- ステップ 1: shadow.py サンプルアプリケーションを実行する
- ステップ 2: shadow.py Device SDK サンプルアプリケーションを確認する
- ステップ 3: shadow.py サンプルアプリケーションの問題をトラブルシューティングする
- ステップ 4: 結果と次のステップを確認する

このチュートリアルの完了には 20 分ほどかかります。

ステップ 1: shadow.py サンプルアプリケーションを実行する

shadow.py サンプルアプリケーションを実行する前に、インストールした証明書ファイルの名前と場所に加えて、次の情報が必要です。

アプリケーションパラメータ値

Parameter	値がある場所
your-iot-thing-name	で前に作成した AWS IoT モノの名前the section called “ステップ 2: モノのリソースを作成し、ポリシーをモノにアタッチする”。 この値を見つけるには、AWS IoT コンソールで [Manage] (管理) を選択し、[Things] (モノ) を選択します。
your-iot-endpoint	*your-iot-endpoint* 値の形式は *endpoint_id* -ats.iot. *region*.amazonaws.com です (例: a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com)。この値を検索するには: 1. AWS IoT コンソールで、[Manage] (管理)、[Things] (モノ) の順に選択します。 2. デバイス用に作成した IoT モノである My_light_bulb (以前に使用しました) を選択してから、[Interact] (操作) を選択します。モノの詳細ページで エンドポイントは、[HTTPS] セクションに表示されます。

サンプルアプリケーションをインストールして実行する

1. サンプルアプリディレクトリに移動します。

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

2. コマンドラインウィンドウで、示されているように *your-iot-endpoint* と *your-iot-thing-name* を置き換えて、このコマンドを実行します。

```
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. サンプルアプリケーションが次のようになっていることを観察します。

   1. アカウントの AWS IoT サービスに接続します。
   2. Delta イベントと Update および Get 応答をサブスクライブします。
   3. ターミナルに必要な値を入力するように求められます。
   4. 次のような出力を表示します。

```
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 サンプルアプリケーションの実行に問題がある場合は、the section called “ステップ 3: shadow.py サンプルアプリケーションの問題をトラブルシューティングする” を確認してください。問題の修正に役立つ可能性のある追加情報を取得するには、コマンドラインに --verbosity debug パラメータを追加して、サンプルアプリケーションが実行内容に関する詳細メッセージを表示するようにします。

値を入力し、Shadow ドキュメントの更新を観察する

ターミナルに desired 値を入力して値を指定すると、reported 値も更新されます。ターミナルで色 yellow を入力するとします。reported 値も色 yellow に更新されます。ターミナルに表示されるメッセージを次に示します。

```
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'.
```

この更新リクエストを発行すると、 はモノのリソースのデフォルトクラシックシャドウ AWS IoT を作成します。 AWS IoT コンソールで reportedおよび desired値に発行した更新リクエストを確認するには、作成したモノのリソースの Shadow ドキュメントを確認します (例: My_light_bulb)。Shadow ドキュメントで更新を表示するには、次の操作を行います。

1. AWS IoT コンソールで、「管理」を選択し、「モノ」を選択します。
2. 表示されるモノのリストで、作成したモノを選択し、[Shadows] (シャドウ) を選択してから、[Classic Shadow] (クラシックシャドウ) を選択します。

Shadow ドキュメントは次のようになり、色 yellow に設定された reported と desired の値が表示されます。これらの値は、ドキュメントの [Shadow の状態] のセクションに表示されます。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "yellow"
},
"reported": {
  "welcome": "aws-iot",
  "color": "yellow"
}
}
```

リクエストのタイムスタンプ情報とバージョン番号を含む [メタデータ] セクションも表示されます。

状態ドキュメントのバージョンを使用して、デバイスのシャドウのドキュメントの最新バージョンを更新していることを確認できます。別の更新リクエストを送信すると、バージョン番号が 1 ずつ増えます。更新リクエストでバージョンを渡したとき、そのバージョンと状態ドキュメントの現在の

バージョンとが一致しない場合、サービスは HTTP 409 conflict レスポンスコードでリクエストを拒否します。

```
{
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  }
},
"version": 10
}
```

Shadow ドキュメントについてさらに学び、状態情報の変化を観察するには、このチュートリアルのステップ 4: 結果と次のステップを確認する セクションで説明されている次のチュートリアル チュートリアル: サンプルアプリケーションと MQTT テストクライアントを使用した Device Shadow とのやり取り に進んでください。必要に応じて、次のセクションで shadow.py サンプルコードとそれが MQTT プロトコルを使用する方法について学ぶこともできます。

ステップ 2: shadow.py Device SDK サンプルアプリケーションを確認する

このセクションでは、このチュートリアルで使用されている AWS IoT Device SDK v2 for Python の shadow.py サンプルアプリケーションを確認します。ここでは、MQTT および MQTT over WSS プロトコル AWS IoT Core を使用して に接続する方法を確認します。AWS 共通ランタイム (AWS-CRT) ライブラリは、低レベルの通信プロトコルサポートを提供し、 AWS IoT Device SDK v2 for Python に含まれています。

このチュートリアルでは MQTT と MQTT over WSS を使用していますが、 は HTTPS リクエストを発行するデバイス AWS IoT をサポートしています。デバイスから HTTP メッセージを送信する

Python プログラムの例については、Python の `requests` ライブラリを使用した HTTPS コード例を参照してください。

デバイス通信に使用するプロトコルについて十分な情報に基づいた決定を行う方法については、デバイス通信用のアプリケーションプロトコルの選択 を確認してください。

MQTT

`shadow.py` サンプルでは、`mqtt_connection_builder` の `mtls_from_path` (以下に示されています) を呼び出して、MQTT プロトコルを使用して AWS IoT Core との接続を確立します。`mtls_from_path` では、X.509 証明書と TLS v1.2 を使用してデバイスを認証します。 AWS CRT ライブラリは、その接続の下位の詳細を処理します。

```
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` はコマンドラインから渡した AWS IoT エンドポイントであり、 でこのデバイスを一意に識別する ID `client_id`です AWS リージョン。
- `cert_filepath`、`pri_key_filepath`、および `ca_filepath` は、デバイスの証明書とプライベートキーファイル、およびルート CA ファイルへのパスです。
- `client_bootstrap` は、ソケット通信アクティビティを処理する共通のランタイムオブジェクトであり、`mqtt_connection_builder.mtls_from_path` の呼び出しの前にインスタンス化されます。
- `on_connection_interrupted` および `on_connection_resumed` は、デバイスの接続が中断されて再開されるときに呼び出すコールバック関数です。
- `clean_session` は、新しい永続セッションを開始するか、既存のセッションに再接続するか (存在する場合) です。`keep_alive_secs` は CONNECT リクエストで送信するキープアライブ値 (秒単位) です。この間隔で ping が自動的に送信されます。サーバーは、この値の 1.5 倍の時間が経過しても ping を受信しなかった場合、接続が失われたとみなします。

この shadow.py サンプルでは、WSS 経由で MQTT プロトコルを使用して AWS IoT Core との接続を確立するために mqtt_connection_builder の websockets_with_default_aws_signing も呼び出します。MQTT over WSS も MQTT と同じパラメータを使用し、次の追加パラメータを受け取ります。

- region は、署名 V4 認証で使用される AWS 署名リージョンであり、認証に使用するために提供される AWS 認証情報credentials_providerです。リージョンはコマンドラインから渡され、credentials_provider オブジェクトは mqtt_connection_builder.websockets_with_default_aws_signing の呼び出しの直前にインスタンス化されます。
- websocket_proxy_options は、プロキシホストを使用する場合の HTTP プロキシオプションです。shadow.py サンプルアプリケーションでは、この値は mqtt_connection_builder.websockets_with_default_aws_signing の呼び出しの直前にインスタンス化されます。

Shadow トピックとイベントをサブスクライブする

shadow.py サンプルは接続の確立を試み、完全に接続されるまで待機します。接続されていない場合、コマンドはキューに入れられます。接続されると、サンプルはデルタイベントをサブスクライブし、メッセージを更新および取得し、サービスの品質 (QoS) レベル 1 (mqtt.QoS.AT_LEAST_ONCE) でメッセージを発行します。

デバイスが QoS レベル 1 のメッセージをサブスクライブすると、メッセージブローカーは、デバイスに送信できるようになるまで、デバイスがサブスクライブしているメッセージを保存します。メッセージブローカーは、デバイスから PUBACK 応答を受信するまでメッセージを再送信します。

MQTT プロトコルの詳細については、MQTT プロトコルを確認する および MQTT を参照してください。

このチュートリアルで使用される MQTT、MQTT over WSS、永続セッション、および QoS レベルの詳細については、「pubsub.py Device SDK サンプルアプリケーションを確認する」を参照してください。

ステップ 3: **shadow.py** サンプルアプリケーションの問題をトラブルシューティングする

shadow.py サンプルアプリケーションを実行すると、ターミナルにいくつかのメッセージが表示され、desired 値を入力するように求められます。プログラムがエラーをスローした場合、エラーをデバッグするには、まずシステムに対して正しいコマンドを実行したかどうかを確認します。

場合によっては、エラーメッセージは接続の問題を示し、`Host name was invalid for dns resolution` または `Connection was closed unexpectedly` のようになることがあります。このような場合、次のことを確認してみてください。

- コマンド内のエンドポイントアドレスを確認する

  サンプルアプリケーションを実行するために入力したコマンドの `endpoint` 引数 (例: `a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`) を確認し、AWS IoT コンソールでこの値を確認します。

  正しい値を使用したかどうかを確認するには、次の手順に従います。

  1. AWS IoT コンソールで、[Manage] (管理)、[Things] (モノ) の順に選択します。
  2. サンプルアプリケーション用に作成したモノ (例: My_light_bulb) を選択してから、[Interact] (操作) を選択します。

  モノの詳細ページで エンドポイントは、[HTTPS] セクションに表示されます。また、以下を記載したメッセージも表示されます: `This thing already appears to be connected.`。

- 証明書の有効化を確認する

  証明書は、 を使用してデバイスを認証します AWS IoT Core。

  証明書がアクティブかどうかを確認するには、次の手順に従います。

  1. AWS IoT コンソールで、[Manage] (管理)、[Things] (モノ) の順に選択します。
  2. サンプルアプリケーション用に作成したモノ (例: My_light_bulb) を選択してから、[Security] (セキュリティ) を選択します。
  3. 証明書を選択し、証明書の詳細ページで [証明書の選択] を選択してから、証明書の詳細ページで [アクション] を選択します。

  ドロップダウンリストで [有効化] が使用できず、[無効化] しか選択できない場合、証明書は有効です。そうでない場合は、[有効化] を選択し、サンプルプログラムを再実行します。

  それでもプログラムが実行されない場合は、`certs` フォルダ内の証明書ファイル名を確認してください。

- モノのリソースにアタッチされているポリシーを確認する

  証明書がデバイスを認証している間、 AWS IoT ポリシーにより、デバイスは MQTT 予約済みトピックへのサブスクライブや発行などの AWS IoT オペレーションを実行できます。

正しいポリシーがアタッチされているかどうかを確認するには、次の手順に従います。

1. 前述したとおりに証明書を検索し、[ポリシー] を選択します。
2. 表示されたポリシーを選択し、デバイスに MQTT 予約トピックへ発行およびサブスクライブするためのアクセス許可を付与する `connect`、`subscribe`、`receive`、および `publish` アクションが説明されているかどうかを確認します。

   サンプルポリシーについては、ステップ 1: Device Shadow の AWS IoT ポリシーを作成する を参照してください。

への接続に問題があることを示すエラーメッセージが表示された場合は AWS IoT、ポリシーに使用しているアクセス許可が原因である可能性があります。その場合は、 AWS IoT リソースへのフルアクセスを提供するポリシーから始めて、サンプルプログラムを再実行することをお勧めします。現在のポリシーを編集するか、現在のポリシーを選択して [Detach] (デタッチ) を選択し、フルアクセスを提供する別のポリシーを作成してモノのリソースにアタッチすることができます。後で、プログラムの実行に必要なアクションとポリシーのみにポリシーを制限できます。

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

- Device SDK のインストールを確認する

  それでもプログラムが実行されない場合は、Device SDK を再インストールして、SDK のインストールが完了していて正しいことを確認できます。

ステップ 4: 結果と次のステップを確認する

このチュートリアルで学習した内容は次のとおりです。

- 必要なソフトウェア、ツール、および AWS IoT Device SDK for Python をインストールします。

- サンプルアプリケーション shadow.py が、シャドウの現在の状態を取得および更新するために MQTT プロトコルを使用する方法を理解します。
- Device Shadows のサンプルアプリケーションを実行し、 AWS IoT コンソールで Shadow ドキュメントの更新を確認します。また、プログラムの実行時に問題をトラブルシューティングし、エラーを修正する方法も学びました。

次の手順

これで、shadow.py サンプルアプリケーションを実行し、Device Shadow を使用して状態を制御できます。 AWS IoT コンソールで Shadow ドキュメントの更新を観察し、サンプルアプリケーションが応答するデルタイベントを観察できます。MQTT テストクライアントを使用して、予約済みのシャドウトピックにサブスクライブし、サンプルプログラムの実行時にトピックが受信するメッセージを観察できます。このチュートリアルの実行方法の詳細については、チュートリアル: サンプルアプリケーションと MQTT テストクライアントを使用した Device Shadow とのやり取り を参照してください。

チュートリアル: サンプルアプリケーションと MQTT テストクライアントを使用した Device Shadow とのやり取り

shadow.py サンプルアプリケーションを操作するには、desired の値のためにターミナルに値を入力します。たとえば、トラフィックライトに似た色を指定してリクエストに AWS IoT 応答し、報告された値を更新できます。

このチュートリアルの学習内容は次のとおりです。

- shadow.py サンプルアプリケーションを使用して、必要な状態を指定し、シャドウの現在の状態を更新します。
- Shadow ドキュメントを編集して、デルタイベントと、shadow.py サンプルアプリケーションがそれにどのように応答するかを観察します。
- MQTT テストクライアントを使用して、シャドウトピックをサブスクライブし、サンプルプログラムを実行するときに更新を確認します。

このチュートリアルを実行する前に、以下の条件を満たす必要があります。

をセットアップし AWS アカウント、Raspberry Pi デバイスを設定し、モノとポリシーを作成 AWS IoT しました。また、必要なソフトウェア、Device SDK、証明書ファイルをインストールし、ターミナルでサンプルプログラムを実行する必要があります。詳細については、前のチュートリアルの

チュートリアル: シャドウアプリケーションを実行するための Raspberry Pi の準備 および ステップ 1: shadow.py サンプルアプリケーションを実行する を参照してください。これらのチュートリアルをまだ完了していない場合は、完了する必要があります。

このチュートリアルでは、次の作業を行います。

- ステップ 1: shadow.py サンプルアプリケーションを使用して、必要な値と報告された値を更新する
- ステップ 2: MQTT テストクライアントで shadow.py サンプルアプリケーションからのメッセージを表示する
- ステップ 3: Device Shadow インタラクションに関するエラーをトラブルシューティングする
- ステップ 4: 結果と次のステップを確認する

このチュートリアルの完了には 45 分ほどかかります。

ステップ 1: **shadow.py** サンプルアプリケーションを使用して、必要な値と報告された値を更新する

前のチュートリアル「」ではステップ 1: shadow.py サンプルアプリケーションを実行する、セクション「」で説明されているように目的の値を入力すると、 AWS IoT コンソールで Shadow ドキュメントに発行されたメッセージを確認する方法を学習しましたチュートリアル: Device SDK のインストールと Device Shadows のサンプルアプリケーションの実行。

前の例では、目的の色を `yellow` に設定しました。各値を入力すると、ターミナルは別の `desired` 値を入力するように求めます。同じ値 (`yellow`) をもう一度入力すると、アプリケーションはこれを認識し、新しい `desired` 値を入力するように求めます。

```
Enter desired value:
yellow
Local value is already 'yellow'.
Enter desired value:
```

ここで、 という色を入力したとしますgreen。 AWS IoT はリクエストに対応し、`reported`値を に更新しますgreen。これは、`desired` 状態が `reported` 状態と異なる場合に更新が行われる方法であり、デルタが発生します。

`shadow.py` サンプルアプリケーションが Device Shadow インタラクションをシミュレートする方法:

1. ターミナルに `desired` 値 (例: `yellow`) を入力して、目的の状態を発行します。

2. desired 状態が reported 状態 (色 green など) と異なるため、デルタが発生し、デルタにサブスクライブしているアプリケーションがこのメッセージを受信します。
3. アプリケーションはメッセージに応答し、その状態を desired 値 yellow に更新します。
4. その後、アプリケーションはデバイスの状態 yellow の新しい報告値を含む更新メッセージを発行します。

以下は、更新リクエストがどのように発行されるかを示す、ターミナルに表示されるメッセージを示しています。

```
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 コンソールでは、Shadow ドキュメントは reportedフィールドと desiredフィールドgreenの両方で に更新された値を反映し、バージョン番号は 1 ずつ増加します。例えば、以前のバージョン番号が 10 と表示されていた場合、現在のバージョン番号は 11 と表示されます。

> **Note**
>
> シャドウを削除しても、バージョン番号は 0 にリセットされません。更新リクエストを発行するか、同じ名前で別のシャドウを作成すると、シャドウバージョンが 1 ずつ増加することがわかります。

Shadow ドキュメントを編集してデルタイベントを観察する

shadow.py サンプルアプリケーションも delta イベントにサブスクライブしており、desired 値が変更されると応答します。例えば、desired 値を色 red に変更できます。これを行うには、AWS IoT コンソールで、編集をクリックして Shadow ドキュメントを編集し、desired値を に設定しながら JSON redでreported値を に設定しますgreen。変更を保存する際には Raspberry Pi のターミナルを開いたままにしておきます。変更が発生するとターミナルにメッセージが表示されます。

```
{
```

```
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

新しい値を保存すると、shadow.py サンプルアプリケーションはこの変更に応答し、デルタを示すメッセージをターミナルに表示します。desired 値を入力するためのプロンプトの下に、次のメッセージが表示されます。

```
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'.
```

ステップ 2: MQTT テストクライアントで **shadow.py** サンプルアプリケーションからのメッセージを表示する

AWS IoT コンソールで MQTT テストクライアントを使用して、 AWS アカウントで渡される MQTT メッセージをモニタリングできます。Device Shadow サービスで使用される予約済みの MQTT トピックにサブスクライブすることで、サンプルアプリケーションの実行時にトピックが受信するメッセージを観察できます。

MQTT テストクライアントをまだ使用していない場合は、MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する を確認できます。これは、AWS IoT コンソールで MQTT テストクライアントを使用して、メッセージブローカーを通過する MQTT メッセージを表示する方法を学ぶのに役立ちます。

1. MQTT テストクライアントを開く

   AWS IoT コンソールの新しいウィンドウで MQTT テストクライアントを開きます。これにより、MQTT テストクライアントの設定を失うことなく、MQTT トピックによって受信された

メッセージを観察できます。MQTT テストクライアントからコンソールの別のページに移動すると、サブスクリプションまたはメッセージログは保持されません。チュートリアルのこのセクションでは、 AWS IoT モノの Shadow ドキュメントと MQTT テストクライアントを別々のウィンドウで開いて、Device Shadows とのやり取りをより簡単に観察できます。

2. MQTT 予約済み Shadow トピックをサブスクライブする

   MQTT テストクライアントを使用して、Device Shadow の MQTT 予約済みトピックの名前を入力し、サブスクライブして、`shadow.py` サンプルアプリケーションの実行時に更新を受け取ることができます。トピックにサブスクライブするには:

   a. AWS IoT コンソールの MQTT テストクライアントで、[Subscribe to a topic] (トピックへサブスクライブする) を選択します。

   b. [Topic filter] (トピックフィルター) のセクションで、$aws/things/***thingname***/shadow/update/# と入力します。ここで、`thingname` は以前に作成したモノのリソースの名前です (例: `My_light_bulb`)。

   c. 追加の構成設定はデフォルト値のままにして、[サブスクライブ] を選択します。

   トピックサブスクリプションで # ワイルドカードを使用すると、複数の MQTT トピックを同時にサブスクライブし、デバイスとその Shadow の間で交換されるすべてのメッセージを単一のウィンドウで観察できます。ワイルドカード文字とその使用の詳細については、「MQTT トピック」を参照してください。

3. **`shadow.py` サンプルプログラムを実行してメッセージを観察する**

   Raspberry Pi のコマンドラインウィンドウで、プログラムを切断した場合は、サンプルアプリケーションを再度実行し、AWS IoT コンソールの MQTT テストクライアントのメッセージを確認します。

   a. 次のコマンドを実行して、サンプルプログラムを再起動します。*`your-iot-thing-name`* と *`your-iot-endpoint`* を、以前に作成した AWS IoT モノの名前 (など`My_light_bulb`) と、デバイスを操作するエンドポイントに置き換えます。

   ```
   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
   ```

その後、shadow.py サンプルアプリケーションが実行され、現在のシャドウ状態が取得されます。シャドウを削除したか、現在の状態をクリアした場合、プログラムは現在の値を off に設定し、desired 値を入力するように求めます。

```
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:
```

一方、プログラムが実行されていて、それを再起動した場合、ターミナルで最新の色の値が報告されるのを確認できます。MQTT テストクライアントでは、トピック $aws/things/***thingname***/shadow/get および $aws/things/***thingname***/shadow/get/accepted の更新が表示されます。

報告された最新の色が green であるとします。以下は、$aws/things/***thingname***/shadow/get/accepted JSON ファイルの内容を示しています。

```
{
"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. yellow など、ターミナルに desired 値を入力します。shadow.py サンプルアプリケーションは応答し、reported 値の yellow への変更を示す次のメッセージをターミナルに表示します。

```
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 コンソールの [MQTT テストクライアント] の [サブスクリプション] で、次のトピックがメッセージを受信したことがわかります。

- $aws/things/***thingname***/shadow/update: desired と updated 値の両方が色 yellow に変化することを示しています。
- $aws/things/***thingname***/shadow/update/accepted: desired および reported 状態の現在の値、およびそれらのメタデータとバージョン情報を表示します。
- $aws/things/***thingname***/shadow/update/documents: desired および reported 状態の以前の値と現在の値、およびそれらのメタデータとバージョン情報を表示します。

ドキュメント $aws/things/***thingname***/shadow/update/documents にも他の 2 つのトピックに含まれる情報が含まれているため、これで状態情報を確認できます。以前の状態は、green に設定された報告された値、そのメタデータとバージョン情報、および yellow に更新された、報告された値を示す現在の状態を示しています。

```
{
"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. ここで、別の desired 値を入力すると、reported 値のさらなる変更と、これらのトピックで受信したメッセージの更新が表示されます。バージョン番号も 1 ずつ増分します。例えば、値 green を入力すると、以前の状態が値 yellow を報告し、現在の状態が値 green を報告します。

4. Shadow ドキュメントを編集してデルタイベントを観察する

デルタトピックへの変更を観察するには、 AWS IoT コンソールで Shadow ドキュメントを編集します。例えば、desired 値を色 red に変更できます。これを行うには、 AWS IoT コンソールで編集を選択し、desired値を に設定したまま JSON でreported値を赤に設定しますgreen。変更を保存する際はターミナルを開いたままにしておいてください。ターミナルに報告されたデルタメッセージが表示されます。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

shadow.py サンプルアプリケーションはこの変更に応答し、デルタを示すメッセージをターミナルに表示します。MQTT テストクライアントでは、update トピックは、desired および reported 値の変更を示すメッセージを受け取ります。

また、トピック $aws/things/***thingname***/shadow/update/delta がメッセージを受信したこともわかります。メッセージを表示するには、[サブスクリプション] の下にリストされているこのトピックを選択します。

```
{
"version": 13,
"timestamp": 1617318480,
"state": {
  "color": "red"
},
"metadata": {
  "color": {
    "timestamp": 1617318480
  }
}
}
```

ステップ 3: Device Shadow インタラクションに関するエラーをトラブルシューティングする

Shadow サンプルアプリケーションを実行すると、Device Shadow サービスとのインタラクションの観察に関する問題が発生する場合があります。

プログラムが正常に実行され、desired 値を入力するように求められた場合は、前述のように Shadow ドキュメントと MQTT テストクライアントを使用して、Device Shadow のインタラクショ

ンを観察できるはずです。ただし、インタラクションが表示されない場合は、次のことを確認できます。

- AWS IoT コンソールでモノの名前とそのシャドウを確認する

  Shadow ドキュメントにメッセージが表示されない場合は、コマンドを確認して、AWS IoT コンソールのモノの名前と一致していることを確認してください。モノのリソースを選択してから [Shadows] (シャドウ) を選択することで、クラシックシャドウがあるかどうかを確認することもできます。このチュートリアルは、主にクラシックシャドウとのインタラクションに焦点を当てています。

  使用したデバイスがインターネットに接続されていることも確認できます。AWS IoT コンソールで、以前に作成したモノを選択してから、[Interact] (操作) を選択します。モノの詳細ページに、次を記載したメッセージが表示されます: `This thing already appears to be connected.`。

- サブスクライブした MQTT 予約済みトピックを確認する

  メッセージが MQTT テストクライアントに表示されない場合は、サブスクライブしたトピックの形式が正しいかどうかを確認してください。MQTT デバイスシャドウトピックの形式は $aws/things/***thingname***/shadow/ であり、シャドウで実行するアクションに応じて、`update`、`get`、または `delete` がそれに続く場合があります。このチュートリアルでは、トピック $aws/things/***thingname***/shadow/# を使用しているため、テストクライアントの [Topic filter] (トピックフィルター) セクションでトピックをサブスクライブするときに、正しく入力したことを確認してください。

  トピック名を入力するときは、*#####*が前に作成した AWS IoT モノの名前と同じであることを確認します。追加の MQTT トピックにサブスクライブして、更新が正常に実行されたかどうかを確認することもできます。例えば、接続の問題をデバッグできるように、トピック $aws/things/***thingname***/shadow/update/rejected にサブスクライブして、更新リクエストが失敗したときにメッセージを受信します。予約されているトピックの詳細については、the section called “シャドウトピック” および Device Shadow MQTT トピック を参照してください。

ステップ 4: 結果と次のステップを確認する

このチュートリアルで学習した内容は次のとおりです。

- `shadow.py` サンプルアプリケーションを使用して、必要な状態を指定し、シャドウの現在の状態を更新します。

- Shadow ドキュメントを編集して、デルタイベントと、shadow.py サンプルアプリケーションがそれにどのように応答するかを観察します。
- MQTT テストクライアントを使用して、シャドウトピックをサブスクライブし、サンプルプログラムを実行するときに更新を確認します。

次のステップ

追加の MQTT 予約トピックにサブスクライブして、シャドウアプリケーションの更新を監視できます。例えば、トピック $aws/things/***thingname***/shadow/update/accepted のみをサブスクライブする場合、更新が正常に実行されると、現在の状態情報のみが表示されます。

追加のシャドウトピックにサブスクライブして、問題をデバッグしたり、Device Shadow のインタラクションの詳細を確認したり、Device Shadow のインタラクションに関する問題をデバッグしたりすることもできます。詳細については、「the section called “シャドウトピック”」および「Device Shadow MQTT トピック」を参照してください。

名前付きシャドウを使用するか、LED 用に Raspberry Pi に接続された追加のハードウェアを使用してアプリケーションを拡張し、ターミナルから送信されたメッセージを使用して状態の変化を観察することもできます。

Device Shadow サービス、デバイスでのサービスの使用、アプリケーションの使用、およびサービスの使用の詳細については、AWS IoT Device Shadow サービス、デバイスでのシャドウの使用、および アプリとサービスでのシャドウの使用 を参照してください。

チュートリアル: のカスタムオーソライザーの作成 AWS IoT Core

このチュートリアルでは、 AWS CLIを使用してカスタム認証を作成、検証、使用する手順を示します。オプションで、このチュートリアルを使用して、HTTP Publish API を使用して AWS IoT Core にデータを送信するために Postman を使用できます。

このチュートリアルでは、トークン署名が有効な create-authorizer 呼び出しを使用して、認可および認証ロジックとカスタムオーソライザーを実装するサンプル Lambda 関数を作成する方法を示します。その後、オーソライザーは を使用して検証されtest-invoke-authorizer、最後に HTTP Publish API AWS IoT Core を使用してテスト MQTT トピックにデータを送信できます。サンプルリクエストでは、`x-amz-customauthorizer-name` ヘッダーを使用して呼び出すオーソライザーを指定し、トークンキー名と `x-amz-customauthorizer-signature` をリクエストヘッダーで渡します。

このチュートリアルでは、次の内容を学習します。

- カスタムオーソライザーハンドラーとなる Lambda 関数を作成する方法

- トークン署名が有効になってい AWS CLI る を使用してカスタムオーソライザーを作成する方法
- test-invoke-authorizer コマンドを使用してカスタムオーソライザーをテストする方法
- Postman を使用して MQTT トピックを発行し、カスタムオーソライザーでリクエストを検証する方法

このチュートリアルの完了には 60 分ほどかかります。

このチュートリアルでは、次の作業を行います。

- ステップ 1: カスタムオーソライザー用の Lambda 関数を作成する
- ステップ 2: カスタムオーソライザーのパブリックキーとプライベートキーのペアを作成する
- ステップ 3: カスタムオーソライザーリソースとその承認を作成する
- ステップ 4: test-invoke-authorizer を呼び出してオーソライザーをテストする
- ステップ 5: Postman を使用して MQTT メッセージの発行をテストする
- ステップ 6: MQTT テストクライアントでメッセージを表示する
- ステップ 7: 結果と次のステップを確認する
- ステップ 8: クリーンアップする

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- セットアップ AWS アカウント

  このチュートリアルを完了するには、 AWS アカウント と AWS IoT コンソールが必要です。

  このチュートリアルで使用するアカウントは、少なくとも次の AWS マネージドポリシーが含まれている場合に最適に機能します。

  - `IAMFullAccess`
  - `AWSIoTFullAccess`
  - `AWSLambda_FullAccess`

  > **Important**
  >
  > このチュートリアルで使用される IAM ポリシーは、本稼働環境で従うべきものよりも許容度が高いです。本稼働環境で、アカウントポリシーとリソースポリシーが必要なアクセス許可のみを付与していることを確認します。

本稼働用の IAM ポリシーを作成する場合は、ユーザーとロールに必要なアクセスを判断し、次にそれらのタスクのみの実行を許可するポリシーを設計します。
詳細については、IAM のセキュリティのベストプラクティスを参照してください。

- をインストールしました AWS CLI

  のインストール方法については AWS CLI、「 AWS CLI のインストール」を参照してください。このチュートリアルには AWS CLI バージョン `aws-cli/2.1.3 Python/3.7.4 Darwin/18.7.0 exe/x86_64` 以降が必要です。

- OpenSSL ツール

  このチュートリアルの例では、LibreSSL 2.6.5 を使用しています。このチュートリアルでは、OpenSSL v1.1.1i ツールを使用することもできます。

- 「AWS Lambda の概要」を確認します。

  AWS Lambda を使用したことがない場合は、AWS Lambda「」と「Lambda の開始方法」を確認して、その用語と概念を確認してください。

- Postman でリクエストを作成する方法を見直しました

  詳細については、「リクエストのビルド」を参照してください。

- 以前のチュートリアルからカスタムオーソライザーを削除しました

  では、一度に設定 AWS アカウント できるカスタムオーソライザーの数は限られています。カスタムオーソライザーの削除方法の詳細については、the section called “ステップ 8: クリーンアップする” を参照してください。

ステップ 1: カスタムオーソライザー用の Lambda 関数を作成する

のカスタム認証 AWS IoT Core では、作成したオーソライザーリソースを使用してクライアントを認証および認可します。このセクションで作成する 関数は、クライアントが AWS IoT リソースに接続してアクセスするときに、クライアントを認証 AWS IoT Core および認可します。

Lambda 関数は以下を実行します。

- リクエストが test-invoke-authorizer からのものである場合、`Deny` アクションを含む IAM ポリシーを返します。

- リクエストが HTTP を使用して Postman から送信され、`actionToken` パラメータの値が `allow` の場合、`Allow` アクションを使用して IAM ポリシーを返します。それ以外の場合は、`Deny` アクションを含む IAM ポリシーを返します。

カスタムオーソライザー用の Lambda 関数を作成するには

1. Lambda コンソールで、[関数] を開きます。
2. [Create function] を選択します。
3. [一から作成] が選択されていることを確認します。
4. [Basic information]:
   a. [関数名] に **`custom-auth-function`** と入力します。
   b. [ランタイム] で、[Node.js 18.x] を確認します
5. [Create function] を選択します。

   Lambda により、Node.js の関数と実行ロールが作成され、ログをアップロードするためのアクセス許可が関数に付与されます。Lambda 関数は、関数を呼び出すときに実行ロールを引き受け、実行ロールを使用して AWS SDK の認証情報を作成し、イベントソースからデータを読み込みます。
6. AWS Cloud9 エディタで関数のコードと設定を表示するには、デザイナーウィンドウで [custom-auth-function] を選択し、エディタのナビゲーションペインで [index.js] を選択します。

   Node.js などのスクリプト言語では、Lambda には成功のレスポンスを返す基本関数が含まれています。ソースコードが 3 MB を超えない限り、AWS Cloud9 エディタを使用して関数を編集できます。
7. エディタの index.js コードを次のコードに置き換えます。

```
// 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. [デプロイ] を選択します。
9. [変更がデプロイされました] が エディタの上に表示されたら、次の操作を実行します。
   a. エディタの上にある [関数の概要] セクションまでスクロールします。
   b. このチュートリアルの後半で使用するために、[関数 ARN] をコピーして保存します。
10. 関数をテストします。
    a. [テスト] タブを選択します。

b. デフォルトのテスト設定を使用して、[呼び出し] を選択します。

c. テストが成功した場合は、[実行結果] で [詳細] ビューを開きます。関数が返したポリシードキュメントが表示されます。

テストが失敗した場合、またはポリシードキュメントが表示されない場合は、コードを確認し、エラーを見つけて修正します。

ステップ 2: カスタムオーソライザーのパブリックキーとプライベートキーのペアを作成する

カスタムオーソライザーでは、認証にパブリックキーとプライベートキーが必要です。このセクションのコマンドは、OpenSSL ツールを使用してこのキーペアを作成します。

カスタムオーソライザーのパブリックキーとプライベートキーのペアを作成するには

1. プライベートキーファイルを作成します。

```
openssl genrsa -out private-key.pem 4096
```

2. 先ほど作成したプライベートキーファイルを検証します。

```
openssl rsa -check -in private-key.pem -noout
```

コマンドがエラーを表示しない場合、プライベートキーファイルは有効です。

3. パブリックキーファイルを作成します。

```
openssl rsa -in private-key.pem -pubout -out public-key.pem
```

4. パブリックキーファイルを確認します。

```
openssl pkey -inform PEM -pubin -in public-key.pem -noout
```

コマンドがエラーを表示しない場合、パブリックキーファイルは有効です。

ステップ 3: カスタムオーソライザーリソースとその承認を作成する

AWS IoT カスタムオーソライザーは、前のステップで作成したすべての要素を結び付けるリソースです。このセクションでは、カスタムオーソライザーリソースを作成し、以前に作成した Lambda

関数を実行するためのアクセス許可を付与します。 AWS IoT コンソール、、または AWS API を使用して AWS CLI、カスタムオーソライザーリソースを作成できます。

このチュートリアルでは、1 つのカスタムオーソライザーを作成するだけで済みます。このセクションでは、 AWS IoT コンソールと を使用して を作成する方法について説明します。これにより AWS CLI、最も便利な メソッドを使用できます。どちらの方法でも作成されたカスタムオーソライザーリソースには違いはありません。

カスタムオーソライザーリソースを作成する

オプションを選択してカスタムオーソライザーリソースを作成する

- AWS IoT コンソールを使用してカスタムオーソライザーを作成する
- を使用してカスタムオーソライザーを作成する AWS CLI

カスタムオーソライザーを作成するには (コンソール)

1. AWS IoT コンソールのカスタムオーソライザーページを開き、オーソライザーの作成を選択します。
2. [オーソライザーの作成] で、次のように操作します。
   a. [オーソライザーの名前] に、**my-new-authorizer** と入力します。
   b. [オーソライザーのステータス] で、[アクティブ] にチェックを入れます。
   c. [オーソライザー関数] で、前に作成した Lambda 関数を選択します。
   d. [トークン検証 - オプション] で次の操作を実行します。
      i. [トークン検証] をオンにします。
      ii. [トークンキー名] に、**tokenKeyName** を入力します。
      iii. [キーの追加] を選択します。
      iv. [キー名] で **FirstKey** を入力します。
      v. [パブリックキー] で、public-key.pem ファイルの内容を入力します。-----BEGIN PUBLIC KEY----- と -----END PUBLIC KEY----- を含むファイルの行を必ず含め、ラインフィード、キャリッジリターン、またはその他の文字をファイルの内容に追加したり、ファイルの内容から削除したりしないでください。入力する文字列は、この例のように表示されます。

```
-----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. [オーソライザーの作成] を選択します。
4. カスタムオーソライザーリソースが作成された場合は、カスタムオーソライザーのリストが表示され、新しいカスタムオーソライザーがそのリストに表示されます。これで、次のセクションに進んでテストできます。

   エラーが表示された場合は、エラーを確認し、カスタムオーソライザーを再度作成して、エントリを再確認してください。各カスタムオーソライザーリソースには一意の名前が必要であることに注意してください。

カスタムオーソライザーを作成するには (AWS CLI)

1. `authorizer-function-arn` と `token-signing-public-keys` の値を置き換えてから、次のコマンドを実行します。

```
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-----"
```

各パラメータの意味は次のとおりです。

- authorizer-function-arn 値は、カスタムオーソライザー用に作成した Lambda 関数の Amazon リソースネーム (ARN) です。
- token-signing-public-keys 値には、キーの名前、**FirstKey**、および public-key.pem ファイルの内容が含まれます。-----BEGIN PUBLIC KEY----- と -----END PUBLIC KEY----- を含むファイルの行を必ず含め、ラインフィード、キャリッジリターン、またはその他の文字をファイルの内容に追加したり、ファイルの内容から削除したりしないでください。

  注: パブリックキーの値を変更すると使用できなくなるため、パブリックキーの入力には注意してください。

2. カスタムオーソライザーが作成されている場合、コマンドは次のような新しいリソースの名前と ARN を返します。

```
{
    "authorizerName": "my-new-authorizer",
    "authorizerArn": "arn:aws:iot:Region:57EXAMPLE833:authorizer/my-new-authorizer"
}
```

次のステップで使用するために authorizerArn 値を保存します。

各カスタムオーソライザーリソースには一意の名前が必要であることに注意してください。

カスタムオーソライザーリソースを承認する

このセクションでは、作成したカスタムオーソライザーリソースに、Lambda 関数を実行するためのアクセス許可を付与します。アクセス許可を付与するには、add-permission CLI コマンドを使用できます。

を使用して Lambda 関数にアクセス許可を付与する AWS CLI

1. 値を挿入したら、次のコマンドを入力します。statement-id 値は一意でなければならないことに注意してください。このチュートリアルを以前に実行したことがある場合、または ResourceConflictException エラーが発生した場合は、*Id-1234* を別の値に置き換えてください。

```
aws lambda add-permission  \
--function-name "custom-auth-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn authorizerArn
```

2. コマンドが成功すると、この例のようなアクセス許可ステートメントが返されます。次のセクションに進んで、カスタムオーソライザーをテストできます。

```
{
    "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\"}}}"
}
```

コマンドが成功しない場合は、この例のようなエラーが返されます。続行する前に、エラーを確認して修正する必要があります。

```
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
```

ステップ 4: test-invoke-authorizer を呼び出してオーソライザーをテストする

すべてのリソースを定義したら、このセクションでは、コマンドラインから test-invoke-authorizer を呼び出して、認可パスをテストします。

コマンドラインからオーソライザーを呼び出す場合、protocolData は定義されていないため、オーソライザーは常に DENY ドキュメントを返すことに注意してください。ただし、このテストは、Lambda 関数を完全にテストしなくても、カスタムオーソライザーと Lambda 関数が正しく設定されていることを確認します。

を使用してカスタムオーソライザーとその Lambda 関数をテストするには AWS CLI

1. 前の手順で作成した private-key.pem ファイルがあるディレクトリで、次のコマンドを実行します。

```
echo -n "tokenKeyValue" | openssl dgst -sha256 -sign private-key.pem | openssl
 base64 -A
```

このコマンドは、次のステップで使用する署名文字列を作成します。署名文字列は次のようになります。

```
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=
```

この署名文字列をコピーして、次の手順で使用します。余分な文字を含めたり、省略したりしないように注意してください。

2. このコマンドで、token-signature 値を前のステップの署名文字列に置き換え、このコマンドを実行してオーソライザーをテストします。

```
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=
```

コマンドが成功すると、この例のように、カスタムオーソライザー関数によって生成された情報が返されます。

```
{
    "isAuthenticated": true,
    "principalId": "principalId",
    "policyDocuments": [
        "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Action\":\"iot:*\",
\"Effect\":\"Deny\",\"Resource\":\"arn:aws:iot:Region:57EXAMPLE833:*\"}]}"
    ],
    "refreshAfterInSeconds": 600,
    "disconnectAfterInSeconds": 3600
}
```

コマンドからエラーが返された場合は、エラーを確認し、このセクションで使用したコマンドを再度確認します。

ステップ 5: Postman を使用して MQTT メッセージの発行をテストする

1. コマンドラインからデバイスデータエンドポイントを取得するには、ここに示すように describe-endpoint を呼び出します

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

   このアドレスを保存して、後のステップで *device_data_endpoint_address* として使用します。
2. 新しい Postman ウィンドウを開き、新しい HTTP POST リクエストを作成します。

   a. コンピュータで、Postman アプリケーションを開きます。

b. Postman の [ファイル] メニューで、[新規] を選択します。

c. [New] (新規) ダイアログボックスで、[Request] (リクエスト) を選択します。

d. [Save] (保存) リクエストで、

i. [Request name] (リクエスト名) で、**`Custom authorizer test request`** と入力します。

ii. [保存先のコレクションまたはフォルダを選択:] で、このリクエストを保存するコレクションを選択または作成します。

iii. [***`collection_name`*** に保存] を選択します。

3. カスタムオーソライザーをテストするための POST リクエストを作成します。

a. URL フィールドの横にあるリクエストメソッドセレクターで、[POST] を選択します。

b. URL フィールドで、前のステップの *`describe-endpoint`* コマンドの device_data_endpoint_address とともに次の URL を使用して、リクエスト用の URL を作成します。

```
https://device_data_endpoint_address:443/topics/test/cust-auth/topic?
qos=0&actionToken=allow
```

この URL には、 AWS IoTへのアクセスを許可するポリシードキュメントを返すように Lambda 関数に指示する `actionToken=allow` クエリパラメータが含まれていることに注意してください。URL を入力すると、Postman の [Params] (パラメータ) タブにもクエリパラメータが表示されます。

c. [認証] タブの [タイプ] フィールドで、[認証なし] を選択します。

d. [Headers] (ヘッダー) タブで次の操作を行います。

i. チェックが入っている [ホスト] キーがある場合は、このチェックを解除します。

ii. ヘッダーのリストの一番下に、これらの新しいヘッダーを追加し、チェックが入っていることを確認します。**`Host`** 値を *`device_data_endpoint_address`* に置き換え、**`x-amz-customauthorizer-signature`** 値を前のセクションの test-invoke-authorize コマンドで使用した署名文字列に置き換えます。

キー	値
`x-amz-customauthorizer-name`	**`my-new-authorizer`**

キー	値
Host	*device_data_endpoint_address*
tokenKeyName	**tokenKeyValue**
x-amz-customauthorizer-signature	*dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjjszEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca+tsDuXf3LzCwQQF/YSUy02u5XkWn+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=*

e. [Body] (本文) タブで、次の操作を行います。

   i. データ形式オプションボックスで、[Raw] を選択します。

ii. データ型リストで、[JavaScript] を選択します。

iii. テキストフィールドで、テストメッセージ用の次の JSON メッセージペイロードを入力します。

```
{
    "data_mode": "test",
    "vibration": 200,
    "temperature": 40
}
```

4. [送信] を選択してリクエストを送信します。

リクエストが成功した場合、次を返します。

```
{
    "message": "OK",
    "traceId": "ff35c33f-409a-ea90-b06f-fbEXAMPLE25c"
}
```

正常なレスポンスは、カスタムオーソライザーが への接続を許可 AWS IoT し、テストメッセージが のブローカーに配信されたことを示します AWS IoT Core。

エラーが返された場合は、エラーメッセージ、*device_data_endpoint_address*、署名文字列、およびその他のヘッダー値を確認してください。

次のセクションで使用するために、このリクエストを Postman で保持します。

ステップ 6: MQTT テストクライアントでメッセージを表示する

前のステップでは、Postman AWS IoT を使用してシミュレートされたデバイスメッセージを に送信しました。成功した応答は、カスタムオーソライザーが AWS IoT への接続を許可し、テストメッセージが AWS IoT Coreのブローカーに配信されたことを示しました。このセクションでは、 AWS IoT コンソールで MQTT テストクライアントを使用して、他のデバイスやサービスと同様に、そのメッセージからのメッセージの内容を確認します。

カスタムオーソライザーによって承認されたテストメッセージを表示するには

1. AWS IoT コンソールで、MQTT テストクライアントを開きます。

2. [Subscribe to topic] (トピックへのサブスクライブ) タブの [Topic filter] (トピックフィルター) で、前のセクションの Postman の例で使用されているメッセージトピックである **`test/cust-auth/topic`** を入力します。
3. [Subscribe] を選択します。

   次のステップのために、このウィンドウを表示したままにします。
4. Postman で、前のセクションで作成したリクエストで、[送信] を選択します。

   応答を確認して、正常に完了したことを確認します。そうでない場合は、前のセクションで説明したようにエラーをトラブルシューティングします。
5. [MQTT テストクライアント] に、メッセージトピックを示す新しいエントリが表示され、展開すると、Postman から送信したリクエストからのメッセージペイロードが表示されます。

   [MQTT テストクライアント] にメッセージが表示されない場合は、次の点を確認してください。

   - Postman リクエストが正常に返されたことを確認します。が接続 AWS IoT を拒否し、エラーを返した場合、リクエスト内のメッセージはメッセージブローカーに渡されません。
   - AWS IoT コンソールを開く AWS リージョン ために使用される AWS アカウント とが、Postman URL で使用しているものと同じであることを確認します。
   - カスタムオーソライザーに適切なエンドポイントを使用していることを確認してください。デフォルトの IoT エンドポイントは、Lambda 関数でのカスタムオーソライザーの使用をサポートしていない場合があります。その場合は、ドメイン設定を使用して新しいエンドポイントを定義し、そのエンドポイントをカスタムオーソライザーに指定できます。
   - MQTT テストクライアントにトピックを正しく入力したことを確認してください。トピックフィルターでは、大文字と小文字が区別されます。疑わしい場合は、 #トピックをサブスクライブすることもできます。トピックは、メッセージブローカーを通過するすべての MQTT メッセージにサブスクライブ AWS アカウント し、 コンソールを開く AWS リージョン AWS IoT ために使用されます。

ステップ 7: 結果と次のステップを確認する

このチュートリアルでは、次の作業を行いました。

- Lambda 関数をカスタムオーソライザーハンドラーとして作成しました
- トークン署名を有効にしてカスタムオーソライザーを作成しました
- test-invoke-authorizer コマンドを使用してカスタムオーソライザーをテストしました

- Postman を使用して MQTT トピックを発行し、カスタムオーソライザーでリクエストを検証しました
- Postman テストから送信されたメッセージを表示するために [MQTT テストクライアント] を使用しました

次の手順

Postman からメッセージを送信してカスタムオーソライザーが機能していることを確認したら、このチュートリアルのさまざまな側面を変更すると結果にどのように影響するかを実験してみてください。手始めにいくつか例を紹介します。

- 署名文字列を変更して、不正な接続の試みがどのように処理されるかを確認できないようにします。このようなエラー応答が返され、メッセージは MQTT テストクライアントに表示されないはずです。

```
{
    "message": "Forbidden",
    "traceId": "15969756-a4a4-917c-b47a-5433e25b1356"
}
```

- AWS IoT ルールの開発中および使用中に発生する可能性のあるエラーを見つける方法の詳細については、「」を参照してくださいモニタリング AWS IoT。

ステップ 8: クリーンアップする

このチュートリアルを繰り返したい場合は、カスタムオーソライザーの一部を削除する必要があります。では、一度に設定 AWS アカウント できるカスタムオーソライザーの数は限られており、既存のカスタムオーソライザーを削除せずに新しいオーソライザーを追加`LimitExceededException`しようとすると、 を取得できます。

カスタムオーソライザーを削除するには (コンソール)

1. AWS IoT コンソールのカスタムオーソライザーページを開き、カスタムオーソライザーのリストで、削除するカスタムオーソライザーを見つけます。
2. [Custom authorizer details] (カスタムオーソライザーの詳細) ページを開き、[Actions] (アクション) メニューから [Edit] (編集) を選択します。
3. [オーソライザーアクティブ化] のチェックを解除し、[更新] を選択します。

アクティブなカスタムオーソライザーを削除することはできません。

4. [Custom authorizer details] (カスタムオーソライザーの詳細) ページで、[Actions] (アクション) メニューを開き、[Delete] (削除) を選択します。

カスタムオーソライザーを削除するには (AWS CLI)

1. インストールしたカスタムオーソライザーの一覧を表示し、削除するカスタムオーソライザーの名前を見つけます。

```
aws iot list-authorizers
```

2. *Custom_Auth_Name* を、削除するカスタムオーソライザーの authorizerName に置き換えた後、このコマンドを実行してカスタムオーソライザーを inactive に設定します。

```
aws iot update-authorizer --status INACTIVE --authorizer-name Custom_Auth_Name
```

3. *Custom_Auth_Name* を、削除するカスタムオーソライザーの authorizerName に置き換えた後、このコマンドを実行してカスタムオーソライザーを削除します。

```
aws iot delete-authorizer --authorizer-name Custom_Auth_Name
```

チュートリアル: AWS IoT と Raspberry Pi による土壌湿度のモニタリング

このチュートリアルでは、Raspberry Pi、湿度センサー、 AWS IoT および を使用して、家の植物や庭の土壌湿度レベルをモニタリングする方法を示します。Raspberry Pi は、センサーから湿度レベルと温度を読み取り、データを送信するコードを実行します AWS IoT。湿度レベル AWS IoT がしきい値を下回ると、Amazon SNS トピックにサブスクライブされているアドレスに E メールを送信するルールを で作成します。

> **Note**
>
> このチュートリアルは最新ではない可能性があります。このトピックの最初の公開以降に、いくつかの参照が置き換えられている可能性があります。

目次

- 前提条件
- セットアップ AWS IoT
  - ステップ 1: AWS IoT ポリシーを作成する
  - ステップ 2: AWS IoT モノ、証明書、プライベートキーを作成する
  - ステップ 3: Amazon SNS トピックおよびサブスクリプションを作成する
  - ステップ 4: E メールを送信する AWS IoT ルールを作成する
- Raspberry Pi と湿度センサーのセットアップ

前提条件

このチュートリアルを完了するには、以下が必要です。

- AWS アカウント。
- 管理者権限を持つ IAM ユーザー。
- AWS IoT コンソールにアクセスするための、Windows、macOS、Linux、または Unix を実行している開発用コンピュータ。
- 最新の Raspberry Pi OS を実行する Raspberry Pi 3B または 4B。インストール手順については、Raspberry Pi ウェブサイトの「Install an operating system」を参照してください。
- Raspberry Pi 用のモニター、キーボード、マウス、Wi-Fi ネットワークまたはイーサネット接続。
- Raspberry Pi 対応の湿度センサー。このチュートリアルで使用するセンサーは、Adafruit STEMMA I2C 容量性湿度センサーで、JST 4 ピンからメスソケットへのケーブルヘッダーを備えています。

セットアップ AWS IoT

このチュートリアルを完了するには、次のリソースを作成する必要があります。デバイスを に接続するには AWS IoT、IoT モノ、デバイス証明書、および AWS IoT ポリシーを作成します。

- AWS IoT モノ。

  モノは物理デバイス (この場合は Rasberry Pi) を表し、デバイスに関する静的メタデータを含みます。
- デバイス証明書。

  AWS IoTに接続して認証するには、すべてのデバイスにデバイス証明書が必要です。

- AWS IoT ポリシー。

  各デバイス証明書には、1 つ以上の AWS IoT ポリシーが関連付けられています。これらのポリシーは、デバイスがアクセスできる AWS IoT リソースを決定します。
- AWS IoT ルート CA 証明書。

  デバイスやその他のクライアントは、 AWS IoT ルート CA 証明書を使用して、通信している AWS IoT サーバーを認証します。詳細については、「サーバー認証」を参照してください。
- AWS IoT ルール。

  ルールには、クエリと 1 つ以上のルールアクションが含まれます。クエリは、デバイスメッセージからデータを抽出して、メッセージデータを処理する必要があるかどうかを判断します。ルールアクションにより、データがクエリに一致する場合の処理が指定されます。
- Amazon SNS トピックおよびトピックサブスクリプション。

  このルールでは、Raspberry Pi からの湿度データがリッスンされます。値がしきい値を下回る場合、Amazon SNS トピックにメッセージを送信します。Amazon SNS は、トピックにサブスクライブしているすべての E メールアドレスにそのメッセージを送信します。

ステップ 1: AWS IoT ポリシーを作成する

Raspberry Pi がメッセージを接続および送信できるようにする AWS IoT ポリシーを作成します AWS IoT。

1. AWS IoT コンソールで、[今すぐ始める] ボタンが表示された場合はそれをクリックします。それ以外の場合は、ナビゲーションペインで [Security] (セキュリティ) を展開し、[Policies] (ポリシー) を選択します。
2. [ポリシーはまだ作成されていません] ダイアログボックスが表示された場合は、[ポリシーの作成] を選択します。それ以外の場合は、[Create (作成)] を選択します。
3. AWS IoT ポリシーの名前を入力します (例: **`MoistureSensorPolicy`**)。
4. [Add statements (ステートメントの追加)] セクションで、既存のポリシーを次の JSON に置き換えます。*`region`* と *`account`* を AWS リージョン と AWS アカウント number に置き換えます。

```
{
    "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. [作成] を選択します。

ステップ 2: AWS IoT モノ、証明書、プライベートキーを作成する

Raspberry Pi を表すモノを AWS IoT レジストリに作成します。

1. AWS IoT コンソールのナビゲーションペインで、[管理]、[モノ] の順に選択します。
2. [まだモノがありません] ダイアログボックスが表示された場合は、[モノの登録] を選択します。それ以外の場合は、[Create (作成)] を選択します。
3. AWS IoT モノの作成ページで、モノを 1 つ作成する を選択します。
4. [Add your device to the device registry (デバイスレジストリへのデバイスの追加)] ページで、IoT モノの名前 (例: **RaspberryPi**) を入力し、[次へ] を選択します。作成後にモノの名前は変更できません。モノの名前を変更するには、新しいモノを作成して、新しい名前を付け、古いモノを削除する必要があります。
5. [モノに証明書を追加] ページで、[証明書の作成] を選択します。
6. [ダウンロード] リンクを選択して、証明書、プライベートキー、ルート CA 証明書をダウンロードします。

> **Important**
>
> これは、証明書とプライベートキーをダウンロードできる唯一の時間です。

7. 証明書を有効にするには、[Activate] (有効化) を選択します。デバイスが AWS IoTに接続するには、証明書がアクティブである必要があります。
8. [ポリシーのアタッチ] を選択します。
9. [Add a policy for your thing (モノのポリシーを追加)] で、[MoistureSensorPolicy]、[Register Thing (モノの登録)] の順に選択します。

ステップ 3: Amazon SNS トピックおよびサブスクリプションを作成する

Amazon SNS トピックおよびサブスクリプションを作成します。

1. AWS SNS コンソールのナビゲーションペインで [Topics] (トピック) を選択し、[Create topic] (トピックの作成) を選択します。
2. タイプを [標準] として選択し、トピックの名前を入力します (例: **`MoistureSensorTopic`**)。
3. トピックの表示名を入力します (例: **`Moisture Sensor Topic`**)。これは、Amazon SNS コンソールでトピックに表示される名前です。
4. [トピックの作成] を選択します。
5. Amazon SNS トピックの詳細ページで、[Create subscription] (サブスクリプションの作成) を選択します。
6. [Protocol (プロトコル)] として [Email (E メール)] を選択してください。
7. [エンドポイント] に E メールアドレスを入力します。
8. [Create subscription] を選択します。
9. E メールクライアントを開き、**`MoistureSensorTopic`** という件名のメッセージを探します。E メールを開き、[サブスクリプションを確認] リンクを選択します。

> **Important**
>
> サブスクリプションを確認するまで、この Amazon SNS トピックからの E メールアラートは受信されません。

入力したテキストが記載された E メールメッセージが届きます。

ステップ 4: E メールを送信する AWS IoT ルールを作成する

AWS IoT ルールは、デバイスからメッセージを受信したときに実行するクエリと 1 つ以上のアクションを定義します。 AWS IoT ルールエンジンは、デバイスによって送信されたメッセージをリッ

スンし、メッセージ内のデータを使用して何らかのアクションを実行する必要があるかどうかを判断します。詳細については、「のルール AWS IoT」を参照してください。

このチュートリアルでは、Raspberry Pi が aws/things/RaspberryPi/shadow/update にメッセージを発行します。これは、デバイスと Thing Shadow サービスで使用される内部 MQTT トピックです。Raspberry Pi は、次の形式のメッセージを発行します。

```
{
    "reported": {
        "moisture" : moisture-reading,
        "temp" : temperature-reading
    }
}
```

受信メッセージから湿度と温度データを抽出するクエリを作成します。また、湿度の読み取り値がしきい値を下回っている場合、データを受け取り、そのデータを Amazon SNS トピックのサブスクライバーに送信する Amazon SNS アクションも作成します。

Amazon SNS ルールを作成する

1. [AWS IoT コンソール] で、[メッセージのルーティング] を選択し、[ルール] を選択します。[ルールはまだ作成されていません] ダイアログボックスが表示された場合は、[ルールの作成] を選択します。それ以外の場合は、[ルールを作成] を選択します。
2. [ルールプロパティ] ページで、**MoistureSensorRule** などの [ルール名] を入力し、**Sends an alert when soil moisture level readings are too low** などの短い [ルールの説明] を入力します。
3. [次へ] を選択し、SQL ステートメントを設定します。SQL バージョンを 2016-03-23 として選択し、次の AWS IoT SQL クエリステートメントを入力します。

   ```
   SELECT * FROM '$aws/things/RaspberryPi/shadow/update/accepted' WHERE
    state.reported.moisture < 400
   ```

   このステートメントは、moisture の読み取り値が 400 より小さい場合にルールアクションをトリガーします。

> **Note**
>
> 別の値の使用が必要になる場合があります。Raspberry Pi でコードを実行したら、センサーに触れたり、水に入れたり、プランター内に置いたりすることで、センサーから取得した値を表示できます。

4. [次へ] を選択し、ルールアクションをアタッチします。[アクション 1] で、[Simple Notification Service] を選択します。このルールアクションの説明は [SNS プッシュ通知としてメッセージを送信します] です。
5. [SNS トピック] では、ステップ 3: Amazon SNS トピックおよびサブスクリプションを作成する、、MoistureSensorTopic で作成したトピックを選択し、[メッセージ形式] を [RAW] のままにします。[IAM role] (IAM ロール) は、[Create a new role] (新しいロールの作成) を選択します。ロールの名前 (例: **`LowMoistureTopicRole`**) を入力し、[ロールを作成] を選択します。
6. [次へ] を選択して確認してから、[作成] を選択してルールを作成します。

Raspberry Pi と湿度センサーのセットアップ

microSD カードを Raspberry Pi に挿入し、モニター、キーボード、マウスを接続し、Wi-Fi を使用していない場合はイーサネットケーブルも接続します。電源ケーブルはまだ接続しないでください。

JST ジャンパーケーブルを湿度センサーに接続します。ジャンパーの反対側には次の 4 本のワイヤがあります。

- 緑: I2C SCL
- 白: I2C SDA
- 赤: 電源 (3.5 V)
- 黒: アース

右側にあるイーサネットジャックで Raspberry Pi を保持します。この向きでは、上部に 2 列の GPIO ピンがあります。次の順序で、湿度センサーのワイヤをピンの下の列に接続します。左端のピンから、赤 (電源)、白 (SDA)、緑 (SCL) を接続します。1 つのピンをスキップし、黒い (アース) ワイヤを接続します。詳細については、「Python Computer Wiring」を参照してください。

電源ケーブルを Raspberry Pi に接続し、もう一方の端をコンセントに接続して電源を入れます。

Raspberry Pi を設定します。

1. [Welcome to Raspberry Pi] で、[Next] を選択します。
2. 国、言語、タイムゾーン、キーボードレイアウトを選択します。[Next] を選択します。
3. Raspberry Pi のパスワードを入力し、[Next] を選択します。
4. Wi-Fi ネットワークを選択し、[Next] を選択します。Wi-Fi ネットワークを使用していない場合は、[Skip] を選択します。
5. [Next] を選択して、ソフトウェアの更新を確認します。更新が完了したら、[Restart] を選択して Raspberry Pi を再起動します。

Raspberry Pi が起動したら、I2C インターフェイスを有効にします。

1. Raspbian デスクトップの左上隅にある Raspberry アイコンをクリックし、[Preferences]、[Raspberry Pi Configuration] の順に選択します。
2. [Interfaces] タブの [I2C] で、[Enable] を選択します。
3. [OK] を選択します。

Adafruit STEMMA 湿度センサーのライブラリは、CircuitPython 向けに記述されています。それらのライブラリを Raspberry Pi で実行するには、最新バージョンの Python 3 をインストールする必要があります。

1. コマンドプロンプトから次のコマンドを実行して、Raspberry Pi ソフトウェアを更新します。

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```
2. 次のコマンドを実行して、Python 3 のインストールを更新します。

   ```
   sudo pip3 install --upgrade setuptools
   ```
3. 次のコマンドを実行して、Raspberry Pi GPIO ライブラリをインストールします。

   ```
   pip3 install RPI.GPIO
   ```
4. 次のコマンドを実行して、Adafruit Blinka ライブラリをインストールします。

   ```
   pip3 install adafruit-blinka
   ```

   詳細については、「Installing CircuitPython Libraries on Raspberry Pi」を参照してください。

5. 次のコマンドを実行して、Adafruit Seesaw ライブラリをインストールします。

   ```
   sudo pip3 install adafruit-circuitpython-seesaw
   ```

6. 次のコマンドを実行して Device AWS IoT SDK for Python をインストールします。

   ```
   pip3 install AWSIoTPythonSDK
   ```

これで、必要なすべてのライブラリが Raspberry Pi にインストールされました。**moistureSensor.py** という名前のファイルを作成し、次の Python コードをファイルにコピーします。

```
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)
```

このファイルを、見つけられる場所に保存します。以下のパラメータを使用して、コマンドラインから moistureSensor.py を実行します。

エンドポイント

カスタム AWS IoT エンドポイント。詳細については、「Device Shadow の REST API」を参照してください。

rootCA

AWS IoT ルート CA 証明書へのフルパス。

cert

AWS IoT デバイス証明書へのフルパス。

key

AWS IoT デバイス証明書のプライベートキーへのフルパス。

thingName

モノの名前 (この場合は `RaspberryPi`)。

clientId

MQTT クライアント ID。`RaspberryPi` を使用します。

コマンドラインは次のようになります。

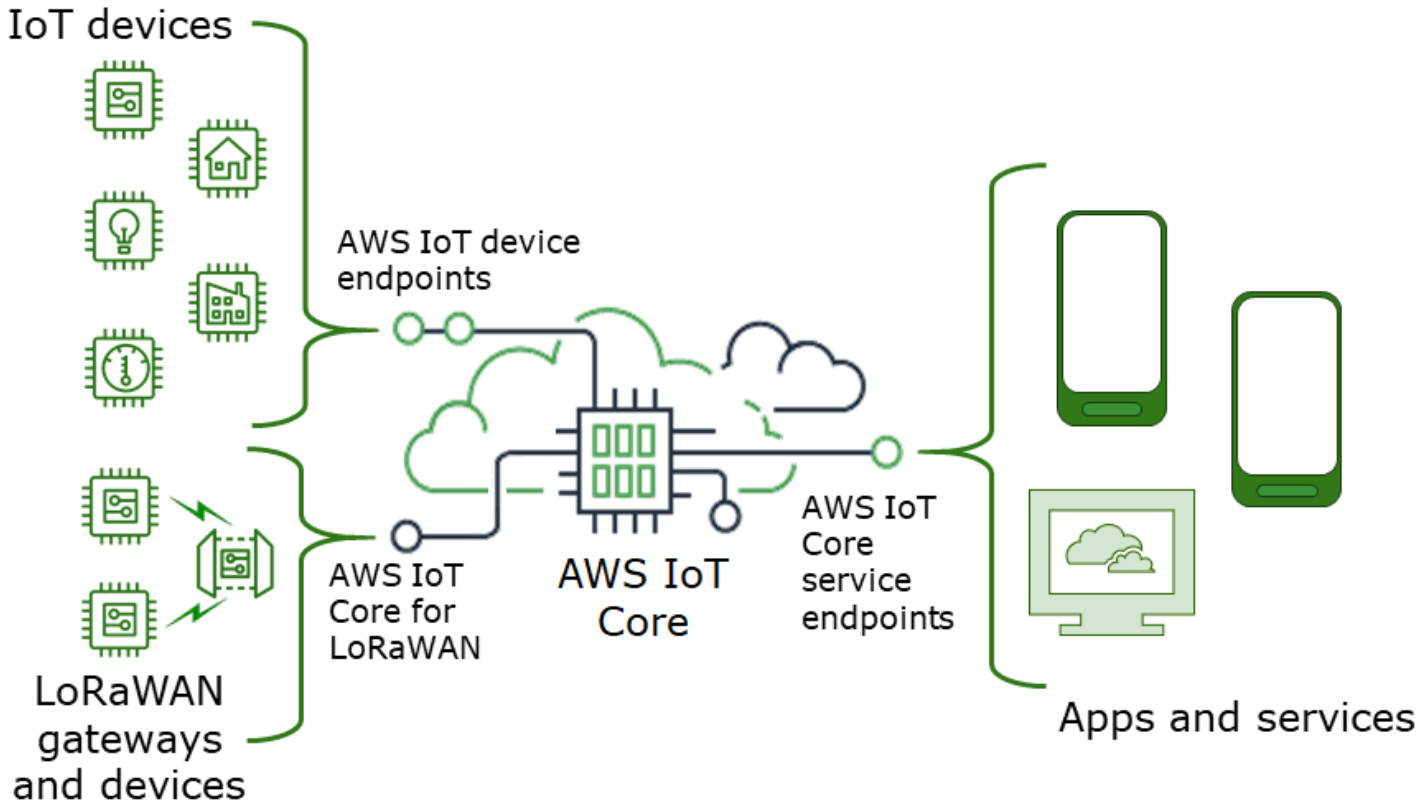
```
python3 moistureSensor.py --endpoint your-endpoint --rootCA ~/certs/
AmazonRootCA1.pem --cert ~/certs/raspberrypi-certificate.pem.crt --key
~/certs/raspberrypi-private.pem.key --thingName RaspberryPi --clientId
RaspberryPi
```

センサーに触れたり、プランター内に置いたり、コップの水に入れたりして、センサーがさまざまなレベルの湿気にどのように反応するかを確認します。必要に応じて、`MoistureSensorRule` でしきい値を変更できます。湿度センサーの読み取り値がルールの SQL クエリステートメントで指定された値を下回ると、 は Amazon SNS トピックにメッセージ AWS IoT を発行します。湿度と温度データが含まれた E メールメッセージが届きます。

Amazon SNS からの E メールメッセージの受信を確認したら、Ctrl+C を押して Python プログラムを停止します。Python プログラムが、料金がかかる量のメッセージを送信することはほとんどありませんが、終了したらプログラムを停止することをお勧めします。

に接続する AWS IoT Core

AWS IoT Core は、IoT デバイス、ワイヤレスゲートウェイ、サービス、アプリケーションとの接続をサポートします。デバイスは に接続 AWS IoT Core し、 AWS IoT サービスや他のデバイスとの間でデータを送受信できるようにします。アプリやその他のサービスも に接続 AWS IoT Core して IoT デバイスを制御および管理し、IoT ソリューションからのデータを処理します。このセクションでは、IoT ソリューションの各側面 AWS IoT Core で に接続して通信する最適な方法を選択する方法について説明します。



を操作する方法はいくつかあります AWS IoT。アプリケーションとサービスは を使用できますAWS IoT Core- コントロールプレーンエンドポイント。デバイスは AWS IoT デバイスエンドポイントまたは AWS IoT Core for LoRaWAN リージョンとエンドポイント AWS IoT Core を使用して に接続できます。

AWS IoT Core- コントロールプレーンエンドポイント

AWS IoT Core - コントロール プレーンエンドポイントは、 AWS IoT ソリューションを制御および管理する 関数へのアクセスを提供します。

- エンドポイント

  AWS IoT Coreコントロールプレーンと AWS IoT Core Device Advisor コントロールプレーンのエンドポイントはリージョン固有であり、「AWS IoT Core エンドポイントとクォータ」に掲載されています。エンドポイントのフォーマットは次のとおりです。

エンドポイントの目的	エンドポイントフォーマット	提供
AWS IoT Core- コントロールプレーン	AWS IoT コントロールプレーンエンドポイント	AWS IoT コントロールプレーン API
AWS IoT Core Device Advisor - コントロールプレーン	`api.iotdeviceadvisor.` *`aws-region`* `.amazonaws.com`	AWS IoT Core Device Advisor コントロールプレーン API

  - IPv4 エンドポイント — IPv4 エンドポイントは IPv4 トラフィックのみをサポートし、すべてのリージョンで使用できます。

    IPv4 エンドポイントでは次の命名規則を使用します。

    ```
    iot.aws-region.amazonaws.com
    ```

    例えば、us-east-1 リージョンの IPv4 エンドポイントは、`iot.us-east-1.amazonaws.com` です。

  - デュアルスタック (IPv4 と IPv6) エンドポイント — デュアルスタックエンドポイントでは、IPv4 トラフィックと IPv6 トラフィックの両方がサポートされます。デュアルスタックエンドポイントにリクエストを行うと、エンドポイント URL は、ネットワークとクライアントが使用するプロトコルに応じて IPv6 または IPv4 アドレスに解決されます。

    デュアルスタックエンドポイントでは、次の命名規則を使用します。

    ```
    iot.aws-region.api.aws
    ```

    例えば、us-east-1 リージョンのデュアルスタックエンドポイント名は、`iot.us-east-1.api.aws` です。

- SDK とツール

AWS SDKs は、 AWS IoT Core APIs および他の AWS サービスの APIs に対して言語固有のサポートを提供します。AWS Mobile SDKs は、アプリ開発者に AWS IoT Core API のプラットフォーム固有のサポートと、モバイルデバイス上のその他の AWS サービスを提供します。

AWS CLI は、 AWS IoT サービスエンドポイントによって提供される関数へのコマンドラインアクセスを提供します。AWS Tools for PowerShell には、PowerShell スクリプト環境で AWS サービスとリソースを管理するためのツールが用意されています。

- 認証

  サービスエンドポイントは、IAM ユーザーと AWS 認証情報を使用してユーザーを認証します。

- 詳細情報

  詳細および SDK リファレンスへのリンクについては、the section called “AWS IoT Core サービスエンドポイントに接続する” を参照してください。

AWS IoT デバイスエンドポイント

AWS IoT デバイスエンドポイントは、IoT デバイスと 間の通信をサポートします AWS IoT。

- エンドポイント

  デバイスエンドポイントは AWS IoT Core および AWS IoT Device Management 関数をサポートします。これらは に固有 AWS アカウント であり、 describe-endpoint コマンドを使用してそれらを確認できます。

エンドポイントの目的	エンドポイントフォーマット	提供
AWS IoT Core- データプレーン	「???」を参照してください。	AWS IoT データプレーン API
AWS IoT Device Management- ジョブのデータ	「???」を参照してください。	AWS IoT ジョブデータプレーン API
AWS IoT Device Advisor - データプレーン	「???」を参照してください。	該当しない

エンドポイントの目的	エンドポイントフォーマット	提供
AWS IoT Device Management- セキュアトンネリング	`api.tunneling.iot.` *`aws-region`* `.amazonaws.com`	AWS IoT セキュアトンネリング API

データプレーンのオペレーションにデュアルスタックエンドポイント (IPv4 と IPv6) を使用している場合は、`iot:Data-ATS` エンドポイントタイプを使用します。`iot:Jobs` は IPv4 にのみ使用できます。これらのエンドポイントとそれらがサポートする機能の詳細については、the section called “AWS IoT デバイスデータとサービスエンドポイント” を参照してください。

- SDK

  AWS IoT Device SDKs は、デバイスが通信に使用する Message Queueing Telemetry Transport (MQTT) および WebSocket Secure (WSS) プロトコルの言語固有のサポートを提供します AWS IoT。 AWS モバイル SDKs は、MQTT デバイス通信、 AWS IoT APIs、およびモバイルデバイス上の他の AWS サービスの APIsもサポートします。

- 認証

  デバイスエンドポイントは、X.509 証明書または認証情報を持つ AWS IAM ユーザーを使用してユーザーを認証します。

- 詳細情報

  詳細および SDK リファレンスへのリンクについては、the section called “AWS IoT デバイス SDKs” を参照してください。

AWS IoT Core for LoRaWAN ゲートウェイとデバイス

AWS IoT Core for LoRaWAN は、ワイヤレスゲートウェイとデバイスを に接続します AWS IoT Core。

- エンドポイント

  AWS IoT Core for LoRaWAN は、アカウントおよびリージョン固有の AWS IoT Core エンドポイントへのゲートウェイ接続を管理します。ゲートウェイは、 AWS IoT Core for LoRaWAN が提供するアカウントの Configuration and Update Server (CUPS) エンドポイントに接続できます。

エンドポイントの目的	エンドポイントフォーマット	提供
Configuration and Update Server (CUPS)	*account-specific-prefix* .cups.lorawan. *aws-region* .amazonaws.com:443	AWS IoT Core for LoRaWAN が提供する Configuration and Update Server とのゲートウェイ通信
LoRaWAN Network Server (LNS)	*account-specific-prefix* .gateway.lorawan. *aws-region* .amazonaws.com:443	AWS IoT Core for LoRaWAN が提供する LoRaWAN ネットワークサーバーとのゲートウェイ通信

- SDK

  AWS IoT Core for LoRaWAN が構築されている AWS IoT Wireless API は、 AWS SDK でサポートされています。詳細については、 「AWS SDK とツールキット」を参照してください。

- 認証

  AWS IoT Core for LoRaWAN デバイス通信では、X.509 証明書を使用して通信を保護します AWS IoT。

- 詳細情報

  ワイヤレスデバイスの設定と接続の詳細については、「AWS IoT Core for LoRaWAN リージョンとエンドポイント」の」を参照してください。

AWS IoT Core サービスエンドポイントに接続する

AWS IoT Core - コントロール プレーンの機能にアクセスするには AWS CLI、、任意の言語の AWS SDK、または REST API を直接呼び出します。 AWS サービスを呼び出すためのベストプラクティスが組み込まれ AWS IoT Core ているため、 AWS CLI または AWS SDK を使用して を操作することをお勧めします。REST API を直接呼び出すことはオプションですが、API へのアクセスを可能にするために必要なセキュリティ認証情報を提供する必要があります。

> **Note**
>
> IoT デバイスは、AWS IoT デバイス SDKs を使用する必要があります。Device SDKs は、デバイスでの使用に最適化されており、 との MQTT 通信をサポートし AWS IoT、デバイスで

最もよく使用される AWS IoT APIsをサポートしています。Device SDK とそれらが提供する機能の詳細については、AWS IoT デバイス SDKs を参照してください。
モバイルデバイスは、AWS モバイル SDKs を使用する必要があります。Mobile SDKs は、AWS IoT APIs、MQTT デバイス通信、およびモバイルデバイス上の他の AWS サービスの APIsをサポートします。Mobile SDK とそれらが提供する機能の詳細については、AWS モバイル SDKs を参照してください。

ウェブおよびモバイルアプリケーションで AWS Amplify ツールとリソースを使用して、より簡単に接続できます AWS IoT Core。Amplify AWS IoT Core を使用して に接続する方法の詳細については、Amplify ドキュメントのPubSub」を参照してください。

以下のセクションでは、 およびその他の SDKs について説明します。 AWS IoT AWS でアプリケーションを構築および管理できる AWS ツールと開発キットの完全なリストについては AWS、「構築するツール AWS」を参照してください。

AWS CLI の AWS IoT Core

AWS CLI は、 AWS APIs。

- インストール

  のインストール方法については AWS CLI、「 のインストール AWS CLI」を参照してください。
- 認証

  は の認証情報 AWS CLI を使用します AWS アカウント。
- リファレンス

  これらの AWS IoT Core サービスの AWS CLI コマンドの詳細については、以下を参照してください。
  - AWS CLI IoT のコマンドリファレンス
  - AWS CLI IoT データのコマンドリファレンス
  - AWS CLI IoT ジョブデータのコマンドリファレンス
  - AWS CLI IoT セキュアトンネリングのコマンドリファレンス

PowerShell スクリプト環境で AWS サービスとリソースを管理するツールについては、AWS「Tools for PowerShell」を参照してください。

AWS SDKs

AWS SDKs を使用すると、アプリケーションと互換性のあるデバイスは、 AWS IoT APIsと他の AWS サービスの APIs を呼び出すことができます。このセクションでは、 AWS SDKsへのリンクと、 AWS IoT Core サービスの API の APIs リファレンスドキュメントへのリンクを提供します。

AWS SDKsこれらの AWS IoT Core APIsをサポート

- AWS IoT
- AWS IoT データプレーン
- AWS IoT ジョブデータプレーン
- AWS IoT セキュアトンネリング
- AWS IoT ワイヤレス

C++

AWS SDK for C++ をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

1. AWS SDK for C++ の使用開始の手順に従ってください。

   次の手順では、次の方法について説明します。

   - ソースファイルから SDK をインストールしてビルドする
   - AWS アカウントで SDK を使用するための認証情報を提供する
   - アプリケーションまたはサービスで SDK を初期化してシャットダウンする
   - アプリケーションまたはサービスを構築するための CMake プロジェクトを作成する
2. サンプルアプリケーションを作成して実行します。 AWS SDK for C++ を使用するサンプルアプリケーションについては、「AWS SDK for C++ コード例」を参照してください。

が AWS SDK for C++ サポートする AWS IoT Core サービスのドキュメント

- AWS::IoTClient" リファレンスドキュメント
- Aws::IoTDataPlane::IoTDataPlaneClient リファレンスドキュメント
- Aws::IoTJobsDataPlane::IoTJobsDataPlaneClient リファレンスドキュメント
- Aws::IoTSecureTunneling::IoTSecureTunnelingClient リファレンスドキュメント

Go

AWS SDK for Go をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

1. 「 の開始方法」の手順に従ってください。 AWS SDK for Go

   次の手順では、次の方法について説明します。

   - のインストール AWS SDK for Go
   - AWS アカウントにアクセスするための SDK のアクセスキーを取得する
   - アプリケーションまたはサービスのソースコードにパッケージをインポートする
2. サンプルアプリケーションを作成して実行します。 AWS SDK for Goを使用するサンプルアプリケーションについては、「AWS SDK for Go コード例」を参照してください。

が AWS SDK for Go サポートする AWS IoT Core サービスのドキュメント

- IoT リファレンスドキュメント
- IoTDataPlane リファレンスドキュメント
- IoTJobsDataPlane リファレンスドキュメント
- IoTSecureTunneling リファレンスドキュメント

Java

AWS SDK for Java をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

1. 「 の開始方法」の手順に従ってください。 AWS SDK for Java 2.x

   次の手順では、次の方法について説明します。

   - にサインアップ AWS して IAM ユーザーを作成する
   - SDK をダウンロードする
   - AWS 認証情報とリージョンの設定
   - Apache Maven で SDK を使用する
   - Gradle とともに SDK を使用する
2. AWS SDK for Java 2.x コード例のいずれかを使用してサンプルアプリケーションを作成して実行する

3. SDK API リファレンスドキュメントを確認する

が AWS SDK for Java サポートする AWS IoT Core サービスのドキュメント

- IotClient リファレンスドキュメント
- IotDataPlaneClient リファレンスドキュメント
- IotJobsDataPlaneClient リファレンスドキュメント
- IoTSecureTunnelingClient リファレンスドキュメント

JavaScript

をインストール AWS SDK for JavaScript し、それを使用して に接続するには AWS IoT:

1. 「AWS SDK for JavaScriptの設定」の手順に従います。これらの手順は、ブラウザおよび Node.JS AWS SDK for JavaScript での の使用に適用されます。インストールに適用される指示に従ってください。

   次の手順では、次の方法について説明します。

   - 前提条件を確認します。
   - SDK for JavaScript をインストールする
   - SDK for JavaScript をロードする
2. ご使用の環境の開始方法のオプションで説明されているように、SDK の使用を開始するためのサンプルアプリケーションを作成して実行します。
   - AWS SDK for JavaScript in the Browse の使用を開始するか、または
   - AWS SDK for JavaScript in Node.js の使用を開始する

が AWS SDK for JavaScript サポートする AWS IoT Core サービスのドキュメント

- `AWS.Iot reference documentation`
- `AWS.IotData reference documentation`
- `AWS.IotJobsDataPlane reference documentation`
- `AWS.IotSecureTunneling reference documentation`

.NET

AWS SDK for .NET をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

1. AWS SDK for .NET 「環境のセットアップ」の手順に従います。
2. AWS SDK for .NET 「プロジェクトのセットアップ」の手順に従います。

   次の手順では、次の方法について説明します。
   - 新しいプロジェクトを開始する
   - AWS 認証情報の取得と設定
   - AWS SDK パッケージをインストールする
3. AWS SDK for .NET の AWS 「サービスの使用」のサンプルプログラムのいずれかを作成して実行する
4. SDK API リファレンスドキュメントを確認する

が AWS SDK for .NET サポートする AWS IoT Core サービスのドキュメント

- Amazon.IoT.Model リファレンスドキュメント
- Amazon.IotData.Model リファレンスドキュメント
- Amazon.IoTJobsDataPlane.Model リファレンスドキュメント
- Amazon.IoTSecureTunneling.Model リファレンスドキュメント

PHP

AWS SDK for PHP をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

1. 「 AWS SDK for PHP バージョン 3 の開始方法」の手順に従ってください。

   次の手順では、次の方法について説明します。
   - 前提条件を確認します。
   - SDK のインストール
   - SDK を PHP スクリプトに適用する
2. AWS SDK for PHP Version 3 コード例のいずれかを使用してサンプルアプリケーションを作成して実行する

が AWS SDK for PHP サポートする AWS IoT Core サービスのドキュメント

- IotClient リファレンスドキュメント
- IoTDataPlaneClient リファレンスドキュメント
- IoTJobsDataPlaneClient リファレンスドキュメント
- IoTSecureTunnelingClient リファレンスドキュメント

Python

AWS SDK for Python (Boto3) をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

1. 「AWS SDK for Python (Boto3) のクイックスタート」の指示に従います、

   次の手順では、次の方法について説明します。

   - SDK のインストール
   - SDK を設定する
   - コードで SDK を使用する
2. AWS SDK for Python (Boto3)を使用するサンプルプログラムを作成し、実行する

   このプログラムは、アカウントの現在設定されているログ記録オプションを表示します。SDK をインストールして、アカウントに合わせて設定したら、このプログラムを実行できます。

   ```
   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))
   ```

   この例で使用されている機能の詳細については、the section called “AWS IoT ログ記録の設定” を参照してください。

が AWS SDK for Python (Boto3) サポートする AWS IoT Core サービスのドキュメント

- IoT リファレンスドキュメント
- IoTDataPlane リファレンスドキュメント
- IoTJobsDataPlane リファレンスドキュメント
- IoTSecureTunneling リファレンスドキュメント

Ruby

AWS SDK for Ruby をインストールし、それを使用して AWS IoTに接続するには以下のようにします。

- 「 の開始方法」の手順に従ってください。 AWS SDK for Ruby

  次の手順では、次の方法について説明します。

  - SDK のインストール
  - SDK を設定する
- Hello World チュートリアルを作成して実行する

AWS SDK for Ruby がサポートする AWS IoT Core サービスのドキュメント

- Aws::IoT::Client リファレンスドキュメント
- Aws::IoTDataPlane::Client リファレンスドキュメント
- Aws::IoTJobsDataPlane::Client リファレンスドキュメント
- Aws::IoTSecureTunneling::Client リファレンスドキュメント

AWS モバイル SDKs

AWS Mobile SDKs は、モバイルアプリ開発者に、 AWS IoT Core サービスの APIs、MQTT を使用した IoT デバイス通信、およびその他の AWS サービスの APIsプラットフォーム固有のサポートを提供します。

Android

AWS Mobile SDK for Android

AWS Mobile SDK for Android には、開発者が を使用して接続されたモバイルアプリケーションを構築するためのライブラリ、サンプル、ドキュメントが含まれています AWS。この SDK には、MQTT デバイス通信のサポートと、 AWS IoT Core サービスの APIsの呼び出しも含まれています。詳細については次を参照してください:

- AWS Mobile SDK for Android on GitHub
- AWS Android Readme 用 Mobile SDK
- AWS Mobile SDK for Android サンプル
- AWS SDK for Android API リファレンス
- AWSIoTClient Class リファレンスドキュメント

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS は、Apache オープンソースライセンスで配布されるオープンソースのソフトウェア開発キットです。SDK for iOS は、開発者が を使用して接続されたモバイルアプリケーションを構築するのに役立つライブラリ、コードサンプル、ドキュメントを提供します AWS。この SDK には、MQTT デバイス通信のサポートと、 AWS IoT Core サービスの APIsの呼び出しも含まれています。詳細については次を参照してください:

- AWS Mobile SDK for iOS GitHub の
- AWS SDK for iOS Readme
- AWS SDK for iOS サンプル
- AWS IoTAWS SDK for iOS のクラスリファレンスドキュメント

AWS IoT Core サービスの REST APIs

AWS IoT Core サービスの REST APIs は、HTTP リクエストを使用して直接呼び出すことができます。

- エンドポイント URL

  AWS IoT Core サービスの REST API を公開するサービスエンドポイントはリージョンによって異なり、「AWS IoT Core エンドポイントとクォータ」にリストされています。 AWS IoT リソースはリージョン固有であるため、アクセスする AWS IoT リソースがあるリージョンのエンドポイントを使用する必要があります。

- 認証

  AWS IoT Core サービスの REST APIs、認証に AWS IAM 認証情報を使用します。詳細については、 AWS 「 全般のリファレンスAWS 」の「 API リクエストの署名」を参照してください。

- API リファレンス

  AWS IoT Core サービスの REST APIs によって提供される特定の関数については、以下を参照してください。

  - IoT 用の API リファレンス。
  - IoT データの API リファレンス。
  - IoT ジョブデータの API リファレンス。
  - IoT セキュアトンネリングの API リファレンス。

デバイスを に接続する AWS IoT

デバイスは、 AWS IoT およびその他の サービスに接続します AWS IoT Core。を通じて AWS IoT Core、デバイスは アカウントに固有のデバイスエンドポイントを使用してメッセージを送受信します。MQTT および WSS プロトコルを使用した the section called “AWS IoT デバイス SDKs” サポートデバイス通信。デバイスが使用できるプロトコルの詳細については、「the section called “デバイス通信プロトコル”」を参照してください。

メッセージブローカー

AWS IoT は、メッセージブローカーを介したデバイス通信を管理します。デバイスおよびクライアントは、メッセージブローカーにメッセージを発行するとともに、メッセージブローカーが発行するメッセージにサブスクライブします。メッセージは、アプリケーション定義のトピックによって識別されます。メッセージブローカーは、デバイスまたはクライアントによって発行されたメッセージを受信すると、そのメッセージのトピックにサブスクライブしているデバイスとクライアントにそのメッセージを再発行します。メッセージブローカーは、メッセージの内容に応じて動作できる AWS IoT ルールエンジンにもメッセージを転送します。

AWS IoT メッセージセキュリティ

認証 AWS IoT に使用するデバイス接続the section called “X.509 クライアント証明書”とAWS 署名 V4。デバイス通信は TLS バージョン 1.3 で保護されており、デバイスは接続時に Server Name Indication (SNI) 拡張機能を送信 AWS IoT する必要があります。詳細については、「 のトランスポートセキュリティ AWS IoT」を参照してください。

AWS IoT デバイスデータとサービスエンドポイント

> **Important**
>
> エンドポイントはデバイスにキャッシュまたは保存できます。このため、新しいデバイスが接続されるたびに `DescribeEndpoint` API にクエリを実行する必要はありません。がアカウントのエンドポイント AWS IoT Core を作成した後も、エンドポイントは変更されません。

各アカウントには、アカウントに固有で、特定の IoT 機能をサポートする複数のデバイスエンドポイントがあります。 AWS IoT デバイスデータエンドポイントは、IoT デバイスの通信ニーズに合わせて設計されたパブリッシュ/サブスクライブプロトコルをサポートします。ただし、アプリケーションがこれらのエンドポイントが提供する特殊な機能を必要とする場合は、アプリケーションやサービスなどの他のクライアントもこのインターフェイスを使用できます。 AWS IoT デバイスサービスエンドポイントは、セキュリティおよび管理サービスへのデバイス中心のアクセスをサポートします。

アカウントのデバイスデータエンドポイントについては、 AWS IoT Core コンソールの設定ページで確認できます。

デバイスデータエンドポイントなど、特定の目的のためにアカウントのデバイスエンドポイントを知るには、ここに示されている describe-endpoint CLI コマンドまたは `DescribeEndpoint` REST API を使用して、次の表の *`endpointType`* パラメータ値を指定します。

```
aws iot describe-endpoint --endpoint-type endpointType
```

このコマンドは、次の形式で *`iot-endpoint`* を返します: *`account-specific-prefix`*`.iot.`*`aws-region`*`.amazonaws.com`。

すべてのカスタマーには `iot:Data-ATS` および `iot:Data` エンドポイントがあります。各エンドポイントは X.509 証明書を使用して、クライアントを認証します。Symantec 認証機関の広範な不信用に関連する問題を避けるために、新しい `iot:Data-ATS` エンドポイントタイプを使用することを強くお勧めします。下位互換性のために VeriSign 証明書を使用する古いエンドポイントからデータを取得するための `iot:Data` エンドポイントをデバイスに提供します。詳細については、「サーバーの認証」を参照してください。

AWS IoT デバイスの エンドポイント

エンドポイントの目的	***endpointType*** 値	説明
AWS IoT Core- データプレーンオペレーション	`iot:Data-ATS`	メッセージブローカー、Device Shadow、および AWS IoTのルールエンジンのコンポーネント間のデータを送受信するために使用されました。 `iot:Data-ATS` は、ATS 署名付きデータエンドポイントを返します。
AWS IoT Core– データプレーンオペレーション (レガシー)	`iot:Data`	`iot:Data` は、下位互換性のために提供された VeriSign 署名付きデータエンドポイントを返します。MQTT 5 は Symantec (`iot:Data`) エンドポイントではサポートされていません。
AWS IoT Core 認証情報アクセス	`iot:CredentialProvider`	直接他の AWS サービスへ接続するために、デバイス組み込みの X.509 証明書を一時的な認証情報と交換するために使用されます。他の AWS サービスへの接続の詳細については、「AWS サービスへの直接呼び出しの承認」を参照してください。
AWS IoT Device Management– ジョブデータオペレーション	`iot:Jobs`	デバイスが AWS IoT Jobs Device HTTPS APIs を使用して Jobs サービスとやり取りできるようにするために使用されます。

エンドポイントの目的	*endpointType* 値	説明
		iot:Jobs は、IPv4 にのみ使用可能です。デュアルスタックエンドポイント (IPv4 と IPv6) を使用している場合は、iot:Data-ATS エンドポイントタイプを使用します。
AWS IoT Device Advisor オペレーション	iot:DeviceAdvisor	Device Advisor を使用してデバイスをテストするために使用されるテストエンドポイントタイプ。詳細については、「???」を参照してください。
AWS IoT Core データベータ (プレビュー)	iot:Data-Beta	ベータリリース用に予約されたエンドポイントの種類。現在の使用については、??? を参照してください。

example.com などの独自の完全修飾ドメイン名 (FQDN) および関連するサーバー証明書を使用して、 AWS IoT を使用してデバイスを に接続することもできますthe section called “ドメイン設定”。

AWS IoT デバイス SDKs

AWS IoT Device SDKs は、IoT デバイスを に接続 AWS IoT Core し、MQTT および MQTT over WSS プロトコルをサポートします。

AWS IoT Device SDKs は、 AWS IoT Device AWS SDKs が IoT デバイスの特殊な通信ニーズをサポートしているが、SDK でサポートされているすべてのサービスをサポートしていないという点で、 AWS SDKs。 SDKs AWS IoT Device SDKs は、すべての AWS サービスをサポートする AWS SDKs と互換性がありますが、異なる認証方法を使用し、異なるエンドポイントに接続するため、IoT デバイスで AWS SDKs を使用することが実用的でない可能性があります。

モバイルデバイス

は、MQTT デバイス通信、一部の AWS IoT サービス APIs、および他の AWS サービスの APIsの両方the section called “AWS モバイル SDKs”をサポートしています。サポートされているモバイルデ

バイスで開発している場合は、その SDK を確認して、それが IoT ソリューションの開発に最適なオプションであるかどうかを確認してください。

C++

AWS IoT C++ デバイス SDK

AWS IoT C++ Device SDK を使用すると、開発者は AWS と AWS IoT Core サービスの APIs を使用して接続されたアプリケーションを構築できます。具具体的には、この SDK にはリソース制約がなく、メッセージキュー、マルチスレッドサポート、最新の言語機能などの高度な機能が必要なデバイス向けに設計されています。詳細については、以下を参照してください:

- AWS IoT GitHub の Device SDK C++ v2
- AWS IoT Device SDK C++ v2 Readme
- AWS IoT Device SDK C++ v2 サンプル
- AWS IoT Device SDK C++ v2 API ドキュメント

Python

AWS IoT Device SDK for Python

AWS IoT Device SDK for Python を使用すると、開発者は Python スクリプトを記述して、デバイスを使用して MQTT または WebSocket Secure (WSS) プロトコル経由で AWS IoT プラットフォームにアクセスできます。デバイスを AWS IoT Core サービスの APIs に接続することで、ユーザーは AWS IoT Core が提供するメッセージブローカー、ルール、Device Shadow サービス、および AWS Lambda、Amazon Kinesis、Amazon S3 などの他の AWS サービスを安全に操作できます。

- AWS IoT GitHub の Device SDK for Python v2
- AWS IoT Device SDK for Python v2 Readme
- AWS IoT Device SDK for Python v2 サンプル
- AWS IoT Device SDK for Python v2 API ドキュメント

JavaScript

AWS IoT Device SDK for JavaScript

AWS IoT Device SDK for JavaScript を使用すると、開発者は WebSocket プロトコルを介して MQTT または MQTT AWS IoT Core を使用して の APIs にアクセスする JavaScript アプリケーションを記述できます。これは、Node.js 環境およびブラウザーアプリケーションで使用できます。詳細については、以下を参照してください。

- AWS IoT GitHub の Device SDK for JavaScript v2
- AWS IoT Device SDK for JavaScript v2 Readme
- AWS IoT Device SDK for JavaScript v2 サンプル
- AWS IoT Device SDK for JavaScript v2 API ドキュメント

Java

AWS IoT Device SDK for Java

AWS IoT Device SDK for Java を使用すると、Java 開発者は WebSocket プロトコルを介して MQTT または MQTT AWS IoT Core を介して の APIs にアクセスできます。SDK では、デバイスシャドウサービスがサポートされています。GET、UPDATE、DELETE を含む HTTP メソッドを使用して、 Shadows にアクセスできます。SDK では、簡略化された Shadow アクセスモデルもサポートしていて、デベロッパーが、JSON ドキュメントをシリアル化または逆シリアル化することなく、ゲッターメソッドとセッターメソッドを使用して Shadows とデータを交換できます。詳細については、以下を参照してください。

- AWS IoT GitHub の Device SDK for Java v2
- AWS IoT Device SDK for Java v2 Readme
- AWS IoT Device SDK for Java v2 サンプル
- AWS IoT Device SDK for Java v2 API ドキュメント

Embedded C

AWS IoT Device SDK for Embedded C

> **Important**
>
> この SDK は、経験豊富な組み込みソフトウェアデベロッパーによる使用を想定しています。

AWS IoT Device SDK for Embedded C (C-SDK) は、MIT オープンソースライセンスに基づく C ソースファイルのコレクションであり、埋め込みアプリケーションで IoT デバイスを IoT AWS IoT Core に安全に接続するために使用できます。これには、MQTT、JSON Parser、 AWS IoT Device Shadow ライブラリなどが含まれます。これはソース形式で配布され、アプリケーションコード、その他のライブラリ、およびオプションで RTOS (Real Time Operating System) とともにお客様のファームウェアに組み込まれることが意図されています。

は通常 AWS IoT Device SDK for Embedded C 、最適化された C 言語ランタイムを必要とするリソースに制約のあるデバイスを対象としています。この SDK は、任意のオペレーティングシステムで使用でき、任意のプロセッサタイプ (MCU や MPU など) でホストできます。デバイスに十分なメモリと処理リソースがある場合は、 AWS IoT Device SDKs for C++、Java、JavaScript、Python など、他の AWS IoT Device および Mobile SDK のいずれかを使用することをお勧めします。

詳細については次を参照してください:

- AWS IoT GitHub の Device SDK for Embedded C
- AWS IoT Device SDK for Embedded C Readme
- AWS IoT Device SDK for Embedded C サンプル

デバイス通信プロトコル

AWS IoT Core は、MQTT および MQTT over WebSocket Secure (WSS) プロトコルを使用してメッセージを発行およびサブスクライブするデバイスとクライアント、および HTTPS プロトコルを使用してメッセージを発行するデバイスとクライアントをサポートします。すべてのプロトコルでは、IPv4 および IPv6 がサポートされています。このセクションでは、デバイスおよびクライアントのさまざまな接続オプションについて説明します。

TLS プロトコルバージョン

AWS IoT Core は、TLS バージョン 1.2 と TLS バージョン 1.3 を使用してすべての通信を暗号化します。ドメイン設定で TLS 設定を設定することで、エンドポイントに追加の TLS ポリシーバージョンを設定できます。デバイスを AWS IoT Coreに接続すると、クライアントはサーバー名表示 (SNI) 拡張機能を送信できます。これは、マルチアカウント登録、設定可能なエンドポイント、カスタムドメイン、VPC エンドポイントなどの機能に必要です。詳細については、「AWS IoTのトランスポートセキュリティ」を参照してください。

AWS IoT デバイス SDKs は、MQTT および MQTT over WSS をサポートし、クライアント接続のセキュリティ要件をサポートします。AWS IoT デバイス SDKs を使用してクライアントを AWS IoTに接続することをお勧めします。

プロトコル、ポートマッピング、認証

デバイスまたはクライアントがメッセージブローカーに接続する方法は、[認証タイプ] を使用して設定できます。デフォルトでは、または SNI 拡張機能が送信されない場合、認証方法は、デバイスが使用するアプリケーションプロトコル、ポート、および Application Layer Protocol Negotiation (ALPN) TLS 拡張機能に基づいています。次の表に、ポート、ポート、ALPN に基づいて予想される認証を示します。

プロトコル、認証、ポートマッピング

プロトコル	サポートされているオペレーション	認証	ポート	ALPN プロトコル名
MQTT over WebSocket	発行、サブスクライブ	署名バージョン 4	443	該当なし
MQTT over WebSocket	発行、サブスクライブ	カスタム認証	443	該当なし
MQTT	発行、サブスクライブ	X.509 クライアント証明書	443†	`x-amzn-mqtt-ca`
MQTT	発行、サブスクライブ	X.509 クライアント証明書	8883	該当なし
MQTT	発行、サブスクライブ	カスタム認証	443†	`mqtt`
HTTPS	発行のみ	署名バージョン 4	443	該当なし
HTTPS	発行のみ	X.509 クライアント証明書	443†	`x-amzn-http-ca`

プロトコル	サポートされているオペレーション	認証	ポート	ALPN プロトコル名
HTTPS	発行のみ	X.509 クライアント証明書	8443	該当なし
HTTPS	発行のみ	カスタム認証	443	該当なし

Application Layer Protocol Negotiation (ALPN)

† デフォルトのエンドポイント設定を使用するとき、ポート 443 で X.509 クライアント証明書認証を使用して接続するクライアントは、Application Layer Protocol Negotiation (ALPN) TLS 拡張を実装し、`ClientHello` メッセージの一部としてクライアントから送信された ALPN ProtocolNameList にリストされている ALPN プロトコル名を使用する必要があります。

ポート 443 では、IoT:Data-ATS エンドポイントは ALPN x-amzn-http-ca HTTP をサポートしますが、IoT:Jobs エンドポイントはサポートしません。

ALPN x-amzn-mqtt-ca を使用するポート 8443 HTTPS およびポート 443 MQTT では、カスタム認証を使用できません。

クライアントは AWS アカウント、デバイスのエンドポイントに接続します。アカウントのデバイスエンドポイントを見つける方法については、the section called “AWS IoT デバイスデータとサービスエンドポイント” を参照してください。

Note

AWS SDKs は URL 全体を必要としません。`pubsub.py` GitHub の AWS IoT Device SDK for Python のサンプルなどのエンドポイントホスト名のみが必要です。次の表にあるような完全な URL を渡すと、無効なホスト名などのエラーが発生する可能性があります。

への接続 AWS IoT Core

プロトコル	エンドポイントまたは URL
MQTT	*iot-endpoint*
MQTT over WSS	wss://*iot-endpoint* /mqtt
HTTPS	https://*iot-endpoint* /topics

デバイス通信用のアプリケーションプロトコルの選択

デバイスエンドポイントを介したほとんどの IoT デバイス通信では、Secure MQTT または MQTT over WebSocket Secure (WSS) プロトコルを使用することができます。ただし、デバイスエンドポイントは HTTPS もサポートします。

次の表は、 がデバイス通信に 2 つの高レベルプロトコル (MQTT と HTTPS) AWS IoT Core を使用する方法を比較したものです。

AWS IoT デバイスプロトコル (MQTT と HTTPS) side-by-side

機能	MQTT	HTTPS
発行/サブスクライブのサポート	発行とサブスクライブ	発行のみ
SDK サポート	AWS Device SDKs MQTT プロトコルと WSS プロトコルをサポート	SDK のサポートはありませんが、言語固有のメソッドを使用して HTTPS リクエストを行うことができます
サービス品質のサポート	MQTT QoS レベル 0 および 1	QoS は、クエリ文字列パラメータを渡すことによってサポートされ、?qos=qosここで、値は0または1にすることができます。このクエリ文字列を追加して、必要な QoS 値を持つメッセージが発行できます。

機能	MQTT	HTTPS
デバイスがオフラインのときに受信しなかったメッセージを受信できる	はい	いいえ
clientId フィールドのサポート	はい	いいえ
デバイスの切断検出	はい	いいえ
安全な通信	はい。「???」を参照してください。	はい。「???」を参照してください。
トピックの定義	定義されているアプリケーション	定義されているアプリケーション
メッセージデータ形式	定義されているアプリケーション	定義されているアプリケーション
プロトコルのオーバーヘッド	Lower	より高い
消費電力	Lower	より高い

デバイス通信の認証タイプの選択

設定可能なエンドポイントを使用して、IoT エンドポイントの認証タイプを設定できます。または、デフォルト設定を使用して、デバイスがアプリケーションプロトコル、ポート、ALPN TLS 拡張機能の組み合わせで認証する方法を決定します。選択した認証タイプによって、接続時のデバイスの認証方法が決まります AWS IoT Core。5 つの認証タイプがあります。

X.509 証明書

デバイスを認証するために AWS IoT Core 検証する X.509 クライアント証明書を使用してデバイスを認証します。この認証タイプは、Secure MQTT (MQTT over TLS) および HTTPS プロトコルで機能します。

カスタムオーソライザーを使用した X.509 証明書

X.509 クライアント証明書を使用してデバイスを認証し、X.509 クライアント証明書情報を受信するカスタムオーソライザーを使用して追加の認証アクションを実行します。この認証タイプは、Secure MQTT (MQTT over TLS) および HTTPS プロトコルで機能します。この認証タイプは、X.509 カスタム認証で設定可能なエンドポイントでのみ使用できます。ALPN オプションはありません。

AWS 署名バージョン 4 (SigV4)

Cognito またはバックエンドサービスを使用してデバイスを認証し、ソーシャルフェデレーションとエンタープライズフェデレーションをサポートします。この認証タイプは、MQTT over WebSocket Secure (WSS) および HTTPS プロトコルで機能します。

カスタムオーソライザー

Lambda 関数を設定してデバイスを認証し、 AWS IoT Coreに送信されるカスタム認証情報を処理します。この認証タイプは、Secure MQTT (MQTT over TLS)、HTTPS、および MQTT over WebSocket Secure (WSS) プロトコルで機能します。

デフォルト

デバイスが使用するポートおよび/または Application Layer Protocol Negotiation (ALPN) 拡張機能に基づいてデバイスを認証します。一部の追加の認証オプションはサポートされていません。詳細については、「???」を参照してください。

以下の表は、サポートされている認証タイプとアプリケーションプロトコルのすべての組み合わせを示しています。

サポートされている認証タイプとアプリケーションプロトコルの組み合わせ

認証タイプ	Secure MQTT (MQTT over TLS)	MQTT over WebSocket Secure (WSS)	HTTPS	デフォルト
X.509 証明書	✓		✓	
カスタムオーソライザーを使用した X.509 証明書	✓		✓	

認証タイプ	Secure MQTT (MQTT over TLS)	MQTT over WebSocket Secure (WSS)	HTTPS	デフォルト
AWS 署名バージョン 4 (SigV4)		✓	✓	
カスタムオーソライザー	✓	✓	✓	
デフォルト				✓

接続時間の制限

HTTPS 接続では、リクエストの受信と応答にかかるよりも長い持続時間が保証されていません。

MQTT 接続時間は、使用する認証機能によって異なります。次の表では、各機能の最適な条件下での最大接続時間が一覧表示されています。

認証機能による MQTT 接続時間

機能	最大時間 *
X.509 クライアント証明書	1～2 週間
カスタム認証	1～2 週間
署名バージョン 4	最長 24 時間

* 保証なし

X.509 証明書やカスタム認証では、接続時間に厳しい制限はありません。数分程度に短縮することも可能です。接続の中断は、さまざまな理由で発生します。次のリストには、最も一般的な理由の一部が含まれています。

- Wi-Fi 可用性の中断
- インターネットサービスプロバイダー (ISP) 接続の中断
- サービスのパッチ

- サービスのデプロイ
- サービスのオートスケーリング
- 使用できないサービスホスト
- ロードバランサーの問題および更新
- クライアント側エラー

デバイスで、切断を検出して再接続するための戦略を実装する必要があります。接続解除イベントとその処理方法に関するガイダンスについては、???の???を参照してください。

MQTT

MQTT (Message Queuing Telemetry Transport) は軽量で広く採用されているメッセージングプロトコルであり、制約のあるデバイス向けに設計されています。MQTT の AWS IoT Core サポートは、the section called “AWS IoT MQTT 仕様との違い” に記載されているように、MQTT v3.1.1 の仕様と MQTT v5.0 の仕様に基づいていますが、いくつかの違いがあります。標準の最新バージョンとして、MQTT 5 には、新しいスケーラビリティの強化、理由コード応答によるエラー報告の改善、メッセージとセッションの有効期限タイマー、カスタムユーザーメッセージヘッダーなど、MQTT ベースのシステムをより堅牢にするいくつかの重要な機能が導入されています。が AWS IoT Core サポートする MQTT 5 機能の詳細については、「MQTT 5 でサポートされている機能」を参照してください。 は、クロス MQTT バージョン (MQTT 3 および MQTT 5) 通信 AWS IoT Core もサポートしています。MQTT 3 パブリッシャーは、MQTT 5 パブリッシュメッセージを受信する MQTT 5 サブスクライバーに MQTT 3 メッセージを送信できます。その逆も可能です。

AWS IoT Core は、MQTT プロトコルと MQTT over WSS プロトコルを使用し、クライアント ID で識別されるデバイス接続をサポートします。AWS IoT デバイス SDKsは両方のプロトコルをサポートしており、デバイスを AWS IoT Coreに接続するための推奨される方法です。 AWS IoT Device SDKs、デバイスとクライアントが サービスに接続してアクセス AWS IoT するために必要な機能をサポートしています。Device SDKs、 AWS IoT サービスが必要とする認証プロトコルと、MQTT プロトコルと MQTT over WSS プロトコルが必要とする接続 ID 要件をサポートします。 AWS Device SDK AWS IoT を使用して に接続する方法と、サポートされている言語 AWS IoT の の例へのリンクについては、「」を参照してくださいthe section called “AWS IoT Device SDKs を使用した MQTT との接続”。 SDKs MQTT メッセージの認証メソッドとポートマッピングの詳細については、??? を参照してください。

Device AWS IoT SDKs を使用して に接続することをお勧めしますが AWS IoT、必須ではありません。ただし、 AWS IoT Device SDKs を使用しない場合は、必要な接続と通信のセキュリティを提供する必要があります。クライアントは、接続リクエストで Server Name Indication (SNI) TLS

extension を送信する必要があります。SNI を含まない接続試行は拒否されます。詳細については、「 のトランスポートセキュリティ AWS IoT」を参照してください。IAM ユーザーと AWS 認証情報を使用してクライアントを認証するクライアントは、正しい署名バージョン 4 認証を提供する必要があります。

クライアントが接続されると、API を使用して MQTT クライアント接続をモニタリングおよび管理できます。詳細については、「the section called “MQTT 接続の管理”」を参照してください。

このトピックの内容

- AWS IoT Device SDKs を使用した MQTT との接続
- MQTT Quality of Service (QoS) オプション
- MQTT 永続的セッション
- 保持された MQTT メッセージ
- MQTT の Last Will and Testament (LWT) メッセージ
- connectAttributes の使用
- MQTT 5 がサポートしている機能
- MQTT 5 プロパティ
- MQTT 理由コード
- AWS IoT MQTT 仕様との違い
- MQTT 接続の管理

AWS IoT Device SDKs を使用した MQTT との接続

このセクションでは、 AWS IoT Device SDKsと、デバイスの接続方法を示すサンプルプログラムのソースコードへのリンクを示します AWS IoT。ここでリンクされているサンプルアプリは、MQTT プロトコルと MQTT over WSS AWS IoT を使用して に接続する方法を示しています。

> **Note**
>
> AWS IoT Device SDKsは MQTT 5 クライアントをリリースしました。

C++

C++ Device SDK AWS IoT を使用したデバイスの接続

- C++ での MQTT 接続例を示すサンプルアプリケーションのソースコード

- AWS IoT GitHub の Device SDK for C++ v2

Python

AWS IoT Device SDK for Python を使用したデバイスの接続

- Python での MQTT 接続例を示すサンプルアプリケーションのソースコード
- AWS IoT GitHub の Device SDK v2 for Python

JavaScript

AWS IoT Device SDK for JavaScript を使用したデバイスの接続

- JavaScriptでの MQTT 接続例を示すサンプルアプリケーションのソースコード
- GitHub のAWS IoT Device SDK for JavaScript v2

Java

AWS IoT Device SDK for Java を使用したデバイスの接続

> **Note**
>
> AWS IoT Device SDK for Java v2 が Android 開発をサポートするようになりました。詳細については、「AWS IoT Device SDK for Android」を参照してください。

- Java での MQTT 接続例を示すサンプルアプリケーションのソースコード
- AWS IoT GitHub の Device SDK for Java v2

Embedded C

AWS IoT Device SDK for Embedded C を使用したデバイスの接続

> **Important**
>
> この SDK は、経験豊富な組み込みソフトウェアデベロッパーによる使用を想定しています。

- Embedded C での MQTT 接続例を示すサンプルアプリケーションのソースコード
- AWS IoT GitHub の Device SDK for Embedded C

MQTT Quality of Service (QoS) オプション

AWS IoT および AWS IoT Device SDKs、MQTT Quality of Service (QoS) レベル0と 1をサポートしています。MQTT プロトコルは、第 3 レベルの QoS、レベル を定義しますが2、サポート AWS IoT していません。MQTT プロトコルのみが QoS 機能をサポートします。HTTPS は、クエリ文字列パラメータ `?qos=qos` を渡すことによって QoS をサポートし、ここで、値は 0 または 1 にすることができます。

この表は、各 QoS レベルが、メッセージブローカーに発行されたメッセージ、およびメッセージブローカーによって発行されたメッセージにどのように影響するかを示しています。

次の QoS レベルを使用..。	メッセージは..。	コメント
QoS レベル 0	送信回数 0 回以上	このレベルは、信頼できる通信リンクを介して送信されるメッセージや、見逃しても問題がないメッセージに使用する必要があります。
QoS レベル 1	少なくとも 1 回送信され、PUBACK 応答が受信されるまで繰り返し送信されます	送信者が正常に配信されたことを示す PUBACK 応答を受信するまで、メッセージは完了したとはみなされません。

MQTT 永続的セッション

永続セッションは、クライアントによって承認されていない、Quality of Service (QoS) が 1 のクライアントのサブスクリプションとメッセージを保存します。コネクテッドデバイスが永続セッションに再接続すると、セッションが再開され、そのサブスクリプションが復元され、再接続前に受信され、クライアントによって確認されていないサブスクライブされたメッセージがクライアントに送信されます。

保存されたメッセージの処理は、CloudWatch メトリクスと CloudWatch Logs に記録されます。CloudWatch と CloudWatch Logs に書き込まれるエントリについては、「メッセージブローカーのメトリクス」および「キューに保存されたログエントリ」を参照してください。

永続セッションの作成

MQTT 3 で、永続セッションを作成するには、CONNECT メッセージを送信して、cleanSession フラグを 0 に設定します。CONNECT メッセージを送信したクライアントのセッションが存在しない場合は、新しい永続セッションが作成されます。クライアントのセッションが既に存在する場合は、クライアントは既存のセッションを再開します。クリーンセッションを作成するには、CONNECT メッセージを送信して cleanSession フラグを 1 に設定します。クライアントが切断してもブローカーはセッション状態を保存しません。

MQTT 5 では、Clean Start フラグと Session Expiry Interval を設定することで永続セッションを処理します。クリーンスタートは、接続セッションの開始と前のセッションの終了を制御します。Clean Start = 1 を設定すると、新しいセッションが作成され、以前のセッションが存在する場合は終了します。Clean Start = 0 を設定すると、接続セッションは以前のセッションが存在する場合はそれを再開します。セッションの有効期限間隔は、接続セッションの終了を制御します。セッション有効期限間隔は、セッションが切断後も持続する時間を秒単位 (4 バイト整数) で指定します。Session Expiry interval = 0 に設定すると、セッションは切断時にすぐに終了します。セッションの有効期限が CONNECT メッセージで指定されていない場合、デフォルトは 0 です。

MQTT 5 クリーンスタートとセッションの有効期限

プロパティ値	説明
Clean Start= 1	新しいセッションを作成し、以前のセッションが存在する場合は終了します。
Clean Start= 0	以前のセッションが存在する場合、セッションを再開します。
Session Expiry Interval> 0	セッションを保持する。
Session Expiry interval= 0	セッションを保持しない。

MQTT 5 では、Clean Start = 1 と Session Expiry Interval = 0 を設定すると、これは MQTT 3 のクリーンセッションと同等になります。Clean Start=0 と Session Expiry Interval > 0 を設定すると、これは MQTT 3 の永続セッションと同等になります。

> **Note**
>
> 複数の MQTT バージョン (MQTT 3 と MQTT 5) の永続セッションはサポートされていません。MQTT 3 永続セッションを MQTT 5 セッションとして再開することはできません。その逆も同様です。

永続セッション中の操作

クライアントは、connection acknowledged (CONNACK) メッセージの `sessionPresent` 属性を調べて、永続的セッションが存在するかどうかを確認します。`sessionPresent` が `1` の場合、永続セッションが存在し、クライアントの保存済みメッセージは、クライアントが CONNACK を受信した後にクライアントに配信されます。これについては、「永続セッションへの再接続後のメッセージトラフィック」を参照してください。`sessionPresent` が `1` の場合、クライアントは再サブスクライブする必要はありません。ただし、`sessionPresent` が `0` である場合は、永続的セッションが存在しないため、クライアントはそのトピックフィルターに再度サブスクライブする必要があります。

クライアントは永続セッションに参加した後も、各オペレーションにフラグを追加することなく、メッセージの発行とトピックフィルターのサブスクライブを行うことができます。

永続セッションへの再接続後のメッセージトラフィック

永続的セッションは、クライアントと MQTT メッセージブローカーの間の継続的な接続を表します。クライアントが永続的セッションを使用してメッセージブローカーに接続すると、クライアントが接続中に作成するすべてのサブスクリプションがメッセージブローカーによって保存されます。クライアントの接続が切断されると、クライアントがサブスクライブしているトピックにパブリッシュされた未確認の QoS 1 メッセージと新しい QoS 1 メッセージが保存されます。メッセージはアカウントの制限に従って保存されます。制限を超えるメッセージは削除されます。永続的なメッセージの制限の詳細については、「AWS IoT Core のエンドポイントとクォータ」を参照してください。クライアントが永続的セッションに再接続すると、すべてのサブスクリプションが回復され、保存されているすべてのメッセージがクライアントに送信されます。その際の最大レートは 1 秒あたり 10 メッセージです。MQTT 5 では、クライアントがオフラインのときにメッセージ有効期限が設定されたアウトバウンド QoS 1 が期限切れになった場合、接続が再開された後、クライアントは期限切れメッセージを受信しません。

再接続後、保存されたメッセージは、`Publish requests per second per connection` 制限に達するまで、現在のメッセージトラフィックとともに、1 秒あたり 10 個の保存されたメッセージに制限されたレートでクライアントに送信されます。保存されたメッセージの配信レートは制限され

ているため、再接続後にセッションに 10 個を超える保存されたメッセージがある場合、すべての保存されたメッセージを配信するには数秒かかります。

共有サブスクライバーの場合、グループの少なくとも 1 人のサブスクライバーが永続セッションを使用し、QoS 1 メッセージを受信するオンラインのサブスクライバーがいない場合、メッセージはキューに入れられます。メッセージのキュー解除は、最大速度 (グループ内のアクティブなサブスクライバーごとに 1 秒あたり 20 件のメッセージ) で行われます。詳細については、「共有サブスクリプションのメッセージキュー機能」を参照してください。

永続セッションの終了

永続セッションは、次の方法で終了できます。

- 永続セッションの有効期限が経過した。永続セッションの有効期限タイマーは、クライアントの切断または接続のタイムアウトによってクライアントが切断されたことをメッセージブローカーが検出すると開始されます。
- クライアントが `cleanSession` フラグを 1 に設定する `CONNECT` メッセージを送信した。
- クライアントを手動で切断し、`DeleteConnection` API を使用してセッションをクリアします。詳細については、「the section called “MQTT 接続の管理”」を参照してください。

MQTT 3 では、永続セッションの有効期限のデフォルト値は 1 時間で、これはアカウント内のすべてのセッションに適用されます。

MQTT 5 では、CONNECT パケットと DISCONNECT パケットのセッション有効期限間隔をセッションごとに設定できます。

DISCONNECT パケットのセッション有効期限間隔について:

- 現在のセッションのセッション有効期限間隔が 0 の場合、DISCONNECT パケットのセッション有効期限間隔を 0 より大きい値に設定することはできません。
- 現在のセッションのセッション有効期限間隔が 0 より大きく、DISCONNECT パケットのセッション有効期限間隔を 0 に設定した場合、セッションは DISCONNECT で終了します。
- そうしないと、DISCONNECT パケットのセッション有効期限間隔が現在のセッションのセッション有効期限間隔を更新します。

> **Note**
>
> セッションの終了時にクライアントに送信されるのを待機している保存済みメッセージは破棄されます。ただし、送信できなかった場合でも、標準のメッセージングレートで請求されます。メッセージの料金の詳細については、「AWS IoT Core の料金」を参照してください。有効期限の時間間隔を設定できます。

永続セッションの有効期限が切れた後の再接続

有効期限が切れる前にクライアントが永続セッションに再接続しない場合、セッションは終了し、保存されたメッセージは破棄されます。セッションの有効期限が切れた後に `cleanSession` フラグを使用してクライアントが `0` に再接続すると、サービスは新しい永続的セッションを作成します。前のセッションのサブスクリプションまたはメッセージは、前のセッションの有効期限が切れたときに破棄されたため、このセッションでは使用できません。

永続セッションのメッセージ料金

メッセージブローカーがクライアントまたはオフラインの永続セッションにメッセージを送信 AWS アカウント すると、メッセージは に課金されます。永続セッションを持つオフラインデバイスが再接続してセッションを再開すると、保存されたメッセージがデバイスに配信され、アカウントに再び課金されます。メッセージの料金の詳細については、「AWS IoT Core の料金 - メッセージング」を参照してください。

標準の制限引き上げプロセスを使用すると、デフォルトの永続セッションの有効期間を 1 時間引き上げることができます。セッションの有効期限を延長すると、メッセージ料金が増加する可能性があることに注意してください。これは、時間を延長するとオフラインデバイスに保存されるメッセージが増える可能性があり、標準のメッセージング料金でこれらの追加のメッセージが課金され、アカウントに請求されるためです。セッションの有効期限は概算であり、セッションはアカウントの制限よりも最長で 30 分長く持続する可能性があります。ただし、セッションはアカウントの制限より短くなることはありません。セッションの制限の詳細については、「AWS Service Quotas」を参照してください。

保持された MQTT メッセージ

AWS IoT Core は、MQTT プロトコルで説明されている `RETAIN`フラグをサポートします。クライアントが発行する MQTT メッセージに `RETAIN`フラグを設定すると、 はメッセージ AWS IoT Core を保存します。その後、新しいサブスクライバーに送信し、`GetRetainedMessage` オペレーションを呼び出して取得し、AWS IoT コンソールで表示できます。

保持された MQTT メッセージの使用例

- 初期設定メッセージとしての使用

  保持されたMQTTメッセージは、クライアントがトピックにサブスクライブした後、クライアントに送信されます。トピックをサブスクライブするすべてのクライアントが、サブスクライブ後すぐに保持された MQTT メッセージを受信できるようにする場合は、設定された RETAIN フラッグで設定されたメッセージを発行することができます。サブスクライブしているクライアントはまた、新しい設定メッセージが発行されるたびに、その設定に対する更新が受信できます。

- 最新のメッセージとして

  デバイスは、 AWS IoT Core が現在の状態メッセージを保存するように、それらに RETAIN フラッグを設定することができます。アプリケーションは、接続または再接続するときに、このトピックをサブスクライブし、保持されているメッセージのトピックをサブスクライブした直後に、最後に報告された状態を取得できます。こうすることで、現在の状態を確認するために、デバイスからの次のメッセージを待つ必要がなくなります。

このセクションの内容:

- での MQTT 保持メッセージの一般的なタスク AWS IoT Core
- 請求と保持メッセージ
- 保持されたMQTTメッセージと永続MQTTセッションの比較
- MQTT 保持メッセージと AWS IoT デバイスシャドウ

での MQTT 保持メッセージの一般的なタスク AWS IoT Core

AWS IoT Core は、 RETAINフラグが設定された MQTT メッセージを保存します。これらの保持されたメッセージは、通常の MQTT メッセージとしてトピックにサブスクライブしたすべてのクライアントに送信されると同時に、トピックへの新しいサブスクライバーに送信するために保存されます。

保持されたMQTTメッセージは、クライアントがメッセージにアクセスすることを許可するために特定のポリシーアクションが必要です。保持されるメッセージポリシーの使用例については、保持されたメッセージポリシーの例を参照してください。

このセクションでは、保持されたメッセージに関連する一般的な操作について説明します。

- 保持されたメッセージの作成

クライアントは、MQTT メッセージを発行するときにメッセージを保持するかどうかを決定します。クライアントは、デバイス SDK を使用してメッセージを発行するときに、RETAIN フラッグを設定できます。アプリケーションおよびサービスは、Publish アクションを使用して MQTT メッセージを発行するときに、RETAIN フラッグを設定することができます。

メッセージは、トピック名ごとに 1 つのみ保持されます。トピックに対して発行されたRETAIN フラッグが付いた新しいメッセージは、以前にトピックに送信されたあらゆる既存の保持メッセージを置き換えます。

> **Note**
>
> RETAIN フラッグが設定された状態で予約済みトピックに対して発行することはできません。

- 保持されたメッセージのトピックのサブスクライブ

  クライアントは、他の MQTT メッセージトピックと同様に、保持されるメッセージトピックをサブスクライブします。保持メッセージのトピックをサブスクライブすることによって受信される保持メッセージには、RETAIN フラグが設定されています。

  保持されたメッセージは、クライアントが 0 バイトのメッセージペイロードを持つ保持されたメッセージを保持されたメッセージトピックに発行 AWS IoT Core すると、 から削除されます。保持されているメッセージのトピックをサブスクライブしたクライアントも、0 バイトのメッセージを受信する事になります。

  保持されたメッセージのトピックが含まれるワイルドカードトピックフィルターにサブスクライブすると、クライアントは保持されたメッセージのトピックに発行された後続のメッセージを受信できるようになりますが、トピックはサブスクライブ時に保持されたメッセージを配信しません。

  > **Note**
  >
  > サブスクライブ時に保持されたメッセージを受信するには、サブスクリプションリクエストのトピックフィルターが、保持されたメッセージのトピックと完全に一致する必要があります。

保持されたメッセージのトピックへのサブスクライブ時に受信する保持されたメッセージには、RETAIN フラグが設定されています。サブスクライブしているクライアントが、サブスクライブ後に受信するメッセージに、このフラグは設定されません。

- 保持されたメッセージの取得

  保持されたメッセージが、保持されたメッセージが含まれたトピックにサブスクライブするときに、クライアントに自動配信されます。クライアントがサブスクライブ時に保持されたメッセージを受信するには、保持されたメッセージの正確なトピック名にサブスクライブする必要があります。保持されたメッセージのトピックが含まれるワイルドカードトピックフィルターにサブスクライブすると、クライアントは保持されたメッセージのトピックに発行された後続のメッセージを受信できるようになりますが、トピックはサブスクライブ時に保持されたメッセージを配信しません。

  サービスとアプリは、ListRetainedMessagesおよびGetRetainedMessageを呼び出す事によって、保持されているメッセージを一覧表示および取得する事ができます。

  クライアントは、RETAIN フラグを設定しなくても、保持されているメッセージのトピックにメッセージを発行できます。これは、保持されたメッセージが、トピックをサブスクライブすることで受信したメッセージと一致しないなど、予期しない結果を発生させる可能性があります。

  MQTT 5 では、保持メッセージにメッセージ有効期限が設定されていて保持メッセージの有効期限が切れると、そのトピックをサブスクライブする新規サブスクライバーは、サブスクリプションが成功しても保持メッセージを受信しません。

- 保持されたメッセージのトピックの一覧表示

  保持されたメッセージは、ListRetainedMessagesを呼び出す事で、リスト化する事ができ、保持されたメッセージはAWS IoT consoleに表示する事ができます。

- 保持されたメッセージの詳細情報の取得

  GetRetainedMessageを呼び出す事で、保持されたメッセージの詳細を取得する事ができ、それらは、AWS IoT consoleに表示する事ができます。

- Will メッセージの保持

  デバイス接続時に作成されるMQTTウィルメッセージはConnect Flag bitsフィールドにWill Retainフラッグを設定する事で保持する事ができます。

- 保持されたメッセージの削除

デバイス、アプリケーション、およびサービスは、RETAIN フラグが設定されたメッセージと、削除するメッセージのトピック名に空の (0 バイト) メッセージペイロードを発行することで、保持されたメッセージを削除することができます。このようなメッセージは、保持されたメッセージを削除し AWS IoT Core、トピックへのサブスクリプションを持つクライアントに送信されますが、によって保持されません AWS IoT Core。

保持されたメッセージは、AWS IoT コンソールで保持されたメッセージにアクセスすることによって、インタラクティブに削除することもできます。AWS IoT コンソールを使用して削除される保持されたメッセージも、保持されたメッセージのトピックにサブスクライブしているクライアントに対して 0 バイトメッセージを送信します。

保持されたメッセージの削除後に、それらを復元することはできません。クライアントは、削除されたメッセージの代わりに新しい保持されたメッセージを発行する必要があります。

- 保持されたメッセージのデバッグとトラブルシューティング

  AWS IoT consoleは、保持されたメッセージのトラブルシューティングに役立つツールをいくつか提供します。

  - 保持されたメッセージのページ

    AWS IoT コンソールの保持されたメッセージのページは、現在の地域で、アカウントによって保存された保持されたメッセージのページ分けされたリストを提供します。このページからは、以下を実行できます。

    - メッセージペイロード、QoS、受信時間など、保持されたメッセージそれぞれの詳細を確認してください。
    - 保持されたメッセージの内容を更新する。
    - 保持されたメッセージを削除する。

  - MQTT テストクライアント

    AWS IoT コンソールのMQTT テストクライアントページは、MQTT トピックにサブスクライブおよび発行する事ができます。公開オプションは、どのようにデバイスが動作するかをシミュレートするために発行するメッセージにRETAIN フラグを設定してくれます。MQTT テストクライアントを使用して、クライアント接続インターフェイスを介して管理する接続されたクライアントからのメッセージをモニタリングすることもできます。クライアント接続の管理の詳細については、「the section called “MQTT 接続の管理”」を参照してください。

一部の予期しない結果は、保持されたメッセージの実装方法に関するこれらの側面の結果である可能性があります AWS IoT Core。

- 保持されるメッセージの制限

  アカウントが保持されたメッセージの最大数を保存すると、 は、保持されたメッセージの一部が削除され、保持されたメッセージ数が制限を下回るまで、RETAIN セットと 0 バイトを超えるペイロードで発行されたメッセージに対してスロットリングされたレスポンス AWS IoT Core を返します。
- 保持されたメッセージの配信順序

  保持されたメッセージとサブスクライブされたメッセージの配信順序は保証されていません。

請求と保持メッセージ

AWS IoT コンソールを使用して、または Publish を呼び出して、クライアントからの RETAIN フラグが設定されたメッセージを発行すると、「AWS IoT Core 料金 - メッセージング」に記載されている追加のメッセージング料金が発生します。

クライアント、 AWS IoT コンソール、または を呼び出して保持されたメッセージを取得すると、通常の API GetRetainedMessage 使用料に加えてメッセージング料金が発生します。追加料金については、AWS IoT Core 料金表-メッセージングに説明されています。

予期せずデバイス接続が切断された際、発行されるMQTTウィルメッセージは、AWS IoT Core 料金表-メッセージングで説明されているメッセージング料金が発生します。

メッセージコストの詳細については、AWS IoT Core の料金 – メッセージングを参照してください。

保持されたMQTTメッセージと永続MQTTセッションの比較

保持メッセージと永続セッションは、MQTT の標準機能であり、オフライン中に発行されたメッセージをデバイスが受信できるようにします。保持されたメッセージは、永続的なセッションから発行する事ができます。このセクションでは、これらの機能の主な側面と、これらがどのように連携するかについて説明します。

	保持されたメッセージ	永続セッション
主な特徴	保持されたメッセージは、接続後に、デバイスを設定または大規模なグループにデバイスを通知するために使用することができます。	永続セッションは、接続が断続的で、いくつかの重要なメッセージを受信しない可能性があるデバイスに役立ちます。

	保持されたメッセージ	永続セッション
	保持されたメッセージは、デバイスが再接続した後で、トピックに発行された最後のメッセージのみを受信するようにしたい場合にも使用できます。	デバイスは、永続セッションで接続して、オフライン中に送信されたメッセージを受信できます。
例	保持メッセージを使用すると、デバイスがオンラインになったときに、デバイスの環境に関する設定情報が提供できます。初期設定には、サブスクライブする他のメッセージのトピックのリストや、ローカルタイムゾーン設定方法についての情報を含める事ができます。	接続が断続的なセルラーネットワーク経由で接続するデバイスは、デバイスがネットワーク圏外またはセルラー無線をオフにする必要があるときに送信される重要なメッセージを受信し損ねる事がないように、永続セッションを使う事ができます。
トピックへの初回サブスクライブ時に受信したメッセージ	保持されたメッセージを持つトピックをサブスクライブすると、最新の保持されたメッセージが受信されます。	保持されたメッセージがないトピックをサブスクライブすると、そのトピックに発行されるまでメッセージを受信しません。
再接続後にサブスクライブされたトピック	永続セッションを使用しない場合、クライアントは再接続後にトピックをサブスクライブする必要があります。	サブスクライブされたトピックは、再接続後に復元されます。
再接続後に受信したメッセージ	保持されたメッセージを持つトピックをサブスクライブすると、最新の保持されたメッセージが受信されます。	デバイスが切断されている間に QOS = 1 で発行され、QOS = 1 でサブスクライブされたすべてのメッセージは、デバイスの再接続後に送信されます。

	保持されたメッセージ	永続セッション
データ/セッションの有効期限	MQTT 3 では、保持されたメッセージに有効期限はありません。これらは、置き換えられる、または削除されるまで保存されます。MQTT 5 では、保持されたメッセージは、設定したメッセージ有効期限を過ぎると期限切れになります。詳細については、「メッセージの有効期限」を参照してください。	永続セッションは、クライアントがタイムアウト期間内に再接続しない場合、期限切れになります。永続セッションの有効期限が切れると、QOS = 1 で発行され、デバイスが切断されている間に QOS = 1 でサブスクライブされたクライアントのサブスクリプションと保存済みメッセージが削除されます。期限切れのメッセージは配信されません。永続的なセッションでのセッションの有効期限の詳細については、「the section called “MQTT 永続的セッション”」を参照してください。。

永続セッションについては、the section called “MQTT 永続的セッション” を参照してください。

Retreated Messages を使用すると、発行するクライアントは、接続後にメッセージを保持してデバイスに配信するかどうか、デバイスに以前のセッションがあったかどうかを判断します。メッセージを保存する選択はパブリッシャーが行い、保存されたメッセージは、QoS 0 または QoS 1 のサブスクリプションをサブスクライブする現在および将来のすべてのクライアントに配信されます。保持されたメッセージで一度に維持できるのは、特定のトピックに関するメッセージ 1 つだけです。

アカウントが保持されるメッセージを最大数保存している場合、 AWS IoT Core は、保持されたメッセージの一部が削除され、保持されたメッセージの数が上限を下回るまで、RETAIN が設定され、ペイロードが 0 バイトを超える状態で発行されたメッセージに対してスロットルされたレスポンスを返します。

MQTT 保持メッセージと AWS IoT デバイスシャドウ

保持されたメッセージとデバイスシャドウのどちらでもデバイスからのデータが保持されますが、どちらも動作が異なり、達成する目的も異なります。このセクションでは、それらの類似点と相違点について説明します。

	保持されたメッセージ	Device Shadow
メッセージペイロードに事前定義された構造またはスキーマがある	実装によって定義されている通り。MQTT は、そのメッセージペイロードの構造やスキーマを指定しません。	AWS IoT は、特定のデータ構造をサポートしています。
メッセージペイロードを更新すると、イベントメッセージが生成されます	保持されたメッセージを発行すると、サブスクライブしているクライアントにメッセージが送信されますが、追加の更新メッセージは生成されません。	Device Shadow を更新すると、変更を説明するメッセージを更新します。
メッセージの更新に番号が付けられる	保持されたメッセージには、自動的に番号が付けられません。	デバイスシャドウドキュメントには、自動のバージョン番号とタイムスタンプがあります。
メッセージペイロードは、モノのリソースにアタッチされます	保持されたメッセージは モノの リソースに添付されません。	デバイスシャドウは モノの リソースにアタッチされます。
メッセージペイロードの個々の要素の更新	メッセージの個々の要素は、メッセージペイロード全体を更新しないと変更できません。	デバイスシャドウドキュメントの個々の要素は、デバイスシャドウドキュメント全体を更新しなくても更新できます。
クライアントは、サブスクリプションすると直ちにメッセージデータを受信します	クライアントは、保持メッセージを持つトピックをサブスクライブすると、保持メッセージを自動的に受信します。	クライアントはデバイスシャドウ更新にサブスクライブできますが、現在のステータスを意図的にリクエストする必要があります。
インデックス作成と検索可能性	保持されたメッセージは、検索用にインデックス化されません。	フリートインデックス作成は、検索および集計のために

	保持されたメッセージ	Device Shadow
		Device Shadowデータをインデックス化します

MQTT の Last Will and Testament (LWT) メッセージ

Last Will and Testament (LWT) は MQTT の機能です。LWT を使用すると、クライアントはブローカーがクライアント定義のトピックに発行し、開始されていない切断が発生したときにそのトピックをサブスクライブしているすべてのクライアントに送信するメッセージを指定できます。クライアントが指定するメッセージは LWT メッセージまたは Will メッセージと呼ばれ、クライアントが定義するトピックは Will トピックと呼ばれます。デバイスがブローカーに接続するときに LWT メッセージを指定できます。これらのメッセージは、接続中に `Connect Flag bits` フィールドで `Will Retain` フラグを設定することで保持できます。例えば、`Will Retain` フラグが `1` に設定されている場合、Will メッセージはブローカーの関連する Will トピックに保存されます。詳細については、「Will メッセージ」を参照してください。

クライアント接続を管理する場合、クライアントを切断するときに LWT メッセージを発行するかどうかを制御できます。詳細については、「the section called “MQTT 接続の管理”」を参照してください。

ブローカーは、開始されていない切断が発生するまで Will メッセージを保存します。その場合、ブローカーは Will トピックにサブスクライブしているすべてのクライアントにメッセージを発行して切断を通知します。MQTT DISCONNECT メッセージを使用してクライアントが開始した切断により、クライアントがブローカーから切断した場合、ブローカーは保存されている LWT メッセージを発行しません。それ以外の場合は、すべて LWT メッセージが送信されます。切断処理は非同期であるため、再接続時に LWT メッセージが順番にディスパッチされるとは限りません。接続状態の検出精度を向上させるために、ライフサイクルイベントを使用することをお勧めします。これらのイベントでは、タイムスタンプやバージョン番号などの属性が提供されるため、順不同のイベントを管理することができます。ブローカーが LWT メッセージを送信するときの切断シナリオの完全なリストについては、「接続/切断イベント」を参照してください。

connectAttributes の使用

`ConnectAttributes` を使用すると、`PersistentConnect` や `LastWill` などの IAM ポリシーの接続メッセージで使用する属性を指定できます。`ConnectAttributes` を使用すると、デフォルトではデバイスに新機能へのアクセスを許可しないポリシーを構築できます。これは、デバイスが侵害された場合に役立ちます。

connectAttributes でサポートされる機能は以下のとおりです。

PersistentConnect

PersistentConnect 機能を使用して、クライアントとブローカー間の接続が中断されたときに、接続中にクライアントが作成したすべてのサブスクリプションを保存します。

LastWill

LastWill 機能を使用して、クライアントが予期せず切断したときにメッセージを LastWillTopic に発行します。

デフォルトでは、ポリシーには非永続的な接続があり、この接続用に渡される属性はありません。永続的な接続が必要な場合は、IAM ポリシーで永続的な接続を指定する必要があります。

クライアント接続を管理するときに、接続されたクライアントの接続属性とセッションの設定を表示することができます。詳細については、「the section called “MQTT 接続の管理”」を参照してください。

ConnectAttributes 例については、接続ポリシーの例を参照してください。

MQTT 5 がサポートしている機能

AWS IoT Core MQTT 5 のサポートは MQTT v5.0 仕様に基づいていますが、「」で説明されているようにいくつかの違いがありますthe section called “AWS IoT MQTT 仕様との違い”。

AWS IoT Core は、次の MQTT 5 機能をサポートしています。

- 共有サブスクリプション
- クリーンスタートとセッションの有効期限
- すべての ACK の理由コード
- トピックエイリアス
- メッセージの有効期限
- その他の MQTT 5 の機能

共有サブスクリプション

AWS IoT Core は、MQTT 3.1.1 と MQTT 5 の両方の共有サブスクリプションをサポートしています。共有サブスクリプションを使用すると、1 つのトピックへのサブスクリプションを複数のクライ

アントで共有できますが、そのトピックに公開されたメッセージをランダム配信を使って受信できるのは 1 つのクライアントのみです。共有サブスクリプションでは、多数のサブスクライバー間で MQTT メッセージを効果的に負荷分散できます。例えば、同じトピックを発行するデバイスが 1,000 台あり、それらのメッセージを処理するバックエンドアプリケーションが 10 台あるとします。その場合、バックエンドアプリケーションは同じトピックをサブスクライブでき、それぞれが共有トピックにデバイスによって発行されたメッセージをランダムに受信します。これにより、それらのメッセージの負荷を効果的に「共有」できます。共有サブスクリプションでは、耐障害性も向上します。バックエンドアプリケーションの接続が切断されると、ブローカーはグループ内の残りのサブスクライバーに負荷を分散します。すべてのサブスクライバーが切断されると、メッセージはキューに入れられます。

メッセージキュー機能は、メッセージ配信の信頼性を高めるために、MQTT 3.1.1 と MQTT 5 のどちらの接続でも共有サブスクリプションで使用できます。

共有サブスクリプションを使用するには、クライアントは次のように共有サブスクリプションのトピックフィルターをサブスクライブします。

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

- $share は共有サブスクリプションのトピックフィルターを示すリテラル文字列です。トピックフィルターは $share で始まる必要があります。
- {ShareName} はサブスクライバーのグループが使用する共有名を指定する文字列です。共有サブスクリプションのトピックフィルターは、ShareName を含み、その後に / の文字が続く必要があります。{ShareName} には、/、+、または # などの文字を含めないでください。{ShareName} の最大長は UTF-8 の 128 文字です。
- {TopicFilter} は、非共有サブスクリプションとして、同じトピックフィルターの構文に従います。{TopicFilter} の最大長は UTF-8 の 256 文字です。
- $share/{ShareName}/{TopicFilter} に必要な 2 つのスラッシュ (/) は、トピックおよびトピックフィルターのスラッシュの最大数の制限に含まれていません。

同じ {ShareName}/{TopicFilter} を持つサブスクリプションは、同じ共有サブスクリプショングループに属します。共有サブスクリプショングループは複数作成できますが、グループあたりの共有サブスクリプションの制限を超えないようにしてください。詳細については、AWS 全般のリファレンスの「AWS IoT Core エンドポイントとクォータ」を参照してください。

次の表では、非共有サブスクリプションと共有サブスクリプションを比較しています。

サブスクリプション	説明	トピックフィルターの例
非共有サブスクリプション	各クライアントは、発行されたメッセージを受信するための個別のサブスクリプションを作成します。メッセージがトピックに発行されると、そのトピックのすべてのサブスクライバーがメッセージのコピーを受け取ります。	`sports/tennis` `sports/#`
共有サブスクリプション	複数のクライアントが 1 つのトピックへのサブスクリプションを共有できますが、そのトピックに公開されたメッセージをランダム配信で受信できるのは 1 つのクライアントのみです。	`$share/consumer/sports/tennis` `$share/consumer/sports/#`

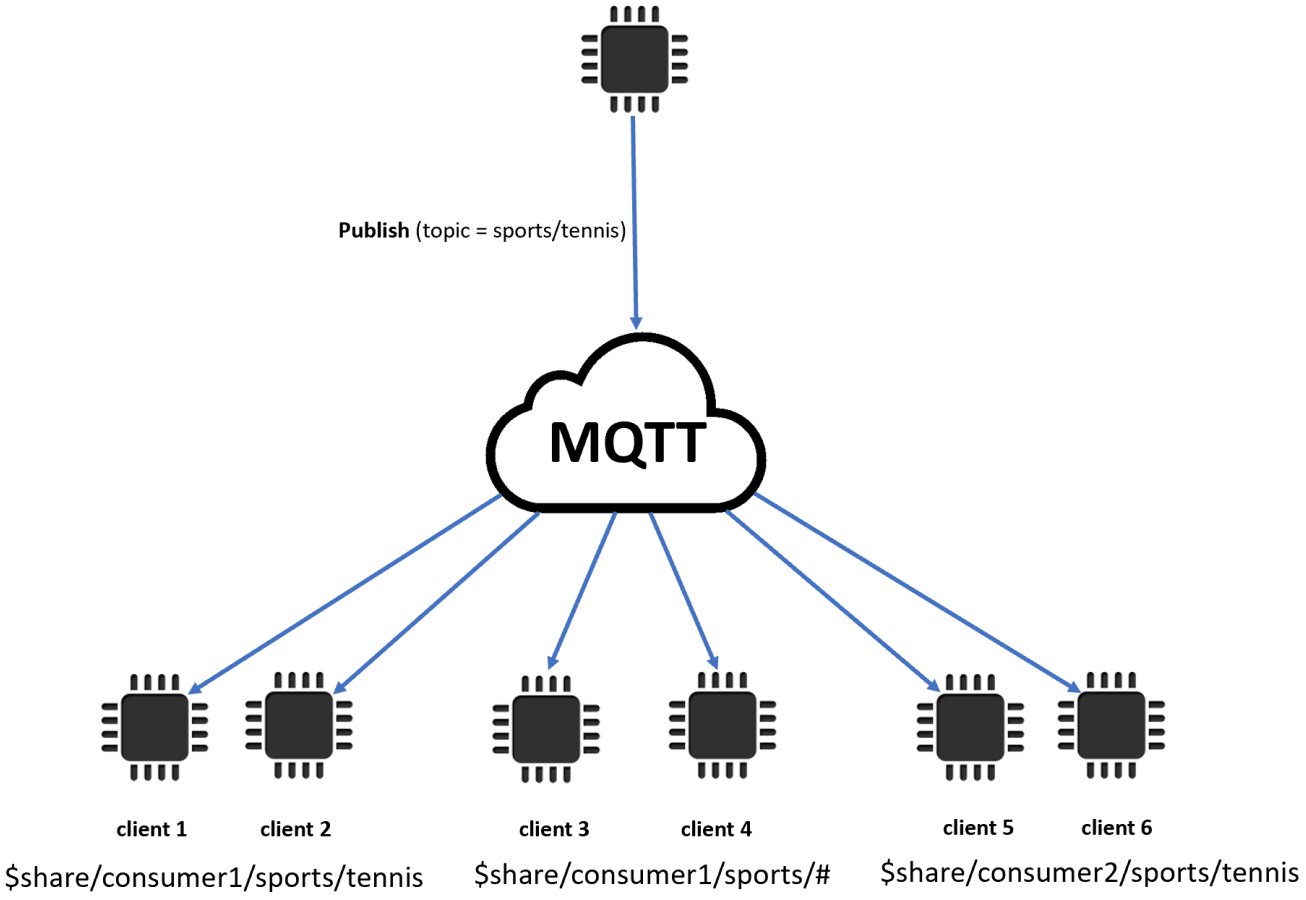
非共有サブスクリプションのフロー	共有サブスクリプションのフロー		
			

共有サブスクリプションを使用する際の重要な注意事項

- 共有サブスクライバーグループが永続セッションサブスクライバーで構成されている場合、共有グループ内のすべてのサブスクライバーが切断されると、またはサブスクライバーが接続毎の 1 秒あたりの発行リクエスト数の制限に違反すると、共有サブスクリプショングループに発行された未確認の QoS 1 メッセージと未配信の QoS 1 メッセージがキューに入れられます。詳細については、「共有サブスクリプションのメッセージキュー機能」を参照してください。
- 共有サブスクリプショングループに発行された QoS 0 メッセージは、障害が発生すると削除されます。
- 共有サブスクリプションは、共有サブスクライバーグループの一部としてトピックパターンにサブスクライブしている場合、保持されたメッセージを受信しません。共有サブスクライバーを持つト

ピックで発行され、RETAIN フラグが設定されているメッセージは、他の発行メッセージと同様に共有サブスクライバーに配信されます。

- 共有サブスクリプションにワイルドカード文字 (# または +) が含まれている場合、1 つのトピックに一致する共有サブスクリプションが複数存在する可能性があります。その場合、メッセージブローカーは発行メッセージをコピーし、一致する共有サブスクリプションごとにランダムなクライアントに送信します。共有サブスクリプションのワイルドカードの動作を、次の図で示しています。



この例では、発行中の MQTT トピック sports/tennis と一致する共有サブスクリプションが 3 つあります。メッセージブローカーは発行されたメッセージをコピーし、一致する各グループのランダムなクライアントにメッセージを送信します。

クライアント 1 とクライアント 2 はサブスクリプション $share/consumer1/sports/tennis を共有します。

クライアント 3 とクライアント 4 はサブスクリプション `$share/consumer1/sports/#` を共有します。

クライアント 5 とクライアント 6 はサブスクリプション `$share/consumer2/sports/tennis` を共有します。

共有サブスクリプションの制限について詳しくは、「AWS 全般のリファレンス」の「AWS IoT Core エンドポイントとクォータ」を参照してください。AWS IoT コンソールで AWS IoT MQTT クライアントを使用して共有サブスクリプションをテストするには、「」を参照してください???。クライアント接続管理機能を使用して、共有サブスクリプションを含む、接続されたクライアントがサブスクライブしているトピックを表示することもできます。詳細については、「the section called “MQTT 接続の管理”」を参照してください。共有サブスクリプションの詳細については、MQTTv5.0 仕様の「共有サブスクリプション」を参照してください。

共有サブスクリプションのメッセージキュー機能

メッセージ配信の信頼性を高めるために、共有サブスクリプションには、オンラインサブスクライバーが不在のときにメッセージを保存するメッセージキュー機能が含まれています。共有サブスクリプショングループに永続セッションを持つメンバーが少なくとも 1 つ含まれている場合に、キュー機能はグループに対して有効になります。メッセージを配信する場合、オンラインのメンバーが受信者として選択されます。QoS 1 メッセージは、オンラインでメンバーが見つからなかった場合、またはサブスクライバーが `Publish requests per second per connection` 制限を超えた場合にキューに入れられます。キューに入れられたメッセージは、既存のメンバーが永続セッションを再開するか、新しいメンバーがグループに参加すると配信されます。キューに入れられたメッセージは、サブスクリプションに従ってサブスクライバーに配信されるその他のメッセージとともに、アクティブなグループサブスクライバーごとに 1 秒あたり最大 20 件 (キューに入れられたメッセージ) の速度で配信されます。

デフォルトでは、キューに入れられたメッセージ保持は `Persistent Session expiry period` クォータに従います。ただし、インバウンド発行メッセージでメッセージの有効期限間隔 (MEI) が設定されている場合、MEI が優先されます。MEI が存在する場合は、永続セッションの有効期限に関係なく、MEI によってメッセージの保持期間が決定されます。

メッセージキューのレートは `Queued messages per second per account` クォータに応じて制限され、メッセージ数は `Maximum number of queued messages per shared subscription group` クォータによって制限されます。クォータを表示および管理するには、Service Quotas コンソールにアクセスします。

AWS/Usage 名前空間で を検索してキュー CloudWatch ApproximateQueueDepthをモニタリングするか、次の CLI コマンドを使用して、各共有サブスクリプショングループのキュー深度に関連付けられたメトリクスを一覧表示できます。

```
aws cloudwatch list-metrics --namespace "AWS/Usage" --dimensions
 Name=Resource,Value='ApproximateQueueDepth'
```

これらの制限を超えると、制限に達する前にキューに入れられたメッセージのみが保持されます。制限を超える新しい受信メッセージは削除されます。システムは、キューに入れられ古いたメッセージを新しいメッセージに置き換えません。

メッセージのキューは、CloudWatch メトリクスと CloudWatch Logs に記録されます。CloudWatch と CloudWatch Logs に書き込まれるエントリについては、「メッセージブローカーのメトリクス」および「キューに保存されたログエントリ」を参照してください。キューに登録されたメッセージは、標準のメッセージング料金で請求されます。メッセージの料金の詳細については、「AWS IoT Core の料金」を参照してください。

共有サブスクリプショングループのセッションライフサイクル

クリーンセッションがグループにサブスクライブすると、そのセッションはグループのオンラインメンバーになります。サブスクリプションを解除または切断すると、クリーンセッションはグループから離脱します。

永続セッションがグループにサブスクライブすると、そのセッションはグループのオンラインメンバーになります。切断しても、グループには残りますが、グループのオフラインメンバーになります。再接続すると、再びオンラインメンバーになります。永続セッションは、明示的にサブスクリプションを解除したとき、または切断後に期限切れになったときにグループから離脱します。

クリーンスタートとセッションの有効期限

クリーンスタートとセッション有効期限を使用すると、永続セッションをより柔軟に処理できます。クリーンスタートフラグは、既存のセッションを使用せずにセッションを開始する必要があるかどうかを示します。セッションの有効期限間隔は、切断後にセッションを保持する期間を示します。セッションの有効期限間隔は、切断時に変更できます。詳細については、「the section called “MQTT 永続的セッション”」を参照してください。

すべての ACK の理由コード

理由コードを使用すると、エラーメッセージをより簡単にデバッグまたは処理できます。理由コードは、ブローカーとのやり取りのタイプ (サブスクライブ、発行、確認) に基づいてメッセージブロー

カーから返されます。詳細については、「MQTT 理由コード」を参照してください。MQTT 理由コードの完全なリストについては、「MQTT v5 の仕様」を参照してください。

トピックエイリアス

トピック名は、2 バイトの整数であるトピックエイリアスに置き換えることができます。トピックエイリアスを使用すると、トピック名の送信を最適化して、計測データサービスのデータコストを削減できます。 には、トピックエイリアスのデフォルト制限が 8 つ AWS IoT Core あります。詳細については、AWS 全般のリファレンスの「AWS IoT Core エンドポイントとクォータ」を参照してください。

メッセージの有効期限

発行されたメッセージには、メッセージの有効期限値を追加できます。これらの値は、メッセージの有効期限間隔 (MEI) を秒単位で表します。その間隔内にメッセージがサブスクライバーに送信されない場合、メッセージは期限切れになり、削除されます。メッセージの有効期限値を設定しない場合、メッセージは期限切れになりません。

アウトバウンドでは、サブスクライバーは有効期限の残り時間を含むメッセージを受信します。例えば、受信した発行メッセージの有効期限が 30 秒で、20 秒後にサブスクライバーにルーティングされた場合、メッセージの有効期限フィールドは 10 に更新されます。サブスクライバーが受信したメッセージの MEI が 0 に更新されている可能性があります。これは、残り時間が 999 ms 以下になるとすぐに 0 に更新されるためです。

では AWS IoT Core、最小 MEI は 1 です。クライアント側から間隔を 0 に設定すると、1 に調整されます。メッセージの最大有効期間は 604800 (7 日) です。これより大きい値はすべて最大値に調整されます。

クロスバージョン通信では、メッセージの有効期限切れの動作は、インバウンド発行メッセージの MQTT バージョンによって決定されます。例えば、MQTT5 経由で接続されたセッションから送信されたメッセージの有効期限付きのメッセージは、MQTT3 セッションでサブスクライブされているデバイスでは期限切れになる可能性があります。次の表は、メッセージ有効期限が次のタイプの発行メッセージをどのようにサポートするかを示しています。

メッセージの種類を発行する	メッセージの有効期限間隔
通常発行	サーバーが指定された時間内にメッセージの配信に失敗すると、期限切れのメッセージは削除され、サブスクライバーはそのメッセージを受信できなくな

メッセージの種類を発行する	メッセージの有効期限間隔
	ります。これには、デバイスが QoS 1 メッセージを発行していない場合などが含まれます。
Retain	保持されたメッセージの有効期限が切れて新しいクライアントがそのトピックをサブスクライブした場合、クライアントはサブスクライブ時にメッセージを受信しません。
ラストウィル	ラストウィルメッセージの間隔は、クライアントが接続を切断し、サーバーがラストウィルメッセージをサブスクライバーに配信しようとした後に始まります。
キューに追加済みのメッセージ	クライアントがオフラインのときにメッセージ有効期限間隔が設定されたアウトバウンド QoS 1 の有効期限が切れた場合、永続セッションが再開されても、クライアントは期限切れメッセージを受信しません。

その他の MQTT 5 の機能

サーバー切断

接続が切断されると、サーバーは事前にクライアントに DISCONNECT を送信して、切断の理由コードを添えて接続の終了を通知できます。

リクエスト/レスポンス

発行者は、受信時に発行者が指定したトピックへの返信を受信者に送信するようリクエストできます。

最大パケットサイズ

クライアントとサーバーは、サポートする最大パケットサイズを個別に指定できます。

ペイロード形式とコンテンツタイプ

メッセージを発行するときのペイロード形式 (バイナリ、テキスト) とコンテンツタイプを指定できます。これらはメッセージの受信者に転送されます。

MQTT 5 プロパティ

MQTT 5 プロパティは、セッションの有効期限やリクエスト/レスポンスパターンなどの MQTT 5 の新機能をサポートするために、MQTT 標準に追加された重要な機能です。では AWS IoT Core、アウトバウンドメッセージのプロパティを転送するルールを作成するか、HTTP Publish を使用して新しいプロパティの一部を含む MQTT メッセージを発行できます。

次の表に、 が AWS IoT Core サポートするすべての MQTT 5 プロパティを示します。

プロパティ	説明	入力タイプ	パケット
ペイロード形式インジケータ	ペイロードが UTF-8 としてフォーマットされているかどうかを示すブール値。	バイト	PUBLISH、CONNECT
コンテンツタイプ	ペイロードの内容を説明する UTF-8 文字列。	UTF-8 文字列	PUBLISH、CONNECT
レスポンストピック	受信者がリクエスト/レスポンスフローの一部として公開すべきトピックを説明する UTF-8 文字列。トピックにはワイルドカード文字を含めないでください。	UTF-8 文字列	PUBLISH、CONNECT
相関データ	リクエストメッセージの送信者が、レスポンスメッセージの対象となるリクエストを識別するために使用するバイナリデータ。	バイナリ	PUBLISH、CONNECT
ユーザーのプロパティ	UTF-8 文字列。このプロパティは、1 つのパケットに複数回表示されることがあります。受信者は、送信されたのと同じ順序でキーと値のペアを受け取ります。	UTF-8 文字列ペア	接続、発行、ウィルのプロパティ、サブスクライブ、切断、サブスクライブ解除
メッセージの有効期限間隔	メッセージの有効期限間隔を秒単位で表す 4 バイトの整数。存在しない場合、メッセージの有効期限はありません。	4 バイト整数	PUBLISH、CONNECT

プロパティ	説明	入力タイプ	パケット
セッションの有効期限間隔	セッションの有効期限を秒単位で表す 4 バイトの整数。 は最大 7 日間 AWS IoT Core をサポートし、デフォルトの最大は 1 時間です。設定した値がアカウントの最大値を超える場合、 AWS IoT Core は CONNACK で調整された値を返します。	4 バイト整数	CONNECT、CONNACK、DISCONNECT
割り当てられたクライアント識別子	クライアント ID がデバイスによって指定されていない場合 AWS IoT Core に によって生成されるランダムなクライアント ID。ランダムクライアント ID は、現在ブローカーによって管理されている他のセッションでは使用されていない新しいクライアント ID でなければなりません。	UTF-8 文字列	CONNACK
サーバーキープアライブ	サーバーによって割り当てられたキープアライブ時間を表す 2 バイトの整数。クライアントがキープアライブ時間を超えて非アクティブになると、サーバーはクライアントを切断します。	2 バイト整数	CONNACK
問題情報をリクエストする	失敗した場合に理由文字列とユーザープロパティのどちらが送信されるかを示すブール値。	バイト	CONNECT
最大受信	PUBACK を受信せずに送信できる PUBLISH QOS > 0 パケットの最大数を表す 2 バイトの整数。	2 バイト整数	CONNECT、CONNACK
トピックエイリアスの最大数	この値は、トピックエイリアスとして受け入れられる最大値を示します。デフォルトは 0 です。	2 バイト整数	CONNECT、CONNACK

プロパティ	説明	入力タイプ	パケット
最大の QoS	が AWS IoT Core サポートする QoS の最大値。デフォルトは 1 です。 AWS IoT Core では、QoS 2 はサポートされていません。	バイト	CONNACK
保持可能	AWS IoT Core メッセージブローカーが保持されたメッセージをサポートしているかどうかを示すブール値。デフォルトは 1 です。	バイト	CONNACK
最大パケットサイズ	が AWS IoT Core 受け入れて送信する最大パケットサイズ。128 KB を超えることはできません。	4 バイト整数	CONNECT、CONNACK
ワイルドカードによるサブスクリプションが利用可能	AWS IoT Core メッセージブローカーがワイルドカードサブスクリプション利用可能をサポートしているかどうかを示すブール値。デフォルトは 1 です。	バイト	CONNACK
サブスクリプション ID が利用可能	AWS IoT Core メッセージブローカーが Subscription Identifier Available をサポートしているかどうかを示すブール値。デフォルトは 0 です。	バイト	CONNACK

MQTT 理由コード

MQTT 5 では、理由コードレスポンスによるエラーレポートが改善されました。 は、パケット別にグループ化された以下を含む理由コードを返す AWS IoT Core ことができます。MQTT 5 でサポートされている理由コードの完全なリストについては、「MQTT 5 specifications」(MQTT 5 の仕様) を参照してください。

CONNACK 理由コード

値	16 進数	理由コード名	説明
0	0x00	Success	接続を受け入れます。
128	0x80	未指定のエラー	サーバーは障害の理由を明らかにしたくないか、他の理由コードのいずれにも当てはまりません。
133	0x85	クライアント ID が無効です	クライアント ID は有効な文字列ですが、サーバーでは許可されていません。
134	0x86	ユーザー名またはパスワードが間違っています	サーバーは、クライアントによって指定されたユーザー名またはパスワードを受け入れません。
135	0x87	権限がありません	クライアントは接続する権限がありません。
144	0x90	トピック名が無効です	ウィルトピック名は正しい形式になっていますが、サーバーでは受け入れられません。
151	0x97	リソースクォータ	実装または管理で指定されている制限を超過しました。
155	0x9B	QoS はサポートされていません	サーバーは Will QoS で設定された QoS をサポートしていません。

PUBACK 理由コード

値	16 進数	理由コード名	説明
0	0x00	Success	メッセージは受け入れられます。QoS 1 メッセージの発行が続行されます。
128	0x80	未指定のエラー	受信者は発行を承認しませんが、理由を明らかにしたくないか、他の値のいずれとも一致しません。

値	16 進数	理由コード名	説明
135	0x87	権限がありません	PUBLISH は許可されていません。
144	0x90	トピック名が無効です	トピック名の形式は正しくありませんが、クライアントまたはサーバーでは受け入れられません。
145	0x91	使用中のパケット識別子	パケット ID は既に使用されています。これは、クライアントとサーバー間のセッション状態が一致していないことを示している可能性があります。
151	0x97	リソースクォータ	実装または管理で指定されている制限を超過しました。

DISCONNECT 理由コード

値	16 進数	理由コード名	説明
129	0x81	正しい形式でないパケット	受信したパケットはこの仕様に準拠していません。
130	0x82	プロトコルエラー	予期しないパケットまたは順不同のパケットが受信されました。
135	0x87	権限がありません	リクエストは承認されていません。
139	0x8B	サーバーのシャットダウン	サーバーはシャットダウン中です。
141	0x8D	キープアライブタイムアウト	キープアライブ時間の 1.5 倍の間、パケットが受信されなかったため、接続は閉じられます。

値	16 進数	理由コード名	説明
142	0x8E	セッションの引き継ぎ	同じ ClientID を使用する別の接続が接続されたため、この接続は閉じられました。
143	0x8F	トピックフィルターが無効です	トピックフィルターは正しく構成されていますが、サーバーでは受け入れられません。
144	0x90	トピック名が無効です	トピック名は正しい形式ですが、このクライアントまたはサーバーでは受け入れられません。
147	0x93	受信上限を超えました	クライアントまたはサーバーが、PUBACK または PUBCOMP を送信していない発行の受信数が最大受信数を超えています。
148	0x94	トピックのエイリアスが無効です	クライアントまたはサーバーが、CONNECT または CONNACK パケットで送信した最大トピックエイリアスを超えるトピックエイリアスを含む PUBLISH パケットを受信しました。
151	0x97	リソースクォータ	実装または管理で指定されている制限を超過しました。
152	0x98	管理アクション	管理アクションにより接続が切断されました。
155	0x9B	QoS はサポートされていません	クライアントが CONNACK の最大 QoS で指定した QoS よりも大きい QoS を指定しました。
161	0xA1	サブスクリプション ID はサポートされていません	サーバーはサブスクリプション ID をサポートしていません。サブスクリプションは受け付けられません。

SUBACK 理由コード

値	16 進数	理由コード名	説明
0	0x00	付与された QoS 0	サブスクリプションが承認され、送信される最大 QoS は QoS 0 になります。これはリクエストされた QoS よりも低い可能性があります。
1	0x01	付与された QoS 1	サブスクリプションが承認され、送信される最大 QoS は QoS 1 になります。これはリクエストされた QoS よりも低い可能性があります。
128	0x80	未指定のエラー	サブスクリプションは受け付けられず、サーバーは理由を明らかにしたくないか、他の理由コードのいずれも適用されません。
135	0x87	権限がありません	クライアントには、このサブスクライブを行う権限がありません。
143	0x8F	トピックフィルターが無効です	トピックフィルターは正しく構成されていますが、このクライアントでは使用できません。
145	0x91	使用中のパケット識別子	指定されたパケット識別子は既に使用されています。
151	0x97	リソースクォータ	実装または管理で指定されている制限を超過しました。

UNSUBACK 理由コード

値	16 進数	理由コード名	説明
0	0x00	Success	サブスクリプションが削除されます。
128	0x80	未指定のエラー	サブスクリプション解除を完了できませんでした。サーバーは理由を明らかにしたくないか、他の理由コードのいずれも適用されません。

値	16 進数	理由コード名	説明
143	0x8F	トピックフィルターが無効です	トピックフィルターは正しく構成されていますが、このクライアントでは使用できません。
145	0x91	使用中のパケット識別子	指定されたパケット識別子は既に使用されています。

AWS IoT MQTT 仕様との違い

メッセージブローカーの実装は MQTT v3.1.1 仕様および MQTT v5.0 仕様に基づいていますが、次の点で仕様とは異なります。

- AWS IoT は MQTT 3 の次のパケットをサポートしていません: PUBREC、PUBREL、PUBCOMP。
- AWS IoT は MQTT 5 の次のパケットをサポートしていません: PUBREC、PUBREL、PUBCOMP、AUTH。
- AWS IoT は MQTT 5 サーバーリダイレクトをサポートしていません。
- AWS IoT は、MQTT サービス品質 (QoS) レベル 0 および 1 のみをサポートします。 AWS IoT は、QoS レベル 2 での発行またはサブスクライブをサポートしていません。QoS 2 レベル 2 がリクエストされると、メッセージブローカーは PUBACK または SUBACK を送信しません。
- では AWS IoT、QoS レベル 0 のトピックにサブスクライブすると、メッセージが 0 回以上配信されます。メッセージは複数回配信される場合があります。複数回配信されるメッセージは、異なるパケット ID を使用して送信される場合があります。これらの場合、DUP フラグは設定されません。
- 接続リクエストに応答するとき、メッセージブローカーは CONNACK メッセージを送信します。このメッセージには、接続で前のセッションを再開するかどうかを示すフラグが含まれます。
- 追加の制御パケットまたは切断リクエストを送信する前に、クライアントは、 AWS IoT メッセージブローカーから CONNACK メッセージがデバイスで受信されるのを待機する必要があります。
- クライアントがトピックをサブスクライブすると、メッセージブローカーは SUBACK を送信してから、クライアントが新しい一致するメッセージの受信を開始するまでに、遅延が生じる場合があります。
- クライアントが、トピックをサブスクライブするために、#トピックフィルターでワイルドカード文字を使用する場合、トピック階層において、そのレベルとそれ以下の文字列はすべて一致しま

す。ただし、親トピックは照合されません。例えば、トピックへのサブスクリプションsensor/#は、トピックsensor/sensor/temperaturesensor/temperature/room1に発行されたメッセージを受信しますが、sensorに発行されたメッセージは受信しません。 ワイルドカードの使用の詳細については、「トピック名フィルター」を参照してください。

- メッセージブローカーは、クライアント ID を使用して、各クライアントを識別します。クライアント ID は MQTT ペイロードの一部としてクライアントからメッセージブローカーに渡されます。クライアント ID が同じ 2 つのクライアントがメッセージブローカーに同時に接続することはできません。あるクライアントが別のクライアントのクライアント ID を使用してメッセージブローカーに接続すると、新しいクライアント接続が受け入れられ、以前に接続されたクライアントは切断されます。API を使用してクライアントを手動で切断することもできます。詳細については、「the section called “MQTT 接続の管理”」を参照してください。
- まれに、メッセージブローカーは、パケット ID が異なる同じ論理 PUBLISH メッセージを再送信する場合があります。
- ワイルドカード文字を含むトピックフィルターへのサブスクリプションでは、保持されたメッセージの受信ができません。保持されたメッセージを受信するには、サブスクライブリクエストに、保持されたメッセージのトピックと完全に一致するトピックフィルターが含まれている必要があります。
- メッセージブローカーはメッセージと ACK の正しい受信順序を確保するわけではありません。
- AWS IoT は、仕様とは異なる制限を持つことができます。詳細については、AWS IoT リファレンスガイドの「AWS IoT Core メッセージブローカーとプロトコルの制限とクォータ」を参照してください。
- MQTT DUP フラグはサポートされていません。

MQTT 接続の管理

AWS IoT Core にはAPIs が用意されています。これらの機能により、 AWS IoT クライアントフリートをより詳細に制御でき、接続の問題のトラブルシューティングに役立ちます。

DeleteConnection API

`DeleteConnection` API を使用して、クライアント ID を指定 AWS IoT Core して MQTT デバイスを から切断します。 IDs クライアントを切断すると、 はクライアントを AWS IoT Core メッセージブローカーから AWS IoT Core 切断し、オプションでセッション状態をクリーンアップして Last Will and Testament (LWT) メッセージを抑制できます。

`DeleteConnection` API を呼び出すと、 AWS IoT Core はクリーンな切断を確保するためにいくつかのアクションを実行します。 AWS IoT Core まず、 は MQTT 切断メッセージをクライアントに送

信して MQTT セッションを終了します。その後、サービスは基盤となる TCP/TLS ソケットを閉じます。

メッセージブローカーはデバイスに DISCONNECT パケットを送信し、切断理由 API_INITIATED_DISCONNECT でライフサイクルイベントを発行します。これにより、切断が開始されたのが、クライアントやネットワークの問題ではなく、API を介したものである場合を特定することができます。これらのイベントは、可視性、トラブルシューティング、監査の目的でモニタリングできます。たとえば、 AWS IoT ルールを使用してこれらのイベントを処理し、クライアントが切断された日時と理由を追跡できます。

cleanSession パラメータを に設定するとtrue、 はすべてのサブスクリプションとキューに入れられたメッセージを含むクライアントのセッション状態 AWS IoT Core を削除します。セッションをクリーンアップすると、永続セッションは終了します。クライアントが永続セッションで、かつ preventWillMessage パラメータが false に設定されている場合、サービスは、使用可能な場合、LWT メッセージをディスパッチします。これは、計画的なメンテナンス実施時に役立ちます。

DeleteConnection API を呼び出すと、切断プロセスはすぐに開始されますが、クライアントが切断を認識する正確なタイミングは、ネットワーク条件とクライアントの実装によって異なります。ほとんどのクライアントは数秒以内に切断を検出しますが、ネットワーク接続が不十分な場合、クライアントが切断されていることを認識するまでに時間がかかることがあります。

> **Note**
>
> 切断を強制的に行うと、クライアントは新しいセッション状態で再認証および再認可する必要があります。API コール自体は、クライアントの再接続を妨げません。そのため、必要に応じて、DeleteConnection API コールを発行する前に、クライアントの認証情報またはポリシーをさらに変更する必要があります。

料金については、「AWS IoT Core の料金」を参照してください。

ユースケース

DeleteConnection API は、問題のある動作を示したり、リソースを過剰に消費したりするクライアントの誤った動作を管理するのに役立ちます。切断を強制することで、クライアントが適切な認証と認可を使用して接続を再確立することができ、リソース消費の問題の解消につながります。

クライアントリダイレクトのシナリオでも、この API は役立ちます。クライアントを別のエンドポイントまたは にリダイレクトする必要がある場合は AWS リージョン、それらをプログラムで切断し、DNS 設定を変更して別の AWS IoT Core エンドポイントに再接続できます。この API は、渋滞

した接続を解決したり、通常のオペレーションを妨げている可能性のある問題のあるセッション状態をクリアしたりするのに役立ちます。

API パラメータ

`DeleteConnection` API は、次のパラメータを受け入れます。

clientId (必須)

切断する MQTT クライアントの一意の識別子。これは URL パスで指定されます。クライアント ID はドル記号 ($) で始めることはできません。

> **Note**
>
> MQTT クライアント ID には、HTTP リクエストで無効な文字を含めることができます。`DeleteConnection` API を使用する場合は、MQTT では有効でも、HTTP では無効なクライアント ID の文字をすべて URL エンコード (パーセントエンコード) する必要があります。これには、スペース、スラッシュ (/)、UTF-8 の文字などの特殊文字が含まれます。たとえば、スペースは %20、スラッシュは %2F、UTF-8 の文字の ü は %C3%BC になります。適切なエンコーディングにより、HTTP ベースの API コールで MQTT クライアント ID が正しく送信されます。

cleanSession (オプション)

切断時にクライアントのセッション状態を削除するかどうかを指定します。サブスクリプションやキューに入れられたメッセージを含むすべてのセッション情報を削除するには、`true` に設定します。セッション状態を保持するには、`false` に設定します。デフォルトでは、これは `false` に設定されます (セッション状態を保持します) 。クリーンセッションの場合、このパラメータは無視されます。

preventWillMessage (オプション)

切断時に利用可能な場合、 が Last Will and Testament (LWT) メッセージを AWS IoT Core ディスパッチするかどうかを制御します。LWT メッセージをディスパッチしないようにする場合は、`true` に設定します。ディスパッチを許可するには、`false` に設定します。デフォルトでは、これは `false` に設定されます (使用可能な場合は LWT をディスパッチします) 。

API 構文

`DeleteConnection` API は次の HTTP リクエスト形式を使用します。

```
DELETE /connections/<clientId>?
cleanSession=<cleanSession>&preventWillMessage=<preventWillMessage> HTTP/1.1
```

リクエストの例:

```
// 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
```

リクエストが成功すると、レスポンス本文なしで「HTTP 200 OK」が返されます。

> **Note**
>
> AWS 署名バージョン 4 がリクエストに署名するために使用するサービス名は、iotdevicegateway です。`aws iot describe-endpoint --endpoint-type iot:Data-ATS` コマンドを使用すると、エンドポイントを見つけることができます。

必要なアクセス許可

`DeleteConnection` API を使用するには、次の IAM アクセス許可が必要です。

```
iot:DeleteConnection
```

リソースベースのポリシーを使用して、特定のクライアント ID にこのアクセス許可の適用範囲を指定できます。例えば、次のようになります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteConnection",
            "Resource": "arn:aws:iot:region:account:client/myDevice*"
        }
```

```
    ]
}
```

重要な考慮事項

切断されたクライアントは、再接続を防ぐために追加のロジックが実装されていない限り、すぐに再接続を試みることができます。切断オペレーションは現在の接続のみを終了し、再接続することをを妨げるものではありません。再接続を防ぐ必要がある場合は、クライアント側のロジックを実装するか、デバイスの認証情報を無効にすることを検討してください。

レート制限は、標準の API レート制限の一部として AWS IoT Core API に適用されます。一括切断オペレーションを計画するときは、これらの制限を考慮し、スロットリングを避けるために適切な再試行ロジックとバッチ処理戦略を実装してください。詳細については、「AWS IoT Core エンドポイントとクォータ」を参照してください。

エラーレスポンス

`DeleteConnection` API は次のエラーレスポンスを返すことができます。

InvalidRequestException

リクエストが無効である。これは、クライアント ID 形式が無効である場合、ドル記号 ($) がプレフィックスとして含まれている場合、または必須パラメータが欠落している場合に発生する可能性があります。

ResourceNotFoundException

指定されたクライアント ID が存在しないか、現在接続されておらず、永続セッションがありません。

UnauthorizedException

この操作を実行する権限がありません。必要な `iot:DeleteConnection` 権限があることを確認してください。

ForbiddenException

発信者にはリクエストを行う権限がありません。これは、IAM アクセス許可が不十分であるか、リソースベースのポリシー制限が原因で発生する可能性があります。

ThrottlingException

レートが制限を超えています。API コールの頻度を減らし、エクスポネンシャルバックオフを使用して適切な再試行ロジックを実装してください。

InternalFailureException

予期しないエラーが発生しました。しばらくしてからリクエストを再試行してください。

ServiceUnavailableException

サービスが一時的に使用できません しばらくしてからリクエストを再試行してください。

HTTPS での発行

クライアントは、HTTP 1.0 または 1.1 プロトコルを使用して REST API にリクエストを実行することで、メッセージをパブリッシュできます。HTTP リクエストで使用される認証およびポートマッピングについては、「???」を参照してください。

> **Note**
>
> HTTPS は MQTT のような `clientId` 値をサポートしていません。`clientId` は MQTT を使用する場合は使用できますが、HTTPS を使用する場合は使用できません。

HTTPS メッセージ URL

デバイスとクライアントは、クライアント固有のエンドポイントとトピック固有の URL に POST リクエストを行うことで、メッセージを発行します。

```
https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1
```

- *IoT_data_endpoint* は、AWS IoT デバイスのデータエンドポイントです。コマンドを使用して AWS CLI 、 AWS IoT コンソールのモノの詳細ページまたはクライアントでエンドポイントを見つけることができます。

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

  エンドポイントは次のようになります: `a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`
- *url_encoded_topic_name* は、送信されるメッセージの完全なトピック名です。

HTTPS メッセージコードの例

AWS IoTに HTTPS メッセージを送信する方法の例をいくつか示します。

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

クライアントまたはデバイスから curl を使用して、 AWS IoTにメッセージを送信できます。

curl を使用して AWS IoT クライアントデバイスからメッセージを送信するには

1. curl バージョンを確認します。
   a. クライアントで、コマンドプロンプトからこのコマンドを実行します。

      curl --help

ヘルプテキストで、TLS オプションを探します。`--tlsv1.2` オプションが表示されます。

b. `--tlsv1.2` オプションが表示された場合は、続行します。

c. `--tlsv1.2` オプションが表示されない場合、または `command not found` エラーが発生した場合は、続行する前にクライアントで curl を更新またはインストールするか、`openssl` をインストールします。

2. クライアントに証明書をインストールします。

AWS IoT コンソールでクライアント (モノ) を登録したときに作成した証明書ファイルをコピーします。続行する前に、クライアントに次の 3 つの証明書ファイルがあることを確認します。

- CA 証明書ファイル (この例では *Amazon-root-CA-1.pem*)。
- クライアントの証明書ファイル (この例では *device.pem.crt*)。
- クライアントのプライベートキーファイル (この例では *private.pem.key*)。

3. curl コマンドラインを作成し、アカウントとシステムの置き換え可能な値を置き換えます。

```
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) を使用します。

--cacert *Amazon-root-CA-1.pem*

ピアを検証する CA 証明書のファイル名とパス (必要な場合)。

--cert *device.pem.crt*

クライアントの証明書のファイル名とパス (必要な場合)。

--key *private.pem.key*

クライアントのプライベートキーのファイル名とパス (必要な場合)。

--request POST

HTTP リクエストのタイプ (この場合は POST)。

--data "*{ \"message\": \"Hello, world\" }*"

パブリッシュ先の HTTP POST データ。この場合、内部の引用符がバックスラッシュ文字 (\) でエスケープされた JSON 文字列です。

"https://*IoT_data_endpoint*:8443/topics/*topic*?qos=1"

この場合、クライアントの AWS IoT デバイスデータエンドポイントの URL の後に HTTPS ポート が続き:8443、その後にキーワード /topics/が続きtopic、トピック名が続きます。サービス品質をクエリパラメータ ?qos=1 として指定します。

4. AWS IoT コンソールで MQTT テストクライアントを開きます。

   MQTT クライアントを使用して AWS IoT MQTT メッセージを表示する の手順に従い、トピック名が curl コマンドで使用されている####のメッセージをサブスクライブするか、ワイルドカードトピックフィルター # を使用するようコンソールを設定します。

5. コマンドをテストします。

   AWS IoT コンソールのテストクライアントでトピックをモニタリングしながらクライアントに移動し、ステップ 3 で作成した curl コマンドラインを発行します。コンソールにクライアントのメッセージが表示されます。

MQTT トピック

MQTT トピックは AWS IoT messages. AWS IoT clients を識別し、メッセージトピック名を指定して発行するメッセージを識別します。クライアントは、トピックフィルターを AWS IoT Coreに登録して、サブスクライブ (受信) するメッセージを識別します。メッセージブローカーはトピック名とトピックフィルターを使用して、パブリッシュするクライアントからサブスクライブするクライアントに、 メッセージを振り分けます。

メッセージブローカーは、トピックを使用して、MQTT を使用して送信されたメッセージと、HTTP を使用して HTTPS メッセージ URL に送信されたメッセージを識別します。

は一部の予約済みシステムトピック AWS IoT をサポートしていますが、ほとんどの MQTT トピックは、次のセクションで説明するように、システムデザイナーであるユーザーが作成および管理します。 はトピック AWS IoT を使用して、発行クライアントから受信したメッセージを識別し、サブスクライブしているクライアントに送信するメッセージを選択します。システムのトピック名前空間を

作成する前に、MQTT トピックの特性を確認して、IoT システムに最適なトピック名の階層を作成します。

トピック名

トピック名とトピックフィルターは、UTF-8 エンコードされた文字列です。スラッシュ (/) 文字を使用して階層のレベルを区切ることにより、情報の階層を表すことができます。たとえば、このトピック名は、部屋 1 の温度センサーを表すことができます。

- `sensor/temperature/room1`

この例では、次のようなトピック名を持つ他の種類のセンサーが他の部屋にある場合もあります。

- `sensor/temperature/room2`
- `sensor/humidity/room1`
- `sensor/humidity/room2`

> **Note**
>
> システム内のメッセージのトピック名を考慮する場合は、次の点に注意してください。
>
> - トピック名とトピックフィルターでは、大文字と小文字が区別されます。
> - トピック名に個人を特定できる情報を含めることはできません。
> - $ で始まるトピック名は、 AWS IoT Coreのみが使用する予約済みのトピックです。
> - AWS IoT Core は、 AWS アカウントまたはリージョン間でメッセージを送受信できません。

トピック名と名前空間の設計の詳細については、ホワイトペーパーの「AWS IoT Coreの MQTT トピックの設計」を参照してください。

アプリケーションがメッセージを発行およびサブスクライブする方法の例については、AWS IoT Core チュートリアルの開始方法 と AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Client から始めてください。

> **Important**
>
> トピック名前空間は、 AWS アカウント および リージョンに制限されています。たとえば、あるリージョン AWS アカウント の で使用されるsensor/temp/room1トピックは、別のリージョンの同じ AWS アカウントで使用されるトピック、または任意のリージョンの他のアカウントで使用されるsensor/temp/room1トピックとは異なり AWS アカウント ます。

トピック ARN

すべてのトピック ARN (Amazon リソースネーム) は、次のフォーマットを備えています。

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

例えば、arn:aws:iot:us-west-2:123EXAMPLE456:topic/application/topic/device/sensor はトピック　application/topic/device/sensor の ARN です。

トピック名フィルター

サブスクライブするクライアントは、 メッセージブローカーにトピック名フィルターを登録して、メッセージブローカーがそのトピック名フィルターに送信する必要のあるメッセージトピックを指定します。トピック名フィルターは、単一のトピック名にサブスクライブする単一のトピック名にすることも、ワイルドカード文字を使用して複数のトピック名に同時にサブスクライブすることもできます。

発行するクライアントは、発行するトピック名にワイルドカード文字を使用できません。

次の表は、トピックフィルターで使用できるワイルドカード文字の一覧です。

トピックのワイルドカード

ワイルドカード文字	マッチ	コメント
#	トピック階層内のそのレベル以下のすべての文字列。	トピックフィルターの最後の文字にする必要があります。 トピック階層のレベルで唯一の文字である必要があります。

ワイルドカード文字	マッチ	コメント
		+ ワイルドカード文字を含むトピックフィルターで使用できます。
+	文字を含むレベル内の任意の文字列。	トピック階層のレベルで唯一の文字である必要があります。 トピックフィルターの複数のレベルで使用できます。

前述のセンサートピック名でのワイルドカードの使用例:

- `sensor/#` へのサブスクリプションでは `sensor/`、`sensor/temperature`、`sensor/temperature/room1` にパブリッシュされたメッセージを受信しますが、`sensor` にパブリッシュされたメッセージは受信しません。
- `sensor/+/room1` のサブスクリプションでは、`sensor/temperature/room1` および `sensor/humidity/room1` にパブリッシュされたメッセージを受信しますが、`sensor/temperature/room2` または `sensor/humidity/room2` に送信されたメッセージは受信しません。

トピックフィルターの ARN

すべてのトピックフィルター ARN (Amazon リソースネーム) は、次の形式になります。

```
arn:aws:iot:aws-region:AWS-account-ID:topicfilter/TopicFilter
```

例えば、`arn:aws:iot:us-west-2:123EXAMPLE456:topicfilter/application/topic/+/sensor` はトピックフィルター `application/topic/+/sensor` の ARN です。

MQTT メッセージペイロード

MQTT メッセージで送信されるメッセージペイロードは、 の 1 つでない限り AWS IoT、 によって指定されませんthe section called “予約済みトピック”。アプリケーションのニーズに対応するため、プロトコルのAWS IoT Core Service Quotas の制約内でトピックのメッセージペイロードを定義することを推奨します。

メッセージペイロードに JSON 形式を使用すると、 AWS IoT ルールエンジンはメッセージを解析し、SQL クエリを適用できます。ルールエンジンがメッセージペイロードに SQL クエリを適用することをアプリケーションが必要としない場合は、アプリケーションで必要な任意のデータ形式を使用できます。SQL クエリで使用される JSON ドキュメントの制限事項と予約されている文字については、JSON 拡張 を参照してください。

MQTT トピックとそれに対応するメッセージペイロードの設計の詳細については、「AWS IoT Core の MQTT トピックの設計」を参照してください。

メッセージサイズの限度がサービスクォータを超えると、理由 `PAYLOAD_LIMIT_EXCEEDED` の `CLIENT_ERROR` になり、「メッセージペイロードがメッセージタイプのサイズ限度を超えています」と表示されます。メッセージサイズ制限の詳細については、「AWS IoT Core メッセージブローカーの限度とクォータ」を参照してください。

予約済みトピック

ドル記号 ($) で始まるトピックは、 で使用するために予約されています AWS IoT。これらの予約済みトピックは、許可されているとおりにサブスクライブおよび発行できます。ただし、ドル記号で始まる新しいトピックを作成することはできません。予約済みトピックへのサポートされていないパブリッシュまたはサブスクライブオペレーションにより、接続が終了することがあります。

アセットモデルのトピック

トピック	クライアントオペレーションを許可する	説明
$aws/sitewise/asset-models/*assetModelId* / assets/*assetId*/properties/*propertyId*	Subscribe	AWS IoT SiteWise は、このトピックにアセットプロパティ通知を発行します。詳細については、「 AWS IoT SiteWise ユーザーガイド」の「他の AWS サービスとやり取りする」を参照してください。

AWS IoT Device Defender トピック

これらのメッセージは、トピックの*#######*に応じて、簡潔なバイナリオブジェクト表現 (CBOR) 形式のレスポンスバッファと JavaScript Object Notation (JSON) をサポートします。 AWS IoT Device Defender トピックは MQTT パブリッシュのみをサポートします。

#######	レスポンス形式のデータ型
cbor	簡潔なバイナリオブジェクトの表現 (CCOR)
json	JavaScript Object Notation (JSON)

詳細については、「デバイスからのメトリクスの送信」を参照してください。

Topic	許可されている操作	説明
$aws/things/*thingName* /defender/metrics/*payload-format*	発行	AWS IoT Device Defender エージェントはこのトピックにメトリクスを発行します。詳細については、「デバイスからのメトリクスの送信」を参照してください。
$aws/things/*thingName* /defender/metrics/*payload-format* /accepted	Subscribe	AWS IoT は、 AWS IoT Device Defender エージェントが成功したメッセージを $aws/things/*thingName* /defender/metrics/*payload-format* に発行した後、このトピックに発行します。詳細については、「デバイスからのメトリクスの送信」を参照してください。
$aws/things/*thingName* /defender/metrics/*payload-format* /rejected	Subscribe	AWS IoT は、 AWS IoT Device Defender エージェントが失敗したメッセージを $aws/things/*thingName* /defender/metrics/*payload-format* に発行した後に、このトピックに発行します。詳細については、「デバイスからのメトリクスの送信」を参照してください。

AWS IoT Core デバイスの場所に関するトピック

AWS IoT Core Device Location は、デバイスからの測定データを解決し、IoT デバイスの推定場所を提供できます。デバイスからの測定データには、GNSS、Wi-Fi、セルラー、IP アドレスを含めることができます。次に AWS IoT Core 、Device Location は、最高の精度を提供し、デバイスの位置情報を解決する測定タイプを選択します。詳細については、「AWS IoT Core デバイスの場所」および「Device Location MQTT トピックを使用した AWS IoT Core デバイスロケーションの解決」を参照してください。

Topic	許可されている操作	説明
$aws/device_location/*customer_device_id*/get_position_estimate	発行	デバイスは、このトピックに発行して、スキャンされた未加工の測定データを AWS IoT Core Device Location で解決します。
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	Subscribe	AWS IoT Core Device Location は、デバイスの場所が正常に解決されると、このトピックに発行されます。
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	Subscribe	AWS IoT Core Device Location は、4xx エラーが原因でデバイスの位置を正常に解決できない場合に、このトピックに発行されます。

イベントのトピック

イベントメッセージは、特定のイベントが発生したときにを発行されます。例えば、モノが追加、更新、または削除されると、イベントがレジストリによって生成されます。この表は、さまざまな AWS IoT イベントとその予約済みトピックを示しています。

Topic	クライアントオペレーションを許可する	説明
$aws/events/certificates/registered/*caCertificateId*	Subscribe	AWS IoT は、 が証明書 AWS IoT を自動的に登録し、クライアントが証明書を PENDING_ACTIVATION ステータ

Topic	クライアントオペレーションを許可する	説明
		スで提示すると、このメッセージを発行します。詳細については、「the section called “自動登録のためのクライアントによる最初の接続の設定します”」を参照してください。
$aws/events/job/*jobID*/canceled	Subscribe	AWS IoT は、ジョブがキャンセルされたときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobID*/cancellation_in_progress	Subscribe	AWS IoT は、ジョブのキャンセルが進行中のときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobID*/completed	Subscribe	AWS IoT は、ジョブが完了するとこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobID*/deleted	Subscribe	AWS IoT は、ジョブが削除されるとこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobID*/deletion_in_progress	Subscribe	AWS IoT は、ジョブの削除が進行中のときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobID*/canceled	Subscribe	AWS IoT は、ジョブ実行がキャンセルされたときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。

Topic	クライアントオペレーションを許可する	説明
$aws/events/jobExecution/*jobID*/deleted	Subscribe	AWS IoT は、ジョブ実行が削除されると、このメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobID*/failed	Subscribe	AWS IoT は、ジョブの実行が失敗したときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobID*/rejected	Subscribe	AWS IoT は、ジョブ実行が拒否されたときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobID*/removed	Subscribe	AWS IoT は、ジョブ実行が削除されたときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobID*/succeeded	Subscribe	AWS IoT は、ジョブの実行が成功したときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobID*/timed_out	Subscribe	AWS IoT は、ジョブ実行がタイムアウトしたときにこのメッセージを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/presence/connected/*clientId*	Subscribe	AWS IoT は、指定されたクライアント ID を持つ MQTT クライアントが接続するときに、このトピックに発行します AWS IoT。詳細については、「接続/切断イベント」を参照してください。

Topic	クライアントオペレーションを許可する	説明
$aws/events/presence/disconnected/*clientId*	Subscribe	AWS IoT は、指定されたクライアント ID を持つ MQTT クライアントが切断されると、このトピックに発行します AWS IoT。詳細については、「接続/切断イベント」を参照してください。
$aws/events/subscriptions/subscribed/*clientId*	Subscribe	AWS IoT は、指定されたクライアント ID を持つ MQTT クライアントが MQTT トピックをサブスクライブするときに、このトピックに発行します。詳細については、「サブスクライブ/サブスクライブ解除イベント」を参照してください。
$aws/events/subscriptions/unsubscribed/*clientId*	Subscribe	AWS IoT 指定されたクライアント ID を持つ MQTT クライアントが MQTT トピックへのサブスクライブを解除すると、 はこのトピックに発行します。詳細については、「サブスクライブ/サブスクライブ解除イベント」を参照してください。
$aws/events/thing/*thingName* /created	Subscribe	AWS IoT *thingName* モノが作成されると、 はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thing/*thingName* /updated	Subscribe	AWS IoT *thingName* モノが更新されると、 はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thing/*thingName* /deleted	Subscribe	AWS IoT *thingName* モノが削除されると、 はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。

Topic	クライアントオペレーションを許可する	説明
$aws/events/thingGroup/*thingGroupName*/created	Subscribe	AWS IoT モノのグループ *thingGroupName* が作成されると、はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingGroup/*thingGroupName*/updated	Subscribe	AWS IoT モノのグループ *thingGroupName* が更新されると、はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingGroup/*thingGroupName*/deleted	Subscribe	AWS IoT モノのグループ *thingGroupName* が削除されると、はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingType/*thingTypeName*/created	Subscribe	AWS IoT *thingTypeName* モノのタイプが作成されると、はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingType/*thingTypeName*/updated	Subscribe	AWS IoT *thingTypeName* モノのタイプが更新されると、はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingType/*thingTypeName*/deleted	Subscribe	AWS IoT *thingTypeName* モノのタイプが削除されると、はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。

Topic	クライアントオペレーションを許可する	説明
$aws/events/thingTypeAssociation/thing/*thingName*/*thingTypeName*	Subscribe	AWS IoT *thingName* が ThingTypeName のモノタイプと関連付けられているか、関連付けが解除されると、 はこのトピックに発行されます*thingTypeName* 。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingGroupMembership/thingGroup/*thingGroupName* /thing/*thingName* /added	Subscribe	AWS IoT *thingName* *thingGroupName* に追加されると、 はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingGroupMembership/thingGroup/*thingGroupName* /thing/*thingName* /removed	Subscribe	AWS IoT *thingName* *thingGroupName* から削除されると、 はこのトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName* /childThingGroup/*childThingGroupName* /added	Subscribe	AWS IoT は、モノのグループ *childThingGroupName* がモノのグループ *parentThingGroupName* に追加されると、このトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。

Topic	クライアントオペレーションを許可する	説明
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName* /childThingGroup/*childThingGroupName* /removed	Subscribe	AWS IoT は、モノのグループ *childThingGroupName* がモノのグループ *parentThingGroupName* から削除されると、このトピックに発行します。詳細については、「the section called “登録イベント”」を参照してください。

フリートプロビジョニングのトピック

> **Note**
>
> この表の「Receive」というクライアントオペレーションは、クライアントがトピックをサブスクライブしているかどうかにかかわらず、リクエストしたクライアントに直接 AWS IoT 発行するトピックを示しています。クライアントは、応答メッセージにサブスクライブしていない場合でも、それらを受信する場合があることを想定する必要があります。これらの応答メッセージはメッセージブローカーを通過せず、他のクライアントまたはルールによってサブスクライブする事はできません。

これらのメッセージは、トピックの*#######*に応じて、簡潔なバイナリオブジェクト表現 (CCOR) 形式および JavaScript Object Notation (JSON) 形式のレスポンスバッファをサポートします。

#######	レスポンス形式のデータ型
cbor	簡潔なバイナリオブジェクトの表現 (CCOR)
json	JavaScript Object Notation (JSON)

詳細については、「デバイスプロビジョニング MQTT API」を参照してください。

トピック	クライアントオペレーションを許可する	説明
$aws/certificates/create/*payload-format*	発行	証明書署名リクエスト (CSR) から証明書を作成するには、このトピックに発行します。
$aws/certificates/create/*payload-format* /accepted	サブスクライブ、受信	AWS IoT は、$aws/certificates/create/*payload-format* を正常に呼び出した後、このトピックに発行します。
$aws/certificates/create/*payload-format* /rejected	サブスクライブ、受信	AWS IoT は、$aws/certificates/create/*payload-format* への呼び出しが失敗した後に、このトピックに発行します。
$aws/certificates/create-from-csr/*payload-format*	発行	このトピックに発行して、CSR から証明書を作成します。
$aws/certificates/create-from-csr/*payload-format* /accepted	サブスクライブ、受信	AWS IoT は、$aws/certificates/create-from-csr/*payload-format* への正常な呼び出しをこのトピックに発行します。
$aws/certificates/create-from-csr/*payload-format* /rejected	サブスクライブ、受信	AWS IoT は、このトピックに $aws/certificates/create-from-csr/*payload-format* への呼び出しに失敗しました。
$aws/provisioning-templates/*templateName* /provision/*payload-format*	発行	モノを登録するには、このトピックに発行します。
$aws/provisioning-templates/*templateName* /provision/*payload-format* /accepted	サブスクライブ、受信	AWS IoT は、$aws/provisioning-templates/*templateName* /provision/*payload-format* を正常に呼び出した後、このトピックに発行します。
$aws/provisioning-templates/*templateN*	サブスクライブ、受信	AWS IoT は、$aws/provisioning-templates/*templateName* /provision/*payload-*

トピック	クライアントオペレーションを許可する	説明
ame /provision/*payload-f ormat* /rejected		*format* への呼び出しが失敗した後に、このトピックに発行します。

ジョブのトピック

> **Note**
>
> この表の「Receive」というクライアントオペレーションは、クライアントがトピックをサブスクライブしているかどうかにかかわらず、リクエストしたクライアントに直接 AWS IoT 発行するトピックを示しています。クライアントは、応答メッセージにサブスクライブしていない場合でも、それらを受信する場合があることを想定する必要があります。
> これらの応答メッセージはメッセージブローカーを通過せず、他のクライアントまたはルールによってサブスクライブする事はできません。ジョブアクティビティ関連のメッセージをサブスクライブするには、notifyおよび notify-nextトピックを使用します。
> ジョブとjobExecutionフリートモニタリングソリューション用のイベントトピックをサブスクライブする際は、まずジョブおよびジョブイベントを実行して、クラウド側でイベント受信する必要があります。
> 詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。

トピック	クライアントオペレーションを許可する	説明
$aws/things/*thingName* / jobs/get	発行	デバイスは、このトピックにメッセージを発行して、GetPendingJobExecu tions リクエストを実行します。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/get/accepted	サブスクライブ、受信	デバイスは、このトピックにサブスクライブして、GetPendingJobExecu tions から正常なレスポンスを受け取ります。詳細については、「ジョブデバ

トピック	クライアントオペレーションを許可する	説明
		イス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/get/rejected	サブスクライブ、受信	`GetPendingJobExecutions` リクエストが拒否されると、デバイスはこのトピックをサブスクライブしてレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/start-next	発行	デバイスは、このトピックにメッセージを発行して、`StartNextPendingJobExecution` リクエストを実行します。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/start-next/accepted	サブスクライブ、受信	デバイスは、このトピックをサブスクライブして、`StartNextPendingJobExecution` リクエストに対する正常なレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/start-next/rejected	サブスクライブ、受信	`StartNextPendingJobExecution` リクエストが拒否されると、デバイスはこのトピックをサブスクライブしてレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。

トピック	クライアントオペレーションを許可する	説明
$aws/things/*thingName* / jobs/*jobId*/get	発行	デバイスは、このトピックにメッセージを発行して、DescribeJobExecution リクエストを実行します。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/*jobId*/get/accepted	サブスクライブ、受信	デバイスは、このトピックをサブスクライブして、DescribeJobExecution リクエストに対する正常なレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/*jobId*/get/rejected	サブスクライブ、受信	DescribeJobExecution リクエストが拒否されると、デバイスはこのトピックをサブスクライブしてレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* / jobs/*jobId*/update	発行	デバイスは、このトピックにメッセージをパブリッシュして、UpdateJobExecution リクエストを実行します。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。

トピック	クライアントオペレーションを許可する	説明
$aws/things/*thingName*/jobs/*jobId*/update/accepted	サブスクライブ、受信	デバイスは、このトピックにサブスクライブして、UpdateJobExecution リクエストに対する正常なレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。 メモ $aws/things/*thingName*/jobs/*jobId*/update にパブリッシュするデバイスのみが、このトピックのメッセージを受信します。
$aws/things/*thingName*/jobs/*jobId*/update/rejected	サブスクライブ、受信	UpdateJobExecution リクエストが拒否されると、デバイスはこのトピックをサブスクライブしてレスポンスを受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。 メモ $aws/things/*thingName*/jobs/*jobId*/update にパブリッシュするデバイスのみが、このトピックのメッセージを受信します。

トピック	クライアントオペレーションを許可する	説明
$aws/things/*thingName* /jobs/notify	サブスクライブ、受信	デバイスは、このトピックにサブスクライブして、モノに対して保留中の実行のリストとの間でジョブの実行が追加または削除されたときに、通知を受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/things/*thingName* /jobs/notify-next	サブスクライブ、受信	デバイスはこのトピックにサブスクライブして、モノに対する次に保留中のジョブの実行が変更されたときに、通知を受け取ります。詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。
$aws/events/job/*jobId*/completed	サブスクライブ	ジョブサービスは、ジョブが完了したときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobId*/canceled	サブスクライブ	ジョブサービスは、ジョブがキャンセルされたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobId*/deleted	サブスクライブ	ジョブサービスは、ジョブが削除されたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/job/*jobId*/cancellation_in_progress	サブスクライブ	ジョブサービスは、ジョブのキャンセルが開始されたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。

トピック	クライアントオペレーションを許可する	説明
$aws/events/job/*jobId*/deletion_in_progress	サブスクライブ	ジョブサービスは、ジョブの削除が開始されたときに、イベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobId*/succeeded	サブスクライブ	ジョブサービスは、ジョブの実行が成功したときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobId*/failed	サブスクライブ	ジョブサービスは、ジョブの実行が失敗したときにイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobId*/rejected	サブスクライブ	ジョブサービスは、ジョブの実行が拒否されたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobId*/canceled	サブスクライブ	ジョブサービスは、ジョブの実行がキャンセルされたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobId*/timed_out	サブスクライブ	ジョブサービスは、ジョブの実行がタイムアウトしたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。

トピック	クライアントオペレーションを許可する	説明
$aws/events/jobExecution/*jobId*/removed	サブスクライブ	ジョブサービスは、ジョブの実行が削除されたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。
$aws/events/jobExecution/*jobId*/deleted	サブスクライブ	ジョブサービスは、ジョブの実行が削除されたときに、このトピックでイベントを発行します。詳細については、「ジョブイベント」を参照してください。

コマンドトのピック

> **Note**
>
> この表の「Receive」というクライアントオペレーションは、クライアントがトピックをサブスクライブしているかどうかにかかわらず、リクエストしたクライアントに直接 AWS IoT 発行するトピックを示しています。クライアントは、応答メッセージにサブスクライブしていない場合でも、それらを受信する場合があることを想定する必要があります。
> これらの応答メッセージはメッセージブローカーを通過せず、他のクライアントまたはルールによってサブスクライブする事はできません。

Topic	クライアントオペレーションを許可する	説明
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/request/*<PayloadFormat>* $aws/commands/*<devices>*	サブスクライブ、受信	コンソールまたは StartCommandExecution API を使用してコマンド実行を開始するリクエストが行われると、デバイスはこのトピックに関するメッセージを受信します。この場合、*<devices>* には IoT のモノまたは MQTT クライアントのいずれかを指定することができ、*<DeviceID>* には

Topic	クライアントオペレーションを許可する	説明
/*<DeviceID>* /executions/*<ExecutionId>* /request		IoT モノ名または MQTT クライアント ID を指定できます。
$aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/*<PayloadFormat>*	発行	デバイスは `UpdateCommandExecution` MQTT API を使用して、コマンド実行に関するメッセージをこのトピックに発行します。メッセージは、コンソールまたは `StartCommandExecution` API を使用してコマンド実行を開始するリクエストへのレスポンスとして発行されます。発行されたメッセージで、JSON または CBOR を *<PayloadFormat>* として使用します。
$aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/accepted /*<PayloadFormat>* $aws/commands/*<devices>* /*<DeviceID>* /executions/*<ExecutionId>* /response/accepted	サブスクライブ、受信	クラウドサービスでコマンド実行結果が正常に処理されると、 AWS IoT Device Management は /accepted トピックにレスポンスを発行します。

Topic	クライアントオペレーションを許可する	説明
$aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/response/rejected/*<PayloadFormat>* $aws/commands/*<devices>*/*<DeviceID>*/executions/*<ExecutionId>*/response/rejected	発行	クラウドサービスでコマンド実行結果の処理が失敗した場合、 AWS IoT Device Management は /rejected トピックにレスポンスを発行します。

ルールのトピック

トピック	クライアントオペレーションを許可する	説明
$aws/rules/*ruleName*	発行	デバイスまたはアプリケーションは、このトピックに発行して、ルールを直接トリガーします。詳細については、「基本的な取り込みによるメッセージングコストの削減」を参照してください。

セキュアトンネリングのトピック

トピック	クライアントオペレーションを許可する	説明
$aws/things/*thing-name*/tunnels/notify	Subscribe	AWS IoT は、このメッセージを IoT エージェントに発行して、リモートデバイスでローカルプロキシを開始します。詳細については、「the section called

トピック	クライアントオペレーションを許可する	説明
		“IoT エージェントスニペット”」を参照してください。

シャドウトピック

このセクションのトピックは、名前付きシャドウと名前のないシャドウで使用されます。それぞれで使用されるトピックは、トピックのプレフィックスでのみ異なります。この表は、各シャドウタイプで使用されるトピックのプレフィックスを示しています。

ShadowTopicPrefix 値	シャドウタイプ
$aws/things/*thingName* /shadow	名前のない (クラシック) シャドウ
$aws/things/*thingName* /shadow/name/*shadowName*	名前付きシャドウ

完全なトピックを作成するには、次の表に示すように、参照するシャドウのタイプの *ShadowTopicPrefix* を選択し、*thingName* と、*shadowName* (該当する場合) を対応する値に置き換え、トピックスタブに追加します。トピックでは大文字と小文字が区別されることに注意してください。

トピック	クライアントオペレーションを許可する	説明
ShadowTopicPrefix /delete	パブリッシュ/サブスクライブ	デバイスまたはアプリケーションは、このトピックにパブリッシュして、シャドウを削除します。詳細については、「/delete」を参照してください。
ShadowTopicPrefix /delete/accepted	サブスクライブ	Device Shadow サービスは、シャドウが削除されると、このトピックにメッセージを送信します。詳細については、

トピック	クライアントオペレーションを許可する	説明
		「/delete/accepted」を参照してください。
ShadowTopicPrefix /delete/rejected	サブスクライブ	Device Shadow サービスは、シャドウの削除リクエストが拒否されると、このトピックにメッセージを送信します。詳細については、「/delete/rejected」を参照してください。
ShadowTopicPrefix /get	パブリッシュ/サブスクライブ	アプリケーションまたはモノは、このトピックに空のメッセージをパブリッシュして、Shadow を取得します。詳細については、「Device Shadow MQTT トピック」を参照してください。
ShadowTopicPrefix /get/accepted	サブスクライブ	Device Shadow サービスは、シャドウに対するリクエストが正常に行われると、このトピックにメッセージを送信します。詳細については、「/get/accepted」を参照してください。
ShadowTopicPrefix /get/rejected	サブスクライブ	Device Shadow サービスは、シャドウのリクエストが拒否されると、このトピックにメッセージを送信します。詳細については、「/get/rejected」を参照してください。
ShadowTopicPrefix /update	パブリッシュ/サブスクライブ	モノまたはアプリケーションは、このトピックにパブリッシュして、Shadow を更新します。詳細については、「/update」を参照してください。

トピック	クライアントオペレーションを許可する	説明
ShadowTopicPrefix / update/accepted	サブスクライブ	Device Shadow サービスは、シャドウに対する更新が正常に行われると、このトピックにメッセージを送信します。詳細については、「/update/accepted」を参照してください。
ShadowTopicPrefix / update/rejected	サブスクライブ	Device Shadow サービスは、シャドウに対する更新が拒否されると、このトピックにメッセージを送信します。詳細については、「/update/rejected」を参照してください。
ShadowTopicPrefix / update/delta	サブスクライブ	Device Shadow サービスは、シャドウの reported セクションと desired セクションとの間で差分が検出されると、このトピックにメッセージを送信します。詳細については、「/update/delta」を参照してください。
ShadowTopicPrefix / update/documents	Subscribe	AWS IoT シャドウの更新が正常に実行されるたびに、 はこのトピックに状態ドキュメントを発行します。詳細については、「/update/documents」を参照してください。

MQTT ベースのファイル配信のトピック

> **Note**
>
> この表の「Receive」というクライアントオペレーションは、クライアントがトピックをサブスクライブしているかどうかにかかわらず、リクエストしたクライアントに直接 AWS IoT 発行するトピックを示しています。クライアントは、応答メッセージにサブスクライブしていない場合でも、それらを受信する場合があることを想定する必要があります。これらの応

答メッセージはメッセージブローカーを通過せず、他のクライアントまたはルールによってサブスクライブする事はできません。

これらのメッセージは、トピックの*#######*に応じて、簡潔なバイナリオブジェクト表現 (CCOR) 形式および JavaScript Object Notation (JSON) 形式のレスポンスバッファをサポートします。

#######	レスポンス形式のデータ型
cbor	簡潔なバイナリオブジェクトの表現 (CCOR)
json	JavaScript Object Notation (JSON)

トピック	クライアントオペレーションを許可する	説明
$aws/things/*ThingName*/streams/*StreamId*/data/*payload-format*	サブスクライブ、受信	AWS デバイスからのGetStream」リクエストが受け入れられた場合、MQTT ベースのファイル配信はこのトピックに発行されます。ペイロードにはストリーミングデータが含まれます。詳細については、「デバイスで AWS IoT の MQTT ベースのファイル配信の使用」を参照してください。
$aws/things/*ThingName*/streams/*StreamId*/get/*payload-format*	発行	デバイスは、このトピックに発行して、「GetStream」リクエストを実行します。詳細については、「デバイスで AWS IoT の MQTT ベースのファイル配信の使用」を参照してください。
$aws/things/*ThingName*/streams/*StreamId*/description/*payload-format*	サブスクライブ、受信	AWS デバイスからのDescribeStream」リクエストが受け入れられた場合、MQTT ベースのファイル配信はこのトピックに発行されます。ペ

トピック	クライアントオペレーションを許可する	説明
		イロードには、ストリーミングの説明が含まれます。詳細については、「デバイスで AWS IoT の MQTT ベースのファイル配信の使用」を参照してください。
$aws/things/*ThingName*/streams/*StreamId*/describe/*payload-format*	発行	デバイスは、このトピックに発行して、「DescribeStream」リクエストを実行します。詳細については、「デバイスで AWS IoT の MQTT ベースのファイル配信の使用」を参照してください。
$aws/things/*ThingName*/streams/*StreamId*/rejected/*payload-format*	サブスクライブ、受信	AWS MQTT ベースのファイル配信は、デバイスからのDescribeStream」またはGetStream」リクエストが拒否された場合に、このトピックに発行されます。詳細については、「デバイスで AWS IoT の MQTT ベースのファイル配信の使用」を参照してください。

予約済みトピック ARN

すべての予約済みトピック ARN (Amazon リソースネーム) は、次の形式です。

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

例えば、`arn:aws:iot:us-west-2:123EXAMPLE456:topic/$aws/things/thingName/jobs/get/accepted` は予約済みトピック `$aws/things/thingName/jobs/get/accepted` の ARN です。

ドメイン設定

では AWS IoT Core、ドメイン設定を使用して、データエンドポイントの動作を設定および管理できます。ドメイン設定では、複数の AWS IoT Core データエンドポイントを生成し、独自の完全修飾ドメイン名 (FQDN) および関連するサーバー証明書でカスタマイズし、カスタムオーソライザーを関連付けることができます。詳細については、「カスタム認証と認可」を参照してください。

> **Note**
>
> この機能は では使用できません AWS GovCloud (US) AWS リージョン。

この章の内容:

- ドメイン設定とは
- AWS マネージドドメインの作成と設定
- カスタマーマネージドドメインの作成と設定
- ドメイン設定の管理
- ドメイン設定での TLS 設定の設定
- OCSP ステープリングのサーバー証明書設定

ドメイン設定とは

では AWS IoT Core、ドメイン設定は、 AWS IoT Core データエンドポイントのドメイン (AWS マネージドドメインまたはカスタマーマネージドドメイン) のセットアップと設定を指します。 は、デバイスが通信するための AWS アカウント (`iot:Data-ATS`) のデフォルトエンドポイント AWS IoT Core も提供します AWS IoT Core。

このトピックの内容

- ユースケース
- 主要なコンセプト
- 重要な注意事項

ユースケース

ドメイン設定を使用して、次のようなタスクを簡素化できます。

- デバイスを に移行します AWS IoT Core。
- 異なるデバイスタイプに対して別々のドメイン設定を維持することにより、異種デバイスフリートをサポートする。
- アプリケーションインフラストラクチャを に移行するときは、ブランドアイデンティティ (ドメイン名など) を維持します AWS IoT Core。

主要なコンセプト

以下の概念、ドメイン設定および関連する詳細を示しています。

- ドメイン設定

  AWS IoT Core エンドポイントのドメインのセットアップと設定。

- デフォルトのエンドポイントドメイン

  などのデフォルトのエンドポイントで AWS IoT が提供するドメイン`iot:Data-ATS`。デフォルトのエンドポイントを検索するには、describe-endpoint または describe-domain-configuration CLI コマンドを実行します。または、 AWS IoT Core コンソールに移動し、左側のナビゲーションで [接続] から [ドメイン設定] を選択します。デフォルトのエンドポイントは、`iot:Data-ATS` という名前でリストされます。

- AWS マネージドドメイン

  AWS が管理するドメイン。 AWS マネージドドメインを選択すると、 が提供するデータエンドポイントを使用してデバイスが接続されます AWS。 AWS はドメインと証明書を管理します。

- カスタマーマネージドドメイン

  ユーザーが管理するドメイン。カスタムドメインとも呼ばれます。カスタマーマネージドドメインを選択すると、デバイスはカスタムドメインデータエンドポイントを使用して接続されます。ユーザーがドメインと証明書を管理します。カスタマーマネージドドメインを使用すると、ニーズに合わせてエンドポイント URL をカスタマイズできます。例えば、カスタムドメイン名 (`your-domain-name.com`) を使用するか、特定のアクセスポリシーを適用できます。

- 認証タイプ

AWS IoT Coreへの接続時にデバイスを認証するために選択する認証タイプ。ドメイン設定を作成するときは、認証タイプを指定する必要があります。詳細については、「???」を参照してください。

- アプリケーションプロトコル

  AWS IoT Coreへの接続時にデバイスが使用するアプリケーションレイヤープロトコル。ドメイン設定を作成するときは、アプリケーションプロトコルを指定する必要があります。詳細については、「???」を参照してください。

重要な注意事項

AWS IoT Core は、サーバー名表示 (SNI) TLS 拡張機能を使用してドメイン設定を適用します。デバイスを に接続すると AWS IoT Core、クライアントはサーバー名表示 (SNI) 拡張機能を送信できます。これは、マルチアカウント登録、設定可能なエンドポイント、カスタムドメイン、VPC エンドポイントなどの機能に必要です。また、ドメイン設定で指定したドメイン名と同じサーバー名を渡す必要があります。このサービスをテストするには、GitHub の AWS IoT デバイス SDK の v2 バージョンを使用します。

で複数のデータエンドポイントを作成すると AWS アカウント、MQTT トピック、デバイスシャドウ、ルールなどのリソースが共有 AWS IoT Core されます。

AWS IoT Core カスタムドメイン設定のサーバー証明書を指定すると、証明書には最大 4 つのドメイン名があります。詳細については、「AWS IoT Core エンドポイントとクォータ」を参照してください。

AWS マネージドドメインの作成と設定

AWS マネージドドメインで設定可能なエンドポイントを作成するには、CreateDomainConfiguration API を使用します。 AWS マネージドドメインのドメイン設定は、以下で構成されます。

- `domainConfigurationName`

  ドメイン設定を識別するユーザー定義の名前と値は、 に一意である必要があります AWS リージョン。`IoT:` で始まるドメイン設定名は、デフォルトのエンドポイント用に予約されており、使用できません。

- `defaultAuthorizerName` (オプション)

エンドポイントで使用するカスタムオーソライザーの名前。

- `allowAuthorizerOverride` (オプション)

  リクエストの HTTP ヘッダーで別のオーソライザーを指定することによって、デバイスがデフォルトのオーソライザーを上書きできるかどうかを指定するブール値。`defaultAuthorizerName` の値が指定されている場合、この値は必須です。

- `serviceType` (オプション)

  エンドポイントが配信するサービスタイプ。 はDATAサービスタイプ AWS IoT Core のみをサポートします。DATA を指定すると、 AWS IoT Core はエンドポイントタイプが `iot:Data-ATS` のエンドポイントを返します。設定可能な `iot:Data` （ VeriSign ）エンドポイントを作成することはできません。

- `TlsConfig` (オプション)

  ドメインの TLS 構成を指定するオブジェクト。詳細については、「???」を参照してください。

次の AWS CLI コマンド例では、Dataエンドポイントのドメイン設定を作成します。

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
```

コマンドの出力は以下のようになります。

```
{
    "domainConfigurationName": "myDomainConfigurationName",
    "domainConfigurationArn": "arn:aws:iot:us-east-1:123456789012:domainconfiguration/
myDomainConfigurationName/itihw"
}
```

カスタマーマネージドドメインの作成と設定

ドメイン設定では、 AWS IoT Coreに接続するカスタムの完全修飾ドメイン名 (FQDN) を指定できます。カスタマーマネージドドメイン (カスタムドメインとも呼ばれます) を使用する多くの利点があります。 ブランド化の目的で、自分のドメインまたは会社のドメインを顧客に公開できます。 新しいブローカーを指すように独自のドメインを簡単に変更できます。 マルチテナンシーをサポートして、同じ 内の異なるドメインを持つ顧客にサービスを提供できます AWS アカウント。 独自のサー

バー証明書の詳細を管理できます。 証明書の署名に使用されるルート認証局 (CA) など、 署名アルゴリズム 証明書チェーンの深さ、 と証明書のライフサイクル。

カスタムドメインを使用してドメイン設定をセットアップするワークフローは、次の 3 つの段階で構成されます。

1. でのサーバー証明書の登録 AWS Certificate Manager
2. ドメイン設定の作成
3. DNS レコードの作成

証明書マネージャーへのサーバー AWS 証明書の登録

カスタムドメインを使用してドメイン設定を作成する前に、サーバー証明書チェーンを AWS Certificate Manager (ACM) に 登録する必要があります。次の 3 種類のサーバー証明書を使用できます。

- ACM によって生成されたパブリック証明書
- 公開 CA によって署名された外部証明書
- プライベート CA によって署名された外部証明書

> **Note**
>
> AWS IoT Core は、証明書が Mozilla の信頼できる ca バンドルに含まれている場合、パブリック CA によって署名されると見なします。

証明書の要件

証明書を ACM にインポートするための要件については、証明書をインポートするための前提条件を参照してください。これらの要件に加えて、 AWS IoT Core では次の要件が追加されます。

- リーフ証明書には、serverAuth (TLS Web Server Authentication) の値を持つ Extended Key Usage x509 v3 拡張が含まれている必要があります。ACM から証明書をリクエストすると、この拡張が自動的に追加されます。
- 証明書チェーンの最大深度は 5 個の証明書です。
- 証明書チェーンの最大サイズは 16 KB です。

- サポートされる暗号化アルゴリズムとキーサイズには、RSA 2048 ビット (RSA_2048) と ECDSA 256 ビット (EC_prime256v1) が含まれます。

複数のドメインに対する 1 つの証明書の使用

1 つの証明書を使用して複数のサブドメインをカバーする場合は、共通名 (CN) フィールドまたは主体者別名 (SAN) フィールドにワイルドカードドメインを使用します。たとえば、**`*.iot.example.com`** を使用して、dev.iot.example.com、qa.iot.example.com、および prod.iot.example.com をカバーします。各 FQDN には独自のドメイン設定が必要ですが、複数のドメイン設定では、同じワイルドカード値が使用できます。CN または SAN は、カスタムドメインとして使用する FQDN をカバーする必要があります。SAN が存在する場合、CN は無視されます。SAN は、カスタムドメインとして使用する FQDN をカバーする必要があります。このカバレッジは、完全一致またはワイルドカード一致です。ワイルドカード証明書が検証され、アカウントに登録されると、そのリージョンの他のアカウントは、証明書と重複するカスタムドメインを作成できなくなります。

次のセクションでは、各種類の証明書を取得する方法について説明します。すべての証明書リソースには、ドメイン設定の作成時に使用する ACM に登録された Amazon リソースネーム (ARN) が必要です。

ACM によって生成されたパブリック証明書

RequestCertificate API を使用して、カスタムドメインのパブリック証明書を生成できます。この方法で証明書を生成すると、ACM はカスタムドメインの所有権を検証します。詳細については、AWS Certificate Manager ユーザーガイドの「パブリック証明書のリクエスト」を参照してください。

公開 CA によって署名された外部証明書

パブリック CA (Mozilla の信頼できる ca-bundle に含まれる CA) によって署名されたサーバー証明書が既にある場合は、ImportCertificate API を使用して証明書チェーンを ACM に直接インポートできます。このタスク、前提条件および証明書形式の要件の詳細については、「証明書のインポート」を参照してください。

プライベート CA によって署名された外部証明書

プライベート CA によって署名されたサーバー証明書または自己署名されたサーバー証明書が既にある場合は、その証明書を使用してドメイン構成を作成できますが、ドメインの所有権を検証するために ACM に追加のパブリック証明書を作成する必要もあります。これを行うには、ImportCertificate API を使用してサーバー証明書チェーンを ACM に登録します。このタスク、前提条件および証明書形式の要件の詳細については、「証明書のインポート」を参照してください。

検証証明書の作成

証明書を ACM にインポートした後で、RequestCertificate API を使用してカスタムドメインのパブリック証明書を生成します。この方法で証明書を生成すると、ACM はカスタムドメインの所有権を検証します。詳細については、「パブリック証明書のリクエスト」を参照してください。ドメイン設定を作成するときは、このパブリック証明書を検証証明書として使用します。

ドメイン設定の作成

CreateDomainConfiguration API を使用して、カスタムドメインで設定可能なエンドポイントを作成します。カスタムドメインのドメイン設定は、次のもので構成されます。

- `domainConfigurationName`

  ドメイン設定を識別するユーザー定義名 `IoT:` で始まるドメイン設定名は、デフォルトのエンドポイント用に予約されており、使用できません。また、この値は に固有である必要があります AWS リージョン。

- `domainName`

  デバイスが接続に使用する FQDN AWS IoT Core。 AWS IoT Core は、サーバー名表示 (SNI) TLS 拡張機能を活用してドメイン設定を適用します。デバイスは、接続時にこの拡張機能を使用し、ドメイン設定で指定されているドメイン名と同じサーバー名を渡す必要があります。

- `serverCertificateArns`

  ACM. に登録したサーバー証明書チェーンの ARN は、 AWS IoT Core 現在 1 つのサーバー証明書のみをサポートしています。

- `validationCertificateArn`

  カスタムドメインの所有権を検証するために ACM で生成したパブリック証明書の ARN。パブリックに署名されたサーバー証明書または ACM によって生成されたサーバー証明書を使用する場合、この引数は不要です。

- `defaultAuthorizerName (optional)`

  エンドポイントで使用するカスタムオーソライザーの名前。

- `allowAuthorizerOverride`

  リクエストの HTTP ヘッダーで別のオーソライザーを指定することによって、デバイスがデフォルトのオーソライザーを上書きできるかどうかを指定するブール値。`defaultAuthorizerName` の値が指定されている場合、この値は必須です。

- `serviceType`

  AWS IoT Core は現在、 DATAサービスタイプのみをサポートしています。を指定するとDATA、はエンドポイントタイプが のエンドポイント AWS IoT を返しますiot:Data-ATS。

- `TlsConfig` (オプション)

  ドメインの TLS 構成を指定するオブジェクト。詳細については、「???」を参照してください。

- `serverCertificateConfig` (オプション)

  ドメインのサーバー証明書設定を指定するオブジェクト。詳細については、「???」を参照してください。

次の AWS CLI コマンドは、iot.example.com のドメイン設定を作成します。

```
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**
>
> ドメイン設定を作成した後、 がカスタムサーバー証明書 AWS IoT Core を提供するまでに最大 60 分かかる場合があります。

詳細については、「???」を参照してください。

DNS レコードの作成

サーバー証明書チェーンを登録してドメイン設定を作成したら、カスタムドメインが AWS IoT ドメインを指すように DNS レコードを作成します。このレコードは、タイプ の AWS IoT エンドポイントを指す必要がありますiot:Data-ATS。エンドポイントを取得するには、DescribeEndpoint API を使用します。

次の AWS CLI コマンドは、エンドポイントを取得する方法を示しています。

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

iot:Data-ATS エンドポイントを取得したら、カスタムドメインからこの AWS IoT エンドポイントにCNAMEレコードを作成します。同じ に複数のカスタムドメインを作成する場合 AWS アカウント、 はそれらを同じiot:Data-ATSエンドポイントにエイリアスします。

トラブルシューティング

デバイスをカスタムドメインに接続できない場合は、 AWS IoT Core がサーバー証明書を受け入れて適用していることを確認します。 AWS IoT Core コンソールまたは を使用して、 AWS IoT Core が証明書を受け入れたことを確認できます AWS CLI。

AWS IoT Core コンソールを使用するには、ドメイン設定ページに移動し、ドメイン設定名を選択します。[Server certificate details] (サーバー証明書の詳細) セクションで、ステータスとステータスの詳細を確認します。証明書が無効な場合は、ACM で、前のセクションでリストされている証明書要件を満たす証明書に置き換えます。証明書の ARN が同じ場合、 AWS IoT Core によって取得され、自動的に適用されます。

を使用して証明書のステータスを確認するには AWS CLI、DescribeDomainConfiguration API を呼び出し、ドメイン名を指定します。

> **Note**
>
> 証明書が無効である場合、 AWS IoT Core は引き続き最後の有効な証明書を提供します。

次の openssl コマンドを使用して、エンドポイントで提供されている証明書を確認できます。

```
openssl s_client -connect custom-domain-name:8883 -showcerts -servername
custom-domain-name
```

ドメイン設定の管理

このトピックでは、ドメイン設定リソースを管理するための主要な操作について説明します。また、ListDomainConfigurations、DescribeDomainConfiguration、UpdateDomainConfiguration DeleteDomainConfiguration の API を使用して、既存の設定のライフサイクルを管理することもできます。

このトピックの内容

- ドメイン設定の表示
- ドメイン設定の更新

- ドメイン設定の削除
- カスタムドメインでの証明書のローテーション

ドメイン設定の表示

内のすべてのドメイン設定のページ分割されたリストを返すには AWS アカウント、ListDomainConfigurations API を使用します。DescribeDomainConfiguration API を使用して、特定のドメイン設定の詳細を表示できます。この API は 1 つの domainConfigurationNameパラメータを受け取り、指定した設定の詳細を返します。

例

ドメイン設定の更新

ドメイン設定のステータスまたはカスタムオーソライザーを更新するには、UpdateDomainConfigurationAPI を使用します。このステータスをENABLED またはDISABLED に対して設定できます。ドメイン設定を無効にすると、そのドメインに接続されているデバイスに認証エラーが表示されます。現在、ドメイン設定でサーバー証明書を更新することはできません。ドメイン設定の証明書を変更するには、証明書を削除して再作成する必要があります。

例

ドメイン設定の削除

ドメイン設定を削除する前に、UpdateDomainConfiguration API を使用して、ステータスを DISABLED に設定します。これにより、エンドポイントが誤って削除されるのを防ぐことができます。ドメイン設定を無効にした後、DeleteDomainConfiguration API を使用してドメイン設定を削除します。削除する前に AWS、マネージドドメインを 7 日間 DISABLEDステータスにする必要があります。カスタムドメインを DISABLED ステータスにして、すぐにそれらを削除することができます。

例

ドメイン設定を削除すると、 はそのカスタムドメインに関連付けられたサーバー証明書を提供し AWS IoT Core なくなります。

カスタムドメインでの証明書のローテーション

お使いのサーバー証明書を更新した証明書に定期的に更新する必要がある場合がございます。これを行う頻度は、証明書の有効期間によって異なります。 AWS Certificate Manager (ACM) を使用し

てサーバー証明書を生成した場合は、証明書が自動的に更新されるように設定できます。ACM が証明書を更新すると、 は新しい証明書 AWS IoT Core を自動的に取得します。追加のアクションを実行する必要はありません。別のソースからサーバー証明書をインポートした場合は、ACM に再インポートすることでサーバー証明書をローテーションできます。証明書の再インポートについては、証明書の再インポートを参照してください。

> **Note**
>
> AWS IoT Core は、次の条件下でのみ証明書の更新を取得します。
>
> - 新しい証明書には、古い証明書と同じ ARN があります。
> - 新しい証明書には、古い証明書と同じ署名アルゴリズム、共通名、またはサブジェクトの別名があります。

ドメイン設定での TLS 設定の設定

AWS IoT Core には、ドメイン設定で TLS 1.2 および TLS 1.3 の Transport Layer Security (TLS) 設定をカスタマイズするための事前定義されたセキュリティポリシーが用意されています。セキュリティポリシーとは、クライアントとサーバー間の TLS ネゴシエーション中にサポートされるプロトコルと暗号を決定する組み合わせのことです。サポートされているセキュリティポリシーを使用すると、デバイスの TLS 設定をより柔軟に管理し、新しいデバイスを接続するときに最新のセキュリティ対策を適用し、既存のデバイスの一貫した TLS 設定を維持できます。

次の表では、セキュリティポリシーとその TLS バージョン、およびサポートされているリージョンについて説明します。

セキュリティポリシー名	サポートされている AWS リージョン
IoTSecurityPolicy_TLS13_1_3_2022_10	すべて AWS リージョン
IoTSecurityPolicy_TLS13_1_2_2022_10	すべて AWS リージョン
IoTSecurityPolicy_TLS12_1_2_2022_10	すべて AWS リージョン

セキュリティポリシー名	サポートされている AWS リージョン
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

のセキュリティポリシーの名前には、リリースされた年月に基づくバージョン情報 AWS IoT Core が含まれます。新しいドメイン設定を作成すると、セキュリティポリシーのデフォルトは `IoTSecurityPolicy_TLS13_1_2_2022_10` になります。プロトコル、TCP ポート、暗号の詳細を含むセキュリティポリシーの完全な表については、「セキュリティポリシー」を参照してください。カスタムセキュリティポリシーはサポートされていません。 AWS IoT Core 詳細については、「???」を参照してください。

ドメイン設定で TLS 設定を構成するには、 AWS IoT コンソールまたは を使用できます AWS CLI。

内容

- ドメイン設定での TLS 設定の設定 (コンソール)
- ドメイン設定での TLS 設定の設定 (CLI)

ドメイン設定での TLS 設定の設定 (コンソール)

AWS IoT コンソールを使用して TLS 設定を構成するには

1. にサインイン AWS マネジメントコンソール し、 AWS IoT コンソールを開きます。
2. 新しいドメイン設定を作成するときに TLS 設定を設定するには、次の手順に従います。
    1. 左側のナビゲーションペインで、ドメイン設定を選択し、ドメイン設定の作成を選択します。
    2. [ドメイン設定を作成] ページの [カスタムドメイン設定 - オプション] セクションで、[セキュリティポリシーを選択] からセキュリティポリシーを選択します。
    3. ウィジェットに従って、残りの設定ステップを完了します。[ドメイン設定を作成] を選択します。

3. 既存のドメイン設定の TLS 設定を更新するには、次の手順に従います。

   1. 左側のナビゲーションペインで、ドメイン設定を選択し、ドメイン設定を選択します。
   2. [ドメイン設定の詳細] ページで、[編集] を選択します。次に、[カスタムドメイン設定 - オプション] セクションの [セキュリティポリシーを選択] で、セキュリティポリシーを選択します。
   3. [ドメイン設定を更新] を選択します。

詳細については、「ドメイン設定の作成」と「ドメイン設定の管理」を参照してください。

ドメイン設定での TLS 設定の設定 (CLI)

create-domain-configuration および update-domain-configuration CLI コマンドを使用して、ドメイン設定の TLS 設定を設定できます。

1. create-domain-configuration CLI コマンドを使用して TLS 設定を指定するには。

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

   このコマンドの出力は以下のようになります。

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

   セキュリティポリシーを指定せずに新しいドメイン設定を作成した場合、値はデフォルトで次のようになります。IoTSecurityPolicy_TLS13_1_2_2022_10

2. describe-domain-configuration CLI コマンドを使用して TLS 設定を記述するには。

```
aws iot describe-domain-configuration \
    --domain-configuration-name domainConfigurationName
```

   このコマンドは、次のような TLS 設定を含むドメイン設定の詳細を返すことができます。

```
{
```

```
 "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 コマンドを使用して TLS 設定を更新するには。

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

このコマンドの出力は以下のようになります。

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

4. ATS エンドポイントの TLS 設定を更新するには、update-domain-configuration CLI コマンドを実行します。ATS エンドポイントのドメイン設定名は `iot:Data-ATS` です。

```
aws iot update-domain-configuration \
    --domain-configuration-name "iot:Data-ATS" \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

コマンドの出力は以下のようになります。

```
{
"domainConfigurationName": "iot:Data-ATS",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
iot:Data-ATS"
```

```
}
```

詳細については、「AWS API リファレンス」の「CreateDomainConfiguration」と「UpdateDomainConfiguration」を参照してください。

OCSP ステープリングのサーバー証明書設定

AWS IoT Core は、サーバー証明書のオンライン証明書ステータスプロトコル (OCSP) ステープリングをサポートします。これは、サーバー証明書の OCSP ステープリングまたは OCSP ステープリングとも呼ばれます。これは、Transport Layer Security (TLS) ハンドシェイクのサーバー証明書の失効ステータスを確認するために使用されるセキュリティメカニズムです。の OCSP ステープリング AWS IoT Core を使用すると、カスタムドメインのサーバー証明書の有効性に検証レイヤーを追加できます。

でサーバー証明書の OCSP ステープリングを有効に AWS IoT Core して、OCSP レスポンダーを定期的にクエリすることで、証明書の有効性を確認できます。OCSP ステープリング設定は、カスタムドメインを使用してドメイン設定を作成または更新するプロセスの一部です。OCSP ステープリングは、サーバー証明書の失効ステータスを継続的にチェックします。これにより、CA によって失効した証明書が、カスタムドメインに接続するクライアントによって信頼されなくなったことを確認できます。詳細については、「???」を参照してください。

サーバー証明書の OCSP ステープリングは、リアルタイムの失効ステータスチェックを提供し、失効ステータスの確認に関連するレイテンシーを減らし、安全な接続のプライバシーと信頼性を向上させます。OCSP ステープリングを使用するメリットの詳細については、「???」を参照してください。

> **Note**
>
> この機能は では使用できません AWS GovCloud (US) Regions。

このトピックの内容

- OCSF とは
- OCSP ステープリングの仕組み
- でサーバー証明書 OCSP を有効にする AWS IoT Core
- でのプライベートエンドポイントのサーバー証明書 OCSP の設定 AWS IoT Core

- でサーバー証明書 OCSP ステープリングを使用するための重要な注意事項 AWS IoT Core
- でのサーバー証明書 OCSP ステープリングのトラブルシューティング AWS IoT Core

OCSF とは

オンライン証明書ステータスプロトコル (OCSP) は、Transport Layer Security (TLS) ハンドシェイクのサーバー証明書の失効ステータスを提供するのに役立ちます。

主要なコンセプト

次の主要な概念は、オンライン証明書ステータスプロトコル (OCSP) の詳細を示しています。

OCSP

OCSP は、Transport Layer Security (TLS) ハンドシェイク中に証明書の失効ステータスをチェックするために使用されます。OCSP では、証明書をリアルタイムで検証できます。これにより、証明書が発行された後も失効または期限切れになっていないことが確認されます。また、OCSP は、従来の証明書失効リスト (CRL) よりもスケーラブルです。OCSP レスポンスは小さく、効率的に生成できるため、大規模なプライベートキーインフラストラクチャ (PKI) に適しています。

OCSP レスポンダー

OCSP レスポンダー (OCSP サーバーとも呼ばれます) は、証明書の失効ステータスを検証しようとするクライアントから OCSP リクエストを受信して応答します。

クライアント側の OCSP

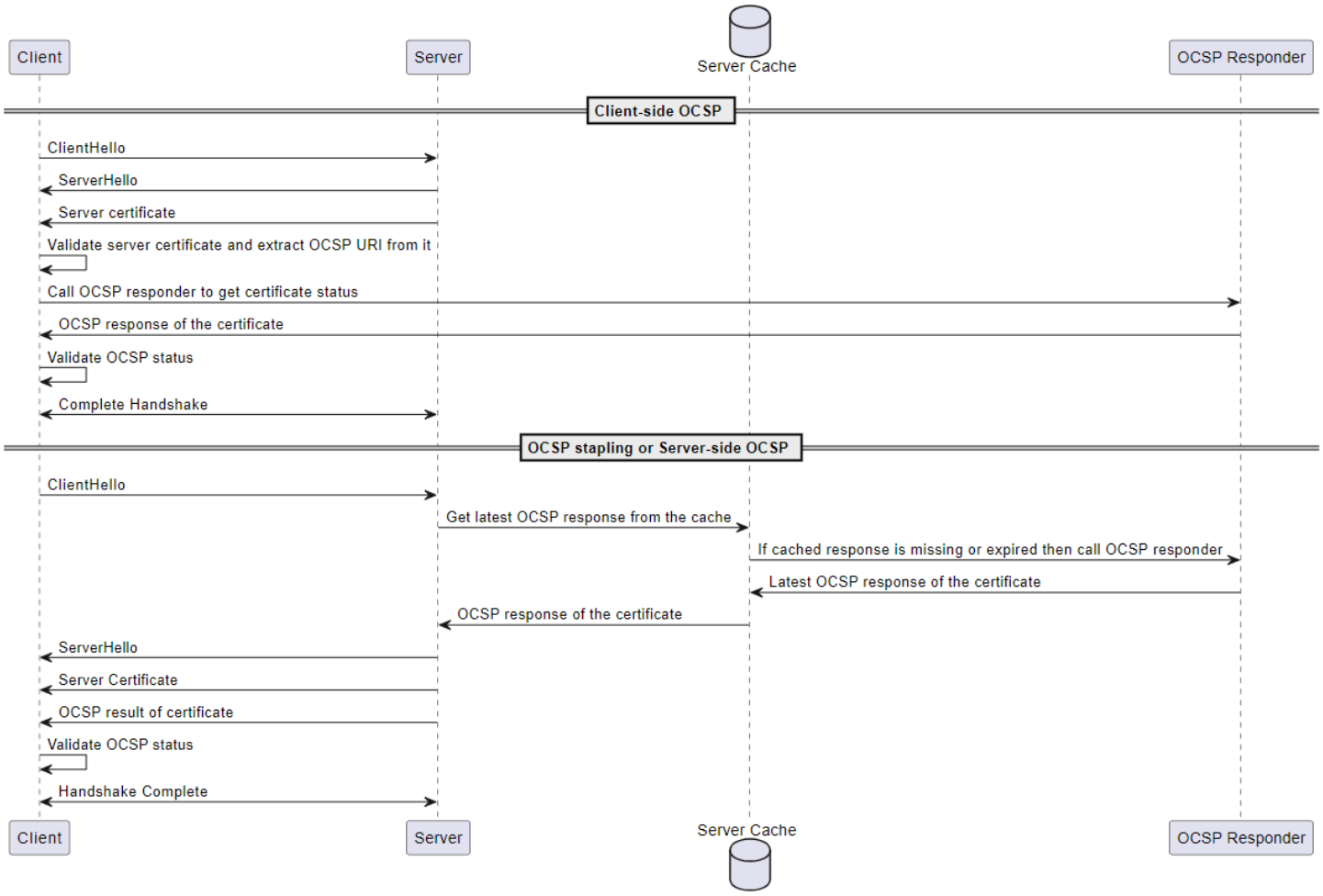
クライアント側の OCSP では、クライアントが OCSP を使用して OCSP レスポンダーに問い合わせを行い、TLS ハンドシェイク中に証明書の失効ステータスを確認します。

サーバー側の OCSP

サーバー側の OCSP (OCSP ステープリングとも呼ばれます) では、サーバーは (クライアントではなく) OCSP レスポンダーにリクエストを行うことができます。サーバーは証明書への OCSP レスポンスをステープルし、TLS ハンドシェイク中にクライアントに返します。

OCSP の図

次の図は、クライアント側の OCSP とサーバー側の OCSP の仕組みを示しています。



クライアント側の OCSP

1. クライアントは、サーバーで TLS ハンドシェイクを開始する `ClientHello` メッセージを送信します。
2. サーバーはメッセージを受信し、`ServerHello` メッセージで応答します。また、サーバーはサーバー証明書をクライアントに送信します。
3. クライアントはサーバー証明書を検証し、そこから OCSP URI を抽出します。
4. クライアントは証明書失効チェックリクエストを OCSP レスポンダーに送信します。
5. OCSP レスポンダーは OCSP レスポンスを送信します。
6. クライアントは OCSP レスポンスから証明書のステータスを検証します。
7. TLS ハンドシェイクが完了しました。

サーバー側の OCSP

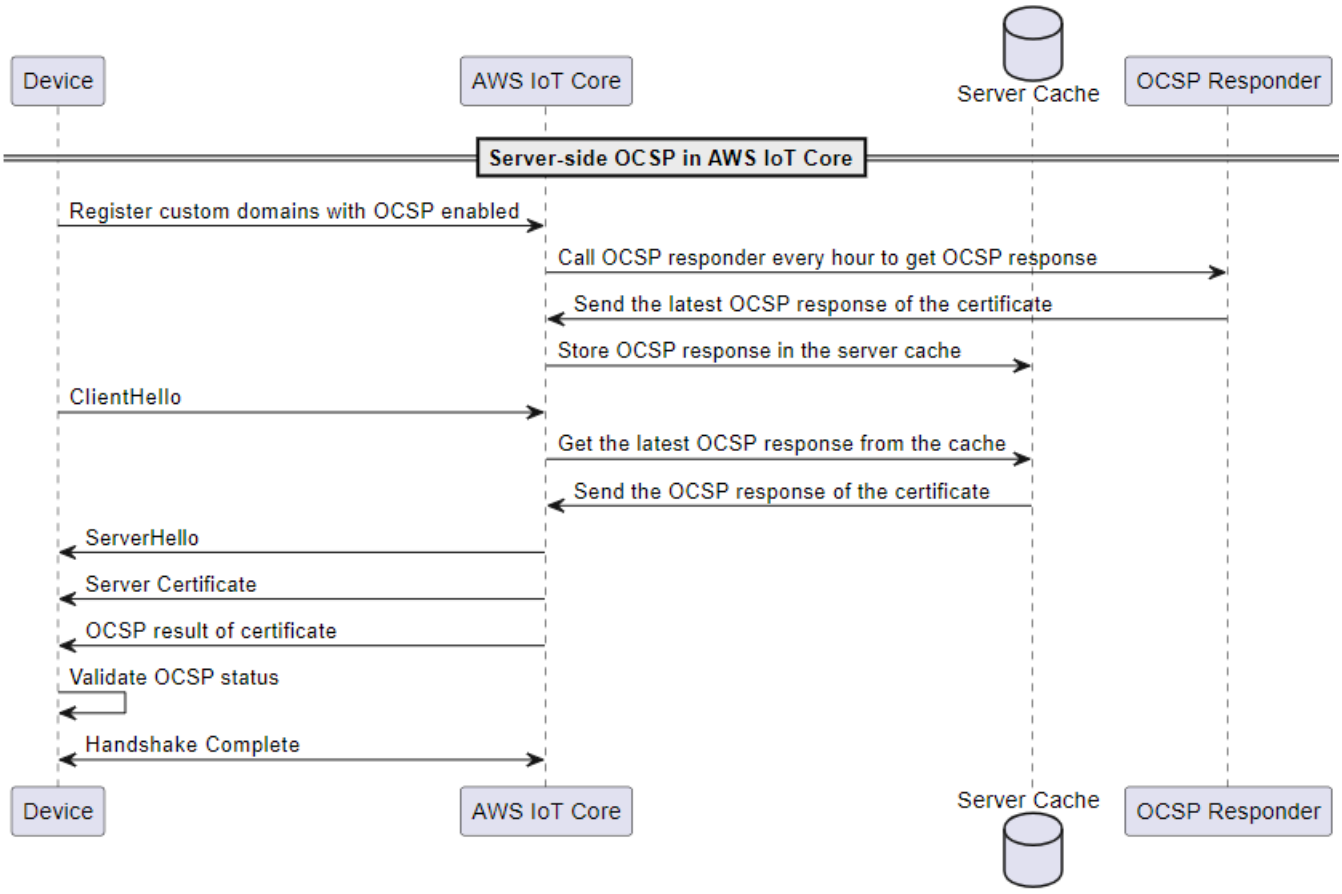
1. クライアントは、サーバーで TLS ハンドシェイクを開始する `ClientHello` メッセージを送信します。
2. サーバーはメッセージを受信し、最新のキャッシュされた OCSP レスポンスを取得します。キャッシュされたレスポンスが欠落しているか、または期限切れの場合、サーバーは証明書ステータスのために OCSP レスポンダーを呼び出します。
3. OCSP レスポンダーは OCSP レスポンスをサーバーに送信します。
4. サーバーは `ServerHello` メッセージを送信します。また、サーバーはサーバー証明書と証明書のステータスをクライアントに送信します。
5. クライアントは OCSP 証明書のステータスを検証します。
6. TLS ハンドシェイクが完了しました。

OCSP ステープリングの仕組み

OCSP Stapling は、クライアントとサーバー間の TLS ハンドシェイク中に使用され、サーバー証明書の失効ステータスをチェックします。サーバーは OCSP レスポンダーに OCSP リクエストを行い、クライアントに返された証明書への OCSP レスポンスをステープルします。サーバーが OCSP レスポンダーにリクエストを行うことで、レスポンスをキャッシュし、多くのクライアントに複数回使用できます。

での OCSP ステープリングの仕組み AWS IoT Core

次の図は、サーバー側の OCSP ステープリングが AWS IoT Coreでどのように機能するかを示しています。



1. デバイスは、OCSP ステープリングが有効になっているカスタムドメインに登録する必要があります。
2. AWS IoT Core は OCSP レスポンダーを 1 時間ごとに呼び出して、証明書のステータスを取得します。
3. OCSP レスポンダーはリクエストを受け取り、最新の OCSP レスポンスを送信し、キャッシュされた OCSP レスポンスを保存します。
4. デバイスは TLS ハンドシェイクを開始する`ClientHello`メッセージを送信します AWS IoT Core。
5. AWS IoT Core はサーバーキャッシュから最新の OCSP レスポンスを取得し、証明書の OCSP レスポンスで応答します。
6. サーバーはデバイスに `ServerHello` メッセージを送信します。また、サーバーはサーバー証明書と証明書のステータスをクライアントに送信します。
7. デバイスは OCSP 証明書のステータスを検証します。
8. TLS ハンドシェイクが完了しました。

クライアント側の OCSP チェックと比較して OCSP ステープリングを使用する利点

サーバー証明書の OCSP Stapling を使用する利点を、以下にいくつか挙げています。

プライバシーの向上

OCSP ステープリングを使用しないと、クライアントのデバイスがサードパーティーの OCSP レスポンダーに情報を公開し、ユーザーのプライバシーが侵害される可能性があります。OCSP ステープリングは、サーバーに OCSP レスポンスを取得させ、クライアントに直接配信させることで、この問題を軽減します。

信頼性の向上

OCSP ステープリングは、OCSP サーバーが停止するリスクを軽減するため、安全な接続の信頼性を向上させることができます。OCSP レスポンスがステープルされると、サーバーには証明書に最新のレスポンスが含まれます。これは、OCSP レスポンダーが一時的に使用できない場合でも、クライアントが失効ステータスにアクセスできるためです。OCSP Stapling は、サーバーが定期的に OCSP レスポンスを取得して、キャッシュされたレスポンスを TLS ハンドシェイクに含めるため、これらの問題の軽減に役立ちます。これにより、OCSP レスポンダーのリアルタイムの可用性への依存が抑えられます。

サーバー負荷の削減

OCSP ステープリングは、OCSP レスポンダーからサーバーへの OCSP リクエストに応答する負担を軽減します。これにより、負荷がより均等に分散され、証明書の検証プロセスの効率性が向上し、スケーラブルになります。

レイテンシーの短縮

OCSP ステープリングは、TLS ハンドシェイク中の証明書の失効ステータスの確認に関連するレイテンシーを減らします。クライアントが OCSP サーバーを個別にクエリする代わりに、サーバーはリクエストを送信し、ハンドシェイク中にサーバー証明書に OCSP レスポンスをアタッチします。

でサーバー証明書 OCSP を有効にする AWS IoT Core

サーバー証明書の OCSP ステープリングを有効にするには AWS IoT Core、カスタムドメインのドメイン設定を作成するか、既存のカスタムドメイン設定を更新します。カスタムドメインを使用してドメイン設定を作成する一般的な方法については、「???」を参照してください。

次の手順を使用して、 AWS マネジメントコンソール または を使用して OCSP サーバーのステープリングを有効にします AWS CLI。

コンソール

AWS IoT コンソールを使用してサーバー証明書の OCSP ステープリングを有効にするには:

1. ナビゲーションメニューから [設定] を選択し、[ドメイン設定を作成] を選択するか、またはカスタムドメインの既存のドメイン設定を選択します。
2. 前のステップで新しいドメイン設定の作成を選択した場合は、[ドメイン設定を作成] ページが表示されます。[ドメイン設定プロパティ] セクションで、[カスタムドメイン] を選択します。ドメイン設定を作成する情報を入力します。

   カスタムドメインの既存のドメイン設定を更新する場合は、[ドメイン設定の詳細] ページが表示されます。[編集] を選択します。
3. OCSP サーバーステープリングを有効にするには、[サーバー証明書の設定] サブセクションの [サーバー証明書の OCSP Stapling を有効にする] を選択します。
4. [ドメイン設定の作成] または [ドメイン設定の更新] を選択します。

AWS CLI

AWS CLIを使用してサーバー証明書の OCSP ステープリングを有効にするには

1. カスタムドメインの新しいドメイン設定を作成する場合、OCSP サーバーのステープリングを有効にするコマンドは次のようになります。

```
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. カスタムドメインの既存のドメイン設定を更新する場合、OCSP サーバーのステープリングを有効にするコマンドは次のようになります。

```
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"
```

詳細については、 AWS IoT API リファレンスのCreateDomainConfiguration」およびUpdateDomainConfiguration」を参照してください。

でのプライベートエンドポイントのサーバー証明書 OCSP の設定 AWS IoT Core

プライベートエンドポイント用の OCSP を使用すると、Amazon Virtual Private Cloud (Amazon VPC) 内のプライベート OCSP リソースを AWS IoT Core オペレーションに使用できます。このプロセスでは、OCSP レスポンダーとして機能する Lambda 関数を設定します。Lambda 関数は、プライベート OCSP リソースを使用して、 AWS IoT Core が使用する OCSP レスポンスを作成する場合があります。

Lambda function

プライベートエンドポイントにサーバー OCSP を設定する前に、基本的な OCSP レスポンスをサポートする Request for Comments (RFC) 6960 準拠のオンライン証明書ステータスプロトコル (OCSP) レスポンダーとして機能する Lambda 関数を作成します。Lambda 関数は、Distinguished Encoding Rules (DER) 形式での OCSP リクエストの base64 エンコードを受け入れます。Lambda 関数のレスポンスもまた、DER 形式の base64 エンコードされた OCSP レスポンスです。レスポンスサイズは 4 キロバイト (KiB) を超えることはできません。Lambda 関数は、ドメイン設定 AWS リージョン と同じ AWS アカウント および にある必要があります。以下は、Lambda 関数の例を示しています。

Lambda 関数の例

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 関数を呼び出す AWS IoT ことを に許可する

Lambda OCSP レスポンダーを使用してドメイン設定を作成するプロセスでは、関数の作成後に Lambda 関数を呼び出す AWS IoT アクセス許可を付与する必要があります。アクセス許可を付与するには、add-permission CLI コマンドを使用できます。

を使用して Lambda 関数にアクセス許可を付与する AWS CLI

1. 値を挿入したら、次のコマンドを入力します。statement-id 値は一意でなければならないことに注意してください。*Id-1234* を正確な値に置き換えます。そうしないと、ResourceConflictException エラーが発生する可能性があります。

   ```
   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 ドメイン設定 ARN は次のパターンに従います。サービスで生成されるサフィックスは作成前にはわからないため、サフィックスを * に置き換える必要があります。ドメイン設定が作成され、正確な ARN がわかった時点で、アクセス許可を更新することができます。

   arn:*aws*:iot:*use-east-1:123456789012*:domainconfiguration/*domain-config-name/service-generated-suffix*

2. コマンドが成功すると、この例のようなアクセス許可ステートメントが返されます。次のセクションに進み、プライベートエンドポイントの OCSP Stapling を設定できます。

   ```
   {
       "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/*\"}}}"
   }
   ```

   コマンドが成功しない場合は、この例のようなエラーが返されます。続行する前に、エラーを確認して修正する必要があります。

```
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
```

プライベートエンドポイントのサーバー OCSP Stapling を設定する

コンソール

AWS IoT コンソールを使用してサーバー証明書の OCSP ステープリングを設定するには:

1. ナビゲーションメニューから [設定] を選択し、[ドメイン設定を作成] を選択するか、カスタムドメインの既存のドメイン設定を選択します。
2. 前のステップで新しいドメイン設定の作成を選択した場合は、[ドメイン設定を作成] ページが表示されます。[ドメイン設定プロパティ] セクションで、[カスタムドメイン] を選択します。ドメイン設定を作成する情報を入力します。

   カスタムドメインの既存のドメイン設定を更新する場合は、[ドメイン設定の詳細] ページが表示されます。[編集] を選択します。
3. OCSP サーバーステープリングを有効にするには、[サーバー証明書の設定] サブセクションの [サーバー証明書の OCSP Stapling を有効にする] を選択します。
4. [ドメイン設定の作成] または [ドメイン設定の更新] を選択します。

AWS CLI

AWS CLIを使用してサーバー証明書の OCSP Stapling を設定にするには:

1. カスタムドメインの新しいドメイン設定を作成する場合は、プライベートエンドポイントのサーバー証明書の OCSP を設定するコマンドは次のようになります。

```
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. カスタムドメインの既存のドメイン設定を更新する場合は、プライベートエンドポイントのサーバー証明書の OCSP を設定するコマンドは次のようになります。

```
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

これは、サーバーの OCSP Stapling チェックが有効になっているかどうかを示すブール値です。サーバー証明書の OCSP Stapling を有効にするには、この値が true である必要があります。

ocspAuthorizedResponderArn

これは、 (ACM) に保存されている X.509 証明書の Amazon リソースネーム AWS Certificate Manager (ARN) の文字列値です。指定した場合、この証明書 AWS IoT Core を使用して、受信した OCSP レスポンスの署名を検証します。指定しない場合、 AWS IoT Core は発行証明書を使用してレスポンスを検証します。証明書は、ドメイン設定 AWS リージョン と同じ AWS アカウント および にある必要があります。承認されたレスポンダー証明書を登録する方法の詳細については、「 AWS Certificate Managerに証明書をインポートする」を参照してください。

ocspLambdaArn

これは、基本的な OCSP レスポンスをサポートする Request for Comments (RFC) 6960 準拠 (OCSP) レスポンダーとして機能する Lambda 関数の Amazon リソースネーム (ARN) の文字列値です。Lambda 関数は、DER 形式を使用してエンコードされた OCSP リクエストの base64 エンコードを受け入れます。Lambda 関数のレスポンスもまた、DER 形式の base64 エンコードされた OCSP レスポンスです。レスポンスサイズは 4 キロバイト (KiB) を超えることはできません。Lambda 関数は、ドメイン設定 AWS リージョン と同じ AWS アカウント および にある必要があります。

詳細については、 AWS IoT API リファレンスのCreateDomainConfiguration」およびUpdateDomainConfiguration」を参照してください。

でサーバー証明書 OCSP ステープリングを使用するための重要な注意事項 AWS IoT Core

でサーバー証明書 OCSP を使用する場合は AWS IoT Core、次の点に注意してください。

1. AWS IoT Core は、パブリック IPv4 アドレス経由で到達可能な OCSP レスポンダーのみをサポートします。
2. の OCSP ステープリング機能は、承認されたレスポンダーをサポート AWS IoT Core していません。すべての OCSP レスポンスは、証明書に署名した CA によって署名される必要があり、CA はカスタムドメインの証明書チェーンの一部である必要があります。
3. の OCSP ステープリング機能は、自己署名証明書を使用して作成されたカスタムドメインをサポート AWS IoT Core していません。
4. AWS IoT Core は OCSP レスポンダーを 1 時間ごとに呼び出し、レスポンスをキャッシュします。応答者への呼び出しが失敗した場合、 AWS IoT Core は最新の有効な応答をステープルします。
5. `nextUpdateTime` が有効でなくなった場合、 AWS IoT Core はキャッシュからレスポンスを削除し、次に OCSP レスポンダーへの呼び出しが成功するまで TLS ハンドシェイクは OCSP レスポンスデータを含めません。これは、サーバーが OCSP レスポンダーから有効なレスポンスを取得する前に、キャッシュされたレスポンスの有効期限が切れた場合に発生する可能性があります。`nextUpdateTime` の値は、OCSP レスポンスがこの時点まで有効であることを示唆しています。`nextUpdateTime` の詳細については、「???」を参照してください。
6. は、OCSP レスポンスの受信に AWS IoT Core 失敗したり、期限切れであるため既存の OCSP レスポンスを削除したりすることがあります。このような状況が発生した場合、 AWS IoT Core は OCSP レスポンスなしでカスタムドメインによって提供されるサーバー証明書を引き続き使用します。
7. OCSP レスポンスのサイズは 4 KiB を超過できません。

でのサーバー証明書 OCSP ステープリングのトラブルシューティング AWS IoT Core

AWS IoT Core は、`RetrieveOCSPStapleData.Success`メトリクスと`RetrieveOCSPStapleData`ログエントリを CloudWatch に出力します。メトリクスとログエントリは、OCSP レスポンスの取得に関連する問題を検出するのに役立ちます。詳細については、「???」および「???」を参照してください。

FIPS AWS IoT エンドポイントに接続する

AWS IoT は、連邦情報処理規格 (FIPS) 140-2 をサポートするエンドポイントを提供します。FIPS 準拠のエンドポイントは、標準 AWS エンドポイントとは異なります。 FIPS 準拠の方法で AWS IoT を操作するには、F IPS準拠のクライアントと共に以下で説明するエンドポイントを使用する必要があります。 AWS IoT コンソールは FIPS に準拠していません。

以下のセクションでは、REST API、 SDK、または を使用して FIPS 準拠の AWS IoT エンドポイントにアクセスする方法について説明します AWS CLI。

トピック

- AWS IoT Core- コントロールプレーンエンドポイント
- AWS IoT Core– データプレーンエンドポイント
- AWS IoT Core- 認証情報プロバイダーエンドポイント
- AWS IoT Device Management– ジョブデータエンドポイント
- AWS IoT Device Management– Fleet Hub エンドポイント
- AWS IoT Device Management- Secure Tunneling API エンドポイント
- AWS IoT Device Management - マネージド統合エンドポイント

AWS IoT Core- コントロールプレーンエンドポイント

AWS IoT操作とその関連するCLI コマンドをサポートするFIPS の準拠AWS IoT Core-コントロールプレーンエンドポイントは、サービス別FIPSエンドポイントに記載されています。サービス別FIPS エンドポイントで、AWS IoT Core-コントロールプレーンサービスを検索し、お使いの AWS リージョン用のエンドポイントを調べてください。

AWS IoT オペレーションにアクセスするときに FIPS 準拠のエンドポイントを使用するには、 に適したエンドポイントで AWS SDK または REST API を使用します AWS リージョン。

aws iotCLI コマンド実行時に、FIPS 準拠のエンドポイントを使用するには、お使いの AWS リージョン 用に適切なエンドポイントを持った--endpointパラメータをコマンドに追加します。

AWS IoT Core– データプレーンエンドポイント

FIPS の準拠AWS IoT Core-データプレーンエンドポイントは、サービス別FIPS エンドポイントに記載されています。サービス別FIPS エンドポイントで、AWS IoT Core-データプレーンサービスを検索し、お使いの AWS リージョン用のエンドポイントを調べてください。

Device AWS IoT SDK を使用し、アカウントのデフォルトのAWS IoT Core - データ プレーンエンドポイントの代わりに SDK の接続関数にエンドポイントを提供することで、FIPS 準拠クライアント AWS リージョン で の FIPS 準拠エンドポイントを使用できます。接続関数は AWS IoT Device SDK に固有です。接続関数の例については、AWS IoT Device SDK for Python の接続関数を参照してください。

> **Note**
>
> AWS IoT は、FIPS 準拠 AWS アカウントの特定のAWS IoT Core - データ プレーンエンドポイントをサポートしていません。Server Name Indication (SNI) で AWS アカウント特定のエンドポイントを必要とするサービス機能は使用できません。FIPS 準拠の AWS IoT Core- データプレーンのエンドポイントは、マルチアカウント登録証明書、カスタムドメイン、カスタムオーソライザー、および設定可能なエンドポイント (サポートされている TLS ポリシーを含む) をサポートできません。

AWS IoT Core- 認証情報プロバイダーエンドポイント

FIPS の準拠AWS IoT Core- データプレーンエンドポイントは、[サービス別の FIPS エンドポイント] に記載されています。[サービス別の FIPS エンドポイント] で、AWS IoT Core- 認証情報プロバイダーサービスを検索し、お使いの AWS リージョンのエンドポイントを調べてください。

> **Note**
>
> AWS IoT は、FIPS 準拠 AWS アカウントの特定のAWS IoT Core 認証情報 プロバイダーエンドポイントをサポートしていません。Server Name Indication (SNI) で AWS アカウント特定のエンドポイントを必要とするサービス機能は使用できません。FIPS 準拠の AWS IoT Core- 認証情報プロバイダーのエンドポイントは、マルチアカウント登録証明書、カスタムドメイン、カスタムオーソライザー、および設定可能なエンドポイント (サポートされている TLS ポリシーを含む) をサポートできません。

AWS IoT Device Management– ジョブデータエンドポイント

FIPS の準拠AWS IoT Device Management-ジョブデータエンドポイントは、サービス別FIPS エンドポイントに記載されています。サービス別FIPS エンドポイントで、AWS IoT Device Management-ジョブデータサービスを検索し、お使いの AWS リージョン用にエンドポイントを調べてください。

aws iot-jobs-data CLI コマンドの実行時に、FIPS 準拠の AWS IoT Device Management - ジョブデータエンドポイントを使用するには、 AWS リージョン に対応する適切なエンドポイントを指定した --endpoint パラメータをコマンドに追加してください。このエンドポイントでは、REST API を使用することもできます。

`iot:Jobs` の代わりに `Data-ATS` を使用することをお勧めします。`iot:Data-ATS` はデュアルスタックエンドポイント (IPv4 と IPv6) をサポートしていますが、`iot:Jobs` は IPv4 のみをサポートしています。

Device AWS IoT SDK を使用し、アカウントのデフォルトのAWS IoT Device Management - ジョブ データエンドポイントの代わりに SDK の接続関数にエンドポイントを提供することで、FIPS 準拠クライアント AWS リージョン で の FIPS 準拠エンドポイントを使用できます。接続関数は、AWS IoT Device SDKに固有です。接続関数の例については、AWS IoT Device SDK for Python の接続関数を参照してください。

AWS IoT Device Management– Fleet Hub エンドポイント

AWS IoT Device Management Fleet Hub for Device Management CLI コマンドで使用する FIPS 準拠 - Fleet Hub AWS IoT エンドポイントは、FIPS Endpoints by Service に記載されています。https://docs.aws.amazon.com//cli/latest/reference/iotfleethub/index.html「サービス別FIPS エンドポイント」で、AWS IoT Device Management- Fleet Hubサービスを検索し、お使いの AWS リージョン向けのエンドポイントを調べてください。

aws iotfleethub CLI コマンドを実行するときに FIPS 準拠AWS IoT Device Management - Fleet Hub エンドポイントを使用するには、 に適切なエンドポイントを持つ --endpointパラメータ AWS リージョン を コマンドに追加します。このエンドポイントでは、REST API を使用することもできます。

AWS IoT Device Management- Secure Tunneling API エンドポイント

AWS IoT – セキュアトンネリング API、および対応する CLI コマンド向けの FIPS 準拠の AWS IoT Device Management- セキュアトンネリングエンドポイントは、「サービス別の FIPS エンドポイント」にリストされています。サービス別の FIPS エンドポイントで、AWS IoT Device Management– セキュアトンネリングサービスを検索し、 AWS リージョンのエンドポイントを調べてください。

aws iotsecuretunneling CLI コマンドの実行時に、FIPS 準拠の AWS IoT Device Management– セキュアトンネリングエンドポイントを使用するには、 AWS リージョン に対応する適切なエンドポイントを指定した --endpoint パラメータをコマンドに追加してください。このエンドポイントでは、REST API を使用することもできます。

AWS IoT Device Management - マネージド統合エンドポイント

マネージド統合オペレーションをサポートする FIPS 準拠のコントロールプレーンエンドポイントとその関連 AWS CLI コマンドは、FIPS Endpoints by Service に記載されています。「サービス別の FIPS エンドポイント」で、AWS IoT Device Management - マネージド統合サービスを見つけて、お使いの AWS リージョン用のエンドポイントを調べてください。

マネージド統合オペレーションにアクセスするときに FIPS 準拠のエンドポイントを使用するには、に適したエンドポイントで AWS SDK または REST API を使用します AWS リージョン。

マネージド統合 CLI コマンド実行時に FIPS 準拠のエンドポイントを使用するには、お使いの AWS リージョン に対応する適切なエンドポイントを指定したパラメータを --endpoint コマンドに追加してください。

を使用したデバイスの管理 AWS IoT

AWS IoT には、モノの管理に役立つレジストリが用意されています。モノは、特定のデバイスまたは論理エンティティの表現です。物理的なデバイスやセンサー (電球や壁のスイッチなど) は、モノとして扱うことができます。また、アプリケーションのインスタンスのような論理エンティティ、または接続しない AWS IoT が接続する他のデバイス (エンジンセンサーやコントロールパネルがある車など) に関連する物理エンティティでもかまいません。

モノに関する情報は、JSON データとして Registry に保存されます。モノの例を次に示します。

```
{
     "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

モノは、名前で識別します。モノには、属性を指定することもできます。属性とは、モノに関する情報 (シリアルナンバーやメーカー名) を格納するために使用する、名前と値のペアです。

一般的なデバイスのユースケースでは、デフォルトの MQTT クライアント ID としてモノの名前が使用されます。MQTT クライアント ID、証明書、またはシャドウ状態をモノのレジストリ名として使用するというマッピングは強制されませんが、レジストリと Device Shadow サービスの両方で、モノの名前を MQTT クライアント ID として使用することをお勧めします。こうすることで、デバイスの証明書モデルや Shadows の柔軟性を失うことなく、IoT 群の秩序や利便性を維持することができます。

AWS IoTにデバイスを接続するために、レジストリでモノを作成する必要はありません。レジストリにモノを追加すると、デバイスの管理や検索が容易になります。

モノのレジストリデータ (属性、モノのタイプ、グループメンバーシップを含む) をルールエンジンで動的に取得して、メッセージの処理とルーティングに使用することもできます。詳細については、「get_registry_data」を参照してください。

レジストリを使用したモノの管理

AWS IoT コンソール、 AWS IoT API、または AWS CLI を使用してレジストリを操作します。以下の各セクションでは、CLI を使用して Registry を操作する方法を示します。

モノのオブジェクトに名前を付ける場合:

- モノの名前で個人を特定できる情報を使用しないでください。モノの名前は、暗号化されていない通信やレポートに表示されることがあります。

トピック

- モノの作成
- モノのリスト表示
- モノを記述する
- モノの更新
- モノの削除
- モノにプリンシパルをアタッチする
- プリンシパルに関連付けられているモノを一覧表示する
- モノに関連付けられているプリンシパルを一覧表示する
- プリンシパルに関連付けられているモノを一覧表示する (V2)
- モノに関連付けられているプリンシパルを一覧表示する (V2)
- モノからプリンシパルをデタッチする

モノの作成

次のコマンドは、 CLI の コマンドを使用して AWS IoT CreateThingモノを作成する方法を示しています。モノの作成後に名前を変更することはできません。モノの名前を変更するには、新しいモノを作成して、新しい名前を付け、古いモノを削除します。

```
$ aws iot create-thing \
    --thing-type-name "MyLightBulb" \
    --attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

CreateThing コマンドを実行すると、新しいモノの名前と Amazon リソースネーム (ARN) が表示されます。

```
{
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678"
}
```

> **Note**
>
> モノの名前に個人を特定できる情報を使用することはお勧めしません。

詳細については、「 AWS CLI コマンドリファレンス」の「create-thing」を参照してください。

モノのリスト表示

ListThings コマンドを使用すると、アカウント内のモノをすべてリスト表示できます。

```
$ aws iot list-things
```

```
{
    "things": [
       {
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyLightBulb"
        },
        {
            "attributes": {
                "numOfStates":"3"
             },
            "version": 11,
            "thingName": "MyWallSwitch"
        }
    ]
}
```

ListThings コマンドを使用して、特定のモノタイプのすべてのものを検索できます。

```
$  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"
        }
    ]
}
```

ListThings コマンドを使用して、特定の属性値を持つすべてのモノを検索できます。このコマンドは、最大 3 つの属性を検索します。

```
$  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"
        }
    ]
}
```

詳細については、「 AWS CLI コマンドリファレンス」の「list-things」を参照してください。

モノを記述する

DescribeThing コマンドを使用して、モノに関するより詳細な情報を表示できます。

```
$ 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"
    }
}
```

インライン関数 を使用して、ルールエンジン内でこの API にアクセスすることもできますget_registry_data()。この関数を使用すると、 および DescribeThing ListThingGroupsForThing APIs を AWS IoT ルール内で直接呼び出すことで、モノのレジストリ情報 (属性、モノのタイプ、グループメンバーシップを含む) に動的にアクセスして利用でき、デバイスレジストリデータに基づいてリアルタイムのメッセージ処理とルーティングが可能になります。詳細については、「get_registry_data」を参照してください。

詳細については、 AWS CLI 「 コマンドリファレンス」の「describe-thing」を参照してください。

モノの更新

UpdateThing コマンドを使用すると、モノを更新できます。このコマンドはモノの属性だけを更新します。モノの名前を変更することはできません。モノの名前を変更するには、新しいモノを作成して、新しい名前を付け、古いモノを削除します。

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"150\", \"model\":\"456\"}}"
```

UpdateThing コマンドでは、出力が生成されません。DescribeThing コマンドを使用して、結果を表示できます。

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

詳細については、「 AWS CLI コマンドリファレンス」の「update-thing」を参照してください。

モノの削除

DeleteThing コマンドを使用すると、モノを削除できます。

```
$ aws iot delete-thing --thing-name "MyThing"
```

このコマンドは、削除が成功した場合、または存在しないモノが指定された場合、エラーなしで正常に終了します。

詳細については、「 AWS CLI コマンドリファレンス」の「delete-thing」を参照してください。

モノにプリンシパルをアタッチする

物理デバイスは、プリンシパルを使用して通信できます AWS IoT。プリンシパルには、X.509 証明書または Amazon Cognito ID を使用できます。証明書または Amazon Cognito ID をデバイスを表すレジストリ内のモノに関連付けるには、attach-thing-principal コマンドを実行します。

attach-thing-principal コマンドを使用して、証明書または Amazon Cognito ID をモノにアタッチする方法:

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

アタッチメントタイプ (排他的アタッチメント、または非排他的アタッチメント) を指定してモノに証明書をアタッチするには、attach-thing-principal コマンドを使用して、`--thing-principal-type` フィールドにタイプを指定します。排他的アタッチメントとは、IoT のモノが証明書にアタッチされている唯一のモノであり、他のモノにこの証明書を関連付けることはできません。非排他的アタッチメントとは、IoT のモノが証明書にアタッチされ、かつ、この証明書を他のモノにも関連付けることができることを意味します。詳細については、「???」を参照してください。

> **Note**
>
> ???機能では、プリンシパルとして使用できるのは X.509 証明書のみです。

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb2" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --thing-principal-type "EXCLUSIVE_THING"
```

アタッチメントが成功した場合は、AttachThingPrincipal コマンドは出力を生成しません。アタッチメントを記述するには、list-thing-principals-v2 CLI コマンドを使用します。

詳細については、「AWS IoT Core API リファレンス」の「AttachThingPrincipal」を参照してください。

プリンシパルに関連付けられているモノを一覧表示する

指定のプリンシパルに関連付けられているモノを一覧表示するには、list-principal-things コマンドを実行します。このコマンドでは、モノと証明書間のアタッチメントタイプは一覧に表示されない点に注意してください。アタッチメントタイプを一覧表示するには、list-principal-things-v2 コマンドを使用します。詳細については、「???」を参照してください。

```
$ aws iot list-principal-things \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

出力は次のようになります。

```
{
    "things": [
        "MyLightBulb1",
        "MyLightBulb2"
    ]
}
```

詳細については、「AWS IoT Core API リファレンス」の「ListPrincipalThings」を参照してください。

モノに関連付けられているプリンシパルを一覧表示する

指定のモノに関連付けられているプリンシパルを一覧表示するには、list-thing-principals コマンドを実行します。このコマンドでは、モノと証明書間のアタッチメントタイプは一覧に表示されない点に注意してください。アタッチメントタイプを一覧表示するには、list-thing-principals-v2 コマンドを使用します。詳細については、「???」を参照してください。

```
$ aws iot list-thing-principals \
    --thing-name "MyLightBulb1"
```

出力は次のようになります。

```
{
    "principals": [
        "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8",
        "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
    ]
}
```

詳細については、「AWS IoT Core API リファレンス」の「ListThingPrincipals」を参照してください。

プリンシパルに関連付けられているモノを一覧表示する (V2)

指定の証明書に関連付けられているモノをアタッチメントタイプとともに一覧表示するには、list-principal-things-v2 コマンドを実行します。アタッチメントタイプは、証明書がモノにアタッチされる方式を指します。

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

出力は次のようになります。

```
{
    "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"
        }

    ]
```

```
}
```

詳細については、「AWS IoT Core API リファレンス」の「ListPrincipalThingsV2」を参照してください。

モノに関連付けられているプリンシパルを一覧表示する (V2)

指定のモノに関連付けられている証明書をアタッチメントタイプとともに一覧表示するには、list-thing-principals-V2 コマンドを実行します。アタッチメントタイプは、証明書がモノにアタッチされる方式を指します。

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

出力は次のようになります。

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

詳細については、「AWS IoT Core API リファレンス」の「ListThingsPrincipalV2」を参照してください。

モノからプリンシパルをデタッチする

DetachThingPrincipal コマンドを使用すると、モノから証明書をデタッチできます。

```
$ aws iot detach-thing-principal \
```

```
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

DetachThingPrincipal コマンドでは、出力が生成されません。

詳細については、「AWS IoT Core API リファレンス」の「detach-thing-principal」を参照してください。

モノのタイプ

モノのタイプを使用すると、同じタイプに関連付けられているすべてのモノに共通した説明および設定情報を格納できます。これにより、Registry でのモノの管理が単純化されます。たとえば、LightBulb というタイプを定義することができます。LightBulb というタイプに関連付けられているすべてのモノは、同じ属性 (シリアルナンバー、メーカー名、ワット数) を使用します。LightBulb というタイプのモノを作成 (または、既存のモノのタイプを LightBulb に変更) する際に、LightBulb タイプで定義されている各属性の値を指定できます。

モノのタイプはオプションですが、使用すると、モノを検出しやすくなります。

- モノのタイプが関連付けられたモノの場合は、最大 50 個の属性を指定できます。
- モノのタイプが関連付けられていないモノの場合は、最大 3 個の属性を指定できます。
- 1 つのモノを関連付けることができるモノのタイプは 1 つだけです。
- アカウント内で作成できるモノのタイプの数に制限はありません。

モノのタイプを作成した後に名前を変更することはできません。特定のタイプに新しいモノが関連付けられないようにするには、いつでもそのタイプを非推奨にできます。また、モノが 1 つも関連付けられていないタイプは、削除できます。

トピック:

- モノのタイプを作成する
- モノのタイプをリスト表示する
- モノのタイプを記述する
- モノのタイプをモノに関連付ける
- モノのタイプを更新する

- モノのタイプを非推奨にする
- モノのタイプを削除する

モノのタイプを作成する

CreateThingType コマンドを使用すると、モノのタイプを作成できます。

```
$ aws iot create-thing-type

                --thing-type-name "LightBulb" --thing-type-properties
 "thingTypeDescription=light bulb type, searchableAttributes=wattage,model"
```

CreateThingType コマンドは、モノのタイプおよび ARN を含む応答を返します。

```
{
    "thingTypeName": "LightBulb",
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb"
}
```

モノのタイプをリスト表示する

ListThingTypes コマンドを使用すると、モノのタイプをリスト表示できます。

```
$ aws iot list-thing-types
```

ListThingTypes コマンドは、 で定義されているモノのタイプのリストを返します AWS アカウント。

```
{
    "thingTypes": [
        {
            "thingTypeName": "LightBulb",
            "thingTypeProperties": {
                "searchableAttributes": [
                    "wattage",
                    "model"
                ],
                "thingTypeDescription": "light bulb type"
```

```
            },
            "thingTypeMetadata": {
                "deprecated": false,
                "creationDate": 1468423800950
            }
        }
    ]
}
```

モノのタイプを記述する

DescribeThingType コマンドを使用すると、モノのタイプに関する情報を取得できます。

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

DescribeThingType コマンドは、指定されたタイプに関する情報を返します。

```
{
    "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
    }
}
```

モノのタイプをモノに関連付ける

CreateThing コマンドを使用すると、モノを作成する際にタイプを指定できます。

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

UpdateThing コマンドを使用すると、モノに関連付けるモノのタイプをいつでも変更できます。

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

UpdateThing コマンドを使用すると、モノのタイプとモノとの関連付けを解除することもできます。

モノのタイプを更新する

UpdateThingType コマンドを使用すると、モノを作成する際にタイプを更新することができます。

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

UpdateThing コマンドを使用すると、モノに関連付けるモノのタイプをいつでも変更できます。

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

UpdateThing コマンドを使用すると、モノのタイプとモノとの関連付けを解除することもできます。

モノのタイプを非推奨にする

モノのタイプは変更不可能です。定義した後に変更することはできません。ただし、特定のタイプにユーザーが新しいモノを関連付けないようにするには、そのタイプを非推奨にすることができます。そのタイプに関連付けられている既存のモノはすべて、変更されません。

モノのタイプを非推奨にするには、DeprecateThingType コマンドを使用します。

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

DescribeThingType コマンドを使用して、結果を表示できます。

```
$ 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
    }
}
```

モノのタイプの廃止は、操作を元に戻すことができます。--undo-deprecate フラッグを DeprecateThingType CLI コマンドと共に使用して、廃止を取り消すことができます。

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

DescribeThingType CLI コマンドを使用して、結果を表示できます。

```
$ 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,
    }
}
```

モノのタイプを削除する

モノのタイプを削除するには、あらかじめそのタイプを非推奨にしておく必要があります。モノのタイプを削除にするには、DeleteThingType コマンドを使用します。

```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

> **Note**
>
> モノのタイプを削除する前に、廃止にしてから 5 分ほど待つ必要があります。

モノの静的グループ

モノの静的グループはこれらをグループに分類することで、複数のモノを一度に管理できるようにします。モノの静的グループには、コンソール、CLI、または API を使用して管理されるモノのグループが含まれます。一方、動的なモノのグループには、指定したクエリに一致するモノが含まれます。モノの静的グループには、他のモノの静的グループを含めることもできます。グループの階層を構築できます。親グループにポリシーを付加することができ、その子グループ、およびそのグループと子グループ内のすべてのモノに継承されます。これにより、多数のモノに対するアクセス許可の制御が容易になります。

> **Note**
>
> モノのグループポリシーは、 AWS IoT Greengrass データプレーンオペレーションへのアクセスを許可しません。 AWS IoT Greengrass データプレーンオペレーションへのモノのアクセスを許可するには、モノの証明書にアタッチする AWS IoT ポリシーに アクセス許可を追加します。詳細については、AWS IoT Greengrass 開発者ガイドの「デバイスの認証と承認」を参照してください。

次に、モノの静的グループで実施可能な操作をご紹介します。

- グループを作成、説明、または削除する。
- モノをグループに追加するか、複数のグループに追加する。
- グループからモノを削除する。
- 作成したグループを一覧表示する。

- グループのすべての子のグループを一覧表示する (直接の子孫と間接の子孫)。
- 子グループ内のすべてのものを含め、グループ内のものを一覧表示する。
- グループのすべての先祖のグループを一覧表示する (直接の先祖と間接の先祖)。
- グループの属性を追加、削除または更新します。(属性は、グループに関する情報を格納するのに使用できる名前と値のペアです。)
- グループにポリシーをアタッチまたはデタッチする。
- グループにアタッチされるポリシーを一覧表示する。
- モノによって継承されたポリシーを一覧表示する (そのグループまたはその親グループの 1 つにアタッチされたポリシーによって)。
- グループ内のモノのログ記録オプションを設定する。「」を参照してくださいAWS IoT ログ記録の設定
- グループとその子グループのすべてのモノに送信され実行されるジョブを作成する。「AWS IoT ジョブ」を参照してください。

> **Note**
>
> AWS IoT Core ポリシーがアタッチされているモノの静的グループにモノがアタッチされている場合、モノの名前はクライアント ID と一致する必要があります。

モノの静的グループのいくつかの制限があります。

- グループは、最大 1 つの直接の親を持つことができます。
- グループが別のグループの子である場合は、作成時に指定します。
- グループの親は後から変更できません。そのため、グループ階層は入念に計画し、親グループを作成してから、その親グループに含む子グループを作成してください
- モノが属することができるグループの最大数は、限られています。
- 1 つのモノを同じ階層の複数のグループに追加することはできません。(つまり、共通の親を共有する 2 つのグループにモノを追加することはできません)。
- グループ名を変更することはできません。
- モノのグループ名には、û、é、ñ などの国際文字を含めることはできません。
- モノのグループ名で個人を特定できる情報を使用しないでください。モノのグループ名は、暗号化されていない通信やレポートに表示される可能性があります。

グループにポリシーをアタッチおよびデタッチすると、 AWS IoT オペレーションのセキュリティをいくつかの重要な方法で強化できます。ポリシーに証明書をアタッチしてからモノにアタッチするような、デバイスごとの方法は時間がかかり、多数のデバイスにわたってポリシーを迅速に更新または変更することが困難です。モノのグループにポリシーをアタッチすると、証明書をモノに回す際のステップが節約されます。また、ポリシーはグループメンバーシップを変更すると動的に適用されるため、デバイスがグループのメンバーシップを変更するたびに複雑なアクセス許可セットを再作成する必要はありません。

モノの静的グループの作成

モノの静的グループを作成するには CreateThingGroup コマンドを使用します。

```
$ aws iot create-thing-group --thing-group-name LightBulbs
```

CreateThingGroup コマンドは、モノの静的グループの名前、ID、および ARN を含むレスポンスを返します。

```
{
    "thingGroupName": "LightBulbs",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
}
```

> **Note**
>
> モノのグループ名に個人を特定できる情報を使用することはお勧めしません。

作成されたときのモノのグループの親を指定している例を次に示します。

```
$ aws iot create-thing-group --thing-group-name RedLights --parent-group-name
 LightBulbs
```

前と同様に、CreateThingGroup コマンドは、モノの静的グループの名前、ID、および ARN を含むレスポンスを返します。

```
{
    "thingGroupName": "RedLights",
```

```
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
}
```

Important

モノのグループ階層を作成するときは、次の制限事項に留意してください。

- モノのグループは、直接の親を 1 つだけ持つことができます。
- モノのグループが持つことができる直接の子グループの数は、限られています。
- グループの階層の最大深度は限られています。
- モノのグループが持つことのできる属性の数は、限られています。(属性は、グループに関する情報を格納するのに使用できる名前と値のペアです。) 各属性名と各値の長さも限られています。

モノのグループの説明

DescribeThingGroup コマンドを使用すると、モノのグループに関する情報を取得できます。

```
$ aws iot describe-thing-group --thing-group-name RedLights
```

DescribeThingGroup コマンドは、指定されたグループに関する情報を返します。

```
{
    "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"
    },
}
```

モノの静的グループにモノを追加する

AddThingToThingGroup コマンドを使用して、モノの静的グループにモノを追加できます。

```
$ aws iot add-thing-to-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

AddThingToThingGroup コマンドでは、出力が生成されません。

> **Important**
>
> 最大 10 個のグループにモノを追加できます。ただし、1 つのモノを同じ階層の複数のグループに追加することはできません。(つまり、共通の親を共有する 2 つのグループにモノを追加することはできません。)
> 1 つのモノができる限り多くのモノのグループに属していて、それらのグループの 1 つ以上がモノの動的グループである場合、overrideDynamicGroups フラグを使用して、静的グループが動的グループより優先されるように指定できます。

モノの静的グループからモノを削除する

RemoveThingFromThingGroup コマンドを使用すると、グループからモノを削除できます。

```
$ aws iot remove-thing-from-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

RemoveThingFromThingGroup コマンドでは、出力が生成されません。

モノのグループ内のモノを一覧表示する

ListThingsInThingGroup コマンドを使用して、グループに属するモノを一覧表示できます。

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs
```

この ListThingsInThingGroup コマンドは、指定されたグループ内のモノのリストを返します。

```
{
    "things":[
        "TestThingA"
    ]
}
```

`--recursive` パラメータを使用すると、グループに属するモノとその子グループに属するモノを一覧表示することができます

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs --recursive
```

```
{
    "things":[
        "TestThingA",
        "MyLightBulb"
    ]
}
```

> **Note**
>
> このオペレーションは結果整合性があります。言い換えれば、モノのグループへの変更はすぐに反映されない可能性があります。

モノのグループの一覧表示

ListThingGroups コマンドを使用して、アカウントのモノのグループを一覧表示できます。

```
$ aws iot list-thing-groups
```

ListThingGroups コマンドは、 内のモノのグループのリストを返します AWS アカウント。

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

オプションのフィルタを使用して、指定された接頭辞 (--parent-group) で始まる名前を持つ、特定のグループを親 (--name-prefix-filter) またはグループとして持つグループを一覧表示します。--recursive パラメータを指定すると、モノのグループの直接の子グループだけでなく、すべての子グループを一覧表示できます。

```
$ aws iot list-thing-groups --parent-group LightBulbs
```

この場合、 ListThingGroups コマンドは、 で定義されたモノのグループの直接の子グループのリストを返します AWS アカウント。

```
{
    "childGroups":[
        {
```

```
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        }
    ]
}
```

--recursive コマンドで ListThingGroups パラメータを使用すると、モノのグループの直接の子グループだけでなく、すべての子グループを一覧表示できます。

```
$ aws iot list-thing-groups --parent-group LightBulbs --recursive
```

この ListThingGroups コマンドは、を使用すると、モノのグループのすべての子グループのリストを返します。

```
{
    "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**
>
> このオペレーションは結果整合性があります。言い換えれば、モノのグループへの変更はすぐに反映されない可能性があります。

モノが属するグループを一覧表示する

ListThingGroupsForThing コマンドを使用して、モノが属する直接グループを一覧表示できます。

```
$ aws iot list-thing-groups-for-thing --thing-name MyLightBulb
```

ListThingGroupsForThing コマンドは、このモノが属する直接モノグループのリストを返します。

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

インライン関数 を使用して、ルールエンジン内でこの API にアクセスすることもできますget_registry_data()。この関数を使用すると、 および DescribeThing ListThingGroupsForThing APIs を AWS IoT ルール内で直接呼び出すことで、モノのレジストリ情報 (属性、モノのタイプ、グループメンバーシップを含む) に動的にアクセスして利用でき、デバイスレジストリデータに基づいてリアルタイムのメッセージ処理とルーティングが可能になります。詳細については、「get_registry_data」を参照してください。

モノの静的グループの更新

UpdateThingGroup コマンドを使用すると、モノの静的グループの属性を更新できます。

```
$ 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 コマンドは、更新後にグループのバージョン番号を含むレスポンスを返します。

```
{
```

```
    "version": 4
}
```

> **Note**
>
> モノが持つことができる属性の数は、限られています。

モノのグループを削除する

モノのグループを削除するには、DeleteThingGroup コマンドを使用します。

```
$ aws iot delete-thing-group --thing-group-name "RedLights"
```

DeleteThingGroup コマンドでは、出力が生成されません。

> **Important**
>
> 子グループを持つモノのグループを削除しようとすると、次のようなエラーが発生します。
>
> ```
> A client error (InvalidRequestException) occurred when calling the
>  DeleteThingGroup
> operation: Cannot delete thing group : RedLights when there are still child
>  groups attached to it.
> ```
>
> グループを削除する前に、すべての子グループを削除する必要があります。

子のモノを持つグループは削除できますが、そのグループのメンバーシップによって付与されているすべてのアクセス許可は適用されなくなります。ポリシーが設定されているグループを削除する前に、それらのアクセス許可を削除してもグループ内の機能が正しく機能しなくなることはありません。また、クラウド内のレコードが更新されている間は、モノが属しているグループを示すコマンド (ListGroupsForThing など) で、そのグループが表示され続けることがあります。

モノの静的グループにポリシーをアタッチする

AttachPolicy コマンドを使用して、モノの静的グループにポリシーをアタッチすることができます。そのため、そのグループ内のすべてのモノやその子グループのモノに拡張機能を割り当てることができます。

```
$ aws iot attach-policy \
  --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs" \
  --policy-name "myLightBulbPolicy"
```

AttachPolicy コマンドでは、出力が生成されません。

> **Important**
>
> 1 つのグループにアタッチできるポリシーは最大 2 つです。

> **Note**
>
> ポリシー名に個人を特定できる情報を使用することはお勧めしません。

`--target` パラメータは、モノのグループ ARN (上記に示す)、証明書の ARN または Amazon Cognito ID とすることができます。ポリシー、証明書、および認証の詳細については、「」を参照してください認証

詳細については、「AWS IoT Core ポリシー」を参照してください。

モノの静的グループからポリシーをデタッチする

DetachPolicy コマンドを使用して、モノのグループからポリシーをデタッチすることができます。そのため、そのグループ内のすべてのモノやその子グループのモノに拡張機能を割り当てることができます。

```
$ aws iot detach-policy --target "arn:aws:iot:us-west-2:123456789012:thinggroup/
LightBulbs" --policy-name "myLightBulbPolicy"
```

DetachPolicy コマンドでは、出力が生成されません。

モノの静的グループにアタッチされているポリシーを一覧表示する

ListAttachedPolicies コマンドを使用すると、モノの静的グループにアタッチされたポリシーを一覧表示できます。

```
$ aws iot list-attached-policies --target "arn:aws:iot:us-
west-2:123456789012:thinggroup/RedLights"
```

`--target` パラメータは、モノのグループ ARN (上記に示す)、証明書の ARN または Amazon Cognito ID とすることができます。

オプションの `--recursive` パラメータを追加して、グループの親グループにアタッチされたすべてのポリシーを含めます。

ListAttachedPolicies コマンドは、ポリシーのリストを返します。

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

ポリシーのグループを一覧表示する

ListTargetsForPolicy コマンドを使用して、ポリシーがアタッチされているすべてのグループを含むターゲットを一覧表示できます。

```
$ aws iot list-targets-for-policy --policy-name "MyLightBulbPolicy"
```

オプションの `--page-size` *`number`* パラメータを追加して返される結果の最大数を指定します。該当する場合は、各クエリの後続の呼び出しで `--marker` *`string`* パラメータを追加して次の結果セットを取得します。

ListTargetsForPolicy コマンドは、より多くの結果を取得するために使用するターゲットとトークンのリストを返します。

```
{
    "nextMarker": "string",
    "targets": [ "string" ... ]
}
```

モノの有効なポリシーを取得する

GetEffectivePolicies コマンドを使用して、(グループが直接の親であるか間接的な祖先であるかにかかわらず) 所属するグループにアタッチされたポリシーを含め、モノで有効なポリシーを一覧表示できます。

```
$ aws iot get-effective-policies \
  --thing-name "MyLightBulb" \
  --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

`--principal` パラメータを使用して、モノにアタッチされた証明書の ARN を指定します。Amazon Cognito ID 認証を使用している場合は、`--cognito-identity-pool-id` パラメータを使用し、オプションで `--principal` パラメータを追加して、特定の Amazon Cognito ID を指定します。`--cognito-identity-pool-id` のみを指定すると、認証されていないユーザーのアイデンティティプールのロールに関連付けられたポリシーが返されます。両方を使用すると、認証されたユーザーの ID プールのロールに関連付けられたポリシーが返されます。

この `--thing-name` パラメータはオプションで、`--principal` パラメータの代わりに使用できます。使用すると、そのモノが属するグループにアタッチされているポリシーと、これらのグループの親グループ (階層内のルートグループまで) にアタッチされているポリシーが返されます。

GetEffectivePolicies コマンドは、ポリシーのリストを返します。

```
{
    "effectivePolicies": [
        {
            "policyArn": "string",
            "policyDocument": "string",
            "policyName": "string"
        }
        ...
    ]
}
```

MQTT アクションテストの認可

TestAuthorization コマンドを使用して、あるモノに対して MQTT アクション (Publish、Subscribe) が許可されているかどうかをテストできます。

```
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` パラメータを使用して、モノにアタッチされた証明書の ARN を指定します。Amazon Cognito ID を使用している場合は、Cognito ID を `--principal` として指定するか、`--cognito-identity-pool-id` パラメータまたはその両方を使用します。`--cognito-identity-pool-id`だけを指定すると、認証されていないユーザーの ID のロールに関連付けられたポリシーが考慮されます。両方を使用すると、認証されたユーザーの ID プールのロールに関連付けられたポリシーが考慮されます。

`--auth-infos` パラメータの後にリソースおよびアクションタイプをリストすることによって、テストする 1 つ以上の MQTT アクションを指定します。この `actionType` フィールドには「PUBLISH」、「SUBSCRIBE」、「RECEIVE」、または「CONNECT」が含まれます。`resources` フィールドには、リソース ARN のリストが含まれている必要があります。詳細については、「AWS IoT Core ポリシー」を参照してください。

`--policy-names-to-add` パラメータを指定することで、ポリシーを追加する効果をテストできます。または、`--policy-names-to-skip` パラメータを使用してポリシーを削除する効果をテストできます。

オプションの `--client-id` パラメータを使用して、結果をさらに絞り込むこともできます。

TestAuthorization コマンドは、指定した `--auth-infos` クエリの各セットに対して許可または拒否されているアクションに関する詳細を返します。

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

モノの動的グループ

モノの動的グループは、レジストリ内の特定の検索クエリから作成されます。デバイス接続、デバイスシャドウ作成、 AWS IoT Device Defender 違反データなどの検索クエリパラメータは、これをサポートします。モノの動的グループでは、デバイスのデータのインデックス作成、検索、集計のためにフリートインデックス作成を有効にする必要があります。フリートのインデックス作成の検索クエリを使用してグループを作成する前に、モノの動的グループに属するモノをプレビューできます。詳細については、「フリートインデックス作成」および「クエリ構文」を参照してください。

> **Note**
>
> モノの動的グループのオペレーションは、レジストリオペレーションで計測されます。詳細については、「AWS IoT Core のその他の詳細」を参照してください。

モノの動的グループは、以下の点でモノの静的グループと異なります。

- モノのメンバーシップが明示的には定義されていません。モノの動的グループを作成するには、グループのメンバーシップを定義する検索クエリ文字列を定義します。
- モノの動的グループは、階層の一部にすることはできません。
- モノの動的グループにはポリシーを適用できません。
- モノの動的グループを作成、更新、および削除するには、異なる一連のコマンドを使用します。他のすべてのオペレーションでは、両方のタイプのモノのグループに同じコマンドを使用します。
- あたりの動的グループの数は限 AWS アカウント られています。

- モノのグループ名で個人を特定できる情報を使用しないでください。モノのグループ名は、暗号化されていない通信やレポートに表示される可能性があります。

モノの静的グループの詳細については、「モノの静的グループ」を参照してください。

モノの動的グループのユースケース

次のユースケースでは、モノの動的グループを使用できます。

モノの動的グループをジョブのターゲットとして指定します。

モノの動的グループをターゲットとする継続的なジョブを作成すると、目的の条件を満たすデバイスを自動的にターゲットにできます。この条件は、接続状態、またはソフトウェアバージョンやモデルなどのレジストリやシャドウに保存されている任意の基準です。モノがモノの動的グループに表示されない場合、ジョブからジョブドキュメントは受信されません。

例えば、デバイスフリートが更新プロセス中の中断リスクを最小限に抑えるためにファームウェアの更新を必要としており、バッテリー寿命が 80% 以上あるデバイスのファームウェアのみを更新したい場合です。80PercentBatteryLife と呼ばれるモノの動的グループを作成し、このグループには、バッテリー寿命が 80% を超えるデバイスのみが含まれ、ジョブのターゲットとして使用できます。バッテリー残量の条件を満たすデバイスだけがファームウェア更新を受け取ります。80% というバッテリー残量の条件に達したデバイスは、そのモノの動的グループに自動的に追加されて、ファームウェア更新を受け取ります。

また、ファームウェアやオペレーティングシステムが異なる複数のデバイスモデルがあり、新しいソフトウェア更新の異なるバージョンが必要になる場合があります。これは、連続ジョブを持つ動的グループで最も一般的なユースケースであり、デバイスモデル、ファームウェア、OS の組み合わせごとに動的グループを作成できます。その後、定義された条件に基づいてデバイスが自動的にこれらの

グループのメンバーになると、これらの動的グループに連続ジョブを設定して、ソフトウェアの更新をプッシュできます。

ジョブターゲットとしてモノのグループを指定する方法については、「CreateJob」を参照してください。

動的グループメンバーシップの変更を使用して目的のアクションを実行する

デバイスがモノの動的グループに追加または削除されるたびに、レジストリイベントの更新の一環として MQTT トピックに通知が送信されます。動的グループメンバーシップの更新に基づいて AWS サービスとやり取りし、必要なアクションを実行するようにAWS IoT Core ルールを設定できます。例えば、 Amazon DynamoDBへの記述、Lambda 関数の呼び出し、または Amazon SNS トピックへの通知の送信などです。

自動違反検出のためにモノの動的グループにデバイスを追加する

AWS IoT Device Defender Detect のお客様は、モノの動的グループでセキュリティプロファイルを定義できます。モノの動的グループのデバイスは、グループで定義されたセキュリティプロファイルによって違反を自動的に検出されます。

モノの動的グループにログレベルを設定して、きめ細かなログ記録でデバイスを観察する

モノの動的グループでログレベルを指定できます。これは、特定の条件を満たすデバイスのログ記録レベルと詳細のみをカスタマイズする場合に便利です。例えば、特定のファームウェアバージョンのデバイスが特定のルールの公開トピックでエラーを引き起こしていると思われる場合は、これらの問題をデバッグするために詳細なログ記録を設定することができます。この場合、このファームウェアバージョンを持つすべてのデバイスに対して動的グループを作成できます。これは、レジストリ属性またはデバイスシャドウとして保存されているものと見なされます。その後、このモノの動的グループとして定義されたログ記録ターゲットを使用して、デバッグレベルを設定できます。詳細なログ記録の詳細については、CloudWatch Logs AWS IoT を使用したモニタリング」を参照してください。特定のモノのグループのログ記録レベルを指定する方法の詳細については、「AWS IoTでリソース固有のログ記録を設定する」を参照してください。

モノの動的グループを作成する

モノの動的グループを作成するには CreateDynamicThingGroup コマンドを使用します。80PercentBatteryLife シナリオのモノの動的グループを作成するには、create-dynamic-thing-groupCLI コマンドを使用します。

```
$ aws iot create-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --query-
string "attributes.batterylife80"
```

> **Note**
>
> モノの動的グループの名前では、個人を特定できる情報を使用しないでください。

CreateDynamicThingGroup コマンドはレスポンスを返します。このレスポンスには、インデックス名、クエリ文字列、クエリバージョン、モノのグループ名、モノのグループ ID、モノのグループの Amazon リソースネーム (ARN) が含まれています。

```
{
    "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"
}
```

モノの動的グループの作成はすぐには行われません。モノの動的グループのバックフィルは完了するまでに時間がかかります。モノの動的グループを作成すると、そのグループの状態は BUILDING に設定されます。バックフィルが完了すると、ステータスは ACTIVE に変わります。モノの動的グループのステータスを確認するには DescribeThingGroup コマンドを使用します。

モノの動的グループを記述する

モノの動的グループに関する情報を取得するには DescribeThingGroup コマンドを使用します。

```
$ aws iot describe-thing-group --thing-group-name "80PercentBatteryLife"
```

DescribeThingGroup コマンドは、指定されたグループに関する情報を返します。

```
{
    "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"
}
```

モノの動的グループに対して DescribeThingGroup を実行すると、そのモノの動的グループに固有の属性が返されます。戻り属性の例は、queryString と ステータスです。

モノの動的グループのステータスは次のいずれかの値です。

`ACTIVE`

モノの動的グループは使用する準備ができています。

`BUILDING`

モノの動的グループは作成中であり、モノのメンバーシップが処理中です。

`REBUILDING`

モノの動的グループは、グループの検索クエリの調整が済んで、メンバーシップが更新中です。

> **Note**
>
> モノの動的グループを作成した後は、そのステータスに関係なくそれを使用できます。ステータスが `ACTIVE` であるモノの動的グループには、そのモノの動的グループの検索クエリに一致するすべてモノが含まれています。ステータスが `BUILDING` または `REBUILDING` であるモノの動的グループには、検索クエリに一致するモノがすべて含まれているとは限りません。

モノの動的グループを更新する

モノの動的グループの属性 (グループの検索クエリなど) を更新するには UpdateDynamicThingGroup コマンドを使用します。次のコマンドは、2 つの属性を更新します。1 つはモノグループの説明で、もう 1 つはメンバーシップ条件をバッテリー寿命 > 85 に変更するクエリ文字列です。

```
$ 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 コマンドは、更新後にグループのバージョン番号を含むレスポンスを返します。

```
{
    "version": 2
}
```

モノの動的グループの更新は一度には行われません。モノの動的グループのバックフィルは完了するまでに時間がかかります。モノの動的グループを更新すると、グループのメンバーシップが更新され、グループのステータスは REBUILDING に変わります。バックフィルが完了すると、ステータスは ACTIVE に変わります。モノの動的グループのステータスを確認するには DescribeThingGroup コマンドを使用します。

モノの動的グループを削除する

モノの動的グループを削除するには DeleteDynamicThingGroup コマンドを使用します。

```
$ aws iot delete-dynamic-thing-group --thing-group-name "80PercentBatteryLife"
```

DeleteDynamicThingGroup コマンドでは、出力が生成されません。

クラウド内のレコードが更新されている間は、モノが属しているグループを示すコマンド (ListGroupsForThing など) で、そのグループが表示され続けることがあります。

モノの動的および静的グループの制限

モノの動的および静的グループには次の共有の制限があります。

- モノのグループが持つことのできる属性の数は限られています。

- モノが属することができるグループの最大数は、限られています。
- モノのグループ名を変更することはできません。
- モノのグループ名には、û、é、ñ などの国際文字を含めることはできません。

モノの動的グループの制限

モノの動的グループには、次の制限があります。

フリートインデックス作成

フリートのインデックス作成サービスを有効にすると、デバイスのフリートに対して検索クエリを実行できます。フリートインデックスのバックフィルが完了したら、モノの動的グループを作成および管理できます。バックフィルプロセスの完了時間は、 AWS クラウドに登録されているデバイスフリートのサイズによって直接指定されます。フリートインデックス作成サービスをモノの動的グループに対して有効にした後は、モノの動的グループをすべて削除するまでは無効にできません。

> **Note**
>
> フリートインデックスに対してクエリを実行するアクセス許可があるユーザーは、フリート全体でモノのデータにアクセスできます。

モノの動的グループの数が限られている

モノの動的グループの数は限られています。

コマンドが成功するとエラーを口グに記録できる

モノの動的グループを作成または更新するときに、モノの動的グループに追加する対象となる可能性があるものの、まだ追加されていないモノが存在する可能性があります。このシナリオでは、エラーの口グ記録と `AddThingToDynamicThingGroupsFailed` メトリクスの生成中に、作成または更新コマンドが成功します。1 つのメトリクスは、複数の口グエントリを表すことができます。

CloudWatch 口グのエラーログエントリは、次の場合に作成されます。

- 対象となるモノをモノの動的グループに追加することはできません。
- モノは動的モノグループから削除され、別のグループに追加されます。

モノがモノの動的グループに追加される対象になると、次のことを考慮します。

- 既に最大限のグループに追加されていますか? (「制限」を参照してください)
  - いいえ: モノはモノの動的グループに追加されます。
  - はい: モノはモノの動的グループのメンバーですか?
    - いいえ: モノをモノの動的グループに追加することはできず、エラーがログに記録され、AddThingToDynamicThingGroupsFailed メトリクスが生成されます。
    - はい: 参加するモノの動的グループは、そのモノが既にメンバーになっているモノの動的グループよりも古いものですか?
      - いいえ: モノをモノの動的グループに追加することはできず、エラーがログに記録され、AddThingToDynamicThingGroupsFailed メトリクスが生成されます。
      - はい: 最新のモノの動的グループからモノを削除して、エラーを ログに記録し、モノをモノの動的グループに追加します。これにより、モノが削除されたモノの動的グループのエラーと AddThingToDynamicThingGroupsFailedメトリクスが生成されます。

モノの動的グループ内のモノが検索クエリに一致しなくなると、そのモノはモノの動的グループから削除されます。同様に、モノが更新されてモノの動的グループの検索クエリを満たすようになると、前述のようにそのモノがそのグループに追加されます。このような追加と削除は正常であり、エラーログエントリは生成されません。

`overrideDynamicGroups` が有効になっている場合は、静的グループの方が動的グループより優先されます。

モノが属することができるグループの最大数は、限られています。AddThingToThingGroup または UpdateThingGroupsForThing コマンドを使用してモノのメンバーシップを更新する場合、`--overrideDynamicGroups` パラメータを追加するとモノの動的グループよりモノの静的グループの方が優先されます。

モノをモノの静的グループに追加する場合、次のことを考慮します。

- モノは既にグループの最大数に属していますか?
  - いいえ: モノがモノの静的グループに追加されます。
  - はい: モノは動的グループに属していますか?
    - いいえ: モノをモノのグループに追加することができません。コマンドにより例外が発生します。
    - はい: --overrideDynamicGroups は有効になっていましたか?

- いいえ: モノをモノのグループに追加することができません。コマンドにより例外が発生します。
- はい: モノは最も新しく作成されたモノの動的グループから削除されて、エラーがログに記録され、モノが削除されたモノの動的グループの`AddThingToDynamicThingGroupsFailed` メトリクスが生成されます。その後、モノがモノの静的グループに追加されます。

作成日時が古いモノの動的グループの方が優先されます。

モノが属することができるグループの最大数は、限られています。作成または更新オペレーションによってモノに対する追加のグループ資格が作成され、モノがグループ制限に達すると、別の動的モノグループから削除して、この追加を有効にすることができます。この動作の詳細については、「コマンドが成功するとエラーを口グに記録できる」と「`overrideDynamicGroups` が有効になっている場合は、静的グループの方が動的グループより優先されます。」で例を参照してください。

モノがモノの動的グループから削除されると、エラーが口グに記録され、イベントが発生します。

モノの動的グループにポリシーを適用することはできません。

モノの動的グループにポリシーを適用しようとすると、例外が生成されます。

モノの動的グループのメンバーシップには結果整合性があります。

レジストリでは、モノの最終的な状態のみが評価されます。複数の状態が急速に更新された場合、中間の状態はスキップされることがあります。ルールまたはジョブを、メンバーシップが中間的な状態に応じて変化するモノの動的グループと関連付けないでください。

AWS IoT モノを MQTT クライアント接続に関連付ける

排他的なモノの関連付けとは、X.509 証明書を単一の AWS IoT のモノにアタッチすることです。この場合、証明書を他のモノに使用することはできません。証明書が 1 つの IoT のモノでのみ使用されるようにすることで、セキュリティの脆弱性を防ぐことができます。

では AWS IoT、クライアント ID は AWS IoT Core MQTT ブローカーに接続するときのモノまたはデバイスの一意の識別子です。非排他的な関連付けを使用する場合は、複数のモノを同じ証明書にアタッチできます。非排他的なモノの関連付けが適用されている場合、明確な関連付けを維持し、潜在的な競合を回避するためには、クライアント ID をモノの名前と一致させる必要があります。

このトピックの内容

- ユースケース
- モノを接続に関連付ける方法

ユースケース

モノを接続に関連付けると、次の機能が提供されます。

> **Note**
>
> IoT のモノとクライアント接続に非排他的な関連付けがある場合は、以下の機能のうち、ライフサイクルイベント機能を除くすべての機能が利用可能な点に留意してください。モノの名前をライフサイクルイベントメッセージに含めるには、IoT のモノとクライアント接続に排他的な関連付けが必要です。

モノのポリシー変数 - モノのポリシー変数を使用して AWS IoT 、API オペレーションへのデバイスアクセスを許可できます。これらの変数を使用すると、名前、タイプ、属性値などのモノのプロパティに基づいてアクセス許可を付与または拒否する AWS IoT Core ポリシーを作成できます。モノのポリシー変数を使用すると、同じポリシーを適用して複数の AWS IoT Core デバイスを制御できます。これによって、ポリシー管理を簡素化し、リソースの重複を減らすことができます。詳細については、「モノのポリシー変数」を参照してください。

ライフサイクルイベント - ライフサイクルイベント (接続、切断、サブスクライブ、サブスクライブ解除など) でモノの名前を受け取ることができます。これにより、ルールなど、メッセージに含まれるモノの名前を処理できます。詳細については、「ライフサイクルイベント」を参照してください。

リソース固有の口グ記録 - モノのグループのリソース固有の口グ記録を設定し、定義されたモノのグループ内のすべてのモノに対して口グ記録の設定を簡単に適用することができます。詳細については、「???」を参照してください。

コスト配分 - コスト配分用にカスタムタグを使用して請求グループを作成し、それらのグループにモノを追加できます。詳細については、「請求グループ」を参照してください。

モノを接続に関連付ける方法

クライアント ID がレジストリ内のモノの名前と一致する場合、その IoT モノに X.509 証明書をアタッチすると、 AWS IoT Core はクライアント接続をモノに関連付けます。クライアント ID がレジストリ内のモノの名前と一致しない場合は、モノに X.509 証明書を排他的にアタッチすることで、この関連付けを確立することができます。この排他的なアタッチメントを持つモノは、排他的なモノ

と呼ばれます。それ以外のモノは、非排他的なモノと呼ばれます。証明書が排他的なモノに関連付けられている場合、この証明書は排他的なモノからデタッチされた場合にのみ、他のモノに関連付けることができます。このセクションでは、 AWS マネジメントコンソール または AWS CLI のいずれかを選択して、モノを接続に関連付けます。

AWS マネジメントコンソール

AWS マネジメントコンソールを使用して証明書を排他的にモノにアタッチする方法

1. AWS IoT コンソールでAWS IoT ホームページを開きます。左のナビゲーションペインの [セキュリティ] で、[証明書] を選択します。
2. [証明書] ページで、モノをアタッチする証明書を選択します。次に、ページの右上隅にある [アクション] から [モノにアタッチする] を選択します。

   または、証明書を選択して、証明書の詳細ページに移動します。[モノ] タブを選択し、[モノにアタッチする] を選択します。
3. [モノに証明書をアタッチする] ページで、[モノを接続に関連付ける] チェックボックスをオンにします。次に、[モノ] ドロップダウンリストから、この証明書をアタッチするモノを選択します。
4. [モノをアタッチ] を選択します。アクションが成功すると、「証明書にモノが正常にアタッチされました」というバナーが表示され、[モノ] タブにモノが追加されます。

を使用して排他的なモノから証明書をデタッチするには AWS マネジメントコンソール

1. AWS IoT コンソールでAWS IoT ホームページを開きます。左のナビゲーションペインの [セキュリティ] で、[証明書] を選択します。
2. [証明書] ページで証明書を選択し、証明書の詳細ページに移動します。
3. 証明書の詳細ページで、[モノ] タブを選択します。次に、証明書をデタッチするモノを選択します。[モノをデタッチする] するを選択します。
4. [モノをデタッチする] ウィンドウで、アクションを確認します。[デタッチ] を選択します。アクションが成功すると、「証明書からモノを正常にデタッチしました」というバナーが表示され、モノは [モノ] タブに表示されなくなります。

AWS CLI

1. を使用してモノに証明書をアタッチするには AWS CLI、attach-thing-principal コマンドを実行します。排他的な証明書からモノへのアタッチメントを指定するには、`--thing-principal-`

type フィールドで EXCLUSIVE_THING を指定する必要があります。以下は、コマンドの一例です。

```
aws iot attach-thing-principal \
    --thing-name "thing_1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-principal-type "EXCLUSIVE_THING"
```

このコマンドでは、出力が生成されません。詳細については、「???」を参照してください。

2. 指定の証明書に関連付けられているモノをアタッチメントタイプとともに一覧表示するには、list-principal-things-v2 コマンドを実行します。アタッチメントタイプは、証明書がモノにアタッチされる方式を指します。以下は、コマンドの一例です。

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

出力は次のようになります。

```
{
    "PrincipalThingObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        }
    ]
}
```

詳細については、「???」を参照してください。

3. 指定のモノに関連付けられているプリンシパルをアタッチメントタイプとともに一覧表示するには、list-thing-principals-v2 コマンドを実行します。アタッチメントタイプは、証明書がモノにアタッチされる方式を指します。以下は、コマンドの一例です。

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

出力は次のようになります。

```
{
    "ThingPrincipalObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        }
    ]
}
```

詳細については、「???」を参照してください。

4. モノから証明書をデタッチするには、detach-thing-principal コマンドを実行します。

```
aws iot detach-thing-principal \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-name "thing_1"
```

このコマンドでは、出力が生成されません。詳細については、「???」を参照してください。

メッセージエンリッチメントための伝播属性の追加

では AWS IoT Core、モノの属性または接続の詳細からのコンテキストメタデータである伝播属性を追加することで、デバイスからの MQTT メッセージを強化できます。メッセージエンリッチメントと呼ばれるこのプロセスは、さまざまなシナリオで役立ちます。たとえば、デバイス側で変更を行ったり、ルールを使用したりすることなく、すべてのインバウンドの発行オペレーションでのメッセージを強化できます。伝播属性を活用することで、ルールの設定や再発行の設定の管理における複雑さをなくして、IoT データをより効率的かつ費用対効果の高い方法で強化できるようになります。

メッセージエンリッチメント機能は、基本的な取り込みとメッセージブローカーを使用する AWS IoT Core お客様が使用できます。発行するデバイスでは任意の MQTT バージョンを使用できますが、サブスクライバー (メッセージを消費するアプリケーションまたはサービス) は、MQTT 5 をサポートして、伝播属性を含むエンリッチされたメッセージを受信する必要があります。エンリッチされたメッセージは、デバイスから発行されたすべてのメッセージに MQTT 5 ユーザープロパティと

して追加されます。ルールを使用する場合は、get_user_properties 関数を使用して、データに基づいてメッセージをルーティングまたは処理するために強化されたデータを取得できます。

では AWS IoT Core、 AWS マネジメントコンソール または を使用して、モノのタイプを作成または更新するときに伝播属性を追加できます AWS CLI。

> **Important**
>
> 伝播属性を追加するときは、メッセージを発行するクライアントが証明書で認証されていることを確認する必要があります。詳細については、「クライアント認証」を参照してください。

> **Note**
>
> コンソール内で MQTT テストクライアントを使用してこの機能をテストしようとすると、この機能には関連付けられた証明書で認証された MQTT クライアントが必要なため、機能しない場合があります。

AWS マネジメントコンソール

を使用してメッセージエンリッチメントの伝播属性を追加するには AWS マネジメントコンソール

1. AWS IoT コンソールでAWS IoT ホームページを開きます。左側のナビゲーションで、[管理] から [すべてのデバイス] を選択します。次に、[モノのタイプ]を選択します。
2. [モノのタイプ] ページで [モノのタイプを作成] を選択します。

   モノのタイプを更新してメッセージエンリッチメントを設定するには、モノのタイプを選択します。次に、モノのタイプの詳細ページで、[更新] を選択します。
3. [モノのタイプを作成] ページにある [モノのタイプのプロパティ] で、モノのタイプの情報を選択または入力します。

   モノのタイプを更新する場合は、前のステップで [更新] を選択すると、[モノのタイプのプロパティ] が表示されます。
4. [追加設定] で、[属性伝播] を展開します。次に、[モノの属性] を選択し、発行される MQTT5 メッセージに入力するモノの属性を入力します。コンソールを使用して、最大 3 つのモノの属性を追加できます。

[属性伝播] セクションで、[接続属性] を選択し、属性タイプとオプションで属性名を入力します。

5. オプションでタグを追加します。次に、[モノのタイプを作成] を選択します。

   モノのタイプを更新する場合は、[モノのタイプをアップデート] を選択します。

AWS CLI

1. を使用して新しいモノのタイプを作成してメッセージエンリッチメントの伝播属性を追加するには AWS CLI、create-thing-type コマンドを実行します。以下は、コマンドの一例です。

```
aws iot create-thing-type \
    --thing-type-name "LightBulb" \
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

   コマンドの出力は以下のようになります。

```
{
 "thingTypeName": "LightBulb",
 "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
 "thingTypeId": "ce3573b0-0a3c-45a7-ac93-4e0ce14cd190"
}
```

2. を使用してモノのタイプを更新してメッセージエンリッチメントを設定するには AWS CLI、update-thing-type コマンドを実行します。このコマンドを実行した場合にのみ、mqtt5Configuration を更新できることに注意してください。以下は、コマンドの一例です。

```
aws iot update-thing-type \
    --thing-type-name "MyThingType" \
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

   このコマンドでは、出力が生成されません。

3. モノのタイプを記述するには、describe-thing-type コマンドを実行します。このコマンドは、thing-type-properties フィールドにメッセージエンリッチメントの設定情報を含む出力を生成します。以下は、コマンドの一例です。

```
aws iot describe-thing-type \
    --thing-type-name "LightBulb"
```

出力は次のようになります。

```
{
 "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"
 }
}
```

詳細については、「???」を参照してください。

AWS IoT リソースのタグ付け

モノのグループ、モノのタイプ、トピックルール、ジョブ、スケジュールによる監査、およびセキュリティプロファイルを管理および整理しやすくするために、これらのリソースのそれぞれにタグという形式で独自のメタデータをオプションで割り当てることができます。このセクションでは、タグとその作成方法について説明します。

モノに関連するコストを管理しやすくするために、モノが含まれる請求グループを作成できます。その後、その請求グループそれぞれに、メタデータが含まれているタグを割り当てることができます。このセクションでは、請求グループ、および請求グループの作成および管理に使用できるコマンドについても説明します。

タグの基本

タグを使用して、さまざまな方法 (目的、所有者、環境など) で AWS IoT リソースを分類できます。同じ型のリソースが多い場合に役立ちます。割り当てたタグに基づいてリソースをすばやく識別できます。タグはそれぞれ、1 つのキーとオプションの値で構成され、どちらもユーザーが定義します。たとえば、デバイスをタイプ別に追跡しやすくするために、モノのタイプに対して一連のタグを定義できます。リソースの種類ごとのニーズを合わせて一連のタグキーを作成することをお勧めします。一貫性のあるタグキーセットを使用することで、リソースの管理が容易になります。

追加または適用したタグに基づいて、リソースを検索およびフィルター処理できます。また、請求グループタグを使用して、コストを分類および追跡することもできます。また、「IAM ポリシーでのタグの使用」で説明しているように、タグを使用してリソースへのアクセスを制御することもできます。

使いやすいように、 AWS マネジメントコンソールのタグエディタは、タグを作成および管理するための一元化された方法を提供します。詳細については、「 マネジメントコンソールの使用」の「タグエディタの使用 AWS」を参照してください。

AWS CLI および AWS IoT API を使用してタグを操作することもできます。以下のコマンドで Tags フィールドを使用して、モノのグループ、モノのタイプ、トピックルール、ジョブ、セキュリティプロファイル、ポリシー、請求グループ、およびモノの作成時にモノと関連付けたパッケージとバージョンにタグを関連付けることができます。

- CreateBillingGroup
- CreateDestination

- CreateDeviceProfile
- CreateDynamicThingGroup
- CreateJob
- CreateOTAUpdate
- CreatePolicy
- CreateScheduledAudit
- CreateSecurityProfile
- CreateServiceProfile
- CreateStream
- CreateThingGroup
- CreateThingType
- CreateTopicRule
- CreateWirelessGateway
- CreateWirelessDevice

以下のコマンドを使用して、タグ付けがサポートされている既存のリソースに対してタグを追加、変更、または削除できます。

- TagResource
- ListTagsForResource
- UntagResource

タグのキーと値は編集でき、タグはリソースからいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはできません。特定のリソースについて既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によってオーバーライドされます。リソースを削除すると、リソースに関連付けられているすべてのタグも削除されます。

タグの制約と制限

タグには以下のような基本制限があります。

- リソースあたりのタグの最大数: 50
- キーの最大長: 127 文字 UnicodeUTF-8)
- 値の最大長: 255 文字 (Unicode) (UTF-8)

- タグのキーと値は大文字と小文字が区別されます。
- タグの名前または値に「aws:」プレフィックスは使用しないでください。 AWS 使用のために予約されています。このプレフィックスが含まれるタグの名前または値は編集または削除できません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。
- 複数のサービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでも許可される文字に制限が適用されることがあるため注意ください。使用できる文字は、UTF-8 で表現できる文字、スペース、および数字と、特殊文字 +、-、=、.、_、:、/、@ です。

IAM ポリシーでのタグの使用

AWS IoT API アクションに対して使用する IAM ポリシーで、タグベースのリソースレベルアクセス許可を適用することができます。これにより、ユーザーがどのリソースを作成、変更、または使用できるかを制御しやすくなります。IAM ポリシーの以下の条件コンテキストのキーと値とともに Condition 要素 (Condition ブロックとも呼ばれる) を使用して、リソースのタグに基づいてユーザーアクセス (アクセス許可) を制御できます。

- 特定のタグを持つリソースに対してユーザーアクションを許可または拒否するには、`aws:ResourceTag/`*`tag-key`*`: `*`tag-value`* を使用します。
- タグが許可されているリソースを作成または変更する API リクエストを作成する場合に、特定のタグが使用されている (または、使用されていない) ことを要求するには、`aws:RequestTag/`*`tag-key`*`: `*`tag-value`* を使用します。
- タグが許可されているリソースを作成または変更する API リクエストを作成する場合に、特定の一連のタグが使用されている または、使用されていないことを要求するには、`aws:TagKeys: [`*`tag-key`*`, ...]` を使用します。

> **Note**
>
> IAM ポリシーの条件コンテキストキーと値は、タグ付けできるリソースの識別子が必須パラメータである AWS IoT アクションにのみ適用されます。例えば、DescribeEndpoint を使用しても、このリクエストではタグ付け可能なリソース (モノのグループ、モノのタイプ、トピックルール、ジョブ、またはセキュリティプロファイル) が参照されていないため、条件コンテキストのキーと値に基づいて許可または拒否されることはありません。タグ付け可能な AWS IoT リソースとサポートする条件キーの詳細については、「アクション、リソース、および条件キー AWS IoT」を参照してください。

タグの使用の詳細については、[AWS Identity and Access Management User Guide](ユーザーガイド)の[Controlling Access Using Tags](タグを使用したアクセスの制御)を参照してください。そのガイドの[IAM JSON Policy Reference](IAM JSON ポリシーリファレンス)セクションには、IAM での JSON ポリシーの要素、変数、および評価ロジックの詳細な構文、説明、および例が記載されています。

次のポリシー例では、ThingGroup アクションにタグベースの 2 つの制約が適用されています。このポリシーによって制限されている IAM ユーザー

- モノグループにタグ「env=prod」を作成することはできません (この例の`"aws:RequestTag/env" : "prod"`の行を参照)。
- 既存のタグ "env=prod" を持つモノグループに対しては、変更またはアクセスできません (この例の `"aws:ResourceTag/env" : "prod"` の行を参照)。

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

次のように、タグ値を1 つのリストとして指定して、1 つのタグキーに対して複数のタグ値を指定することもできます。

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

> **Note**
>
> タグに基づいてリソースへのユーザーのアクセスを許可または拒否する場合は、ユーザーが同じリソースに対してそれらのタグを追加または削除することを明示的に拒否することを検討する必要があります。そうしないと、ユーザーはそのリソースのタグを変更することで、制限を回避してリソースにアクセスできてしまいます。

請求グループ

AWS IoT では、個々のモノにタグを直接適用することはできませんが、請求グループにモノを配置したり、タグを適用したりできます。の場合 AWS IoT、タグに基づくコストと使用状況データの配分は請求グループに限定されます。

AWS IoT Core ワイヤレスデバイスやゲートウェイなどの for LoRaWAN リソースを請求グループに追加することはできません。ただし、請求グループに追加できる AWS IoT モノに関連付けることができます。

以下のコマンド使用できます。

- AddThingToBillingGroup は、モノを請求グループに追加します。
- CreateBillingGroup は請求グループを作成します。
- DeleteBillingGroup は請求グループを削除します。
- DescribeBillingGroup は請求グループに関する情報を返します。
- ListBillingGroups は作成済みの請求グループをリスト化します。
- ListThingsInBillingGroup 指定された請求グループにユーザーが追加したモノをリスト化します。
- RemoveThingFromBillingGroup は指定されたモノを請求グループから削除します。
- UpdateBillingGroup は請求グループに関する情報を更新します。
- CreateThing を使用すると、モノを作成するときにその請求グループを指定できます。
- DescribeThing は、モノが属する請求グループ (存在する場合) も含めて、モノの説明を返します。

AWS IoT Wireless API は、ワイヤレスデバイスとゲートウェイを AWS IoT モノに関連付けるためのこれらのアクションを提供します。

- AssociateWirelessDeviceWithThing
- AssociateWirelessGatewayWithThing

コスト配分と使用状況データの表示

請求グループタグを使用して、コストを分類および追跡できます。請求グループ (およびそれらに含まれるモノ) にタグを適用すると、 はコスト配分レポートを、タグによって集計された使用量とコストを含むカンマ区切り値 (CSV) ファイルとして AWS 生成します。自社のカテゴリ (たとえばコストセンター、アプリケーション名、所有者) を表すタグを適用すると、複数のサービスにわたってコストを分類することができます。コスト割り当てでのタグの使用の詳細については、「AWS 請求とコスト管理ユーザーガイド」の「コスト割り当てタグの使用」を参照してください。

> **Note**
>
> 使用状況やコストのデータを、請求グループに配置したモノと正確に関連付けるには、各デバイスおよびアプリケーションが以下の要件を満たしている必要があります。
>
> - モノとして登録されている AWS IoT。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。

- モノの名前のみをクライアント ID として使用して、MQTT 経由で AWS IoT メッセージブローカーに接続します。詳細については、「the section called “デバイス通信プロトコル”」を参照してください。クライアント ID がモノの名前と一致しない場合は、排他的なモノのアタッチメントを有効にして関連付けを確立できます。詳細については、「???」を参照してください。
- モノに関連付けられているクライアント証明書を使用して認証されている。

請求グループでは (請求グループと関連付けられているモノのアクティビティに基づいて) 以下の価格ディメンションを利用できます。

- 接続 (接続するクライアント ID として使用されたモノの名前に基づく)。
- メッセージング (モノからのインバウンドメッセージおよびモノへのアウトバウンドメッセージに基づく。MQTT のみ)。
- シャドウオペレーション (シャドウ更新をトリガーしたメッセージの発信元のモノに基づく)。
- トリガーされたルール (インバウンドメッセージがルールをトリガーしたモノに基づく。MQTT のライフサイクルイベントによってトリガーされたルールには適用されない)。
- モノのインデックス更新 (インデックスに追加されたモノに基づく)。
- リモートアクション (モノの更新に基づく)。
- AWS IoT Device Defender detect レポート(アクティビティが報告されたモノに基づく)。

タグに基づく (および、請求グループに対してレポートされた) コストと使用状況のデータには、以下のアクティビティは反映されません。

- デバイスレジストリオペレーション (モノ、モノのグループ、モノのタイプの更新を含む)。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。
- モノのグループインデックス更新 (モノのグループを追加した場合)。
- インデックス検索クエリ。
- デバイスプロビジョニング.
- AWS IoT Device Defender audit レポート。

のセキュリティ AWS IoT

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを活用できます。

セキュリティは、 AWS とお客様の間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 AWS また、 では、安全に使用できるサービスも提供しています。「AWS 」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。が適用されるコンプライアンスプログラムの詳細については AWS IoT、AWS 「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、 を使用する際の責任共有モデルの適用方法を理解するのに役立ちます AWS IoT。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成する AWS IoT ように を設定する方法を示します。また、 AWS IoT リソースのモニタリングや保護に役立つ他の AWS サービスの使用方法についても説明します。

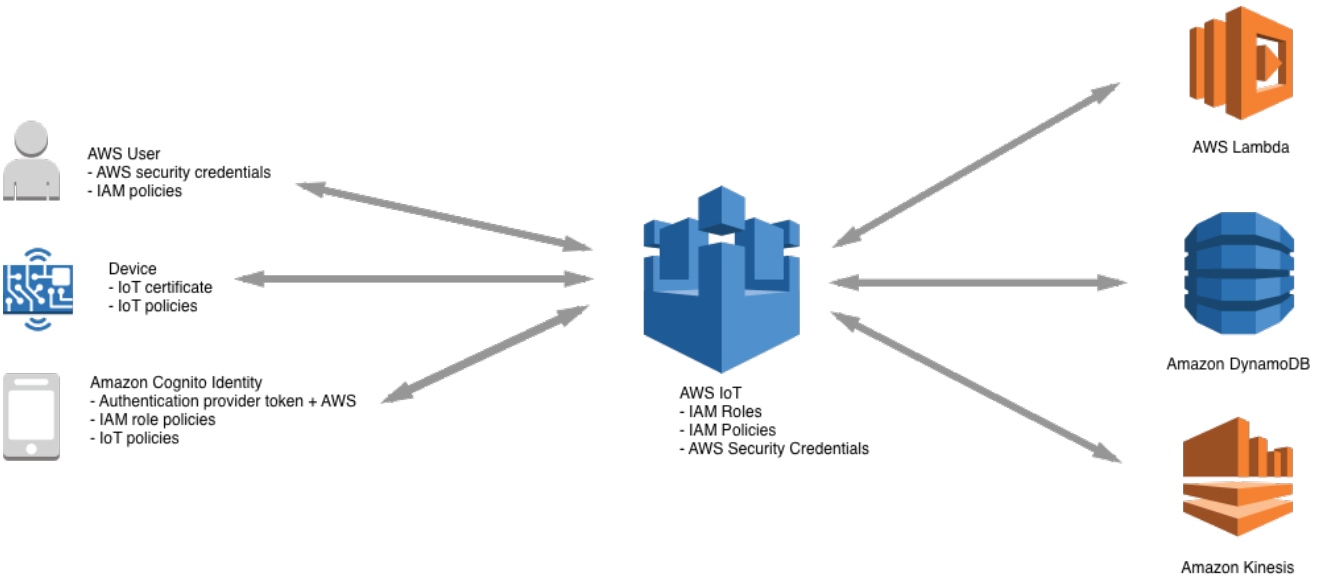
トピック

- AWS IoT セキュリティ
- 認証
- Authorization
- でのデータ保護 AWS IoT Core
- の ID とアクセスの管理 AWS IoT
- ログ記録とモニタリング
- AWS IoT Core のコンプライアンス検証
- AWS IoT Core の耐障害性
- インターフェイス VPC エンドポイント AWS IoT Core での の使用
- のインフラストラクチャセキュリティ AWS IoT

- AWS IoT Core を使用した本番稼働用フリートまたはデバイスのセキュリティモニタリング
- のセキュリティのベストプラクティス AWS IoT Core
- AWS トレーニングと認定

AWS IoT セキュリティ

接続された各デバイスまたはクライアントには、 AWS IoTと対話する認証情報が必要です。との間のすべてのトラフィック AWS IoT は、Transport Layer Security (TLS) を介して安全に送信されます。 AWS クラウドセキュリティメカニズムは、 AWS IoT と他の AWS サービスの間を移動するデータを保護します。



- AWS IoTでは、デバイスの認証情報 (X.509 証明書、 AWS 認証情報、Amazon Cognito アイデンティティ、フェデレーテッドアイデンティティ、またはカスタム認証トークン) とポリシーを管理する責任があります。また、各デバイスへの一意の ID の割り当てと、各デバイスまたはデバイスグループに対するアクセス許可の管理も担当します。
- デバイスは、安全な TLS 接続を介して X.509 証明書または Amazon Cognito ID AWS IoT を使用して に接続します。研究および開発中、および API コールを行うか WebSockets を使用する一部のアプリケーション用に、IAM ユーザーとグループ、またはカスタム認証トークンを使用して認証することもできます。詳細については、「IAM ユーザー、グループ、ロール」を参照してください。
- AWS IoT 認証を使用する場合、メッセージブローカーは、デバイスの認証、デバイスデータの安全な取り込み、および AWS IoT ポリシーを使用してデバイスに対して指定したアクセス許可の付与または拒否を行います。

- カスタム認証を使用する場合、カスタムオーソライザーはデバイスを認証し、 または IAM AWS IoT ポリシーを使用してデバイスに指定したアクセス許可を付与または拒否します。
- AWS IoT ルールエンジンは、定義したルールに従って、デバイスデータを他のデバイスまたは他の AWS サービスに転送します。を使用して AWS Identity and Access Management 、最終送信先にデータを安全に転送します。詳細については、「の ID とアクセスの管理 AWS IoT」を参照してください。

認証

認証は、クライアントまたはサーバーの ID を確認するメカニズムです。サーバー認証は、デバイスまたは他のクライアントが実際の AWS IoT エンドポイントと通信していることを確認するプロセスです。クライアント認証は、デバイスまたは他のクライアントが自身を認証するプロセスです AWS IoT。

X.509 証明書の概要

X.509 証明書は、X.509 パブリックキーインフラストラクチャ規格を使用して、パブリックキーと証明書内の ID を関連付けるための、デジタル証明書です。X.509 証明書は、認証機関 (CA) と呼ばれる信頼済みエンティティによって発行されます。CA は、CA 証明書と呼ばれる 1 つ以上の特別な証明書を管理しており、この証明書は X.509 証明書を発行するために使用されます。認証機関にのみ CA 証明書に対するアクセス許可があります。X.509 証明書チェーンは、クライアントによるサーバー認証とサーバーによるクライアント認証の両方に使用されます。

サーバー認証

デバイスまたは他のクライアントが に接続しようとすると AWS IoT Core AWS IoT Core 、サーバーはデバイスがサーバーを認証するために使用する X.509 証明書を送信します。認証は、X.509 証明書チェーンの検証を通じて、TLS レイヤーで行われます。これは、HTTPS URL にアクセスしたときにブラウザが使用するのと同じ方法です。独自の認証機関からの証明書を使用する場合は、「CA 証明書の管理」を参照してください。

デバイスまたは他のクライアントがエンドポイントへの TLS 接続を確立すると AWS IoT Core 、 はデバイスが別のサーバー偽装ではなく AWS IoT Core 、通信していることを検証するために使用する証明書チェーン AWS IoT Core を表します AWS IoT Core。表示されるチェーンは、デバイスが接続しているエンドポイントのタイプと、TLS ハンドシェイク中にクライアントと AWS IoT Core がネゴシエートした暗号スイートの組み合わせによって異なります。

エンドポイントタイプ

AWS IoT Core は をサポートしますiot:Data-ATS。 iot:Data-ATSエンドポイントは、Amazon Trust Services CA によって署名されたサーバー証明書を提供します。

ATS エンドポイントによって提示された証明書は、Starfield によってクロス署名されています。一部の TLS クライアント実装では、信頼のルートを検証する必要があり、Starfield CA 証明書がクライアントの信頼ストアにインストールされている必要があります。

> **Warning**
>
> 証明書全体 (発行者名など) をハッシュする証明書固定方法を使用することはお勧めしません。これは、提供する ATS 証明書が Starfield によってクロス署名され、発行者名が異なるため、証明書の検証に失敗するためです。

> **Important**
>
> iot:Data-ATS エンドポイントを使用します。Symantec 証明書と Verisign 証明書は廃止され、 ではサポートされなくなりました AWS IoT Core。

describe-endpoint コマンドを使用して ATS エンドポイントを作成できます。

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

この describe-endpoint コマンドは、次の形式でエンドポイントを返します。

```
account-specific-prefix.iot.your-region.amazonaws.com
```

> **Note**
>
> describe-endpoint が初めて呼び出されると、エンドポイントが作成されます。以降の describe-endpoint への呼び出しはすべて、同じエンドポイントを返します。

> **Note**
>
> AWS IoT Core コンソールでiot:Data-ATSエンドポイントを表示するには、「設定」を選択します。コンソールには iot:Data-ATS エンドポイントのみが表示されます。

AWS SDK for Java **IotDataPlaneClient**を使用した の作成

iot:Data-ATS エンドポイントを使用する IotDataPlaneClient を作成するには、以下を実行する必要があります。

- DescribeEndpoint API を使用して iot:Data-ATS エンドポイントを作成します。
- IotDataPlaneClient を作成するときに、そのエンドポイントを指定します。

次の例では、これらのオペレーションの両方を実行します。

```
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();
}
```

サーバー認証用の CA 証明書

使用しているデータエンドポイントのタイプとネゴシエートした暗号スイートに応じて、 AWS IoT Core サーバー認証証明書は次のいずれかのルート CA 証明書によって署名されます。

Amazon Trust Services エンドポイント (推奨)

> **Note**
>
> 場合によって、以下のリンクを右クリックし、[Save link as...] (名前を付けてリンク先を保存) を選択して、これらの証明書をファイルとして保存する必要があります。

- RSA 2048 ビットキー: Amazon Root CA 1。
- RSA 4096 ビットキー: Amazon Root CA 2。将来の利用のために予約されています。
- ECC 256 ビットキー: Amazon Root CA 3。
- ECC 384 ビットキー: Amazon Root CA 4。将来の利用のために予約されています。

これらの証明書はすべて、 Starfield ルート CA 証明書によってクロス署名されています。アジアパシフィック (ムンバイ) AWS IoT Core リージョン AWS IoT Core での の 2018 年 5 月 9 日に開始されたすべての新しいリージョンは、ATS 証明書のみを提供します。

VeriSign エンドポイント (レガシー)

- RSA 2048 ビットキー: VeriSign クラス 3 Public Primary G5 ルート CA 証明書

サーバー認証のガイドライン

AWS IoT Core サーバー認証証明書を検証するデバイスの機能に影響を与える可能性のある多くの変数があります。例えば、デバイスのメモリ制限が大きすぎてルート CA 証明書をすべて保持できない場合や、デバイスが証明書検証の標準以外の方法を実装している場合があります。これらの理由から、次のガイドラインに従うことをお勧めします。

- ATS エンドポイントを使用して、サポートされているすべての Amazon Root CA 証明書をインストールすることをお勧めします。
- これらの証明書をすべてデバイスに保存できず、デバイスが ECC ベースの検証を使用していない場合は、Amazon Root CA 3 および Amazon Root CA 4 ECC 証明書を省略できます。デバイスが RSA ベースの証明書検証を実装していない場合は、Amazon Root CA 1 および Amazon Root CA 2 RSA 証明書を省略できます。場合によって、以下のリンクを右クリックし、[Save link as... (名前を付けてリンク先を保存)] を選択して、これらの証明書をファイルとして保存する必要があります。
- ATS エンドポイントに接続するときにサーバー証明書の検証の問題が発生した場合は、関連するクロス署名付き Amazon ルート CA 証明書を信頼ストアに追加してみてください。場合によって、以下のリンクを右クリックし、[Save link as... (名前を付けてリンク先を保存)] を選択して、これらの証明書をファイルとして保存する必要があります。
  - 相互署名済み Amazon Root CA 1
  - 相互署名済みの Amazon Root CA 2 - 将来の使用のために予約済みです。
  - 相互署名済み Amazon Root CA 3

  - 相互署名済みの Amazon Root CA 4 - 将来の使用のために予約済みです。
- サーバー証明書の検証の問題が発生した場合は、デバイスがルート CA を明示的に信頼する必要がある可能性があります。Starfield Root CA Certificate を信頼ストアに追加してみてください。
- 上記の手順を実行しても問題が解決しない場合は、AWS デベロッパーサポートにお問い合わせください。

> **Note**
>
> CA 証明書には有効期限があり、その後、サーバー証明書の検証には使用できません。有効期限が切れる前に CA 証明書を交換する必要がある場合があります。すべてのデバイスまたはクライアントにインストールされているルート CA 証明書をアップデートして、進行中の接続を安全にし、セキュリティベストプラクティスを最新の状態に保つ必要があります。

> **Note**
>
> デバイスコード AWS IoT Core で に接続するときは、接続に使用している API に証明書を渡します。使用する API は SDK によって異なります。詳細については、「AWS IoT Core デバイス SDK」を参照してください。

クライアント認証

AWS IoT は、デバイス認証またはクライアント認証用に 3 種類の ID プリンシパルをサポートしています。

- X.509 クライアント証明書
- IAM ユーザー、グループ、ロール
- Amazon Cognito ID

これらの ID は、デバイス、モバイル、ウェブ、またはデスクトップアプリケーションで使用できます。これらは、ユーザーが AWS IoT コマンドラインインターフェイス (CLI) コマンドを入力することによっても使用できます。通常、 AWS IoT デバイスは X.509 証明書を使用し、モバイルアプリケーションは Amazon Cognito ID を使用します。ウェブアプリケーションとデスクトップアプリケーションでは、IAM またはフェデレーテッドアイデンティティを使用します。 AWS CLI コマンド

は IAM を使用します。IAM ID の詳細については、の ID とアクセスの管理 AWS IoT を参照してください。

X.509 クライアント証明書

X.509 証明書 AWS IoT は、クライアントとデバイスの接続を認証する機能を提供します。クライアント証明書は、クライアントが通信 AWS IoT する前に に登録する必要があります AWS IoT。クライアント証明書は、同じリージョン内の AWS アカウント間でデバイスを移動しやすく AWS リージョン するために、同じ 内の複数の AWS アカウントに登録できます。詳細については、「マルチアカウント登録で複数の で X.509 AWS アカウントクライアント証明書を使用する」を参照してください。

証明書が失効した場合などにきめ細かくクライアント管理アクションを有効にできるように、各デバイスまたはクライアントで一意の証明書を指定することをお勧めします。証明書失効時にオペレーションを円滑に行うために、デバイスまたはクライアントが証明書のローテーションおよび置換に対応している必要もあります。

X.509 証明書を使用して複数のデバイスをサポートする方法については、デバイスプロビジョニングを参照して、 AWS IoT がサポートするさまざまな証明書管理およびプロビジョニングオプションを確認してください。

AWS IoT は、次のタイプの X.509 クライアント証明書をサポートしています。

- によって生成された X.509 証明書 AWS IoT
- に登録されている CA によって署名された X.509 証明書 AWS IoT。
- AWS IoTに登録されていない CA によって署名された X.509 証明書。

このセクションでは、 AWS IoTで X.509 証明書を管理する方法について説明します。 AWS IoT コンソールまたは を使用して AWS CLI 、次の証明書オペレーションを実行できます。

- AWS IoT クライアント証明書を作成する
- 独自のクライアント証明書を作成する
- クライアント証明書の登録
- クライアント証明書を有効または無効する
- クライアント証明書の取り消し

これらのオペレーションを実行する AWS CLI コマンドの詳細については、「AWS IoT CLI リファレンス」を参照してください。

X.509 クライアント証明書の使用

X.509 証明書は、クライアントとデバイスの接続を認証します AWS IoT。X.509 証明書には、他の識別および認証メカニズムに比べて、いくつかの利点があります。X.509 証明書では、非対称キーをデバイスで使用できます。例えば、プライベートキーをデバイス上の安全なストレージに書き込むと、デバイスから機密の暗号化情報が持ち出されることがないようにすることができます。X.509 証明書により、ユーザー名とパスワード、ベアラートークンなどの他のスキーマよりも、強力なクライアント認証が可能になります。これは、プライベートキーがデバイスから持ち出されることがないためです。

AWS IoT は、TLS プロトコルのクライアント認証モードを使用してクライアント証明書を認証します。TLS サポートは、多くのプログラミング言語とオペレーティングシステムに対応しており、データの暗号化に一般に使用されます。TLS クライアント認証では、 は X.509 クライアント証明書を AWS IoT リクエストし、証明書のステータスと を証明書のレジストリ AWS アカウント と照合します。次に、証明書に含まれるパブリックキーに対応するプライベートキーの所有権の証明をクライアントに要求します。 では、クライアントがサーバー名表示 (SNI) 拡張機能を Transport Layer Security (TLS) プロトコルに送信 AWS IoT する必要があります。SNI 拡張機能の設定の詳細については、「のトランスポートセキュリティ AWS IoT Core」を参照してください。

AWS IoT コアへの安全で一貫性のあるクライアント接続を容易にするには、X.509 クライアント証明書に次のものが必要です。

- AWS IoT Core に登録されています。詳細については、「クライアント証明書の登録」を参照してください。
- ステータスは ACTIVE 状態である。詳細については、「クライアント証明書を有効または無効する」を参照してください。
- 証明書の有効期限にまだ達していない。

Amazon Root CA を使用するクライアント証明書を作成し、別の認証機関 (CA) によって署名された独自のクライアント証明書を使用できます。 AWS IoT コンソールを使用して Amazon ルート CA を使用する証明書を作成する方法の詳細については、「」を参照してくださいAWS IoT クライアント証明書を作成する。独自の X.509 証明書の使用の詳細については、「独自のクライアント証明書を作成する」を参照してください。

CA 証明書によって署名された証明書の有効期限が切れる日付と時刻は、証明書の作成時に設定されます。によって生成された X.509 証明書は、2049 年 12 月 31 日午前 0 時に AWS IoT 期限切れになります (2049-12-31T23:59:59Z)。

AWS IoT Device Defender は、一般的な IoT セキュリティのベストプラクティスをサポートする AWS アカウント および デバイスで監査を実行できます。これには、CA または Amazon Root CA によって署名された X.509 証明書の有効期限の管理が含まれます。証明書の有効期限の管理の詳細については、「デバイス証明書の有効期限切れ」および「CA 証明書の有効期限切れ」を参照してください。

公式 AWS IoT ブログでは、「 を使用して IoT デバイス証明書のローテーションを管理する方法」で、デバイス証明書のローテーション AWS IoTとセキュリティのベストプラクティスの管理について詳しく説明します。

マルチアカウント登録で複数の で X.509 AWS アカウントクライアント証明書を使用する

マルチアカウント登録により、同じリージョン内、または異なるリージョン内の AWS アカウント間で、デバイスの移動が可能になります。実稼働前のアカウントでデバイスを登録、テスト、設定した後、実稼働アカウントで同じデバイスとデバイス証明書を登録して使用することができます。クライアント証明書をデバイスに登録することも、登録されている CA なしでデバイス証明書を登録することもできます AWS IoT。詳細については、「登録していない CA によって署名したクライアント証明書を登録する (CLI)」を参照してください。

> **Note**
>
> マルチアカウント登録に使用される証明書は、`iot:Data-ATS`、`iot:Data` (レガシー)、`iot:Jobs`、および `iot:CredentialProvider` エンドポイントタイプでサポートされています。 AWS IoT デバイスエンドポイントの詳細については、「」を参照してくださいAWS IoT デバイスデータとサービスエンドポイント。

マルチアカウント登録を使用するデバイスは、Server Name Indication (SNI) 拡張機能を Transport Layer Security (TLS) プロトコルに送信し、 `host_name`フィールドで完全なエンドポイントアドレスを指定する必要があります AWS IoT。 は、 でエンドポイントアドレス AWS IoT を使用して接続を正しい AWS IoT アカウントに`host_name`ルーティングします。`host_name` で有効なエンドポイントアドレスを送信しなかった既存デバイスは、引き続き動作しますが、この情報を必要とする機能を使用することはできません。SNI 拡張の詳細と、 `host_name` フィールドのエンドポイントアドレスを特定する方法については、「のトランスポートセキュリティ AWS IoT Core」を参照してください。

マルチアカウント登録を使用するには

1. CA を指定してデバイス証明書を登録することができます。署名 CA を SNI_ONLY モードで複数のアカウントに登録し、その CA を使用して同じクライアント証明書を複数のアカウントに登録できます。詳細については、「SNI_ONLY モードでの CA 証明書の登録 (CLI) – 推奨」を参照してください。
2. デバイス証明書は CA を指定せずに登録することができます。「登録していない CA によって署名したクライアント証明書を登録する (CLI)」を参照してください。CA の登録はオプションです。デバイス証明書に署名した CA を登録する必要はありません AWS IoT。

でサポートされている証明書署名アルゴリズム AWS IoT

AWS IoT は、次の証明書署名アルゴリズムをサポートしています。

- SHA256WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- SHA256WITHRSAANDMGF1 (RSASSA-PSS)
- SHA384WITHRSAANDMGF1 (RSASSA-PSS)
- SHA512WITHRSAANDMGF1 (RSASSA-PSS)
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256
- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

証明書認証およびセキュリティの詳細については、「デバイス証明書のキー品質」を参照してください。

> **Note**
>
> 証明書署名リクエスト (CSR) にはパブリックキーを含める必要があります。キーは、少なくとも 2,048 ビット長の RSA キー、または NIST P-256、NIST P-384、NIST P-521 カーブの ECC キーとすることができます。詳細については、「AWS IoT API リファレンスガイド」の「CreateCertificateFromCsr API」を参照してください。

でサポートされているキーアルゴリズム AWS IoT

次の表は、キーアルゴリズムのサポート方法を示しています。

[キーアルゴリズム]	証明書署名アルゴリズム	TLS バージョン	サポート対象? はい/いいえ
キーサイズが 2048 ビット以上の RSA	すべて	TLS 1.2 TLS 1.3	はい
ECC NIST P-256/P-384/P-521	すべて	TLS 1.2 TLS 1.3	はい
キーサイズが 2048 ビット以上の RSA-PSS	すべて	TLS 1.2	いいえ
キーサイズが 2048 ビット以上の RSA-PSS	すべて	TLS 1.3	はい

CreateCertificateFromCSR を使用して証明書を作成するには、サポートされているキーアルゴリズムを使用して CSR のパブリックキーを生成できます。RegisterCertificate または RegisterCertificateWithoutCA を使用して独自の証明書を登録するには、サポートされているキーアルゴリズムを使用して証明書のパブリックキーを生成できます。

詳細については、「セキュリティポリシー」を参照してください。

AWS IoT クライアント証明書を作成する

AWS IoT は、Amazon ルート認証局 (CA) によって署名されたクライアント証明書を提供します。

このトピックでは、Amazon ルート認証局によって署名されたクライアント証明書を作成し、証明書ファイルをダウンロードする方法について説明します。クライアント証明書ファイルを作成したら、クライアントにインストールする必要があります。

> **Note**
>
> によって提供される各 X.509 クライアント証明書は、証明書の作成時に設定した発行者とサブジェクト属性 AWS IoT を保持します。証明書の属性は、証明書が作成された後にのみイミュータブルです。

AWS IoT コンソールまたは を使用して AWS CLI 、Amazon ルート AWS IoT 認証局によって署名された証明書を作成できます。

AWS IoT 証明書を作成する (コンソール)

AWS IoT コンソールを使用して AWS IoT 証明書を作成するには

1. にサインイン AWS マネジメントコンソール し、 AWS IoT コンソールを開きます。
2. ナビゲーションペインで、[セキュリティ] を選択し、[証明書] を選択してから [作成] を選択します。
3. [1-Click 証明書作成 (推奨)] - [証明書の作成] を選択します。
4. [証明書が作成されました] ページで、モノ、パブリックキー、およびプライベートキーのクライアント証明書ファイルを安全な場所にダウンロードします。によって生成されたこれらの証明書 AWS IoT は、 AWS IoT サービスでのみ使用できます。

   Amazon Root CA 証明書ファイルも必要であれば、このページにダウンロードできるページへのリンクがあります。
5. これで、クライアント証明書が作成され、 AWS IoTに登録されました。証明書をクライアントで使用する前に、証明書をアクティブ化する必要があります。

   クライアント証明書を今すぐ有効化するには、[有効化] を選択します。今すぐ証明書を有効化しない場合、証明書を後で有効化する方法について、「クライアント証明書のアクティブ化 (コンソール)」を参照してください。
6. 証明書にポリシーをアタッチする場合は、[Attach a policy] (ポリシーをアタッチ) を選択します。

   ポリシーを今すぐアタッチしない場合は、[Done] (完了) を選択して完了します。ポリシーは後でアタッチできます。

手順が完了したら、証明書ファイルをクライアントにインストールします。

AWS IoT 証明書を作成する (CLI)

AWS CLI は、Amazon ルート認証局によって署名されたクライアント証明書を作成するための create-keys-and-certificate コマンドを提供します。ただし、このコマンドでは Amazon ルート CA 証明書ファイルはダウンロードされません。Amazon ルート CA 証明書ファイルは、 からダウンロードできますサーバー認証用の CA 証明書

このコマンドは、プライベートキー、パブリックキー、および X.509 証明書ファイルを作成し、証明書を に登録してアクティブ化します 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
```

証明書の作成と登録時に証明書を有効化しない場合、このコマンドはプライベートキー、パブリックキー、および X.509 証明書ファイルを作成し、証明書を登録しますが、有効化しません。クライアント証明書のアクティブ化 (CLI) では、証明書を後で有効化する方法について説明します。

```
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
```

証明書ファイルをクライアントにインストールします。

独自のクライアント証明書を作成する

AWS IoT は、ルートまたは中間認証機関 (CA) によって署名されたクライアント証明書をサポートします。 は CA 証明書 AWS IoT を使用して証明書の所有権を検証します。Amazon の CA ではない CA によって署名されたデバイス証明書を使用するには、デバイス証明書の所有権を検証 AWS IoT できるように、CA の証明書を に登録する必要があります。

AWS IoT は、独自の証明書 (BYOC) を持ち込むための複数の方法をサポートしています。

- まず、クライアント証明書の署名に使用する CA を登録し、次に個々のクライアント証明書を登録します。デバイスまたはクライアントをクライアント証明書に初めて接続するときに登録する場合は AWS IoT (Just-in-Timeプロビジョニングとも呼ばれます)、署名 CA を AWS IoT に登録し、自動登録を有効にする必要があります。
- 署名 CA を登録できない場合は、CA なしでクライアント証明書を登録することを選択できます。CA なしで登録されたデバイスの場合、 AWS IoTへの接続時に Server Name Indication (SNI) を提示する必要があります。

> **Note**
>
> CA を使用してクライアント証明書を登録するには、階層内の他の CA ではなく AWS IoT、署名 CAsを登録する必要があります。

> **Note**
>
> CA 証明書は、リージョン内の 1 つのアカウントでのみ DEFAULT モードで登録できます。CA 証明書は、リージョン内の複数のアカウントで SNI_ONLY モードで登録できます。

X.509 証明書を使用して複数のデバイスをサポートする方法の詳細については、「デバイスプロビジョニング」を参照して、 AWS IoT がサポートするさまざまな証明書管理およびプロビジョニングオプションを確認してください。

トピック

- CA 証明書の管理
- CA 証明書を使用してクライアント証明書を作成する

CA 証明書の管理

このセクションでは、独自の認証局 (CA) 証明書を管理するための一般的なタスクについて説明します。

が認識 AWS IoT しない CA によって署名されたクライアント証明書を使用している場合 AWS IoT は、認証機関 (CA) を に登録できます。

クライアントが最初に接続 AWS IoT するときにクライアント証明書を に自動的に登録する場合は、クライアント証明書に署名した CA を登録する必要があります AWS IoT。それ以外の場合は、クライアント証明書に署名した CA 証明書を登録する必要はありません。

> **Note**
>
> CA 証明書は、リージョン内の 1 つのアカウントでのみ DEFAULT モードで登録できます。CA 証明書は、リージョン内の複数のアカウントで SNI_ONLY モードで登録できます。

トピック:

- CA 証明書を作成する
- CA 証明書の登録
- CA 証明書の非アクティブ化

CA 証明書を作成する

CA 証明書がない場合は、OpenSSL v1.1.1i ツールを使用して作成できます。

> **Note**
>
> この手順は AWS IoT コンソールでは実行できません。

OpenSSL v1.1.1i ツールを使用して CA 証明書を作成するには

1. キーペアを生成します。

```
openssl genrsa -out root_CA_key_filename.key 2048
```

2. キーペアのプライベートキーを使用して CA 証明書を生成します。

```
openssl req -x509 -new -nodes \
    -key root_CA_key_filename.key \
    -sha256 -days 1024 \
    -out root_CA_cert_filename.pem
```

CA 証明書の登録

これらの手順では、Amazon の CA ではない認証機関 (CA) から証明書を登録する方法について説明します。 は CA 証明書 AWS IoT Core を使用して証明書の所有権を検証します。Amazon の CA ではない CA によって署名されたデバイス証明書を使用するには、デバイス証明書の所有権を検証 AWS IoT Core できるように、CA 証明書を に登録する必要があります。

CA 証明書の登録 (コンソール)

> **Note**
>
> コンソールで CA 証明書を登録するには、[Register CA certificate] (CA 証明書の登録) をコンソールで開始します。CA はマルチアカウントモードで登録でき、検証証明書を提供したり、プライベートキーにアクセスしたりする必要はありません。マルチアカウントモードでは、同一 AWS リージョン内の複数の AWS アカウント で CA を登録することができます。検証証明書と CA のプライベートキーの所有権の証明を提供することで、CA をシングルアカウントモードで登録できます。

CA 証明書の登録 (CLI)

CA 証明書を DEFAULT モードまたは SNI_ONLY モードで登録できます。CA は、1 つの AWS アカウント に 1 つずつ、 DEFAULT モードで登録できます AWS リージョン。CA は、 AWS アカウント 同じ 内の複数の によって SNI_ONLY モードで登録できます AWS リージョン。CA 証明書の詳細については、「certificateMode」を参照してください。

> **Note**
>
> CA を SNI_ONLY モードで登録することをお勧めします。検証証明書やプライベートキーへのアクセスを提供する必要はなく、同じ AWS アカウント 内の複数の で CA を登録できます AWS リージョン。

SNI_ONLY モードでの CA 証明書の登録 (CLI) – 推奨

前提条件

続行する前に、コンピュータで次のものが揃っていることを確認してください。

- ルート CA の証明書ファイル (次の例では *root_CA_cert_filename.pem* と表記します)
- OpenSSL v1.1.1i またはそれ以降

を使用して CA 証明書を **SNI_ONLY** モードで登録するには AWS CLI

1. CA 証明書を に登録します AWS IoT。register-ca-certificate コマンドを使用して、CA 証明書ファイル名を入力します。詳細については、「AWS CLI コマンドリファレンス」の「register-ca-certificate」を参照してください。

   ```
   aws iot register-ca-certificate \
       --ca-certificate file://root_CA_cert_filename.pem \
       --certificate-mode SNI_ONLY
   ```

   成功した場合、このコマンドは *certificateId* を返します。

2. この時点で、CA 証明書は に登録されています AWS IoT が、非アクティブです。CA 証明書によって署名されたクライアント証明書を登録できるようにするには、CA 証明書をアクティブにする必要があります。

   このステップにより CA 証明書がアクティブ化されます。

   CA 証明書をアクティブ化するには、次のように update-certificate コマンドを使用します。詳細については、「AWS CLI コマンドリファレンス」の「update-certificate」を参照してください。

   ```
   aws iot update-ca-certificate \
       --certificate-id certificateId \
       --new-status ACTIVE
   ```

CA 証明書のステータスを表示するには、describe-ca-certificate コマンドを使用します。詳細については、「AWS CLI コマンドリファレンス」の「describe-ca-certificate」を参照してください。

DEFAULT モードでの CA 証明書の登録 (CLI)

前提条件

続行する前に、コンピュータで次のものが揃っていることを確認してください。

- ルート CA の証明書ファイル (次の例では *root_CA_cert_filename.pem* と表記します)
- ルート CA 証明書のプライベートキーファイル (次の例では *root_CA_key_filename.key* と表記します)
- OpenSSL v1.1.1i またはそれ以降

を使用して CA 証明書を **DEFAULT** モードで登録するには AWS CLI

1. から登録コードを取得するには AWS IoT、 を使用しますget-registration-code。プライベートキー検証証明書の Common Name として使用するために、返された registrationCode を保存します。詳細については、「AWS CLI コマンドリファレンス」の「get-registration-code」を参照してください。

   ```
   aws iot get-registration-code
   ```

2. プライベートキー検証証明書のキーペアを生成します。

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

3. プライベートキー検証証明書の証明書署名リクエスト (CSR) を作成します。証明書の Common Name フィールドに、registrationCode によって返された get-registration-code を設定します。

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   証明書に関するいくつかの情報 (例: Common Name) の入力を求められます。

   ```
   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. CSR を使用して、プライベートキー検証証明書を作成します。

```
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 証明書を に登録します AWS IoT。次のように、CA 証明書のファイル名とプライベートキー検証証明書のファイル名を register-ca-certificate コマンドに渡します。詳細については、「AWS CLI コマンドリファレンス」の「register-ca-certificate」を参照してください。

```
aws iot register-ca-certificate \
    --ca-certificate file://root_CA_cert_filename.pem \
    --verification-cert file://verification_cert_filename.pem
```

   成功した場合、このコマンドは *certificateId* を返します。

6. この時点で、CA 証明書は に登録されています AWS IoT が、アクティブではありません。CA 証明書によって署名されたクライアント証明書を登録できるようにするには、CA 証明書をアクティブにする必要があります。

   このステップにより CA 証明書がアクティブ化されます。

   CA 証明書をアクティブ化するには、次のように update-certificate コマンドを使用します。詳細については、「AWS CLI コマンドリファレンス」の「update-certificate」を参照してください。

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

CA 証明書のステータスを表示するには、describe-ca-certificate コマンドを使用します。詳細については、「AWS CLI コマンドリファレンス」の「describe-ca-certificate」を参照してください。

コンソールに CA 証明書を登録するための CA 検証証明書を作成する

> **Note**
>
> この手順は、 AWS IoT コンソールから CA 証明書を登録する場合にのみ使用します。AWS IoT コンソールからこの手順にアクセスしなかった場合は、コンソールの「CA 証明書の登録」で CA 証明書の登録プロセスを開始します。

続行する前に、同じコンピュータで次のものが揃っていることを確認してください :

- ルート CA の証明書ファイル (次の例では *root_CA_cert_filename.pem* と表記します)
- ルート CA 証明書のプライベートキーファイル (次の例では *root_CA_key_filename.key* と表記します)
- OpenSSL v1.1.1i またはそれ以降

コマンドラインインターフェイスを使用して CA 検証証明書を作成し、コンソールに CA 証明書を登録するには

1. *verification_cert_key_filename.key* を、作成する検証証明書のキーファイルのファイル名 (例えば **verification_cert.key** など) で置き換えます。次にこのコマンドを実行して、プライベートキー検証証明書のキーペアを生成します。

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

2. *verification_cert_key_filename.key* をステップ 1 で作成したキーファイルの名前で置き換えます。

   verification_cert_csr_filename.csr を作成する証明書署名リクエスト (CSR) ファイルの名前で置き換えます。例えば、**verification_cert.csr**。

   このコマンドを実行して、CSR ファイルを作成します。

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   このコマンドでは、後で説明する追加情報の入力を求めるプロンプトが表示されます。

3. AWS IoT コンソールの検証証明書コンテナで、登録コードをコピーします。
4. openssl コマンドが入力を求める情報を以下の例に示します。Common Name フィールド以外では、独自の値を入力することも、空白のままにすることもできます。

   Common Name フィールドに、前のステップでコピーした登録コードを貼り付けます。

```
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 []:
```

   終了すると、コマンドによって CSR ファイルが作成されます。
5. *verification_cert_csr_filename.csr* を前のステップで使用した *verification_cert_csr_filename.csr* で置き換えます。

   root_CA_cert_filename.pem を登録する CA 証明書のファイル名で置き換えます。

   root_CA_key_filename.key を CA 証明書のプライベートキーファイルのファイル名で置き換えます。

   verification_cert_filename.pem を作成する検証証明書のファイル名で置き換えます。例えば、**verification_cert.pem**。

```
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 コマンドが完了したら、コンソールに戻ったときにこれらのファイルを使用できるようになります。

   - CA 証明書ファイル (前のコマンドで使用した *root_CA_cert_filename.pem*)
   - 前のステップで作成した検証証明書 (前のコマンドで使用した *verification_cert_filename.pem*)

CA 証明書の非アクティブ化

認証機関 (CA) 証明書の自動クライアント証明書登録が有効になっている場合、 は CA 証明書 AWS IoT をチェックして CA が であることを確認しますACTIVE。CA 証明書が の場合INACTIVE、クライアント証明書の登録は許可 AWS IoT されません。

CA 証明書を INACTIVE に設定すると、CA によって発行された新しいクライアント証明書が自動的に登録されなくなります。

> **Note**
>
> 疑わしい CA 証明書によって署名された登録済みのクライアント証明書は、明示的にそれぞれのクライアント証明書が取り消されるまで、引き続き使用されます。

CA 証明書の非アクティブ化 (コンソール)

AWS IoT コンソールを使用して CA 証明書を無効にするには

1. にサインイン AWS マネジメントコンソール し、 AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性]、[CA] の順に選択します。
3. 認証局の一覧で、非アクティブ化する認証局を探し、省略記号のアイコンを選択してオプションメニューを開きます。
4. オプションメニューで、[無効化] を選択します。

認証局は、リストに [非アクティブ] と表示されます。

> **Note**
>
> AWS IoT コンソールには、非アクティブ化した CA によって署名された証明書を一覧表示する方法はありません。これらの証明書を一覧表示する AWS CLI オプションについては、CA 証明書の非アクティブ化 (CLI) を参照してください。

CA 証明書の非アクティブ化 (CLI)

AWS CLI は、CA 証明書を非アクティブ化する update-ca-certificate コマンドを提供します。

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

list-certificates-by-ca コマンドを使用して、指定した CA によって署名されたすべての登録済みクライアント証明書のリストを取得します。指定した CA 証明書によって署名されたクライアント証明書ごとに、update-certificate コマンドを使用します。これによってクライアント証明書が失効し、使用できなくなります。

CA 証明書のステータスを表示するには、describe-ca-certificate コマンドを使用します。

CA 証明書を使用してクライアント証明書を作成する

独自の認証局 (CA) を使用してクライアント証明書を作成できます。クライアント証明書は、使用 AWS IoT 前に に登録する必要があります。クライアント証明書の登録オプションの詳細については、 を参照してくださいクライアント証明書の登録

クライアント証明書の作成 (CLI)

> **Note**
>
> この手順は AWS IoT コンソールでは実行できません。

を使用してクライアント証明書を作成するには AWS CLI

1. キーペアを生成します。

   ```
   openssl genrsa -out device_cert_key_filename.key 2048
   ```

2. クライアント証明書の CSR を作成します。

```
openssl req -new \
    -key device_cert_key_filename.key \
    -out device_cert_csr_filename.csr
```

以下に示しているように、いくつかの情報の入力を求められます。

```
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. CSR からクライアント証明書を作成します。

```
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
```

この時点で、クライアント証明書は作成されていますが、まだ登録されていません AWS IoT。クライアント証明書を登録する方法とタイミングについては、 を参照してくださいクライアント証明書の登録

クライアント証明書の登録

クライアントと 間の通信を有効にする AWS IoT には、クライアント証明書を に登録する必要があります AWS IoT。各クライアント証明書を手動で登録することも、クライアントが AWS IoT 初めてに接続するときに自動的に登録するようにクライアント証明書を設定することもできます。

クライアントとデバイスが最初に接続するときにクライアント証明書を登録する場合は、CA 証明書の登録 を使用するリージョンで AWS IoT を使用してクライアント証明書に署名する必要があります。Amazon ルート CA は自動的に に登録されます AWS IoT。

クライアント証明書は、 AWS アカウント および リージョンで共有できます。これらのトピックの手順は、クライアント証明書を使用する各アカウントおよびリージョンで実行する必要があります。あるアカウントまたはリージョンでのクライアント証明書の登録は、別のアカウントでは自動的に認識されません。

> **Note**
>
> AWS IoT への接続に Transport Layer Security (TLS) プロトコルを使用するクライアントは、TLS に対する Server Name Indication (SNI) 拡張機能をサポートしている必要があります。詳細については、「のトランスポートセキュリティ AWS IoT Core」を参照してください。

トピック

- クライアント証明書を手動で登録する
- クライアントがジャ AWS IoT just-in-time登録 (JITR) に接続するときにクライアント証明書を登録する

クライアント証明書を手動で登録する

AWS IoT コンソールと を使用して、クライアント証明書を手動で登録できます AWS CLI。

使用する登録手順は、証明書が AWS アカウントおよびリージョンによって共有されるかどうかによって異なります。あるアカウントまたはリージョンでのクライアント証明書の登録は、別のアカウントでは自動的に認識されません。

このトピックの手順は、クライアント証明書を使用する各アカウントおよびリージョンで実行する必要があります。クライアント証明書は、 AWS アカウントおよび リージョンで共有できます。

登録した CA によって署名したクライアント証明書を登録する (コンソール)

> **Note**
>
> この手順を実行する前に、クライアント証明書の .pem ファイルがあり、クライアント証明書が登録 AWS IoTした CA によって署名されていることを確認してください。

コンソール AWS IoT を使用して既存の証明書を に登録するには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. ナビゲーションペインの [Manage] (管理) セクションで、[Security] (セキュリティ)、[Certificates] (証明書) の順に選択します。
3. [Certificates] (証明書) ページの [Certificates] (証明書) ダイアログボックスで、[Add certificate] (証明書を追加)、[Register certificates] (証明書を登録) の順に選択します。
4. [Register certificate] (証明書を登録) ページの [Certificates to upload] (アップロードする証明書) ダイアログボックスで、次の操作を行います。
   - [CA is registered with AWS IoT] (CA が IoT に登録されています) を選択します。
   - [Choose a CA certificate] (CA 証明書を選択) から [Certification authority] (認証機関) を選択します。
     - AWS IoTに登録されていない新しい認証機関を登録するには、[Register a new CA] (新しい CA を登録) を選択します。
     - 認証機関として Amazon ルート認証機関を使用する場合は、[Choose a CA certificate] (CA 証明書を選択) を空白のままにします。
   - アップロードして登録する証明書を最大 10 個選択します AWS IoT。
     - 「AWS IoT クライアント証明書を作成する」と「CA 証明書を使用してクライアント証明書を作成する」で作成した証明書ファイルを使用します。
   - [Activate] (有効化) または [Deactivate] (無効化) を選択します。[Deactive] (無効化) を選択すると、証明書の登録後に証明書を有効化する方法についての説明が クライアント証明書を有効または無効する に表示されます。
   - [登録] を選択します。

[Certificates] (証明書) ページの [Certificates] (証明書) ダイアログボックスに、登録した証明書が表示されます。

登録していない CA によって署名したクライアント証明書を登録する (コンソール)

> **Note**
>
> この手順を実行する前に、クライアント証明書の.pem ファイルがあることを確認してください。

コンソール AWS IoT を使用して既存の証明書を に登録するには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択してから [作成] を選択します。
3. [Create a certificate] (証明書を作成する) で、[Use my certificate] (お持ちの証明書を使用する) エントリを見つけて、[Get started] (今すぐ始める) を選択します。
4. [Select a CA] (CA の選択) で、[Next] (次へ) を選択します。
5. [Register existing device certificates] (既存のデバイス証明書を登録する) で [Select certificates] (証明書の選択) を選択し、登録する証明書ファイルを最大 10 個選択します。
6. ファイルダイアログボックスを閉じた後、クライアント証明書を登録するときにアクティブ化するか取り消すかどうかを選択します。

   登録時に証明書をアクティブ化しない場合、後で証明書をアクティブ化する方法については、「クライアント証明書のアクティブ化 (コンソール)」で説明します。

   登録時に証明書が失効した場合、後でアクティブ化することはできません。

   登録する証明書ファイルを選択し、登録後に実行するアクションを選択したら、[Register certificates(証明書の登録)] を選択します。

正常に登録されたクライアント証明書が証明書の一覧に表示されます。

登録した CA によって署名したクライアント証明書を登録する (CLI)

> **Note**
>
> この手順を実行する前に、認証局 (CA) .pem とクライアント証明書の.pem ファイルがあることを確認してください。クライアント証明書は、登録 AWS IoTした認証機関 (CA) によって署名されている必要があります。

register-certificate コマンドを使用して、クライアント証明書を登録します (アクティブ化しません)。

```
aws iot register-certificate \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

クライアント証明書は に登録されていますが AWS IoT、まだアクティブではありません。後で有効化する方法については、「クライアント証明書のアクティブ化 (CLI)」を参照してください。

このコマンドを使用して、クライアント証明書を登録するときにクライアント証明書をアクティブ化することもできます。

```
aws iot register-certificate \
    --set-as-active \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

証明書をアクティブ化して に接続できるようにする方法の詳細については AWS IoT、「」を参照してください。 クライアント証明書を有効または無効する

登録していない CA によって署名したクライアント証明書を登録する (CLI)

> **Note**
>
> この手順を実行する前に、証明書の.pem ファイルがあることを確認してください。

register-certificate-without-ca コマンドを使用して、クライアント証明書を登録します (アクティブ化しません)。

```
aws iot register-certificate-without-ca \
    --certificate-pem file://device_cert_filename.pem
```

クライアント証明書は に登録されていますが AWS IoT、まだアクティブではありません。後で有効化する方法については、「クライアント証明書のアクティブ化 (CLI)」を参照してください。

このコマンドを使用して、クライアント証明書を登録するときにクライアント証明書をアクティブ化することもできます。

```
aws iot register-certificate-without-ca \
    --status ACTIVE \
    --certificate-pem file://device_cert_filename.pem
```

証明書をアクティブ化して に接続できるようにする方法の詳細については AWS IoT、「」を参照してくださいクライアント証明書を有効または無効する。

クライアントがジャ AWS IoT just-in-time登録 (JITR) に接続するときにクライアント証明書を登録する

CA 証明書を設定して、クライアントが初めて接続したときに AWS IoT に自動的に登録するために署名したクライアント証明書を有効にすることができます AWS IoT。

クライアントが AWS IoT 初めて に接続するときにクライアント証明書を登録するには、CA 証明書の自動登録を有効にし、必要な証明書を提供するようにクライアントによる最初の接続を設定する必要があります。

自動登録をサポートするための CA 証明書の設定 (コンソール)

AWS IoT コンソールを使用してクライアント証明書の自動登録をサポートするように CA 証明書を設定するには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性]、[CA] の順に選択します。
3. 認証局のリストで、自動登録を有効にする認証局を探し、省略記号アイコンを使用してオプションメニューを開きます。
4. オプションメニューで、[自動登録を有効にする] を選択します。

> **Note**
>
> 自動登録ステータスは、認証局の一覧に表示されません。認証局の自動登録ステータスを表示するには、認証局の [詳細] ページを開く必要があります。

自動登録をサポートするための CA 証明書の設定 (CLI)

CA 証明書を に登録済みの場合は AWS IoT、 update-ca-certificate コマンドを使用して CA 証明書`autoRegistrationStatus`を に設定します`ENABLE`。

```
aws iot update-ca-certificate \
--certificate-id caCertificateId \
--new-auto-registration-status ENABLE
```

CA 証明書を登録するときに `autoRegistrationStatus` を有効にする場合は、register-ca-certificate コマンドを使用します。

```
aws iot register-ca-certificate \
--allow-auto-registration  \
--ca-certificate file://root_CA_cert_filename.pem \
--verification-cert file://verification_cert_filename.pem
```

CA 証明書のステータスを表示するには、describe-ca-certificate コマンドを使用します。

自動登録のためのクライアントによる最初の接続の設定します

クライアントが AWS IoT 初めて に接続しようとする場合、Transport Layer Security (TLS) ハンドシェイク中に CA 証明書によって署名されたクライアント証明書がクライアントに存在する必要があります。

クライアントが に接続するときは AWS IoT、「クライアント証明書の作成」または「独自のクライアント証明書の作成」で作成した AWS IoT クライアント証明書を使用します。 は CA 証明書を登録済み CA 証明書として AWS IoT 認識し、クライアント証明書を登録して、そのステータスを に設定します`PENDING_ACTIVATION`。これは、クライアント証明書が自動的に登録され、アクティベーションの待機中という事です。クライアント証明書が `ACTIVE` 状態になると、 AWS IoTへの接続に使用できるようになります。クライアント証明書の有効化については、「クライアント証明書を有効または無効する」を参照してください。

> **Note**
>
> デバイスの最初の接続で信頼チェーン全体を送信しなくても、ジャ AWS IoT Core just-in-time登録 (JITR) 機能を使用してデバイスをプロビジョニングできます AWS IoT Core。CA 証明書の提示はオプションですが、[Server Name Indication (SNI)](サーバーネームインディケーション (SNI)) エクステンションを接続する時にそれらを送信するために、そのデバイスが必要です。

が証明書 AWS IoT を自動的に登録するか、クライアントが証明書を PENDING_ACTIVATIONステータスで提示すると、 は次の MQTT トピックにメッセージ AWS IoT を発行します。

`$aws/events/certificates/registered/`*caCertificateId*

ここで、*caCertificateId* は、クライアント証明書を発行した CA 認定の ID です。

このトピックに発行されたメッセージには、以下の構造があります。

```
{
        "certificateId": "certificateId",
        "caCertificateId": "caCertificateId",
        "timestamp": timestamp,
        "certificateStatus": "PENDING_ACTIVATION",
        "awsAccountId": "awsAccountId",
        "certificateRegistrationTimestamp": "certificateRegistrationTimestamp"
}
```

このトピックをリッスンし、いくつかのアクションを実行するルールを作成できます。クライアント証明書が証明書失効リスト (CRL) に含まれていないことを確認し、証明書を有効にし、ポリシーを作成して、証明書にアタッチする、Lambda ルールを作成することをお勧めします。ポリシーによって、クライアントがアクセスできるリソースが決まります。作成するポリシーで接続デバイスからのクライアント ID が必要な場合は、ルールの clientid() 関数を使用してクライアント ID を取得できます。ルールの例として、以下のようなものがあります。

```
SELECT *,
   clientid() as clientid
from $aws/events/certificates/registered/caCertificateId
```

この例では、ルールは JITR トピック `$aws/events/certificates/registered/`*caCertificateID* をサブスクライブし、clientid() 関数を使用してクライアント

ID を取得します。次に、ルールは JITR ペイロードにクライアント ID を追加します。ルールの clientid() 関数の詳細については、「clientid()」を参照してください。

`$aws/events/certificates/registered/caCertificateID` トピックでリッスンして該当アクションを実行する Lambda ルールの作成方法については、「Just-in-Time Registration of Client Certificates on AWS IoT」を参照してください。

クライアント証明書の自動登録中にエラーまたは例外が発生した場合、 はイベントまたはメッセージを CloudWatch Logs のログ AWS IoT に送信します。アカウントのログ設定の詳細については、Amazon CloudWatch のドキュメントを参照してください。

クライアント証明書の管理

AWS IoT には、クライアント証明書を管理する機能が用意されています。

このトピックの内容

- クライアント証明書を有効または無効する
- クライアント証明書へのモノまたはポリシーのアタッチ
- クライアント証明書の取り消し
- 別のアカウントに証明書を移転する

クライアント証明書を有効または無効する

AWS IoT は、接続を認証するときにクライアント証明書がアクティブであることを確認します。

クライアント証明書をアクティブ化せずに作成および登録できるため、使用するときまで使用されないようにすることができます。アクティブなクライアント証明書を無効にして、一時的に無効にすることもできます。最後に、クライアント証明書を取り消して、今後の使用を防ぐことができます。

クライアント証明書のアクティブ化 (コンソール)

AWS IoT コンソールを使用してクライアント証明書をアクティブ化するには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。
3. 証明書の一覧で、アクティブ化する証明書を探し、省略記号アイコンを使用してオプションメニューを開きます。
4. オプションメニューで、[有効化] を選択します。

証明書の一覧で、証明書が [アクティブ] と表示されます。

クライアント証明書の非アクティブ化 (コンソール)

AWS IoT コンソールを使用してクライアント証明書を無効にするには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。
3. 証明書の一覧で、非アクティブ化する証明書を探し、省略記号アイコンを使用してオプションメニューを開きます。
4. オプションメニューで、[無効化] を選択します。

証明書のリストでは、証明書が [非アクティブ] と表示されます。

クライアント証明書のアクティブ化 (CLI)

AWS CLI は、証明書をアクティブ化するための update-certificate コマンドを提供します。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

コマンドが成功した場合、証明書の状態は ACTIVE になります。describe-certificate を実行して、証明書のステータスを確認します。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

クライアント証明書の非アクティブ化 (CLI)

AWS CLI は、証明書を非アクティブ化する update-certificate コマンドを提供します。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

コマンドが成功した場合、証明書の状態は INACTIVE になります。describe-certificate を実行して、証明書のステータスを確認します。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

クライアント証明書へのモノまたはポリシーのアタッチ

AWS IoT モノとは別の証明書を作成して登録する場合、 AWS IoT オペレーションを許可するポリシーはなく、 AWS IoT モノのオブジェクトにも関連付けられません。このセクションでは、登録済みの証明書にこれらの関連付けを追加する方法について説明します。

> **Important**
>
> これらの手順を完了するには、証明書にアタッチするモノまたはポリシーを事前に作成しておく必要があります。

証明書は、接続 AWS IoT できるようにデバイスを で認証します。モノのリソースに証明書をアタッチすると、デバイスとモノのリソースとの関係が (証明書を介して) 確立されます。デバイスがメッセージに接続して発行することを許可するなどの AWS IoT アクションを実行することを許可するには、適切なポリシーをデバイスの証明書にアタッチする必要があります。

クライアント証明書へのモノのアタッチ (コンソール)

この手順を完了するには、モノオブジェクトの名前が必要です。

登録済み証明書にモノオブジェクトをアタッチするには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。
3. 証明書のリストで、ポリシーをアタッチする証明書を見つけ、省略記号アイコンを選択して証明書のオプションメニューを開き、[モノをアタッチ] を選択します。
4. ポップアップで、証明書にアタッチするモノの名前を見つけ、そのチェックボックスをオンにして [Attach] (アタッチ) を選択します。

これで、モノオブジェクトが証明書の詳細ページのモノリストに表示されます。

クライアント証明書へのポリシーのアタッチ (コンソール)

この手順を完了するには、ポリシーオブジェクトの名前が必要です。

登録済みの証明書にポリシーオブジェクトをアタッチするには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。
3. 証明書のリストで、ポリシーをアタッチする証明書を見つけ、省略記号アイコンを選択して証明書のオプションメニューを開き、[ポリシーをアタッチ] を選択します。
4. ポップアップで、証明書にアタッチするポリシーの名前を見つけ、そのチェックボックスをオンにして [Attach] (アタッチ) を選択します。

これで、ポリシーオブジェクトが証明書の詳細ページのポリシーリストに表示されます。

クライアント証明書へのモノのアタッチ (CLI)

AWS CLI は、モノのオブジェクトを証明書にアタッチする attach-thing-principal コマンドを提供します。

```
aws iot attach-thing-principal \
    --principal certificateArn \
    --thing-name thingName
```

クライアント証明書へのポリシーのアタッチ (CLI)

AWS CLI は、ポリシーオブジェクトを証明書にアタッチする attach-policy コマンドを提供します。

```
aws iot attach-policy \
    --target certificateArn \
    --policy-name policyName
```

クライアント証明書の取り消し

登録済みのクライアント証明書で不審なアクティビティを検出した場合は、再使用できないように無効にすることができます。

> **Note**
>
> 証明書が取り消されると、そのステータスを変更することはできません。つまり、証明書のステータスを `Active` や他のステータスに変更することはできません。

クライアント証明書の取り消し (コンソール)

AWS IoT コンソールを使用してクライアント証明書を取り消すには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。
3. 証明書の一覧で、取り消す証明書を探し、省略記号アイコンを使用してオプションメニューを開きます。
4. オプションメニューで [取り消し] を選択します。

証明書が正常に取り消された場合、証明書の一覧に [失効済み] と表示されます。

クライアント証明書 (CLI) の取り消し

AWS CLI は、証明書を取り消すupdate-certificateコマンドを提供します。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status REVOKED
```

コマンドが成功した場合、証明書の状態は REVOKED になります。describe-certificate を実行して、証明書のステータスを確認します。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

別のアカウントに証明書を移転する

1 つの に属する X.509 証明書は、別の に転送 AWS アカウント できます AWS アカウント。

X.509 証明書を 1 つの から別の に転送 AWS アカウント するには

1. the section called “証明書の転送を開始する”

   転送を開始する前に、証明書を非アクティブにし、すべてのポリシーとモノからデタッチする必要があります。
2. the section called “証明書の転送を承諾または拒否する”

   受信側のアカウントは、転送された証明書を明示的に承諾または拒否する必要があります。受信側のアカウントが証明書を承諾したら、使用前に証明書を有効化する必要があります。

3. the section called “証明書の転送をキャンセルする”

   証明書が承諾されない場合、元のアカウントは転送をキャンセルできます。

証明書の転送を開始する

AWS IoT コンソールまたは AWS アカウント を使用して、証明書の別の への転送を開始できます AWS CLI。

証明書の転送を開始する (コンソール)

この手順を完了するには、転送する証明書の ID が必要です。

転送する証明書を持つアカウントからこの手順を実行します。

別の への証明書の転送を開始するには AWS アカウント

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。

   ステータスが [Active] (アクティブ) または [Inactive] (無効) である、転送する証明書を選択し、その詳細ページを開きます。
3. 証明書の [詳細] ページの [アクション] メニューで、[無効化] オプションが使用可能な場合は、[無効化] オプションを選択して証明書を無効化します。
4. 証明書の [Details] (詳細) ページの左側のメニューで、[Policies] (ポリシー) を選択します。
5. 証明書の [Policies] (ポリシー) ページで、証明書にポリシーがアタッチされている場合は、ポリシーのオプションメニューを開いて [Detach] (デタッチ) を選択し、各ポリシーをデタッチします。

   続行する前に、証明書にポリシーがアタッチされていてはなりません。
6. 証明書の [Policies] (ポリシー) ページの左側のメニューで、[Things] (モノ) を選択します。
7. 証明書の [Things] (モノ) ページで、証明書にモノがアタッチされている場合は、モノのオプションメニューを開いて [Detach] (デタッチ) を選択し、各モノをデタッチします。

   続行する前に、証明書にモノがアタッチされていてはなりません。
8. 証明書の [Things] (モノ) ページの左側のメニューで、[Details] (詳細) を選択します。
9. 証明書の [詳細] ページの [アクション] メニューで、[転送を開始する] を選択して [転送を開始する] ダイアログボックスを開きます。

10. 転送の開始ダイアログボックスに、証明書を受信するアカウントの AWS アカウント 番号とオプションのショートメッセージを入力します。
11. [Start transfer] (転送を開始する) を選択して、証明書を転送します。

転送の成功または失敗を示すメッセージがコンソールに表示されます。転送が開始された場合、証明書のステータスは [Transferred] (転送済み) に更新されます。

証明書の転送を開始する (CLI)

この手順を完了するには、転送する証明書の *certificateId* と *certificateArn* が必要です。

転送する証明書を持つアカウントからこの手順を実行します。

別の AWS アカウントへの証明書の移管を開始するには

1. update-certificate コマンドを使用して証明書を非アクティブ化します。

```
aws iot update-certificate --certificate-id certificateId --new-status INACTIVE
```

2. すべてのポリシーをデタッチします。
   1. list-attached-policies コマンドを使用して、証明書にアタッチされているポリシーを一覧表示します。

```
aws iot list-attached-policies --target certificateArn
```

   2. アタッチされたポリシーごとに、detach-policy コマンドを使用してポリシーをデタッチします。

```
aws iot detach-policy --target certificateArn --policy-name policy-name
```

3. すべてのモノをデタッチします。
   1. list-principal-things コマンドを使用して、証明書にアタッチされているモノを一覧表示します。

```
aws iot list-principal-things --principal certificateArn
```

   2. アタッチされた各モノに対して detach-thing-principal コマンドを実行してモノをデタッチします。

```
aws iot detach-thing-principal --principal certificateArn --thing-name thing-name
```

4. transfer-certificate コマンドを使用して、証明書の転送を開始します。

```
aws iot transfer-certificate --certificate-id certificateId --target-aws-
account account-id
```

証明書の転送を承諾または拒否する

AWS IoT コンソールまたは AWS アカウント を使用して、別の から転送された AWS アカウント 証明書を承認または拒否できます AWS CLI。

証明書の転送を承諾または拒否する (コンソール)

この手順を完了するには、アカウントに転送された証明書の ID が必要です。

転送された証明書を受信するアカウントからこの手順を実行します。

に転送された証明書を承諾または拒否するには AWS アカウント

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。

   承認または拒否するステータスが [Pending Transfer] (転送を保留中) の証明書を選択し、その詳細ページを開きます。
3. 証明書の [Details] (詳細) ページの [Actions] (アクション) メニューで、次の操作を実行します。

   - 証明書を受け入れるには、[Accept transfer] (転送を許可する) を選択します。
   - 証明書を承諾しない場合は、[Reject transfer] (転送を拒否する) を選択します。

証明書の転送を承諾または拒否する (CLI)

この手順を完了するには、承認または拒否する証明書の転送の *certificateId* が必要です。

転送された証明書を受信するアカウントからこの手順を実行します。

に転送された証明書を承諾または拒否するには AWS アカウント

1. accept-certificate-transfer コマンドを使用して証明書を受け入れます。

```
aws iot accept-certificate-transfer --certificate-id certificateId
```

2. reject-certificate-transfer コマンドを使用して証明書を拒否します。

```
aws iot reject-certificate-transfer --certificate-id certificateId
```

証明書の転送をキャンセルする

AWS IoT コンソールまたは AWS CLIを使用して、承認される前に証明書の転送をキャンセルできます。

証明書の転送をキャンセルする (コンソール)

この手順を完了するには、キャンセルする証明書の転送の ID が必要です。

証明書の転送を開始したアカウントから、この手順を実行します。

証明書の転送をキャンセルするには

1. AWS マネジメントコンソールにサインインし、AWS IoT コンソールを開きます。
2. 左のナビゲーションペインで、[安全性] を選択し、[証明書] を選択します。

   転送をキャンセルする [Transferred] (転送済み) のステータスの証明書を選択し、そのオプションメニューを開きます。
3. 証明書のオプションメニューで、[Revoke transfer] (転送を取り消す) オプションを選択して、証明書の転送をキャンセルします。

> **Important**
>
> [Revoke transfer] (転送を取り消す) オプションと [Revoke] (取り消し) オプションを間違えないように注意してください。
> [Revoke transfer] (転送を取り消す) オプションは証明書の転送をキャンセルしますが、[Revoke] (取り消し) オプションは証明書を AWS IoTで不可逆的に使用できなくします。

証明書の転送をキャンセルする (CLI)

この手順を完了するには、キャンセルする証明書の転送の *certificateId* が必要です。

証明書の転送を開始したアカウントから、この手順を実行します。

cancel-certificate-transfer コマンドを使用して、証明書の転送をキャンセルします。

```
aws iot cancel-certificate-transfer --certificate-id certificateId
```

カスタムクライアント証明書の検証

AWS IoT Core は、X.509 クライアント証明書のカスタムクライアント証明書検証をサポートしているため、クライアント認証の管理が向上します。この証明書の検証方法は認証前証明書チェックとも呼ばれ、独自の基準 (Lambda 関数で定義) に基づいてクライアント証明書を評価し、クライアント証明書または証明書の署名認証局 (CA) 証明書を取り消して、クライアントが AWS IoT Coreに接続できないようにします。例えば、オンライン証明書ステータスプロトコル (OCSP) または証明書失効リスト (CRL) エンドポイントをサポートする検証機関に対して証明書のステータスを検証する独自の証明書失効チェックを作成し、失効した証明書を持つクライアントへの接続を防ぐことができます。クライアント証明書の評価に使用される基準は、Lambda 関数 (認証前 Lambda とも呼ばれます) で定義されます。ドメイン設定で設定されたエンドポイントを使用し、[認証タイプ] は X.509 証明書である必要があります。さらに、クライアントは接続時に Server Name Indication (SNI) 拡張機能を提供する必要があります AWS IoT Core。

> **Note**
>
> この機能は、 AWS GovCloud (US) リージョンではサポートされていません。

カスタムクライアント証明書の検証を実行するプロセスには、次のステップが含まれます。

- ステップ 1: X.509 クライアント証明書を に登録する AWS IoT Core
- ステップ 2: Lambda 関数を作成する
- ステップ 3: AWS IoT に Lambda 関数の呼び出しを許可する
- ステップ 4: ドメインの認証設定を設定する

ステップ 1: X.509 クライアント証明書を に登録する AWS IoT Core

これをまだ実行していない場合は、X.509 クライアント証明書を登録してアクティブ化します AWS IoT Core。それ以外の場合は、次のステップに進みます。

クライアント証明書を に登録してアクティブ化するには AWS IoT Core、次の手順に従います。

1. でクライアント証明書を直接作成 AWS IoTする場合。これらのクライアント証明書は、 に自動的に登録されます AWS IoT Core。
2. 独自のクライアント証明書を作成する場合は、以下の手順に従って登録します AWS IoT Core。
3. クライアント証明書をアクティブ化するには、以下の手順に従います。

ステップ 2: Lambda 関数を作成する

証明書の検証を実行し、設定されたエンドポイントのクライアント接続試行ごとに呼び出される Lambda 関数を作成する必要があります。この Lambda 関数を作成するときは、「最初の Lambda 関数を作成する」の一般的なガイダンスに従ってください。さらに、Lambda 関数が、次のように予想されるリクエストとレスポンスの形式に準拠していることを確認します。

Lambda 関数のイベントの例

```
{
 "connectionMetadata": {
  "id": "string"
 },
 "principalId": "string",
 "serverName": "string",
 "clientCertificateChain": [
  "string",
  "string"
 ]
}
```

`connectionMetadata`

クライアントから AWS IoT Coreへの接続に関連するメタデータまたは追加情報。

`principalId`

TLS 接続のクライアントに関連付けられたプリンシパル識別子。

`serverName`

[サーバー名表示 (SNI)] ホスト名文字列。 AWS IoT Core では、デバイスが SNI 拡張を Transport Layer Security (TLS) に送信し、host_name フィールドに完全なエンドポイントアドレスを指定する必要があります。

`clientCertificateChain`

クライアントの X.509 証明書チェーンを表す文字列の配列。

Lambda 関数のレスポンスの例

```
{
 "isAuthenticated": "boolean"
}
```

`isAuthenticated`

リクエストが認証されるかどうかを示すブール値。

> **Note**
>
> Lambda レスポンスでは、さらなる認証および承認に進むには `isAuthenticated` が `true` である必要があります。それ以外の場合は、IoT クライアント証明書を無効にし、X.509 クライアント証明書によるカスタム認証をブロックして、さらなる認証および承認を行うことができます。

ステップ 3: AWS IoT に Lambda 関数の呼び出しを許可する

Lambda 関数を作成したら、add-permission CLI コマンドを使用して、関数を呼び出す AWS IoT アクセス許可を に付与する必要があります。この Lambda 関数は、設定されたエンドポイントへの接続試行ごとに呼び出されることに注意してください。詳細については、「Lambda 関数 AWS IoT の呼び出しを許可する」を参照してください。

ステップ 4: ドメインの認証設定を設定する

次のセクションでは、 AWS CLIを使用してカスタムドメインの認証設定を設定する方法について説明します。

ドメインのクライアント証明書設定を設定する (CLI)

ドメイン設定がない場合は、create-domain-configuration CLI コマンドを使用してドメイン設定を作成します。既にドメイン設定がある場合は、update-domain-configuration CLI コマンドを使用してドメインのクライアント証明書設定を更新します。前のステップで作成した Lambda 関数の ARN を追加する必要があります。

```
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`

ドメイン構成の名前。

`authentication-type`

ドメイン設定の認証タイプ。詳細については、「認証タイプの選択」を参照してください。

`application-protocol`

デバイスが AWS IoT Coreとの通信に使用するアプリケーションプロトコル。詳細については、「アプリケーションプロトコルの選択」を参照してください。

`client-certificate-config`

ドメインのクライアント認証設定を指定するオブジェクト。

`clientCertificateCallbackArn`

新しい接続が確立されるときに が TLS レイヤーで AWS IoT 呼び出す Lambda 関数の Amazon リソースネーム (ARN)。カスタムクライアント証明書の検証を実行するようにクライアント認証をカスタマイズするには、前のステップで作成した Lambda 関数の ARN を追加する必要があります。

詳細については、「AWS IoT API リファレンス」の「CreateDomainConfiguration」と「UpdateDomainConfiguration」を参照してください。ドメイン設定の詳細については、「ドメイン設定」を参照してください。

IAM ユーザー、グループ、ロール

IAM ユーザー、グループ、ロールは、 AWSで ID と認証を管理するための標準的なメカニズムです。これらを使用して、 AWS SDK と を使用して HTTP AWS IoT インターフェイスに接続できます AWS CLI。

IAM ロールでは AWS IoT 、 がユーザーに代わってアカウント内の他の AWS リソースにアクセスすることもできます。たとえば、デバイスがその状態を DynamoDB テーブルに発行する場合、IAM ロールは AWS IoT が Amazon DynamoDB とやり取りできるようにします。詳細については、「IAM ロール」を参照してください。

HTTP 経由のメッセージブローカー接続の場合、 は署名バージョン 4 の署名プロセスを使用して ユーザー、グループ、ロールを AWS IoT 認証します。詳細については、AWS 「 API リクエストの署名」を参照してください。

で AWS 署名バージョン 4 を使用する場合 AWS IoT、クライアントは TLS 実装で以下をサポートする必要があります。

- TLS 1.2
- SHA-256 RSA 証明書の署名の検証
- サポートされているいずれかの TLS 暗号スイート

詳細については、の ID とアクセスの管理 AWS IoT を参照してください。

Amazon Cognito ID

Amazon Cognito Identity を使用すると、モバイルアプリケーションやウェブアプリケーションで使用するために、権限が制限された一時的な AWS 認証情報を作成できます。Amazon Cognito ID を使用する場合、ユーザーに対して一意の ID を作成し、Login with Amazon、Facebook、Google などの ID プロバイダーで認証するアイデンティティプールを作成します。独自のデベロッパーが認証した ID で Amazon Cognito ID を使用することもできます。詳細については、Amazon Cognito ID を参照してください。

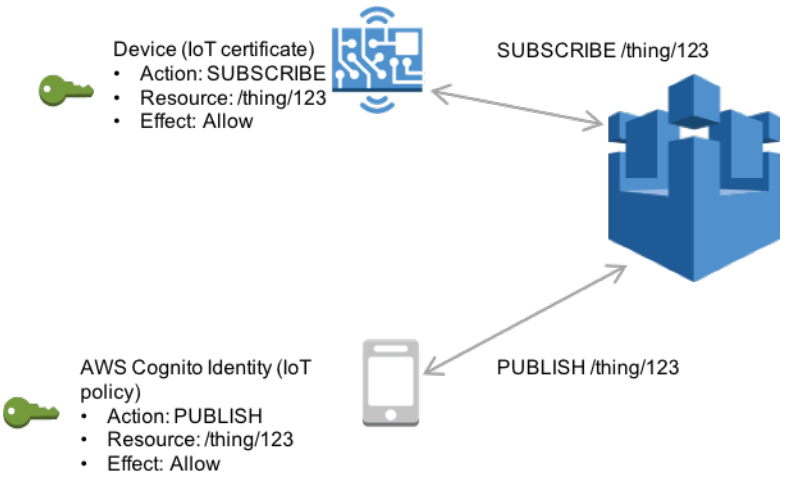
Amazon Cognito ID を使用するには、IAM ロールに関連付けられた Amazon Cognito アイデンティティプールを定義します。IAM ロールは、 サービスの呼び出し AWS などの AWS リソースにアクセスするためのアクセス許可を ID プールから ID に付与する IAM ポリシーに関連付けられています。

Amazon Cognito ID では、認証されていない ID と認証された ID が作成されます。認証されていない ID は、サインインせずにアプリを使用するモバイルアプリケーションまたはウェブアプリケー

ションのゲストユーザーに使用されます。認証されていないユーザーには、ID プールに関連付けられた IAM ポリシーで指定されているアクセス許可のみが付与されます。

認証された ID を使用する場合は、ID プールにアタッチされた IAM ポリシーに加えて、Amazon Cognito ID に AWS IoT ポリシーをアタッチする必要があります。 AWS IoT ポリシーをアタッチするには、AttachPolicy API を使用し、 AWS IoT アプリケーションの個々のユーザーにアクセス許可を付与します。 AWS IoT ポリシーを使用して、特定の顧客とそのデバイスにきめ細かなアクセス許可を割り当てることができます。

認証されたユーザーと未認証のユーザーは、異なる ID タイプです。Amazon Cognito ID に AWS IoT ポリシーをアタッチしない場合、認証されたユーザーは での認可に失敗 AWS IoT し、 AWS IoT リソースとアクションにアクセスできません。Amazon Cognito ID のポリシー作成の詳細については、パブリッシュ/サブスクライブポリシーの例 および Amazon Cognito ID を使用した承認 を参照してください。



カスタム認証と認可

AWS IoT Core では、独自のクライアントの認証と認可を管理できるように、カスタムオーソライザーを定義できます。これは、 が AWS IoT Core ネイティブにサポートしている認証メカニズム以外の認証メカニズムを使用する必要がある場合に便利です。(ネイティブでサポートされているメカニズムの詳細については、the section called “クライアント認証” を参照してください)。

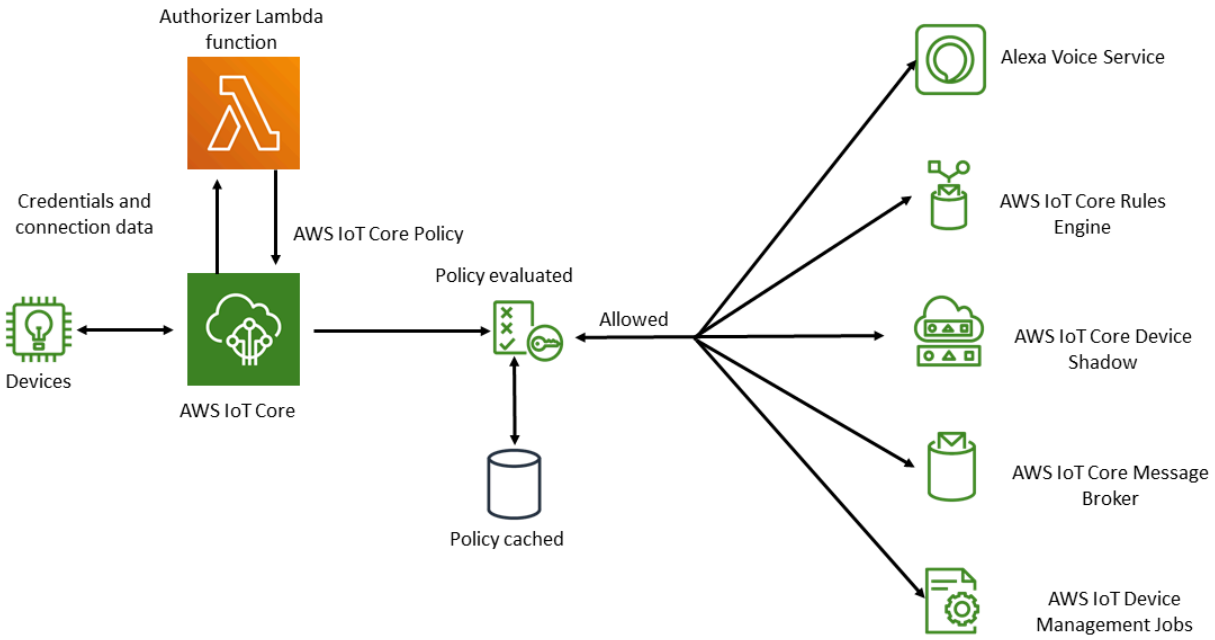
たとえば、 フィールドの既存のデバイスを に移行 AWS IoT Core し、これらのデバイスがカスタムベアラートークンまたは MQTT ユーザー名とパスワードを使用して認証 する場合、新しい ID をプロビジョニング AWS IoT Core しなくても に移行できます。カスタム認証は、 が AWS IoT Core サポートする任意の通信プロトコルで使用できます。 AWS IoT Core がサポートするプロトコルの詳細については、「the section called “デバイス通信プロトコル”」を参照してください。

トピック

- カスタム認証ワークフローについて
- カスタムオーソライザー (CLI) の作成と管理
- X.509 クライアント証明書を使用したカスタム認証
- カスタム認証 AWS IoT Core を使用した への接続
- オーソライザーのトラブルシューティング

カスタム認証ワークフローについて

カスタム認証を使用すると、オーソライザーリソースを使用してクライアントを認証および認可する方法を定義できます。 各オーソライザーには、カスタマー管理の Lambda 関数への参照、デバイスの認証情報を検証するためのオプションのパブリックキー、および追加の設定情報が含まれています。次の図は、 AWS IoT Coreでのカスタム認証の認可ワークフローを示しています。



AWS IoT Core カスタム認証と認可ワークフロー

次のリストでは、カスタム認証および認可ワークフローの各ステップについて説明します。

1. デバイスは、サポートされている のいずれかを使用して、顧客の AWS IoT Core データエンドポイントに接続しますthe section called “デバイス通信プロトコル”。デバイスは、リクエストのヘッダーフィールドまたはクエリパラメータ (HTTP Publish または MQTT over WebSocket プロトコルの場合)、または MQTT CONNECT メッセージのユーザー名とパスワードフィールド (MQTT および MQTT over WebSockets プロトコルの場合) で認証情報を渡します。
2. AWS IoT Core は、次の 2 つの条件のいずれかをチェックします。
   - 受信リクエストはオーソライザーを指定します。
   - リクエストを受信する AWS IoT Core データエンドポイントには、デフォルトのオーソライザーが設定されています。

   がこれらのいずれかの方法でオーソライザー AWS IoT Core を検出すると、 AWS IoT Core はオーソライザーに関連付けられた Lambda 関数をトリガーします。
3. (オプション) トークン署名を有効にしている場合、 は Lambda 関数をトリガーする前に、オーソライザーに保存されている パブリックキーを使用してリクエスト署名 AWS IoT Core を検証します。検証が失敗した場合、 AWS IoT Core は Lambda 関数を呼び出さずにリクエストを停止します。
4. Lambda 関数は、リクエスト内の認証情報と接続メタデータを受け取り、認証の可否を判断します。
5. Lambda 関数は、認証決定の結果と、接続で許可されるアクションを指定する AWS IoT Core ポリシードキュメントを返します。Lambda 関数は、Lambda 関数を呼び出してリクエスト内の認証情報 AWS IoT Core を再検証する頻度を指定する情報も返します。
6. AWS IoT Core は、Lambda 関数から受信したポリシーに対して接続のアクティビティを評価します。
7. 接続が確立され、カスタムオーソライザー Lambda が最初に呼び出された後、MQTT オペレーションなしでアイドル状態の接続では、次の呼び出しが最大 5 分間遅延する可能性があります。その後、それ以降の呼び出しはカスタムオーソライザー Lambda の更新間隔に従います。このアプローチにより、 の Lambda 同時実行数の制限を超える可能性のある過剰な呼び出しを防ぐことができます AWS アカウント。

スケーリングに関する考慮事項

Lambda 関数はオーソライザーの認証と認可を処理するため、関数は Lambda の料金とサービスの制限 (同時実行率など) の対象となります。Lambda の料金の詳細については、「Lambda の料金」を参照してください。Lambda 関数の応答の `refreshAfterInSeconds` および `disconnectAfterInSeconds` パラメータを調整することで、Lambda 関数の負荷を管理できま

す。Lambda 関数のレスポンスの内容の詳細については、the section called “Lambda 関数の定義” を参照してください。

> **Note**
>
> 署名を有効にしておくと、認識されないクライアントによる Lambda の過度なトリガーを防ぐことができます。オーソライザーで署名を無効にする前に、これを考慮してください。

> **Note**
>
> カスタムオーソライザーの Lambda 関数のタイムアウト制限は 5 秒です。

カスタムオーソライザー (CLI) の作成と管理

AWS IoT Core は、カスタムオーソライザーを使用してカスタム認証および認可スキームを実装します。カスタムオーソライザーは、特定の要件に基づいてルールとポリシーを柔軟に定義および実装できる AWS IoT Core リソースです。ステップバイステップの手順でカスタムオーソライザーを作成するには、「チュートリアル: AWS IoT Coreのカスタムオーソライザーの作成」を参照してください。

各オーソライザーは、次のコンポーネントで構成されています。

- 名前: オーソライザーを識別する一意のユーザー定義文字列。
- Lambda 関数 ARN: 認可および認証ロジックを実装する Lambda 関数の Amazon リソースネーム (ARN)。
- トークンキー名: 署名の検証を実行するために、HTTP ヘッダー、クエリパラメータ、または MQTT CONNECT ユーザー名からトークンを抽出するために使用されるキー名。オーソライザーで署名が有効になっている場合、この値は必須です。
- 署名無効フラグ (オプション): 認証情報の署名要件を無効にするかどうかを指定するブール値。これは、MQTT ユーザー名とパスワードを使用する認証スキームなど、認証情報への署名が意味をなさないシナリオで役立ちます。デフォルト値は `false` であるため、署名はデフォルトで有効になっています。
- トークン署名パブリックキー: AWS IoT Core がトークン署名を検証するために使用するパブリックキー。最小長は 2,048 ビットです。オーソライザーで署名が有効になっている場合、この値は必須です。

Lambda では、Lambda 関数の実行回数と、関数内のコードの実行にかかった時間に対する請求が発生します。Lambda の料金の詳細については、「Lambda の料金」を参照してください。Lambda 関数の作成の詳細については、Lambda デベロッパーガイドを参照してください。

> **Note**
>
> 署名を有効にしておくと、認識されないクライアントによる Lambda の過度なトリガーを防ぐことができます。オーソライザーで署名を無効にする前に、これを考慮してください。

> **Note**
>
> カスタムオーソライザーの Lambda 関数のタイムアウト制限は 5 秒です。

この章の内容:

- Lambda 関数の定義
- オーソライザーを作成する
- Lambda 関数を呼び出す AWS IoT ことを に許可する
- オーソライザーのテスト
- カスタムオーソライザーの管理

Lambda 関数の定義

がオーソライザーを AWS IoT Core 呼び出すと、次の JSON オブジェクトを含むイベントで、オーソライザーに関連付けられた Lambda がトリガーされます。サンプルの JSON オブジェクトには、可能なフィールドがすべて含まれています。接続リクエストに関係のないフィールドは含まれていません。

```
{
    "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 関数は、この情報を使用して着信接続を認証し、接続で許可されるアクションを決定する必要があります。関数は、次の値を含む応答を送信する必要があります。

- `isAuthenticated`: リクエストが認証されるかどうかを示すブール値。
- `principalId`: カスタム認可リクエストによって送信されるトークンの識別子として機能する英数字の文字列。値は、1 ~ 128 文字以内の英数字の文字列で、正規表現 (regex) パターン (`[a-zA-Z0-9]){1,128}` と一致する必要があります。英数字以外の特殊文字は、 `principalId`では使用できません AWS IoT Core。英数字以外の特殊文字が で許可されている場合は、他の AWS サービスのドキュメントを参照してくださいprincipalId。
- `policyDocuments`: JSON 形式の AWS IoT Core ポリシードキュメントのリスト AWS IoT Core ポリシーの作成の詳細については、「」を参照してくださいthe section called “AWS IoT Core ポリシー”。ポリシードキュメントの最大数は 10 個です。各ポリシードキュメントでは最大 2,048 文字を使用できます。
- `disconnectAfterInSeconds`: AWS IoT Core ゲートウェイへの接続の最大期間 (秒単位) を指定する整数。最小値は 300 秒で、最大値は 86,400 秒です。デフォルト値は 86,400 です。

> **Note**
>
> (Lambda 関数によって返される) `disconnectAfterInSeconds` の値は、接続が確立されると設定されます。この値は、後続のポリシー更新 Lambda 呼び出し中に変更することはできません。

- `refreshAfterInSeconds`: ポリシーの更新の間隔を指定する整数。この時間間隔が経過すると、 AWS IoT Core が Lambda 関数を呼び出してポリシーの更新を許可します。最小値は 300 秒で、最大値は 86,400 秒です。

次の JSON オブジェクトには、Lambda 関数が送信できる応答の例が含まれています。

```
{ "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` 値には有効な AWS IoT Core ポリシードキュメントが含まれている必要があります。 AWS IoT Core ポリシーの詳細については、「」を参照してくださいthe section called "AWS IoT Core ポリシー"。 MQTT over TLS および MQTT over WebSockets 接続では、 は、`refreshAfterInSeconds`フィールドの値で指定された間隔についてこのポリシーを AWS IoT Core キャッシュします。HTTP 接続の場合、オーソライザーの設定時に選択するとキャッシュを有効にできる HTTP 持続的接続 (HTTP キープアライブまたは HTTP 接続の再利用とも呼ばれる) をデバイスで使用していない限り、認可リクエストごとに Lambda 関数が呼び出されます。この間隔中、 は Lambda 関数を再度トリガーすることなく、このキャッシュされたポリシーに対して確立された接続のアクション AWS IoT Core を承認します。カスタム認証中に障害が発生した場合、 は接続 AWS IoT Core を終了します。 AWS IoT Core また、 `disconnectAfterInSeconds`パラメータで指定された値よりも長く開いている場合は接続も終了します。

次の JavaScript には、 の値を持つ MQTT Connect メッセージでパスワードを検索`test`し、 という名前のクライアント AWS IoT Core を使用して に接続`myClientName`し、同じクライアント名を含むトピックに発行するアクセス許可を付与するポリシーを返すサンプル Node.js Lambda 関数が含ま

れています。想定されるパスワードが見つからない場合、これらの 2 つのアクションを拒否するポリシーを返します。

```
// 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;
}
```

上の Lambda 関数は、MQTT Connect メッセージの `test` で想定されるパスワードを受け取ると、次の JSON を返します。`password` プロパティと `principalId` プロパティの値は、MQTT Connect メッセージの値です。

```
{
  "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
```

```
}
```

オーソライザーを作成する

CreateAuthorizer API を使用してオーソライザーを作成できます。次の例は、このコマンドを示しています。

```
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 パラメータを使用して、オーソライザーの呼び出しごとに署名の検証をオプトアウトできます。必要がない限り、署名を無効にしないことを強くお勧めします。署名の検証により、不明なデバイスからの Lambda 関数の過剰な呼び出しを防ぐことができます。オーソライザーを作成した後で、signing-disabled ステータスを更新することはできません。この動作を変更するには、signing-disabled パラメータの異なる値を持つ別のカスタムオーソライザーを作成する必要があります。

署名を無効にしている場合、tokenKeyName パラメータと tokenSigningPublicKeys パラメータの値はオプションです。署名が有効になっている場合、これらは必須の値です。

Lambda 関数とカスタムオーソライザーを作成したら、ユーザーに代わって関数を呼び出すアクセス許可を AWS IoT Core サービスに明示的に付与する必要があります。これを行う には、次のコマンドを使用します。

> **Note**
>
> デフォルトの IoT エンドポイントは、Lambda 関数でのカスタムオーソライザーの使用をサポートしていない場合があります。その場合は、ドメイン設定を使用して新しいエンドポイントを定義し、そのエンドポイントをカスタムオーソライザーに指定できます。

```
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 関数を呼び出す AWS IoT ことを に許可する

このセクションでは、作成したカスタムオーソライザーリソースに、Lambda 関数を実行するためのアクセス許可を付与します。アクセス許可を付与するには、add-permission CLI コマンドを使用できます。

を使用して Lambda 関数にアクセス許可を付与する AWS CLI

1. 値を挿入したら、次のコマンドを入力します。statement-id 値は一意でなければならないことに注意してください。*Id-1234* を正確な値に置き換えます。そうしないと、ResourceConflictException エラーが発生する可能性があります。

   ```
   aws lambda add-permission  \
   --function-name "custom-auth-function" \
   --principal "iot.amazonaws.com" \
   --action "lambda:InvokeFunction" \
   --statement-id "Id-1234" \
   --source-arn authorizerArn
   ```

2. コマンドが成功すると、この例のようなアクセス許可ステートメントが返されます。次のセクションに進んで、カスタムオーソライザーをテストできます。

   ```
   {
       "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\"}}}"
   }
   ```

   コマンドが成功しない場合は、この例のようなエラーが返されます。続行する前に、エラーを確認して修正する必要があります。

   ```
   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
```

オーソライザーのテスト

TestInvokeAuthorizer API を使用して、オーソライザーの呼び出しと戻り値をテストできます。この API を使用すると、プロトコルメタデータを指定し、オーソライザーで署名の検証をテストできます。

次のタブは、 を使用してオーソライザーを AWS CLI テストする方法を示しています。

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` パラメータの値は署名付きトークンです。この値を取得する方法については、「the section called “トークンへの署名”」を参照してください。

オーソライザーがユーザー名とパスワードを受け取る場合、`--mqtt-context` パラメータを使用してこの情報を渡すことができます。次のタブは、`TestInvokeAuthorizer` API を使用して、ユーザー名、パスワード、およびクライアント名を含む JSON オブジェクトをカスタムオーソライザーに送信する方法を示しています。

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"}'
```

パスワードは base64 でエンコードされている必要があります。次の例は、Unix 系の環境でパスワードをエンコードする方法を示しています。

```
echo -n PASSWORD | base64
```

カスタムオーソライザーの管理

次の API を使用して、オーソライザーを管理できます。

- ListAuthorizers: アカウント内のすべてのオーソライザーを表示します。
- DescribeAuthorizer: 指定されたオーソライザーのプロパティを表示します。これらの値には、作成日、最終更新日、およびその他の属性が含まれます。
- SetDefaultAuthorizer: AWS IoT Core データエンドポイントのデフォルトのオーソライザーを指定します。デバイスが AWS IoT Core 認証情報を渡さず、オーソライザーを指定しない場合、 はこのオーソライザー AWS IoT Core を使用します。 AWS IoT Core 認証情報の使用の詳細については、「」を参照してくださいthe section called “クライアント認証”。
- UpdateAuthorizer: 指定されたオーソライザーのステータス、トークンキー名、またはパブリックキーを変更します。
- DeleteAuthorizer: 指定されたオーソライザーを削除します。

> **Note**
>
> オーソライザーの署名要件を更新することはできません。これは、署名を必要とする既存のオーソライザーでの署名を無効にすることはできないことを意味します。また、署名を必要としない既存のオーソライザーで署名を要求することもできません。

X.509 クライアント証明書を使用したカスタム認証

デバイスを に接続する場合 AWS IoT Core、複数の認証タイプを使用できます。X.509 クライアント証明書を使用すると、クライアントとデバイス接続を認証したり、カスタムオーソライザーを定義して独自のクライアント認証と承認ロジックを管理したりできます。このトピックでは、X.509 クライアント証明書でカスタム認証を使用する方法について説明します。

X.509 証明書でカスタム認証を使用すると、X.509 証明書を使用してデバイスを既に認証していて、追加の検証とカスタム認証を実行したい場合に役立ちます。例えば、X.509 クライアント証明書にシリアル番号などのデバイスのデータを保存した場合、 AWS IoT Core が X.509 クライアント証明書を認証した後、カスタムオーソライザーを使用して、証明書の CommonName フィールドに保存されている情報に基づいて特定のデバイスを識別できます。X.509 証明書でカスタム認証を使用すると、デバイスを に接続する際のデバイスセキュリティ管理を強化 AWS IoT Core し、認証ロジックと認可ロジックをより柔軟に管理できます。 は、MQTT プロトコルと HTTPS プロトコルの両方で動作する X.509 証明書とカスタムオーソライザー認証タイプを使用した X.509 証明書でのカスタム認証 AWS IoT Core をサポートします。 AWS IoT Core デバイスエンドポイントがサポートする認証タイプとアプリケーションプロトコルの詳細については、「デバイス通信プロトコル」を参照してください。

> **Note**
>
> X.509 クライアント証明書によるカスタム認証は、 AWS GovCloud (US) リージョンではサポートされていません。

> **Important**
>
> [ドメイン設定] を使用して作成されたエンドポイントを使用する必要があります。さらに、クライアントは接続時に Server Name Indication (SNI) 拡張機能を提供する必要があります AWS IoT Core。

X.509 クライアント証明書を使用したカスタム認証を使用してデバイスを認証するプロセスは、次のステップで構成されます。

- ステップ 1: X.509 クライアント証明書を に登録する AWS IoT Core
- ステップ 2: Lambda 関数を作成する
- ステップ 3: カスタムオーソライザーを作成する
- ステップ 4: ドメイン設定で認証タイプとアプリケーションプロトコルを設定する

ステップ 1: X.509 クライアント証明書を に登録する AWS IoT Core

これをまだ実行していない場合は、X.509 クライアント証明書を登録してアクティブ化します AWS IoT Core。それ以外の場合は、次のステップに進みます。

クライアント証明書を に登録してアクティブ化するには AWS IoT Core、次の手順に従います。

1. でクライアント証明書を直接作成 AWS IoTする場合。これらのクライアント証明書は、 に自動的に登録されます AWS IoT Core。
2. 独自のクライアント証明書を作成する場合は、以下の手順に従って登録します AWS IoT Core。
3. クライアント証明書をアクティブ化するには、以下の手順に従います。

ステップ 2: Lambda 関数を作成する

AWS IoT Core はカスタムオーソライザーを使用してカスタム認証および認可スキームを実装します。カスタムオーソライザーは、デバイスが認証されているかどうか、およびデバイスが実行できるオペレーションを決定する Lambda 関数に関連付けられます。デバイスが に接続すると AWS IoT Core、 はオーソライザー名と関連する Lambda 関数を含むオーソライザーの詳細 AWS IoT Core を取得し、Lambda 関数を呼び出します。Lambda 関数は、デバイスの X.509 クライアント証明書データを含む JSON オブジェクトを含むイベントを受信します。Lambda 関数は、このイベント JSON オブジェクトを使用して認証リクエストを評価し、実行するアクションを決定してレスポンスを送り返します。

Lambda 関数のイベントの例

次の JSON オブジェクトの例には、含めることができるすべてのフィールドが含まれています。実際の JSON オブジェクトには、特定の接続リクエストに関連するフィールドのみが含まれます。

```
{
 "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`

オーソライザーの Lambda 関数を呼び出す前に、オーソライザーで設定されたトークン署名を検証するかどうかを示すブール値。オーソライザーがトークン署名を無効にするように設定されている場合、このフィールドは false になります。

`protocols`

リクエストに期待されるプロトコルを含む配列。

`protocolData`

接続で使用されるプロトコルの情報を含むオブジェクト。認証、承認などに役立つプロトコル固有の詳細を提供します。

`tls` - このオブジェクトには、TLS (Transport Layer Security) プロトコルに関連する情報が含まれます。

- `serverName` - [サーバー名表示 (SNI)] ホスト名文字列。 AWS IoT Core では、デバイスが SNI 拡張を Transport Layer Security (TLS) に送信し、`host_name` フィールドに完全なエンドポイントアドレスを指定する必要があります。
- `x509CertificatePem` - PEM 形式の X.509 証明書。TLS 接続でのクライアント認証に使用されます。

- `principalId` - TLS 接続内のクライアントに関連付けられたプリンシパル識別子。

`mqtt` - このオブジェクトは、MQTT プロトコルに関連する情報を保持します。

- `clientId` - 文字列は、デバイスがこの値を送信するイベントにのみ含める必要があります。
- `username` - MQTT Connect パケットで指定されたユーザー名。
- `password` - MQTT Connect パケットで提供されたパスワード。

`connectionMetadata`

接続のメタデータ。

`id` - ログ記録とトラブルシューティングに使用できる接続 ID。

> **Note**
>
> このイベントでは、JSON オブジェクト、`x509CertificatePem` および `principalId` はリクエスト内の 2 つの新しいフィールドです。`principalId` の値は、`certificateId` の値と同じです。詳細については、「証明書」を参照してください。

Lambda 関数のレスポンスの例

Lambda 関数は、イベント JSON オブジェクトからの情報を使用して受信接続を認証し、接続で許可されるアクションを決定する必要があります。

次の JSON オブジェクトには、Lambda 関数が送信できるレスポンスの例が含まれています。

```
{
 "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"
```

```
    }
   ]
  }
 ]
}
```

この例では、この関数は、次の値を含むレスポンスを送信する必要があります。

`isAuthenticated`

リクエストが認証されるかどうかを示すブール値。

`principalId`

カスタム認可リクエストによって送信されるトークンの識別子として機能する英数字の文字列。値は、1 ~ 128 文字以内の英数字の文字列である必要があります。ログ内の接続を識別します。`principalId` の値は、イベント JSON オブジェクト (X.509 証明書の certificateId) の `principalId` の値と同じである必要があります。

`policyDocuments`

JSON 形式の AWS IoT Core ポリシードキュメントのリスト。値はオプションで、モノのポリシー変数と証明書のポリシー変数をサポートしています。ポリシードキュメントの最大数は 10 です。各ポリシードキュメントでは最大 2,048 文字を使用できます。クライアント証明書と Lambda 関数に複数のポリシーがアタッチされている場合、アクセス許可はすべてのポリシーのコレクションです。 AWS IoT Core ポリシーの作成の詳細については、「ポリシー」を参照してください。

`disconnectAfterInSeconds`

AWS IoT Core ゲートウェイへの接続の最大時間 (秒単位) を指定する整数。最小値は 300 秒、最大値は 86,400 秒です。`disconnectAfterInSeconds` は接続の存続期間中であり、連続するポリシー更新では更新されません。

`refreshAfterInSeconds`

ポリシーの更新の間隔を指定する整数。この間隔が経過すると、 は Lambda 関数を AWS IoT Core 呼び出してポリシーの更新を許可します。最小値は 300 秒で、最大値は 86,400 秒です。

Lambda 関数の例

次に示すのは、Node.js Lambda 関数の例です。この関数は、クライアントの X.509 証明書を調べてシリアル番号、フィンガープリント、件名などの関連情報を抽出します。抽出された情報が期待値

と一致する場合、クライアントに接続するためのアクセス許可が付与されます。このメカニズムにより、有効な証明書を持つ承認されたクライアントのみが接続を確立できます。

```
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;
}
```

前述の Lambda 関数は、予想されるシリアル、フィンガープリント、およびサブジェクトを含む証明書を受信すると、次の JSON を返します。x509CertificatePem の値は、TLS ハンドシェイクで提供されるクライアント証明書になります。詳細については、「Lambda 関数の定義」を参照してください。

```
{
 "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
}
```

ステップ 3: カスタムオーソライザーを作成する

Lambda 関数 を定義した後、カスタムオーソライザーを作成して、独自のクライアント認証および承認ロジックを管理します。ステップ 3: カスタマーオーソライザーリソースとその承認を作成するの詳細な指示に従ってください。詳細については、「オーソライザーの作成」を参照してください。

カスタムオーソライザーを作成するプロセスでは、作成後に Lambda 関数を呼び出すアクセス許可を AWS IoT に付与する必要があります。詳細な手順については、「Lambda 関数 AWS IoT の呼び出しを許可する」を参照してください。

ステップ 4: ドメイン設定で認証タイプとアプリケーションプロトコルを設定する

X.509 クライアント証明書によるカスタム認証を使用してデバイスを認証するには、ドメイン設定で認証タイプとアプリケーションプロトコルを設定し、SNI 拡張機能を送信する必要があります。authenticationType の値は CUSTOM_AUTH_X509 にする必要があり、applicationProtocol の値は SECURE_MQTT または HTTPS にすることができます。

ドメイン設定 (CLI) で認証タイプとアプリケーションプロトコルを設定する

ドメイン設定がない場合は、create-domain-configuration コマンドを使用してドメイン設定を作成します。authenticationType の値は CUSTOM_AUTH_X509 にする必要があり、applicationProtocol の値は SECURE_MQTT または HTTPS にすることができます。

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
    }'
```

ドメイン設定が既にある場合は、update-domain-configuration コマンドの更新 authenticationType と必要に応じて applicationProtocol を使用します。デフォルトのエンドポイント (iot:Data-ATS) では、認証タイプやプロトコルを変更できないことに注意してください。

```
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

ドメイン構成の名前。

authentication-type

ドメイン設定の認証タイプ。詳細については、「認証タイプの選択」を参照してください。

`application-protocol`

デバイスが AWS IoT Coreとの通信に使用するアプリケーションプロトコル。詳細については、「アプリケーションプロトコルの選択」を参照してください。

`--authorizer-config`

ドメイン設定でオーソライザー設定を指定するオブジェクト。

`defaultAuthorizerName`

ドメイン構成のオーソライザー名。

詳細については、「AWS IoT API リファレンス」の「CreateDomainConfiguration」と「UpdateDomainConfiguration」を参照してください。ドメイン設定の詳細については、「ドメイン設定」を参照してください。

カスタム認証 AWS IoT Core を使用した への接続

デバイスは、 AWS IoT Core がデバイスメッセージング用に AWS IoT Core サポートする任意のプロトコルでカスタム認証を使用して に接続できます。サポートされる通信プロトコルの詳細については、the section called “デバイス通信プロトコル” を参照してください。 オーソライザーの Lambda 関数に渡す接続データは、使用するプロトコルによって異なります。オーソライザーの Lambda 関数の作成の詳細については、「the section called “Lambda 関数の定義”」を参照してください。次のセクションでは、サポートされている各プロトコルを使用して接続して認証する方法について説明します。

HTTPS

HTTP Publish API を使用して AWS IoT Core にデータを送信するデバイスは、HTTP POST リクエストのリクエストヘッダーまたはクエリパラメータを介して認証情報を渡すことができます。デバイスは、`x-amz-customauthorizer-name` ヘッダーまたはクエリパラメータを使用して呼び出すオーソライザーを指定できます。オーソライザーでトークン署名を有効にしている場合は、リクエストヘッダーまたはクエリパラメータで *`token-key-name`* と `x-amz-customauthorizer-signature` を渡す必要があります。ブラウザ内から JavaScript を使用する場合、*`token-signature`* の値は URL エンコードする必要があります。

> **Note**
>
> HTTPS プロトコルのカスタマーオーソライザーは、発行オペレーションのみをサポートします。HTTPS プロトコルの詳細については、「the section called “デバイス通信プロトコル”」を参照してください。

次のリクエスト例は、これらのパラメータをリクエストヘッダーとクエリパラメータの両方で渡す方法を示しています。

```
//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 接続 AWS IoT Core を使用して に接続するデバイスは、MQTT メッセージの username および passwordフィールドを介して認証情報を渡すことができます。username の値には、追加の値 (トークン、署名、オーソライザー名など) をオーソライザーに渡すクエリ文字列をオプションで含めることもできます。username と password の値の代わりにトークンベースの認証スキームを使用する場合は、このクエリ文字列を使用できます。

> **Note**
>
> パスワードフィールドのデータは、 によって base64 でエンコードされます AWS IoT Core。Lambda 関数はそれをデコードする必要があります。

次の例では、トークンと署名を指定する追加のパラメータを含む username 文字列が含まれています。

```
username?x-amz-customauthorizer-name=authorizer-name&x-amz-customauthorizer-
signature=token-signature&token-key-name=token-value
```

オーソライザーを呼び出すには、MQTT とカスタム認証 AWS IoT Core を使用して に接続するデバイスがポート 443 に接続する必要があります。また、 の値を持つ Application Layer Protocol Negotiation (ALPN) TLS 拡張機能mqttと、 AWS IoT Core データエンドポイントのホスト名を持つ Server Name Indication (SNI) 拡張機能を渡す必要があります。潜在的なエラーを回避するために、x-amz-customauthorizer-signature の値は URL エンコードされている必要があります。また、x-amz-customauthorizer-name と token-key-name の値も URL エンコードすることを強くお勧めします。これらの値の詳細については、「the section called “デバイス通信プロトコル”」を参照してください。V2 AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Client は、これらの拡張の両方を設定できます。

MQTT over WebSockets

MQTT over WebSockets AWS IoT Core を使用して に接続するデバイスは、次の 2 つの方法のいずれかで認証情報を渡すことができます。

- HTTP UPGRADE リクエストのリクエストヘッダーまたはクエリパラメータを介して、WebSocket 接続を確立します。
- MQTT CONNECT メッセージの username フィールドと password フィールド経由。

MQTT 接続メッセージを介して認証情報を渡す場合、ALPN および SNI TLS 拡張が必要です。これらの拡張の詳細については、「the section called “MQTT”」を参照してください。次の例は、HTTP Upgrade リクエストを介して認証情報を渡す方法を示しています。

```
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
```

トークンへの署名

`create-authorizer` 呼び出しで使用したパブリックキーとプライベートキーのペアのプライベートキーを使用してトークンに署名する必要があります。次の例では、Unix 系のコマンドと JavaScript を使用してトークン署名を作成する方法を示します。SHA-256 ハッシュアルゴリズムを使用して、署名をエンコードします。

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')
```

オーソライザーのトラブルシューティング

このトピックでは、カスタム認証ワークフローで問題を引き起こす可能性のある一般的な問題と、それらを解決するための手順について説明します。問題を最も効果的にトラブルシューティングするには、 の CloudWatch Logs を有効に AWS IoT Core し、ログレベルを DEBUG に設定します。 AWS IoT Core コンソール (https://console.aws.amazon.com/iot/) で CloudWatch ログを有効にできます。AWS IoT Coreのログの有効化と設定の詳細については、「the section called “AWS IoT ログ記録の設定”」を参照してください。

> **Note**
>
> ログレベルを長期間 DEBUG のままにしておくと、CloudWatch は大量のログデータを保存する可能性があります。これにより、CloudWatch の料金が増加する可能性があります。リ

ソースベースのログ記録を使用して、特定のモノのグループ内のデバイスのみの詳細度を高めることを検討してください。リソースベースのログ記録の詳細については、「the section called “AWS IoT ログ記録の設定”」を参照してください。また、トラブルシューティングが完了したら、ログレベルの詳細度を引き下げます。

トラブルシューティングを開始する前に、the section called “カスタム認証ワークフローについて” でカスタム認証プロセスの概要を確認してください。これは、問題の原因を見つけるのにどこを確認すればよいかを理解するのに役立ちます。

このトピックでは、調査する次の 2 つの領域について説明します。

- オーソライザーの Lambda 関数に関連する問題。
- デバイスに関連する問題。

オーソライザーの Lambda 関数に問題がないか確認する

次の手順を実行して、デバイスの接続試行が Lambda 関数を呼び出していることを確認します。

1. オーソライザーに関連付けられている Lambda 関数を確認します。

   これを行うには、DescribeAuthorizer API を呼び出すか、 AWS IoT Core コンソールの [Secure] (安全性) セクションで目的のオーソライザーをクリックします。
2. Lambda 関数の呼び出しメトリクスを確認します。これを行うには、次の手順を実行します。

   a. AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開き、オーソライザーに関連付けられている関数を選択します。

   b. [Monitor] (監視) タブを選択し、問題に関連する時間枠のメトリクスを表示します。
3. 呼び出しが表示されない場合は、 AWS IoT Core に Lambda 関数を呼び出すアクセス許可があることを確認します。呼び出しが表示された場合は、次の手順に進みます。次の手順を実行して、Lambda 関数に必要なアクセス許可があることを確認します。

   a. AWS Lambda コンソールで 関数のアクセス許可タブを選択します。

   b. ページの下部にある [Resource-based Policy] (リソースベースのポリシー) セクションを見つけます。Lambda 関数に必要なアクセス許可がある場合、ポリシーは次の例のようになります。

```
{
  "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. このポリシーは、関数に対する アクセスInvokeFunction許可を AWS IoT Core プリンシパルに付与します。表示されない場合は、AddPermission API を使用して追加する必要があります。次の例では、 AWS CLIを使用してこの操作を行う方法を示しています。

```
aws lambda add-permission --function-name FunctionName --principal
 iot.amazonaws.com --source-arn AuthorizerARn --statement-id Id-123 --action
 "lambda:InvokeFunction"
```

4. 呼び出しが表示された場合は、エラーがないことを確認します。エラーは、Lambda 関数が AWS IoT Core が送信する接続イベントを適切に処理していないことを示している可能性があります。

   Lambda 関数でのイベントの処理については、the section called “Lambda 関数の定義” を参照してください。 AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) のテスト機

能を使用して、関数のテスト値をハードコードし、関数がイベントを正しく処理していることを確認します。

5. エラーのない呼び出しが表示されても、デバイスが接続 (またはメッセージを発行、サブスクライブ、および受信) できない場合、問題は、Lambda 関数が返すポリシーが、デバイスが実行しようとしているアクションのためのアクセス許可を付与しないことである可能性があります。関数が返すポリシーに問題があるかどうかを判断するには、次の手順を実行します。

   a. Amazon CloudWatch Logs Insights クエリを使用して、短期間にログをスキャンし、エラーがないか確認します。次のクエリ例では、イベントをタイムスタンプでソートし、エラーを探します。

   ```
   display clientId, eventType, status, @timestamp | sort @timestamp desc | filter
    status = "Failure"
   ```

   b. Lambda 関数を更新して、返されるデータと、関数 AWS IoT Core をトリガーするイベントを口グに記録します。これらの口グを使用して、関数が作成するポリシーを検査できます。

6. 呼び出しにエラーがないにもかかわらず、デバイスが接続できない (またはメッセージを発行、サブスクライブ、受信できない) 場合は、別の原因として、Lambda 関数がタイムアウト制限を超えている可能性があります。カスタムオーソライザーの Lambda 関数のタイムアウト制限は 5 秒です。関数の期間は、CloudWatch 口グまたはメトリクスで確認できます。

デバイスの問題の調査

Lambda 関数の呼び出しに問題がない場合や、関数が返すポリシーに問題がない場合は、デバイスの接続試行に問題がないか確認してください。不正な形式の接続リクエストにより、 がオーソライザーをトリガー AWS IoT Core しない可能性があります。接続の問題は、TLS レイヤーとアプリケーションレイヤーの両方で発生する可能性があります。

考えられる TLS レイヤーの問題:

- お客様は、すべてのカスタム認証リクエストで、ホスト名ヘッダー (HTTP、MQTT over WebSockets) または Server Name Indication TLS 拡張 (HTTP、MQTT over WebSockets、MQTT) を渡す必要があります。どちらの場合も、渡される値はアカウントの AWS IoT Core データエンドポイントのいずれかと一致する必要があります。これらは、次の CLI コマンドを実行したときに返されるエンドポイントです。
  - `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

  - `aws iot describe-endpoint --endpoint-type iot:Data` (レガシー VeriSign エンドポイント)
- MQTT 接続にカスタム認証を使用するデバイスは、`mqtt` の値の Application Layer Protocol Negotiation (ALPN) TLS 拡張も渡す必要があります。
- カスタム認証は現在、ポート 443 でのみ使用できます。

考えられるアプリケーションレイヤーの問題:

- 署名が有効になっている場合 (オーソライザーで `signingDisabled` フィールドが false の場合)、次の署名の問題がないかを確認します。
  - `x-amz-customauthorizer-signature` ヘッダーまたはクエリ文字列パラメータのいずれかでトークン署名を渡していることを確認してください。
  - サービスがトークン以外の値に署名していないことを確認してください。
  - オーソライザーの `token-key-name` フィールドで指定したヘッダーまたはクエリパラメータでトークンを渡すようにしてください。
- `x-amz-customauthorizer-name` ヘッダーまたはクエリ文字列パラメータで渡すオーソライザー名が有効であること、またはアカウント用にデフォルトのオーソライザーが定義されていることを確認してください。

Authorization

認可とは、認証された ID にアクセス許可を付与するプロセスです。 AWS IoT Core および IAM ポリシー AWS IoT Core を使用して、 でアクセス許可を付与します。このトピックでは、 AWS IoT Core ポリシーについて説明します。IAM ポリシーの詳細については、の ID とアクセスの管理 AWS IoT および が IAM と AWS IoT 連携する方法 を参照してください。

AWS IoT Core ポリシーは、認証された ID が実行できる操作を決定します。認証済みの ID は、デバイス、モバイルアプリケーション、ウェブアプリケーション、デスクトップアプリケーションで使用されます。認証された ID は、 AWS IoT Core CLI コマンドを入力するユーザーでもかまいません。ID は、それらの AWS IoT Core オペレーションに対するアクセス許可を付与するポリシーがある場合にのみ、オペレーションを実行できます。

AWS IoT Core ポリシーと IAM ポリシーはどちらも、アイデンティティ (プリンシパルとも呼ばれます) が実行できるオペレーションを制御する AWS IoT Core ために とともに使用されます。使用するポリシータイプは、認証に使用する ID のタイプによって異なります AWS IoT Core。

AWS IoT Core オペレーションは 2 つのグループに分けられます。

- コントロールプレーン API では、証明書、モノ、ルールなどの作成または更新などの管理タスクを行うことができます。
- データプレーン API を使用すると、 との間でデータを送受信できます AWS IoT Core。

使用するポリシーのタイプは、コントロールプレーン API とデータプレーン API のどちらを使用しているかによって異なります。

次の表に、ID タイプ、使用しているプロトコル、認可時に使用することのできるポリシータイプを示します。

AWS IoT Core データプレーン API とポリシータイプ

プロトコルと認証メカニズム	SDK	ID のタイプ	ポリシータイプ
TLS/TCP を介した MQTT、TLS 相互認証 (ポート 8883 または 443)[†])	AWS IoT デバイス SDK	X.509 証明書	AWS IoT Core ポリシー
MQTT over HTTPS/WebSocket、AWS SigV4 認証 (ポート 443)	AWS モバイル SDK	認証された Amazon Cognito ID	IAM および AWS IoT Core ポリシー
		認証されていない Amazon Cognito ID	IAM ポリシー
		IAM、またはフェデレーテッド ID	IAM ポリシー

プロトコルと認証メカニズム	SDK	ID のタイプ	ポリシータイプ		
HTTPS、AWS 署名バージョン 4 認証 (ポート 443)	AWS CLI	Amazon Cognito、IAM、またはフェデレーテッド ID	IAM ポリシー		
HTTPS、TLS 相互認証 (ポート 8443)	SDK はサポートしていません	X.509 証明書	AWS IoT Core ポリシー		
カスタム認証を介した HTTPS (ポート 443)	AWS IoT デバイス SDK	カスタムオーソライザー	カスタムオーソライザーポリシー		

AWS IoT Core コントロールプレーン API とポリシータイプ

プロトコルと認証メカニズム	SDK	ID のタイプ	ポリシータイプ		
HTTPS AWS 署名バージョン 4 認証 (ポート 443)	AWS CLI	Amazon Cognito ID	IAM ポリシー		
		IAM、またはフェデレーテッド ID	IAM ポリシー		

AWS IoT Core ポリシーは、X.509 証明書、Amazon Cognito ID、またはモノのグループにアタッチされます。IAM ポリシーは、IAM ユーザー、グループ、ロールにアタッチされます。 AWS IoT コンソールまたは CLI AWS IoT Core を使用して (証明書、Amazon Cognito Identity、またはモノのグループに) ポリシーをアタッチする場合は、 AWS IoT Core ポリシーを使用します。それ以外の場合は、モノのグループにアタッチされた IAM policy. AWS IoT Core policies が、そのモノのグループ内

のすべてのモノに適用されます。 AWS IoT Core ポリシーを有効にするには、 clientIdとモノの名前が一致している必要があります。

ポリシーベースの権限付与は強力なツールになります。これにより、デバイス、ユーザー、アプリケーションが AWS IoT Coreでできることを完全に制御できます。例えば、証明書 AWS IoT Core を使用して に接続するデバイスを考えてみましょう。この場合、デバイスを使用して、すべての MQTT トピックへのアクセスを許可するか、1 つのトピックにアクセスを制限できます。または、コマンドラインで CLI コマンドを入力することもできます。ポリシーを使用すると、ユーザーの任意のコマンドまたは AWS IoT Core リソースへのアクセスを許可または拒否できます。また、 AWS IoT Core リソースへのアプリケーションのアクセスを制御することもできます。

AWS IoT がポリシードキュメントをキャッシュする方法によっては、ポリシーに加えられた変更が有効になるまでに数分かかる場合があります。つまり、最近アクセス権が付与されたリソースにアクセスするには数分かかる場合があり、アクセスが取り消された後、数分間リソースにアクセスできる場合があります。

AWS トレーニングと認定

での認可については AWS IoT Core、 AWS 「トレーニングと認定」ウェブサイトの「認証と認可の詳細 AWS IoT Core」コースを受講してください。

AWS IoT Core ポリシー

AWS IoT Core ポリシーは JSON ドキュメントです。IAM ポリシーと同じ規則に従います。 は名前付きポリシー AWS IoT Core をサポートしているため、多くの ID が同じポリシードキュメントを参照できます。名前付きポリシーは、簡単にロールバックされるようにバージョン管理されます。

AWS IoT Core ポリシーを使用すると、 AWS IoT Core データプレーンへのアクセスを制御できます。 AWS IoT Core のデータプレーンは、 AWS IoT Core メッセージブローカーへの接続、MQTT メッセージの送受信、デバイスのシャドウの取得または更新を可能にするオペレーションで構成されます。

AWS IoT Core ポリシーは、1 つ以上のポリシーステートメントを含む JSON ドキュメントです。各ステートメントには、次の内容が含まれます。

- `Effect`: アクションが許可されるか拒否されるかを指定します。
- `Action` では、ポリシーで許可または拒否されているアクションを指定します。
- `Resource` では、アクションを許可または拒否するリソースを 1 つ以上指定します。

ポリシーに加えられた変更は、 がポリシードキュメントを AWS IoT キャッシュするため、有効になるまでに 6~8 分かかる場合があります。つまり、最近アクセス権が付与されたリソースにアクセスするには数分かかる場合があり、アクセスが取り消された後、数分間リソースにアクセスできる場合があります。

AWS IoT Core ポリシーは、X.509 証明書、Amazon Cognito ID、およびモノのグループにアタッチできます。モノのグループにアタッチされたポリシーは、そのグループ内のあらゆるものに適用されます。ポリシーを有効にするには、`clientId` とモノの名前が一致している必要があります。 AWS IoT Core ポリシーは、IAM ポリシーと同じポリシーの評価ロジックに従います。デフォルトでは、すべてのポリシーが明示的に拒否されます。アイデンティティベースのポリシーまたはリソースベースのポリシーに対する明示的な許可は、このデフォルト設定を上書きします。ポリシー内の明示的な拒否は、すべての許可に優先します。詳細については、AWS Identity and Access Management ユーザーガイドの「ポリシーの評価論理」を参照してください。

トピック

- AWS IoT Core ポリシーアクション
- AWS IoT Core アクションリソース
- AWS IoT Core ポリシー変数
- サービス間での不分別な代理処理の防止
- AWS IoT Core ポリシーの例
- Amazon Cognito ID を使用した承認

AWS IoT Core ポリシーアクション

次のポリシーアクションは、 AWS IoT Coreによって定義されています。

MQTT ポリシーアクション

`iot:Connect`

AWS IoT Core メッセージブローカーに接続するアクセス許可を表します。`iot:Connect`アクセス許可は、`CONNECT` リクエストがブローカーに送信される度に確認されます。メッセージブローカーは、同じクライアント ID を持つ 2 つのクライアントが同時に接続を維持することを許可しません。2 番目のクライアントが接続すると、ブローカーは既存の接続を閉じます。`iot:Connect` アクセス許可を使い、特定のクライアント ID を使用している権限を持つクライアントのみが接続できる事を確認します。

`iot:DeleteConnection`

接続された MQTT クライアントを AWS IoT Coreから切断するアクセス許可を表します。`iot:DeleteConnection` アクセス許可は、クライアントを強制的に切断するリクエストが行われるたびにチェックされます。クライアントを切断すると、 はクライアントのネットワーク接続を AWS IoT Core 閉じ、オプションでセッション状態をクリーンアップします。

`iot:GetRetainedMessage`

保持されている単一のメッセージの内容を取得するためのアクセス許可を表します。保持されたメッセージは、RETAIN フラグを設定して公開され、 によって保存されたメッセージです AWS IoT Core。アカウントの保持されているすべてのメッセージのリストを取得するためのアクセス許可については、`iot:ListRetainedMessages`を参照。

`iot:ListRetainedMessages`

アカウントの保持メッセージの内容ではなく、それに関する概要情報を取得するためのアクセス許可を表します。保持されたメッセージは、RETAIN フラグを設定して公開され、 によって保存されたメッセージです AWS IoT Core。このアクションのために指定されたリソース ARN は*のはずです。保持されている単一のメッセージの内容を取得するためのアクセス許可については、`iot:GetRetainedMessage` を参照。

`iot:Publish`

MQTT トピックを発行するためのアクセス許可を表します。このアクセス権限は、PUBLISH リクエストがブローカーに送信される度に確認されます。このアクセス許可を使用して、クライアントが特定のトピックパターンに対し発行できるようにします。

> **Note**
>
> `iot:Publish` アクセス許可を付与するには、`iot:Connect` アクセス許可も付与する必要があります。

`iot:Receive`

メッセージを受信するアクセス許可を表します AWS IoT Core。`iot:Receive` アクセス許可は、メッセージがクライアントに配信されるたびに確認されます。このアクセス許可は配信ごとに確認されるため、この権限を利用し、現在トピックにサブスクライブしているクライアントに対してアクセス許可を取り消すことができます。

`iot:RetainPublish`

設定された RETAIN フラッグで MQTT メッセージを発行するためのアクセス許可を表します。

> **Note**
>
> `iot:RetainPublish` アクセス許可を付与するには、`iot:Publish` アクセス許可も付与する必要があります。

`iot:Subscribe`

トピックフィルターにサブスクライブするアクセス権限を表します。このアクセス権限は、SUBSCRIBE リクエストがブローカーに送信される度に確認されます。このアクセス許可を使用して、クライアントが、特定のトピックパターンに一致するトピックにサブスクライブできるようにします。

> **Note**
>
> `iot:Subscribe` アクセス許可を付与するには、`iot:Connect` アクセス許可も付与する必要があります。

シャドウポリシーアクション

`iot:DeleteThingShadow`

thing のデバイスシャドウを削除するアクセス権限を表します。`iot:DeleteThingShadow` アクセス権限は、thing のデバイスシャドウのコンテンツの削除リクエストが行われるたびに確認されます。

`iot:GetThingShadow`

thing のデバイスシャドウを取得するアクセス権限を表します。`iot:GetThingShadow` アクセス権限は、thing のデバイスシャドウコンテンツの取得リクエストが行われるたびに確認されます。

`iot:ListNamedShadowsForThing`

thing の名前付きのシャドウを削除するアクセス権限を表します。`iot:ListNamedShadowsForThing` アクセス権限は、thing の名前付きシャドウの一覧表示リクエストが行われる度に確認されます。

`iot:UpdateThingShadow`

デバイスのシャドウを更新するアクセス権限を表します。`iot:UpdateThingShadow` アクセス権限は、thing のデバイスシャドウコンテンツの更新リクエストが行われるたびに確認されます。

> **Note**
>
> ジョブ実行ポリシーアクションは、HTTP TLS エンドポイントにのみ適用されます。MQTT エンドポイントを使用する場合は、このトピックで定義された MQTT ポリシーアクションを使用する必要があります。
> これを示すジョブ実行ポリシーの例については、MQTT プロトコルと連携する the section called “基本的なジョブポリシーの例” を参照してください。

ジョブ実行 AWS IoT Core ポリシーアクション

`iotjobsdata:DescribeJobExecution`

特定のモノのジョブの実行を取得するアクセス権限を表します。`iotjobsdata:DescribeJobExecution` アクセス許可は、ジョブの実行の取得リクエストが行われるたびに確認されます。

`iotjobsdata:GetPendingJobExecutions`

モノの終了のステータスではないジョブのリストを取得するアクセス権限を表します。`iotjobsdata:GetPendingJobExecutions` アクセス権限は、リストの取得リクエストが行われるたびに確認されます。

`iotjobsdata:UpdateJobExecution`

ジョブの実行を更新するアクセス権限を表します。`iotjobsdata:UpdateJobExecution` アクセス権限は、ジョブ実行の状態の更新リクエストが行われるたびに確認されます。

`iotjobsdata:StartNextPendingJobExecution`

モノに対して保留中の次のジョブ実行を取得および開始するアクセス権限を表します (つまり、ステータスが QUEUED から IN_PROGRESS であるジョブの実行を更新します)。`iotjobsdata:StartNextPendingJobExecution` アクセス許可は、保留中の次のジョブ実行を開始するリクエストが行われるたびに確認されます。

AWS IoT Core 認証情報プロバイダーポリシーアクション

`iot:AssumeRoleWithCertificate`

AWS IoT Core 認証情報プロバイダーを呼び出して、証明書ベースの認証で IAM ロールを引き受けるアクセス許可を表します。アクセス`iot:AssumeRoleWithCertificate`許可は、 AWS IoT Core 認証情報プロバイダーにロールを引き受けるリクエストが行われるたびにチェックされます。

AWS IoT Core アクションリソース

AWS IoT Core ポリシーアクションのリソースを指定するには、リソースの Amazon リソースネーム (ARN) を使用します。リソースの ARN はすべて、次のような形式になります。

```
arn:partition:iot:region:AWS-account-ID:Resource-type/Resource-name
```

以下の表では、アクションタイプごとに指定するリソースを示しています。ARN の例は、パーティション aws 内のアカウント ID 123456789012 で、リージョン us-east-1 に固有です。ARN の形式の詳細については、 AWS Identity and Access Management ユーザーガイドの「Amazon リソースネーム (ARNs)」を参照してください。

Action	リソースタイプ	リソース名	ARN の例
iot:Connect	client	クライアントのクライアント ID	arn:aws:iot:us-east-1:123456789012:client/myClientId
iot:DeleteConnection	client	クライアントのクライアント ID	arn:aws:iot:us-east-1:123456789012:client/myClientId
iot:DeleteThingShadow	thing	モノの名前、および該当する場合はシャドウの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-

Action	リソースタイプ	リソース名	ARN の例
			east-1:123456789012:thing/thingOne/shadowOne
iotjobsdata:DescribeJobExecution	thing	モノの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iotjobsdata:GetPendingJobExecutions	thing	モノの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:GetRetainedMessage	topic	保持されたメッセージトピック	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:GetThingShadow	thing	モノの名前、および該当する場合はシャドウの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:ListNamedShadowsForThing	すべて	すべて	*
iot:ListRetainedMessages	すべて	すべて	*

Action	リソースタイプ	リソース名	ARN の例
iot:Publish	topic	トピック文字列	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:Receive	topic	トピック文字列	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:RetainPublish	topic	設定された RETAIN フラグと共に発行するためのトピック	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iotjobsdata:StartNextPendingJobExecution	thing	モノの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:Subscribe	topicfilter	トピックフィルター文字列	arn:aws:iot:us-east-1:123456789012:topicfilter/myTopicFilter
iotjobsdata:UpdateJobExecution	thing	モノの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:UpdateThingShadow	thing	モノの名前、および該当する場合はシャドウの名前	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne

Action	リソースタイプ	リソース名	ARN の例
iot:AssumeRoleWithCertificate	rolealias	ロール ARN をポイントするロールエイリアス	arn:aws:iot:us-east-1:123456789012:rolealias/CredentialProviderRole_alias

AWS IoT Core ポリシー変数

AWS IoT Core は、 Resource または Conditionブロックのポリシーで使用できる AWS IoT Core ポリシー変数を定義します。ポリシーが評価されると、ポリシー変数は実際の値に置き換えられます。たとえば、デバイスがクライアント ID が 100-234-3456 の AWS IoT Core メッセージブローカーに接続されている場合、iot:ClientIdポリシードキュメントのポリシー変数は 100-234-3456 に置き換えられます。

AWS IoT Core ポリシーはワイルドカード文字を使用し、IAM ポリシーと同様の規則に従うことができます。文字列に * (アスタリスク) を挿入すると、任意の文字に一致するワイルドカードとして扱うことができます。例えば、* を使用してポリシーの Resource 属性で複数の MQTT トピック名を記述できます。+ と # の文字は、ポリシーの中でリテラル文字列として扱われます。ワイルドカードを使用する方法を示したポリシー例については、「MQTT および AWS IoT Core ポリシーでのワイルドカード文字の使用」を参照してください。

また、固定値を持つ事前定義されたポリシー変数を使用して、それ以外の場合に特別な意味を持つ文字を表現することができます。これらの特殊文字には、$(*)、$(?)、および $($) が含まれます。ポリシー変数と特殊文字の詳細については、「IAM ポリシーの要素: 変数とタグ」および「複数のキーまたは値による条件の作成」を参照してください。

トピック

- 基本的な AWS IoT Core ポリシー変数
- モノのポリシー変数
- X.509 証明書 AWS IoT Core ポリシー変数

基本的な AWS IoT Core ポリシー変数

AWS IoT Core は、以下の基本ポリシー変数を定義します。

- `aws:SourceIp`: AWS IoT Core メッセージブローカーに接続されているクライアントの IP アドレス。
- `iot:ClientId`: このクライアント ID は、 AWS IoT Core メッセージブローカーに接続するために使用されます。
- `iot:DomainName`: 接続先のクライアントのドメイン名 AWS IoT Core。

例

- ClientId および SourceIp ポリシー変数の例
- iot:DomainName ポリシー変数の例

`ClientId` および **`SourceIp`** ポリシー変数の例

次の AWS IoT Core ポリシーは、ポリシー変数を使用するポリシーを示しています。 は、ポリシーの条件要素で使用して、プリンシパルが特定のアドレス範囲内でのみ API リクエストを行うことを許可`aws:SourceIp`できます。例については「ユーザーとクラウドサービスに AWS IoT ジョブの使用を許可する」を参照してください。

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

これらの例では、ポリシー${iot:ClientId}が評価されると、 は AWS IoT Core メッセージブローカーに接続されたクライアントの ID に置き換えられます。${iot:ClientId} などのポリシー変数を使用すると、アクセス可能にしないトピックにアクセスすることがあります。例えば、${iot:ClientId} を使用するポリシーでトピックフィルターを指定する場合です。

```
{
 "Effect": "Allow",
 "Action": [
  "iot:Subscribe"
 ],
 "Resource": [
  "arn:aws:iot:us-east-1:123456789012:topicfilter/my/${iot:ClientId}/topic"
 ]
}
```

クライアントは、クライアント ID として + を使用して接続できます。これにより、ユーザーはトピックフィルター my/+/topic に一致する任意のトピックにサブスクライブできます。このようなセキュリティギャップから保護するには、iot:Connect ポリシーアクションを使用して、どのクライアント ID が接続できるかを制御します。例えば、このポリシーにより、これらのクライアント ID が clientid1 のクライアントのみが接続できるようになります。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
```

```
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/clientid"
    ]
   }
 ]
}
```

> **Note**
>
> ポリシー変数 ${iot:ClientId} を Connect と併用することはお勧めしません。ClientId の値はチェックされないため、別のクライアントの ID のアタッチャーが検証に合格しても接続が切断されることがあります。どのクライアント ID ClientId も許可されているため、ランダムなクライアント ID を設定すると、モノグループのポリシーがバイパスされる可能性があります。

iot:DomainName ポリシー変数の例

iot:DomainName ポリシー変数を追加して、使用できるドメインを制限できます。iot:DomainName ポリシー変数を追加すると、デバイスは特定の設定済みエンドポイントにのみ接続できます。

次のポリシーでは、デバイスが指定されたドメインに接続することを許可します。

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

```
}
```

次のポリシーは、デバイスが指定されたドメインに接続することを拒否します。

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

ポリシー条件演算子の詳細については、「IAM JSON ポリシーエレメント: 条件演算子」を参照してください。ドメイン設定の詳細については、「ドメイン設定とは」を参照してください。

モノのポリシー変数

モノのポリシー変数を使用すると、モノの名前、モノのタイプ、モノの属性値などのモノのプロパティに基づいてアクセス許可を付与または拒否する AWS IoT Core ポリシーを作成できます。モノのポリシー変数を使用して、同じポリシーを適用して多くの AWS IoT Core デバイスを制御できます。デバイスのプロビジョニングの詳細については、「デバイスプロビジョニング」を参照してください。

非排他的なモノの関連付けを使用する場合、同じ証明書を複数のモノにアタッチすることができます。明確な関連付けを維持し、潜在的な競合を回避するには、クライアント ID をモノの名前と一致させる必要があります。この場合、モノの接続時に送信される MQTT Connect メッセージ内のクライアント ID からモノの名前を取得します AWS IoT Core。

AWS IoT Core ポリシーでモノのポリシー変数を使用する場合は、次の点に注意してください。

- AttachThingPrincipal API を使用して、証明書やプリンシパル (認証済みの Amazon Cognito ID) をモノにアタッチします。
- モノの名前をモノのポリシー変数に置き換える場合、MQTT 接続メッセージまたは TLS 接続の `clientId` の値がモノの名前と完全に一致している必要があります。

以下のモノのポリシー変数が利用可能です。

- `iot:Connection.Thing.ThingName`

  これは、ポリシーが評価されている AWS IoT Core レジストリ内のモノの名前に解決されます。は、デバイスが認証時に提示する証明書 AWS IoT Core を使用して、接続の検証に使用するモノを決定します。このポリシー変数は、デバイスが MQTT または MQTT over WebSocket プロトコルに接続するときにのみ使用できます。

- `iot:Connection.Thing.ThingTypeName`

  これは、ポリシーが評価されているモノと関連付けられるモノのタイプに解決されます。MQTT/WebSocket 接続のクライアント ID は、モノの名前と同じである必要があります。このポリシー変数は、MQTT または MQTT over WebSocket プロトコルに接続するときにのみ使用できます。

- `iot:Connection.Thing.Attributes[`*`attributeName`*`]`

  これは、ポリシーが評価されているモノと関連付けられる指定した属性値に解決されます。モノには最大 50 個の属性を指定できます。各属性はポリシー変数として使用できます。`iot:Connection.Thing.Attributes[`*`attributeName`*`]` *`attributeName`* は属性の名前です。MQTT/WebSocket 接続のクライアント ID は、モノの名前と同じである必要があります。このポリシー変数は、MQTT または MQTT over WebSocket プロトコルに接続するときにのみ使用できます。

- `iot:Connection.Thing.IsAttached`

  `iot:Connection.Thing.IsAttached: ["true"]` は、 に登録 AWS IoT され、プリンシパルにアタッチされているデバイスのみがポリシー内のアクセス許可にアクセスできるようにします。この変数を使用して、デバイスがレジストリ内の AWS IoT Core IoT モノにアタッチされていない証明書を提示 AWS IoT Core する場合、デバイスが に接続できないようにすることができます。この変数には値 `true` または があり、接続するモノが AttachThingPrincipal API を使用してレジストリ内の証明書または Amazon Cognito ID にアタッチされている`false`ことを示します。モノの名前はクライアント ID として使用されます。

クライアント ID がモノの名前と一致する場合、または証明書をモノに排他的にアタッチする場合は、ポリシー定義でポリシー変数を使用すると、ポリシー管理を簡素化できます。IoT のモノごとに個別のポリシーを作成する代わりに、モノのポリシー変数を使用して単一のポリシーを定義できます。このポリシーは、すべてのデバイスに動的に適用できます。以下は、この仕組みを示すポリシーの例です。詳細については、「???」を参照してください。

```
{
 "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/*"
  }
 ]
}
```

このポリシー例では、クライアント ID が envType 属性の値で終わる AWS IoT Core と、モノは に接続できます。接続できるのは、一致するクライアント ID パターンを持つモノのみです。

X.509 証明書 AWS IoT Core ポリシー変数

X.509 証明書ポリシー変数は、 AWS IoT Core ポリシーの記述に役立ちます。これらのポリシーは、X.509 証明書属性に基づいてアクセス許可を付与します。次のセクションでは、それらの証明書のポリシー変数を使用する方法を説明します。

> **Important**
>
> X.509 証明書に特定の証明書属性が含まれていないものの、対応する証明書ポリシー変数がポリシードキュメントで使用されている場合、ポリシー評価によって予期しない動作が発生する可能性があります。

CertificateId

RegisterCertificate API では、レスポンス本文に certificateId が表示されます。証明書に関する情報を取得するには、DescribeCertificate で certificateId を使用できます。

発行元の属性

次の AWS IoT Core ポリシー変数は、証明書発行者が設定した証明書属性に基づいて、アクセス許可の許可または拒否をサポートします。

- 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

件名の属性

次の AWS IoT Core ポリシー変数は、証明書発行者が設定した証明書のサブジェクト属性に基づいて、アクセス許可の付与または拒否をサポートします。

- 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 証明書は、これらの属性に 1 つ以上の値を含むオプションを提供します。
デフォルトでは、複数値の各属性用のポリシー変数は最初の値を返します。例えば、Certificate.Subject.Country 属性には国名のリストが含まれる場合がありますが、ポリシーで評価されると、iot:Certificate.Subject.Country は最初の国名に置き換えられます。

1 から始めるインデックスを使用して、最初の値以外の特定の属性値をリクエストできます。例えば、iot:Certificate.Subject.Country.1 は、Certificate.Subject.Country 属性の 2 番目の国名に置き換えられます。存在していないインデックス値を指定する場合、(例えば、属性に割り当てられた値が 2 つのみのとき 3 番目の値を要求すると) 置き換えはされず、認可は失敗します。ポリシーの変数名で、.List サフィックスを使用して、属性の値をすべて指定できます。

発行元の代替名属性

次の AWS IoT Core ポリシー変数は、証明書発行者が設定した発行者の代替名属性に基づいて、アクセス許可の付与または拒否をサポートします。

- 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

件名の代替名属性

次の AWS IoT Core ポリシー変数は、証明書発行者が設定したサブジェクト代替名属性に基づいて、アクセス許可の付与または拒否をサポートします。

- 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

その他の属性

を使用してiot:Certificate.SerialNumber、証明書のシリアル番号に基づいて、 AWS IoT Core リソースへのアクセスを許可または拒否できます。iot:Certificate.AvailableKeys ポリシー変数には、値を含むすべての証明書のポリシー変数の名前が含まれます。

X.509 証明書のポリシー変数の使用

このトピックでは、証明書のポリシー変数の使用方法について詳しく説明します。X.509 証明書のポリシー変数は、X.509 証明書の属性に基づいてアクセス許可を付与する AWS IoT Core ポリシーを作成するときに必須です。X.509 証明書に特定の証明書属性が含まれていないものの、対応する証明書ポリシー変数がポリシードキュメントで使用されている場合、ポリシー評価によって予期しない動作が発生する可能性があります。これは、欠落しているポリシー変数がポリシーステートメントで評価されないためです。

このトピックの内容

- X.509 証明書の例
- 証明書のポリシー変数としての証明書発行者の属性の使用
- 証明書のポリシー変数としての証明書のサブジェクト属性の使用
- 証明書ポリシー変数として証明書発行者の代替名属性を使用
- 証明書ポリシー変数として証明書サブジェクトの代替名属性を使用
- 証明書ポリシー変数として他の証明書属性を使用
- X.509 証明書のポリシー変数の制限
- 証明書ポリシー変数を使用したポリシーの例

X.509 証明書の例

一般的な X.509 証明書は次のように表示されます。この証明書の例には、証明書の属性が含まれています。 AWS IoT Core ポリシーの評価中、Serial Number、Issuer、Subject、X509v3

Issuer Alternative Name、および X509v3 Subject Alternative Name の証明書の属性が証明書ポリシー変数として入力されます。

```
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 >>
```

証明書のポリシー変数としての証明書発行者の属性の使用

次の表は、証明書発行者の属性を AWS IoT Core ポリシーに入力する方法の詳細を示しています。

ポリシーに入力する発行者の属性

証明書発行者の属性	証明書のポリシー変数
• 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	• `iot:Certificate.Issuer.Country = US` • `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.DistinguishedNameQualifier = 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`

証明書のポリシー変数としての証明書のサブジェクト属性の使用

次の表は、証明書のサブジェクト属性を AWS IoT Core ポリシーに入力する方法の詳細を示しています。

ポリシーに入力されるサブジェクト属性

証明書のサブジェクト属性	証明書のポリシー変数
• C=US	• `iot:Certificate.Subject.Country = US`

証明書のサブジェクト属性	証明書のポリシー変数
• O=IoT Devices • ST=NY • 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.Organization = IoT Devices` • `iot:Certificate.Subject.State = NY` • `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`

証明書ポリシー変数として証明書発行者の代替名属性を使用

次の表は、証明書発行者の代替名属性が AWS IoT Core ポリシーに入力される方法の詳細を示しています。

ポリシーに入力する発行者の代替名属性

X509v3 発行者の代替名	ポリシー内の属性
• DNS:issuer.com • IP Address:5.6.7.8 • URI:PrimarySignerCA • email:primary@issuer.com	• `iot:Certificate.Issuer.AlternativeName.DNSName = issuer.com` • `iot:Certificate.Issuer.AlternativeName.IPAddress = 5.6.7.8`

X509v3 発行者の代替名	ポリシー内の属性
• DirName:/C=US/O=Issuer/OU=IoT Devices/CN=Primary Issuer CA	• `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier = PrimarySignerCA` • `iot:Certificate.Issuer.AlternativeName.RFC822Name = primary@issuer.com` • `iot:Certificate.Issuer.AlternativeName.DirectoryName = cn=Primary Issuer CA,ou=IoT Devices,o=Issuer,c=US`

証明書ポリシー変数として証明書サブジェクトの代替名属性を使用

次の表は、証明書サブジェクトの代替名属性が AWS IoT Core ポリシーに入力される方法の詳細を示しています。

ポリシーに入力されるサブジェクトの代替名属性

X509v3 サブジェクトの代替名	ポリシー内の属性
• 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`

証明書ポリシー変数として他の証明書属性を使用

次の表は、他の証明書属性が AWS IoT Core ポリシーに入力される方法の詳細を示しています。

ポリシーに入力されるその他の属性

その他の証明書属性	証明書のポリシー変数
Serial Number: 92:12:85:cb:b7:a5:e0:86	iot:Certificate.SerialNumber = 105256223 89124227206

X.509 証明書のポリシー変数の制限

次の制限は、X.509 証明書のポリシー変数に適用されます。

欠落しているポリシー変数

X.509 証明書に特定の証明書属性が含まれていないものの、対応する証明書ポリシー変数がポリシードキュメントで使用されている場合、ポリシー評価によって予期しない動作が発生する可能性があります。これは、欠落しているポリシー変数がポリシーステートメントで評価されないためです。

証明書の SerialNumber 形式

AWS IoT Core は、証明書のシリアル番号を 10 進数の整数の文字列表現として扱います。例えば、ポリシーで証明書のシリアル番号に一致するクライアント ID のみの接続が許可されている場合、クライアント ID は 10 進数のシリアル番号である必要があります。

ワイルドカード

ワイルドカード文字が証明書の属性にある場合は、ポリシー変数は証明書属性値に置き換えられません。そのため、ポリシードキュメントに ${policy-variable} テキストが残ります。これにより、承認が失敗する場合があります。ワイルドカード文字として、*、$、+、?、# を使用できます。

配列フィールド

配列を含む証明書の属性は、5 つの項目に制限されます。追加項目は無視されます。

文字列の長さ

すべての文字列値は 1024 文字に制限されています。証明書の属性に、1024 文字以上の文字列が含まれている場合、ポリシー変数は証明書の属性値に置き換えられません。そのため、ポリシードキュメントに ${policy-variable} が残ります。これにより、承認が失敗する場合があります。

特殊文字

,、"、\、+、=、<、>、; などの特殊文字をポリシー変数で使用する場合は、先頭にバックスラッシュ (\) を付ける必要があります。例えば、Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US は Amazon Web Service O\=Amazon.com Inc. L\=Seattle ST\=Washington C\=US になります。

証明書ポリシー変数を使用したポリシーの例

次のポリシードキュメントでは、証明書のシリアル番号に一致するクライアント ID を使用して接続し、${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/* のパターンに一致するトピックに発行できます。

> **Important**
>
> X.509 証明書に特定の証明書属性が含まれていないものの、対応する証明書ポリシー変数がポリシードキュメントで使用されている場合、ポリシー評価によって予期しない動作が発生する可能性があります。これは、欠落しているポリシー変数がポリシーステートメントで評価されないためです。例えば、iot:Certificate.Subject.Organization 属性を含まない証明書に次のポリシードキュメントをアタッチした場合、ポリシー評価中に iot:Certificate.Subject.Organization 証明書ポリシー変数は入力されません。

```
{
 "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 条件演算子を使用して、ポリシーの評価中にポリシーで使用される証明書ポリシー変数が入力されるようにすることもできます。次のポリシードキュメントでは、証明書のシリアル番号と証明書のサブジェクトの共通名属性が存在する場合にのみ、証明書で iot:Connect を許可します。

すべての証明書ポリシー変数には文字列値があるため、すべての文字列条件演算子がサポートされます。

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

サービス間での不分別な代理処理の防止

「混乱した代理」問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の

問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、あるサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスが操作され、それ自身のアクセス許可を使用して、本来アクセス許可が付与されるべきではない方法で別の顧客のリソースに対して働きかけることがあります。これを防ぐため、 AWS では、アカウント内のリソースへのアクセス許可が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールを提供しています。

が別のサービス AWS IoT に付与するアクセス許可をリソースに制限するには、リソースポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用することをお勧めします。両方のグローバル条件コンテキストキーを同じポリシーステートメントで使用する場合は、aws:SourceAccount 値と、aws:SourceArn 値に含まれるアカウントが、同じアカウント ID を示している必要があります。

「混乱した代理」問題から保護するための最も効果的な方法は、リソースの完全なAmazon リソースネーム (ARN)を指定しながら、グローバル条件コンテキストキー aws:SourceArn を使用することです。の場合 AWS IoT、 aws:SourceArnはリソース固有のアクセス許可arn:*aws*:iot:*region*:*account-id*:*resource-type/resource-id*の場合は または の形式に従う必要がありますarn:*aws*:iot:*region*:*account-id*:*。resource-id は、許可されたリソースの名前または ID、または許可されたリソース ID のワイルドカードステートメントにすることができます。*#####*がお客様の AWS IoT リージョンと一致し、*account-id* がお客様のアカウント ID と一致していることを確認します。

次の例は、 AWS IoT ロール信頼ポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題を防ぐ方法を示しています。その他の例については、「混乱した代理の防止の詳細な例」を参照してください。

```
{
   "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**
>
> アクセス拒否エラーが発生した場合は、 AWS Security Token Service (STS) とのサービス統合が aws:SourceArn および aws:SourceAccount コンテキストキーをサポートしていない可能性があります。

混乱した代理の防止の詳細な例

このセクションでは、 AWS IoT ロール信頼ポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題を防ぐ方法の詳細な例を示します。

- フリートプロビジョニング
- JITP
- 認証情報プロバイダー

フリートプロビジョニング

プロビジョニングテンプレートリソースを使用してフリートプロビジョニングを設定できます。プロビジョニングテンプレートがプロビジョニングロールを参照する場合、そのロールの信頼ポリシーに aws:SourceArnおよび aws:SourceAccount 条件キーを含めることができます。これらのキーは、設定が sts:AssumeRole リクエストを呼び出すことができるリソースを制限します。

次の信頼ポリシーを持つロールは、SourceArn で指定されたプロビジョニングテンプレートの IoT プリンシパル (iot.amazonaws.com) によってのみ引き受けることができます。

```
{
   "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

ジャストインタイムプロビジョニング (JITP) では、プロビジョニングテンプレートを CA とは別のリソースとして使用することも、テンプレート本文とロールを CA 証明書設定の一部として定義することもできます。 AWS IoT ロール信頼ポリシーaws:SourceArnの の値は、プロビジョニングテンプレートの定義方法によって異なります。

プロビジョニングテンプレートを別のリソースとして定義する

プロビジョニングテンプレートを別のリソースとして定義した場合、aws:SourceArn の値は "arn:aws:iot:*region*:*account-id*:provisioningtemplate/*example_template*" になります。フリートプロビジョニング では同じポリシーの例を利用できます。

CA 証明書でのプロビジョニングテンプレートの定義

CA 証明書リソース内でプロビジョニングテンプレートを定義した場合、aws:SourceArn の値は "arn:aws:iot:*region*:*account-id*:cacert/*cert_id*" または "arn:aws:iot:*region*:*account-id*:cacert/*" になります。CA 証明書の ID などのリソース識別子が作成時に不明な場合は、ワイルドカードを使用できます。

次の信頼ポリシーを持つロールは、SourceArn で指定された CA 証明書の IoT プリンシパル (iot.amazonaws.com) によってのみ引き受けることができます。

```
{
   "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 証明書を作成するときは、登録設定でプロビジョニングロールを参照できます。プロビジョニングロールの信頼ポリシーは、ロールを引き受けることができるリソースを制限するために aws:SourceArn を使用できます。ただし、CA 証明書を登録するための最初の RegisterCACertificate 呼び出し中に、aws:SourceArn 条件に指定する CA 証明書の ARN はありません。

これを回避するには、つまり、 に登録されている特定の CA 証明書にプロビジョニングロールの信頼ポリシーを指定するには AWS IoT Core、以下を実行します。

- まず、RegistrationConfig パラメータを指定せずに RegisterCACertificate を呼び出します。
- CA 証明書が に登録されたら AWS IoT Core、その証明書で UpdateCACertificate を呼び出します。

  UpdateCACertificate 呼び出しで、新しく登録された CA 証明書の ARN に aws:SourceArn が設定されたプロビジョニングロール信頼ポリシーを含む RegistrationConfig を指定します。

認証情報プロバイダー

AWS IoT Core 認証情報プロバイダーの場合、 でロールエイリアスを作成する AWS アカウント のと同じ を使用しaws:SourceAccount、 でロールエイリアスリソースタイプのリソース ARN に一致するステートメントを指定しますaws:SourceArn。 AWS IoT Core 認証情報プロバイダーで使用する IAM ロールを作成するときは、ロールを引き受ける必要があるロールエイリアスの ARNs を aws:SourceArn条件に含めて、クロスサービスsts:AssumeRoleリクエストを承認する必要があります。

次の信頼ポリシーを持つロールは、 SourceArnで指定された roleAlias の AWS IoT Core 認証情報プロバイダー (credentials.iot.amazonaws.com) のプリンシパルのみが引き受けることができます。プリンシパルが aws:SourceArn 条件に指定されている以外のロールエイリアスの認証情報を取得しようとすると、その他のロールエイリアスが同じ IAM ロールを参照している場合でも、リクエストは拒否されます。

```
{
  "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 ポリシーの例

このセクションのポリシーの例は、 AWS IoT Coreで一般的なタスクを完了するために使われるポリシードキュメントを説明しています。ソリューションのポリシーを作成している際、開始時の例としてそれらが使用できます。

このセクションの例では、次のポリシー要素が使用されています。

- the section called “AWS IoT Core ポリシーアクション”
- the section called “AWS IoT Core アクションリソース”
- the section called “アイデンティティベースのポリシーの例”
- the section called “基本的な AWS IoT Core ポリシー変数”
- the section called “X.509 証明書 AWS IoT Core ポリシー変数”

このセクションのポリシーの例：

- 接続ポリシーの例
- パブリッシュ/サブスクライブポリシーの例
- 接続および公開ポリシーの例
- 保持されたメッセージポリシーの例
- 証明書のポリシーの例
- モノのポリシーの例
- 基本的なジョブポリシーの例

接続ポリシーの例

次のポリシーは、クライアント IDs `client1`と への接続許可を拒否`client2`し AWS IoT Core、デバイスがクライアント ID を使用して接続できるようにします。クライアント ID は、 AWS IoT Core レジストリに登録され、接続に使用されるプリンシパルにアタッチされているモノの名前と一致します。

> **Note**
>
> 登録済みデバイスの場合、`Connect` アクションにはモノのポリシー変数を使用し、接続に使用されるプリンシパルにモノをアタッチすることをおすすめします。

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

次のポリシーは、クライアント ID AWS IoT Core を使用して に接続するアクセス許可を付与しますclient1。このポリシーの例は、未登録のデバイス向けです。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
```

```
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/client1"
    ]
   }
 ]
}
```

MQTT 永続セッションポリシーの例

connectAttributes を使用すると、PersistentConnect や LastWill などの IAM ポリシーの接続メッセージで使用する属性を指定できます。詳細については、「connectAttributes の使用」を参照してください。

次のポリシーは、PersistentConnect 機能との接続を許可します。

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

次のポリシーでは PersistentConnect は許可されていませんが、他の機能は許可されています。

```
{
```

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

上記のポリシーは、StringEquals を使用して表現することもできます。新機能を含む他の機能はすべて許可されます。

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

次のポリシーは、PersistentConnect と LastWill の両方による接続を許可します。その他の新機能は許可されません。

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

次のポリシーは、LastWill の有無にかかわらず、クライアントによるクリーン接続を許可します。他の機能は許可されません。

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

次のポリシーは、デフォルト機能を使用した接続のみを許可します。

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

次のポリシーでは、PersistentConnect を使用した接続のみが許可されます。接続が PersistentConnect を使用する限り、新しい機能はすべて許可されます。

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

次のポリシーでは、接続には PersistentConnect と LastWill の両方の使用が必要であり、新機能は許可されないことが示されています。

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

次のポリシーには、PersistentConnect を含めることはできませんが、LastWill を含めることはできます。その他の新機能は許可されません。

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

次のポリシーは、トピック "my/lastwill/topicName" と一緒にLastWill を持つクライアントによる接続のみを許可します。LastWill トピックを使用する限り、すべての機能が許可されます。

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

次のポリシーは、特定の LastWillTopic を使用したクリーン接続のみを許可します。LastWillTopic を使用する限り、すべての機能が許可されます。

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

パブリッシュ/サブスクライブポリシーの例

使用するポリシーは、接続方法によって異なります AWS IoT Core。MQTT クライアント、HTTP、または WebSocket AWS IoT Core を使用して に接続できます。MQTT クライアントを使用して接続すると、X.509 証明書で認証されます。HTTP または WebSocket プロトコルを使用して接続すると、署名バージョン 4 および Amazon Cognito で認証されます。

> **Note**
>
> 登録済みデバイスの場合、Connect アクションにはモノのポリシー変数を使用し、接続に使用されるプリンシパルにモノをアタッチすることをおすすめします。

このセクションの内容:

- MQTT および AWS IoT Core ポリシーでのワイルドカード文字の使用
- 特定のトピックとの間でメッセージを発行、サブスクライブ、および受信するためのポリシー
- 特定のプレフィックスを持つトピックとの間でメッセージを発行、サブスクライブ、および受信するためのポリシー
- 各デバイスに固有のトピックの間でメッセージを発行、サブスクライブ、および受信するためのポリシー
- トピック名にモノ属性を含むトピックとの間でのメッセージの発行、サブスクライブ、受信に関するポリシー
- トピック名のサブトピックへのメッセージの発行を拒否するポリシー
- トピック名のサブトピックからのメッセージの受信を拒否するポリシー

- MQTT ワイルドカード文字を使用してトピックにサブスクライブするポリシー
- HTTP および WebSocket クライアントのポリシー

MQTT および AWS IoT Core ポリシーでのワイルドカード文字の使用

MQTT ポリシーと AWS IoT Core ポリシーではワイルドカード文字が異なるため、慎重に検討した上で選択する必要があります。MQTT では、ワイルドカード文字 +と #が MQTT トピックフィルターで使用され、複数のトピック名をサブスクライブします。 AWS IoT Core ポリシーは * とをワイルドカード文字?として使用し、IAM ポリシーの規則に従います。ポリシードキュメントでは、* は任意の文字の組み合わせを表し、疑問符 ? は任意の 1 文字を表します。ポリシードキュメントでは、MQTT ワイルドカード文字である + と # は、特別な意味を持たないこれらの文字として扱われます。ポリシーの `resource` 属性に複数のトピック名とトピックフィルターを記述するには、MQTT ワイルドカード文字の代わりに * と ? ワイルドカード文字を使用します。

ポリシードキュメントで使用するワイルドカード文字を選択するとき、* 文字は、単一のトピックレベルに限定されないことを考慮してください。+ 文字は、MQTT トピックフィルターの 1 つのトピックレベルに制限されます。ワイルドカードの仕様を単一の MQTT トピックフィルターレベルに制約するには、複数の ? 文字の使用を検討してください。ポリシーリソースでのワイルドカード文字の使用、およびワイルドカード文字が一致するその他の例については、「リソース ARN でのワイルドカードの使用」を参照してください。

次の表は、MQTT で使用されるさまざまなワイルドカード文字と MQTT クライアントの AWS IoT Core ポリシーを示しています。

ワイルドカード文字	MQTT のワイルドカード文字	MQTT での例	AWS IoT Core ポリシーのワイルドカード文字	MQTT クライアントの AWS IoT Core ポリシーの例
#	はい	`some/#`	いいえ	該当なし
+	はい	`some/+/topic`	いいえ	該当なし
*	いいえ	該当なし	はい	`topicfilter/some/*/topic` `topicfilter/some/sensor*/topic`

ワイルドカード文字	MQTT のワイルドカード文字	MQTT での例	AWS IoT Core ポリシーのワイルドカード文字	MQTT クライアントの AWS IoT Core ポリシーの例

?	いいえ	該当なし	はい	`topic/some/?????/topic`
				
				`topicfilter/some/sensor???/topic`

特定のトピックとの間でメッセージを発行、サブスクライブ、および受信するためのポリシー

以下に、登録済みデバイスと未登録デバイスで、「some_speciic_topic」という名前のトピックとの間でメッセージを発行、サブスクライブ、受信する例を示します。例では、Publish と Receive がリソースとして「トピック」を使用し、Subscribe がリソースとして「トピックフィルター」を使用することも強調しています。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。また、「some_specific_topic」という名前のトピックに対する Publish、Subscribe、および Receive のアクセス許可も提供します。

```
{
 "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 レジストリに登録されていないデバイスの場合、次のポリシーにより、デバイスは clientId1, clientId2または clientId3 のいずれかを使用して接続できます。また、「some_specific_topic」という名前のトピックに対する `Publish`、`Subscribe`、および `Receive` のアクセス許可も提供します。

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

特定のプレフィックスを持つトピックとの間でメッセージを発行、サブスクライブ、および受信するためのポリシー

以下に、登録済みデバイスと未登録デバイスで、「topic_prefix」というプレフィックスが付いたトピックとの間でメッセージを発行、サブスクライブ、受信する例を示します。

> **Note**
>
> この例では、ワイルドカード文字 * を使用することに注意してください。* は、1 つのステートメントで複数のトピック名にアクセス許可を与えるのに便利ですが、必要以上の権限をデバイスに与えると、意図しない結果につながる可能性があります。そのため、慎重に検討したうえで、ワイルドカード文字 * のみを使用することをお勧めします。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。また、「some_specific_topic」というプレフィックスが付いたトピックに対する `Publish`、`Subscribe`、および `Receive` のアクセス許可も提供します。

```
{
 "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 レジストリに登録されていないデバイスの場合、次のポリシーにより、デバイスは clientId1, clientId2または clientId3 のいずれかを使用して接続できます。また、「some_specific_topic」というプレフィックスが付いたトピックに対する `Publish`、`Subscribe`、および `Receive` のアクセス許可も提供します。

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

各デバイスに固有のトピックの間でメッセージを発行、サブスクライブ、および受信するためのポリシー

以下は、登録済みデバイスと未登録デバイスで、特定のデバイスに固有のトピックとの間でメッセージの発行、サブスクライブ、および受信を行う例を示しています。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。これにより、モノ固有のトピック (`sensor/device/${iot:Connection.Thing.ThingName}`) への発行、モノ固有のトピック (`command/device/${iot:Connection.Thing.ThingName}`) との間でサブスクライブおよび受信を行う許可が提供されます。レジストリ内のモノの名前が「thing1」の場合、デバイスは「sensor/device/thing1」トピックに発行できます。また、デバイスは「command/device/thing1」トピックにサブスクライブして受信することもできます。

```
{
 "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 レジストリに登録されていないデバイスの場合、次のポリシーにより、デバイスは clientId1, clientId2または clientId3 のいずれかを使用して接続できます。これにより、クライアント固有のトピック (sensor/device/${iot:ClientId}) への発行、クライアント固有のトピック (command/device/${iot:ClientId}) との間でサブスクライブおよび受信を行う許可が提供されます。デバイスが clientId1 として clientId で接続すると、トピック「sensor/device/clientId1」に発行できるようになります。デバイスは、トピック device/clientId1/command にサブスクライブして から受信することもできます。

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

トピック名にモノ属性を含むトピックとの間でのメッセージの発行、サブスクライブ、受信に関するポリシー

以下に、登録済みデバイスが、名前にモノの属性を含むトピックとの間でメッセージを発行、サブスクライブ、受信する例を示します。

> **Note**
>
> モノの属性は、 AWS IoT Core レジストリに登録されているデバイスにのみ存在します。未登録のデバイスに対応する例はありません。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。これにより、トピック (`sensor/${iot:Connection.Thing.Attributes[version]}`) への発行、トピック (`command/${iot:Connection.Thing.Attributes[location]}`) との間でのサブスクライブと受信を許可します。トピック名にはモノの属性が含まれます。レジストリ内のモノの名前に `version=v1` と `location=Seattle` がある場合、デバイスはトピック「sensor/v1」に発行し、トピック「command/Seattle」との間でサブスクライブして受信することができます。

```
{
 "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

モノの属性は AWS IoT Core のレジストリに登録されているデバイスにしか存在しないため、未登録のモノに対応する例はありません。

トピック名のサブトピックへのメッセージの発行を拒否するポリシー

以下は、登録済みデバイスと未登録デバイスで、特定のサブトピックを除くすべてのトピックにメッセージを発行する例を示しています。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。「department/」というプレフィックスが付いているすべてのトピックには発行できますが、「department/admins」サブトピックには発行できません。

```
{
 "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 レジストリに登録されていないデバイスの場合、次のポリシーにより、デバイスは clientId1, clientId2または clientId3 のいずれかを使用して接続できます。「department/」というプレフィックスが付いているすべてのトピックには発行できますが、「department/admins」サブトピックには発行できません。

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

トピック名のサブトピックからのメッセージの受信を拒否するポリシー

以下に、登録済みデバイスと未登録デバイスで、特定のサブトピックを除く特定のプレフィックスを持つトピックをサブスクライブしたり、トピックからメッセージを受信したりする例を示します。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。このポリシーにより、デバイスは「topic_prefix」というプレフィックスが付いた任意のトピックをサブスクライブできます。`iot:Receive` のステートメントで `NotResource` を使用すると、「topic_prefix/restricted」というプレフィックスが付いたトピックを除いて、デバイスがサブスクライブしているすべてのトピックからのメッセージをデバイスが受信できるようになります。例えば、このポリシーでは、デバイスは「topic_prefix/topic1」や「topic_prefix/restricted」をサブスクライブできますが、トピック「topic_prefix/topic1」からのメッセージのみを受信し、トピック「topic_prefix/restricted」からのメッセージは受信しません。

```
{
 "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 レジストリに登録されていないデバイスの場合、次のポリシーにより、デバイスは clientId1, clientId2または clientId3 のいずれかを使用して接続できます。このポリシーにより、デバイスは「topic_prefix」というプレフィックスが付いた任意のトピックをサブスクライブできます。`iot:Receive` のステートメントで `NotResource` を使用すると、「topic_prefix/restricted」というプレフィックスが付いたトピックを除いて、デバイスがサブスクライブしているすべてのトピックからのメッセージをデバイスが受信できるようになります。例えば、このポリシーでは、デバイスは「topic_prefix/topic1」と「topic_prefix/restricted」をサブスクライ

ブできます。ただし、トピック「topic_prefix/topic1」からのメッセージのみを受信し、トピック「topic_prefix/restricted」からのメッセージは受信しません。

```
{
    "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 ワイルドカード文字を使用してトピックにサブスクライブするポリシー

MQTT ワイルドカード文字 + と # はリテラル文字列として扱われますが、 AWS IoT Core ポリシーで使用するとワイルドカードとして扱われません。MQTT では、+ と # はトピックフィルターに登録する場合にのみワイルドカードとして扱われ、それ以外のコンテキストではリテラル文字列として扱われます。これらの MQTT ワイルドカードは、慎重に検討した上で AWS IoT Core ポリシーの一部としてのみ使用することをお勧めします。

AWS IoT Core ポリシーで MQTT ワイルドカードを使用する登録済みおよび未登録のモノの例を次に示します。これらのワイルドカードはリテラル文字列として扱われます。

Registered devices

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスはレジストリ内のモノの名前と一致する clientId に接続できます。このポリシーにより、デバイスは「department/+/employees」と「location/#」のトピックをサブスクライブできるようになります。+ と # は、 AWS IoT Core ポリシーではリテラル文字列として扱われるため、デバイスは「department/+/employees」というトピックにはサブスクライブできますが、「department/engineering/employees」のトピックにはサブスクライブできません。同様に、デバイスはトピック「location/#」をサブスクライブできますが、「location/Seattle」というトピックには登録できません。ただし、デバイスが「department/+/employees」というトピックに登録すると、ポリシーにより、「department/engineering/employees」というトピックからのメッセージを受信できるようになります。同様に、デバイスが「location/#」というトピックをサブスクライブすると、「location/Seattle」というトピックからのメッセージも受信するようになります。

```
{
 "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 レジストリに登録されていないデバイスの場合、次のポリシーにより、デバイスは clientId1, clientId2または clientId3 のいずれかを使用して接続できます。このポリシーにより、デバイスは「department/+/employees」と「location/#」のトピックをサブスクライブできるようになります。+ と # は、 AWS IoT Core ポリシーではリテラル文字列として扱われるため、デバイスは「department/+/employees」というトピックにはサブスクライブできますが、「department/engineering/employees」のトピックにはサブスクライブできません。同様に、デバイスはトピック「ロケーション/#」をサブスクライブできますが、「ロケーション/シアトル」はサブスクライブできません。ただし、デバイスが「department/+/employees」というトピックに登録すると、ポリシーにより、「department/engineering/employees」というトピックからのメッセージを受信できるようになります。同様に、デバイスが「location/#」というトピックをサブスクライブすると、「location/Seattle」というトピックからのメッセージも受信するようになります。

```
{
    "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 および WebSocket クライアントのポリシー

HTTP または WebSocket プロトコルを使用して接続すると、署名バージョン 4 および Amazon Cognito で認証されます。Amazon Cognito ID は、認証されている場合と認証されていない場合があります。認証された ID は任意のサポートされている認証プロバイダーで認証されたユーザーに属します。認証されていない ID は、通常、ID プロバイダーで認証しないゲストユーザーに属します。Amazon Cognito は、認証されていない ID をサポートするために一意の識別子と AWS 認証情報を提供します。詳細については、「the section called “Amazon Cognito ID を使用した承認”」を参照してください。

以下のオペレーションでは、 `AttachPolicy` API を介して Amazon Cognito ID にアタッチされた AWS IoT Core ポリシー AWS IoT Core を使用します。これにより、認証された ID を持つ Amazon Cognito アイデンティティプールにアタッチされたアクセス許可がスコープダウンされます。

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

つまり、Amazon Cognito ID には IAM ロールポリシーと AWS IoT Core ポリシーからのアクセス許可が必要です。`AttachPolicy` API を使用して、IAM ロールポリシーをプールにアタッチし、 AWS IoT Core ポリシーを Amazon Cognito ID にアタッチします AWS IoT Core 。

認証されたユーザーと未認証のユーザーは、異なる ID タイプです。 AWS IoT ポリシーを Amazon Cognito ID にアタッチしない場合、認証されたユーザーが AWS IoT での認可に失敗し、 AWS IoT リソースとアクションにアクセスできません。

> **Note**
>
> 他の AWS IoT Core オペレーションまたは認証されていない ID の場合、 AWS IoT Core は Amazon Cognito ID プールロールにアタッチされたアクセス許可の範囲を絞り込みません。認証済みの ID と非認証の ID の両方に対して、これは、Amazon Cognito プールのロールにアタッチすることをお勧めする最も緩いポリシーです。

HTTP

非認証の Amazon Cognito ID が Amazon Cognito ID に固有のトピックで HTTP を介してメッセージを発行できるようにするには、Amazon Cognito ID プールロールに以下の IAM ポリシーをアタッチします。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"
            ]
        }
```

```
    ]
}
```

認証されたユーザーを許可するには、 AWS IoT Core AttachPolicy API を使用して、前述のポリシーを Amazon Cognito ID プールロールと Amazon Cognito ID にアタッチします。

> **Note**
>
> Amazon Cognito ID を承認する場合、 は両方のポリシー AWS IoT Core を検討し、指定された最小限の権限を付与します。アクションは、両方のポリシーで要求されたアクションが許可されている場合にのみ許可されます。いずれかのポリシーでアクションが許可されていない場合、そのアクションは許可されません。

MQTT

非認証の Amazon Cognito ID がアカウントで Amazon Cognito ID に固有のトピックで WebSockets を介して MQTT メッセージをパブリッシュできるようにするには、Amazon Cognito ID プールロールに以下の IAM ポリシーをアタッチします。

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

```
}
```

認証されたユーザーを許可するには、 AWS IoT Core AttachPolicy API を使用して、前述のポリシーを Amazon Cognito ID プールロールと Amazon Cognito ID にアタッチします。

> **Note**
>
> Amazon Cognito ID を承認するときは、 と の両方 AWS IoT Core を考慮し、指定された最小限の権限を付与します。アクションは、両方のポリシーで要求されたアクションが許可されている場合にのみ許可されます。いずれかのポリシーでアクションが許可されていない場合、そのアクションは許可されません。

接続および公開ポリシーの例

AWS IoT Core レジストリでモノとして登録されたデバイスの場合、次のポリシーは、モノの名前に一致するクライアント ID AWS IoT Core を使用して に接続するアクセス許可を付与し、デバイスをクライアント ID またはモノの名前固有の MQTT トピックへの発行に制限します。接続を成功させるには、モノの名前を AWS IoT Core レジストリに登録し、モノにアタッチされた ID またはプリンシパルを使用して認証する必要があります。

```
{
  "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 レジストリにモノとして登録されていないデバイスの場合、次のポリシーはクライアント ID AWS IoT Core を使用して に接続するアクセス許可を付与`client1`し、デバイスを clientID 固有の MQTT トピックでの発行に制限します。

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

保持されたメッセージポリシーの例

保持されているメッセージの使用には、特定のポリシーが必要です。保持メッセージは、RETAIN フラグが設定され、 によって保存される MQTT メッセージです AWS IoT Core。このセクションでは、保持されたメッセージの一般的な使用を許可するポリシーの例を紹介します。

このセクションの内容:

- 保持されたメッセージを接続して発行するためのポリシー
- 保持された Will メッセージを接続して発行するためのポリシー
- 保持されたメッセージを一覧表示して取得するためのポリシー

保持されたメッセージを接続して発行するためのポリシー

デバイスが保持されたメッセージを発行するには、デバイスが、接続、発行 (任意のMQTTメッセージ)、そして保持されたMQTTメッセージを発行できる必要があります。次のポリシーにより、トピックに対するアクセス権限が付与されます。クライアント**device1**に`device/sample/configuration`。接続のアクセス許可を付与する別の例については、the section called “接続および公開ポリシーの例”を参照。

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

保持された Will メッセージを接続して発行するためのポリシー

クライアントは、クライアント AWS IoT Core が予期せず切断したときに発行するメッセージを設定できます。MQTT はこのようなメッセージをウィルメッセージと呼びます。それらを包括するために、クライアントは、接続許可に追加して追加条件を持つ必要があります。

次のポリシードキュメントは、 AWS IoT Core も保持するであろうトピックwillにより明確になった Will メッセージに接続発行する権限をすべてのクライアントに付与します。

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

保持されたメッセージを一覧表示して取得するためのポリシー

サービスとアプリケーションは、MQTT クライアントをサポートしなくても、ListRetainedMessagesおよびGetRetainedMessageを呼び出す事によって保持されたメッセージにアクセスする事ができます。これらのアクションを呼び出すサービスとアプリケーションは、次の例のようなポリシーを使用して認証される必要があります。

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

証明書のポリシーの例

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーは、モノの名前に一致するクライアント ID AWS IoT Core で に接続し、デバイスがそれ自体を認証するために使用した証明書certificateIdの と名前が等しいトピックに発行するアクセス許可を付与します。

```
{
    "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 レジストリに登録されていないデバイスの場合、次のポリシーは、クライアント IDs、、client3および client1 AWS IoT Core を使用して に接続しclient2、デバイスが自身を認証するために使用した証明書certificateIdの と名前が等しいトピックに発行するアクセス許可を付与します。

```
{
    "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 レジストリに登録されているデバイスの場合、次のポリシーは、モノの名前に一致するクライアント ID AWS IoT Core で に接続し、デバイスが自身を認証するために使用した証明書のサブジェクトの CommonNameフィールドと名前が等しいトピックに発行するアクセス許可を付与します。

```
{
    "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**
>
> この例では、証明書のサブジェクト共通名が、サブジェクト共通名が登録された各証明書に対して一意であることを前提にして、トピック ID として使用されます。複数のデバイス間

で証明書が共有されている場合、サブジェクト共通名は、その証明書を共有するすべてのデバイスで同じになります。したがって、複数のデバイスの同じトピックに発行する権限が許可されます (推奨されません)。

AWS IoT Core レジストリに登録されていないデバイスの場合、次のポリシーは、クライアント IDs、、`client3`および AWS IoT Core を使用して に接続し`client1``client2`、デバイスが自身を認証するために使用した証明書のサブジェクト`CommonName`フィールドと名前が等しいトピックに発行するアクセス許可を付与します。

```
{
    "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**
>
> この例では、証明書のサブジェクト共通名が、サブジェクト共通名が登録された各証明書に対して一意であることを前提にして、トピック ID として使用されます。複数のデバイス間

で証明書が共有されている場合、サブジェクト共通名は、その証明書を共有するすべてのデバイスで同じになります。したがって、複数のデバイスの同じトピックに発行する権限が許可されます (推奨されません)。

AWS IoT Core レジストリに登録されたデバイスの場合、次のポリシーは、モノの名前に一致するクライアント ID AWS IoT Core で に接続し、デバイスの認証に使用される証明書のSubject.CommonName.2フィールドが に設定されているadmin/ときに、名前にプレフィックスが付けられたトピックに発行するアクセス許可を付与します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 レジストリに登録されていないデバイスの場合、次のポリシーは、クライアント IDs client1、、client3および AWS IoT Core を使用して に接続しclient2、デバイスの認証に使用

される証明書のSubject.CommonName.2フィールドが に設定されている場合admin/、名前にプレフィックスが付けられたトピックに発行するアクセス許可を付与します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 レジストリに登録されているデバイスの場合、次のポリシーにより、デバイスの認証に使用される証明書のいずれかのSubject.CommonNameフィールドThingNameが に設定されている場合、デバイスはモノの名前を使用して、 admin/が続く特定のトピックに発行できます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/
${iot:Connection.Thing.ThingName}"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
        }
    ]
}
```

AWS IoT Core レジストリに登録されていないデバイスの場合、次のポリシーは、デバイスの認証に使用される証明書のSubject.CommonNameフィールドのいずれかが に設定されているadmin場合に、クライアント IDs client1、client2、client3および AWS IoT Core で に接続してトピックに発行するアクセス許可を付与します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"
            }
        }
        }
    ]
}
```

モノのポリシーの例

次のポリシーでは、 で認証に使用される証明書 AWS IoT Core が、ポリシーが評価されているモノにアタッチされている場合、デバイスが接続することを許可します。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":["iot:Connect"],
            "Resource":[ "*" ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
            }
        }
        }
    ]
}
```

以下のポリシーでは、証明書が特定のモノのタイプを持つモノにアタッチされ、そのモノが値 `attributeValue` を持つ `attributeName` の属性を持つ場合、デバイスの公開を許可します。モ

ノのポリシー変数の詳細については、[Thing policy variables] (モノのポリシー変数) を参照してください。

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

次のポリシーでは、デバイスはモノの属性で始まるトピックにパブリッシュすることができます。デバイス証明書がモノに関連付けられていない場合、この変数は解決されず、アクセス拒否エラーが発生します。モノのポリシー変数の詳細については、[Thing policy variables] (モノのポリシー変数) を参照してください。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.Attributes[attributeName]}/*"
```

```
    }
  ]
}
```

基本的なジョブポリシーの例

このサンプルでは、単一のデバイスであるジョブターゲットがジョブリクエストを受信し、ジョブ実行ステータスを AWS IoTと通信するために必要なポリシーステートメントについて説明します。

us-west-2:57EXAMPLE833 を AWS リージョン、コロン文字 (:)、12 桁の AWS アカウント 数字に置き換え、*uniqueThingName* をデバイスを表すモノのリソースの名前に置き換えます 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 ID を使用した承認

Amazon Cognito ID には、未認証と認証済みの 2 種類あります。アプリが認証されていない Amazon Cognito ID をサポートしている場合、認証が実行されないため、ユーザーが誰であるかを知りません。

認証されていない ID: 認証されていない Amazon Cognito ID の場合、認証されていない ID プールに IAM ロールをアタッチしてアクセス許可を付与します。不明なユーザーが使用できるようにするそれらのリソースにのみアクセスを許可する必要があります。

> **Important**
>
> 認証されていない Amazon Cognito ユーザーが に接続する場合は AWS IoT Core、IAM ポリシーで非常に制限されたリソースへのアクセスを許可することをお勧めします。

認証された ID: 認証された Amazon Cognito ID には、2 つの場所でアクセス許可を指定する必要があります。

- 認証された Amazon Cognito ID に IAM ポリシーをアタッチし、
- Amazon Cognito ID (認証済みユーザー) に AWS IoT Core ポリシーをアタッチします。

に接続する認証されていない Amazon Cognito ユーザーのポリシー例 AWS IoT Core

次の例は、Amazon Cognito ID の IAM ポリシーと IoT ポリシーの両方におけるアクセス許可を示しています。認証されたユーザーは、デバイス固有のトピック (デバイス/DEVICE_ID/ステータスなど) に公開したいと考えています。

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

次の例は、Amazon Cognito の認証されていないロールの IAM ポリシー内のアクセス許可を示しています。認証されていないユーザーは、認証を必要としない非デバイス固有のトピックに発行したいと考えています。

```
{
    "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 の例

次の GitHub でのウェブアプリケーションの例は、認証されたユーザーへのポリシーの添付をユーザーのサインアップと認証プロセスに組み込む方法を示しています。

- AWS Amplify と を使用した MQTT パブリッシュ/サブスクライブ React ウェブアプリケーション AWS IoT Device SDK for JavaScript
- AWS Amplify、、 AWS IoT Device SDK for JavaScriptおよび Lambda 関数を使用した MQTT パブリッシュ/サブスクライブ React ウェブアプリケーション

Amplify は、 サービスと統合するウェブおよびモバイルアプリケーションの構築に役立つ一連のツールと AWS サービスです。Amplify の詳細については、Amplify Framework Documentation を参照してください。

どちらの例でも、次の手順を実行します。

1. ユーザーがアカウントにサインアップすると、アプリケーションが Amazon Cognito ユーザープールおよび ID を作成します。
2. ユーザーが認証されると、アプリケーションがポリシーを作成し、ID にアタッチします。これにより、ユーザーは発行およびサブスクライブのアクセス許可を与えられます。
3. ユーザーは、アプリケーションを使用して MQTT トピックを発行およびサブスクライブできます。

最初の例では、認証オペレーション内で直接 `AttachPolicy` API オペレーションを使用します。次の例は、Amplify および AWS IoT Device SDK for JavaScriptを使用する React ウェブアプリケーション内に、この API コールを実装する方法を示しています。

```
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);
           }
      });
}
```

このコードは、AuthDisplay.js ファイルに表示されます。

2 番目の例では、Lambda 関数に AttachPolicy API オペレーションを実装します。次の例は、Lambda がこの API コールを使用する方法を示しています。

```
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});
     }
 });
```

このコードは、app.js ファイルの iot.GetPolicy 関数内に表示されます。

> **Note**
>
> Amazon Cognito ID プールを介して取得した AWS 認証情報を使用して関数を呼び出すと、Lambda 関数のコンテキストオブジェクトに の値が含まれますcontext.cognito_identity_id。詳細については、以下を参照してください。
>
> - AWS Lambda Node.js のコンテキストオブジェクト
> - AWS Lambda Python のコンテキストオブジェクト
> - AWS Lambda Ruby のコンテキストオブジェクト

- AWS Lambda Java のコンテキストオブジェクト
- AWS Lambda Go のコンテキストオブジェクト
- AWS Lambda C# のコンテキストオブジェクト
- AWS Lambda PowerShell のコンテキストオブジェクト

AWS IoT Core 認証情報プロバイダーを使用した AWS サービスへの直接呼び出しの許可

デバイスは X.509 証明書を使用して、TLS 相互認証プロトコル AWS IoT Core を使用して に接続できます。他の AWS サービスは証明書ベースの認証をサポートしていませんが、AWS 署名バージョン 4 形式の AWS 認証情報を使用して呼び出すことができます。署名バージョン 4 アルゴリズムでは、通常、呼び出し元にアクセスキー ID とシークレットアクセスキーが必要です。 には、組み込みの X.509 証明書を一意のデバイス ID として使用して AWS リクエストを認証できる認証情報プロバイダー AWS IoT Core があります。これによって、デバイスにアクセスキー ID およびシークレットアクセスキーを保存する必要がなくなります。

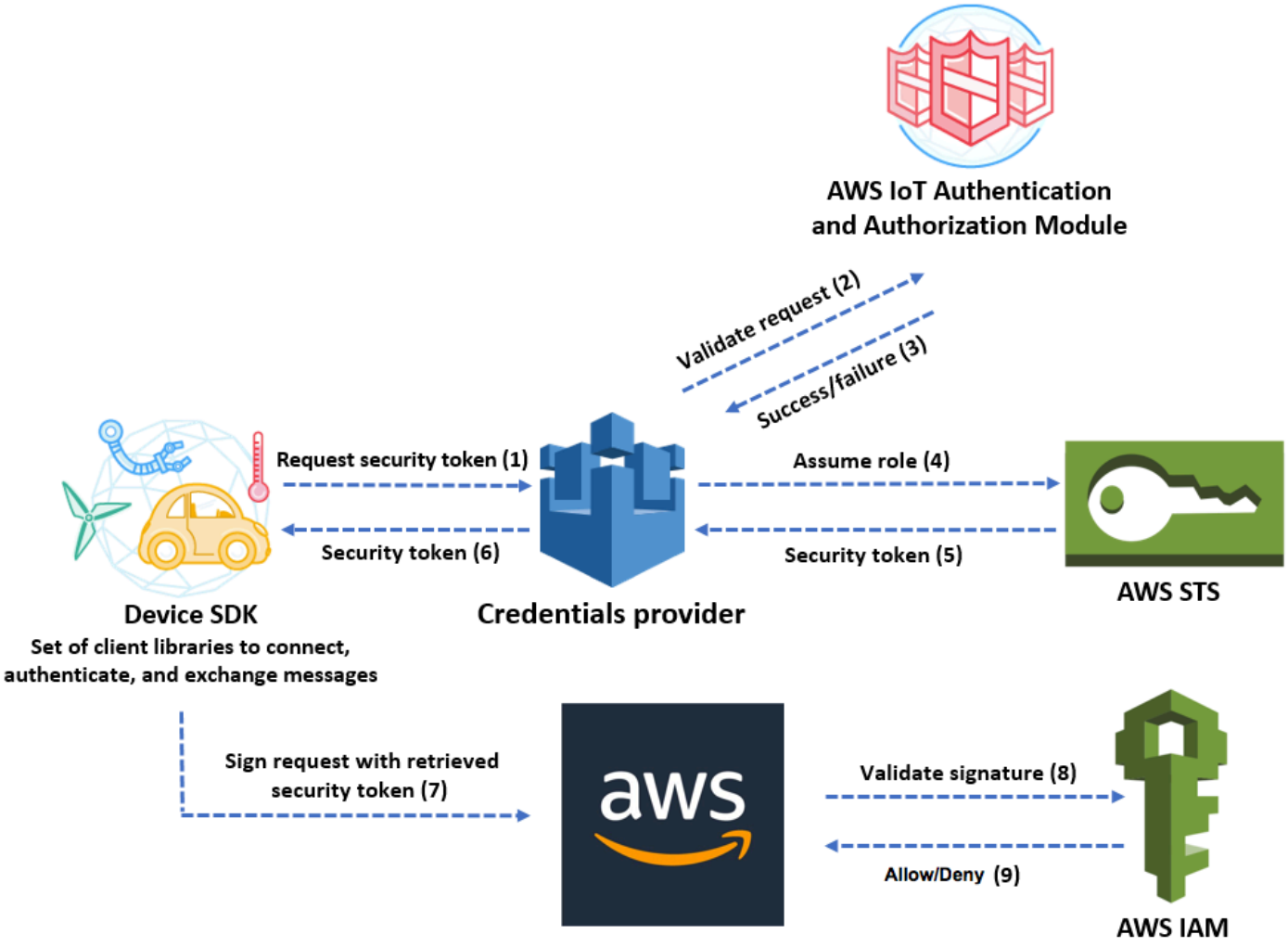
認証情報プロバイダーは、X.509 証明書を使用して発信者を認証し、一時的で制限された権限のセキュリティトークンを発行します。トークンは、任意の AWS リクエストに署名して認証するために使用できます。この方法で AWS リクエストを認証するには、 AWS Identity and Access Management (IAM) ロールを作成して設定し、適切な IAM ポリシーをロールにアタッチして、認証情報プロバイダーがユーザーに代わってロールを引き受けられるようにする必要があります。 AWS IoT Core と IAM の詳細については、の ID とアクセスの管理 AWS IoT を参照してください。

AWS IoT では、デバイスが Server Name Indication (SNI) 拡張機能を Transport Layer Security (TLS) プロトコルに送信し、 `host_name`フィールドに完全なエンドポイントアドレスを指定する必要があります。`host_name` フィールドには、呼び出すエンドポイントが含まれている必要があります。次のようになる必要があります。

- `aws iot describe-endpoint --endpoint-type iot:CredentialProvider` によって返される `endpointAddress`。

正しい `host_name` 値を持たないデバイスによって試行された接続は失敗します。

次の図は認証情報プロバイダーのワークフローを示しています。



1. AWS IoT Core デバイスは、セキュリティトークンの認証情報プロバイダーに HTTPS リクエストを行います。このリクエストには、認証のためのデバイスの X.509 証明書が含まれています。
2. 認証情報プロバイダーは AWS IoT Core 認証および認可モジュールにリクエストを転送して証明書を検証し、デバイスにセキュリティトークンをリクエストする権限があることを検証します。
3. 証明書が有効であり、セキュリティトークンをリクエストする許可がある場合、 AWS IoT Core 認証および認可モジュールは成功を返します。それ以外の場合は、デバイスに例外が送信されます。
4. 証明書の検証が成功したら、認証情報プロバイダーは AWS Security Token Service (AWS STS) を呼び出して、そのために作成した IAM ロールを引き受けます。
5. AWS STS は、権限が制限された一時的なセキュリティトークンを認証情報プロバイダーに返します。
6. 認証情報プロバイダーは、デバイスにセキュリティトークンを返します。

7. デバイスはセキュリティトークンを使用して、署名バージョン 4 で AWS AWS リクエストに署名します。
8. リクエストされたサービスは IAM を呼び出して署名を検証し、認証情報プロバイダーに作成した IAM ロールにアタッチされたアクセスポリシーに対するリクエストを認可します。
9. IAM が署名の検証に成功してリクエストを認可したら、リクエストは成功します。それ以外の場合、IAM は例外を送信します。

次のセクションでは、証明書を使用してセキュリティトークンを取得する方法を説明します。これは、すでにデバイスが登録され、そのデバイスに独自の証明書を作成して有効化していることを前提に記述されています。

証明書を使用してセキュリティトークンを取得する方法

1. デバイスに代わって認証情報プロバイダーが引き受ける IAM ロールを設定します。次の信頼ポリシーをロールにアタッチします。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"Service": "credentials.iot.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }
}
```

   呼び出す AWS サービスごとに、ロールにアクセスポリシーをアタッチします。認証情報プロバイダーは次のポリシー変数をサポートしています。

   - `credentials-iot:ThingName`
   - `credentials-iot:ThingTypeName`
   - `credentials-iot:AwsCertificateId`

   デバイスが AWS サービスへのリクエストでモノの名前を提供すると、認証情報プロバイダーは `credentials-iot:ThingName` および `credentials-iot:ThingTypeName` をコンテキスト変数としてセキュリティトークンに追加します。デバイスがリクエストでモノの名前を提供していなくても、認証情報プロバイダーは `credentials-iot:AwsCertificateId` をコンテキ

スト変数として提供します。x-amzn-iot-thingname HTTP リクエストヘッダーの値としてモノの名前を渡します。

上記の 3 つの変数は IAM ポリシーのみで機能し、 AWS IoT Core ポリシーには使用できません。

2. 次のステップ (ロールエイリアスの作成) を実行するユーザーに新しく作成したロールを AWS IoT Coreに渡す権限があることを確認します。次のポリシーは、 iam:GetRoleと の両方のiam:PassRoleアクセス許可を AWS ユーザーに付与します。iam:GetRole アクセス許可は、今作成したロールに関する情報をユーザーが取得できるようにします。アクセスiam:PassRole許可により、ユーザーはロールを別の AWS サービスに渡すことができます。

```
{
    "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 ロールエイリアスを作成します。 AWS サービスへの直接呼び出しを行うデバイスは、 への接続時に使用するロール ARN を認識している必要があります AWS IoT Core。ロールの ARN のロールをハードコーディングすることは、ロールの ARN が変わるたびにデバイスを更新する必要があるため、良いソリューションではありません。CreateRoleAlias API を使用してロールの ARN を指し示すロールエイリアスを作成することが、より優れたソリューションです。ロールの ARN が変更された場合には、ロールエイリアスを更新だけです。デバイスに変更を加える必要はありません。この API には以下のパラメータがあります。

   roleAlias

   必須。ロールのエイリアスを識別する任意の文字列。これは、ロールエイリアスデータモデルのプライマリキーとして機能します。これには 1 ~ 128 文字を使用でき、英数字および =、@、- 記号のみを含めることができます。大文字および小文字のアルファベット文字を使用できます。ロールエイリアス名では、大文字と小文字が区別されます。

`roleArn`

必須。ロールエイリアスが参照するロールの ARN。

`credentialDurationSeconds`

オプション。認証情報が有効な時間 (秒単位) です。最小値は 900 秒 (15 分) です。最大値は 43,200 秒 (12 時間) です。デフォルト値は 3,600 秒 (1 時間) です。

> **Important**
>
> AWS IoT Core 認証情報プロバイダーは、最大有効期間が 43,200 秒 (12 時間) の認証情報を発行できます。認証情報を最大 12 時間まで有効にすると、クレデンシャルを長くキャッシュすることで、クレデンシャルプロバイダーへの呼び出し回数を減らすことができます。
>
> `credentialDurationSeconds` 値は、ロールのエイリアスが参照する IAM ロールの最長セッション時間と同じかそれ以下である必要があります。詳細については、「 Identity and Access Management ユーザーガイド」の「Modifying a role maximum session duration (AWS API) AWS 」を参照してください。

この API の詳細については、「CreateRoleAlias」を参照してください。

4. ポリシーをデバイス証明書にアタッチします。デバイス証明書にアタッチされているポリシーは、デバイスにそのロールを引き受ける権限を付与する必要があります。これを行うには、次の例のように、`iot:AssumeRoleWithCertificate` アクションへのアクセス許可をロールエイリアスに付与します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/your role
 alias"
        }
    ]
}
```

5. 認証情報プロバイダーにセキュリティトークンを取得する HTTPS リクエストを行います。以下の情報を提供します。

   - Certificate: これは、TLS 相互認証を介した HTTP リクエストであるため、リクエスト作成中に証明書およびプライベートキーをクライアントに提供する必要があります。証明書の登録に使用したのと同じ証明書とプライベートキーを使用します AWS IoT Core。

     デバイスが AWS IoT Core (偽装するサービスではなく) と通信していることを確認するには、「サーバー認証」を参照し、リンクに従って適切な CA 証明書をダウンロードしてから、デバイスにコピーします。
   - RoleAlias: 認証情報プロバイダーに作成したロールエイリアスの名前。ロールエイリアス名では大文字と小文字が区別され、 で作成されたロールエイリアスと一致する必要がありますAWS IoT Core。
   - ThingName: モノを登録したときに作成した AWS IoT Core モノの名前。これは、`x-amzn-iot-thingname` HTTP ヘッダーの値として渡されます。この値は、 AWS IoT Core または IAM ポリシーでポリシー変数としてモノの属性を使用している場合にのみ必要です。

     > **Note**
     >
     > `x-amzn-iot-thingname` で提供する ThingName は証明書に割り当てられた AWS IoT Thing リソースの名前と一致する必要があります。一致しない場合は、403 エラーが返されます。

   で次のコマンドを実行して AWS CLI 、 の認証情報プロバイダーエンドポイントを取得しますAWS アカウント。この API についての詳細は、「DescribeEndpoint」を参照してください。FIPS 対応のエンドポイントについては、「AWS IoT Core- 認証情報プロバイダーエンドポイント」を参照してください。

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

   次の JSON オブジェクトは、describe-endpoint コマンドの出力サンプルです。これには、セキュリティトークンをリクエストするために使用する `endpointAddress` が含まれています。

   ```
   {
       "endpointAddress": "your_aws_account_specific_prefix.credentials.iot.your
    region.amazonaws.com"
   ```

```
}
```

エンドポイントを使用して、セキュリティトークンを返す HTTPS リクエストを認証情報プロバイダーに作成します。次のコマンド例では、curl を使用していますが、任意の HTTP クライアントを使用できます。

> **Note**
>
> roleAlias 名では大文字と小文字が区別され、作成されたロールエイリアスと一致する必要があります 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
```

このコマンドは、accessKeyId、secretAccessKey、sessionToken および有効期限を含むセキュリティトークンオブジェクトを返します。次の JSON オブジェクトは、curl コマンドの出力サンプルです。

```
   {"credentials":{"accessKeyId":"access key","secretAccessKey":"secret access
 key","sessionToken":"session token","expiration":"2018-01-18T09:18:06Z"}}
```

その後、accessKeyId、、secretAccessKeyおよび sessionToken値を使用して、 AWS サービスへのリクエストに署名できます。end-to-endのデモンストレーションについては、AWS セキュリティブログのAWS 「認証情報プロバイダーを使用してデバイス内のハードコードされた AWS IoT 認証情報の必要性を排除する方法」ブログ記事を参照してください。

IAM を使用したクロスアカウントアクセス

AWS IoT Core では、プリンシパルがプリンシパルが所有 AWS アカウント していない で定義されているトピックを発行またはサブスクライブできるようにします。IAM ポリシーと IAM ロールを作成し、そのポリシーをそのロールにアタッチすることで、クロスアカウントアクセスを設定します。

最初に、 AWS アカウントの他のユーザーと証明書に対して行うのと同様に、「IAM ポリシーの作成」の説明に従ってカスタマー管理 IAM ポリシーを作成します。

AWS IoT Core レジストリに登録されているデバイスの場合、次のポリシーは、デバイスのモノの名前に一致するクライアント ID AWS IoT Core を使用して に接続するデバイスと、`#####`がデバイスのモノの名前`my/topic/thing-name` である に発行するアクセス許可を付与します。

```
{
    "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 レジストリに登録されていないデバイスの場合、次のポリシーは、アカウントの (123456789012) AWS IoT Core レジストリに登録`client1`されているモノの名前を使用して、 に接続 AWS IoT Core し、 というプレフィックスが付いた名前のクライアント ID 固有のトピックに発行するアクセス許可をデバイスに付与します`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}"
            ]
        }
    ]
}
```

次に、「IAM ユーザーにアクセス許可を委任するロールの作成」のステップに従います。アクセスを共有する AWS アカウント のアカウント ID を入力します。最後に、作成したポリシーをロールにアタッチします。後で、アクセス許可を付与する AWS アカウント ID を変更する必要がある場合は、以下の信頼ポリシーの形式を使用できます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::567890123456:user/MyUser"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

でのデータ保護 AWS IoT Core

責任 AWS 共有モデル、 でのデータ保護に適用されます AWS IoT Core。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須ですが、TLS 1.3 を推奨します。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して AWS CLI AWS IoT または他の AWS のサービス を使用する場合も同様です。AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

データ保護の詳細についてはAWS 責任共有モデルと AWS セキュリティブログ GDPR の GDPR ブログ投稿を参照してください。

AWS IoT デバイスはデータを収集し、そのデータに対して何らかの操作を実行し、そのデータを別のウェブサービスに送信します。デバイスにデータを短期間保存することを選択することもできます。お客様は、保管中のデータに対するデータ保護を提供する責任があります。デバイスが にデータを送信すると AWS IoT、このセクションで後述するように、TLS 接続を介して送信されます。 AWS IoT デバイスは任意の AWS サービスにデータを送信できます。各サービスのデータセキュリティの詳細については、そのサービスのドキュメントを参照してください。CloudWatch Logs にログを書き込んで AWS IoT API コールをログに記録するように AWS IoT を設定できます AWS CloudTrail。これらのサービスのデータセキュリティの詳細については、Amazon CloudWatch の認証とアクセスコントロール」およびAWS KMS 「マネージドキーを使用した CloudTrail ログファイルの暗号化」を参照してください。

でのデータの暗号化 AWS IoT

デフォルトでは、転送中および保管中のすべての AWS IoT データは暗号化されます。転送中のデータは TLS を使用して暗号化され、保管中のデータは AWS 所有キーを使用して暗号化されます。 は Key Management Service AWS KMS keys () のカスタマー管理 (KMS AWS キー) AWS IoT をサポートしますAWS KMS。ただし、Device Advisor と AWS IoT Wireless は、顧客データの暗号化 AWS 所有のキー に のみを使用します。

のトランスポートセキュリティ AWS IoT Core

TLS (Transport Layer Security) は、コンピュータネットワーク上での安全な通信のために設計された暗号化プロトコルです。 AWS IoT Core Device Gateway では、デバイスから Gateway への接続に TLS を使用して、転送中にすべての通信を暗号化する必要があります。TLS は、 AWS IoT Core でサポートされているアプリケーションプロトコル (MQTT、HTTP、および WebSocket) の機密性を達成するために使用されます。TLS サポートは、多くのプログラミング言語とオペレーティングシステムで使用できます。内のデータは AWS 、特定の AWS サービスによって暗号化されます。他の AWS サービスのデータ暗号化の詳細については、そのサービスのセキュリティドキュメントを参照してください。

内容

- TLS プロトコル
- セキュリティポリシー
- でのトランスポートセキュリティに関する重要な注意事項 AWS IoT Core

- LoRaWAN ワイヤレスデバイスのトランスポートセキュリティ

TLS プロトコル

AWS IoT Core は、TLS プロトコルの次のバージョンをサポートしています。

- TLS 1.3
- TLS 1.2

では AWS IoT Core、ドメイン設定で TLS 設定 (TLS 1.2 および TLS 1.3 の場合) を設定できます。詳細については、「???」を参照してください。

セキュリティポリシー

セキュリティポリシーは、クライアントとサーバー間の TLS ネゴシエーション中にサポートされるプロトコルと暗号を決定する TLS プロトコルとその暗号の組み合わせです。必要に応じて、事前定義されたセキュリティポリシーを使用するようにデバイスを設定できます。 AWS IoT Core はカスタムセキュリティポリシーをサポートしていないことに注意してください。

デバイスの事前定義されたセキュリティポリシーの 1 つを接続時に選択できます AWS IoT Core。の最新の事前定義されたセキュリティポリシーの名前には、リリースされた年月に基づくバージョン情報 AWS IoT Core が含まれます。事前に定義されたデフォルトのセキュリティポリシーは `IoTSecurityPolicy_TLS13_1_2_2022_10` です。セキュリティポリシーを指定するには、 AWS IoT コンソールまたは を使用できます AWS CLI。詳細については、「???」を参照してください。

次の表は、 AWS IoT Core でサポートされる、事前に定義された最新のセキュリティポリシーの詳細を示しています。見出し行に収まるようにポリシー名から `IotSecurityPolicy_` を削除しました。

セキュリティポリシー	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 ポート	443/8443/8883	443/8443/8883	443/8443/8883	443	8443/8883	443	8443/8883
TLS Protocols							

セキュリティポリシー	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*	
TLS 1.2		✓	✓	✓	✓	✓	✓
TLS 1.3	✓	✓					
TLS Ciphers							
TLS_AES_128_GCM_SHA256	✓	✓					
TLS_AES_256_GCM_SHA384	✓	✓					
TLS_CHACHA20_POLY1305_SHA256	✓	✓					
ECDHE-RSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA		✓	✓	✓	✓	✓	✓

セキュリティポリシー	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-RSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA		✓	✓	✓	✓	✓	✓
AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
AES128-SHA256		✓	✓	✓		✓	✓
AES128-SHA		✓	✓	✓	✓	✓	✓
AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
AES256-SHA256		✓	✓	✓	✓	✓	✓
AES256-SHA		✓	✓	✓	✓	✓	✓

セキュリティポリシー	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*	
DHE-RSA-AES256-SHA						✓	✓
ECDHE-ECDSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-SHA384		✓	✓	✓	✓	✓	✓

セキュリティポリシー	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-AES256-SHA		✓	✓	✓	✓	✓	✓

> **Note**
>
> TLS12_1_0_2016_01 は、 AWS リージョン 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-1 でのみ使用できます。
> TLS12_1_0_2015_01 は AWS リージョン、ap-northeast-1、ap-southeast-1、eu-central-1、eu-west-1、us-east-1、us-west-2 でのみ使用できます。

でのトランスポートセキュリティに関する重要な注意事項 AWS IoT Core

MQTT AWS IoT Core を使用して に接続するデバイスの場合、TLS はデバイスとブローカー間の接続を暗号化し、TLS クライアント認証 AWS IoT Core を使用してデバイスを識別します。詳細については、「クライアント認証」を参照してください。HTTP AWS IoT Core を使用して に接続するデバイスの場合、TLS はデバイスとブローカー間の接続を暗号化し、認証は AWS 署名バージョン 4 に委任されます。詳細については、「AWS 全般リファレンス」で署名バージョン 4 でのリクエストの署名に関する情報を参照してください。

デバイスを に接続する場合 AWS IoT Core、サーバー名表示 (SNI) 拡張機能の送信は必要ありませんが、強くお勧めします。マルチアカウント登録、カスタムドメイン、VPC エンドポイント、設定済み TLS ポリシーなどの機能を使用するには、SNI 拡張機能を使用し、host_name フィールドに完全なエンドポイントアドレスを指定する必要があります。host_name フィールドには、呼び出すエンドポイントが含まれている必要があります。そのエンドポイントは、次のいずれかである必要があります。

- endpointAddress によって返される `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

- domainName によって返される `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"`

正しくない、または無効なhost_name値を持つデバイスで試行された接続は失敗します。 AWS IoT Core は、カスタム認証の認証タイプについて CloudWatch に失敗を記録します。

AWS IoT Core は SessionTicket TLS 拡張機能をサポートしていません。

LoRaWAN ワイヤレスデバイスのトランスポートセキュリティ

LoRaWAN デバイスは、LoRaWAN ™ SECURITY: Gemalto、Actility、および Semtech による LoRa Alliance™ 向けに作成されたホワイトペーパーで説明されているセキュリティ プラクティスに従います。

LoRaWAN デバイスでのトランスポートセキュリティの詳細については、「LoRaWAN データおよびトランスポートセキュリティ」を参照してください。

でのデータの暗号化 AWS IoT

データ保護とは、転送中 (との間で送受信される) AWS IoT Coreおよび保管中 (デバイスや他の AWS サービスに保存される) のデータを保護することです。に送信されるすべてのデータは、MQTT、HTTPS、および WebSocket プロトコルを使用して TLS 接続を介して送信され、転送中にデフォルトで保護されます。 AWS IoT Core AWS IoT Core はデバイスからデータを収集し、さらに処理するために他の AWS サービスに送信します。他の AWS のサービスのデータ暗号化の詳細については、そのサービスのセキュリティドキュメントを参照してください。詳細については、「保管時のデータ暗号化」を参照してください。

FreeRTOS は、キーストレージ、暗号化オブジェクトへのアクセス、およびセッションの管理を抽象化する PKCS #11 ライブラリを提供します。このライブラリを使用して、デバイスに保存されているデータを暗号化することはお客様の責任です。詳細については、「FreeRTOS 公開キー暗号化標準 (PKCS) #11 ライブラリ」を参照してください。

での保管時のデータ暗号化 AWS IoT Core

デフォルトでは、保管中のすべての AWS IoT Core データは AWS 所有キーを使用して暗号化されます。 は AWS Key Management Service () の対称カスタマーマネージドキー AWS IoT Core もサポートしていますAWS KMS。カスタマーマネージドキーを使用すると、 AWS アカウントの AWS KMS キーを作成、所有、管理できます。 AWS IoT Core は KMS キーを使用して保管中のデータを暗号化

します。ユーザーは、キーポリシーの作成や保守など、これらの KMS キーを完全に制御できます。これらのキーのアクセス許可を制御する AWS KMS ために にアクセスするロールの IAM ポリシーを設定することもできます。

AWS 所有キー

AWS 所有キーは、 AWS サービスが複数の AWS アカウントで使用するために所有および管理する KMS キーのコレクションです。 AWS サービスは AWS 、所有キーを使用してデータを保護します。デフォルトでは、 は AWS 所有キーを使用して保管中のデータを AWS IoT Core 暗号化します。これらのキーはサービスによって管理されます。 AWS 所有キーを表示、管理、使用することはできません。ユーザーがこれらのキーを保護するために実行する必要のあるアクションはありません。

AWS 所有キーの詳細については、「 AWS Key Management Service デベロッパーガイド」のAWS「 所有キー」を参照してください。

カスタマーマネージドキー

カスタマーマネージドキーは、作成、所有、管理する AWS アカウントの KMS キーです。ユーザーは、キーポリシーの作成や保守など、これらの AWS KMS キーを完全に制御できます。これらのキーのアクセス許可を制御する AWS KMS ために にアクセスするロールの IAM ポリシーを設定することもできます。カスタマーマネージド KMS キー AWS IoT Core を使用してデータを暗号化するように を設定できます。

カスタマーマネージドキーの詳細については、AWS Key Management Service デベロッパーガイドの「カスタマーマネージドキー」を参照してください。

でカスタマーマネージドキーにオプトインするには AWS IoT Core、次の手順に従います。

トピック

- ステップ 1 : カスタマーマネージドキーを作成する
- ステップ 2: KMS キーを使用する AWS IoT Core ためのアクセス許可を付与する IAM ロールを作成する
- ステップ 3: でカスタマーマネージドキーにオプトインする AWS IoT Core
- ステップ 4: AWS IoT Core コントロールプレーンオペレーションに必要な追加のアクセス許可
- ステップ 5: キーの管理
- ステップ 6: キーヘルスのモニタリング

ステップ 1：カスタマーマネージドキーを作成する

対称カスタマーマネージドキーは、 AWS KMS コンソールまたは AWS KMS CLI コマンドを使用して作成できます。keySpec は SYMMETRIC_DEFAULT、keyUsage は ENCRYPT_DECRYPT である必要があります。

> **Note**
>
> AWS IoT Core は、カスタマーマネージド AWS KMS キーのSYMMETRIC_DEFAULTキー仕様とENCRYPT_DECRYPTキー使用状況を持つキーのみをサポートします。

以下は、 で AWS IoT Core カスタマーマネージドキーに使用できる KMS キーを作成する AWS CLI コマンドの例です。

```
aws kms create-key --key-spec SYMMETRIC_DEFAULT --key-usage ENCRYPT_DECRYPT --region
 us-west-2
```

以下は、コマンドの出力例です。

```
{
    "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
    }
}
```

詳細については、「AWS Key Management Service デベロッパーガイド」の「対称カスタマーマネージドキーを作成する」を参照してください。

キーポリシー

カスタマーマネージドキーの作成時に、キーポリシーを指定することができます。キーポリシーは、カスタマーマネージドキーへのアクセスを制御します。すべてのカスタマーマネージドキーには、キーポリシーが 1 つだけ必要です。このポリシーには、そのキーを使用できるユーザーとその使用方法を決定するステートメントが含まれています。詳細については、「AWS Key Management Service デベロッパーガイド」の「キーポリシー」を参照してください。

AWS IoT Core は、アカウントの IAM ロールを使用してカスタマーマネージドキーにアクセスします。カスタムキーポリシーを使用している場合は、このキーで作成された IAM ロールに次のアクセス許可があることを確認してください。

- `kms:DescribeKey`
- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:ReEncryptTo`
- `kms:ReEncryptFrom`

ステップ 2: KMS キーを使用する AWS IoT Core ためのアクセス許可を付与する IAM ロールを作成する

が作成した KMS キーを使用して保管中のデータを暗号化 AWS IoT Core するには、アカウントに IAM ロールも作成する必要があります。このロールは、 が KMS キーにアクセスするために引き受け AWS IoT Core ることができます。

がロールを引き受ける AWS IoT Core には、ロールに次の信頼ポリシーが必要です。

```
{
    "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 ロールにアタッチされた IAM ポリシーに、KMS キーに対する次のアクセス許可が設定されていることを確認します。

- kms:DescribeKey
- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo
- kms:ReEncryptFrom

以下は、カスタマーマネージドキーに必要なアクセス許可が設定された IAM ポリシーの例です。

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

詳細については、「AWS Identity and Access Management ユーザーガイド」の「IAM ユーザーにアクセス許可を付与するロールを作成する」を参照してください。

ステップ 3: でカスタマーマネージドキーにオプトインする AWS IoT Core

前のすべてのステップを完了したら、update-encryption-configuration CLI コマンドを実行し、 AWS IoT Coreでカスタマーマネージドキーを使用してオプトインします。カスタマーマネージドキーにオプトインすると、 AWS アカウント内のすべての AWS IoT Core リソースは指定された AWS KMS キーを使用して暗号化されます。

1. AWS IoT Core を使用して でカスタマーマネージドキーにオプトインするには AWS CLI、update-encryption-configuration CLI コマンドを実行します。

   ```
   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. AWS IoT Core を使用して でカスタマーマネージドキーを検証するには AWS CLI、 describe-encryption-configuration CLI コマンドを実行します。

   ```
   aws iot describe-encryption-configuration --region us-west-2
   ```

   でカスタマーマネージドキーを有効にしている場合 AWS IoT Core、出力は次のようになります。

   ```
   {
       "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` フィールドは、暗号化設定が最後に更新された日付を示します。

カスタマーマネージドキーを有効にしていない場合、出力は次のようになります。

```
{
    "encryptionType": "AWS_OWNED_KMS_KEY",
    "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
}
```

ステップ 4: AWS IoT Core コントロールプレーンオペレーションに必要な追加のアクセス許可

カスタマーマネージドキーにオプトインすると、 AWS アカウントに属するすべての AWS IoT Core リソースは、提供された KMS キーで暗号化されます。すべてのコントロールプレーンオペレーションでは、 AWS IoT Core リソースに対する特定のオペレーションに必要な`kms:Decrypt`アクセス許可に加えて、呼び出し元に KMS キーに対するアクセス許可が必要になりました。発信者に `kms:Decrypt` アクセス許可がなく、データの暗号化または復号化を必要とする API コール (`GetPolicy` など) を行った場合、発信者は `UnauthorizedException` を受け取ります。

たとえば、`GetPolicy` を呼び出す場合、API コールを成功させるには、カスタマーマネージド KMS キーに対する `iot:GetPolicy` と `kms:Decrypt` の両方のアクセス許可が必要です。

> **Note**
>
> IAM ユーザーまたはロールを更新して暗号化設定に使用されるキーに対する AWS KMS アクセス許可を付与する場合は、KMS キーポリシーによって、各 IAM ユーザーまたはロールに必要なアクセス許可も付与されていることを確認してください。

AWS KMS の アクセス許可 **`UpdateEncryptionConfiguration`**

`UpdateEncryptionConfiguration` API コールでは、カスタマーマネージドキーにオプトインしたり、キー設定を変更したりするために、KMS キーに対する次の AWS KMS アクセス許可が必要です。

- kms:DescribeKey
- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKeyWithoutPlaintext
- kms:ReEncryptTo
- kms:ReEncryptFrom

AWS KMS 他のすべてのコントロールプレーン APIsの アクセス許可

ほとんどのコントロールプレーン API では、カスタマーマネージドキーが有効になっている場合に kms:Decrypt アクセス許可が必要です。ただし、特定の API では、これらの追加のアクセス許可は必要ありません。

AWS KMS アクセス許可を必要としない APIs

List* および Delete* API はこの区分には該当しません。お客様はいつでも List* または Delete* コントロールプレーン API を呼び出すことができます。発信者に kms:Decrypt アクセス許可がない場合でも、これらの API コールは成功します。これらの API コールは、カスタマーマネージドキーが異常である場合でも成功します。これは、List* と Delete* API が復号化を一切行わないためです。

- List* API – すべての一覧表示操作 (例: ListThings、ListPolicies、ListCertificates)
- Delete* API – すべての削除操作 (例: DeleteThing、DeletePolicy、DeleteCertificate)

ステップ 5: キーの管理

AWS IoT Core は、カスタマーマネージドキー設定を定期的にチェックして、暗号化および復号オペレーションに影響がないことを確認します。これらのヘルスチェックは 1 AWS IoT Core分に 1 回実行され、 AWS KMS キーおよび関連する IAM ロールの両方にアクセスして使用してオペレーションを暗号化および復号する機能を検証します。

HEALTHY

AWS IoT Core は、指定された IAM ロールを介して AWS KMS キーに正常にアクセスし、暗号化/復号オペレーションを実行できます。すべてのコンポーネントが正しく機能しています。

UNHEALTHY

AWS IoT Core は AWS KMS キーにアクセスまたは使用できません。これにより、新しい暗号化操作を実行できなくなり、サービス機能に影響が出る可能性があります。errorCode フィールドは、問題がキーにあるのか、IAM ロールにあるのかを示します。

キーのヘルスに影響を与える可能性のあるお客様によるアクション

いくつかのお客様のアクションが引き金となって、キーのヘルスステータスが HEALTHY から UNHEALTHY に変わることがあります。

キー関連のアクション

- AWS KMS キーの削除 – キーの削除をスケジュールすると、そのキーは Pending deletionステータスになり、使用できません。
- AWS KMS キーの無効化 – KMS キーを無効にすると、暗号化/復号オペレーションに使用できなくなります。
- 削除のためのキーのスケジュール – 削除が完了するとキーは使用できなくなります。
- キーポリシーの変更 – AWS IoT Core アクセスに必要なアクセス許可の削除
- キー使用許可の変更 – 必要な AWS KMS アクションの制限

IAM ロール関連のアクション

- IAM ロールの削除 – AWS IoT Core キーにアクセスするためのロールを引き受けることはできません
- ロールのアクセス許可の変更 — ロールポリシーから必要な AWS KMS アクセス許可を削除する
- 信頼ポリシーの変更 — AWS IoT Core サービスがロールを引き受けるのを防ぐ
- 制限条件の追加 – がロール AWS IoT Core を使用できないようにする条件

アカウントレベルのアクション

- クロスアカウントキーアクセスの変更 – 異なるアカウントのキーのアクセス許可の変更。
- サービスコントロールポリシー (SCPs) – AWS KMS アクセスを制限する組織レベルのポリシー
- アカウントレベルの IAM ポリシー – キーアクセスを上書きする、またはキーアクセスと競合するポリシー。

> **Important**
>
> で使用される AWS KMS キー、IAM ロール、またはポリシーへの変更は、まず開発環境でテスト AWS IoT Core する必要があります。変更を加えた後、主要なヘルスステータスを注意深くモニタリングして、 AWS IoT Core 機能に影響がないことを確認します。

暗号化設定の更新

の暗号化設定を更新 AWS IoT Core して、あるカスタマーマネージドキーから別のカスタマーマネージドキーに変更するか、 AWS 所有キーとカスタマーマネージドキーの間で変更します。

設定を別のカスタマーマネージドキーに変更するには:

1. 「the section called “ステップ 1：カスタマーマネージドキーを作成する”」の手順に従って、新しいカスタマーマネージドキーを作成します。
2. 更新期間中には古いキーと新しいキーの両方のアクセス許可を含めるように、IAM ロールポリシーを更新します。
3. 新しいキーを使用するように暗号化設定を更新します。

```
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"
```

カスタマーマネージドキーから AWS 所有キーに戻すには:

```
aws iot update-encryption-configuration --encryption-type "AWS_OWNED_KMS_KEY"
```

> **Note**
>
> 新しいカスタマーマネージドキーの暗号化設定を更新するときは、操作を成功させるために、古いキーと新しいキーの両方がアクセス可能な状態であることを確認します。

一般的な障害シナリオとその影響

次の表は、キーが削除または非アクティブ化される一般的な障害シナリオを示しています。

シナリオ	即時的な影響	長期的な影響
キーの無効化	すべての新しい暗号化/復号化の操作がすぐに失敗する	キーが再有効化または置き換えられるまでサービスが中断
キー削除のスケジュール	キーのステータスが削除保留中に変更され、すべての暗号化/復号化操作が失敗する	削除完了時に自動サービスの障害が発生する
キーの完全削除	すべての操作の即時および永続的な障害	永久的にデータが損失され、暗号化されたデータの復元が不可能
キーポリシーの不適切な変更	AWS IoT Core はキーへのアクセス許可を失います	ポリシーが修正されるまでのサービス障害が続く
IAM ロールの削除	AWS IoT Core はキーにアクセスするためのロールを引き受けることができません	暗号化サービスが完全に停止
IAM ロールが正しく変更されていない	AWS IoT Core がロールを引き受けたり、ロールを使用してキーにアクセスしたりできない	IAM ロールが修正されるまでサービス障害が続く

予防とベストプラクティス

キーの偶発的な削除や非アクティブ化を防ぎ、サービス障害のリスクを最小限に抑えるには:

キーのライフサイクルポリシーを実装する

キーの作成、ローテーション、廃止に関する明確な手順を確立します。どのキーがどの AWS IoT Core リソースで使用されるかを文書化し、アクティブなキーのインベントリを維持します。

IAM ポリシーを使用してキーの削除を制限する

権限のないユーザーが重要な暗号化キーを削除または無効化できないようにする IAM ポリシーを作成します。条件を使用して、キー削除の操作に対し追加承認を要求します。

CloudTrail ログ記録を有効にする

CloudTrail を使用してすべての AWS KMS キーオペレーションをモニタリングし、不正または偶発的なキー管理アクティビティを検出します。キーの削除、無効化、またはポリシーの変更に関するアラートを設定します。

キーの交換手順をテストする

非本番環境でキーの交換手順を定期的にテストして、キー関連の障害から迅速に復旧できることを確認します。

キーのバックアップを維持する

AWS KMS キーマテリアルをエクスポートすることはできませんが、必要に応じて迅速なキー交換を容易にするために、キー ARNs、ポリシー、および関連する AWS IoT Core 設定の詳細な記録を維持します。

キーのヘルスをモニタリングする

`CMK.Health` メトリクスを継続的にモニタリングし、主要なヘルスステータスの変更に関する自動アラートを設定します。キー関連の問題に迅速に対処するための自動応答を実装します。

> **Important**
>
> キーの更新手順は、本番環境に実装する前に、必ず開発環境でテストしてください。ロールバック計画を文書化し、緊急時にキー交換手順を迅速に実行できるようにします。

ステップ 6: キーヘルスのモニタリング

定期的なチェック AWS IoT Core の実行の一環として、CloudWatch メトリクスとログが出力され、カスタマーマネージドキー設定のヘルスステータスが可視化されます。

AWS IoT Core は、少なくとも 1 分に 1 回 CloudWatch に`CMK.Health`メトリクスを出力します。メトリクスは、 がデータの暗号化と復号 AWS IoT Core に使用するカスタマーマネージドキーのヘルスステータスに関する情報を提供します。

この `CMK.Health` パラメータは以下の値を取ることができます。

- 値は です`1`。 AWS IoT Core データの暗号化と復号に暗号化キーを正常に使用できます。
- 値は です`0`。 AWS IoT Core はデータの暗号化と復号に暗号化キーを使用できません。

AWS IoT Core 暗号化キーのヘルスステータスが変更されると、 も AWS IoT V2 ログを出力します。これらのログは、ヘルスステータスの更新に関する追加の詳細を提供します。これらのログを表示するには、 AWS IoT V2 ログを有効にする必要があります。HEALTHY ログは INFO レベルで出力され、UNHEALTHY ログは ERROR レベルで出力されます。ログレベルの詳細については、「ログレベル」を参照してください。

次の例は、カスタマーマネージドキーのヘルスステータスの更新を示す AWS IoT Core ために から出力される CloudWatch ログエントリです。

重要なヘルスステータスの変更を効果的にモニタリングして対応するには:

1. CMK.Health メトリクスの CloudWatch アラームを設定します。

```
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. 詳細なヘルスステータス変更イベントをエラーコードとメッセージでキャプチャするための Enable AWS IoT V2 ログ記録。
3. トラブルシューティングの設定ステータスを確認します。

```
aws iot describe-encryption-configuration --region us-west-2
```

4. errorCode フィールドを確認して UNHEALTHY ステータスを調べます。
   - KMS_KEY_VALIDATION_ERROR – AWS KMS キーの問題 (無効、削除、またはポリシーの問題)
   - ROLE_VALIDATION_ERROR – IAM ロールの問題 (削除やポリシーの問題、または信頼の問題)

UNHEALTHY から HEALTHY へ変更

暗号化キーのステータスが から UNHEALTHYに更新されるとHEALTHY、 AWS IoT Core は次の形式で AWS IoT V2 ログメッセージを出力します。

```
{
    "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"
}
```

HEALTHY から UNHEALTHY へ変更

暗号化キーのステータスが から HEALTHYに更新されるとUNHEALTHY、 AWS IoT Core は次の形式で AWS IoT V2 ログメッセージを出力します。

```
{
    "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**
>
> キーの状態が になるとUNHEALTHY、 AWS IoT Core オペレーションはすぐに失敗します。この場合は、キー設定、IAM ロールのアクセス許可、およびポリシーを確認してください。CMK.Health メトリクスでステータスの変更をモニタリングします。設定を確認してもオペレーションが失敗し続ける場合は、アカウントマネージャーまたは AWS サポートセンターにお問い合わせください。

AWS CloudTrail イベント

また、復号オペレーションを暗号化するための KMS キー AWS IoT Coreの使用状況をモニタリングすることもできます。 AWS IoT Core はDescribeKey、KMS キーに対して Decrypt、ReEncrypt、、および GenerateDataKeyWithoutPlaintextオペレーションを行い、保管中の AWS アカウントに属するデータを暗号化/復号します。

DescribeKey、Decrypt、ReEncryptおよび GenerateDataKeyWithoutPlaintext には CloudTrail イベントがあります。これらのイベントは、カスタマーマネージドキーによって暗号化されたデータにアクセス AWS IoT Core するために によって呼び出される AWS KMS オペレーションをモニタリングします。

Decrypt の例

```
{
    "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"
}
```


の ID とアクセスの管理 AWS IoT

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS IoT リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- オーディエンス
- IAM アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理

- が IAM と AWS IoT 連携する方法
- AWS IoT アイデンティティベースのポリシーの例
- AWS の 管理ポリシー AWS IoT
- AWS IoT ID とアクセスのトラブルシューティング

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「AWS IoT ID とアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「が IAM と AWS IoT 連携する方法」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「AWS IoT アイデンティティベースのポリシーの例」を参照)

IAM アイデンティティを使用した認証

AWS IoT ID には、デバイス (X.509) 証明書、Amazon Cognito ID、IAM ユーザーまたはグループを使用できます。このトピックでは、IAM ID のみについて説明します。が AWS IoT サポートする他の ID の詳細については、「」を参照してくださいクライアント認証。

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM Identity Center (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサインインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティか

ら始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスすることを人間のユーザーに要求する AWS」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

アイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティ (ユーザー、グループ、またはロール) にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイド の アクセスコントロールリスト (ACL) の概要 を参照してください。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

が IAM と AWS IoT 連携する方法

IAM を使用して へのアクセスを管理する前に AWS IoT、使用できる IAM 機能を理解しておく必要があります AWS IoT。 AWS IoT およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

トピック

- AWS IoT ID ベースのポリシー
- AWS IoT リソースベースのポリシー
- AWS IoT タグに基づく認可
- AWS IoT IAM ロール

AWS IoT ID ベースのポリシー

IAM アイデンティティベースのポリシーでは、許可または拒否されたアクションとリソースを指定でき、さらにアクションが許可または拒否された条件を指定できます。 AWS IoT は、特定のアク

ション、リソース、および条件キーをサポートします。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクション

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

次の表に、IAM IoT アクション、関連付けられた AWS IoT API、およびアクションが操作するリソースを示します。

ポリシーアクション	AWS IoT API	リソース
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** ARN で AWS アカウント 指定された は、証明書が転送されるアカウントである必要があります。
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	なし

ポリシーアクション	AWS IoT API	リソース
iot:AttachPolicy	AttachPolicy	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* または 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** ARN で AWS アカウント 指定された は、証明書が転送されるアカウントである必要があります。
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*

ポリシーアクション	AWS IoT API	リソース
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	なし
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*

ポリシーアクション	AWS IoT API	リソース
iot:CreatePolicyVersion	CreatePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** これは IAM AWS IoT ポリシーではなく、 ポリシーである必要があります。
iot:CreateRoleAlias	CreateRoleAlias	(パラメータ :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* 作成されているグループと親グループ用、使用されている場合
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*

ポリシーアクション	AWS IoT API	リソース
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*

ポリシーアクション	AWS IoT API	リソース
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* (パラメータ: authorizerName) なし
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	なし

ポリシーアクション	AWS IoT API	リソース
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	なし
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	なし
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	なし
iot:DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*

ポリシーアクション	AWS IoT API	リソース
iot:DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* または arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
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	なし
iot:GetJobDocument	GetJobDocument	arn:aws:iot: *region*:*account-id* :job/*job-id*

ポリシーアクション	AWS IoT API	リソース
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*
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* または arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	なし
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	なし

ポリシーアクション	AWS IoT API	リソース
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	なし
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	なし
iot:ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* thingGroupName パラメータが使用される場合
iot:ListJobTemplates	ListJobs	なし
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	なし

ポリシーアクション	AWS IoT API	リソース
iot:ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:ListThingGroups	ListThingGroups	なし
iot:ListThingGroupsForThing	ListThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	なし
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	なし
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	なし

ポリシーアクション	AWS IoT API	リソース
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	なし
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*certificate-id*
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*

ポリシーアクション	AWS IoT API	リソース
iot:StartThingRegistrationTask	StartThingRegistrationTask	なし
iot:StopThingRegistrationTask	StopThingRegistrationTask	なし
iot:TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	なし
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	なし
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	なし

ポリシーアクション	AWS IoT API	リソース
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*
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*

のポリシーアクションは、アクションの前にプレフィックス AWS IoT を使用しますiot:。たとえば、 ListThings API AWS アカウント を使用して に登録されているすべての IoT モノを一覧表示するアクセス許可をユーザーに付与するには、ポリシーに iot:ListThingsアクションを含めます。ポリシーステートメントには、 Actionまたは NotAction element を含める必要があります。は、このサービスで実行できるタスクを記述する独自のアクションのセット AWS IoT を定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": [
      "ec2:action1",
      "ec2:action2"
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "iot:Describe*"
```

AWS IoT アクションのリストを確認するには、IAM ユーザーガイドの「 で定義されるアクション AWS IoT」を参照してください。

Device Advisor のアクション

次の表は、IAM IoT Device Advisor のアクション、関連する AWS IoT Device Advisor API、およびアクションで使用されるリソースを示しています。

ポリシーアクション	AWS IoT API	リソース
iotdeviceadvisor:CreateSuiteDefinition	CreateSuiteDefinition	なし
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:L	ListSuiteDefinitions	なし

ポリシーアクション	AWS IoT API	リソース
istSuiteDefinitions		
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*
iotdeviceadvisor:StartSuiteRun	StartSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceadvisor: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*
iotdeviceadvisor: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*
iotdeviceadvisor:UpdateSuiteDefinition	UpdateSuiteDefinition	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*

ポリシーアクション	AWS IoT API	リソース
iotdeviceadvisor:StopSuiteRun	StopSuiteRun	arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*

AWS IoT Device Advisor のポリシーアクションは、アクションの前にプレフィックス を使用しますiotdeviceadvisor:。たとえば、ListSuiteDefinitions API AWS アカウント を使用して に登録されているすべてのスイート定義を一覧表示するアクセス許可をユーザーに付与するには、ポリシーにiotdeviceadvisor:ListSuiteDefinitionsアクションを含めます。

リソース

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

AWS IoT リソース

ポリシーアクション	AWS IoT API	リソース
iot:AcceptCertificateTransfer	AcceptCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id* **Note** ARN で AWS アカウント 指定された は、証明書が転送されるアカウントである必要があります。

ポリシーアクション	AWS IoT API	リソース
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	なし
iot:AttachPolicy	AttachPolicy	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* または arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:AttachPrincipalPolicy	AttachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
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** ARN で AWS アカウント 指定された は、証明書が転送されるアカウントである必要があります。
iot:CancelJob	CancelJob	arn:aws:iot: *region*:*account-id* :job/*job-id*

ポリシーアクション	AWS IoT API	リソース
iot:CancelJobExecution	CancelJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ClearDefaultAuthorizer	ClearDefaultAuthorizer	なし
iot:CreateAuthorizer	CreateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
iot:CreateCertificateFromCsr	CreateCertificateFromCsr	*
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*

ポリシーアクション	AWS IoT API	リソース
CreatePolicyVersion	iot:CreatePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** これは IAM AWS IoT ポリシーではなく、 ポリシーである必要があります。
iot:CreateRoleAlias	CreateRoleAlias	(パラメータ :roleAlias) arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-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* 作成されているグループと親グループ用、使用されている場合
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*

ポリシーアクション	AWS IoT API	リソース
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*
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*

ポリシーアクション	AWS IoT API	リソース
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* (パラメータ: authorizerName) なし
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	なし
iot:DescribeEndpoint	DescribeEndpoint	*
iot:DescribeEventConfigurations	DescribeEventConfigurations	なし
iot:DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
iot:DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*

ポリシーアクション	AWS IoT API	リソース
iot:DescribeJobExecution	DescribeJobExecution	なし
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	なし
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* または 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*

ポリシーアクション	AWS IoT API	リソース
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	なし
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*
iot:GetRegistrationCode	GetRegistrationCode	*
iot:GetTopicRule	GetTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*

ポリシーアクション	AWS IoT API	リソース
iot:ListAttachedPolicies	ListAttachedPolicies	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* または arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:ListAuthorizers	ListAuthorizers	なし
iot:ListCACertificates	ListCACertificates	*
iot:ListCertificates	ListCertificates	*
iot:ListCertificatesByCA	ListCertificatesByCA	*
iot:ListIndices	ListIndices	なし
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	なし
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	なし
iot:ListJobs	ListJobs	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* thingGroupName パラメータが使用される場合
iot:ListJobTemplates	ListJobTemplates	なし

ポリシーアクション	AWS IoT API	リソース
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	なし
iot:ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot:ListThingGroups	ListThingGroups	なし
iot:ListThingGroupsForThing	ListThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot:ListThingPrincipals	ListThingPrincipals	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

ポリシーアクション	AWS IoT API	リソース
iot:ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	なし
iot:ListThingRegistrationTasks	ListThingRegistrationTasks	なし
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	なし
iot:RegisterCACertificate	RegisterCACertificate	*
iot:RegisterCertificate	RegisterCertificate	*
iot:RegisterThing	RegisterThing	なし
iot:RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

ポリシーアクション	AWS IoT API	リソース
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	なし
iot:StopThingRegistrationTask	StopThingRegistrationTask	なし

ポリシーアクション	AWS IoT API	リソース
iot:TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot:TestInvokeAuthorizer	TestInvokeAuthorizer	なし
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	なし
iot:UpdateIndexingConfiguration	UpdateIndexingConfiguration	なし
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*

ポリシーアクション	AWS IoT API	リソース
iot:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

ARN の形式の詳細については、「Amazon リソースネーム (ARNs AWS 「サービス名前空間」を参照してください。

リソースを作成するためのアクションなど、一部の AWS IoT アクションは、特定のリソースで実行できません。このような場合はワイルドカード *を使用する必要があります。

```
"Resource": "*"
```

AWS IoT リソースタイプとその ARNs」を参照してください。 AWS IoT どのアクションで各リソースの ARN を指定できるかについては、「AWS IoTで定義されるアクション」を参照してください。

Device Advisor のリソース

AWS IoT Device Advisor IAM ポリシーのリソースレベルの制限を定義するには、スイート定義とスイート実行に次のリソース ARN 形式を使用します。

スイート定義リソース ARN 形式

```
arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id
```

スイート実行リソース ARN 形式

```
arn:aws:iotdeviceadvisor:region:account-id:suiterun/suite-definition-id/suite-run-id
```

条件キー

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

AWS IoT は独自の条件キーのセットを定義し、いくつかのグローバル条件キーの使用もサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドのAWS 「グローバル条件コンテキストキー」を参照してください。

AWS IoT 条件キー

AWS IoT 条件キー	説明	タイプ
aws:RequestTag/${*tag-key*}	ユーザーが AWS IoTに対して行うリクエストに含まれるタグキー。	String
aws:ResourceTag/${*tag-key*}	AWS IoT リソースにアタッチされたタグのタグキーコンポーネント。	String
aws:TagKeys	リクエスト内のリソースに関連付けられているすべてのタグキー名のリスト。	String

AWS IoT 条件キーのリストを確認するには、IAM ユーザーガイドの「 の条件キー AWS IoT」を参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS IoT」を参照してください。

例

AWS IoT アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS IoT アイデンティティベースのポリシーの例。

AWS IoT リソースベースのポリシー

リソースベースのポリシーは、指定されたプリンシパルが AWS IoT リソースに対して実行できるアクションと条件を指定する JSON ポリシードキュメントです。

AWS IoT は IAM リソースベースのポリシーをサポートしていません。ただし、 AWS IoT リソースベースのポリシーをサポートしています。詳細については、「AWS IoT Core ポリシー」を参照してください。

AWS IoT タグに基づく認可

AWS IoT リソースにタグをアタッチするか、 へのリクエストでタグを渡すことができます AWS IoT。タグに基づいてアクセスを制御するには、`iot:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。詳細については、「IAM ポリシーでのタグの使用」を参照してください。 AWS IoT リソースのタグ付けの詳細については、「」を参照してくださいAWS IoT リソースのタグ付け。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例を表示するには、「タグに基づく AWS IoT リソースの表示」を参照してください。

AWS IoT IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のエンティティです。

での一時的な認証情報の使用 AWS IoT

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole や GetFederationToken などの AWS STS API オペレーションを呼び出します。

AWS IoT では、一時的な認証情報の使用がサポートされています。

サービスリンクロール

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

AWS IoT は、サービスにリンクされたロールをサポートしていません。

サービス役割

この機能により、ユーザーに代わってサービスがサービスロールを引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービスロールはIAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者はこの役割の権限を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

AWS IoT アイデンティティベースのポリシーの例

デフォルトでは、IAM ユーザーおよびロールには、 AWS IoT リソースを作成または変更するアクセス許可はありません。また、 AWS マネジメントコンソール、 AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらのアクセス許可が必要なユーザーまたはグループにそのポリシーをアタッチします。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- AWS IoT コンソールの使用
- ユーザーが自分の許可を表示できるようにする
- タグに基づく AWS IoT リソースの表示
- タグに基づく AWS IoT Device Advisor リソースの表示

ポリシーに関するベストプラクティス

アイデンティティベースのポリシーは、アカウント内の AWS IoT リソースを誰かが作成、アクセス、または削除できるかどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイド の AWS マネージドポリシー または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

AWS IoT コンソールの使用

AWS IoT コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の AWS IoT リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリ

シーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

これらのエンティティが引き続き AWS IoT コンソールを使用できるようにするには、エンティティに次の AWS 管理ポリシーもアタッチします: AWSIoTFullAccess。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーが自分の許可を表示できるようにする

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

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

タグに基づく AWS IoT リソースの表示

アイデンティティベースのポリシーの条件を使用して、タグに基づいて AWS IoT リソースへのアクセスをコントロールできます。この例では、モノを表示できるポリシーを作成する方法を示します。ただし、アクセス許可は、モノタグ Owner にそのユーザーのユーザー名の値がある場合のみ、付与されます。このポリシーでは、このアクションをコンソールで実行するために必要なアクセス権限も付与します。

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

このポリシーをアカウントの IAM ユーザーにアタッチできます。という名前のユーザーが AWS IoT 請求グループを表示richard-roeしようとする場合、請求グループには Owner=richard-roeま

たは のタグを付ける必要がありますowner=richard-roe。それ以外の場合、アクセスは拒否されます。条件キー名では大文字と小文字は区別されないため、条件タグキー Owner は Owner と owner に一致します。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素：条件」を参照してください。

タグに基づく AWS IoT Device Advisor リソースの表示

アイデンティティベースのポリシーの条件を使用して、タグに基づいて AWS IoT Device Advisor リソースへのアクセスをコントロールできます。次の例は、特定のスイート定義の表示を許可するポリシーを作成する方法を示しています。ただし、アクセス許可が付与されるのは、スイート定義タグが SuiteType を MQTT の値に設定している場合のみです。このポリシーでは、このアクションをコンソールで実行するために必要なアクセス許可も付与します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewSuiteDefinition",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:GetSuiteDefinition",
            "Resource": "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/SuiteType": "MQTT"}
        }
        }
    ]
}
```

AWS の 管理ポリシー AWS IoT

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを記述するよりも AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには時間と専門知識が必要です。すぐに開始するには、 AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「 AWS 管理ポリシー」を参照してください。

AWS サービスは、 AWS 管理ポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスでは新しい機能を利用できるようにするために、 AWS マネージドポリシーに権限が追加されることがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS マネージドポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が損なわれることはありません。

さらに、 は、複数のサービスにまたがるジョブ関数の マネージドポリシー AWS をサポートしています。例えば、ReadOnlyAccess AWS 管理ポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新しい機能を起動する場合、 AWS は新たなオペレーションとリソース用に、読み取り専用の許可を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

> **Note**
>
> AWS IoT は、 AWS IoT と IAM ポリシーの両方で動作します。このトピックでは、コントロールプレーン API とデータプレーン API オペレーションのポリシーアクションを定義する IAM ポリシーのみについて説明します。AWS IoT Core ポリシー も参照してください。

AWS マネージドポリシー: AWSIoTConfigAccess

`AWSIoTConfigAccess` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、すべての AWS IoT 設定オペレーションへのアクセスを許可する、関連付けられた ID のアクセス許可を付与します。このポリシーは、データの処理とストレージに影響を与える可能性があります。でこのポリシーを表示するには AWS マネジメントコンソール、「AWSIoTConfigAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `iot` – AWS IoT データを取得し、IoT 設定アクションを実行します。

```
{
    "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 マネージドポリシー: AWSIoTConfigReadOnlyAccess

`AWSIoTConfigReadOnlyAccess` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、すべての AWS IoT 設定オペレーションへの読み取り専用アクセスを許可する、関連付けられた ID のアクセス許可を付与します。でこのポリシーを表示するには AWS マネジメントコンソール、「AWSIoTConfigReadOnlyAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- iot — IoT 設定アクションの読み取り専用オペレーションを実行します。

```
{
    "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 マネージドポリシー: AWSIoTDataAccess

AWSIoTDataAccess ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、すべての AWS IoT データオペレーションへのアクセスを許可する、関連付けられた ID アクセス許可を付与します。データオペレーションでは、MQTT または HTTP プロトコルを介してデータを送信します。 AWS マネジメントコンソールのこのポリシーを表示するには、「AWSIoTDataAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- iot – AWS IoT データを取得し、 AWS IoT メッセージングアクションへのフルアクセスを許可します。

```
{
    "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 マネージドポリシー: AWSIoTFullAccess

AWSIoTFullAccess ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、すべての AWS IoT 設定およびメッセージングオペレーションへのアクセスを許可する、関連付けられた ID のアクセス許可を付与します。でこのポリシーを表示するには AWS マネジメントコンソール、「」を参照してくださいAWSIoTFullAccess。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- iot – AWS IoT データを取得し、設定アクションとメッセージングアクションへの AWS IoT フルアクセスを許可します。
- iotjobsdata – AWS IoT Jobs データを取得し、 AWS IoT Jobs データプレーン API オペレーションへのフルアクセスを許可します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:*",
                "iotjobsdata:*"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

AWS マネージドポリシー: AWSIoTLogging

AWSIoTLogging ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、Amazon CloudWatch Logs グループを作成してグループにログをストリーミングするためのアクセスを許可する関連付けられた ID のアクセス許可を付与します。このポリシーは CloudWatch ログ記録用のロールにアタッチされます。でこのポリシーを表示するには AWS マネジメントコンソール、「」を参照してくださいAWSIoTLogging。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- logs — CloudWatch ログを取得します。また、CloudWatch Logs のグループを作成し、グループにログをストリーミングすることができます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
```

```
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS マネージドポリシー: AWSIoTOTAUpdate

AWSIoTOTAUpdate ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、ジョブの作成 AWS IoT 、署名ジョブの AWS IoT コード、および AWS コード署名者のジョブを記述するためのアクセスを許可する、関連付けられた ID アクセス許可を付与します。でこのポリシーを表示するには AWS マネジメントコンソール、「」を参照してくださいAWSIoTOTAUpdate。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- iot – AWS IoT ジョブとコード署名ジョブを作成します。
- signer – AWS コード署名者ジョブの作成を実行します。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateJob",
            "signer:DescribeSigningJob"
        ],
        "Resource": "*"
    }
}
```

AWS マネージドポリシー: AWSIoTRuleActions

`AWSIoTRuleActions` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、 AWS IoT ルールアクションでサポートされているすべての へのアクセスを許可する、関連付けられた ID アクセス許可を付与 AWS のサービスします。でこのポリシーを表示するには AWS マネジメントコンソール、「」を参照してください`AWSIoTRuleActions`。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `iot` - ルールアクションメッセージを公開するためのアクションを実行します。
- `dynamodb` - DynamoDB テーブルにメッセージを挿入するか、メッセージを DynamoDB テーブルの複数の列に分割します。
- `s3` - Amazon S3 バケットにオブジェクトを保存します。
- `kinesis` - Amazon Kinesis ストリームオブジェクトにメッセージを送信します。
- `firehose` - Firehose ストリームオブジェクトにレコードを挿入します。
- `cloudwatch` - CloudWatch アラームの状態を変更するか、CloudWatch メトリクスにメッセージデータを送信します。
- `sns` - Amazon SNS を使用して通知を発行するオペレーションを実行します。このオペレーションは SNS AWS IoT トピックを対象としています。
- `sqs` - SQS キューに追加するメッセージを挿入します。
- `es` - OpenSearch Service のサービスにメッセージを送信します。

```
{
    "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 マネージドポリシー: AWSIoTThingsRegistration

AWSIoTThingsRegistration ポリシーは IAM ID にアタッチできます。

このポリシーは、StartThingRegistrationTask API を使用してモノを一括して登録するアクセスを許可する、関連付けられた ID のアクセス許可を付与します。このポリシーは、データの処理とストレージに影響を与える可能性があります。でこのポリシーを表示するには AWS マネジメントコンソール、「」を参照してくださいAWSIoTThingsRegistration。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- iot - 一括登録時に、モノを作成し、ポリシーと証明書をアタッチするためのアクションを実行します。

```
{
    "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 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始 AWS IoT してからの の AWS 管理ポリシーの更新に関する詳細を表示します。このページの変更に関する自動アラートについては、 AWS IoT ドキュメント履歴ページの RSS フィードにサブスクライブしてください。

変更	説明	日付
AWSIoTFullAccess — 既存のポリシーへの更新	AWS IoT は、ユーザーが HTTP プロトコルを使用して AWS IoT Jobs データプレーン API オペレーションにアクセスできるようにする新しいアクセス許可を追加しました。 新しい IAM ポリシープレフィックス を使用するとiotjobsdata: 、 AWS IoT Jobs データプレーンエンドポイントにアクセスするためのきめ細かなアクセスコントロールが提供されます。コントロールプレーン API オペレーションには、これまでどおり、iot: プレフィックスを使用します。詳細については、「AWS IoT Core HTTPS プロトコルの ポリシー」を参照してください。	2022 年 5 月 11 日
AWS IoT が変更の追跡を開始しました	AWS IoT は、 AWS 管理ポリシーの変更の追跡を開始しました。	2022 年 5 月 11 日

AWS IoT ID とアクセスのトラブルシューティング

以下の情報は、 および IAM の使用時に発生する可能性がある一般的な問題の診断 AWS IoT と修正に役立ちます。

トピック

- でアクションを実行する権限がありません AWS IoT
- iam:PassRole を実行する権限がありません
- 自分の 以外のユーザーに自分の AWS IoT リソース AWS アカウント へのアクセスを許可したい

でアクションを実行する権限がありません AWS IoT

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

以下の例のエラーは、IAM ユーザーの mateojackson がコンソールを使用して、モノのリソースの詳細を表示しようとしているが、iot:*DescribeThing* アクセス許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iot:DescribeThing on resource: MyIoTThing
```

この場合、iot:*DescribeThing* アクションを使用してモノのリソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

AWS IoT Device Advisor の使用

AWS IoT Device Advisor を使用している場合、次の例のエラーは、ユーザーがコンソールを使用してスイート定義の詳細を表示mateojacksonしようとしているが、 アクセスiotdeviceadvisor:*GetSuiteDefinition*許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotdeviceadvisor:GetSuiteDefinition on resource: MySuiteDefinition
```

この場合、iotdeviceadvisor:*GetSuiteDefinition* アクションを使用して *MySuiteDefinition* リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWS IoTにロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して AWS IoTでアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分の AWS IoT リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- がこれらの機能 AWS IoT をサポートしているかどうかを確認するには、「」を参照してくださいが IAM と AWS IoT 連携する方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。

- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティーが所有する へのアクセスを提供する AWS アカウント」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、IAM ユーザーガイド の IAM でのクロスアカウントのリソースへのアクセス を参照してください。

ログ記録とモニタリング

モニタリングは、 および AWS IoT AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ログ記録および監視手順の詳細については、「モニタリング AWS IoT」を参照してください。

モニタリングツール

AWS には、モニタリングに使用できるツールが用意されています AWS IoT。自動的にモニタリングが行われるように、これらのツールを設定できます。手動操作を必要とするツールもあります。モニタリングタスクをできるだけ自動化することをお勧めします。

自動モニタリングツール

次の自動モニタリングツールを使用して、問題が発生したときに監視 AWS IoT および報告できます。

- Amazon CloudWatch アラーム - 指定した期間にわたって単一のメトリクスをモニタリングし、複数の期間にわたる特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) のトピックまたは Amazon EC2 Auto Scaling のポリシーに送信される通知です。CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。詳細については、「Amazon CloudWatch を使用して AWS IoT アラームとメトリクスをモニタリングする」を参照してください。
- Amazon CloudWatch Logs – AWS CloudTrail またはその他のソースからログファイルをモニタリング、保存、およびアクセスします。Amazon CloudWatch Logs では、Device Advisor のテストケースが実行する重要なステップ AWS IoT 、生成されたイベント、デバイスから送信された

MQTT メッセージ、またはテストの実行 AWS IoT Core 中に送信された MQTT メッセージを確認することもできます。これらのログにより、デバイスでデバッグして是正措置を講じることができます。詳細については、CloudWatch Logs AWS IoT を使用したモニタリング を参照してください。Amazon CloudWatch の使用の詳細については、Amazon CloudWatch ユーザーガイドのログファイルのモニタリングを参照してください。

- Amazon CloudWatch Events – イベントに一致したものを 1 つ以上のターゲットの関数またはストリームに渡して、変更、状態の情報の収集、是正措置を行います。詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch Events とは」を参照してください。
- AWS CloudTrail ログモニタリング – アカウント間でログファイルを共有し、CloudWatch Logs に送信CloudWatch CloudTrail ログファイルをリアルタイムでモニタリングし、Java でログ処理アプリケーションを書き込み、CloudTrail による配信後にログファイルが変更されていないことを確認します。詳細については、「を使用した AWS IoT API コールのログ記録 AWS CloudTrail」と、AWS CloudTrail ユーザーガイドの「CloudTrail ログファイルの操作」を参照してください。

手動モニタリングツール

モニタリングのもう 1 つの重要な部分は AWS IoT 、CloudWatch アラームでカバーされていない項目を手動でモニタリングすることです。 AWS IoT、CloudWatch、およびその他の AWS サービスコンソールダッシュボードには、 AWS 環境の状態がat-a-glanceビューが表示されます。ログファイルも確認することをお勧めします AWS IoT。

- AWS IoT ダッシュボードには以下が表示されます。
  - CA 証明書
  - 証明書
  - ポリシー
  - ルール
  - モノ
- CloudWatch ホームページには、次の内容が表示されます。
  - 現在のアラームとステータス。
  - アラームとリソースのグラフ。
  - サービス状態ステータス。

  CloudWatch を使用して以下のことを実行できます。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成する。
  - メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する。

- すべての AWS リソースメトリクスを検索して参照します。
- 問題があることを通知するアラームを作成および編集する。

AWS IoT Core のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス「コンプライアンスプログラムによるスコープ」の「コンプライアンス」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

AWS IoT Core の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、およびスケーラビリティが優れています。

とアベイラビリティーゾーンの詳細については、 AWS リージョンAWS 「 グローバルインフラストラクチャ」を参照してください。

AWS IoT Core は、デバイスに関する情報をデバイスレジストリに保存します。また、CA 証明書、デバイス証明書、およびデバイスシャドウデータも保存されます。このデータは、ハードウェアまたはネットワークに障害が発生しても、このデータはアベイラビリティーゾーン間で自動的に複製されますが、リージョン間では複製されません。

AWS IoT Core デバイスレジストリが更新されると、 は MQTT イベントを発行します。これらのメッセージを使用して、レジストリデータをバックアップし、DynamoDB テーブルなどに保存でき

ます。が作成する証明書、または自分で作成する証明書を保存するのは AWS IoT Core お客様の責任です。Device Shadow はデバイスに関する状態データを保存し、デバイスがオンラインに戻ったときに再送信できます。 AWS IoT Device Advisor は、テストスイートの設定に関する情報を保存します。このデータは、ハードウェアまたはネットワークに障害が発生しても、自動的にはレプリケートされます。

AWS IoT Core リソースはリージョン固有であり、特にそう AWS リージョン しない限り、 間でレプリケートされません。

セキュリティのベストプラクティスの詳細については、「のセキュリティのベストプラクティス AWS IoT Core」を参照してください。

インターフェイス VPC エンドポイント AWS IoT Core での の使用

を使用すると AWS IoT Core、インターフェイス VPC エンドポイントを使用して、仮想プライベートクラウド (VPC) 内に IoT コントロールプレーンエンドポイントと IoT データエンドポイントを作成できます。 https://docs.aws.amazon.com//vpc/latest/userguide/vpce-interface.html#create-interface-endpointインターフェイス VPC エンドポイントは AWS PrivateLink、プライベート IP アドレス AWS を使用して で実行されているサービスにアクセスするために使用できるテクノロジーである AWS を利用しています。詳細については、「Amazon Virtual Private Cloud」を参照してください。

リモートネットワーク上の現場にあるデバイス (企業ネットワークなど) を Amazon VPC の詳細については、「ネットワークから Amazon VPC への接続マトリックス」に掲載されている様々なリストを参照してください。

内容

- AWS IoT Core コントロールプレーン用の VPC エンドポイントの作成
- AWS IoT Core データプレーンの VPC エンドポイントの作成
- AWS IoT Core 認証情報プロバイダーの VPC エンドポイントの作成
- Amazon VPC インターフェイスエンドポイントの作成
- プライベートホストゾーンを設定
- VPC エンドポイント AWS IoT Core を介した へのアクセスの制御
- 制限事項
- を使用した VPC エンドポイントのスケーリング AWS IoT Core
- VPC エンドポイントでのカスタムドメインの使用

- の VPC エンドポイントの可用性 AWS IoT Core
- インターフェイス VPC エンドポイントでの AWS IoT Device Management セキュアトンネリングの使用

AWS IoT Core コントロールプレーン用の VPC エンドポイントの作成

AWS IoT Core コントロールプレーン API の VPC エンドポイントを作成して、デバイスを AWS IoT サービスやその他の AWS サービスに接続できます。VPC エンドポイントの使用を開始するには、インターフェイス VPC エンドポイントを作成し、 を AWS サービス AWS IoT Core として選択します。CLI を使用している場合は、まず describe-vpc-endpoint-services を呼び出して、 AWS IoT Core が特定の に存在するアベイラビリティーゾーンを選択していることを確認します AWS リージョン。例えば、us-east-1 では、このコマンドは以下のようになります。

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.api
```

以下の詳細な手順を参照して、 AWS IoT Core コントロールプレーンの Amazon VPC インターフェイスエンドポイントを作成します。

AWS IoT Core データプレーンの VPC エンドポイントの作成

AWS IoT Core データプレーン API 用の VPC エンドポイントを作成して、デバイスを AWS IoT サービスやその他の AWS サービスに接続できます。VPC エンドポイントの使用を開始するには、インターフェイス VPC エンドポイントを作成し、 を AWS サービス AWS IoT Core として選択します。CLI を使用している場合は、まず describe-vpc-endpoint-services を呼び出して、 AWS IoT Core が特定の に存在するアベイラビリティーゾーンを選択していることを確認します AWS リージョン。例えば、us-east-1 では、このコマンドは以下のようになります。

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.data
```

> **Note**
>
> DNS レコードを自動的に作成するための VPC 機能は無効になっています。これらのエンドポイントを接続するには、プライベート DNS レコードを手動で作成する必要があります。プライベート VPC DNS レコードの詳細については、インターフェイスエンドポイントのプライベート DNS を参照してください。 AWS IoT Core VPC の制限の詳細については、制限事項「」を参照してください。

MQTT クライアントを VPC エンドポイントインターフェイスに接続するには

- VPC にアタッチされているプライベートホストゾーンに DNS レコードを手動で作成する必要があります。開始するには、「プライベートホストゾーンの作成」を参照してください。

- プライベートホストゾーン内で、VPC エンドポイントの各 Elastic Network Interface IP のエイリアスレコードを作成します。複数の VPC エンドポイントに複数のネットワークインターフェイス IP がある場合、すべての加重レコードを通して等しい加重の DNS レコードを作成する必要があります。これらの IP アドレスは、説明フィールドの VPC エンドポイント ID でフィルタリングした場合に、DescribeNetworkInterfaces API コールから取得できます。

以下の詳細な手順を参照して、Amazon VPC インターフェイスエンドポイントを作成し、 AWS IoT Core データプレーンのプライベートホストゾーンを設定します。

AWS IoT Core 認証情報プロバイダーの VPC エンドポイントの作成

VPC エンドポイントの AWS IoT Core 認証情報プロバイダーを作成して、クライアント証明書ベースの認証を使用してデバイスに接続し、AWS 署名バージョン 4 形式で一時的な AWS 認証情報を取得できます。 AWS IoT Core 認証情報プロバイダーの VPC エンドポイントの使用を開始するには、create-vpc-endpoint CLI コマンドを実行してインターフェイス VPC エンドポイントを作成し、AWS サービスとして AWS IoT Core 認証情報プロバイダーを選択します。特定の AWS IoT Core に存在するアベイラビリティーゾーンを選択するには AWS リージョン、まず describe-vpc-endpoint-services コマンドを実行します。例えば、us-east-1 では、このコマンドは以下のようになります。

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.iot.credentials
```

> **Note**
>
> DNS レコードを自動的に作成するための VPC 機能は無効になっています。これらのエンドポイントを接続するには、プライベート DNS レコードを手動で作成する必要があります。プライベート VPC DNS レコードの詳細については、インターフェイスエンドポイントのプライベート DNS を参照してください。 AWS IoT Core VPC の制限の詳細については、制限事項「」を参照してください。

HTTP クライアントを VPC エンドポイントインターフェイスに接続するには

- VPC にアタッチされているプライベートホストゾーンに DNS レコードを手動で作成する必要があります。開始するには、「プライベートホストゾーンの作成」を参照してください。

- プライベートホストゾーン内で、VPC エンドポイントの各 Elastic Network Interface IP のエイリアスレコードを作成します。複数の VPC エンドポイントに複数のネットワークインターフェイス IP がある場合、すべての加重レコードを通して等しい加重の DNS レコードを作成する必要があります。これらの IP アドレスは、説明フィールドの VPC エンドポイント ID でフィルタリングした場合に、DescribeNetworkInterfaces API コールから取得できます。

以下の詳細な手順を参照して、Amazon VPC インターフェイスエンドポイントを作成し、 AWS IoT Core 認証情報プロバイダーのプライベートホストゾーンを設定します。

Amazon VPC インターフェイスエンドポイントの作成

インターフェイス VPC エンドポイントを作成して、 AWS のサービスに接続できます AWS PrivateLink。次の手順を使用して、 AWS IoT Core データプレーンまたは AWS IoT Core 認証情報プロバイダーに接続するインターフェイス VPC エンドポイントを作成します。詳細については、「インターフェイス VPC エンドポイントを使用して AWS サービスにアクセスする」を参照してください。

> **Note**
>
> AWS IoT Core データプレーンと AWS IoT Core 認証情報プロバイダーの Amazon VPC インターフェイスエンドポイントを作成するプロセスは似ていますが、接続を機能させるにはエンドポイント固有の変更を行う必要があります。

VPC エンドポイントコンソールを使用して VPC エンドポイントを使用してインターフェイスを作成するには

1. VPC エンドポイントコンソールに移動し、左側のメニューの [仮想プライベートクラウド] で [エンドポイント]、[エンドポイントを作成] を選択します。
2. [エンドポイントの作成] ページで、以下の情報を指定します。
   - [Service category] (サービスカテゴリ) には [AWS のサービス s] を選択します。
   - [Service Name] (サービス名) については、キーワード `iot` を入力して検索します。表示された `iot` サービスのリストで、エンドポイントを選択します。

AWS IoT Core コントロールプレーンの VPC エンドポイントを作成する場合は、 の AWS IoT Core コントロールプレーン API エンドポイントを選択します AWS リージョン。エンドポイントは com.amazonaws.*region*.iot.api の形式です。

AWS IoT Core データプレーンの VPC エンドポイントを作成する場合は、リージョン AWS IoT Core のデータプレーン API エンドポイントを選択します。エンドポイントは com.amazonaws.*region*.iot.data の形式です。

AWS IoT Core 認証情報プロバイダーの VPC エンドポイントを作成する場合は、リージョンの AWS IoT Core 認証情報プロバイダーエンドポイントを選択します。エンドポイントは com.amazonaws.*region*.iot.credentials の形式です。

連邦情報処理規格 (FIPS) リージョンの VPC エンドポイントを作成する場合は、の FIPS API エンドポイントを選択します AWS リージョン。エンドポイントは com.amazonaws.*region*.iot-fips.api の形式です。これはコントロールプレーン専用です。

> **Note**
>
> 中国リージョン AWS IoT Core のデータプレーンのサービス名は の形式になりますcn.com.amazonaws.*region*.iot.data。中国リージョンの AWS IoT Core コントロールプレーンのサービス名は、 の形式になりますcom.amazonaws.*region*.iot.api。

- [VPC] と [Subnets] (サブネット) には、エンドポイントを作成する VPC と、エンドポイントネットワークを作成するアベイラビリティーゾーン (AZ) を選択します。
- DNS 名を有効にする では、このエンドポイントの有効化が AWS IoT Core データプレーンと AWS IoT Core 認証情報プロバイダーに対して選択されていないことを確認してください。AWS IoT Core データプレーンも AWS IoT Core 認証情報プロバイダーもプライベート DNS 名をまだサポートしていません。

  AWS IoT Core コントロールプレーンの場合、デフォルトで Enable DNS name が選択されます。これにより、コントロールプレーンの AWS IoT Core パブリックエンドポイントへのリクエストは、代わりに VPC エンドポイントを介してルーティングされます。これを有効にすると、プライベートホストゾーンを設定する必要はありません。
- [Security group] (セキュリティグループ) には、エンドポイントネットワークインターフェイスに関連付けるセキュリティグループを選択します。

- オプションで、タグを追加または削除できます。タグとは名前と値のペアで、エンドポイントに関連付けるために使用します。

3. [Create Endpoint] (エンドポイントの作成) をクリックして、VPC エンドポイントを作成します。

AWS PrivateLink エンドポイントを作成すると、エンドポイントの詳細タブに DNS 名のリストが表示されます。このセクションで作成したこれらの DNS 名のいずれかを使用して、プライベートホストゾーンを設定できます。 AWS IoT Core コントロールプレーンを使用している場合は、プライベートホストゾーンを設定する必要はありません。

プライベートホストゾーンを設定

> **Note**
>
> AWS IoT Core コントロールプレーンを使用していて、DNS 名の有効化が選択されている場合は、プライベートホストゾーンを設定する必要はありません。無効にする場合は、以下の手順に従ってプライベートホストゾーンを設定する必要があります。

前のセクションで作成したこれらの DNS 名のいずれかを使用して、プライベートホストゾーンを設定できます。

AWS IoT Core データプレーンの場合

DNS 名は、ドメイン設定名または `IoT:Data-ATS` エンドポイントである必要があります。例えば、DNS 名は、 *xxx*`-ats.data.iot.`*region*`.amazonaws.com` などがあります。

AWS IoT Core 認証情報プロバイダーの場合

DNS 名は `iot:CredentialProvider` エンドポイントである必要があります。例えば、DNS 名は、 *xxxx*`.credentials.iot.`*region*`.amazonaws.com` などがあります。

AWS IoT Core コントロールプレーンの場合

DNS 名は AWS IoT Core コントロールプレーンエンドポイントである必要があります。 AWS IoT Core コントロールプレーンの DNS 名の例は です *xxxx*`.api.iot.`*region*`.amazonaws.com`。

> **Note**
>
> AWS IoT Core データプレーンと AWS IoT Core 認証情報プロバイダーのプライベートホストゾーンを設定するプロセスは似ていますが、接続を機能させるにはエンドポイント固有の変更を行う必要があります。

プライベートホストゾーンを作成します。

Route 53 コンソールを使用してプライベートホストゾーンを作成するには

1. Route 53 の [Hosted zones] (ホストゾーン) コンソールに移動して、[Create hosted zone] (ホストゾーンの作成) をクリックします。
2. [Create hosted zone] (ホストゾーンの作成) ページで、以下の情報を指定します。
   - [ドメイン名] には、`iot:Data-ATS` または `iot:CredentialProvider` エンドポイントのエンドポイントアドレスを入力します。次の AWS CLI コマンドは、パブリックネットワークの `aws iot describe-endpoint --endpoint-type iot:Data-ATS` または `aws iot describe-endpoint --endpoint-type iot:CredentialProvider` を通じてエンドポイントを取得する方法を示しています。

     > **Note**
     >
     > カスタムドメインを使用している場合は、「VPC コマンドでのカスタムドメインの使用」を参照してください。カスタムドメインは、 AWS IoT Core 認証情報プロバイダーではサポートされていません。

   - [タイプ] には、[プライベートホストゾーン] を選択します。
   - 必要に応じて、タグを追加または削除してホストゾーンに関連付けることができます。
3. プライベートホストゾーンを作成するには、ホストゾーンの作成を選んでください。

詳細については、「プライベートホストゾーンの作成」を参照してください。

レコードを作成する

プライベートホストゾーンを作成したら、そのドメインにトラフィックをルーティングする方法を DNS に指示するレコードを作成できます。

レコードを作成するには

1. 表示されるホストゾーンのリストで、前に作成したプライベートホストゾーンを選び、レコードを作成するを選びます。
2. ウィザードを使用してレコードを作成します。コンソールに [Quick create] (クイック作成) 方式が表示された場合は、[Switch to wizard] (ウィザードに切り替える) をクリックします。
3. [Routing policy] (ルーティングポリシー) に [Simple Routing] (シンプルルーティング) を選択し、[Next] (次へ) を選びます。
4. [Configure records] (レコードを設定)ページで、[Define simple record] (シンプルなレコードを定義) を選択します。
5. [Define simple record] (シンプルなレコードを定義) ページで
   - [レコード名] に `iot:Data-ATS` エンドポイントまたは `iot:CredentialProvider` エンドポイントを入力します。これは、プライベートホストゾーン名と同じである必要があります。
   - [レコードタイプ] では、IPv4 サポートのみが必要な場合は、値を `A - Routes traffic to an IPv4 address and some AWS resources` にします。IPv6 サポートのみが必要な場合は、値を `AAAA - Routes traffic to an IPv6 address and some AWS resources` にします。デュアルスタックのサポート (IPv4 と IPv6 の両方) が必要な場合は、同じレコード名と値/トラフィックのルーティング先を使用して 2 つのレコード (`A` と `AAAA`) をホストゾーンに作成します。
   - [Value/Route traffic to (値/トラフィックのルーティング先)] には、[Alias to VPC endpoint (VPC エンドポイントへのエイリアス)] を選択します。次に、お使いの [Region] (地域) を選択し、表示されたエンドポイントのリストから、「???」の説明に従って先ほど作成したエンドポイントを選択します。
6. [Define simple record] (シンプルなレコードを定義) をクリックしてレコードを作成します。

VPC エンドポイント AWS IoT Core を介した へのアクセスの制御

VPC 条件コンテキストキーを使用して、VPC エンドポイントを介してのみデバイスアクセスを許可する AWS IoT Core ように制限できます。 は、次の VPC 関連のコンテキストキー AWS IoT Core をサポートしています。

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core は VPC エンドポイントのエンドポイントポリシーをサポートしていません。

例えば、次のポリシーは、モノの名前に一致するクライアント ID AWS IoT Core を使用して に接続し、モノの名前のプレフィックスが付いた任意のトピックに発行するアクセス許可を付与します。これは、特定の VPC エンドポイント ID を持つ VPC エンドポイントに接続するデバイスを条件とします。このポリシーでは、パブリック IoT データエンドポイントへの接続試行が拒否されます。

```
{
    "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}/*"
            ]
        }
    ]
}
```

制限事項

VPC エンドポイントは現在、AWS IoT Core コントロールプレーンエンドポイント、AWS IoT Core データエンドポイント、AWS IoT Core 認証情報プロバイダーエンドポイントでサポートされています。VPC エンドポイントは、コントロールプレーンを使用する場合、連邦情報処理規格 (FIPS) エンドポイントでのみサポートされます AWS IoT Core 。

IoT コントロールプレーン VPC エンドポイントの制限事項

このセクションでは、IoT コントロールプレーン VPC エンドポイントの制限について説明します。

- VPC エンドポイントは ATS 証明書のみを提供します。
- コントロールプレーンエンドポイントでは、カスタムドメインはサポートされていません。
- FIPS セキュリティポリシーの詳細については、「FIPS セキュリティポリシー」を参照してください。

IoT データ VPC エンドポイントの制限

このセクションでは、IoT データ VPC エンドポイントの制限について説明します。

- MQTT キープアライブ期間は 230 秒に制限されています。これ以上のキープアライブ期間は、自動的に 230 秒に短縮されます。
- 各 VPC エンドポイントは、合計で 100,000 台の同時接続デバイスをサポートします。さらに接続が必要な場合は、を使用した VPC エンドポイントのスケーリング AWS IoT Core「」を参照してください。
- VPC エンドポイントは ATS 証明書のみに対応します (カスタムドメインを除く)。
- VPC エンドポイントポリシーはサポートされません。
- AWS IoT Core データプレーン用に作成された VPC エンドポイントの場合、 AWS IoT Core はゾーンまたはリージョンのパブリック DNS レコードの使用をサポートしていません。

認証情報プロバイダーエンドポイントの制限

このセクションでは、認証情報プロバイダー VPC エンドポイントの制限について説明します。

- VPC エンドポイントは ATS 証明書のみを提供します。
- VPC エンドポイントポリシーはサポートされません。
- 認証情報プロバイダーエンドポイントでは、カスタムドメインはサポートされていません。

- AWS IoT Core 認証情報プロバイダー用に作成された VPC エンドポイントの場合、 AWS IoT Core はゾーンまたはリージョンのパブリック DNS レコードの使用をサポートしていません。

を使用した VPC エンドポイントのスケーリング AWS IoT Core

AWS IoT Core インターフェイス VPC エンドポイントは、1 つのインターフェイスエンドポイントで 100,000 台の接続デバイスに制限されています。ユースケースでブローカーに対する同時接続がこれ以上必要になる場合は、複数の VPC エンドポイントを使用して、インターフェイスエンドポイント間でのデバイスのルーティングを手動で行うことをお勧めします。VPC エンドポイントにトラフィックをルーティングするプライベート DNS レコードを作成するときは、VPC エンドポイントと同じ数の加重レコードを作成して、複数のエンドポイント全体にトラフィックを分散するようにしてください。

VPC エンドポイントでのカスタムドメインの使用

VPC エンドポイントでカスタムドメインを使用する場合は、プライベートホストゾーンにカスタムドメイン名レコードを作成し、Route53 でルーティングレコードを作成する必要があります。詳細については、「プライベートホストゾーンの作成」を参照してください。

> **Note**
>
> カスタムドメインは、 AWS IoT Core データエンドポイントでのみサポートされています。

の VPC エンドポイントの可用性 AWS IoT Core

AWS IoT Core インターフェイス VPC エンドポイントは、AWS IoT Core サポートされているすべてのリージョンで使用できます。認証情報プロバイダーの AWS IoT Core AWS IoT Core インターフェイス VPC エンドポイントは、中国リージョンおよび ではサポートされていません AWS GovCloud (US) Regions。

インターフェイス VPC エンドポイントでの AWS IoT Device Management セキュアトンネリングの使用

AWS IoT Device Management セキュアトンネリングは、インターフェイス VPC エンドポイントをサポートします。VPC エンドポイントを使用すると、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を必要とせずに、VPC と AWS ネットワーク AWS IoT Secure Tunneling 間のトラフィックを維持できます。

インターフェイス VPC エンドポイントはAWS PrivateLink、プライベート IP アドレスを使用してサービスにプライベートにアクセスできるテクノロジーである を利用しています。詳細については、「 AWS PrivateLink ガイド」の「インターフェイス VPC エンドポイントを使用して AWS サービスにアクセスする」を参照してください。

内容

- 前提条件
- VPC エンドポイントを介したトンネル通知の受信
- セキュアトンネリング用の VPC エンドポイントの作成
- Proxy Server での VPC エンドポイントポリシーの設定
- 次の手順

前提条件

の VPC エンドポイントを作成する前に AWS IoT Secure Tunneling、以下があることを確認します。

- VPC エンドポイントを作成するために必要なアクセス許可を持つ AWS アカウント。
- AWS アカウントの VPC。
- AWS IoT Device Management セキュアトンネリングの概念の理解。
- VPC エンドポイントポリシーと AWS Identity and Access Management (IAM) に精通していること

VPC エンドポイントを介したトンネル通知の受信

VPC エンドポイントを介してトンネル通知を受信するために、デバイスは VPC エンドポイントを介して AWS IoT Core データプレーンに接続し、セキュアトンネリング予約 MQTT トピックにサブスクライブできます。

AWS IoT Core データプレーンで VPC エンドポイントを作成して設定する手順については、「 AWS IoT デベロッパーガイド」の「インターフェイス VPC エンドポイント AWS IoT Core で を使用する」を参照してください。

セキュアトンネリング用の VPC エンドポイントの作成

セキュアトンネリングコントロールプレーンとプロキシサーバーの両方の VPC エンドポイントを作成できます。

セキュアトンネリング用の VPC エンドポイントを作成するには

1. Amazon VPC デベロッパーガイドの「インターフェイスエンドポイントの作成」のステップに従います。
2. サービス名で、エンドポイントタイプに基づいて次のいずれかのオプションを選択します。

   コントロールプレーン

   - 標準: `com.amazonaws.<region>.iot.tunneling.api`
   - FIPS (FIPS リージョンで利用可能): `com.amazonaws.<region>.iot-fips.tunneling.api`

   プロキシサーバー

   - 標準: `com.amazonaws.<region>.iot.tunneling.data`
   - FIPS (FIPS リージョンで利用可能): `com.amazonaws.<region>.iot-fips.tunneling.data`

   <region> を に置き換えます AWS リージョン。例えば、`us-east-1`。
3. ネットワーク要件に従って、VPC エンドポイント作成プロセスの残りのステップを完了します。

Proxy Server での VPC エンドポイントポリシーの設定

トンネルへの接続を許可するために使用されるクライアントアクセストークンベースの認可に加えて、VPC エンドポイントポリシーを使用して、デバイスが VPC エンドポイントを使用して Secure Tunneling Proxy Server に接続する方法をさらに制限できます。VPC エンドポイントポリシーは IAM のような構文に従い、VPC エンドポイント自体で設定されます。

プロキシサーバー VPC エンドポイントポリシーでサポートされている IAM アクションは のみですiot:ConnectToTunnel。

以下は、さまざまな VPC エンドポイントポリシーの例です。

プロキシサーバー VPC エンドポイントポリシーの例

次の例は、一般的なユースケースの Proxy Server VPC エンドポイントポリシー設定を示しています。

Example- デフォルトポリシー

このポリシーにより、VPC 内のデバイスは、任意のアカウントで AWS リージョン エンドポイントが作成されたのと同じ 内の任意の AWS トンネルに接続できます。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Example- 特定の AWS アカウントへのアクセスを制限する

このポリシーにより、VPC エンドポイントは特定の AWS アカウントのトンネルにのみ接続できます。

```
{
    "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- トンネルエンドポイントによる接続の制限

VPC エンドポイントへのアクセスを制限して、デバイスがトンネルの送信元または送信先の端に接続することのみを許可できます。

ソースのみ:

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "source"
                }
            }
        }
    ]
}
```

送信先のみ:

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
            }
        }
    ]
}
```

Example- リソースタグに基づいてアクセスを制限する

このポリシーにより、VPC エンドポイントは、特定のキーと値のペアでタグ付けされたトンネルにのみ接続できます。

```
{
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Environment": "Production"
                }
            }
        }
    ]
}
```

Example- ポリシー条件の組み合わせ

このポリシーは、複数のポリシー要素を組み合わせる方法を示しています。これにより、特定の AWS アカウントの任意のトンネルへの接続が許可されます。ただし、トンネルが`AllowConnectionsThroughPrivateLink`に設定され`true`、クライアントがトンネルの送信先端に接続していないというタグが付けられている場合に限られます。

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

次の手順

の VPC エンドポイントを作成して設定したら AWS IoT Secure Tunneling、次の点を考慮してください。

- エンドポイントを介してデバイスを接続して、VPC エンドポイント設定をテストします。
- Amazon CloudWatch メトリクスを使用して VPC エンドポイントの使用状況をモニタリングします。
- セキュリティ要件に応じて、VPC エンドポイントポリシーを確認して更新します。

AWS IoT Device Management セキュアトンネリングの詳細については、「」を参照してくださいAWS IoT Secure Tunneling。

のインフラストラクチャセキュリティ AWS IoT

マネージドサービスのコレクションとして、 AWS IoT は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティ手順で保護されています。

AWS が公開した API コールを使用して、ネットワーク AWS IoT 経由で にアクセスします。クライアントは、Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。詳細については、「のトランスポートセキュリティ AWS IoT Core」を参照してください。

リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットのアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

AWS IoT Core を使用した本番稼働用フリートまたはデバイスのセキュリティモニタリング

IoT フリートは、多様な機能を持ち、存続期間が長く、地理的に分散される多数のデバイスで構成されることがあります。このような特性によってフリートのセットアップが複雑になり、エラーを起こしやすくなります。また、デバイスの計算能力、メモリ、ストレージの機能には制約があるため、デバイス自体での暗号化や他の形式のセキュリティの使用が制限されます。さらに、デバイスは多く場合、既知の脆弱性を持つソフトウェアを使用しています。このような要素が原因で、IoT フリートはハッカーの魅力的なターゲットとなり、デバイスフリートを継続的に保護することが困難になっています。

AWS IoT Device Defender セキュリティの問題やベストプラクティスからの逸脱を特定するためのツールを提供することで、これらの課題に対処します。 AWS IoT Device Defender を使用すると、接続されたデバイスを分析、監査、モニタリングして異常な動作を検出し、セキュリティリスクを軽減できます。 AWS IoT Device Defender は、デバイスフリートを監査して、セキュリティのベストプラクティスに準拠していることを確認して、デバイスの異常な動作を検出できます。これにより、AWS IoT デバイスフリート全体に一貫したセキュリティポリシーを適用し、デバイスが侵害されたときに迅速に対応できます。詳細については、「AWS IoT Device Defender」を参照してください。

AWS IoT Device Advisor は、必要に応じてフリートの更新とパッチをプッシュします。 AWS IoT Device Advisor はテストケースを自動的に更新します。選択するテストケースには常に最新バージョンが使用されています。詳細については、「Device Advisor」を参照してください。

のセキュリティのベストプラクティス AWS IoT Core

このセクションでは、 のセキュリティのベストプラクティスについて説明します AWS IoT Core。産業における IoT ソリューションのセキュリティルールについては、「産業における IoT ソリューションにおける 10 のセキュリティゴールデンルール」を参照してください。

での MQTT 接続の保護 AWS IoT

AWS IoT Core は、接続されたデバイスがクラウドアプリケーションやその他のデバイスと簡単かつ安全にやり取りできるようにするマネージドクラウドサービスです。 は、断続的な接続を許容するように特別に設計された軽量通信プロトコルである HTTP、WebSocket、および MQTT AWS IoT Core をサポートしています。MQTT AWS IoT を使用して に接続する場合は、各接続をクライアント ID と呼ばれる識別子に関連付ける必要があります。MQTT クライアント ID は、MQTT 接続を一意に識別します。別の接続に対して既に要求されているクライアント ID を使用して新しい接続が確

立された場合、 AWS IoT メッセージブローカーは古い接続を削除して新しい接続を許可します。クライアント IDsは、それぞれ内で一意である必要があります AWS アカウント AWS リージョン。つまり、 の外部 AWS アカウント または 内のリージョン間でクライアント IDs のグローバル一意性を適用する必要はありません AWS アカウント。

デバイスフリートでの MQTT 接続の中断の影響と重大度は、多くの要因によって異なります。具体的には次のとおりです。

- ユースケース (デバイスが送信するデータ、データ AWS IoTの量、データの送信頻度など) 。
- MQTT クライアント設定 (例えば、自動再接続設定、関連するバックオフタイミング、MQTT 永続セッションの使用など)。
- デバイスリソースの制約。
- 切断の根本原因、その積極性、永続性。

クライアント ID の競合とその潜在的な悪影響を回避するには、各デバイスまたはモバイルアプリケーションに、 AWS IoT メッセージブローカーへの MQTT 接続に使用できるクライアント IDs を制限する AWS IoT または IAM ポリシーがあることを確認してください。例えば、IAM ポリシーを使用すると、既に使用中のクライアント ID を使用することによりデバイスが意図せず別のデバイスの接続を閉じないようにすることができます。詳細については、「Authorization」を参照してください。

フリート内のすべてのデバイスには、意図したアクションのみを認可する権限を持つ認証情報が必要です。これには、メッセージの発行や特定のスコープとコンテキストを持つトピックへのサブスクライブなどの AWS IoT MQTT アクションが含まれますが、これらに限定されません。特定のアクセス許可ポリシーは、ユースケースによって異なる場合があります。ビジネス要件とセキュリティ要件に最も合うアクセス許可ポリシーを特定します。

アクセス許可ポリシーの作成と管理を簡素化するために、AWS IoT Core ポリシー変数 および IAM ポリシー変数を使用できます。ポリシー変数はポリシーに配置でき、ポリシーが評価されると、変数はデバイスのリクエストから取得された値に置き換えられます。ポリシー変数を使用して、複数のデバイスにアクセス許可を付与する単一のポリシーを作成できます。 AWS IoT メッセージブローカーへの接続に使用される AWS IoT アカウント設定、認証メカニズム、ネットワークプロトコルに基づいて、ユースケースに関連するポリシー変数を特定できます。ただし、最良のアクセス許可ポリシーを記述するには、使用状況と脅威モデルの詳細を考慮してください。

たとえば、デバイスを AWS IoT レジストリに登録した場合、 AWS IoT ポリシーでモノのポリシー変数を使用して、モノの名前、モノのタイプ、モノの属性値などのモノのプロパティに基づいてアクセス許可を付与または拒否できます。モノの名前は、モノの接続時に送信される MQTT 接続メッセージのクライアント ID から取得されます AWS IoT。モノのポリシー

変数は、モノが TLS 相互認証を使用して MQTT AWS IoT 経由で に接続するか、認証された Amazon Cognito ID を使用して WebSocket プロトコル経由で MQTT に接続すると置き換えられます。AttachThingPrincipal API を使用して、証明書と認証された Amazon Cognito ID をモノにアタッチできます。iot:Connection.Thing.ThingName は、クライアント ID の制限を適用するために便利なモノのポリシー変数です。次の AWS IoT ポリシー例では、 AWS IoT メッセージブローカーへの MQTT 接続のクライアント ID として登録済みモノの名前を使用する必要があります。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

進行中のクライアント ID の競合を特定する場合は、CloudWatch Logs AWS IoTを有効にして使用できます。クライアント ID の競合が原因で AWS IoT メッセージブローカーが切断する MQTT 接続ごとに、次のようなログレコードが生成されます。

```
{
    "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"
}
```

`{$.reason= "DUPLICATE_CLIENT_ID" }` などの CloudWatch Logs フィルターを使用して、クライアント ID が競合するインスタンスを検索したり、CloudWatch メトリクスフィルターおよび対応する CloudWatch アラームを設定して、継続的なモニタリングとレポートを作成したりできます。

AWS IoT Device Defender を使用して、過度に寛容なポリシー AWS IoT と IAM ポリシーを特定できます。 AWS IoT Device Defender には、フリート内の複数のデバイスが同じクライアント ID を使用して AWS IoT メッセージブローカーに接続しているかどうかを通知する監査チェックも用意されています。

AWS IoT Device Advisor を使用して、デバイスが確実に に接続 AWS IoT Core し、セキュリティのベストプラクティスに従うことができることを検証できます。

関連情報

- AWS IoT Core
- AWS IoTのセキュリティ機能
- AWS IoT Core ポリシー変数
- IAM ポリシー変数
- Amazon Cognito ID
- AWS IoT Device Defender
- の CloudWatch Logs AWS IoT

デバイスのクロックを同期させる

デバイスの時刻を正確に保つことが重要です。X.509 証明書には有効期限の日時があります。デバイスのクロックは、サーバー証明書が現在も有効であることを確認するために使用されます。商用 IoT デバイスを構築する場合は、製品が販売される前に長期間保管される可能性があることに注意してください。この間、リアルタイムクロックがドリフトし、電池が放電する可能性があるため、工場での設定時間では不十分です。

ほとんどのシステムでは、デバイスのソフトウェアに Network Time Protocol (NTP) クライアントを含める必要があります。デバイスは、 AWS IoT Coreへの接続を試行する前に、NTP サーバーと同期するまで待機する必要があります。これが不可能な場合は、後続の接続が成功するように、ユーザーがデバイスの時刻を設定する方法を提供する必要があります。

デバイスが NTP サーバーと同期されると、 AWS IoT Coreとの接続を開くことができます。許容されるクロックスキューの量は、接続で何をしようとしているかによって異なります。

サーバー証明書の検証

デバイスが最初に操作するのは AWS IoT 、安全な接続を開くことです。デバイスを に接続するときは AWS IoT、別のサーバーを偽装するのではなく AWS IoT 、 と話していることを確認してください AWS IoT。各 AWS IoT サーバーは、`iot.amazonaws.com`ドメイン用に発行された証明書でプロビジョニングされます。この証明書は、ドメインのアイデンティティと所有権を検証した信頼できる認証機関 AWS IoT によって に発行されました。

デバイスが接続するときに最初に AWS IoT Core 行うことの 1 つは、デバイスにサーバー証明書を送信することです。デバイスは、`iot.amazonaws.com` に接続する予定だったことと、その接続の最後のサーバーが、そのドメインの信頼された機関からの証明書を持っていることを確認できます。

TLS 証明書は X.509 形式で、組織の名前、場所、ドメイン名、有効期間などのさまざまな情報が含まれています。有効期間は、`notBefore` と `notAfter` と呼ばれる時間値のペアとして指定されます。などのサービスは、サーバー証明書に限られた有効期間 (1 年など) AWS IoT Core を使用し、古い証明書の有効期限が切れる前に新しい証明書の提供を開始します。

デバイスごとの単一の ID を使用する

クライアントごとに 1 つの ID を使用します。通常、デバイスでは X.509 クライアント証明書が使用されます。ウェブおよびモバイルアプリケーションは Amazon Cognito ID を使用します。これにより、デバイスにきめ細かなアクセス許可を適用できます。

例えば、電球とサーモスタットの 2 つの異なるスマートホームオブジェクトからステータス更新を受け取る携帯電話デバイスで構成されるアプリケーションがあるとします。電球は、バッテリーレベルのステータスを送信し、サーモスタットは温度を報告するメッセージを送信します。

AWS IoT はデバイスを個別に認証し、各接続を個別に処理します。承認ポリシーを使用してきめ細かなアクセス制御を適用できます。サーモスタットのポリシーを定義して、トピックスペースに公開することができます。電球に別のポリシーを定義して、別のトピックスペースに公開することができます。最後に、これらのデバイスからのメッセージを受信するために、サーモスタットと電球のトピックへの接続とサブスクライブのみを許可するモバイルアプリのポリシーを定義できます。

最小権限の原則を適用し、デバイスごとのアクセス許可を可能な限り絞り込みます。すべてのデバイスまたはユーザーには、既知のクライアント ID との接続と、識別および固定されたトピックセットの発行とサブスクライブ AWS IoT のみを許可する AWS IoT ポリシーが必要です。

バックアップ AWS リージョン として 2 番目の を使用する

データのコピーをバックアップ AWS リージョン として 1 秒に保存することを検討してください。Disaster Recovery for AWS IoTという名前の AWS ソリューションは使用できなくなりました。関連付けられた GitHub ライブラリは引き続きアクセス可能ですが、2023 年 7 月に AWS 廃止され、メンテナンスやサポートは提供されなくなりました。独自のソリューションを実装したり、追加のサポートオプションを確認したりするには、「AWSへのお問い合わせ」を参照してください。アカウントに関連付けられている AWS Technical Account Manager がある場合は、サポートを依頼してください。

ジャストインタイムプロビジョニングの使用

各デバイスの手動作成とプロビジョニングには時間がかかる場合があります。 は、最初に接続するときにデバイスをプロビジョニングするテンプレートを定義する方法 AWS IoT を提供します AWS IoT。詳細については、「ジャストインタイムプロビジョニング」を参照してください。

AWS IoT Device Advisor テストを実行するアクセス許可

次のポリシーテンプレートは、 AWS IoT Device Advisor テストケースの実行に必要な最小限のアクセス許可と IAM エンティティを示しています。*your-device-role-arn* を、前提条件の下で作成したデバイスロール Amazon Resource Name (ARN) に置き換える必要があります。

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

デバイスアドバイザーのクロスサービスでの混乱した代理の防止

混乱した代理問題は、アクションを実行する許可を持たないエンティティが、より特権のあるエンティティにアクションを実行するように強制できるセキュリティの問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、 は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを持つすべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシー内のグローバル条件コンテキストキー aws:SourceArn と aws:SourceAccount を使用して、リソースについてデバイスアドバイザーが別のサービスに付与する許可を制限することをお勧めします。両方のグローバル条件コンテキストキーを同じポリシーステートメントで使用する場合は、aws:SourceAccount 値と、aws:SourceArn 値に含まれるアカウントが、同じアカウント ID を示している必要があります。

aws:SourceArn の値は、スイート定義リソースの ARN である必要があります。スイート定義リソースは、デバイスアドバイザーで作成したテストスイートを指します。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合は、aws:SourceArn グローバルコンテキスト条件キーを使用して、ARN の未知部分をワイルドカード (*) で表します。例えば、arn:aws:iotdeviceadvisor:*:*account-id*:suitedefinition/* です

次の例では、デバイスアドバイザーで aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題を回避する方法を示します。

```
{
    "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 トレーニングと認定

AWS IoT セキュリティの主要な概念については、Security AWS IoT Primer を参照してください。

モニタリング AWS IoT

モニタリングは、 および AWS IoT AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。

マルチポイント障害が発生した場合にデバッグしやすくするために、 AWS ソリューションのすべての部分からモニタリングデータを収集することを強くお勧めします。まず、以下の質問に答えて監視計画を作成します。どのように答えるべきかわからない場合でも、引き続き ログ記録を有効化して、パフォーマンスのベースラインを確立できます。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを使用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまなタイミングと負荷条件で AWS IoT パフォーマンスを測定することで、ロギングを有効にし、環境で通常のパフォーマンスのベースラインを確立します。モニタリング中は AWS IoT、現在のパフォーマンスデータと比較できるように、モニタリングデータの履歴を保持します。これにより、通常のパフォーマンスパターンとパフォーマンスの異常を特定し、問題に対処するための方法を考えることができます。

のベースラインパフォーマンスを確立するには AWS IoT、これらのメトリクスをモニタリングして開始する必要があります。監視対象のメトリクスは、必要時にいつでも増やすことができます。

- PublishIn.Success
- PublishOut.Success
- Subscribe.Success
- Ping.Success
- Connect.Success
- GetThingShadow.Accepted
- UpdateThingShadow.Accepted
- DeleteThingShadow.Accepted

- RulesExecuted

このセクションのトピックは、ログ記録と AWS IoTの監視を開始するために役立ちます。

トピック

- AWS IoT ログ記録の設定
- Amazon CloudWatch を使用して AWS IoT アラームとメトリクスをモニタリングする
- CloudWatch Logs AWS IoT を使用したモニタリング
- Amazon CloudWatch にデバイス側のログをアップロード
- を使用した AWS IoT API コールのログ記録 AWS CloudTrail

AWS IoT ログ記録の設定

AWS IoT アクティビティをモニタリングおよびログ記録する前に、 AWS IoT コンソール、CLI、または API を使用してログ記録を有効にする必要があります。のログ記録は、アカウントレベル、イベントレベル、またはリソース固有の AWS IoT 3 つのレベルで設定できます。イベントレベルとリソース固有のログ記録は、V2 ログ記録でのみ使用できます。V1 ログ記録を使用するお客様は、これらの機能にアクセスするには V2 への移行を実行する必要があります。詳細については、「」を参照してください。

AWS IoT ログ記録の設定方法を検討する場合、特に指定がない限り、アカウントレベルのログ記録設定によって AWS IoT アクティビティのログ記録方法が決まります。最初は、INFO または DEBUG のデフォルトのログレベルで詳細なログを取得できます。初期ログを確認したら、デフォルトのログレベルをアカウントまたはイベントレベルで WARN や ERROR などのより詳細なレベルに変更し、より注意が必要なリソースに対してより詳細なリソース固有のログレベルを設定できます。ログレベルはいつでも変更できます。

このトピックでは、クラウド側のログインについて説明します AWS IoT。デバイス側のログ記録とモニタリングについては、「CloudWatch にデバイス側のログをアップロード」を参照してください。

ログ記録とモニタリングの詳細については AWS IoT Greengrass、「 でのログ記録とモニタリング AWS IoT Greengrass」を参照してください。

V2 ロギングの設定 AWS IoT

ログ記録のバージョンの確認

VGetV2LoggingOptions API は NotConfiguredException を返します。 V2 このエラーは、V1 ログ記録が使用中であるか、ログ記録が設定されていない場合に発生します。

V2 ログ記録機能について

V2 ログ記録には、イベントレベルのログ記録とリソース固有のログ記録の 2 つの主要な機能があります。イベントレベルのログ記録により、カスタマイズ可能なログレベルと CloudWatch ロググループの送信先を使用して、ターゲットを絞ったログ記録設定が可能になります。リソース固有のログ記録を使用すると、モノのグループ、ソース IP、クライアント ID、またはプリンシパル ID でログをフィルタリングできます。これらの機能により、IoT オペレーションをきめ細かく制御し、包括的に可視化できるため、不要なログ記録アクティビティがなくなるため、ログの検索可能性が向上し、コストを削減できます。

V1 から V2 への移行

AWS CLI または SDK を使用して、SetV2LoggingOptions API を使用して V2 ログ記録に移行できます。 SetV2LoggingOptions 移行後、 はすべてのログを CloudWatch ロググループAWSIotLogsV2 AWS IoT 」に自動的にルーティングします。重要: ダウンストリームアプリケーションまたはリソースがAWSIotLogs」の情報を使用する場合は、対応するロググループパスを使用するように更新します。

ログ記録ロールとポリシーの構成

ログインを有効にする前に AWS IoT、ユーザーに代わって CloudWatch ロググループに AWS IoT ログアクティビティを書き込む AWS IoT アクセス許可を付与する IAM ロールとポリシーを作成する必要があります。AWS IoT コンソールの Logs セクションで必要なポリシーを使用して IAM ロールを生成することもできます。

> **Note**
>
> AWS IoT ログ記録を有効にする前に、CloudWatch Logs アクセス許可を理解していることを確認してください。CloudWatch Logs に対するアクセス許可のあるユーザーはデバイスからのデバッグ情報を表示できます。詳細については、「Amazon CloudWatch Logs に対する認証とアクセスコントロール」を参照してください。

負荷テスト AWS IoT Core により でトラフィックパターンが高くなることが予想される場合は、スロットリングを防ぐために IoT ログ記録を無効にすることを検討してください。大量のトラフィックが検出された場合、当社のサービスはアカウントでのログ記録を無効にする場合があります。

AWS IoT Core リソースのログ記録ロールとポリシーを作成する方法を以下に示します。

ログ記録ロールの作成

ログ記録ロールを作成するには、IAM コンソールの [Roles] (ロール) ハブを開き、[Create role] (ロールの作成) を選択します。

1. [信頼されたエンティティを選択] で、[AWS サービス] を選択します。次に、[Use case] (ユースケース) で [IoT] を選択します。[IoT] が表示されない場合は、[他の AWS サービスのユースケース] ドロップダウンメニューで [IoT] と入力して検索してください。[次へ] を選択してください。
2. [Add permissions] (権限の追加) ページでは、サービスロールに自動的に付加されるポリシーが表示されます。[次へ] を選択します。
3. [Name, review, and create] (名前、確認、作成) ページで、ロールの [Role name] (ロール名) と [Role description] (ロールの説明) を入力し、次に [Create role] (ロールの作成) を選択します。

ログ記録ロールのポリシー

以下のポリシードキュメントは、お客様に代わって AWS IoT から CloudWatch にログエントリを送信するためのロールポリシーと信頼ポリシーを示しています。カスタム CloudWatch ロググループでイベントレベルのログ記録を設定する場合は、ロールポリシーを更新してカスタムリソース ARN を含める必要があります。

ログエントリ AWS IoT Core for LoRaWAN の送信も許可されている場合は、両方のアクティビティをログに記録するポリシードキュメントが作成されます。

> **Note**
>
> これらのドキュメントは、ログ記録ロールを作成したときに作成されたものです。ドキュメントには変数 *${partition}#${region}*、 *${accountId}* があり、 を値に置き換える必要があります。
>
> - partition をリージョンのパーティションに置き換えます。

- region を、使用する AWS リージョンに置き換えます。デバイスで CLI AWS の設定に使用したリージョンと同じ AWS リージョンを使用していることを確認してください。
- account-id を自分の AWS アカウント ID に置き換えます。

ロールポリシー:

```
{
    "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 アクティビティのみをログに記録する信頼ポリシー:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

AWS IoT IAM ロールのアクセス許可が不十分なため、ログ記録が CloudWatch Logs へのログ記録の発行に失敗する可能性があります。これが発生した場合は、CloudWatch ログ記録メトリクスをチェックして、障害を調査およびトラブルシューティングします。

でログ記録を設定する AWS IoT (コンソール)

このセクションでは、 AWS IoT コンソールを使用して AWS IoT ログ記録を設定する方法について説明します。アカウントレベル、イベントレベル、リソース固有のログ記録を設定できます。

AWS IoT ログ記録を設定するには:

1. AWS IoT コンソールにサインインします。詳細については、「AWS IoT コンソールを開く」を参照してください。
2. 左側のナビゲーションペインで、ログ (以前の設定のセクション) を選択します。
3. アカウントレベルのログ記録を設定する: イベントレベルまたはリソース固有の設定によって上書きされない限り、アカウントレベルのログ記録はすべての AWS IoT フリート (デバイスまたはエンドポイント) に適用されます。

    a. アカウントレベルのログ記録で、アカウントレベルのログ記録の管理を選択して更新を行います。

    b. CloudWatch へのログの送信を開始するには、「ログ記録を有効にする」チェックボックスを選択します。「ログ記録を有効にする」がチェックされていない場合、イベントレベルまたはリソースレベルのログ記録設定に関係なく、 AWS IoT は CloudWatch ロググループにログを送信しません。

    c. IAM ログロールで、ドロップダウンリストから既存のロールを選択します。ロールの詳細を表示して、ロールのアクセス許可を検査できます。または、新しいロールの作成を選択して新しい IAM ロールを設定します。ログ記録ロールは、 がユーザーに代わって CloudWatch AWS IoT にログエントリを送信できるようにするポリシーを提供します。カス

タム CloudWatch ロググループでイベントレベルのログ記録を設定する場合は、ロールポリシーを更新して、このロググループの ARN を含める必要があります。

d. CloudWatch Logs に表示するログエントリの詳細レベルに対応するデフォルトのログレベルを選択します。 https://docs.aws.amazon.com/iot/latest/developerguide/configure-logging.html#log-level注: ログレベル「DEBUG」は最も詳細ですが、CloudWatch のコストが増加します。CloudWatch ロググループの送信先をアカウントレベルで設定することはできません。ただし、次のセクションで説明するように、個々のイベントタイプのカスタムロググループを指定できます。

e. ログ記録の更新を選択して変更を保存します。

4. イベントレベルのログ記録を使用すると、関連するイベントのログを選択的にキャプチャし、専用の CloudWatch ロググループに送信できます。これにより、ログを有効にし、イベントの重要度に基づいてログレベルを設定することで、検出可能性を高め、さまざまな対象者と共有し、CloudWatch コストを削減するために、ユースケースごとにログを柔軟に整理できます。

   イベントレベルのログ記録を設定する: イベントレベルのログ記録は、クライアント認証の試行などの特定の AWS IoT イベントをキャプチャします。これらの設定は、アカウントレベルのログ記録を上書きします。

   a. イベントレベルのログ記録セクションで、イベントレベルのログ記録を管理を選択して更新を行います。

   b. デフォルトでは、イベントタイプはアカウントレベルのログ記録設定を継承します。注: リソース固有のログ記録を設定すると、アカウントとイベントレベルの設定が上書きされます。

   c. 個々のイベントの設定を変更するには、対応するイベント行の値をクリックします。ログレベルと CloudWatch ロググループの送信先の両方を調整できます。カスタム CloudWatch ロググループの送信先を指定するときは、IAM ロールポリシーに新しいロググループのアクセス許可が含まれていることを確認する必要があります。ロールポリシーを更新しないと、 AWS IoT はカスタムロググループにログを書き込むことができません。選択したら、チェックマークをクリックして選択を確定します。「Is Modified」列に「Yes」と表示され、保留中の変更を示します。

   d. ログ記録の更新をクリックして変更を適用するか、キャンセルを選択して破棄します。

5. リソース固有のオーバーライドを設定する: リソース固有のオーバーライドは、選択したリソースにログ記録設定を適用します。リソースには、モノのグループ、ソース IP、クライアント ID、またはプリンシパル ID を指定できます。リソース固有のログ記録設定は、アカウントレベルとイベントレベルの両方の設定を上書きします。有効にすると、指定されたリソースに対して設定されたログ記録レベルですべてのイベントタイプのログが生成されます。たとえば、他のす

べてのモノの情報レベルのログ記録を維持しながら、特定のモノのデバッグレベルのログ記録を設定できます。

a. 「リソース固有のオーバーライドの追加」セクションの「リソース固有のオーバーライドの追加」を選択します。
b. ログターゲットを選択します: モノのグループ、ソース IP、クライアント ID、またはプリンシパル ID。
c. 選択したターゲットタイプに対応するログターゲット値を入力します。
d. リソース固有のログレベルセクションのドロップダウンメニューから目的のログレベルを選択します。
e. 送信をクリックしてオーバーライドを追加するか、キャンセルをクリックして変更を破棄します。
f. 既存のリソース固有のオーバーライドを変更するには、リソースの横にあるチェックボックスを選択し、「削除」をクリックしてオーバーライドを削除するか、「編集」をクリックして変更します。

ログ記録を有効にしたら、CloudWatch コンソールでの AWS IoT ログの表示 にアクセスして、ログエントリの表示方法の詳細を確認します。

アカウントとイベントレベルのログインを設定する AWS IoT (CLI)

このセクションでは、 CLI AWS IoT を使用して のグローバルログ記録を設定する方法について説明します。

オプションで、イベントレベルのログ記録を設定できます。イベントレベルのログ記録は、認証、認可、証明書作成イベントなどのログ情報をイベントレベルでキャプチャします。ログレベルと CloudWatch ロググループの送信先の両方をイベントレベルでカスタマイズできます。イベントレベルのログ記録は、アカウントレベルのログ記録よりもターゲットを絞ったレベルで機能するため、アカウントレベルのログ記録設定を上書きします。この階層的アプローチにより、運用上の重要性とコストの考慮事項に基づいて、さまざまなタイプのイベントに対してさまざまなログ記録戦略を維持できます。

> **Note**
>
> 使用するロールの Amazon リソースネーム (ARN) が必要です。ログ記録に使用するロールを作成する必要がある場合は、先へ進む前に「ログ記録ロールの作成」を参照してくださ

い。任意のイベントタイプにカスタム CloudWatch ロググループを指定するときは、ログ記録ロールにターゲットロググループに必要なアクセス許可があることを確認してください。API を呼び出すために使用されるプリンシパルには、ログ記録ロール用のロールのアクセス許可の適用が必要です。

ここに示す CLI コマンドに対応する API のメソッドを使用して、 AWS API でこの手順を実行することもできます。

CLI を使用して のデフォルトログ記録を設定するには AWS IoT

1. set-v2-logging-options コマンドを使用して、アカウントのログ記録オプションを設定します。

```
aws iot set-v2-logging-options \
    --event-configurations event-configuration-list \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

各パラメータの意味は次のとおりです。

--role-arn

CloudWatch Logs のログに書き込むアクセス AWS IoT 許可を付与するロール ARN。初期設定には Role-arn 設定が必要です。

--default-log-level

使用するログレベル。有効な値は、ERROR、、WARN、INFODEBUG、または ですDISABLED。初期設定には、Default-log-level設定が必要です。

--no-disable-all-logs

すべての AWS IoT ログ記録を有効にするオプションのパラメータ。このパラメータは、現在無効になっているログ記録を有効にする場合に使用します。

--disable-all-logs

すべての AWS IoT ログ記録を無効にするオプションのパラメータ。このパラメータは、現在有効になっているログ記録を無効にする場合に使用します。

--event-configurations

このパラメータはオプションであり、個々のイベントタイプのログ記録設定をカスタマイズできます。

- eventType: アカウントレベルの設定を上書きするイベントのタイプ
- logLevel: DEBUG、INFO、ERROR、WARN、または DISABLED でアカウントレベルの設定を上書きする
- logDestination: ログ配信用のカスタム CloudWatch ロググループを指定する

ログ記録レベルとログ送信先は、イベントタイプごとに個別に設定できます。指定しない場合、イベントはアカウントレベルの設定を継承します。

```
aws iot set-v2-logging-options \
    --event-configurations "[{\"eventType\":\"Publish-In\",\"logLevel\":\"INFO\",
\"logDestination\":\"examplePublishInLogGroup\"}]"
```

2. 現在のログ記録オプションを取得するには、get-v2-logging-options コマンドを使用します。

```
aws iot get-v2-logging-options \
    --verbose
```

各パラメータの意味は次のとおりです。

--verbose

すべての eventTypes とその設定を取得できるオプションのパラメータ。

ログ記録を有効にしたら、CloudWatch コンソールでの AWS IoT ログの表示 にアクセスして、ログエントリの表示方法の詳細を確認します。

> **Note**
>
> AWS IoT は、 アカウントでグローバルログを設定および取得するための古いコマンド (set-logging-options および get-logging-options) を引き続きサポートします。これらのコマンドを使用すると、作成されるログには JSON ペイロードではなくプレーンテキストが含まれ、一般的にログ記録のレイテンシーが高くなることに注意してください。これらの古いコマンドの実装は今後強化されません。「v2」バージョンを使用してログ記録オプションを設定し、

可能であれば古いバージョンを使用するレガシーアプリケーションを変更することをお勧めします。

AWS IoT (CLI) でリソース固有のオーバーライドを設定する

このセクションでは、 CLI AWS IoT を使用して のリソース固有のオーバーライドを設定する方法について説明します。リソース固有のオーバーライドを使用すると、モノのグループ、クライアント ID、ソース IP、またはプリンシパル ID で識別される特定のリソースのログ記録レベルを指定できます。リソース固有のログ記録を有効にすると、アカウントレベルとイベントレベルの両方の設定が上書きされます。すべてのイベントタイプは、イベントレベルの設定でそれらのイベントが無効になっている場合でも、設定されたログ記録レベルで指定されたリソースのログを生成します。

モノグループには別のモノグループを含めて、階層的な関係を作成することができます。この手順では、単一のモノグループのログ記録を設定する方法について説明します。階層内の親モノグループにこの手順を適用することにより、階層内のすべてのモノグループに対してログ記録を設定することもできます。この手順を子モノグループに適用して、親のログ記録設定をオーバーライドすることもできます。

モノはモノグループのメンバーにすることができます。メンバーになることで、そのモノのグループに適用される設定、ポリシー、および設定をモノが継承できるようになります。モノのグループは、各モノを個別に処理するのではなく、複数のモノをまとめて管理および適用するために使用されます。クライアント ID がモノの名前と一致すると、 AWS IoT Core はクライアントセッションを対応するモノのリソースに自動的に関連付けます。これによりクライアントセッションは、モノが属するモノのグループに適用される構成や設定 (ログ記録レベルなど) を継承できるようになります。クライアント ID がモノの名前と一致しない場合は、排他的なモノのアタッチメントを有効にして関連付けを確立することができます。詳細については、「???」を参照してください。

モノグループに加えて、デバイスのクライアント ID、ソース IP、プリンシパル ID などのターゲットをログに記録することもできます。

> **Note**
>
> 使用するロールの Amazon リソースネーム (ARN) が必要です。ログ記録に使用するロールを作成する必要がある場合は、先へ進む前に「ログ記録ロールの作成」を参照してください。
> API を呼び出すために使用されるプリンシパルには、ログ記録ロール用のロールのアクセス許可の適用が必要です。

ここに示す CLI コマンドに対応する API のメソッドを使用して、 AWS API でこの手順を実行することもできます。

CLI を使用して のリソース固有のオーバーライドを設定するには AWS IoT

1. 次のコマンドを使用してリソース固有のログ記録を設定する前に、アカウントレベルのログ記録を有効にします。aws iot set-v2-logging-options コマンド
2. set-v2-logging-level コマンドを使用して、リソース固有のオーバーライドを設定します。モノのグループ設定については、次の例を参照してください。

```
aws iot set-v2-logging-level \
              --log-target targetType=THING_GROUP,targetName=thing_group_name \
              --log-level log_level
```

--log-target

ログ記録を設定するリソースの名前とタイプ。targetTypeの値は次のいずれかである必要があります : THING_GROUP | CLIENT_ID | SOURCE_IP | PRINCIPAL_ID。log-target パラメータ値には、前のコマンド例で示したテキストや、次の例のような JSON 文字列を指定できます。

```
aws iot set-v2-logging-level \
              --log-target '{"targetType": "THING_GROUP","targetName":
 "thing_group_name"}' \
              --log-level log_level
```

--log-level

指定されたリソースのログを生成する際に使用されるログ記録レベル。有効な値は、DEBUG、INFO、ERROR、WARN、DISABLED です。

3. 現在設定されているログ記録レベルを一覧表示するには、list-v2-logging-levels コマンドを使用します。

```
aws iot list-v2-logging-levels
```

4. 次の例のように、リソース固有のログ記録レベルを削除するには、delete-v2-logging-level コマンドを使用します。

```
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の値は次のいずれかである必要があります : THING_GROUP | CLIENT_ID | SOURCE_IP | PRINCIPAL_ID。

--target-name

ログ記録レベルを削除するモノグループの名前。

ログレベル

以下のログレベルは、ログにどのイベントを記録するかを決定します。デフォルトおよびリソース固有のログレベルに適用されます。

ERROR

オペレーションの失敗につながるすべてのエラー。

例: 証明書の有効期限が切れたため、デバイスの認証に失敗しました。

ログには ERROR 情報のみが含まれます。

WARN

オペレーションの失敗につながるわけではないが、システムの不整合を引き起こす可能性のあるすべての要因。

例: メッセージレート制限に近づいています。

ログには ERROR および WARN 情報が含まれます。

INFO

モノのフローに関する概要。

例: クライアントが MQTT トピックに正常にサブスクライブしました。

ログには INFO、ERROR および WARN 情報が含まれます。

DEBUG

問題のデバッグに役立つ場合のある情報。

例: IoT Rules Engine がルールトピック「rule/test」に発行されたメッセージを検出し、正常に実行を開始しました。ルールは RepublishAction で設定されます。

ログには DEBUG、INFO、ERROR、WARN 情報が含まれます。

DISABLED

すべての口ギングが無効です。

Amazon CloudWatch を使用して AWS IoT アラームとメトリクスをモニタリングする

CloudWatch AWS IoT を使用してCloudWatch は から raw データを収集し、ほぼリアルタイムの読み取り可能なメトリクス AWS IoT に加工します。これらの統計は 2 週間記録されるため、履歴情報にアクセスしてウェブアプリケーションまたはサービスの動作をより的確に把握することができます。デフォルトでは、 AWS IoT メトリクスデータは 1 分間隔で CloudWatch に自動的に送信されます。詳細については、Amazon CloudWatch ユーザーガイドの Amazon CloudWatch、Amazon CloudWatch Events、Amazon CloudWatch Logs とはをご覧ください。

AWS IoT メトリクスの使用

によってレポートされるメトリクスは、さまざまな方法で分析できる情報 AWS IoT を提供します。以下のユースケースは、10 個のモノが 1 日に 1 回インターネットに接続するというシナリオに基づいています。毎日:

- 10 AWS IoT 個のモノがほぼ同時に に接続します。
- 各モノはトピックフィルタにサブスクライブし、接続を切断するまで 1 時間待機します。この期間中、モノは相互に情報をやり取りし、現在の状態について詳細を取得します。
- 各モノは、`UpdateThingShadow` を使用して、その新たに取得したデータに基づいて、何らかのパーセプションをパブリッシュします。
- 各モノは から切断されます AWS IoT。

これらのトピックでは、作業を開始しやすくするために、いくつかの質問について解説します。

- モノが毎日正常に接続していない場合に通知されるようにするには?
- モノが毎日正常にデータをパブリッシュしていない場合に通知されるようにするには?
- Thing Shadow の更新が毎日拒否されている場合に通知されるようにするには?
- ジョブ用の CloudWatch アラームを作成するにはどうすればよいですか。

CloudWatch のアラームとメトリクスの詳細

- モニタリングする CloudWatch アラームの作成 AWS IoT
- AWS IoT メトリクスとディメンション

モニタリングする CloudWatch アラームの作成 AWS IoT

アラームの状態が変わったら、Amazon SNS メッセージを送信する Amazon CloudWatch のアラームを作成することができます。1 つのアラームで、指定した期間中、1 つのメトリクスを監視します。複数の期間にわたってメトリクスの値が一定のしきい値を超えると、1 つ以上のアクションが実行されます。アクションは、Amazon SNS のトピックまたは Auto Scaling のポリシーに送信される通知とすることができます。アラームは、持続している状態変化に対してのみアクションをトリガーします。CloudWatch のアラームは、メトリクスが特定の状態になっただけではアクションをトリガーしません。アクションを呼び出すには、状態が変化して、指定した期間継続している必要があります。

次のトピックでは、CloudWatch アラームの使用例について解説します。

- モノが毎日正常に接続していない場合に通知されるようにするには?
- モノが毎日正常にデータをパブリッシュしていない場合に通知されるようにするには?
- Thing Shadow の更新が毎日拒否されている場合に通知されるようにするには?
- ジョブ用の CloudWatch アラームを作成するにはどうすればよいですか?

CloudWatch アラームがモニタリングできるすべてのメトリクスは、AWS IoT メトリクスとディメンション で確認できます。

モノが毎日正常に接続していない場合に通知されるようにするには?

1. `things-not-connecting-successfully` という名前の Amazon SNS トピックを作成し、対応する Amazon リソースネーム (ARN) を記録します。この手順では、トピックの ARN を *`sns-topic-arn`* とします。

Amazon SNS 通知の作成方法については、[Getting Started with Amazon SNS](Amazon SNS の開始方法) を参照してください。

2. アラームを作成します。

```
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. アラームのテストを行います。

```
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. アラームが CloudWatch コンソールに表示されていることを確認します。

モノが毎日正常にデータをパブリッシュしていない場合に通知されるようにするには?

1. things-not-publishing-data という名前の Amazon SNS トピックを作成し、対応する Amazon リソースネーム (ARN) を記録します。この手順では、トピックの ARN を *sns-topic-arn* とします。

   Amazon SNS 通知の作成方法については、[Getting Started with Amazon SNS](Amazon SNS の開始方法) を参照してください。

2. アラームを作成します。

```
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. アラームのテストを行います。

```
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. アラームが CloudWatch コンソールに表示されていることを確認します。

Thing Shadow の更新が毎日拒否されている場合に通知されるようにするには?

1. things-shadow-updates-rejected という名前の Amazon SNS トピックを作成し、対応する Amazon リソースネーム (ARN) を記録します。この手順では、トピックの ARN を *sns-topic-arn* とします。

   Amazon SNS 通知の作成方法については、[Getting Started with Amazon SNS](Amazon SNS の開始方法) を参照してください。

2. アラームを作成します。

```
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. アラームのテストを行います。

```
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. アラームが CloudWatch コンソールに表示されていることを確認します。

ジョブ用の CloudWatch アラームを作成するにはどうすればよいですか?

ジョブサービスは、ジョブをモニタリングするための CloudWatch メトリクスを提供します。CloudWatch アラームを作成して、任意の ジョブのメトリクス をモニタリングできます。

次のコマンドは、Job *SampleOTAJob* の失敗したジョブ実行の合計数を監視する CloudWatch アラームを作成し、20 を超えるジョブ実行が失敗した場合に通知します。アラームは、レポートされた値を 300 秒ごとにチェックして、ジョブメトリクス FailedJobExecutionTotalCount をモニタリングします。報告された単一の値が 20 より大きい場合にアクティブになります。つまり、ジョブの開始以降、ジョブの実行が 20 回以上失敗したことを示します。アラームがオフになると、提供された Amazon SNS トピックに通知が送信されます。

```
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
```

次のコマンドは、CloudWatch アラームを作成して、特定の期間に Job *SampleOTAJob* の失敗したジョブの実行数を監視します。その後、その期間中に 5 つ以上のジョブの実行が失敗すると通知します。アラームは、レポートされた値を 3600 秒ごとにチェックして、ジョブメトリクス FailedJobExecutionCount をモニタリングします。レポートされた単一の値が 5 より大きい場合にアクティブになります。つまり、過去 1 時間で 5 回以上ジョブ実行に失敗したということです。アラームがオフになると、提供された Amazon SNS トピックに通知が送信されます。

```
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 メトリクスとディメンション

を操作すると AWS IoT、サービスはメトリクスとディメンションを 1 分ごとに CloudWatch に送信します。CloudWatch コンソールまたは を使用して AWS IoT、これらのメトリクス AWS CLI を表示できます。

CloudWatch コンソールを使用してメトリクスを表示するには、[CloudWatch コンソール] を開きます。ナビゲーションペインで、[Metrics] (メトリクス) を選択し、次に、[All metrics] (すべてのメトリクス) を選択します。参照タブで、 を検索 AWS IoT してメトリクスのリストを表示します。メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

を使用してメトリクスを表示するには AWS CLI、次のコマンドを実行します。

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch には、次のメトリクスグループが表示されます AWS IoT。

- AWS IoT メトリクス
- AWS IoT Core 認証情報プロバイダーメトリクス
- 認証メトリクス
- サーバー証明書の OCSP ステープリングメトリクス
- ルールのメトリクス
- ルールアクションのメトリクス
- HTTP アクション固有のメトリクス
- メッセージブローカーのメトリクス
- Device Shadow のメトリクス
- メトリクスのログ記録
- ジョブのメトリクス
- Device Defender の監査メトリクス
- Device Defender の検出メトリクス
- デバイスプロビジョニングのメトリクス
- LoRaWAN メトリクス
- フリートのインデックス作成メトリクス
- メトリクスのディメンション

AWS IoT メトリクス

メトリクス	説明
AddThingToDynamicThingGroupsFailed	モノの動的グループへのモノの追加に関連する失敗イベントの数。DynamicThingGroupName ディメンションには、モノの追加に失敗した動的グループの名前が含まれています。
NumLogBatchesFailedToPublishThrottled	スロットリングエラーのために発行に失敗したログイベントの単数のバッチ。

メトリクス	説明
NumLogEventsFailedToPublishThrottled	バッチ内でスロットリングエラーのために発行に失敗したログイベントの数。

AWS IoT Core 認証情報プロバイダーメトリクス

メトリクス	説明
CredentialExchangeSuccess	AWS IoT Core 認証情報プロバイダーへの AssumeRoleWithCertificate リクエストが成功した数。

認証メトリクス

> **Note**
> 認証メトリクスは、CloudWatch コンソールの [プロトコルメトリクス] に表示されます。

メトリクス	説明
Connection.AuthNError	認証の失敗により が AWS IoT Core 拒否する接続試行の数。このメトリクスは、 AWS アカウントのエンドポイントに一致するサーバー名表示 (SNI) 文字列を送信する接続のみを考慮します。このメトリクスには、インターネットスキャンツールやプローブアクティビティなどの外部ソースからの接続試行が含まれます。Protocol ディメンションには、接続の試行の送信に使用されたプロトコルが含まれます。

サーバー証明書の OCSP ステープリングメトリクス

メトリクス	説明
RetrieveOCSPStapleData.Success	OCSP レスポンスは正常に受信および処理されました。このレスポンスは、設定されたドメインの TLS ハンドシェイク中に含まれます。DomainConfigurationName ディメンションには、サーバー証明書 OCSP ステープリングが有効になっている設定済みドメインの名前が含まれます。

ルールのメトリクス

メトリクス	説明
ParseError	ルールがリッスンしているトピックで発行されたメッセージで発生した JSON 解析エラーの数。RuleName ディメンションにはルールの名前が含まれます。
RuleExecutionThrottled	悪意のある動作のため、またはメッセージの数がルールエンジンのスロットル制限を超えているために、ルールエンジンによってスロットリングされたメッセージの数。RuleName ディメンションには、トリガーされるルールの名前が入っています。
RuleNotFound	トリガーされるルールが見つかりませんでした。RuleName ディメンションにはルールの名前が含まれます。
RulesExecuted	実行された AWS IoT ルールの数。
TopicMatch	ルールがリッスンしているトピックで発行された受信メッセージの数。RuleName ディメンションにはルールの名前が含まれます。

ルールアクションのメトリクス

メトリクス	説明
Failure	失敗したルールアクションの呼び出しの数。RuleName ディメンションには、アクションを指定するルールの名前が含まれます。ActionType ディメンションには、呼び出されたアクションのタイプが含まれます。
Success	正常なルールアクションの呼び出しの数。RuleName ディメンションには、アクションを指定するルールの名前が含まれます。ActionType ディメンションには、呼び出されたアクションのタイプが含まれます。
ErrorActionFailure	失敗したエラーアクションの数。RuleName ディメンションには、アクションを指定するルールの名前が含まれます。ActionType ディメンションには、呼び出されたアクションのタイプが含まれます。
ErrorActionSuccess	成功したエラーアクションの数。RuleNameディメンションには、アクションを指定するルールの名前が含まれます。ActionType ディメンションには、呼び出されたアクションのタイプが含まれます。

HTTP アクション固有のメトリクス

メトリクス	説明
HttpCode_Other	ダウンストリームウェブサービス/アプリケーションからの応答のステータスコードが 2xx、4xx、または 5xx でない場合に生成されます。
HttpCode_4XX	ダウンストリームウェブサービス/アプリケーションからの応答のステータスコードが 400 ~ 499 の場合に生成されます。

メトリクス	説明
HttpCode_5XX	ダウンストリームウェブサービス/アプリケーションからの応答のステータスコードが 500 ~ 599 の場合に生成されます。
HttpInvalidUrl	置換テンプレートが置き換えられた後にエンドポイント URL が https:// で始まらない場合に生成されます。
HttpRequestTimeout	ダウンストリームウェブサービス/アプリケーションが要求タイムアウト制限内で応答を返さない場合に生成されます。詳細については、「Service Quotas」を参照してください。
HttpUnknownHost	URL が有効であるが、サービスが存在しないか、到達不能である場合に生成されます。

メッセージブローカーのメトリクス

> **Note**
>
> メッセージブローカーのメトリクスは、CloudWatch コンソールの [Protocol Metrics] (プロトコルメトリクス) に表示されます。

メトリクス	説明
Connect.AuthError	メッセージブローカーが承認できなかった接続リクエストの数。Protocol ディメンションには、CONNECT メッセージの送信に使用されたプロトコルが含まれます。
Connect.ClientError	MQTT メッセージが で定義された要件を満たさなかったために拒否された接続リクエストの数AWS IoT のクォータ。Protocol ディメンションには

メトリクス	説明
	、CONNECT メッセージの送信に使用されたプロトコルが含まれます。
Connect.ClientIDThrottle	特定のクライアント ID で許可された接続リクエストの頻度をクライアントが超えたために調整された接続リクエストの数。Protocol ディメンションには、CONNECT メッセージの送信に使用されたプロトコルが含まれます。
Connect.ServerError	内部エラーが発生したために失敗した接続リクエストの数。Protocol ディメンションには、CONNECT メッセージの送信に使用されたプロトコルが含まれます。
Connect.Success	メッセージブローカーへ正常な接続の数。Protocol ディメンションには、CONNECT メッセージの送信に使用されたプロトコルが含まれます。
Connect.Throttle	許可された接続リクエストの頻度をアカウントが超えたために調整された接続リクエストの数。Protocol ディメンションには、CONNECT メッセージの送信に使用されたプロトコルが含まれます。
Ping.Success	メッセージブローカーによって受け取られた ping メッセージの数。Protocol ディメンションには、ping メッセージの送信に使用されたプロトコルが含まれます。
PublishIn.AuthError	メッセージブローカーが承認できなかった発行リクエストの数。Protocol ディメンションには、メッセージの発行に使用されたプロトコルが含まれます。HTTP Publish はこのメトリクスをサポートしていません。

メトリクス	説明
PublishIn.ClientError	メッセージが で定義された要件を満たさなかったためにメッセージブローカーによって拒否された発行リクエストの数AWS IoT のクォータ。Protocol ディメンションには、メッセージの発行に使用されたプロトコルが含まれます。
PublishIn.ServerError	内部エラーが発生したためにメッセージブローカーが処理に失敗した発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishIn.Success	メッセージブローカーによって正常に処理された発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishIn.Throttle	許可されたインバウンドメッセージの頻度をクライアントが超えたために調整された発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。HTTP Publish はこのメトリクスをサポートしていません。
PublishOut.AuthError	AWS IoTが承認できなかった、メッセージブローカーによって行われた発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishOut.ClientError	メッセージブローカーによって行われた発行リクエストのうち、メッセージが AWS IoT のクォータ で定義された要件を満たしていないために拒否されたものの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。

メトリクス	説明
PublishOut.Success	メッセージブローカーによって正常に行われた発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishOut.Throttle	許可されたアウトバウンドメッセージの頻度をクライアントが超えたために調整された発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishRetained.AuthError	メッセージブローカーが承認できなかった、RETAIN フラグが設定されている発行リクエストの数。Protocol ディメンションには、PUBLISHメッセージの送信に使用されたプロトコルが含まれます。
PublishRetained.ServerError	内部エラーが発生したためにメッセージブローカーが処理に失敗した、保持されている発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishRetained.Success	メッセージブローカーによって正常に処理された、RETAIN フラグが設定されている発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。
PublishRetained.Throttle	クライアントが許可されたインバウンドメッセージレートを超過したためにスロットルされた、RETAIN フラグが設定されている発行リクエストの数。Protocol ディメンションには、PUBLISH メッセージの送信に使用されたプロトコルが含まれます。

メトリクス	説明
Queued.Success	永続セッションから切断されたクライアントのメッセージブローカーによって正常に処理された、保存されているメッセージの数。QoS が 1 のメッセージは、クライアントの永続セッションが切断されている間も保存されます。
Queued.Throttle	クライアントの永続セッションが接続が切断されている間に保存できず、スロットリングされたメッセージの数。これは、クライアントのアカウントごとの 1 秒あたりのキューメッセージ数が上限を超えた場合に発生します。QoS が 1 のメッセージは、クライアントの永続セッションが切断されている間も保存されます。
Queued.ServerError	内部エラーのために永続セッションに保存されなかったメッセージの数。クライアントの永続セッションが切断されると、サービス品質 (QoS) が 1 のメッセージが保存されます。
Subscribe.AuthError	クライアントによって行われ、承認できなかったサブスクリプションリクエストの数。Protocol ディメンションには、SUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。
Subscribe.ClientError	SUBSCRIBE メッセージが で定義された要件を満たさなかったために拒否されたサブスクライブリクエストの数AWS IoT のクォータ。Protocol ディメンションには、SUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。
Subscribe.ServerError	内部エラーが発生したために拒否されたサブスクリプションのリクエストの数。Protocol ディメンションには、SUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。

メトリクス	説明
Subscribe.Success	メッセージブローカーによって正常に処理されたサブスクリプションのリクエストの数。Protocol ディメンションには、SUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。
Subscribe.Throttle	許可されたサブスクリプションリクエストの頻度が AWS アカウントに対して超えたためにスロットリングされたサブスクリプションリクエストの数。これらの制限には、「AWS IoT Core メッセージブローカーとプロトコルの制限とクォータ」に記載されている、アカウントごとの 1 秒あたりのサブスクリプション、アカウントごとのサブスクリプション、接続あたりのサブスクリプションが含まれます。Protocol ディメンションには、SUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。
Throttle.Exceeded	このメトリクスは、MQTT クライアントが接続レベルの制限ごとに 1 秒あたりのパケット数をスロットリングされている場合に CloudWatch に表示されます。このメトリクスは HTTP 接続には適用されません。
Unsubscribe.ClientError	UNSUBSCRIBE メッセージが で定義された要件を満たさなかったために拒否されたサブスクリプション解除リクエストの数AWS IoT のクォータ。Protocol ディメンションには、UNSUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。
Unsubscribe.ServerError	内部エラーが発生したために拒否されたサブスクリプション解除リクエストの数。Protocol ディメンションには、UNSUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。

メトリクス	説明
Unsubscribe.Success	メッセージブローカーによって正常に処理されたサブスクリプション解除リクエストの数。Protocol ディメンションには、UNSUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。
Unsubscribe.Throttle	許可されたサブスクリプション解除リクエストの頻度をクライアントが超えたために拒否されたサブスクリプション解除リクエストの数。Protocol ディメンションには、UNSUBSCRIBE メッセージの送信に使用されたプロトコルが含まれます。

Device Shadow のメトリクス

> **Note**
>
> Device Shadow メトリクスは、CloudWatch コンソールの [Protocol Metrics] (プロトコルメトリクス) に表示されます。

メトリクス	説明
DeleteThingShadow.Accepted	正常に処理された DeleteThingShadow リクエストの数。Protocol ディメンションには、リクエストの作成に使用されたプロトコルが含まれます。
GetThingShadow.Accepted	正常に処理されたGetThingShadow リクエストの数。Protocol ディメンションには、リクエストの作成に使用されたプロトコルが含まれます。
ListThingShadow.Accepted	正常に処理されたListThingShadow リクエストの数。Protocol ディメンションには、リクエストの作成に使用されたプロトコルが含まれます。

メトリクス	説明
UpdateThingShadow.Accepted	正常に処理されたUpdateThingShadow リクエストの数。Protocol ディメンションには、リクエストの作成に使用されたプロトコルが含まれます。

メトリクスのログ記録

メトリクス	説明
CloudwatchLogs:LogGroupCreationFailed	CreateLogGroup リクエストの失敗の数。AWS IoT ログ記録では、CloudWatch ロググループが存在しない場合は作成します。LogGroup ディメンションには、リクエストの実行に使用されるロググループが含まれます。
CloudwatchLogs:LogStreamCreationFailed	CreateLogStream リクエストの失敗の数。AWS IoT ログ記録は、ログイベントをプッシュするための CloudWatch ログストリームを作成します。LogGroup ディメンションには、リクエストの実行に使用されるロググループが含まれます。
CloudwatchLogs:PutRetentionPolicyFailed	PutRetentionPolicy リクエストの失敗の数。AWS IoT ログ記録では、CloudWatch ロググループが存在しない場合は作成され、保持ポリシーは 30 日間に設定されます。LogGroup ディメンションには、リクエストの実行に使用されるロググループが含まれます。
CloudwatchLogs:PutLogEventsFailed	PutLogEvents リクエストの失敗の数。LogGroup ディメンションには、リクエストの実行に使用される CloudWatch ロググループが含まれます。

ジョブのメトリクス

メトリクス	説明
CanceledJobExecutionCount	CloudWatch によって決定された期間中にステータスが CANCELED に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。
CanceledJobExecutionTotalCount	指定されたジョブのステータスが CANCELED であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。
ClientErrorCount	ジョブの実行中に生成されたクライアントエラーの数。JobId ディメンションには、ジョブの ID が含まれます。
FailedJobExecutionCount	CloudWatch によって決定された期間中にステータスが FAILED に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。
FailedJobExecutionTotalCount	指定されたジョブのステータスが FAILED であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。
InProgressJobExecutionCount	CloudWatch によって決定された期間中にステータスが IN_PROGRESS に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。

メトリクス	説明
InProgressJobExecutionTotalCount	指定されたジョブのステータスが IN_PROGRESS であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。
RejectedJobExecutionTotalCount	指定されたジョブのステータスが REJECTED であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。
RemovedJobExecutionTotalCount	指定されたジョブのステータスが REMOVED であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。
QueuedJobExecutionCount	CloudWatch によって決定された期間中にステータスが QUEUED に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。
QueuedJobExecutionTotalCount	指定されたジョブのステータスが QUEUED であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。
RejectedJobExecutionCount	CloudWatch によって決定された期間中にステータスが REJECTED に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。
RemovedJobExecutionCount	CloudWatch によって決定された期間中にステータスが REMOVED に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。

メトリクス	説明
ServerErrorCount	ジョブの実行中に生成されたサーバーエラーの数。JobId ディメンションには、ジョブの ID が含まれます。
SuccededJobExecutionCount	CloudWatch によって決定された期間中にステータスが SUCCESS に変更されたジョブ実行の数。(CloudWatch メトリクスの詳細については、「Amazon CloudWatch メトリクス」を参照してください。) JobId ディメンションには、ジョブの ID が含まれます。
SuccededJobExecutionTotalCount	指定されたジョブのステータスが SUCCESS であるジョブ実行の総数。JobId ディメンションには、ジョブの ID が含まれます。

Device Defender の監査メトリクス

メトリクス	説明
NonCompliantResources	チェックの結果、準拠していないことが判明したリソースの数。システムは実行した各監査のチェックごとに非準拠のリソースの数を報告します。
ResourcesEvaluated	準拠状況を評価したリソースの数。システムは実行した各監査のチェックごとに評価したリソースの数を報告します。
MisconfiguredDeviceDefenderNotification	の SNS 設定 AWS IoT Device Defender が正しく設定されていない場合に通知します。 ディメンション

Device Defender の検出メトリクス

メトリクス	説明
NumOfMetricsExported	クラウド側、デバイス側、またはカスタムメトリクスにエクスポートされたメトリクスの数。システムは、特定のメトリクスについて、アカウントにエクスポートされたメトリクスの数を報告します。このメトリクスは、メトリクスのエクスポートを使用するカスタマーのみが使用できます。
NumOfMetricsSkipped	クラウド側、デバイス側、またはカスタムメトリクスでスキップされたメトリクスの数。Device Defender Detect が mqtt トピックに発行するアクセス許可が不十分であるために、システムはアカウントでスキップされたメトリクスの数を特定のメトリクスについてレポートします。このメトリクスは、メトリクスのエクスポートを使用するカスタマーのみが使用できます。
NumOfMetricsExceedingSizeLimit	サイズが MQTT メッセージサイズの制約を超えたために、クラウド側、デバイス側、またはカスタムメトリクスのエクスポートのためにスキップされたメトリクスの数。サイズが MQTT メッセージサイズの制約を超えたために、特定のメトリクスについて、アカウントのエクスポートでスキップされたメトリクスの数が報告されます。このメトリクスは、メトリクスのエクスポートを使用するカスタマーのみが使用できます。
Violations	前回実施した評価以降に判明したセキュリティプロファイル動作の新しい違反の数。システムは、セキュリティプロファイル別およびその動作別に、アカウントの新しい違反の数を報告します。
ViolationsCleared	前回実施した評価以降に解決されたセキュリティプロファイル動作の違反の数。システムは、セキュリ

メトリクス	説明
	ティプロファイル別およびその動作別に、アカウントの解決された違反の数を報告します。
ViolationsInvalidated	前回実施した評価以降に (レポートデバイスがレポートを停止したか、何らかの理由でモニタリングが中止されたために) 情報が使用できなくなったセキュリティプロファイル動作の違反の数。システムは、セキュリティプロファイル別およびその動作別に、アカウント全体の無効化された違反の数を報告します。
MisconfiguredDeviceDefenderNotification	の SNS 設定 AWS IoT Device Defender が正しく設定されていない場合に通知します。 ディメンション

デバイスプロビジョニングのメトリクス

AWS IoT フリートプロビジョニングメトリクス

メトリクス	説明
ApproximateNumberOfThingsRegistered	フリートプロビジョニングによって登録されたモノの数。 カウントは一般的に正確ですが、 AWS IoT Core の分散アーキテクチャでは、登録されたモノの正確なカウントを維持することが難しくなります。 このメトリクスに使用する統計は次のとおりです。 • Max を使用して、登録されているモノの合計数を報告します。CloudWatch 集計期間中に登録されたモノの数については、RegisterThingFailed メトリクスを参照してください。

メトリクス	説明
	ディメンション: ClaimCertificateId
CreateKeysAndCertificateFailed	CreateKeysAndCertificate MQTT API の呼び出しで発生した失敗の数。 このメトリクスは、成功した場合 (値 = 0) と失敗した場合 (値 = 1) の両方で出力されます。このメトリクスを使用して、CloudWatch がサポートする集計期間中 (5 分、あるいは 1 時間など) に作成および登録された証明書の数を追跡できます。 このメトリクスに使用できる統計は次のとおりです。 • Sum を使用して、失敗した呼び出しの数を報告します。 • SampleCount を使用して、成功した呼び出しと失敗した呼び出しの合計数を報告します。
CreateCertificateFromCsrFailed	CreateCertificateFromCsr MQTT API の呼び出しで発生した失敗の数。 このメトリクスは、成功した場合 (値 = 0) と失敗した場合 (値 = 1) の両方で出力されます。このメトリクスを使用して、CloudWatch がサポートする集計期間中 (5 分、あるいは 1 時間など) に登録されたモノの数を追跡できます。 このメトリクスに使用できる統計は次のとおりです。 • Sum を使用して、失敗した呼び出しの数を報告します。 • SampleCount を使用して、成功した呼び出しと失敗した呼び出しの合計数を報告します。

メトリクス	説明
RegisterThingFailed	RegisterThing MQTT API の呼び出しで発生した失敗の数。 このメトリクスは、成功した場合 (値 = 0) と失敗した場合 (値 = 1) の両方で出力されます。このメトリクスを使用して、CloudWatch がサポートする集計期間中 (5 分、あるいは 1 時間など) に登録されたモノの数を追跡できます。登録されたモノの総数については、ApproximateNumberOfThingsRegistered メトリクスを参照してください。 このメトリクスに使用できる統計は次のとおりです。 • Sum を使用して、失敗した呼び出しの数を報告します。 • SampleCount を使用して、成功した呼び出しと失敗した呼び出しの合計数を報告します。 ディメンション: TemplateName

ジャストインタイムのプロビジョニングメトリクス

メトリクス	説明
ProvisionThing.ClientError	クライアントエラーが原因でデバイスがプロビジョニングに失敗した回数。例えば、テンプレートで指定されたポリシーが存在しませんでした。
ProvisionThing.ServerError	サーバーエラーが原因でデバイスがプロビジョニングに失敗した回数。お客様は待機してからデバイスのプロビジョニングを再試行することができるとともに、問題が解決しない場合は AWS IoT に問い合わせることができます。

メトリクス	説明
`ProvisionThing.Success`	デバイスが正常にプロビジョニングされた回数。

LoRaWAN メトリクス

次の表は、 のメトリクスを示しています AWS IoT Core for LoRaWAN。詳細については、「AWS IoT Core for LoRaWAN メトリクス」を参照してください。

AWS IoT Core for LoRaWAN メトリクス

メトリクス	説明
アクティブなデバイス/ゲートウェイ	アカウント内のアクティブな LoRaWAN デバイスとゲートウェイの数。
アップリンクメッセージ数	AWS アカウント内のすべてのアクティブなゲートウェイとデバイスで、指定された期間内に送信されるアップリンクメッセージの数。アップリンクメッセージは、デバイスから に送信されるメッセージです AWS IoT Core for LoRaWAN。
ダウンリンクメッセージ数	AWS アカウント内のすべてのアクティブなゲートウェイとデバイスで、指定された期間内に送信されるダウンリンクメッセージの数。ダウンリンクメッセージは、 からデバイスに送信されるメッセージ AWS IoT Core for LoRaWAN です。
メッセージロス率	デバイスを追加して に接続すると AWS IoT Core for LoRaWAN、デバイスはアップリンクメッセージを開始して、クラウドとのメッセージの交換を開始できます。このメトリクスを使用して、アップリンクメッセージのロス率を追跡できます。
結合メトリクス	デバイスとゲートウェイを追加したら、結合手順を実行して、デバイスがアップリンクデータを送信して と通信できるようにします AWS IoT Core for LoRaWAN。このメトリクスを使用して、 AWS アカ

メトリクス	説明
	ウント内のすべてのアクティブなデバイスの結合メトリクスに関する情報を取得できます。
平均受信信号強度インジケータ (RSSI)	このメトリクスを使用して、指定された時間内の平均 RSSI (受信信号強度インジケータ) をモニタリングできます。RSSI は、信号が良好なワイヤレス接続に十分な強度があるかどうかを示す測定です。この値は負であり、強力な接続にはゼロに近い必要があります。
信号対雑音比 (SNR)	このメトリクスを使用して、指定された時間内の平均 SNR (信号対雑音比) をモニタリングできます。SNR は、受信した信号が良好なワイヤレス接続の雑音レベルと比較して十分に強いかどうかを示す測定値です。SNR 値は正であり、信号電力が雑音電力よりも強いことを示すには、0 より大きい必要があります。
ゲートウェイの可用性	このメトリクスを使用して、指定された期間内にこのゲートウェイの可用性に関する情報を取得できます。このメトリクスは、指定された期間におけるこのゲートウェイのウェブソケット接続時間を表示します。

ジャストインタイムのプロビジョニングメトリクス

メトリクス	説明
`ProvisionThing.ClientError`	クライアントエラーが原因でデバイスがプロビジョニングに失敗した回数。例えば、テンプレートで指定されたポリシーが存在しませんでした。
`ProvisionThing.ServerError`	サーバーエラーが原因でデバイスがプロビジョニングに失敗した回数。お客様は待機してからデバイスのプロビジョニングを再試行することができるとと

メトリクス	説明
	もに、問題が解決しない場合は AWS IoT に問い合わせることができます。
ProvisionThing.Success	デバイスが正常にプロビジョニングされた回数。

フリートのインデックス作成メトリクス

AWS IoT フリートインデックス作成メトリクス

メトリクス	説明
NamedShadowCountForDynamicGroupQueryLimitExceeded	動的なモノのグループのデータソース固有ではないクエリ条件については、モノごとに最大 25 の名前付きシャドウが処理されます。モノに対してこの制限に違反すると、NamedShadowCountForDynamicGroupQueryLimitExceeded イベントタイプが出力されます。

メトリクスのディメンション

メトリクスは名前空間を使用し、以下のディメンションのメトリクスを提供します

ディメンション	説明
ActionType	リクエストによってトリガーされたルールで指定されたアクションのタイプ。
BehaviorName	モニタリングされている Device Defender Detect セキュリティプロファイルの動作の名前。
ClaimCertificateId	デバイスのプロビジョニングに使用されるクレームの certificateId 。
CheckName	結果がモニタリングされている Device Defender 監査チェックの名前。

ディメンション	説明
JobId	進行状況またはメッセージ接続の成功/失敗が監視されているジョブの ID。
Protocol	リクエストを行うために使用されるプロトコル。有効な値: MQTT または HTTP
RuleName	リクエストによってトリガーされたルールの名前。
ScheduledAuditName	チェックの結果がモニタリングされている Device Defender のスケジュールされた監査の名前。オンデマンドで実行された監査の結果がレポートされた場合、この値は OnDemand になります。
SecurityProfileName	動作がモニタリングされている Device Defender Detect セキュリティプロファイルの名前。
TemplateName	プロビジョニングテンプレートの名前。
SourceArn	検出用のセキュリティプロファイルまたは監査用のアカウント ARN を指します。
RoleArn	Device Defender が引き受けようとしたロールを指します。
TopicArn	Device Defender が発行を試みた SNS トピックを指します。

ディメンション	説明
Error	SNS トピックに発行しようとしたときに受信したエラーの簡単な説明を提供します。可能な値は以下のとおりです。 • 「KMSKeyNotFound」: トピックに KMS キーが存在しないことを示します。 • 「InvalidTopicName」: SNS トピックが有効でないことを示します。 • 「KMSAccessDenied」: ロールにトピックの KMS キーに対するアクセス許可がないことを示します。 • 「AuthorizationError」: 指定されたロールが Device Defender に SNS トピックへの発行を許可しないことを示します。 • 「SNSTopicNotFound」: 指定された SNS トピックが存在しないことを示します。 • 「FailureToAssumeRole」: 指定されたロールが Device Defender にロールの引き受けを許可しないことを示します。 • 「CrossRegionSNSTopic」: SNS トピックが別のリージョンに存在することを示します。

CloudWatch Logs AWS IoT を使用したモニタリング

AWS IoT ログ記録を有効にすると、 は、メッセージブローカーとルールエンジンを介してデバイスから送信される各メッセージに関する進行状況イベント AWS IoT を送信します。CloudWatch コンソールで、CloudWatch のログは AWSIotLogsという名前のロググループに表示されます。

CloudWatch Logs の詳細については、CloudWatch Logs を参照してください。サポートされている AWS IoT CloudWatch Logs の詳細については、CloudWatch Logs AWS IoT ログエントリ「」を参照してください。

CloudWatch コンソールでの AWS IoT ログの表示

> **Note**
>
> AWSIotLogsV2 ロググループは、次の状態になるまで CloudWatch コンソールに表示されません。
>
> - ログインを有効にしました AWS IoT。ログインを有効にする方法の詳細については AWS IoT、「」を参照してください。 AWS IoT ログ記録の設定
> - 一部のログエントリは AWS IoT オペレーションによって書き込まれています。

CloudWatch コンソールで AWS IoT ログを表示するには

1. https://console.aws.amazon.com/cloudwatch/ を参照します。ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
2. [フィルター] テキストボックスで、「**AWSIotLogsV2**」と入力して、Enter キーを押します。
3. [AWSIotLogsV2] ロググループをダブルクリックします。
4. [Search All] (すべて検索) を選択します。アカウント用に生成された AWS IoT ログの完全なリストが表示されます。
5. 個々のストリームを表示するには、展開アイコンを選択します。

また、[フィルタイベント] テキストボックスにクエリを入力することもできます。興味深いクエリの例がいくつかあります。

- `{ $.logLevel = "INFO" }`

  ログレベルが INFO のすべてのログを検索します。
- `{ $.status = "Success" }`

  Success のステータスを持つすべてのログを検索します。
- `{ $.status = "Success" && $.eventType = "GetThingShadow" }`

  ステータスが Success で、イベントタイプが GetThingShadow であるすべてのログを検索します。

フィルター式の作成の詳細については、CloudWatch Logs クエリを参照してください。

CloudWatch Logs AWS IoT ログエントリ

の各コンポーネントは、独自のログエントリ AWS IoT を生成します。各ログエントリには、ログエントリが生成される原因となった操作を示す eventType が含まれています。このセクションでは、次の AWS IoT コンポーネントによって生成されるログエントリについて説明します。

トピック

- メッセージブローカーのログエントリ
- サーバー証明書 OCSP ログエントリ
- Device Shadow のログエントリ
- ルールエンジンのログエントリ
- ジョブのログエントリ
- デバイスプロビジョニングのログエントリ
- モノの動的グループのログエントリ
- フリートのインデックス作成ログエントリ
- CloudWatch Logs の共通属性

メッセージブローカーのログエントリ

AWS IoT メッセージブローカーは、次のイベントのログエントリを生成します。

トピック

- Connect ログエントリ
- Disconnect ログエントリ
- DeleteConnection ログエントリ
- GetRetainedMessage ログエントリ
- ListretainedMessage ログエントリ
- Publish-In ログエントリ
- Publish-Out ログエントリ
- キューに保存されたログエントリ
- Subscribe ログエントリ
- Unsubscribe ログエントリ

Connect ログエントリ

AWS IoT メッセージブローカーは、MQTT クライアントが接続Connectするときに、 が eventType のログエントリを生成します。

Connect ログエントリの例

```
{
    "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
}
```

に加えてCloudWatch Logs の共通属性、Connectログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

Disconnect ログエントリ

AWS IoT メッセージブローカーは、MQTT クライアントが切断Disconnectされると、eventType の でログエントリを生成します。

Disconnect ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、Disconnectログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

理由

クライアントが切断する理由。

details

エラーの簡単な説明。

disconnectReason

クライアントが切断する理由。

DeleteConnection ログエントリ

AWS IoT メッセージブローカーは、MQTT クライアント接続が削除されDeleteConnectionると、が eventType のログエントリを生成します。

DeleteConnection ログエントリの例

```
{
    "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
}
```

に加えてCloudWatch Logs の共通属性、DeleteConnectionログエントリには次の属性が含まれます。

clientId

切断されるクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。プリンシパル ID を使用してプリンシパルを識別する方法については、「IAM ユーザーガイド」の「IAM ID と認証情報を比較する」を参照してください。

protocol

リクエストを行うために使用されるプロトコル。有効値は HTTP です。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

GetRetainedMessage ログエントリ

AWS IoT メッセージブローカーは、 GetRetainedMessageが呼び出GetRetainedMessageされると、 eventType の を持つログエントリを生成します。

GetRetainedMessage ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、GetRetainedMessageログエントリには次の属性が含まれます。

lastModifiedDate

保持されたメッセージが保存されたエポック日時をミリ秒単位で表します AWS IoT。

protocol

リクエストを行うために使用されるプロトコル。有効な値: HTTP。

qos

発行要求で使用されるサービス品質 (QoS) レベル。有効な値は 0 または 1 です。

topicName

サブスクライブされたトピックの名前。

ListretainedMessage ログエントリ

AWS IoT メッセージブローカーは、 ListRetainedMessagesが呼び出ListRetainedMessageされると、 eventType の を持つログエントリを生成します。

ListRetainedMessage ログエントリの例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ListRetainedMessage",
    "protocol": "HTTP"
}
```

に加えてCloudWatch Logs の共通属性、ListRetainedMessageログエントリには次の属性が含まれます。

protocol

リクエストを行うために使用されるプロトコル。有効な値: HTTP。

Publish-In ログエントリ

AWS IoT メッセージブローカーは MQTT メッセージを受信すると、 eventTypeの を持つログエントリを生成しますPublish-In。

Publish-In ログエントリの例

```
{
        "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"
    }
```

に加えてCloudWatch Logs の共通属性、Publish-Inログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

保持

メッセージに RETAIN フラグが True の値で設定されている場合に使用される属性。メッセージに RETAIN フラグが設定されていない場合、この属性はログエントリに表示されません。詳細については、「保持された MQTT メッセージ」を参照してください。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

topicName

サブスクライブされたトピックの名前。

Publish-Out ログエントリ

メッセージブローカーは、MQTT メッセージを公開する際に、eventType という Publish-Out のログエントリを生成します。

Publish-Out ログエントリの例

```
{
    "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
}
```

に加えてCloudWatch Logs の共通属性、Publish-Outログエントリには次の属性が含まれます。

clientId

その MQTT トピックでメッセージを受信するサブスクライブクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

topicName

サブスクライブされたトピックの名前。

キューに保存されたログエントリ

永続セッションを持つデバイスが切断されると、MQTT メッセージブローカーはデバイスのメッセージを保存し、eventType が のログエントリ AWS IoT を生成しますQueued。MQTT 永続セッションの詳細については、MQTT 永続的セッション「」を参照してください。

キューに保存されたサーバーエラーログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、Queuedサーバーエラーログエントリには次の属性が含まれます。

clientId

メッセージがキューに保存されたクライアントの ID。

details

`Server Error`

サーバーエラーにより、メッセージの保存が妨げられました。

protocol

リクエストを行うために使用されるプロトコル。値は常に MQTT です。

qos

リクエストのサービス品質 (QoS) レベル。QoS が 0 のメッセージは保存されないため、値は常に 1 になります。

topicName

サブスクライブされたトピックの名前。

キューに保存された成功ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、Queued成功ログエントリには次の属性が含まれます。

clientId

メッセージがキューに保存されたクライアントの ID。

protocol

リクエストを行うために使用されるプロトコル。値は常に MQTT です。

qos

リクエストのサービス品質 (QoS) レベル。QoS が 0 のメッセージは保存されないため、値は常に 1 になります。

topicName

サブスクライブされたトピックの名前。

キューに保存されているスロットリングされたログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、Queuedスロットリングされたログエントリには次の属性が含まれます。

clientId

メッセージがキューに保存されたクライアントの ID。

details

`Throttled while queueing offline message`

クライアントが `Queued messages per second per account` 制限を超えたため、メッセージは保存されませんでした。

protocol

リクエストを行うために使用されるプロトコル。値は常に `MQTT` です。

qos

リクエストのサービス品質 (QoS) レベル。QoS が 0 のメッセージは保存されないため、値は常に 1 になります。

topicName

サブスクライブされたトピックの名前。

Subscribe ログエントリ

AWS IoT メッセージブローカーは、MQTT クライアントがトピックをサブスクライブSubscribeするときに、 eventTypeの を持つログエントリを生成します。

MQTT 3 Subscribe ログエントリの例

```
{
    "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
}
```

に加えてCloudWatch Logs の共通属性、Subscribeログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

protocol

リクエストを行うために使用されるプロトコル。値は常に MQTT です。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

topicName

サブスクライブされたトピックの名前。

MQTT 5 Subscribe ログエントリの例

```
{
 "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 Subscribe オペレーションでは、CloudWatch Logs の共通属性 および MQTT 3 Subscribe ログエントリ属性に加えて、MQTT 5 Subscribe ログエントリに次の属性が含まれます。

サブスクリプション

Subscribe リクエストでリクエストされたトピックと個々の MQTT 5 理由コード間のマッピングのリスト。詳細については、「MQTT 理由コード」を参照してください。

Unsubscribe ログエントリ

AWS IoT メッセージブローカーは、MQTT クライアントが MQTT トピックへのサブスクライブを解除Unsubscribeすると、 eventTypeの を持つログエントリを生成します。

MQTT unsubscribe ログエントリの例

```
{
    "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
}
```

に加えてCloudWatch Logs の共通属性、Unsubscribeログエントリには次の属性が含まれます。

protocol

リクエストを行うために使用されるプロトコル。値は常に MQTT です。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

sourceIp

リクエストが発生した IP アドレス。

sourcePort

リクエストが発生したポート。

サーバー証明書 OCSP ログエントリ

AWS IoT Core は、次のイベントのログエントリを生成します。

トピック

- RetrieveOCSPStapleData ログエントリ
- プライベートエンドポイントの RetrieveOCSPStapleData ログエントリ

RetrieveOCSPStapleData ログエントリ

AWS IoT Core サーバーが OCSP ステープルデータを取得するeventTypeRetrieveOCSPStapleDataと、 は のログエントリを生成します。

RetrieveOCSPStapleData ログエントリの例

以下は Success のログエントリの例です。


```
{
 "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"
 }
```

```
}
```

以下は Failure のログエントリの例です。

```
{
 "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 オペレーションの場合、 CloudWatch Logs の共通属性 に加えて、ログエントリには次の属性が含まれます。

理由

オペレーションが失敗する理由。

domainConfigName

ドメイン構成の名前。

connectionDetails

接続の詳細の簡潔な説明。

- httpStatusCode

  サーバーに対するクライアントのリクエストに応じて OCSP レスポンダーによって返される HTTP ステータスコード。

- ocspResponderUri

  サーバー証明書から AWS IoT Core 取得する OCSP レスポンダー URI。

- sourceIp

  AWS IoT Core サーバーのソース IP アドレス。

- targetIp

  OCSP レスポンダーのターゲット IP アドレス。

ocspRequestDetails

OCSP リクエストの詳細。

- requesterName

  OCSP レスポンダーにリクエストを送信する AWS IoT Core サーバーの識別子。

- requestCertId

  リクエストの証明書 ID。これは、OCSP レスポンスがリクエストされている証明書の ID です。

ocspResponseDetails

OCSP レスポンスの詳細。

- responseCertId

  OCSP レスポンスの証明書 ID。

- ocspResponseStatus

  OCSP レスポンスのステータス。

- certStatus

  証明書のステータス。

- 署名

  信頼されたエンティティによるレスポンスに適用される署名。

- thisUpdateTime

  表示されるステータスが正しいことがわかっている時刻。

- nextUpdateTime

証明書のステータスに関する新しい情報が利用可能になる時刻またはそれ以前。

- producedAtTime

  OCSP レスポンダーがこのレスポンスに署名した時刻。

- stapledDataPayloadSize

  ステープルデータのペイロードサイズ。

プライベートエンドポイントの RetrieveOCSPStapleData ログエントリ

AWS IoT Core サーバーが OCSP ステープルデータを取得するeventTypeRetrieveOCSPStapleDataと、 は のログエントリを生成します。

プライベートエンドポイントの RetrieveOCSPStapleData ログエントリの例

以下は Success のログエントリの例です。

```
{
 "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"
 }
}
```

以下は Failure のログエントリの例です。

```
{
 "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 オペレーションでは、CloudWatch Logs の共通属性および RetrieveOCSPStapleData ログエントリの属性に加えて、プライベートエンドポイントのログエントリには次の属性が含まれます。

lambdaDetails

Lambda 関数の詳細。

- LambdaArn

  Lambda 関数の ARN。

- sourceArn

  ドメイン設定の ARN。

authorizedResponderArn

ドメイン設定で設定されている場合、オーソライザーレスポンダーの ARN。

Device Shadow のログエントリ

AWS IoT Device Shadow サービスは、次のイベントのログエントリを生成します。

トピック

- DeleteThingShadow ログエントリ
- GetThingShadow ログエントリ
- UpdateThingShadow ログエントリ

DeleteThingShadow ログエントリ

Device Shadow サービスは、デバイスのシャドウ削除リクエストを受信すると、`eventType` という `DeleteThingShadow` のログエントリを生成します。

DeleteThingShadow ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、DeleteThingShadowログエントリには次の属性が含まれます。

deviceShadowName

更新する Shadow の名前。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

公開されたリクエストのトピックの名前。

GetThingShadow ログエントリ

デバイスシャドウサービスは、シャドウの取得リクエストを受信すると、eventType という GetThingShadow のログエントリを生成します。

GetThingShadow ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、GetThingShadowログエントリには次の属性が含まれます。

deviceShadowName

リクエストした Shadow の名前。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

公開されたリクエストのトピックの名前。

UpdateThingShadow ログエントリ

Device Shadow サービスは、デバイスのシャドウ更新リクエストを受信すると、eventType という UpdateThingShadow のログエントリを生成します。

UpdateThingShadow ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、UpdateThingShadowログエントリには次の属性が含まれます。

deviceShadowName

更新する Shadow の名前。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

公開されたリクエストのトピックの名前。

ルールエンジンのログエントリ

AWS IoT ルールエンジンは、次のイベントのログを生成します。

トピック

- FunctionExecution ログエントリ
- RuleExecution ログエントリ
- RuleMatch ログエントリ
- RuleExecutionThrottled ログエントリ
- RuleNotFound ログエントリ
- StartingRuleExecution ログエントリ

FunctionExecution ログエントリ

ルールエンジンは、ルールの SQL クエリが外部関数を呼び出すと、`eventType` という `FunctionExecution` のログエントリを生成します。外部関数は、ルールのアクションが AWS IoT または別のウェブサービスに対して HTTP リクエストを行うときに呼び出されます (たとえば、`get_thing_shadow` または を呼び出す)`machinelearning_predict`。

FunctionExecution ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、`FunctionExecution`ログエントリには次の属性が含まれます。

clientId

N/A ログの FunctionExecution。

principalId

リクエストを実行するプリンシパルの ID。

リソース

ルールのアクションによって使用されるリソースの集合。

ruleName

一致ルールの名前。

topicName

サブスクライブされたトピックの名前。

RuleExecution ログエントリ

AWS IoT ルールエンジンがルールのアクションをトリガーすると、RuleExecutionログエントリが生成されます。

RuleExecution ログエントリの例

```
{
    "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"
}
```

バッチログエントリを使用した RuleExecution の例


```
{
    "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
    }
```

```
}
```

CloudWatch Logs の共通属性 に加えて、RuleExecution ログ項目には次の属性が含まれています。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

リソース

ルールのアクションによって使用されるリソースの集合。

ruleAction

トリガーされるアクションの名前。

ruleName

一致ルールの名前。

topicName

サブスクライブされたトピックの名前。

RuleMatch ログエントリ

AWS IoT ルールエンジンは、メッセージブローカーがルールに一致するメッセージを受信するRuleMatchと、 eventType の を持つログエントリを生成します。

RuleMatch ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、RuleMatchログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

ruleName

一致ルールの名前。

topicName

サブスクライブされたトピックの名前。

RuleExecutionThrottled ログエントリ

実行がスロットリングされると、 AWS IoT ルールエンジンは eventTypeの を使用してログエントリを生成しますRuleExecutionThrottled。

RuleExecutionThrottled ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、RuleExecutionThrottledログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

details

エラーの簡単な説明。

principalId

リクエストを実行するプリンシパルの ID。

理由

文字列「RuleExecutionThrottled」。

ruleName

トリガーされるルールの名前。

topicName

発行されたトピックの名前。

RuleNotFound ログエントリ

AWS IoT ルールエンジンは、指定された名前のルールを見つけることができない場合、eventTypeの を持つログエントリを生成しますRuleNotFound。

RuleNotfound ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、RuleNotFoundログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

details

エラーの簡単な説明。

principalId

リクエストを実行するプリンシパルの ID。

理由

文字列 "RuleNotFound"。

ruleName

見つからなかったルールの名前。

topicName

発行されたトピックの名前。

StartingRuleExecution ログエントリ

AWS IoT ルールエンジンがルールのアクションのトリガーを開始すると、 eventType の を持つログエントリが生成されますStartingRuleExecution。

StartingRuleExecution ログエントリの例

```
{
    "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"
}
```


に加えてCloudWatch Logs の共通属性、`rule-`ログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

principalId

リクエストを実行するプリンシパルの ID。

ruleAction

トリガーされるアクションの名前。

ruleName

一致ルールの名前。

topicName

サブスクライブされたトピックの名前。

ジョブのログエントリ

AWS IoT ジョブサービスは、次のイベントのログエントリを生成します。デバイスからの MQTT または HTTP リクエストが受信されたときに、ログエントリが生成されます。

トピック

- DescribeJobExecution ログエントリ
- GetPendingJobExecution ログエントリ
- ReportFinalJobExecutionCount ログエントリ

- StartNextPendingJobExecution ログエントリ
- UpdateJobExecution ログエントリ

DescribeJobExecution ログエントリ

AWS IoT Jobs サービスは、サービスDescribeJobExecutionがジョブ実行を記述するリクエストを受信すると、 eventTypeの を持つログエントリを生成します。

DescribeJobExecution ログエントリの例

```
{
    "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."
}
```

に加えてCloudWatch Logs の共通属性、GetJobExecutionログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

clientToken

リクエストのべき等のための一意の識別子 (大文字と小文字を区別)。詳細については、べき等を確実にする方法のページを参照してください。

details

Jobs サービスからの追加情報。

jobId

ジョブ実行のジョブ ID。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

リクエストを行うために使用されるトピック。

GetPendingJobExecution ログエントリ

AWS IoT Jobs サービスは、サービスGetPendingJobExecutionがジョブ実行リクエストを受信すると、 eventTypeの を持つログエントリを生成します。

GetPendingJobExecution ログエントリの例

```
{
    "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."
}
```

に加えてCloudWatch Logs の共通属性、GetPendingJobExecutionログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

clientToken

リクエストのべき等のための一意の識別子 (大文字と小文字を区別)。詳細については、べき等を確実にする方法のページを参照してください。

details

Jobs サービスからの追加情報。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

サブスクライブされたトピックの名前。

ReportFinalJobExecutionCount ログエントリ

AWS IoT Jobs サービスは、ReportFinalJobExecutionCountジョブが完了すると entryType のログエントリを生成します。

ReportFinalJobExecutionCount ログエントリの例

```
{
    "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"
}
```

に加えてCloudWatch Logs の共通属性、ReportFinalJobExecutionCountログエントリには次の属性が含まれます。

details

Jobs サービスからの追加情報。

jobId

ジョブ実行のジョブ ID。

StartNextPendingJobExecution ログエントリ

次の保留中のジョブ実行を開始するリクエストを受信すると、 AWS IoT ジョブサービスは eventTypeの を使用してログエントリを生成しますStartNextPendingJobExecution。

StartNextPendingJobExecution ログエントリの例

```
{
    "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."
}
```

に加えてCloudWatch Logs の共通属性、StartNextPendingJobExecutionログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

clientToken

リクエストのべき等のための一意の識別子 (大文字と小文字を区別)。詳細については、べき等を確実にする方法のページを参照してください。

details

Jobs サービスからの追加情報。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

リクエストを行うために使用されるトピック。

UpdateJobExecution ログエントリ

AWS IoT Jobs サービスは、サービスUpdateJobExecutionがジョブ実行の更新リクエストを受信すると、 eventType のログエントリを生成します。

UpdateJobExecution ログエントリの例

```
{
    "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."
}
```

に加えてCloudWatch Logs の共通属性、UpdateJobExecutionログエントリには次の属性が含まれます。

clientId

リクエストを実行するクライアントの ID。

clientToken

リクエストのべき等のための一意の識別子 (大文字と小文字を区別)。詳細については、べき等を確実にする方法のページを参照してください。

details

Jobs サービスからの追加情報。

jobId

ジョブ実行のジョブ ID。

protocol

リクエストを行うために使用されるプロトコル。有効な値は MQTT または HTTP です。

topicName

リクエストを行うために使用されるトピック。

versionNumber

ジョブ実行のバージョン。

デバイスプロビジョニングのログエントリ

AWS IoT Device Provisioning サービスは、次のイベントのログを生成します。

トピック

- GetDeviceCredentials ログエントリ
- ProvisionDevice ログエントリ

GetDeviceCredentials ログエントリ

AWS IoT Device Provisioning サービスは、クライアントが を呼び出すeventTypeGetDeviceCredentialときに のログエントリを生成しますGetDeviceCredential。

GetDeviceCredentials ログエントリの例

```
{
  "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."
}
```

に加えてCloudWatch Logs の共通属性、GetDeviceCredentialsログエントリには次の属性が含まれます。

details

エラーの簡単な説明。

deviceCertificateId

デバイス証明書の ID。

ProvisionDevice ログエントリ

AWS IoT Device Provisioning サービスは、クライアントが を呼び出すeventTypeProvisionDeviceときに、 のログエントリを生成しますProvisionDevice。

ProvisionDevice ログエントリの例

```
{
  "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."
 }
```

に加えてCloudWatch Logs の共通属性、ProvisionDeviceログエントリには次の属性が含まれます。

details

エラーの簡単な説明。

deviceCertificateId

デバイス証明書の ID。

provisioningTemplateName

プロビジョニングテンプレートの名前。

モノの動的グループのログエントリ

AWS IoT モノの動的グループは、次のイベントのログを生成します。

トピック

- AddThingToDynamicThingGroupsFailed ログエントリ

AddThingToDynamicThingGroupsFailed ログエントリ

AWS IoT が指定された動的グループにモノを追加できなかった場合、 eventTypeの を使用してログエントリを生成しますAddThingToDynamicThingGroupsFailed。この場合、モノがモノの動的グループの条件を満たしていたにもかかわらず、動的グループに追加できなかったか、動的グループから削除されたことを意味します。これは、次の原因で発生した可能性があります。

- モノが既に最大数のグループに属している。
- --override-dynamic-groups オプションを使用してモノがモノの静的グループに追加された。これを可能にするために、モノの動的なグループから削除された。

詳細については、「モノの動的グループの制限と競合」を参照してください。

AddThingToDynamicThingGroupsFailed ログエントリの例

この例は、AddThingToDynamicThingGroupsFailed エラーのログエントリを示しています。この例では、TestThing は「dynamicThingGroupNames」にリストされているモノの動的グループに追加される条件を満たしていましたが、reason で説明されているとおり、これらの動的グループに追加できませんでした。

```
{
 "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."
}
```

に加えてCloudWatch Logs の共通属性、AddThingToDynamicThingGroupsFailedログエントリには次の属性が含まれます。

dynamicThingGroupNames

モノを追加できなかったモノの動的グループの配列。

理由

モノをモノの動的グループに追加できなかった理由。

thingName

モノの動的グループに追加できなかったモノの名前。

フリートのインデックス作成ログエントリ

AWS IoT フリートインデックス作成は、次のイベントのログエントリを生成します。

トピック

- NamedShadowCountForDynamicGroupQueryLimitExceeded ログエントリ

NamedShadowCountForDynamicGroupQueryLimitExceeded ログエントリ

動的グループ内のデータソース固有ではないクエリ条件では、モノごとに最大 25 の名前付きシャドウが処理されます。モノに対してこの制限に違反すると、NamedShadowCountForDynamicGroupQueryLimitExceeded イベントタイプが出力されます。

NamedShadowCountForDynamicGroupQueryLimitExceeded ログエントリの例

この例は、NamedShadowCountForDynamicGroupQueryLimitExceeded エラーのログエントリを示しています。この例では、reason フィールドに示されているように、すべての値に基づいた DynamicGroup の結果は不正確である可能性があります。

```
{
"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."
}
```

CloudWatch Logs の共通属性

すべての CloudWatch Logs ログエントリには、以下の属性が含まれます。

accountId

AWS アカウント ID。

eventType

ログが生成されたイベントタイプ。イベントタイプの値は、ログエントリが生成される原因となったイベントによって異なります。各ログエントリの説明には、そのログエントリの `eventType` の値が含まれます。

logLevel

使用されているログレベル。詳細については、「the section called “ログレベル”」を参照してください。

status

リクエストのステータス。

timestamp

AWS IoT メッセージブローカーに接続した時刻のクライアントの、人間が読み取り可能な UNIX タイムスタンプ。

traceId

特定のリクエストのすべてのログを関連付けるために使用できる、ランダムに生成された識別子。

Amazon CloudWatch にデバイス側のログをアップロード

過去のデバイス側のログを Amazon CloudWatch にアップロードして、現場でのデバイスのアクティビティを監視および分析できます。デバイス側のログには、システム、アプリケーション、およびデバイスログファイルを含めることができます。このプロセスでは、CloudWatch Logs ルールアクションパラメータを使用して、デバイス側のログを顧客定義の[ロググループ](#)に公開します。

仕組み

このプロセスは、 AWS IoT デバイスがフォーマットされたログファイルを含む MQTT メッセージを AWS IoT トピックに送信したときに開始されます。 AWS IoT ルールはメッセージトピックをモニタリングし、定義した CloudWatch Logs グループにログファイルを送信します。その後、情報を確認して分析できます。

トピック

- MQTT トピック
- ルールアクション

MQTT トピック

ログの発行に使用する MQTT トピック名前空間を選択します。一般的なトピックスペース、`$aws/rules/things/thing_name/logs` にはこの形式を使用し、エラートピック、`$aws/rules/things/thing_name/logs/errors` にはこの形式を使用することをお勧めします。ログとエラートピックの命名構造は推奨されますが、必須ではありません。詳細については、「AWS IoT Coreのための MQTT トピックの設計」を参照してください。

推奨される共通トピックスペースを使用すると、 AWS IoT 基本的な取り込み予約トピックを利用できます。 AWS IoT 基本的な取り込みは、 AWS IoT ルールアクションでサポートされている AWS サービスにデバイスデータを安全に送信します。これにより、パブリッシュ/サブスクライブのメッセージブローカーが取り込みパスから除外され、コスト効率が向上します。詳細については、「基本的な取り込みによるメッセージングコストの削減」を参照してください。

batchMode を使用してログファイルをアップロードする場合、メッセージは UNIX タイムスタンプとメッセージを含む特定の形式に従う必要があります。詳細については、「CloudWatch Logs rule action」(CloudWatch Logs ルールアクション) 内の「MQTT message format requirements for batchMode」(BatchMode の MQTT メッセージフォーマット要件) トピックを参照してください。

ルールアクション

がクライアントデバイスから MQTT メッセージ AWS IoT を受信すると、 AWS IoT ルールはカスタマー定義のトピックをモニタリングし、定義した CloudWatch ロググループにコンテンツを公開します。このプロセスでは、CloudWatch Logs ルールアクションを使用して MQTT のログファイルのバッチを監視します。詳細については、CloudWatch Logs AWS IoT ルールアクション」を参照してください。

バッチモード

batchMode は、 AWS IoT CloudWatch Logs ルールアクション内のブールパラメータです。このパラメータはオプションで、デフォルトはオフ (false) です。デバイス側のログファイルをバッチでアップロードするには、 AWS IoT ルールの作成時にこのパラメータを (true) をオンにする必要があります。詳細については、AWS IoT ルールアクションセクションの「CloudWatch Logs」を参照してください。

AWS IoT ルールを使用したデバイス側のログのアップロード

AWS IoT ルールエンジンを使用して、既存のデバイス側のログファイル (システム、アプリケーション、デバイスクライアントログ) から Amazon CloudWatch にログレコードをアップロードできます。デバイス側のログが MQTT トピックに公開されると、CloudWatch Logs ログルールアクションはメッセージを CloudWatch Logs に転送します。このプロセスでは、ルールアクション batchModeパラメータをオン (true に設定) にして、デバイスログをバッチでアップロードする方法の概要を説明します。

CloudWatch へのデバイス側のログのアップロードを開始するには、次の前提条件を満たしてください。

前提条件

開始する前に、以下を実行します。

- モノ AWS IoT Core として に登録されているターゲット IoT デバイスを少なくとも 1 つ作成します AWS IoT 。詳細については、「モノのオブジェクトを作成する」を参照してください。
- 取り込みやエラーが起きる MQTT トピックスペースを決定します。MQTT トピックと推奨される命名規則の詳細については、「Upload device-side logs to Amazon CloudWatch」(Amazon CloudWatch へのデバイス側のログのアップロード) の「MQTT トピック MQTT topics」(MQTT トピック) セクションを参照してください。

前提条件の詳細については、「Upload device-side logs to CloudWatch」(CloudWatch へのデバイス側のログのアップロード) を参照してください。

CloudWatch Logs ロググループの作成

CloudWatch ロググループを作成するには、次のステップを完了してください。 AWS マネジメントコンソール または AWS Command Line Interface () でステップを実行するかどうかに応じて、適切なタブを選択しますAWS CLI。

AWS マネジメントコンソール

を使用して CloudWatch ロググループを作成するには AWS マネジメントコンソール

1. を開き AWS マネジメントコンソール 、CloudWatch に移動します。
2. ナビゲーションバーで、[Logs] (ログ)、[Log groups] (ロググループ) の順に選択します。
3. [ロググループの作成] を選択します。
4. ロググループ名を更新し、オプションで保存設定フィールドを更新します。
5. [作成] を選択します。

AWS CLI

を使用して CloudWatch ロググループを作成するには AWS CLI

1. ロググループを作成するには、以下のコマンドを実行します。詳細については、v2 コマンドリファレンス create-log-group AWS CLI の「」を参照してください。

   例 (uploadLogsGroup) のロググループ名を任意の名前に置き換えます。

   ```
   aws logs create-log-group --log-group-name uploadLogsGroup
   ```

2. ロググループが正しく作成されたことを確認するには、次のコマンドを実行します。

   ```
   aws logs describe-log-groups --log-group-name-prefix uploadLogsGroup
   ```

   サンプル出力:

   ```
   {
       "logGroups": [
           {
               "logGroupName": "uploadLogsGroup",
               "creationTime": 1674521804657,
               "metricFilterCount": 0,
               "arn": "arn:aws:logs:us-east-1:111122223333:log-
   group:uploadLogsGroup:*",
               "storedBytes": 0
           }
       ]
   }
   ```

トピックルールの作成

AWS IoT ルールを作成するには、次の手順を実行します。 AWS マネジメントコンソール または AWS Command Line Interface () でステップを実行するかどうかに応じて、適切なタブを選択します AWS CLI。

AWS マネジメントコンソール

を使用してトピックルールを作成するには AWS マネジメントコンソール

1. ルールハブを開きます。
   a. [AWS マネジメントコンソール] を開き、AWS IoT に移動します。
   b. ナビゲーションバーで、[Message routing] (メッセージルーティング) を選択し、次に [Rules] (ルール) を選択します。
   c. [ルールを作成] を選択します。
2. ルールプロパティを入力します。
   a. 英数字のルール名を入力します。
   b. (オプション) ルールの説明とタグを入力します。
   c. [次へ] を選択します。
3. SQL ステートメントを入力します。
   a. 取り込み用に定義した MQTT トピックを使用して SQL ステートメントを入力します。

      例: `SELECT * FROM '$aws/rules/things/thing_name/logs'`
   b. [次へ] を選択します。
4. ルールアクションを入力します。
   a. [Action 1] (アクション 1) メニューで、[CloudWatch logs] を選択します。
   b. [Log group name] (ロググループ名) を選択し、選択したロググループを選択します。
   c. [Use batch mode] (バッチモードを使用) を選択します。
   d. ルールの IAM ロールを指定します。

      ルールの IAM ロールがある場合は、次の操作を行います。

      1. [IAM role] (IAM ロール) メニューで、IAM ロールを選択します。

ルールの IAM ロールがない場合は、次の操作を行います。

1. [Create new role (新しいロールの選択)] を選択します。
2. [Role name] (ロール名) に、一意の名前を入力して [Create] (作成) を選択します。
3. [IAM role] (IAM ロール) フィールドで、IAM ロール名が正しいことを確認します。

e. [次へ] を選択します。

5. テンプレート設定を確認します。

a. ジョブテンプレートの設定を確認して、設定が正しいことを確認します。

b. 終了したら、[作成] を選択します。

AWS CLI

を使用して IAM ロールとトピックルールを作成するには AWS CLI

1. AWS IoT ルールに権限を付与する IAM ロールを作成します。

a. IAM ポリシーを作成します。

IAM ポリシーを作成するには、次のコマンドを実行します。policy-name パラメータ値を必ず更新してください。詳細については、v2 コマンドリファレンス[create-policy](#) AWS CLI の「」を参照してください。

> **Note**
>
> Microsoft Windows オペレーティングシステムを使用している場合は、行末マーカー (\) をチェックマーク (`) または別の文字に置き換える必要がある場合があります。

```
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. 出力されたポリシーの ARN をテキストエディターにコピーします。

サンプル出力:

```
{
    "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 ロールと信頼ポリシーを作成します。

IAM ポリシーを作成するには、次のコマンドを実行します。role-name パラメータ値を必ず更新してください。詳細については、v2 コマンドリファレンスcreate-role AWS CLI の「」を参照してください。

```
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 ポリシーをルールにアタッチします。

IAM ポリシーを作成するには、次のコマンドを実行します。role-name および policy-arn パラメータ値を必ず更新してください。詳細については、v2 コマンドリファレンスattach-role-policy AWS CLI の「」を参照してください。

```
aws iam attach-role-policy \
--role-name uploadLogsRole \
--policy-arn arn:aws:iam::111122223333:policy/uploadLogsPolicy
```

e. ロールを確認してください。

IAM ロールが正しく作成されたことを確認するには、次のコマンドを実行します。role-name パラメータ値を必ず更新してください。詳細については、v2 コマンドリファレンスget-role AWS CLI の「」を参照してください。

```
aws iam get-role --role-name uploadLogsRole
```

サンプル出力:

```
{
    "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. で AWS IoT トピックルールを作成します AWS CLI。

   a. AWS IoT トピックルールを作成するには、次のコマンドを実行します。--rule-name、sql ステートメント、descriptionroleARN 、および logGroupNameパラメータ値を必ず更新してください。詳細については、「 v2 コマンドリファレンス AWS CLI 」のcreate-topic-rule」を参照してください。

```
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. ルールが正しく作成されたことを検証するには、次のコマンドを実行します。role-name パラメータ値を必ず更新してください。詳細については、「 v2 コマンドリファレンス AWS CLI 」のget-topic-rule」を参照してください。

```
aws iot get-topic-rule --rule-name uploadLogsRule
```

サンプル出力:

```
{
    "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"
    }
}
```

へのデバイス側のログの送信 AWS IoT

デバイス側のログを に送信するには AWS IoT

1. 履歴ログを送信するには AWS IoT、デバイスと通信して以下を確認します。

   - ログ情報は、この手順の「前提条件」セクションで指定されている正しいトピック名前空間に送信されます。

     例: `$aws/rules/things/thing_name/logs`

   - MQTT メッセージペイロードは正しくフォーマットされています。MQTT トピックと推奨される命名規則の詳細については、 内のAmazon CloudWatch にデバイス側のログをアップロードMQTT トピック「」セクションを参照してください。

2. MQTT メッセージが AWS IoT MQTT クライアント内で受信されていることを確認します。

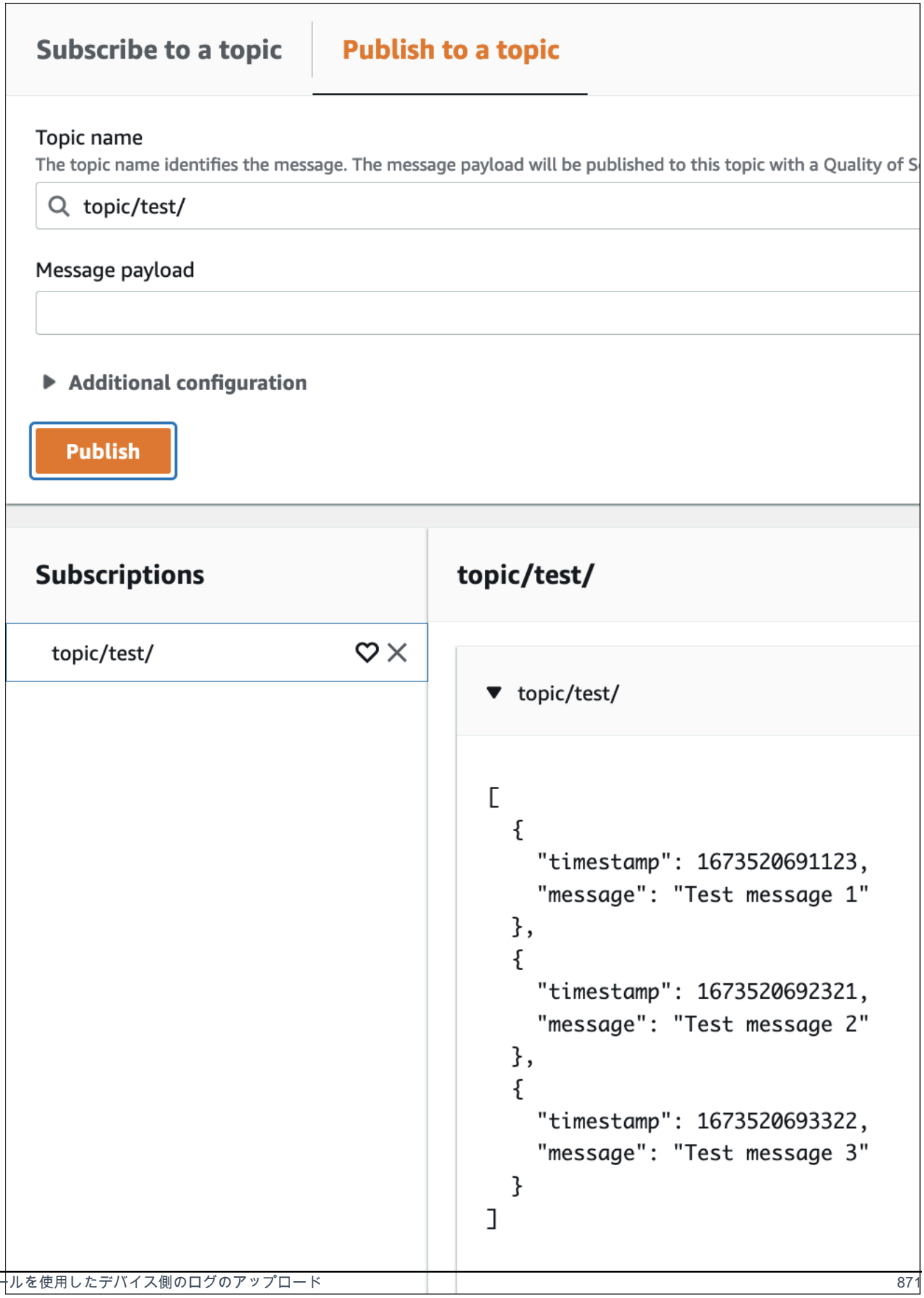
   a. を開き AWS マネジメントコンソール 、 に移動しますAWS IoT。

   b. MQTT テストクライアントを表示するには、ナビゲーションバーでテスト、MQTT テストクライアントを選択します。

   c. トピックをサブスクライブするには、トピックフィルターにトピック名前空間を入力します。

   d. [サブスクライブ] を選択します。

      MQTT メッセージは、次に示すように、サブスクリプションとトピックの表に表示されます。これらのメッセージが表示されるまでに最大 5 分かかります。


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"
}
]

ログデータの表示

CloudWatch Logs でログレコードを確認するには

1. を開き AWS マネジメントコンソール、CloudWatch に移動します。
2. ナビゲーションバーで、Logs、Logs Insights を選択します。
3. ロググループの選択 (複数可) メニューで、 AWS IoT ルールで指定したロググループを選択します。
4. [Logs insights] (ログインサイト) ページで、[Run query] (クエリを実行) を選択します。

を使用した AWS IoT API コールのログ記録 AWS CloudTrail

AWS IoT は、ユーザー AWS CloudTrail、ロール、または のサービスによって実行されたアクションを記録する AWS サービスである と統合されています AWS IoT。CloudTrail は、 AWS IoT コンソールからの呼び出しや API へのコード呼び出しを含む、 のすべての API コールをイベント AWS IoT としてキャプチャします。 AWS IoT APIs 証跡を作成する場合は、イベントを含む Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます AWS IoT。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail によって収集された情報を使用して、リクエストの実行元の IP アドレス AWS IoT、リクエストの実行者、リクエストの実行日時などの詳細を確認できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

AWS IoT CloudTrail の情報

CloudTrail は、アカウントの作成 AWS アカウント 時に で有効になります。アクティビティが発生すると AWS IoT、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

AWS IoTのイベントなど、 AWS アカウントのイベントの継続的な記録に対して、追跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべての AWS リージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。CloudTrail ログで収集されたイベントデータをさらに分析して処理するように、他の AWS サービスを設定できます。詳細については、以下を参照してください。

- 証跡を作成するための概要
- CloudTrail がサポートするサービスと統合
- CloudTrail 用 Amazon SNS 通知の構成
- 複数のリージョンから CloudTrail ログファイルを受け取るおよび複数のアカウントから CloudTrail ログファイルを受け取る

> **Note**
>
> AWS IoT データプレーンアクション (デバイス側) は CloudTrail によってログに記録されません。これらのアクションを監視するために CloudWatch を使用します。

一般的に、変更を行う AWS IoT コントロールプレーンアクションは CloudTrail によってログに記録されます。CreateThing、CreateKeysAndCertificate、UpdateCertificate などの呼び出しは CloudTrail エントリを残しますが、ListThings や ListTopicRules などの呼び出しは残しません。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

AWS IoT アクションは AWS IoT API リファレンスに記載されています。 AWS IoT Wireless アクションは AWS IoT Wireless API リファレンスに記載されています。

AWS IoT ログファイルエントリについて

「トレイル」は、指定した Amazon S3 バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次は、AttachPolicy アクションを示す CloudTrail ログエントリの例です。

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

のルール AWS IoT

ルールにより、デバイスは とやり取りできるようになります AWS のサービス。ルールの分析とアクションの実行は、MQTT トピックストリーミングに基づいて行われます。ルールを使用すると、次のようなタスクをサポートできます。

- デバイスから受け取ったデータの加工またはフィルター処理を行う。
- デバイスから受け取ったデータを Amazon DynamoDB データベースに書き込む。
- Amazon S3 にファイルを保存します。
- Amazon SNS を使用しているすべてのユーザーにプッシュ通知を送信します。
- Amazon SQS キューにデータを発行します。
- Lambda 関数を呼び出してデータを抽出する。
- Amazon Kinesis を使用して、デバイスからの多数のメッセージを処理する。
- Amazon OpenSearch Service にデータを送信します。
- CloudWatch メトリクスを取得します。
- CloudWatch アラームを変更します。
- MQTT メッセージから Amazon SageMaker AI にデータを送信して、機械学習 (ML) モデルに基づいて予測を行います。
- Salesforce の IoT 入力ストリーミングにメッセージを送信します。
- Step Functions ステートマシンのプロセスを開始します。
- メッセージデータを AWS IoT Events 入力に送信します。
- AWS IoT SiteWiseでアセットプロパティにメッセージデータを送信します
- ウェブアプリケーションまたはサービスにメッセージデータを送信します。

ルールには、the section called “デバイス通信プロトコル” がサポートするパブリッシュ/サブスクライブプロトコルを通過する MQTT メッセージを使用できます。また、基本的な取り込み機能を使用して、メッセージングコストを発生させることなく、 AWS のサービス 前述の にデバイスデータを安全に送信することもできます。基本的な取り込み機能では、取り込みパスからパブリッシュ/サブスクライブのメッセージブローカーを除外することによってデータフローが最適化されます。これにより、 のセキュリティとデータ処理機能を維持しながら、コスト効率が向上します AWS IoT。

AWS IoT がこれらのアクションを実行する前に、ユーザーに代わって AWS リソースにアクセスするアクセス許可を付与する必要があります。アクションを実行すると、 AWS のサービス 使用するの標準料金が発生します。

内容

- 必要なアクセスを AWS IoT ルールに付与する
- ロールのアクセス許可の適用
- AWS IoT ルールの作成
- AWS IoT ルールの管理
- AWS IoT ルールアクション
- ルールのトラブルシューティング
- AWS IoT ルールを使用したクロスアカウントリソースへのアクセス
- エラー処理 (エラーアクション)
- 基本的な取り込みによるメッセージングコストの削減
- AWS IoT SQL リファレンス

必要なアクセスを AWS IoT ルールに付与する

IAM ロールを使用して、各ルールがアクセスできる AWS リソースを制御します。ルールを作成する前に、必要な AWS リソースへのアクセスを許可するポリシーを持つ IAM ロールを作成する必要があります。 は、ルールを実装するときにこのロールを AWS IoT 引き受けます。

以下のステップを実行して、必要なアクセスを AWS IoT ルールに付与する IAM ロールと AWS IoT ポリシーを作成します (AWS CLI)。

1. ロールを引き受ける AWS IoT アクセス許可を付与する次の信頼ポリシードキュメントを、 という名前のファイルに保存します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 コマンドを使用し、iot-role-trust.json ファイルを指定して IAM ロールを作成します。

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document
 file://iot-role-trust.json
```

このコマンドの出力は以下のようになります。

```
{
 "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. 次の JSON を my-iot-policy.json という名前のファイルに保存します。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
```

```
    "Action": "dynamodb:*",
    "Resource": "*"
   }
  ]
}
```

この JSON は、DynamoDB への AWS IoT 管理者アクセスを許可するポリシードキュメントの例です。

create-policy コマンドを使用して、ロールを引き受けるときに AWS リソース AWS IoT へのアクセスを許可し、 `my-iot-policy.json` ファイルを渡します。

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-policy.json
```

のポリシー AWS のサービス で へのアクセスを許可する方法の詳細については AWS IoT、「」を参照してくださいAWS IoT ルールの作成。

create-policy コマンドの出力には、ポリシーの ARN が含められます。 ロールにポリシーをアタッチします。

```
{
 "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. ポリシーをロールにアタッチするには、attach-role-policy コマンドを使用します。

```
aws iam attach-role-policy --role-name my-iot-role --policy-arn
 "arn:aws:iam::123456789012:policy/my-iot-policy"
```

ルールエンジンアクセスを取り消す

ルールエンジンへのアクセスを即時に取り消すには、次の手順を実行します。

1. 信頼ポリシーから iot.amazonaws.com を削除する
2. ステップに従って iot ロールセッションを取り消す

ロールのアクセス許可の適用

ルール定義の一部として、ルールのアクションで指定されたリソースにアクセスする権限を付与する IAM ロールがあります。ルールエンジンは、ルールのアクションが呼び出されたときに、そのロールを引き受けます。ロールは、ルール AWS アカウント と同じ で定義する必要があります。

ルールを作成または置き換えるときに、実質的にロールをルールエンジンに渡します。このオペレーションを実行するには `iam:PassRole` アクセス許可が必要です。このアクセス許可を検証するには、`iam:PassRole` アクセス許可を付与し、それを IAM ユーザーにアタッチするポリシーを作成します。次のポリシーは、ロールに `iam:PassRole` 権限を付与する方法を示しています。

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt1",
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": [
    "arn:aws:iam::123456789012:role/myRole"
   ]
  }
 ]
}
```

このポリシー例では、`iam:PassRole` ロールに `myRole` 許可が付与されます。ロールは、ロールの ARN を使用して指定されます。このポリシーを IAM ユーザーまたはユーザーが所属するロールにアタッチします。詳細については、「管理ポリシーの使用」を参照してください。

> **Note**
>
> Lambda 関数はリソースベースのポリシーを使用し、このポリシーは Lambda 関数自体に直接アタッチされます。Lambda 関数を呼び出すルールを作成する場合は、ロールを適用しないため、ルールを作成するユーザーに `iam:PassRole` アクセス許可は必要ありません。Lambda 関数の承認については、「リソースポリシーを使用したアクセス許可の付与」を参照してください。

AWS IoT ルールの作成

接続されたモノからデータをルーティングして他の AWS サービスとやり取りする AWS IoT ルールを作成できます。 AWS IoT ルールは、次のコンポーネントで構成されます。

ルールのコンポーネント

コンポーネント	説明	必須またはオプション
ルール名	ルールの名前。ルールの名前に個人を特定できる名前や情報を使用することはお勧めしません。	必須。
ルールの説明	ルールのテキスト説明。ルールの説明に個人を特定できる名前や情報を使用することはお勧めしません。	オプション。
SQL ステートメント	MQTT トピックで受け取ったメッセージをフィルター処理し、別の場所にデータを送るための単純な SQL 構文。詳細については、「AWS IoT SQL リファレンス」を参照してください。	必須。
SQL バージョン	ルールを評価する際に使用する SQL ルールエンジンのバージョン。このプロパティはオプションですが、SQL バージョンを指定することを強くお勧めします。 AWS IoT Core コンソールは、このプロパティを `2016-03-23` デフォルトで に設定します。 AWS CLI コマンドや CloudFormation テンプレートなど、このプロパティが設定されていない場合 `2015-10-08` は、 が使用されます。詳細	必須。

コンポーネント	説明	必須またはオプション
	については、「SQL バージョン」を参照してください。	
1 つ以上のアクション	アクションは、ルールを作成するときに AWS IoT 実行されます。例えば、データを DynamoDB テーブルに挿入する、データを Amazon S3 バケットに書き込む、Amazon SNS トピックに発行する、Lambda 関数を呼び出すなどのアクションを指定できます。	必須。
エラーアクション	アクションは AWS IoT 、ルールのアクションを実行できない場合に実行されます。	オプション。

AWS IoT ルールを作成する前に、必要な AWS リソースへのアクセスを許可するポリシーを持つ IAM ロールを作成する必要があります。 は、ルールを実装するときにこのロール AWS IoT を引き受けます。詳細については、「必要なアクセス許可を AWS IoT ルールに付与する」および「ロールのアクセス許可を渡す」を参照してください。

ルールを作成する際には、トピックに対して公開されるデータの量に注意してください。ワイルドカードのトピックパターンが含まれるルールを作成すると、一致するメッセージの割合が大きくなる可能性があります。この場合、場合によってはターゲットアクションに使用する AWS リソースの容量を増やす必要があります。処理の重複を防ぎ、コストを削減するために、再発行のルールでワイルドカードのトピックパターンを回避することをお勧めします。

> **Note**
>
> ルールの作成と更新は、管理者レベルのアクションです。ルールを作成または更新するアクセス権限のあるユーザーは、そのルールで処理されたデータにもアクセスできます。

ルールを作成する (コンソール)

ルール (AWS マネジメントコンソール) を作成するには

AWS マネジメントコンソール コマンドを使用してルールを作成します。

1. AWS IoT コンソールを開きます。
2. 左側のナビゲーションで、[管理] セクションから [メッセージルーティング] を選択します。次に、[ルール] を選択します。
3. [Rules] (ルール) ページで、[Create rule] (ルールの作成) を選択します。
4. [ルールのプロパティを指定] ページで、ルールの名前を入力します。[ルールの説明] と [タグ] はオプションです。[次へ] を選択します。
5. [SQL ステートメントを設定] ページで、SQL バージョンを選択し、SQL ステートメントを入力します。例えば、SQL ステートメントには `SELECT temperature FROM 'iot/topic' WHERE temperature > 50` などがあります。詳細については、「SQL バージョン」と「AWS IoT SQL リファレンス」を参照してください。
6. ルールアクションのアタッチページで、他の AWS サービスにデータをルーティングするルールアクションを追加します。

    1. [ルールアクション] で、ドロップダウンリストからルールアクションを選択します。例えば、[Kinesis Stream] を選択できます。ルールアクションの詳細については、「AWS IoT ルールアクション」を参照してください。
    2. 選択したルールアクションに応じて、関連する設定の詳細を入力します。例えば、[Kinesis Stream] を選択した場合、データストリームリソースを選択または作成し、オプションでパーティションキーなどの設定の詳細を入力する必要があります。パーティションキーは、スチーム内のシャード別にデータをグループ化するために使用されます。
    3. IAM ロールで、エンドポイント AWS IoT へのアクセスを許可するロールを選択または作成します。 AWS IoT は、選択した IAM ロール`aws-iot-rule`の下に というプレフィックスを持つポリシーを自動的に作成します。[表示] を選択して、IAM コンソールから IAM ロールとポリシーを表示できます。[エラーアクション] はオプションです。詳細については、[エラー処理 (エラーアクション)] を参照してください。ルールの IAM ロールの作成の詳細については、「必要とするアクセスをルールに付与する」を参照してください。[次へ] を選択します。
7. [確認と作成] ページで、すべての設定を確認し、必要に応じて編集します。[作成] を選択します。

ルールを正常に作成すると、[ルール] ページにそのルールが表示されます。ルールを選択して [詳細] ページを開き、ルールの表示、ルールの編集、ルールの無効化、ルールの削除を行うことができます。

ルール (CLI) を作成する

ルール (AWS CLI) を作成するには

ルールを作成するには、create-topic-rule コマンドを使用します。

```
aws iot create-topic-rule --rule-name myrule --topic-rule-payload file://myrule.json
```

以下のペイロードファイル例では、iot/test トピックに送信されたすべてのメッセージを指定の DynamoDB テーブルに挿入するルールが指定されています。SQL ステートメントはメッセージをフィルタリングし、ロール ARN は DynamoDB テーブルに書き込む AWS IoT アクセス許可を付与します。

```
{
 "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()}"
   }
  }
 ]
}
```

以下のペイロードファイル例では、iot/test トピックに送信されたすべてのメッセージを指定の S3 バケットに挿入するルールが指定されています。SQL ステートメントはメッセージをフィルタリングし、ロール ARN は Amazon S3 バケットに書き込む AWS IoT アクセス許可を付与します。

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

以下のペイロードファイル例では、データを Amazon OpenSearch Service にプッシュするルールが指定されています。

```
{
 "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()}"
   }
  }
 ]
}
```

以下のペイロードファイル例では、Lambda 関数を呼び出すルールが指定されています。

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function"
   }
  }
 ]
```

```
}
```

以下のペイロードファイル例では、Amazon SNS トピックに発行するルールが指定されています。

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

以下のペイロードファイル例では、別の MQTT トピックに再発行するルールが指定されています。

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "topic": "my-mqtt-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

以下のペイロードファイル例では、データを Amazon Data Firehose ストリーミングにプッシュするルールが指定されています。

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

以下のペイロードファイル例では、MQTT ペイロード内のデータが 1 として分類されている場合、Amazon SageMaker AI の machinelearning_predict 関数を使用してトピックに再発行するルールが指定されています。

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

以下は、Salesforce IoT クラウド入力ストリームにメッセージを発行するルールを持つペイロードファイルの例です。

```
{
 "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"
   }
```

```
  }
 ]
}
```

以下のペイロードファイル例では、Step Functions ステートマシンの実行を開始するルールが指定されています。

```
{
 "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 ルールの管理

次のアクションを使用してルールを管理できます AWS IoT 。

このトピックの内容

- ルールのタグ付け
- ルールの表示
- ルールの削除

ルールのタグ付け

新規のルールまたは既存のルールにさらに詳細性のレイヤーを追加するには、タグ付けを適用できます。タグ付けでは、ルール内のキーと値のペアを活用して、 AWS IoT リソースとサービスにルールを適用する方法と場所をより細かく制御できます。例えば、プレリリーステスト (`Key=environment, Value=beta`) のために、ルールのスコープをベータ環境でのみ適用するように制限したり、特定のエンドポイントのみから `iot/test` トピックに送信されたすべてのメッセージをキャプチャして、Amazon S3 バケットに保存したりできます。

IAM ポリシーの例

ルールのタグ付けのアクセス許可を付与する方法を示す例として、次のコマンドを実行してルールを作成し、タグ付けしてベータ環境にのみ適用する方法について考えてみましょう。

この例では、次のように置き換えます。

- *MyTopicRuleName* をルールの名前に置き換えます。
- *myrule.json* をポリシードキュメントの名前に置き換えます。

```
aws iot create-topic-rule
    --rule-name MyTopicRuleName
    --topic-rule-payload file://myrule.json
    --tags "environment=beta"
```

この例では、次の IAM ポリシーを使用する必要があります。

```
{
    "Version":"2012-10-17",
    "Statement":
    {
        "Action": [ "iot:CreateTopicRule", "iot:TagResource" ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:rule/MyTopicRuleName"
        ]
    }
}
```

上の例は、ベータ環境にのみ適用される、MyTopicRuleName という新しく作成されたルールを示しています。MyTopicRuleName を明確に呼び出したポリシーステートメントの iot:TagResource により、MyTopicRuleName の作成時または更新時にタグ付けが可能になります。ルールの作成時にパラメータ --tags "environment=beta" を使用すると、MyTopicRuleName のスコープがベータ環境のみに制限されます。パラメータ --tags "environment=beta" を削除した場合、MyTopicRuleName はすべての環境に適用されます。

AWS IoT ルールに固有の IAM ロールとポリシーの作成に関する詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

リソースのタグ付けの詳細については、「AWS IoT リソースのタグ付け」を参照してください。

ルールの表示

ルールをリスト表示するには、list-topic-rules コマンドを使用します。

```
aws iot list-topic-rules
```

ルールに関する情報を取得するには、get-topic-rule コマンドを使用します。

```
aws iot get-topic-rule --rule-name myrule
```

ルールの削除

不要になったルールは、削除することができます。

ルール (AWS CLI) を削除するには

delete-topic-rule コマンドを使用して、ルールを削除します。

```
aws iot delete-topic-rule --rule-name myrule
```

AWS IoT ルールアクション

AWS IoT ルールアクションは、ルールが呼び出されたときに何をするかを指定します。Amazon DynamoDB データベースへのデータの送信、Amazon Kinesis Data Streams へのデータの送信、 関数の AWS Lambda 呼び出しなどのアクションを定義できます。 は、アクションのサービス AWS リージョン が利用可能な で以下のアクション AWS IoT をサポートします。

ルールアクション	説明	API での名前
Apache Kafka	Apache Kafka クラスターにメッセージを送信します。	`kafka`
CloudWatch アラーム	Amazon CloudWatch アラームの状態を変更します。	`cloudwatchAlarm`
CloudWatch Logs	Amazon CloudWatch Logs にメッセージを送信します。	`cloudwatchLogs`

ルールアクション	説明	API での名前
CloudWatch メトリクス	CloudWatch メトリクスにメッセージを送信します。	`cloudwatchMetric`
DynamoDB	DynamoDB テーブルにメッセージを送信します。	`dynamoDB`
DynamoDBv2	DynamoDB テーブル内の複数の列に、メッセージデータを送信します。	`dynamoDBv2`
Elasticsearch	OpenSearch エンドポイントにメッセージを送信します。	`OpenSearch`
HTTP	HTTPS エンドポイントに、メッセージをポストします。	`http`
AWS IoT Events	AWS IoT Events 入力にメッセージを送信します。	`iotEvents`
AWS IoT SiteWise	メッセージデータを AWS IoT SiteWise アセットプロパティに送信します。	`iotSiteWise`
Firehose	Firehose 配信ストリーミングにメッセージを送信します。	`firehose`
Kinesis Data Streams	Kinesis データストリーミングにメッセージを送信します。	`kinesis`
Lambda	メッセージデータを入力として Lambda 関数を呼び出します。	`lambda`
ロケーション	位置データを Amazon Location Service に送信します。	`location`

ルールアクション	説明	API での名前
OpenSearch	Amazon OpenSearch Service エンドポイントにメッセージを送信します。	`OpenSearch`
再発行	メッセージを別の MQTT トピックに再発行します。	`republish`
S3	Amazon Simple Storage Service (Amazon S3) バケットにメッセージを保存します。	`s3`
Salesforce IoT	Salesforce の IoT 入力ストリーミングにメッセージを送信します。	`salesforce`
SNS	Amazon Simple Notification Service (Amazon SNS) プッシュ通知としてメッセージを発行します。	`sns`
SQS	Amazon Simple Queue Service (Amazon SQS) キューにメッセージを送信します。	`sqs`
ステップ関数	AWS Step Functions ステートマシンを起動します。	`stepFunctions`
the section called “Timestream”	Amazon Timestream データベーステーブルに、メッセージを送信します。	`timestream`

注意事項

- ルールアクションがそのリソースとやり取りできるように、別のサービスのリソース AWS リージョン と同じ でルールを定義します。
- 断続的なエラーが発生した場合、 AWS IoT ルールエンジンはアクションの実行を複数回試みることがあります。すべての試行が失敗すると、メッセージは破棄され、CloudWatch ログにエラーが表示されます。障害が発生した後に呼び出される各ルールに対して、エラーアクションを指定できます。詳細については、「エラー処理 (エラーアクション)」を参照してください。
- 一部のルールアクションは、 AWS Key Management Service (AWS KMS) と統合されたサービスでアクションをアクティブ化して、保管時のデータ暗号化をサポートします。カスタマーマネージド AWS KMS key (KMS キー) を使用して保管中のデータを暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。カスタマーマネージド KMS キー許可を管理する方法については、該当するサービスガイドのデータ暗号化トピックを参照してください。カスタマーマネージド KMS キーの詳細については、[AWS Key Management Service Developer Guide](デベロッパーガイド)の「AWS Key Management Service concepts」(コンセプト) を参照してください。

  エラーアクションの SQL ステートメントでは、、`aws_lambda()`、`get_dynamodb()`、、、`get_registry_data()`、`get_thing_shadow()` `get_secret()` `machinelearning_predict()`などの外部関数を含む任意の関数または置換テンプレートを使用できます`decode()`。エラーアクションで外部関数を呼び出す必要がある場合、エラーアクションを呼び出すと、外部関数に追加の請求が発生する可能性があります。

Apache Kafka

Apache Kafka (Kafka) アクションは、Amazon Managed Streaming for Apache Kafka (Amazon MSK)、Confluent Cloud などのサードパーティープロバイダーによって管理される Apache Kafka クラスター、またはセルフマネージド Apache Kafka クラスターに直接メッセージを送信します。Kafka ルールアクションを使用すると、IoT データを Kafka クラスターにルーティングできます。これにより、ストリーミング分析、データ統合、視覚化、ミッションクリティカルなビジネス用途など、さまざまな目的で高性能データパイプラインを構築できます。

> **Note**
>
> このトピックでは、Apache Kafka プラットフォームおよび関連概念について精通していることを前提としています。Apache Kafka の詳細については、「Apache Kafka」を参照してください。MSK Serverless はサポートされていません。MSK Serverless クラスターは、Apache Kafka ルールアクションが現在サポートしていない IAM 認証を介してのみ実行できます。Confluent AWS IoT Core で を設定する方法の詳細については、「Confluent の活用」およびIoT デバイスとデータ管理の課題の解決 AWS」を参照してください。

要件

このルールアクションには、以下の要件があります。

- が 、ec2:CreateNetworkInterface、、ec2:DescribeNetworkInterfaces、、、ec2:Describ
  および ec2:DescribeSecurityGroupsオペレーションを実行するために引き受
  け AWS IoT ることができる IAM ec2:CreateNetworkInterfacePermission
  ec2:DeleteNetworkInterface ec2:DescribeSubnets ec2:DescribeVpcsロール。この
  ロールは、Kafka ブローカーに到達するために、Amazon Virtual Private Cloud への伸縮自在な
  ネットワークインターフェイスを作成および管理します。詳細については、「必要なアクセスを
  AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT Core するロールを選択または作成できます。

  ネットワークインターフェイスの詳細については、Amazon EC2 ユーザーガイドの「Elastic Network Interface」を参照してください。

  指定したロールにアタッチされるポリシーは次の例のようになります。

```
{
    "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 を使用して Kafka ブローカーへの接続に必要な認証情報を保存する場合は、 が `secretsmanager:GetSecretValue`および `secretsmanager:DescribeSecret`オペレーションを実行するために引き受け AWS IoT Core ることができる IAM ロールを作成する必要があります。

  指定したロールにアタッチされるポリシーは次の例のようになります。

```
{
    "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 (Amazon VPC) 内で Apache Kafka クラスターを実行できます。Amazon VPC 送信先を作成し、サブネットで NAT ゲートウェイを使用して、 からパブリック Kafka クラスター AWS IoT にメッセージを転送する必要があります。 AWS IoT ルールエンジンは、VPC 送信先にリストされている各サブネットにネットワークインターフェイスを作成し、VPC に直接トラフィックをルーティングします。VPC 送信先を作成すると、 AWS IoT ルー

ルエンジンは自動的に VPC ルールアクションを作成します。VPC ルールアクションの詳細については、Virtual private cloud (仮想プライベートクラウド)(VPC) 送信先 を参照してください。

- カスタマーマネージド AWS KMS key (KMS キー) を使用して保管中のデータを暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。詳細については、Amazon Managed Streaming for Apache Kafka デベロッパーガイドの Amazon MSK 暗号化を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

destinationArn

VPC 送信先の Amazon リソースネーム (ARN)。VPC 送信先の作成方法の詳細については、「Virtual private cloud (仮想プライベートクラウド)(VPC) 送信先」を参照してください。

トピック

Kafka のブローカーに送信されるメッセージの Kafka のトピック。

このフィールドは、置換テンプレートを使用して置換できます。詳細については、「the section called “置換テンプレート”」を参照してください。

キー (オプション)

Kafka のメッセージキー

このフィールドは、置換テンプレートを使用して置換できます。詳細については、「the section called “置換テンプレート”」を参照してください。

ヘッダー (オプション)

指定した Kafka ヘッダーのリスト。各ヘッダーは、Kafka アクションを作成するときに指定できるキーと値のペア (1 つのキーと 1 つの値) です。これらのヘッダーを使用して、メッセージペイロードを変更せずに IoT クライアントからダウンストリーム Kafka クラスターにデータをルーティングできます。

このフィールドは、置換テンプレートを使用して置換できます。インラインルールの関数を Kafka Action のヘッダーで代替テンプレートとして渡す方法については、「例」を参照してください。詳細については、「the section called “置換テンプレート”」を参照してください。

> **Note**
>
> バイナリ形式のヘッダーはサポートされていません。

パーティション (オプション)

Kafka のメッセージパーティション。

このフィールドは、置換テンプレートを使用して置換できます。詳細については、「the section called “置換テンプレート”」を参照してください。

clientProperties

Apache Kafka プロデューサークライアントのプロパティを定義するオブジェクト。

acks (オプション)

リクエストが完了したとみなされる前に、プロデューサーがサーバーに受信することを求める確認応答の数。

値として 0 を指定すると、プロデューサーはサーバーからの確認応答を待機しなくなります。サーバーがメッセージを受信しない場合、プロデューサーはメッセージの送信を再試行しません。

有効な値は、`-1`、`0`、`1`、`all` です。デフォルト値は `1` です。

bootstrap.servers

Kafka クラスターへの初期接続を確立するために使用されるホストとポートのペア (`host1:port1`、`host2:port2` など) のリスト。

compression.type (optional)

プロデューサーによって生成されるすべてのデータの圧縮タイプ。

有効な値: `none`、`gzip`、`snappy`、`lz4`、`zstd`。デフォルト値は `none` です。

security.protocol

Kafka ブローカーにアタッチするために使用されるセキュリティプロトコル。

有効な値: `SSL`、`SASL_SSL`。デフォルト値は `SSL` です。

key.serializer

ProducerRecord で提供するキーオブジェクトをバイトに変換する方法を指定します。

有効な値: StringSerializer。

value.serializer

ProducerRecord で提供する値オブジェクトをバイトに変換する方法を指定します。

有効な値: ByteBufferSerializer。

ssl.truststore

base64 形式のトラストストアファイル、または AWS Secrets Manager 内のトラストストアファイルの場所。トラストストアが Amazon 認証機関 (CA) によって信頼されている場合は、この値は必須ではありません。

このフィールドは置換テンプレートをサポートしています。Secrets Manager を使用して Kafka ブローカーへの接続に必要な認証情報を保存する場合、get_secret SQL 関数を使用してこのフィールドの値を取得できます。置換テンプレートの詳細については、「the section called “置換テンプレート”」を参照してください。get_secret SQL 関数の詳細については、「the section called “get_secret(secretId, secretType, key, roleArn)”」を参照してください。トラストストアがファイル形式の場合は、SecretBinary パラメータを使用します。トラストストアが文字列の形式である場合は、SecretString パラメータを使用します。

この値の最大サイズは 65 KB です。

ssl.truststore.password

信頼ストアのパスワード。この値は、トラストストアのパスワードを作成した場合にのみ必要です。

ssl.keystore

キーストアファイル。security.protocol の値として SSL を指定する場合、この値は必須です。

このフィールドは置換テンプレートをサポートしています。Secrets Manager を使用して、Kafka ブローカーへの接続に必要な認証情報を保存します。このフィールドの値を取得するには、get_secret 関数を使用します。置換テンプレートの詳細については、「the section called “置換テンプレート”」を参照してください。get_secret SQL 関数の詳細については、「the section called “get_secret(secretId, secretType, key, roleArn)”」を参照してください。SecretBinary パラメータを使用します。

ssl.keystore.password

キーストアファイルのストアパスワード。ssl.keystore の値を指定している場合、この値は必須です。

このフィールドの値はプレーンテキストにすることができます。このフィールドは、代替テンプレートもサポートします。Secrets Manager を使用して、Kafka ブローカーへの接続に必要な認証情報を保存します。このフィールドの値を取得するには、get_secret 関数を使用します。置換テンプレートの詳細については、「the section called “置換テンプレート”」を参照してください。get_secret SQL 関数の詳細については、「the section called “get_secret(secretId, secretType, key, roleArn)”」を参照してください。SecretString パラメータを使用します。

ssl.key.password

キーストアファイル内のプライベートキーのパスワード。

このフィールドは置換テンプレートをサポートしています。Secrets Manager を使用して、Kafka ブローカーへの接続に必要な認証情報を保存します。このフィールドの値を取得するには、get_secret 関数を使用します。置換テンプレートの詳細については、「the section called “置換テンプレート”」を参照してください。get_secret SQL 関数の詳細については、「the section called “get_secret(secretId, secretType, key, roleArn)”」を参照してください。SecretString パラメータを使用します。

sasl.mechanism

Kafka のブローカーに接続するために使用されるセキュリティメカニズム。この値は、security.protocol の SASL_SSL を指定する場合に必要です。

有効な値: PLAIN、SCRAM-SHA-512、GSSAPI。

> **Note**
>
> SCRAM-SHA-512 は、cn-north-1、cn-northwest-1、us-gov-east-1、および us-gov-west-1 リージョンでサポートされている唯一のセキュリティメカニズムです。

sasl.plain.username

Secrets Manager からシークレット文字列を取得するために使用されるユーザー名。この値は、security.protocol の SASL_SSL および sasl.mechanism の PLAIN を指定する場合に必要です。

sasl.plain.password

Secrets Manager からシークレット文字列を取得するために使用されるパスワード。この値は、security.protocol の SASL_SSL および sasl.mechanism の PLAIN を指定する場合に必要です。

sasl.scram.username

Secrets Manager からシークレット文字列を取得するために使用されるユーザー名。この値は、security.protocol の SASL_SSL および sasl.mechanism の SCRAM-SHA-512 を指定する場合に必要です。

sasl.scram.password

Secrets Manager からシークレット文字列を取得するために使用されるパスワード。この値は、security.protocol の SASL_SSL および sasl.mechanism の SCRAM-SHA-512 を指定する場合に必要です。

sasl.kerberos.keytab

Secrets Manager の Kerberos 認証用のキータブファイル。この値は、security.protocol の SASL_SSL および sasl.mechanism の GSSAPI を指定する場合に必要です。

このフィールドは置換テンプレートをサポートしています。Secrets Manager を使用して、Kafka ブローカーへの接続に必要な認証情報を保存します。このフィールドの値を取得するには、get_secret 関数を使用します。置換テンプレートの詳細については、「the section called “置換テンプレート”」を参照してください。get_secret SQL 関数の詳細については、「the section called “get_secret(secretId, secretType, key, roleArn)”」を参照してください。SecretBinaryパラメータを使用します。

sasl.kerberos.service.name

Apache Kafka が実行される Kerberos プリンシパル名。この値は、security.protocol の SASL_SSL および sasl.mechanism の GSSAPI を指定する場合に必要です。

sasl.kerberos.krb5.kdc

Apache Kafka プロデューサークライアントが接続するキー配布センター (KDC) のホスト名。この値は、security.protocol の SASL_SSL および sasl.mechanism の GSSAPI を指定する場合に必要です。

sasl.kerberos.krb5.realm

Apache Kafka プロデューサークライアントが接続する領域。この値は、security.protocol の SASL_SSL および sasl.mechanism の GSSAPI を指定する場合に必要です。

sasl.kerberos.principal

Kerberos 対応サービスにアクセスするためのチケットを Kerberos が割り当てることができる一意の Kerberos ID。この値は、security.protocol の SASL_SSL および sasl.mechanism の GSSAPI を指定する場合に必要です。

例

次の JSON 例では、 AWS IoT ルールで Apache Kafka アクションを定義します。次の例では、sourceIp () インライン関数を Kafka Action ヘッダーの代替テンプレートとして渡します。

```
{
 "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 セットアップに関する重要な注意事項

- キー配布センター (KDC) は、ターゲット VPC 内のプライベートドメインネームシステム (DNS) を介して解決可能である必要があります。考えられる方法の 1 つは、KDC DNS エントリをプライベートホストゾーンに追加することです。このアプローチの詳細については、「プライベートホストゾーンの使用」を参照してください。
- 各 VPC で DNS 解決が有効になっている必要があります。詳細については、「Using DNS with Your VPC」を参照してください。
- VPC 送信先のネットワークインターフェイスセキュリティグループとインスタンスレベルのセキュリティグループは、次のポートで VPC 内からのトラフィックを許可する必要があります。
  - ブートストラップブローカーのリスナーポート上の TCP トラフィック (通常は 9092 ですが、9000 ~ 9100 の範囲内である必要があります)
  - KDC のポート 88 の TCP および UDP トラフィック
- SCRAM-SHA-512 は、cn-north-1、cn-northwest-1、us-gov-east-1、および us-gov-west-1 リージョンでサポートされている唯一のセキュリティメカニズムです。

Virtual private cloud (仮想プライベートクラウド)(VPC) 送信先

Apache Kafka ルールアクションは、データを Amazon Virtual Private Cloud (Amazon VPC) の Apache Kafka クラスターにルーティングします。Apache Kafka ルールアクションで使用される VPC 設定は、ルールアクションの VPC 送信先を指定すると自動的に有効になります。

VPC の送信先は、VPC 内のサブネットのリストに含まれています。ルールエンジンは、このリストで指定する各サブネットで Elastic Network Interface を作成します。ネットワークインターフェイスの詳細については、Amazon EC2 ユーザーガイドの Elastic Network Interface （伸縮自在のネットワークインターフェース ）を参照してください。

要件と考慮事項

- インターネット経由でパブリックエンドポイントを使用してアクセスされるセルフマネージド型 Apache Kafka クラスターを使用している場合。
  - サブネット内のインスタンス用に NAT ゲートウェイを作成します。NAT ゲートウェイには、インターネットに接続できるパブリック IP アドレスがあるため、ルールエンジンがメッセージをパブリック Kafka クラスターに転送できます。
  - VPC の宛先によって作成された Elastic Network Interface (ENI) を使用して Elastic IP アドレスを割り当てます。使用するセキュリティグループは、着信トラフィックをブロックするように設定する必要があります。

    > **Note**
    >
    > VPC の宛先が無効になってから再度有効化されている場合は、Elastic IP を新しい ENI に再度関連付ける必要があります。

- 30 日間連続してトラフィックを受信しなかった VPC トピックルールの送信先は、無効になります。
- VPC の送信先で使用されるリソースが変更された場合、送信先は無効になり、使用できなくなります。
- VPC 送信先を無効にすることができる変更には、次のようなものがあります。
  - VPC、サブネット、セキュリティグループ、または使用されるロールを削除します。
  - 必要なアクセス許可がなくなるようにロールを変更します。
  - サブネット容量に近いため、FedRAMP パッチを適用できません。
  - 送信先を無効にします。

料金

請求のために、リソースが VPC 内にある場合にリソースにメッセージを送信するアクションに加えて、VPC ルールアクションが計測されます。料金については、AWS IoT Core 料金を参照してください。

仮想プライベートクラウド (VPC) トピックルールの送信先の作成

CreateTopicRuleDestination API または AWS IoT Core コンソールを使用して、仮想プライベートクラウド (VPC) の送信先を作成します。

VPC 送信先を作成する場合は、次の情報を指定する必要があります。

vpcId

VPC 送信先の一意の ID。

subnetIds

ルールエンジンが Elastic Network Iinterfaces を作成するサブネットのリスト。ルールエンジンは、リスト内のサブネットごとに 1 つのネットワークインターフェイスを割り当てます。

securityGroups (オプション)

ネットワークインターフェイスに適用するセキュリティグループのリスト。

roleArn

ユーザーに代わってネットワークインターフェイスを作成するための許可を持つロールの Amazon リソースネーム (ARN)。

この ARN には、次の例のようなポリシーがアタッチされている必要があります。

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

を使用して VPC 送信先を作成する AWS CLI

以下の例は、 AWS CLIを使用して VPC 送信先を作成する方法を示しています。

```
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"}'
```

このコマンドを実行すると、VPC 送信先のステータスは、 IN_PROGRESS になります。数分後、そのステータスはERROR (コマンドが成功しなかった場合) または ENABLEDに変わります。送信先ステータスが ENABLED の場合、使用可能です。

次のコマンドを使用して、VPC 送信先のステータスを取得できます。

```
aws --region region iot get-topic-rule-destination --arn "VPCDestinationARN"
```

AWS IoT Core コンソールを使用した VPC 送信先の作成

次の手順では、 AWS IoT Core コンソールを使用して VPC 送信先を作成する方法について説明します。

1. AWS IoT Core コンソールに移動します。左のペインの[Act] タブで、[送信先] を選択します。
2. 以下のフィールドに値を入力します。
   - VPC ID
   - サブネット ID
   - セキュリティグループ
3. ネットワークインターフェイスの作成に必要な許可を持つロールを選択します。上記のポリシー例には、これらの許可が含まれています。

VPC 送信先ステータスが、[ENABLED](有効) の場合、使用する準備ができています。

CloudWatch アラーム

CloudWatch アラーム (cloudWatchAlarm) アクションは、Amazon CloudWatch アラームの状態を変更します。状態変更の理由と、この呼び出しでの値を指定できます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールcloudwatch:SetAlarmState。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`alarmName`

CloudWatch アラーム名。

置換テンプレートをサポート: API および AWS CLI のみ

`stateReason`

アラーム変更の理由。

置換テンプレートをサポート: はい

`stateValue`

アラーム状態の値。有効な値: `OK`、`ALARM`、`INSUFFICIENT_DATA`。

置換テンプレートをサポート: はい

`roleArn`

CloudWatch アラームへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON 例は、 AWS IoT ルールで CloudWatch アラームアクションを定義します。

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

以下の資料も参照してください。

- Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」
- Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch アラームの使用」

CloudWatch Logs

CloudWatch Logs (`cloudwatchLogs`) アクションは、データを Amazon CloudWatch Logs に送信します。`batchMode` を使用して、1 つのメッセージで複数のデバイスログレコードをアップロードし、タイムスタンプを付けることができます。アクションがデータを送信するロググループを指定できます。

要件

このルールアクションには、以下の要件があります。

- が `logs:CreateLogStream`、、`logs:DescribeLogStreams`および `logs:PutLogEvents`オペレーションを実行するために引き受け AWS IoT ることができる IAM ロール。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- カスタマーマネージド AWS KMS key (KMS キー) を使用して CloudWatch Logs のログデータを暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。詳しくは、Amazon CloudWatch Logs ユーザーガイドのAWS KMSを使用して CloudWatch Logs のログデータを暗号化するを参照してください。

`batchMode` の MQTT メッセージ形式の要件

`batchMode` をオフにして CloudWatch Logs ルールアクションを使用する場合、MQTT メッセージフォーマットの要件はありません。(注意: `batchMode` パラメータのデフォルト値は `false` です)。ただし、`batchMode` をオンにして、CloudWatch Logs ルールアクションを使用する場合 (パラメータ値は `true`) 、デバイス側のログを含む MQTT メッセージはタイムスタンプとメッセージペイロードを含むようにフォーマットする必要があります。注意: `timestamp` は、イベントの発生時刻を表し、1970 年 1 月 1 日 00:00:00 UTC からのミリ秒数で表されます。

発行形式の例を次に示します。

```
[
  {"timestamp": 1673520691093, "message": "Test message 1"},
  {"timestamp": 1673520692879, "message": "Test message 2"},
  {"timestamp": 1673520693442, "message": "Test message 3"}
]
```

デバイス側のログの生成方法によっては、この要件を満たすために送信する前にフィルタリングして再フォーマットする必要がある場合があります。詳細については、「MQTT メッセージペイロード」を参照してください。

`batchMode` パラメータに関係なく、`message`コンテンツは AWS IoT メッセージサイズの制限に準拠する必要があります。詳細については、「AWS IoT Core エンドポイントとクォータ」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`logGroupName`

アクションがデータを送信する CloudWatch ロググループ。

置換テンプレートをサポート: API および AWS CLI のみ

`roleArn`

CloudWatch ロググループへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

(オプション) batchMode

ログレコードのバッチを抽出して CloudWatch にアップロードするかどうかを示します。値には trueまたは false (デフォルト) が含まれます。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで CloudWatch Logs アクションを定義します。

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

関連情報

- Amazon CloudWatch Logs ユーザーガイドの「Amazon CloudWatch Logs とは」

CloudWatch メトリクス

CloudWatch メトリクス (cloudwatchMetric) アクションは Amazon CloudWatch メトリクスを取得します。メトリクスの名前空間、名前、値、単位、タイムスタンプを指定できます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールcloudwatch:PutMetricData。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`metricName`

CloudWatch メトリクス名前空間。

置換テンプレートをサポート: はい

`metricNamespace`

CloudWatch メトリクス名前空間。

置換テンプレートをサポート: はい

`metricUnit`

CloudWatch でサポートされているメトリクス単位。

置換テンプレートをサポート: はい

`metricValue`

CloudWatch メトリクス値を含む文字列。

置換テンプレートをサポート: はい

`metricTimestamp`

(オプション) タイムスタンプ (秒単位) を含む文字列 (Unix エポック時間)。デフォルトは現在の Unix エポック時間です。

置換テンプレートをサポート: はい

`roleArn`

CloudWatch メトリクスへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON 例では、 AWS IoT ルールで CloudWatch メトリクスアクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して CloudWatch メトリクスアクションを定義します。

```
{
    "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"
```

```
                }
            }
        ]
    }
}
```

関連情報

- Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」
- Amazon CloudWatch ユーザーガイドの Amazon CloudWatch メトリクスの使用

DynamoDB

DynamoDB (dynamoDB) アクションは、MQTT メッセージの全部または一部を Amazon DynamoDB テーブルに書き込みます。

DynamoDB アクションでルールを作成してテストする方法を示すチュートリアルに従うことができます。詳細については、「チュートリアル: デバイスデータの DynamoDB テーブルへの保存」を参照してください。

> **Note**
>
> このルールは、JSON 以外のデータをバイナリデータとして DynamoDB に書き込みます。DynamoDB コンソールでは、base64 でエンコードされたテキストとしてデータが表示されます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールdynamodb:PutItem。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- カスタマーマネージド AWS KMS key (KMS キー) を使用して DynamoDB に保管中のデータを暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。詳

細については、[Amazon DynamoDB Getting Started Guide](Amazon DynamoDB 開始方法ガイド) の [Customer Managed KMS key](カスタマー管理の CMK) を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`tableName`

DynamoDB テーブルの名前。

置換テンプレートをサポート: API および AWS CLI のみ

`hashKeyField`

ハッシュキー (パーティションキーとも呼ばれます) の名前。

代替テンプレートをサポート: API および AWS CLI のみ

`hashKeyType`

(オプション) ハッシュキー (パーティションキーとも呼ばれます) のデータ型。有効な値: `STRING`、`NUMBER`。

置換テンプレートをサポート: API および AWS CLI のみ

`hashKeyValue`

ハッシュキーの値。`${topic()}` や `${timestamp()}` などの置換テンプレートの使用を検討してください。

置換テンプレートをサポート: はい

`rangeKeyField`

(オプション) 範囲キー (ソートキーとも呼ばれます) の名前。

置換テンプレートをサポート: API および AWS CLI のみ

`rangeKeyType`

(オプション) 範囲キー (ソートキーとも呼ばれます) のデータ型。有効な値: `STRING`、`NUMBER`。

置換テンプレートをサポート: API および AWS CLI のみ

`rangeKeyValue`

(オプション) 範囲キーの値。`${topic()}` や `${timestamp()}` などの置換テンプレートの使用を検討してください。

置換テンプレートをサポート: はい

`payloadField`

(オプション) ペイロードが書き込まれる列の名前。この値を省略した場合、ペイロードは `payload` という名前の列に書き込まれます。

置換テンプレートをサポート: はい

`operation`

(オプション) 実行する操作の種類。有効な値: `INSERT`、`UPDATE`、`DELETE`。

置換テンプレートをサポート: はい

`roleARN`

DynamoDB テーブルへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

DynamoDB テーブルに書き込まれるデータは、ルールの SQL ステートメントの結果です。

例

次の JSON の例では、 AWS IoT ルールで DynamoDB アクションを定義します。

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

関連情報

- Amazon DynamoDB デベロッパーガイドの Amazon DynamoDB とは?
- Amazon DynamoDB デベロッパーガイドの DynamoDB の開始方法
- チュートリアル: デバイスデータの DynamoDB テーブルへの保存

DynamoDBv2

DynamoDBv2 (dynamoDBv2) アクションは、MQTT メッセージの全部または一部を Amazon DynamoDB テーブルに書き込みます。ペイロードの各属性は、DynamoDB データベースの個別の列に書き込まれます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールdynamodb:PutItem。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- MQTT メッセージのペイロードは、定義されている場合、テーブルのプライマリパーティションキーおよびテーブルのプライマリソートキーに一致するルートレベルキーが含まれる必要があります。
- カスタマーマネージド AWS KMS key (KMS キー) を使用して DynamoDB に保管中のデータを暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。詳細については、[Amazon DynamoDB Getting Started Guide](Amazon DynamoDB 開始方法ガイド) の [Customer Managed KMS key](カスタマー管理の CMK) を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`putItem`

メッセージデータを書き込む DynamoDB テーブルを指定するオブジェクト。このオブジェクトには、次の情報が含まれている必要があります。

`tableName`

DynamoDB テーブルの名前。

置換テンプレートをサポート: API および AWS CLI のみ

`roleARN`

DynamoDB テーブルへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

DynamoDB テーブルに書き込まれるデータは、ルールの SQL ステートメントの結果です。

例

次の JSON の例では、 AWS IoT ルールで DynamoDBv2 アクションを定義します。

```
{
    "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",
                }
            }
```

```
        ]
    }
}
```

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して DynamoDB アクションを定義します。

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

関連情報

- Amazon DynamoDB デベロッパーガイドの Amazon DynamoDB とは?
- Amazon DynamoDB デベロッパーガイドの DynamoDB の開始方法

Elasticsearch

Elasticsearch (`elasticsearch`) アクションは、Amazon OpenSearch Service ドメインに MQTT メッセージのデータを書き込みます。その後、OpenSearch Dashboards などのツールを使用して、OpenSearch Service のデータをクエリおよび視覚化できます。

> **Warning**
>
> `Elasticsearch`アクションは、既存のルールアクションのみで使用できます。新しいルールアクションを作成したり、既存のルールアクションを更新したりする

には、OpenSearchルールアクションを代わりに使用します。詳細については、「OpenSearch」を参照してください。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールes:ESHttpPut。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- カスタマー管理 AWS KMS key (KMS キー) を使用して OpenSearch で保管中のデータを暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。詳細については、Amazon OpenSearch Service デベロッパーガイドの「Amazon OpenSearch Service の保管中のデータの暗号化」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`endpoint`

サービスドメインのエンドポイント。

代替テンプレートをサポート: API および AWS CLI のみ

`index`

データを保存したい インデックス。

置換テンプレートをサポート: はい

`type`

保存するドキュメントのタイプ。

置換テンプレートをサポート: はい

`id`

各ドキュメントの一意の識別子。

置換テンプレートをサポート: はい

`roleARN`

OpenSearch Service ドメインへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで Elasticsearch アクションを定義し、`elasticsearch`アクションのフィールドを指定する方法を定義します。詳しくは、ElasticsearchAction を参照してください。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して Elasticsearch アクションを定義します。

```
{
```

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

関連情報

- OpenSearch
- Amazon OpenSearch Service とは

HTTP

HTTPS (http) アクションは、MQTT メッセージからウェブアプリケーションまたはサービスを指すことができる HTTPS エンドポイントにデータを送信します。

要件

このルールアクションには、以下の要件があります。

- ルールエンジンが HTTPS エンドポイントを使用する前に、それらの HTTPS エンドポイントを確認して有効にする必要があります。詳細については、「HTTP トピックルール送信先の使用」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`url`

HTTP POST メソッドを使用してメッセージを送信する HTTPS エンドポイント。ホスト名の代わりに IP アドレスを使用する場合は、IPv4 アドレスである必要があります。IPv6 アドレスはサポートされません。

置換テンプレートをサポート: はい

`confirmationUrl`

(オプション) 指定した場合、確認 URL AWS IoT を使用して一致するトピックルールの送信先を作成します。トピックルールの送信先は、HTTPS アクションで使用する前に有効にする必要があります。詳細については、「HTTP トピックルール送信先の使用」を参照してください。置換テンプレートを使用する場合、`http` アクションを使用する前に手動でトピックルールの送信先を作成する必要があります。`confirmationUrl` は `url` のプレフィックスである必要があります。

`url` との関係および `confirmationUrl` は、次のように記述されます。

- `url` がハードコードされており、 が指定されていない場合`confirmationUrl`、 `url`フィールドは暗黙的に として扱われます`confirmationUrl`。 は のトピックルールの送信先 AWS IoT を作成します`url`。
- `url` と `confirmationUrl`がハードコードされている場合、 は で始まる`url`必要があります`confirmationUrl`。 は のトピックルールの送信先 AWS IoT を作成します`confirmationUrl`。
- `url` に置換テンプレートが含まれている場合は、`confirmationUrl` を指定し、`url` は `confirmationUrl` で始まる必要があります。`confirmationUrl` に置換テンプレートが含まれている場合、`http` アクションを使用する前に手動でトピックルールの送信先を作成する必要があります。`confirmationUrl` に置換テンプレートが含まれていない場合、 は のトピックルールの送信先 AWS IoT を作成します`confirmationUrl`。

置換テンプレートをサポート: はい

`headers`

(オプション) エンドポイントへの HTTP リクエストに含めるヘッダーのリスト。各ヘッダーには、以下の情報が必要です。

`key`

ヘッダーのキー。

置換テンプレートをサポート: いいえ

`value`

ヘッダーの値

置換テンプレートをサポート: はい

> **Note**
>
> ペイロードが JSON 形式の場合、デフォルトのコンテンツタイプは application/json です。それ以外の場合は、application/octet-stream です。キー content-type (大文字と小文字を区別しない) でヘッダー内の正確なコンテンツタイプを指定することで、上書きできます。

`auth`

(オプション) `url` 引数で指定されたエンドポイント URL に接続するためにルールエンジンが使用する認証。現在、サポートされている認証タイプは署名バージョン 4 のみです。認可の詳細については、「HTTP 承認」を参照してください。

置換テンプレートをサポート: いいえ

`enableBatching`

(オプション) HTTP アクションメッセージを特定の URL の単一のリクエストに処理するかどうか。値は true または false にすることができます。バッチ処理の詳細については、「HTTP アクションメッセージのバッチ処理」を参照してください。

ブール値

置換テンプレートをサポート: いいえ

`batchConfig`

(オプション) バッチ処理の設定。有効にしたら、`batchConfig`パラメータを指定する必要があります。`batchConfig` パラメータを指定しない場合、デフォルト値が使用されます。

`maxBatchOpenMs`

送信メッセージが他のメッセージがバッチを作成するのを待機する最大時間 (ミリ秒単位) 。設定が大きいほど、バッチ処理された HTTP アクションのレイテンシーが長くなります。

最小値: 5 ミリ秒。最大値: 200 ミリ秒。

デフォルト値: 20 ミリ秒

置換テンプレートをサポート: いいえ

`maxBatchSize`

1 回のアクション実行でまとめてバッチ処理されるメッセージの最大数。

置換テンプレートをサポート: いいえ

最小値: 2 メッセージ。最大値: 10 メッセージ

デフォルト値: 10 メッセージ

`maxBatchSizeBytes`

メッセージバッチの最大サイズ、バイト単位。

最小値: 100 バイト。最大値: 131,072 バイト

デフォルト値: 5,120 バイト

置換テンプレートをサポート: いいえ

> **Note**
>
> ペイロードが JSON 形式の場合、デフォルトのコンテンツタイプは application/json です。それ以外の場合は、application/octet-stream です。キー content-type (大文字と小文字を区別しない) でヘッダー内の正確なコンテンツタイプを指定することで、上書きできます。

例

次の JSON の例では、HTTP アクションを使用して AWS IoT ルールを定義します。

```
{
    "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 アクションの再試行ロジック

AWS IoT ルールエンジンは、次のルールに従って HTTP アクションを再試行します。

- ルールエンジンは、1 回以上メッセージの送信を試みます。
- ルールエンジンは、最大で 2 回再試行します。最大試行回数は 3 回です。
- 次の場合、ルールエンジンは再試行を試行しません。
  - 前回の試行で 16,384 バイトを超える応答が提供された場合。
  - ダウンストリームウェブサービスまたはアプリケーションが試行後に TCP 接続を閉じた場合。
  - 再試行を含むリクエストを完了するための合計時間が、リクエストタイムアウト制限を超えた場合。
  - リクエストが 429、500～599 以外の HTTP ステータスコードを返した場合。

> **Note**
>
> 再試行には、標準データ転送コストが適用されます。

関連情報

- HTTP アクションメッセージのバッチ処理
- HTTP トピックルール送信先の使用
- ブログのモノのインターネット AWSで、 からウェブサービス AWS IoT Core にデータを直接ルーティングする

HTTP アクションメッセージのバッチ処理

バッチ処理を使用して、1 つのリクエストで複数の HTTP アクションメッセージを送信できます。

概要:

バッチ処理を使用すると、 AWS IoT Core ルールエンジンから HTTP エンドポイントにバッチでメッセージを送信できます。この機能は、HTTP アクションの実行数を減らすことでコストを削減し、新しい接続の確立に関連するオーバーヘッドを減らすことで効率を向上させるのに役立ちます。

> **Note**
>
> バッチ処理された HTTP アクションは 1 つのアクションとして計測されます。 AWS IoT Core ルールエンジンからダウンストリームサービスに出力されるバッチ処理されたアウトバ

ウンドペイロードのサイズに基づいて、5 kiB 単位で計測されます。詳細については、AWS IoT Core 料金表ページ を参照してください。

IoT ルールアクションの定義でバッチ処理を有効にすると、次のパラメータを設定できるようになります。

`maxBatchOpenMs`

送信メッセージが他のメッセージがバッチを作成するのを待機する最大時間 (ミリ秒単位) 。設定が大きいほど、バッチ処理された HTTP アクションのレイテンシーが長くなります。

最小値: 5 ミリ秒。最大値: 200 ミリ秒。

デフォルト値: 20 ミリ秒

置換テンプレートをサポート: いいえ

`maxBatchSize`

1 回の IoT ルールアクション実行でまとめてバッチ処理されるメッセージの最大数。

最小値: 2 メッセージ。最大値: 10 メッセージ

デフォルト値: 10 メッセージ

置換テンプレートをサポート: いいえ

`maxBatchSizeBytes`

メッセージバッチの最大サイズ、バイト単位。

最小値: 100 バイト。最大値: 131,072 バイト

デフォルト値: 5120 バイト

置換テンプレートをサポート: いいえ

Important

複数のバッチパラメータを指定すると、最初の制限に達するとバッチ処理が完了します。たとえば、最大バッチオープン時間として 100 ミリ秒、最大バッチサイズとして 5 kiB を指定

し、ルールエンジンが 100 ミリ秒以内に 2 kiB のみをバッチ処理する場合、2 kiB バッチが作成されて送信されます。

バッチでの HTTP ヘッダーの使用

HTTP アクションでヘッダーを使用する場合、バッチ処理されたリクエストは、バッチに追加された最後のメッセージ (最後に発行したメッセージではない) のヘッダー値を使用します。次のいずれかのヘッダー値を使用することをお勧めします。

- バッチ内のすべてのメッセージで同一
- すべてのメッセージに適用可能 (認証情報など)

ヘッダーは HTTP リクエストとともに送信され、メッセージ本文の一部ではありません。

> **Note**
>
> バッチ処理が有効になっている場合:
>
> - バッチ処理されたリクエストには、バッチが JSON 配列として送信されるため、`Content-Type: application/json`ヘッダーが自動的に含まれます。
> - バッチ内の最後のメッセージが、発行した最後のメッセージであることを保証することはできません。これは、バッチに書き込まれた最後のメッセージです。

ペイロードの例

次の例は、HTTP エンドポイントに送信されるバッチ処理されたメッセージペイロードの構造を示しています。

```
[
  {
    "user_id": "user1",
    "steps_today": 1000
  },
  {
    "user_id": "user2",
    "steps_today": 21000
  },
  {
```

```
    "user_id": "user8",
    "steps_today": 1500
  },
  ...
]
```

制限事項

バッチ処理の制限は次のとおりです。

- AWS IoT Core は、メッセージの全体的な順序付けを保証しません。バッチ処理は各ホストでローカルに実行されるため、バッチ内のメッセージが受信された順序とは異なる順序で処理される可能性があります。
- AWS IoT Core は、受信者側でメッセージ処理をサポートしていません。ダウンストリームサービスがデータをバッチで受け入れて処理するように設定されていることを確認する責任があります。
- クロスアカウントバッチ処理は、メッセージが同じリソース識別子 (HTTP URL またはリソース ARN) 宛てであってもサポートされていません。
- AWS IoT Core は、バッチサイズが指定した設定を満たすことを保証するものではありません。バッチは、タイミングとメッセージフローに基づいて設定された制限よりも小さい場合があります。
- バッチ処理が有効になっている場合、バイナリペイロード (非 UTF-8 データ) はサポートされていません。UTF-8 テキストペイロード (JSON など) のみが受け入れられます。バイナリデータを送信するには、base64 でエンコードしてから HTTP アクションに送信し、受信エンドポイントでデコードします。たとえば、IoT ルールのエンコード関数を使用してバイナリペイロードをエンコードできます。または、IoT デバイスでバイナリペイロードをエンコードして公開することもできます AWS IoT Core。

バッチ処理のエラーアクション

エラーアクション定義で別のバッチロジックを定義することはできません。ただし、プライマリアクションでバッチ処理ロジックを定義している場合、エラーアクションはバッチ処理をサポートします。

バッチリクエストが失敗すると、 AWS IoT Core Rules エンジンは HTTP アクションの再試行ロジックに従います。最後の再試行後、失敗したバッチ全体に対してエラーアクションが呼び出されます。

バッチ処理が有効になっているエラーメッセージの例を次に示します。

```
{
```

```
   "ruleName": "FailedTopicRule",
   "topic": "topic/rulesengine",
   "payloadsWithMetadata": [
       {
           "id": 1,
           "cloudwatchTraceId": "bebd6d93-6d4a-899e-9e40-56e82252d2be",
           "clientId": "Test",
           "sourceIp": "10.0.0.0",
           "base64OriginalPayload":
"eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
       },
       {
           "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**
>
> バッチ処理されたアクションの失敗は、より大きなエラーアクションペイロードも生成するため、サイズによるエラーアクションの失敗の可能性が高くなる可能性があります。ErrorActionFailure メトリクスを使用して、エラーアクションの失敗をモニタリングできます。詳細については「ルールアクションのメトリクス」を参照してください。

を使用した HTTP アクションメッセージのバッチ処理 AWS CLI

バッチ処理によるルールアクションの作成または更新

1. 適切な AWS CLI コマンドを使用して、ルールを作成または更新します。
   - 新しいルールを作成するには、create-topic-rule コマンドを使用します。

     ```
     aws iot create-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

   - 既存のルールを更新するには、 replace-topic-rule コマンドを使用します。

     ```
     aws iot replace-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

2. トピックルールペイロードで enableBatching パラメータを true に設定して、バッチ処理機能を有効にします。

```
{
        "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. バッチ処理パラメータを設定します。すべてのバッチパラメータを指定する必要はありません。1、2、または 3 つのバッチパラメータすべてを指定できます。バッチパラメータを指定しない場合、ルールエンジンはそのパラメータをデフォルト値で更新します。バッチパラメータとそのデフォルト値の詳細については、「HTTP パラメータ」を参照してください。

HTTP トピックルール送信先の使用

HTTP トピックルールの送信先は、ルールエンジンがトピックルールからデータをルーティングできる ウェブサービスです。 AWS IoT Core リソースは、 のウェブサービスを記述します AWS IoT。トピックルールの送信先のリソースは、異なるルールで共有できます。

AWS IoT Core が別のウェブサービスにデータを送信する前に、サービスのエンドポイントにアクセスできることを確認する必要があります。

この章の内容:

- HTTP トピックルールの送信先の概要
- HTTP トピックルールの送信先の管理
- トピックルールの送信先の HTTPS エンドポイントでサポートされている認証機関

HTTP トピックルールの送信先の概要

HTTP トピックルールの送信先とは、確認 URL と 1 つ、または複数のデータ収集 URL をサポートする ウェブサービスを指します。HTTP トピックルールの送信先リソースには、Web サービスの確認 URL が含まれています。HTTP トピックルールアクションを設定するときは、データを受信するエンドポイントの実際のURL と、ウェブサービスの確認 URL を指定します。送信先が確認されると、トピックルールは、SQL ステートメントの結果を HTTPS エンドポイントに送信します (確認 URL ではない)。

HTTP トピックルール送信先の状態は、以下のいずれかになります :

有効

送信先は確認済みで、ルールアクションによって使用できます。送信先をルールで使用するには、送信先は、ENABLEDの状態である必要があります。有効にできるのは、「DISABLED」ステータスの送信先のみです。

無効

送信先は確認されましたが、ルールアクションでは使用できません。これは、確認プロセスを再度実行することなく、エンドポイントへのトラフィックを一時的に防止する場合に便利です。無効にできるのは、有効ステータスの送信先のみです。

IN_PROGRESS

送信先の確認は進行中です。

ERROR

送信先の確認がタイムアウトしました。

HTTP トピックルール送信先が確認されて有効化されると、アカウントのどのルールでも使用できるようになります。

次のセクションでは、HTTP トピックルールの送信先に関する一般的なアクションについて説明します。

HTTP トピックルールの送信先の管理

次のオペレーションを使用して、HTTP トピックルールの送信先を管理できます。

このトピックの内容

- HTTP トピックルールの送信先の作成
- HTTP トピックルールの送信先の確認
- 新しい確認リクエストの送信
- トピックルールの送信先の無効化

HTTP トピックルールの送信先の作成

HTTP トピックルール送信先は、CreateTopicRuleDestination オペレーションを呼び出す、または AWS IoT コンソールを使用することによって作成します。

送信先を作成すると、 は確認 URL に確認リクエスト AWS IoT を送信します。確認リクエストの形式は次のとおりです。

```
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"
}
```

確認リクエストの内容には、以下の情報が含まれます。

arn

確認するトピックルール送信先の Amazon リソースネーム (ARN)。

confirmationToken

によって送信された確認トークン AWS IoT Core。この例のトークンは切り捨てられます。トークンは長くなります。 AWS IoT Coreで目的地を確認するには、このトークンが必要です。

enableUrl

トピックルールの送信先を確認するために参照する URL。

messageType

メッセージのタイプ。

HTTP トピックルールの送信先の確認

エンドポイントの確認プロセスを完了するには、 AWS CLIを使用している場合、確認 URL が確認リクエストを受信した後、次のいずれかの手順を実施する必要があります。

1. 送信先がメッセージを受信する意思があることを確認する

   トピックルールの送信先が IoT メッセージを受信する意思があることを確認するには、確認リクエストで `enableUrl` を呼び出すか、`ConfirmTopicRuleDestination` API オペレーションを実行して確認リクエストから `confirmationToken` を渡します。

2. トピックルールのステータスを有効に設定する

   送信先がメッセージを受信できることを確認したら、`UpdateTopicRuleDestination` API オペレーションを実行してトピックルールのステータスを `ENABLED` に設定する必要があります。

AWS IoT コンソールを使用している場合は、 をコピーconfirmationTokenし、 AWS IoT コンソールの送信先の確認ダイアログに貼り付けます。その後、トピックルールを有効にできます。

新しい確認リクエストの送信

送信先の新しい確認メッセージをアクティブ化するには、UpdateTopicRuleDestination を呼び出 して、トピックルールの送信先のステータスを IN_PROGRESS に設定します。

新しい確認リクエストを送信した後、確認プロセスを繰り返します。

トピックルールの送信先の無効化

送信先を無効にするには、UpdateTopicRuleDestinationを呼び出して、トピックルールの送信先のステータスを DISABLED に設定します。無効状態のトピックルールは、新しい確認要求を送信しなくても、再度有効にすることができます。

トピックルールの送信先を削除するには、DeleteTopicRuleDestination を呼び出します。

トピックルールの送信先の HTTPS エンドポイントでサポートされている認証機関

トピックルールの送信先の HTTPS エンドポイントでは、次の証明機関がサポートされています。これらのサポートされている証明機関のいずれかを選択できます。署名は参考用です。自己署名証明書は機能しないため、使用できないことに注意してください。

このトピックの改善にご協力ください
フィードバックをお寄せください。

```
Alias name: swisssignplatinumg2ca
Certificate fingerprints:
  MD5:  C9:98:27:77:28:1E:3D:0E:15:3C:84:00:B8:85:03:E6
  SHA1: 56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
  SHA256:
 3B:22:2E:56:67:11:E9:92:30:0D:C0:B1:5A:B9:47:3D:AF:DE:F8:C8:4D:0C:EF:7D:33:17:B4:C1:82:1D:14:3

Alias name: hellenicacademicandresearchinstitutionsrootca2011
Certificate fingerprints:
  MD5:  73:9F:4C:4B:73:5B:79:E9:FA:BA:1C:EF:6E:CB:D5:C9
  SHA1: FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
  SHA256:
 BC:10:4F:15:A4:8B:E7:09:DC:A5:42:A7:E1:D4:B9:DF:6F:05:45:27:E8:02:EA:A9:2D:59:54:44:25:8A:FE:7

Alias name: teliasonerarootcav1
Certificate fingerprints:
  MD5:  37:41:49:1B:18:56:9A:26:F5:AD:C2:66:FB:40:A5:4C
```

```
  SHA1: 43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
  SHA256:
 DD:69:36:FE:21:F8:F0:77:C1:23:A1:A5:21:C1:22:24:F7:22:55:B7:3E:03:A7:26:06:93:E8:A2:4B:0F:A3:8

Alias name: geotrustprimarycertificationauthority
Certificate fingerprints:
  MD5:  02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  SHA256:
 37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6

Alias name: trustisfpsrootca
Certificate fingerprints:
  MD5:  30:C9:E7:1E:6B:E6:14:EB:65:B2:16:69:20:31:67:4D
  SHA1: 3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
  SHA256:
 C1:B4:82:99:AB:A5:20:8F:E9:63:0A:CE:55:CA:68:A0:3E:DA:5A:51:9C:88:02:A0:D3:A6:73:BE:8F:8E:55:7

Alias name: quovadisrootca3g3
Certificate fingerprints:
  MD5:  DF:7D:B9:AD:54:6F:68:A1:DF:89:57:03:97:43:B0:D7
  SHA1: 48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
  SHA256:
 88:EF:81:DE:20:2E:B0:18:45:2E:43:F8:64:72:5C:EA:5F:BD:1F:C2:D9:D2:05:73:07:09:C5:D8:B8:69:0F:4

Alias name: buypassclass2ca
Certificate fingerprints:
  MD5:  46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  SHA256:
 9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:4

Alias name: secureglobalca
Certificate fingerprints:
  MD5:  CF:F4:27:0D:D4:ED:DC:65:16:49:6D:3D:DA:BF:6E:DE
  SHA1: 3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
  SHA256:
 42:00:F5:04:3A:C8:59:0E:BB:52:7D:20:9E:D1:50:30:29:FB:CB:D4:1C:A1:B5:06:EC:27:F1:5A:DE:7D:AC:6

Alias name: chunghwaepkirootca
Certificate fingerprints:
  MD5:  1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
```

```
  SHA256:
 C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D

Alias name: verisignclass2g2ca
Certificate fingerprints:
  MD5:  2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
  SHA1: B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
  SHA256:
 3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21:74:2E:AC:2B:75:C2:0F:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A

Alias name: szafirrootca2
Certificate fingerprints:
  MD5:  11:64:C1:89:B0:24:B1:8C:B1:07:7E:89:9E:51:9E:99
  SHA1: E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
  SHA256:
 A1:33:9D:33:28:1A:0B:56:E5:57:D3:D3:2B:1C:E7:F9:36:7E:B0:94:BD:5F:A7:2A:7E:50:04:C8:DE:D7:CA:F

Alias name: quovadisrootca1g3
Certificate fingerprints:
  MD5:  A4:BC:5B:3F:FE:37:9A:FA:64:F0:E2:FA:05:3D:0B:AB
  SHA1: 1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
  SHA256:
 8A:86:6F:D1:B2:76:B5:7E:57:8E:92:1C:65:82:8A:2B:ED:58:E9:F2:F2:88:05:41:34:B7:F1:F4:BF:C9:CC:7

Alias name: utndatacorpsgcca
Certificate fingerprints:
  MD5:  B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06
  SHA1: 58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
  SHA256:
 85:FB:2F:91:DD:12:27:5A:01:45:B6:36:53:4F:84:02:4A:D6:8B:69:B8:EE:88:68:4F:F7:11:37:58:05:B3:4

Alias name: autoridaddecertificacionfirmaprofesionalcifa62634068
Certificate fingerprints:
  MD5:  73:3A:74:7A:EC:BB:A3:96:A6:C2:E4:E2:C8:9B:C0:C3
  SHA1: AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
  SHA256:
 04:04:80:28:BF:1F:28:64:D4:8F:9A:D4:D8:32:94:36:6A:82:88:56:55:3F:3B:14:30:3F:90:14:7F:5D:40:E

Alias name: securesignrootca11
Certificate fingerprints:
  MD5:  B7:52:74:E2:92:B4:80:93:F2:75:E4:CC:D7:F2:EA:26
  SHA1: 3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
  SHA256:
 BF:0F:EE:FB:9E:3A:58:1A:D5:F9:E9:DB:75:89:98:57:43:D2:61:08:5C:4D:31:4F:6F:5D:72:59:AA:42:16:1
```

```
Alias name: amazon-ca-g4-acm2
Certificate fingerprints:
  MD5:  B2:F1:03:2B:93:64:05:80:B8:A8:17:36:B9:1B:52:3C
  SHA1: A7:E6:45:32:1F:7A:B7:AD:C0:70:EA:73:5F:AB:ED:C3:DA:B4:D0:C8
  SHA256:
 D7:A8:7C:69:95:D0:E2:04:2A:32:70:A7:E2:87:FE:A7:E8:F4:C1:70:62:F7:90:C3:EB:BB:53:F2:AC:39:26:B

Alias name: isrgrootx1
Certificate fingerprints:
  MD5:  0C:D2:F9:E0:DA:17:73:E9:ED:86:4D:A5:E3:70:E7:4E
  SHA1: CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
  SHA256:
 96:BC:EC:06:26:49:76:F3:74:60:77:9A:CF:28:C5:A7:CF:E8:A3:C0:AA:E1:1A:8F:FC:EE:05:C0:BD:DF:08:C

Alias name: amazon-ca-g4-acm1
Certificate fingerprints:
  MD5:  E2:F1:18:19:61:5C:43:E0:D4:A8:5D:0B:FA:7C:89:1B
  SHA1: F2:0D:28:B6:29:C2:2C:5E:84:05:E6:02:4D:97:FE:8F:A0:84:93:A0
  SHA256:
 B0:11:A4:F7:29:6C:74:D8:2B:F5:62:DF:87:D7:28:C7:1F:B5:8C:F4:E6:73:F2:78:FC:DA:F3:FF:83:A6:8C:8

Alias name: etugracertificationauthority
Certificate fingerprints:
  MD5:  B8:A1:03:63:B0:BD:21:71:70:8A:6F:13:3A:BB:79:49
  SHA1: 51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
  SHA256:
 B0:BF:D5:2B:B0:D7:D9:BD:92:BF:5D:4D:C1:3D:A2:55:C0:2C:54:2F:37:83:65:EA:89:39:11:F5:5E:55:F2:3

Alias name: geotrustuniversalca2
Certificate fingerprints:
  MD5:  34:FC:B8:D0:36:DB:9E:14:B3:C2:F2:DB:8F:E4:94:C7
  SHA1: 37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
  SHA256:
 A0:23:4F:3B:C8:52:7C:A5:62:8E:EC:81:AD:5D:69:89:5D:A5:68:0D:C9:1D:1C:B8:47:7F:33:F8:78:B9:5B:0

Alias name: digicertglobalrootca
Certificate fingerprints:
  MD5:  79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
  SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
  SHA256:
 43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:7F:89:34:A4:43:C7:01:6

Alias name: staatdernederlandenevrootca
```

```
Certificate fingerprints:
  MD5:  FC:06:AF:7B:E8:1A:F1:9A:B4:E8:D2:70:1F:C0:F5:BA
  SHA1: 76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
  SHA256:
 4D:24:91:41:4C:FE:95:67:46:EC:4C:EF:A6:CF:6F:72:E2:8A:13:29:43:2F:9D:8A:90:7A:C4:CB:5D:AD:C1:5

Alias name: utnuserfirstclientauthemailca
Certificate fingerprints:
  MD5:  D7:34:3D:EF:1D:27:09:28:E1:31:02:5B:13:2B:DD:F7
  SHA1: B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
  SHA256:
 43:F2:57:41:2D:44:0D:62:74:76:97:4F:87:7D:A8:F1:FC:24:44:56:5A:36:7A:E6:0E:DD:C2:7A:41:25:31:A

Alias name: actalisauthenticationrootca
Certificate fingerprints:
  MD5:  69:C1:0D:4F:07:A3:1B:C3:FE:56:3D:04:BC:11:F6:A6
  SHA1: F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
  SHA256:
 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:6

Alias name: amazonrootca4
Certificate fingerprints:
  MD5:  89:BC:27:D5:EB:17:8D:06:6A:69:D5:FD:89:47:B4:CD
  SHA1: F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
  SHA256:
 E3:5D:28:41:9E:D0:20:25:CF:A6:90:38:CD:62:39:62:45:8D:A5:C6:95:FB:DE:A3:C2:2B:0B:FB:25:89:70:9

Alias name: amazonrootca3
Certificate fingerprints:
  MD5:  A0:D4:EF:0B:F7:B5:D8:49:95:2A:EC:F5:C4:FC:81:87
  SHA1: 0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
  SHA256:
 18:CE:6C:FE:7B:F1:4E:60:B2:E3:47:B8:DF:E8:68:CB:31:D0:2E:BB:3A:DA:27:15:69:F5:03:43:B4:6D:B3:A

Alias name: amazonrootca2
Certificate fingerprints:
  MD5:  C8:E5:8D:CE:A8:42:E2:7A:C0:2A:5C:7C:9E:26:BF:66
  SHA1: 5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
  SHA256:
 1B:A5:B2:AA:8C:65:40:1A:82:96:01:18:F8:0B:EC:4F:62:30:4D:83:CE:C4:71:3A:19:C3:9C:01:1E:A4:6D:B

Alias name: amazonrootca1
Certificate fingerprints:
  MD5:  43:C6:BF:AE:EC:FE:AD:2F:18:C6:88:68:30:FC:C8:E6
```

```
  SHA1: 8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
  SHA256:
 8E:CD:E6:88:4F:3D:87:B1:12:5B:A3:1A:C3:FC:B1:3D:70:16:DE:7F:57:CC:90:4F:E1:CB:97:C6:AE:98:19:6

Alias name: affirmtrustpremium
Certificate fingerprints:
  MD5:  C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  SHA256:
 70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9

Alias name: keynectisrootca
Certificate fingerprints:
  MD5:  CC:4D:AE:FB:30:6B:D8:38:FE:50:EB:86:61:4B:D2:26
  SHA1: 9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
  SHA256:
 42:10:F1:99:49:9A:9A:C3:3C:8D:E0:2B:A6:DB:AA:14:40:8B:DD:8A:6E:32:46:89:C1:92:2D:06:97:15:A3:3

Alias name: equifaxsecureglobalebusinessca1
Certificate fingerprints:
  MD5:  51:F0:2A:33:F1:F5:55:39:07:F2:16:7A:47:C7:5D:63
  SHA1: 3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
  SHA256:
 86:AB:5A:65:71:D3:32:9A:BC:D2:E4:E6:37:66:8B:A8:9C:73:1E:C2:93:B6:CB:A6:0F:71:63:40:A0:91:CE:A

Alias name: affirmtrustpremiumca
Certificate fingerprints:
  MD5:  C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  SHA256:
 70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9

Alias name: baltimorecodesigningca
Certificate fingerprints:
  MD5:  90:F5:28:49:56:D1:5D:2C:B0:53:D4:4B:EF:6F:90:22
  SHA1: 30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
  SHA256:
 A9:15:45:DB:D2:E1:9C:4C:CD:F9:09:AA:71:90:0D:18:C7:35:1C:89:B3:15:F0:F1:3D:05:C1:3A:8F:FB:46:8

Alias name: gdcatrustauthr5root
Certificate fingerprints:
  MD5:  63:CC:D9:3D:34:35:5C:6F:53:A3:E2:08:70:48:1F:B4
  SHA1: 0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
```

```
  SHA256:
 BF:FF:8F:D0:44:33:48:7D:6A:8A:A6:0C:1A:29:76:7A:9F:C2:BB:B0:5E:42:0F:71:3A:13:B9:92:89:1D:38:9

Alias name: certinomisrootca
Certificate fingerprints:
  MD5:  14:0A:FD:8D:A8:28:B5:38:69:DB:56:7E:61:22:03:3F
  SHA1: 9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
  SHA256:
 2A:99:F5:BC:11:74:B7:3C:BB:1D:62:08:84:E0:1C:34:E5:1C:CB:39:78:DA:12:5F:0E:33:26:88:83:BF:41:5

Alias name: verisignclass3publicprimarycertificationauthorityg5
Certificate fingerprints:
  MD5:  CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  SHA256:
 9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:D

Alias name: verisignclass3publicprimarycertificationauthorityg4
Certificate fingerprints:
  MD5:  3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  SHA256:
 69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:7

Alias name: verisignclass3publicprimarycertificationauthorityg3
Certificate fingerprints:
  MD5:  CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  SHA256:
 EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:4

Alias name: swisssignsilverg2ca
Certificate fingerprints:
  MD5:  E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  SHA256:
 BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D

Alias name: swisssignsilvercag2
Certificate fingerprints:
  MD5:  E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  SHA256:
 BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D
```

```
Alias name: atostrustedroot2011
Certificate fingerprints:
  MD5:  AE:B9:C4:32:4B:AC:7F:5D:66:CC:77:94:BB:2A:77:56
  SHA1: 2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
  SHA256:
 F3:56:BE:A2:44:B7:A9:1E:B3:5D:53:CA:9A:D7:86:4A:CE:01:8E:2D:35:D5:F8:F9:6D:DF:68:A6:F4:1A:A4:7

Alias name: comodoecccertificationauthority
Certificate fingerprints:
  MD5:  7C:62:FF:74:9D:31:53:5E:68:4A:D5:78:AA:1E:BF:23
  SHA1: 9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
  SHA256:
 17:93:92:7A:06:14:54:97:89:AD:CE:2F:8F:34:F7:F0:B6:6D:0F:3A:E3:A3:B8:4D:21:EC:15:DB:BA:4F:AD:C

Alias name: securetrustca
Certificate fingerprints:
  MD5:  DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
  SHA1: 87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
  SHA256:
 F1:C1:B5:0A:E5:A2:0D:D8:03:0E:C9:F6:BC:24:82:3D:D3:67:B5:25:57:59:B4:E7:1B:61:FC:E9:F7:37:5D:7

Alias name: soneraclass1ca
Certificate fingerprints:
  MD5:  33:B7:84:F5:5F:27:D7:68:27:DE:14:DE:12:2A:ED:6F
  SHA1: 07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
  SHA256:
 CD:80:82:84:CF:74:6F:F2:FD:6E:B5:8A:A1:D5:9C:4A:D4:B3:CA:56:FD:C6:27:4A:89:26:A7:83:5F:32:31:3

Alias name: cadisigrootr2
Certificate fingerprints:
  MD5:  26:01:FB:D8:27:A7:17:9A:45:54:38:1A:43:01:3B:03
  SHA1: B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
  SHA256:
 E2:3D:4A:03:6D:7B:70:E9:F5:95:B1:42:20:79:D2:B9:1E:DF:BB:1F:B6:51:A0:63:3E:AA:8A:9D:C5:F8:07:0

Alias name: cadisigrootr1
Certificate fingerprints:
  MD5:  BE:EC:11:93:9A:F5:69:21:BC:D7:C1:C0:67:89:CC:2A
  SHA1: 8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
  SHA256:
 F9:6F:23:F4:C3:E7:9C:07:7A:46:98:8D:5A:F5:90:06:76:A0:F0:39:CB:64:5D:D1:75:49:B2:16:C8:24:40:C

Alias name: verisignclass3g5ca
```

```
Certificate fingerprints:
  MD5:  CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  SHA256:
 9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:D

Alias name: utnuserfirsthardwareca
Certificate fingerprints:
  MD5:  4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
  SHA1: 04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
  SHA256:
 6E:A5:47:41:D0:04:66:7E:ED:1B:48:16:63:4A:A3:A7:9E:6E:4B:96:95:0F:82:79:DA:FC:8D:9B:D8:81:21:3

Alias name: addtrustqualifiedca
Certificate fingerprints:
  MD5:  27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
  SHA1: 4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
  SHA256:
 80:95:21:08:05:DB:4B:BC:35:5E:44:28:D8:FD:6E:C2:CD:E3:AB:5F:B9:7A:99:42:98:8E:B8:F4:DC:D0:60:1

Alias name: verisignclass3g3ca
Certificate fingerprints:
  MD5:  CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  SHA256:
 EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:4

Alias name: thawtepersonalfreemailca
Certificate fingerprints:
  MD5:  53:4B:1D:17:58:58:1A:30:A1:90:F8:6E:5C:F2:CF:65
  SHA1: E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
  SHA256:
 5B:38:BD:12:9E:83:D5:A0:CA:D2:39:21:08:94:90:D5:0D:4A:AE:37:04:28:F8:DD:FF:FF:FA:4C:15:64:E1:8

Alias name: certplusclass3pprimaryca
Certificate fingerprints:
  MD5:  E1:4B:52:73:D7:1B:DB:93:30:E5:BD:E4:09:6E:BE:FB
  SHA1: 21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
  SHA256:
 CC:C8:94:89:37:1B:AD:11:1C:90:61:9B:EA:24:0A:2E:6D:AD:D9:9F:9F:6E:1D:4D:41:E5:8E:D6:DE:3D:02:8

Alias name: swisssigngoldg2ca
Certificate fingerprints:
  MD5:  24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
```

```
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  SHA256:
 62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:9

Alias name: swisssigngoldcag2
Certificate fingerprints:
  MD5:  24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  SHA256:
 62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:9

Alias name: dtrustrootclass3ca22009
Certificate fingerprints:
  MD5:  CD:E0:25:69:8D:47:AC:9C:89:35:90:F7:FD:51:3D:2F
  SHA1: 58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
  SHA256:
 49:E7:A4:42:AC:F0:EA:62:87:05:00:54:B5:25:64:B6:50:E4:F4:9E:42:E3:48:D6:AA:38:E0:39:E9:57:B1:C

Alias name: acraizfnmtrcm
Certificate fingerprints:
  MD5:  E2:09:04:B4:D3:BD:D1:A0:14:FD:1A:D2:47:C4:57:1D
  SHA1: EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
  SHA256:
 EB:C5:57:0C:29:01:8C:4D:67:B1:AA:12:7B:AF:12:F7:03:B4:61:1E:BC:17:B7:DA:B5:57:38:94:17:9B:93:F

Alias name: securitycommunicationevrootca1
Certificate fingerprints:
  MD5:  22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  SHA256:
 A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:3

Alias name: starfieldclass2ca
Certificate fingerprints:
  MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:5

Alias name: opentrustrootcag3
Certificate fingerprints:
  MD5:  21:37:B4:17:16:92:7B:67:46:70:A9:96:D7:A8:13:24
  SHA1: 6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
```

```
  SHA256:
 B7:C3:62:31:70:6E:81:07:8C:36:7C:B8:96:19:8F:1E:32:08:DD:92:69:49:DD:8F:57:09:A4:10:F7:5B:62:9

Alias name: opentrustrootcag2
Certificate fingerprints:
  MD5:  57:24:B6:59:24:6B:AE:C8:FE:1C:0C:20:F2:C0:4E:EB
  SHA1: 79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
  SHA256:
 27:99:58:29:FE:6A:75:15:C1:BF:E8:48:F9:C4:76:1D:B1:6C:22:59:29:25:7B:F4:0D:08:94:F2:9E:A8:BA:F

Alias name: buypassclass2rootca
Certificate fingerprints:
  MD5:  46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  SHA256:
 9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:4

Alias name: opentrustrootcag1
Certificate fingerprints:
  MD5:  76:00:CC:81:29:CD:55:5E:88:6A:7A:2E:F7:4D:39:DA
  SHA1: 79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
  SHA256:
 56:C7:71:28:D9:8C:18:D9:1B:4C:FD:FF:BC:25:EE:91:03:D4:75:8E:A2:AB:AD:82:6A:90:F3:45:7D:46:0E:B

Alias name: globalsignr2ca
Certificate fingerprints:
  MD5:  94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  SHA256:
 CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9

Alias name: buypassclass3rootca
Certificate fingerprints:
  MD5:  3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  SHA256:
 ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4

Alias name: ecacc
Certificate fingerprints:
  MD5:  EB:F5:9D:29:0D:61:F9:42:1F:7C:C2:BA:6D:E3:15:09
  SHA1: 28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
  SHA256:
 88:49:7F:01:60:2F:31:54:24:6A:E2:8C:4D:5A:EF:10:F1:D8:7E:BB:76:62:6F:4A:E0:B7:F9:5B:A7:96:87:9
```

```
Alias name: epkirootcertificationauthority
Certificate fingerprints:
  MD5:  1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  SHA256:
 C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D

Alias name: verisignclass1g2ca
Certificate fingerprints:
  MD5:  DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
  SHA1: 27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
  SHA256:
 34:1D:E9:8B:13:92:AB:F7:F4:AB:90:A9:60:CF:25:D4:BD:6E:C6:5B:9A:51:CE:6E:D0:67:D0:0E:C7:CE:9B:7

Alias name: certigna
Certificate fingerprints:
  MD5:  AB:57:A6:5B:7D:42:82:19:B5:D8:58:26:28:5E:FD:FF
  SHA1: B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
  SHA256:
 E3:B6:A2:DB:2E:D7:CE:48:84:2F:7A:C5:32:41:C7:B7:1D:54:14:4B:FB:40:C1:1F:3F:1D:0B:42:F5:EE:A1:2

Alias name: camerfirmaglobalchambersignroot
Certificate fingerprints:
  MD5:  C5:E6:7B:BF:06:D0:4F:43:ED:C4:7A:65:8A:FB:6B:19
  SHA1: 33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
  SHA256:
 EF:3C:B4:17:FC:8E:BF:6F:97:87:6C:9E:4E:CE:39:DE:1E:A5:FE:64:91:41:D1:02:8B:7D:11:C0:B2:29:8C:E

Alias name: cfcaevroot
Certificate fingerprints:
  MD5:  74:E1:B6:ED:26:7A:7A:44:30:33:94:AB:7B:27:81:30
  SHA1: E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
  SHA256:
 5C:C3:D7:8E:4E:1D:5E:45:54:7A:04:E6:87:3E:64:F9:0C:F9:53:6D:1C:CC:2E:F8:00:F3:55:C4:C5:FD:70:F

Alias name: soneraclass2rootca
Certificate fingerprints:
  MD5:  A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  SHA256:
 79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:2

Alias name: certumtrustednetworkca
```

```
Certificate fingerprints:
  MD5:  D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
  SHA1: 07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
  SHA256:
 5C:58:46:8D:55:F5:8E:49:7E:74:39:82:D2:B5:00:10:B6:D1:65:37:4A:CF:83:A7:D4:A3:2D:B7:68:C4:40:8

Alias name: securitycommunicationrootca2
Certificate fingerprints:
  MD5:  6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F

Alias name: globalsigneccrootcar5
Certificate fingerprints:
  MD5:  9F:AD:3B:1C:02:1E:8A:BA:17:74:38:81:0C:A2:BC:08
  SHA1: 1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
  SHA256:
 17:9F:BC:14:8A:3D:D0:0F:D2:4E:A1:34:58:CC:43:BF:A7:F5:9C:81:82:D7:83:A5:13:F6:EB:EC:10:0C:89:2

Alias name: globalsigneccrootcar4
Certificate fingerprints:
  MD5:  20:F0:27:68:D1:7E:A0:9D:0E:E6:2A:CA:DF:5C:89:8E
  SHA1: 69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
  SHA256:
 BE:C9:49:11:C2:95:56:76:DB:6C:0A:55:09:86:D7:6E:3B:A0:05:66:7C:44:2C:97:62:B4:FB:B7:73:DE:22:8

Alias name: chambersofcommerceroot2008
Certificate fingerprints:
  MD5:  5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  SHA256:
 06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C

Alias name: pscprocert
Certificate fingerprints:
  MD5:  E6:24:E9:12:01:AE:0C:DE:8E:85:C4:CE:A3:12:DD:EC
  SHA1: 70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
  SHA256:
 3C:FC:3C:14:D1:F6:84:FF:17:E3:8C:43:CA:44:0C:00:B9:67:EC:93:3E:8B:FE:06:4C:A1:D7:2C:90:F2:AD:B

Alias name: thawteprimaryrootcag3
Certificate fingerprints:
  MD5:  FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31
```

```
  SHA1: F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
  SHA256:
 4B:03:F4:58:07:AD:70:F2:1B:FC:2C:AE:71:C9:FD:E4:60:4C:06:4C:F5:FF:B6:86:BA:E5:DB:AA:D7:FD:D3:4

Alias name: quovadisrootca
Certificate fingerprints:
  MD5:  27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
  SHA1: DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
  SHA256:
 A4:5E:DE:3B:BB:F0:9C:8A:E1:5C:72:EF:C0:72:68:D6:93:A2:1C:99:6F:D5:1E:67:CA:07:94:60:FD:6D:88:7

Alias name: thawteprimaryrootcag2
Certificate fingerprints:
  MD5:  74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F
  SHA1: AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
  SHA256:
 A4:31:0D:50:AF:18:A6:44:71:90:37:2A:86:AF:AF:8B:95:1F:FB:43:1D:83:7F:1E:56:88:B4:59:71:ED:15:5

Alias name: deprecateditsecca
Certificate fingerprints:
  MD5:  A5:96:0C:F6:B5:AB:27:E5:01:C6:00:88:9E:60:33:E5
  SHA1: 12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
  SHA256:
 9A:59:DA:86:24:1A:FD:BA:A3:39:FA:9C:FD:21:6A:0B:06:69:4D:E3:7E:37:52:6B:BE:63:C8:BC:83:74:2E:C

Alias name: usertrustrsacertificationauthority
Certificate fingerprints:
  MD5:  1B:FE:69:D1:91:B7:19:33:A3:72:A8:0F:E1:55:E5:B5
  SHA1: 2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
  SHA256:
 E7:93:C9:B0:2F:D8:AA:13:E2:1C:31:22:8A:CC:B0:81:19:64:3B:74:9C:89:89:64:B1:74:6D:46:C3:D4:CB:D

Alias name: entrustrootcag2
Certificate fingerprints:
  MD5:  4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  SHA256:
 43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:3

Alias name: networksolutionscertificateauthority
Certificate fingerprints:
  MD5:  D3:F3:A6:16:C0:FA:6B:1D:59:B1:2D:96:4D:0E:11:2E
  SHA1: 74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
```

```
  SHA256:
 15:F0:BA:00:A3:AC:7A:F3:AC:88:4C:07:2B:10:11:A0:77:BD:77:C0:97:F4:01:64:B2:F8:59:8A:BD:83:86:0

Alias name: trustcenterclass4caii
Certificate fingerprints:
  MD5:  9D:FB:F9:AC:ED:89:33:22:F4:28:48:83:25:23:5B:E0
  SHA1: A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
  SHA256:
 32:66:96:7E:59:CD:68:00:8D:9D:D3:20:81:11:85:C7:04:20:5E:8D:95:FD:D8:4F:1C:7B:31:1E:67:04:FC:3

Alias name: oistewisekeyglobalrootgaca
Certificate fingerprints:
  MD5:  BC:6C:51:33:A7:E9:D3:66:63:54:15:72:1B:21:92:93
  SHA1: 59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
  SHA256:
 41:C9:23:86:6A:B4:CA:D6:B7:AD:57:80:81:58:2E:02:07:97:A6:CB:DF:4F:FF:78:CE:83:96:B3:89:37:D7:F

Alias name: verisignuniversalrootcertificationauthority
Certificate fingerprints:
  MD5:  8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  SHA256:
 23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3

Alias name: ttelesecglobalrootclass3ca
Certificate fingerprints:
  MD5:  CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  SHA256:
 FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:B

Alias name: starfieldservicesrootg2ca
Certificate fingerprints:
  MD5:  17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  SHA256:
 56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B

Alias name: addtrustexternalroot
Certificate fingerprints:
  MD5:  1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  SHA256:
 68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F
```

```
Alias name: turktrustelektroniksertifikahizmetsaglayicisih5
Certificate fingerprints:
  MD5:  DA:70:8E:F0:22:DF:93:26:F6:5F:9F:D3:15:06:52:4E
  SHA1: C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
  SHA256:
 49:35:1B:90:34:44:C1:85:CC:DC:5C:69:3D:24:D8:55:5C:B2:08:D6:A8:14:13:07:69:9F:4A:F0:63:19:9D:7

Alias name: camerfirmachambersca
Certificate fingerprints:
  MD5:  5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  SHA256:
 06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C

Alias name: certsignrootca
Certificate fingerprints:
  MD5:  18:98:C0:D6:E9:3A:FC:F9:B0:F5:0C:F7:4B:01:44:17
  SHA1: FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
  SHA256:
 EA:A9:62:C4:FA:4A:6B:AF:EB:E4:15:19:6D:35:1C:CD:88:8D:4F:53:F3:FA:8A:E6:D7:C4:66:A9:4E:60:42:B

Alias name: verisignuniversalrootca
Certificate fingerprints:
  MD5:  8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  SHA256:
 23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3

Alias name: geotrustuniversalca
Certificate fingerprints:
  MD5:  92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
  SHA1: E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
  SHA256:
 A0:45:9B:9F:63:B2:25:59:F5:FA:5D:4C:6D:B3:F9:F7:2F:F1:93:42:03:35:78:F0:73:BF:1D:1B:46:CB:B9:1



Alias name: luxtrustglobalroot2
Certificate fingerprints:
  MD5:  B2:E1:09:00:61:AF:F7:F1:91:6F:C4:AD:8D:5E:3B:7C
  SHA1: 1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
  SHA256:
 54:45:5F:71:29:C2:0B:14:47:C4:18:F9:97:16:8F:24:C5:8F:C5:02:3B:F5:DA:5B:E2:EB:6E:1D:D8:90:2E:D
```

```
Alias name: twcaglobalrootca
Certificate fingerprints:
  MD5:  F9:03:7E:CF:E6:9E:3C:73:7A:2A:90:07:69:FF:2B:96
  SHA1: 9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
  SHA256:
 59:76:90:07:F7:68:5D:0F:CD:50:87:2F:9F:95:D5:75:5A:5B:2B:45:7D:81:F3:69:2B:61:0A:98:67:2F:0E:1

Alias name: tubitakkamusmsslkoksertifikasisurum1
Certificate fingerprints:
  MD5:  DC:00:81:DC:69:2F:3E:2F:B0:3B:F6:3D:5A:91:8E:49
  SHA1: 31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
  SHA256:
 46:ED:C3:68:90:46:D5:3A:45:3F:B3:10:4A:B8:0D:CA:EC:65:8B:26:60:EA:16:29:DD:7E:86:79:90:64:87:1

Alias name: affirmtrustnetworkingca
Certificate fingerprints:
  MD5:  42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  SHA256:
 0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1

Alias name: affirmtrustcommercialca
Certificate fingerprints:
  MD5:  82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
  SHA256:
 03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A

Alias name: godaddyrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  SHA256:
 45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:D

Alias name: starfieldrootg2ca
Certificate fingerprints:
  MD5:  D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  SHA256:
 2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F
```

```
Alias name: dtrustrootclass3ca2ev2009
Certificate fingerprints:
  MD5:  AA:C6:43:2C:5E:2D:CD:C4:34:C0:50:4F:11:02:4F:B6
  SHA1: 96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
  SHA256:
 EE:C5:49:6B:98:8C:E9:86:25:B9:34:09:2E:EC:29:08:BE:D0:B0:F3:16:C2:D4:73:0C:84:EA:F1:F3:D3:48:8

Alias name: buypassclass3ca
Certificate fingerprints:
  MD5:  3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  SHA256:
 ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4

Alias name: verisignclass2g3ca
Certificate fingerprints:
  MD5:  F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
  SHA1: 61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
  SHA256:
 92:A9:D9:83:3F:E1:94:4D:B3:66:E8:BF:AE:7A:95:B6:48:0C:2D:6C:6C:2A:1B:E6:5D:42:36:B6:08:FC:A1:B

Alias name: digicerttrustedrootg4
Certificate fingerprints:
  MD5:  78:F2:FC:AA:60:1F:2F:B4:EB:C9:37:BA:53:2E:75:49
  SHA1: DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
  SHA256:
 55:2F:7B:DC:F1:A7:AF:9E:6C:E6:72:01:7F:4F:12:AB:F7:72:40:C7:8E:76:1A:C2:03:D1:D9:D2:0A:C8:99:8

Alias name: quovadisrootca2g3
Certificate fingerprints:
  MD5:  AF:0C:86:6E:BF:40:2D:7F:0B:3E:12:50:BA:12:3D:06
  SHA1: 09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
  SHA256:
 8F:E4:FB:0A:F9:3A:4D:0D:67:DB:0B:EB:B2:3E:37:C7:1B:F3:25:DC:BC:DD:24:0E:A0:4D:AF:58:B4:7E:18:4

Alias name: geotrustprimarycertificationauthorityg3
Certificate fingerprints:
  MD5:  B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  SHA256:
 B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D

Alias name: geotrustprimarycertificationauthorityg2
```

```
Certificate fingerprints:
  MD5:  01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:6

Alias name: godaddyclass2ca
Certificate fingerprints:
  MD5:  91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
  SHA1: 27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
  SHA256:
 C3:84:6B:F2:4B:9E:93:CA:64:27:4C:0E:C6:7C:1E:CC:5E:02:4F:FC:AC:D2:D7:40:19:35:0E:81:FE:54:6A:E

Alias name: trustcoreca1
Certificate fingerprints:
  MD5:  27:92:23:1D:0A:F5:40:7C:E9:E6:6B:9D:D8:F5:E7:6C
  SHA1: 58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
  SHA256:
 5A:88:5D:B1:9C:01:D9:12:C5:75:93:88:93:8C:AF:BB:DF:03:1A:B2:D4:8E:91:EE:15:58:9B:42:97:1D:03:9

Alias name: hellenicacademicandresearchinstitutionseccrootca2015
Certificate fingerprints:
  MD5:  81:E5:B4:17:EB:C2:F5:E1:4B:0D:41:7B:49:92:FE:EF
  SHA1: 9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
  SHA256:
 44:B5:45:AA:8A:25:E6:5A:73:CA:15:DC:27:FC:36:D2:4C:1C:B9:95:3A:06:65:39:B1:15:82:DC:48:7B:48:3

Alias name: utnuserfirstobjectca
Certificate fingerprints:
  MD5:  A7:F2:E4:16:06:41:11:50:30:6B:9C:E3:B4:9C:B0:C9
  SHA1: E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
  SHA256:
 6F:FF:78:E4:00:A7:0C:11:01:1C:D8:59:77:C4:59:FB:5A:F9:6A:3D:F0:54:08:20:D0:F4:B8:60:78:75:E5:8

Alias name: ttelesecglobalrootclass3
Certificate fingerprints:
  MD5:  CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  SHA256:
 FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:B

Alias name: ttelesecglobalrootclass2
Certificate fingerprints:
  MD5:  2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
```

```
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  SHA256:
 91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:5

Alias name: addtrustclass1ca
Certificate fingerprints:
  MD5:  1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC
  SHA1: CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
  SHA256:
 8C:72:09:27:9A:C0:4E:27:5E:16:D0:7F:D3:B7:75:E8:01:54:B5:96:80:46:E3:1F:52:DD:25:76:63:24:E9:A

Alias name: amzninternalrootca
Certificate fingerprints:
  MD5:  08:09:73:AC:E0:78:41:7C:0A:26:33:51:E8:CF:E6:60
  SHA1: A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
  SHA256:
 0E:DE:63:C1:DC:7A:8E:11:F1:AB:BC:05:4F:59:EE:49:9D:62:9A:2F:DE:9C:A7:16:32:A2:64:29:3E:8B:66:A

Alias name: starfieldrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  SHA256:
 2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F

Alias name: camerfirmachambersignca
Certificate fingerprints:
  MD5:  9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  SHA256:
 13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:C

Alias name: secomscrootca2
Certificate fingerprints:
  MD5:  6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F

Alias name: entrustevca
Certificate fingerprints:
  MD5:  D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
```

```
  SHA256:
 73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4

Alias name: secomscrootca1
Certificate fingerprints:
  MD5:  F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  SHA256:
 E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6

Alias name: affirmtrustcommercial
Certificate fingerprints:
  MD5:  82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
  SHA256:
 03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A

Alias name: digicertassuredidrootg3
Certificate fingerprints:
  MD5:  7C:7F:65:31:0C:81:DF:8D:BA:3E:99:E2:5C:AD:6E:FB
  SHA1: F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
  SHA256:
 7E:37:CB:8B:4C:47:09:0C:AB:36:55:1B:A6:F4:5D:B8:40:68:0F:BA:16:6A:95:2D:B1:00:71:7F:43:05:3F:C

Alias name: affirmtrustnetworking
Certificate fingerprints:
  MD5:  42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  SHA256:
 0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1

Alias name: izenpecom
Certificate fingerprints:
  MD5:  A6:B0:CD:85:80:DA:5C:50:34:A3:39:90:2F:55:67:73
  SHA1: 2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
  SHA256:
 25:30:CC:8E:98:32:15:02:BA:D9:6F:9B:1F:BA:1B:09:9E:2D:29:9E:0F:45:48:BB:91:4F:36:3B:C0:D4:53:1

Alias name: amazon-ca-g4-legacy
Certificate fingerprints:
  MD5:  6C:E5:BD:67:A4:4F:E3:FD:C2:4C:46:E6:06:5B:6D:55
  SHA1: EA:E7:DE:F9:0A:BE:9F:0B:68:CE:B7:24:0D:80:74:03:BF:6E:B1:6E
  SHA256:
 CD:72:C4:7F:B4:AD:28:A4:67:2B:E1:86:47:D4:40:E9:3B:16:2D:95:DB:3C:2F:94:BB:81:D9:09:F7:91:24:5
```

```
Alias name: digicertassuredidrootg2
Certificate fingerprints:
  MD5:  92:38:B9:F8:63:24:82:65:2C:57:33:E6:FE:81:8F:9D
  SHA1: A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
  SHA256:
 7D:05:EB:B6:82:33:9F:8C:94:51:EE:09:4E:EB:FE:FA:79:53:A1:14:ED:B2:F4:49:49:45:2F:AB:7D:2F:C1:8

Alias name: comodoaaaservicesroot
Certificate fingerprints:
  MD5:  49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  SHA256:
 D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F

Alias name: entrustnetpremium2048secureserverca
Certificate fingerprints:
  MD5:  EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  SHA256:
 6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:7

Alias name: trustcorrootcertca2
Certificate fingerprints:
  MD5:  A2:E1:F8:18:0B:BA:45:D5:C7:41:2A:BB:37:52:45:64
  SHA1: B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
  SHA256:
 07:53:E9:40:37:8C:1B:D5:E3:83:6E:39:5D:AE:A5:CB:83:9E:50:46:F1:BD:0E:AE:19:51:CF:10:FE:C7:C9:6

Alias name: entrust2048ca
Certificate fingerprints:
  MD5:  EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  SHA256:
 6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:7

Alias name: trustcorrootcertca1
Certificate fingerprints:
  MD5:  6E:85:F1:DC:1A:00:D3:22:D5:B2:B2:AC:6B:37:05:45
  SHA1: FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
  SHA256:
 D4:0E:9C:86:CD:8F:E4:68:C1:77:69:59:F4:9E:A7:74:FA:54:86:84:B6:C4:06:F3:90:92:61:F4:DC:E2:57:5

Alias name: baltimorecybertrustroot
```

```
Certificate fingerprints:
  MD5:  AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  SHA256:
 16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:E

Alias name: eecertificationcentrerootca
Certificate fingerprints:
  MD5:  43:5E:88:D4:7D:1A:4A:7E:FD:84:2E:52:EB:01:D4:6F
  SHA1: C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
  SHA256:
 3E:84:BA:43:42:90:85:16:E7:75:73:C0:99:2F:09:79:CA:08:4E:46:85:68:1F:F1:95:CC:BA:8A:22:9B:8A:7

Alias name: dstacescax6
Certificate fingerprints:
  MD5:  21:D8:4C:82:2B:99:09:33:A2:EB:14:24:8D:8E:5F:E8
  SHA1: 40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
  SHA256:
 76:7C:95:5A:76:41:2C:89:AF:68:8E:90:A1:C7:0F:55:6C:FD:6B:60:25:DB:EA:10:41:6D:7E:B6:83:1F:8C:4

Alias name: comodocertificationauthority
Certificate fingerprints:
  MD5:  5C:48:DC:F7:42:72:EC:56:94:6D:1C:CC:71:35:80:75
  SHA1: 66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
  SHA256:
 0C:2C:D6:3D:F7:80:6F:A3:99:ED:E8:09:11:6B:57:5B:F8:79:89:F0:65:18:F9:80:8C:86:05:03:17:8B:AF:6

Alias name: thawteserverca
Certificate fingerprints:
  MD5:  EE:FE:61:69:65:6E:F8:9C:C6:2A:F4:D7:2B:63:EF:A2
  SHA1: 9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
  SHA256:
 87:C6:78:BF:B8:B2:5F:38:F7:E9:7B:33:69:56:BB:CF:14:4B:BA:CA:A5:36:47:E6:1A:23:25:BC:10:55:31:6

Alias name: secomvalicertclass1ca
Certificate fingerprints:
  MD5:  65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB
  SHA1: E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
  SHA256:
 F4:C1:49:55:1A:30:13:A3:5B:C7:BF:FE:17:A7:F3:44:9B:C1:AB:5B:5A:0A:E7:4B:06:C2:3B:90:00:4C:01:0

Alias name: godaddyrootg2ca
Certificate fingerprints:
  MD5:  80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
```

```
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  SHA256:
 45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:D

Alias name: globalchambersignroot2008
Certificate fingerprints:
  MD5:  9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  SHA256:
 13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:C

Alias name: equifaxsecureebusinessca1
Certificate fingerprints:
  MD5:  14:C0:08:E5:A3:85:03:A3:BE:78:E9:67:4F:27:CA:EE
  SHA1: AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
  SHA256:
 2E:3A:2B:B5:11:25:05:83:6C:A8:96:8B:E2:CB:37:27:CE:9B:56:84:5C:6E:E9:8E:91:85:10:4A:FB:9A:F5:9

Alias name: quovadisrootca3
Certificate fingerprints:
  MD5:  31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF
  SHA1: 1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
  SHA256:
 18:F1:FC:7F:20:5D:F8:AD:DD:EB:7F:E0:07:DD:57:E3:AF:37:5A:9C:4D:8D:73:54:6B:F4:F1:FE:D1:E1:8D:3

Alias name: usertrustecccertificationauthority
Certificate fingerprints:
  MD5:  FA:68:BC:D9:B5:7F:AD:FD:C9:1D:06:83:28:CC:24:C1
  SHA1: D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
  SHA256:
 4F:F4:60:D5:4B:9C:86:DA:BF:BC:FC:57:12:E0:40:0D:2B:ED:3F:BC:4D:4F:BD:AA:86:E0:6A:DC:D2:A9:AD:7

Alias name: quovadisrootca2
Certificate fingerprints:
  MD5:  5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B
  SHA1: CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
  SHA256:
 85:A0:DD:7D:D7:20:AD:B7:FF:05:F8:3D:54:2B:20:9D:C7:FF:45:28:F7:D6:77:B1:83:89:FE:A5:E5:C4:9E:8

Alias name: soneraclass2ca
Certificate fingerprints:
  MD5:  A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
```

```
  SHA256:
 79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:2

Alias name: twcarootcertificationauthority
Certificate fingerprints:
  MD5:  AA:08:8F:F6:F9:7B:B7:F2:B1:A7:1E:9B:EA:EA:BD:79
  SHA1: CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
  SHA256:
 BF:D8:8F:E1:10:1C:41:AE:3E:80:1B:F8:BE:56:35:0E:E9:BA:D1:A6:B9:BD:51:5E:DC:5C:6D:5B:87:11:AC:4

Alias name: baltimorecybertrustca
Certificate fingerprints:
  MD5:  AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  SHA256:
 16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:E

Alias name: cia-crt-g3-01-ca
Certificate fingerprints:
  MD5:  E3:66:DD:D6:A0:D5:40:8F:FF:29:E2:C0:CB:6E:62:1A
  SHA1: 2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
  SHA256:
 20:48:AD:4C:EC:90:7F:FA:4A:15:D4:CE:45:E3:C8:E4:2C:EA:78:33:DC:C7:D3:40:48:FC:60:47:27:42:99:E

Alias name: entrustrootcertificationauthorityg2
Certificate fingerprints:
  MD5:  4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  SHA256:
 43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:3

Alias name: verisignclass3g4ca
Certificate fingerprints:
  MD5:  3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  SHA256:
 69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:7

Alias name: xrampglobalcaroot
Certificate fingerprints:
  MD5:  A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A
```

```
Alias name: identrustcommercialrootca1
Certificate fingerprints:
  MD5:  B3:3E:77:73:75:EE:A0:D3:E3:7E:49:63:49:59:BB:C7
  SHA1: DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
  SHA256:
 5D:56:49:9B:E4:D2:E0:8B:CF:CA:D0:8A:3E:38:72:3D:50:50:3B:DE:70:69:48:E4:2F:55:60:30:19:E5:28:A

Alias name: camerfirmachamberscommerceca
Certificate fingerprints:
  MD5:  B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  SHA256:
 0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C

Alias name: verisignclass3g2ca
Certificate fingerprints:
  MD5:  A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
  SHA1: 85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
  SHA256:
 83:CE:3C:12:29:68:8A:59:3D:48:5F:81:97:3C:0F:91:95:43:1E:DA:37:CC:5E:36:43:0E:79:C7:A8:88:63:8

Alias name: deutschetelekomrootca2
Certificate fingerprints:
  MD5:  74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08
  SHA1: 85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
  SHA256:
 B6:19:1A:50:D0:C3:97:7F:7D:A9:9B:CD:AA:C8:6A:22:7D:AE:B9:67:9E:C7:0B:A3:B0:C9:D9:22:71:C1:70:D

Alias name: certumca
Certificate fingerprints:
  MD5:  2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
  SHA1: 62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
  SHA256:
 D8:E0:FE:BC:1D:B2:E3:8D:00:94:0F:37:D2:7D:41:34:4D:99:3E:73:4B:99:D5:65:6D:97:78:D4:D8:14:36:2

Alias name: cybertrustglobalroot
Certificate fingerprints:
  MD5:  72:E4:4A:87:E3:69:40:80:77:EA:BC:E3:F4:FF:F0:E1
  SHA1: 5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
  SHA256:
 96:0A:DF:00:63:E9:63:56:75:0C:29:65:DD:0A:08:67:DA:0B:9C:BD:6E:77:71:4A:EA:FB:23:49:AB:39:3D:A

Alias name: globalsignrootca
```

```
Certificate fingerprints:
  MD5:  3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  SHA256:
 EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:9

Alias name: secomevrootca1
Certificate fingerprints:
  MD5:  22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  SHA256:
 A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:3

Alias name: globalsignr3ca
Certificate fingerprints:
  MD5:  C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  SHA256:
 CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3

Alias name: staatdernederlandenrootcag3
Certificate fingerprints:
  MD5:  0B:46:67:07:DB:10:2F:19:8C:35:50:60:D1:0B:F4:37
  SHA1: D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
  SHA256:
 3C:4F:B0:B9:5A:B8:B3:00:32:F4:32:B8:6F:53:5F:E1:72:C1:85:D0:FD:39:86:58:37:CF:36:18:7F:A6:F4:2

Alias name: staatdernederlandenrootcag2
Certificate fingerprints:
  MD5:  7C:A5:0F:F8:5B:9A:7D:6D:30:AE:54:5A:E3:42:A2:8A
  SHA1: 59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
  SHA256:
 66:8C:83:94:7D:A6:3B:72:4B:EC:E1:74:3C:31:A0:E6:AE:D0:DB:8E:C5:B3:1B:E3:77:BB:78:4F:91:B6:71:6

Alias name: aolrootca2
Certificate fingerprints:
  MD5:  D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF
  SHA1: 85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
  SHA256:
 7D:3B:46:5A:60:14:E5:26:C0:AF:FC:EE:21:27:D2:31:17:27:AD:81:1C:26:84:2D:00:6A:F3:73:06:CC:80:B

Alias name: dstrootcax3
Certificate fingerprints:
  MD5:  41:03:52:DC:0F:F7:50:1B:16:F0:02:8E:BA:6F:45:C5
```

```
  SHA1: DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
  SHA256:
 06:87:26:03:31:A7:24:03:D9:09:F1:05:E6:9B:CF:0D:32:E1:BD:24:93:FF:C6:D9:20:6D:11:BC:D6:77:07:3

Alias name: trustcenteruniversalcai
Certificate fingerprints:
  MD5:  45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C
  SHA1: 6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
  SHA256:
 EB:F3:C0:2A:87:89:B1:FB:7D:51:19:95:D6:63:B7:29:06:D9:13:CE:0D:5E:10:56:8A:8A:77:E2:58:61:67:E

Alias name: aolrootca1
Certificate fingerprints:
  MD5:  14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E
  SHA1: 39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
  SHA256:
 77:40:73:12:C6:3A:15:3D:5B:C0:0B:4E:51:75:9C:DF:DA:C2:37:DC:2A:33:B6:79:46:E9:8E:9B:FA:68:0A:E

Alias name: affirmtrustpremiumecc
Certificate fingerprints:
  MD5:  64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  SHA256:
 BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:2

Alias name: microseceszignorootca2009
Certificate fingerprints:
  MD5:  F8:49:F4:03:BC:44:2D:83:BE:48:69:7D:29:64:FC:B1
  SHA1: 89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
  SHA256:
 3C:5F:81:FE:A5:FA:B8:2C:64:BF:A2:EA:EC:AF:CD:E8:E0:77:FC:86:20:A7:CA:E5:37:16:3D:F3:6E:DB:F3:7

Alias name: verisignclass1g3ca
Certificate fingerprints:
  MD5:  B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
  SHA1: 20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
  SHA256:
 CB:B5:AF:18:5E:94:2A:24:02:F9:EA:CB:C0:ED:5B:B8:76:EE:A3:C1:22:36:23:D0:04:47:E4:F3:BA:55:4B:6

Alias name: certplusrootcag2
Certificate fingerprints:
  MD5:  A7:EE:C4:78:2D:1B:EE:2D:B9:29:CE:D6:A7:96:32:31
  SHA1: 4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
```

```
  SHA256:
 6C:C0:50:41:E6:44:5E:74:69:6C:4C:FB:C9:F8:0F:54:3B:7E:AB:BB:44:B4:CE:6F:78:7C:6A:99:71:C4:2F:1

Alias name: certplusrootcag1
Certificate fingerprints:
  MD5:  7F:09:9C:F7:D9:B9:5C:69:69:56:D5:37:3E:14:0D:42
  SHA1: 22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
  SHA256:
 15:2A:40:2B:FC:DF:2C:D5:48:05:4D:22:75:B3:9C:7F:CA:3E:C0:97:80:78:B0:F0:EA:76:E5:61:A6:C7:43:3

Alias name: addtrustexternalca
Certificate fingerprints:
  MD5:  1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  SHA256:
 68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F

Alias name: entrustrootcertificationauthority
Certificate fingerprints:
  MD5:  D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  SHA256:
 73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4

Alias name: verisignclass3ca
Certificate fingerprints:
  MD5:  EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
  SHA1: A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
  SHA256:
 A4:B6:B3:99:6F:C2:F3:06:B3:FD:86:81:BD:63:41:3D:8C:50:09:CC:4F:A3:29:C2:CC:F0:E2:FA:1B:14:03:0

Alias name: digicertassuredidrootca
Certificate fingerprints:
  MD5:  87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
  SHA1: 05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
  SHA256:
 3E:90:99:B5:01:5E:8F:48:6C:00:BC:EA:9D:11:1E:E7:21:FA:BA:35:5A:89:BC:F1:DF:69:56:1E:3D:C6:32:5

Alias name: globalsignrootcar3
Certificate fingerprints:
  MD5:  C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  SHA256:
 CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3
```

```
Alias name: globalsignrootcar2
Certificate fingerprints:
  MD5:  94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  SHA256:
 CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9

Alias name: verisignclass1ca
Certificate fingerprints:
  MD5:  86:AC:DE:2B:C5:6D:C3:D9:8C:28:88:D3:8D:16:13:1E
  SHA1: CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1
  SHA256:
 51:84:7C:8C:BD:2E:9A:72:C9:1E:29:2D:2A:E2:47:D7:DE:1E:3F:D2:70:54:7A:20:EF:7D:61:0F:38:B8:84:2

Alias name: thawtepremiumserverca
Certificate fingerprints:
  MD5:  A6:6B:60:90:23:9B:3F:2D:BB:98:6F:D6:A7:19:0D:46
  SHA1: E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
  SHA256:
 3F:9F:27:D5:83:20:4B:9E:09:C8:A3:D2:06:6C:4B:57:D3:A2:47:9C:36:93:65:08:80:50:56:98:10:5D:BC:E

Alias name: verisigntsaca
Certificate fingerprints:
  MD5:  F2:89:95:6E:4D:05:F0:F1:A7:21:55:7D:46:11:BA:47
  SHA1: 20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
  SHA256:
 CB:6B:05:D9:E8:E5:7C:D8:82:B1:0B:4D:B7:0D:E4:BB:1D:E4:2B:A4:8A:7B:D0:31:8B:63:5B:F6:E7:78:1A:9

Alias name: thawteprimaryrootca
Certificate fingerprints:
  MD5:  8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
  SHA1: 91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
  SHA256:
 8D:72:2F:81:A9:C1:13:C0:79:1D:F1:36:A2:96:6D:B2:6C:95:0A:97:1D:B4:6B:41:99:F4:EA:54:B7:8B:FB:9

Alias name: visaecommerceroot
Certificate fingerprints:
  MD5:  FC:11:B8:D8:08:93:30:00:6D:23:F9:7E:EB:52:1E:02
  SHA1: 70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
  SHA256:
 69:FA:C9:BD:55:FB:0A:C7:8D:53:BB:EE:5C:F1:D5:97:98:9F:D0:AA:AB:20:A2:51:51:BD:F1:73:3E:E7:D1:2

Alias name: digicertglobalrootg3
```

```
Certificate fingerprints:
  MD5:  F5:5D:A4:50:A5:FB:28:7E:1E:0F:0D:CC:96:57:56:CA
  SHA1: 7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
  SHA256:
 31:AD:66:48:F8:10:41:38:C7:38:F3:9E:A4:32:01:33:39:3E:3A:18:CC:02:29:6E:F9:7C:2A:C9:EF:67:31:D

Alias name: xrampglobalca
Certificate fingerprints:
  MD5:  A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A

Alias name: digicertglobalrootg2
Certificate fingerprints:
  MD5:  E4:A6:8A:C8:54:AC:52:42:46:0A:FD:72:48:1B:2A:44
  SHA1: DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
  SHA256:
 CB:3C:CB:B7:60:31:E5:E0:13:8F:8D:D3:9A:23:F9:DE:47:FF:C3:5E:43:C1:14:4C:EA:27:D4:6A:5A:B1:CB:5

Alias name: valicertclass2ca
Certificate fingerprints:
  MD5:  A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
  SHA1: 31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
  SHA256:
 58:D0:17:27:9C:D4:DC:63:AB:DD:B1:96:A6:C9:90:6C:30:C4:E0:87:83:EA:E8:C1:60:99:54:D6:93:55:59:6

Alias name: geotrustprimaryca
Certificate fingerprints:
  MD5:  02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  SHA256:
 37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6

Alias name: netlockaranyclassgoldfotanusitvany
Certificate fingerprints:
  MD5:  C5:A1:B7:FF:73:DD:D6:D7:34:32:18:DF:FC:3C:AD:88
  SHA1: 06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
  SHA256:
 6C:61:DA:C3:A2:DE:F0:31:50:6B:E0:36:D2:A6:FE:40:19:94:FB:D1:3D:F9:C8:D4:66:59:92:74:C4:46:EC:9

Alias name: geotrustglobalca
Certificate fingerprints:
  MD5:  F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5
```

```
  SHA1: DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
  SHA256:
 FF:85:6A:2D:25:1D:CD:88:D3:66:56:F4:50:12:67:98:CF:AB:AA:DE:40:79:9C:72:2D:E4:D2:B5:DB:36:A7:3

Alias name: oistewisekeyglobalrootgbca
Certificate fingerprints:
  MD5:  A4:EB:B9:61:28:2E:B7:2F:98:B0:35:26:90:99:51:1D
  SHA1: 0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
  SHA256:
 6B:9C:08:E8:6E:B0:F7:67:CF:AD:65:CD:98:B6:21:49:E5:49:4A:67:F5:84:5E:7B:D1:ED:01:9F:27:B8:6B:D

Alias name: certumtrustednetworkca2
Certificate fingerprints:
  MD5:  6D:46:9E:D9:25:6D:08:23:5B:5E:74:7D:1E:27:DB:F2
  SHA1: D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
  SHA256:
 B6:76:F2:ED:DA:E8:77:5C:D3:6C:B0:F6:3C:D1:D4:60:39:61:F4:9E:62:65:BA:01:3A:2F:03:07:B6:D0:B8:0

Alias name: starfieldservicesrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  SHA256:
 56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B

Alias name: comodorsacertificationauthority
Certificate fingerprints:
  MD5:  1B:31:B0:71:40:36:CC:14:36:91:AD:C4:3E:FD:EC:18
  SHA1: AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
  SHA256:
 52:F0:E1:C4:E5:8E:C6:29:29:1B:60:31:7F:07:46:71:B8:5D:7E:A8:0D:5B:07:27:34:63:53:4B:32:B4:02:3

Alias name: comodoaaaca
Certificate fingerprints:
  MD5:  49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  SHA256:
 D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F

Alias name: identrustpublicsectorrootca1
Certificate fingerprints:
  MD5:  37:06:A5:B0:FC:89:9D:BA:F4:6B:8C:1A:64:CD:D5:BA
  SHA1: BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
```

```
  SHA256:
 30:D0:89:5A:9A:44:8A:26:20:91:63:55:22:D1:F5:20:10:B5:86:7A:CA:E1:2C:78:EF:95:8F:D4:F4:38:9F:2

Alias name: certplusclass2primaryca
Certificate fingerprints:
  MD5:  88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
  SHA1: 74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
  SHA256:
 0F:99:3C:8A:EF:97:BA:AF:56:87:14:0E:D5:9A:D1:82:1B:B4:AF:AC:F0:AA:9A:58:B5:D5:7A:33:8A:3A:FB:C

Alias name: ttelesecglobalrootclass2ca
Certificate fingerprints:
  MD5:  2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  SHA256:
 91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:5

Alias name: accvraiz1
Certificate fingerprints:
  MD5:  D0:A0:5A:EE:05:B6:09:94:21:A1:7D:F1:B2:29:82:02
  SHA1: 93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
  SHA256:
 9A:6E:C0:12:E1:A7:DA:9D:BE:34:19:4D:47:8A:D7:C0:DB:18:22:FB:07:1D:F1:29:81:49:6E:D1:04:38:41:1

Alias name: digicerthighassuranceevrootca
Certificate fingerprints:
  MD5:  D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A
  SHA1: 5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
  SHA256:
 74:31:E5:F4:C3:C1:CE:46:90:77:4F:0B:61:E0:54:40:88:3B:A9:A0:1E:D0:0B:A6:AB:D7:80:6E:D3:B1:18:C

Alias name: amzninternalinfoseccag3
Certificate fingerprints:
  MD5:  E9:34:94:02:BA:BB:31:6B:22:E6:2B:A9:C4:F0:26:04
  SHA1: B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
  SHA256:
 81:03:0B:C7:E2:54:DA:7B:F8:B7:45:DB:DD:41:15:89:B5:A3:81:86:FB:4B:29:77:1F:84:0A:18:D9:67:6D:6

Alias name: cia-crt-g3-02-ca
Certificate fingerprints:
  MD5:  FD:B9:23:FD:D3:EB:2D:3E:57:EF:56:FF:DB:D3:E4:B9
  SHA1: 96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
  SHA256:
 93:F1:72:FB:BA:43:31:5C:06:EE:0F:9F:04:89:B8:F6:88:BC:75:15:3C:BE:B4:80:AC:A7:14:3A:F6:FC:4A:C
```

```
Alias name: entrustrootcertificationauthorityec1
Certificate fingerprints:
  MD5:  B6:7E:1D:F0:58:C5:49:6C:24:3B:3D:ED:98:18:ED:BC
  SHA1: 20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
  SHA256:
 02:ED:0E:B2:8C:14:DA:45:16:5C:56:67:91:70:0D:64:51:D7:FB:56:F0:B2:AB:1D:3B:8E:B0:70:E5:6E:DF:F

Alias name: securitycommunicationrootca
Certificate fingerprints:
  MD5:  F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  SHA256:
 E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6

Alias name: globalsignca
Certificate fingerprints:
  MD5:  3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  SHA256:
 EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:9

Alias name: trustcenterclass2caii
Certificate fingerprints:
  MD5:  CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23
  SHA1: AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
  SHA256:
 E6:B8:F8:76:64:85:F8:07:AE:7F:8D:AC:16:70:46:1F:07:C0:A1:3E:EF:3A:1F:F7:17:53:8D:7A:BA:D3:91:B

Alias name: camerfirmachambersofcommerceroot
Certificate fingerprints:
  MD5:  B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  SHA256:
 0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C

Alias name: geotrustprimarycag3
Certificate fingerprints:
  MD5:  B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  SHA256:
 B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D

Alias name: geotrustprimarycag2
```

```
Certificate fingerprints:
  MD5:  01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:6

Alias name: hongkongpostrootca1
Certificate fingerprints:
  MD5:  A8:0D:6F:39:78:B9:43:6D:77:42:6D:98:5A:CC:23:CA
  SHA1: D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
  SHA256:
 F9:E6:7D:33:6C:51:00:2A:C0:54:C6:32:02:2D:66:DD:A2:E7:E3:FF:F1:0A:D0:61:ED:31:D8:BB:B4:10:CF:B

Alias name: affirmtrustpremiumeccca
Certificate fingerprints:
  MD5:  64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  SHA256:
 BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:2

Alias name: hellenicacademicandresearchinstitutionsrootca2015
Certificate fingerprints:
  MD5:  CA:FF:E2:DB:03:D9:CB:4B:E9:0F:AD:84:FD:7B:18:CE
  SHA1: 01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
  SHA256:
 A0:40:92:9A:02:CE:53:B4:AC:F4:F2:FF:C6:98:1C:E4:49:6F:75:5E:6D:45:FE:0B:2A:69:2B:CD:52:52:3F:3
```

AWS IoT Events

AWS IoT Events (`iotEvents`) アクションは、MQTT メッセージから AWS IoT Events 入力にデータを送信します。

> **Important**
>
> ペイロードが AWS IoT Core なしで に送信された場合`Input attribute Key`、またはキーがキーで指定された同じ JSON パスにない場合、IoT ルールはエラー で失敗しますFailed to send message to Iot Events。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールiotevents:BatchPutMessage。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`batchMode`

(オプション) イベントアクションをバッチとして処理するかどうか。デフォルト値は `false` です。

`batchMode` が `true`で、ルール SQL ステートメントが配列に評価される場合、各配列要素は、 を呼び出して AWS IoT イベントに送信したときに個別のメッセージとして扱われますBatchPutMessage。結果の配列には 10 を超えるメッセージを含めることはできません。

`batchMode` が `true` の場合、`messageId` を指定することはできません。

置換テンプレートをサポート: いいえ

`inputName`

AWS IoT Events 入力の名前。

置換テンプレートをサポート: API および AWS CLI のみ

`messageId`

(オプション) これを使用して、特定の を持つ 1 つの入力 (メッセージ) `messageId` のみがディ AWS IoT Events テクターによって処理されることを確認します。`${newuuid()}` 置換テンプレートを使用して、リクエストごとに一意の ID を生成できます。

`batchMode` が `true` である場合、`messageId` を指定することはできません。新しい UUID 値が割り当てられます。

置換テンプレートをサポート: はい

`roleArn`

がディテクターに入力 AWS IoT Events を送信 AWS IoT できるようにする IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON 例では、 AWS IoT ルールで IoT Events アクションを定義します。

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

関連情報

- デAWS IoT Events ベロッパーガイドの「 AWS IoT Eventsとは」

AWS IoT SiteWise

AWS IoT SiteWise (`iotSiteWise`) アクションは、MQTT メッセージからアセットプロパティにデータを送信します AWS IoT SiteWise。

AWS IoT モノからデータを取り込む方法を示すチュートリアルに従うことができます。詳細については、「 ユーザーガイド」の AWS IoT 「モノから への AWS IoT SiteWise データの取り込み」チュートリアルまたはAWS IoT 「 Core ルールを使用したデータの取り込み」セクションを参照してください。 AWS IoT SiteWise

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールiotsitewise:BatchPutAssetPropertyValue。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  次の信頼ポリシーの例をロールにアタッチできます。

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

  セキュリティを向上させるために、 Conditionプロパティで AWS IoT SiteWise アセット階層パスを指定できます。次の例は、アセット階層パスを指定する信頼ポリシーです。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
            }
        }
        }
    ]
```

```
}
```

- このアクション AWS IoT SiteWise を使用して にデータを送信する場合、データは BatchPutAssetPropertyValueオペレーションの要件を満たしている必要があります。詳細については、「AWS IoT SiteWise API リファレンス」の「BatchPutAssetPropertyValue」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`putAssetPropertyValueEntries`

アセットプロパティ値エントリのリストで、それぞれに次の情報が含まれています。

`propertyAlias`

(オプション) アセットプロパティに関連付けられたプロパティエイリアス。`propertyAlias` または `assetId` と `propertyId` の両方のいずれかを指定します。プロパティのエイリアスの詳細については、AWS IoT SiteWise ユーザーガイドの Mapping industrial data streams to asset properties を参照してください。

置換テンプレートをサポート: はい

`assetId`

(オプション) AWS IoT SiteWise アセットの ID。`propertyAlias` または `assetId` と `propertyId` の両方のいずれかを指定します。

置換テンプレートをサポート: はい

`propertyId`

(オプション) アセットのプロパティの ID。`propertyAlias` または `assetId` と `propertyId` の両方のいずれかを指定します。

置換テンプレートをサポート: はい

`entryId`

(オプション) このエントリの一意な識別子。`entryId` を定義して、障害発生時にエラーの原因となったメッセージをより正確に追跡します。デフォルトは新しい UUID です。

置換テンプレートをサポート: はい

`propertyValues`

次の形式でタイムスタンプ、品質、値 (TQV) を含み、挿入するプロパティ値のリスト。

`timestamp`

次の情報を含むタイムスタンプ構造体。

`timeInSeconds`

時間を秒単位で含む文字列 (Unix エポック時間)。メッセージペイロードにタイムスタンプがない場合は、timestamp() を使用して、現在の時間をミリ秒単位で返すことができます。この時間を秒に変換するには、次の置換テンプレートを使用できます: **`${floor(timestamp() / 1E3)}`**。

置換テンプレートをサポート: はい

`offsetInNanos`

(オプション) 秒単位の時間からのナノ秒の時間オフセットを含む文字列。メッセージペイロードにタイムスタンプがない場合は、timestamp() を使用して、現在の時間をミリ秒単位で返すことができます。その時点からのナノ秒のオフセットを計算するには、次の置換テンプレートを使用できます: **`${(timestamp() % 1E3) * 1E6}`**。

置換テンプレートをサポート: はい

Unix エポック時間については、 はタイムスタンプが過去最大 7 日、5 分先までのエントリのみ AWS IoT SiteWise を受け入れます。

`quality`

(オプション) 値の品質を表す文字列。有効な値: `GOOD`、`BAD`、`UNCERTAIN`。

置換テンプレートをサポート: はい

`value`

アセットプロパティのデータ型に応じて、次のいずれかの値フィールドを含む値構造体。

`booleanValue`

(オプション) 値エントリのブール値を含む文字列。

置換テンプレートをサポート: はい

`doubleValue`

(オプション) 値エントリの double 値を含む文字列。

置換テンプレートをサポート: はい

`integerValue`

(オプション) 値エントリの整数値を含む文字列。

置換テンプレートをサポート: はい

`stringValue`

(オプション) 値エントリの文字列値。

置換テンプレートをサポート: はい

`roleArn`

アセットプロパティ値を送信する AWS IoT アクセス許可を付与する IAM ロールの ARN AWS IoT SiteWise。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON 例では、 AWS IoT ルールで基本的な IoT SiteWise アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで IoT SiteWise アクションを定義します。この例では、トピックをプロパティエイリアスおよび timestamp() 関数として使用します。たとえば、/company/windfarm/3/turbine/7/rpm にデータをパブリッシュする場合、このアクションは、指定したトピックと同じプロパティエイリアスを持つアセットプロパティにデータを送信します。

```
{
    "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"
                }
            }
```

```
        ]
    }
}
```

関連情報

- 「 AWS IoT SiteWiseユーザーガイド」の「AWS IoT SiteWise とは」
- AWS IoT SiteWise 「 ユーザーガイド」の「 AWS IoT Core ルールを使用したデータの取り込み」
- 「 ユーザーガイド」の AWS IoT 「 AWS IoT SiteWise へのデータの取り込みAWS IoT SiteWise 」
- AWS IoT SiteWise ユーザーガイドの AWS IoT SiteWise ルールアクションのトラブルシューティング

Firehose

Firehose (`firehose`) アクションは、MQTT メッセージから Amazon Data Firehose ストリーミングにデータを送信します。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールfirehose:PutRecord。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- Firehose を使用して Amazon S3 バケットにデータを送信し、カスタマーマネージド AWS KMS key を使用して AWS KMS Amazon S3 に保管中のデータを暗号化する場合、Firehose はバケットへのアクセス権と、発信者に代わって AWS KMS key を使用するアクセス許可を持っている必要があります。詳細については、「Amazon Firehose データファイアハウスデベロッパーガイド」の「Amazon S3 送信先に Firehose アクセスを付与する」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`batchMode`

(オプション) PutRecordBatch を使用して、Firehose ストリーミングをバッチとして配信するかどうか。デフォルト値は `false` です。

`batchMode` が `true` で、ルールの SQL ステートメントが配列に評価される場合、各配列要素は PutRecordBatch リクエストで 1 つのレコードを形成します。結果の配列には 500 を超えるレコードを含めることはできません。

置換テンプレートをサポート: いいえ

`deliveryStreamName`

メッセージデータの書き込み先として指定する Firehose ストリーム。

置換テンプレートをサポート: API および AWS CLI のみ

`separator`

(オプション) Firehose ストリーミングに書き込まれたレコードを区切るために使用される文字区切り記号。このパラメータを省略すると、ストリーミングは区切り記号を使用しません。有効な値: `,` (カンマ)、`\t` (タブ)、`\n` (改行)、`\r\n` (Windows 改行)。

置換テンプレートをサポート: いいえ

`roleArn`

Firehose ストリーミングへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで Firehose アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して Firehose アクションを定義します。

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

関連情報

- Amazon Data Firehose デベロッパーガイドの Amazon Data Firehose とは

Kinesis Data Streams

Kinesis Data Streams (`kinesis`) アクションは、Amazon Kinesis Data Streams に MQTT メッセージのデータを書き込みます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールkinesis:PutRecord。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- AWS KMS カスタマーマネージド AWS KMS key (KMS キー) を使用して Kinesis Data Streams に保管中のデータを暗号化する場合、サービスには AWS KMS key 発信者に代わって を使用するアクセス許可が必要です。詳細については、[Amazon Kinesis Data Streams Developer Guide](Amazon Kinesis Data Streams デベロッパーガイド)の[AWS KMS keys Permissions to use user-generated](ユーザーが生成した アクセス許可)を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`stream`

データを書き込む Kinesis データストリーミング。

置換テンプレートをサポート: API および AWS CLI のみ

`partitionKey`

どのシャードにデータを書き込むかを決定するために使用されるパーティションキー。パーティションキーは通常、式 (例: `${topic()}` または `${timestamp()}`) で構成されます。

置換テンプレートをサポート: はい

`roleArn`

Kinesis データストリームへのアクセス AWS IoT 許可を付与する IAM ロールの ARN。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで Kinesis Data Streams アクションを定義します。

```
{
```

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して Kinesis アクションを定義します。

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

関連情報

- Amazon Kinesis Data Streams デベロッパーガイドの Amazon Kinesis Data Streams とは

Lambda

Lambda (`lambda`) アクションは AWS Lambda 関数を呼び出し、MQTT メッセージを渡します。 は Lambda 関数を非同期的に AWS IoT 呼び出します。

Lambda アクションを使用してルールを作成およびテストする方法を示すチュートリアルに従うことができます。詳細については、「チュートリアル: AWS Lambda 関数を使用して通知をフォーマットする」を参照してください。

要件

このルールアクションには、以下の要件があります。

- AWS IoT が Lambda 関数を呼び出すには、 アクセス`lambda:InvokeFunction`許可を付与するポリシーを設定する必要があります AWS IoT。Lambda ポリシーが存在する AWS リージョン のと同じ で定義された Lambda 関数のみを呼び出すことができます。Lambda 関数はリソースベースのポリシーを使用するため、ポリシーを Lambda 関数自体にアタッチする必要があります。

  次の AWS CLI コマンドを使用して、 アクセス`lambda:InvokeFunction`許可を付与するポリシーをアタッチします。このコマンドを、以下のように置き換えます。

  - *`function_name`* を Lambda 関数の名前に置き換えます。関数のリソースポリシーを更新するための新しいアクセス許可を追加します。
  - 関数 AWS リージョン の を持つ *`#####`*。
  - *`account-id`* は、ルールが定義されている AWS アカウント 番号です。
  - *`rule-name`* は、Lambda アクションを定義する AWS IoT ルールの名前です。
  - *`unique_id`* を一意のステートメント識別子で置き換えます。

  > **Important**
  >
  > `source-arn` または を指定せずにプリン AWS IoT シパルのアクセス許可を追加するとと`source-account`、Lambda アクションを使用してルールを作成する AWS アカウントは、Lambda 関数を呼び出すルールをアクティブ化できます AWS IoT。

  詳細については、「AWS Lambda のアクセス許可」を参照してください。

```
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 コンソールを使用して Lambda ルールアクションのルールを作成すると、Lambda 関数が自動的にトリガーされます。 AWS CloudFormation の代わりに を使用する場合は[AWS::IoT::TopicRule LambdaAction](#)、 [AWS::lambda::Permission](#)リソースを追加する必要があります。このリソースにより、Lambda 関数をトリガーするアクセス許可が付与されます。

  次のコードは、このリソースを追加する方法を示しています。この例では、次のように置き換えます。

  - *function_name* を Lambda 関数の名前に置き換えます。
  - *region* と 関数 AWS リージョン の 。
  - *account-id* は、ルールが定義されている AWS アカウント 番号です。
  - *rule-name* は、Lambda アクションを定義する AWS IoT ルールの名前です。

```
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 カスタマーマネージドを使用して Lambda AWS KMS key に保管中のデータを暗号化する場合、サービスには AWS KMS key 発信者に代わって を使用するアクセス許可が必要です。詳しくは、[AWS Lambda Developer Guide](デベロッパーガイド)の[Encryption at rest](保管時の暗号化)を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`functionArn`

呼び出す Lambda 関数の ARN。 には、関数を呼び出すアクセス許可 AWS IoT が必要です。詳細については、「要件」を参照してください。

Lambda 関数のバージョンまたはエイリアスを指定しない場合、関数の最新バージョンがシャットダウンされます。Lambda 関数の特定のバージョンをシャットダウンする場合は、バージョンまたはエイリアスを指定できます。バージョンまたはエイリアスを指定するには、Lambda 関数の ARN にバージョンまたはエイリアスを追加します。

```
arn:aws:lambda:us-east-2:123456789012:function:myLambdaFunction:someAlias
```

バージョニングとエイリアスの詳細については、「AWS Lambda 関数のバージョン」を参照してください。

代替テンプレートをサポート: API および AWS CLI のみ

例

次の JSON の例では、 AWS IoT ルールで Lambda アクションを定義します。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-
east-2:123456789012:function:myLambdaFunction"
                }
            }
        ]
    }
}
```

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して Lambda アクションを定義します。

```
{
```

```
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:
${topic()}"
                }
            }
        ]
    }
}
```

関連情報

- デAWS Lambda ベロッパーガイドの「 AWS Lambdaとは」
- チュートリアル: AWS Lambda 関数を使用して通知をフォーマットする

ロケーション

Location (`location`) アクションによって、地理的位置データを Amazon Location Service に送信します。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールgeo:BatchUpdateDevicePosition。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`deviceId`

位置データを提供するデバイスの一意の ID。詳細については、「Amazon Location Service API リファレンス」の「DeviceId」を参照してください。

置換テンプレートをサポート: はい

`latitude`

デバイスの位置の緯度を表す double 値として評価される文字列。

置換テンプレートをサポート: はい

`longitude`

デバイスの位置の経度を表す double 値として評価される文字列。

置換テンプレートをサポート: はい

`roleArn`

Amazon Location Service ドメインへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

`timestamp`

位置データがサンプリングされた時刻。デフォルト値は MQTT メッセージが処理された時間です。

`timestamp` 値は、次の 2 つの値で構成されます。

- `value`: 長いエポック時間の値を返す式。the section called “time_to_epoch(String, String)” 関数を使用して、メッセージペイロードで渡される日付または時刻の値から有効なタイムスタンプを作成できます。置換テンプレートのサポート: はい
- `unit` (オプション): `value` で説明されている式の結果として生じるタイムスタンプ値の精度。有効な値: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`。デフォルトは`MILLISECONDS`です。置換テンプレートをサポート: API および AWS CLI のみ。

`trackerName`

Location が更新される Amazon Location トラッカーリソースの名前。詳細については、「Amazon Location Service デベロッパーガイド」の「トラッカー」を参照してください。

代替テンプレートをサポート: API および AWS CLI のみ

例

次の JSON の例では、 AWS IoT ルールで Location アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して Location アクションを定義します。

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

次の MQTT ペイロードの例では、上記の例の置換テンプレートがデータにアクセスする方法を示しています。get-device-position-history CLI コマンドを使用して、MQTT ペイロードデータがロケーショントラッカーに配信されていることを確認できます。

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

関連情報

- Amazon Location Service デベロッパーガイドの「Amazon Location Service とは?」

OpenSearch

OpenSearch (`openSearch`) アクションは、Amazon OpenSearch Service ドメインに MQTT メッセージからデータを書き込みます。その後、OpenSearch Dashboards などのツールを使用して、OpenSearch Service のデータをクエリおよび視覚化できます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールes:ESHttpPut。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- OpenSearch Service でカスタマーマネージドを使用して保管中のデータを AWS KMS key 暗号化する場合、サービスには発信者に代わって KMS キーを使用するアクセス許可が必要です。詳細については、Amazon OpenSearch Service デベロッパーガイドの「Amazon OpenSearch Service の保管中のデータの暗号化」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`endpoint`

Amazon OpenSearch Service ドメインのエンドポイント。

置換テンプレートをサポート: API および AWS CLI のみ

`index`

データを保存する OpenSearch インデックス。

置換テンプレートをサポート: はい

`type`

保存するドキュメントのタイプ。

> **Note**
>
> 1.0 以降の OpenSearch バージョンでは、`type` パラメータの値は `_doc` である必要があります。詳細については、「OpenSearch ドキュメント」を参照してください。

置換テンプレートをサポート: はい

`id`

各ドキュメントの一意の識別子。

置換テンプレートをサポート: はい

`roleARN`

OpenSearch Service ドメインへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

制限事項

OpenSearch (`openSearch`) アクションを使用して VPC Elasticsearch クラスターにデータを配信することはできません。

例

次の JSON の例では、 AWS IoT ルールで OpenSearch アクションを定義し、`OpenSearch`アクションのフィールドを指定する方法を定義します。詳細については、[OpenSearchAction]を参照してください。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して OpenSearch アクションを定義します。

```
{
    "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**
>
> 置換した type フィールドは OpenSearch バージョン 1.0 で機能します。1.0 以降のバージョンでは、type の値は _doc である必要があります。

関連情報

Amazon OpenSearch Service デベロッパーガイドの「Amazon OpenSearch Service とは?」

再発行

再発行 (`republish`) アクションは、MQTT メッセージを別の MQTT トピックに再発行します。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールiot:Publish。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`headers`

MQTT バージョン 5.0 のヘッダー情報。

詳細については、「AWS API リファレンス」の「RepublishAction」と「MqttHeaders」を参照してください。

`topic`

メッセージの再発行先として指定する MQTT トピック。

`$` で始まる予約済みトピックに再発行するには、代わりに `$$` を使用します。例えば、デバイスシャドウトピック `$aws/things/MyThing/shadow/update` に再発行する場合は、トピックを `$$aws/things/MyThing/shadow/update` として指定します。

> **Note**
>
> 予約済みのジョブトピックへの再発行はサポートされていません。

> AWS IoT Device Defender 予約トピックは HTTP パブリッシュをサポートしていません。

置換テンプレートをサポート: はい

`qos`

(オプション) メッセージを再発行するときに使用する Quality of Service (QoS) レベル。有効な値: `0`、`1`。デフォルト値は `0` です。MQTT QoS の詳細については、「MQTT」を参照してください。

置換テンプレートをサポート: いいえ

`roleArn`

が MQTT トピックに発行 AWS IoT できるようにする IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで再発行アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して再発行アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールheadersで を使用して再発行アクションを定義します。

```
{
    "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**
>
> 元の送信元 IP は Republish アクションには渡されません。

S3

S3 (s3) アクションは、MQTT メッセージのデータを Amazon Simple Storage Service (Amazon S3) バケットに書き込みます。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールs3:PutObject。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- AWS KMS カスタマーマネージド を使用して Amazon S3 AWS KMS key に保管中のデータを暗号化する場合、サービスには発信者に代わって AWS KMS key を使用するアクセス許可が必要です。詳細については、「Amazon Simple Storage Service デベロッパーガイド」のAWS 「 マネージド AWS KMS keys およびカスタマーマネージド AWS KMS keys」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`bucket`

データの書き込み先として指定する Amazon S3 バケット。

代替テンプレートをサポート: API および AWS CLI のみ

`cannedacl`

(オプション) オブジェクトキーによって識別されるオブジェクトへのアクセスをコントロールする Amazon S3 既定 ACL。使用できる値などの詳細については、既定 ACL を参照してください (許可値の一覧が記載されています)。

置換テンプレートをサポート: いいえ

`key`

データの書き込み先として指定するファイルのパス。

このパラメータが `${topic()}/${timestamp()}` であり、トピックが `some/topic` であるメッセージをルールが受信する例を考えてください。現在のタイムスタンプが `1460685389` の場合、このアクションはデータを S3 バケットの `some/topic` フォルダの `1460685389` というファイルに書き込みます。

> **Note**
>
> 静的キーを使用する場合、 はルールが呼び出されるたびに 1 つのファイルを AWS IoT 上書きします。受信したメッセージごとに新しいファイルが Amazon S3 に保存されるように、メッセージのタイムスタンプまたは別の一意のメッセージ識別子を使用することをお勧めします。

置換テンプレートをサポート: はい

`roleArn`

Amazon S3 バケットへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで S3 アクションを定義します。

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

関連情報

- Amazon Simple Storage Service デベロッパーガイドの Amazon S3 とは?

Salesforce IoT

Salesforce IoT (`salesforce`) アクションは、ルールをトリガーした MQTT メッセージから Salesforce IoT 入力ストリーミングにデータを送信します。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`url`

Salesforce IoT 入力ストリームによって公開される URL。この URL は、入力ストリームの作成時に Salesforce IoT プラットフォームから入手できます。詳細については、Salesforce IoT ドキュメントを参照してください。

置換テンプレートをサポート: いいえ

`token`

指定した Salesforce IoT 入力ストリームへのアクセスを認証するために使用されるトークン。このトークンは、入力ストリームの作成時に Salesforce IoT プラットフォームから入手できます。詳細については、Salesforce IoT ドキュメントを参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON 例では、 AWS IoT ルールで Salesforce IoT アクションを定義します。

```
{
    "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`) アクションは、Amazon Simple Notification Service (Amazon SNS) プッシュ通知として MQTT メッセージからデータを送信します。

SNS アクションを使用してルールを作成およびテストする方法を示すチュートリアルに従うことができます。詳細については、「チュートリアル: Amazon SNS 通知の送信」を参照してください。

> **Note**
>
> SNS アクションは、Amazon SNS FIFO (First-In-First-Out) トピックをサポートしていません。ルールエンジンは完全に分散されたサービスであるため、SNS アクションが呼び出されたときのメッセージ順序の保証はありません。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールsns:Publish。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- AWS KMS カスタマーマネージドマネージドを使用して Amazon SNS AWS KMS key に保管中のデータを暗号化する場合、サービスには AWS KMS key 発信者に代わって を使用するアクセス許可が必要です。詳細については、[Amazon Simple Notification Service Developer Guide](Amazon 簡易通知サービスデベロッパーガイド)の[Key management](キー管理)を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`targetArn`

プッシュ通知の送信先として指定する SNS トピックまたは個々のデバイス。

置換テンプレートをサポート: API および AWS CLI のみ

`messageFormat`

(オプション) メッセージ形式。Amazon SNS ではこの設定を使用して、ペイロードを解析して関連するプラットフォーム固有の部分をペイロードから抽出するかどうかを判断します。有効な値: `JSON`、`RAW`。デフォルトは `RAW` です。

置換テンプレートをサポート: いいえ

`roleArn`

SNS へのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで SNS アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して SNS アクションを定義します。

```
{
    "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"
                }
            }
```

```
        ]
    }
}
```

関連情報

- Amazon Simple Notification Service デベロッパーガイドの Amazon Simple Notification Service とは
- チュートリアル: Amazon SNS 通知の送信

SQS

SQS (sqs) アクションは、Amazon Simple Queue Service (Amazon SQS) キューに MQTT メッセージのデータを送信します。

> **Note**
>
> SQS アクションは、Amazon SQS FIFO (First-In-First-Out) キューをサポートしていません。ルールエンジンは完全に分散されたサービスであるため、SQS アクションがトリガーされたときのメッセージ順序の保証はありません。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールsqs:SendMessage。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。
- AWS KMS カスタマーマネージドを使用して Amazon SQS で保管中のデータを AWS KMS key 暗号化する場合、サービスには AWS KMS key 発信者に代わって を使用するアクセス許可が必要です。詳細については、Amazon Simple Storage Service デベロッパーガイドのキー管理を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`queueUrl`

データの書き込み先として指定する Amazon SQS キューの URL。この URL のリージョンは、AWS IoT ルール AWS リージョン と同じである必要はありません。

> **Note**
>
> SQS ルールアクションを使用したデータ転送クロス AWS リージョン には追加料金が発生する場合があります。詳細については、「Amazon SQSの料金」を参照してください。

置換テンプレートをサポート: API および AWS CLI のみ

`useBase64`

このパラメータを `true` に設定して、データを Amazon SQS キューに書き込む前にメッセージデータを base64 エンコードするルールアクションを設定します。デフォルトは `false` です。

置換テンプレートをサポート: いいえ

`roleArn`

Amazon SQS キューへのアクセスを許可する IAM ロール。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで SQS アクションを定義します。

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

次の JSON の例では、 AWS IoT ルールで置換テンプレートを使用して SQS アクションを定義します。

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

関連情報

- Amazon Simple Queue Service デベロッパーガイドの Amazon Simple Queue Service とは

ステップ関数

Step Functions (`stepFunctions`) アクションは AWS Step Functions ステートマシンを起動します。

要件

このルールアクションには、以下の要件があります。

- オペレーションを実行するために が引き受け AWS IoT ることができる IAM ロールstates:StartExecution。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールで、このルールアクションを実行することを に許可 AWS IoT するロールを選択または作成できます。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

`stateMachineName`

開始する Step Functions ステートマシンの名前。

置換テンプレートをサポート: API および AWS CLI のみ

`executionNamePrefix`

(オプション) ステートマシンの実行に指定される、このプレフィックスとそれに続く UUID で構成される名前。Step Functions は、各ステートマシンの実行用に一意の名前を作成します (指定されていない場合)。

置換テンプレートをサポート: はい

`roleArn`

ステートマシンを起動する AWS IoT アクセス許可を付与するロールの ARN。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

例

次の JSON の例では、 AWS IoT ルールで Step Functions アクションを定義します。

```
{
```

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

関連情報

- デAWS Step Functions ベロッパーガイドの「AWS Step Functionsとは」

Timestream

Timestream ルールアクションは、MQTT メッセージの属性 (メジャー) を Amazon Timestream テーブルに書き込みます。Amazon Timestream の詳細については、Amazon Timestream とは を参照してください。

> **Note**
>
> Amazon Timestream は、すべての AWS リージョンで利用できるわけではありません。リージョンで Amazon Timestream を利用できない場合は、ルールアクションのリストには表示されません。

このルールが Timestream データベースに保存する属性は、ルールのクエリステートメントから得られた属性です。クエリステートメントの結果の各属性の値は、そのデータタイプを (the section called “DynamoDBv2” アクションにおけるのと同様に) 推測するために解析されます。各属性の値は、Timestream テーブルの独自のレコードに書き込まれます。属性のデータ型を指定または変更するには、クエリステートメントで`cast()`関数を使用します。各 Timestream レコードの内容の詳細については、「the section called “Timestream レコードコンテンツ”」を参照してください。

> **Note**
>
> SQL V2 (2016-03-23) では、`10.0` などの整数である数値は整数表現 (`10`) に変換されます。cast() 関数を使用するなどして、それらを `Decimal`値に明示的にキャストしても、この動作は妨げられず、結果は `Integer` 値のままです。これにより、データが Timestream データベースに記録されるのを妨げる、タイプの不一致エラーが発生する可能性があります。整数の数値を `Decimal` 値として処理するには、ルールクエリステートメントに SQL V1 (2015-10-08) を使用します。

> **Note**
>
> タイムストリームルールアクションが Amazon Timestream テーブルに書き込むことができる値の最大数は 100 です。詳細については、Amazon Timestream クォータのリファレンスを参照してください。

要件

このルールアクションには、以下の要件があります。

- が `timestream:DescribeEndpoints`および `timestream:WriteRecords`オペレーションを実行するために引き受け AWS IoT ることができる IAM ロール。詳細については、「必要なアクセスを AWS IoT ルールに付与する」を参照してください。

  AWS IoT コンソールでは、このルールアクションを実行 AWS IoT することを に許可するロールを選択、更新、または作成できます。
- Timestream で顧客 AWS KMS を使用して保管中のデータを暗号化する場合、サービスには AWS KMS key 発信者に代わって を使用するアクセス許可が必要です。詳細については、「 AWS サービスが KMS AWS を使用する方法」を参照してください。

パラメータ

このアクションで AWS IoT ルールを作成するときは、次の情報を指定する必要があります。

databaseName

このアクションが作成したレコードを受信するテーブルを持つ Amazon Timestream データベースの名前。「**tableName**」も参照してください。

置換テンプレートをサポート: API および AWS CLI のみ

dimensions

各メジャーレコードに書き込まれる時系列のメタデータ属性。例えば、EC2 インスタンスの名前とアベイラビリティーゾーン、または風力タービンの製造元の名前はディメンションです。

name

メタデータディメンション名。これはデータベーステーブルレコード内の列の名前です。

ディメンションに、`measure_name`、`measure_value`、`time` の名前を付けることはできません。これらの名前は予約されています。ディメンション名は、`ts_` または `measure_value` で始めることはできず、コロン (`:`) 文字を含めることはできません。

置換テンプレートをサポート: いいえ

value

データベースレコードのこの列に書き込む値。

置換テンプレートをサポート: はい

roleArn

Timestream データベーステーブルに書き込むためのアクセス許可を AWS IoT に付与するロールの Amazon リソースネーム (ARN)。詳細については、「要件」を参照してください。

置換テンプレートをサポート: いいえ

tableName

メジャーレコードを書き込むデータベーステーブルの名前。「**databaseName**」も参照してください。

置換テンプレートをサポート: API および AWS CLI のみ

timestamp

エントリのタイムスタンプに使用する値。空白の場合、エントリが処理された時刻が使用されます。

`unit`

`value` で説明されている式から得られるタイムスタンプ値の精度。

有効な値: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`。デフォルトは`MILLISECONDS`です。

`value`

長いエポック時間の値を返す式。

the section called “time_to_epoch(String, String)” 関数を使用して、メッセージペイロードで渡される日付または時刻の値から有効なタイムスタンプを作成できます。

Timestream レコードコンテンツ

このアクションによって Amazon Timestream テーブルに書き込まれるデータには、タイムスタンプ、Timestream ルールアクションからのメタデータ、およびルールのクエリステートメントの結果が含まれます。

クエリステートメントの結果の各属性 (メジャー) について、このルールアクションは、これらの列を持つ指定された Timestream テーブルにレコードを書き込みます。

列名	属性タイプ	値	コメント
dimension-name	ディメンション	Timestream ルールアクションエントリで指定された値。	ルールアクションエントリで指定された各 [Dimension] (ディメンション) は、ディメンションの名前で Timestream データベースに列を作成します。
measure_name	MEASURE_NAME	属性の名前	`measure_value::` *data-type* 列で値が指定されているクエリステートメントの結果の属性の名前。

列名	属性タイプ	値	コメント
measure_value::*data-type*	MEASURE_VALUE	クエリステートメントの結果に含まれる属性の値。属性の名前は measure_name 列にあります。	値は解釈*され、次に対する最適なマッチとしてキャストされます: bigint、boolean、double、Timestream は、データ型ごとに個別の列を作成します。メッセージ内の値は、ルールのクエリステートメントで cast() 関数を使用して別のデータ型にキャストできます。
time	TIMESTAMP	データベース内のレコードの日時。	この値は、ルールエンジンまたは timestamp プロパティ (定義されている場合) によって割り当てられます。

* メッセージペイロードから読み取られた属性値は以下のように解釈されます。これらの各ケースの図については、「the section called “例”」を参照してください。

- true または false の引用符で囲まれていない値は、boolean タイプとして解釈されます。
- 10 進数値は double タイプとして解釈されます。
- 小数点のない数値は bigint タイプとして解釈されます。
- 引用符で囲まれた文字列は varchar タイプとして解釈されます。
- オブジェクトと配列の値は JSON 文字列に変換され、varchar タイプとして保存されます。

例

次の JSON の例では、 ルールで置換テンプレートを使用して Timestream AWS IoT ルールアクションを定義します。

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

前の例で定義された Timestream トピックルールアクションを次のメッセージペイロードで使用すると、Amazon Timestream レコードが次の表に書き込まれます。

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

次の表は、指定されたトピックルールアクションを使用して以前のメッセージペイロードが作成したものの処理を行うデータベースの列とレコードを示しています。device_firmware_sku および device_id 列は、トピックルールアクションで定義された DIMENSIONS です。Timestream トピックルールアクションは、time 列と measure_name および measure_value::* 列を作成し、トピックルールアクションのクエリステートメントの結果からの値を入力します。

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	time
静的メタデータ	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
静的メタデータ	iotconsole-159EXAMPLE738-0	integer_value_as_string	-	123456789012	-	-	2020-08-26 22:42:16.423000000
静的メタデータ	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	time
静的メタデータ	iotconsole-159EXAMPLE738-0	integer_value	123456789012	-	-	-	2020-08-26 22:42:16.423000000
静的メタデータ	iotconsole-159EXAMPLE738-0	string_value	-	文字列値	-	-	2020-08-26 22:42:16.423000000
静的メタデータ	iotconsole-159EXAMPLE738-0	array_of_integers	-	[23,36,56,72]	-	-	2020-08-26 22:42:16.423000000
静的メタデータ	iotconsole-159EXAMPLE738-0	文字列の配列	-	["red","green","blue"]	-	-	2020-08-26 22:42:16.423000000
静的メタデータ	iotconsole-159EXAMPLE738-0	boolean_value_as_string	-	TRUE	-	-	2020-08-26 22:42:16.423000000
静的メタデータ	iotconsole-159EXAMPLE738-0	double_value	-	-	123.456789012	-	2020-08-26 22:42:16.423000000
静的メタデータ	iotconsole-159EXAMPLE738-0	double_value_as_string	-	123.45679	-	-	2020-08-26 22:42:16.423000000

ルールのトラブルシューティング

ルールに問題が発生した場合は、CloudWatch Logs を有効にすることをお勧めします。ログを分析して、認可に関する問題かどうか、WHERE 句の条件に一致する結果が見つからない問題かどうかなどを判断できます。詳細については、CloudWatch Logs のセットアップを参照してください。

AWS IoT ルールを使用したクロスアカウントリソースへのアクセス

クロスアカウントアクセスの AWS IoT ルールを設定して、あるアカウントの MQTT トピックに取り込まれたデータを別のアカウントの Amazon SQS や Lambda などの AWS サービスにルーティングできます。次に、あるアカウントの MQTT トピックから別のアカウントの宛先へのクロスアカウントデータ取り込みの AWS IoT ルールを設定する方法について説明します。

クロスアカウントルールは、宛先リソースのリソースベースのアクセス許可を使用して設定できます。したがって、 AWS IoT ルールを使用してクロスアカウントアクセスを有効にすることができるのは、リソースベースのアクセス許可をサポートする送信先のみです。サポートされる宛先には、Amazon SQS、Amazon SNS、Amazon S3、および AWS Lambdaがあります。

> **Note**
>
> Amazon SQS を除くサポートされている送信先については、ルールアクションがそのリソースとやり取りできるように、別のサービスのリソース AWS リージョン と同じ でルールを定義する必要があります。 AWS IoT ルールアクションの詳細については、AWS IoT 「 ルールアクション」を参照してください。ルールの SQS アクションの詳細については、「???」を参照してください。

前提条件

- AWS IoT ルールに詳しいこと
- IAM ユーザー,ロール,およびリソースベースのアクセス許可を理解していること
- AWS CLI がインストールされていること

Amazon SQS のクロスアカウント設定

シナリオ: アカウント A が、MQTT メッセージからアカウント B の Amazon SQS キューにデータを送信します。

AWS アカウント	アカウントの呼び方	説明
1111-1111-1111	アカウント A	ルールアクション: `sqs:SendMessage`
2222-2222-2222	アカウント B	Amazon SQS キュー • ARN: *arn:aws:sqs:region:2222-2222-2222:ExampleQueue* • URL: *https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue*

> **Note**
>
> 送信先の Amazon SQS キューがAWS IoT ルール AWS リージョン と同じ に存在する必要はありません。ルールの SQS アクションの詳細については、「???」を参照してください。

アカウント A のタスクを実行する

> **メモ**
>
> 次のコマンドを実行するには、リソースがルールの Amazon リソースネーム (ARN) である `iot:CreateTopicRule` へのアクセス許可と、リソースがロールの ARN である `iam:PassRole` アクションへのアクセス許可が IAM ユーザーに必要です。

1. アカウント A の IAM ユーザーを使用して、AWS CLIを設定します。
2. AWS IoT ルールエンジンを信頼し、アカウント B の Amazon SQS キューへのアクセスを許可するポリシーをアタッチする IAM ロールを作成します。「必要なアクセス権を付与する」の「コマンドとポリシードキュメントの例 AWS IoT」を参照してください。
3. トピックにアタッチされるルールを作成するために、create-topic-rule コマンドを実行します。

```
aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
```

次のペイロードファイル例では、iot/test トピックに送信されたすべてのメッセージを指定の Amazon SQS キューに挿入するルールが指定されています。SQL ステートメントによってメッセージがフィルター処理され、ロールの ARN によって Amazon SQS キューにメッセージを追加するアクセス許可が AWS IoT に付与されます。

```
{
 "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 ルールで Amazon SQS アクションを定義する方法の詳細については、AWS IoT「ルールアクション - Amazon SQS」を参照してください。

アカウント B のタスクを実行する

1. アカウント B の IAM ユーザーを使用して、AWS CLIを設定します。
2. Amazon SQS キューリソースへのアクセス許可をアカウント A に付与するために、add-permission コマンドを実行します。

```
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 SNS のクロスアカウント設定

シナリオ: アカウント A が、MQTT メッセージからアカウント B の Amazon SNS トピックにデータを送信します。

AWS アカウント	アカウントの呼び方	説明
1111-1111-1111	アカウント A	ルールアクション: `sns:Publish`
2222-2222-2222	アカウント B	Amazon SNS トピックの ARN: *arn:aws:sns:region:2222-2222-2222:ExampleTopic*

アカウント A のタスクを実行する

> **注意事項**
>
> 次のコマンドを実行するには、リソースがルールの ARN である `iot:CreateTopicRule` へのアクセス許可と、リソースがロールの ARN である `iam:PassRole` アクションへのアクセス許可が IAM ユーザーに必要です。

1. アカウント A の IAM ユーザーを使用して、AWS CLIを設定します。
2. AWS IoT ルールエンジンを信頼し、アカウント B の Amazon SNS トピックへのアクセスを許可するポリシーをアタッチする IAM ロールを作成します。コマンドとポリシードキュメントの例については、「必要なアクセス権 AWS IoT の付与」を参照してください。
3. トピックにアタッチされるルールを作成するために、create-topic-rule コマンドを実行します。

   ```
   aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
   ```

   次のペイロードファイル例では、`iot/test` トピックに送信されたすべてのメッセージを指定の Amazon SNS トピックに挿入するルールが指定されています。SQL ステートメントによってメッセージがフィルター処理され、ロールの ARN によって Amazon SNS トピックへのアクセス許可が AWS IoT に付与されます。

   ```
   {
    "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 ルールで Amazon SNS アクションを定義する方法の詳細については、AWS IoT 「ルールアクション - Amazon SNS」を参照してください。

アカウント B のタスクを実行する

1. アカウント B の IAM ユーザーを使用して、AWS CLIを設定します。
2. Amazon SNS トピックリソースに対するアクセス許可をアカウント A に付与するために、add-permission コマンドを実行します。

```
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 S3 のクロスアカウント設定

シナリオ: アカウント A が、MQTT メッセージからアカウント B の Amazon S3 バケットにデータを送信します。

AWS アカウント	アカウントの呼び方	説明
1111-1111-1111	アカウント A	ルールアクション: `s3:PutObject`
2222-2222-2222	アカウント B	Amazon S3 バケットの ARN: *arn:aws:s3:::amzn-s3-demo-bucket*

アカウント A のタスクを実行する

> **メモ**
>
> 次のコマンドを実行するには、リソースがルールのARN である `iot:CreateTopicRule` に対するアクセス許可と、リソースがロールの ARN である `iam:PassRole` アクションに対するアクセス許可が IAM ユーザーに必要です。

1. アカウント A の IAM ユーザーを使用して、AWS CLIを設定します。
2. AWS IoT ルールエンジンを信頼し、アカウント B の Amazon S3 バケットへのアクセスを許可するポリシーをアタッチする IAM ロールを作成します。コマンドとポリシードキュメントの例については、「必要なアクセス権 AWS IoT の付与」を参照してください。
3. ターゲット S3 バケットにアタッチされるルールを作成するために、create-topic-rule コマンドを実行します。

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
```

次のペイロードファイル例では、`iot/test` トピックに送信されたすべてのメッセージを指定の Amazon S3 バケットに挿入するルールが指定されています。SQL ステートメントによってメッセージがフィルター処理され、ロールの ARN によって Amazon S3 バケットにメッセージを追加するアクセス許可が AWS IoT に付与されます。

```
{
 "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 ルールで Amazon S3 アクションを定義する方法の詳細については、AWS IoT 「ルールアクション - Amazon S3」を参照してください。

アカウント B のタスクを実行する

1. アカウント B の IAM ユーザーを使用して、AWS CLIを設定します。
2. アカウント A のプリンシパルを信頼するバケットポリシーを作成します。

   次のペイロードファイル例では、別のアカウントのプリンシパルを信頼するバケットポリシーを定義します。

```
{
    "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/*"
        }
    ]
}
```

   詳しくは、バケットポリシーの例を参照してください。
3. 指定したバケットにバケットポリシーをアタッチするために、put-bucket-policy コマンドを実行します。

```
aws s3api put-bucket-policy --bucket amzn-s3-demo-bucket --policy file://./amzn-s3-demo-bucket-policy.json
```

4. クロスアカウントアクセスが機能するように、パブリックアクセスをすべてブロックが正しく設定されていることを確認してください。詳しくは、Amazon S3 のセキュリティベストプラクティスを参照してください。

のクロスアカウント設定 AWS Lambda

シナリオ: アカウント A はアカウント B の AWS Lambda 関数を呼び出し、MQTT メッセージを渡します。

AWS アカウント	アカウントの呼び方	説明
1111-1111-1111	アカウント A	ルールアクション: `lambda:InvokeFunction`
2222-2222-2222	アカウント B	Lambda 関数の ARN: *arn:aws:lambda:region:2222-2222-2222:function:example-function*

アカウント A のタスクを実行する

> **注意事項**
>
> 次のコマンドを実行するには、リソースがルールのARN である `iot:CreateTopicRule` に対するアクセス許可と、リソースがロールの ARN である `iam:PassRole` アクションに対するアクセス許可が IAM ユーザーに必要です。

1. アカウント A の IAM ユーザーを使用して、AWS CLIを設定します。
2. create-topic-rule コマンドを実行して、アカウント B の Lambda 関数へのクロスアカウントアクセスを定義するルールを作成します。

   ```
   aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
   ```

   次のペイロードファイル例では、`iot/test` トピックに送信されたすべてのメッセージを指定の Lambda 関数に挿入するルールが指定されています。SQL ステートメントによってメッセージがフィルター処理され、ロールの ARN によって Lambda 関数にデータを渡すアクセス許可が AWS IoT に付与されます。

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
   ```

```
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:region:2222-2222-2222:function:example-function"
   }
  }
 ]
}
```

ルールで AWS Lambda AWS IoT アクションを定義する方法の詳細については、AWS IoT ルールアクション - Lambda を参照してください。

アカウント B のタスクを実行する

1. アカウント B の IAM ユーザーを使用して、AWS CLIを設定します。
2. Lambda の add-permission コマンドを実行して、Lambda 関数をアクティブ化するアクセス許可を AWS IoT ルールに付与します。次のコマンドを実行するには、`lambda:AddPermission` アクションに対するアクセス許可が IAM ユーザーに必要です。

```
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"
```

オプション:

--プリンシパル

このフィールドは、Lambda 関数を呼び出すアクセス許可を AWS IoT (によって表されますiot.amazonaws.com) に付与します。

--source-arn

このフィールドは、 AWS IoT の `arn:aws:iot:region:1111-1111-1111:rule/example-rule` のみがこのLambda 関数をトリガーし、同じアカウントまたは異なるアカウント内の他のルールはこの Lambda 関数をアクティブ化できないことを確認します。

--source-account

このフィールドは、 が1111-1111-1111アカウントに代わってのみこの Lambda 関数を AWS IoT アクティブ化することを確認します。

> **注意事項**
>
> AWS Lambda 関数のコンソールの [Configuration] (設定) の下に「ルールが見つかりませんでした」というエラーメッセージが表示される場合は、エラーメッセージを無視して、接続のテストに進みます。

エラー処理 (エラーアクション)

がデバイスからメッセージ AWS IoT を受信すると、ルールエンジンはメッセージがルールと一致するかどうかを確認します。一致する場合は、そのルールのクエリステートメントが評価され、ルールのアクションがアクティブ化され、クエリステートメントの結果が渡されます。

アクションをアクティブ化するときに問題が発生した場合、ルールエンジンはエラーアクションを呼び出します (ルールに指定されている場合)。次の場合に、この問題が発生することがあります。

- ルールに Amazon S3 バケットにアクセスする権限がない。
- ユーザーエラーにより、DynamoDB のプロビジョニングされたスループットを超える。

> **Note**
>
> このトピックで説明するエラー処理は、ルールアクションに関するものです。外部関数を含む SQL の問題をデバッグするには、 AWS IoT ログ記録を設定できます。詳細については、「???」を参照してください。

エラーアクションメッセージ形式

ルールとメッセージごとに 1 つのメッセージが生成されます。たとえば、同じルール内の 2 つのルールアクションが失敗した場合、エラーアクションは両方のエラーを含む 1 つのメッセージを受け取ります。

エラーアクションメッセージは次の例のようになります。

```
{
  "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

エラーアクションをトリガーしたルールの名前。

トピック

元のメッセージが受信されたトピック。

cloudwatchTraceId

CloudWatch でエラーログを参照する一意の ID。

clientId

メッセージの発行元のクライアント ID。

base64OriginalPayload

Base64 でエンコードされた元のメッセージペイロード。

エラー

failedAction

完了に失敗したアクションの名前 (たとえば「S3Action」)。

failedResource

リソースの名前 (たとえば S3 バケットの名前)。

errorMessage

エラーの記述と説明。

エラーアクションの例

次に、エラーアクションが追加されたルールの例を示します。次のルールには、メッセージデータを DynamoDB テーブルに書き込むアクションと、Amazon S3 バケットにデータを書き込むエラーアクションがあります。

```
{
    "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()}"
        }
    }
}
```

エラーアクションの SQL ステートメントでは、、aws_lambda()、get_dynamodb()、、get_registry_data()、、get_thing_shadow() get_secret() machinelearning_predict()などの外部関数を含む任意の関数または置換テンプレートを使用できますdecode()。エラーアクションで外部関数を呼び出す必要がある場合、エラーアクションを呼び出すと、外部関数に追加の請求が発生する可能性があります。

ルールとエラーアクションの指定方法の詳細については、AWS IoT 「ルールの作成」を参照してください。

CloudWatch を使用してルールの成功または失敗を監視する方法の詳細については、AWS IoT メトリクスとディメンション を参照してください。

基本的な取り込みによるメッセージングコストの削減

基本的な取り込みを使用して、メッセージングコストを発生させることなくAWS IoT ルールアクション、 で AWS のサービス サポートされている にデバイスデータを安全に送信できます。基本的な取り込みでは、取り込みパスからパブリッシュ/サブスクライブのメッセージブローカーを除外することによってデータフローが最適化されます。

基本的な取り込みでは、デバイスまたはアプリケーションからメッセージを送信できます。メッセージには、最初の 3 つのレベルに `$aws/rules/`*`rule_name`* で始まるトピック名があり、ここで *`rule_name`* は呼び出す AWS IoT ルールの名前です。

基本的な取り込みのプレフィックス (`$aws/rules/`*`rule_name`*) を、ルールを呼び出すために使用するメッセージのトピックに追加するだけで、既存のルールを基本的な取り込みで使用することができます。例えば、`Buildings/Building5/Floor2/Room201/Lights` (`"sql": "SELECT * FROM 'Buildings/#'"`) などのトピックでメッセージによって呼び出される `BuildingManager` という名前のルールがある場合、トピック `$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights` でメッセージを送信することで、基本的な取り込みで同じルールを呼び出せます。

> **Note**
>
> - デバイスとルールでは、基本的な取り込みの予約されたトピックをサブスクライブできません。たとえば、 AWS IoT Device Defender `num-messages-received`メトリクスはトピックへのサブスクライブをサポートしていないため、出力されません。詳細については、「予約済みトピック」を参照してください。
> - 複数のサブスクライバーにメッセージを配信する (たとえば、他のデバイスやルールエンジンにメッセージを配信する) パブリッシュ/サブスクライブブローカーが必要な場合は、AWS IoT 引き続きメッセージブローカーを使用してメッセージ配信を処理する必要があります。しかし、基本的な取り込みトピック以外のトピックにメッセージを公開するようにしてください。

基本的な取り込みの使用

基本的な取り込みを使用する前に、デバイスまたはアプリケーションが、$aws/rules/* に対する発行アクセス許可を持つポリシーを使用していることを確認してください。または、ポリシーに $aws/rules/*rule_name*/* で個別ルールに対するアクセス許可を指定できます。それ以外の場合は、デバイスとアプリケーションは AWS IoT Coreに対して既存の接続を引き続き使用できます。

メッセージがルールエンジンに到達すると、基本的な取り込みから呼び出されたルールと、メッセージブローカーサブスクリプションを通じて呼び出されたルールによる実装またはエラー処理に違いはありません。

基本的な取り込みで使用するためのルールを作成できます。以下に留意してください。

- 基本的な取り込みトピックの最初のプレフィックス ($aws/rules/*rule_name*) は topic(10 進数) 関数には使用できません。
- 基本的な取り込みでのみ呼び出されるルールを定義する場合、FROM 句は rule 定義の sql フィールドのオプションです。これは、メッセージブローカーを使用してメッセージを送信する必要がある他のメッセージによりルールが呼び出される場合でも (例えば、他のメッセージが複数の受信対象に配信される必要があるため) 必要になります。詳細については、「AWS IoT SQL リファレンス」を参照してください。
- 基本的な取り込みトピックの最初の 3 つのレベル ($aws/rules/*rule_name*) はトピックの 8 つのセグメント長制限または 256 文字の文字数制限にカウントされません。それ以外の場合は、「AWS IoT の制限」で説明されているように、同じ制限が適用されます。
- 無効なルールまたは存在しないルールを指定する基本的な取り込みトピックでメッセージが受信された場合は、エラーログは Amazon CloudWatch ログに作成され、デバッグに利用できます。詳細については、「」を参照してくださいルールエンジンのログエントリ RuleNotFound メトリクスが表示され、このメトリクスでアラームを作成できます。詳細については、「ルールのメトリクス」の「ルールメトリクス」を参照してください。
- この場合でも、基本的な取り込みトピックに QoS 1 で発行できます。メッセージがルールエンジンに正常に配信された後、PUBACK を受信します。PUBACK の受信はルールのアクションが正常に完了したことを意味するわけではありません。エラーアクションを設定して、アクションの実行中にエラーを処理することができます。詳細については、「エラー処理 (エラーアクション)」を参照してください。

AWS IoT SQL リファレンス

では AWS IoT、ルールは SQL のような構文を使用して定義されます。SQL ステートメントは、次の 3 種類の句で構成されます。

SET

(オプション) SQL ステートメントと置換テンプレート全体で再利用できる変数を定義します。式を使用して変数に値を割り当てます。これらの変数は、SELECT 句と WHERE 句、およびアクション置換テンプレートで参照します。

SET 句は、データ型、オペレータ、、関数リテラル、Case ステートメント、、JSON 拡張変数、および をサポートしますネストされたオブジェクトのクエリ。

SELECT

(必須) 受信メッセージのペイロードから情報を抽出し、その情報に対して変換を実行します。使用するメッセージは、FROM 句で指定されたトピックフィルターによって識別されます。

SELECT 句は、データ型、、オペレータ、関数、リテラルCase ステートメント、JSON 拡張、置換テンプレート、、変数、ネストされたオブジェクトのクエリ、および をサポートしていますバイナリペイロード。

FROM

データを抽出するメッセージを識別する MQTT メッセージトピックフィルター。ここで指定されたトピックフィルタに一致する MQTT トピックに送信されるメッセージごとに、ルールがアクティブ化されます。メッセージブローカーを通過するメッセージによってアクティブ化されるルールに必要です。基本的な取り込み機能を使用してのみアクティブ化されるルールの場合はオプションです。

WHERE

(オプション) ルールで指定されたアクションが実行されるかどうかを決定する条件付きロジックを追加します。

WHERE 句は、データ型、オペレータ、、関数リテラル、Case ステートメント、JSON 拡張、変数、および をサポートしますネストされたオブジェクトのクエリ。

SQL ステートメントの例は、次のようになります。

```
SELECT color AS rgb FROM 'topic/subtopic' WHERE temperature > 50
```

MQTT メッセージ (入力ペイロードとも呼ばれる) の例は、次のようになります。

```
{
    "color":"red",
    "temperature":100
}
```

このメッセージが 'topic/subtopic' トピックに発行された場合、ルールはトリガーされ、SQL ステートメントは評価されます。SQL ステートメントでは、color プロパティが 50 より大きい場合は、"temperature" プロパティの値を抽出します。WHERE 句は、条件 temperature > 50 を指定します。AS キーワードは "color" プロパティの名前を "rgb" に変更します。結果は、(出力ペイロードとも呼ばれる) 次のようになります。

```
{
    "rgb":"red"
}
```

このデータは、さらなる処理のためにデータを送信するルールアクションに転送されます。ルールアクションの詳細については、「AWS IoT ルールアクション」を参照してください。

> **Note**
>
> コメントは、現在 AWS IoT SQL 構文ではサポートされていません。
> 空白を含む属性名は、SQL ステートメントのフィールド名として使用できません。受信ペイロードにはスペースを含む属性名を使用できますが、SQL ステートメントではそのような名前を使用できません。ただし、ワイルドカード (*) フィールド名指定を使用すると、送信ペイロードに渡されます。

SELECT 句

AWS IoT SELECT 句は基本的に ANSI SQL SELECT 句と同じですが、いくつかの小さな違いがあります。

SELECT 句は、データ型、、オペレータ、関数リテラル、Case ステートメント、、JSON 拡張変数、ネストされたオブジェクトのクエリ、および をサポートしていますバイナリペイロード。

SELECT 句を使用して、受信 MQTT メッセージから情報を抽出できます。SELECT * を使用して、受信メッセージのペイロード全体を取得することもできます。以下に例を示します。

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT * FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50}
```

ペイロードが JSON オブジェクトの場合、オブジェクトのキーを参照できます。出力ペイロードにはキーと値のペアが含まれます。以下に例を示します。

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT color FROM 'topic/subtopic'
Outgoing payload: {"color":"red"}
```

AS キーワードを使用してキーの名前を変更できます。以下に例を示します。

```
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"}
```

カンマで区切ることで、複数の項目を選択できます。以下に例を示します。

```
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}
```

「*」を含む複数の項目を選択して、受信ペイロードに項目を追加できます。以下に例を示します。

```
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" キーワードを使用して、JSON オブジェクトではない出力ペイロードを作成できます。SQL バージョン 2015-10-08 では、選択できる項目は 1 つのみです。SQL バージョン 2016-03-23 以降では、最上位オブジェクトとして出力する配列を選択することもできます。

Example

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT VALUE color FROM 'topic/subtopic'
Outgoing payload: "red"
```

'.' 構文を使用して、受信ペイロードで入れ子になっている JSON オブジェクトを詳細に調べることができます。以下に例を示します。

```
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}
```

数字やハイフン (マイナス) 文字などの予約文字を含む JSON オブジェクトおよびプロパティ名の使用方法については、「JSON 拡張」を参照してください。

関数を使用して受信ペイロードを変換できます (関数 を参照)。グループ化するには括弧を使用します。以下に例を示します。

```
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 句

FROM 句はルールをトピックやトピックのフィルターに受信登録します。トピックまたはトピックフィルタは、一重引用符 (') で囲みます。ここで指定されたトピックフィルタに一致する MQTT トピックに送信されるメッセージごとに、ルールがトリガーされます。トピックのフィルタを使って、類似のトピックのグループにサブスクライブできます。

例:

トピックに公開された受信ペイロード 'topic/subtopic': {temperature: 50}。

トピックに公開された受信ペイロード 'topic/subtopic-2': {temperature: 50}。

SQL: "SELECT temperature AS t FROM 'topic/subtopic'"。

ルールが 'topic/subtopic' に受信登録されるため、受信ペイロードがルールに渡されます。ルールアクションに渡される発信ペイロードは、{t: 50} です。ルールが 'topic/subtopic-2' に受信登録されていないので、ルールは 'topic/subtopic-2' で公開されるメッセージにトリガーされません。

ワイルドカードの例:

「#」(複数レベル) ワイルドカード文字を使用して、1 つ以上の特定のパス要素に一致させることができます。

トピックに公開された受信ペイロード 'topic/subtopic': {temperature: 50}。

トピックに公開された受信ペイロード 'topic/subtopic-2': {temperature: 60}。

トピックに公開された受信ペイロード 'topic/subtopic-3/details': {temperature: 70}。

トピックに公開された受信ペイロード 'topic-2/subtopic-x': {temperature: 80}。

SQL: "SELECT temperature AS t FROM 'topic/#'"。

ルールは 'topic' で始まるすべてのトピックにサブスクライブされるため、3 回実行され、{t: 50} (topic/subtopic)、{t: 60} (topic/subtopic-2)、および {t: 70} (topic/subtopic-3/details) の送信ペイロードがアクションに送信されます。'topic-2/subtopic-x' に受信登録されていないので、{temperature: 80} のメッセージにルールはトリガーされません。

+ ワイルドカードの例:

「+」(単数レベル) ワイルドカード文字を使用して、1 つの特定のパス要素に一致させることができます。

トピックに公開された受信ペイロード 'topic/subtopic': {temperature: 50}。

トピックに公開された受信ペイロード 'topic/subtopic-2': {temperature: 60}。

トピックに公開された受信ペイロード 'topic/subtopic-3/details': {temperature: 70}。

トピックに公開された受信ペイロード 'topic-2/subtopic-x': {temperature: 80}。

SQL: "SELECT temperature AS t FROM 'topic/+'"。

ルールは、2 つのパス要素のあるすべてのトピックに受信登録されていて、最初の要素は 'topic' です。ルールは 'topic/subtopic' および 'topic/subtopic-2'に送信されたメッセージに対して実行されますが、'topic/subtopic-3/details' (トピックフィルターよりも多くのレベルがある) または 'topic-2/subtopic-x' (topic で始まらない) に対しては実行されません。

SET 句

SET 句を使用して、式の結果を保存する変数を定義します。これらの変数は、SELECT 句と WHERE 句、および置換テンプレートで再利用できます。これにより、複雑な式が重複するのを防ぎ、SQL ステートメント内の関数呼び出しの数を減らすことができます。

SET 句は、データ型、、オペレータ、関数リテラル、Case ステートメント、JSON 拡張、変数、および をサポートしますネストされたオブジェクトのクエリ。

SET 句構文

SET 句は、SQL ステートメントの SELECT 句の前に表示する必要があります。次の構文を使用します。

```
SET @variable_name = expression [, @variable_name2 = expression2]
```

構文ルール:

- で変数名を開始する @
- 変数名には、文字、数字、アンダースコアを含めることができます
- 変数名は最大 64 文字です
- 複数の変数をカンマで区切って 1 つの SET 句に設定できます。
- 各変数は 1 回のみ割り当てることができます (変数はイミュータブルです)
- SET キーワードは SQL ステートメントごとに 1 回のみ使用できます

変数の使用

変数を定義したら、次の場所で使用できます。

- SELECT 句
- WHERE 句
- その他の SET 変数の割り当て
- アクション置換テンプレート
- エラーアクション置換テンプレート
- ネストされた SELECT クエリ
- 関数パラメータ (roleArn パラメータなどの特定のパラメータ、および のような関数のモードを切り替えるパラメータは変数をサポート`transform("enrichArray", attributes, values)`していません)

変数は、SET 句で使用されるのと同じ`@variable_name`構文を使用して参照されます。JSON 拡張構文を使用して、 などのオブジェクトを含む変数のプロパティにアクセスすることもできます`@variable_name.property`。

SET 句の例

基本的な変数の使用法

次の例は、トピック で公開されたペイロードを示していますdevice/data。
`{"temp_fahrenheit": 75, "humidity": 60}`

SQL ステートメント:

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

送信ペイロード: `{"celsius": 23.89, "humidity": 60}`

埋め込み JSON オブジェクトのメンバーにアクセスする

次の例は、トピック で公開されたペイロードを示していますdevice/data。 `{"device1": {"deviceId":"weather_sensor", "deviceData": {"sensors": {"temp_fahrenheit": 75, "humidity": 60}, "location": [47.606,-122.332]}}}`

SQL ステートメント:

```
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'
```

送信ペイロード:
`{"temp_fahrenheit":75,"humidity":60,"deviceId":"weather_sensor"}`

JSON 拡張機能の操作方法の詳細については、「」を参照してください。 JSON 拡張

重複する関数呼び出しの回避

SET 変数は、複雑なデコードオペレーションの重複を回避するのに役立ちます。

```
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 変数がない場合、関数呼び出し制限を超えるデコード関数を 3 回繰り返す必要があります。

複数の変数

カンマで区切ることで、1 つの SET 句に複数の変数を定義できます。

```
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
```

置換テンプレートでの変数の使用

変数はアクション置換テンプレートでも使用できます。これにより、SQL ステートメントとルールアクションの両方で計算された値を再利用できます。

SQL ステートメント:

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

アクション設定:

```
{
  "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/testRuleRole",
    "bucketName": "bucket",
    "key": "temperature-data/${device_id}/temp-${@temp_celsius}C.json"
  }
}
```

この例では、SET 変数`@temp_celsius`が置換テンプレートで使用され、S3 アクションのキーフィールドが作成されます。

JSON 以外のペイロードの使用

SET 変数は非 JSON ペイロードを直接サポートしないため、ペイロードは最初にエンコードまたはデコードする必要があります。

```
SET @encoded_payload = encode(*, 'base64')
SELECT @encoded_payload AS raw_data FROM 'device/binary'
```

JSON 以外のペイロードの操作方法の詳細については、「」を参照してください。 バイナリペイロードの使用

SET 句の制限

SET 変数には、次の制限が適用されます。

- SQL ステートメントあたり最大 10 個の一意の変数
- 最大変数値サイズは 128 KiB (UTF-8 JSON 文字列の最小化)
- すべての変数の最大合計値サイズは 128 KiB です
- 変数名は 64 文字に制限されています
- 変数は JSON ペイロードをそのまま直接受け入れることができます (非 JSON ペイロードは最初にエンコード/デコードする必要があります)

WHERE 句

WHERE 句は、ルールで指定されたアクションが実行されるかどうかを決定します。WHERE 句が true と評価する場合、ルールアクションが実行されます。それ以外の場合、ルールアクションは実行されません。

WHERE 句は、データ型、、オペレータ、関数リテラル、Case ステートメント、JSON 拡張、変数、および をサポートしますネストされたオブジェクトのクエリ。

例:

`topic/subtopic` に公開された受信ペイロード: `{"color":"red", "temperature":40}`。

SQL: `SELECT color AS my_color FROM 'topic/subtopic' WHERE temperature > 50 AND color <> 'red'`。

この場合、ルールはトリガーされますが、ルールで指定されたアクションは実行されません。出力ペイロードはありません。

WHERE 句の関数と演算子を使用できます。ただし、SELECT に AS キーワードで作成された エイリアスを参照することはできません。(SELECT が評価されたかどうかを判断するため、WHERE 句は最初に評価されます。)

非 JSON ペイロードのある例:

`topic/subtopic` で公開された受信非 JSON ペイロード: `80`

SQL: ``SELECT decode(encode(*, 'base64'), 'base64') AS value FROM 'topic/subtopic' WHERE decode(encode(*, 'base64'), 'base64') > 50``

この場合、ルールはトリガーされますが、ルールで指定されたアクションは実行されません。送信ペイロードは SELECT 句によって JSON ペイロード `{"value":80}` として変換されます。

データ型

AWS IoT ルールエンジンは、すべての JSON データ型をサポートしています。

サポートされているデータの種類

タイプ	意味
`Int`	個別の `Int`。最大 34 桁。
`Decimal`	0 以外の最小サイズ 1E-999 および最大サイズ 9.999...E999 の、精度が 34 桁の `Decimal`。 **Note** 一部の関数は、34 桁の精度ではなく、倍精度の `Decimal` の値を返します。SQL V2 (2016-03-23) では、`10.0` などの整数である数値は、想定される `Decimal` 値 (`10.0`) ではなく `Int` 値 (`10`) として処理されます。整数の数値を `Decimal` 値として確実に処理するには、ルールクエリステートメントに SQL V1 (2015-10-08) を使用します。
`Boolean`	`True`-または-`False`
`String`	A UTF-8 文字列。
`Array`	同じ型を持つ必要のない一連の値。
`Object`	キーと値で構成される JSON 値。キーは文字列である必要があります。値は任意のタイプにできます。
`Null`	`Null` JSON で定義されます。値がないことを表す、実際の値です。SQL ステートメントの

タイプ	意味
	Null キーワードを使用して、Null の値を明示的に作成できます。例: "SELECT NULL AS n FROM 'topic/subtopic'"

Undefined	値ではなく。値を省略しない限り、これは、JSON では明示的には表現できません。たとえば、{"foo": null} のオブジェクトでは、キー「foo」は NULL を返しますが、キー「bar」は Undefined を返します。内部的には、SQL 言語は、Undefined を値として処理しますが、JSON では表現ができないため、JSON にシリアル化される場合は、結果は Undefined になります。
	
	`{"foo":null, "bar":undefined}`
	
	JSON に以下のようにシリアル化されます。
	
	`{"foo":null}`
	
	同様に、単独でシリアル化すると、Undefined は空の文字列に変換されます。無効な引数 (例: 間違った種類、間違った数値の引数など) を使用して呼び出された関数は Undefined を返します。

変換

次の表では 1 つのタイプの値が別のタイプに変換されると (正しくない型の値が関数に付与された場合) 結果を表示します。たとえば、絶対値関数「abs」(Int または Decimal を想定します) が String に与えられると、以下のルールにしたがって、String を Decimal に変換するように試みます。この場合、「abs("-5.123")」は「abs(-5.123)」として処理されます。

> **Note**
>
> Array、Object、Null、Undefined へ試みられた変換はありません。

Decimal に

引数の型	結果
Int	小数点のない Decimal。
Decimal	ソース値。
Boolean	Undefined 。(キャスト関数を明示的に使用して、true = 1.0、false = 0.0. を変換できます。)
String	SQL エンジンは、文字列を Decimal. AWS IoT attempts として解析し、正規表現に一致する文字列を解析しようとします^-?\d+(\.\d+)?((?i)E-?\d+)?$ 。「0」、「-1.2」、「5E-12」は、Decimal に自動的に変換される文字列の例です。
配列	Undefined .
オブジェクト	Undefined .
Null	Null.
未定義	Undefined .

Int に

引数の型	結果
Int	ソース値。
Decimal	最も近い Int に丸められたソース値。

引数の型	結果
Boolean	Undefined 。(キャスト関数を明示的に使用して、true = 1.0、false = 0.0. を変換できます。)
String	SQL エンジンは、文字列を Decimal. AWS IoT attempts として解析し、正規表現に一致する文字列を解析しようとします^-?\d+(\.\d+)?((?i)E-?\d+)?$ 。「0」、「-1.2」、「5E-12」はすべて、 を String に変換するために Decimals. AWS IoT attempts に自動的に変換されDecimal、その小数点以下を切り捨てDecimalて を作成しますInt。
配列	Undefined .
オブジェクト	Undefined .
Null	Null.
未定義	Undefined .

ブールに

引数の型	結果
Int	Undefined (cast 関数を明示的に使用して、0 = False、any_nonzero_value = True を変換できます)。
Decimal	Undefined (キャスト関数を明示的に使用して、0 = False、any_nonzero_value = True を変換できます)。
Boolean	元の値。

引数の型	結果
String	"true" = True および "false" = False (大文字小文字の区別なし)。その他の文字列値 = Undefined 。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

文字列に

引数の型	結果
Int	標準表記の Int の文字列表現。
Decimal	おそらく科学的表記での Decimal 値の文字列表現。
Boolean	"true" または "false"。すべて小文字。
String	元の値。
配列	JSON にシリアル化された Array。結果として生じる文字列は、角括弧で囲まれたカンマ区切りリストです。String は引用符で囲まれます。Decimal、Int、Boolean、Null は引用符で囲まれません。
オブジェクト	JSON にシリアル化されたオブジェクト。結果として生じる文字列は、キーと値のペアのカンマ区切りリストで、中括弧で開始し、終了します。String は引用符で囲まれます。Decimal、Int、Boolean、Null は引用符で囲まれません。

引数の型	結果
Null	`Undefined` .
未定義	未定義。

オペレータ

次の演算子は、SELECT 句および WHERE 句で使用できます。

AND 演算子

Boolean の結果を返します。論理 AND 演算を実行します。左右オペランドが true の場合は true を返します。それ以外の場合は、false を返します。Boolean オペランド、または、大文字小文字を区別しない「true」または「false」文字列オペランドが必要です。

構文: *`expression`* `AND` *`expression`*

AND 演算子

左のオペランド	右のオペランド	出力
`Boolean`	`Boolean`	`Boolean`。両方のオペランドが true の場合は true。それ以外の場合は、false を返します。
`String/Boolean`	`String/Boolean`	すべての文字列が「true」または「false」 (大文字と小文字が区別されません) の場合、それらは `Boolean` に変換され、*`boolean`* `AND` *`boolean`* として、正常に処理されます。
その他の値	その他の値	`Undefined` .

OR 演算子

Boolean の結果を返します。論理 OR 演算を実行します。左または右オペランドのいずれかが true の場合 true を返します。それ以外の場合は、false を返します。Boolean オペランド、または、大文字小文字を区別しない「true」または「false」文字列オペランドが必要です。

構文: *`expression`* `OR` *`expression`*

OR 演算子

左のオペランド	右のオペランド	出力
Boolean	Boolean	Boolean。どちらかのオペランドが true の場合は true。それ以外の場合は、false を返します。
String/Boolean	String/Boolean	すべての文字列が「true」または「false」(大文字と小文字が区別されません) の場合、それらはブール値に変換され、*boolean* OR *boolean* として、正常に処理されます。
その他の値	その他の値	Undefined .

NOT 演算子

Boolean の結果を返します。論理 NOT 演算を実行します。オペランドが false の場合は true を返します。それ以外の場合は true を返します。Boolean オペランド、または、大文字小文字を区別しない「true」または「false」文字列オペランドが必要です。

構文: NOT *expression*

NOT 演算子

オペランド	出力
Boolean	Boolean。オペランドが false の場合は true。それ以外の場合は、true。
String	文字列が「true」または「false」(大文字と小文字は区別されません) の場合は、対応するブール値に変換され、反対の値は返されます。
その他の値	Undefined .

IN 演算子

Boolean の結果を返します。WHERE 句の IN 演算子を使用して、値が配列内の値と一致するかどうかを確認できます。一致が見つかった場合は true、それ以外の場合は false を返します。

構文: *expression* IN *expression*

IN 演算子

左のオペランド	右のオペランド	Output
Int/Decimal/String/A	Array	Integer/Decimal/String/Array/Object 要素が配列で見つかった場合は true。それ以外の場合は、false を返します。

例:

```
SQL: "select * from 'a/b' where 3 in arr"

JSON: {"arr":[1, 2, 3, "three", 5.7, null]}
```

この例では、arr という名前の配列に 3 が存在するため、条件句 `where 3 in arr` は true に評価されます。したがって、SQL ステートメントでは、`select * from 'a/b'` が実行されます。この例は、配列が異種である可能性も示しています。

EXISTS 演算子

Boolean の結果を返します。EXISTS 演算子を条件句で使用して、サブクエリ内の要素の存在をテストできます。サブクエリが 1 つ以上の要素を返す場合は true を返し、サブクエリが要素を返さない場合は false を返します。

構文: *expression*

例:

```
SQL: "select * from 'a/b' where exists (select * from arr as a where a = 3)"

JSON: {"arr":[1, 2, 3]}
```

この例では、arr という名前の配列に 3 が存在するため、条件句 `where exists (select * from arr as a where a = 3)` は true に評価されます。したがって、SQL ステートメントでは、`select * from 'a/b'` が実行されます。

例:

```
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}]}
```

この例では、JSON オブジェクト内の配列 e にオブジェクト {"foo":2} が含まれているため、条件句 `where exists (select * from e as e where foo = 2)` は true に評価されます。したがって、SQL ステートメントでは、`select * from 'a/b'` が実行されます。

> operator

Boolean の結果を返します。左のオペランドが右のオペランドより大きい場合は true を返します。両方のオペランドが、Decimal に変換され、比較されます。

構文: *expression* > *expression*

> operator

左のオペランド	右のオペランド	出力
Int/Decimal	Int/Decimal	Boolean。左のオペランドが右のオペランドより大きい場合は true。それ以外の場合は、false を返します。
String/Int/Deci	String/Int/Deci	すべての文字列を Decimal に変換できると、その後は Boolean です。左のオペランドが右のオペランドより大きい場合は true を返します。それ以外の場合は、false を返します。
その他の値	Undefined .	Undefined .

>= operator

Boolean の結果を返します。左のオペランドが右のオペランド以上の場合は true を返します。両方のオペランドが、Decimal に変換され、比較されます。

構文: *expression* >= *expression*

>= operator

左のオペランド	右のオペランド	出力
Int/Decimal	Int/Decimal	Boolean。左のオペランドが右のオペランド以上の場合は true。それ以外の場合は、false を返します。
String/Int/Deci	String/Int/Deci	すべての文字列を Decimal に変換できると、その後は Boolean です。左のオペランドが右のオペランド以上の場合は true を返します。それ以外の場合は、false を返します。
その他の値	Undefined .	Undefined .

< 演算子

Boolean の結果を返します。左のオペランドが右のオペランド未満の場合は true を返します。両方のオペランドが、Decimal に変換され、比較されます。

構文: *expression* < *expression*

< 演算子

左のオペランド	右のオペランド	出力
Int/Decimal	Int/Decimal	Boolean。左のオペランドが右のオペランド未満の場合は true。それ以外の場合は、false を返します。
String/Int/Deci	String/Int/Deci	すべての文字列を Decimal に変換できると、その後は Boolean です。左のオペランドが右のオペランド未満の場合は true を返します。それ以外の場合は、false を返します。
その他の値	Undefined	Undefined

<= 演算子

Boolean の結果を返します。左のオペランドが右のオペランド以下の場合は true を返します。両方のオペランドが、Decimal に変換され、比較されます。

構文: *expression* <= *expression*

<= 演算子

左のオペランド	右のオペランド	出力
Int/Decimal	Int/Decimal	Boolean。左のオペランドが右のオペランド以下の場合は true。それ以外の場合は、false を返します。
String/Int/Deci	String/Int/Deci	すべての文字列を Decimal に変換できると、その後は Boolean です。左のオペランドが右のオペランド以下の場合は true を返します。それ以外の場合は、false を返します。
その他の値	Undefined	Undefined

<> 演算子

Boolean の結果を返します。左右オペランドの両方が等しくない場合に true を返します。それ以外の場合は、false を返します。

構文: *expression* <> *expression*

<> 演算子

左のオペランド	右のオペランド	出力
Int	Int	左のオペランドが右のオペランドと等しくなかったら true。それ以外の場合は、false を返します。
Decimal	Decimal	左のオペランドが右のオペランドと等しくなかったら true。それ以外の場合は、false を返します。Int は比較される前に Decimal に変換されます。
String	String	左のオペランドが右のオペランドと等しくなかったら true。それ以外の場合は、false を返します。
配列	配列	各オペランドの項目が等しくなく、同じ順序でない場合、true。それ以外の場合は、false を返します。

左のオペランド	右のオペランド	出力
オブジェクト	オブジェクト	各オペランドのキーと値が等しくなかったら true。それ以外の場合は、false を返します。キーと値の順序は重要でありません。
Null	Null	False。
任意の値	`Undefined`	未定義。
`Undefined`	任意の値	未定義。
不一致の型	不一致の型	True。

= operator

`Boolean` の結果を返します。左右オペランドの両方が等しい場合に true を返します。それ以外の場合は、false を返します。

構文: *`expression`* `=` *`expression`*

= operator

左のオペランド	右のオペランド	出力
`Int`	`Int`	左のオペランドが右のオペランドと等しかったら true。それ以外の場合は、false を返します。
`Decimal`	`Decimal`	左のオペランドが右のオペランドと等しかったら true。それ以外の場合は、false を返します。`Int` は比較される前に `Decimal` に変換されます。
`String`	`String`	左のオペランドが右のオペランドと等しかったら true。それ以外の場合は、false を返します。
配列	配列	各オペランドの項目が等しく、同じ順序の場合、true。それ以外の場合は、false を返します。

左のオペランド	右のオペランド	出力
オブジェクト	オブジェクト	各オペランドのキーと値が等しかったら true。それ以外の場合は、false を返します。キーと値の順序は重要でありません。
任意の値	`Undefined`	`Undefined` .
`Undefined`	任意の値	`Undefined` .
不一致の型	不一致の型	False。

+ operator

「+」は、オーバーロードされた演算子です。文字列の連結または追加に使用できます。

構文: *`expression`* `+` *`expression`*

+ operator

左のオペランド	右のオペランド	出力
`String`	任意の値	右のオペランドを文字列に変換してから、左のオペランドの末尾に連結します。
任意の値	`String`	左のオペランドを文字列に変換して、右のオペランドを変換された左のオペランドの末尾に連結します。
`Int`	`Int`	`Int` 値。オペランドを共に追加します。
`Int/Decimal`	`Int/Decimal`	`Decimal` 値。オペランドを共に追加します。
その他の値	その他の値	`Undefined` .

- operator

左のオペランドから右のオペランドを減算します。

構文: *`expression`* `-` *`expression`*

- operator

左のオペランド	右のオペランド	出力
Int	Int	Int 値。左のオペランドから右のオペランドを減算します。
Int/Decimal	Int/Decimal	Decimal 値。左のオペランドから右のオペランドを減算します。
String/Int/Deci	String/Int/Deci	すべての文字列が小数に正しく変換されると、Decimal の値が返されます。左のオペランドから右のオペランドを減算します。それ以外の場合は Undefined を返します。
その他の値	その他の値	Undefined .
その他の値	その他の値	Undefined .

* operator

左のオペランドを右のオペランドで乗算します。

構文: *expression* * *expression*

* operator

左のオペランド	右のオペランド	出力
Int	Int	Int 値。左のオペランドを右のオペランドで乗算します。
Int/Decimal	Int/Decimal	Decimal 値。左のオペランドを右のオペランドで乗算します。
String/Int/Deci	String/Int/Deci	すべての文字列が小数に正しく変換されると、Decimal の値が返されます。左のオペランドを右のオペランドで乗算します。それ以外の場合は Undefined を返します。

左のオペランド	右のオペランド	出力
その他の値	その他の値	Undefined .

/ operator

左のオペランドを右のオペランドで除算します。

構文: *expression* / *expression*

/ operator

左のオペランド	右のオペランド	出力
Int	Int	Int 値。左のオペランドを右のオペランドで除算します。
Int/Decimal	Int/Decimal	Decimal 値。左のオペランドを右のオペランドで除算します。
String/Int/Deci	String/Int/Deci	すべての文字列が小数に正しく変換されると、Decimal の値が返されます。左のオペランドを右のオペランドで除算します。それ以外の場合は Undefined を返します。
その他の値	その他の値	Undefined .

% operator

左のオペランドを右のオペランドで除算した剰余を返します。

構文: *expression* % *expression*

% operator

左のオペランド	右のオペランド	出力
Int	Int	Int 値。左のオペランドを右のオペランドで除算した剰余を返します。

左のオペランド	右のオペランド	出力
String/Int/Deci	String/Int/Deci	すべての文字列が小数に正しく変換されると、Decimal の値が返されます。左のオペランドを右のオペランドで除算した剰余を返します。そうでない場合は 、Undefined です。
その他の値	その他の値	Undefined .

関数

SQL 式の SELECT 句または WHERE 句では、以下の組み込み関数を使用できます。

次の外部関数は、ルールアクションの aws_lambda、、get_dynamodb()get_registry_data()、および と同等の料金が請求されますget_thing_shadow()。また、Protobuf メッセージを JSON にデコードしている場合にのみ、decode() 関数に対して課金されます。詳細については、AWS IoT Core 料金表のページを参照してください。

abs(10 進数)

数値の絶対値を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例: abs(-5) は 5 を返します。

引数の型	結果
Int	Int、引数の絶対値。
Decimal	Decimal、引数の絶対値。
Boolean	Undefined .
String	Decimal。結果は、引数の絶対値です。文字列が変換できない場合、結果は Undefined です。
配列	Undefined .

引数の型	結果
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

accountid()

String としてこのルールを所有するアカウントの ID を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

accountid() = "123456789012"

acos(10 進数)

数値の逆コサインをラジアンで返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: acos(0) = 1.5707963267948966

引数の型	結果
Int	Decimal 「倍精度で」、引数の逆コサイン。仮想上の結果は、Undefined として返されます。
Decimal	Decimal 「倍精度で」、引数の逆コサイン。仮想上の結果は、Undefined として返されます。
Boolean	Undefined .
String	Decimal、引数の逆コサイン。文字列が変換できない場合、結果は Undefined で

引数の型	結果
	す。仮想上の結果は、Undefined として返されます。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

asin(10 進数)

数値の逆サインをラジアンで返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `asin(0)` = 0.0

引数の型	結果
Int	Decimal「倍精度」、引数の逆サイン。仮想上の結果は、Undefined として返されます。
Decimal	Decimal「倍精度」、引数の逆サイン。仮想上の結果は、Undefined として返されます。
Boolean	Undefined .
String	Decimal「倍精度」、引数の逆サイン。文字列が変換できない場合、結果は Undefined です。仮想上の結果は、Undefined として返されます。
配列	Undefined .

引数の型	結果
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

atan(10 進数)

数値の逆タンジェントをラジアンで返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `atan(0)` = 0.0

引数の型	結果
Int	Decimal「倍精度で」、引数の逆タンジェント。仮想上の結果は、Undefined として返されます。
Decimal	Decimal「倍精度で」、引数の逆タンジェント。仮想上の結果は、Undefined として返されます。
Boolean	Undefined .
String	Decimal、引数の逆タンジェント。文字列が変換できない場合、結果は Undefined です。仮想上の結果は、Undefined として返されます。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

atan2(10 進数、10 進数)

正の X 軸と 2 つの引数で定義された点 (x、y) の間の角度をラジアンで返します。 反時計回りの角度では正で (上半面、y > 0)、時計回りの角度では負です (下反面、y < 0)。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `atan2(1, 0)` = 1.5707963267948966

引数の型	引数の型	結果
Int/Decimal	Int/Decimal	Decimal (倍精度で)、X 軸と指 (x、y) の間の角度。
Int/Decimal/String	Int/Decimal/String	Decimal、示された点の逆タン ト。文字列が変換できない場合 Undefined です。
その他の値	その他の値	Undefined .

aws_lambda(functionArn, inputJson)

inputJson を Lambda 関数に渡して指定された Lambda 関数を呼び出し、Lambda 関数で生成された JSON を返します。

引数

引数	説明
functionArn	呼び出す Lambda 関数の ARN。Lambda 関数は JSON データを返す必要があります。
inputJson	Lambda 関数に渡される JSON 入力です。ネストされたオブジェクトクエリとリテラルを渡すには、SQL バージョン 2016-03-23 を使用する必要があります。

指定された Lambda 関数を呼び出すアクセス許可を付与 AWS IoT lambda:InvokeFunctionする必要があります。次の例は、lambda:InvokeFunction を使用して AWS CLI権限を付与する方法を示しています。

```
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"
```

以下は、add-permission コマンドの引数です。

--function-name

Lambda 関数の名前。関数のリソースポリシーを更新するための新しいアクセス許可を追加します。

--region

AWS リージョン アカウントの 。

--principal

アクセス権限を取得するプリンシパル。これは、Lambda 関数を呼び出す AWS IoT アクセス許可を付与`iot.amazonaws.com`するためです。

--source-arn

ルールの ARN。get-topic-rule AWS CLI コマンドを使用して、ルールの ARN を取得できます。

--source-account

AWS アカウント ルールが定義されている 。

--statement-id

一意のステートメント ID。

--action

このステートメントで許可する Lambda アクション。 AWS IoT が Lambda 関数を呼び出すことを許可するには、`lambda:InvokeFunction` を指定します。

> **Important**
>
> `source-arn` または を指定せずにプリン AWS IoT シパルのアクセス許可を追加するとsource-account、Lambda アクションでルールを作成する は AWS アカウン

ト 、Lambda 関数を呼び出すルールをトリガーできます AWS IoT。詳細については、[Lambda Permission Model](ラムダのアクセス許可モデル)を参照してください。

次のような JSON メッセージペイロードがあるとします。

```
{
    "attribute1": 21,
    "attribute2": "value"
}
```

aws_lambda 関数は、以下のように Lambda 関数を呼び出すために使用できます。

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function",
 {"payload":attribute1}) as output FROM 'topic-filter'
```

完全な MQTT メッセージペイロードを渡したい場合は、次の例のように、「*」を使用して JSON ペイロードを指定できます。

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function", *) as output
 FROM 'topic-filter'
```

payload.inner.element は、'topic/subtopic' トピックで発行されたメッセージからデータを選択します。

some.value は、Lambda 関数によって生成された出力からデータを選択します。

> **Note**
>
> ルールエンジンは、Lambda 関数の実行時間を制限します。ルールからの Lambda 関数の呼び出しは、2000 ミリ秒以内に完了する必要があります。

bitand(Int、Int)

2 つの Int (-converted) 引数のビット表現におけるビット単位の AND を実行します。SQL バージョン 2015-10-08 以降でサポートされています。

例: `bitand(13, 5) = 5`

引数の型	引数の型	結果
Int	Int	Int、2 つの引数のビット単位の
Int/Decimal	Int/Decimal	Int、2 つの引数のビット単位の べての非 Int 数は、最も近い I 下げられます。いずれかの引数 換できない場合、結果は Undef す。
Int/Decimal/String	Int/Decimal/String	Int、2 つの引数のビット単位の べての文字列は小数に変換され Int に切り下げられます。変換 場合、結果は Undefined です
その他の値	その他の値	Undefined .

bitor(Int、Int)

2 つの引数のビット表現のビット単位の OR を実行します。SQL バージョン 2015-10-08 以降でサポートされています。

例: `bitor(8, 5) = 13`

引数の型	引数の型	結果
Int	Int	Int、2 つの引数のビット単位の
Int/Decimal	Int/Decimal	Int、2 つの引数のビット単位の ての非 Int 数は、最も近い Int げられます。変換が失敗した場 Undefined です。
Int/Decimal/String	Int/Decimal/String	Int、2 つの引数におけるビット OR。すべての文字列は小数に変

引数の型	引数の型	結果
		も近い Int に切り下げられます 敗した場合、結果は Undefine
その他の値	その他の値	Undefined .

bitxor(Int、Int)

2 つの Int (-converted) 引数のビット表現におけるビット単位の XOR を実行します。SQL バージョン 2015-10-08 以降でサポートされています。

例: `bitor(13, 5) = 8`

引数の型	引数の型	結果
Int	Int	Int、2 つの引数におけるビット XOR。
Int/Decimal	Int/Decimal	Int、2 つの引数におけるビット XOR。非 Int 数は、最も近い I 下げられます。
Int/Decimal/String	Int/Decimal/String	Int、2 つの引数におけるビット XOR。文字列は小数に変換され Int に切り下げられます。いず が失敗した場合、結果は Undef す。
その他の値	その他の値	Undefined .

bitnot(Int)

Int (-converted) 引数のビット表現におけるビット単位の NOT を実行します。SQL バージョン 2015-10-08 以降でサポートされています。

例: `bitnot(13) = 2`

引数の型	結果
Int	Int、引数のビット単位の NOT。
Decimal	Int、引数のビット単位の NOT。Decimal の値は、最も近い Int に切り下げられます。
String	Int、引数のビット単位の NOT。文字列は小数に変換され、最も近い Int に切り下げられます。いずれかの変換が失敗した場合、結果は Undefined です。
その他の値	その他の値。

cast()

値を 1 つのデータ型から別のデータ型へ変換します。cast は通常の変換と同様に動作し、数値をブールへ/からキャストする追加機能があります。が 1 つのタイプを別のタイプにキャストする方法を決定 AWS IoT できない場合、結果は になりますUndefined。SQL バージョン 2015-10-08 以降でサポートされています。形式: cast(# as #)。

例:

```
cast(true as Int) = 1
```

cast を呼び出す際、次のキーワードが「as」の後に表示される場合があります。

SQL バージョン 2015-10-08 および 2016-03-23 向け

キーワード	結果
String	値を String へキャストします。
Nvarchar	値を String へキャストします。
Text	値を String へキャストします。
Ntext	値を String へキャストします。

キーワード	結果
varchar	値を String へキャストします。
Int	値を Int へキャストします。
整数	値を Int へキャストします。
Double	値を (倍精度で) Decimal にキャストします。

さらに、SQL バージョン 2016-03-23 向け

キーワード	結果
Decimal	値を Decimal へキャストします。
Bool	値を Boolean へキャストします。
Boolean	値を Boolean へキャストします。

キャストのルール:

Decimal へのキャスト

引数の型	結果
Int	小数点のない Decimal。
Decimal	ソース値。 **Note** SQL V2 (2016-03-23) では、10.0 などの整数である数値は、想定される Int 値 (10) の代わりに Decimal 値 (10.0) を返します。整数の数値を Decimal 値として確実にキャストするには、ルール

引数の型	結果
	クエリステートメントに SQL V1 (2015-10-08) を使用します。
Boolean	true = 1.0, false = 0.0.
String	文字列を Decimal として解析しようとします。 AWS IoT は、正規表現 (^-?\d+(\.\d+)?((?i)E-?\d+)?$) と一致する文字列を解析するよう試みます。「0」、「-1.2」、「5E-12」は、小数に自動的に変換される文字列の例です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

Int へのキャスト

引数の型	結果
Int	ソース値。
Decimal	最も近い Int へ切り下げられたソース値。
Boolean	true = 1.0, false = 0.0.
String	文字列を Decimal として解析しようとします。 AWS IoT は、正規表現 (^-?\d+(\.\d+)?((?i)E-?\d+)?$) と一致する文字列を解析するよう試みます。「0」「-1.2」「5E-12」はすべて自動的に小数に変換される文字列の例です。 AWS IoT は文字列

引数の型	結果
	を Decimal に変換するよう試み、最も近い Int へ切り下げます。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

Boolean へのキャスト

引数の型	結果
Int	0 = False, any_nonzero_value = True。
Decimal	0 = False, any_nonzero_value = True。
Boolean	ソース値。
String	"true" = True および "false" = False (大文字小文字の区別なし)。その他の文字列値 = Undefined 。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

String へのキャスト

引数の型	結果
Int	Int の文字列表現、標準表記。

引数の型	結果
Decimal	おそらく科学的表記での Decimal 値の文字列表現。
Boolean	「true」または「false」、すべて小文字。
String	ソース値。
配列	JSON へシリアル化された配列。結果の文字列は角括弧で囲まれたカンマ区切りのリストです。String は引用符で囲まれます。Decimal、Int、Boolean は囲まれません。
オブジェクト	JSON にシリアル化されたオブジェクト。JSON 文字列はキーと値のペアのカンマ区切りのリストで、最初と最後に中括弧があります。String は引用符で囲まれます。Decimal、Int、Boolean、Null は囲まれません。
Null	Undefined .
未定義	Undefined .

ceil(10 進数)

指定の Decimal を最も近い Int に切り上げます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

ceil(1.2) = 2

ceil(-1.2) = -1

引数の型	結果
Int	Int、引数値。
Decimal	Int、最も近い Decimal へ切り上げられたInt 値。
String	Int。文字列は Decimal に変換され、最も近い Int へ切り上げられます。文字列が Decimal に変換できない場合、結果は Undefined です。
その他の値	Undefined .

chr(文字列)

指定の Int 引数に対応する ASCII 文字を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

chr(65) = "A"。

chr(49) = "1"。

引数の型	結果
Int	指定された ASCII 値に対応する文字。引数が有効な ASCII 値でない場合、結果は Undefined です。
Decimal	指定された ASCII 値に対応する文字。Decimal 引数は、最も近い Int に切り下げられます。引数が有効な ASCII 値でない場合、結果は Undefined です。
Boolean	Undefined .

引数の型	結果
String	String が Decimal に変換できる場合、最も近い Int に切り下げられます。引数が有効な ASCII 値でない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
その他の値	Undefined .

clientid()

メッセージを送信している MQTT クライアントの ID を返すか、または、メッセージが MQTT クライアント経由で送られていない場合は、n/a を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

clientid() = "123456789012"

concat()

配列または文字列を連結します。この関数は、任意の数の引数を受け取り、String または Array を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

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]

引数の数	結果
0	`Undefined` .
1	引数は変更されずに返されます。
2+	引数が `Array` の場合、結果はすべての引数を含む 1 つの配列です。どの引数も配列ではなく、少なくとも 1 つの引数が `String` の場合、結果はすべての引数の `String` 表現の連結です。引数は、前にリストされた標準変換を使用して、文字列に変換されます。

cos(10 進数)

数値のコサインをラジアンで返します。`Decimal` 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`cos(0)` = 1.

引数の型	結果
`Int`	`Decimal` 「倍精度で」、引数のコサイン。仮想上の結果は、`Undefined` として返されます。
`Decimal`	`Decimal` 「倍精度で」、引数のコサイン。仮想上の結果は、`Undefined` として返されます。

引数の型	結果
Boolean	Undefined .
String	Decimal「倍精度で」、引数のコサイン。文字列が Decimal に変換できない場合、結果は Undefined です。仮想上の結果は、Undefined として返されます。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

cosh(10 進数)

数値のハイパーボリックコサインをラジアンで返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: cosh(2.3) = 5.037220649268761。

引数の型	結果
Int	Decimal「倍精度で」、引数の双曲線コサイン。仮想上の結果は、Undefined として返されます。
Decimal	Decimal「倍精度で」、引数の双曲線コサイン。仮想上の結果は、Undefined として返されます。
Boolean	Undefined .
String	Decimal「倍精度で」、引数の双曲線コサイン。文字列が Decimal に変換できない場合、結果は Undefined です。仮想

引数の型	結果
	上の結果は、Undefined として返されます。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

decode(value, decodingScheme)

decode 関数を使用して、エンコードされた値をデコードします。デコードされた文字列が JSON ドキュメントの場合、アドレス指定可能なオブジェクトが返されます。それ以外の場合、デコードされた文字列は文字列として返されます。文字列がデコードできない場合、この関数は NULL を返します。この関数は、base64 でエンコードされた文字列とプロトコルバッファ (protobuf) メッセージ形式のデコードをサポートします。

SQL バージョン 2016-03-23 以降でサポートされています。

値

文字列を返す文字列値、または AWS IoT SQL リファレンス で定義されている有効な式のいずれか。

decodingScheme

値をデコードするために使用されるスキームを表すリテラル文字列。現在 'base64' そして 'proto' のみがサポートされています。

base64 でエンコードされた文字列のデコード

この例では、メッセージペイロードにエンコードされた値が含まれています。

```
{
    encoded_temp: "eyAidGVtcGVyYXR1cmUiOiAzMyB9Cg=="
}
```

この SQL ステートメントの decode 関数は、メッセージペイロードの値をデコードします。

```
SELECT decode(encoded_temp,"base64").temperature AS temp from 'topic/subtopic'
```

encoded_temp 値をデコードすると、次の有効な JSON ドキュメントが生成され、SELECT ステートメントで温度値を読み取ることができます。

```
{ "temperature": 33 }
```

この例では、SELECT ステートメントの結果は以下のとおりです。

```
{ "temp": 33 }
```

デコードされた値が有効な JSON ドキュメントでない場合、デコードされた値は文字列として返されます。

protobuf メッセージペイロードのデコード

decode SQL 関数を使用して、protobuf メッセージペイロードをデコードできるルールを設定できます。詳細については、「protobuf メッセージペイロードのデコード」を参照してください。

> **Important**
>
> AWS IoT プリンシパルのアクセス許可を設定するsource-accountときに source-arnまたは を省略すると、 は他の AWS IoT ルールを通じてデコード関数を呼び出す AWS アカウント ことができます。関数を保護するには、「Amazon Simple Storage Service ユーザーガイド」の「バケットポリシー」を参照してください。

この関数の署名は次のようになります。

```
decode(<ENCODED DATA>, 'proto', '<S3 BUCKET NAME>', '<S3 OBJECT KEY>', '<PROTO NAME>',
 '<MESSAGE TYPE>')
```

ENCODED DATA

デコードする protobuf でエンコードされたデータを指定します。ルールに送信されるメッセージ全体が protobuf でエンコードされたデータである場合は、* を使用して raw バイナリ受信ペイロードを参照できます。それ以外の場合は、このフィールドは base-64 でエンコードされた JSON 文字列である必要があり、文字列への参照を直接渡すことができます。

1) raw バイナリ protobuf 受信ペイロードをデコードするには次のように入力します。

```
decode(*, 'proto', ...)
```

2) base64 でエンコードされた文字列「a.b」で表される protobuf エンコードされたメッセージをデコードするには、次のように入力します。

```
decode(a.b, 'proto', ...)
```

`proto`

protobuf メッセージ形式でデコードするデータを指定します。`proto` の代わりに `base64` を指定すると、この関数は base64 でエンコードされた文字列を JSON としてデコードします。

`S3 BUCKET NAME`

`FileDescriptorSet` ファイルをアップロードした Amazon S3 バケットの名前。

`S3 OBJECT KEY`

Amazon S3 バケット内の `FileDescriptorSet` ファイルを指定するオブジェクトキー。

`PROTO NAME`

`FileDescriptorSet` ファイルを生成した `.proto` ファイルの名前 (拡張子を除く)。

`MESSAGE TYPE`

`FileDescriptorSet` ファイル内の protobuf メッセージ構造の名前。デコードするデータはこの構造に従う必要があります。

decode SQL 関数を使用した SQL 式の例は次のようになります。

```
SELECT VALUE decode(*, 'proto', 's3-bucket', 'messageformat.desc', 'myproto',
 'messagetype') FROM 'some/topic'
```

- `*`

  `mymessagetype` と呼ばれる protobuf メッセージタイプに準拠するバイナリの受信ペイロードを表します。

- `messageformat.desc`

s3-bucket という名前の Amazon S3 バケットに保存されている FileDescriptorSet ファイル。

- myproto

  myproto.proto という名前の FileDescriptorSet ファイルを生成するために使用されたオリジナルの .proto ファイル。

- messagetype

  myproto.proto で定義された messagetype と呼ばれるメッセージタイプ (インポートされた依存関係を含む)。

encode(value, encodingScheme)

エンコードスキームに基づいて、ペイロード (非 JSON データの場合もある) を文字列表現にエンコードするには、encode 関数を使用します。SQL バージョン 2016-03-23 以降でサポートされています。

値

AWS IoT SQL リファレンス で定義されている任意の有効な式。JSON 形式かどうかに関係なくペイロード全体をエンコードするには、* を指定できます。式を指定した場合は、まず評価の結果が文字列に変換され、その後でエンコードされます。

encodingScheme

使用するエンコードスキームを表すリテラル文字列。現在は、'base64' のみがサポートされます。

endswith(文字列、文字列)

Boolean を返し、最初の String 引数が 2 番目の String 引数で終わるかどうかを示します。どちらかの引数が Null または Undefined の場合、結果は Undefined です。SQL バージョン 2015-10-08 以降でサポートされています。

例: endswith("cat","at") = true。

引数の型 1	引数の型 2	結果
String	String	最初の引数が 2 番目の引数で終 は true。それ以外の場合は、fal す。
その他の値	その他の値	両方の引数は標準変換ルールを 字列に変換されます。最初の引 の引数で終わる場合は true。そ 合は、false を返します。どちら Null または Undefined の場 Undefined です。

exp(10 進数)

e を Decimal 引数の累乗にして返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: exp(1) = e。

引数の型	結果
Int	Decimal(倍精度で)、e ^ 引数。
Decimal	Decimal(倍精度で)、e ^ 引数。
String	Decimal(倍精度で)、e ^ 引数。String が Decimal に変換できない場合、結果は Undefined です。
その他の値	Undefined .

floor(Decimal)

指定の Decimal を最も近い Int に切り下げます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

floor(1.2) = 1

floor(-1.2) = -2

引数の型	結果
Int	Int、引数値。
Decimal	Int、最も近い Decimal へ切り下げられた Int 値。
String	Int。文字列は Decimal に変換され、最も近い Int へ切り下げられます。文字列が Decimal に変換できない場合、結果は Undefined です。
その他の値	Undefined .

get

コレクションのような型から値を抽出します (配列、文字列、オブジェクト)。最初の引数に変換は適用されません。テーブルで説明されているように、変換は 2 番目の引数に適用されます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

get(["a", "b", "c"], 1) = "b"

get({"a":"b"}, "a") = "b"

get("abc", 0) = "a"

引数の型 1	引数の型 2	結果
配列	Any Type (Int に変換)	2 番目の引数により提供されるのゼロベースインデックスの項に変換)。変換が失敗した場合、は Undefined です。インデックスArray の境界の外にある場合 (n

引数の型 1	引数の型 2	結果
		>= array.length)、結果は Undef す。
文字列	Any Type (Int に変換)	2 番目の引数により提供される のゼロベースインデックスの文 に変換)。変換が失敗した場合、 Undefined です。インデック 列の境界の外にある場合 (negat string.length)、結果は Undefin
オブジェクト	String (変換の適用なし)	2 番目の引数として提供される に対応する最初の引数のオブジ 存されている値。
その他の値	任意の値	Undefined .

get_dynamodb (tableName、partitionKeyName、partitionKeyValue、sortKeyName、sortKeyValue, roleArn)

DynamoDB テーブルからデータを取得します。get_dynamodb() を使用すると、ルールが評価されている間に DynamoDB テーブルを照会できます。DynamoDB から取得したデータを使用して、メッセージペイロードをフィルタリングまたは拡張できます。SQL バージョン 2016-03-23 以降でサポートされています。

get_dynamodb() には以下のパラメータがあります。

tableName

クエリする DynamoDB テーブルの名前。

partitionKeyName

パーティションキーの名前。詳細については、「DynamoDB Keys」を参照してください。

partitionKeyValue

レコードを識別するために使用されるパーティションキーの値。詳細については、「DynamoDB Keys」を参照してください。

sortKeyName

(オプション) ソートキーの名前。このパラメータは、クエリされた DynamoDB テーブルが複合キーを使用する場合にのみ必要です。詳細については、「DynamoDB Keys」を参照してください。

sortKeyValue

(オプション) ソートキーの値。このパラメータは、クエリされた DynamoDB テーブルが複合キーを使用する場合にのみ必要です。詳細については、「DynamoDB Keys」を参照してください。

roleArn

DynamoDB テーブルへのアクセスを付与する IAM ロールの ARN。ルールエンジンは、ユーザーに代わって DynamoDB テーブルにアクセスするためにこのロールを引き受けます。過度に許容されるロールの使用は避けてください。ルールで必要なアクセス許可のみをロールに付与します。1 つの DynamoDB テーブルへのアクセスを許可するポリシーの例を次に示します。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/table-
name"
        }
    ]
}
```

get_dynamodb() の使用方法の例として、 AWS IoTに接続されているすべてのデバイスのデバイス ID と位置情報を含む DynamoDB テーブルがあるとします。次の SELECT 文は、get_dynamodb() 関数を使用して、指定したデバイス ID の場所を取得します。

```
SELECT *, get_dynamodb("InServiceDevices", "deviceId", id,
"arn:aws:iam::12345678910:role/getdynamo").location AS location FROM 'some/
topic'
```

> **Note**
>
> - 1 つの SQL 文につき最大 1 回 get_dynamodb( ) を呼び出すことができます。1 つの SQL 文で get_dynamodb( ) 複数回呼び出すと、アクションを呼び出さずにルールが終了します。

get_mqtt_property(name)

次の MQTT5 ヘッダーのいずれかを参照します: contentType、payLoadFormatIndicator、responseTopic、correlationData。この関数は、content_type、format_indicator、response_topic、correlation_data いうリテラル文字列のいずれかを引数としてとります。詳細については、以下の関数の引数表を参照してください。

contentType

文字列: 公開メッセージの内容を説明する UTF-8 でエンコードされた文字列。

payLoadFormatIndicator

文字列: ペイロードが UTF-8 としてフォーマットされているかどうかを示す列挙型文字列の値。有効な値は、UNSPECIFIED_BYTES および UTF8_DATA です。

responseTopic

文字列: 応答メッセージのトピック名として使用される UTF-8 でエンコードされた文字列。レスポンストピックは、受信者がリクエスト/レスポンスフローの一部として公開すべきトピックを説明するために使用されます。トピックにワイルドカード文字を含めることはできません。

correlationData

文字列: 要求メッセージの送信者によって base64 でエンコードされたバイナリデータで、応答メッセージを受信した際に、それがどの要求に対するものかを識別するために使用されます。

次の表は、get_mqtt_property 関数で使用できる関数の引数と、それに関連する戻り値の型を示しています。

関数の引数

SQL	返されるデータ型 (存在する場合)	返されるデータ型 (存在しない場合)
`get_mqtt_property("format_indicator")`	文字列 (UNSPECIFIED_BYTES または UTF8_DATA)	文字列 (UNSPECIFIED_BYTES)
`get_mqtt_property("content_type")`	String	未定義
`get_mqtt_property("response_topic")`	String	未定義
`get_mqtt_property("correlation_data")`	base64 でエンコードされた文字列	未定義
`get_mqtt_property("some_invalid_name")`	未定義	未定義

以下の SQL ルール例では、`contentType`、`payLoadFormatIndicator`、`responseTopic`、`correlationData` のいずれかの MQTT5 ヘッダーを参照しています。

```
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(式、defaultValue)

指定された場合は 2 番目のパラメータのデフォルト値を返します。それ以外の場合は、最初のパラメータの式が null、undefined、または fail を返すときに undefined を返します。SQL バージョン 2016-03-23 以降でサポートされています。

> **Important**
>
> `get_or_default` は、JSON 以外のペイロードをそのまま直接サポートしていません。JSON 以外のペイロードを使用している場合は、 `encode`または `decode`関数を使用します。

`get_or_default()` には以下のパラメータがあります。

expression

データ型、関数、、変数リテラル、、ネストされたオブジェクトのクエリまたは を含む有効な式JSON 拡張。

defaultValue

(オプション) データ型、関数、、変数リテラル、、ネストされたオブジェクトのクエリまたは を含む有効な式JSON 拡張。これは、最初の引数が null、未定義、または失敗を返すたびに返される値です。

> **Note**
>
> get_secret、get_dynamodb、aws_lambda、get_thing_shadow、decode-protobuf、Machinelearning_predict など、顧客所有のリソースからデータを取得する関数は、defaultValue パラメータでは許可されません。

次の表は、各引数と関連する出力に許容される関数引数を示しています。

最初の引数	2 番目の引数	Output
成功した評価	任意の値または指定なし	最初の引数値。
未定義、Null、または失敗	Undefined または Null を含む任意の値	2 番目の引数値。
未定義、Null、または失敗	指定されていません	`Undefined`

例:

例 1:

次の例では、DynamoDB テーブルまたはクエリが失敗した場合の defaultValue 値を示します。

```
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'
```

例 2:

次の例では、ステータスが未定義の場合、安全なデフォルト値「UNKNOWN」を示します。

```
SELECT
  get_or_default( CASE status
    WHEN 'active' THEN 'GOOD'
    WHEN 'inactive' THEN 'BAD'/
    ELSE 'UNKNOWN'
  END, 'UNKNOWN') as status_category
FROM 'topic/subtopic'
```

例 3:

次の例は、単一のパラメータで get_or_default を使用する方法を示しています。これは、明確なデフォルト値がないが、ルールの実行を失敗させたくない場合に役立ちます。

```
SELECT
  get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME") as config
FROM 'device/telemetry'
```

DynamoDB ルックアップが失敗した場合、ルールの実行は失敗し、アクションは実行されません。代わりに次の SQL を使用する場合:

```
SELECT
```

```
  get_or_default(get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME")) as config
FROM 'device/telemetry'
```

get_or_default ステートメントは に評価されるためUndefined、この例では SELECT ステートメント全体が に評価{}され、すべてのルールアクションが試行されます。

> **Important**
>
> この機能を使用する際にセキュリティを維持するために、以下のベストプラクティスに従うことをお勧めします。
>
> - デフォルト値を含むルール定義でハードコードされたシークレットを使用しない
> - 機密情報の管理 AWS Secrets Manager に を使用する

get_registry_data(registryAPI, thingName, roleArn)

AWS IoT ルール内の AWS IoT モノのレジストリデータを取得します。レジストリデータ (デバイスが属する属性、モノのタイプ、モノのグループなど) を読み取って、この情報を使用してメッセージをフィルタリング、強化、または動的にルーティングできます。SQL バージョン 2016-03-23 以降でサポートされています。

`get_registry_data()` には以下のパラメータがあります。

registryAPI

呼び出されるレジストリ API。有効な値は、`DescribeThing` および `ListThingGroupsForThing` です。これらの値は定数文字列である必要があります。

thingName

文字列: レジストリデータを取得するモノの名前。

roleArn

文字列: 呼び出される API に基づく`iot:DescribeThing`アクセス許可またはアクセス`iot:ListThingGroupsForThing`許可を持つロール ARN。

`get_registry_data` 関数のレスポンス形式は、 というレジストリ API と同じです。詳細については、DescribeThing API と ListThingGroupsForThing APIsを参照してください。

例:

モノのタイプ情報を取得して、モノのタイプが `testenv` である (モノの名前が MQTT クライアント ID と一致する) モノの AWS IoT Core ライフサイクルイベントメッセージをフィルタリングできます。

```
SELECT *
FROM '$aws/events/lifecycle/+'
WHERE
    get_registry_data("DescribeThing",clientId,[roleArn]).thingTypeName='testenv'
```

例:

ゲートウェイデバイス `gateway1` によって送信される `sensor1` すべてのメッセージのモノの名前を持つデバイスのモノ属性を取得できます。

```
SELECT *, get_registry_data("DescribeThing","sensor1",
[roleArn]).attributes.temperature_threhold AS device1_tempthreshold
FROM home1/gateway1/sensor1/#
```

> **Note**
>
> - アクションとエラーアクションの SQL ステートメントと置換テンプレートごとに `get_registry_data()` 最大 1 回呼び出すことができます。

get_secret(secretId, secretType, key, roleArn)

AWS Secrets Manager の現在のバージョンのシークレットの暗号化された `SecretString` または `SecretBinary` フィールドの値を取得します。シークレットの作成と維持の詳細については、CreateSecret、UpdateSecret、および PutSecretValue を参照してください。

`get_secret()` には以下のパラメータがあります。

secretId

文字列: 取得するシークレットの Amazon リソースネーム (ARN) またはフレンドリ名。

secretType

文字列: シークレットタイプ。有効な値: `SecretString` | `SecretBinary`。

SecretString

- APIs、、 AWS CLIまたは AWS Secrets Manager コンソールを使用して JSON オブジェクトとして作成するシークレットの場合:
  - key パラメータの値を指定すると、この関数は指定されたキーの値を返します。
  - key パラメータの値を指定しない場合、この関数は JSON オブジェクト全体を返します。
- API または AWS CLIを使用して非 JSON オブジェクトとして作成するシークレットについては、以下を実行します。
  - key パラメータの値を指定すると、この関数は例外で失敗します。
  - key パラメータの値を指定しない場合、この関数はシークレットの内容を返します。

SecretBinary

- key パラメータの値を指定すると、この関数は例外で失敗します。
- key パラメータの値を指定しない場合、この関数は base64 でエンコードされた UTF-8 文字列としてシークレット値を返します。

key

(オプション) 文字列: シークレットの SecretString フィールドに保存されている JSON オブジェクト内のキー名。JSON オブジェクト全体ではなく、シークレットに保存されているキーの値のみを取得する場合は、この値を使用します。

このパラメータの値を指定し、シークレットの SecretString フィールド内に JSON オブジェクトが含まれていない場合、この関数は例外で失敗します。

roleArn

文字列: secretsmanager:GetSecretValue および secretsmanager:DescribeSecret アクセス許可を持つロール ARN。

> **Note**
>
> この関数は常に、シークレットの現在のバージョン (AWSCURRENT タグ付きのバージョン) を返します。 AWS IoT ルールエンジンは、各シークレットを最大 15 分間キャッシュします。その結果、ルールエンジンがシークレットを更新するのに最長で 15 分かかることがあります。つまり、 での更新から最大 15 分後にシークレットを取得した場合 AWS Secrets Manager、この関数は以前のバージョンを返す可能性があります。

この関数は計測されませんが、 AWS Secrets Manager 料金が適用されます。シークレットキャッシュメカニズムにより、ルールエンジンが AWS Secrets Managerを呼び出すことがあります。ルールエンジンは完全に分散されたサービスであるため、15 分のキャッシュウィンドウ中にルールエンジンから複数の Secrets Manager API 呼び出しが実行される場合があります。

例:

次の API キー認証の例のように、HTTPS ルールアクションの認証ヘッダーで `get_secret` 関数を使用できます。

```
"API_KEY": "${get_secret('API_KEY', 'SecretString', 'API_KEY_VALUE',
 'arn:aws:iam::12345678910:role/getsecret')}"
```

HTTPS ルールアクションの詳細については、「the section called “HTTP”」を参照してください。

get_thing_shadow(thingName、shadowName、roleArn)

指定されたモノの指定されたシャドウを返します。SQL バージョン 2016-03-23 以降でサポートされています。

thingName

文字列: シャドウを取得する対象のモノの名前。

shadowName

(オプション) 文字列: シャドウの名前。このパラメータは、名前の付きのシャドウを参照する場合にのみ必要です。

roleArn

文字列: `iot:GetThingShadow` アクセス権限を持つロール ARN。

例:

名前付きシャドウとともに使用する場合は、`shadowName` パラメータを指定します。

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","MyThingShadow","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

名前のないシャドウとともに使用する場合は、shadowName パラメータを省略します。

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

get_user_properties(userPropertyKey)

MQTT5 でサポートされているプロパティヘッダーの一種であるユーザープロパティを参照します。

userProperty

文字列: ユーザープロパティは、キーと値のペアです。この関数は、キーを引数にとり、関連するキーと一致するすべての値の配列を返します。

関数の引数

メッセージヘッダー内の以下のユーザープロパティの場合:

キー	値
あるキー	ある値
別のキー	別の値
あるキー	重複したキーを持つ値

次の表に、予想される SQL の動作を示します。

SQL	返されるデータ型	返されるデータ値
get_user_properties('あるキー')	文字列の配列	`['some value', 'value with duplicate key']`
get_user_properties('別のキー')	文字列の配列	`['a different value']`
get_user_properties( )	キーと値のペアのオブジェクトの配列	`[{'"some key": "some value"'}, {"other key": "a different value"}, {"some key": "value with duplicate key"}]`
get_user_properties('存在しないキー')	未定義	

以下の SQL ルール例では、ユーザープロパティ (MQTT5 プロパティヘッダーの一種) をペイロードに参照しています。

```
SELECT *, get_user_properties('user defined property key') as userProperty
FROM 'some/topic'
```

ハッシュ関数

AWS IoT には、次のハッシュ関数が用意されています。

- md2
- md5
- sha1
- sha224
- sha256
- sha384
- sha512

1 つの文字列引数を除くすべてのハッシュ関数。結果はその文字列のハッシュ値です。標準文字列変換は非文字列引数に適用されます。すべてのハッシュ関数は、SQL バージョン 2015-10-08 以降でサポートされます。

例:

`md2("hello")` = "a9046c73e00331af68917d3804f70655"

`md5("hello")` = "5d41402abc4b2a76b9719d911017c592"

indexof(文字列、文字列)

最初の引数のサブ文字列として、2 番目の引数の最初のインデックス (ゼロベース) を返します。両方の引数は文字列であることが期待されます。文字列ではない引数は、標準文字列変換ルールに従います。この関数は、配列にではなく、文字列にのみ適用されます。SQL バージョン 2016-03-23 以降でサポートされています。

例:

`indexof("abcd", "bc")` = 1

isNull()

引数が `Null` 値である場合は true を返します。SQL バージョン 2016-03-23 以降でサポートされています。

例:

`isNull(5)` = false。

`isNull(Null)` = true。

引数の型	結果
`Int`	false
`Decimal`	false
`Boolean`	false

引数の型	結果
String	false
Array	false
Object	false
Null	true
Undefined	false

isUndefined()

引数が Undefined である場合は true を返します。SQL バージョン 2016-03-23 以降でサポートされています。

例:

`isUndefined(5)` = false。

`isUndefined(floor([1,2,3]))` = true。

引数の型	結果
Int	false
Decimal	false
Boolean	false
String	false
Array	false
Object	false
Null	false
Undefined	true

length(文字列)

指定された文字列の文字数を返します。標準変換ルールは、非 String 引数に適用されます。SQL バージョン 2016-03-23 以降でサポートされています。

例:

`length("hi")` = 2

`length(false)` = 5

ln(10 進数)

引数の自然対数を返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `ln(e)` = 1。

引数の型	結果
Int	Decimal (倍精度で)、引数の自然対数。
Decimal	Decimal (倍精度で)、引数の自然対数。
Boolean	Undefined .
String	Decimal (倍精度で)、引数の自然対数。文字列が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

log(10 進数)

引数の 10 を底とする対数を返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `log(100)` = 2.0。

引数の型	結果
Int	Decimal 「倍精度で」、引数の 10 を底とした対数。
Decimal	Decimal 「倍精度で」、引数の 10 を底とした対数。
Boolean	Undefined .
String	Decimal 「倍精度で」、引数の 10 を底とした対数。String が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

lower(文字列)

指定した String の小文字バージョンを返します。非文字列引数は標準変換ルールを使用して文字列に変換されます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`lower("HELLO")` = "hello"。

`lower(["HELLO"])` = "[\"hello\"]"。

lpad(文字列、Int)

2 番目の引数で指定された数のスペースを左詰めにした String 引数を返します。Int 引数は、0 ~ 1000 の間である必要があります。提供された値がこの有効な範囲の外にある場合は、引数は最も近い有効な値に設定されます (0 または 1000)。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`lpad("hello", 2) = "  hello".`

`lpad(1, 3) = "  1"`

引数の型 1	引数の型 2	結果
String	Int	String、提供された String のスペースを左詰めにして提供さ
String	Decimal	Decimal 引数は最も近い Int に られ、String には指定された スを左詰めにします。
String	String	2 番目の引数は Decimal に変換 最も近い Int に切り下げられま た、String には指定された数の が左詰めにされます。2 番目の に変換できない場合、結果は Un です。
その他の値	Int/Decimal/String	最初の値は、標準変換ルールを String に変換された後、LPAD の String に適用されます。そ きない場合、結果は Undefine
任意の値	その他の値	Undefined .

ltrim(文字列)

提供される String から先頭の空白 (タブとスペース) をすべて削除します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

Ltrim(" h i ") = "hi "。

引数の型	結果
Int	先頭の空白がすべて削除された String の Int 表現。
Decimal	先頭の空白がすべて削除された String の Decimal 表現。
Boolean	先頭の空白がすべて削除されたブール (「true」または「false」) の String 表現。
String	先頭の空白がすべて削除された引数。
配列	先頭の空白がすべて削除された String の Array 表現 (標準変換ルールを使用)。
オブジェクト	先頭の空白がすべて削除されたオブジェクトの String 表現 (標準変換ルールを使用)。
Null	Undefined .
未定義	Undefined .

machinelearning_predict (modelId, roleArn, record)

Amazon SageMaker AI モデルに基づいて、MQTT メッセージからのデータを使用して予測を行うには、machinelearning_predict 関数を使用します。SQL バージョン 2015-10-08 以降でサポートされています。machinelearning_predict 関数の引数は次のとおりです。

modelId

予測を実行する対象となるモデルの ID。このモデルのリアルタイムエンドポイントを有効にしておく必要があります。

roleArn

`machinelearning:Predict` アクセス許可および `machinelearning:GetMLModel` アクセス許可を許可するポリシーが指定され、予測を実行する対象となるモデルへのアクセスを許可する IAM ロール。

record

SageMaker AI Predict API に渡されるデータ。単一レイヤーの JSON オブジェクトとして表す必要があります。record が複数レベルの JSON オブジェクトの場合は、値のシリアル化によって平坦化されます。たとえば、次の JSON の場合:

```
{ "key1": {"innerKey1": "value1"}, "key2": 0}
```

次のようになります。

```
{ "key1": "{\"innerKey1\": \"value1\"}", "key2": 0}
```

この関数では、次のフィールドを持つ JSON オブジェクトが返されます。

predictedLabel

モデルに基づく入力の区分。

details

次の属性が含まれています。

PredictiveModelType

モデルのタイプ。有効な値は、REGRESSION、BINARY、MULTICLASS です。

Algorithm

予測を行うために SageMaker AI で使用されるアルゴリズム。この値は SGD にする必要があります。

predictedScores

各ラベルに対応する、未加工の分類スコアを格納します。

predictedValue

SageMaker AI によって予測された値。

mod(10 進数、10 進数)

最初の引数を 2 番目の引数で割ったときの剰余を返します。remainder(Decimal, Decimal) と同等です。同じモジュロ機能に挿入演算子として「%」も使用できます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `mod(8, 3)` = 2。

左のオペランド	右のオペランド	出力
`Int`	`Int`	`Int`、最初の引数を 2 番目の引 ときの剰余。
`Int/Decimal`	`Int/Decimal`	`Decimal`、最初の引数を 2 番目 ドで割ったときの剰余。
`String/Int/Decimal`	`String/Int/Decimal`	すべての文字列を小数に変換し 結果は、最初の引数を 2 番目の 割ったときの剰余です。そうで は、`Undefined` です。
その他の値	その他の値	`Undefined` .

nanvl(AnyValue, AnyValue)

最初の引数が有効な `Decimal` ならば、最初の引数が返されます。それ以外の場合、2 番目の引数が返されます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `Nanvl(8, 3)` = 8。

引数の型 1	引数の型 2	出力
未定義	任意の値	2 番目の引数。
Null	任意の値	2 番目の引数。

引数の型 1	引数の型 2	出力
Decimal (NaN)	任意の値	2 番目の引数。
Decimal (not NaN)	任意の値	最初の引数。
その他の値	任意の値	最初の引数。

newuuid()

16 バイトのランダムな UUID を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例: `newuuid()` = `123a4567-b89c-12d3-e456-789012345000`

numbytes(文字列)

指定された文字列の UTF-8 エンコードのバイト数を返します。標準変換ルールは、非 `String` 引数に適用されます。SQL バージョン 2016-03-23 以降でサポートされています。

例:

`numbytes("hi")` = 2

`numbytes("€")` = 3

parse_time(String, Long[, String])

`parse_time` 機能を使用して、タイムスタンプを人間が判読可能な日付/時刻形式にフォーマットします。SQL バージョン 2016-03-23 以降でサポートされています。タイムスタンプ文字列をミリ秒に変換するには、「time_to_epoch(String, String)」を参照してください。

`parse_time` 関数は次の引数を想定します。

pattern

(文字列) Joda-Time フォーマットに従う日付/時刻パターン。

timestamp

(ロング) Unix エポックからミリ秒単位でフォーマットされる時間。関数「timestamp()」を参照してください。

timezone

(文字列) フォーマットされた日付/時刻のタイムゾーン。デフォルトは「UTC」です。この関数は、「Joda-Time のタイムゾーン」をサポートしています。この引数はオプションです。

例:

このメッセージがトピック「A/B」に公開されると、ペイロード {"ts": "1970.01.01 AD at 21:46:40 CST"} が S3 バケットに送信されます。

```
{
    "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"
    }
}
```

このメッセージがトピック「A/B」に公開されると、{"ts": "2017.06.09 AD at 17:19:46 UTC"} (ただし、現在の日付/時刻) と同様のペイロードが S3 バケットに送信されます。

```
{
    "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() は、置換テンプレートとしても使用できます。たとえば、このメッセージがトピック「A/B」に公開されると、ペイロードは S3 バケットに key = 「2017」で送信されます。

```
{
    "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(10 進数、10 進数)

最初の引数を 2 番目の引数の累乗にして返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。SQL バージョン 2015-10-08 以降でサポートされています。

例: power(2, 5) = 32.0。

引数の型 1	引数の型 2	出力
Int/Decimal	Int/Decimal	Decimal「倍精度で」、最初の 目の引数の累乗にします。
Int/Decimal/String	Int/Decimal/String	Decimal(倍精度で)、最初の引 の引数の累乗にします。すべて 小数に変換されます。いずれか が Decimal への変換に失敗し は Undefined です。
その他の値	その他の値	Undefined .

principal()

トリガーメッセージの発行方法に基づいて、デバイスが認証に使用するプリンシパルを返します。次の表は、発行方法とプロトコルごとに返されるプリンシパルについて説明しています。

メッセージの発行方法	プロトコル	認証情報のタイプ
MQTT クライアント	MQTT	X.509 デバイス証明書
AWS IoT コンソール MQTT クライアント	MQTT	IAM ユーザーまたはロール
AWS CLI	HTTP	IAM ユーザーまたはロール
AWS IoT デバイス SDK	MQTT	X.509 デバイス証明書
AWS IoT デバイス SDK	MQTT over WebSocket	IAM ユーザーまたはロール

次の例は、principal() が返すことができるさまざまなタイプの値を示しています。

- X.509 証明書のサムプリント: ba67293af50bf2506f5f93469686da660c7c844e7b3950bfb16813e0d31e9373
- IAM ロール ID とセッション名: ABCD1EFG3HIJK2LMNOP5:my-session-name
- ユーザー ID: ABCD1EFG3HIJK2LMNOP5 を返します

rand()

0.0 から 1.0 までの間で疑似ランダムで、均等に分散された倍を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`rand( ) = 0.8231909191640703`

regexp_matches(文字列、文字列)

文字列 (最初の引数) に、正規表現 (2 番目の引数) の一致が含まれている場合は、true を返します。正規表現で | を使用する場合は、( ) で使用します。

例:

`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。

最初の引数:

引数の型	結果
`Int`	`String` の `Int` 表現。
`Decimal`	`String` の `Decimal` 表現。
`Boolean`	ブール (「true」または「false」) の `String` 表現。
`String`	。`String`
配列	`String` の `Array` 表現 (標準変換ルールを使用)。
オブジェクト	オブジェクトの `String` 表現 (標準変換ルールを使用)。

引数の型	結果
Null	Undefined .
未定義	Undefined .

2 番目の引数:

有効な正規表現の式である必要があります。非文字列型は標準変換ルールを使用して String に変換されます。型により、結果として生じる文字列が有効な正規表現でない場合もあります。(変換された) 引数が有効な正規表現でない場合、結果は Undefined です。

regexp_replace(文字列、文字列、文字列)

最初の引数にある 2 番目の引数 (正規表現) の出現すべてを 3 番目の引数で置き換えます。「$」でキャプチャグループを参照します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`regexp_replace("abcd", "bc", "x")` = "axd"。

`regexp_replace("abcd", "b(.*)d", "$1")` = "ac"。

最初の引数:

引数の型	結果
Int	String の Int 表現。
Decimal	String の Decimal 表現。
Boolean	ブール (「true」または「false」) の String 表現。
String	ソース値。
配列	String の Array 表現 (標準変換ルールを使用)。
オブジェクト	オブジェクトの String 表現 (標準変換ルールを使用)。

引数の型	結果
Null	Undefined .
未定義	Undefined .

2 番目の引数:

有効な正規表現の式である必要があります。非文字列型は標準変換ルールを使用して String に変換されます。型により、結果として生じる文字列が有効な正規表現でない場合もあります。(変換された) 引数が有効な正規表現でない場合、結果は Undefined です。

3 番目の引数:

有効な正規表現の置換文字列である必要があります。(キャプチャグループを参照できます。) 非文字列型は標準変換ルールを使用して String に変換されます。(変換された) 引数が有効な正規表現の置換文字列でない場合、結果は Undefined です。

regexp_substr(文字列、文字列)

最初のパラメータにある 2 番目のパラメータ (正規表現) の最初の一致を見つけます。「$」でキャプチャグループを参照します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`regexp_substr("hihihello", "hi")` = "hi"

`regexp_substr("hihihello", "(hi)*")` = "hihi"

最初の引数:

引数の型	結果
Int	String の Int 表現。
Decimal	String の Decimal 表現。
Boolean	ブール (「true」または「false」) の String 表現。
String	String引数。

引数の型	結果
配列	String の Array 表現 (標準変換ルールを使用)。
オブジェクト	オブジェクトの String 表現 (標準変換ルールを使用)。
Null	Undefined .
未定義	Undefined .

2 番目の引数:

有効な正規表現の式である必要があります。非文字列型は標準変換ルールを使用して String に変換されます。型により、結果として生じる文字列が有効な正規表現でない場合もあります。(変換された) 引数が有効な正規表現でない場合、結果は Undefined です。

remainder(Decimal, Decimal)

最初の引数を 2 番目の引数で割ったときの剰余を返します。mod(10 進数、10 進数) と同等です。同じモジュロ機能に挿入演算子として「%」も使用できます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `remainder(8, 3)` = 2。

左のオペランド	右のオペランド	出力
Int	Int	Int、最初の引数を 2 番目の引 ときの剰余。
Int/Decimal	Int/Decimal	Decimal、最初の引数を 2 番目 ドで割ったときの剰余。
String/Int/Decimal	String/Int/Decimal	すべての文字列を小数に変換し 結果は、最初の引数を 2 番目の 割ったときの剰余です。そうで は、Undefined です。

左のオペランド	右のオペランド	出力
その他の値	その他の値	Undefined .

replace(String, String, String)

最初の引数にある 2 番目の引数の出現すべてを 3 番目の引数で置き換えます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`replace("abcd", "bc", "x")` = `"axd"`.

`replace("abcdabcd", "b", "x")` = `"axcdaxcd"`.

すべての引数

引数の型	結果
Int	String の Int 表現。
Decimal	String の Decimal 表現。
Boolean	ブール (「true」または「false」) の String 表現。
String	ソース値。
配列	String の Array 表現 (標準変換ルールを使用)。
オブジェクト	オブジェクトの String 表現 (標準変換ルールを使用)。
Null	Undefined .
未定義	Undefined .

rpad(文字列、Int)

2 番目の引数で指定された数のスペースを右詰めにした文字列の引数を返します。Int 引数は、0 ~ 1000 の間である必要があります。提供された値がこの有効な範囲の外にある場合は、引数は最も近い有効な値に設定されます (0 または 1000)。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`rpad("hello", 2)` = "hello ".

`rpad(1, 3)` = "1 ".

引数の型 1	引数の型 2	結果
String	Int	String は、提供された Int と同じ数のスペースを右詰めにされています。
String	Decimal	Decimal 引数は最も近い Int に切り下げられ、文字列は、提供された Int と同じ数のスペースを右詰めにされています。
String	String	2 番目の引数は Decimal に変換され、最も近い Int に切り下げられます。String は、Int 値と同じ数のスペースを右詰めにされています。
その他の値	Int/Decimal/String	最初の値は、標準変換ルールを使用して String に変換され

引数の型 1	引数の型 2	結果
		た後、rpad 関数がその String に適用されます。それが変換できない場合、結果は Undefined です。
任意の値	その他の値	Undefined .

round(10 進数)

指定の Decimal を最も近い Int に丸めます。Decimal が 2 つの Int 値と等距離である場合 (例: 0.5)、Decimal は丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: Round(1.2) = 1。

Round(1.5) = 2.

Round(1.7) = 2.

Round(-1.1) = -1.

Round(-1.5) = -2.

引数の型	結果
Int	引数。
Decimal	Decimal は、最も近い Int に切り下げられます。
String	Decimal は、最も近い Int に切り下げられます。文字列が Decimal に変換できない場合、結果は Undefined です。
その他の値	Undefined .

rtrim(文字列)

提供される String から末尾の空白 (タブとスペース) をすべて削除します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

```
rtrim(" h i ") = "h i"
```

引数の型	結果
Int	String の Int 表現。
Decimal	String の Decimal 表現。
Boolean	ブール (「true」または「false」) の String 表現。
配列	String の Array 表現 (標準変換ルールを使用)。
オブジェクト	オブジェクトの String 表現 (標準変換ルールを使用)。
Null	Undefined .
未定義	Undefined

sign(10 進数)

指定された数値の符号を返します。引数の符号が正の場合、1 を返します。引数の符号が負の場合、-1 を返します。引数が 0 の場合、0 を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

```
sign(-7) = -1.
```

```
sign(0) = 0.
```

```
sign(13) = 1.
```

引数の型	結果
Int	Int、Int の値の符号。
Decimal	Int、Decimal の値の符号。
String	Int、Decimal の値の符号。文字列は Decimal の値に変換され、Decimal の値の符号が返されます。String が Decimal に変換できない場合、結果は Undefined です。SQL バージョン 2015-10-08 以降でサポートされています。
その他の値	Undefined .

sin(10 進数)

数値のサインをラジアンで返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: sin(0) = 0.0

引数の型	結果
Int	Decimal 「倍精度で」、引数のサイン。
Decimal	Decimal (倍精度で)、引数のサイン。
Boolean	Undefined .
String	Decimal (倍精度で)、引数のサイン。文字列が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .

引数の型	結果
Undefined	Undefined .

sinh(10 進数)

数値のハイパーボリックサインを返します。Decimal 値は関数適用の前に倍精度に丸められます。結果は倍精度の Decimal 値です。SQL バージョン 2015-10-08 以降でサポートされています。

例: sinh(2.3) = 4.936961805545957

引数の型	結果
Int	Decimal (倍精度で)、引数のハイパーボリックサイン。
Decimal	Decimal (倍精度で)、引数のハイパーボリックサイン。
Boolean	Undefined .
String	Decimal (倍精度で)、引数のハイパーボリックサイン。文字列が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

sourceip()

デバイスまたはそれに接続するルーターの IP アドレスを取得します。デバイスがインターネットに直接接続されている場合、この関数はデバイスの送信元 IP アドレスを返します。デバイスがイン

ターネットに接続するルーターに接続されている場合、関数はルーターの送信元 IP アドレスを返します。SQL バージョン 2016-03-23 でサポートされています。`sourceip()` はパラメータを取得しません。

> **Important**
>
> デバイスの公開送信元 IP アドレスは、多くの場合、インターネットサービスプロバイダーのルーターやケーブルモデムなどの最新のネットワークアドレス変換 (NAT) ゲートウェイの IP アドレスです。

例:

```
sourceip()="192.158.1.38"
```

```
sourceip()="1.102.103.104"
```

```
sourceip()="2001:db8:ff00::12ab:34cd"
```

SQL 例:

```
SELECT *, sourceip() as deviceIp FROM 'some/topic'
```

AWS IoT Core ルールアクションで sourceip() 関数を使用する方法の例:

例 1

次の例は、DynamoDB アクションで () 関数を代替テンプレートとして呼び出す方法を示しています。

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

例 2

次の例は、代替テンプレートを使用して sourceip () 関数を MQTT ユーザープロパティとして追加する方法を示しています。

```
{
 "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()}"
       }
      ]
     }
    }
   }
  ]
 }
}
```

メッセージブローカーと基本的な取り込みパスの両方から AWS IoT Core ルールに渡すメッセージからソース IP アドレスを取得できます。IPv4 と IPv6 の両方のメッセージの両方の送信元 IP アドレスを使用できます。送信元 IP は次のように表示されます。

IPv6: `yyyy:yyyy:yyyy::yyyy:yyyy`

IPv4:`xxx.xxx.xxx.xxx`

> **Note**
>
> 元の送信元 IP は Republish アクションには渡されません。

substring(String, Int[, Int])

1 つか 2 つの `String` 値が続く `Int` を予想します。`String` および単一の `Int` 引数では、この関数は提供された `String` インデックス (ゼロベース、包括) から `Int` の端までの提供された `String` のサブストリングを返します。`String` および 2 つの `Int` 引数では、この関数は、最初の `String` インデックス引数 (ゼロベース、包括) から 2 番目の `Int` インデックス引数 (ゼロベース、除外) までの提供された `Int` のサブストリングを返します。ゼロ以下のインデックスはゼロに設定されます。`String` の長さを超過するインデックスは `String` の長さに設定されます。3 つの引数バージョンでは、最初のインデックスが 2 番目インデックスより大きい (または等しい) 場合、結果は空の `String` です。

提供された引数が (*`String`*, *`Int`*)、または (*`String`*, *`Int`*, *`Int`*) でない場合、引数に標準変換が適用され、正しい型への変換が試行されます。型が変換できない場合、関数の結果は `Undefined` です。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`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()

このルールで指定されている SQL バージョンを返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`sql_version()` = "2016-03-23"

sqrt(10 進数)

数値の平方根を返します。Decimal 引数は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `sqrt(9)` = 3.0。

引数の型	結果
Int	引数の平方根。
Decimal	引数の平方根。
Boolean	Undefined .
String	引数の平方根。文字列が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

startswith(文字列、文字列)

最初の文字列引数が 2 番目の文字列引数で始まるかどうか、Boolean を返します。どちらかの引数が Null または Undefined の場合、結果は Undefined です。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`startswith("ranger","ran")` = true

引数の型 1	引数の型 2	結果
String	String	最初の文字列が 2 番目の文字列どうか。
その他の値	その他の値	両方の引数は標準変換ルールを文字列に変換されます。最初の2 番目の文字列で始まる場合はします。どちらかの引数が Nul Undefined の場合、結果は U です。

tan(10 進数)

数値のタンジェントをラジアンで返します。Decimal 値は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: `tan(3)` = -0.1425465430742778

引数の型	結果
Int	Decimal 「倍精度で」、引数のタンジェント。
Decimal	Decimal 「倍精度で」、引数のタンジェント。
Boolean	Undefined .

引数の型	結果
String	Decimal「倍精度で」、引数のタンジェント。文字列が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .
Null	Undefined .
未定義	Undefined .

tanh(10 進数)

数値のハイパーボリックタンジェントをラジアンで返します。Decimal 値は関数適用の前に倍精度に丸められます。SQL バージョン 2015-10-08 以降でサポートされています。

例: tanh(2.3) = 0.9800963962661914

引数の型	結果
Int	Decimal「倍精度で」、引数の双曲線タンジェント。
Decimal	Decimal「倍精度で」、引数の双曲線タンジェント。
Boolean	Undefined .
String	Decimal「倍精度で」、引数の双曲線タンジェント。文字列が Decimal に変換できない場合、結果は Undefined です。
配列	Undefined .
オブジェクト	Undefined .

引数の型	結果
Null	`Undefined .`
未定義	`Undefined .`

time_to_epoch(String, String)

`time_to_epoch` 関数を使用して、タイムスタンプ文字列を Unix エポック時間のミリ秒数に変換します。SQL バージョン 2016-03-23 以降でサポートされています。ミリ秒をフォーマットされたタイムスタンプ文字列に変換するには、「parse_time(String, Long[, String])」を参照してください。

`time_to_epoch` 関数は次の引数を想定します。

timestamp

(文字列) Unix エポックからミリ秒に変換されるタイムスタンプ文字列。タイムスタンプ文字列がタイムゾーンを指定しない場合、関数は UTC タイムゾーンを使用します。

pattern

(文字列) JDK11 Time フォーマットに従う日付/時刻パターン。

例:

`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 ルールエンジンによって観測された 1970 年 1 月 1 日木曜日の協定世界時 (UTC) からの現在のタイムスタンプをミリ秒単位で返します。SQL バージョン 2015-10-08 以降でサポートされています。

例: `timestamp( ) = 1481825251155`

topic(10 進数)

ルールをトリガーしたメッセージが送信されたトピックを返します。パラメータが指定されていない場合、トピック全体が返されます。Decimal パラメータは、特定のトピックセグメントを指定するために使用され、1 は最初のセグメントを表します。foo/bar/baz トピックでは、topic(1) は foo を返し、topic(2) は bar を返す、と続いていきます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`topic()` = "things/myThings/thingOne"

`topic(1)` = "things"

基本的な取り込みが使用されている場合、トピック (`$aws/rules/rule-name`) の最初のプレフィックスは topic() 関数では使用できません。たとえば、次のトピックがあるとします。

`$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`

`topic()` = "Buildings/Building5/Floor2/Room201/Lights"

`topic(3)` = "Floor2"

traceid()

MQTT メッセージのトレース ID (UUID) を返すか、または、メッセージが MQTT 経由で送信されなかった場合は Undefined を返します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`traceid()` = "12345678-1234-1234-1234-123456789012"

transform(String, Object, Array)

Array パラメータの Object パラメータの指定された変換の結果を含むオブジェクトの配列を返します。

SQL バージョン 2016-03-23 以降でサポートされています。

文字列

使用する変換モード。サポートされている変換モードと、Object および Array パラメータから Result を作成する方法については、次の表を参照してください。

オブジェクト

Array の各要素に適用する属性を含むオブジェクト。

配列

Object の属性が適用されるオブジェクトの配列。

この配列内の各オブジェクトは、関数の応答内のオブジェクトに対応します。関数の応答の各オブジェクトには、元のオブジェクトに存在する属性と、String で指定された変換モードによって決定される Object によって提供される属性が含まれます。

String パラメータ	**Object** パラメータ	**Array** パラメータ	結果
enrichArray	オブジェクト	オブジェクトの配列	各オブジェクトに Array パラメータの要素の属性と Object パラメータの属性が含まれるオブジェクトの配列。
その他の値	任意の値	任意の値	未定義

> **Note**
>
> この関数によって返される配列は 128 KiB に制限されています。

変換関数の例 1

この例では、transform() 関数がデータオブジェクトと配列からオブジェクトの単一の配列を生成する方法を示します。

この例では、次のメッセージが MQTT トピック A/B に発行されます。

```
{
    "attributes": {
        "data1": 1,
```

```
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

トピックルールアクションのこの SQL ステートメントでは、String 値が enrichArray である transform() 関数を使用します。この例では、Object はメッセージペイロードの attributes プロパティで、Array は 3 つのオブジェクトを含む values 配列です。

```
select value transform("enrichArray", attributes, values) from 'A/B'
```

メッセージペイロードを受信すると、SQL ステートメントは次の応答と評価されます。

```
[
  {
    "a": 3,
    "data1": 1,
    "data2": 2
  },
  {
    "b": 4,
    "data1": 1,
    "data2": 2
  },
  {
    "c": 5,
    "data1": 1,
    "data2": 2
  }
]
```

変換関数の例 2

この例では、transform() 関数がリテラル値を使用して、メッセージペイロードの個々の属性を含めて、名前を変更する方法を示します。

この例では、次のメッセージが MQTT トピック A/B に発行されます。これは the section called “変換関数の例 1” で使用されたメッセージと同じものです。

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

トピックルールアクションのこの SQL ステートメントでは、String 値が enrichArray である transform() 関数を使用します。transform() 関数の Object にはメッセージペイロードの値が attributes.data1 である key という単一属性が含まれており、Array は、前述の例で使用されたものと同じ 3 つのオブジェクトを含む values 配列です。

```
select value transform("enrichArray", {"key": attributes.data1}, values) from 'A/B'
```

メッセージペイロードを受信すると、SQL ステートメントは次の応答と評価されます。応答で data1 プロパティの名前が key になっていることに注意してください。

```
[
  {
    "a": 3,
    "key": 1
  },
  {
```

```
    "b": 4,
    "key": 1
  },
  {
    "c": 5,
    "key": 1
  }
]
```

変換関数の例 3

この例では、ネストされた SELECT 句で transform() 関数を使用して、複数の属性を選択し、後続の処理のために新しいオブジェクトを作成する方法を示します。

この例では、次のメッセージが MQTT トピック A/B に発行されます。

```
{
  "data1": "example",
  "data2": {
    "a": "first attribute",
    "b": "second attribute",
    "c": [
      {
        "x": {
          "someInt": 5,
          "someString": "hello"
        },
        "y": true
      },
      {
        "x": {
          "someInt": 10,
          "someString": "world"
        },
        "y": false
      }
    ]
  }
}
```

この変換関数の Object は、メッセージの data2 オブジェクトの a 要素と b 要素を含む SELECT ステートメントによって返されるオブジェクトです。Array パラメータは、元のメッセージの data2.c 配列の 2 つのオブジェクトで構成されます。

```
select value transform('enrichArray', (select a, b from data2), (select value c from
 data2)) from 'A/B'
```

前述のメッセージにより、SQL ステートメントは次の応答に評価されます。

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

この応答で返される配列は、batchMode をサポートするトピックルールアクションで使用できます。

trim(文字列)

提供された String から、すべての先頭および末尾の空白を削除します。SQL バージョン 2015-10-08 以降でサポートされています。

例:

Trim(" hi ") = "hi"

引数の型	結果
Int	すべての先頭および末尾の空白が削除された String の Int 表現。
Decimal	すべての先頭および末尾の空白が削除された String の Decimal 表現。
Boolean	すべての先頭および末尾の空白が削除された String (「true」または「false」) の Boolean 表現。
String	すべての先頭および末尾の空白が削除された String。
配列	標準変換ルールを使用した String の Array 表現。
オブジェクト	標準変換ルールを使用したオブジェクトの String 表現。
Null	Undefined .
未定義	Undefined .

trunc(10 進数、Int)

2 番目の引数で指定された Decimal の場所の数で最初の引数を切り捨てます。2 番目の引数がゼロよりより少ない場合は、ゼロに設定されます。2 番目の引数が 34 より大きい場合は、34 に設定されます。末尾のゼロは結果から省かれます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`trunc(2.3, 0)` = 2.

`trunc(2.3123, 2)` = 2.31.

`trunc(2.888, 2)` = 2.88.

`trunc(2.00, 5)` = 2.

引数の型 1	引数の型 2	結果
`Int`	`Int`	ソース値。
`Int/Decimal`	`Int/Decimal`	最初の引数は 2 番目の引数で説 長さに切り捨てられます。2 番 は、`Int` でなければ、最も近い 下げられます。
`Int/Decimal/String`	`Int/Decimal`	最初の引数は 2 番目の引数で説 長さに切り捨てられます。2 番 は、`Int` でなければ、最も近い り下げられます。`String` は De に変換されます。文字列変換が 合、結果は `Undefined` です。
その他の値		`Undefined` .

upper(文字列)

指定した `String` の大文字バージョンを返します。非 `String` 引数は標準変換のルールを使用して `String` に変換されます。SQL バージョン 2015-10-08 以降でサポートされています。

例:

`upper("hello")` = "HELLO"

`upper(["hello"])` = "[\"HELLO\"]"

リテラル

ルール SQL の SELECT および WHERE 句で直接リテラルオブジェクトを指定でき、情報を渡すのに役立ちます。

> **Note**
>
> リテラルは SQL バージョン 2016-03-23 以降を使用する場合にのみ利用可能です。

JSON オブジェクトの構文が使用されています (キーと値のペア、カンマ区切り、キーが文字列で値が JSON 値なら波括弧 {} で囲む)。以下に例を示します。

トピックに公開された受信ペイロード topic/subtopic: `{"lat_long": [47.606,-122.332]}`

SQL ステートメント: `SELECT {'latitude': get(lat_long, 0),'longitude':get(lat_long, 1)} as lat_long FROM 'topic/subtopic'`

結果の出力ペイロードは次のとおりです: `{"lat_long": {"latitude":47.606,"longitude":-122.332}}`。

ルール SQL の SELECT および WHERE 句で直接配列を指定でき、情報をグループ化できます。JSON の構文が使用されています (カンマ区切りの項目を角括弧 [] でラップし、配列リテラルを作成する)。以下に例を示します。

トピックに公開された受信ペイロード topic/subtopic: `{"lat": 47.696, "long": -122.332}`

SQL ステートメント: `SELECT [lat,long] as lat_long FROM 'topic/subtopic'`

結果の出力ペイロードは次のとおりです: `{"lat_long": [47.606,-122.332]}`。

Case ステートメント

Case ステートメントを使用して、switch ステートメントと同様に、実行のブランチができます。

構文:

```
CASE v WHEN t[1] THEN r[1]
  WHEN t[2] THEN r[2] ...
  WHEN t[n] THEN r[n]
  ELSE r[e] END
```

v 式が評価され、各 WHEN 句の *t[i]* 値と等しいかどうか照合されます。一致がある場合は、対応する *r[i]* 式が CASE ステートメントの結果になります。WHEN 句は順番に評価され、一致する句が複数ある場合、最初に一致した句の結果が CASE ステートメントの結果になります。一致するものがない場合は、ELSE 句の *r[e]* が結果です。一致するものがなく、ELSE 句もない場合、結果は Undefined です。

CASE ステートメントには少なくとも 1 つの WHEN 句が必要です。ELSE 句はオプションです。

例:

トピック topic/subtopic に公開された受信ペイロード:

```
{
    "color":"yellow"
}
```

SQL ステートメント:

```
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'
```

結果の出力ペイロードは次のとおりです。

```
{
    "instructions":"caution"
}
```

> **Note**
>
> *v* が Undefined の場合、CASE ステートメントの結果は Undefined です。

JSON 拡張

入れ子になった JSON オブジェクトに対応するには、次に示す ANSI SQL 構文への拡張を使用できます。

"." 演算子

この演算子は、埋め込み JSON オブジェクトのメンバーにアクセスします。ANSI SQL や JavaScript の場合と同じ機能があります。以下に例を示します。

```
SELECT foo.bar AS bar.baz FROM 'topic/subtopic'
```

topic/subtopic トピックに送信される次のメッセージペイロードから、foo オブジェクト内の bar プロパティの値を選択します。

```
{
  "foo": {
    "bar": "RED",
    "bar1": "GREEN",
    "bar2": "BLUE"
  }
}
```

JSON プロパティ名にハイフン文字または数字が含まれている場合、「ドット」表記は機能しません。代わりに、get 関数を使用してプロパティの値を抽出します。

この例では、次のメッセージが iot/rules トピックに送信されます。

```
{
  "mydata": {
    "item2": {
      "0": {
        "my-key": "myValue"
      }
    }
  }
}
```

通常、my-key の値はこのクエリのように識別されます。

```
SELECT * from iot/rules WHERE mydata.item2.0.my-key= "myValue"
```

ただし、プロパティ名my-key にはハイフンが含まれ、item2には数字が含まれるため、次のクエリが示すように get 関数を使用する必要があります。

```
SELECT * from 'iot/rules' WHERE get(get(get(mydata,"item2"),"0"),"my-key") = "myValue"
```

* 演算子

ANSI SQL の * ワイルドカードと同じ機能があります。SELECT 句のみで使用され、メッセージデータを含む新しい JSON オブジェクトを作成します。メッセージペイロードが JSON 形式でない場合、* はメッセージペイロード全体を raw バイトとして返します。以下に例を示します。

```
SELECT * FROM 'topic/subtopic'
```

属性値に対する関数の適用

以下に、デバイスから発行される JSON ペイロードの例を示します。

```
{
    "deviceid" : "iot123",
    "temp" : 54.98,
    "humidity" : 32.43,
    "coords" : {
        "latitude" : 47.615694,
        "longitude" : -122.3359976
    }
}
```

次の例では、JSON ペイロード内の属性値に関数を適用しています。

```
SELECT temp, md5(deviceid) AS hashed_id FROM topic/#
```

このクエリの結果は、次の JSON オブジェクトです。

```
{
   "temp": 54.98,
   "hashed_id": "e37f81fb397e595c4aeb5645b8cbbbd1"
}
```

置換テンプレート

代替テンプレートを使用して、ルールがトリガーされて AWS IoT アクションを実行したときに返される JSON データを拡張できます。置換テンプレートの構文は${式}です。式は、SELECT 句、WHERE 句、および AWS IoT で でサポートされている任意の式にすることができますAWS IoT ルールアクション。この式をルールのアクションフィールドに接続して、アクションを動的に構成できます。実際には、この機能はアクションの情報の一部を置き換えます。これには、関数、演算子、および元のメッセージペイロードに存在する情報が該当します。

> **Important**
>
> 置換テンプレート内の式は "SELECT..." ステートメントとは独立して評価されるため、AS 句を使用して作成されたエイリアスを参照することはできません。元のペイロード、関数、および演算子に存在する情報のみを参照できます。

サポートされる式の詳細については、「AWS IoT SQL リファレンス」を参照してください。

次のルールアクションは、置換テンプレートをサポートします。各アクションは、置換可能なさまざまなフィールドをサポートしています。

- Apache Kafka
- CloudWatch アラーム
- CloudWatch Logs
- CloudWatch メトリクス
- DynamoDB
- DynamoDBv2
- Elasticsearch
- HTTP
- AWS IoT Events
- AWS IoT SiteWise
- Kinesis Data Streams
- Firehose
- Lambda
- ロケーション
- OpenSearch
- 再発行
- S3
- SNS
- SQS
- ステップ関数
- Timestream

置換テンプレートは、ルール内のアクションパラメータに表示されます。

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

このルールが my/iot/topic に発行された次の JSON によってトリガーされた場合:

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    }
}
```

次にmy/iot/topic/republish、このルールは次の JSON を に発行します。これは から AWS IoT 置き換えられます${topic()}/republish。

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    },
    "timestamp": 1579637878451
}
```

ネストされたオブジェクトのクエリ

ネストされた SELECT 句を使用して、配列および内部 JSON オブジェクト内の属性を照会できます。SQL バージョン 2016-03-23 以降でサポートされています。

次の MQTT メッセージを考えてみます。

```
{
    "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

次のルールを使用して、値を新しい配列に変換できます。

```
SELECT (SELECT VALUE n FROM e) as sensors FROM 'my/topic'
```

この場合、ルールにより、次の出力が生成されます。

```
{
    "sensors": [
        "temperature",
        "light",
        "acidity"
    ]
}
```

Example

同じ MQTT メッセージを使用して、次のルールを使い、ネストされたオブジェクト内の特定の値を照会することもできます。

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

この場合、ルールにより、次の出力が生成されます。

```
{
    "temperature": [
        {
            "v": 22.5
        }
    ]
}
```

Example

また、より複雑なルールで出力を平坦化することもできます。

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'), 0).v as temperature FROM 'topic'
```

この場合、ルールにより、次の出力が生成されます。

```
{
    "temperature": 22.5
}
```

バイナリペイロードの使用

メッセージのペイロードを raw バイナリデータとして (JSON オブジェクトではなく) 処理するには、* 演算子を使用して SELECT 句で参照できます。

このトピックの内容

- バイナリペイロードの例
- protobuf メッセージペイロードのデコード

バイナリペイロードの例

メッセージペイロードを raw バイナリデータとして参照するために * を使用するときは、ルールにデータを追加できます。空のペイロードまたは JSON ペイロードがある場合、結果のペイロードには、ルールを使用してデータを追加できます。以下は、サポートされる SELECT 句の例です

- バイナリペイロードに * のみを使用した以下の SELECT 句を使用できます。
  - ```
    SELECT * FROM 'topic/subtopic'
    ```
  - ```
    SELECT * FROM 'topic/subtopic' WHERE timestamp() % 12 = 0
    ```
- データを追加して、以下の SELECT 句を使用することもできます。
  - ```
    SELECT *, principal() as principal, timestamp() as time FROM 'topic/subtopic'
    ```
  - ```
    SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'topic/subtopic'
    ```
- これらの SELECT 句をバイナリペイロードと使用することもできます。
  - 以下は、WHERE 句の device_type を参照します。

```
SELECT * FROM 'topic/subtopic' WHERE device_type = 'thermostat'
```

- 以下もサポートされています。

```
{
 "sql": "SELECT * FROM 'topic/subtopic'",
 "actions": [
  {
   "republish": {
    "topic": "device/${device_id}"
   }
  }
 ]
}
```

次のルールアクションはバイナリペイロードをサポートしていないので、それらをデコードする必要があります。

- Lambdaアクションなど、バイナリペイロード入力をサポートしないルールアクションの場合は、バイナリペイロードをデコードする必要があります。Lambda ルールアクションは、base64 エンコード済みで JSON ペイロードの場合、バイナリデータを受け取ることができます。ルールを以下のように変更することで、これを実行できます。

```
SELECT encode(*, 'base64') AS data FROM 'my_topic'
```

- SQL ステートメントは、文字列を入力としてサポートしていません。文字列入力を JSON に変換するには、次のコマンドが実行できます。

```
SELECT decode(encode(*, 'base64'), 'base64') AS payload FROM 'topic'
```

protobuf メッセージペイロードのデコード

プロトコルバッファ (protobuf) は、構造化データをコンパクトなバイナリ形式でシリアル化するために使用されるオープンソースのデータ形式です。データをネットワーク経由で送信したり、ファイルに保存したりするために使用されます。Protobuf を使用すると、小さなパケットサイズで他のメッセージング形式よりも高速にデータを送信できます。 AWS IoT Core ルールは、decode(value, decodingScheme) SQL 関数を提供することで protobuf をサポートします。これにより、protobuf

エンコードされたメッセージペイロードを JSON 形式にデコードし、ダウンストリームサービスにルーティングできます。このセクションでは、 AWS IoT Core ルールで protobuf デコードを設定する手順について、順を追って説明します。

このセクションの内容:

- 前提条件
- 記述子ファイルの作成
- 記述子ファイルを S3 バケットにアップロードする
- ルールで protobuf デコードを設定する
- 制限事項
- ベストプラクティス

前提条件

- プロトコルバッファ (protobuf) に関する基本事項の理解
- メッセージタイプと関連する依存関係を定義する `.proto` ファイル
- システムへの protobuf コンパイラ (protoc) のインストール

記述子ファイルの作成

記述子ファイルが既にある場合は、このステップを省略できます。記述子ファイル (`.desc`) は `.proto` ファイルのコンパイル版で、protobuf のシリアル化で使用されるデータ構造とメッセージタイプを定義するテキストファイルです。記述子ファイルを生成するには、`.proto` ファイルを定義し、protoc コンパイラを使用してそれをコンパイルする必要があります。

1. メッセージタイプを定義する `.proto` ファイルを作成します。`.proto` ファイルの例として、以下のようなものがあります。

```
syntax = "proto3";

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

この例の .proto ファイルでは、proto3 構文を使用してメッセージタイプ Person を定義します。Person メッセージ定義では、3 つのフィールド (名前、ID、E メール) を指定します。.proto ファイルメッセージ形式の詳細については、言語ガイド (proto3)を参照してください。

2. protoc コンパイラを使用して、.proto ファイルをコンパイルし、記述子ファイルを生成します。descriptor (.desc) ファイルを作成するコマンドの例として、次のものがあります。

```
protoc --descriptor_set_out=<FILENAME>.desc \
    --proto_path=<PATH_TO_IMPORTS_DIRECTORY> \
    --include_imports \
    <PROTO_FILENAME>.proto
```

このコマンド例では、記述子ファイル を生成します。このファイル を使用して<FILENAME>.desc、 AWS IoT Core で定義されたデータ構造に準拠する protobuf ペイロードをデコードできます<PROTO_FILENAME>.proto。

- --descriptor_set_out

  生成する記述子ファイル (<FILENAME>.desc) の名前を指定します。

- --proto_path

  コンパイル中の .proto ファイルから参照するインポートされたファイルの場所を指定します。インポートされた .proto ファイルの場所が異なる場合は、フラグを複数回指定できます。

- --include_imports

  インポートされた .proto ファイルもすべてコンパイルして、<FILENAME>.desc 記述ファイルに含めるように指定します。

- <PROTO_FILENAME>.proto

  コンパイルする .proto ファイルの名前を指定します。

protoc リファレンスの詳細については、API リファレンスを参照してください。

記述子ファイルを S3 バケットにアップロードする

記述子ファイル を作成したら<FILENAME>.desc、 AWS API、 AWS SDK、または を使用して、記述子ファイルを <FILENAME>.desc Amazon S3 バケットにアップロードします AWS マネジメントコンソール。

重要な考慮事項

- ルールを設定する AWS リージョン のと同じ AWS アカウント の Amazon S3 バケットに記述子ファイルをアップロードしてください。
- S3 FileDescriptorSetから を読み取る AWS IoT Core ためのアクセス許可を必ず付与してください。S3 バケットでサーバー側の暗号化が無効になっている場合、または S3 バケットが Amazon S3 管理キー (SSE-S3) を使用して暗号化されている場合、追加のポリシー設定は必要ありません。これは、バケットポリシーの例で実現できます。

```
{
 "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 バケットが AWS Key Management Service キー (SSE-KMS) を使用して暗号化されている場合は、S3 バケットにアクセスするときにキーを使用するアクセス AWS IoT Core 許可を付与してください。そのためには、次のステートメントをキーポリシーに追加してください。

```
{
 "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"

}
```

ルールで protobuf デコードを設定する

記述子ファイルを S3 バケットにアップロードしたら、decode(value, decodingScheme) SQL 関数を使用して、protobuf メッセージペイロード形式をデコードできるルールを設定します。詳細な関数の署名と例は、「AWS IoT SQL リファレンス」の「decode(value, decodingScheme) SQL 関数」に記載されています。

decode(value, decodingScheme) 関数を使用する SQL 式の例としては、次のようなものがあります。

```
SELECT VALUE decode(*, 'proto', '<BUCKET NAME>', '<FILENAME>.desc', '<PROTO_FILENAME>',
 '<PROTO_MESSAGE_TYPE>') FROM '<MY_TOPIC>'
```

この式の例:

- decode(value, decodingScheme) SQL 関数を使用して、* から参照されるバイナリメッセージペイロードをデコードします。これは、protobuf でエンコードされたバイナリのペイロード、または base64 でエンコードされた protobuf ペイロードを表す JSON 文字列です。
- 提供されたメッセージペイロードは、PROTO_FILENAME.proto で定義されている Person メッセージタイプを使用してエンコードされます。
- BUCKET NAME という名前の Amazon S3 バケットには、PROTO_FILENAME.proto から生成された FILENAME.desc が含まれます。

設定が完了したら、ルールがサブスクライブされているトピック AWS IoT Core のメッセージを に発行します。

制限事項

AWS IoT Core ルールは protobuf をサポートしますが、以下の制限があります。

- 置換テンプレート内の protobuf メッセージペイロードのデコードはサポートされていません。
- protobuf メッセージペイロードをデコードする場合、1 つの SQL 式内で decode SQL 関数を最大 2 回使用できます。
- インバウンドペイロードの最大サイズは 128 KiB (1KiB = 1024 バイト)、アウトバウンドペイロードの最大サイズは 128 KiB、Amazon S3 バケットに保存される `FileDescriptorSet` オブジェクトの最大サイズは 32 KiB です。
- SSE-C 暗号化を使用して暗号化された Amazon S3 バケットはサポートされていません。

ベストプラクティス

ここでは、ベストプラクティスおよびトラブルシューティングのヒントを説明します。

- Amazon S3 バケットに proto ファイルをバックアップする。

  問題が発生した場合に備えて、proto ファイルをバックアップすることをお勧めします。例えば、protoc の実行中にバックアップせずに proto ファイルを誤って変更すると、本稼働スタックで問題が発生する可能性があります。Amazon S3 バケットのファイルをバックアップする方法は複数あります。例えば、S3 バケットでバージョニングを使用できます。Amazon S3 バケット内のファイルをバックアップする方法の詳細については、「Amazon S3 デベロッパーガイド」を参照してください。
- AWS IoT ログエントリを表示するようにログ記録を設定します。

  CloudWatch でアカウントの AWS IoT ログをチェックできるように、 AWS IoT ログ記録を設定することをお勧めします。ルールの SQL クエリが外部関数を呼び出すと、 AWS IoT Core Rules は `eventType`の を持つログエントリを生成します。これには`FunctionExecution`、障害のトラブルシューティングに役立つ理由フィールドが含まれます。Amazon S3 オブジェクトが見つからない、または無効な protobuf ファイル記述子が含まれていることが考えられます。 AWS IoT ロギングを設定する方法とログエントリを確認する方法の詳細については、「AWS IoT ロギングの設定」と「ルールエンジンのログエントリ」を参照してください。
- 新しいオブジェクトキーを使用して `FileDescriptorSet` を更新し、ルール内のオブジェクトキーを更新する。

  更新された記述子ファイルを Amazon S3 バケットにアップロードすることで `FileDescriptorSet` を更新できます。`FileDescriptorSet` への更新が反映されるまで、最大 15 分かかる場合があります。この遅延を避けるため、新しいオブジェクトキーを使用して更新した `FileDescriptorSet` をアップロードし、ルール内のオブジェクトキーを更新することをお勧めします。

SQL バージョン

AWS IoT ルールエンジンは、SQL のような構文を使用して MQTT メッセージからデータを選択します。SQL ステートメントは、ルールが記述されている JSON ドキュメント内の `awsIotSqlVersion` プロパティで指定された SQL バージョンに基づいて解釈されます。JSON ルールドキュメントの構造については、「ルールの作成」を参照してください。`awsIotSqlVersion` プロパティを使用すると、使用する AWS IoT SQL ルールエンジンのバージョンを指定できます。新しいバージョンをデプロイした場合は、引き続き古いバージョンを使用することも、新しいバージョンを使用できるようにルールを変更することもできます。現在のルールでは、ルールの作成時のバージョンが引き続き使用されます。

以下の JSON 例は、`awsIotSqlVersion` プロパティを使用して SQL バージョンを指定する方法を示しています。

```
{
    "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 は現在、次の SQL バージョンをサポートしています。

- `2016-03-23` – 2016 年 3 月 23 日にビルドされた SQL バージョン (推奨)。
- `2015-10-08` – 2015 年 10 月 8 日にビルドされた元の SQL バージョン。
- `beta` – 最新のベータ SQL バージョン。このバージョンでは、ルールへの変更が必要になる場合があります。

SQL ルールエンジン 2016-03-23 バージョンの最新情報

- 入れ子になっている JSON オブジェクトの選択が修正されました。
- 配列クエリに関する修正が行われました。

- オブジェクト間でのクエリがサポートされるようになりました。詳細については、「ネストされたオブジェクトのクエリ」を参照してください。
- 最上位レベルのオブジェクトとして配列を出力できるようになりました。
- JSON および非 JSON 形式のデータに適用できる encode(value, encodingScheme) 関数の追加。詳細については、「encode 関数」を参照してください。

最上位レベルのオブジェクトとして **Array** を出力する

この機能を使用すると、ルールから、最上位レベルのオブジェクトとして配列を返すことができます。たとえば、次の MQTT メッセージの場合:

```
{
    "a": {"b":"c"},
    "arr":[1,2,3,4]
}
```

次のルールを使用します。

```
SELECT VALUE arr FROM 'topic'
```

この場合、ルールにより、次の出力が生成されます。

```
[1,2,3,4]
```

AWS IoT Device Shadow サービス

AWS IoT Device Shadow サービスは、 AWS IoT モノのオブジェクトにシャドウを追加します。シャドウは、デバイスが に接続されている AWS IoT かどうかにかかわらず、デバイスの状態をアプリケーションやその他のサービスで利用できるようにします。 AWS IoT モノのオブジェクトには複数の名前付きシャドウを含めることができるため、IoT ソリューションには、デバイスを他のアプリケーションやサービスに接続するためのより多くのオプションがあります。

AWS IoT Thing オブジェクトには、明示的に作成されるまでシャドウはありません。シャドウは、AWS IoT コンソールを使用して作成、更新、削除できます。デバイス、その他のウェブクライアント、サービスは、MQTT および予約された MQTT トピック、Device Shadow REST API を使用する HTTP、AWS CLI for AWS IoT を使用して、シャドウを作成、更新、削除できます。シャドウは によってクラウド AWS に保存されるため、デバイスが接続されているかどうかにかかわらず、アプリケーションやその他のクラウドサービスからデバイスの状態データを収集してレポートできます。

シャドウの使用

シャドウは、デバイス、アプリ、その他のクラウドサービスでデータを共有するための信頼性の高いデータストアを提供します。これにより、デバイス、アプリ、その他のクラウドサービスが、デバイスの状態を失うことなく接続および切断できます。

デバイス、アプリケーション、およびその他のクラウドサービスが接続されている間は AWS IoT、シャドウを介してデバイスの現在の状態にアクセスして制御できます。たとえば、アプリはシャドウを更新することでデバイスの状態の変更をリクエストできます。 は、デバイスへの変更を示すメッセージ AWS IoT を発行します。デバイスはこのメッセージを受信し、一致するように状態を更新し、更新された状態のメッセージを発行します。Device Shadow サービスは、この更新された状態を対応するシャドウに反映します。アプリはシャドウの更新をサブスクライブすることも、シャドウに現在の状態を照会することもできます。

デバイスがオフラインになっても、アプリは引き続き AWS IoT およびデバイスのシャドウと通信できます。デバイスは再接続すると、シャドウの現在の状態を受信し、シャドウの状態と一致するように状態を更新し、更新された状態のメッセージを発行します。同様に、アプリがオフラインになり、デバイスがオフライン中に状態が変わると、デバイスはシャドウを更新したままにして、アプリが再接続したときに現在の状態のシャドウを照会できるようにします。

デバイスが頻繁にオフラインで、再接続後にデルタメッセージを受信するようにデバイスを設定する場合は、永続セッション機能を使用できます。永続セッションの有効期間の詳細については、「永続的セッションの有効期間」を参照してください。

名前付きのシャドウまたは名前のないシャドウの使用の選択

Device Shadow サービスは、名前付きと名前のない、またはクラシックな、シャドウをサポートします。Thing オブジェクトは、複数の名前付きシャドウを持つことができます。また、名前のないシャドウを 1 つ以上持つことはできません。Thing オブジェクトは、予約済みの名前付きシャドウを持つこともできます。名前を更新できないという点を除けば、名前付きシャドウと同様に機能します。詳細については、「予約済みの名前付きシャドウ」を参照してください。

Thing オブジェクトは、名前付きのシャドウと名前のないシャドウを同時に持つことができますが、それぞれにアクセスするために使用する API は若干異なるため、ソリューションに最適なシャドウのタイプを決定し、そのタイプのみを使用する方が効率的です。シャドウにアクセスするための API の詳細については、「シャドウトピック」を参照してください。

名前付きシャドウを使用すると、Thing オブジェクトの状態をさまざまなビューで作成できます。たとえば、多数のプロパティを持つ Thing オブジェクトを、それぞれがシャドウ名で識別される論理的なプロパティのグループを持つシャドウに分割できます。また、プロパティを別のシャドウにグループ化し、ポリシーを使用してアクセスを制御することで、プロパティへのアクセスを制限することもできます。デバイスシャドウで使用するポリシーの詳細については、「AWS IoTのアクション、リソース、および条件キー」と「AWS IoT Core のポリシー」を参照してください。

クラシックな名前のないシャドウは単純ですが、名前付きシャドウよりも多少制限があります。各 AWS IoT モノのオブジェクトには、名前のないシャドウを 1 つだけ含めることができます。IoT ソリューションでシャドウデータの必要性が限られている場合は、そのようにシャドウの使用を開始することができます。ただし、将来、シャドウを追加すると思われる場合は、最初から名前付きのシャドウを使用することを検討してください。

フリートインデックスによるサポートは、名前のないシャドウと名前付きシャドウで異なります。詳細については、「Manage fleet indexing」(フリートインデックスの管理) を参照してください。

シャドウにアクセスする

すべてのシャドウには、予約された MQTT トピックと HTTP URL があり、シャドウに対する `get`、`update`、`delete` のアクションをサポートします。

シャドウは JSON シャドウドキュメントを使用してデータを格納および取得します。シャドウのドキュメントには、デバイスの状態の次の側面を説明する state プロパティが含まれています。

- `desired`

アプリは、desired オブジェクトを更新することによって、デバイスプロパティの desired 状態を指定します。

- reported

  デバイスは、reported オブジェクト内の現在の状態を報告します。

- delta

  AWS IoT は、deltaオブジェクト内の目的の状態と報告された状態の違いを報告します。

シャドウに格納されるデータは、更新アクションのメッセージ本文の state プロパティによって決まります。それ以降の更新アクションでは、既存のデータオブジェクトの値を変更したり、シャドウの state オブジェクト内のキーやその他の要素を追加および削除したりできます。シャドウへのアクセス方法の詳細については、「デバイスでのシャドウの使用」および「アプリとサービスでのシャドウの使用」を参照してください。

> **Important**
>
> 更新リクエストを行うアクセス許可は、信頼できるアプリとデバイスに制限する必要があります。これにより、シャドウの state プロパティが予期せず変更されるのを防ぐことができます。そうしないと、シャドウを使用するデバイスおよびアプリは、state プロパティのキーが変更されることを期待するように設計する必要があります。

デバイス、アプリ、その他のクラウドサービスでのシャドウの使用

デバイス、アプリ、その他のクラウドサービスでシャドウを使用するには、これらすべての一貫性と調整が必要です。 AWS IoT Device Shadow サービスはシャドウ状態を保存し、シャドウ状態が変更されたときにメッセージを送信し、状態を変更するメッセージに応答します。IoT ソリューション内のデバイス、アプリ、その他のクラウドサービスは、その状態を管理し、デバイスシャドウの状態と整合性を維持する必要があります。

シャドウ状態データは動的であり、シャドウへのアクセス許可を持つデバイス、アプリ、その他のクラウドサービスによって変更できます。このため、各デバイス、アプリ、その他のクラウドサービスがシャドウとどのようにやり取りするかを検討することが重要です。以下に例を示します。

- デバイスは、状態データをシャドウに伝達するときに、シャドウ状態の reported プロパティにのみ書き込む必要があります。

- アプリおよびその他のクラウドサービスは、状態変更リクエストをシャドウを介してデバイスに通信するときのみ、desired プロパティに書き込む必要があります。

> **Important**
>
> シャドウデータオブジェクトに含まれるデータは、他のシャドウや Thing オブジェクトのプロパティ (Thing の属性や MQTT メッセージのコンテンツなど) から独立しています。ただし、デバイスは、必要に応じて、異なる MQTT トピックとシャドウで同じデータを報告できます。
> 複数のシャドウをサポートするデバイスは、異なるシャドウで報告するデータの整合性を維持する必要があります。

メッセージの順序

AWS IoT サービスからのメッセージが特定の順序でデバイスに到着する保証はありません。次のシナリオは、この場合に何が起こるかを示しています。

初期状態ドキュメント:

```
{
  "state": {
    "reported": {
      "color": "blue"
    }
  },
  "version": 9,
  "timestamp": 123456776
}
```

更新 1:

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
```

```
}
```

更新 2:

```
{
  "state": {
    "desired": {
      "color": "GREEN"
    }
  },
  "version": 11,
  "timestamp": 123456778
}
```

最終状態ドキュメント:

```
{
  "state": {
    "reported": {
      "color": "GREEN"
    }
  },
  "version": 12,
  "timestamp": 123456779
}
```

これにより、2 つの差分メッセージが生成されます。

```
{
  "state": {
    "color": "RED"
  },
  "version": 11,
  "timestamp": 123456778
}
```

```
{
  "state": {
    "color": "GREEN"
  },
  "version": 12,
```

```
  "timestamp": 123456779
}
```

デバイスは順不同でこれらのメッセージを受信する場合があります。これらのメッセージ内の状態は累積的であるため、デバイスは追跡しているものより古いバージョン番号が含まれるメッセージをすべて安全に破棄できます。デバイスはバージョン 11 の前にバージョン 12 の差分を受信した場合、バージョン 11 のメッセージを安全に破棄できます。

シャドウメッセージのトリム

デバイスに送信されるシャドウメッセージのサイズを小さくするには、デバイスに必要なフィールドのみを選択してから、デバイスのリッスン対象の MQTT トピックにメッセージを再パブリッシュするルールを定義します。

ルールは JSON で指定し、以下のようになります。

```
{
  "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 ステートメントにより、指定したトピックにメッセージのどのフィールドが再パブリッシュされるかが決まります。すべての Shadow 名に一致させるには、"+" のワイルドカードを使用します。ルールでは、一致するすべてのメッセージが指定したトピックに再パブリッシュされるように定義しています。この場合、"topic()" 関数を使用して、再パブリッシュする先のトピックを指定しています。topic(3) は、元のトピック内のモノ名に評価されます。ルール作成の詳細については、「のルール AWS IoT」を参照してください。

デバイスでのシャドウの使用

このセクションでは、デバイスが Device Shadow サービスと通信するための推奨方法である MQTT メッセージを使用したシャドウとの AWS IoT デバイス通信について説明します。

シャドウ通信は、MQTT のパブリッシュ/サブスクライブ通信モデルを使用して、リクエスト/レスポンスモデルをエミュレートします。すべてのシャドウアクションは、リクエストトピック、成功したレスポンストピック (accepted)、エラーレスポンストピック (rejected) で構成されます。

アプリとサービスで、デバイスが接続されているかどうかを判別できるようにする場合は、「デバイスが接続されているかどうかの検出」を参照してください。

> **Important**
>
> MQTT は発行/サブスクライブ通信モデルを使用するため、リクエストトピックを発行する前にレスポンストピックをサブスクライブする必要があります。そうでない場合、発行するリクエストに対するレスポンスを受信しない可能性があります。
> AWS IoT Device SDK を使用して Device Shadow サービス API を呼び出す場合、これは自動的に処理されます。

このセクションの例では、次の表で説明されているように、*ShadowTopicPrefix* が名前付きのシャドウまたは名前のないシャドウを参照できるトピックの省略形を使用します。

シャドウは、名前付き、または名前のないもの (クラシック) にすることができます。それぞれで使用されるトピックは、トピックのプレフィックスでのみ異なります。この表は、各シャドウタイプで使用されるトピックのプレフィックスを示しています。

ShadowTopicPrefix 値	シャドウタイプ
`$aws/things/ thingName /shadow`	名前のない (クラシック) シャドウ
`$aws/things/ thingName /shadow/name/ shadowName`	名前付きシャドウ

> **Important**
>
> アプリまたはサービスによるシャドウの使用が一貫しており、デバイス内の対応する実装でサポートされていることを確認してください。たとえば、シャドウの作成、更新、削除方法を考えてみましょう。また、デバイスおよびシャドウを介してデバイスにアクセスするアプリまたはサービスでの更新の処理方法も考慮してください。デバイスの状態がどのように更

新され、報告され、アプリやサービスがデバイスとそのシャドウとどのように相互作用するかを明確に設計する必要があります。

完全なトピックを作成するには、次の表に示すように、参照するシャドウのタイプの *ShadowTopicPrefix* を選択し、*thingName* と、*shadowName* (該当する場合) を対応する値に置き換え、トピックスタブに追加します。トピックでは大文字と小文字が区別されることに注意してください。

シャドウ用に予約されているトピックの詳細については、シャドウトピック を参照してください。

への最初の接続時にデバイスを初期化する AWS IoT

デバイスが に登録されると AWS IoT、サポートされているシャドウの MQTT メッセージをサブスクライブする必要があります。

Topic	意味	このトピックの受信時にデバイスが実行するアクション
ShadowTopicPrefix / delete/accepted	delete リクエストが受け入れられ、シャドウ AWS IoT が削除されました。	更新の発行を停止するなど、削除されたシャドウに対応するために必要なアクション。
ShadowTopicPrefix / delete/rejected	delete リクエストは によって拒否 AWS IoT され、シャドウは削除されませんでした。メッセージ本文には、エラー情報が含まれています。	メッセージ本文内のエラーメッセージに応答します。
ShadowTopicPrefix / get/accepted	get リクエストは によって受け入れられ AWS IoT、メッセージ本文には現在のシャドウドキュメントが含まれます。	メッセージ本文内の状態ドキュメントを処理するために必要なアクション。
ShadowTopicPrefix / get/rejected	get リクエストは によって拒否され AWS IoT、メッセージ	メッセージ本文内のエラーメッセージに応答します。

Topic	意味	このトピックの受信時にデバイスが実行するアクション
	本文にエラー情報が含まれています。	
ShadowTopicPrefix /update/accepted	update リクエストは によって受け入れられ AWS IoT、メッセージ本文には現在のシャドウドキュメントが含まれます。	メッセージ本文の更新されたデータがデバイスの状態と一致することを確認します。
ShadowTopicPrefix /update/rejected	update リクエストは によって拒否され AWS IoT、メッセージ本文にエラー情報が含まれています。	メッセージ本文内のエラーメッセージに応答します。
ShadowTopicPrefix /update/delta	シャドウドキュメントは へのリクエストによって更新され AWS IoT、メッセージ本文にはリクエストされた変更が含まれています。	メッセージ本文内の目的の状態と一致するようにデバイスの状態を更新します。
ShadowTopicPrefix /update/documents	シャドウの更新が最近完了し、メッセージ本文に現在のシャドウドキュメントが含まれています。	メッセージ本文の更新された状態が、デバイスの状態と一致することを確認します。

各シャドウの前の表のメッセージにサブスクライブした後、デバイスがサポートするシャドウがすでに作成されているかどうかをテストして、各シャドウに /get トピックを発行します。/get/accepted メッセージを受信すると、メッセージ本文にはシャドウドキュメントが含まれます。シャドウドキュメントは、デバイスがその状態を初期化するために使用できます。/get/rejected メッセージを受信した場合は、現在のデバイス状態の /update メッセージを発行してシャドウを作成する必要があります。

例えば、モノ My_IoT_Thing があるとします。クラシックシャドウや名前の付いたシャドウはありません。今予約済みトピック $aws/things/My_IoT_Thing/shadow/get に /get リクエストを発行する場合、モノにシャドウがないため $aws/things/My_IoT_Thing/shadow/get/

rejected トピックのエラーが返されます。このエラーを解決するには、まず次のペイロードなど、現在のデバイス状態の $aws/things/My_IoT_Thing/shadow/update トピックを使用して /update メッセージを発行します。

```
{
 "state": {
  "reported": {
   "welcome": "aws-iot",
   "color": "yellow"
  }
 }
}
```

これで モノのクラシックシャドウが作成され、$aws/things/My_IoT_Thing/shadow/update/accepted トピックへメッセージが発行されます。トピック $aws/things/My_IoT_Thing/shadow/get に発行する場合、デバイスの状態に関する $aws/things/My_IoT_Thing/shadow/get/accepted トピックを返します。

名前付きシャドウの場合、Get リクエストを使用する前に、最初に名前付きシャドウを作成するか、シャドウ名で更新を発行する必要があります。例えば、名前の付いたシャドウ namedShadow1 を作成するには、まず、デバイスの状態情報をトピック $aws/things/My_IoT_Thing/shadow/name/namedShadow1/update に発行します。状態情報を取得するには、名前付きシャドウ $aws/things/My_IoT_Thing/shadow/name/namedShadow1/get の /get リクエストを使用します。

デバイスが に接続されている間のメッセージの処理 AWS IoT

デバイスが に接続されている間は AWS IoT、/update/delta メッセージを受信でき、次の方法でデバイスの状態をシャドウの変更と一致させる必要があります。

1. 受信したすべての /update/delta メッセージを読み取り、一致するようにデバイスの状態を同期します。
2. デバイスの状態が変更されるたびに、デバイスの現在の状態を含む reported メッセージ本文を含む /update メッセージを発行します。

デバイスが接続されている間は、これらのメッセージが表示されたら発行する必要があります。

表示	トピック	Payload
デバイスの状態が変更されました。	*ShadowTopicPrefix* /update	reported プロパティを持つシャドウドキュメント。
デバイスがシャドウと同期していない可能性があります。	*ShadowTopicPrefix* /get	(空)
デバイスに対するアクションは、デバイスの削除または交換時など、デバイスによってシャドウがサポートされなくなることを示します	*ShadowTopicPrefix* /delete	(空)

デバイスが に再接続したときのメッセージの処理 AWS IoT

1 つ以上のシャドウを持つデバイスが に接続する場合 AWS IoT、その状態を、以下によってサポートされるすべてのシャドウの状態と同期する必要があります。

1. 受信したすべての /update/delta メッセージを読み取り、一致するようにデバイスの状態を同期します。
2. デバイスの現在の状態を記載した reported メッセージ本文を含む /update メッセージを発行します。

アプリとサービスでのシャドウの使用

このセクションでは、アプリケーションまたはサービスが AWS IoT Device Shadow サービスとやり取りする方法について説明します。この例では、アプリまたはサービスがシャドウと、シャドウを介してデバイスとのみ相互作用していることを前提としています。この例には、シャドウの作成や削除などの管理アクションは含まれていません。

この例では、 AWS IoT Device Shadow サービスの REST API を使用してシャドウを操作します。発行/サブスクライブ通信モデルを使用する デバイスでのシャドウの使用 で使用される例とは異なり、この例では REST API のリクエスト/レスポンス通信モデルを使用します。つまり、アプリケーションまたはサービスは、レスポンスを受信する前にリクエストを行う必要があります AWS IoT。ただし、このモデルの欠点は、通知をサポートしていないことです。アプリまたはサービスで、デバイス

の状態の変更をタイムリーに通知する必要がある場合は、「デバイスでのシャドウの使用」で説明されているように、パブリッシュ/サブスクライブ通信モデルをサポートする WSS プロトコル経由の MQTT または MQTT を検討してください。

> **Important**
>
> アプリまたはサービスによるシャドウの使用が、デバイス内の対応する実装と一致し、サポートされていることを確認します。たとえば、シャドウの作成、更新、削除方法、デバイスとシャドウにアクセスするアプリまたはサービスで更新がどのように処理されるかを検討します。デザインでは、デバイスの状態の更新と報告方法、アプリやサービスがデバイスとそのシャドウとどのように相互作用するかを明確に指定する必要があります。

名前付きシャドウの REST API の URL は次のとおりです。

```
https://endpoint/things/thingName/shadow?name=shadowName
```

名前のないシャドウの場合：

```
https://endpoint/things/thingName/shadow
```

各パラメータの意味は次のとおりです。

エンドポイント

CLI コマンドによって返されるエンドポイント。

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

thingName

シャドウが属する Thing オブジェクトの名前

shadowName

名前付きシャドウの名前。このパラメータは、名前のないシャドウでは使用されません。

への接続時にアプリまたはサービスを初期化する AWS IoT

アプリが最初に接続するときは AWS IoT、使用しているシャドウの現在の状態を取得するために使用するシャドウの URLs に HTTP GET リクエストを送信する必要があります。これにより、アプリまたはサービスをシャドウと同期させることができます。

アプリケーションまたはサービスが に接続されている間の処理状態の変更 AWS IoT

アプリケーションまたはサービスが接続されている間は AWS IoT、使用するシャドウの URLs に HTTP GET リクエストを送信することで、定期的に現在の状態をクエリできます。

エンドユーザーがアプリまたはサービスと対話してデバイスの状態を変更すると、アプリまたはサービスは、シャドウの `desired` 状態を更新するために使用するシャドウの URL に HTTP POST リクエストを送信できます。このリクエストは受け入れられた変更を返しますが、デバイスがシャドウを新しい状態で更新するまで HTTP GET リクエストを行い、シャドウをポーリングする必要がある場合があります。

デバイスが接続されているかどうかの検出

デバイスが現在接続されているかどうかを確認するには、シャドウドキュメントに `connected` プロパティを含めて、MQTT Last Wy and Testament (LWT) メッセージを使用して、デバイスがエラーにより切断された場合に、`connected` プロパティを `false` に設定します。

> **Note**
>
> AWS IoT 予約済みトピック ($ で始まるトピック) に送信される MQTT LWT メッセージは、AWS IoT Device Shadow サービスによって無視されます。ただし、これらはサブスクライブされたクライアントと AWS IoT ルールエンジンによって処理されるため、予約されていないトピックに送信される LWT メッセージと、シャドウの予約された更新トピック にシャドウ更新メッセージとして MQTT LWT メッセージを再発行するルールを作成する必要があります*ShadowTopicPrefix*`/update`。

Device Shadow サービスに LWT メッセージを送信するには

1. 予約されたトピックで MQTT LWT メッセージを再発行するルールを作成します。次の例は、`my/things/myLightBulb/update` トピックに関するメッセージをリッスンし、`$aws/things/myLightBulb/shadow/update` に再発行するルールです。

```
{
    "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. デバイスが に接続すると AWS IoT、再発行ルールが認識できるように、予約されていないトピックに LWT メッセージを登録します。この例では、トピックが my/things/myLightBulb/update であり、connected プロパティを false に設定します。

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

3. 接続後、デバイスはシャドウ更新トピック、$aws/things/myLightBulb/shadow/update に関するメッセージを発行し、connected プロパティを true に設定することを含む現在の状態を報告します。

```
{
     "state": {
        "reported": {
            "connected":"true"
        }
    }
}
```

4. デバイスは正常に切断する前に、シャドウ更新トピック、$aws/things/myLightBulb/shadow/update に関するメッセージを発行し、connected プロパティを false に設定することを含む最新の状態を報告します。

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

5. エラーによりデバイスが切断された場合、 AWS IoT メッセージブローカーはデバイスに代わってデバイスの LWT メッセージを発行します。再発行ルールはこのメッセージを検出し、シャドウ更新メッセージを発行してデバイスシャドウの connected プロパティを更新します。

> **Note**
>
> 切断処理は非同期であるため、再接続時に LWT メッセージが順番にディスパッチされるとは限りません。接続状態の検出精度を向上させるために、ライフサイクルイベントを使用することをお勧めします。これらのイベントでは、順不同のイベントを管理するための属性が提供されています。

デバイスシャドウサービス通信のシミュレーション

このトピックでは、Device Shadow サービスが仲介として動作する方法を示し、デバイスおよびアプリはシャドウを使用してデバイスの状態を更新、保存、取得できます。

このトピックで説明されているやり取りをデモンストレーションし、さらに詳しく調べるには、AWS アカウント と、 を実行できるシステムが必要です AWS CLI。これらがない場合は、コード例で相互作用を確認できます。

この例では、 AWS IoT コンソールは デバイスを表します。は、シャドウを介してデバイスにアクセスするアプリまたはサービス AWS CLI を表します。 AWS CLI インターフェイスは、アプリケーションが通信に使用する API と非常によく似ています AWS IoT。この例のデバイスはスマート電球で、アプリは電球の状態を表示し、電球の状態を変更できます。

シミュレーションの設定

これらの手順では、デバイスをシミュレートする AWS IoT コンソールと、アプリをシミュレートするコマンドラインウィンドウを開いてシミュレーションを初期化します。

シミュレーション環境を設定するには

1. このトピックの例を自分で実行する AWS アカウント には、 が必要です。がない場合は AWS アカウント、「」の説明に従って作成しますセットアップ AWS アカウント。
2. AWS IoT コンソールを開き、左側のメニューで [テスト] を選択して MQTT クライアントを開きます。
3. 別のウィンドウで、 AWS CLI がインストールされているシステムでターミナルウィンドウを開きます。

2 つのウィンドウが開いている必要があります。1 つはテストページの AWS IoT コンソール、もう 1 つはコマンドラインプロンプトです。

デバイスの初期化

このシミュレーションでは、mySimulatedThing という名前の Thing オブジェクトと、SimShadow1 という名前のシャドウを使って作業します。

モノのオブジェクトとその IoT ポリシーを作成する

AWS IoT コンソールでモノのオブジェクトを作成するには

1. [Manage] (管理) を選択し、[Things] (モノ) を選択します。
2. モノが一覧表示されている場合は「作成」ボタンをクリックするか、「モノの登録」をクリックしてモノを 1 つ作成 AWS IoT します。
3. 名前 `mySimulatedThing` を入力し、その他の設定はデフォルトのままにして、[Next] (次へ) をクリックします。
4. ワンクリックの証明書作成を使用して、 AWS IoTへのデバイスの接続を認証する証明書を生成します。[Activate] (有効化) をクリックして証明書を有効化します。
5. MQTT 予約トピックを発行およびサブスクライブするためのアクセス許可をデバイスに付与するポリシー `My_IoT_Policy` をアタッチできます。 AWS IoT モノの作成方法とこのポリシーの作成方法の詳細については、「」を参照してくださいモノのオブジェクトを作成する。

モノのオブジェクトの名前の付いたシャドウを作成する

以下に示すように、トピック `$aws/things/mySimulatedThing/shadow/name/simShadow1/update` に更新リクエストを発行することで、モノの名前付きシャドウを作成できます。

または、名前付きシャドウを作成するには:

1. AWS IoT コンソールで、表示されたモノのリストから自分のモノオブジェクトを選択し、[Shadows] (シャドウ) を選択します。
2. [Add a shadow] (シャドウの追加) を選択し、名前 `simShadow1` を入力してから、[Create] (作成) を選択して名前付きシャドウを追加します。

予約済みの MQTT トピックをサブスクライブして発行する

コンソールで、予約された MQTT シャドウのトピックをサブスクライブします。これらのトピックは `get`、`update`、`delete` アクションに対するレスポンスです。これにより、デバイスがアクションを発行した後にレスポンスを受信できます。

MQTT クライアントで MQTT トピックをサブスクライブするには

1. MQTT クライアント で、[Subscribe to topic] (トピックのサブスクライブ) を選択します。
2. サブスクライブする `get`、`update`、および `delete` トピックを入力します。次のリストから一度に 1 つのトピックをコピーして [Topic filter] (トピックフィルター) フィールドに貼り付け、[Subscribe] (サブスクライブ) をクリックします。[Subscriptions] (サブスクリプション) の下にトピックが表示されます。

   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/accepted`
   - `$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`

   この時点で、シミュレートされたデバイスは、 AWS IoTによって発行されるトピックを受信する準備が整いました。

MQTT クライアントで MQTT トピックを発行するには

デバイス自体を初期化してレスポンストピックにサブスクライブした後、サポートしているシャドウを照会する必要があります。このシミュレーションは、simShadow1 という名前の mySimulatedThing というモノのオブジェクトをサポートする 1 つのシャドウのみをサポートします。

MQTT クライアントから現在のシャドウ状態を取得するには

1. MQTT クライアントで、[トピックへの発行] を選択します。
2. [Publish] (発行) で、次のトピックを入力し、取得するトピックを入力した下のメッセージ本文ウィンドウからコンテンツを削除します。その後、[Publish to topic] (トピックに発行) を選択してリクエストを発行できます。`$aws/things/mySimulatedThing/shadow/name/simShadow1/get`。

   名前付きシャドウ 、`simShadow1` 、を作成していない場合は、`$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected`トピックでメッセージを受信し、`code` が`404` の場合 (この例のように)、シャドウは作成されていないため、次に作成します。

   ```
   {
     "code": 404,
     "message": "No shadow exists with name: 'simShadow1'"
   }
   ```

デバイスの現在のステータスを持つシャドウを作成するには

1. [MQTT client] (MQTT クライアント) で、[Publish to a topic] (トピックへの発行) を選択し、このトピックを入力します。

   ```
   $aws/things/mySimulatedThing/shadow/name/simShadow1/update
   ```

2. トピックを入力した下のメッセージ本文ウィンドウで、このシャドウドキュメントを入力して、デバイスが ID と現在の色を RGB 値で報告していることを示します。[Publish] (発行) を選択してリクエストを発行します。

   ```
   {
     "state": {
       "reported": {
   ```

```
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

トピックでメッセージを受信した場合は、次のようになります。

- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted: これは、シャドウが作成され、メッセージ本文に現在のシャドウドキュメントが含まれていることを意味します。
- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected: メッセージ本文内のエラーを確認します。
- $aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted: シャドウは既に存在し、この例のようにメッセージ本文には現在のシャドウ状態があります。これにより、デバイスを設定したり、シャドウ状態と一致していることを確認したりできます。

```
{
  "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"
}
```

アプリからアップデートを送信する

このセクションでは、 を使用して AWS CLI 、アプリがシャドウとやり取りする方法を示します。

を使用してシャドウの現在の状態を取得するには AWS CLI

コマンドラインで、次のコマンドを入力します。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1 /
dev/stdout
```

Windows プラットフォームでは、/dev/stdoutの代わりに con を使用できます。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1
 con
```

シャドウが存在し、デバイスによって現在の状態を反映するように初期化されているため、次のシャドウドキュメントが返されます。

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
```

```
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591141111
}
```

アプリはこのレスポンスを使用して、デバイスの状態の表現を初期化できます。

エンドユーザーがスマート電球の色を黄色に変更した場合など、アプリが状態を更新した場合、アプリは update-thing-shadow コマンドを送信します。このコマンドは UpdateThingShadow REST API に対応します。

アプリからシャドウを更新するには

コマンドラインで、次のコマンドを入力します。

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
```

成功した場合、このコマンドは次のシャドウドキュメントを返します。

```
{
  "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"
}
```

デバイスでの更新に応答

AWS コンソールで MQTT クライアントに戻ると、前のセクションで AWS IoT 発行された更新コマンドを反映するために が発行したメッセージが表示されます。

MQTT クライアントで更新メッセージを表示するには

[MQTT client] (MQTT クライアント) の、[Subscriptions] (サブスクリプション) 列で $aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta を選択します。トピック名が切り詰められている場合は、トピック名を一時停止してトピック全体を表示できます。このトピックのトピックログには、次のような /delta メッセージが表示されます。

```
{
  "version": 4,
  "timestamp": 1591141596,
  "state": {
    "ColorRGB": [
      255,
      255,
      0
    ]
  },
  "metadata": {
    "ColorRGB": [
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      }
    ]
  },
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

デバイスはこのメッセージの内容を処理して、デバイスの状態がメッセージ内の desired 状態と一致するように設定します。

デバイスは、メッセージ内の状態と一致するようにdesired状態を更新した後、更新メッセージを公開 AWS IoT して、新しい報告された状態を に送信する必要があります。この手順では、MQTT クライアントでこれをシミュレートします。

デバイスからシャドウを更新するには

1. MQTT クライアントで、[トピックへの発行] を選択します。
2. メッセージ本文ウィンドウで、メッセージ本文ウィンドウの上にあるトピックフィールドに、シャドウのトピックを入力し、その後に /update アクションを入力します: $aws/things/mySimulatedThing/shadow/name/simShadow1/updateとメッセージ本文に、デバイスの現在の状態を説明するこの更新されたシャドウドキュメントを入力します。[Publish] (発行) をクリックして、更新されたデバイス状態を発行します。

```
{
  "state": {
    "reported": {
      "ColorRGB": [255,255,0]
      }
  },
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

メッセージが によって正常に受信された場合 AWS IoT、この例のように、MQTT クライアントの $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted メッセージログに、シャドウの現在の状態を含む新しいレスポンスが表示されます。

```
{
  "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"
}
```

デバイスの報告された状態を正常に更新 AWS IoT すると、 は、前述の手順でデバイスによって実行されたシャドウ更新に起因するこのメッセージ本文など、メッセージのシャドウ状態の包括的な説明を update/documentsトピックに送信します。

```
{
  "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"
}
```

アプリで更新を確認する

アプリは、デバイスによって報告された現在の状態をシャドウに照会できるようになりました。

を使用してシャドウの現在の状態を取得するには AWS CLI

1. コマンドラインで、次のコマンドを入力します。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 /dev/stdout
```

Windows プラットフォームでは、/dev/stdoutの代わりに conを使用できます。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 con
```

2. シャドウは現在の状態を反映するようにデバイスによって更新されたばかりなので、次のシャドウドキュメントを返します。

```
{
  "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
}
```

シミュレーションを超える

AWS CLI (アプリを表す) とコンソール (デバイスを表す) の間の相互作用を試して、IoT ソリューションをモデル化します。

シャドウとの相互作用

このトピックでは、シャドウを操作するために AWS IoT が提供する 3 つの方法のそれぞれに関連するメッセージについて説明します。これらの方法には、次のものがあります。

UPDATE

存在しない場合はシャドウを作成します。メッセージ本文に指定された状態情報で既存のシャドウの内容を更新します。 AWS IoT は、更新ごとにタイムスタンプを記録して、状態が最後に更新された日時を示します。シャドウの状態が変更されると、 AWS IoT はすべての MQTT サブス

クライバーに、 状態desiredと reported状態の違いとともに/deltaメッセージを送信します。/delta メッセージを受信するデバイスまたはアプリは、違いに基づいてアクションを実行できます。たとえば、デバイスは自らの状態を desired 状態に更新でき、アプリケーションはデバイスの状態の変更を表示するようにその UI を更新できます。

GET

メタデータを含むシャドウの完全な状態を含む現在のシャドウドキュメントを取得します。

DELETE

デバイスシャドウとそのコンテンツを削除します。

削除したデバイスシャドウドキュメントを復元することはできませんが、削除したデバイスシャドウドキュメントの名前で新しいデバイスシャドウを作成することはできます。過去 48 時間以内に削除されたものと同じ名前のデバイスシャドウドキュメントを作成した場合、新しいデバイスシャドウドキュメントのバージョン番号は、削除されたデバイスのシャドウドキュメントのバージョン番号の続きになります。デバイスシャドウドキュメントが 48 時間より前に削除されている場合、同じ名前の新しいデバイスシャドウドキュメントのバージョン番号は 0 になります。

プロトコルサポート

AWS IoT は、MQTT と HTTPS プロトコル経由の REST API をサポートして shadows とやり取りします。 は、MQTT 発行およびサブスクライブアクション用に予約されたリクエストとレスポンスのトピックのセット AWS IoT を提供します。デバイスとアプリは、 がリクエスト AWS IoT を処理した方法に関する情報についてリクエストトピックに発行する前に、レスポンストピックをサブスクライブする必要があります。詳細については、「Device Shadow MQTT トピック」および「Device Shadow の REST API」を参照してください。

状態の要求と報告

AWS IoT および シャドウを使用して IoT ソリューションを設計する場合は、変更をリクエストするアプリケーションまたはデバイスと、変更を実装するアプリケーションまたはデバイスを決定する必要があります。通常、デバイスは変更をシャドウに実装して報告し、アプリとサービスはシャドウの変更に応答して要求します。ソリューションは異なる場合がありますが、このトピックの例では、クライアントアプリまたはサービスがシャドウの変更を要求し、デバイスがその変更を実行してシャドウに報告することを前提としています。

シャドウの更新

アプリまたはサービスは、UpdateThingShadow API を使用するか、/update トピックに発行することで、シャドウの状態を更新できます。更新は、リクエストで指定したフィールドにのみ反映されます。

クライアントが状態の変更を要求したときのシャドウの更新

クライアントが MQTT プロトコルを使用してシャドウの状態の変更を要求した場合

1. クライアントには、変更するプロパティを識別できるように、現在のシャドウドキュメントが必要です。現在のシャドウドキュメントを取得する方法については、/get アクションを参照してください。
2. クライアントは、次の MQTT トピックにサブスクライブします。
   - $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. クライアントは、シャドウの desired 状態を含む状態ドキュメントを持つ $aws/things/*thingName*/shadow/name/*shadowName*/update リクエストトピックを発行します。変更するプロパティのみをドキュメントに含める必要があります。これは、desired 状態のドキュメントの例です。

   ```
   {
     "state": {
       "desired": {
         "color": {
           "r": 10
         },
         "engine": "ON"
       }
     }
   }
   ```

4. 更新リクエストが有効な場合、 はシャドウ内の目的の状態 AWS IoT を更新し、これらのトピックに関するメッセージを発行します。
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted

- $aws/things/*thingName*/shadow/name/*shadowName*/update/delta

/update/accepted メッセージには /accepted レスポンス状態ドキュメント シャドウドキュメントが含まれ、/update/delta メッセージには /delta レスポンス状態ドキュメント シャドウドキュメントが含まれます。

5. 更新リクエストが有効でない場合、 はエラーを説明するエラーレスポンスドキュメントシャドウドキュメントを含む $aws/things/*thingName*/shadow/name/*shadowName*/update/rejectedトピックを含むメッセージ AWS IoT を発行します。

クライアントが API を使用してシャドウの状態の変更を要求した場合

1. クライアントは、UpdateThingShadow API と リクエスト状態ドキュメント 状態ドキュメントをメッセージ本文として使用します。
2. リクエストが有効な場合、 は HTTP 成功レスポンスコードと/accepted レスポンス状態ドキュメントシャドウドキュメントをレスポンスメッセージ本文として AWS IoT 返します。

   AWS IoT は、それをサブスクライブするデバイスまたはクライアントの/delta レスポンス状態ドキュメントシャドウドキュメントを含む MQTT メッセージも$aws/things/*thingName*/shadow/name/*shadowName*/update/deltaトピックに発行します。
3. リクエストが有効でない場合、 はレスポンスメッセージ本文エラーレスポンスドキュメントとして HTTP エラーレスポンスコード AWS IoT を返します。

デバイスが /desired トピックに関する /update/delta 状態を受信すると、デバイス内で必要な変更を行います。次に、/update トピックにメッセージが送信され、現在の状態がシャドウに報告されます。

デバイスが現在の状態を報告したときにシャドウを更新する

デバイスが MQTT プロトコルを使用して現在の状態をシャドウに報告する場合

1. デバイスは、シャドウを更新する前に、次の MQTT トピックにサブスクライブする必要があります。

   - $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. デバイスは、この例のように、現在の状態を報告する $aws/things/*thingName*/shadow/name/*shadowName*/update トピックにメッセージを発行することによって、現在の状態を報告します。

```
{
    "state": {
        "reported" : {
            "color" : { "r" : 10 },
            "engine" : "ON"
        }
    }
}
```

3. が更新 AWS IoT を受け入れると、シャ/accepted レスポンス状態ドキュメントドウドキュメントを使用して$aws/things/*thingName*/shadow/name/*shadowName*/update/acceptedトピックにメッセージを発行します。

4. 更新リクエストが有効でない場合、 はエラーを説明するエラーレスポンスドキュメントシャドウドキュメントを含む $aws/things/*thingName*/shadow/name/*shadowName*/update/rejectedトピックを含むメッセージ AWS IoT を発行します。

デバイスが API を使用して現在の状態をシャドウに報告する場合

1. デバイスは、リクエスト状態ドキュメント 状態ドキュメントをメッセージ本文として使用して UpdateThingShadow API を呼び出します。

2. リクエストが有効な場合、 はシャドウ AWS IoT を更新し、レスポンスメッセージ本文としてシャドウドキュメントを含む HTTP /accepted レスポンス状態ドキュメント 成功レスポンスコードを返します。

   AWS IoT は、それをサブスクライブするデバイスまたはクライアントの/delta レスポンス状態ドキュメントシャドウドキュメントを含む MQTT メッセージも$aws/things/*thingName*/shadow/name/*shadowName*/update/deltaトピックに発行します。

3. リクエストが有効でない場合、 はレスポンスメッセージ本文エラーレスポンスドキュメントとして HTTP エラーレスポンスコード AWS IoT を返します。

オプティミスティックロック

状態ドキュメントのバージョンを使用して、デバイスのシャドウドキュメントの最新バージョンを更新していることを確認できます。更新リクエストでバージョンを渡したとき、そのバージョンと状態ドキュメントの現在のバージョンとが一致しない場合、サービスは HTTP 409 conflict レスポンスコードでリクエストを拒否します。競合レスポンスコードは、ThingShadow を変更するすべての API (DeleteThingShadow を含む) でも発生する可能性があります。

例えば、次のようになります。

初期ドキュメント:

```
{
  "state": {
    "desired": {
      "colors": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

更新: (バージョンが一致しないと、リクエストは拒否される)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 9
}
```

結果:

```
{
  "code": 409,
```

```
  "message": "Version conflict",
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

更新: (バージョンが一致すると、リクエストは受け入れられる)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

最終状態:

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 11
}
```

シャドウキュメントの取得

シャドウドキュメントは、GetThingShadow API を使用するか、/get トピックをサブスクライブして発行することによって取得できます。これにより、desired 状態と reported 状態の間の delta を含む、完全なシャドウドキュメントが取得されます。このタスクの手順は、デバイスまたはクライアントがリクエストを行っているかどうかにかかわらず同じです。

MQTT プロトコルを使用してシャドウドキュメントを取得するには

1. デバイスまたはクライアントは、シャドウを更新する前に、次の MQTT トピックにサブスクライブする必要があります。

- $aws/things/*thingName*/shadow/name/*shadowName*/get/accepted
- $aws/things/*thingName*/shadow/name/*shadowName*/get/rejected

2. デバイスまたはクライアントは、空のメッセージ本文を持つ $aws/things/*thingName*/shadow/name/*shadowName*/get トピックにメッセージを発行します。
3. リクエストが成功すると、 はメッセージ本文/accepted レスポンス状態ドキュメントに を含むメッセージを$aws/things/*thingName*/shadow/name/*shadowName*/get/acceptedトピックに AWS IoT 発行します。
4. リクエストが有効でない場合、 はメッセージ本文に を含むメッセージを$aws/things/*thingName*/shadow/name/*shadowName*/get/rejectedトピックエラーレスポンスドキュメントに AWS IoT 発行します。

REST API を使用してシャドウドキュメントを取得するには

1. デバイスまたはクライアントは、空のメッセージ本文で GetThingShadow API を呼び出します。
2. リクエストが有効な場合、 はレスポンスメッセージ本文としてシャドウドキュメントを含む HTTP /accepted レスポンス状態ドキュメント 成功レスポンスコード AWS IoT を返します。
3. リクエストが有効でない場合、 はレスポンスメッセージ本文エラーレスポンスドキュメントとして HTTP エラーレスポンスコード AWS IoT を返します。

シャドウデータの削除

シャドウデータを削除するには、シャドウドキュメント内の特定のプロパティを削除する方法と、シャドウを完全に削除する方法の 2 つがあります。

- シャドウから特定のプロパティを削除するには、シャドウを更新します。ただし、削除するプロパティの値を null に設定します。値が null のフィールドは、シャドウドキュメントから削除されます。
- シャドウ全体を削除するには、DeleteThingShadow API を使用するか、/delete トピックに発行します。

> **Note**
>
> シャドウを削除しても、すぐにバージョン番号がゼロにリセットされるわけではありません。48 時間後にゼロにリセットされます。

シャドウドキュメントからのプロパティの削除

MQTT プロトコルを使用してシャドウからプロパティを削除するには

1. デバイスまたはクライアントには、変更するプロパティを識別できるように、現在のシャドウドキュメントが必要です。現在のシャドウドキュメントを取得する方法については、「シャドウドキュメントの取得」を参照してください。
2. デバイスまたはクライアントは、次の MQTT トピックにサブスクライブします。
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
3. デバイスまたはクライアントは、削除するシャドウのプロパティに `$aws/things/thingName/shadow/name/shadowName/update` 値を割り当てる状態ドキュメントを含む null リクエストトピックを発行します。変更するプロパティのみをドキュメントに含める必要があります。これは、engine プロパティを削除するドキュメントの例です。

   ```
   {
     "state": {
       "desired": {
         "engine": null
       }
     }
   }
   ```
4. 更新リクエストが有効な場合、 はシャドウ内の指定されたプロパティ AWS IoT を削除し、メッセージ本文にシャド/accepted レスポンス状態ドキュメントウドキュメントを含む `$aws/things/thingName/shadow/name/shadowName/update/accepted`トピックを含むメッセージを発行します。
5. 更新リクエストが有効でない場合、 はエラーを説明するエラーレスポンスドキュメントシャドウドキュメントを含む `$aws/things/thingName/shadow/name/shadowName/update/rejected`トピックを含むメッセージ AWS IoT を発行します。

REST API を使用してシャドウからプロパティを削除するには

1. デバイスまたはクライアントは、削除するシャドウのプロパティに null 値を割り当てる リクエスト状態ドキュメント を使用して UpdateThingShadow API を呼び出します。削除するプロパティのみをドキュメントに含めます。これは、engine プロパティを削除するドキュメントの例です。

```
{
  "state": {
    "desired": {
      "engine": null
    }
  }
}
```

2. リクエストが有効な場合、 は HTTP 成功レスポンスコードと/accepted レスポンス状態ドキュメントシャドウドキュメントをレスポンスメッセージ本文として AWS IoT 返します。
3. リクエストが有効でない場合、 はレスポンスメッセージ本文エラーレスポンスドキュメントとして HTTP エラーレスポンスコード AWS IoT を返します。

シャドウの削除

デバイスのシャドウを削除に関する考慮事項を次に示します。

- デバイスのシャドウ状態を null に設定しても、シャドウは削除されません。シャドウのバージョンは、次の更新時に増分されます。
- デバイスのシャドウを削除しても、Thing オブジェクトは削除されません。Thing オブジェクトを削除しても、対応するデバイスのシャドウは削除されません。
- シャドウを削除しても、すぐにバージョン番号がゼロにリセットされるわけではありません。48 時間後にゼロにリセットされます。

MQTT プロトコルを使用してシャドウを削除するには

1. デバイスまたはクライアントは、次の MQTT トピックにサブスクライブします。
   - $aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected

2. デバイスまたはクライアントは、空のメッセージバッファを持つ $aws/things/*thingName*/shadow/name/*shadowName*/delete を発行します。
3. 削除リクエストが有効な場合、 はシャドウ AWS IoT を削除し、メッセージ本文に $aws/things/*thingName*/shadow/name/*shadowName*/delete/acceptedトピックと省略されたシャド/accepted レスポンス状態ドキュメントウドキュメントを含むメッセージを発行します。次に、受け入れられた削除メッセージの例を示します。

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

4. 更新リクエストが有効でない場合、 はエラーを説明するエラーレスポンスドキュメントシャドウドキュメントを含む $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejectedトピックを含むメッセージ AWS IoT を発行します。

REST API を使用してシャドウを削除するには

1. デバイスまたはクライアントは、空のメッセージバッファを使用して DeleteThingShadow API を呼び出します。
2. リクエストが有効な場合、 はメッセージ本文に HTTP 成功レスポンスコードと /accepted レスポンス状態ドキュメント /accepted レスポンス状態ドキュメントと省略されたシャドウドキュメント AWS IoT を返します。次に、受け入れられた削除メッセージの例を示します。

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

3. リクエストが有効でない場合、 はレスポンスメッセージ本文エラーレスポンスドキュメントとして HTTP エラーレスポンスコード AWS IoT を返します。

Device Shadow の REST API

Shadow は状態情報の更新用に以下の URI を公開します。

```
https://account-specific-prefix-ats.iot.region.amazonaws.com/things/thingName/shadow
```

エンドポイントは に固有です AWS アカウント。エンドポイントを見つけるには、以下を実行します。

- AWS CLIの describe-endpoint コマンドを使用します。
- AWS IoT コンソール設定を使用します。[Settings] (設定) で、エンドポイントは [Custom endpoint] (カスタムエンドポイント) の下に表示されます
- AWS IoT コンソールのモノの詳細ページを使用します。コンソールで:
  1. [Manage] (管理) を展開し、[Manage] (管理) で [Things] (モノ) をクリックします。
  2. モノのリストで、エンドポイント URI を取得するモノを選択します。
  3. [Device Shadows] (デバイスシャドウ) タブをクリックし、シャドウを選択します。エンドポイント URI は、[Device Shadow details] (デバイスシャドウの詳細) ページの [Device Shadow URL] (デバイスシャドウの URL) セクションで確認できます。

エンドポイントの形式は以下のとおりです。

```
identifier.iot.region.amazonaws.com
```

シャドウの REST API は、「デバイス通信プロトコル」で説明されているものと同じ HTTPS プロトコル/ポートマッピングに従います。

> **Note**
>
> API を使用するには、`iotdevicegateway` を認証のサービス名として使用する必要があります。詳細については、「IOTDataPlane」を参照してください。

API アクション

- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

また、API のクエリパラメータの一部として name=*shadowName* を指定し、API を使用して名前付きシャドウを作成することもできます。

GetThingShadow

指定したモノの Shadow を取得します。

レスポンス状態ドキュメントには、desired 状態と reported 状態との差分が含まれます。

リクエスト

リクエストには標準の HTTP ヘッダーに加えて、以下の URI が含まれます。

```
HTTP GET https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

名前なし (クラシック) シャドウでは、name クエリパラメータは必要ありません。

応答

成功した場合、レスポンスには標準の HTTP ヘッダーに加えて、以下のコードと本体が含まれます。

```
HTTP 200
Response Body: response state document
```

詳細については、「Example Response State Document」を参照してください。

Authorization

Shadow を取得するには、呼び出し元に iot:GetThingShadow アクションの実行を許可するポリシーが必要です。Device Shadow サービスは、IAM 認証情報による署名バージョン 4、クライアント証明書による TLS 相互認証という、2 つの形式の認証を受け入れます。

以下に示しているのは、呼び出し元にデバイスのシャドウの取得を許可するポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:GetThingShadow",
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

UpdateThingShadow

指定したモノの Shadow を更新します。

更新は、リクエスト状態ドキュメントで指定したフィールドにのみ反映されます。値が null のフィールドはすべてデバイスのシャドウから削除されます。

リクエスト

リクエストには標準の HTTP ヘッダーに加えて、以下の URI と本体が含まれます。

```
HTTP POST https://endpoint/things/thingName/shadow?name=shadowName
Request body: request state document
```

名前なし (クラシック) シャドウでは、name クエリパラメータは必要ありません。

詳細については、「リクエスト状態ドキュメントの例」を参照してください。

応答

成功した場合、レスポンスには標準の HTTP ヘッダーに加えて、以下のコードと本体が含まれます。

```
HTTP 200
Response body: response state document
```

詳細については、「Example Response State Document」を参照してください。

Authorization

Shadow を更新するには、呼び出し元に iot:UpdateThingShadow アクションの実行を許可するポリシーが必要です。Device Shadow サービスは、IAM 認証情報による署名バージョン 4、クライアント証明書による TLS 相互認証という、2 つの形式の認証を受け入れます。

以下に示しているのは、呼び出し元にデバイスのシャドウの更新を許可するポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:UpdateThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

DeleteThingShadow

指定したモノの Shadow を削除します。

リクエスト

リクエストには標準の HTTP ヘッダーに加えて、以下の URI が含まれます。

```
HTTP DELETE https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

名前なし (クラシック) シャドウでは、name クエリパラメータは必要ありません。

応答

成功した場合、レスポンスには標準の HTTP ヘッダーに加えて、以下のコードと本体が含まれます。

```
HTTP 200
Response body: Empty response state document
```

シャドウを削除しても、バージョン番号は 0 にリセットされないことに注意してください。

Authorization

デバイスのシャドウを削除するには、呼び出し元に iot:DeleteThingShadow アクションの実行を許可するポリシーが必要です。Device Shadow サービスは、IAM 認証情報による署名バージョン 4、クライアント証明書による TLS 相互認証という、2 つの形式の認証を受け入れます。

以下に示しているのは、呼び出し元にデバイスのシャドウの削除を許可するポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

ListNamedShadowsForThing

指定されたモノのシャドウを一覧表示します。

リクエスト

リクエストには標準の HTTP ヘッダーに加えて、以下の URI が含まれます。

```
HTTP GET /api/things/shadow/ListNamedShadowsForThing/thingName?
nextToken=nextToken&pageSize=pageSize
Request body: (none)
```

nextToken

次の結果セットを取得するためのトークン。

この値は、ページングされた結果で返され、次のページを返す呼び出しで使用されます。

pageSize

各呼び出しで返すシャドウ名の数。「nextToken」も参照してください。

thingName

名前の付いたシャドウを一覧表示するモノの名前。

応答

成功した場合、レスポンスには標準の HTTP ヘッダーに加えて、以下のレスポンスコードと シャドウ名リストレスポンスドキュメント が含まれます。

> **Note**
>
> 名前なし (クラシック) シャドウは、このリストに表示されません。クラシックシャドウしかない、または指定したthingName が存在しない場合、レスポンスは空のリストになります。

```
HTTP 200
Response body: Shadow name list document
```

Authorization

デバイスのシャドウをリスト化するには、呼び出し元に iot:ListNamedShadowsForThing アクションの実行を許可するポリシーが必要です。Device Shadow サービスは、IAM 認証情報による署名バージョン 4、クライアント証明書による TLS 相互認証という、2 つの形式の認証を受け入れます。

以下に示しているのは、呼び出し元にモノの名前付きシャドウの削除を許可するポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:ListNamedShadowsForThing",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
```

```
}
```

Device Shadow MQTT トピック

Device Shadow サービスでは、予約された MQTT トピックを使用することで、アプリケーションやデバイスがデバイスの状態情報 (シャドウ) を取得、更新、削除できるようにします。

Shadow トピックへのパブリッシュとサブスクライブにはトピックベースの権限付与が必要です。AWS IoT には、既存のトピック構造に新しいトピックを追加する権限があります。この理由から、Shadow トピックへのサブスクリプションにワイルドカードを使用しないことをお勧めします。例えば、 が新しいシャドウトピック AWS IoT を導入`$aws/things/thingName/shadow/#`すると、このトピックフィルターに一致するトピックの数が増える可能性があるため、 のようなトピックフィルターへのサブスクライブは避けてください。これらのトピックでパブリッシュされたメッセージの例については、「シャドウとの相互作用」を参照してください。

シャドウは、名前付き、または名前のないもの (クラシック) にすることができます。それぞれで使用されるトピックは、トピックのプレフィックスでのみ異なります。この表は、各シャドウタイプで使用されるトピックのプレフィックスを示しています。

ShadowTopicPrefix 値	シャドウタイプ
`$aws/things/` *thingName* `/shadow`	名前のない (クラシック) シャドウ
`$aws/things/` *thingName* `/shadow/name/` *shadowName*	名前付きシャドウ

完全なトピックを作成するには、次のセクションに示すように、参照するシャドウのタイプの *ShadowTopicPrefix* を選択し、*thingName* と、*shadowName* (該当する場合) を対応する値に置き換え、トピックスタブにそれを追加します。

以下に示しているのは、Shadow とのやり取りに使用される MQTT トピックです。

トピック

- /get
- /get/accepted
- /get/rejected

- /update
- /update/delta
- /update/accepted
- /update/documents
- /update/rejected
- /delete
- /delete/accepted
- /delete/rejected

/get

デバイスのシャドウを取得するには、このトピックに空のメッセージをパブリッシュします。

```
ShadowTopicPrefix/get
```

AWS IoT は、 /get/acceptedまたは のいずれかに発行することで応答します/get/rejected。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 は、デバイスのシャドウを返すときに、このトピックにレスポンスシャドウドキュメントを発行します。

```
ShadowTopicPrefix/get/accepted
```

詳細については、「レスポンス状態ドキュメント」を参照してください。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 は、デバイスのシャドウを返せない場合に、このトピックにエラーレスポンスドキュメントを発行します。

```
ShadowTopicPrefix/get/rejected
```

詳細については、「エラーレスポンスドキュメント」を参照してください。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
  "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

このトピックにリクエスト状態ドキュメントをパブリッシュして、デバイスのシャドウを更新します。

```
ShadowTopicPrefix/update
```

メッセージ本文には、部分的なリクエスト状態ドキュメントが含まれています。

デバイスの状態を更新しようとするクライアントは、次のような desired プロパティを持つ JSON リクエスト状態ドキュメントを送信します。

```
{
  "state": {
    "desired": {
      "color": "red",
      "power": "on"
    }
  }
}
```

シャドウを更新するデバイスは、次のような reported プロパティを持つ JSON リクエスト状態ドキュメントを送信します。

```
{
  "state": {
    "reported": {
      "color": "red",
      "power": "on"
    }
  }
}
```

AWS IoT は、 /update/acceptedまたは のいずれかに発行することで応答します/update/rejected。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
```

```
    "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 は、デバイスのシャドウの変更を受け入れると、レスポンス状態ドキュメントをこのトピックに発行します。レスポンス状態ドキュメントには、 desired と reportedの状態の異なる値が含まれています。

```
ShadowTopicPrefix/update/delta
```

メッセージバッファには /delta レスポンス状態ドキュメント が含まれています

メッセージ本文の詳細

- update/delta にパブリッシュされたメッセージには、desired セクションと reported セクションとで異なる属性のみが含まれます。それら属性が、現在の更新メッセージに含まれていたか、 AWS IoTにすでに保存されていたかには関係ありません。desired セクションと reported セクションとで同じ属性は含まれません。
- 属性が reported セクションにあるが、desired セクションに同じ属性がない場合、その属性は含まれません。
- 属性が desired セクションにあるが、reported セクションに同じ属性がない場合、その属性は含まれません。
- 属性が reported セクションから削除されたが、desired セクションにまだある場合、その属性は含まれます。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 は、デバイスのシャドウの変更を受け入れると、このトピックに応答状態ドキュメントを発行します。

```
ShadowTopicPrefix/update/accepted
```

メッセージバッファには /accepted レスポンス状態ドキュメント が含まれています

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 シャドウの更新が正常に実行されるたびに、 はこのトピックに状態ドキュメントを発行します。

```
ShadowTopicPrefix/update/documents
```

メッセージ本文には、 が含まれています/documents レスポンス状態ドキュメント

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 は、デバイスのシャドウの変更を拒否すると、このトピックにエラーレスポンスドキュメントを発行します。

```
ShadowTopicPrefix/update/rejected
```

メッセージ本文には、エラーレスポンスドキュメント が含まれています。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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

デバイスのシャドウを削除するには、削除トピックに空のメッセージをパブリッシュします。

```
ShadowTopicPrefix/delete
```

メッセージの内容は無視されます。

シャドウを削除しても、バージョン番号は 0 にリセットされないことに注意してください。

AWS IoT は、 /delete/acceptedまたは のいずれかに発行することで応答します/delete/rejected。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 は、デバイスのシャドウが削除されると、このトピックにメッセージを発行します。

```
ShadowTopicPrefix/delete/accepted
```

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 は、デバイスのシャドウを削除できない場合に、このトピックにエラーレスポンスドキュメントを発行します。

```
ShadowTopicPrefix/delete/rejected
```

メッセージ本文には、エラーレスポンスドキュメント が含まれています。

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

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

Device Shadow サービスドキュメント

Device Shadow サービスは JSON 仕様のすべてのルールに準拠します。値、オブジェクト、配列はデバイスのシャドウドキュメントに保存されます。

目次

- シャドウドキュメントの例
- ドキュメントのプロパティ
- delta 状態
- シャドウドキュメントのバージョニング
- シャドウドキュメント内のクライアントトークン
- 空のシャドウドキュメントのプロパティ
- シャドウドキュメント内の配列値

シャドウドキュメントの例

Device Shadow サービスは、REST API または MQTT パブリッシュ/サブスクライブメッセージを使用する UPDATE、GET、DELETE オペレーションで、以下のドキュメントを使用します。

例

- リクエスト状態ドキュメント
- レスポンス状態ドキュメント

- エラーレスポンスドキュメント
- シャドウ名リストレスポンスドキュメント

リクエスト状態ドキュメント

リクエスト状態ドキュメントの形式は次のとおりです。

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "clientToken": "token",
    "version": version
}
```

- `state` — 更新は、指定したフィールドにのみ反映されます。通常、同じリクエストで `desired` または `reported` プロパティのいずれかを使用しますが、両方を使用することはありません。
  - `desired` — デバイスで更新がリクエストされた state のプロパティと値。
  - `reported` — デバイスによってレポートされた state のプロパティと値。
- `clientToken` — 使用する場合は、クライアントトークンによってリクエストとレスポンスを対応付けることができます。
- `version` — 使用した場合、指定したバージョンが最新バージョンと一致すると、Device Shadow サービスは更新を処理します。

レスポンス状態ドキュメント

レスポンス状態ドキュメントはレスポンスタイプに応じて以下の形式になります。

/accepted レスポンス状態ドキュメント

```
{
    "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 レスポンス状態ドキュメント

```
{
    "state": {
        "attribute1": integer2,
        "attribute2": "string2",
        ...
        "attributeN": boolean2
    },
    "metadata": {
        "attribute1": {
            "timestamp": timestamp
        },
```

```
        "attribute2": {
            "timestamp": timestamp
        },
        ...
        "attributeN": {
            "timestamp": timestamp
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/documents レスポンス状態ドキュメント

```
{
  "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"
}
```

レスポンス状態ドキュメントのプロパティ

- previous — 更新が成功した後、更新前のオブジェクトの state が含まれます。
- current — 更新が成功した後、更新後のオブジェクトの state が含まれます。
- state
  - reported — モノが reported セクション内の任意のデータを報告し、リクエスト状態ドキュメントにあったフィールドだけを含む場合にのみ存在します。
  - desired — デバイスが desired セクション内の任意のデータを報告し、リクエスト状態ドキュメントにあったフィールドだけを含む場合にのみ存在します。
- metadata — いつ状態が更新されたかを決定できるように、desired および reported セクションの属性ごとにタイムスタンプが含まれます。
- timestamp — レスポンスが生成されたエポック日時 AWS IoT。
- clientToken — /update トピックに有効な JSON を発行するときにクライアントトークンが使用された場合にのみ存在します。
- version — AWS IoTで共有されるデバイスのシャドウのドキュメントの現在のバージョン。ドキュメントの前バージョンから 1 ずつインクリメントされます。

エラーレスポンスドキュメント

エラーレスポンスドキュメントの形式は次のとおりです。

```
{
    "code": error-code,
    "message": "error-message",
    "timestamp": timestamp,
    "clientToken": "token"
}
```

- code — エラーのタイプを示す HTTP レスポンスコード。
- message — 追加の情報を提供するテキストメッセージ。
- timestamp — レスポンスが生成された日時 AWS IoT。このプロパティはすべてのエラーレスポンスドキュメントに存在するわけではありません。
- clientToken — 発行されたメッセージでクライアントトークンが使用された場合にのみ表示されます。

詳細については、「Device Shadow エラーメッセージ」を参照してください。

シャドウ名リストレスポンスドキュメント

シャドウ名リストレスポンスドキュメントの形式は次のとおりです。

```
{
    "results": [
        "shadowName-1",
        "shadowName-2",
        "shadowName-3",
        "shadowName-n"
    ],
    "nextToken": "nextToken",
    "timestamp": timestamp
}
```

- results — シャドウ名の配列。
- nextToken — シーケンス内の次のページを取得するためにページングされたリクエストで使用するトークン値。返すシャドウ名がなくなると、このプロパティは存在しません。
- timestamp — レスポンスが生成された日時 AWS IoT。

ドキュメントのプロパティ

デバイスのシャドウドキュメントには、以下のプロパティがあります。

`state`

`desired`

デバイスの desired 状態。アプリケーションは、ドキュメントのこの部分に書き込んで、デバイスの状態を直接更新できます。そのために、デバイスに接続する必要はありません。

`reported`

デバイスの reported 状態。デバイスはドキュメントのこの部分に書き込んで、デバイスの新しい状態を報告します。アプリは、ドキュメントのこの部分を読み取り、デバイスの last-reported 状態を判断します。

`metadata`

ドキュメントの `state` セクションに保存されたデータに関する情報。この情報には `state` セクション内の属性ごとにタイムスタンプ (エポック時間) が含まれており、いつそれらの属性が更新されたかを決定できます。

> **Note**
>
> メタデータは、サービスの制限または料金に関連するドキュメントサイズに影響を与えません。詳細については、「AWS IoT サービス制限」を参照してください。

`timestamp`

メッセージがいつ送信されたかを示します AWS IoT。メッセージ内のタイムスタンプと、`desired` または `reported` セクションの個々の属性のタイムスタンプを使用すると、デバイスに内部クロックがない場合でも、デバイスはプロパティの経過時間を判断できます。

`clientToken`

デバイスに一意の文字列。MQTT 環境でレスポンスをリクエストに関連付けるために使用されます。

`version`

ドキュメントのバージョン。ドキュメントが更新されるたびに、このバージョン番号がインクリメントされます。更新されるドキュメントのバージョンが最新になるように使用されます。

詳細については、「シャドウドキュメントの例」を参照してください。

delta 状態

delta 状態は、desired 状態と reported 状態の違いを含む virtual 型の状態です。desired セクションに含まれるが、reported セクションに含まれないフィールドは、差分に含まれます。reported セクションに含まれるが、desired セクションに含まれないフィールドは、差分に含まれません。差分にはメタデータが含まれ、その値は desired フィールドのメタデータに一致します。以下に例を示します。

```
{
  "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
  }
}
```

ネストされたオブジェクトが異なると、差分にはルートへのパスが含まれます。

```
{
  "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
}
```

Device Shadow サービスは、desired 状態の各フィールドを反復処理し、reported 状態と比較することで、差分を計算します。

配列は値のように扱われます。desired セクションの配列が reported セクションの配列に一致しない場合は、desired 配列全体が差分にコピーされます。

シャドウドキュメントのバージョニング

Device Shadow サービスは、リクエストとレスポンスの両方の更新メッセージごとにバージョニングをサポートします。つまり、シャドウが更新されるたびに、JSON ドキュメントのバージョンが増分されます。これにより、以下の 2 つのことが確実になります。

- クライアントは、古いバージョン番号を使用して Shadow を上書きしようとした場合、エラーを受け取ることができます。デバイスのシャドウを更新するには、再同期する必要があることがクライアントに通知されます。
- クライアントは、受信したメッセージ内のバージョンが自らで保存しているバージョンより古い場合、そのメッセージに基づいてアクションを実行しないことを決定できます。

クライアントは、シャドウドキュメントにバージョンを含めないことで、バージョン一致を回避できます。

シャドウドキュメント内のクライアントトークン

クライアントトークンを MQTT ベースのメッセージングで使用して、リクエストとそのレスポンスに同じクライアントトークンが含まれていることを確認できます。これにより確実にレスポンスとリクエストが関連付けられます。

> **Note**
>
> クライアントトークンは 64 バイトを超えない範囲にします。64 バイトを超えるクライアントトークンは、400 (不適切なリクエスト) レスポンスとInvalid clientToken エラーメッセージの原因となります。

空のシャドウドキュメントのプロパティ

シャドウドキュメントの reported プロパティと desired プロパティは、現在のシャドウ状態に適用されない場合は空にすることも、省略することもできます。たとえば、シャドウドキュメントには、desired 状態がある場合にのみ、desired プロパティが含まれます。次に、desired プロパティのない状態ドキュメントの有効な例を示します。

```
{
    "reported" : { "temp": 55 }
}
```

デバイスによってシャドウが更新されていない場合など、reported プロパティは空にすることもできます。

```
{
    "desired" : { "color" : "RED" }
}
```

更新によって desired または reported プロパティが null になると、ドキュメントから削除されます。次に、desired プロパティを null に設定して削除する方法を示します。たとえば、デバイスの状態を更新するときにこれを行うことができます。

```
{
    "state": {
        "reported": {
            "color": "red"
        },
        "desired": null
    }
}
```

シャドウドキュメントには、desired プロパティと reported プロパティのいずれも持たないため、シャドウドキュメントは空になります。これは、空で有効なシャドウドキュメントの例です。

```
{
}
```

シャドウドキュメント内の配列値

シャドウでは、配列がサポートされていますが、配列に対する更新により配列全体が置き換わる点では、通常の値として扱われます。配列の一部を更新することはできません。

初期状態:

```
{
    "desired" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

更新:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

最終状態:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

配列に null 値を含めることはできません。たとえば、以下の配列は有効ではなく、拒否されます。

```
{
    "desired" : {
        "colors" : [ null, "RED", "GREEN" ]
    }
}
```

Device Shadow エラーメッセージ

Device Shadow サービスは、状態ドキュメントの変更の試みが失敗したときに、MQTT を介して error トピックにメッセージをパブリッシュします。このメッセージは、いずれかの予約された $aws トピックへの発行リクエストに対するレスポンスとしてのみ発行されます。クライアントが REST API を使用してドキュメントを更新する場合は、そのレスポンスの一部として HTTP エラーコードを受け取り、MQTT エラーメッセージは発行されません。

HTTP エラーコード	エラーメッセージ
400 (Bad Request)	• JSON が無効です • Missing required node: state • State node must be an object • Desired node must be an object • Reported node must be an object • Invalid version • Invalid clientToken **Note** 64 バイトを超えるクライアントトークンは、このメッセージの原因となります。 • JSON contains too many levels of nesting; maximum is 6 • State contains an invalid node
401 (Unauthorized)	• Unauthorized
403 (Forbidden)	• Forbidden
404 (Not Found)	• Thing not found • 次の名前のシャドウは存在しません: *shadowName*
409 (Conflict)	• Version conflict
413 (Payload Too Large)	• The payload exceeds the maximum size allowed
415 (Unsupported Media Type)	• Unsupported documented encoding; supported encoding is UTF-8
429 (Too Many Requests)	• 単一の接続に 10 個を超える処理中のリクエストがある場合、Device Shadow サービスはこのエラーメッセージを生成します。実行中のリクエストとは、開始され

HTTP エラーコード	エラーメッセージ
	たがまだ完了していない進行中のリクエストのことです。
500 (Internal Server Error)	• Internal service failure

AWS IoT Device Management ソフトウェアパッケージカタログ

AWS IoT Device Management Software Package Catalog を使用すると、ソフトウェアパッケージとそのバージョンのインベントリを維持できます。パッケージバージョンを個々のモノと AWS IoT モノの動的グループに関連付け、社内プロセスまたはAWS IoT ジョブを通じてデプロイできます。

ソフトウェアパッケージには、1 つ以上のパッケージバージョンが含まれます。パッケージバージョンは、1 つのユニットとしてデプロイできるファイルの集まりです。パッケージバージョンには、ファームウェア、オペレーティングシステムの更新、デバイスアプリケーション、設定、およびセキュリティパッチを含めることができます。ソフトウェアが時間の経過とともに進化するにつれて、新しいパッケージバージョンを作成してフリートにデプロイできます。

AWS IoT ソフトウェアパッケージハブは 内にあります AWS IoT Core。ハブを使用して、ソフトウェアパッケージのインベントリとメタデータを一元的に登録および保守できます。これにより、ソフトウェアパッケージとそのバージョンのカタログが作成されます。デバイスにデプロイされたソフトウェアパッケージとパッケージバージョンに基づいてデバイスをグループ化することを選択できます。この機能により、デバイス側のパッケージインベントリを名前付きシャドウとして保持したり、バージョンに基づいてデバイスを関連付けてグループ化したり、フリートメトリクスを使用してフリート全体のパッケージバージョン分布を視覚化したりできます。

社内でソフトウェアデプロイシステムを確立している場合は、引き続きそのプロセスを使用してパッケージバージョンをデプロイできます。デプロイプロセスがまだ確立されていない場合や、希望する場合は、AWS IoT ジョブを使用して Software Package Catalog の機能を使用することをお勧めします。詳細については、AWS IoT 「ジョブの準備」を参照してください。

この章には、以下のセクションが含まれています。

- Software Package Catalog の使用準備
- セキュリティの準備
- フリートインデックス作成の準備
- AWS IoT ジョブの準備
- Software Package Catalog の開始方法

Software Package Catalog の使用準備

次のセクションでは、パッケージバージョンのライフサイクルの概要と、 AWS IoT Device Management Software Package Catalog を使用するための情報について説明します。

パッケージバージョンライフサイクル

パッケージバージョンは、draft、published、deprecated などのライフサイクルステータスを経て進化する可能性があります。deleted である可能性もあります。



- 下書き

  パッケージバージョンが作成されると、draft 状態になります。この状態は、ソフトウェアパッケージが準備中であるか、不完全であることを示します。

  この状態のパッケージバージョンでは、デプロイできません。パッケージバージョンの説明、属性、およびタグを編集することはできます。

  コンソールを使用するか、UpdatePackageVersion または DeletePackageVersion API オペレーションのいずれかを発行することで、現在 draft の状態のパッケージバージョンを published に移行するか deleted にできます。

- 公開済み

  パッケージバージョンをデプロイする準備ができたら、パッケージバージョンを published 状態に移行します。この状態では、コンソールでソフトウェアパッケージを編集する

か、UpdatePackage API オペレーションを使用して、パッケージバージョンをデフォルトバージョンとして指定できます。この状態では、説明とタグのみを編集できます。

コンソールを使用するか、UpdatePackageVersion または DeletePackageVersion API オペレーションのいずれかを発行することで、現在 published の状態のパッケージバージョンを deprecated に移行するか、deleted にできます。

- 廃止済み

  新しいパッケージバージョンがある場合は、以前のパッケージバージョンを deprecated に移行できます。廃止されたパッケージバージョンを使用してジョブをデプロイすることもできます。廃止されたパッケージバージョンをデフォルトバージョンとして命名し、説明とタグのみを編集することもできます。

  パッケージバージョンが古い場合、またはランタイムの依存関係のためにパッケージバージョンを維持する必要がある場合は、パッケージバージョンを deprecated に移行することを検討してください。

  コンソールを使用するか、UpdatePackageVersion または DeletePackageVersion API オペレーションのいずれかを発行することで、現在 deprecated の状態のパッケージバージョンを published に移行するか、deleted にできます。

- [Deleted] (削除済み)

  パッケージバージョンを使用する意図がなくなった場合は、コンソールまたは DeletePackageVersion API オペレーションを使用して削除できます。

  > **Note**
  >
  > そのパッケージバージョンを参照している保留中のジョブがあるときにパッケージバージョンを削除した場合、ジョブが正常に完了して予約済みの名前付きシャドウを更新しようとしたときにエラーメッセージが表示されます。
  > 削除するソフトウェアパッケージバージョンがデフォルトパッケージバージョンとして指定されている場合は、最初にパッケージを更新して別のバージョンをデフォルトとして指定するか、フィールドを名前なしのままにする必要があります。これを行うには、コンソールまたは UpdatePackageVersion API オペレーションを使用します。(指定されたパッケージバージョンをデフォルトとして削除するには、UpdatePackage API オペレーションを発行するときに unsetDefaultVersion パラメーターを true に設定します)。

デフォルトバージョンとして指定されていない限り、コンソールからソフトウェアパッケージを削除すると、そのパッケージに関連付けられているすべてのパッケージバージョンが削除されます。

パッケージバージョンの命名規則

パッケージバージョンに名前を付けるときは、自分や他の人が最新のパッケージバージョンとバージョンの進行状況を簡単に識別できるように、論理的な命名戦略を計画して適用することが重要です。パッケージバージョンを作成するときはバージョン名を指定する必要がありますが、戦略と形式はビジネスケースによって大きく異なります。

ベストプラクティスとして、セマンティックバージョニング SemVer 形式を使用することをお勧めします。例えば、`1.2.3` の場合、`1` は機能的に互換性のない変更のメジャーバージョン、`2` は機能的に互換性のある変更のメジャーバージョン、`3` は (バグ修正用の) パッチバージョンです。詳細については、「セマンティックバージョニング 2.0.0」を参照してください。パッケージバージョン名の要件の詳細については、 AWS IoT API リファレンスガイドのversionName」を参照してください。

デフォルトバージョン

バージョンをデフォルトとして設定することは任意です。デフォルトのパッケージバージョンを追加または削除できます。デフォルトバージョンとして指定されていないパッケージバージョンをデプロイすることもできます。

パッケージバージョンを作成すると、そのバージョンは `draft` 状態になり、パッケージバージョンを公開済みに移行するまでデフォルトバージョンとして指定することはできません。Software Package Catalog は、デフォルトとしてバージョンを自動的に選択したり、新しいパッケージバージョンをデフォルトとして更新したりしません。コンソールから、または UpdatePackageVersion API オペレーションを発行して、選択したパッケージバージョンに意図的に名前を付ける必要があります。

バージョン属性

バージョン属性とその値には、パッケージバージョンに関する重要な情報が格納されます。パッケージまたはパッケージバージョンの汎用属性を定義することをお勧めします。例えば、プラットフォーム、アーキテクチャ、オペレーティングシステム、リリース日、作成者、または Amazon S3 URL の名前と値のペアを作成できます。

AWS IoT ジョブドキュメントを使用してジョブを作成するときに、属性の値を参照する置換変数 (`$parameter`) を使用することもできます。詳細については、AWS IoT「ジョブの準備」を参照してください。

パッケージバージョンで使用されるバージョン属性は、予約された名前付きシャドウに自動的に追加されず、フリートインデックス作成から直接インデックスを作成またはクエリすることはできません。フリートのインデックス作成を使用してパッケージバージョン属性をインデックス作成またはクエリするには、予約された名前付きシャドウにバージョン属性を入力します。

オペレーションシステムやインストール時間など、予約された名前付きシャドウキャプチャデバイスレポートプロパティのバージョン属性パラメータを使用することをお勧めします。また、フリートのインデックス作成を使用してインデックスを作成し、クエリを実行することもできます。

バージョン属性は、特定の命名規則に従う必要はありません。ビジネスニーズに合わせて名前と値のペアを作成できます。パッケージバージョンのすべての属性の合計サイズは 3 KB に制限されています。詳細については、「Software Package Catalog ソフトウェアパッケージとパッケージバージョンの制限」を参照してください。

ジョブドキュメント内のすべての属性の使用

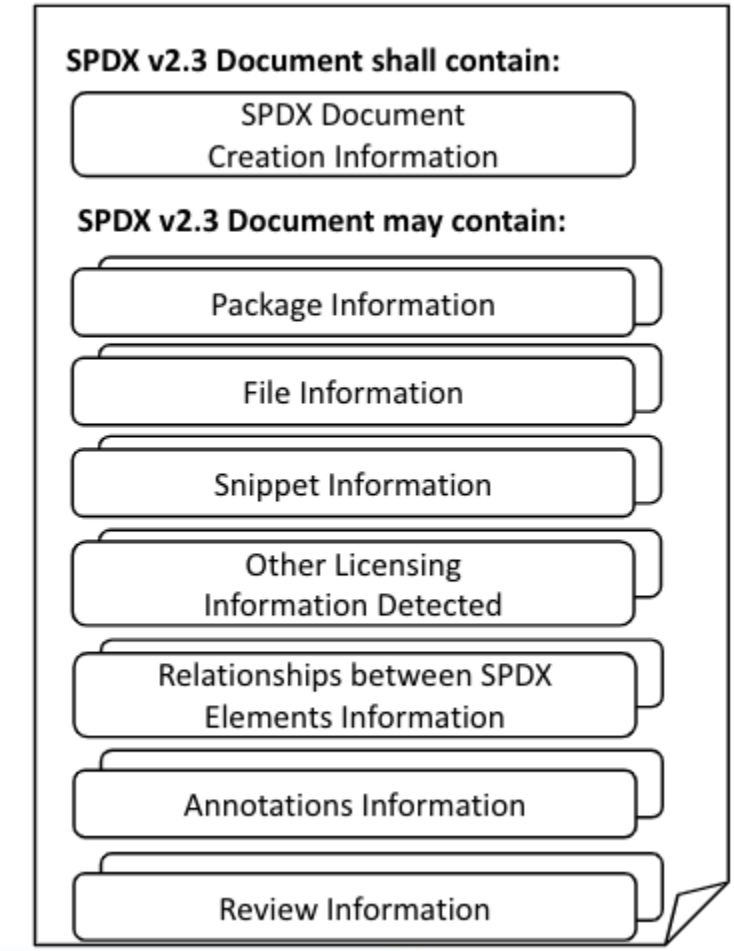
選択したデバイスのジョブデプロイに、すべてのパッケージバージョン属性を自動的に追加できます。API または CLI コマンドでプログラムですべてのパッケージバージョン属性を自動的に使用するには、次のジョブドキュメントの例を参照してください。

```
"TestPackage": "${aws:iot:package:TestPackage:version:PackageVersion:attributes}"
```

ソフトウェア部品表

ソフトウェア部品表 (SBOM) は、ソフトウェアパッケージのあらゆる側面の一元化されたリポジトリを提供します。ソフトウェアパッケージとパッケージバージョンを保存するだけでなく、各パッケージバージョンに関連付けられたソフトウェア部品表 (SBOM) を AWS IoT Device Management Software Package Catalog に保存できます。ソフトウェアパッケージには 1 つ以上のパッケージバージョンが含まれ、各パッケージバージョンは 1 つ以上のコンポーネントで構成されます。特定のパッケージバージョンの構成をサポートする各コンポーネントは、ソフトウェア部品表を使用して説明およびカタログ化できます。サポートされるソフトウェア部品表の業界標準は、SPDX と CycloneDX です。SBOM が最初に作成されると、SPDX および CycloneDX 業界標準形式に対する検証が行われます。SPDX の詳細については、「System Package Data Exchange」を参照してください。CycloneDX の詳細については、「CycloneDX」を参照してください。

ソフトウェア部品表には、パッケージ情報、ファイル情報、その他の関連メタデータなど、特定のパッケージバージョンのコンポーネントのあらゆる側面が記載されています。SPDX 形式のソフトウェア部品表ドキュメント構造の以下の例を参照してください。



ソフトウェア部品表の利点

Software Package Catalog にパッケージバージョンのソフトウェア部品表を追加する主な利点の 1 つは、脆弱性管理です。

脆弱性管理

ソフトウェアコンポーネントの明らかなセキュリティリスクに対する脆弱性を評価して軽減することは、デバイスフリートの整合性を保護する上で依然として重要です。パッケージバージョンごとに Software Package Catalog に保存されているソフトウェア部品表を追加すると、独自の社内脆弱性管理ソリューションを使用して、パッケージバージョンと SBOM に基づいてリスクにさらされているデバイスを把握して、セキュリティ上のギャップを事前に公開できます。影響を受けるデバイスに修正をデプロイし、デバイスのフリートを保護できます。

ソフトウェア部品表のストレージ

各ソフトウェアパッケージバージョンのソフトウェア部品表 (SBOM) は、Amazon S3 バージョニング機能を使用して Amazon S3 バケットに保存されます。SBOM を格納する Amazon S3 バケットは、パッケージバージョンが作成されたのと同じリージョンに配置する必要があります。バージョニング機能を使用する Amazon S3 バケットは、同じバケットにオブジェクトの複数のバリアントを保持します。Amazon S3 バケットでのバージョニングの使用の詳細については、「Amazon S3 バケットでのバージョニングの使用」を参照してください。

> **Note**
>
> 各ソフトウェアパッケージバージョンには複数の SBOM ファイルをアタッチできますが、SBOM ファイルは 1 つの zip アーカイブファイルに保存されている必要があります。

バケットの特定の Amazon S3 キーとバージョン ID は、パッケージバージョンのソフトウェア部品表の各バージョンを一意に識別するために使用されます。

> **Note**
>
> 単一の SBOM ファイルを持つパッケージバージョンの場合、その SBOM ファイルを Amazon S3 バケットに zip アーカイブファイルとして保存できます。
> 複数の SBOM ファイルを含むパッケージバージョンでは、すべての SBOM ファイルを 1 つの zip アーカイブファイルに配置し、その zip アーカイブファイルを Amazon S3 バケットに保存する必要があります。
> 両方のシナリオで単一の zip アーカイブファイルに保存されているすべての SBOM ファイルは、SPDX または CycloneDX の .json ファイルとしてフォーマットされます。

アクセス許可ポリシー

指定されたプリンシパルとして AWS IoT Amazon S3 バケットに保存されている SBOM zip アーカイブファイルにアクセスするには、リソースベースのアクセス許可ポリシーが必要です。正しいリソースベースのアクセス許可ポリシーについては、次の例を参照してください。

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iot.amazonaws.com"
                ]
        },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::bucketName/*"
        }
    ]
}
```

リソースベースのアクセス許可ポリシーの詳細については、「AWS IoT リソースベースのポリシー」を参照してください。

SBOM の更新

ソフトウェア部品表は、デバイスフリートを保護し強化するために必要な頻度で更新できます。Amazon S3 バケットでソフトウェア部品表が更新されるたびに、バージョン ID が変更されます。このため、新しい Amazon S3 バケット URL を適切なパッケージバージョンに関連付ける必要があります。新しいバージョン ID は、 AWS マネジメントコンソールのパッケージバージョンページの Amazon S3 オブジェクトバージョン ID 列に表示されます。さらに、API オペレーション `GetPackageVersion` または CLI コマンド `get-package-version` を使用して、新しいバージョン ID を表示できます。

> **Note**
>
> 新しいバージョン ID の原因となるソフトウェア部品表を更新しても、新しいパッケージバージョンは作成されません。

Amazon S3 オブジェクトキー名の詳細については、「オブジェクトキー名の作成」を参照してください。

AWS IoT フリートインデックス作成の有効化

AWS IoT フリートインデックス作成を有効にすることは、 AWS IoT Device Management Software Package Catalog を使用するための要件です。Software Package Catalog で AWS IoT フリートのイ

ンデックス作成をするには、インデックスを作成してメトリクスを収集する各デバイスのデータソースとして予約済み名前付きシャドウ ($package) を設定します。予約済みの名前付きシャドウの詳細については、「予約済みの名前付きシャドウ」を参照してください。

フリートインデックス作成は、ソフトウェアパッケージバージョンでフィルタリングされたモノの動的グループを通じて AWS IoT モノをグループ化できるようにするサポートを提供します。例えば、フリートインデックス作成では、特定のパッケージバージョンがインストールされているモノ、されていないモノ、パッケージバージョンが何もインストールされていないモノ、特定の名前と値のペアに一致するモノを識別できます。最後に、フリートインデックスを作成すると、デバイスフリートの状態に関するインサイトを得るのに使用できる標準およびカスタムメトリクスが提供されます。詳細については、「フリートインデックス作成の準備」を参照してください。

> **Note**
>
> Software Package Catalog のフリートインデックス作成を有効にすると、標準のサービスコストが発生します。詳細については、「AWS IoT Device Management, Pricing」(、料金) を参照してください。

予約済みの名前付きシャドウ

予約済みの名前付きシャドウ、$package は、デバイスにインストールされているソフトウェアパッケージとパッケージバージョンの状態を反映しています。フリートインデックス作成では、予約済みの名前付きシャドウをデータソースとして使用して標準メトリクスとカスタムメトリクスを構築し、フリートの状態をクエリできるようにします。詳細については、「フリートインデックス作成の準備」を参照してください。

予約済みの名前付きシャドウは、名前があらかじめ定義されていて変更できないという点を除いて、名前付きのシャドウと似ています。さらに、予約済みの名前付きシャドウはメタデータを更新せず、`version` および `attributes` のキーワードのみを使用します。

`description` など、他のキーワードを含む更新リクエストには、`rejected` トピックの下でエラーレスポンスが返されます。詳細については、「Device Shadow エラーメッセージ」を参照してください。

コンソールから AWS IoT モノを作成するとき、 AWS IoT ジョブが正常にシャドウを完了して更新するとき、および `UpdateThingShadow` API オペレーションを発行するときに作成できます。詳細については、 デ AWS IoT Core ベロッパーガイドのUpdateThingShadow」を参照してください。

> **Note**
>
> 予約済みの名前付きシャドウをインデックスしても、フリートインデックスでインデックスできる名前付きシャドウの数にはカウントされません。クォータと制限の詳細については、「AWS IoT Device Management フリートインデックス作成の制限とクォータ」を参照してください。さらに、 AWS IoT ジョブが正常に完了したときに予約された名前付きシャドウをジョブで更新することを選択した場合、API コールは Device Shadow およびレジストリオペレーションにカウントされ、コストが発生する可能性があります。詳細については、「AWS IoT Device Management jobs limits and quotas」(ジョブの制限とクォータ) および IndexingFilter API データタイプを参照してください。

$package シャドウの構造

予約済みの名前付きシャドウには次のものが含まれます。

```
{
    "state": {
        "reported": {
            "<packageName>": {
                "version": "",
                "attributes": {
                }
            }
        }
    },
    "version" : 1
    "timestamp" : 1672531201
}
```

シャドウのプロパティは次の情報で更新されます。

- <packageName>: インストール済みのソフトウェアパッケージの名前。packageName パラメータで更新されます。
- version: インストールされているパッケージバージョンの名前。versionName パラメータで更新されます。
- attributes: デバイスに保存され、フリートインデックスによってインデックス化されるオプションのメタデータ。これにより、保存されたデータに基づいてインデックスにクエリを実行できます。

- `version`: シャドウのバージョン番号。シャドウが更新されるたびに自動的に増加し、1 から開始します。
- `timestamp`: シャドウが最後に更新された日時を示し、Unix 時間で記録されます。

名前付きシャドウの形式と動作の詳細については、「AWS IoT Device Shadow サービス メッセージの順序」を参照してください。

ソフトウェアパッケージとそのパッケージバージョンの削除

ソフトウェアパッケージを削除する前に、次の操作を行います。

- パッケージとそのバージョンがアクティブにデプロイされていないことを確認します。
- 最初に、関連するすべてのバージョンを削除します。いずれかのバージョンがデフォルトバージョンとして指定されている場合は、指定されたデフォルトバージョンをパッケージから削除する必要があります。デフォルトバージョンの指定は任意であり、削除しても競合はありません。ソフトウェアパッケージからデフォルトバージョンを削除するには、コンソールからパッケージを編集するか、 UpdatePackageVersion API オペレーションを使用します。

指定されたデフォルトパッケージバージョンがない限り、コンソールを使用してソフトウェアパッケージを削除でき、そのパッケージバージョンもすべて削除されます。API コールを使用してソフトウェアパッケージを削除する場合は、最初にパッケージバージョンを削除してからソフトウェアパッケージを削除する必要があります。

セキュリティの準備

このセクションでは、 AWS IoT Device Management Software Package Catalog の主なセキュリティ要件について説明します。

リソースベースの認証

Software Package Catalog では、リソースベースの認可を使用して、フリートのソフトウェアを更新する際のセキュリティを強化します。つまり、ソフトウェアパッケージとパッケージバージョンに対して `create`、、`readupdate`、`delete`、および `list`アクションを実行する権限を付与する AWS Identity and Access Management (IAM) ポリシーを作成し、 `Resources`セクションでデプロイする特定のソフトウェアパッケージとパッケージバージョンを参照する必要があります。予約済みの名前付きシャドウを更新するには、これらの権限も必要です。各エンティティに Amazon リソー

スネーム (ARN) を含めることで、ソフトウェアパッケージとパッケージバージョンを参照できます。

> **Note**
>
> パッケージバージョンの API コール (CreatePackageVersion, UpdatePackageVersion, DeletePackageVersion など) の権限をポリシーで付与するようにする場合は、ソフトウェアパッケージとパッケージバージョン ARN の両方をポリシーに含める必要があります。ポリシーでソフトウェアパッケージ API コール (CreatePackage、UpdatePackage、DeletePackage など) の権限を付与する場合は、ソフトウェアパッケージ ARN のみをポリシーに含める必要があります。

ソフトウェアパッケージとパッケージバージョン ARN を次のように構成します。

- ソフトウェアパッケージ: `arn:aws:iot:<region>:<accountID>:package/<packageName>/package`
- パッケージバージョン: `arn:aws:iot:<region>:<accountID>:package/<packageName>/version/<versionName>`

> **Note**
>
> このポリシーに含めることができる、その他の関連する権利もあります。例えば、job、thinggroup、jobtemplate に ARN を含めることができます。ポリシーオプションの詳細と詳細なリストについては、AWS IoT 「ジョブによるユーザーとデバイスの保護」を参照してください。

例えば、次のような名前のソフトウェアパッケージとパッケージバージョンがあるとします。

- AWS IoT モノ: `myThing`
- パッケージ名: `samplePackage`
- バージョン `1.0.0`

ポリシーは以下の例のようになります。

```
{
    "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 パッケージバージョンをデプロイするためのジョブ権限

セキュリティ上の理由から、パッケージとパッケージバージョンをデプロイする権限を付与し、デプロイが許可されている特定のパッケージとパッケージバージョンに名前を付けることが重要です。そのためには、パッケージバージョンでジョブをデプロイするアクセス許可を付与する IAM ロールとポリシーを作成します。ポリシーでは、リソースとしてターゲットパッケージのバージョンを指定する必要があります。

IAM ポリシー

IAM ポリシーは、Resource セクションで指定されているパッケージとバージョンを含むジョブを作成する権限を付与します。

```
{
    "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**
>
> ソフトウェアパッケージとパッケージのバージョンをアンインストールするジョブをデプロイする場合は、以下のようにパッケージバージョンが $null の ARN を認可する必要があります。

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

AWS IoT 予約済み名前付きシャドウを更新するジョブ権限

ジョブが正常に完了したときにジョブがモノの予約名シャドウを更新できるようにするには、IAM ロールとポリシーを作成する必要があります。 AWS IoT コンソールでこれを行うには 2 つの方法があります。1 つ目は、コンソールでソフトウェアパッケージを作成するときです。[パッケージ管理の依存関係を有効にする] ダイアログボックスが表示されたら、既存のロールを使用するか、新しいロールを作成するかを選択できます。または、 AWS IoT コンソールで [設定] を選択し、[インデック

ス作成の管理] を選択し、次に [デバイスパッケージとバージョンのインデックス作成の管理] を選択します。

> **Note**
>
> AWS IoT ジョブが正常に完了したときに予約された名前付きシャドウをジョブサービスで更新することを選択した場合、API コールは Device Shadow およびレジストリオペレーションにカウントされ、コストが発生する可能性があります。詳細については、AWS IoT Core 料金表を参照してください。

[ロールを作成] オプションを使用すると、生成されるロールの名前は `aws-iot-role-update-shadows` で始まり、次のポリシーが含まれています。

ロールのセットアップ

アクセス許可

アクセス許可ポリシーにより、モノのシャドウへのクエリと更新を行う権限が付与されます。リソース ARN の $package パラメータは、予約済みの名前付きシャドウを対象としています。

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

信頼関係

アクセス許可ポリシーに加えて、エンティティがロールを引き継いで予約済みの名前付きシャドウを更新できるように、ロールと AWS IoT Core の信頼関係も必要です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

ユーザーポリシーの設定

iam:PassRole アクセス許可

最後に、UpdatePackageConfiguration API オペレーションを呼び出す AWS IoT Core ときに にロールを渡すアクセス許可が必要です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iot:UpdatePackageConfiguration"
            ],
            "Resource": "arn:aws:iam::111122223333:role/<roleName>"
        }
    ]
}
```

AWS IoT Amazon S3 からダウンロードするジョブのアクセス許可

ジョブドキュメントは Amazon S3 に保存されます。 AWS IoT ジョブを介してディスパッチするときに、このファイルを参照します。ファイルをダウンロードする権限を AWS IoT Jobs に提供する必要があります (`s3:GetObject`)。また、Amazon S3 と AWS IoT ジョブ間の信頼関係を設定する必要があります。これらのポリシーを作成する手順については、「ジョブを管理する」の「署名付き URL」を参照してください。

パッケージバージョンのソフトウェア部品表を更新するアクセス許可

`Draft`、、または `Deprecated`ライフサイクル状態のパッケージバージョンのソフトウェア部品表を更新するには、Amazon S3 で新しいソフトウェア部品表を検索し`Published`、 でパッケージバージョンを更新するための AWS Identity and Access Management ロールとポリシーが必要です AWS IoT Core。

まず、更新されたソフトウェア部品表をバージョニングされた Amazon S3 バケットに配置し、`sboms` パラメータを含む `UpdatePackageVersion` API オペレーションを呼び出します。次に、承認されたプリンシパルが作成した IAM ロールを引き受け、Amazon S3 で更新されたソフトウェア部品表を見つけ、ソフトウェアパッケージカタログ AWS IoT Core の でパッケージ検証を更新します。

この更新を実行するには、次のポリシーが必要です。

ポリシー

- 信頼ポリシー IAM ロールを引き受ける承認されたプリンシパルとの信頼関係を確立し、Amazon S3 のバージョニングされたバケットから更新されたソフトウェア部品表を見つけ、 AWS IoT Coreのパッケージバージョンを更新できるようにします。
  -

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

- アクセス許可ポリシー: ソフトウェア部品表がパッケージバージョン用に保存されている Amazon S3 バージョニングバケットにアクセスし、 AWS IoT Coreでパッケージバージョンを更新するポリシー。

  - ```
    {
        "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>"
            ]
        }
    ]
}
```

- ロールのアクセス許可を渡す: UpdatePackageVersion API オペレーションを呼び出す AWS IoT Core ときに IAM ロールを Amazon S3 に渡すアクセス許可を付与するポリシー。

  -
```
{
  "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` ライフサイクル状態に移行したパッケージバージョンでは、ソフトウェア部品表を更新することはできません。

AWS サービスの IAM ロールの作成の詳細については、AWS 「 サービスにアクセス許可を委任するロールの作成」を参照してください。

Amazon S3 バケットの作成とそれへのオブジェクトのアップロードの詳細については、「バケットの作成」と「オブジェクトのアップロード」を参照してください。

フリートインデックス作成の準備

AWS IoT フリートインデックス作成では、予約された名前付きシャドウ () を使用してデータを検索および集計できます`$package`。 AWS IoT およびモノの予約済みの名前付きシャドウ動的グループをクエリすることで、モノをグループ化することもできます。たとえば、特定のパッケージバージョンを使用しているモノ、特定のパッケージバージョンがインストールされていないモノ、またはパッケージバージョンがインストールされていない AWS IoT モノに関する情報を確認できます。属性を組み合わせることで、さらに詳しいインサイトを得ることができます。例えば、特定のバージョンで特定のタイプのもの (バージョン 1.0.0 や pump_sensor のモノのタイプなど) を識別します。詳細については、「フリートインデックス作成」を参照してください。

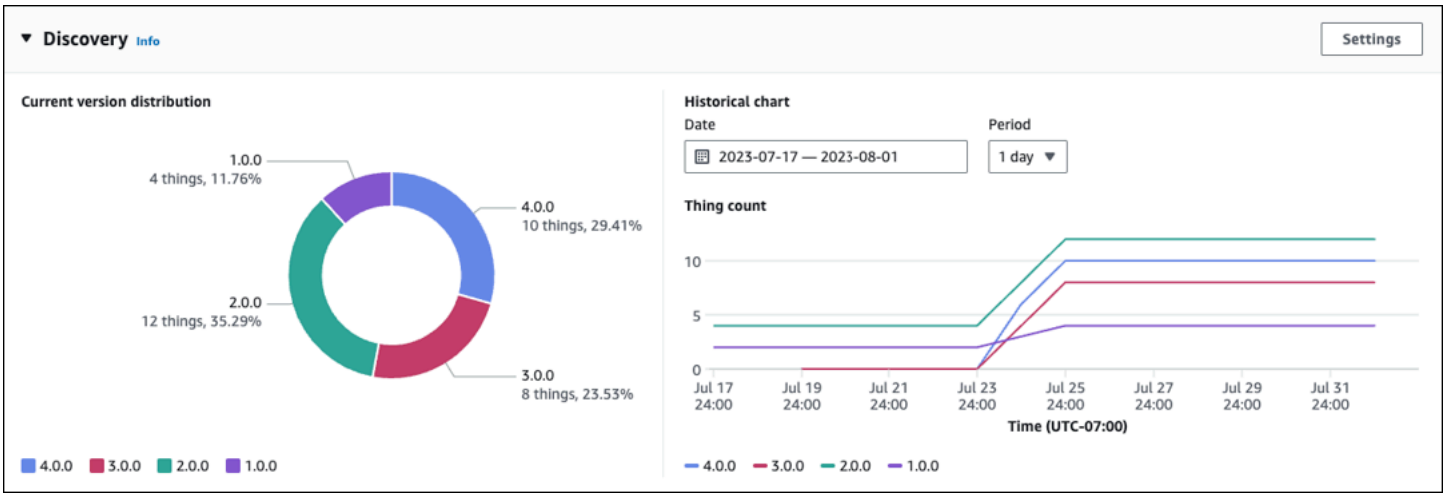
$package シャドウをデータソースとして設定する

Software Package Catalog でフリートインデックス作成を使用するには、フリートインデックス作成を有効にし、名前付きシャドウをデータソースとして設定して、名前付きシャドウフィルターとして `$package` を定義する必要があります。フリートインデックス作成をまだ有効にしていない場合は、このプロセス内で有効にできます。コンソールの AWS IoT Core から [設定] を開き、[インデックス作成の管理]、[名前付きシャドウの追加]、[デバイスソフトウェアパッケージとバージョンの追加]、[更新] の順に選択します。詳細については、「モノのインデックス作成の管理」を参照してください。

または、最初のパッケージを作成するときにフリートインデックス作成を有効にすることもできます。[パッケージ管理の依存関係を有効にする] ダイアログボックスが表示されたら、デバイスソフトウェアのパッケージとバージョンをデータソースとしてフリートインデックス作成に追加するオプションを選択します。このオプションを選択すると、フリートインデックス作成も有効になります。

> **Note**
>
> Software Package Catalog のフリートインデックス作成を有効にすると、標準のサービスコストが発生します。詳細については、「AWS IoT Device Management, Pricing」(、料金) を参照してください。

コンソールに表示されるメトリクス



AWS IoT コンソールソフトウェアパッケージの詳細ページでは、検出パネルにシャ$packageドウを介して取り込まれた標準メトリクスが表示されます。

- 現行バージョンのディストリビューションチャートには、このソフトウェアパッケージに関連付けられているすべてのデバイスの AWS IoT モノに関連付けられている最新のパッケージバージョン 10 のデバイス数と割合が表示されます。注: ソフトウェアパッケージに含まれるパッケージバージョンが、グラフに表示されているバージョンよりも多い場合は、その他にグループ分けして表示できます。
- 履歴グラフには、指定した期間における選択したパッケージバージョンに関連するデバイスの数が表示されます。最大 5 つのパッケージバージョンを選択し、日付範囲と時間間隔を定義するまで、最初はグラフは空です。グラフのパラメータを選択するには、[設定] を選択します。[履歴グラフ] に表示されるデータは、表示されるパッケージバージョンの数が異なることと、[履歴グラフ] で分析するパッケージバージョンを選択できるため、[現在のバージョン分布] グラフと異なる場合があります。注: 視覚化するパッケージバージョンを選択すると、そのバージョンはフリートメトリクスの最大数制限にカウントされます。クォータと制限の詳細については、「フリートインデックス作成の制限とクォータ」を参照してください。

パッケージバージョン分布を収集する方法についてインサイトを得る別の方法については、「Collecting package version distribution through getBucketsAggregation」(によるパッケージバージョン配布の収集) を参照してください。

クエリパターン

Software Package Catalog によるフリートのインデックス作成では、標準的にサポートされている大抵の機能 (用語、フレーズ、検索フィールドなど) を使用します。ただし、予約済みの名前付きシャドウ ($package) version キーでは、比較演算子 (小なり「<」や、大なり「>」など) と range クエリは使用できません。ただし、attributes キーではこれらのクエリを使用できます。詳細については、「クエリ構文」を参照してください。

サンプルデータ

注: 予約済みの名前付きシャドウとその構造については、「予約済みの名前付きシャドウ」を参照してください。

この例では、最初のデバイスに AnyThing という名前が付けられ、次のパッケージがインストールされています。

- ソフトウェアパッケージ: SamplePackage

  パッケージバージョン: 1.0.0

  パッケージ ID: 1111

シャドウは次のようになります。

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                    }
            }
        }
```

```
    }
}
```

2 番目のデバイスには AnotherThing という名前が付けられ、次のパッケージがインストールされています。

- ソフトウェアパッケージ: SamplePackage

  パッケージバージョン: 1.0.0

  パッケージ ID: 1111
- ソフトウェアパッケージ: OtherPackage

  パッケージバージョン: 1.2.5

  パッケージ ID: 2222

シャドウは次のようになります。

```
{
    "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"
                    }
            },
        }
    }
}
```

サンプルクエリ

次の表は、AnyThing および AnotherThing のデバイスシャドウの例に基づいたクエリの例を示しています。詳細については、「モノのクエリの例」を参照してください。

AWS IoT Device Tester for FreeRTOS の最新バージョン

リクエストされた情報	クエリ	結果
特定のパッケージバージョンがインストールされているモノ	`shadow.name.$package.reported.SamplePackage.version:1.0.0`	AnyThing, OtherThing
特定のパッケージバージョンがインストールされていないモノ	`NOT shadow.name.$package.reported.OtherPackage.version:1.2.5`	AnyThing
パッケージ ID が 1500 を超えるパッケージバージョンを使用するすべてのデバイス	`shadow.name.$package.reported.*.attributes.packageID>1500"`	OtherThing
特定のパッケージがインストールされていて、複数のパッケージがインストールされているモノ	`shadow.name.$package.reported.SamplePackage.version:1.0.0 AND shadow.name.$package.reported.totalCount:2`	OtherThing

getBucketsAggregation によるパッケージバージョン配布の収集

AWS IoT コンソール内の Discovery パネルに加えて、 GetBucketsAggregation API オペレーションを使用してパッケージバージョンのディストリビューション情報を取得することもできます。パッケージバージョンの配布情報を取得するには、以下を実行する必要があります。

- 各ソフトウェアパッケージのフリートインデックス作成内のカスタムフィールドを定義します。注: カスタムフィールドの作成は、AWS IoT フリートインデックス作成のサービスクォータにカウントされます。
- カスタムフィールドを次のようにフォーマットします。

  ```
  shadow.name.$package.reported.<packageName>.version
  ```

詳細については、フリー AWS IoT トインデックス作成の「カスタムフィールド」セクションを参照してください。

AWS IoT ジョブの準備

AWS IoT Device Management Software Package Catalog は、置換パラメータ、 AWS IoT フリートインデックス作成、モノの動的グループ、モノ AWS IoT の予約済み名前付きシャドウとの統合を通じて AWS IoT ジョブを拡張します。

> **Note**
>
> Software Package Catalog が提供するすべての機能を使用するには、次の AWS Identity and Access Management (IAM) ロールとポリシーを作成する必要があります: AWS IoT パッケージバージョンをデプロイするジョブ権限とAWS IoT 、予約された名前付きシャドウを更新するジョブ権限。詳細については、「セキュリティの準備」を参照してください。

AWS IoT ジョブの置換パラメータ

置換パラメータは、 AWS IoT ジョブドキュメント内のプレースホルダーとして使用できます。ジョブサービスが代替パラメータを検出すると、そのジョブは、指定されたソフトウェアバージョンのパラメータ値の属性を指します。このプロセスを使用して 1 つのジョブドキュメントを作成し、汎用属性を介してメタデータをジョブに渡すことができます。例えば、Amazon Simple Storage Service (Amazon S3) URL、ソフトウェアパッケージ Amazon リソースネーム (ARN)、または署名をパッケージバージョン属性を介してジョブドキュメントに渡すことができます。

代替パラメータは、ジョブドキュメントで次のようにフォーマットする必要があります。

- ソフトウェアパッケージ名とパッケージバージョン
  - `package::version` 間の空の文字列は、ソフトウェアパッケージ名の置換パラメータを表します。`version::attribute` 間の空の文字列は、ソフトウェアパッ

ケージバージョンの置換パラメータを表します。ジョブドキュメントでパッケージ名とパッケージ検証置換パラメータを使用するには、次の例を参照してください。${aws:iot:package::version::attributes:<attributekey>}

- ジョブドキュメントは、パッケージバージョンの詳細からバージョン ARN を使用して、これらの置換パラメータを自動入力します。API または CLI コマンドを使用して単一パッケージデプロイ用のジョブまたはジョブテンプレートを作成する場合、パッケージバージョンのバージョン ARN は CreateJob および DescribeJob の destinationPackageVersions パラメータで表されます。

- ソフトウェアパッケージバージョンのすべての属性
  - ジョブドキュメントでソフトウェアパッケージバージョン置換パラメータのすべての属性を使用するには、次の例を参照してください。${aws:iot:package:<packageName>:version:<versionName>:attributes}

> **Note**
>
> パッケージ名、パッケージバージョン、およびすべての属性置換パラメータを一緒に使用できます。ジョブドキュメントで 3 つの置換パラメータをすべて使用する方法については、次の例を参照してください。${aws:iot:package::version::attributes}

次の例では、samplePackage という名前のソフトウェアパッケージがあり、そのパッケージバージョンには以下の属性を持つ 2.1.5 という名前のパッケージバージョンがあります。

- 名前: s3URL、値: https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/exampleCodeFile
  - この属性は、Amazon S3 内に保存されているコードファイルの場所を識別します。
- 名前: signature、値: aaaaabbbbbccccddddeeeeeffffgggggghhhhhiiiiijjjj
  - この属性は、デバイスがセキュリティ対策として必要とするコード署名値を提供します。詳細については、「ジョブのコード署名」を参照してください。注: この属性は一例であり、Software Package Catalog やジョブの一部としては必須ではありません。

s3URL の場合、ジョブドキュメントパラメータは次のように記述されます。

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
```

```
}
```

signature の場合、ジョブドキュメントパラメータは次のように記述されます。

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
}
```

ジョブドキュメント全体は次のように記述されます。

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

置換が行われると、次のジョブドキュメントがデバイスにデプロイされます。

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage": "https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/
exampleCodeFile"
      },
    ],
    "signature": [
      "samplePackage" : "aaaaabbbbbcccccdddddeeeeeffffffgggggghhhhhiiiiijjjj"
    ]
```

```
  }
}
```

置換パラメータ (表示前と表示後)

代替パラメータは、デフォルトのパッケージバージョンの $default など、さまざまなフラグを使用してジョブドキュメントの作成を効率化します。これにより、特定のパッケージ検証で参照されるメタデータがそれらのフラグに自動入力されるため、ジョブデプロイごとに特定のパッケージバージョンメタデータを手動で入力する必要がなくなります。デフォルトのパッケージバージョンの $default など、パッケージバージョンの属性の詳細については、「デプロイ用のジョブドキュメントとパッケージバージョンの準備」を参照してください。

で AWS マネジメントコンソール、パッケージバージョンのジョブデプロイ中にデプロイ指示ファイルエディタウィンドウのプレビュー置換ボタンを切り替えると、置換パラメータの有無にかかわらずジョブドキュメントが表示されます。

DescribeJob および GetJobDocument API の「置換前」パラメータを使用すると、置換パラメータが削除される前後の API レスポンスを表示できます。DescribeJob および GetJobDocument API で次の例を参照してください。

- DescribeJob
  - デフォルトビュー

```
{
    "jobId": "<jobId>",
    "description": "<description>",
    "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/1.0.2"]
}
```

  - 置換前ビュー

```
{
    "jobId": "<jobId>",
    "description": "<description>",
    "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/$default"]
}
```

- GetJobDocument
  - デフォルトビュー

```
{
    "attributes": {
        "location": "prod-artifacts.s3.us-east-1.amazonaws.com/mqtt-core",
        "signature": "IQoJb3JpZ2luX2VjEIrwEaCXVzLWVhc3QtMSJHMEUCIAofPNPpZ9cI",
        "streamName": "mqtt-core",
        "fileId": "0"
    },
}
```

- 置換前ビュー

```
{
    "attributes": "${aws:iot:package:TestPackage:version:$default:attributes}",
}
```

AWS IoT ジョブ、ジョブドキュメントの作成、ジョブのデプロイの詳細については、「ジョブ」を参照してください。

デプロイ用のジョブドキュメントとパッケージバージョンの準備

パッケージバージョンが作成されると、デプロイの準備中であることを示す draft 状態になります。デプロイのパッケージバージョンを準備するには、ジョブドキュメントを作成し、ジョブがアクセスできる場所 (Amazon S3 など) にドキュメントを保存して、パッケージバージョンにジョブドキュメントで使用したい属性値があることを確認し、パッケージバージョンを公開する必要があります。(注意: draft 状態にあるパッケージバージョンの属性のみを更新できます)。

単一パッケージデプロイ用の AWS IoT ジョブまたは ジョブテンプレートを作成する場合、ジョブドキュメントをカスタマイズするための以下のオプションがあります。

デプロイ指示ファイル (**recipe**)

- パッケージバージョンのデプロイ指示ファイルには、パッケージバージョンを複数のデバイスにデプロイするためのインラインジョブドキュメントを含むデプロイ指示が含まれています。ファイルは、迅速かつ効率的なジョブデプロイのために、特定のデプロイ手順をパッケージバージョンに関連付けます。

  では AWS マネジメントコンソール、新しいパッケージの作成ワークフローの「バージョンデプロイ設定」タブの「デプロイ手順ファイルプレビュー」ウィンドウで ファイルを作成できます。AWS IoT を使用して、推奨ファイルから AWS IoT 開始を使用してパッケージバージョン属性から

指示ファイルを自動的に生成するか、独自のデプロイ指示ファイルを使用して Amazon S3 バケットに保存されている既存のジョブドキュメントを使用できます。

> **Note**
>
> 独自のジョブドキュメントを使用する場合は、[デプロイ手順ファイルプレビュー] ウィンドウで直接更新できますが、Amazon S3 バケットに保存されている元のジョブドキュメントは自動的に更新されません。

AWS CLI または `CreatePackageVersion`、、 `GetPackageVersion`などの API コマンドを使用する場合`UpdatePackageVersion`、 はインラインジョブドキュメントを含むデプロイ指示ファイル`recipe`を表します。

ジョブドキュメントの詳細については、「基本概念」を参照してください。

`recipe` で表されるデプロイ指示ファイルについては、次の例を参照してください。

```
{
    "packageName": "sample-package-name",
    "versionName": "sample-package-version",
    ...
    "recipe": "{...}"
}
```

> **Note**
>
> `recipe` で表されるデプロイインストラクションファイルは、パッケージバージョンがパッケージバージョンメタデータとは別に published ステータスの状態にあるときに更新できます。ジョブのデプロイ中は変更できなくなります。

`Artifact` バージョン属性

- ソフトウェアパッケージバージョンのバージョン属性 `artifact` を使用して、パッケージバージョンアーティファクトの Amazon S3 の場所を追加できます。 AWS IoT ジョブを使用してパッケージバージョンのジョブデプロイがトリガーされると、ジョブドキュメントの署名付き URL プレースホルダー `${aws:iot:package:<packageName>:version:<versionName>:artifact-`

location:*s3-presigned-url*} は、Amazon S3 バケット、バケットキー、および Amazon S3 バケットに保存されているファイルのバージョンを使用して更新されます。パッケージバージョンのアーティファクトを格納する Amazon S3 バケットは、パッケージバージョンが作成されたのと同じリージョンに配置する必要があります。

> **Note**
>
> Amazon S3 バケットに同じファイルの複数のオブジェクトバージョンを保存するには、バケットでバージョニングを有効にする必要があります。詳細については、「バケットでのバージョニングの有効化」を参照してください。

CreatePackageVersion または UpdatePackageVersion API オペレーションを使用するときに Amazon S3 バケット内のパッケージバージョンアーティファクトにアクセスするには、次のアクセス許可が必要です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObjectVersion",
            "Resource": "arn:aws:s3:::bucket-name/key-name"
        }
    ]
}
```

CreatePackageVersion および UpdatePackageVersion API オペレーションのバージョン属性 artifact の詳細については、「CreatePackageVersion」および「UpdatePackageVersion」を参照してください。

新しいパッケージバージョンを作成するときに Amazon S3 のアーティファクトの場所をサポートするバージョン属性 artifact を示す次の例を参照してください。

```
{
    "packageName": "sample package name",
    "versionName": "1.0",
    "artifact": {
        "s3Location": {
```

```
            "bucket": "firmware",
            "key": "image.bin",
            "version": "12345"
        }
    }
}
```

> **Note**
>
> パッケージバージョンが draft ステータス状態から published ステータス状態に更新されると、パッケージバージョンの属性とアーティファクトの場所は変更不可能になります。この情報を更新するには、新しいパッケージバージョンを作成し、draft ステータス状態で更新を実行する必要があります。

パッケージバージョン

- デフォルトのソフトウェアパッケージバージョンは、安全で安定したパッケージバージョンを提供するソフトウェアパッケージの利用可能なバージョンに表示できます。これは、 AWS IoT ジョブを使用してデフォルトのパッケージバージョンをデバイスフリートにデプロイするときのソフトウェアパッケージのベースラインバージョンとして機能します。ソフトウェアパッケージの $default パッケージバージョンをデプロイするジョブを作成する場合、ジョブドキュメントと新しいジョブデプロイのパッケージバージョンは $default と一致する必要があります。ジョブデプロイのパッケージバージョンは、API コマンドと CLI コマンドの destinationPackageVersions と AWS マネジメントコンソールの VersionARN で表されます。ジョブドキュメントのパッケージバージョンは、以下に示すジョブドキュメントプレースホルダーによって表されます。

  ```
  arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$default
  ```

  デフォルトのパッケージバージョンを使用してジョブまたはジョブテンプレートを作成するには、以下に示すように、CreateJob または CreateJobTemplate API コマンドで $default フラグを使用します。

  ```
  "$ aws iot create-job \
      --destination-package-versions "arn:aws:iot:us-west-2:123456789012:package/
  TestPackage/version/$default"
      --document file://jobdoc.json
  ```

> **Note**
>
> デフォルトバージョンを参照する `$default` パッケージバージョン属性は、 AWS IoT ジョブを介したジョブデプロイのデフォルトパッケージバージョンを参照する場合にのみ必要なオプションの属性です。

パッケージバージョンに満足したら、 AWS IoT コンソールのソフトウェアパッケージの詳細ページまたは UpdatePackageVersion API オペレーションを発行して公開します。その後、 AWS IoT コンソールから、または CreateJob API オペレーションを発行してジョブを作成するときに、パッケージバージョンを参照できます。

デプロイ時のパッケージとバージョンの命名

ソフトウェアパッケージバージョンをデバイスにデプロイするには、ジョブドキュメントで参照されているソフトウェアパッケージとパッケージバージョンが、`CreateJob` API オペレーションの `destinationPackageVersions` パラメータに記載されているソフトウェアパッケージとパッケージバージョンと一致していることを確認します。一致しない場合は、両方の参照を一致させるように促すエラーメッセージが表示されます。Software Package Catalog のエラーメッセージの詳細については、「エラーメッセージの一般的なトラブルシューティング」を参照してください。

ジョブドキュメントで参照されているソフトウェアパッケージとパッケージバージョンに加えて、ジョブドキュメントで参照されていない `CreateJob` API オペレーションの `destinationPackageVersions` パラメータに追加のソフトウェアパッケージとパッケージバージョンを含めることができます。追加のソフトウェアパッケージバージョンを適切にインストールするために、必要なインストール情報がジョブドキュメントに含まれていることを確認します。`CreateJob` API オペレーションの詳細については、「CreateJob」を参照してください。

モノの AWS IoT 動的グループによるジョブのターゲット設定

Software Package Catalog は、フリートインデックス作成、AWS IoT のジョブ、AWS IoT のモノの動的グループと連携して、フリート内のデバイスをフィルタリングしてターゲットにし、デバイスにデプロイするパッケージバージョンを選択します。デバイスの現在のパッケージ情報に基づいてフリートインデックス作成クエリを実行し、それらのモノを AWS IoT ジョブにターゲットにすることができます。ソフトウェアアップデートをリリースすることもできますが、対象となるターゲットデバイスに対してのみです。例えば、現在 `iot-device-client 1.5.09` を実行しているデバイスに

のみ設定をデプロイするように指定できます。詳細については、「モノの動的グループを作成する」を参照してください。

予約済みの名前付きシャドウとパッケージバージョン

設定されている場合、 AWS IoT ジョブが正常に完了すると、ジョブはモノの予約済み名前付きシャドウ ($package) を更新できます。そうすれば、パッケージバージョンをモノの予約済みの名前付きシャドウに手動で関連付ける必要がなくなります。

次のような状況では、パッケージバージョンをモノの予約済みの名前付きシャドウに手動で関連付けるか、更新することを選択できます。

- インストールされたパッケージバージョンを関連付け AWS IoT Core ずに、モノを に登録します。
- AWS IoT ジョブは、モノの予約済み名前付きシャドウを更新するように設定されていません。
- 社内プロセスを使用してパッケージバージョンをフリートにディスパッチすると、そのプロセスは完了 AWS IoT Core しても更新されません。

> **Note**
>
> AWS IoT ジョブを使用して、予約済み名前付きシャドウ () のパッケージバージョンを更新することをお勧めします$package。 AWS IoT Jobs $package もシャドウを更新するように設定されている場合、他のプロセス (手動またはプログラムによる API コールなど) を通じてシャドウ内のバージョンパラメータを更新すると、デバイス上の実際のバージョンと予約された名前付きシャドウに報告されたバージョンとの間に不整合が発生する可能性があります。

コンソールまたは `UpdateThingShadow` API オペレーションを使用して、モノの予約済みの名前付きシャドウ ($package) にパッケージバージョンを追加または更新できます。詳細については、「 AWS IoT モノへのパッケージバージョンの関連付け」を参照してください。

> **Note**
>
> パッケージバージョンを AWS IoT モノに関連付けても、デバイスソフトウェアは直接更新されません。デバイスソフトウェアを更新するには、パッケージバージョンをデバイスにデプロイする必要があります。

ソフトウェアパッケージとそのパッケージバージョンのアンインストール

$null は、 AWS IoT Jobs サービスに、デバイスの予約済み名前付きシャドウ から既存のソフトウェアパッケージとパッケージバージョンを削除するように求める予約済みプレースホルダーです$package。詳細については、「予約済みの名前付きシャドウ」を参照してください。

この機能を使用するには、destinationPackageVersion Amazon リソースネーム (ARN) を $null に置き換えます。その後、デバイスからソフトウェアを削除するようにサービスに指示する必要があります。

承認された ARN は次の形式を使用します。

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

例えば、

```
$ aws iot create-job \
    ... \
    --destinationPackageVersions ["arn:aws:iot:us-east-1:111122223333:package/
samplePackage/version/$null"]
```

Software Package Catalog の開始方法

AWS IoT Device Management Software Package Catalog は AWS マネジメントコンソール、、AWS IoT Core API オペレーション、および AWS Command Line Interface () を使用して構築および保守できますAWS CLI。

> **Note**
>
> AWS IoT フリートインデックス作成を有効にすることは、 Software Package Catalog を使用するための要件です。でソフトウェアパッケージバージョンを作成し、 CreatePackage API コマンド AWS マネジメントコンソール を使用するなどの基本的な操作は、 AWS IoT フリートインデックス作成を有効にしないと失敗します。
> Software Package Catalog での AWS IoT フリートインデックス作成の使用の詳細については、「」を参照してくださいフリートインデックス作成の準備。

コンソールの使用

を使用するには AWS マネジメントコンソール、 AWS アカウントにサインインし、 に移動しますAWS IoT Core。ナビゲーションペインで、[ソフトウェアパッケージ] を選択します。その後で、このセクションからパッケージとそのバージョンを作成および管理できます。

API または CLI オペレーションの使用

AWS IoT Core API オペレーションを使用して、 Software Package Catalog 機能を作成および管理できます。詳細については、「AWS IoT API リファレンス」および「AWS SDK とツールキット」を参照してください。 AWS CLI コマンドはカタログも管理します。詳細については、「AWS IoT CLI コマンドリファレンス」を参照してください。

この章には、以下のセクションが含まれています。

- ソフトウェアパッケージとパッケージバージョンの作成
- AWS IoT ジョブによるパッケージバージョンのデプロイ
- パッケージバージョンを AWS IoT モノに関連付ける

ソフトウェアパッケージとパッケージバージョンの作成

次の手順を使用して、 AWS マネジメントコンソールからパッケージと初期バージョンのモノを作成できます。

ソフトウェアパッケージを作成するには

1. AWS アカウントにサインインし、 AWS IoT コンソールに移動します。
2. ナビゲーションペインで、[ソフトウェアパッケージ] を選択します。
3. [AWS IoT ソフトウェアパッケージ] ページで、[パッケージの作成] を選択します。[パッケージ管理の依存関係を有効にする] ダイアログボックスが表示されます。
4. [フリートインデックス作成] で、[デバイスソフトウェアパッケージとバージョンの追加] を選択します。これは、Software Package Catalog に必須で、フリートインデックス作成とフリートに関するメトリクスを提供します。
5. [オプション] AWS IoT ジョブが正常に完了したときにリザーブドの名前付きシャドウを更新する場合は、ジョブからシャドウの自動更新を選択します。 AWS IoT ジョブでこの更新を行わない場合は、このチェックボックスはオフのままにします。
6. [オプション] 予約済み名前付きシャドウを更新する権限を AWS IoT ジョブに付与するには、「ロールの選択」で「ロールの作成」を選択します。 AWS IoT ジョブでこの更新を行わない場合は、このロールは必要ありません。

7. ロールを作成または選択します。

   a. この目的のロールがない場合: [ロールを作成] ダイアログボックスが表示されたら、[ロール名] を入力して [作成] を選択します。

   b. この目的のロールがある場合: [ロールを選択] でロールを選択し、[IAM ロールにポリシーをアタッチ] チェックボックスがオンになっていることを確認します。

8. [確認] を選択します。[新しいパッケージを作成] ページが表示されます。

9. [パッケージの詳細] に、パッケージ名を入力します。

10. [パッケージの説明] に、このパッケージの識別と管理に役立つ情報を入力します。

11. [オプション] タグを使用すると、このパッケージを分類および管理しやすくなります。タグを追加するには、[タグ] を展開して [タグを追加] を選択し、キーと値のペアを入力します。最大 50 個のタグを入力できます。詳細については、「AWS IoT リソースのタグ付け」を参照してください。

新しいパッケージの作成時にパッケージバージョンを追加するには

1. [初期バージョン] に、[バージョン名] を入力します。

   パッケージのバージョンを一意に識別するには、SemVer 形式 (例: `1.0.0.0`) を使用することをお勧めします。また、ユースケースに適した別のフォーマット戦略を使用することもできます。詳細については、「パッケージバージョンライフサイクル」を参照してください。

2. [バージョンの説明] に、このパッケージバージョンの識別と管理に役立つ情報を入力します。

   > **Note**
   >
   > パッケージバージョンは `draft` 状態で作成されるため、[デフォルトバージョン] チェックボックスはオフになっています。パッケージバージョンの作成後、および状態を `published` に変更したときに、デフォルトバージョンに名前を付けることができます。詳細については、「パッケージバージョンライフサイクル」を参照してください。

3. [オプション] このバージョンを管理したり、デバイスに情報を伝えたりするには、[バージョン属性] に名前と値のペアを 1 つ以上入力します。入力する名前と値のペアごとに [属性の追加] を選択します。詳細については、「バージョン属性」を参照してください。

4. [オプション] タグを使用すると、このパッケージを分類および管理しやすくなります。タグを追加するには、[タグ] を展開して [タグを追加] を選択し、キーと値のペアを入力します。最大 50

個のタグを入力できます。詳細については、「AWS IoT リソースのタグ付け」を参照してください。

5. [次へ] を選択します。

ソフトウェア部品表をパッケージバージョンに関連付ける (オプション)

1. [SBOM 設定] ウィンドウの [ステップ 3: バージョン SBOM (オプション)] で、パッケージバージョンに関連付ける前に、ソフトウェア部品表の検証に使用されるデフォルトの SBOM ファイル形式と検証モードを選択します。
2. [SBOM ファイルを追加] ウィンドウで、バージョニングされた Amazon S3 バケットを表す Amazon リソースネーム (ARN) と、デフォルトのタイプが機能しない場合は任意の SBOM ファイル形式を入力します。

> **Note**
>
> パッケージバージョンに複数のソフトウェア部品表がある場合は、1 つの SBOM ファイルまたは複数の SBOM を含む 1 つの zip ファイルを追加できます。

3. [追加された SBOM ファイル] ウィンドウで、パッケージバージョンに追加した SBOM ファイルを表示できます。
4. [パッケージとバージョンの作成] を選択します。パッケージバージョンページが表示され、[追加された SBOM ファイル] ウィンドウで SBOM ファイルの検証ステータスを確認できます。SBOM ファイルが検証されるときに初期ステータスは `In progress` になります。

> **Note**
>
> SBOM ファイルの検証ステータスは、`Invalid file`、`Not started`、`In progress`、`Validated (SPDX)`、`Validated (CycloneDX)` および検証失敗の理由です。

AWS IoT ジョブによるパッケージバージョンのデプロイ

次の手順を使用して、 AWS マネジメントコンソールからパッケージバージョンをデプロイできます。

前提条件:

開始する前に、以下を実行します。

- AWS IoT モノを に登録します AWS IoT Core。デバイスを追加する手順については AWS IoT Core、「モノオブジェクトの作成」を参照してください。
- [オプション] AWS IoT モノのグループまたはモノの動的グループを作成して、パッケージバージョンをデプロイするデバイスをターゲットにします。モノのグループの作成方法については、「モノの静的グループの作成」を参照してください。モノの動的グループの作成方法については、「モノの動的グループを作成する」を参照してください。
- ソフトウェアパッケージとパッケージバージョンを作成します。詳細については、「ソフトウェアパッケージとパッケージバージョンの作成」を参照してください。
- ジョブドキュメントを作成します。詳細については、「デプロイ用のジョブドキュメントとパッケージバージョンの準備」を参照してください。

AWS IoT ジョブをデプロイするには

1. AWS IoT コンソールで、[ソフトウェアパッケージ] を選択します。
2. デプロイするソフトウェアパッケージを選択します。[ソフトウェアパッケージ詳細] ページが表示されます。
3. [バージョン] でデプロイするパッケージバージョンを選択し、[ジョブバージョンのデプロイ] を選択します。
4. このポータルから初めてジョブをデプロイする場合は、要件を説明するダイアログボックスが表示されます。情報を確認してから、[確認] を選択します。
5. デプロイの名前を入力するか、自動生成された名前を [名前] フィールドに残します。
6. [オプション] [説明] フィールドに、デプロイの目的や内容を特定する説明を入力するか、自動生成された情報をそのまま残します。

   注意: ジョブ名および説明のフィールドに個人を特定できる情報を使用しないことをお勧めします。
7. [オプション] このジョブに関連付けるタグをすべて追加します。
8. [次へ] を選択します。
9. [ジョブターゲット] で、ジョブを受け取るモノまたはモノのグループを選択します。
10. [ジョブファイル] フィールドで、ジョブドキュメントの JSON ファイルを指定します。
11. [Package Catalog サービスとのジョブの統合] を開きます。
12. ジョブドキュメント内で指定されているパッケージとバージョンを選択します。

> **Note**
>
> ジョブドキュメント内で指定されているのと同じパッケージとパッケージバージョンを選択する必要があります。さらに多くを含めることもできますが、ジョブから発行される指示は、ジョブドキュメントに含まれるパッケージとバージョンについてのみです。詳細については、「デプロイ時のパッケージとバージョンの命名」を参照してください。

13. [次へ] を選択します。
14. [ジョブ設定] ダイアログボックスの [ジョブ設定] ページで、次のいずれかのジョブタイプを選択します。

    - スナップショットジョブ: スナップショットジョブは、ターゲットデバイスおよびグループでの実行が終了すると完了します。
    - 連続ジョブ: 連続ジョブはモノのグループに適用され、指定したターゲットグループに後に追加するあらゆるデバイス上で実行されます。

15. [その他の設定 - オプション] ダイアログボックスで、以下のオプションのジョブ設定を確認し、それに応じて選択してください。詳細については、「ジョブのロールアウト、スケジュール、中止の設定」および「ジョブ実行タイムアウトと再試行の設定」を参照してください。

    - ロールアウト設定
    - スケジューリング設定
    - ジョブ実行タイムアウトの設定
    - ジョブ実行再試行設定
    - 中止設定

16. 選択したジョブを確認して、[送信] を選択します。

ジョブを作成した後、コンソールにより JSON 署名が生成され、ジョブドキュメントに入力されます。 AWS IoT コンソールを使用して、ジョブのステータスを表示したり、ジョブをキャンセルまたは削除したりできます。ジョブを管理するには、コンソールの [Job hub] (ジョブハブ) に移動します。

パッケージバージョンを AWS IoT モノに関連付ける

デバイスにソフトウェアをインストールしたら、パッケージバージョンを AWS IoT モノの予約済みの名前付きシャドウに関連付けることができます。 AWS IoT ジョブがデプロイされて正常に完了した後に、モノの予約済み名前付きシャドウを更新するようにジョブが設定されている場合は、この手順を完了する必要はありません。詳細については、「予約済みの名前付きシャドウ」を参照してください。

前提条件:

開始する前に、以下を実行します。

- AWS IoT モノを作成し、テレメトリを確立します AWS IoT Core。詳細については、「 の開始方法 AWS IoT Core」を参照してください。
- ソフトウェアパッケージとパッケージバージョンを作成します。詳細については、「ソフトウェアパッケージとパッケージバージョンの作成」を参照してください。
- パッケージバージョンソフトウェアをデバイスにインストールします。

> **Note**
>
> パッケージバージョンを AWS IoT モノに関連付けても、物理デバイスにソフトウェアは更新またはインストールされません。パッケージバージョンはデバイスにデプロイする必要があります。

パッケージバージョンを AWS IoT モノに関連付けるには

1. AWS IoT コンソールのナビゲーションペインで、[すべてのデバイス] メニューを展開して [モノ] を選択します。
2. リストから更新する AWS IoT モノを特定し、モノの名前を選択して詳細ページを表示します。
3. [詳細] セクションで、[パッケージとバージョン] を選択します。
4. [パッケージとバージョンに追加] を選択します。
5. [デバイスパッケージの選択] で、必要なソフトウェアパッケージを選択します。
6. [バージョンを選択] で、必要なソフトウェアバージョンを選択します。
7. [デバイスパッケージの追加] を選択します。

   パッケージとバージョンが [選択したパッケージとバージョン] リストに表示されます。

8. このモノに関連付けるパッケージとバージョンごとに上記の手順を繰り返します。
9. 完了したら、[パッケージとバージョンの詳細を追加] を選択します。[モノの詳細] ページが開き、新しいパッケージとバージョンがリストに表示されます。

AWS IoT ジョブ

AWS IoT ジョブを使用して、接続されている 1 つ以上のデバイスに送信および実行できる一連のリモートオペレーションを定義します AWS IoT。例えば、一連のデバイスに対して、アプリケーションのダウンロードとインストール、ファームウェア更新の実行、再起動、証明書のローテーション、またはリモートトラブルシューティングオペレーションの実行を指示するジョブを定義できます。

AWS IoT ジョブへのアクセス

AWS IoT ジョブの使用を開始するには、 コンソールまたは AWS IoT Core API を使用します。

コンソールを使用する

にサインインし AWS マネジメントコンソール、 AWS IoT コンソールに移動します。ナビゲーションペインで、[管理]、[ジョブ] の順に選択します。このセクションからジョブを作成し、管理できます。ジョブテンプレートを作成および管理するには、ナビゲーションペインで、[Job templates] (ジョブテンプレート) を選択します。詳細については、「を使用してジョブを作成および管理する AWS マネジメントコンソール」を参照してください。

API または CLI の使用

AWS IoT Core API オペレーションを使用して開始できます。詳細については、「AWS IoT API リファレンス」を参照してください。ジョブが構築されている AWS IoT Core AWS IoT API は AWS SDK でサポートされています。詳細については、「AWS SDK とツールキット」を参照してください。

を使用して AWS CLI 、ジョブとジョブテンプレートを作成および管理するためのコマンドを実行できます。詳細については、「AWS IoT CLI リファレンス」を参照してください。

AWS IoT ジョブのリージョンとエンドポイント

AWS IoT ジョブは、 に固有のコントロールプレーンとデータプレーン API エンドポイントをサポートします AWS リージョン。データプレーン API エンドポイントは、 AWS アカウント と に固有です AWS リージョン。 AWS IoT ジョブエンドポイントの詳細については、AWS 「 全般のリファレンス」の「 AWS IoT Device Management - jobs data endpoints」を参照してください。

リモートオペレーションとは

> **Note**
>
> 無線通信 (OTA) の更新は、以下のリージョンでは利用できません。
>
> - 欧州 (スペイン)
> - アジアパシフィック (マレーシア)

リモートオペレーションは、オペレーターや技術者の物理的な存在を必要とせずにリモートで実行できる、物理デバイス、仮想デバイス、エンドポイントで実行できる更新やアクションです。リモートオペレーションは無線通信 (OTA) 更新を使用して実行されるため、デバイスが物理的に存在する必要はありません。でデバイスフリートを管理する AWS クラウド と、登録時にデバイスに対してリモートオペレーションを実行できます AWS IoT Core。

AWS IoT Device Management ジョブは、 に登録されているデバイスでリモートアクションを実行するためのスケーラブルなアプローチを提供します AWS IoT Core。ジョブは で作成 AWS クラウドされ、MQTT または HTTP プロトコルを介して OTA 更新を使用してすべてのターゲットデバイスにプッシュアウトされます。

AWS IoT Device Management ジョブを使用すると、工場出荷時のリセット、デバイスの再起動、ソフトウェアの OTA 更新などのリモートオペレーションを、安全でスケーラブルでコスト効率の高い方法で実行できます。

詳細については AWS IoT Core、「」を参照してくださいとは AWS IoT。

AWS IoT Device Management ジョブの詳細については、「AWS IoT ジョブとは」を参照してください。

リモートオペレーションに AWS IoT Device Management ジョブを使用する利点

AWS IoT Device Management ジョブを使用してリモートオペレーションを実行すると、デバイスフリートの管理が効率化されます。次のリストでは、 AWS IoT Device Management ジョブを使用してリモートオペレーションを実行する主な利点をいくつか示します。

- 他の AWS のサービスとのシームレスな統合

- AWS IoT Device Management ジョブは、以下の付加価値のある AWS のサービス および機能と緊密に統合されます。
  - Amazon S3: リモートオペレーションの指示は、そのコンテンツのアクセス許可を制御する安全な Amazon S3 バケットに保存します。Amazon S3 バケットを使用すると、スケーラブルで耐久性のあるストレージソリューションが提供され、 AWS IoT Device Management Software Package Catalog とネイティブに統合されるため AWS IoT Device Management 、ジョブは更新手順を参照して置き換えることができます。詳細については、「Amazon S3 とは」を参照してください。
  - Amazon CloudWatch: 他のデバイスアクティビティに加えて、各デバイスのジョブ実行のリモートオペレーション実装ステータスをモニタリングしてログに記録し、 AWS IoT Device Management ジョブの全体的なジョブパフォーマンスを追跡および分析します。詳細については、「Amazon CloudWatch とは」を参照してください。ジョブログのモニタリングとトラブルシューティングのための履歴データの取得。ジョブと連携する方法。
  - AWS IoT Device Shadow サービス: AWS IoT Device Management ジョブを使用してデバイスシャドウを介して AWS IoT モノのデジタル表現を維持し、デバイスの接続に関係なく、デバイスの状態をアプリケーションやその他のサービスで利用できるようにします。詳細については、「AWS IoT Device Shadow サービス」を参照してください。
- セキュリティベストプラクティス
  - アクセス許可コントロール: Amazon S3 を使用してリモート操作指示へのアクセス許可を制御し、 AWS IoT ポリシーと IAM ユーザーロールを使用してリモート操作指示をデバイスフリートにデプロイできる IAM ユーザーを決定します。
    - AWS IoT ポリシーの詳細については、「」を参照してくださいAWS IoT ポリシーを作成する。
    - IAM ユーザーロールの詳細については、「の ID とアクセスの管理 AWS IoT」を参照してください。
- スケーラビリティ
  - ターゲットジョブのデプロイ: ジョブの作成時にジョブドキュメントに入力された特定のデバイスグループ化基準を使用して、ターゲットジョブのデプロイのあるジョブからジョブドキュメントを受信するデバイスを制御します。デバイスごとに AWS IoT モノを作成し、その情報を AWS IoT レジストリに保存することで、フリートインデックス作成を使用してターゲットを絞った検索を実行できます。フリートのインデックス作成の検索結果に基づいてカスタムグループを作成して、ターゲットジョブのデプロイをサポートできます。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。ジョブを使用してスナップショットジョブと連続ジョブを実行します。

  - ジョブのステータス: 各デバイスのジョブドキュメントの個々の実装ステータスに加えて、デバイスフリートへのジョブドキュメントのロールアウトのステータスと全体的なジョブステータスをデバイスフリートレベルから追跡します。詳細については、「Jobs とジョブ実行の状態」を参照してください。
  - 新しいデバイスのスケーラビリティ: 継続的なジョブを介したフリートのインデックス作成を使用して作成された既存のカスタムグループに追加することで、ジョブドキュメントを新しいデバイスに簡単にデプロイできます。これにより、ジョブドキュメントを新しいデバイスに個別にデプロイする必要がなくなります。または、ジョブドキュメントを所定のデバイスグループにデプロイしてからジョブが完了すると、スナップショットのショットで、よりターゲットを絞ったアプローチが可能になります。
- 柔軟性
  - ジョブ設定: 特定のニーズに合わせて、オプションのジョブ設定のロールアウト、スケジューリング、中止、タイムアウト、再試行を使用して、ジョブとジョブドキュメントをカスタマイズします。詳細については、「ジョブの 設定」を参照してください。
- コスト効率が良い
  - AWS IoT Device Management ジョブを活用して重要な更新をデプロイし、定期的なメンテナンスタスクを実行することで、デバイスフリートを維持するためのより効率的なコスト構造を導入します。デバイスフリートを維持するための Do It Yourself (DIY) ソリューションには、DIY ソリューションのホストと管理に必要なインフラストラクチャ、DIY ソリューションの開発、維持、スケーリングにかかる労働コスト、データ転送コストなど、定期的に発生する変動コストが含まれます。 AWS IoT Device Management ジョブの透過的で固定コスト構造を活用することで、デバイスフリートへのジョブドキュメントのロールアウトを容易にし、各デバイスのジョブ実行ステータスを追跡するために必要なデータ転送コストに加えて、デバイスの各ジョブ実行のコストを正確に把握できます。詳細については、AWS IoT Core 料金表を参照してください。

AWS IoT ジョブとは

AWS IoT ジョブを使用して、接続されている 1 つ以上のデバイスに送信および実行できる一連のリモートオペレーションを定義します AWS IoT。

ジョブを作成するには、まず、デバイスがリモートで実行する必要があるオペレーションに関する指示のリストを含むジョブドキュメントを定義します。これらのオペレーションを実行するには、ターゲットのリストを指定します。ターゲットとは、個々のモノか、モノのグループか、またはその両方です。ジョブドキュメントとターゲットが一緒になって、デプロイを構成します。

各デプロイには、次の設定を追加できます。

- ロールアウト: この設定では、毎分ジョブドキュメントを受信するデバイスの数を定義します。
- 中止: 特定の数のデバイスがジョブ通知を受信しない場合、この設定を使用してジョブをキャンセルします。これにより、フリート全体に不適切なアップデートが送信されるのを防ぐことができます。
- タイムアウト: 特定の期間内にジョブターゲットから応答が受信されない場合、ジョブが失敗する可能性があります。これらのデバイスで実行されているジョブを追跡できます。
- 再試行: デバイスが失敗またはジョブのタイムアウトを報告した場合は、 AWS IoT ジョブを使用してジョブドキュメントをデバイスに自動的に再送信できます。
- スケジューリング: この設定では、将来の日付と時間にジョブをスケジューリングできます。また、事前に定義したトラフィックの少ない期間にデバイスを更新する定期的なメンテナンスウィンドウを作成することもできます。

AWS IoT ジョブは、ジョブが利用可能であることをターゲットに通知するメッセージを送信します。ターゲットは、ジョブドキュメントをダウンロードし、指定したオペレーションを実行し、その進行状況を報告することで、ジョブの実行を開始します AWS IoT。Jobs によって提供されるコマンドを実行することで、特定のターゲットまたはすべてのターゲットの AWS IoT ジョブの進行状況を追跡できます。ジョブが開始されると、ステータスは、[In progress] (進行中) になります。その後、デバイスは、ジョブが成功、失敗、またはタイムアウトするまで、このステータスを表示しながら変化分の更新を報告します。

次のトピックでは、ジョブの主要な概念と、ジョブとジョブ実行のライフサイクルについて説明します。

トピック

- ジョブの主要な概念
- Jobs とジョブ実行の状態

ジョブの主要な概念

以下の概念では、 AWS IoT ジョブの詳細と、 デバイスでリモートオペレーションを実行するジョブを作成してデプロイする方法を説明します。

基本概念

AWS IoT ジョブを使用する際に知っておくべき基本概念を次に示します。

ジョブ

ジョブは、 AWS IoTに接続された 1 つ以上のデバイスに送信され実行されるリモート操作です。例えば、一連のデバイスに対して、アプリケーションのダウンロードとインストール、ファームウェア更新の実行、再起動、証明書のローテーション、またはリモートトラブルシューティングオペレーションの実行を指示するジョブを定義できます。

ジョブドキュメント

ジョブを作成するには、デバイスによって実行されるリモートオペレーションの説明であるジョブドキュメントを最初に作成する必要があります。

ジョブドキュメントは、UTF-8 でエンコードされた JSON ドキュメントであり、デバイスがジョブを実行するために必要な情報を含みます。ジョブドキュメントには、デバイスが更新やその他のデータをダウンロードできる URL が 1 つ以上含まれています。ジョブドキュメントは、Amazon S3 バケットに格納することも、ジョブを作成するコマンドにインラインで含めることもできます。

> **Note**
>
> MQTT レスポンスでは、`jobDocument` フィールドは JSON オブジェクトです。HTTP レスポンスの場合は、JSON オブジェクトの文字列表現です。

ターゲット

ジョブを作成するとき、オペレーションを実行する必要があるデバイスであるターゲットのリストを指定します。ターゲットはモノ、モノのグループ、またはその両方にすることができます。AWS IoT Jobs サービスは各ターゲットにメッセージを送信し、ジョブが利用可能であることを通知します。

デプロイメント

ジョブドキュメントを指定し、ターゲットのリストを指定してジョブを作成した後、ジョブドキュメントは、更新を実行するリモートターゲットデバイスにデプロイされます。スナップショットジョブでは、ジョブは、ターゲットデバイスにデプロイした後に完了します。連続ジョブでは、ジョブは、グループに追加されたデバイスのグループにデプロイされます。

ジョブの実行

ジョブの実行は、ターゲットデバイスでのジョブのインスタンスです。ターゲットは、ジョブのドキュメントをダウンロードしてジョブの実行を開始します。次に、ドキュメントで指定され

たオペレーションを実行し、その進行状況を に報告します AWS IoT。実行番号は、特定のターゲットでのジョブ実行の一意の識別子です。 AWS IoT Jobs サービスは、ターゲットでのジョブ実行の進行状況と、すべてのターゲットでのジョブの進行状況を追跡するコマンドを提供します。

ジョブタイプの概念

以下の概念は、 ジョブで作成できるさまざまなタイプの AWS IoT ジョブについて理解を深めるのに役立ちます。

スナップショットジョブ

デフォルトでは、ジョブの作成時に指定したすべてのターゲットにジョブが送信されます。それらのターゲットがジョブを完了した後 (または完了できないと報告した場合)、ジョブは完了となります。

連続ジョブ

ジョブの作成時に指定したすべてのターゲットにジョブが送信されます。引き続き実行され、ターゲットグループに追加された新しいデバイス (モノなど) に送信されます。例えば、連続ジョブを使用して、グループに追加されたデバイスをオンボードまたはアップグレードできます。ジョブを作成するときにオプションのパラメータを設定することにより、連続ジョブを作成することができます。

> **Note**
>
> モノの動的グループを使用して IoT フリートをターゲットにする場合、スナップショットジョブではなく、連続ジョブを使用することをお勧めします。連続ジョブを使用すると、グループに参加しているデバイスは、ジョブが作成された後でもジョブの実行を受け取ります。

署名付き URL

ジョブドキュメントに含まれないデータへのアクセスのセキュリティを確保し、時間制限を付けるには、署名付き Amazon S3 URL を使用できます。データを Amazon S3 バケットに配置し、ジョブドキュメント内のデータにプレースホルダーリンクを追加することができます。 AWS IoT ジョブは、ジョブドキュメントのリクエストを受け取ると、プレースホルダーリンクを検索してジョブドキュメントを解析し、リンクを署名付き Amazon S3 URL URLs。

プレースホルダーリンクの形式は次のとおりです。

```
${aws:iot:s3-presigned-url:https://s3.amazonaws.com/bucket/key}
```

ここで、#### はバケット名で、##はリンク先のバケット内のオブジェクトです。

北京および寧夏リージョンでは、署名付き URL は、リソース所有者が ICP (Internet Content Provider) ライセンスを持っている場合にのみ機能します。詳細については、「Getting Started with AWS Services in China」ドキュメントの「Amazon Simple Storage Service」を参照してください。

ジョブ設定の概念

次の概念は、ジョブの設定方法を理解するのに役立ちます。

ロールアウト

保留中のジョブの実行がターゲットに通知される速度を指定できます。これにより、ステージングされたロールアウトを作成し、更新、再起動、その他のオペレーションをよりよく管理できます。静的ロールアウトレートまたは指数関数的ロールアウトレートを使用して、ロールアウト設定を作成できます。1 分あたりに通知するジョブターゲットの最大数を指定するには、静的ロールアウトレートを使用します。

ロールアウトレート設定の例とジョブロールアウトの設定の詳細については、「ジョブのロールアウト、スケジュール、中止の設定」を参照してください。

スケジューリング

ジョブのスケジューリングにより、ターゲットグループ内のすべてのデバイスへのジョブドキュメントのロールアウト期間を、連続ジョブとスナップショットジョブ用にスケジューリングできます。さらに、オプションのメンテナンスウィンドウを作成して、ジョブがターゲットグループ内のすべてのデバイスへのジョブドキュメントのロールアウトを行う特定の日付と時刻を含めることができます。メンテナンスウィンドウとは、最初のジョブあるいはジョブテンプレートの作成時に選択した、日次、週次、月次、またはカスタムの日付と時刻の頻度で繰り返されるインスタンスです。メンテナンスウィンドウ中にロールアウトを実行するようにスケジュールできるのは、連続ジョブのみです。

ジョブスケジューリングはジョブによって異なります。個々のジョブの実行をスケジューリングすることはできません。詳細については、「ジョブのロールアウト、スケジュール、中止の設定」を参照してください。

中止

指定した基準が満たされたときにロールアウトを中止する一連の条件を作成できます。詳細については、「ジョブのロールアウト、スケジュール、中止の設定」を参照してください。

タイムアウト

ジョブタイムアウトは、予期せず長時間ジョブのデプロイが IN_PROGRESS 状態になったときに通知します。タイマーには、進捗タイマーとステップタイマーの 2 種類があります。ジョブが IN_PROGRESS のとき、ジョブデプロイの進行状況をモニタリングして追跡できます。

ロールアウトおよび中止の設定はジョブに固有ですが、タイムアウト設定はジョブのデプロイに固有です。詳細については、「ジョブ実行タイムアウト設定と再試行の設定」を参照してください。

再試行

ジョブ再試行により、ジョブが失敗したり、タイムアウトになったり、その両方になった場合、ジョブの実行を再試行できます。ジョブの実行は、最大 10 回まで再試行できます。再試行の進行状況、およびジョブ実行が成功したかどうかをモニタリングして追跡できます。

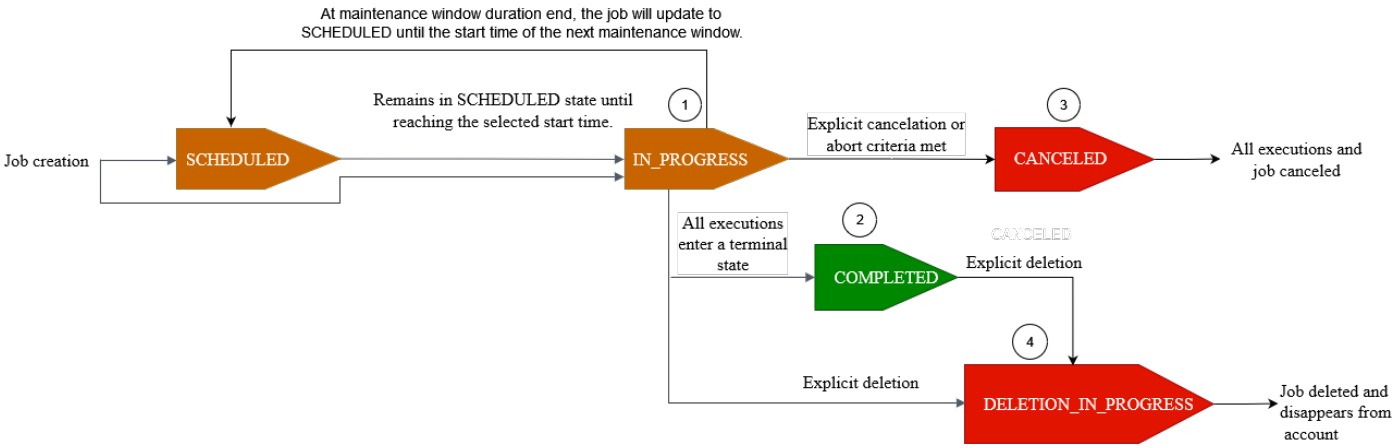
ロールアウトおよび中止の設定はジョブに固有ですが、タイムアウト設定と再試行設定はジョブの実行に固有です。詳細については、「ジョブ実行タイムアウト設定と再試行の設定」を参照してください。

Jobs とジョブ実行の状態

以下のセクションでは、 AWS IoT ジョブのライフサイクルとジョブ実行のライフサイクルについて説明します。

ジョブの状態

次の図は、 AWS IoT ジョブのさまざまな状態を示しています。



ジョブを使用して作成する AWS IoT ジョブは、次のいずれかの状態になります。

- SCHEDULED

  コンソール、CreateJob API、または CreateJobTemplate API を使用した AWS IoT 最初のジョブまたはジョブテンプレートの作成中に、 AWS IoT コンソールまたは CreateJob API または CreateJobTemplate API の `SchedulingConfig` でオプションのスケジューリング設定を選択できます。特定の `startTime`、`endTime`、`endBehavior` を含むスケジューリングされたジョブを開始すると、ジョブのステータスは `SCHEDULED` に更新されます。ジョブが選択した `startTime` または次のメインテナンスウィンドウの `startTime` に到達すると (メンテナンスウィンドウ中にジョブのロールアウトを選択した場合)、ステータスが `SCHEDULED` から `IN_PROGRESS` に更新され、ターゲットグループ内のすべてのデバイスへのジョブドキュメントのロールアウトが開始されます。

- IN_PROGRESS

  AWS IoT コンソールまたは CreateJob API を使用してジョブを作成すると、ジョブのステータスは に更新されます`IN_PROGRESS`。ジョブ作成中、 AWS IoT ジョブは、ターゲットグループ内のデバイスへのジョブ実行のロールアウトを開始します。すべてのジョブ実行がロールアウトされたら、 AWS IoT Jobs は、デバイスがリモートアクションを完了するまで待機します。

  進行中のジョブに適用される同時実行性と制限については、「AWS IoT ジョブの制限」を参照してください。

> **Note**
>
> IN_PROGRESS ジョブが現在のメンテナンスウィンドウの終わりに達すると、ジョブドキュメントのロールアウトは停止します。ジョブは次のメンテナンスウィンドウの startTime まで SCHEDULED に更新されます。

- COMPLETED

  連続ジョブは、以下のいずれかの方法で処理されます。

  - オプションのスケジュール設定が選択されていない連続ジョブは、常に実行され、ターゲットグループに追加された新しいデバイスに対して引き続き実行されます。ステータスが COMPLETED になることはありません。
  - オプションのスケジュール設定が選択されている連続ジョブの場合、次のことが当てはまります。
    - endTime が指定されている場合、連続ジョブは、endTime が経過し、すべてのジョブ実行が終了ステータスに達した時点で COMPLETED ステータスになります。
    - オプションのスケジュール設定で endTime が指定されていない場合、連続ジョブは引き続きジョブドキュメントのロールアウトを実行します。

  スナップショットジョブの場合、すべてのジョブ実行が SUCCEEDED、FAILED、TIMED_OUT、REMOVED、または CANCELED などの終了状態になると、ジョブステータスは COMPLETED に変わります。

- CANCELED

  AWS IoT コンソール、CancelJob API、または を使用してジョブをキャンセルすると ジョブの中止設定、ジョブのステータスは に変わりますCANCELED。ジョブのキャンセル中、 AWS IoT Jobs は以前に作成したジョブ実行のキャンセルを開始します。

  キャンセルされるジョブに適用される同時実行性と制限については、「AWS IoT ジョブの制限」を参照してください。

- DELETION_IN_PROGRESS

  AWS IoT コンソールまたは DeleteJob API を使用してジョブを削除すると、ジョブのステータスは に変わりますDELETION_IN_PROGRESS。ジョブの削除中、 AWS IoT Jobs は以前に作成したジョブ実行の削除を開始します。すべてのジョブ実行が削除されると、ジョブは AWS アカウントから消えます。

Job 実行の状態

次の表は、 AWS IoT ジョブ実行のさまざまな状態と、状態の変更がデバイスによって開始されるか、 AWS IoT ジョブによって開始されるかを示しています。

Job 実行の状態とソース

Job 実行の状態	デバイスによって開始されましたか?	AWS IoT ジョブによって開始されましたか?	ターミナルステータス?	再試行できますか。
QUEUED	いいえ	あり	なし	該当しない
IN_PROGRESS	はい	なし	いいえ	該当しない
SUCCEEDED	はい	なし	はい	該当しない
FAILED	はい	なし	はい	あり
TIMED_OUT	なし	はい	はい	はい
REJECTED	あり	なし	あり	なし
REMOVED	なし	はい	あり	なし
CANCELED	なし	はい	あり	なし

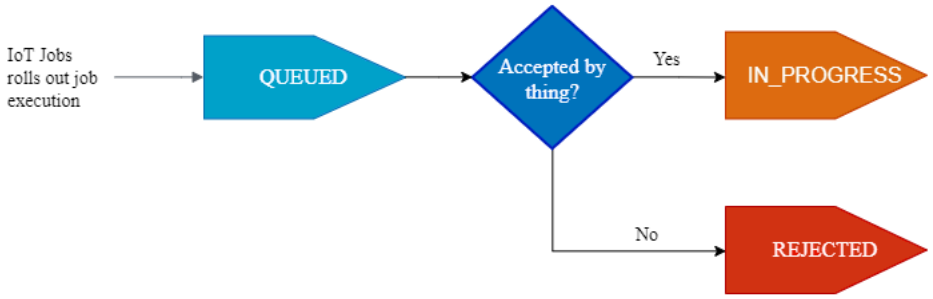
次のセクションでは、ジョブを使用してジョブを作成するときにロールアウトされるジョブ実行の状態について詳しく説明します AWS IoT 。

- QUEUED

  AWS IoT Jobs がターゲットデバイスのジョブ実行をロールアウトすると、ジョブ実行ステータスは に設定されますQUEUED。ジョブ実行は、以下を行うまで QUEUED 状態のままになります。

  - デバイスでジョブの実行を受け取り、ジョブ API オペレーションを呼び出し、ステータスを IN_PROGRESS とレポートします。
  - ジョブまたはジョブ実行をキャンセルするか、指定した中止基準を満たした場合、ステータスが CANCELED に変わります。

- デバイスがターゲットグループから削除され、ステータスが REMOVED に変わります。



- IN_PROGRESS

  IoT デバイスがリザーブド ジョブのトピック $notify と をサブスクライブし$notify-next、デバイスが StartNextPendingJobExecution API または ステータスの UpdateJobExecution API を呼び出す場合IN_PROGRESS、 AWS IoT ジョブはジョブ実行ステータスを に設定しますIN_PROGRESS。

  UpdateJobExecution API は、ステータス IN_PROGRESS で複数回呼び出すことができます。statusDetails オブジェクトを使用して、実行ステップに関する追加の詳細を指定できます。

  > **Note**
  >
  > デバイスごとに複数のジョブを作成する場合、 AWS IoT ジョブと MQTT プロトコルは配信順序を保証しません。

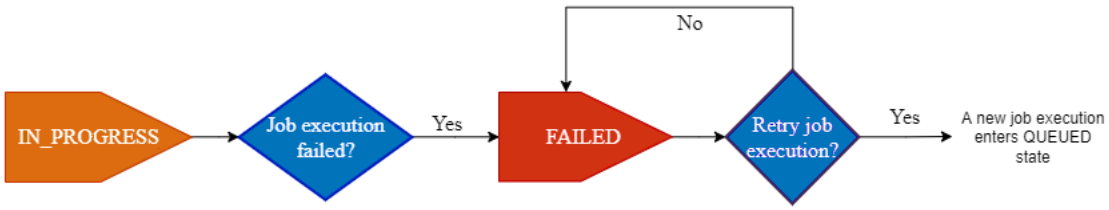
- 成功

  デバイスがリモートオペレーションを正常に完了すると、デバイスは ステータスの UpdateJobExecution API を呼び出しSUCCEEDEDて、ジョブの実行が成功したことを示す必要があります。 AWS IoT ジョブは を更新し、ジョブの実行ステータスを として返しますSUCCEEDED。



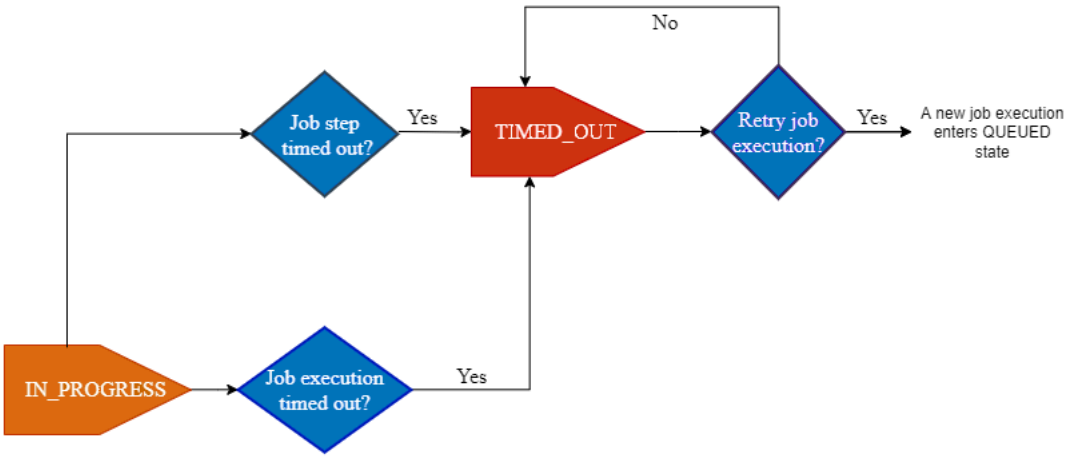
- FAILED

デバイスがリモートオペレーションを完了できなかった場合、デバイスは ステータスの UpdateJobExecution API を呼び出しFailedて、ジョブの実行が失敗したことを示す必要があります。 AWS IoT ジョブは を更新し、ジョブの実行ステータスを として返しますFailed。このジョブの実行は、ジョブ実行再試行設定 を使用してデバイスに対して再試行することができます。



- TIMED_OUT

  ステータスが の場合にデバイスがジョブステップを完了できない場合IN_PROGRESS、または進行中のタイマーのタイムアウト期間内にリモートオペレーションを完了できない場合、 AWS IoT Jobs はジョブ実行ステータスを に設定しますTIMED_OUT。また、進行中のジョブの各ジョブステップにステップタイマーがあり、ジョブの実行にのみ適用されます。進行中のタイマー時間はジョブ実行タイムアウトの設定 の inProgressTimeoutInMinutes プロパティを使用して指定されます。このジョブの実行は、ジョブ実行再試行設定 を使用してデバイスに対して再試行することができます。



- 拒否

  デバイスが無効なリクエストまたは互換性のないリクエストを受信すると、デバイスは ステータスの UpdateJobExecution API を呼び出す必要がありますREJECTED。 AWS IoT ジョブは を更新し、ジョブ実行ステータスを として返しますREJECTED。

- 削除済み

  デバイスが、モノの動的モノグループからデタッチされた場合など、ジョブ実行の有効なターゲットでなくなった場合、 AWS IoT Jobs は、ジョブの実行ステータスを REMOVED に設定します。モノをターゲットグループに再アタッチし、デバイスのジョブ実行を再開できます。

- CANCELED

  コンソール、 または CancelJobExecution API を使用してジョブをキャンセルCancelJobするか、ジョブ実行をキャンセルする場合、または を使用して指定された中止基準が満たされた場合、 AWS IoT Jobs ジョブの中止設定はジョブをキャンセルし、ジョブ実行ステータスを に設定しますCANCELED。

ジョブの管理

ジョブを使用して、ソフトウェアまたはファームウェアの更新をデバイスに通知します。AWS IoT コンソール、ジョブ管理と制御の API オペレーション、AWS Command Line Interface、または AWS SDK を使用して、ジョブを作成し管理することができます。

ジョブのコード署名

デバイスにコードを送信するときに、コードが転送中に変更されたかどうかをデバイスが検出できるように、 AWS CLIを使用してコードファイルに署名することをお勧めします。手順については、「AWS CLIを使用したジョブの作成および管理」を参照してください。

詳細については、「What Is Code Signing for AWS IoT?」を参照してください。

ジョブドキュメント

ョブを作成する前に、ジョブドキュメントを作成する必要があります。コード署名を使用している場合は AWS IoT、バージョニングされた Amazon S3 バケットにジョブドキュメントをアップロードする必要があります。Amazon S3 バケットの作成とファイルのアップロードの詳細については、Amazon S3 開始方法のガイドの Amazon Simple Storage Service の開始方法を参照してください。

> **Tip**
>
> ジョブドキュメントの例については、 AWS IoT SDK for JavaScript の jobs-agent.js の例を参照してください。

署名付き URL

ジョブドキュメントにコードファイル (またはその他のファイル) をポイントする署名付き Amazon S3 URL を含めることができます。署名付き Amazon S3 URL は一定の期間のみ有効であり、デバイスがジョブドキュメントをリクエストしたときに生成されます。ジョブドキュメントを作成するときには署名付き URL が作成されていないため、代わりにプレースホルダー URL をジョブドキュメントで使用します。プレースホルダー URL は次のようになります。

```
${aws:iot:s3-presigned-url-v2:https://
s3.region.amazonaws.com/<bucket>/<code file>}
```

各パラメータの意味は次のとおりです。

- *bucket* は、コードファイルが含まれる Amazon S3 バケットです。
- *code file* は、コードファイルの Amazon S3 キーです。

デバイスがジョブドキュメントをリクエストすると、 は署名付き URL AWS IoT を生成し、プレースホルダー URL を署名付き URL に置き換えます。それからジョブドキュメントは、デバイスに送信されます。

S3 からファイルをダウンロードするアクセス許可を付与する IAM ロール

事前署名済みの Amazon S3 URL を使用するジョブを作成する場合は、IAM ロールを指定する必要があります。ロールは、データまたはアップデートが保管されている Amazon S3 バケットからファイルをダウンロードするアクセス許可を付与する必要があります。ロールは、 AWS IoT がロールを引き受けるための権限も付与する必要があります。

オプションで、署名付き URL のタイムアウトを指定することもできます。詳細については、「CreateJob」を参照してください。

ロールを引き受けるアクセス許可を AWS IoT ジョブに付与する

1. [Roles hub of the IAM console] (IAM コンソールのロールハブ) に移動して、ロールを選択します。
2. [Trust Relationships] (信頼関係) タブで、[Edit Trust Relationship] (信頼関係の編集) を選択し、ポリシードキュメントを以下の JSON に置き換えます。信頼ポリシーの更新 を選択します。

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iot.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. Confused Deputy Problem (混乱した代理の問題) から保護するために、グローバル条件コンテキストキー [aws:SourceArn](#) と [aws:SourceAccount](#) をポリシーに追加します。

> **Important**
>
> aws:SourceArn は、arn:aws:iot:*region*:*account-id*:* の形式に従う必要があります。*#####*がお客様の AWS IoT リージョンと一致し、*account-id* がお客様のアカウント ID と一致していることを確認します。詳細については、[クロスサービスでの混乱した代理処理を防止する](#)を参照してください。

```
{
  "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. ジョブが Amazon S3 オブジェクトであるジョブドキュメントを使用している場合は、[アクセス許可] を選択し、次の JSON を使用してください。これにより、Amazon S3 バケットからファイルをダウンロードするためのアクセス許可を付与するポリシーが追加されます。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your_S3_bucket/*"
        }
    ]
}
```

ファイルアップロード用の署名付き URL

ジョブのデプロイ中にデバイスが Amazon S3 バケットにファイルをアップロードする必要がある場合は、ジョブドキュメントに次の署名付き URL プレースホルダーを含めることができます。

```
${aws:iot:s3-presigned-url-upload:https://s3.region.amazonaws.com/<bucket>/<key>}
```

ジョブドキュメントにあるファイルアップロードプレースホルダー URL の key 属性内で、${thingName}、${jobId}、および ${executionNumber} の各 2 つの予約キーワードを使用できます。key 属性内のこれらの予約キーワードを表すローカルプレースホルダーは、ジョブ実行の作成時に解析され、置き換えられます。各デバイスに固有の予約キーワードを持つローカルプレースホルダーを使用すると、デバイスからアップロードされた各ファイルがそのデバイスに固有であり、同じジョブデプロイの対象となる別のデバイスからアップロードされた同様のファイルによって上書きされないようにします。ジョブのデプロイ中にファイルをアップロードするための署名付き URL プレースホルダー内のローカルプレースホルダーのトラブルシューティングについては、「エラーメッセージの一般的なトラブルシューティング」を参照してください。

> **Note**
>
> Amazon S3 バケット名には、アップロードされたファイルの予約済みドキーワードを表すローカルプレースホルダーを含めることはできません。ローカルプレースホルダーは key 属性に配置する必要があります。

この署名付き URL プレースホルダーは、デバイスがそれを受信すると、ジョブドキュメント内の Amazon S3 署名付きアップロード URL に変換されます。デバイスはこれを使用して、送信先の Amazon S3 バケットにファイルをアップロードします。

> **Note**
>
> Amazon S3 バケットとキーが上記のプレースホルダー URL で指定されていない場合、AWS IoT ジョブは `${thingName}`、`${jobId}`、および `${executionNumber}` のそれぞれの最大 2 つを使用して各デバイスごとにキーを自動的に生成します。

Amazon S3 バージョニングを使用した署名付き URL

Amazon S3 バケットに保存されているファイルの整合性を保護することは、そのファイルを使用してデバイスフリートに安全なジョブデプロイを確保するために不可欠です。Amazon S3 バージョニングを使用すると、Amazon S3 バケットに保存されているファイルの各バリアントのバージョン識別子を追加して、ファイルの各バージョンを追跡できます。これにより、 AWS IoT Jobs を使用してデバイスフリートにデプロイされるファイルのバージョンに関するインサイトが得られます。Amazon S3 バケットでのバージョニングを使用に関する詳細については、「Amazon S3 バケットでのバージョニングの使用」を参照してください。

ファイルが Amazon S3 に保存されており、ジョブドキュメントに署名済み URL プレースホルダーが含まれている場合、 AWS IoT Jobs は Amazon S3 バケット、バケットキー、および Amazon S3 バケットに保存されているファイルのバージョンを使用して、ジョブドキュメントにAmazon S3を生成します。ジョブドキュメントで生成されたこの署名付き URL は、ジョブドキュメントで最初に作成された署名付き URL プレースホルダーに置き換わります。Amazon S3 バケットに保存されているファイルを更新すると、新しいバージョンのファイルとそれ以降の `versionId` が作成され、更新が行われたことを知らせて、将来のジョブデプロイでその特定のファイルをターゲットにする機能が提供されます。

versionId を使用したジョブドキュメントでの Amazon S3 署名付き URL のジョブのデプロイ前とデプロイ中の例については、以下を参照してください。

Amazon S3 署名付き URL プレースホルダー (ジョブデプロイ前)

```
//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 署名付き URL (ジョブデプロイ中)

```
//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 仮想ホストおよびパススタイルのオブジェクト URL の詳細については、「仮想ホスト形式のリクエスト」および「パス形式のリクエスト」を参照してください。

> **Note**
>
> Amazon S3 署名付き URL に versionId を追加する場合は、 AWS SDK for Java 2.x をサポートする URL エンコーディングに準拠する必要があります。詳細については、「Changes in parsing Amazon S3 URIs from version 1 to version 2」を参照してください。

Amazon S3 の署名付き URL プレースホルダーのバージョンの違い

次のリストは、Amazon S3 の署名付き URL プレースホルダー ${aws:iot:s3-presigned-url-v1 (バージョン 1) と ${aws:iot:s3-presigned-url-v2 (バージョン 2) の違いの概要を示しています。

- Amazon S3 の署名付き URL プレースホルダー ${aws:iot:s3-presigned-url-v1 は、version-id をサポートしていません。

- Amazon S3 の署名付き URL プレースホルダー `${aws:iot:s3-presigned-url-v1` は、Amazon S3 URL をエンコードされていないものとして受け取ります。Amazon S3 の署名付き URL プレースホルダー `${aws:iot:s3-presigned-url-v2` では、Amazon S3 SDK 標準に準拠するように Amazon S3 URL をエンコードする必要があります。

トピック

- を使用してジョブを作成および管理する AWS マネジメントコンソール
- を使用してジョブを作成および管理する AWS CLI

を使用してジョブを作成および管理する AWS マネジメントコンソール

このセクションでは、 AWS IoT コンソールからジョブを作成および管理する方法を説明します。ジョブを作成したら、詳細ページでジョブに関する情報を表示し、ジョブを管理できます。

> **Note**
>
> AWS IoT ジョブのコード署名を実行する場合は、 を使用します AWS CLI。詳細については、「Create and manage jobs by using the AWS CLI」を参照してください。

トピック

- を使用して管理ジョブを作成する AWS マネジメントコンソール
- を使用してジョブを表示および管理する AWS マネジメントコンソール

を使用して管理ジョブを作成する AWS マネジメントコンソール

ジョブを作成するには、 AWS IoT コンソールにログインし、リモートアクションセクションのジョブハブに移動します。次に、以下のステップを実行します。

1. [ジョブ] ダイアログボックスの [ジョブ] ページで、[ジョブを作成する] を選択します。
2. 使用しているデバイスに応じて、カスタムジョブ、FreeRTOS OTA 更新ジョブ、または AWS IoT Greengrass ジョブを作成できます。この例では、[Create a custom job] (カスタムジョブの作成) を選択します。[次へ] を選択します。
3. [Custom job properties] (カスタムジョブのプロパティ) ページの [Job properties] (ジョブのプロパティ) ダイアログボックスで、次のフィールドに情報を入力します。

- 名前: ジョブの名前を一意の英数字で入力します。
- 説明 - オプション: ジョブに関する説明を任意で入力します。
- タグ - オプション:

> **Note**
> ジョブ ID または説明に個人を特定できる情報を使用しないようにお勧めします。

[次へ] を選択します。

4. [ジョブターゲット] ダイアログボックスの [ファイル設定] ページで、このジョブを実行する [モノ] または [モノのグループ] を選択します。

   [ジョブドキュメント] ダイアログボックスで、次のいずれかのオプションを選択します。

   - ファイルから: 以前に Amazon S3 バケットにアップロードした JSON ジョブファイル
     - コード署名

       Amazon S3 URL にあるジョブドキュメントでは、コード署名プロファイルを使用して署名付きコードファイルパスに置き換えられるまで、`${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}` がプレースホルダーとして必要です。新しい署名付きコードファイルは、Amazon S3 ソースバケットの `SignedImages` フォルダに最初に表示されます。`Codesigned_` プレフィックスを含む新しいジョブドキュメントが、コード署名プレースホルダーの代わりに署名付きコードファイルパスを使用して作成され、新しいジョブを作成するために Amazon S3 URL に配置されます。
     - 事前署名リソース URL

       [署名付きロール] ドロップダウンで、「署名付き URL」で作成した IAM ロールを選択します。`${aws:iot:s3-presigned-url:` を使用して Amazon S3 にあるオブジェクトの URL を事前署名することは、デバイスが Amazon S3 からオブジェクトをダウンロードする場合のセキュリティのベストプラクティスです。

       コード署名のプレースホルダーとして署名付き URL を使用する場合は、次のサンプルテンプレートを使用してください。

```
${aws:iot:s3-presigned-url:${aws:iot:code-sign-signature:<S3 URL>}
```

- テンプレートから: ジョブドキュメントとジョブ設定を含むジョブテンプレート。ジョブテンプレートは、作成したカスタムジョブテンプレートまたは AWS マネージドテンプレートにすることができます。

  デバイスの再起動など、頻繁に使用するリモートアクションを実行するためのジョブを作成する場合は、 AWS マネージドテンプレートを使用できます。これらのテンプレートには、使用するための事前設定が既に行われています。詳細については、「カスタムジョブテンプレートを作成する」および「管理テンプレートからカスタムジョブテンプレートを作成する」を参照してください。

5. [Job configuration] (ジョブ設定) ダイアログボックスの [Job configuration] (ジョブ設定) ページで、次のいずれかのジョブタイプを選択します。

   - スナップショットジョブ: スナップショットジョブは、ターゲットデバイスおよびグループでの実行が終了すると完了します。
   - 連続ジョブ: 連続ジョブはモノのグループに適用され、指定したターゲットグループに後に追加するあらゆるデバイス上で実行されます。

6. [Additional configurations - optional] (その他の設定 - オプション) ダイアログボックスで、以下のオプションのジョブ設定を確認し、それに応じて選択してください。

   - ロールアウト設定
   - スケジューリング設定
   - ジョブ実行タイムアウトの設定
   - ジョブ実行リトライ設定 - 新規ジョブ
   - 中止設定

   ジョブ設定の詳細については、次のセクションを参照してください。

   - ジョブのロールアウト、スケジュール、中止の設定
   - ジョブ実行タイムアウト設定と再試行の設定

   選択したジョブをすべて確認し、[Submit] (送信) を選択してジョブを作成します。

を使用してジョブを表示および管理する AWS マネジメントコンソール

ジョブを作成した後、コンソールにより JSON 署名が生成され、ジョブドキュメントに入力されます。ステータスを表示、ジョブをキャンセル、削除するには、AWS IoT コンソールを使用できます。

作成したジョブを選択すると、以下が表示されます。

- ジョブ名、説明、タイプ、作成日時、最終更新日時、推定開始時刻などの一般的なジョブの詳細。
- 指定したジョブ設定とそのステータス。
- ジョブのドキュメント。
- 指定したジョブ実行とオプションのタグ。

ジョブを管理するには、[コンソールのジョブハブ] に移動し、ジョブを編集、削除、またはキャンセルするかどうかを選択します。

を使用してジョブを作成および管理する AWS CLI

このセクションでは、ジョブを作成して管理する方法について説明します。

ジョブの作成

AWS IoT ジョブを作成するには、 CreateJob コマンドを使用します。ジョブは、指定したターゲット (モノまたはモノのグループ) で実行キューに登録されます。 AWS IoT ジョブを作成するには、リクエストの本文に含めることができるジョブドキュメント、または Amazon S3 ドキュメントへのリンクとして含めることができるジョブドキュメントが必要です。ジョブに署名付き Amazon S3 URL を使用したファイルのダウンロードが含まれている場合は、ファイルをダウンロードするアクセス許可を持ち、ロールを引き受けるアクセス許可を AWS IoT Jobs サービスに付与する IAM ロール Amazon リソースネーム (ARN) が必要です。 URLs

API コマンドまたは を使用して日付と時刻を入力する際の構文の詳細については AWS CLI、「タイムスタンプ」を参照してください。

コード署名とジョブ

コード署名を使用している場合は AWS IoT、コード署名ジョブを開始し、ジョブドキュメントに出力を含める必要があります。これは、コード署名プロファイルを使用して署名されたコードファイルパスに置き換えるまで、プレースホルダーとして必要とされる、ジョブドキュメント内のコード署名

の署名プレースホルダーを置き換えます。コード署名の署名プレースホルダーは、次のようになります。

```
    ${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}
```

start-signing-job コマンドを使用してコード署名付きジョブを作成します。start-signing-job はジョブ ID を返します。署名が保存されている Amazon S3 の場所を取得するには、describe-signing-job コマンドを使用します。その後、署名は Amazon S3 からダウンロードできます。コード署名付きジョブの詳細については、AWS IoTのコード署名を参照してください。

次のように、ジョブドキュメントにはコードファイルの署名付き URL プレースホルダーおよび start-signing-job コマンドを使用して Amazon S3 バケットに配置された JSON 署名出力が含まれる必要があります。

```
{
    "presign": "${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/bucket/
image}",
}
```

ジョブドキュメントを使用したジョブの作成

次のコマンドは、Amazon S3 バケット (*jobBucket*) に保存されているジョブドキュメント (*job-document.json*) と、Amazon S3 からファイルをダウンロードするためのアクセス許可を持つロール (*S3DownloadRole*) を使用してジョブを作成する方法を示しています。

```
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}"
```

ジョブは *thingOne* で実行されます。

オプションの timeout-config パラメータは、各デバイスがジョブの実行を終了する必要のある時間を指定します。ジョブの実行ステータスが IN_PROGRESS に設定されると、タイマーが開始されます。タイマーが時間切れになるまでにジョブの実行ステータスが別の終了状態に設定されない場合は、TIMED_OUT に設定されます。

進捗タイマーは更新できず、ジョブのすべての実行に適用されます。ジョブの実行がこの間隔より長く IN_PROGRESS状態のままになるたびに失敗し、終了TIMED_OUTステータスに切り替わります。AWS IoT また、 は MQTT 通知を発行します。

ジョブロールアウトと中止に関する設定の作成の詳細については、「ジョブロールアウトと中止設定」を参照してください。

> **Note**
>
> ジョブを作成するときに、Amazon S3 ファイルとして指定されたジョブドキュメントが取得されます。ジョブドキュメントを作成した後でジョブドキュメントのソースとして使用した Amazon S3 ファイルの内容を変更しても、ジョブのターゲットに送信されるものは変更されません。

ジョブの更新

ジョブを更新するには、UpdateJob コマンドを使用します。ジョブの description、presignedUrlConfig、jobExecutionsRolloutConfig、abortConfig および timeoutConfig の各フィールドを更新できます。

```
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}"
```

詳細については、「ジョブのロールアウトと中止設定」を参照してください。

ジョブのキャンセル

ジョブをキャンセルするには、CancelJob コマンドを使用します。ジョブのキャンセルは、ジョブの新しいジョブ実行の AWS IoT ロールアウトを停止します。また、 QUEUED状態のジョブ実行もキャンセルされます。デバイスがすでに AWS IoT ジョブを完了しているため、終了状態のジョブ実行は変更されません。ジョブ実行のステータスが IN_PROGRESS の場合、オプションの --force パラメータを使用しない限り、これも変更されません。

次のコマンドでは、ID が 010 のジョブをキャンセルする方法を示しています。

```
aws iot cancel-job --job-id 010
```

コマンドによって以下の出力が表示されます。

```
{
    "jobArn": "string",
    "jobId": "string",
    "description": "string"
}
```

ジョブをキャンセルすると、QUEUED 状態のジョブ実行がキャンセルされます。IN_PROGRESS 状態のジョブ実行は、オプションの --force パラメータを指定した場合にのみキャンセルされます。終了状態のジョブ実行はキャンセルされません。

> **Warning**
>
> (--force パラメータを設定して) IN_PROGRESS 状態のジョブをキャンセルすると、実行中のすべてのジョブ実行がキャンセルされ、このジョブを実行しているデバイスでジョブ実行ステータスを更新することができなくなります。キャンセルするジョブを実行している各デバイスが有効な状態に必ず戻ることができるように注意してください。

キャンセルされたジョブまたはジョブ実行のいずれかのステータスは、結果整合性があります。は、そのジョブの新しいQUEUEDジョブ実行とジョブ実行のデバイスへのスケジューリングをできるだけ早く AWS IoT 停止します。ジョブの実行ステータスを CANCELED に変更するには、デバイスの数やその他の要因によっては時間がかかることがあります。

AbortConfig オブジェクトによって定義された基準を満たすため、ジョブがキャンセルされた場合、サービスにより comment および reasonCode フィールドに自動的に値が入力されます。カスタマーは、自らジョブをキャンセルしたときの reasonCode に対して独自の値を作成できます。

ジョブ実行をキャンセルする

デバイスでジョブ実行をキャンセルするには、CancelJobExecution コマンドを使用します。QUEUED 状態にあるジョブの実行をキャンセルします。進行中のジョブ実行をキャンセルするには、--force パラメータを使用する必要があります。

次のコマンドでは、myThing で実行されている 010 ジョブのジョブ実行をキャンセルする方法を示しています。

```
aws iot cancel-job-execution --job-id 010 --thing-name myThing
```

コマンドは出力を表示しません。

QUEUED 状態にあるジョブの実行をキャンセルします。IN_PROGRESS 状態のジョブ実行は、オプションの --force パラメータを指定した場合にのみキャンセルされます。終了状態のジョブ実行をキャンセルすることはできません。

> **Warning**
>
> IN_PROGRESS 状態のジョブ実行をキャンセルすると、そのデバイスでジョブ実行ステータスを更新できなくなります。デバイスが有効な状態に必ず戻ることができるように注意してください。

ジョブ実行が終了状態にあるか、ジョブ実行が IN_PROGRESS 状態にあり、--force パラメータが true に設定されていない場合に、このコマンドによって InvalidStateTransitionException になります。

キャンセルされたジョブ実行のステータスは結果整合性があります。ジョブ実行のステータスの CANCELED への変更は、さまざまな要因によっては時間がかかることがあります。

ジョブを削除する

ジョブおよびジョブ実行を削除するには、DeleteJob コマンドを使用します。デフォルトでは、終了状態 (SUCCEEDED または CANCELED) にあるジョブのみが削除できます。他の場合は、例外が発生し

ます。ただし、force パラメータが true に設定されている場合のみ、IN_PROGRESS 状態のジョブを削除できます。

ジョブを削除するには、次のコマンドを実行します。

```
aws iot delete-job --job-id 010 --force|--no-force
```

コマンドは出力を表示しません。

> **Warning**
>
> IN_PROGRESS 状態のジョブを削除すると、ジョブをデプロイしているデバイスはジョブ情報にアクセスすることもジョブ実行ステータスを更新することもできなくなります。削除されたジョブをデプロイしている各デバイスが有効な状態に必ず戻ることができるように注意してください。

ジョブに作成されたジョブ実行の数およびその他の要因に応じて、ジョブの削除には少し時間がかかる場合があります。ジョブの削除中、そのジョブのステータスは DELETION_IN_PROGRESS として表示されます。ステータスが既に DELETION_IN_PROGRESS のジョブを削除あるいはキャンセルしようとすると、エラーになります。

同時に 10 個のジョブのみが DELETION_IN_PROGRESS ステータスになることができます。それ以外の場合は、LimitExceededException が発生します。

ジョブドキュメントの取得

ジョブのジョブドキュメントを取得するには、GetJobDocument コマンドを使用します。ジョブドキュメントは、デバイスによって実行されるリモートオペレーションの説明です。

ジョブドキュメントを取得するには、次のコマンドを実行します。

```
aws iot get-job-document --job-id 010
```

次のコマンドは、指定したジョブのジョブドキュメントを返します。

```
{
    "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**
>
> このコマンドを使用してジョブドキュメントを取得すると、プレースホルダー URL は署名付き Amazon S3 URL に置き換えられません。デバイスが GetPendingJobExecutions API オペレーションを呼び出すときに、プレースホルダー URL は、ジョブドキュメント内の署名付き Amazon S3 URL に置き換えられます。

ジョブの一覧表示

内のすべてのジョブのリストを取得するには AWS アカウント、 ListJobs コマンドを使用します。ジョブデータとジョブ実行データは、 期間限定で保持されます。次のコマンドを実行して、 内のすべてのジョブを一覧表示します AWS アカウント。

```
aws iot list-jobs
```

このコマンドは、ジョブステータスでソートされたアカウント内のすべてのジョブを返します。

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

ジョブの説明

ジョブのステータスを取得するには、DescribeJob コマンドを実行します。次のコマンドでは、ジョブの説明を表示する方法を示しています。

```
$ aws iot describe-job --job-id 010
```

このコマンドは、指定されたジョブのステータスを返します。以下に例を示します。

```
{
    "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
          }
    }
}
```

ジョブ実行リスト

特定のデバイス上で実行されているジョブは、ジョブ実行オブジェクトによって表されます。ジョブのすべてのジョブ実行を一覧表示するには、ListJobExecutionsForJob コマンドを実行します。ジョブの実行をリストする方法を次に示しています。

```
aws iot list-job-executions-for-job --job-id 010
```

このコマンドは、ジョブの実行リストを返します。

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

モノのジョブ実行リスト

モノのジョブ実行をすべて一覧表示するには、ListJobExecutionsForThing コマンドを実行します。モノのジョブの実行をリストする方法を次に示しています。

```
aws iot list-job-executions-for-thing --thing-name thingOne
```

このコマンドは、指定されたモノの実行中または実行済みのジョブの実行リストを返します。

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

ジョブ実行の説明

ジョブ実行のステータスを取得するには、DescribeJobExecution コマンドを実行します。ジョブの実行を識別するために、ジョブ ID とモノの名前、またオプションで実行番号を指定する必要があります。ジョブ実行を記述する方法を次に示しています。

```
aws iot describe-job-execution --job-id 017 --thing-name thingOne
```

コマンドは JobExecution を返します。以下に例を示します。

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

ジョブ実行を削除する

ジョブ実行を削除するには、DeleteJobExecution コマンドを実行します。ジョブの実行を識別するために、ジョブ ID、モノの名前、および実行番号を指定する必要があります。ジョブ実行を削除する方法を次に示しています。

```
aws iot delete-job-execution --job-id 017 --thing-name thingOne --execution-number
 1234567890 --force|--no-force
```

コマンドは出力を表示しません。

デフォルトでは、ジョブ実行のステータスは QUEUED または終了状態 (SUCCEEDED、FAILED、REJECTED、TIMED_OUT、REMOVED または CANCELED) である必要があります。それ以外の場合は、エラーが発生します。IN_PROGRESS 状態のジョブ実行を削除するには、force パラメータを true に設定できます。

> **Warning**
>
> ステータスが IN_PROGRESS のジョブ実行を削除すると、ジョブを実行しているデバイスはジョブ情報にアクセスする、ジョブ実行ステータスを更新することはできません。デバイスが有効な状態に必ず戻ることができるように注意してください。

ジョブテンプレート

ジョブテンプレートを使用すると、ジョブを事前設定して複数のターゲットデバイスセットにデプロイできます。アプリケーションの再起動やインストールなど、頻繁に実行されるリモートアクションをデバイスにデプロイするには、テンプレートを使用して標準設定を定義できます。セキュリティパッチやバグ修正のデプロイなどのオペレーションを行うには、既存のジョブからテンプレートを作成することができます。

ジョブテンプレートを作成する際には、以下の追加の設定やリソースを指定できます。

- ジョブプロパティ
- ジョブドキュメントとターゲット
- ロールアウト、スケジュール、キャンセルの基準
- タイムアウトとリトライの基準

カスタムテンプレートと AWS マネージドテンプレート

実行するリモートアクションに応じて、カスタムジョブテンプレートを作成するか、 AWS マネージドテンプレートを使用できます。カスタムジョブテンプレートを使用して独自のカスタムジョブドキュメントを提供し、デバイスにデプロイする再利用可能なジョブを作成します。 AWS マネージドテンプレートは、一般的に実行されるアクションのために AWS IoT Jobs によって提供されるジョブテンプレートです。これらのテンプレートには、リモートアクション用の事前定義されたジョブドキュメントがあるため、独自のジョブドキュメントを作成する必要はありません。管理テンプレートを使用すると、再利用可能なジョブを作成して、デバイスへの迅速な起動が可能になります。

トピック

- AWS マネージドテンプレートを使用して一般的なリモートオペレーションをデプロイする
- カスタムジョブテンプレートを作成する

AWS マネージドテンプレートを使用して一般的なリモートオペレーションをデプロイする

AWS マネージドテンプレートは、 が提供するジョブテンプレートです AWS。これは、再起動、ファイルのダウンロード、デバイスへのアプリケーションのインストールなど、頻繁に実行されるリモートオペレーションのために使用されます。これらのテンプレートには、リモートアクションごとに事前定義されたジョブドキュメントがあるため、独自のジョブドキュメントを作成する必要はありません。

事前定義された設定のセットから選択し、追加のコードを記述しなくても、これらのテンプレートを使用してジョブを作成できます。管理テンプレートを使用すると、フリートにデプロイされたジョブドキュメントを表示できます。これらのテンプレートを使用してジョブを作成し、リモートオペレーションで再利用できるカスタムジョブテンプレートを作成できます。

管理テンプレートには何が含まれていますか。

各 AWS マネージドテンプレートには、以下が含まれています。

- ジョブドキュメントでコマンドを実行するための環境。
- オペレーションの名前とそのパラメータを指定するジョブドキュメント。例えば、Download file テンプレートの場合、オペレーション名は Download file で、パラメータは次のとおりです。
  - デバイスにダウンロードするファイルの URL。インターネットリソースか、パブリックまたは署名付き Simple Storage Service (Amazon S3) の URL です。
  - ダウンロードしたファイルを保存するデバイス上のローカルファイルパス。

  ジョブドキュメントとそのパラメータの詳細については、「管理テンプレートのリモートアクションとジョブドキュメント」を参照してください。

前提条件

管理テンプレートジョブドキュメントで指定されたリモートアクションをデバイスで実行するには、次の操作を行う必要があります。

- デバイスに特定のソフトウェアをインストールする

  独自のデバイスソフトウェアとジョブハンドラー、または AWS IoT Device Client を使用します。ビジネスケースによっては、両方を実行して異なる機能を実行することもできます。
  - 独自のデバイスソフトウェアとジョブハンドラーを使用する

リモートオペレーションをサポートする AWS IoT Device SDK とそのハンドラーのライブラリを使用して、デバイス用の独自のコードを書き込むことができます。ジョブをデプロイして実行するには、デバイスエージェントライブラリが正しくインストールされ、デバイスで実行されていることを検証します。

リモートオペレーションをサポートする独自のハンドラーを使用することもできます。詳細については、 AWS IoT Device Client GitHub リポジトリの「サンプルジョブハンドラー」を参照してください。

- AWS IoT Device Client を使用する

  または、デバイス上に AWS IoT Device Client をインストールして実行することもできます。これは、デフォルトですべてのマネージドテンプレートをコンソールから直接使用できるためです。

  Device Client は C++ で書かれたオープンソースソフトウェアで、組み込み Linux ベースの IoT デバイスにコンパイルしてインストールできます。Device Client には、ベースクライアントと個別のクライアント側の機能があります。ベースクライアントは MQTT プロトコル AWS IoT 経由で との接続を確立し、さまざまなクライアント側の機能に接続できます。

  デバイスでリモートオペレーションを行うには、Device Client の クライアント側の Jobs 機能を使用します。この機能には、ジョブドキュメントを受信するためのパーサーと、ジョブドキュメントで指定されたリモートアクションを実装するジョブハンドラーが含まれます。Device Client とその機能の詳細については、「AWS IoT Device Client」を参照してください。

  デバイス上で実行すると、Device Client はジョブドキュメントを受信し、ドキュメント内のコマンドを実行するために使用するプラットフォーム固有の実装ができるようになります。Device Client のセットアップおよび Jobs 機能の使用の詳細については、「AWS IoT のチュートリアル」を参照してください。

- サポートされている環境を使用する

  管理テンプレートごとに、リモートアクションの実行に使用できる環境に関する情報が表示されます。テンプレートで指定されているように、サポートされている Linux 環境でテンプレートを使用することをお勧めします。 AWS IoT Device Client を使用してマネージドテンプレートのリモートアクションを実行します。これは、Debian や Ubuntu などの一般的なマイクロプロセッサと Linux 環境をサポートするためです。

管理テンプレートのリモートアクションとジョブドキュメント

次のセクションでは、 AWS IoT ジョブのさまざまな AWS マネージドテンプレートを一覧表示し、デバイスで実行できるリモートアクションについて説明します。次のセクションでは、ジョブドキュメントに関する情報と、各リモートアクションのジョブドキュメントパラメータの説明が記載されています。デバイス側のソフトウェアは、テンプレート名とパラメータを使用してリモートアクションを実行します。

AWS マネージドテンプレートは、テンプレートを使用してジョブを作成するときに値を指定する入力パラメータを受け入れます。すべての管理テンプレートには、共通して次の 2 つのオプションの入力パラメータ (`runAsUser` および `pathToHandler`) があります。`AWS-Reboot` 以外のテンプレートには、追加の入力パラメータが必要です。このパラメータには、テンプレートを使用してジョブを作成するときに値を指定する必要があります。これらの必須入力パラメータは、選択したテンプレートによって異なります。たとえば、`AWS-Download-File` テンプレートを選択する場合、インストールするパッケージのリストと、ファイルのダウンロード元の URL を指定する必要があります。

AWS IoT コンソールまたは AWS Command Line Interface (AWS CLI) を使用してマネージドテンプレートを使用するジョブを作成するときに、入力パラメータの値を指定します。CLI を使用する場合は、`document-parameters` オブジェクトを使用してこれらの値を指定します。詳細については、「documentParameters」を参照してください。

> **Note**
>
> `document-parameters` は、 AWS 管理テンプレートからジョブを作成する場合にのみ使用します。このパラメータは、カスタムジョブテンプレートで使用したり、カスタムジョブテンプレートからジョブを作成したりすることはできません。

次に、一般的なオプションの入力パラメータについて説明します。各管理テンプレートが必要とするその他の入力パラメータについては、次のセクションを参照してください。

`runAsUser`

このパラメータは、ジョブハンドラーを別のユーザーとして実行するかどうかを指定します。ジョブ作成時に指定されていない場合、ジョブハンドラーは Device Client と同じユーザーとして実行されます。ジョブハンドラーを別のユーザーとして実行する場合は、256 文字以下の文字列で値を指定します。

`pathToHandler`

デバイスで実行されているジョブハンドラーへのパス。ジョブの作成時に指定されなかった場合、Device Client は現在の作業ディレクトリを使用します。

以下に、さまざまなリモートアクション、それらのジョブドキュメント、およびそれらが受け入れるパラメータを示します。これらのテンプレートはすべて、デバイス上でリモートオペレーションを実行するための Linux 環境をサポートしています。

AWS–Download–File

[テンプレート名]

`AWS-Download-File`

テンプレートの説明

ファイルをダウンロード AWS するために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータがあります。`runAsUser` および `pathToHandler` のオプションパラメータを指定することもできます。

`downloadUrl`

ファイルのダウンロード元の URL。これは、インターネットリソース、パブリックにアクセスできる Amazon S3 内のオブジェクト、またはお客様のデバイスだけが署名付き URL を使用してアクセスできる Amazon S3 内のオブジェクトです。署名付き URL の使用およびアクセス許可の付与の詳細については、署名付き URL を参照してください。

`filePath`

ダウンロードしたファイルを保存するデバイス内の場所を示すローカルファイルパス。

デバイスの動作

デバイスは、指定された場所からファイルをダウンロードし、ダウンロードが完了したことを確認し、ローカルに保存します。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラへのパスとシェルスクリプト download-file.sh が表示されます。ファイルをダウンロードするには、ジョブハンドラでこのシェルスクリプトを実行する必要があります。また、必須パラメータ downloadUrl および filePath も表示されます。

```
{
  "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

[テンプレート名]

AWS-Install-Application

テンプレートの説明

1 つ以上のアプリケーションをインストール AWS するために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータ packages があります。runAsUser および pathToHandler のオプションパラメータを指定することもできます。

packages

インストールされる 1 つ以上のアプリケーションのスペース区切りのリスト。

デバイスの動作

デバイスは、ジョブドキュメントで指定されたとおりにアプリケーションをインストールします。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト `install-packages.sh` が表示されます。ファイルをダウンロードするには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。また、必須パラメータ `packages` も表示されます。

```
{
  "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

[テンプレート名]

AWS-Reboot

テンプレートの説明

デバイスを再起動 AWS するために が提供する マネージドテンプレート。

入力パラメータ

このテンプレートには必須パラメータはありません。runAsUser および pathToHandler のオプションパラメータを指定できます。

デバイスの動作

デバイスは正常に再起動します。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト reboot.sh が表示されます。デバイスを再起動するには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。

```
{
  "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

[テンプレート名]

AWS-Remove-Application

テンプレートの説明

1 つ以上のアプリケーションをアンインストール AWS するために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータ packages があります。runAsUser および pathToHandler のオプションパラメータを指定することもできます。

packages

アンインストールされる 1 つ以上のアプリケーションのスペース区切りのリスト。

デバイスの動作

デバイスは、ジョブドキュメントで指定されたとおりにアプリケーションをアンインストールします。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト remove-packages.sh が表示されます。ファイルをダウンロードするには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。また、必須パラメータ packages も表示されます。

```
{
  "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

[テンプレート名]

AWS-Restart-Application

テンプレートの説明

1 つ以上のサービスを停止および再起動 AWS するために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータ services があります。runAsUser および pathToHandler のオプションパラメータを指定することもできます。

サービス

再起動される 1 つ以上のアプリケーションのスペース区切りのリスト。

デバイスの動作

指定されたアプリケーションが停止し、その後デバイスで再起動されます。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト restart-services.sh が表示されます。システムサービスを再起動するには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。また、必須パラメータ services も表示されます。

```
{
  "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

[テンプレート名]

AWS-Start-Application

テンプレートの説明

1 つ以上のサービスを開始 AWS するために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータ services があります。runAsUser および pathToHandler のオプションパラメータを指定することもできます。

services

起動される 1 つ以上のアプリケーションのスペース区切りのリスト。

デバイスの動作

指定されたアプリケーションがデバイス上で実行を開始します。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト start-services.sh が表示されます。システムサービスを起動するには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。また、必須パラメータ services も表示されます。

```
{
  "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

[テンプレート名]

AWS-Stop-Application

テンプレートの説明

1 つ以上のサービスを停止 AWS するために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータ services があります。runAsUser および pathToHandler のオプションパラメータを指定することもできます。

services

停止される 1 つ以上のアプリケーションのスペース区切りのリスト。

デバイスの動作

指定されたアプリケーションは、デバイスでの実行を停止します。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト stop-services.sh が表示されます。システムサービスを停止するには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。また、必須パラメータ services も表示されます。

```
{
  "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

[テンプレート名]

AWS-Run-Command

テンプレートの説明

シェルコマンドを実行する AWS ために が提供するマネージドテンプレート。

入力パラメータ

このテンプレートには以下の必須パラメータ command があります。runAsUser のオプションパラメータを指定することもできます。

command

コンマで区切られたコマンドの文字列。コマンド自体に含まれるカンマはすべてエスケープする必要があります。

デバイスの動作

デバイスでは、ジョブドキュメントで指定されたとおりにシェルコマンドを実行します。

ジョブドキュメント

以下は、ジョブドキュメントとその最新バージョンを示しています。テンプレートには、デバイスで実行されるジョブコマンドと、提供されたコマンドへのパスが表示されます。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Run-Command",
        "type": "runCommand",
        "input": {
          "command": "${aws:iot:parameter:command}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

トピック

- を使用して AWS マネージドテンプレートからジョブを作成する AWS マネジメントコンソール
- を使用して AWS マネージドテンプレートからジョブを作成する AWS CLI

を使用して AWS マネージドテンプレートからジョブを作成する AWS マネジメントコンソール

AWS マネジメントコンソール を使用してマネージド AWS テンプレートに関する情報を取得し、これらのテンプレートを使用してジョブを作成します。作成したジョブは、独自のカスタムテンプレートとして保存できます。

管理テンプレートの詳細を取得する

AWS IoT コンソールから使用できるさまざまなマネージドテンプレートに関する情報を取得できます。

1. 使用可能なマネージドテンプレートを表示するには、AWS IoT コンソールのジョブテンプレートハブに移動し、マネージドテンプレートタブを選択します。
2. 詳細を表示するには、管理テンプレートを選択します。

詳細ページには次の情報が含まれています。

- 管理テンプレートの名前、説明、および Amazon リソースネーム (ARN)。

- Linux など、リモートオペレーションを実行できる環境。
- ジョブハンドラーへのパスとデバイスで実行するコマンドを指定する JSON ジョブドキュメント。例えば、次の例は、AWS-Reboot テンプレートのジョブドキュメントです。テンプレートには、ジョブハンドラーへのパスとシェルスクリプト `reboot.sh` が表示されます。デバイスを再起動するには、ジョブハンドラーでこのシェルスクリプトを実行する必要があります。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

  さまざまなリモートアクションのジョブドキュメントおよびそのパラメータの詳細については、管理テンプレートのリモートアクションとジョブドキュメント を参照してください。
- ジョブドキュメントの最新バージョン。

管理テンプレートを使用してジョブを作成する

AWS マネジメントコンソールを使用して、ジョブの作成に使用する AWS マネージドテンプレートを選択できます。このセクションでは、その方法を説明します。

ジョブ作成ワークフローを開始し、ジョブの作成時に使用する AWS マネージドテンプレートを選択することもできます。ワークフローの詳細については、「を使用してジョブを作成および管理する AWS マネジメントコンソール」を参照してください。

1. AWS マネージドテンプレートを選択する

   AWS IoT コンソールのジョブテンプレートハブに移動し、マネージドテンプレートタブを選択し、テンプレートを選択します。

2. 管理テンプレートを使用してジョブを作成する

   1. テンプレートの詳細ページで [Create job] (ジョブを作成) を選択します。

      コンソールは、テンプレート設定が追加されている [Create job] (ジョブを作成) ワークフローの [Custom job properties] (カスタムジョブのプロパティ) のステップに切り替わります。
   2. 一意の英数字のジョブ名、オプションの説明とタグを入力し、[Next] (次へ) をクリックします。
   3. このジョブで実行するジョブターゲットとしてモノまたはモノのグループを選択します。
   4. [Job document] (ジョブドキュメント) セクションで、テンプレートがその設定と入力パラメータとともに表示されます。選択したテンプレートの入力パラメータの値を入力します。例えば、[AWS-Download-File] テンプレートを選択した場合:

      - [downloadUrl] には、ダウンロードするファイルの URL を入力します。例: `https://example.com/index.html`。
      - [filePath] には、ダウンロードしたファイルを保存するデバイス上のパスを入力します。例: *path/to/file*。

      オプションで、パラメータ `runAsUser` および `pathToHandler` の値を入力することもできます。各テンプレートの入力パラメータの詳細については、「管理テンプレートのリモートアクションとジョブドキュメント」を参照してください。
   5. [Job configuration] (ジョブ設定) ページで、ジョブタイプとして連続またはスナップショットジョブを選択します。スナップショットジョブは、ターゲットデバイスおよびグループでの実行が終了すると完了します。連続ジョブはモノのグループに適用され、指定したターゲットグループに追加したいずれかのデバイス上で実行されます。
   6. 引き続きジョブに追加の設定を追加し、確認してジョブを作成します。追加の設定については、以下を参照してください。

      - ジョブのロールアウト、スケジュール、中止の設定
      - ジョブ実行タイムアウト設定と再試行の設定

管理テンプレートからカスタムジョブテンプレートを作成する

AWS マネージドテンプレートとカスタムジョブを出発点として使用して、独自のカスタムジョブテンプレートを作成できます。カスタムジョブテンプレートを作成するには、前のセクションで説明したように、まず AWS マネージドテンプレートからジョブを作成します。

その後、カスタムジョブをテンプレートとして保存して、独自のカスタムジョブテンプレートを作成できます。テンプレートとして保存するには:

1. AWS IoT コンソールのジョブハブに移動し、マネージドテンプレートを含むジョブを選択します。
2. [Save as a job template] (ジョブテンプレートとして保存) を選択し、カスタムジョブテンプレートを作成します。カスタムジョブテンプレートの作成の詳細については、「既存のジョブからジョブテンプレートを作成する」を参照してください。

を使用して AWS マネージドテンプレートからジョブを作成する AWS CLI

AWS CLI を使用してマネージド AWS テンプレートに関する情報を取得し、これらのテンプレートを使用してジョブを作成します。次に、ジョブをテンプレートとして保存し、独自のカスタムテンプレートを作成できます。

管理テンプレートを一覧表示する

`list-managed-job-templates` AWS CLI コマンドは、 内のすべてのジョブテンプレートを一覧表示します AWS アカウント。

```
aws iot list-managed-job-templates
```

デフォルトでは、このコマンドを実行すると、使用可能なすべての AWS マネージドテンプレートとその詳細が表示されます。

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

詳細については、「ListManagedJobTemplates」を参照してください。

管理テンプレートの詳細を取得する

describe-managed-job-template AWS CLI コマンドは、指定されたジョブテンプレートの詳細を取得します。ジョブテンプレート名とオプションのテンプレートバージョンを指定します。テンプレートのバージョンを指定しなかった場合は、定義済みのデフォルトバージョンが返されます。

以下は、コマンドを実行して AWS-Download-File テンプレートの詳細を取得する例を示しています。

```
aws iot describe-managed-job-template \
    --template-name AWS-Download-File
```

このコマンドは、テンプレートの詳細と ARN、そのジョブドキュメント、およびテンプレートの入力パラメータのキーと値のペアのリストである documentParameters パラメータを表示します。さまざまなテンプレートと入力パラメータの詳細については、「管理テンプレートのリモートアクションとジョブドキュメント」を参照してください。

> **Note**
>
> この API の使用時に返されるdocumentParametersオブジェクトは、 AWS マネージドテンプレートからジョブを作成する場合にのみ使用する必要があります。このオブジェクトをカスタムジョブテンプレートに使用しないでください。このパラメータの使用方法を示す例については、「管理テンプレートを使用してジョブを作成する」を参照してください。

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

詳細については、「DescribeManagedJobTemplate」を参照してください。

管理テンプレートを使用してジョブを作成する

create-job AWS CLI コマンドを使用して、ジョブテンプレートからジョブを作成できます。これは、thingOne というデバイスをターゲットとし、ジョブのベースとして使用する管理テンプレートの Amazon リソースネーム (ARN) を指定します。create-job コマンドの関連パラメータを渡すことで、タイムアウトやキャンセルの設定といった詳細設定を上書きできます。

この例では、AWS-Download-File テンプレートを使用するジョブを作成する方法を示しています。また、document-parameters パラメータを使用して、テンプレートの入力パラメータを指定する方法も示します。

> **Note**
>
> document-parameters オブジェクトは、 AWS マネージドテンプレートでのみ使用します。このオブジェクトをカスタムジョブテンプレートに使用しないでください。

```
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
```

各パラメータの意味は次のとおりです。

- *region* は です AWS リージョン。
- *account-id* は、一意の AWS アカウント 番号です。
- *thingOne* は、ジョブの対象となる IoT のモノの名前です。
- *AWS-Download-File:1.0* は、管理テンプレートの名前です。
- https://*example.com/index*.html は、ファイルのダウンロード元の URL です。
- https://*pathto/file/index* は、ダウンロードしたファイルを保存するデバイス上のローカルファイルパスです。

次のコマンドを実行して、テンプレートを作成します。*AWS-Download-File*

```
{
    "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."
}
```

管理テンプレートからカスタムジョブテンプレートを作成する

1. 前のセクションで説明したように、管理テンプレートを使用してジョブを作成します。
2. 作成したジョブの ARN を使用して、カスタムジョブテンプレートを作成します。詳細については、「既存のジョブからジョブテンプレートを作成する」を参照してください。

カスタムジョブテンプレートを作成する

および AWS CLI AWS IoT コンソールを使用してジョブテンプレートを作成できます。、 AWS IoT コンソール AWS CLI、および Fleet Hub for AWS IoT Device Management ウェブアプリケーションを使用して、ジョブテンプレートからジョブを作成することもできます。Fleet Hub アプリケーションでのジョブテンプレートの操作の詳細については、Fleet Hub for AWS IoT Device Management の「ジョブテンプレートの操作」を参照してください。

> **Note**
>
> ジョブドキュメント内の代替パターンの合計数は、10 以下である必要があります。

トピック

- を使用してカスタムジョブテンプレートを作成する AWS マネジメントコンソール
- を使用してカスタムジョブテンプレートを作成する AWS CLI

を使用してカスタムジョブテンプレートを作成する AWS マネジメントコンソール

このトピックでは、 AWS IoT コンソールを使用してジョブテンプレートの詳細を作成、削除、表示する方法について説明します。

カスタムジョブテンプレートを作成する

オリジナルのカスタムジョブテンプレートを作成することも、既存のジョブからジョブテンプレートを作成することもできます。 AWS マネージドテンプレートを使用して作成された既存のジョブからカスタムジョブテンプレートを作成することもできます。詳細については、「管理テンプレートからカスタムジョブテンプレートを作成する」を参照してください。

オリジナルのジョブテンプレートを作成する

1. ジョブテンプレートの作成を開始する

    1. AWS IoT コンソールのジョブテンプレートハブに移動し、カスタムテンプレートタブを選択します。
    2. [Create job template] (ジョブテンプレートの作成) を選択します。

> **Note**
>
> また、[Fleet Hub] の [Related services] (関連サービス) ページから [Job templates] (ジョブテンプレート) に移動することもできます。

2. ジョブテンプレートのプロパティを指定する

   [Create job template] (ジョブテンプレートを作成) ページで、ジョブ名の英数字識別子と英数字の説明を入力して、テンプレートに関する追加情報を指定します。

> **Note**
>
> ジョブ ID または説明に個人を特定できる情報を使用しないようにお勧めします。

3. ジョブドキュメントを提供する

   S3 バケットに保存されている JSON ジョブファイル、またはジョブ内で指定されたインラインジョブドキュメントである JSON ジョブファイルを指定します。このテンプレートを使用してジョブを作成すると、このジョブファイルがジョブドキュメントになります。

   ジョブファイルが S3 バケットに保存されている場合は、S3 の URL を入力するか、[Browse S3] (S3 をブラウズ) をクリックし、ジョブドキュメントに移動して選択します。

> **Note**
>
> 現在のリージョンにある S3 バケットのみ選択できます。

4. 引き続きジョブに追加の設定を追加し、確認してジョブを作成します。その他のオプション設定などの詳細については、次のリンクを参照してください。

   - ジョブのロールアウト、スケジュール、中止の設定
   - ジョブ実行タイムアウト設定と再試行の設定

既存のジョブからジョブテンプレートを作成する

1. ジョブを選択する

   1. AWS IoT コンソールのジョブハブに移動し、ジョブテンプレートの基礎として使用するジョブを選択します。
   2. [Save as a job template] (ジョブテンプレートとして保存) を選択します。

   > **Note**
   >
   > 必要に応じて、別のジョブドキュメントを選択したり、元のジョブから高度な設定を編集したりしてから、[Create job template] (ジョブテンプレートの作成) を選択することもできます。新しいジョブテンプレートが [Job templates] (ジョブテンプレート) ページに表示されます。

2. ジョブテンプレートのプロパティを指定する

   [Create job template] (ジョブテンプレートを作成) ページで、ジョブ名の英数字識別子と英数字の説明を入力して、テンプレートに関する追加情報を指定します。

   > **Note**
   >
   > ジョブドキュメントは、テンプレートの作成時に指定したジョブファイルです。ジョブドキュメントが S3 の場所ではなくジョブ内で指定されている場合、ジョブドキュメントはこのジョブの詳細ページに表示されます。

3. 引き続きジョブに追加の設定を追加し、確認してジョブを作成します。追加の設定については、以下を参照してください。

   - ジョブのロールアウト、スケジュール、中止の設定
   - ジョブ実行タイムアウト設定と再試行の設定

カスタムジョブテンプレートからジョブを作成する

このトピックで説明するように、ジョブテンプレートの詳細ページに移動して、カスタムジョブテンプレートからジョブを作成できます。ジョブを作成することも、ジョブ作成ワークフローの実行時に使用するジョブテンプレートを選択することもできます。詳細については、「を使用してジョブを作成および管理する AWS マネジメントコンソール」を参照してください。

このトピックでは、カスタムジョブテンプレートの詳細ページからジョブを作成する方法について説明します。 AWS マネージドテンプレートからジョブを作成することもできます。詳細については、「管理テンプレートを使用してジョブを作成する」を参照してください。

1. カスタムジョブテンプレートを選択する

   AWS IoT コンソールのジョブテンプレートハブに移動し、カスタムテンプレートタブを選択し、テンプレートを選択します。

2. カスタムテンプレートを使用してジョブを作成する

   ジョブを作成するには:

   1. テンプレートの詳細ページで [Create job] (ジョブを作成) を選択します。

      コンソールは、テンプレート設定が追加されている [Create job] (ジョブを作成) ワークフローの [Custom job properties] (カスタムジョブのプロパティ) のステップに切り替わります。
   2. 一意の英数字のジョブ名、オプションの説明とタグを入力し、[Next] (次へ) をクリックします。
   3. このジョブで実行するジョブターゲットとしてモノまたはモノのグループを選択します。

      [Job document] (ジョブドキュメント) セクションで、テンプレートがその設定とともに表示されます。別のジョブドキュメントを使用する場合は、[Browse] (参照) をクリックし、別のバケットとドキュメントを選択します。[次へ] を選択します。
   4. [Job configuration] (ジョブ設定) ページで、ジョブタイプとして連続またはスナップショットジョブを選択します。スナップショットジョブは、ターゲットデバイスおよびグループでの実行が終了すると完了します。連続ジョブはモノのグループに適用され、指定したターゲットグループに追加したいずれかのデバイス上で実行されます。
   5. 引き続きジョブに追加の設定を追加し、確認してジョブを作成します。追加の設定については、以下を参照してください。
      - ジョブのロールアウト、スケジュール、中止の設定
      - ジョブ実行タイムアウト設定と再試行の設定

> **Note**
>
> ジョブテンプレートから作成されたジョブがジョブテンプレートによって提供される既存のパラメータを更新すると、それらの更新されたパラメータは、そのジョブのジョブテンプレートによって提供される既存のパラメータも上書きします。

Fleet Hub Web アプリケーションを使用して、ジョブテンプレートからジョブを作成することもできます。Fleet Hub でのジョブの作成については、Fleet Hub for AWS IoT Device Management の「ジョブテンプレートの使用」を参照してください。

ジョブテンプレートを削除する

ジョブテンプレートを削除するには、まずAWS IoT コンソールのジョブテンプレートハブに移動し、カスタムテンプレートタブを選択します。削除するジョブテンプレートを選択し、[次へ] をクリックします。

> **Note**
>
> 完全に削除され、ジョブテンプレートは [Custom templates] (カスタムテンプレート) タブに表示されなくなります。

を使用してカスタムジョブテンプレートを作成する AWS CLI

このトピックでは、 AWS CLIを使用してジョブテンプレートの作成、削除、詳細情報の取得を行う方法について説明します。

ーからジョブテンプレートを作成する

次の AWS CLI コマンドは、S3 バケットに保存されているジョブドキュメント (*job-document.json*) と、Amazon S3 (*S3DownloadRole*) からファイルをダウンロードするアクセス許可を持つロールを使用してジョブを作成する方法を示しています。

```
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}"
```

オプションの timeout-config パラメータは、各デバイスがジョブの実行を終了すべき時間を指定します。ジョブの実行ステータスが IN_PROGRESS に設定されると、タイマーが開始されます。タイマーが時間切れになるまでにジョブの実行ステータスが別の終了状態に設定されない場合は、TIMED_OUT に設定されます。

進捗タイマーは更新できず、ジョブのすべての起動に適用されます。ジョブの起動がこの間隔より長く IN_PROGRESS状態のままになるたびに、ジョブの起動は失敗し、終了TIMED_OUTステータスに切り替わります。 AWS IoT も MQTT 通知を発行します。

ジョブロールアウトと中止に関する設定の作成の詳細については、「ジョブのロールアウトと中止設定」を参照してください。

> **Note**
>
> ジョブを作成するときに、Amazon S3 ファイルとして指定されたジョブドキュメントが取得されます。ジョブを作成した後でジョブドキュメントのソースとして使用した Amazon S3 ファイルの内容を変更しても、ジョブのターゲットに送信されるものは変更されません。

既存のジョブからジョブテンプレートを作成する

次の AWS CLI コマンドは、既存のジョブの Amazon リソースネーム (ARN) を指定してジョブテンプレートを作成します。新しいジョブテンプレートでは、ジョブで指定されたすべての設定が使用されます。必要に応じて、任意のオプションのパラメータを使用して既存のジョブの構成を変更できます。

```
aws iot create-job-template  \
      --job-arn arn:aws:iot:region:123456789012:job/job-name  \
```

```
      --timeout-config inProgressTimeoutInMinutes=100
```

ジョブテンプレートの詳細を取得する

次の AWS CLI コマンドは、指定されたジョブテンプレートの詳細を取得します。

```
aws iot describe-job-template \
      --job-template-id template-id
```

コマンドによって以下の出力が表示されます。

```
{
   "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
   }
}
```

ジョブテンプレートを一覧表示する

次の AWS CLI コマンドは、 内のすべてのジョブテンプレートを一覧表示します AWS アカウント。

```
 aws iot list-job-templates
```

コマンドによって以下の出力が表示されます。

```
{
   "jobTemplates": [
      {
         "createdAt": number,
         "description": "string",
         "jobTemplateArn": "string",
         "jobTemplateId": "string"
      }
   ],
   "nextToken": "string"
}
```

結果の追加ページを取得するには、nextToken フィールドの値を使用します。

ジョブテンプレートを削除する

次の AWS CLI コマンドは、指定されたジョブテンプレートを削除します。

```
aws iot delete-job-template \
      --job-template-id template-id
```

コマンドは出力を表示しません。

カスタムジョブテンプレートからジョブを作成する

次の AWS CLI コマンドは、カスタムジョブテンプレートからジョブを作成します。これは、thingOne というデバイスをターゲットとし、ジョブのベースとして使用するジョブテンプレートの Amazon リソースネーム (ARN) を指定します。create-job コマンドの関連パラメータを渡すことで、タイムアウトやキャンセルの設定といった詳細設定を上書きできます。

> **Warning**
>
> document-parameters オブジェクトは、 AWS 管理テンプレートからジョブを作成する場合のみ create-job コマンドで使用する必要があります。このオブジェクトをカスタムジョブテンプレートに使用しないでください。このパラメータを使用してジョブを作成する方法を示す例については、「管理テンプレートを使用してジョブを作成する」を参照してください。

```
aws iot create-job \
      --targets arn:aws:iot:region:123456789012:thing/thingOne  \
      --job-template-arn arn:aws:iot:region:123456789012:jobtemplate/template-id
```

ジョブの 設定

指定したターゲットにデプロイするジョブごとに、次の追加設定ができます。

- ロールアウト: 毎分ジョブドキュメントを受信するデバイスの数を定義します。
- スケジューリング: 定期的なメンテナンスウィンドウを使用する場合に加えて、将来の日時にジョブをスケジューリングします。
- 中止: 一部のデバイスがジョブ通知を受信しない場合や、デバイスがジョブ実行の失敗を報告する場合などに、ジョブをキャンセルします。
- タイムアウト : ジョブの実行の開始後、特定の期間内にジョブターゲットから応答が受信されない場合、ジョブが失敗する可能性があります。
- 再試行: ジョブの実行を完了しようとしたときにデバイスが失敗を報告した、またはジョブの実行がタイムアウトした場合に、ジョブ実行を再試行します。

これらの設定を使用すると、ジョブ実行のステータスをモニタリングし、不正な更新がフリート全体に送信されるのを防ぐことができます。

トピック

- ジョブ設定の仕組み
- 追加の設定を指定する

ジョブ設定の仕組み

ジョブのデプロイ時にロールアウト設定と中止設定を使用し、タイムアウトと再試行設定をジョブの実行に使用します。以下のセクションに、これらの設定の仕組みの詳細を示します。

トピック

- ジョブのロールアウト、スケジュール、中止の設定
- ジョブ実行タイムアウト設定と再試行の設定

ジョブのロールアウト、スケジュール、中止の設定

ジョブのロールアウト、スケジューリング、中止の設定を使用して、ジョブドキュメントを受信するデバイスの数を定義し、ジョブのロールアウトをスケジューリングして、ジョブをキャンセルする基準を決定できます。

ジョブのロールアウト設定

保留中のジョブの実行がターゲットに通知される速度を指定できます。また、ステージングされたロールアウトを作成し、更新、再起動、その他のオペレーションを管理できます。ターゲットの通知方法を指定するには、ジョブのロールアウトレートを使用します。

ジョブロールアウトレート

一定のロールアウトレートまたは指数関数的ロールアウトレートを使用して、ロールアウト設定を作成できます。1 分あたりに通知するジョブターゲットの最大数を指定するには、一定のロールアウトレートを使用します。

AWS IoT ジョブは、さまざまな基準としきい値が満たされると、指数ロールアウトレートを使用してデプロイできます。失敗したジョブの数が、指定した基準のセットと一致する場合、ジョブのロールアウトをキャンセルできます。ジョブのロールアウトレート基準は、ジョブを作成時

に、JobExecutionsRolloutConfig オブジェクトを使用して設定します。ジョブの中止基準も、ジョブの作成時に、AbortConfig オブジェクトを使用して設定します。

次の例は、ロールアウトレートの仕組みを示しています。例えば、基本レートが 50 分あたり、増分係数 2、通知および成功したデバイスの数がそれぞれ 1,000 のジョブロールアウトは、次のように機能します。ジョブは、1 分あたり 50 のジョブ実行の割合で開始され、1,000 個のモノがジョブの実行通知を受信するか、1,000 のジョブ実行が成功するまでその割合で継続されます。

次の表で、最初の 4 つのインクリメントでロールアウトを進める方法を示します。

分あたりのロールアウトレート	50	100	200	400
レートの増加を満たすための通知されたデバイス数または成功したジョブ実行数	1,000	2,000	3,000	4,000

> **Note**
>
> 同時に実行できるジョブの最大数が 500 (isConcurrent = True) の場合、同時実行ジョブの数が 499 以下 (isConcurrent = False) になるまで、すべてのアクティブなジョブのステータスは IN-PROGRESS のままで、新しいジョブは実行されません。これは、スナップショットジョブと連続ジョブに適用されます。
> isConcurrent = True の場合、このジョブによってターゲットグループ内のすべてのデバイスへのジョブ実行をロールアウトしています。isConcurrent = False の場合、ジョブがターゲットグループ内のすべてのデバイスへのジョブ実行のロールアウトを完了しました。ターゲットグループのすべてのデバイスが終了状態になるか、ジョブ停止設定を選択した場合は、ターゲットグループのしきい値 (パーセント) に達すると、ステータスが更新されます。isConcurrent = True および isConcurrent = False のジョブレベルのステータスは両方ともに IN_PROGRESS です。
> アクティブジョブと同時実行ジョブの制限の詳細については、「アクティブなジョブおよび同時ジョブの制限」を参照してください。

モノの動的グループを使用した継続的ジョブのジョブロールアウトレート

連続ジョブを使用してフリートにリモートオペレーションをロールアウトすると、 AWS IoT Jobs はターゲットモノグループ内のデバイスのジョブ実行をロールアウトします。モノの動的グループに新

たに追加されたデバイスについては、ジョブ作成後もこれらのジョブ実行がそれらのデバイスに対して引き続きロールアウトされます。

ロールアウト設定では、ジョブが作成されるまでグループに追加されたデバイスのみ、ロールアウトレートを制御できます。ジョブが作成されると、新しいデバイスについては、デバイスがターゲットグループに参加すると、ジョブ実行がほぼ同時に作成されます。

ジョブスケジューリングの設定

あらかじめ設定された開始時刻、終了時刻、終了時刻に達した際に各ジョブの実行がどのようになるかを示す終了動作を使用して、最大 1 年先まで連続ジョブまたはスナップショットのジョブをスケジューリングできます。さらに、オプションの定期メンテナンスウィンドウを作成して、継続的なジョブの頻度、開始時間、期間を柔軟に設定して、ジョブドキュメントをターゲットグループ内のすべてのデバイスにロールアウトできます。

ジョブスケジューリングの設定

[開始時刻]

スケジューリングされたジョブの開始時刻は、そのジョブがターゲットグループ内のすべてのデバイスへのジョブドキュメントのロールアウトを開始する将来の日付と時刻です。スケジューリングされたジョブの開始時刻は、連続ジョブとスナップショットジョブに適用されます。スケジューリングされたジョブが最初に作成されると、SCHEDULED のステータスは維持されます。選択した `startTime` に達すると、`IN_PROGRESS` に更新され、ジョブドキュメントのロールアウトが開始されます。`startTime` は、スケジューリングされたジョブを最初に作成した日から 1 年以内にする必要があります。

API コマンドまたは `startTime`を使用する場合の の構文の詳細については AWS CLI、「タイムスタンプ」を参照してください。

夏時間 (DST) が適用されている場所での定期的なメンテナンスウィンドウでオプションのスケジュール設定が選択されているジョブの場合は、DST と標準時間を切り替える際に開始時間が 1 時間変わります。

> **Note**
>
> に表示されるタイムゾーン AWS マネジメントコンソール は、現在のシステムのタイムゾーンです。ただし、これらのタイムゾーンはシステムによって UTC に変換されます。

[終了時刻]

スケジューリングされたジョブの終了時刻は、そのジョブがターゲットグループ内に残っているすべてのデバイスへのジョブドキュメントのロールアウトを停止する将来の日付と時刻です。スケジューリングされたジョブの終了時刻は、連続ジョブとスナップショットジョブに適用されます。スケジューリングされたジョブが選択した endTime に達し、すべてのジョブの実行が終了状態になると、ステータスが IN_PROGRESS から COMPLETED に更新されます。endTime は、スケジューリングされたジョブを最初に作成した日から 2 年以内にする必要があります。startTime と endTime との間の最短時間は 30 分です。ジョブ実行の再試行は、ジョブが endTime に到達するまで行われます。その後、endBehavior が処理方法を決定します。

API コマンドまたは endTimeを使用する場合の の構文の詳細については AWS CLI、「タイムスタンプ」を参照してください。

夏時間 (DST) が適用されている場所での定期的なメンテナンスウィンドウでオプションのスケジュール設定が選択されているジョブの場合は、DST と標準時間を切り替える際に開始時間が 1 時間変わります。

> **Note**
>
> に表示されるタイムゾーン AWS マネジメントコンソール は、現在のシステムのタイムゾーンです。ただし、これらのタイムゾーンはシステムによって UTC に変換されます。

終了動作

スケジューリングされたジョブの終了動作によって、ジョブが選択した endTime に到達した際に、そのジョブと未完了のすべてのジョブ実行がどうなるかが決まります。

ジョブまたはジョブテンプレートの作成時に選択できる終了動作は次のとおりです。

- STOP_ROLLOUT
  - STOP_ROLLOUT によって、ターゲットグループ内の残りのすべてのデバイスに対し、ジョブドキュメントのロールアウトを停止します。さらに、すべての QUEUED および IN_PROGRESS のジョブ実行はターミナル状態になるまで継続されます。CANCEL または FORCE_CANCEL を選択しない限り、これがデフォルトの終了動作になります。
- CANCEL
  - CANCEL によって、ターゲットグループ内の残りのすべてのデバイスに対し、ジョブドキュメントのロールアウトを停止します。さらに、すべての QUEUED のジョブ実行はキャンセルされ、すべての IN_PROGRESS のジョブ実行は終了状態になるまで継続されます。

- FORCE_CANCEL
  - FORCE_CANCEL によって、ターゲットグループ内の残りのすべてのデバイスに対し、ジョブドキュメントのロールアウトを停止します。さらに、QUEUED および IN_PROGRESS のジョブ実行はすべてキャンセルされます。

> **Note**
>
> endbehavior を選択するには、endtime を選択する必要があります。

最大継続時間

スケジューリングされたジョブの最大継続時間は、startTime および endTime に関係なく 2 年以下にする必要があります。

スケジューリングされたジョブの一般的な継続時間のシナリオの表を以下に示します。

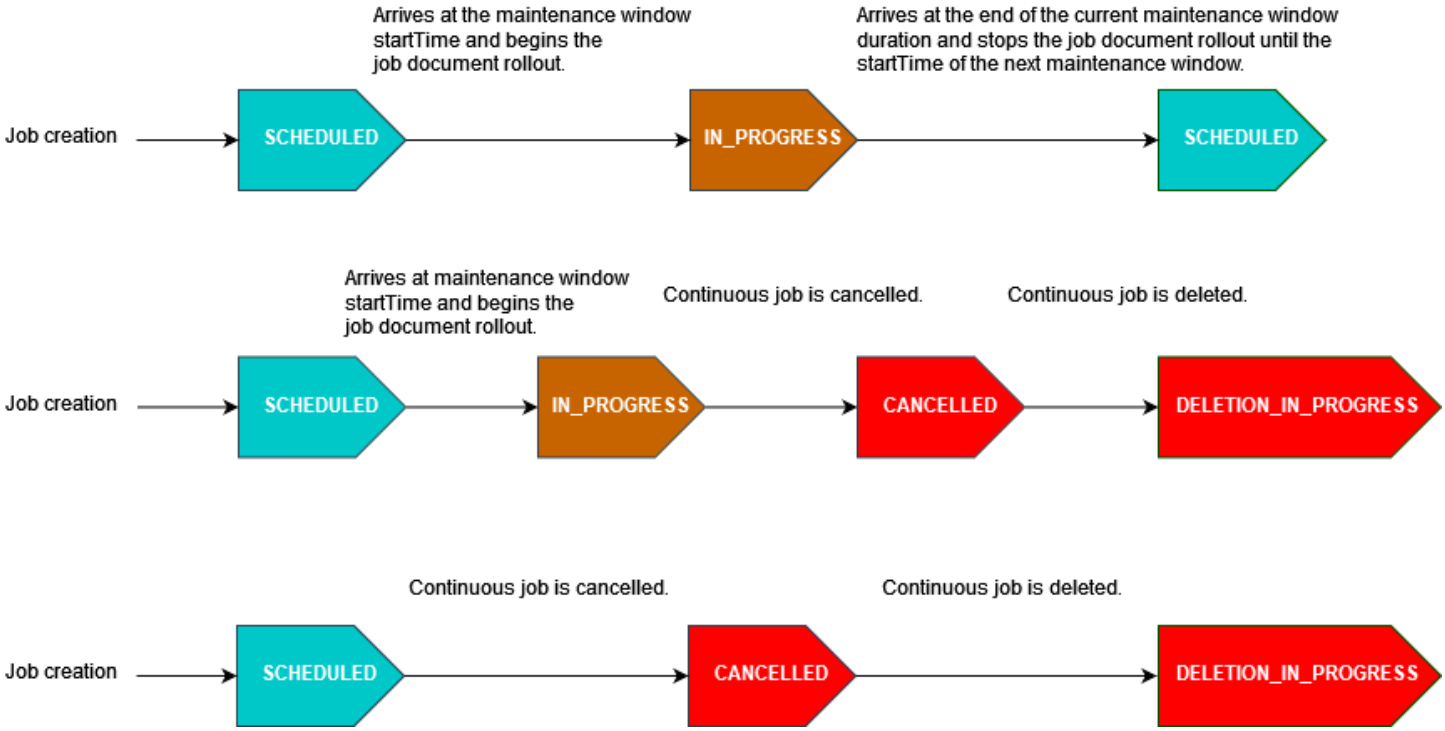
スケジュールされたジョブサンプル番号	startTime	endTime	最大継続時間
1	最初のジョブ作成直後。	最初のジョブ作成から 1 年後。	1 年
2	最初のジョブ作成から 1 か月後。	最初のジョブ作成から 13 か月後。	1 年
3	最初のジョブ作成から 1 年後。	最初のジョブ作成から 2 年後。	1 年
4	最初のジョブ作成直後。	最初のジョブ作成から 2 年後。	2 年

定期メンテナンスウィンドウ

メンテナンスウィンドウは、 のスケジュール設定内CreateJob、 AWS マネジメントコンソール および CreateJobTemplate API SchedulingConfig内のオプション設定APIs。開始時間、期間、およびメンテナンスウィンドウが発生する頻度 (毎日、毎週、または毎月) をあらかじめ設定して、

定期的なメンテナンスウィンドウを設定できます。メンテナンスウィンドウは連続ジョブにのみ適用されます。定期的なメンテナンスウィンドウの最大所要時間は 23 時間 50 分です。

次のリストでは、オプションのメンテナンスウィンドウを使用したスケジューリングされたさまざまなジョブシナリオのジョブステータスを示しています。



ジョブのステータス状態の詳細については、「Jobs とジョブ実行の状態」を参照してください。

> **Note**
>
> メンテナンスウィンドウ中にジョブが `endTime` に到着すると、そのジョブは `IN_PROGRESS` から `COMPLETED` に更新されます。さらに、残りのジョブ実行は、そのジョブの `endBehavior` 後に実行されます。

Cron 式

メンテナンスウィンドウ中にカスタム頻度でジョブドキュメントをロールアウトするようにスケジュールされたジョブの場合、カスタム頻度は cron 式を使用して入力されます。Cron 式には 6 つの必須フィールドがあり、それらはスペースで区切られます。

[Syntax] (構文)

```
cron(fields)
```

フィールド	値	ワイルドカード
分	0-59	, - * /
時間	0-23	, - * /
日	1-31	, - * ? / L W
月	1-12 または JAN-DEC	, - * /
曜日	1-7 または SUN-SAT	, - * ? L #
年	1970-2199	, - * /

ワイルドカード

- , (カンマ) のワイルドカードには、追加の値が含まれます。月フィールドの、「JAN,FEB,MAR」は、1 月、2 月、3 月を含みます。
- - (ダッシュ) のワイルドカードは、範囲を指定します。日フィールドの、「1-15」は、指定した月の 1 日から 15 日を含みます。
- [*] (アスタリスク) のワイルドカードには、フィールドのすべての値が含まれます。時間フィールドの、* にはすべての時間が含まれています。[*] を日および曜日フィールドの両方に使用することはできません。一方に使用する場合は、もう一方に [?] を使用する必要があります。
- / (スラッシュ) のワイルドカードは、増分を指定します。分フィールドで、「1/10」と入力して、その時間の最初の分から始めて、10 分毎を指定できます (11 分、21 分、31 分など)。
- [?] (疑問符) のワイルドカードは、任意を意味します。[日] フィールドに 7 と入力し、7 日が何曜日であってもかまわない場合、[曜日] フィールドに ? を入力できます。
- Day-of-month フィールドまたは Day-of-week フィールドの、ワイルドカード L は月または週の最終日を指定します。
- Day-of-month フィールドのワイルドカード **W** は、平日を指定します。Day-of-month フィールドで、**3W** は月の 3 日目に最も近い平日を指定します。
- Day-of-week フィールドの # ワイルドカードは、月の指定された曜日の特定のインスタンスを指定します。例えば、3#2 は、月の第 2 火曜日を示します。3 は週の 3 番目の日 (火曜日) を示し、2 は月のそのタイプの 2 番目の日を示します。

> **Note**
>
> 「#」文字を使用する場合、曜日フィールドには 1 つの式しか定義できません。例えば、"3#1,6#3" は 2 つの式として解釈されるため、無効です。

制限事項

- cron 式の日フィールドと曜日フィールドを同時に指定することはできません。一方のフィールドに値 (または *) を指定する場合、もう一方のフィールドで ? を使用する必要があります。

例

定期的なメンテナンスウィンドウの `startTime` に cron 式を使用する場合は、次の cron 文字列のサンプルを参照してください。

分	時間	日	月	曜日	年	意味
0	10	*	*	?	*	毎日午前 10:00 (UTC) に実行
15	12	*	*	?	*	毎日午後 12:15 (UTC) に実行
0	18	?	*	MON-FRI	*	毎週月曜日から金曜日まで午後 6:00 (UTC) に実行
0	8	1	*	?	*	毎月 1 日の午前 8:00

分	時間	日	月	曜日	年	意味
						(UTC) に実行

定期メンテナンスウィンドウ期間の終了ロジック

メンテナンスウィンドウ中のジョブのロールアウトが現在のメンテナンスウィンドウ発生期間の終わりに達すると、次のアクションが実行されます。

- ターゲットグループ内の残りのデバイスに対し、ジョブドキュメントのロールアウトをすべて停止します。次のメンテナンスウィンドウの `startTime` に再開されます。
- ステータスが `QUEUED` のすべてのジョブの実行は、次のメンテナンスウィンドウ発生の `startTime`まで `QUEUED` のままになります。次のウィンドウでは、ジョブドキュメントで指定されたアクションの実行をデバイスが開始できる状態になった時点で `IN_PROGRESS` に切り替えることができます。
- `IN_PROGRESS` ステータスのすべてのジョブ実行は、ターミナル状態になるまでジョブドキュメントで指定されたアクションを実行し続けます。`JobExecutionsRetryConfig` で指定されている再試行は、次のメンテナンスウィンドウの `startTime` に行われます。

ジョブの中止設定

この設定を使用して、デバイスのしきい値の割合がその基準を満たした場合にジョブをキャンセルする基準を作成します。例えば、次の場合にこの設定を使用してジョブをキャンセルできます :

- デバイスのしきい値の割合がジョブ実行通知を受信しない場合 (デバイスが 無線通信 (OTA) アップデートに対応していない場合など) 。この場合、デバイスは `REJECTED` ステータスを報告します。
- Amazon S3 URL からジョブドキュメントをダウンロードしようとしたときにデバイスが切断された場合など、デバイスのしきい値の割合によってジョブ実行の失敗が報告された場合。このような場合、デバイスを `FAILURE` ステータスを AWS IoTにレポートするようにプログラムする必要があります。
- ジョブの実行が開始された後、デバイスのしきい値の割合でジョブ実行がタイムアウトしたために `TIMED_OUT` ステータスが報告された場合。
- 複数の再試行が失敗した場合。再試行設定を追加すると、再試行のたびに AWS アカウントに追加料金が発生することがあります。このような場合、ジョブをキャンセルすると、キューに入れられ

たジョブの実行がキャンセルされ、これらの実行の再試行を回避できます。再試行設定と、中止設定との使用の詳細については、「ジョブ実行タイムアウト設定と再試行の設定」を参照してください。

AWS IoT コンソールまたは AWS IoT Jobs API を使用して、ジョブ中止条件を設定できます。

ジョブ実行タイムアウト設定と再試行の設定

ジョブ実行タイムアウト設定を使用して、ジョブの実行が設定された期間より長く進行中の場合ジョブの通知 を送信します。ジョブが失敗またはタイムアウトしたときに実行を再試行するには、ジョブ実行リトライ設定を使用します。

ジョブ実行タイムアウトの設定

ジョブ実行タイムアウト設定を使用すると、予期せず長時間、ジョブの実行が IN_PROGRESS ステータスのままになるたびに、通知を受けることができます。ジョブが IN_PROGRESS のとき、ジョブ実行の進行状況をモニタリングできます。

ジョブタイムアウトのタイマー

タイマーには、進捗タイマーとステップタイマーの 2 種類があります。

進捗タイマー

ジョブまたはジョブテンプレートを作成するときに、進捗タイマーの値を 1 分から 7 日間の間で指定できます。ジョブ実行の開始まで、このタイマーの値を更新できます。タイマーが開始されると、タイマーは更新できず、タイマー値はジョブのすべての実行に適用されます。ジョブの実行がこの間隔より長く IN_PROGRESSステータスのままになるたびに、ジョブの実行は失敗し、終了TIMED_OUTステータスに切り替わります。 AWS IoT も MQTT 通知を発行します。

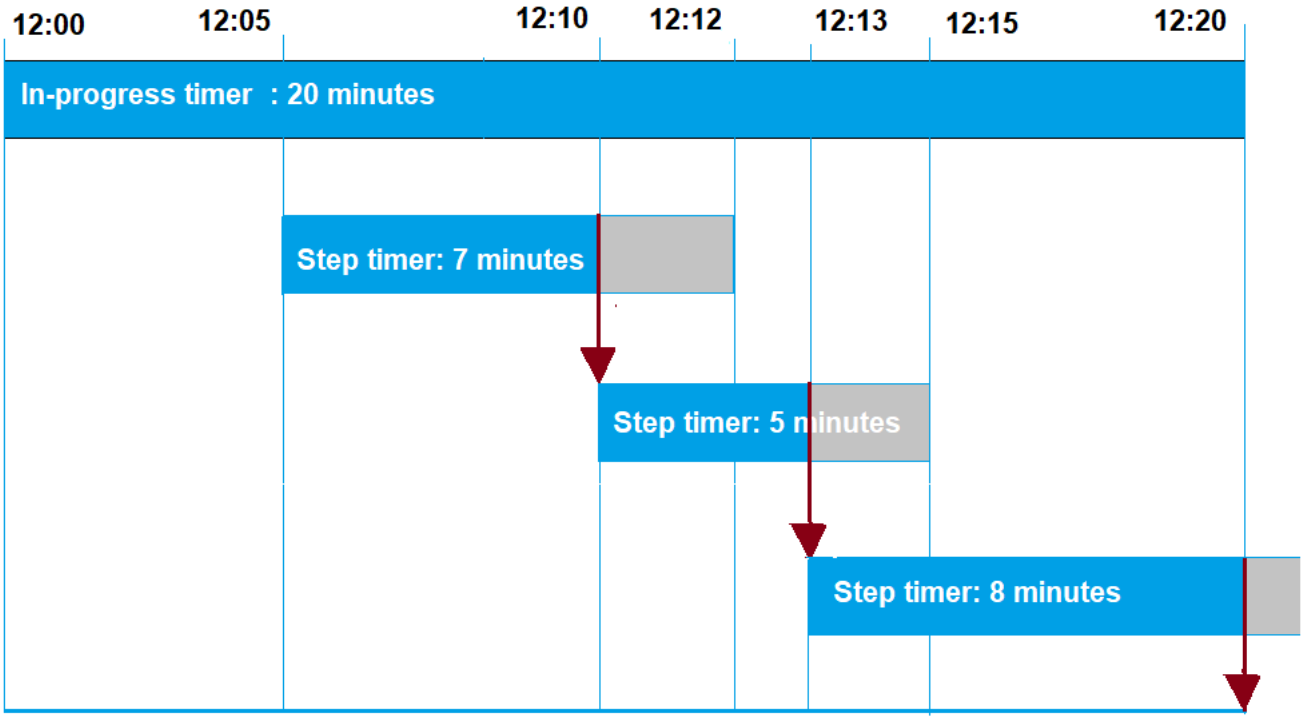
ステップタイマー

更新するジョブ実行のみに適用されるステップタイマーを設定することもできます。このタイマーは、進捗タイマーには影響を与えません。ジョブの実行を更新するたびに、このステップタイマーに新しい値を設定できます。モノに対して保留中の次のジョブ実行を開始するときに、新しいステップタイマーを作成することもできます。ジョブの実行がステップタイマーの間隔より長い間、IN_PROGRESS ステータスのままになる場合、ジョブの実行は失敗し、終了ステータス TIMED_OUT に切り替わります。

> **Note**
>
> コンソール AWS IoT または AWS IoT Jobs API を使用して、進行中のタイマーを設定できます。ステップタイマーを指定するには、API を使用します。

ジョブタイムアウトのタイマーの仕組み

以下の図は、20 分のタイムアウト期間の間に、進捗タイムアウトとステップタイムアウトが相互に影響を与えている例を示しています。



以下は異なるステップを示します。

1. 12:00

   新しいジョブが作成され、ジョブの作成時に 20 分の進捗タイマーが開始されます。進捗タイマーが開始され、ジョブの実行が `IN_PROGRESS` に切り替えられます。

2. 午後 12:05

   値 7 分の新しいステップタイマーを作成します。ジョブの実行は午後 12:12 にタイムアウトします。

3. 午後 12:10

値 5 分の新しいステップタイマーを作成します。新しいステップタイマーが作成されると、前のステップタイマーは破棄され、ジョブの実行は午後 12:15 にタイムアウトになります。

4. 午後 12:13

   値 9 分の新しいステップタイマーを作成します。前のステップタイマーは破棄され、進捗タイマーが午後 12:20 にタイムアウトするため、ジョブの実行は午後 12:20 にタイムアウトします。ステップタイマーは、進捗タイマーによって作成された絶対限度時間を超えることはできません。

ジョブ実行再試行設定

再試行設定を使用して、特定の条件セットが満たされたときにジョブの実行を再試行できます。再試行は、ジョブがタイムアウトしたとき、またはデバイスで失敗が発生したときに試行できます。タイムアウト失敗が原因で実行を再試行するには、タイムアウト設定を有効にする必要があります。

再試行設定の使用方法

設定を完了するには、次の手順を再試行します。

1. FAILED、TIMED_OUT、または両方の失敗基準に対して再試行設定を使用するかどうかを決定します。TIMED_OUT ステータスの場合、ステータスが報告されると、 AWS IoT Jobs は自動的にデバイスのジョブ実行を再試行します。
2. FAILED ステータスの場合、ジョブの実行失敗を再試行できるかどうかを確認します。再試行可能な場合は、FAILURE ステータスを AWS IoTに報告するようデバイスをプログラムします。次のセクションでは、再試行可能な失敗と復元不可能な失敗について詳しく説明します。
3. 前述の情報を使用して、各失敗タイプに使用する再試行回数を指定します。1 つのデバイスに対して、両方の失敗タイプを合わせて最大 10 回の再試行を指定できます。再試行は、実行が成功したとき、または指定された試行回数に達すると、自動的に停止します。
4. 再試行が繰り返し失敗した場合、ジョブをキャンセルする中止設定を追加して、再試行回数が多い場合でも追加料金が発生しないようにします。

> **Note**
>
> ジョブが繰り返し発生するメンテナンスウィンドウの終わりに達すると、すべての IN_PROGRESS ジョブ実行は、ターミナル状態に達するまでジョブドキュメントで指定されたアクションを実行し続けます。ジョブの実行がウィンドウの外で FAILED または

TIMED_OUT のターミナル状態になった場合は、再試行回数の上限に達するまで次のメンテナンスウィンドウで再試行されます。次のメンテナンスウィンドウが発生する startTime に、新しいジョブ実行が作成され、デバイスの起動準備が整うまで QUEUED のステータスに入ります。

再試行と中止設定

再試行するたびに、 に追加料金が発生します AWS アカウント。繰り返しの再試行失敗による追加料金が発生しないように、中止設定を追加することを推奨します。料金の詳細については、「AWS IoT Device Management の料金」を参照してください。

デバイスのしきい値が高い割合でタイムアウトしたり、失敗を報告したりすると、複数の再試行が失敗することがあります。この場合、中止設定を使用してジョブをキャンセルし、キューに入っているジョブの実行やそれ以上の再試行を回避できます。

> **Note**
>
> ジョブ実行をキャンセルするための中止基準が満たされた場合のみ、QUEUED ジョブ実行はキャンセルされます。デバイスのキューに入れられた再試行は試行されません。ただし、現在の IN_PROGRESS ステータスのジョブ実行はキャンセルされません。

失敗したジョブ実行を再試行する前に、以下のセクションで説明するように、ジョブ実行の失敗が再試行可能かどうかをチェックすることを推奨します。

FAILED の失敗タイプの再試行

FAILED の失敗タイプの再試行をする場合は、失敗したジョブ実行の FAILURE ステータスを AWS IoTにレポートするようにデバイスがプログラムされている必要があります。FAILED ジョブを再試行する条件再試行設定で設定し、また再試行回数を指定します。 AWS IoT Jobs がFAILUREステータスを検出すると、デバイスのジョブ実行を自動的に再試行します。再試行は、ジョブの実行が成功するか、最大再試行回数に達するまで継続されます。

各再試行とこれらのデバイスで実行されているジョブを追跡できます。実行ステータスを追跡することで、指定した回数の再試行が試行された後、デバイスを使用して失敗を報告し、別の再試行を開始できます。

再試行可能な失敗と復元不可能な失敗

ジョブ実行の失敗は、再試行可能または復元不可能の可能性があります。再試行するたびに、 AWS アカウントに料金が発生します。複数の再試行による追加料金が発生しないようにするには、まず、ジョブ実行の失敗が再試行可能かどうかをチェックすることを検討してください。再試行可能な失敗の例には、Amazon S3 URL からジョブドキュメントをダウンロードしようとしたときにデバイスに発生する接続エラーが含まれます。ジョブ実行の失敗が再試行可能な場合は、デバイスをプログラムして、ジョブの実行に失敗した場合に FAILURE ステータスを報告するように設定します。次に、FAILED の実行を再試行するように再試行を設定します。

実行を再試行できない場合、再試行してアカウントに追加料金が発生する可能性を回避するために、デバイスを REJECTED ステータスを AWS IoTに報告するようプログラムすることを推奨します。再試行不可能な障害の例としては、デバイスがジョブの更新を受信できない場合や、ジョブの実行中にメモリエラーが発生した場合などがあります。このような場合、 AWS IoT ジョブは FAILEDまたは TIMED_OUTステータスを検出した場合にのみジョブ実行を再試行するため、ジョブ実行を再試行しません。

ジョブ実行の失敗が再試行可能であると判断した後、再試行が失敗する場合は、デバイスログを確認することを検討してください。

> **Note**
>
> オプションのスケジューリング設定によるジョブが endTime に達すると、選択された endBehavior はターゲットグループ内の残りのすべてのデバイスへのジョブドキュメントのロールアウトを停止し、残りのジョブ実行で再試行を進める方法を指示します。再試行設定で選択した場合、試行は再試行されます。

TIMEOUT の失敗タイプの再試行

ジョブの作成時にタイムアウトを有効にすると、ジョブ AWS IoT はステータスが から IN_PROGRESSに変わったときにデバイスのジョブ実行を再試行しようとしますTIMED_OUT。このステータスの変化は、進捗タイマーがタイムアウトしたとき、または指定したステップタイマーが IN_PROGRESS である場合に発生し、その後タイムアウトします。再試行は、ジョブの実行が成功するか、この失敗タイプの再試行の最大回数に達するまで継続されます。

継続的なジョブとモノグループメンバーシップの更新

継続的なジョブのジョブステータスが IN_PROGRESS の場合、モノグループメンバーシップが更新されると、再試行回数はゼロにリセットされます。例えば、再試行を 5 回指定し、3 回の再試行がすでに実行されているとします。モノがモノグループから削除され、ダイナミックモノグループの場合

などで、モノがグループに再登録されると、再試行回数はゼロにリセットされます。これで、残りの 2 回の試行ではなく、モノグループに対して 5 回の再試行を実行できます。さらに、モノがモノグループから削除されると、追加の再試行がキャンセルされます。

追加の設定を指定する

ジョブまたはジョブテンプレートを作成する場合、これらの追加設定を指定できます。次に、これらの設定を指定できる時期を示します。

- カスタムジョブテンプレートの作成時 テンプレートからジョブを作成するときに、指定した追加の構成設定が保存されます。
- ジョブファイルを使用したカスタムジョブの作成時 ジョブファイルには、S3 バケットにアップロードされる JSON ファイルを使用できます。
- カスタムジョブテンプレートを使用したカスタムジョブの作成時 テンプレートにこれらの設定が既に指定されている場合は、それらを再利用するか、新しい構成設定を指定して上書きできます。
- AWS マネージドテンプレートを使用してカスタムジョブを作成する場合。

トピック

- を使用してジョブ設定を指定する AWS マネジメントコンソール
- Jobs API を使用して AWS IoT ジョブ設定を指定する

を使用してジョブ設定を指定する AWS マネジメントコンソール

AWS IoT コンソールを使用して、ジョブにさまざまな設定を追加できます。ジョブを作成した後、ジョブ設定のステータスの詳細を [job details] (ジョブ詳細) ページで確認できます。さまざまな設定とその動作の詳細については、「ジョブ設定の仕組み」を参照してください。

ジョブまたはジョブテンプレートを作成するときに、ジョブ設定を追加します。

カスタムジョブテンプレートの作成時

カスタムジョブテンプレートの作成時にロールアウト設定を指定するには

1. AWS IoT コンソールのジョブテンプレートハブに移動し、ジョブテンプレートの作成を選択します。
2. ジョブテンプレートのプロパティを指定し、ジョブドキュメントを指定し、追加する設定を展開して、設定パラメータを指定します。

カスタムジョブの作成時

カスタムジョブの作成時にロールアウト設定を指定するには

1. AWS IoT コンソールのジョブハブに移動し、ジョブの作成を選択します。
2. [Create a custom job] (カスタムジョブの作成) を選択し、ジョブのプロパティ、ターゲット、およびジョブドキュメントにジョブファイルまたはテンプレートを使用するかどうかを指定します。カスタムテンプレートまたは AWS マネージドテンプレートを使用できます。
3. ジョブ設定を選択し、[Rollout configuration] (ロールアウト設定) を展開し、[Constant rate] (一定のレート) または [Exponential rate] (指数関数的レート) を使用するかどうか指定します。次に、設定パラメータを指定します。

次のセクションでは、各設定に指定できるパラメータを示します。

ロールアウト設定

一定のロールアウトレートを使用するか、指数関数的レートを使用するかを指定できます。

- 一定のロールアウトレートを設定する

  ジョブ実行の一定レートを設定するには、[一定間隔] を選択し、[1 分あたりの最大数] でレートの上限を指定します。この値はオプションで、1 ~ 1000 の範囲です。設定しないと、デフォルト値として 1000 が使用されます。
- 指数関数的なロールアウトレートを設定する

  指数関数的レートを設定するには、[Exponential rate] (指数関数的レート) を選択し、次に、パラメータを指定します。

  - 1 分あたりのベースレート

    [通知されたデバイス数] または [成功したデバイスの数] が [レート増価基準] のしきい値を満たすまでジョブが実行されるレート。
  - 増分係数

    ロールアウトレートが、[Number of notified devices] (通知されたデバイスの数) または[Number of succeeded devices] (成功したデバイスの数) が [Rate increase criteria] (レート上げ基準) のしきい値を満たした後にロールアウトレートが増加する指数係数。
  - レート上げ基準

[Number of notified devices] (通知されたデバイスの数) または[Number of succeeded devices] (成功したデバイスの数) どちらかのしきい値。

中止設定

[Add new configuration] (新しい設定の追加) を選択し、各設定の以下のパラメータを指定します。

- 失敗タイプ

  ジョブの中止を開始する失敗の種類を指定します。[FAILED] (失敗)、[REJECTED] (拒否)、[TIMED_OUT] (タイムアウト)、または [ALL] (すべて) が含まれます。
- 増分係数

  ジョブ中止基準が満たされる前に、完了が必要なジョブの数を指定します。
- しきい値割合

  ジョブ中止を開始する、実行されたモノの合計数を指定します。

スケジューリング設定

各ジョブは、最初の作成時にすぐに開始することも、後の日時に開始するようにスケジューリングすることも、定期的なメンテナンスウィンドウで実行することもできます。

[Add new configuration] (新しい設定の追加) を選択し、各設定の以下のパラメータを指定します。

- ジョブ開始

  ジョブを開始する日時を指定します。
- 定期メンテナンスウィンドウ

  定期的なメンテナンスウィンドウは、ジョブがジョブ内のターゲットデバイスにジョブドキュメントを配信できる特定の日付と時刻を定義します。メンテナンスウィンドウは、毎日、毎週、毎月、またはカスタムの曜日と時間の繰り返しが可能です。
- ジョブ終了

  ジョブを終了する日時を指定します。
- ジョブ終了動作

  ジョブ終了時の未完了のすべてのジョブ実行の終了動作を選択します。

> **Note**
>
> オプションのスケジューリング設定があるジョブと選択した終了時刻が終了時刻に達すると、ジョブがターゲットグループ内の残りのすべてのデバイスへのジョブドキュメントのロールアウトを停止します。また、再試行設定に基づく残りのジョブ実行とその再試行回数をどのように進めるかについて、選択された終了動作を使用します。

タイムアウト設定

デフォルトでは、タイムアウトはなく、ジョブの実行はキャンセルまたは削除されます。タイムアウトを使用するには、[タイムアウトの有効化] を選択し、1 分から 7 日の間のタイムアウト値を指定します。

再試行設定

> **Note**
>
> ジョブの作成後は、再試行回数は更新できません。すべての失敗タイプの再試行設定のみを削除できます。ジョブを作成するときは、設定に使用する適切な再試行回数を考慮してください。再試行失敗の可能性による過剰なコストが発生しないようにするには、中止設定を追加します。

[Add new configuration] (新しい設定の追加) を選択し、各設定の以下のパラメータを指定します。

- 失敗タイプ

  ジョブ実行の再試行をトリガーする失敗の種類を指定します。[Failed] (失敗)、[Timeout] (タイムアウト)、[All] (すべて) が含まれます。
- 再試行回数

  選択した [Failure type] (失敗タイプ) の再試行回数を指定します。両方の失敗タイプを組み合わせると、最大 10 回の再試行が可能です。

Jobs API を使用して AWS IoT ジョブ設定を指定する

[CreateJob] (ジョブの作成) または[CreateJobTemplate] (ジョブテンプレートの作成) API を使用してさまざまなジョブ設定を指定できます。次のセクションでは、これらの設定を追加する

方法について説明します。設定を追加後、[JobExecutionSummary] (ジョブ実行概要) および [JobExecutionSummaryForJob] (ジョブ用のジョブ実行概要) を使用して、そのステータスを表示します。

さまざまな設定とその動作の詳細については、「ジョブ設定の仕組み」を参照してください。

ロールアウト設定

ロールアウト設定には、一定のロールアウトレートまたは指数関数的なロールアウトレートを指定できます。

- 一定のロールアウトレートを設定する

  一定のロールアウトレートを設定するには、JobExecutionsRolloutConfig オブジェクトを使用してmaximumPerMinute パラメータを CreateJob リクエストに追加します。このパラメータはジョブ実行が発生するレートの上限を指定します。この値はオプションで、1 ~ 1000 の範囲です。値を設定しないと、デフォルト値として 1000 が使用されます。

  ```
    "jobExecutionsRolloutConfig": {
        "maximumPerMinute": 1000
    }
  ```

- 指数関数的なロールアウトレートを設定する

  可変ジョブロールアウトレートを設定するには、JobExecutionsRolloutConfig オブジェクトを使用します。CreateJob API オペレーション実行時の ExponentialRolloutRate プロパティを設定できます。次の例では、exponentialRate パラメータを使用して指数関数的ロールアウトレートを設定します。パラメータの詳細については、「 」を参照してくださいExponentialRolloutRate

  ```
  {
  ...
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 50,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 1000,
          "numberOfSucceededThings": 1000
        },
        "maximumPerMinute": 1000
      }
  ```

```
  }
...
}
```

パラメータが以下の場合 :

baseRatePerMinute

numberOfNotifiedThings または numberOfSucceededThings しきい値に達するまで、ジョブを実行するレートを指定します。

incrementFactor

numberOfNotifiedThings または numberOfSucceededThings しきい値に達した後にロールアウトレートを増加する指数係数を指定します。

rateIncreaseCriteria

numberOfNotifiedThings または numberOfSucceededThings しきい値のいずれかを指定します。

中止設定

API を使用してこの設定を追加するには、CreateJob または CreateJobTemplate API オペレーションを実行するときに AbortConfig パラメータを指定します。次の例は、CreateJob API オペレーションで指定された複数回失敗した実行のジョブロールアウトに対する中止設定を示しています。

> **Note**
>
> ジョブ実行を削除すると、完了した実行の合計数の計算値に影響します。ジョブを中止するとき、サービスは自動で comment と reasonCode を作成し、ジョブ中止のキャンセルとユーザーによるキャンセルを区別します。

```
  "abortConfig": {
     "criteriaList": [
        {
           "action": "CANCEL",
           "failureType": "FAILED",
```

```
            "minNumberOfExecutedThings": 100,
            "thresholdPercentage": 20
        },
        {
            "action": "CANCEL",
            "failureType": "TIMED_OUT",
            "minNumberOfExecutedThings": 200,
            "thresholdPercentage": 50
        }
    ]
}
```

パラメータが以下の場合 :

action

中止基準が満たされたときに実行するアクションを指定します。このパラメータは必須であり、CANCEL が唯一の有効な値です。

failureType

ジョブの中止を開始する失敗の種類を指定します。有効な値は、FAILED、REJECTED、TIMED_OUT、および ALL です。

minNumberOfExecutedThings

ジョブ中止基準が満たされる前に、完了が必要なジョブの数を指定します。この例では、 AWS IoT は、ジョブ実行が完了しているデバイスが少なくとも 100 個になるまで、ジョブ中止が必要かどうかをチェックしません。

thresholdPercentage

ジョブ中止を開始する、ジョブが実行されたモノの合計数を指定します。この例では、 は順番に AWS IoT チェックし、しきい値の割合が満たされた場合にジョブの中止を開始します。100 回の実行が完了した後に完了した実行の 20% 以上が失敗した場合、ジョブのロールアウトはキャンセルされます。この基準が満たされない場合、 AWS IoT は、200 回の実行が完了した後に、完了した実行の少なくとも 50% がタイムアウトしたかどうかを確認します。この場合、ジョブのロールアウトがキャンセルされます。

スケジューリング設定

API を使用してこの設定を追加するには、CreateJob または CreateJobTemplate API オペレーションを実行する際にオプションの SchedulingConfig を指定します。

```
  "SchedulingConfig": {
    "endBehavior": string
    "endTime": string
    "maintenanceWindows": string
    "startTime": string
  }
```

パラメータが以下の場合 :

startTime

ジョブを開始する日時を指定します。

endTime

ジョブを終了する日時を指定します。

maintenanceWindows

対象グループ内のすべてのデバイスに対し、ジョブドキュメントのロールアウトスケジュールジョブに、オプションのメンテナンスウィンドウが選択されていた場合に指定します。`maintenanceWindow` の文字列形式は、日付は YYYY/MM/DD、時刻は hh:mm です。

endBehavior

スケジュールされたジョブが `endTime` に達したときのジョブの動作を指定します。

> **Note**
>
> ジョブのオプション `SchedulingConfig` は、`DescribeJob` および `DescribeJobTemplate` API で表示できます。

タイムアウト設定

API を使用してこの設定を追加するには、`CreateJob` または `CreateJobTemplate` API オペレーションを実行するときに `TimeoutConfig` パラメータを指定します。

タイムアウト設定を使用するには

1. ジョブまたはジョブテンプレートの作成時に進捗タイマーを設定するには、オプションの `TimeoutConfig` オブジェクトの `inProgressTimeoutInMinutes` プロパティの値を設定します。

```
    "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
```

2. ジョブ実行のステップタイマーを設定するには、UpdateJobExecution を呼び出すときの stepTimeoutInMinutesの値を設定します。ステップタイマーは更新するジョブ実行にのみ適用されます。ジョブの実行を更新するたびに、このタイマーに新しい値を設定できます。

> **Note**
>
> UpdateJobExecution は、値 -1 の新しいステップタイマーを作成することで、すでに作成されたステップタイマーを破棄できます。

```
{
   ...
    "statusDetails": {
      "string" : "string"
   },
   "stepTimeoutInMinutes": number
}
```

3. 新しいステップタイマーを作成するには、StartNextPendingJobExecution API オペレーションを呼び出すこともできます。

再試行設定

> **Note**
>
> ジョブを作成するときは、設定に使用する適切な再試行回数を考慮してください。再試行失敗の可能性による過剰なコストが発生しないようにするには、中止設定を追加します。ジョブの作成後は、再試行回数は更新できません。UpdateJob API オペレーションを使用する場合、再試行回数は 0 のみに設定できます。

API を使用してこの設定を追加するには、CreateJob または CreateJobTemplate API オペレーションを実行するときに jobExecutionsRetryConfig パラメータを指定します。

```
{
...
  "jobExecutionsRetryConfig": {
      "criteriaList": [
         {
            "failureType": "string",
            "numberOfRetries": number
         }
      ]
  }
...
}
```

criteriaList は、ジョブの失敗タイプごとに許可される再試行回数を決定する基準のリストを指定する配列です。

デバイスとジョブ

デバイスは、MQTT、HTTP 署名バージョン 4、または HTTP TLS を使用してAWS IoTジョブと通信できます。デバイスが AWS IoTジョブと通信するときに使用するエンドポイントを確認するには、DescribeEndpoint コマンドを実行します。たとえば、次のコマンドを実行するとします。

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

以下のような応答が得られます。

```
{
    "endpointAddress": "a1b2c3d4e5f6g7-ats.iot.us-west-2.amazonaws.com"
}
```

MQTT プロトコルを使用する

デバイスは MQTT プロトコルを使用してAWS IoTジョブと通信できます。デバイスは MQTT トピックをサブスクライブして、新しいジョブの通知を受け取り、AWS IoTジョブサービスからレスポンスを受信します。デバイスは、MQTT トピックを発行して、ジョブ起動の状態をクエリまたは更新します。各デバイスには、全般的な MQTT トピックがあります。MQTT トピックのパブリッシュとサブスクライブの詳細については、「デバイス通信プロトコル」を参照してください。

この通信方法では、デバイスはデバイス固有の証明書とプライベートキーを使用して AWS IoTジョブで認証します。

デバイスは次のトピックをサブスクライブできます。thing-name はデバイスに関連付けられたモノの名前です。

- **$aws/things/*thing-name*/jobs/notify**

  このトピックをサブスクライブして、保留中のジョブ起動のリストに対してジョブ起動が追加または削除されたときに、通知を受け取ります。

- **$aws/things/*thing-name*/jobs/notify-next**

  このトピックをサブスクライブして、次の保留中のジョブ実行が変更されたときに通知を受信します。

- **$aws/things/*thing-name*/jobs/*request-name*/accepted**

  AWS IoTJobs サービスは、MQTT トピックに成功メッセージと失敗メッセージを発行します。このトピックは、accepted または rejected を、リクエストを行うために使用されたトピックに追加することで構成されます。ここで、 request-nameは などのリクエストの名前Getで、トピックは です。$aws/things/myThing/jobs/get.AWS IoTJobs はトピックに成功メッセージを発行します$aws/things/myThing/jobs/get/accepted。

- **$aws/things/*thing-name*/jobs/*request-name*/rejected**

  ここで、request-name は Get のようなリクエスト名です。リクエストが失敗した場合、AWS IoTJobs は$aws/things/myThing/jobs/get/rejectedトピックに失敗メッセージを発行します。

以下の HTTPS API オペレーションを使用することもできます。

- UpdateJobExecution API を呼び出してジョブ実行のステータスを更新します。
- DescribeJobExecution API を呼び出してジョブ実行のステータスをクエリします。
- GetPendingJobExecutions API を呼び出して保留中のジョブの実行リストを取得します。
- jobId を $next として DescribeJobExecution API を呼び出して次の保留中のジョブ実行を取得します。
- StartNextPendingJobExecution API を呼び出して次の保留中のジョブ実行を取得して開始します。

HTTP 署名バージョン 4 の使用

デバイスは、ポート 443 の HTTP 署名バージョン 4 を使用してAWS IoTジョブと通信できます。これは、AWS SDK と CLI で使用されるメソッドです。これらのツールの詳細については、AWS CLI コマンドリファレンス: iot-jobs-data またはAWS SDK とツール、および希望する言語の IotJobsDataPlane セクションを参照してください。

この通信方法では、デバイスは IAM 認証情報を使用して AWS IoTジョブで認証します。

この方法では次のコマンドを使用できます。

- 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 の使用

デバイスは、このプロトコルをサポートするサードパーティーのソフトウェアクライアントを使用して、ポート 8443 で HTTP TLS を使用してAWS IoTジョブと通信できます。

この方法では、デバイスは X.509 証明書ベースの認証 (たとえば、デバイス固有の証明書とプライベートキー) を使用します。

この方法では次のコマンドを使用できます。

- DescribeJobExecution
- GetPendingJobExecutions
- StartNextPendingJobExecution
- UpdateJobExecution

ジョブを処理するデバイスのプログラミング

このセクションの例では、MQTT を使用してデバイスと AWS IoT ジョブサービスの連携を示します。または、対応する API または CLI コマンドを使用できます。これらの例では、MyThing と呼ばれるデバイスが以下の MQTT トピックをサブスクライブすることを想定しています。

- $aws/things/*MyThing*/jobs/notify+ または $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

のコード署名を使用している場合AWS IoT、デバイスコードはコードファイルの署名を検証する必要があります。署名は codesign プロパティのジョブドキュメントにあります。コードファイル署名の検証の詳細については、「Device Agent Sample」を参照してください。

トピック

- デバイスのワークフロー
- ジョブワークフロー
- ジョブの通知

デバイスのワークフロー

デバイスは、次のいずれかの方法を使用してデバイスが実行するジョブを処理できます。

- 次のジョブを取得する

  1. デバイスが最初にオンラインになると、デバイスの notify-next トピックに登録する必要があります。
  2. jobId を $next として DescribeJobExecution MQTT API を呼び出して、次のジョブ、そのジョブのドキュメント、および statusDetails に保存されている状態を含むその他の詳細を取得します。ジョブドキュメントにコードファイル署名がある場合、ジョブリクエストの処理を続行する前に、署名を確認する必要があります。

3. ジョブステータスを更新するには、UpdateJobExecution MQTT API を呼び出します。または、これと前のステップを 1 回の呼び出しで組み合わせるには、デバイスは StartNextPendingJobExecution を呼び出すことができます。
4. (オプション) ステップタイマーを追加するには、`stepTimeoutInMinutes` または UpdateJobExecution のいずれかを呼び出すときに StartNextPendingJobExecution の値を設定します。
5. UpdateJobExecution MQTT API を使用してジョブドキュメントで指定されたアクションを実行して、ジョブの進行状況を報告します。
6. この jobId で DescribeJobExecutionMQTT API を呼び出して、ジョブ実行のモニタリングを続行します。ジョブ実行が削除された場合、DescribeJobExecution は `ResourceNotFoundException` を返します。

   デバイスがジョブを実行中にジョブ実行がキャンセルされたまたは削除された場合、このデバイスは有効な状態に復旧することができる必要があります。
7. ジョブが終了したら、UpdateJobExecution MQTT API を呼び出してジョブのステータスを更新し、成功または失敗を報告します。
8. このジョブの実行ステータスが終了状態に変更されたため、実行可能な次のジョブ (存在する場合) が変更されます。デバイスは、次の保留中のジョブ実行が変更されたことを通知されます。この時点で、デバイスはステップ 2 の説明に従って続行する必要があります。

デバイスがオンラインのままであれば、後続の保留中のジョブ実行の通知が引き続き受信されます。これには、ジョブ完了時や、保留中の新しいジョブ実行が追加された時点の、ジョブ実行データが含まれます。これが発生すると、デバイスはステップ 2 の説明に従って続行されます。

- 利用可能なジョブを選択する

1. デバイスが最初にオンラインになると、モノの `notify` トピックに登録する必要があります。
2. GetPendingJobExecutions MQTT API を呼び出して、保留中のジョブ実行のリストを取得します。
3. リストに 1 つまたは複数のジョブの実行が含まれている場合は、1 つのジョブを選択します。
4. DescribeJobExecution MQTT API を呼び出して、`statusDetails` に保存されている状態を含め、ジョブドキュメントおよびその他の詳細を取得します。
5. ジョブステータスを更新するには、UpdateJobExecution MQTT API を呼び出します。このコマンドで `includeJobDocument` フィールドが `true` に設定されている場合、デバイスは前のステップをスキップして、この時点でジョブドキュメントを取得できます。

6. オプションで、ステップタイマーを追加するには、`stepTimeoutInMinutes` を呼び出すときに UpdateJobExecution の値を設定します。
7. UpdateJobExecution MQTT API を使用してジョブドキュメントで指定されたアクションを実行して、ジョブの進行状況を報告します。
8. この jobId で DescribeJobExecutionMQTT API を呼び出して、ジョブ実行のモニタリングを続行します。デバイスがジョブを実行中にそのジョブの実行がキャンセルまたは削除された場合は、このデバイスを有効な状態に復帰できる必要があります。
9. ジョブが終了したら、UpdateJobExecution MQTT API を呼び出してジョブのステータスを更新し、成功または失敗を報告します。

デバイスがオンラインのままになると、新しい保留中のジョブ実行が利用可能になるとき、保留中のジョブの実行がすべて通知されます。これが発生すると、デバイスはステップ 2 の説明に従って続行できます。

デバイスがジョブを実行できない場合は、UpdateJobExecution MQTT API を呼び出してジョブのステータスを `REJECTED` に更新する必要があります。

ジョブワークフロー

以下に、新しいジョブの開始からジョブ実行の完了ステータスのレポートまで、ジョブワークフローのさまざまなステップを示しています。

新しいジョブを開始する

新しいジョブが作成されると、AWS IoTJobs は各ターゲットデバイスの `$aws/things/thing-name/jobs/notify`トピックにメッセージを発行します。

メッセージには、次に示す情報が含まれます。

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
```

```
    }
}
```

デバイスは、ジョブの実行がキューに入れられたときに、'$aws/things/*thingName*/jobs/notify' トピックでこのメッセージを受け取ります。

> **Note**
>
> オプションの SchedulingConfig 付きのジョブの場合、そのジョブの初期状態は SCHEDULED に維持されます。ジョブが選択した startTime に達すると、次の処理が実行されます。
>
> - ジョブの状態は IN_PROGRESS に更新されます。
> - ジョブによって、対象グループ内のすべてのデバイスに対し、ジョブドキュメントのロールアウトを開始します。

ジョブ情報を取得する

ジョブの実行に関する詳細情報を取得するには、デバイスは includeJobDocument フィールドを true に設定して DescribeJobExecution MQTT API を呼び出します (デフォルト)。

リクエストが成功すると、AWS IoTジョブサービスは$aws/things/MyThing/jobs/0023/get/acceptedトピックにメッセージを発行します。

```
{
    "clientToken" : "client-001",
    "timestamp" : 1489097434407,
    "execution" : {
        "approximateSecondsBeforeTimedOut": number,
        "jobId" : "023",
        "status" : "QUEUED",
        "queuedAt" : 1489097374841,
        "lastUpdatedAt" : 1489097374841,
        "versionNumber" : 1,
        "jobDocument" : {
            < contents of job document >
        }
    }
}
```

リクエストが失敗すると、AWS IoTジョブサービスは$aws/things/MyThing/jobs/0023/get/rejectedトピックにメッセージを発行します。

デバイスにはジョブドキュメントがあるようになり、これを使用してジョブのリモートオペレーションを実行できます。ジョブドキュメントに Amazon S3 の署名付き URL が含まれている場合、デバイスはその URL を使用してジョブに必要なファイルをダウンロードできます。

ジョブの実行ステータスレポート

デバイスがジョブを実行しているときに、UpdateJobExecution MQTT API を呼び出してジョブの実行ステータスを更新できます。

たとえば、デバイスは、IN_PROGRESS トピックに次のメッセージを公開することによって、ジョブの実行ステータスを $aws/things/MyThing/jobs/0023/update に更新することができます。

```
{
    "status":"IN_PROGRESS",
    "statusDetails": {
        "progress":"50%"
    },
    "expectedVersion":"1",
    "clientToken":"client001"
}
```

ジョブは、$aws/things/MyThing/jobs/0023/update/accepted トピックまたは $aws/things/MyThing/jobs/0023/update/rejected トピックにメッセージを公開して応答します。

```
{
    "clientToken":"client001",
    "timestamp":1476289222841
}
```

デバイスは、StartNextPendingJobExecution を呼び出すことによって以前の 2 つのリクエストを組み合わせることができます。これは、次の保留中のジョブ実行を取得して開始し、デバイスがジョブ実行ステータスを更新できるようにします。このリクエストは、ジョブの実行が保留中の場合にもジョブドキュメントを返します。

ジョブに TimeoutConfig が含まれている場合、進捗タイマーが開始されます。UpdateJobExecution を呼び出すときの stepTimeoutInMinutesの値を設定することで、ジョブ実行のステッ

プタイマーを設定することもできます。ステップタイマーは更新するジョブ実行にのみ適用されます。ジョブの実行を更新するたびに、このタイマーに新しい値を設定できます。StartNextPendingJobExecution を呼び出すときに、ステップタイマーを作成することもできます。ジョブの実行がステップタイマーの間隔より長い間、IN_PROGRESS ステータスのままになる場合、ジョブの実行は失敗し、終了ステータス TIMED_OUT に切り替わります。ステップタイマーは、ジョブの作成時に設定した進捗タイマーには影響を与えません。

status フィールドは、IN_PROGRESS、SUCCEEDED、または FAILED に設定できます。すでにターミナル状態になっているジョブの実行ステータスは更新できません。

レポート実行の完了

ジョブの実行が終了すると、デバイスは UpdateJobExecution MQTT API を呼び出します。ジョブが正常に行われた場合は、status を SUCCEEDED に設定し、メッセージペイロードの statusDetails にジョブに関する他の情報を名前と値のペアとして追加します。ジョブの実行が完了すると、進捗タイマーとステップタイマーが終了します。

以下に例を示します。

```
{
    "status":"SUCCEEDED",
    "statusDetails": {
        "progress":"100%"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

ジョブが失敗した場合は、status を FAILED に設定し、statusDetails に発生したエラーに関する情報を追加します。

```
{
    "status":"FAILED",
    "statusDetails": {
        "errorCode":"101",
        "errorMsg":"Unable to install update"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

> **Note**
>
> statusDetails 属性には、任意の数の名前と値のペアを含めることができます。

AWS IoTジョブサービスは、この更新を受け取ると、 $aws/things/MyThing/jobs/notifyトピックにメッセージを発行して、ジョブの実行が完了したことを示します。

```
{
    "timestamp":1476290692776,
    "jobs":{}
}
```

その他のジョブ

デバイスに対して保留中の他のジョブの実行がある場合、それらは $aws/things/MyThing/jobs/notify に公開されたメッセージに含まれます。

以下に例を示します。

```
{
    "timestamp":1476290692776,
    "jobs":{
        "QUEUED":[{
            "jobId":"0002",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }],
        "IN_PROGRESS":[{
            "jobId":"0003",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }]
    }
}
```

ジョブの通知

AWS IoTJobs サービスは、ジョブが保留中の場合、またはリストの最初のジョブ実行が変更された場合、予約済みトピックに MQTT メッセージを発行します。デバイスでは、これらのトピックにサブスクライブすることによって、保留中のジョブを追跡できます。

ジョブ通知タイプ

ジョブ通知は、MQTT トピックに JSON ペイロードとして発行されます。通知は 2 種類あります。

ListNotification

`ListNotification` には、15 件を超えない保留中のジョブの実行リストが含まれます。ステータスによってソートされた後 (`IN_PROGRESS` ジョブ実行の前に `QUEUED` ジョブ実行)、キューに入れられた時刻によってソートされます。

`ListNotification` は、次のいずれかの条件が満たされると必ず発行されます。

- 新しいジョブ実行がキューに登録されたか、非ターミナルステータス (`IN_PROGRESS` または `QUEUED`) に変わった。
- 古いジョブの実行が終了ステータスに変わった (`FAILED`、`SUCCEEDED`、`CANCELED`、`TIMED_OUT`、`REJECTED`、または `REMOVED`)。

スケジュール設定をする場合としない場合の制限の詳細については、「ジョブ実行の制限」を参照してください。

NextNotification

- `NextNotification` には、キュー内の次のジョブ実行に関する概要が含まれています。

  `NextNotification` は、リスト内の最初のジョブ実行が変更されると必ず発行されます。

  - 新しいジョブ実行は `QUEUED` としてリストに追加され、リスト内の最初の項目になります。
  - リスト内の最初の項目でない既存のジョブ実行のステータスは、`QUEUED` から `IN_PROGRESS` に変わり、リストにある最初の項目になります。(これは、リスト内に他の `IN_PROGRESS` ジョブ実行がない場合や、ステータスが `QUEUED` から `IN_PROGRESS` に変わったジョブ実行がリスト内の他の `IN_PROGRESS` ジョブ実行より早くキューに登録された場合に発生します)。
  - リスト内の最小にあるジョブ実行のステータスがターミナルステータスに変更され、リストから削除されます。

MQTT トピックのパブリッシュとサブスクライブの詳細については、「the section called “デバイス通信プロトコル”」を参照してください。

> **Note**
>
> ジョブとの通信に HTTP 署名バージョン 4 あるいは HTTP TLS を使用する場合、通知は利用できません。

ジョブの保留

AWS IoTJobs サービスは、モノの保留中のジョブ実行のリストからジョブが追加または削除されるか、リスト内の最初のジョブ実行が変更されると、MQTT トピックにメッセージを発行します。

- `$aws/things/thingName/jobs/notify`
- `$aws/things/thingName/jobs/notify-next`

メッセージには、次のペイロード例が含まれています。

`$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-job というジョブの実行が、オプションのスケジュール設定が選択されているジョブから開始され、ジョブドキュメントのロールアウトがメンテナンスウィンドウ中に実行されるようにスケ

ジュールされている場合、そのジョブは定期的なメンテナンスウィンドウ中にのみ表示されます。メンテナンスウィンドウ以外では、this-job というジョブは、次の例に示すように、保留中のジョブの実行リストから除外されます。

```
{
  "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-job というジョブの実行が、オプションのスケジュール設定が選択されているジョブから開始され、ジョブドキュメントのロールアウトがメンテナンスウィンドウ中に実行されるようにスケジュールされている場合、そのジョブは定期的なメンテナンスウィンドウ中にのみ表示されます。メンテナンスウィンドウ以外では、other-job というジョブは、次の例に示すように、次のジョブの実行として表示されません。

```
{} //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"
```

有効なジョブの実行ステータス値は
QUEUED、IN_PROGRESS、FAILED、SUCCEEDED、CANCELED、TIMED_OUT、REJECTED、REMOVED です。

以下の一連の例では、ジョブ実行が作成、および 1 つの状態から別の状態に変更される各トピックに発行される通知を示しています。

まず、job1 という名の 1 つのジョブが作成されます。この通知は、jobs/notify トピックに発行されます。

```
{
  "timestamp": 1517016948,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

この通知は、jobs/notify-next トピックに発行されます。

```
{
  "timestamp": 1517016948,
  "execution": {
```

```
    "jobId": "job1",
    "status": "QUEUED",
    "queuedAt": 1517016947,
    "lastUpdatedAt": 1517016947,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

別のジョブが作成されると (job2)、この通知が jobs/notify トピックに発行されます。

```
{
  "timestamp": 1517017192,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

キューの次のジョブ (jobs/notify-next) が変更されていないため、通知は job1 トピックに発行されません。job1 が実行を開始した場合、そのステータスは IN_PROGRESS に変わります。ジョブのリストおよびキューの次のジョブが変更されないため、通知は発行されません。

3 番目のジョブが追加されると (job3)、この通知が jobs/notify トピックに発行されます。

```
{
```

```
  "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
      }
    ]
  }
}
```

キューの次のジョブがまだ jobs/notify-next であるため、通知は job1 トピックに発行されません。

job1 が完了すると、そのステータスは SUCCEEDED に変わり、この通知は jobs/notify トピックに発行されます。

```
{
  "timestamp": 1517186269,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job2",
```

```
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

この時点で、job1 はキューから削除され、実行する次のジョブが job2 になります。この通知は、jobs/notify-next トピックに発行されます。

```
{
  "timestamp": 1517186269,
  "execution": {
    "jobId": "job2",
    "status": "QUEUED",
    "queuedAt": 1517017191,
    "lastUpdatedAt": 1517017191,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

job3 よりも前に job2 の実行を開始する必要がある場合 (非推奨)、job3 のステータスを IN_PROGRESS に変更できます。この場合、job2 はキューの次のジョブではなくなり、この通知が jobs/notify-next トピックに発行されます。

```
{
  "timestamp": 1517186779,
  "execution": {
    "jobId": "job3",
```

```
    "status": "IN_PROGRESS",
    "queuedAt": 1517017905,
    "startedAt": 1517186779,
    "lastUpdatedAt": 1517186779,
    "versionNumber": 2,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

追加あるいは削除されたジョブがないため、jobs/notify トピックに発行される通知はありません。

デバイスが job2 を拒否し、そのステータスを REJECTED に変更した場合、この通知は jobs/notify トピックに発行されます。

```
{
  "timestamp": 1517189392,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517186779,
        "startedAt": 1517186779,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ]
  }
}
```

job3 (まだ進行中) が強制的に削除される場合、この通知は jobs/notify トピックに発行されます。

```
{
  "timestamp": 1517189551,
  "jobs": {}
}
```

この時点で、キューは空です。この通知は、jobs/notify-next トピックに発行されます。

```
{
  "timestamp": 1517189551
}
```

AWS IoT ジョブ API オペレーション

AWS IoT Jobs API は、次のいずれかのカテゴリに使用できます。

- ジョブの管理や制御などの管理タスク。これは、コントロールプレーンです。
- それらのジョブを実行するデバイス。これは、データプレーンです。これにより、データの送受信が可能になります。

ジョブの管理と制御には HTTPS プロトコル API を使用します。デバイスは、MQTT または HTTPS プロトコル API を使用できます。コントロールプレーンは、ジョブの作成と追跡の際に一般的な少数の呼び出しに対応するように設計されています。これは、通常、単一のリクエストの接続を開き、レスポンスが受信された後で接続を閉じます。データプレーンの HTTPS および MQTT API により、長時間のポーリングが可能になります。これらの API オペレーションは、数百万のデバイスに拡張できる大量のトラフィック用に設計されています。

各 AWS IoT Jobs HTTPS API には、 AWS Command Line Interface () から API を呼び出すことができる対応するコマンドがありますAWS CLI。コマンドは小文字で、API の名前を構成する単語の間にハイフンが付きます。たとえば、CLI で CreateJob API を呼び出すには、次のように入力します。

```
aws iot create-job ...
```

オペレーション中にエラーが発生した場合は、エラーに関する情報を含むエラーレスポンスが返されます。

ErrorResponse

AWS IoT ジョブサービスオペレーション中に発生したエラーに関する情報が含まれます。

以下の例は、このオペレーションの構文を示しています。

```
{
    "code": "ErrorCode",
    "message": "string",
```

```
    "clientToken": "string",
    "timestamp": timestamp,
    "executionState": JobExecutionState
}
```

以下は、この ErrorResponse の説明です。

code

ErrorCode を以下に設定できます。

InvalidTopic

リクエストは、どの API オペレーションにもマッピングされない AWS IoT Jobs 名前空間のトピックに送信されました。

InvalidJson

リクエストの内容を有効な UTF-8 エンコード JSON として解釈できませんでした。

InvalidRequest

リクエストの内容が無効です。たとえば、このコードは、UpdateJobExecution リクエストに無効なステータスの詳細が含まれている場合に返されます。メッセージには、エラーに関する詳細情報が含まれています。

InvalidStateTransition

ジョブ実行の現在の状態が原因で、ジョブ実行を有効でない状態に変更しようとするアップデートが試行されました。たとえば、「SUCCEEDED」状態のリクエストを「IN_PROGRESS」状態に変更しようとします。この場合、エラーメッセージの本文には executionState フィールドも含まれます。

ResourceNotFound

リクエストトピックによって指定された JobExecution が存在しません。

VersionMismatch

リクエストで指定された想定バージョンが、 AWS IoT ジョブサービスのジョブ実行のバージョンと一致しません。この場合、エラーメッセージの本文には executionState フィールドも含まれます。

InternalError

リクエストの処理中に内部エラーが発生しました。

RequestThrottled

リクエストがスロットリングされました。

TerminalStateReached

ターミナル状態のジョブでジョブについて説明するコマンドが実行されたときに発生します。

`message`

エラーメッセージ文字列。

`clientToken`

リクエストをその応答に関連付けるために使用される任意の文字列。

`timestamp`

エポックからの秒単位の時間。

`executionState`

JobExecutionState オブジェクト。このフィールドは、code フィールドの値が InvalidStateTransition または VersionMismatch の場合にのみ含まれます。これにより、これらの場合、現在のジョブ実行ステータスデータを取得するための別の DescribeJobExecution リクエストを実行する必要はありません。

ジョブの API オペレーションとデータ型は以下のとおりです。

- ジョブ管理と制御のデータ型
- ジョブデバイス MQTT および HTTPS API オペレーションとデータ型

ジョブ管理と制御のデータ型

次のコマンドは、CLI または HTTPS プロトコルを介してジョブの管理および制御で使用できます。

- ジョブ管理と制御のデータ型
- ジョブ管理と制御の API オペレーション

CLI コマンドの *endpoint-url* パラメータを特定するには、次のコマンドを実行します。

```
aws iot describe-endpoint --endpoint-type=iot:Jobs
```

このコマンドは、次の出力を返します。

```
{
"endpointAddress": "account-specific-prefix.jobs.iot.aws-region.amazonaws.com"
}
```

> **Note**
>
> ジョブエンドポイントは ALPN x-amzn-http-ca をサポートしていません。
> デュアルスタックエンドポイント (IPv6 と IPv6) を使用している場合は、iot:Data-ATS エンドポイントを使用します。iot:Jobs エンドポイントは IPv4 のみをサポートします。

ジョブ管理と制御のデータ型

次のデータ型は、管理および制御アプリケーションが AWS IoT ジョブと通信するために使用されます。

ジョブ

Job オブジェクトにはジョブの詳細が含まれています。以下に構文例を示します。

```
{
    "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
    }
}
```

詳細については、「Job」または「job」を参照してください。

JobSummary

JobSummary オブジェクトにはジョブの概要が含まれています。以下に構文例を示します。

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED|SCHEDULED",
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "thingGroupId": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp
}
```

詳細については、「JobSummary」または「job-summary」を参照してください。

JobExecution

JobExecution オブジェクトは、デバイスでのジョブの実行を表します。以下に構文例を示します。

> **Note**
>
> コントロールプレーン API オペレーションを使用する場合、JobExecution データ型には JobDocument フィールドが含まれません。この情報を取得するには、GetJobDocument API オペレーションまたは get-job-document CLI コマンドを使用できます。

```
{
    "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
}
```

詳細については、「JobExecution」または「job-execution」を参照してください。

JobExecutionSummary

JobExecutionSummary オブジェクトには、ジョブ実行の概要情報が含まれています。以下に構文例を示します。

```
{
    "executionNumber": 1234567890,
    "queuedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|REMOVED"
}
```

詳細については、「JobExecutionSummary」または「job-execution-summary」を参照してください。

JobExecutionSummaryForJob

JobExecutionSummaryForJob オブジェクトには、特定のジョブのジョブ実行に関する情報の概要が含まれています。以下に構文例を示します。

```
{
    "executionSummaries": [
        {
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyThing",
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            }
        },
```

```
        ...
    ]
}
```

詳細については、「JobExecutionSummaryForJob」または「job-execution-summary-for-job」を参照してください。

JobExecutionSummaryForThing

JobExecutionSummaryForThing オブジェクトには、特定のモノのジョブ実行に関する情報の概要が含まれています。以下に構文例を示します。

```
{
    "executionSummaries": [
        {
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            },
            "jobId": "MyThingJob"
        },
        ...
    ]
}
```

詳細については、「JobExecutionSummaryForThing」または「job-execution-summary-for-thing」を参照してください。

ジョブ管理と制御の API オペレーション

次の API オペレーションまたは CLI コマンドを使用します。

AssociateTargetsWithJob

グループを連続ジョブに関連付けます。以下の条件を満たす必要があります。

- ジョブは、targetSelection フィールドを CONTINUOUS に設定して作成しておく必要があります。
- ジョブのステータスは、現在 IN_PROGRESS です。

- ジョブに関連付けられたターゲットの合計数が 100 を超えることはできません。

HTTPS request

```
POST /jobs/jobId/targets

{
"targets": [ "string" ],
"comment": "string"
}
```

詳細については、「AssociateTargetsWithJob」を参照してください。

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 形式:

```
{
"targets": [
"string"
],
"jobId": "string",
"comment": "string"
}
```

詳細については、「associate-targets-with-job」を参照してください。

CancelJob

ジョブをキャンセルします。

HTTPS request

```
PUT /jobs/jobId/cancel
```

```
{
"force": boolean,
"comment": "string",
"reasonCode": "string"
}
```

詳細については、「CancelJob」を参照してください。

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 形式:

```
{
    "jobId": "string",
    "force": boolean,
    "comment": "string"
}
```

詳細については、「cancel-job」を参照してください。

CancelJobExecution

デバイスでジョブ実行をキャンセルする。

HTTPS request

```
PUT /things/thingName/jobs/jobId/cancel

{
"force": boolean,
"expectedVersion": "string",
"statusDetails": {
    "string": "string"
```

```
    ...
}
}
```

詳細については、「CancelJobExecution」を参照してください。

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 形式:

```
{
"jobId": "string",
"thingName": "string",
"force": boolean,
"expectedVersion": long,
"statusDetails": {
"string": "string"
}
}
```

詳細については、「cancel-job-execution」を参照してください。

CreateJob

ジョブを作成します。ジョブドキュメントは、Amazon S3 バケット (documentSource パラメータ) またはリクエストの本文 (document パラメータ) のファイルへのリンクとして提供できます。

オプションの targetSelection パラメータを CONTINUOUS に設定することで、ジョブを連続にすることができます (デフォルトは SNAPSHOT)。連続ジョブは実行され続け、新規追加されたモノで起動するため、連続ジョブを使用して、グループに追加されたデバイスをオンボードまたはアップグレードできます。これは、ジョブ作成時点のグループ内のモノがジョブを完了した後でも発生する可能性があります。

ジョブでは、オプションの TimeoutConfig に進捗タイマーの値を設定できます。進捗タイマーは更新できず、ジョブのすべての実行に適用されます。

CreateJob API の引数に対して、次の検証が実行されます。

- targets 引数は、有効なモノまたはモノのグループの ARN のリストでなければなりません。すべてのモノとモノのグループは に存在する必要があります AWS アカウント。
- documentSource 引数は、ジョブドキュメント有効な Amazon S3 URL である必要があります。Amazon S3 URL は https://s3.amazonaws.com/*bucketName*/*objectName* という形式です。
- documentSource 引数で指定された URL に格納されているドキュメントは、UTF-8 でエンコードされた JSON ドキュメントである必要があります。
- ジョブドキュメントのサイズは、MQTT メッセージのサイズ (128 KB) と暗号化の制限から、32 KB に制限されています。
- は 内で一意jobIdである必要があります AWS アカウント。

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
}
}
```

詳細については、「CreateJob」を参照してください。

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 形式:

```
{
    "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"
    }
```

```
}
```

詳細については、「create-job」を参照してください。

DeleteJob

ジョブおよびそれに関連するジョブの実行を削除します。

ジョブに作成されたジョブ実行の数およびその他さまざまな要素に応じて、ジョブの削除には時間がかかる場合があります。ジョブの削除中、そのジョブのステータスは「DELETION_IN_PROGRESS」として表示されます。ステータスがすでに「DELETION_IN_PROGRESS」のジョブを削除あるいはキャンセルしようとすると、エラーになります。

HTTPS request

```
DELETE /jobs/jobId?force=force
```

詳細については、「DeleteJob」を参照してください。

CLI syntax

```
aws iot  delete-job \
--job-id <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 形式:

```
{
"jobId": "string",
"force": boolean
}
```

詳細については、「delete-job」を参照してください。

DeleteJobExecution

ジョブの実行を削除します。

HTTPS request

```
DELETE /things/thingName/jobs/jobId/executionNumber/executionNumber?force=force
```

詳細については、「DeleteJobExecution」を参照してください。

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 形式:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long,
"force": boolean
}
```

詳細については、「delete-job-execution」を参照してください。

DescribeJob

ジョブ実行の詳細を取得します。

HTTPS request

```
GET /jobs/jobId
```

詳細については、「DescribeJob」を参照してください。

CLI syntax

```
aws iot describe-job \
--job-id <value>  \
```

```
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 形式:

```
{
"jobId": "string"
}
```

詳細については、「describe-job」を参照してください。

DescribeJobExecution

ジョブ実行の詳細を取得します。ジョブの実行ステータスは、SUCCEEDED または FAILED である必要があります。

HTTPS request

```
GET /things/thingName/jobs/jobId?executionNumber=executionNumber
```

詳細については、「DescribeJobExecution」を参照してください。

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 形式:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long
}
```

詳細については、「describe-job-execution」を参照してください。

GetJobDocument

ジョブのジョブドキュメントを取得します。

> **Note**
>
> プレースホルダーの URL は、返されたドキュメントの署名付き Amazon S3 URL に置き換えられません。署名付き URLs は、 AWS IoT ジョブサービスが MQTT 経由でリクエストを受け取った場合にのみ生成されます。

HTTPS request

```
GET /jobs/jobId/job-document
```

詳細については、「GetJobDocument」を参照してください。

CLI syntax

```
aws iot get-job-document \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` 形式:

```
{
"jobId": "string"
}
```

詳細については、「get-job-document」を参照してください。

ListJobExecutionsForJob

ジョブのジョブ実行リストを取得します。

HTTPS request

```
GET /jobs/jobId/things?status=status&maxResults=maxResults&nextToken=nextToken
```

詳細については、「ListJobExecutionsForJob」を参照してください。

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 形式:

```
{
"jobId": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

詳細については、「list-job-executions-for-job」を参照してください。

ListJobExecutionsForThing

モノのジョブ実行リストを取得します。

HTTPS request

```
GET /things/thingName/jobs?status=status&maxResults=maxResults&nextToken=nextToken
```

詳細については、「ListJobExecutionsForThing」を参照してください。

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 形式:

```
{
"thingName": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

詳細については、「list-job-executions-for-thing」を参照してください。

ListJobs

内のジョブのリストを取得します AWS アカウント。

HTTPS request

```
GET /jobs?
status=status&targetSelection=targetSelection&thingGroupName=thingGroupName&thingGroupId=thi
```

詳細については、「ListJobs」を参照してください。

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 形式:

```
{
"status": "string",
"targetSelection": "string",
```

```
"maxResults": "integer",
"nextToken": "string",
"thingGroupName": "string",
"thingGroupId": "string"
}
```

詳細については、「list-jobs」を参照してください。

UpdateJob

指定されたジョブのサポート対象フィールドを更新する。timeoutConfig の更新された値は、新しく進行中の起動に対してのみ有効になります。現在、進行中の起動は、以前のタイムアウト設定で引き続き起動します。

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
}
}
```

詳細については、「UpdateJob」を参照してください。

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 形式:

```
{
"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
}
}
```

詳細については、「update-job」を参照してください。

ジョブデバイス MQTT および HTTPS API オペレーションとデータ型

次のコマンドは、MQTT および HTTPS プロトコル経由で利用できます。ジョブを実行するデバイスのデータプレーンで、これらの API オペレーションを使用します。

ジョブデバイス MQTT および HTTPS データ型

次のデータ型は、MQTT および HTTPS プロトコルを介して AWS IoT Jobs サービスと通信するために使用されます。

JobExecution

JobExecution オブジェクトは、デバイスでのジョブの実行を表します。以下に構文例を示します。

> **Note**
>
> MQTT および HTTP データプレーン API オペレーションを使用する場合、JobExecution データ型には JobDocument フィールドが含まれます。デバイスはこの情報を使用して、ジョブ実行からジョブドキュメントを取得できます。

```
{
    "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
}
```

詳細については、「JobExecution」または「job-execution」を参照してください。

JobExecutionState

JobExecutionState には、ジョブの実行状態に関する情報が含まれます。以下に構文例を示します。

```
{
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
        ...
    }
    "versionNumber": "number"
}
```

詳細については、「JobExecutionState」または「job-execution-state」を参照してください。

JobExecutionSummary

ジョブの実行に関する情報のサブセットが含まれています。以下に構文例を示します。

```
{
    "jobId": "string",
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "versionNumber": "number",
    "executionNumber": long
}
```

詳細については、「JobExecutionSummary」または「job-execution-summary」を参照してください。

MQTT および HTTPS API オペレーションの詳細については、以下のセクションを参照してください。

- ジョブデバイス MQTT API オペレーション
- ジョブデバイス MQTT API

ジョブデバイス MQTT API オペレーション

ジョブデバイスコマンドは、ジョブコマンドに使用される予約済みトピックに MQTT メッセージを発行することで発行できます。

デバイス側のクライアントは、これらのコマンドの応答メッセージトピックにサブスクライブする必要があります。 AWS IoT Device Client を使用する場合、デバイスはレスポンストピックに自動的にサブスクライブします。つまり、クライアントが応答メッセージトピックをサブスクライブしているかどうかにかかわらず、メッセージブローカーはコマンドメッセージを発行したクライアントに応答メッセージトピックを発行することになります。これらの応答メッセージはメッセージブローカーを通過せず、他のクライアントまたはルールによってサブスクライブする事はできません。

フリートモニタリングソリューション用のジョブと `jobExecution` イベントトピックをサブスクライブする際は、まずジョブおよびジョブ実行イベントを有効にして、クラウド側でイベント受信します。メッセージブローカーを介して処理され、 AWS IoT ルールで使用できるジョブ進行状況メッセージは ジョブイベント として発行されます。メッセージブローカーは、応答メッセージを発行するため、明示的なサブスクリプションがない場合でも、クライアントは、メッセージを受信し、受信したメッセージを識別するように設定される必要があります。また、クライアントは、クライアントがメッセージを処理する前に、受信メッセージトピックの *`thingName`* がクライアントの Thing 名に適用されることを確認する必要もあります。

> **Note**
>
> MQTT Jobs API コマンドメッセージに応答して が AWS IoT 送信するメッセージは、明示的にサブスクライブしたかどうかにかかわらず、アカウントに課金されます。

以下は、MQTT API オペレーションとそのリクエストとレスポンスの構文を示しています。すべての MQTT API オペレーションには次のパラメータがあります。

clientToken

リクエストとレスポンスを関連付けるために使用されるオプションのクライアントトークン。ここに任意の値を入力すると、レスポンスに反映されます。

timestamp

メッセージが送信されたときの、エポックからの秒単位の時間。

GetPendingJobExecutions

特定のモノについて、ターミナル状態にないすべてのジョブのリストを取得します。

この API を呼び出すには、$aws/things/*thingName*/jobs/get にメッセージを発行します。

リクエストペイロード:

```
{ "clientToken": "string" }
```

メッセージブローカーは、特定のサブスクリプションがなくても $aws/things/*thingName*/jobs/get/accepted および $aws/things/*thingName*/jobs/get/rejected を発行します。ただし、クライアントがメッセージを受信するには、それらをリッスンしている必要があります。詳細については、[the note about Jobs API messages](Jobs API メッセージに関する注記) を参照してください。

レスポンスペイロード:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ],
"timestamp" : 1489096425069,
"clientToken" : "client-001"
}
```

ここで、inProgressJobs と queuedJobs は、IN_PROGRESS または QUEUED のステータスを持つ JobExecutionSummary オブジェクトのリストを返します。

StartNextPendingJobExecution

モノの次の保留中のジョブ実行を取得し、開始します (ステータスが IN_PROGRESS または QUEUED)。

- ステータスが IN_PROGRESS のジョブの実行が最初に返されます。
- ジョブの実行は、キューに保存された順に返されます。ジョブのターゲットグループにモノが追加または削除された場合は、既存のジョブ実行と比較して、新しいジョブ実行のロールアウト順序を確認します。
- 次の保留中のジョブの実行が QUEUED の場合、そのステータスは IN_PROGRESS に変更され、ジョブの実行の詳細は指定どおりに設定されます。
- 次の保留中のジョブの実行がすでに IN_PROGRESS である場合、そのステータスの詳細は変更されません。
- 保留中のジョブの実行がない場合、レスポンスに execution フィールドは含まれません。
- オプションで、stepTimeoutInMinutes プロパティの値を設定してステップタイマーを作成できます。UpdateJobExecution を実行してこのプロパティの値を更新しない場合、ステップタイマーが時間切れになると、ジョブの実行がタイムアウトになります。

この API を呼び出すには、$aws/things/*thingName*/jobs/start-next にメッセージを発行します。

リクエストペイロード:

```
{
"statusDetails": {
    "string": "job-execution-state"
    ...
},
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

statusDetails

ジョブ実行のステータスについて説明する名前と値のペアの集合。指定しない場合、statusDetails は変更されません。

stepTimeOutInMinutes

このデバイスがこのジョブの実行を終了する必要のある時間を指定します。このタイマーが時間切れになるか、再設定される (UpdateJobExecution を呼び出し、ステータスを IN_PROGRESS に設定して、stepTimeoutInMinutes フィールドで新しいタイムアウト値を指定する) までに、ジョブの実行ステータスが終了状態に設定されない場合、ジョブの実行ステータスは TIMED_OUT に設定されます。このタイムアウトを設定しても、ジョブの作成時に指定し

たジョブの実行タイムアウト (CreateJob で timeoutConfig フィールドを使用することにより) には影響を与えません。

このパラメータの有効な値の範囲は 1 ～ 10,080 (1 分～ 7 日) です。-1 の値も有効で、現在のステップタイマー (先ほど UpdateJobExecutionRequest を使用して作成したもの) をキャンセルします。

メッセージブローカーは、特定のサブスクリプションがなくても $aws/things/*thingName*/jobs/start-next/accepted および $aws/things/*thingName*/jobs/start-next/rejected を発行します。ただし、クライアントがメッセージを受信するには、それらをリッスンしている必要があります。詳細については、[the note about Jobs API messages](Jobs API メッセージに関する注記) を参照してください。

レスポンスペイロード:

```
{
"execution" : JobExecutionData,
"timestamp" : timestamp,
"clientToken" : "string"
}
```

ここで、execution は JobExecution オブジェクトです。例:

```
{
"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

ジョブの実行に関する詳細情報を取得します。

jobId を $next に設定して、モノ (ステータスが IN_PROGRESS または QUEUED のもの) に対して保留中の次のジョブ実行を返すことができます。

この API を呼び出すには、$aws/things/*thingName*/jobs/*jobId*/get にメッセージを発行します。

リクエストペイロード:

```
{
"jobId" : "022",
"thingName" : "MyThing",
"executionNumber": long,
"includeJobDocument": boolean,
"clientToken": "string"
}
```

thingName

デバイスに関連付けられたモノの名前。

jobId

作成時にこのジョブに割り当てた一意の識別子。

または、$next を使用して、モノ (ステータスが IN_PROGRESS または QUEUED のもの) に対して保留中の次のジョブ実行を返すことができます。この場合は、ステータスが IN_PROGRESS のジョブの実行が最初に返されます。ジョブの実行は、作成された順に返されます。

executionNumber

(オプション) デバイスでジョブの実行を識別する番号。指定しない場合、最新のジョブ実行が返されます。

includeJobDocument

(オプション) false に設定しない限り、レスポンスにはジョブドキュメントが含まれます。デフォルトは true です。

メッセージブローカーは、特定のサブスクリプションがなくても $aws/things/*thingName*/jobs/*jobId*/get/accepted および $aws/things/*thingName*/jobs/*jobId*/get/rejected を発行します。ただし、クライアントがメッセージを受信するには、それらをリッスンしている必要があります。詳細については、[the note about Jobs API messages](Jobs API メッセージに関する注記) を参照してください。

レスポンスペイロード:

```
{
"execution" : JobExecutionData,
"timestamp": "timestamp",
"clientToken": "string"
}
```

ここで、execution は JobExecution オブジェクトです。

UpdateJobExecution

ジョブ実行のステータスを更新します。オプションで、ステップタイマーを作成するには、stepTimeoutInMinutes プロパティの値を設定します。UpdateJobExecution を再び実行してこのプロパティの値を更新しない場合、ステップタイマーが時間切れになると、ジョブの実行がタイムアウトになります。

この API を呼び出すには、$aws/things/*thingName*/jobs/*jobId*/update にメッセージを発行します。

リクエストペイロード:

```
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"executionNumber": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

status

ジョブ実行の新しいステータス (IN_PROGRESS、FAILED、SUCCEEDED、または REJECTED)。これはすべての更新時に指定する必要があります。

statusDetails

ジョブ実行のステータスについて説明する名前と値のペアの集合。指定しない場合、statusDetails は変更されません。

expectedVersion

ジョブ実行の予想される現在のバージョン。ジョブの実行を更新するたびに、そのバージョンがインクリメントされます。 AWS IoT Jobs サービスに保存されているジョブ実行のバージョンが一致しない場合、更新はVersionMismatch工ラーで拒否されます。現在のジョブ実行ステータスデータを含む ErrorResponse も返されます。(これにより、ジョブ実行ステータスデータを取得するために別の DescribeJobExecution リクエストを実行する必要はありません。)

executionNumber

(オプション) デバイスでジョブの実行を識別する番号。指定しない場合、最新のジョブ実行が使用されます。

includeJobExecutionState

(オプション) これが含まれ、true に設定されている場合、レスポンスには JobExecutionState フィールドが含まれます。デフォルトは false です。

includeJobDocument

(オプション) これが含まれ、true に設定されている場合、レスポンスには JobDocument が含まれます。デフォルトはfalseです。

stepTimeoutInMinutes

このデバイスがこのジョブの実行を終了する必要のある時間を指定します。タイマーが期限切れになるかタイマーがリセットされる前にジョブ実行ステータスが終了状態に設定されない場合、ジョブ実行ステータスは TIMED_OUT に設定されます。このタイムアウトを設定または再設定しても、ジョブの作成時に指定したジョブの実行タイムアウトには影響を与えません。

メッセージブローカーは、特定のサブスクリプションがなくても $aws/things/*thingName*/jobs/*jobId*/update/accepted および $aws/things/*thingName*/jobs/*jobId*/update/rejected を発行します。ただし、クライアントがメッセージを受信するには、それらをリッスンしている必要があります。詳細については、[the note about Jobs API messages](Jobs API メッセージに関する注記) を参照してください。

レスポンスペイロード:

```
{
"executionState": JobExecutionState,
"jobDocument": "string",
"timestamp": timestamp,
"clientToken": "string"
}
```

`executionState`

JobExecutionState オブジェクト。

`jobDocument`

ジョブドキュメントオブジェクト。

> **Note**
>
> MQTT レスポンスでは、`jobDocument` フィールドは JSON オブジェクトです。HTTP レスポンスの場合は、JSON オブジェクトの文字列表現です。

`timestamp`

メッセージが送信されたときの、エポックからの秒単位の時間。

`clientToken`

リクエストとレスポンスを相関させるために使用されるクライアントトークン。

MQTT プロトコルを使用する場合、以下の更新を実行することもできます。

JobExecutionsChanged

あるジョブの実行中のジョブのリストに、ジョブの実行が追加または削除されるたびに送信されます。

トピックを使用します。

`$aws/things/`*`thingName`*`/jobs/notify`

メッセージペイロード:

```
{
```

```
"jobs" : {
    "JobExecutionState": [ JobExecutionSummary ... ]
         },
    "timestamp": timestamp
}
```

NextJobExecutionChanged

DescribeJobExecution に対して jobId $next で定義されている、モノに対する保留されているジョブ実行のリストで、次の順番のジョブに変更があったときに送信されます。このメッセージは、次のジョブの実行の詳細が変更されたときには送信されません。これは、jobId $next とともに DescribeJobExecution で返される次のジョブが変更されたときだけです。ステータスが QUEUED のジョブ実行 J1 と J2 を考えてみましょう。保留中のジョブの実行リストの次に J1 が表示されます。J1 の状態が変更されないまま J2 のステータスが IN_PROGRESS に変更された場合、この通知が送信され、J2 の詳細が含まれます。

トピックを使用します。

$aws/things/*thingName*/jobs/notify-next

メッセージペイロード:

```
{
"execution" : JobExecution,
"timestamp": timestamp,
}
```

ジョブデバイス MQTT API

デバイスは、ポート 443 の HTTP 署名バージョン 4 を使用して AWS IoT ジョブと通信できます。これは、 AWS SDKs と CLI で使用される方法です。これらのツールの詳細については、「AWS CLI コマンドリファレンス: iot-jobs-data」または「AWS SDK とツール」を参照してください。

ジョブを実行するデバイスでは、次のコマンドを使用できます。MQTT プロトコルでの API オペレーションの使用方法については、「ジョブデバイス MQTT API オペレーション」を参照してください。

GetPendingJobExecutions

特定のモノについて、ターミナル状態にないすべてのジョブのリストを取得します。

HTTPS request

```
GET /things/thingName/jobs
```

レスポンス:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ]
}
```

詳細については、「GetPendingJobExecutions」を参照してください。

CLI syntax

```
aws iot-jobs-data get-pending-job-executions \
--thing-name <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 形式:

```
{
"thingName": "string"
}
```

詳細については、「get-pending-job-executions」を参照してください。

StartNextPendingJobExecution

モノ (ステータスが IN_PROGRESS または QUEUED のもの) に対する次の保留中のジョブ実行を取得し、開始します。

- ステータスが IN_PROGRESS のジョブの実行が最初に返されます。
- ジョブの実行は、作成された順に返されます。
- 次の保留中のジョブの実行が QUEUED の場合、そのステータスは IN_PROGRESS に変更され、ジョブの実行のステータスの詳細は指定どおりに設定されます。
- 次の保留中のジョブの実行がすでに IN_PROGRESS である場合、そのステータスの詳細は変更されません。

- 保留中のジョブの実行がない場合、レスポンスに execution フィールドは含まれません。
- オプションで、stepTimeoutInMinutes プロパティの値を設定してステップタイマーを作成できます。UpdateJobExecution を実行してこのプロパティの値を更新しない場合、ステップタイマーが時間切れになると、ジョブの実行がタイムアウトになります。

HTTPS request

以下に構文例を示します。

```
PUT /things/thingName/jobs/$next
{
"statusDetails": {
    "string": "string"
    ...
},
"stepTimeoutInMinutes": long
}
```

詳細については、「StartNextPendingJobExecution」を参照してください。

CLI syntax

概要:

```
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 形式:

```
{
"thingName": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": long
}
```

詳細については、「start-next-pending-job-execution」を参照してください。

DescribeJobExecution

ジョブの実行に関する詳細情報を取得します。

jobId を $next に設定して、モノに対して保留中の次のジョブ実行を返すことができます。ジョブの実行ステータスは、QUEUED または IN_PROGRESS である必要があります。

HTTPS request

リクエスト:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument
```

レスポンス:

```
{
"execution" : JobExecution,
}
```

詳細については、「DescribeJobExecution」を参照してください。

CLI syntax

概要:

```
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 形式:

```
{
"jobId": "string",
```

```
"thingName": "string",
"includeJobDocument": boolean,
"executionNumber": long
}
```

詳細については、「describe-job-execution」を参照してください。

UpdateJobExecution

ジョブ実行のステータスを更新します。オプションで、stepTimeoutInMinutes プロパティの値を設定してステップタイマーを作成できます。UpdateJobExecution を再び実行してこのプロパティの値を更新しない場合、ステップタイマーが時間切れになると、ジョブの実行がタイムアウトになります。

HTTPS request

リクエスト:

```
POST /things/thingName/jobs/jobId
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"executionNumber": long
}
```

詳細については、「UpdateJobExecution」を参照してください。

CLI syntax

概要:

```
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 形式:

```
{
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": number,
"expectedVersion": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"executionNumber": long
}
```

詳細については、「update-job-execution」を参照してください。

AWS IoT ジョブによるユーザーとデバイスの保護

ユーザーがデバイスで AWS IoT ジョブを使用することを許可するには、IAM ポリシーを使用してアクセス許可を付与する必要があります。その後、デバイスは、 AWS IoT Core ポリシーを使用して、安全に接続し AWS IoT、ジョブ実行を受信し、実行ステータスを更新することで承認される必要があります。

AWS IoT ジョブに必要なポリシータイプ

次の表に、認可に使用する必要があるさまざまなポリシーのタイプを示します。必要なポリシーの詳細については、「Authorization」を参照してください。

必要なポリシータイプ

ユースケース	プロトコル	認証	コントロールプレーン/データプレーン	ID のタイプ	必要なポリシータイプ
管理者、オペレーター、またはクラウドサービスがジョブで安全に作業できるように承認する	HTTPS	AWS 署名バージョン 4 認証 (ポート 443)	コントロールプレーンとデータプレーンの両方	Amazon Cognito ID、IAM、またはフェデレーティッドユーザー	IAM ポリシー
ジョブで安全に使用できるようにIoT デバイスを承認する	MQTT/HTTPS	TCP または TLS 相互認証 (ポート 8883 または 443)	データプレーン	X.509 証明書	AWS IoT Core ポリシー

コントロールプレーンとデータプレーンの両方で実行できる AWS IoT ジョブオペレーションを許可するには、IAM ポリシーを使用する必要があります。これらのオペレーションを行うためには、ID は AWS IoT で認証されている必要があり、Amazon Cognito ID または IAM ユーザー、グループ、ロール である必要があります。認証の詳細については、「認証」を参照してください。

これで、 AWS IoT Core ポリシーを使用してデバイスゲートウェイに安全に接続することで、デバイスがデータプレーンで承認されるようになりました。デバイスゲートウェイを使用すると、デバイスは安全に と通信し AWS IoT、ジョブ実行を受信し、ジョブ実行ステータスを更新できます。デバイス通信は、セキュアな MQTT または HTTPS での発行 通信プロトコルを使用することで保護されます。これらのプロトコルはX.509 クライアント証明書、 が提供する AWS IoT を使用してデバイス接続を認証します。

以下は、ユーザー、c loud サービス、およびデバイスに AWS IoT ジョブの使用を許可する方法を示しています。コントロールプレーンとデータプレーンの API オペレーションの詳細については、AWS IoT ジョブ API オペレーション を参照してください。

トピック

- ユーザーとクラウドサービスに AWS IoT ジョブの使用を許可する
- データプレーンで AWS IoT ジョブを安全に使用することをデバイスに許可する

ユーザーとクラウドサービスに AWS IoT ジョブの使用を許可する

ユーザーとクラウドサービスを承認するには、コントロールプレーンとデータプレーンの両方で IAM ポリシーを使用する必要があります。ポリシーは HTTPS プロトコルと一緒に使用し、ユーザー認証には AWS 署名バージョン 4 認証 (ポート 443) を使用する必要があります。

> **Note**
>
> AWS IoT Core ポリシーをコントロールプレーンで使用することはできません。ユーザーおよびクラウドサービスの承認には、IAM ポリシーのみが使用されます。必要なポリシータイプの詳細については、「AWS IoT ジョブに必要なポリシータイプ」を参照してください。

IAM ポリシーは、ポリシーステートメントを含む JSON ドキュメントです。ポリシーステートメントでは、効果、アクション、およびリソースの各要素を使用して、リソース、許可または拒否するアクション、およびアクションが許可または拒否される条件を指定します。詳細については、IAM ユーザーガイドの「IAM JSON ポリシー要素のリファレンス」を参照してください。

> **Warning**
>
> IAM ポリシーや ポリシーなど`"Action": ["iot:*"]`では、ワイルドカードアクセス許可を使用しないことをお勧めします AWS IoT Core 。ワイルドカードアクセス許可の使用は、セキュリティ上のベストプラクティスとして推奨されません。詳細については、「アクセス許可が過度に寛容なAWS IoT ポリシー」を参照してください。

コントロールプレーンにおける IAM ポリシー

コントロールプレーンでは、IAM ポリシーは、対応するジョブ API オペレーションを承認するために、アクションと一緒に`iot:` プレフィックスを使用します。例えば、`iot:CreateJob` ポリシーアクションは、`CreateJob` API を使用するアクセス許可をユーザーに付与します。

ポリシーアクション

次の表に API アクションを使用するための IAM ポリシーアクションとアクセス許可のリストを示します。リソースタイプの詳細については、「 で定義されるリソースタイプ AWS IoT」を参照してください。アクションの詳細については AWS IoT 、「 で定義されるアクション AWS IoT」を参照してください。

コントロールプレーンにおける IAM ポリシーアクション

ポリシーアクション	API オペレーション:	リソースタイプ	説明
iot:AssociateTargetsWithJob	AssociateTargetsWithJob	• ジョブ • thing • thinggroup	グループを連続ジョブに関連付けるためのアクセス許可を表します。iot:AssociateTargetsWithJob アクセス許可は、ターゲットの関連付けのリクエストが行われるたびに確認されます。
iot:CancelJob	CancelJob	ジョブ	ジョブをキャンセルするアクセス許可を表します。iot:CancelJob アクセス許可は、ジョブのキャンセルリクエストが行われるたびに確認されます。
iot:CancelJobExecution	CancelJobExecution	• ジョブ • thing	ジョブの実行をキャンセルするアクセス許可を表します。iot: CancelJobExecution アクセス許可は、ジョブの実行のキャンセルリクエストが行われるたびに確認されます。
iot:CreateJob	CreateJob	• ジョブ • thing • thinggroup • jobtemplate	ジョブを作成するためのアクセス許可を表します。iot: CreateJob アクセス許可は、ジョブの作成リクエストが行われるたびに確認されます。

ポリシーアクション	API オペレーション:	リソースタイプ	説明
		• パッケージ	
iot:CreateJobTemplate	CreateJobTemplate	• ジョブ • jobtemplate • パッケージ	ジョブテンプレートを作成するためのアクセス許可を表します。iot: CreateJobTemplate アクセス許可は、ジョブテンプレートの作成リクエストが行われるたびに確認されます。
iot:DeleteJob	DeleteJob	ジョブ	ジョブを削除するアクセス許可を表します。iot: DeleteJob アクセス許可は、ジョブの削除リクエストが行われるたびに確認されます。
iot:DeleteJobTemplate	DeleteJobTemplate	jobtemplate	ジョブテンプレートを削除するアクセス許可を表します。iot: CreateJobTemplate アクセス許可は、ジョブテンプレートの削除リクエストが行われるたびに確認されます。
iot:DeleteJobExecution	DeleteJobTemplate	• ジョブ • thing	ジョブの実行を削除するアクセス許可を表します。iot: DeleteJobExecution アクセス許可は、ジョブの実行の削除リクエストが行われるたびに確認されます。
iot:DescribeJob	DescribeJob	ジョブ	ジョブを説明するアクセス許可を表します。iot: DescribeJob アクセス許可は、ジョブの説明のリクエストが行われるたびに確認されます。

ポリシーアクション	API オペレーション:	リソースタイプ	説明
iot:DescribeJobExecution	DescribeJobExecution	• ジョブ • thing	ジョブの実行を説明するアクセス許可を表します。iot: DescribeJobExecution アクセス許可は、ジョブの実行の説明のリクエストが行われるたびに確認されます。
iot:DescribeJobTemplate	DescribeJobTemplate	jobtemplate	ジョブテンプレートを説明するためのアクセス許可を表します。iot: DescribeJobTemplate アクセス許可は、ジョブテンプレートの説明のリクエストが行われるたびに確認されます。
iot:DescribeManagedJobTemplate	DescribeManagedJobTemplate	jobtemplate	マネージドジョブテンプレートを説明するアクセス許可を表します。iot: DescribeManagedJobTemplate アクセス許可は、マネージドジョブテンプレートの説明のリクエストが行われるたびに確認されます。
iot:GetJobDocument	GetJobDocument	ジョブ	ジョブのジョブドキュメントを取得するアクセス許可を表します。iot:GetJobDocument アクセス許可は、ジョブドキュメントの取得リクエストが行われるたびに確認されます。
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	ジョブ	ジョブのジョブ実行を一覧表示するためのアクセス許可を表します。iot:ListJobExecutionsForJob アクセス許可は、ジョブ実行を一覧表示するリクエストが行われるたびに確認されます。

ポリシーアクション	API オペレーション:	リソースタイプ	説明
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	thing	ジョブのジョブ実行を一覧表示するためのアクセス許可を表します。iot:ListJobExecutionsForThing アクセス許可は、モノのジョブ実行を一覧表示するリクエストが行われるたびに確認されます。
iot:ListJobs	ListJobs	なし	ジョブを一覧表示するためのアクセス許可を表します。iot:ListJobs アクセス許可は、ジョブを一覧表示するリクエストが行われるたびに確認されます。
iot:ListJobTemplates	ListJobTemplates	なし	ジョブテンプレートを一覧表示するためのアクセス許可を表します。iot:ListJobTemplates アクセス許可は、ジョブテンプレートを一覧表示するリクエストが行われるたびに確認されます。
iot:ListManagedJobTemplates	ListManagedJobTemplates	なし	マネージドジョブテンプレートを一覧表示するアクセス許可を表します。iot:ListManagedJobTemplates アクセス許可は、マネージドジョブテンプレートを一覧表示するリクエストが行われるたびに確認されます。
iot:UpdateJob	UpdateJob	ジョブ	ジョブを更新するアクセス許可を表します。iot:UpdateJob アクセス許可は、ジョブの更新リクエストが行われるたびに確認されます。

ポリシーアクション	API オペレーション:	リソースタイプ	説明
iot:TagResource	TagResource	• ジョブ • jobtemplate • thing	特定のリソースにタグを付けるアクセス許可を表します。
iot:UntagResource	UntagResource	• ジョブ • jobtemplate • thing	特定のリソースのタグを解除するためのアクセス許可を表します。

基本的な IAM ポリシーの例

IoT のモノとモノのグループに対して、ユーザーに以下のアクションを実行する許可を与える IAM ポリシーの例を以下に示します。

この例では、次のように置き換えます。

- AWS リージョンなどの の*#####*us-east-1。
- *account-id* は、 などの AWS アカウント 番号で指定します57EXAMPLE833。
- *Thing-group-name* をジョブのターゲットとなる IoT のモノグループの名前 (例: FirmwareUpdateGroup) に。
- *thing-name* をジョブのターゲットとなる IoT のモノの名前 (例: MyIoTThing) に。

```
{
    "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 ベースの認可用の IAM ポリシーの例

プリンシパルに対して、特定の IP アドレスからのコントロールプレーンエンドポイントへの API コールを制限できます。許可できる IP アドレスを指定するには、IAM ポリシーの Condition 要素で、aws:SourceIp グローバル条件キーを使用します。

この条件キーを使用すると、 など、ユーザーに代わってこれらの API コールを行うことを他の AWS のサービスへのアクセスを拒否することもできます AWS CloudFormation。これらのサービスへのアクセスを許可するには、aws:ViaAWSService グローバル条件キーを aws:SourceIp キーで 使用します。これにより、送信元 IP アドレスのアクセス制限は、プリンシパルによって直接行われた要求にのみ適用されます。詳細については、AWS「: ソース IP AWS に基づいて へのアクセスを拒否する」を参照してください。

次の例は、コントロールプレーンエンドポイントへの API コールを特定の IP アドレスにのみ許可する方法を示しています。aws:ViaAWSService キーを true に設定します。これにより、ユーザーに代わって API コールを行うことを他のサービスに許可します。

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

データプレーンにおける IAM ポリシー

データプレーンの IAM ポリシーでは、ユーザーが実行できるジョブ API オペレーションを承認するために iotjobsdata: プレフィックスを使用します。データプレーンでは、iotjobsdata:DescribeJobExecution ポリシーアクションを使用することで、ユーザーに DescribeJobExecution API を使用するアクセス許可を付与することができます。

> **Warning**
>
> AWS IoT ジョブをデバイスのターゲットにする場合、データプレーンで IAM ポリシーを使用することは推奨されません。ユーザーがジョブを作成および管理するには、コントロールプレーンで IAM ポリシーを使用することをお勧めします。データプレーンでは、ジョブ実行

の取得と実行ステータスの更新をデバイスに承認するには、AWS IoT Core HTTPS プロトコルの ポリシー を使用します。

基本的な IAM ポリシーの例

承認する必要がある API オペレーションは、通常、CLI コマンドを入力して実行します。ユーザーが DescribeJobExecution オペレーションを実行した場合の例を以下に示します。

この例では、次のように置き換えます。

- AWS リージョンなどの の *#####*us-east-1。
- *account-id* は、 などの AWS アカウント 番号で指定します57EXAMPLE833。
- *thing-name* をジョブのターゲットとなる IoT のモノの名前 (例: myRegisteredThing) に。
- *job-id* は、API を使用してターゲットとするジョブの一意の識別子です。

```
aws iot-jobs-data describe-job-execution \
    --endpoint-url "https://account-id.jobs.iot.region.amazonaws.com" \
    --job-id jobID --thing-name thing-name
```

以下に、このアクションを承認する IAM ポリシーの例を示します。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iotjobsdata:DescribeJobExecution"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
    }
}
```

IP ベースの認可用の IAM ポリシーの例

プリンシパルに対して、特定の IP アドレスからのデータプレーンエンドポイントへの API コールを制限できます。許可できる IP アドレスを指定するには、IAM ポリシーの Condition 要素で、aws:SourceIp グローバル条件キーを使用します。

この条件キーを使用すると、 など、ユーザーに代わってこれらの API コールを行うことを他の AWS のサービスへのアクセスを拒否することもできます AWS CloudFormation。これらのサービスへのアクセスを許可するには、aws:ViaAWSService グローバル条件キーを aws:SourceIp 条件キーで使用します。これにより、IP アドレスのアクセス制限は、プリンシパルによって直接行われた要求にのみ適用されます。詳細については、AWS「: ソース IP AWS に基づいて へのアクセスを拒否する」を参照してください。

次の例は、データプレーンエンドポイントへの API コールを特定の IP アドレスにのみ許可する方法を示しています。

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iotjobsdata:*"
        ],
        "Resource": [
            "*"
        ],
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "123.45.167.89"
        },
            "Bool": {
                "aws:ViaAWSService": "false"
        }
        }
    }]
}
```

次の例は、特定の IP アドレスまたはアドレス範囲に対して、データプレーンエンドポイントへの API コールを制限する方法を示しています。

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

コントロールプレーンとデータプレーン両方の IAM ポリシーの例

コントロールプレーンとデータプレーンの両方で API オペレーションを実行する場合、コントロールプレーンのポリシーアクションは `iot:` プレフィックスを使用する必要があり、データプレーンのポリシーアクションは `iotjobsdata:` プレフィックスを使用する必要があります。

たとえば、`DescribeJobExecution` API は、コントロールプレーンとデータプレーン両方で使用できます。コントロールプレーン上では、DescribeJobExecution API は、ジョブ実行を説明するために使用されます。データプレーン上では、DescribeJobExecution API は、ジョブ実行の詳細を取得するために使用されます。

以下の IAM ポリシーは、コントロールプレーンとデータプレーンの両方で `DescribeJobExecution` API を使用するアクセス許可をユーザーに付与します。

この例では、次のように置き換えます。

- AWS リージョンなどの の `#####us-east-1`。

- *account-id* は、 などの AWS アカウント 番号を使用します57EXAMPLE833。
- *thing-name* をジョブのターゲットなる IoT のモノの名前 (例: MyIoTThing) に。

```
{
    "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 リソースのタグ付けを承認する

作成、変更、使用できるジョブおよびジョブテンプレートをより適切に制御するために、ジョブまたはジョブテンプレートにタグをアタッチできます。タグは、所有権を識別し、請求グループに配置してタグをアタッチすることで、コストを割り当てるのにも役立ちます。

ユーザーが または を使用して作成したジョブまたはジョブテンプレートにタグを付ける場合 AWS マネジメントコンソール AWS CLI、IAM ポリシーはユーザーにタグを付けるアクセス許可を付与する必要があります。アクセス許可を付与するには、IAM ポリシーで `iot:TagResource` アクションを使用する必要があります。

> **Note**
>
> IAM ポリシーに `iot:TagResource` アクションが含まれていない場合、タグ付きの `CreateJob` または `CreateJobTemplate` は `AccessDeniedException` エラーを返します。

または を使用して作成したジョブまたはジョブテンプレートにタグを付ける場合 AWS CLI、IAM ポリシーはそれらのジョブ AWS マネジメントコンソール またはジョブテンプレートにタグを付けるアクセス許可を付与する必要があります。アクセス許可を付与するには、IAM ポリシーで `iot:TagResource` アクションを使用する必要があります。

リソースのタグ付けの詳細については、「AWS IoT リソースのタグ付け」を参照してください。

IAM ポリシーの例

タグ付けのアクセス許可を付与する次の IAM ポリシーの例を参照してください。

例 1

次のコマンドを実行してジョブを作成し、特定の環境にタグ付けするユーザー。

この例では、次のように置き換えます。

- AWS リージョンなどの の*#####*us-east-1。
- *account-id* は、 などの AWS アカウント 番号を使用します57EXAMPLE833。
- *thing-name* をジョブをターゲットにしている IoT のモノの名前 (例: MyIoTThing) に。

```
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
```

この例では、次の IAM ポリシーを使用する必要があります。

```
{
    "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/*"
        ]
    }
}
```

データプレーンで AWS IoT ジョブを安全に使用することをデバイスに許可する

デバイスがデータプレーン上の AWS IoT ジョブと安全にやり取りすることを許可するには、 AWS IoT Core policies. AWS IoT Core policies for jobs をポリシーステートメントを含む JSON ドキュメントとして使用する必要があります。これらのポリシーでは効果、アクション、およびリソースの各要素を使用し、IAM ポリシーと同様の規則に従います。要素の詳細については、IAM ユーザーガイドの「IAM JSON ポリシーの要素のリファレンス」を参照してください。

ポリシーは MQTT プロトコルと HTTPS プロトコルの両方で使用でき、デバイスを認証するには TCP または TLS 相互認証を使用する必要があります。以下に、さまざまな通信プロトコルでこれらのポリシーを使用する方法を示します。

> **Warning**
>
> IAM ポリシーや ポリシーなど`"Action": ["iot:*"]`では、ワイルドカードアクセス許可を使用しないことをお勧めします AWS IoT Core 。ワイルドカードアクセス許可の使用は、セキュリティ上のベストプラクティスとして推奨されません。詳細については、「アクセス許可が過度に寛容なAWS IoT ポリシー」を参照してください。

AWS IoT Core MQTT プロトコルの ポリシー

AWS IoT Core MQTT プロトコルの ポリシーは、ジョブデバイスの MQTT API アクションを使用するアクセス許可を付与します。MQTT API オペレーションは、ジョブコマンド用に予約されている

MQTT トピックを操作するために使用されます。これらの API オペレーションの詳細については、「ジョブデバイス MQTT API オペレーション」を参照してください。

MQTT ポリシーは、iot:Connect、iot:Publish、iot:Subscribe、iot:Receieve などのポリシーアクションを使用して、ジョブトピックと連携します。これらのポリシーにより、メッセージブローカーへの接続、ジョブズ MQTT トピックのサブスクライブ、デバイスとクラウド間の MQTT メッセージの送受信を行うことができます。これらのアクションの詳細については、「AWS IoT Core ポリシーアクション」を参照してください。

AWS IoT ジョブのトピックについては、「」を参照してくださいジョブのトピック。

基本的な MQTT ポリシーの例

次の例は、iot:Publish と iot:Subscribe を使用して、ジョブおよびジョブ実行を発行およびサブスクライブする方法を示します。

この例では、次のように置き換えます。

- AWS リージョンなどの の *#####*us-east-1。
- *account-id* は、 などの AWS アカウント 番号を使用します57EXAMPLE833。
- *thing-name* をジョブのターゲットとなる IoT のモノの名前 (例: MyIoTThing) に。

```
{
    "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 プロトコルの ポリシー

データプレーン上のAWS IoT Core ポリシーは、TLS 認証メカニズムを備えた HTTPS プロトコルを使用して、デバイスを認可することもできます。データプレーンでは、ポリシーは `iotjobsdata:` プレフィックスを使用して、デバイスが実行できるジョブ API オペレーションを承認します。例えば、`iotjobsdata:DescribeJobExecution` ポリシーアクションは、`DescribeJobExecution` API を使用するアクセス許可をユーザーに付与します。

> **Note**
>
> データプレーンポリシーアクションは、`iotjobsdata:` プレフィックスを使用する必要があります。コントロールプレーンでは、アクションは `iot:` プレフィックスを使用する必要があります。コントロールプレーンとデータプレーンの両方のポリシーアクションを使用する場合の IAM ポリシーの例については、「コントロールプレーンとデータプレーン両方の IAM ポリシーの例」を参照してください。

ポリシーアクション

次の表は、デバイスに API アクションの使用を許可するための AWS IoT Core ポリシーアクションとアクセス許可のリストを示しています。データプレーンで実行できる API オペレーションのリストについては、「ジョブデバイス MQTT API」を参照してください。

> **Note**
>
> これらのジョブ実行ポリシーアクションは、HTTP TLS エンドポイントにのみ適用されます。MQTT エンドポイントを使用する場合は、以前に定義された MQTT ポリシーアクションを使用する必要があります。

AWS IoT Core データプレーンでの ポリシーアクション

ポリシーアクション	API オペレーション:	リソースタイプ	説明
iotjobsdata:DescribeJobExecution	DescribeJobExecution	• ジョブ • thing	ジョブの実行を取得するアクセス許可を表します。iotjobsdata:DescribeJobExecution アクセス許可は、ジョブの実行の取得リクエストが行われるたびに確認されます。
iotjobsdata:GetPendingJobExecutions	GetPendingJobExecutions	thing	モノの終了のステータスではないジョブのリストを取得するアクセス権限を表します。iotjobsdata:GetPendingJobExecutions アクセス権限は、リストの取得リクエストが行われるたびに確認されます。
iotjobsdata:StartNextPendingJobExecution	StartNextPendingJobExecution	thing	モノに対して保留中の次のジョブ実行を取得および開始するアクセス権限を表します (つまり、ステータスが QUEUED のジョブ実行を IN_PROGRESS に更新します)。iotjobsdata:StartNextPendingJobExecution アクセス許可は、保留中の次のジョブ実行を開始するリクエストが行われるたびに確認されます。
iotjobsdata:UpdateJobExecution	UpdateJobExecution	thing	ジョブの実行を更新するアクセス権限を表します。iotjobsdata:UpdateJobExecution アクセス権限は、ジョブ実行の状態の更新リクエストが行われるたびに確認されます。

基本的なポリシーの例

以下は、任意のリソースのデータプレーン API オペレーションでアクションを実行するアクセス許可を付与する AWS IoT Core ポリシーの例です。特定のリソース (IoT のモノなど) にポリシーのスコープを設定することができます。例では、以下を置き換えます。

- AWS リージョン などの を使用する *#####*us-east-1。
- *account-id* は、 などの AWS アカウント 番号を使用します57EXAMPLE833。
- *thing-name* を IoT のモノの名前 (例: MyIoTthing) に。

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

これらのポリシーを使用する必要がある場合の一例として、IoT デバイスが AWS IoT Core ポリシーを使用して、これらの API オペレーションの 1 つにアクセスする場合 (DescribeJobExecution APIの次の例のような場合) が挙げられます。

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument&namespaceId=namespaceId
 HTTP/1.1
```

AWS IoT ジョブの制限

AWS IoT ジョブには、 のサービスリソースまたはオペレーションの最大数に対応するサービスクォータまたは制限があります AWS アカウント。

トピック

- ジョブ実行の制限
- アクティブなジョブおよび同時ジョブの制限

ジョブ実行の制限

このセクションでは、 AWS IoT Device Managementのジョブ実行の制限について説明します。

> **Note**
>
> これらの制限は、AWS IoT Device Management の Service Quotas に関するドキュメントに記載されている Service Quotas の一部ではありません。

保留中のジョブ実行の数に関する情報を取得するには、 `GetPendingJobExecutions` API を使用するか、 AWS IoT ジョブの MQTT 予約済みトピックをサブスクライブして を受け取りますジョブ通知タイプ。

アカウントで保留中のジョブ実行の数は、スケジューリング設定を有効にして定期的なメンテナンスウィンドウを使用するかどうかによって異なります。

保留中のジョブ実行の最大数

API/通知名	説明	スケジューリング設定なし	スケジューリング設定あり
`ListNotification`	`ListNotification` は、古いジョブ実行が終了ステータスになる、または新しいジョブ実行がキューに登録される、あるいは非終了ステータスに変更されるたびに発行されます。`QUEUED` または `IN_PROGRESS` のいずれかの保留中のジョブ実行を最大 15 件表示できます。	10	15 (メンテナンスウィンドウの間は、最大 5 つのジョブ実行のみが `ListNotification` に表示されます)。
`GetPendingJobExecutions`	`GetPendingJobExecutions` API を呼び出すと、まだ開始されていないジョブ実行のリストが返され、API コール後に開始できます。API は、最大 10 個の	10	15

API/通知名	説明	スケジューリング設定なし	スケジューリング設定あり
	保留中のジョブ実行を返すことができます。 • 保留中の 10 件のジョブ実行のうち、IN_PROGRESS の実行は結果から除外されます。 • 保留中の 10 件のジョブ実行のうち、ジョブが SCHEDULED ステータスのものは、結果から除外されます。		

アクティブなジョブおよび同時ジョブの制限

このセクションでは、アクティブなジョブと同時ジョブ、およびそれらに適用される制限について詳しく説明します。

アクティブなジョブとアクティブなジョブの制限

AWS IoT コンソールまたは CreateJob API を使用してジョブを作成すると、ジョブのステータスは に変わりますIN_PROGRESS。進行中のジョブはすべて アクティブなジョブ であり、アクティブなジョブの制限にカウントされます。これには、新しいジョブ実行をロールアウトしているジョブ、またはデバイスがジョブ実行を完了するのを待っているジョブが含まれます。この制限は、連続ジョブとスナップショットジョブの両方に適用されます。

同時ジョブとジョブの同時実行制限

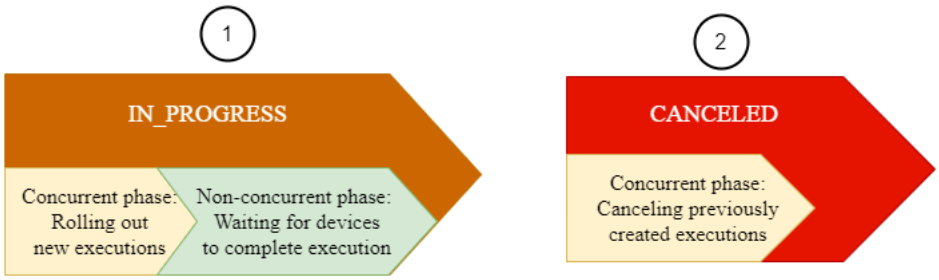
新しいジョブ実行をロールアウトしている進行中のジョブ、または以前に作成したジョブ実行をキャンセルしているジョブは同時ジョブであり、ジョブの同時実行数の制限にカウントされます。 AWS IoT ジョブは、1 分あたり 1000 デバイスの割合でジョブ実行を迅速にロールアウトおよびキャンセルできます。各ジョブは concurrent であり、ジョブの同時実行数の制限の対象になるのは、短期間だけです。ジョブの実行がロールアウトまたはキャンセルされると、ジョブは同時実行されなくなり、ジョブの同時実行制限にはカウントされません。ジョブの同時実行を使用して、デバイスがジョブの実行を完了するのを待っている間に、多数のジョブを作成できます。

> **Note**
>
> オプションのスケジュール設定が選択されているジョブで、メンテナンスウィンドウ中に実行されるようにスケジュールしたジョブドキュメントのロールアウトが選択した `startTime` に達し、さらにジョブの同時実行数が上限に達した場合、そのスケジュールされたジョブのステータスは `CANCELED` になります。

ジョブが同時に実行されるかどうかを判断するには、 AWS IoT コンソールから、または `DescribeJob`または `ListJob` API を使用して、ジョブの `IsConcurrent`プロパティを使用できます。この制限は、連続ジョブとスナップショットジョブの両方に適用されます。

のアクティブなジョブとジョブ同時実行制限、およびその他の AWS IoT ジョブクォータを表示 AWS アカウント し、制限の引き上げをリクエストするには、「」のAWS IoT 「デバイス管理エンドポイントとクォータ」を参照してください AWS 全般のリファレンス。

次の図は、ジョブの同時実行が、進行中のジョブとキャンセルされるジョブにどのように適用されるかを示しています。



> **Note**
>
> オプション `SchedulingConfig` 付きの新規のジョブは、初期ステータス `SCHEDULED` を維持し、選択した `startTime` に達すると `IN_PROGRESS` に更新されます。オプション `SchedulingConfig` 付きの新規のジョブが選択した `startTime` に達して `IN_PROGRESS` に更新されると、アクティブなジョブの制限とジョブの同時実行数の制限にカウントされます。ステータスが `SCHEDULED` のジョブはアクティブなジョブの制限にはカウントされますが、ジョブの同時実行数の制限にはカウントされません。

次の表では、アクティブなジョブと同時ジョブ、およびジョブ状態の同時フェーズと非同時フェーズに適用される制限を示しています。

アクティブなジョブおよび同時ジョブの制限

ジョブのステータス	[Phase] (フェーズ)	アクティブなジョブ制限	ジョブ同時実行数の制限
SCHEDULED	非同時フェーズ: AWS IoT ジョブは、スケジュールされたジョブ`startTime` がデバイスへのジョブ実行通知を開始するのを待ちます。このフェーズのジョブは、アクティブなジョブの制限にのみカウントされ、`IsConcurrent` プロパティは false に設定されています。	申請済み	適用されません
IN_PROGRESS	同時フェーズ: AWS IoT Jobs はジョブの作成リクエストを受け入れ、デバイスへのジョブ実行通知のロールアウトを開始します。このフェーズのジョブは同時実行され、`IsConcurrent` プロパティが true に設定され、アクティブなジョブとジョブの同時実行制限の両方に対してカウントされます。	申請済み	申請済み
	非同時フェーズ: AWS IoT ジョブは、デバイスがジョブ実行の結果を報告するのを待ちます。このフェーズのジョブは、アクティブなジョブの制限にのみカウントされ、`IsConcurrent` プロパティは false に設定されています。	申請済み	適用されません
Canceled	同時フェーズ: AWS IoT Jobs はジョブをキャンセルするリクエストを受け入れ、デバイス用に以前に作成されたジョブ実行のキャンセルを開始します。このフェーズのジョブは同時実行され、`IsConcurrent` プロパティを true に設定します。ジョブとジョブ実行がキ	適用されません	申請済み

ジョブのステータス	[Phase] (フェーズ)	アクティブなジョブ制限	ジョブ同時実行数の制限
	ャンセルされると、ジョブは同時実行されなくなり、ジョブの同時実行制限にはカウントされません。		

> **Note**
>
> 定期的なメンテナンスウィンドウの最大所要時間は 23 時間 50 分です。

AWS IoT Device Management コマンド

> **Important**
>
> このドキュメントでは、 AWS IoT Device Managementでコマンド機能を使用する方法について説明します。 AWS IoT FleetWiseでこの機能を使用する方法については、「リモートコマンド」を参照してください。
> お客様は、安全かつ適用法に準拠した方法でコマンドをデプロイすることに全責任を負います。お客様の責任に関する詳細については、AWS IoT サービスのAWS サービス規約をご覧ください。

AWS IoT Device Management コマンドを使用して、クラウドから に接続されているデバイスに命令を送信します AWS IoT。コマンドは一度に 1 つのデバイスをターゲットとし、デバイス側のログの取得やデバイスの状態変更の開始など、低レイテンシーで高スループットのアプリケーションに使用できます。

コマンドは、 によって管理される再利用可能なリソースです AWS IoT Device Management。これには、デバイスに発行される前に適用される設定が含まれています。電球のオンや車両のドアのロック解除など、特定のユースケースの一連のコマンドを事前に定義できます。

AWS IoT コマンド機能を使用すると、次のことができます。

- 静的または動的なペイロードを使用して再利用可能なコマンドテンプレートを作成し、特定のデバイスで実行します。
- AWS IoT モノまたは未登録の MQTT クライアントとして登録されたターゲットデバイス。
- 同じデバイスで複数のコマンドを同時に実行します。
- イベント通知を有効にし、デバイスがコマンドを処理するときに実行ステータスを追跡します。

以下のトピックでは、コマンドの作成、デバイスへのコマンドの送信、デバイスによって報告されたステータスの取得方法について説明します。

トピック

- クイックスタート
- コマンドの概念とステータス
- 高レベルのコマンドワークフロー

- コマンドの作成と管理
- コマンド実行を開始してモニタリングする
- コマンドリソースを非推奨にする

クイックスタート

最初のコマンドを送信するには、次の手順に従います。

1. 前提条件 - IoT Core、IoT Core に接続されたデバイス、 コマンド API オペレーションの IAM アクセス許可、デバイスにインストールされた MQTT クライアントライブラリにオンボードされていることを確認します。
2. コマンドの作成 - 静的ペイロード、またはデバイスアクションを指定するペイロードテンプレートを使用してコマンドを定義します。「コマンドリソースを作成する」を参照してください。
3. デバイスを識別する - デバイスを IoT モノとして登録するか、MQTT クライアント ID を書き留めます。「ターゲットデバイスに関する考慮事項」を参照してください。
4. アクセス許可の設定 - デバイスがコマンドを受信してレスポンスを発行できるようにする IAM ポリシーを設定します。
5. トピックをサブスクライブする - コマンドのリクエストおよびレスポンストピックをサブスクライブするようにデバイスを設定します。「コマンドのターゲットデバイスを選択して MQTT トピックをサブスクライブする」を参照してください。
6. コマンドを実行する - ターゲットデバイスでコマンドの実行を開始します。「コマンド実行を開始する」を参照してください。

一般的なユースケース:

- リモート診断 - ログの取得、診断の実行、デバイスのヘルスチェック
- 設定の更新 - 設定の変更、パラメータの更新、機能の切り替え
- デバイスコントロール - ロック/ロック解除、電源オン/オフ、モードの変更

コマンドの概念とステータス

AWS IoT コマンドを使用して、クラウドから接続されたデバイスに指示を送信します。この機能を使用する方法

1. デバイスで実行するために必要な設定を含むペイロードを使用してコマンドを作成します。

2. ペイロードを受信し、アクションを実行するターゲットデバイスを指定します。
3. ターゲットデバイスで コマンドを実行し、ステータス情報を取得します。問題のトラブルシューティングについては、CloudWatch logs」を参照してください。

ワークフローの詳細については、「高レベルのコマンドワークフロー」を参照してください。

トピック

- コマンドキーの概念
- コマンドの状態
- コマンド実行ステータス

コマンドキーの概念

以下の主要な概念は、 コマンド機能を理解するのに役立ちます。用語は、このドキュメント全体で一貫して使用されます。

- コマンド - デバイスの指示を定義する再利用可能なテンプレート
- 実行 - デバイスで実行されているコマンドのインスタンス
- モノの名前 - IoT レジストリに登録されているデバイスの識別子
- クライアント ID - 未登録デバイスの MQTT 識別子
- ペイロード - デバイスに送信される命令データ
- トピック - コマンド通信用の MQTT チャネル

コマンド

コマンドは、MQTT メッセージとしてクラウドから IoT デバイスに送信される手順です。ペイロードを受信すると、デバイスは指示を処理し、設定の変更、センサーの読み取り値の送信、ログのアップロードなどの対応するアクションを実行します。その後、デバイスは結果をクラウドに返し、リモートモニタリングと制御を可能にします。

名前空間

コマンドを作成するときは、その名前空間を指定します。 AWS IoT Device Management コマンドには、デフォルトの`AWS-IoT`名前空間を使用し、ペイロードまたは payloadTemplate を指定します。 AWS IoT FleetWise コマンドには、 `AWS-IoT-FleetWise`名前空間を使用します。詳細

については、「 AWS IoT FleetWise デベロッパーガイド」の「リモートコマンド」を参照してください。

ペイロード

コマンドを作成するときは、デバイスが実行する必要があるアクションを定義する静的ペイロードを指定します。ペイロードは、サポートされている任意の形式を使用できます。デバイスがペイロードを正しく解釈できるように、ペイロード形式タイプを指定することをお勧めします。MQTT5 プロトコルを使用するデバイスは、MQTT 標準に従って形式を識別できます。JSON または CBOR の形式インジケータは、 コマンドリクエストトピックで使用できます。

ペイロードテンプレート

ペイロードテンプレートは、指定したパラメータ値に基づいて実行時に異なるペイロードを生成するプレースホルダーを持つコマンドペイロードを定義します。たとえば、温度値ごとに個別のペイロードを作成する代わりに、温度プレースホルダーを使用して 1 つのテンプレートを作成し、実行時に値を指定します。これにより、複数の同様のペイロードが維持されなくなります。

ターゲットデバイス

コマンドを実行するには、モノの名前 (に登録されているデバイスの場合 AWS IoT) または MQTT クライアント ID (未登録デバイスの場合) を使用してターゲットデバイスを指定します。クライアント ID は、デバイスの接続に使用されるMQTTプロトコルで定義された一意の識別子です AWS IoT。詳細については、「ターゲットデバイスに関する考慮事項」を参照してください。

コマンドトのピック

コマンドを実行する前に、デバイスはコマンドリクエストトピックをサブスクライブする必要があります。コマンドを実行すると、ペイロードはこのトピックのデバイスに送信されます。実行後、デバイスはコマンドレスポンストピックに結果とステータスを発行します。詳細については、「コマンドトのピック」を参照してください。

コマンドの実行

実行は、ターゲットデバイスで実行されるコマンドのインスタンスです。実行を開始すると、ペイロードがデバイスに配信され、一意の実行 ID が生成されます。デバイスは コマンドを実行し、進捗状況を に報告します AWS IoT。デバイス側のロジックは、予約済みトピックに対する実行動作とステータスレポートを決定します。

パラメータ値の条件

ペイロードテンプレートを使用してコマンドを作成するときは、値条件を定義して、実行前にパラメータ値を検証します。値条件により、パラメータが要件を満たし、無効な実行が防止されます。

CommandParameterValue タイプでサポートされている演算子

数値型 (INTEGER、LONG、DOUBLE、UNSIGNEDLONG)

- EQUALS - 値は指定された数に等しくなければなりません
- NOT_EQUALS - 値は指定された数と等しくすることはできません
- GREATER_THAN - 値は指定された数より大きい必要があります
- GREATER_THAN_EQUALS - 値は指定された数以上である必要があります
- LESS_THAN - 値は指定された数より小さくする必要があります
- LESS_THAN_EQUALS - 値は指定された数以下である必要があります
- IN_RANGE - 値は指定された範囲内 (含む) である必要があります
- NOT_IN_RANGE - 値は指定された範囲外である必要があります (含む)
- IN_SET - 値は指定された数値のいずれかと一致する必要があります
- NOT_IN_SET - 値は指定された数値のいずれにも一致しない必要があります

文字列タイプ (STRING)

- EQUALS - 値は指定された文字列と等しくなければなりません
- NOT_EQUALS - 値は指定された文字列と等しくすることはできません
- IN_SET - 値は指定された文字列のいずれかと一致する必要があります
- NOT_IN_SET - 値は指定された文字列のいずれにも一致してはいけません

ブール型

- 値条件はサポートされていません

バイナリタイプ

- 値条件はサポートされていません

例: 温度制御コマンド

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

この例では、 temperatureパラメータは 60 ~ 80 (両端を含む) である必要があります。この範囲外の値の実行リクエストは検証に失敗します。

> **Note**
>
> 値条件は、StartCommandExecution API の呼び出し時に評価されます。失敗した検証はエラーを返し、実行の作成を防ぎます。

パラメータ値の優先度と評価

ペイロードテンプレートを使用してコマンド実行を開始する場合、パラメータ値は次の優先度を使用して解決されます。

1. 実行リクエストパラメータ - StartCommandExecutionリクエストで指定された値が最優先されます
2. コマンドのデフォルト値 - 実行リクエストでパラメータが指定されていない場合、パラメータの defaultValueが使用されます。
3. 値なし - どちらも指定されていない場合、実行リクエストの生成に必要なパラメータとして実行が失敗します

値条件は、優先度で上記で導出された最終的なパラメータ値と、実行の作成前に評価されます。検証が失敗した場合、実行リクエストはエラーを返します。

例: を使用した SetTemperature コマンド **defaultValue**

```
{
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "defaultValue": {"I": 72},
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {"numberRange": {"min": "60", "max": "80"}}
        }
      ]
    }
  ]
}
```

実行を開始する場合:

- リクエスト`"temperature": {"I": 75}`で を指定すると、75 が使用されます。
- 温度パラメータを省略すると、デフォルト値の 72 が使用されます。
- 両方の値が範囲条件 [60,80] に対して検証されます

コマンドの状態

のコマンドは AWS アカウント 、使用可能、非推奨、または保留中の 3 つの状態のいずれかになります。

使用可能

作成が成功すると、コマンドは使用可能状態になり、デバイスで実行できます。

非推奨

不要になったときは、非推奨のコマンドにマークを付けます。廃止されたコマンドは新しい実行を開始できませんが、保留中の実行は完了し続けます。新しい実行を有効にするには、 コマンドを Available 状態に復元します。

削除保留中

コマンドを削除対象としてマークすると、最大タイムアウト (デフォルト: 12 時間) より長く非推奨になると、自動的に削除されます。このアクションは永続的です。非推奨または非推奨がタイ

ムアウトより短い場合、コマンドは削除保留中状態になり、タイムアウトの有効期限が切れた後に削除されます。

コマンド実行ステータス

ターゲットデバイスで実行を開始すると、 CREATEDステータスになり、デバイスレポートに基づいて他のステータスに移行できます。ステータス情報を取得し、実行を追跡できます。

> **Note**
>
> デバイスで複数のコマンドを同時に実行できます。同時実行制御を使用して、デバイスごとの実行を制限し、オーバーロードを防止します。デバイスあたりの最大同時実行数については、AWS IoT Device Management「 コマンドクォータ」を参照してください。

次の表は、実行ステータスと実行の進行状況に基づく移行を示しています。

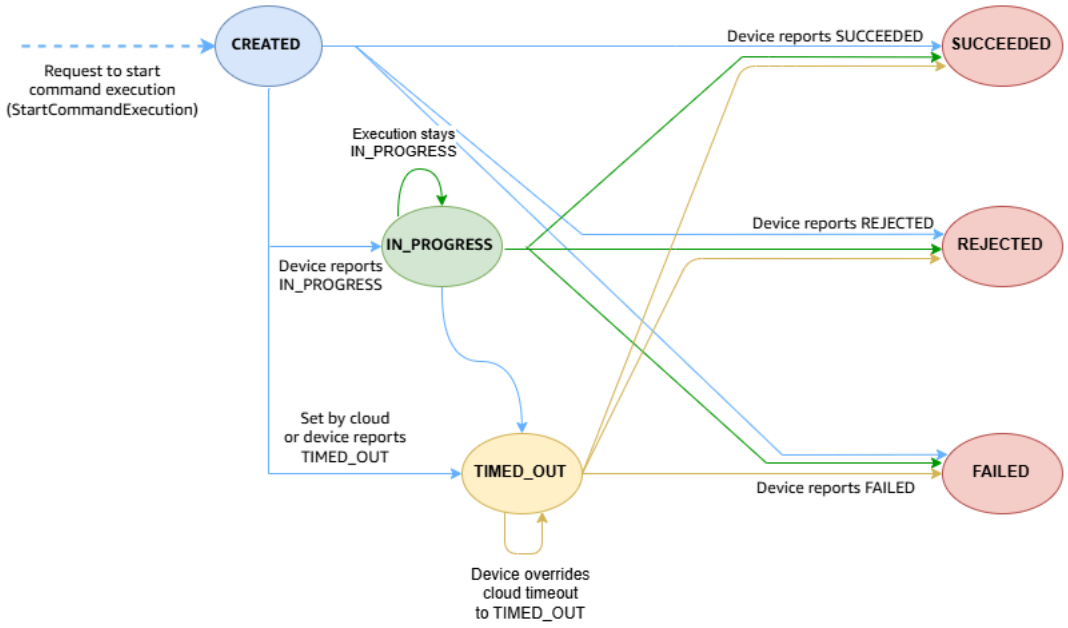
コマンド実行のステータスと開始元

コマンド実行ステータス	開始元はデバイスかクラウドか	実行の終了かどうか	許可されたステータスの移行
CREATED	Cloud	いいえ	• IN_PROGRESS • 成功 • FAILED • 拒否 • TIMED_OUT
IN_PROGRESS	デバイス	いいえ	• IN_PROGRESS • 成功 • FAILED • 拒否 • TIMED_OUT

コマンド実行ステータス	開始元はデバイスかクラウドか	実行の終了かどうか	許可されたステータスの移行
TIMED_OUT	デバイスとクラウド	いいえ	• 成功 • FAILED • 拒否 • TIMED_OUT
SUCCEEDED	デバイス	はい	該当しない
FAILED	デバイス	はい	該当しない
REJECTED	デバイス	はい	該当しない

デバイスは、コマンド予約 MQTT トピックを使用して、いつでもステータスと結果の更新を発行できます。追加のコンテキストを提供するために、デバイスは statusReason オブジェクトのフィールドreasonCodeと reasonDescriptionフィールドを使用できます。

次の図は、実行ステータスの遷移を示しています。



> **Note**
>
> がタイムアウト期間内にデバイスレスポンス AWS IoT を検出しない場合、 を一時的なステータスTIMED_OUTとして設定し、再試行と状態の変更を許可します。デバイスが を明示的に報告した場合TIMED_OUT、これはそれ以上の移行なしでターミナルステータスになります。詳細については、「非終了のコマンド実行」を参照してください。

以下のセクションでは、ターミナルと非ターミナルの実行とそのステータスについて説明します。

トピック

- 非終了のコマンド実行
- 終了のコマンド実行

非終了のコマンド実行

デバイスからの更新を受け入れることができる場合、実行は非ターミナルです。非ターミナル実行はアクティブと見なされます。次のステータスは非ターミナルです。

- CREATED

  AWS IoT コンソールから実行を開始するか、 StartCommandExecution API を使用すると、リクエストが成功するとステータスが に変更されますCREATED。このステータスから、実行は他の非ターミナルステータスまたはターミナルステータスに移行できます。

- IN_PROGRESS

  ペイロードを受信すると、デバイスは指示の実行と指定されたアクションの実行を開始できます。実行中、デバイスはコマンドレスポンストピックにレスポンスを発行し、ステータスを に更新できますIN_PROGRESS。からIN_PROGRESS、実行は 以外の任意のターミナルまたは非ターミナルステータスに移行できますCREATED。

  > **Note**
  >
  > UpdateCommandExecution API は、 IN_PROGRESSステータスで複数回呼び出すことができます。statusReason オブジェクトを使用して追加の実行の詳細を指定します。

- TIMED_OUT

クラウドとデバイスの両方がこのステータスをトリガーできます。CREATED または IN_PROGRESSステータスの実行は、次の理由で TIMED_OUT に変わる可能性があります。

- コマンドを送信すると、タイマーが開始されます。デバイスが指定された期間内に応答しない場合、クラウドはステータスを に変更しますTIMED_OUT。この場合、実行は非ターミナルです。
- デバイスは、ステータスを任意のターミナルステータスに上書きしたり、タイムアウトを報告してステータスを に設定したりできますTIMED_OUT。この場合、ステータスは のままですがTIMED_OUT、StatusReasonオブジェクトフィールドはデバイス情報に基づいて変わります。実行はターミナルになります。

詳細については、「タイムアウト値と TIMED_OUT 実行ステータス」を参照してください。

終了のコマンド実行

デバイスからの更新を受け入れなくなると、実行はターミナルになります。次のステータスは終了です。実行は、、CREATED、IN_PROGRESSまたは の非ターミナルステータスからターミナルステータスに移行できますTIMED_OUT。

- 成功

  デバイスが正常に実行を完了すると、コマンドレスポンストピックへのレスポンスを発行し、ステータスを に更新できますSUCCEEDED。

- FAILED

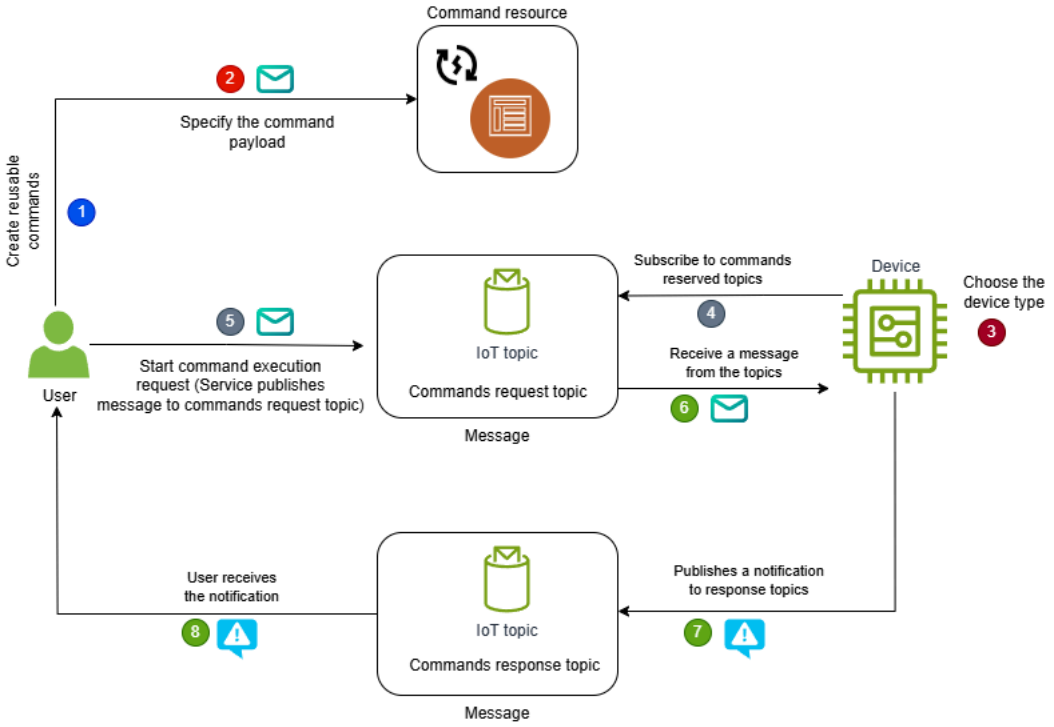
  デバイスが実行を完了できなかった場合、コマンドレスポンストピックにレスポンスを発行し、ステータスを に更新できますFAILED。statusReason オブジェクトの フィールドreasonCodeとreasonDescriptionフィールド、または CloudWatch ログを使用して、障害のトラブルシューティングを行います。

- 拒否

  デバイスが無効なリクエストまたは互換性のないリクエストを受信すると、ステータスが の UpdateCommandExecution API を呼び出すことができますREJECTED。statusReason オブジェクトの フィールドreasonCodeと reasonDescriptionフィールド、または CloudWatch ログを使用して、問題のトラブルシューティングを行います。

高レベルのコマンドワークフロー

このワークフローは、デバイスが AWS IoT Device Management コマンドとやり取りする方法を示しています。すべての HTTP API リクエストは、署名に Sigv4 認証情報を使用します。



ワークフローの概要

- コマンドの作成と管理
- コマンドのターゲットデバイスを選択して MQTT トピックをサブスクライブする
- ターゲットデバイスのコマンド実行を開始してモニタリングする
- (オプション) コマンドイベントの通知を有効にする

コマンドの作成と管理

デバイスのコマンドを作成および管理するには、次の手順を実行します。

1. コマンドリソースを作成する

   コマンドハブから、または API を使用してコマンドを作成します。 `CreateCommand`

2. ペイロードを指定する

   ペイロードを任意の形式で指定します。デバイスが正しく解釈されるようにコンテンツタイプを指定します。

   ペイロードテンプレートを持つ動的コマンドの場合、最終的なペイロードは、指定したパラメータを使用して実行時に生成されます。テンプレートは JSON 形式のみをサポートしますが、生成されたペイロードは JSON または CBOR として送信できます。

3. (オプション) 作成したコマンドを管理する

   作成後に表示名と説明を更新します。不要になったコマンドを非推奨としてマークするか、完全に削除します。ペイロード情報を変更するには、新しいコマンドを作成します。

コマンドのターゲットデバイスを選択して MQTT トピックをサブスクライブする

ターゲットデバイスを選択し、コマンドを受信してレスポンスを発行するための MQTT トピックを設定します。

1. コマンドのターゲットデバイスを選択する

   コマンドを受信して実行するターゲットデバイスを選択します。登録済みデバイスにはモノの名前、未登録デバイスにはクライアント ID を使用します。詳細については、「ターゲットデバイスに関する考慮事項」を参照してください。

2. AWS IoT デバイスポリシーを設定する

   実行を受信して更新を発行するアクセス許可を付与する IAM ポリシーを設定します。ポリシーの例サンプルの IAM ポリシーについては、「」を参照してください。

3. MQTT 接続を確立する

   デバイスをメッセージブローカーに接続し、リクエストとレスポンスのトピックにサブスクライブします。デバイスには アクセスiot:Connect許可が必要です。DescribeEndpoint API または describe-endpoint CLI コマンドを使用してデータプレーンエンドポイントを検索します。

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

このコマンドを実行すると、以下に示すように、アカウント固有のデータプレーンエンドポイントが返されます。

```
account-specific-prefix.iot.region.amazonaws.com
```

4. コマンドトピックをサブスクライブする

コマンドリクエストトピックをサブスクライブします。実行を開始すると、メッセージブローカーはこのトピックにペイロードを発行します。デバイスは コマンドを受け取り、処理します。

(オプション) レスポンストピック (accepted または rejected) をサブスクライブして、クラウドサービスがデバイスのレスポンスを承認したか拒否したかの確認を受け取ります。

この例では、次のように置き換えます。

- *<device>* を、ターゲットとするデバイスが IoT のモノとして登録されているか、MQTT クライアントとして指定されているかに応じて、thing または client に置き換えます。
- *<DeviceID>* をターゲットデバイスの一意の識別子に置き換えます。この ID には、一意の MQTT クライアント ID またはモノの名前を使用することができます。

> **Note**
>
> ペイロードタイプが JSON または CBOR でない場合は、コマンドリクエストトピックに *<PayloadFormat>* フィールドが存在しない可能性があります。ペイロード形式を取得するには、MQTT5 を使用して MQTT メッセージヘッダーから形式情報を取得することをお勧めします。詳細については、「コマンドトのピック」を参照してください。

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
$aws/commands/<devices>/<DeviceID>/executions/+/response/accepted/<PayloadFormat>
$aws/commands/<devices>/<DeviceID>/executions/+/response/rejected/<PayloadFormat>
```

ターゲットデバイスのコマンド実行を開始してモニタリングする

コマンドを作成し、コマンドのターゲットを指定したら、次の手順を実行してターゲットデバイスで実行を開始できます。

1. ターゲットデバイスでコマンド実行を開始する

   Command Hub から、またはアカウント固有のエンドポイントで `StartCommandExecution` API を使用して実行を開始します。`iot:Data-ATS` デュアルスタック (IPv4/IPv6) または IPv4 にのみ `iot:Jobs`を使用します。

   API は、ペイロードを コマンドリクエストトピックに発行します。

   > **Note**
   >
   > デバイスがオフラインで MQTT 永続セッションを使用している場合、 コマンドはメッセージブローカーで待機します。タイムアウト前にデバイスが再接続すると、 コマンドを処理して結果を発行できます。タイムアウトの有効期限が切れると、実行はタイムアウトし、ペイロードは破棄されます。

2. コマンド実行の結果を更新する

   デバイスはペイロードを受け取り、 コマンドを処理し、指定されたアクションを実行し、`UpdateCommandExecution`MQTT ベースの API を使用してコマンドレスポンストピックに結果を発行します。受け入れられたトピックと拒否されたトピックにサブスクライブすると、デバイスはクラウドサービスが応答を承諾したか拒否したかの確認を受け取ります。

   リクエストトピックで指定した内容に応じて、*<devices>* はモノまたはクライアントにすることができ、*<DeviceID>* は AWS IoT モノの名前または MQTT クライアント ID にすることができます。

   > **Note**
   >
   <PayloadFormat> は、コマンドレスポンストピックで JSON または CBOR のみを使用できます。

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/
response/<PayloadFormat>
```

3. (オプション) コマンド実行結果を取得する

   AWS IoT コンソールまたは を使用して実行結果を取得しますGetCommandExecution。デバイスは、最新の情報について、 コマンドレスポンストピックに結果を発行する必要があります。最終更新時刻、結果、完了時刻などの詳細を表示します。

(オプション) コマンドイベントの通知を有効にする

実行ステータスが変更されたときの通知のコマンドイベントにサブスクライブします。

1. トピックルールを作成する

   ステータス変更通知の コマンドイベントトピックにサブスクライブします。 AWS IoT コンソールまたは を使用して AWS Lambda、Amazon SQS や AWS Step Functions などの他の AWS IoT サービスにデバイスデータをルーティングするトピックルールを作成しますAWS IoT ルールの作成。

   この例では、*<CommandID>* を通知を受信するコマンドの識別子に置き換え、*<CommandExecutionStatus>* をコマンド実行のステータスに置き換えます。

   ```
   $aws/events/commandExecution/<CommandID>/<CommandExecutionStatus>
   ```

   > **Note**
   >
   > すべてのコマンドとコマンド実行ステータスの通知を受け取るには、ワイルドカード文字を使用し、次のトピックにサブスクライブします。

   ```
   $aws/events/commandExecution/+/#
   ```

2. コマンドイベントを受信して処理する

   コマンドプッシュ通知を管理し、サブスクライブされたイベントを使用してアプリケーションを構築します。

次のコードは、受信するコマンドイベント通知のサンプルペイロードを示しています。

```
{
    "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
}
```

コマンドの作成と管理

AWS IoT Device Management コマンドを使用して再利用可能なリモートアクションを設定するか、デバイスにすぐに指示を送信します。 AWS IoT コンソールまたは を使用して、 コマンドを作成および管理します AWS CLI。

コマンドオペレーションの作成と管理

- コマンドリソースを作成する
- コマンドに関する情報を取得する
- でコマンドを一覧表示する AWS アカウント
- コマンドリソースを更新する
- コマンドリソースを非推奨にする、または復元する
- コマンドリソースを削除する

コマンドリソースを作成する

コマンドを作成するときは、次の情報を入力します。

- 一般的な情報

ターゲットデバイスで実行するときにコマンドを識別する一意のコマンド ID を指定します。必要に応じて、管理用の表示名、説明、タグを指定します。

- ペイロード

  静的コマンドの場合は、デバイスアクションを定義するペイロードを指定します。正しいデバイス解釈のためにペイロード形式タイプを指定します。

  動的コマンドについては、「ペイロードテンプレート属性」を参照してください。

- ペイロードテンプレート

  動的コマンドの場合は、プレースホルダーとパラメータを含む payloadTemplate を指定します。オプションで defaultValueおよび 条件を指定します。 AWS IoT Device Management コマンドは、実行時にプレースホルダーを置き換えます。欠落しているパラメータは defaultValue を使用します。すべての値は、定義された条件を満たす必要があります。

  以下の大文字と小文字を区別するプレースホルダータイプがサポートされています。

  - ${aws:iot:commandexecution::parameter:*parameter1*} – という名前のパラメータの値のプレースホルダーparameter1。
  - ${aws:iot:commandexecution::executionTimeoutSec} – 実行時に提供されるコマンド実行タイムアウトパラメータのプレースホルダー。

ペイロードとコマンドのトピック

コマンドリザーブドトピックは、ペイロード形式タイプに基づく形式を使用します。

- application/json または application/cborコンテンツタイプの場合は、次のリクエストトピックを使用します。

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
  ```

- 他のコンテンツタイプまたは指定されていない形式の場合は、このリクエストトピックを使用します。ペイロード形式が MQTT メッセージヘッダーに表示されます。

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request
  ```

コマンドレスポンス Topic は、ペイロードタイプに関係なく jsonまたは cbor形式を使用します。*<PayloadFormat>* は jsonまたは である必要がありますcbor。

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

コマンドリソースの作成 (コンソール)

以下のセクションでは、ペイロード形式の考慮事項とコンソールからのコマンド作成について説明します。

トピック

- 静的コマンドペイロード形式
- 動的コマンドペイロード形式
- コマンドの作成方法 (コンソール)

静的コマンドペイロード形式

ペイロードは、最大 32 KB の任意の形式をサポートします。安全で正しいデバイス解釈のためにペイロード形式タイプを指定します。

形式 (例: `application/json`または `application/cbor`) を使用してペイロード`type/subtype`形式タイプを指定します。デフォルト: `application/octet-stream`。サポートされている形式については、「一般的な MIME タイプ」を参照してください。

動的コマンドペイロード形式

payloadTemplate は、少なくとも 1 つのプレースホルダー、最大 32 KB の有効な JSON である必要があります。

`AwsJsonSubstitution` プリプロセッサの場合、 AWS IoT Device Management コマンドはプリプロセッサ設定に基づいて JSON または CBOR 形式で通知を送信します。

コマンドの作成方法 (コンソール)

コンソールから コマンドを作成するには、Command Hub に移動し、次の手順に従います。

1. [コマンドを作成] を選択します。
2. 一意のコマンド ID を指定します。
3. (オプション) 表示名、説明、タグを指定します。
4. デバイスアクションを含むペイロードファイルをアップロードします。正しいデバイス解釈のためにペイロード形式タイプを指定します。

5. (オプション) プレースホルダーを含む JSON ペイロードテンプレートの場合、パラメータは編集のためにインラインテーブルに事前入力されます。
6. (オプション) パラメータ値タイプ (必須) 、デフォルト値、および条件を設定します。
7. [コマンドを作成] を選択します。

コマンドリソースの作成 (CLI)

`CreateCommand` API または `create-command` CLI コマンドを使用して コマンドを作成します。

トピック

- コマンドのペイロード
- サンプルの IAM ポリシー
- 静的コマンド作成の例
- 動的コマンド作成の例

コマンドのペイロード

静的ペイロードまたはペイロードテンプレートを指定します。静的ペイロードは base64 でエンコードされます。ペイロードテンプレートの場合、最終的なペイロードは実行時にパラメータ値を使用して生成します。デバイスはペイロードを処理し、指定されたアクションを実行します。正しいデバイス受信のためのペイロードコンテンツタイプを指定します。

> **Note**
>
> コマンドの作成後にペイロードを変更することはできません。ペイロードを変更する新しいコマンドを作成します。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、`CreateCommand` アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *`region`* を *`us-east-1`* などお客様ご自身の AWS リージョンに置き換えます。
- *`account-id`* を *`123456789012`* などの AWS アカウント 番号に置き換えます。

- *command-id* など、 AWS IoT コマンド ID の一意の識別子を持つ *LockDoor*。複数のコマンドを送信する場合は、IAM ポリシーの「Resource」セクションでこれらのコマンドを指定できます。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:CreateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

静的コマンド作成の例

次の例は、静的コマンドを作成する方法を示しています。用途に応じて、以下を置き換えてください。

- *<command-id>* をコマンドの一意の識別子で置き換えます。たとえば、家のドアをロックするには、 を指定できます*LockDoor*。UUID を使用することをお勧めします。英数字、「-」、「_」を使用することもできます。
- (オプション) *<display-name>* および *<description>* は、*Lock the doors of my home* など、コマンドのわかりやすい名前と意味を持つ説明を提供するために使用できるオプションのフィールドです。
- namespace は、コマンドの名前空間を指定するために使用できます。これは AWS-IoT である必要があります。でこの機能を使用する方法については AWS IoT FleetWise、「リモートコマンド」を参照してください。
- payload には、コマンドの実行時に使用するペイロードとそのコンテンツタイプに関する情報が含まれています。

```
aws iot create-command \
    --command-id <command-id> \
    --display-name <display-name> \
    --description <description> \
    --namespace AWS-IoT \
    --payload
 '{"content":"eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=","contentType":"application/json"}'
```

このコマンドを実行すると、コマンドの ID と ARN (Amazon リソース名) を含むレスポンスが生成されます。たとえば以下は、作成時に *LockDoor* コマンドを指定した場合の、コマンド実行の出力例です。

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor"
}
```

動的コマンド作成の例

次の例は、動的コマンドを作成する方法を示しています。用途に応じて、以下を置き換えてください。

- *<command-id>* をコマンドの一意の識別子で置き換えます。例えば、照明の電源ステータスを設定するには、 を指定できます*Light_Power_Status*。UUID を使用することをお勧めします。英数字、「-」、「_」を使用することもできます。
- (オプション) *<display-name>* および *<description>* は、*Turn a light ON or OFF* など、コマンドのわかりやすい名前と意味を持つ説明を提供するために使用できるオプションのフィールドです。
- namespace は、コマンドの名前空間を指定するために使用できます。これは AWS-IoT である必要があります。でこの機能を使用する方法については AWS IoT FleetWise、「リモートコマンド」を参照してください。
- payloadTemplate には、プレースホドルを含む JSON 形式の playoad テンプレートが含まれています。
- preprocessor には、payloadTemplateの処理方法を決定するプリプロセッサの設定が含まれています。
- mandatoryParameter には、payloadTemplate のプレースホルダーに対応するパラメータ、そのタイプ、デフォルト値、および条件が含まれます。

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

このコマンドを実行すると、コマンドの ID と ARN (Amazon リソース名) を含むレスポンスが生成されます。たとえば以下は、作成時に *Light_Power_Status* コマンドを指定した場合の、コマンド実行の出力例です。

```
{
    "commandId": "Light_Power_Status",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status"
}
```

コマンドに関する情報を取得する

コマンドを作成した後は、 AWS IoT コンソールから AWS CLIを使用してそのコマンドに関する情報を取得できます。取得できる情報は次のとおりです。

- コマンド ID、Amazon リソースネーム (ARN)、コマンドに指定した表示名と説明。
- ターゲットデバイス上でコマンドを実行できるかどうか、または非推奨なのか、削除されているかを示すコマンドの状態。
- 指定したペイロードまたは payloadTemplate。
- 指定したプリプロセッサ。
- 指定した mandatoryParameters。
- コマンドが作成され、最後にアップデートされた時刻。

コマンドリソースを取得する (コンソール)

コンソールからコマンドを取得するには、 AWS IoT コンソールの コマンドハブ に移動し、作成したコマンドを選択してその詳細を表示します。

コマンドの詳細に加えて、ターゲットデバイスでのコマンドの実行に関する情報を提供するコマンド履歴を確認できます。デバイスでこのコマンドを実行すると、このタブで実行に関する情報を確認できます。

コマンドリソースを取得する (CLI)

HTTP コントロールプレーン API `GetCommand` オペレーションまたは `get-command` AWS CLI コマンドを使用して、コマンドリソースに関する情報を取得します。このコマンドは、`CreateCommand`

API リクエスト、または create-command CLI を使用してあらかじめ作成しておく必要があります。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、GetCommand アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *region* を us-east-1 などお客様ご自身の AWS リージョンに置き換えます。
- *account-id* を *123456789023* などの AWS アカウント 番号に置き換えます。
- *command-id* *LockDoor*または などの一意のコマンド識別子を使用します AWS IoT *Light_Power_Status*。複数のコマンドを取得する場合は、IAM ポリシーの「Resource」セクションでこれらのコマンドを指定できます。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:GetCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

コマンド取得の例 (AWS CLI)

次の例は、 を使用してコマンドに関する情報を取得する方法を示していますget-command AWS CLI。用途に応じて、*<command-id>* を情報を取得するコマンドの識別子に置き換えてください。この情報は、create-command CLI のレスポンスから取得できます。

```
aws iot get-command --command-id <command-id>
```

このコマンドを実行すると、コマンド、ペイロード、作成時刻と最終更新時刻に関する情報を含むレスポンスが生成されます。また、コマンドが非推奨になっているか、削除されているかを示す情報も提供されます。

以下のコードは、レスポンスの例です。

```
{
    "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
}
```

でコマンドを一覧表示する AWS アカウント

コマンドを作成すると、アカウントで作成したコマンドを表示することができます。リストでは、以下の情報を確認できます。

- コマンド ID、およびコマンドに指定した表示名。
- コマンドの Amazon リソースネーム (ARN)。
- ターゲットデバイス上でコマンドを実行できるかどうか、またはコマンドが非推奨なのかを示すコマンドの状態。

> **Note**
>
> アカウントから削除されているコマンドは、リストに表示されません。コマンドが保留中の削除の場合は、コマンド ID を使用することでこれらのコマンドの詳細を表示できます。

- コマンドが作成され、最後に更新された時刻。

アカウントのコマンドを一覧表示する (コンソール)

AWS IoT コンソールでは、Command Hub に移動して、作成したコマンドのリストとその詳細を確認できます。

アカウントのコマンドを一覧表示する (CLI)

作成したコマンドを一覧表示するには、ListCommands API オペレーションまたは list-commands CLI を使用します。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、ListCommands アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *region* を us-east-1 などお客様ご自身の AWS リージョンに置き換えます。
- *account-id* を *123456789012* などの AWS アカウント 番号に置き換えます。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:ListCommands",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/*"
    }
}
```

アカウントのコマンドを一覧表示する例

アカウントのコマンドを一覧表示するには、次のコマンドを使用できます。

```
aws iot list-commands --namespace "AWS-IoT"
```

このコマンドを実行すると、作成したコマンドのリスト、コマンドが作成された時刻、最後に更新された時刻を含むレスポンスが生成されます。また、コマンドが非推奨になっているのか、またはターゲットデバイス上で実行可能な状態なのかどうかを示す、コマンドの状態も示されます。それぞれのステータスとステータスの理由については、「コマンド実行ステータス」を参照してください。

コマンドリソースを更新する

コマンドを作成した後に、コマンドの表示名と説明を更新することができます。

> **Note**
>
> コマンドのペイロードは更新できません。この情報を更新したり、変更されたペイロードを使用したりするには、新しいコマンドを作成する必要があります。

コマンドリソースを更新する (コンソール)

コンソールからコマンドを更新するには、 AWS IoT コンソールの コマンドハブに移動し、次の手順を実行します。

1. 既存のコマンドリソースを更新するには、更新するコマンドを選択し、[アクション] で [編集] を選択します。
2. 使用する表示名と説明、および名前と値のペアをコマンドのタグとして指定します。
3. [編集] を選択して、コマンドを新しい設定で保存します。

コマンドリソースを更新する (CLI)

`UpdateCommand` コントロールプレーン API オペレーションまたは `update-command` AWS CLI を使用して、コマンドリソースを更新します。この API を使用すると、次のことができます。

- 作成したコマンドの表示名と説明を編集する。
- コマンドリソースを非推奨にする、または、既に非推奨になっているコマンドを復元する。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、`UpdateCommand` アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *`region`* を `us-east-1` などお客様ご自身の AWS リージョンに置き換えます。
- *`account-id`* を *`123456789012`* などの AWS アカウント 番号に置き換えます。
- *`command-id`* を *`LockDoor`* などの AWS IoT の一意のコマンド識別子に置き換えます。複数のコマンドを取得する場合は、IAM ポリシーの「Resource」セクションでこれらのコマンドを指定できます。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:UpdateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

コマンドに関する情報を更新する例 (AWS CLI)

次の例は、 コマンドを使用してupdate-command AWS CLI コマンドに関する情報を更新する方法を示しています。この API を使用してコマンドリソースを非推奨にする、または復元する方法については、「コマンドリソースを更新する (CLI)」を参照してください。

この例では、コマンドの表示名と説明を更新する方法を示しています。用途に応じて、*<command-id>* を情報を取得するコマンドの識別子に置き換えてください。

```
aws iot update-command \
    --command-id <command-id>
    --displayname <display-name> \
    --description <description>
```

このコマンドを実行すると、コマンドに関する更新された情報と最後に更新された時刻を含むレスポンスが生成されます。次のコードは、AC をオフにするコマンドの表示名と説明を更新するためのリクエストとレスポンスの例を示しています。

```
aws iot update-command \
    --command-id <LockDoor> \
    --displayname <Secondary lock door> \
    --description <Locks doors to my home>
```

このコマンドを実行すると、次のレスポンスが生成されます。

```
{
    "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"
}
```

コマンドリソースを非推奨にする、または復元する

コマンドを作成した後、コマンドの使用を継続しない場合は、非推奨としてマークできます。コマンドを非推奨にすると、保留中のすべてのコマンド実行は、終了ステータスに達するまでターゲットデバイスで実行され続けます。コマンドを非推奨にした後で、新しいコマンド実行をターゲットデバイスに送信するなどにコマンドを使用する場合は、コマンドを復元する必要があります。

> **Note**
>
> 非推奨にしたコマンドを編集したり、そのコマンドに新しい実行を実行したりすることはできません。デバイスで新しいコマンドを実行するには、コマンドを復元して、コマンドの状態を [利用可能] にする必要があります。

コマンドの非推奨化と復元、およびその考慮事項の詳細については、「コマンドリソースを非推奨にする」を参照してください。

コマンドリソースを削除する

コマンドが不要になった場合は、コマンドをアカウントから完全に削除することができます。削除アクションが成功した場合、以下が適用されます。

- コマンドが最大タイムアウトの 12 時間より長い期間非推奨になっている場合、コマンドは即時に削除されます。
- コマンドが非推奨になっていない場合、または非推奨状態にある期間が最大タイムアウト期間より短い場合、そのコマンドは [pending deletion] の状態になります。最大タイムアウトの 12 時間後に、アカウントから自動的に削除されます。

> **Note**
>
> コマンドは、保留中のコマンドの実行がある場合でも、削除される可能性があります。コマンドは保留中の削除状態になり、アカウントから自動的に削除されます。

コマンドリソースを削除する (コンソール)

コンソールからコマンドを削除するには、 AWS IoT コンソールの コマンドハブ に移動し、次の手順を実行します。

1. 削除するコマンドを選択し、[アクション] で [削除] を選択します。
2. コマンドを削除することを確認し、[削除] を選択します。

コマンドは削除対象としてマークされ、12 時間後にアカウントから完全に削除されます。

コマンドリソースを削除する (CLI)

HTTP コントロールプレーン API `DeleteCommand` オペレーションまたは `delete-command` AWS CLI コマンドを使用して、コマンドリソースを削除します。削除アクションが成功すると、HTTP `statusCode` が 204 または 202 になり、最大タイムアウト期間の 12 時間後にコマンドは自動的にアカウントから削除されます。ステータスが 204 の場合、これはコマンドが削除されたことを示しています。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、`DeleteCommand` アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *`region`* を `us-east-1` などお客様ご自身の AWS リージョンに置き換えます。
- *`account-id`* を *`123456789012`* などの AWS アカウント 番号に置き換えます。
- *`command-id`* を *`LockDoor`* などの AWS IoT の一意のコマンド識別子に置き換えます。複数のコマンドを取得する場合は、IAM ポリシーの「Resource」セクションでこれらのコマンドを指定できます。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:DeleteCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
```

```
    }
}
```

コマンドを削除する例 (AWS CLI)

次の例は、 コマンドを使用してdelete-command AWS CLI コマンドを削除する方法を示しています。用途に応じて、*<command-id>* を、削除するコマンドの識別子に置き換えてください。

```
aws iot delete-command --command-id <command-id>
```

API リクエストが成功すると、コマンドは 202 または 204 のステータスコードを生成します。GetCommand API を使用して、コマンドがアカウントに存在していないことを確認できます。

コマンド実行を開始してモニタリングする

コマンドを作成したら、ターゲットデバイスで実行を開始します。デバイスは結果を更新し、MQTT 予約済みトピックにステータスを発行します。アカウントから実行ステータスを取得してモニタリングします。

AWS IoT コンソールまたは を使用して、 コマンドを起動およびモニタリングします AWS CLI。

コマンドオペレーションの開始とモニタリング

- コマンド実行を開始する
- コマンド実行の結果を更新する
- コマンド実行を取得する
- MQTT テストクライアントを使用したコマンドの更新の表示
- でコマンド実行を一覧表示する AWS アカウント
- コマンド実行を削除する

コマンド実行を開始する

> **Important**
>
> お客様は、安全かつ適用法に準拠した方法でコマンドをデプロイすることに全責任を負います。

実行を開始する前に、以下を確認してください。

- ペイロード情報を使用して AWS IoT 名前空間に コマンドを作成しました。実行を開始すると、デバイスはペイロード命令を処理し、指定されたアクションを実行します。コマンドの作成コマンドリソースを作成するについては、「」を参照してください。
- デバイスが コマンド用に MQTT 予約トピックにサブスクライブしました。実行を開始すると、ペイロード情報はこの予約された MQTT リクエストトピックに発行されます。

  <devices> は Things または MQTT クライアントです。*<DeviceID>* はモノの名前またはクライアント ID です。サポートされている *<PayloadFormat>* 値: JSON および CBOR。詳細については、「コマンドトのピック」を参照してください。

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
  ```

  JSON/CBOR *<PayloadFormat>* 以外の場合は、次のコマンドトピック形式を使用します。

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request
  ```

ターゲットデバイスに関する考慮事項

コマンドを受信して実行するターゲットデバイスを指定します。登録済みデバイスにはモノの名前、未登録デバイスにはクライアント ID を使用します。ペイロードを受信すると、デバイスは コマンドを実行し、指定されたアクションを実行します。

AWS IoT モノ

ターゲットデバイスは、 AWS IoT レジストリに登録されているモノにすることができます。モノはデバイスの検索と管理を簡素化します。

Connect デバイスページまたは を使用して、デバイスをモノとして登録しますCreateThing。Thing Hub または を使用して既存のモノを検索しますDescribeThing。登録の詳細については、「レジストリを使用したモノの管理」を参照してください。

クライアント ID

未登録のデバイスの場合は、クライアント ID を使用します。

クライアント ID は、デバイスに割り当てる一意の識別子です。MQTT プロトコルで定義され、英数字、アンダースコア、またはダッシュが含まれます。に接続する各デバイスには、一意のクライアント ID AWS IoT が必要です。

> **Note**
>
> - 登録済みのモノの場合、クライアント ID はモノの名前と一致することができます。
> - 特定のクライアント ID をターゲットにする場合、デバイスはペイロードを受け取るためにそのクライアント ID AWS IoT を使用して に接続する必要があります。

クライアント ID は、デバイスが に接続するときに使用する MQTT クライアント ID です AWS IoT Core。 はこの ID AWS IoT を使用してデバイスを識別し、接続とサブスクリプションを管理します。

コマンド実行のタイムアウトに関する考慮事項

タイムアウトは、実行結果を提供するデバイスの期間 (秒単位) を指定します。

実行を作成すると、タイマーが開始されます。デバイスがオフラインになるか、タイムアウト内に結果を報告できない場合、実行はステータス でタイムアウトしますTIMED_OUT。

デフォルト: 10 秒。最大: 12 時間。

タイムアウト値と **TIMED_OUT** 実行ステータス

クラウドとデバイスの両方がタイムアウトをレポートできます。

コマンドを送信すると、タイマーが開始されます。タイムアウト内にデバイスレスポンスが到着しない場合、クラウドは理由コード TIMED_OUTを使用して実行ステータスを に設定します$NO_RESPONSE_FROM_DEVICE。

これは、次の場合に発生します。

- 実行中にデバイスがオフラインになりました。
- デバイスはタイムアウト内に実行を完了できませんでした。
- デバイスがタイムアウト内にステータスをレポートできませんでした。

この場合、TIMED_OUT の実行ステータスがクラウドから報告されると、コマンドの実行は非終了の状態です。デバイスは、ステータスをターミナルステータス SUCCEEDED、、FAILEDまたは のいずれかに上書きするレスポンスを発行できますREJECTED。その後、コマンド実行はターミナルになり、それ以降の更新は受け付けません。

デバイスは、コマンドの実行時にタイムアウトが発生したことを報告することで、クラウドによって開始されたTIMED_OUTステータスを更新することもできます。この場合、コマンドの実行ステータスは のままですがTIMED_OUT、statusReasonオブジェクトはデバイスによって報告された情報に基づいて更新されます。その後、コマンド実行はターミナルになり、それ以上の更新は受け入れられません。

MQTT 永続的セッションの使用

AWS IoT Device Management コマンド機能で使用するように MQTT 永続セッションを設定できます。この機能は、デバイスがオフラインになり、タイムアウト時間前にオンラインに戻ったときに、デバイスがコマンドを受信して指定された手順を実行できるようにする場合などに、特に便利です。

デフォルトでは、MQTT 永続セッションの有効期限は 60 分に設定されています。コマンド実行タイムアウトがこの期間を超える値に設定されている場合、60 分以上実行されるコマンド実行はメッセージブローカーによって拒否され、失敗する可能性があります。実行時間が 60 分を超えるコマンドを実行するには、永続セッションの有効期限の延長をリクエストできます。

> **Note**
>
> MQTT 永続セッション機能を正しく使用するには、クリーンスタートフラグをゼロに設定します。詳細については、「MQTT 永続セッション」を参照してください。

コマンド実行を開始する (コンソール)

コンソールからコマンドの実行を開始するには、 AWS IoT コンソールの コマンドハブページに移動し、次の手順を実行します。

1. 作成したコマンドを実行するには、[コマンドの実行] を選択します。
2. MQTT 予約トピックや該当する場合はパラメータなど、作成したコマンドに関する情報を確認します。

   動的コマンドの場合は、パラメータ値を入力するか、デフォルトのままにします。デフォルト値を持たないパラメータの場合は、この実行の一部として送信される値を指定する必要があります。
3. コマンドを受信して実行するターゲットデバイスを指定します。デバイスが に登録されている場合は AWS IoT モノとして指定でき AWS IoT、デバイスがまだ登録されていない場合はクライアント ID を使用できます。詳細については、ターゲットデバイスに関する考慮事項を参照してください。

4. (オプション) コマンドのタイムアウト値を設定し、タイムアウトまでにコマンドを実行する期間を決定します。コマンドを 60 分よりも長く実行させる場合は、MQTT 永続セッションの有効期間を延長する必要がある可能性があります。詳細については、「コマンド実行のタイムアウトに関する考慮事項」を参照してください。
5. [Run command] を選択してください。

コマンド実行を開始する (AWS CLI)

`StartCommandExecution` HTTP データプレーン API オペレーションを使用して、コマンド実行を開始します。API リクエストとレスポンスは、コマンド実行 ID と相関しています。デバイスは、コマンドの実行を完了すると、コマンドレスポンストピックにメッセージを発行することで、ステータスと実行結果をクラウドに報告できます。カスタムレスポンスコードの場合、所有しているアプリケーションコードはレスポンスメッセージを処理して結果を投稿できます AWS IoT。

デバイスがコマンドリクエストトピックをサブスクライブしている場合は、`StartCommandExecution` API がペイロードメッセージをトピックに発行します。ペイロードには、任意の形式を使用できます。詳細については、「コマンドのペイロード」を参照してください。

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
```

ペイロード形式が JSON または CBOR でない場合、コマンドリクエストトピックの形式を次に示します。

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、`StartCommandExecution` アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *region* を us-east-1 などお客様ご自身の AWS リージョンに置き換えます。
- *account-id* を *123456789012* などの AWS アカウント 番号に置き換えます。
- *command-id* など、 AWS IoT コマンドの一意の識別子を持つ *LockDoor*。複数のコマンドを送信する場合は、IAM ポリシーでこれらのコマンドを指定できます。

- *devices* デバイスが AWS IoT モノとして登録されているか、MQTT クライアントとして指定されているかclientに応じて、 thingまたは のいずれかを使用します。
- *device-id* thing-nameまたは AWS IoT を使用します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"
  ]
}
```

でサポートされている条件キーのリストを確認するにはStartCommandExecution、IAM ユーザーガイドの「 の条件キー AWS IoT」を参照してください。

アカウント固有のデータプレーンエンドポイントを取得する

API コマンドを実行する前に、エンドポイントのアカウント固有のエンドポイント URL を取得する必要があります。デュアルスタックエンドポイント (IPv4 と IPv6) を使用している場合は、iot:Data-ATS を使用します。iot:Jobs エンドポイントは IPv4 専用です。たとえば、次のコマンドを実行するとします。

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

以下のレスポンスの例に示すように、アカウント固有のエンドポイント URL を返します。

```
{
    "endpointAddress":
    "<account-specific-prefix>-ats.iot.<region>.api.com"
}
```

コマンド実行を開始する例 (AWS CLI)

次の例は、 コマンドを使用してstart-command-execution AWS CLI コマンドの実行を開始する方法を示しています。

この例では、次のように置き換えます。

- *<command-arn>* を実行するコマンドの ARN に置き換えます。この情報は、create-command CLI コマンドのレスポンスから取得できます。たとえば、ステアリングホイールモードを変更するためのコマンドを実行する場合は、arn:aws:iot:*region*:*account-id*:command/*SetComfortSteeringMode* を使用します。
- *<target-arn>* をターゲットデバイスのモノの ARN に置き換えます。これは、コマンド実行のターゲットとなる IoT のモノ、または MQTT クライアントです。たとえば、ターゲットデバイス myRegisteredThing に対してコマンドを実行する場合は、arn:aws:iot:*region*:*account-id*:thing/*myRegisteredThing* を使用します。
- *<endpoint-url>* を、アカウント固有のデータプレーンエンドポイントを取得する で取得したアカウント固有のエンドポイントに https:// のプレフィックスを付けて置き換えます。例えば、https://*123456789012abcd*.jobs.iot.*us-east-1*.amazonaws.com。
- (オプション) StartCommandExecution API オペレーションを実行するときに、追加のパラメータ executionTimeoutSeconds を指定することもできます。このオプションフィールドは、デバイスがコマンドの実行を完了する必要がある時間枠を秒単位で指定します。デフォルト値は 10 秒です。コマンドの実行ステータスが CREATED になると、タイマーが開始されます。タイマーの有効期限が切れる前にコマンド実行結果が受信されない場合は、ステータスは自動的に TIMED_OUT に変わります。
-

```
aws iot-jobs-data start-command-execution \
    --command-arn <command-arn>  \
    --target-arn <target-arn> \
    --endpoint <endpoint-url> \
    --execution-timeout-seconds 900
```

- (オプション) 動的コマンドの場合は、置換に使用するパラメータとその値を指定します。コマンドの作成時に defaultValue が設定されていないパラメータの値を指定する必要があります。パラメータに defaultValue がある場合、ここで指定するパラメータ値が優先されます。valueConditions が設定されているパラメータの場合、ここで指定するパラメータ値は 条件を満たす必要があります。

  Light_Power_Status 動的コマンドの例に基づく:
-

```
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}"
```

このコマンドを実行すると、コマンド実行 ID が返されます。この ID を使用して、コマンド実行ステータス、詳細、およびコマンド実行履歴をクエリすることができます。

> **Note**
>
> コマンドが非推奨になった場合、StartCommandExecution API リクエストは検証の例外で失敗します。このエラーを修正するには、まず UpdateCommand API を使用してコマンドを復元してから、StartCommandExecution リクエストを実行します。

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542"
}
```

コマンド実行の結果を更新する

コマンド実行のステータスまたは結果を更新するには、UpdateCommandExecution MQTT データプレーン API オペレーションを使用します。

> **Note**
>
> この API を使用する前に:
>
> - デバイスで MQTT 接続が確立されており、コマンドのリクエストとレスポンスのトピックがサブスクライブされている必要があります。詳細については、「高レベルのコマンドワークフロー」を参照してください。
> - StartCommandExecution API オペレーションを使用して、このコマンドがあらかじめ実行されている必要があります。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスがこれらのアクションを実行することを IAM ポリシーで許可していることを確認してください。以下に、デバイスがアクションを実行することを許可するポリシーの例を示しています。UpdateCommandExecution MQTT アクションを実行するア

クセス許可をユーザーに付与する追加のサンプル IAM ポリシーについては、「」を参照してください[接続および公開ポリシーの例](#)。

この例では、次のように置き換えます。

- *Region* など AWS リージョン、 で を使用しますus-east-1。
- *AccountID* を *123456789012* などの AWS アカウント 番号に置き換えます。
- *ThingName* は、 など、コマンド実行をターゲットとする AWS IoT モノの名前に置き換えます*myRegisteredThing*。
- *commands-request-topic* および AWS IoT コマンドリクエストおよびレスポンストピック*commands-response-topic*の名前。詳細については、「[高レベルのコマンドワークフロー](#)」を参照してください。

MQTT クライアント ID の IAM ポリシーの例

次のコードは、MQTT クライアント ID を使用する場合のデバイスポリシーの例を示しています。

```
{
  "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 のモノの IAM ポリシーの例

次のコードは、 AWS IoT のモノを使用する場合のデバイスポリシーの例を示しています。

```
{
  "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 の使用方法

コマンド実行がリクエストトピックで受信されると、デバイスはコマンドを処理します。デバイスはその後、UpdateCommandExecution API を使用して、コマンド実行のステータスと結果を次のレスポンストピックに更新します。

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

この例では、*<DeviceID>* はターゲットデバイスの一意の識別子であり、*<execution-id>* はターゲットデバイスでのコマンド実行の識別子です。*<PayloadFormat>* は JSON または CBOR です。

> **Note**
>
> デバイスを に登録していない場合は AWS IoT、モノの名前の代わりにクライアント ID を識別子として使用できます。

```
$aws/commands/clients/<ClientID>/executions/<ExecutionId>/response/<PayloadFormat>
```

実行ステータスに対する更新をデバイスで報告する

デバイスは、API を使用することで、コマンド実行に対する以下のステータス更新を報告できます。これらのステータスの詳細については、「コマンド実行ステータス」を参照してください。

- IN_PROGRESS: デバイスでコマンドの実行が開始されると、デバイスはステータスを IN_PROGRESS に更新できます。
- SUCCEEDED: コマンドが正常に処理されて実行が完了すると、デバイスは応答トピックに SUCCEEDED としてメッセージを発行できます。
- FAILED: コマンドの実行に失敗した場合は、デバイスはレスポンストピックに FAILED としてメッセージを発行できます。
- REJECTED: デバイスがコマンドを受け入れなかった場合、デバイスはレスポンストピックに REJECTED としてメッセージを発行できます。

- TIMED_OUT: コマンドの実行ステータスは、次のいずれかの理由で TIMED_OUT に変わる場合があります。
  - コマンド実行の結果が受信されなかった。これは、実行が指定された期間内に完了しなかったか、デバイスがレスポンストピックにステータス情報を発行できなかったために発生する可能性があります。
  - デバイスは、コマンドの実行を試行したときにタイムアウトが発生したことを報告します。

TIMED_OUT ステータスの詳細については、「タイムアウト値と TIMED_OUT 実行ステータス」を参照してください。

UpdateCommandExecution API を使用するときの考慮事項

UpdateCommandExecution API を使用する際の重要な考慮事項を以下に示しています。

- デバイスは、オプションの statusReason オブジェクトを使用して、実行に関する追加情報を提供できます。デバイスがこのオブジェクトを提供する場合、 オブジェクトの reasonCode フィールドは必須ですが、 reasonDescriptionフィールドはオプションです。
- デバイスが statusReason オブジェクトを使用する場合、 は パターンを使用し[A-Z0-9_-]+、64 文字を超えないようにreasonCodeする必要があります。を指定する場合はreasonDescription、1,024 文字を超えないようにしてください。改行などのコントロール文字を除く任意の文字を使用できます。
- デバイスは、オプションの result オブジェクトを使用することで、リモート関数呼び出しの戻り値など、コマンド実行の結果に関する情報を提供できます。result を指定する場合は、少なくとも 1 つのエントリが必要です。
- result フィールドでは、エントリをキーと値のペアとして指定します。エントリごとに、データ型情報を文字列、ブール値、またはバイナリとして指定する必要があります。文字列データ型はキー s を使用し、ブール値データ型はキー b を使用し、バイナリデータ型はキー bin を使用する必要があります。これらのキーが小文字であることを確認します。
- UpdateCommandExecution API の実行中にエラーが発生した場合は、Amazon CloudWatch の AWSIoTLogsV2 ロググループでエラーを表示できます。ログの有効化とログの表示については、「AWS IoT ログ記録の設定」を参照してください。

UpdateCommandExecution API の例

次のコードは、デバイスが UpdateCommandExecution API を使用して実行ステータスを報告する方法の例を示しています。statusReason フィールドには、ステータスに関する追加情報が提供さ

れ、result フィールドには、この例の場合は自動車のバッテリー残量のパーセントなど、実行の結果に関する情報が提供されます。

```
{
  "status": "IN_PROGRESS",
  "statusReason": {
    "reasonCode": "200",
    "reasonDescription": "Execution_in_progress"
  },
  "result": {
        "car_battery": {
            "s": "car battery at 50 percent"
        }
    }
}
```

コマンド実行を取得する

コマンドを実行したら、 AWS IoT コンソールと を使用してコマンド実行に関する情報を取得できます AWS CLI。取得できる情報は次のとおりです。

> **Note**
>
> 最新のコマンド実行ステータスを取得するには、以下で説明するように、デバイスが UpdateCommandExecution MQTT API を使用してレスポンストピックにステータス情報を発行する必要があります。デバイスがこのトピックに発行するまで、GetCommandExecution API はステータスを CREATED または TIMED_OUT として報告します。

作成する各コマンドの実行には以下が含まれます。

- 実行 ID。これはコマンド実行の一意の識別子です。
- コマンド実行のステータス。ターゲットデバイスでコマンドを実行すると、コマンド実行は CREATED ステータスになります。その後、以下で説明するように、他のコマンド実行ステータスに移行できます。
- コマンド実行の結果。
- 一意のコマンド ID と、実行が作成されたターゲットデバイス。

- 開始日。コマンド実行が作成された日時を示します。

コマンド実行を取得する (コンソール)

次のいずれかの方法を使用して、コンソールからコマンド実行を取得できます。

- [コマンドハブ] ページから

  AWS IoT コンソールの Command Hub ページに移動し、以下の手順を実行します。

  1. ターゲットデバイスで実行を作成したコマンドを選択します。
  2. コマンドの詳細ページの [コマンド履歴] タブに、作成した実行が表示されます。情報を取得する実行を選択します。
  3. デバイスが UpdateCommandExecution API を使用して結果情報を提供した場合は、このページの [結果] タブでこの情報を確認できます。
- [モノのハブ] ページから

  コマンドの実行時にターゲットデバイスとして AWS IoT モノを選択した場合は、モノのハブページから実行の詳細を表示できます。

  1. AWS IoT コンソールの Thing Hub ページに移動し、コマンド実行を作成したモノを選択します。
  2. モノの詳細ページの [コマンド履歴] に、作成した実行が表示されます。情報を取得する実行を選択します。
  3. デバイスが UpdateCommandExecution API を使用して結果情報を提供した場合は、このページの [結果] タブでこの情報を確認できます。

コマンド実行を取得する (CLI)

GetCommandExecution AWS IoT Core コントロールプレーンの HTTP API オペレーションを使用して、コマンド実行に関する情報を取得します。StartCommandExecution API オペレーションを使用して、このコマンドがあらかじめ実行されている必要があります。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、GetCommandExecution アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *region* を us-east-1 などお客様ご自身の AWS リージョンに置き換えます。
- *account-id* を *123456789012* などの AWS アカウント 番号に置き換えます。
- *command-id* などの一意の AWS IoT コマンド識別子を使用します*LockDoor*。
- *devices* デバイスが AWS IoT モノとして登録されているか、MQTT クライアントとして指定されているかclientに応じて、 thingまたは のいずれかを使用します。
- *device-id* thing-nameまたは AWS IoT を使用します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"
  ]
}
```

コマンド実行を取得する例

次の例は、 コマンドを使用して実行されたstart-command-execution AWS CLI コマンドに関する情報を取得する方法を示しています。次の例では、ステアリングホイールモードをオフにするために実行されたコマンドに関する情報を取得する方法が示されています。

この例では、次のように置き換えます。

- *<execution-id>* を情報を取得するコマンド実行の識別子に置き換えます。
- *<target-arn>* を実行のターゲットとするデバイスの Amazon リソース番号 (ARN) に置き換えます。この情報は、start-command-execution CLI コマンドのレスポンスから取得できます。
- オプションで、デバイスが UpdateCommandExection API を使用して実行結果を提供した場合、GetCommandExecution API を使用して GetCommandExecution API のレスポンスにコマンド実行結果を含めるかどうかを指定できます。

```
aws iot get-command-execution
    --execution-id <execution-id> \
```

```
    --target-arn <target-arn> \
    --include-result
```

このコマンドを実行すると、コマンド実行の ARN、実行ステータス、実行開始時刻、完了時刻に関する情報を含むレスポンスが生成されます。また、ステータスに関する追加情報を含む statusReason オブジェクトも提供されます。それぞれのステータスとステータスの理由については、「コマンド実行ステータス」を参照してください。

次のコードは、API リクエストからのレスポンスの例を示しています。

> **Note**
>
> 実行レスポンスの completedAt フィールドは、デバイスが終了ステータスをクラウドに報告する時間に対応します。TIMED_OUT ステータスの場合、このフィールドはデバイスがタイムアウトを報告した場合にのみ設定されます。TIMED_OUT ステータスがクラウドによって設定された場合は、TIMED_OUT ステータスは更新されません。このタイムアウトの動作の詳細については、「コマンド実行のタイムアウトに関する考慮事項」を参照してください。

```
{
    "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 テストクライアントを使用したコマンドの更新の表示

MQTT テストクライアントを使用すると、コマンド機能を使用の際に、MQTT 経由でのメッセージの送受信を表示できます。デバイスが との MQTT 接続を確立したら AWS IoT、コマンドを作成してペイロードを指定し、デバイスで実行できます。コマンドを実行すると、デバイスがコマンドの MQTT 予約済みリクエストトピックをサブスクライブしている場合、このトピックに発行されたペイロードメッセージが表示されます。

その後、デバイスはペイロードの指示を受け取り、 AWS IoT デバイスで指定されたオペレーションを実行します。次に、 `UpdateCommandExecution` API を使用してコマンド実行結果とステータス情報を command. AWS IoT Device Management listens の MQTT 予約レスポンストピックに発行し、レスポンストピックを更新して更新された情報を保存し、 AWS CloudTrail と Amazon CloudWatch にログを発行します。その後、コンソールから、または `GetCommandExecution` API を使用して、最新のコマンド実行情報を取得できます。

次の手順は、MQTT テストクライアントを使用してメッセージを観察する方法を示しています。

1. AWS IoT コンソールで MQTT テストクライアントを開きます。
2. Subscribe タブで、次のトピックを入力し、Subscribe を選択します。*<thingId>* は、登録したデバイスのモノの名前です AWS IoT。

   > **Note**
   >
   > デバイスのモノの名前は、 AWS IoT コンソールの Thing Hub ページから確認できます。デバイスをモノとして登録していない場合は、Connect device ページ AWS IoT から に接続するときにデバイスを登録できます。 https://console.aws.amazon.com/iot/home#/connect-overview

   ```
   $aws/commands/things/<thingId>/executions/+/request
   ```

3. (オプション) [サブスクライブ] タブで、次のトピックを入力し、[サブスクライブ] を選択することもできます。

   ```
   $aws/commands/things/+/executions/+/response/accepted/json
   $aws/commands/things/+/executions/+/response/rejected/json
   ```

4. コマンド実行が開始されると、デバイスがサブスクライブしているリクエストトピック `$aws/commands/things/<thingId>/executions/+/request` を使用して、メッセージペイロー

ドがデバイスに送信されます。MQTT テストクライアントには、デバイスがコマンドを処理する手順を含むコマンドペイロードが表示されます。

5. コマンドの実行を開始した後、デバイスは、次の MQTT 予約済みレスポンストピックにコマンドのステータス更新を発行することができます。

```
$aws/commands/<devices>/<device-id>/executions/<executionId>/response/json
```

例えば、車の空調をオンにして温度を目的の値まで下げるために実行したコマンドを考えてみましょう。次の JSON は、車両がレスポンストピックに発行した、コマンドの実行に失敗したことを示すサンプルメッセージです。

```
{
  "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"
  }
}
```

この場合、車のバッテリーを充電した後に、コマンドを再度実行できます。

でコマンド実行を一覧表示する AWS アカウント

コマンドを実行したら、 AWS IoT コンソールと を使用してコマンド実行に関する情報を取得できます AWS CLI。取得できる情報は次のとおりです。

- 実行 ID。これはコマンド実行の一意の識別子です。
- コマンド実行のステータス。ターゲットデバイスでコマンドを実行すると、コマンド実行は CREATED ステータスになります。その後、以下で説明するように、他のコマンド実行ステータスに移行できます。
- 一意のコマンド ID と、実行が作成されたターゲットデバイス。
- 開始日。コマンド実行が作成された日時を示します。

アカウントのコマンド実行を一覧表示する (コンソール)

次のいずれかの方法を使用して、コンソールからすべてのコマンド実行を表示できます。

- [コマンドハブ] ページから

  AWS IoT コンソールの Command Hub ページに移動し、以下の手順を実行します。

  1. ターゲットデバイスで実行を作成したコマンドを選択します。
  2. コマンドの詳細ページで、[コマンド履歴] タブに移動すると、作成した実行が一覧表示されます。
- [モノのハブ] ページから

  コマンドの実行時にターゲットデバイスとして AWS IoT モノを選択し、1 つのデバイスに複数のコマンド実行を作成した場合は、モノのハブページからデバイスの実行を表示できます。

  1. AWS IoT コンソールの Thing Hub ページに移動し、実行を作成したモノを選択します。
  2. モノの詳細ページの [コマンド履歴] に、デバイスに対して作成した実行のリストが表示されます。

アカウントのコマンド実行を一覧表示する (CLI)

`ListCommandExecutions` AWS IoT Core コントロールプレーンの HTTP API オペレーションを使用して、アカウント内のすべてのコマンド実行を一覧表示します。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスでこのアクションを実行することを IAM ポリシーが許可していることを確認してください。以下に、`ListCommandExecutions` アクションの実行をユーザーに許可する IAM ポリシーの例を示しています。

この例では、次のように置き換えます。

- *region* を us-east-1 などお客様ご自身の AWS リージョンに置き換えます。
- *account-id* を *123456789012* などの AWS アカウント 番号に置き換えます。
- *command-id* などの一意の AWS IoT コマンド識別子を使用します*LockDoor*。

```
{
```

```
  "Effect": "Allow",
  "Action": "iot:ListCommandExecutions",
  "Resource": *
}
```

コマンド実行を一覧表示する例

次の例は、 AWS アカウントのコマンド実行を一覧表示する方法を示しています。

コマンドを実行するときは、リストをフィルタリングして、targetArn を使用して特定のデバイスに対して作成されたコマンド実行のみを表示するか、commandArn を使用して指定された特定のコマンドの実行のみを表示するかを指定する必要があります。

この例では、次のように置き換えます。

- *<target-arn>* を、arn:aws:iot:*us-east-1*:*123456789012*:thing/*b8e4157c98f332cffb37627f* などの、実行のターゲットとするデバイスの Amazon リソース番号 (ARN) に置き換えます。
- *<target-arn>* を、arn:aws:iot:*us-east-1*:*123456789012*:thing/*b8e4157c98f332cffb37627f* などの、実行のターゲットとするデバイスの Amazon リソース番号 (ARN) に置き換えます。
- *<after>* は、特定の日時以降に作成された実行を一覧表示する場合の日時に置き換えます。たとえば、2024-11-01T03:00。

```
aws iot list-command-executions \
--target-arn <target-arn> \
--started-time-filter '{after=<after>}' \
--sort-order "ASCENDING"
```

このコマンドを実行すると、作成したコマンド実行のリスト、実行の開始時刻、完了時刻を含むレスポンスが生成されます。また、ステータス情報と、ステータスに関する追加情報を含む statusReason オブジェクトも提供されます。

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


それぞれのステータスとステータスの理由については、「コマンド実行ステータス」を参照してください。

コマンド実行を削除する

コマンド実行が不要になった場合は、アカウントから完全に削除することができます。

> **Note**
>
> - コマンド実行は、SUCCEEDED、FAILED、REJECTED などの終了ステータスになった場合にのみ削除できます。
> - このオペレーションは、 AWS IoT Core API または を使用してのみ実行できます AWS CLI。これはコンソールからは利用できません。

サンプルの IAM ポリシー

この API オペレーションを使用する前に、デバイスがこれらのアクションを実行することを IAM ポリシーで許可していることを確認してください。以下に、デバイスがアクションを実行することを許可するポリシーの例を示しています。

この例では、次のように置き換えます。

- *Region* など AWS リージョン、 で を使用しますus-east-1。
- *AccountID* を *123456789012* などの AWS アカウント 番号に置き換えます。
- *CommandID* を実行を削除するコマンドの識別子に置き換えます。
- *devices* デバイスが AWS IoT モノとして登録されているか、MQTT クライアントとして指定されているかclientに応じて、 thingまたは のいずれかを使用します。
- *device-id* thing-nameまたは AWS IoT を使用します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"
  ]
}
```

コマンド実行を削除する例

次の例は、 コマンドを使用してdelete-command AWS CLI コマンドを削除する方法を示しています。用途に応じて、*<execution-id>* を削除するコマンド実行の識別子に置き換え、*<target-arn>* をターゲットデバイスの ARN に置き換えてください。

```
aws iot delete-command-execution \
--execution-id <execution-id> \
--target-arn <target-arn>
```

API リクエストが成功すると、コマンド実行は 200 のステータスコードを生成します。GetCommandExecution API を使用して、コマンド実行がアカウントに存在していないことを確認できます。

コマンドリソースを非推奨にする

コマンドを非推奨にして、古いため使用しないでください。例えば、同じ ID でペイロードが異なる新しいコマンドを作成するときに、コマンドがアクティブに維持されなくなった場合や、非推奨になります。

主な考慮事項

コマンドを廃止する際の重要な考慮事項:

- コマンドを非推奨にしても削除されません。コマンドは、その ID を使用して取得し、再利用のために復元できます。
- 廃止されたコマンドで新しい実行を開始しようとすると、エラーが生成され、古いコマンドが使用できなくなります。
- 廃止されたコマンドを実行するには、まず復元します。復元後、コマンドはターゲットデバイスで定期的に使用および実行できるようになります。
- 実行中にコマンドを非推奨にすると、コマンドは完了するまで実行され続けます。実行ステータスは引き続き取得できます。

コマンドリソースを非推奨にする (コンソール)

コンソールからコマンドを非推奨にするには、 AWS IoT コンソールの コマンドハブに移動し、次の手順を実行します。

1. 非推奨にするコマンドを選択し、アクションで非推奨を選択します。
2. コマンドを非推奨にすることを確認し、[非推奨] を選択します。

コマンドリソースを非推奨にする(CLI)

update-command CLI を使用して、コマンドを非推奨としてマークします。削除する前に、コマンドを非推奨にする必要があります。廃止されたコマンドを使用するには、最初に復元します。

```
aws iot update-command \
    --command-id <command-id> \
    --deprecated
```

たとえば、上記の例で更新した *ACSwitch* コマンドを非推奨にした場合、次のコードはこのコマンドを実行した際のサンプル出力を示しています。

```
{
    "commandId": "turnOffAc",
    "deprecated": true,
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
```

```
}
```

非推奨の時間とステータスを確認する

GetCommand API を使用して、コマンドが廃止されたかどうか、および最後に廃止されたのはいつかを判断します。

```
aws iot get-command --command-id <turnOffAC>
```

このコマンドは、最後に更新されたフィールドの作成と廃止のタイムスタンプなど、コマンド情報を含むレスポンスを生成します。これにより、コマンドの有効期間と、それを削除するか再利用するかが決まります。コマンドのレスポンス例を次に示します*turnOffAc*。

```
{
    "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
}
```

コマンドリソースを復元する

ACSwitch コマンドを使用またはデバイスに送信するには、まず復元します。

コンソールからコマンドを復元するには、 AWS IoT コンソールの コマンドハブに移動し、復元するコマンドを選択し、アクションで復元を選択します。

AWS IoT Core API または を使用してコマンドを復元するには AWS CLI、 UpdateCommand API オペレーションまたは CLI update-command を使用します。以下の例は、リクエストとレスポンスの例を示しています。

```
aws iot update-command \
    --command-id <command-id>
    --no-deprecated
```

次の例は、出力の例を示しています。

```
{
    "commandId": "ACSwitch",
    "deprecated": false,
    "lastUpdatedAt": "2024-05-09T23:17:21.954000-07:00"
}
```

AWS IoT セキュアトンネリング

リモートサイトの制限付きファイアウォールの背後でデバイスをデプロイする場合、トラブルシューティング、設定の更新、およびその他のオペレーションタスクのために、それらのデバイスにアクセスする方法が必要です。セキュアトンネリングを使用して、 が管理する安全な接続を介してリモートデバイスへの双方向通信を確立します AWS IoT。セキュアトンネリングでは、既存のインバウンドファイアウォールルールを更新する必要がないため、リモートサイトのファイアウォールルールで提供されるのと同じセキュリティレベルを維持できます。

たとえば、数百マイル離れた工場にあるセンサー装置で工場の温度測定に問題があるとします。セキュアトンネリングを使用すると、そのセンサーデバイスへのセッションを開いて、すばやく開始できます。問題（不正な設定ファイルなど）を特定したら、同じセッションでファイルをリセットし、センサーデバイスを再起動できます。従来のトラブルシューティング（センサーデバイスを調査するために技術者を工場に派遣するなど）と比較して、セキュアトンネリングでは、インシデント対応と復旧の時間および運用コストが削減されます。

セキュアトンネリングとは?

セキュアトンネリングを使用して、リモートサイトのポート制限ファイアウォールの背後にデプロイされているデバイスにアクセスします。 AWS クラウドを使用して、ノートパソコンまたはデスクトップコンピュータから送信先デバイスにソースデバイスとして接続できます。送信元と送信先は、各デバイスで実行されるオープンソースのローカルプロキシを使用して通信します。ローカルプロキシは、ファイアウォールで許可されているオープンポート、通常は 443 AWS クラウド を使用してと通信します。トンネルを介して送信されるデータは、Tranported Layer Security（TLS）を使用して暗号化されています。

トピック

- セキュアトンネリングの概念
- セキュアトンネリングの仕組み
- 安全なトンネルのライフサイクル

セキュアトンネリングの概念

リモートデバイスとの通信を確立する際に、セキュアトンネリングで使用される条件は次のとおりです。セキュアトンネリングの仕組みについては、「セキュアトンネリングの仕組み」を参照してください。

クライアントアクセストークン (CAT)

新しいトンネルが作成されたときにセキュアトンネリングによって生成されるトークンのペア。CAT は、送信元および宛先デバイスが、セキュアトンネリングサービスに接続するために使用されます。CAT は、トンネルに接続するためだけに使用できます。トンネルに再接続するには、RotateTunnelAccessToken API オペレーションまたは rotate-tunnel-access-token CLI コマンドを使用してクライアントアクセストークンをローテーションします。

クライアントトークン

AWS IoT セキュアトンネリングが、同じトンネルへの後続のすべての再試行接続に使用できるクライアントによって生成された一意の値。このフィールドはオプションです。クライアントトークンが指定されていない場合、クライアントアクセストークン (CAT) は同じトンネルに対して一度だけ使用できます。同じ CAT を使用したその後の接続試行は拒否されます。クライアントトークンの使用方法の詳細については、「GitHub でのローカルプロキシリファレンスの実装」を参照してください。

宛先アプリケーション

宛先デバイスで実行されているアプリケーション。たとえば、宛先アプリケーションは、セキュアトンネリングを使用して SSH セッションを確立するための SSH デーモンにすることができます。

宛先デバイス

アクセスするリモートデバイス。

デバイスエージェント

AWS IoT デバイスゲートウェイに接続し、MQTT 経由で新しいトンネル通知をリッスンする IoT アプリケーション。詳細については、「IoT エージェントスニペット」を参照してください。

ローカルプロキシ

送信元および宛先デバイスで実行され、セキュアトンネリングとデバイスアプリケーション間でデータストリームをリレーするソフトウェアプロキシ。ローカルプロキシは、送信元モードまたは宛先モードで実行できます。詳細については、「ローカルプロキシ」を参照してください。

送信元デバイス

オペレータが宛先デバイス (通常はラップトップまたはデスクトップコンピュータ) へのセッションを開始するために使用するデバイス。

トンネル

ソースデバイスと宛先デバイス間の双方向通信 AWS IoT を可能にする 経由の論理パス。

セキュアトンネリングの仕組み

次に、セキュアトンネリングが送信元と宛先デバイス間の接続を確立する方法を示します。クライアントアクセストークン (CAT) などのさまざまな用語については、「セキュアトンネリングの概念」を参照してください。

1. トンネルを開く

   リモート送信先デバイスとのセッションを開始するためのトンネルを開くには、 AWS マネジメントコンソール、AWS CLI open-tunnel コマンド、または OpenTunnel API を使用できます。

2. クライアントアクセストークンペアをダウンロードする

   トンネルを開いた後、送信元と宛先のクライアントアクセストークン (CAT) をダウンロードし、送信元デバイスに保存できます。ローカルプロキシを開始する前に、CAT を取得して保存する必要があります。

3. 宛先モードでのローカルプロキシの起動

   インストール済みで宛先デバイスで実行されている IoT エージェントは、予約済み MQTT トピック `$aws/things/thing-name/tunnels/notify` をサブスクライブし、CAT を受け取ります。ここで thing-*name* は、送信先用に作成する AWS IoT モノの名前です。詳細については、「セキュアトンネリングのトピック」を参照してください。

   IoT エージェントは、CAT を使用して宛先モードでローカルプロキシを起動し、トンネルの宛先側の接続を設定します。詳細については、「IoT エージェントスニペット」を参照してください。

4. 送信元モードでのローカルプロキシの実行

   トンネルを開くと、 はソースデバイスにダウンロードできるソースの CAT AWS IoT Device Management を提供します。CAT を使用して、ローカルプロキシを送信元モードで起動し、トンネルの送信元側を接続します。ローカルプロキシの詳細については、「ローカルプロキシ」を参照してください。

5. SSH セッションを開く

   トンネルの両側が接続されると、送信元側のローカルプロキシを使用して SSH セッションを開始できます。

を使用してトンネル AWS マネジメントコンソール を開き、SSH セッションを開始する方法の詳細については、「」を参照してくださいトンネルを開き、リモートデバイスへの SSH セッションを開始します。

次の動画では、セキュアトンネリングの仕組みと、Raspberry Pi デバイスへの SSH セッションを設定するプロセスを説明します。

安全なトンネルのライフサイクル

トンネルはステータス OPEN または CLOSED を持つことができます。トンネルへの接続は、CONNECTED または DISCONNECTED ステータスを持つことができます。次に、さまざまなトンネルと接続ステータスの仕組みを示します。

1. トンネルを開くと、ステータスは OPEN になります。トンネルの送信元および宛先の接続ステータスは DISCONNECTED に設定されます。
2. デバイス (送信元または宛先) がトンネルに接続すると、対応する接続ステータスが CONNECTED に変更されます。
3. トンネルのステータスが OPEN のままデバイスがトンネルとの接続を切断すると、対応する接続ステータスが DISCONNECTED に戻ります。デバイスは、トンネルが OPEN のままである限り、トンネルに対して繰り返し接続および切断できます。

> **Note**
>
> クライアントアクセストークン (CAT) は、トンネルに接続するためだけに使用できます。トンネルに再接続するには、RotateTunnelAccessToken API オペレーションまたは rotate-tunnel-access-token CLI コマンドを使用してクライアントアクセストークンをローテーションします。例については「クライアントアクセストークンをローテーションして AWS IoT 安全なトンネリング接続の問題を解決する」を参照してください。

4. CloseTunnel を呼び出すか、トンネルが MaxLifetimeTimeout 値よりも長く OPEN のままであるときにトンネルのステータスは、CLOSED になります。MaxLifetimeTimeout を呼び出すときに OpenTunnel を設定できます。値を指定しない場合、MaxLifetimeTimeout はデフォルトで 12 時間に設定されます。

> **Note**
>
> トンネルが CLOSED である場合、再び開くことはできません。

5. トンネルが表示されている間は、DescribeTunnel と ListTunnels を呼び出して、トンネルメタデータを表示できます。トンネルは、削除されるまで少なくとも 3 時間 AWS IoT コンソールで表示できます。

AWS IoT セキュアトンネリングのチュートリアル

AWS IoT セキュアトンネリングは、 が管理する安全な接続を介して、ファイアウォールの内側にあるリモートデバイスへの双方向通信を確立するのに役立ちます AWS IoT。

AWS IoT セキュアトンネリングをデモするには、AWS IoT GitHub のセキュアトンネリングデモを使用します。

以下のチュートリアルは、セキュアトンネリングの使用開始方法と使用方法の学習に役立ちます。次の方法について説明します。

1. リモートデバイスにアクセスするためには、クイックセットアップ方式と手動セットアップ方式を使用して、セキュアトンネルを作成します。
2. 手動セットアップ方式を使用する場合は、ローカルプロキシを設定し、トンネルに接続して送信先デバイスにアクセスします。
3. ローカルプロキシを設定しなくても、SSH でブラウザからリモートデバイスに接続できます。
4. AWS CLI または手動セットアップ方式を使用して作成されたトンネルを変換し、クイックセットアップ方式を使用します。

このセクションのチュートリアル

このセクションのチュートリアルでは、 AWS マネジメントコンソール と AWS IoT API リファレンスを使用してトンネルを作成することに重点を置いています。 AWS IoT コンソールでは、トンネルハブページまたは作成したモノの詳細ページからトンネルを作成できます。詳細については、「AWS IoT コンソールでのトンネル作成方法」を参照してください。

このセクションのチュートリアルを以下に示します。

- トンネルを開き、ブラウザベースの SSH を使用してリモートデバイスにアクセスします。

  このチュートリアルでは、クイックセットアップ方式を使用して トンネルハブ ページからトンネルを開く方法を説明します。また、ブラウザベースの SSH を使用して、 AWS IoT コンソール内のコンテキスト内コマンドラインインターフェイスを使用してリモートデバイスにアクセスする方法についても説明します。

- 手動セットアップを使用してトンネルを開き、リモートデバイスに接続する

  このチュートリアルでは、手動セットアップ方式を使用して トンネルハブ ページからトンネルを開く方法を説明します。また、送信元デバイスのターミナルからローカルプロキシを設定して起動し、トンネルに接続する方法について学習します。

- リモートデバイス用のトンネルを開き、ブラウザベースの SSH を使用する

  このチュートリアルでは、作成したモノの詳細ページからトンネルを開く方法を説明します。新しいトンネルを作成し、既存のトンネルを使用する方法を学習します。既存のトンネルは、デバイス用に作成された最新のオープントンネルに対応しています。ブラウザベースの SSH を使用してリモートデバイスにアクセスすることもできます。

AWS IoT セキュアトンネリングのチュートリアル

- トンネルを開き、リモートデバイスへの SSH セッションを開始します
- リモートデバイス用のトンネルを開き、ブラウザベースの SSH を使用する

トンネルを開き、リモートデバイスへの SSH セッションを開始します

これらのチュートリアルでは、ファイアウォールの内側にあるデバイスにリモートアクセスする方法を学習します。ファイアウォールですべてのインバウンドトラフィックをブロックしているため、デバイスへの SSH セッションを直接開始することはできません。このチュートリアルでは、トンネルを開き、そのトンネルを使用してリモートデバイスへの SSH セッションを開始する方法について説明します。

チュートリアルの前提条件

チュートリアルを実行するための前提条件は、トンネルを開いてリモートデバイスにアクセスする際に、手動セットアップ方式またはクイックセットアップ方式のどちらを使用するかによって異なります。

> **Note**
>
> どちらのセットアップ方式でも、ポート 443 でアウトバウンドトラフィックを許可する必要があります。

- クイックセットアップ方式のチュートリアルに関する前提条件については、「クイックセットアップ方式の前提条件」を参照してください。
- 手動セットアップ方式のチュートリアルに関する前提条件については、「手動セットアップ方式の前提条件」を参照してください。このセットアップ方式を使用する場合は、送信元デバイスでローカルプロキシを設定する必要があります。ローカルプロキシのソースコードをダウンロードするには、「Local proxy reference implementation on GitHub」(GitHub のローカルプロキシリファレンスの実装) を参照してください。

トンネルのセットアップ方式

これらのチュートリアルでは、トンネルを開いてリモートデバイスに接続するための手動セットアップ方式およびクイックセットアップ方式について学習します。次の表では、セットアップ方式の違いを示しています。トンネルを作成したら、ブラウザ内のコマンドラインインターフェイスを使用してリモートデバイスに SSH 接続できます。トークンを紛失した場合やトンネルの接続が切断された場合、新しいアクセストークンを送信してトンネルに再接続できます。

クイックセットアップ方式および手動セットアップ方式

条件	Quick Setup	手動セットアップ
トンネルの作成	デフォルトの編集可能な構成で新しいトンネルを作成します。リモートデバイスにアクセスするには、送信先サービスとして SSH のみを使用できます。	トンネル構成を手動で指定してトンネルを作成します。この方式を使用すると、SSH 以外のサービスを使用してリモートデバイスに接続できます。
アクセストークン	トンネルの作成時にモノの名前が指定されている場合、送信先アクセストークンは、予約済み MQTT トピック上のデバイスに自動的に配信されます。送信元デバイスでのトークンのダウンロードや管理は必要ありません。	送信元デバイスでトークンを手動でダウンロードし、管理する必要があります。トンネルの作成時にモノの名前が指定されている場合、送信先アクセストークンは、予約済み MQTT トピック上のリモートデバイスに自動的に配信されます。
ローカルプロキシ	ウェブベースのローカルプロキシが自動的に設定され、デバイスとやり取りできるようになります。ローカ	ローカルプロキシを手動で設定し、起動する必要があります。ローカルプロキシを設定するには、 AWS IoT Device Client を使

条件	Quick Setup	手動セットアップ
	ルプロキシを手動で設定する必要はありません。	用するか、GitHub でローカルプロキシリファレンスの実装をダウンロードします。

AWS IoT コンソールでのトンネル作成方法

このセクションのチュートリアルでは、 AWS マネジメントコンソール および OpenTunnel API を使用したトンネルの作成方法について説明します。トンネルの作成時に送信先を設定すると、 AWS IoT セキュアトンネリングは MQTT および予約済み MQTT トピック を介して送信先クライアントアクセストークンをリモートデバイスに配信します`$aws/things/RemoteDeviceA/tunnels/notify`)。MQTT メッセージを受信すると、リモートデバイス上の IoT エージェントがローカルプロキシを送信先モードで起動します。詳細については、「予約済みトピック」を参照してください。

> **Note**
>
> 宛先クライアントアクセストークンを別の方法でリモートデバイスに配信する場合は、宛先設定を省略できます。詳細については、「リモートデバイスの設定と IoT エージェントの使用」を参照してください。

AWS IoT コンソールでは、次のいずれかの方法を使用してトンネルを作成できます。これらの方式を使用してトンネルを作成する方法を学習できるチュートリアルの詳細については、「このセクションのチュートリアル」を参照してください。

- トンネルハブ

  トンネルを作成する際に、クイックセットアップ方式または手動セットアップ方式のどちらを使用するかを指定し、オプションのトンネル設定の詳細を提供できるようになります。設定の詳細には、送信先デバイスの名前と、デバイスへの接続に使用するサービスも含まれます。トンネルを作成したら、ブラウザ内で SSH を使用するか、 AWS IoT コンソールの外部でターミナルを開いてリモートデバイスにアクセスできます。
- モノの詳細ページ

  トンネル作成時に、セットアップ方式の選択とオプションのトンネル設定の詳細を指定することに加え、最新のオープントンネルを使用するか、デバイス用に新しいトンネルを作成するかを指定することもできます。既存のトンネルの設定の詳細は編集できません。クイックセットアップ方式を使用して、アクセストークンと SSH をブラウザ内のリモートデバイスにローテー

ションできます。この方法を使用してトンネルを開くには、 AWS IoT レジストリに IoT モノ (などRemoteDeviceA) を作成しておく必要があります。詳細については、AWS IoT「レジストリにデバイスを登録する」を参照してください。

このセクションのチュートリアル

- トンネルを開き、ブラウザベースの SSH を使用してリモートデバイスにアクセスします。
- 手動セットアップを使用してトンネルを開き、リモートデバイスに接続する

トンネルを開き、ブラウザベースの SSH を使用してリモートデバイスにアクセスします。

トンネルを作成するには、クイックセットアップ方式または手動セットアップ方式を使用できます。このチュートリアルでは、クイックセットアップ方式を使用してトンネルを開き、ブラウザベースの SSH を使用してリモートデバイスに接続する方法を説明しています。手動セットアップ方式を使用してトンネルを開く方法を示す例については、「手動セットアップを使用してトンネルを開き、リモートデバイスに接続する」を参照してください。

クイックセットアップ方式を使用すると、編集可能なデフォルト構成で新しいトンネルを作成できます。ウェブベースのローカルプロキシが設定され、アクセストークンは MQTT を使用してリモート送信先デバイスに自動的に配信されます。トンネルを作成したら、コンソール内のコマンドラインインターフェイスを使用してリモートデバイスとのやり取りを開始できます。

クイックセットアップ方式では、リモートデバイスにアクセスする送信先サービスとして SSH を使用する必要があります。さまざまなセットアップ方式の詳細については、「トンネルのセットアップ方式」を参照してください。

クイックセットアップ方式の前提条件

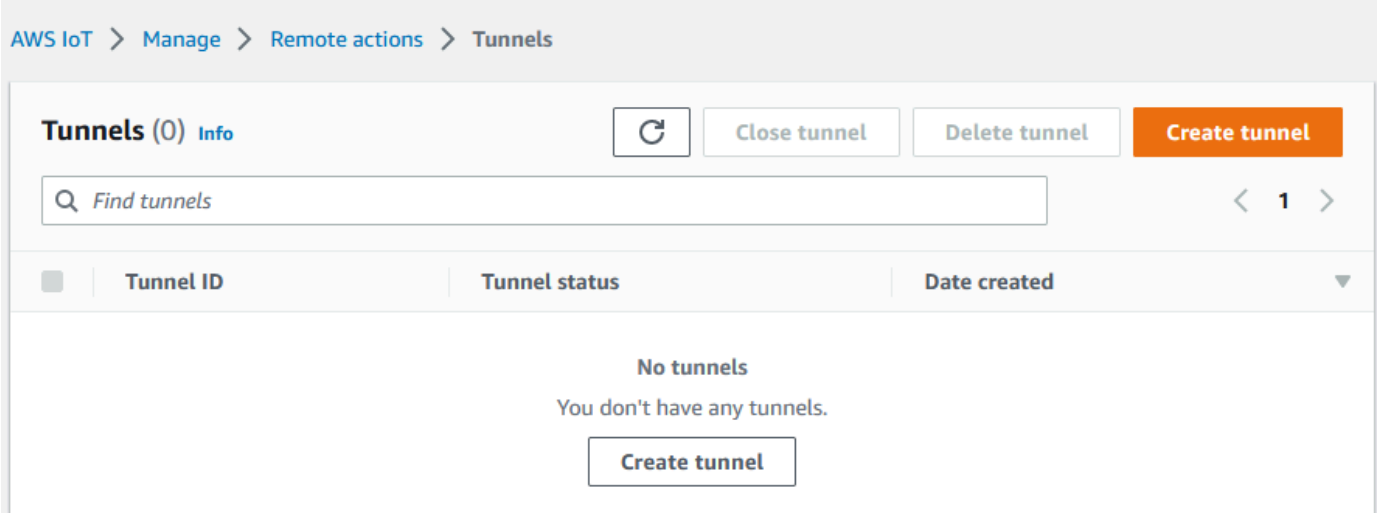
- リモートデバイスが背後にあるファイアウォールは、ポート 443 でアウトバウンドトラフィックを許可する必要があります。作成されたトンネルは、このポートを使用してリモートデバイスに接続します。
- デバイスゲートウェイに接続し、MQTT トピックサブスクリプションで設定されているリモートデバイスで実行されている IoT AWS IoT デバイスエージェント (「」を参照IoT エージェントスニペット) がある。詳細については、「デバイスを AWS IoT デバイスゲートウェイに接続する」を参照してください。
- リモートデバイスで SSH デーモンが実行されている必要があります。

トンネルを開く

セキュアトンネルは AWS マネジメントコンソール、、 AWS IoT API リファレンス、または を使用して開くことができます AWS CLI。送信先名はオプションで設定できますが、このチュートリアルでは不要です。送信先を設定すると、MQTT を使用して、セキュアトンネリングによってアクセストークンをリモートデバイスに自動的に配信します。詳細については、「AWS IoT コンソールでのトンネル作成方法」を参照してください。

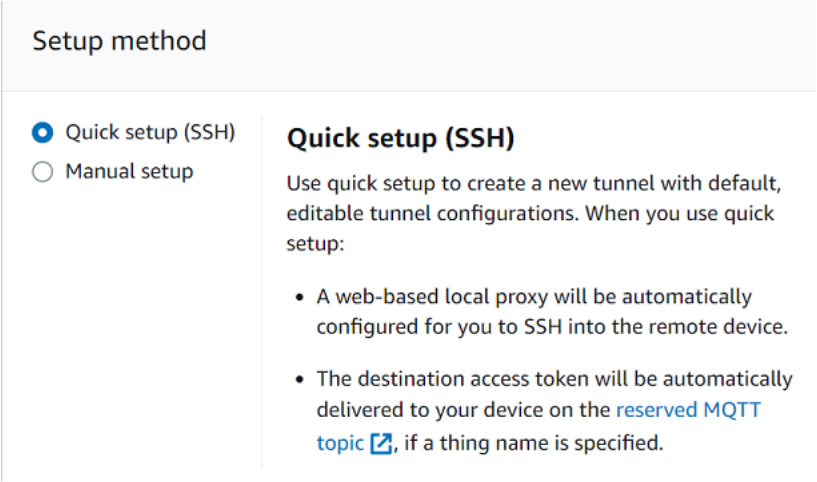
コンソールを使用してトンネルを開くには

1. AWS IoT コンソールのトンネルハブ に移動し、[Create tunnel] (トンネルの作成) を選択します。



2. このチュートリアルでは、トンネルの作成方式として [Quick setup] (クイックセットアップ) を選択し、[Next] (次へ) を選択します。

> **Note**
>
> 作成したモノの詳細ページからセキュアトンネルを作成する場合、新しいトンネルを作成するか、既存のトンネルを使用するかを選択できます。詳細については、「リモートデバイス用のトンネルを開き、ブラウザベースの SSH を使用する」を参照してください。

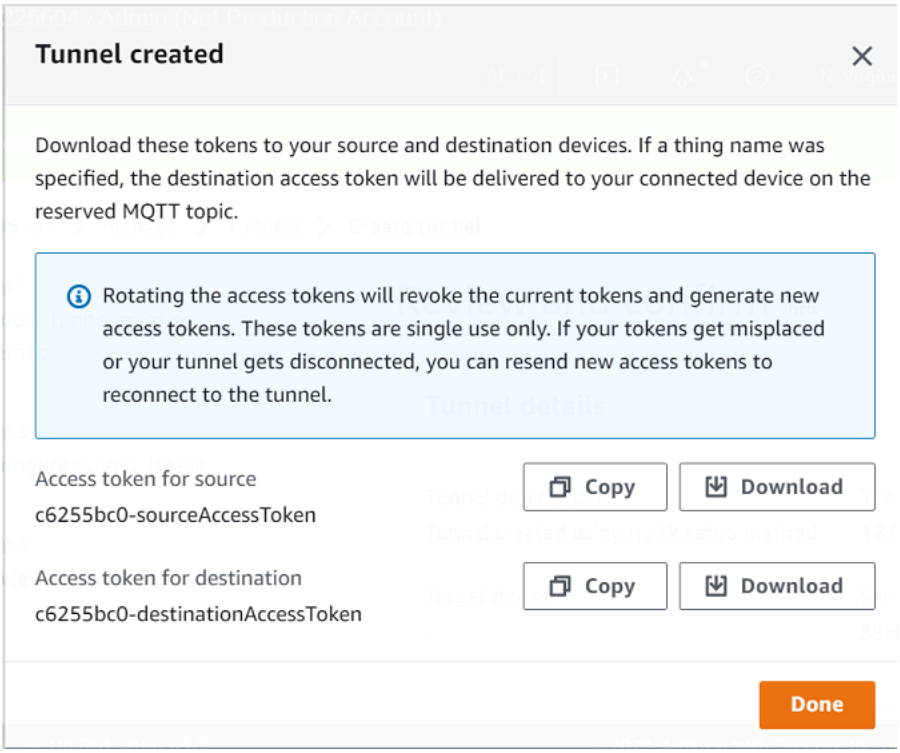


3. トンネル設定の詳細を確認して確定します。トンネルを作成するには、[Confirm and create] (確認と作成) を選択します。これらの詳細を編集する場合は、[Previous] (前へ) を選択して前のページに戻り、確認してトンネルを作成します。

   > **Note**
   >
   > クイックセットアップを使用する場合は、サービス名は編集できません。[Service] (サービス) として、[SSH] を使用する必要があります。

4. トンネルを作成するには、[Done] (完了) を選択します。

   このチュートリアルでは、送信元または送信先のアクセストークンをダウンロードする必要はありません。これらのトークンは、トンネルに接続するためだけに使用できます。トンネルが切断された場合は、トンネルに再接続するために新しいトークンを生成してリモートデバイスに送信できます。詳細については、「トンネルのアクセストークンを再送信する」を参照してください。



API を使用してトンネルを開くには

新しいトンネルを開くには、OpenTunnel API オペレーションを使用できます。

> **Note**
>
> クイックセットアップ方式を使用してトンネルを作成できるのは、 AWS IoT コンソールからのみです。 AWS IoT API リファレンス API または を使用すると AWS CLI、手動セットアップ方式が使用されます。作成した既存のトンネルを開いてから、クイックセットアップ方式を使用するようにトンネルの設定方式を変更できます。詳細については、「既存のトンネルを開き、ブラウザベースの SSH を使用する」を参照してください。

API オペレーションを実行する方法の例を次に示します。オプションで、モノの名前と送信先サービスを指定する場合は、DestinationConfig パラメータを使用します。このパラメータの使用方法を示す例については、「リモートデバイス用の新しいトンネルを開く」を参照してください。

```
aws iotsecuretunneling open-tunnel
```

このコマンドを実行すると、新しいトンネルの作成と、送信元と送信先のアクセストークンが提供されます。

```
{
    "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>"
}
```

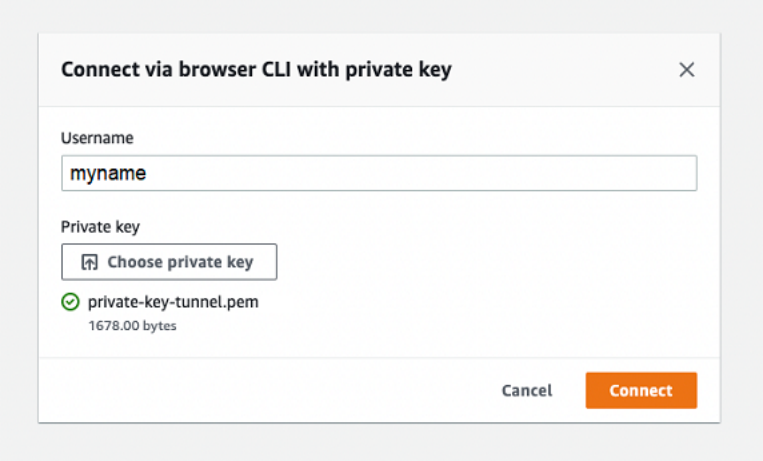
ブラウザベースの SSH を使用する

クイックセットアップ方式でトンネルを作成し、送信先デバイスがトンネルに接続されると、ブラウザベースの SSH を使用してリモートデバイスにアクセスできます。ブラウザベースの SSH を使用すると、コンソール内のコンテキスト内のコマンドラインインターフェイスにコマンドを入力することで、リモートデバイスと直接通信できます。この機能により、コンソール外でターミナルを開いたり、ローカルプロキシを設定したりする必要がないため、リモートデバイスとのやり取りがしやすくなります。

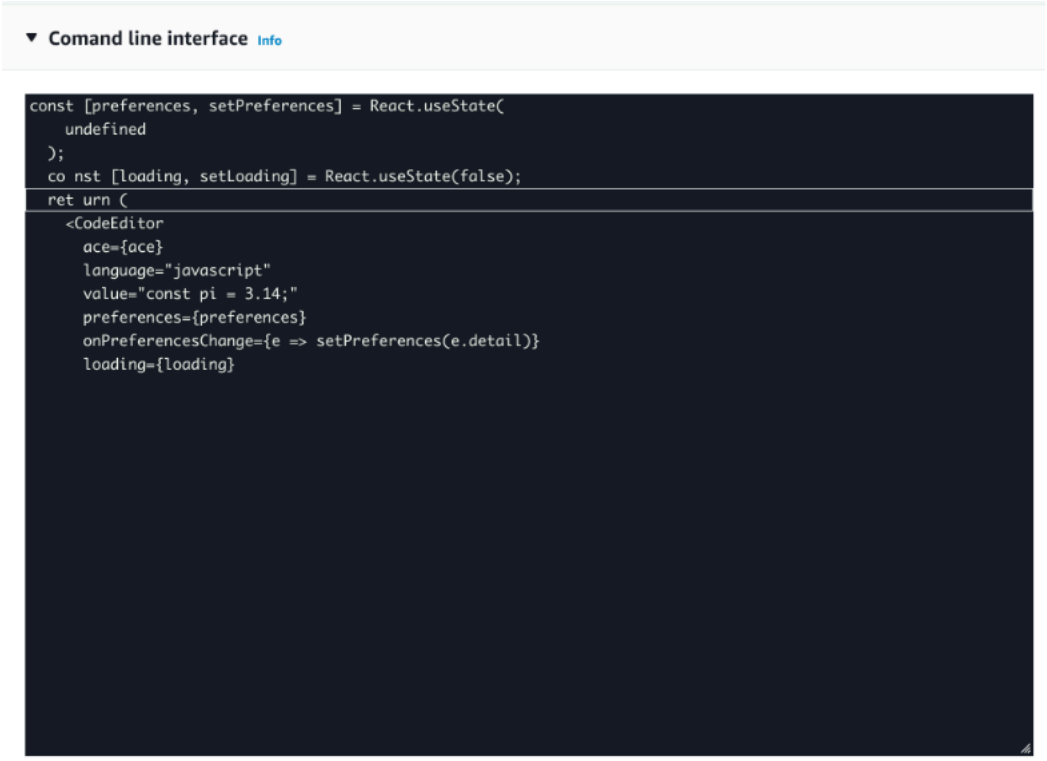
ブラウザベースの SSH を使用するには

1. AWS IoT コンソールのトンネルハブ に移動して、作成したトンネルを選択し、その詳細を表示します。
2. [Secure Shell (SSH)] (セキュアシェル (SSH)) セクションを展開し、[Connect] (接続) を選択します。
3. ユーザー名とパスワードを入力して SSH 接続を認証するか、デバイスのプライベートキーを使用してより安全な認証を行うかを選択します。プライベートキーを使用して認証する場合は、PEM 形式 (256 および 512) の RSA キーのみが SSH AWS IoT Secure Tunneling コンソールで動作することに注意してください。

   - ユーザー名とパスワードを使用して接続するには、[Use password] (パスワードを使用する) を選択します。その後、ユーザー名とパスワードを入力して、ブラウザ内 CLI の使用を開始できます。
   - 送信先デバイスのプライベートキーを使用して接続するには、[Use private key] (プライベートキーを使用する) を選択します。ユーザー名を指定してデバイスのプライベートキーファイルをアップロードし、[Connect] (接続) を選択してブラウザ内 CLI の使用を開始します。



ローカルプロキシが既に設定されているため、SSH 接続への認証が完了すると、コマンドの入力やブラウザ CLI によるデバイスとのやり取りをすぐに開始できます。



トンネルの継続時間が経過してもブラウザ CLI が開いたままになっていると、タイムアウトになり、コマンドラインインターフェイスが切断される可能性があります。トンネルを複製して別のセッションを開始し、コンソール自体でリモートデバイスとやり取りすることができます。

ブラウザベースの SSH を使用する場合のトラブルシューティング

ブラウザベースの SSH を使用する際に発生する可能性のある問題のトラブルシューティング方法を次に示します。

- コマンドラインインターフェイスの代わりにエラーが表示される

  送信先のデバイスが切断されたため、エラーが表示されている可能性があります。[Generate new access tokens] (新しいアクセストークンの生成) を選択して新しいアクセストークンを生成し、MQTT を使用してそのトークンをリモートデバイスに送信できます。この新しいトークンを使用して、トンネルに再接続できます。トンネルに再接続すると、履歴がクリアされ、コマンドラインセッションが更新されます。
- プライベートキーを使用して認証すると、トンネルが切断されたというエラーが表示される

  プライベートキーが送信先デバイスで許可されなかったことが原因で、エラーが表示されている可能性があります。このエラーをトラブルシューティングするには、認証用にアップロードしたプライベートキーファイルを確認してください。それでもエラーが表示される場合は、デバイスのログを確認してください。リモートデバイスに新しいアクセストークンを送信して、トンネルに再接続してみることもできます。
- セッション使用中にトンネルが閉じた

  トンネルが指定した継続時間を超えて開いたままになっていたために閉じられた場合、コマンドラインセッションが切断される可能性があります。トンネルが閉じた場合、再び開くことはできません。再接続するには、デバイスに対して別のトンネルを開く必要があります。

  トンネルを複製することで、閉じたトンネルと同じ構成の新しいトンネルを作成できます。 AWS IoT コンソールから閉じたトンネルを複製できます。トンネルを複製するには、閉じているトンネルを選択して詳細を表示し、[Duplicate tunnel] (トンネルの複製) を選択します。使用するトンネルの継続時間を指定し、新しいトンネルを作成します。

クリーンアップ

- トンネルを閉じる

  トンネルの使用が終わったら、トンネルを閉じることをお勧めします。トンネルを指定した継続期間を超えて開いたままにしていると、閉じた状態になることもあります。トンネルが閉じた場合、再び開くことはできません。閉じたトンネルを選択して、[Duplicate tunnel] (トンネルの複製) を選択すれば、トンネルを複製できます。使用するトンネルの継続時間を指定し、新しいトンネルを作成します。

- AWS IoT コンソールから個々のトンネルまたは複数のトンネルを閉じるには、トンネルハブに移動し、閉じるトンネルを選択して、[Close tunnel] (トンネルを閉じる) を選択します。
- AWS IoT API リファレンス API を使用して個々のトンネルまたは複数のトンネルを閉じるには、CloseTunnel API を使用します。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- トンネルの削除

  からトンネルを完全に削除できます AWS アカウント。

> **Warning**
>
> 削除の操作は永続的で、元には戻せません。

- AWS IoT コンソールから個々のトンネルまたは複数のトンネルを削除するには、トンネルハブに移動し、削除するトンネルを選択して、[Delete tunnel] (トンネルを削除する) を選択します。
- AWS IoT API リファレンス API を使用して個々のトンネルまたは複数のトンネルを削除するには、CloseTunnel API を使用します。API を使用する場合は、`delete` フラグを `true` に設定します。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

手動セットアップを使用してトンネルを開き、リモートデバイスに接続する

トンネルを開く場合、クイックセットアップ方式または手動セットアップ方式を選択して、リモートデバイスへのトンネルを開くことができます。このチュートリアルでは、手動セットアップ方式を使用してトンネルを開き、ローカルプロキシを設定して起動し、リモートデバイスに接続する方法を説明しています。

手動セットアップ方式を使用する場合は、トンネルを作成する際にトンネル構成を手動で指定する必要があります。トンネルを作成したら、ブラウザ内で SSH を実行するか、 AWS IoT コンソールの外部でターミナルを開くことができます。このチュートリアルでは、コンソール外のターミナル

を使用してリモートデバイスにアクセスする方法について説明します。また、ローカルプロキシを設定し、そのローカルプロキシに接続してリモートデバイスとやり取りする方法について学習します。ローカルプロキシに接続するには、トンネルを作成する際に送信元アクセストークンをダウンロードする必要があります。

このセットアップ方式では、FTP などの SSH 以外のサービスを使用してリモートデバイスに接続できます。さまざまなセットアップ方式の詳細については、「トンネルのセットアップ方式」を参照してください。

手動セットアップ方式の前提条件

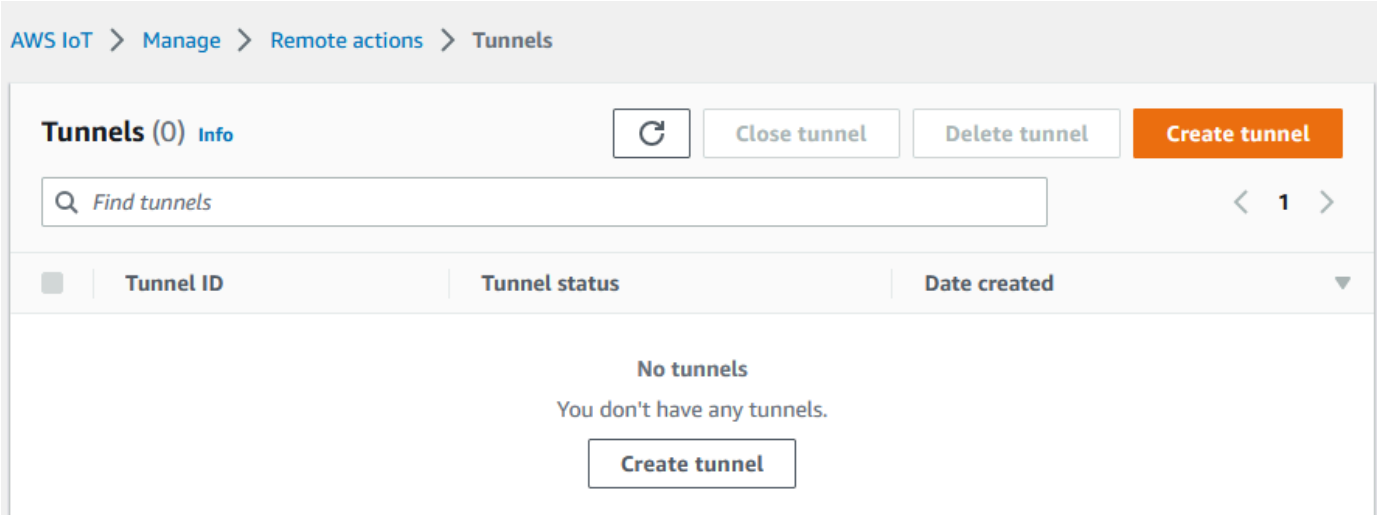
- リモートデバイスが背後にあるファイアウォールは、ポート 443 でアウトバウンドトラフィックを許可する必要があります。作成されたトンネルは、このポートを使用してリモートデバイスに接続します。
- デバイスゲートウェイに接続し、MQTT トピックサブスクリプションで設定されているリモートデバイスで実行されている IoT AWS IoT デバイスエージェント (「」を参照IoT エージェントスニペット) がある。詳細については、「デバイスを AWS IoT デバイスゲートウェイに接続する」を参照してください。
- リモートデバイスで SSH デーモンが実行されている必要があります。
- GitHub からローカルプロキシソースコードをダウンロードし、選択したプラットフォーム用にビルドしました。このチュートリアルでは、ビルドされたローカルプロキシ実行可能ファイルを `localproxy` と呼びます。

トンネルを開く

セキュアトンネルは AWS マネジメントコンソール、、 AWS IoT API リファレンス、または を使用して開くことができます AWS CLI。送信先名はオプションで設定できますが、このチュートリアルでは不要です。送信先を設定すると、MQTT を使用して、セキュアトンネリングによってアクセストークンをリモートデバイスに自動的に配信します。詳細については、「AWS IoT コンソールでのトンネル作成方法」を参照してください。

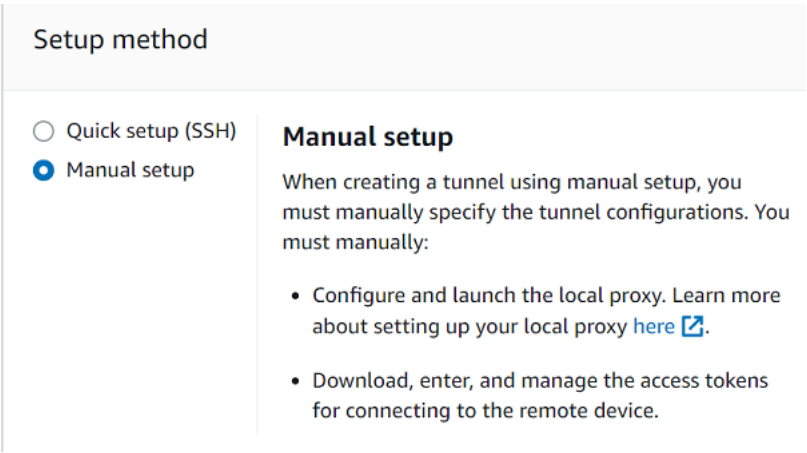
コンソールでトンネルを開くには

1. AWS IoT コンソールのトンネルハブ に移動し、[Create tunnel] (トンネルの作成) を選択します。



2. このチュートリアルでは、トンネルの作成方法として [Manual setup] (手動セットアップ) を選択し、[Next] (次へ) を選択します。クイックセットアップ方式を使用してトンネルを作成する方法についての詳細は、「トンネルを開き、ブラウザベースの SSH を使用してリモートデバイスにアクセスします。」を参照してください。

> **Note**
>
> モノの詳細ページからセキュアトンネルを作成する場合、新しいトンネルを作成するか、既存のトンネルを使用するかを選択できます。詳細については、「リモートデバイス用のトンネルを開き、ブラウザベースの SSH を使用する」を参照してください。



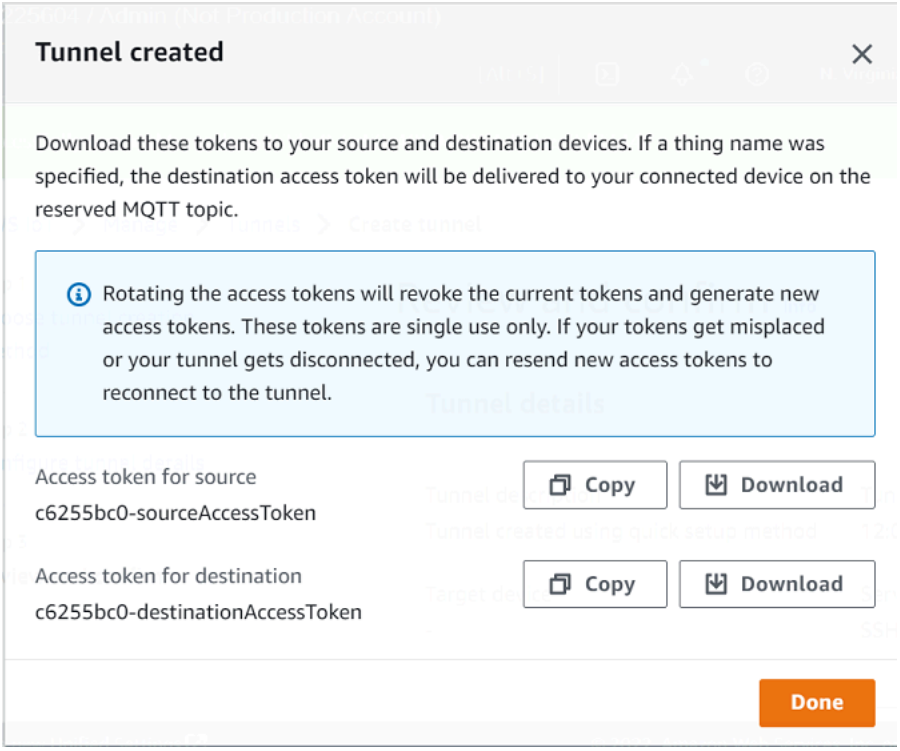
3. (オプション) トンネル構成の設定を入力します。このステップをスキップして、次のステップに進み、トンネルの作成を行うこともできます。

トンネルの説明、トンネルのタイムアウト時間、さらに送信元のタグをキーと値のペアで入力することで、送信元を識別しやすくします。このチュートリアルでは、送信先の設定をスキップできます。

> **Note**
>
> トンネルが開いている時間に基づいて料金が請求されることはありません。料金が発生するのは、新しいトンネルを作成するときだけです。料金情報については、AWS IoT Device Management 料金表の「セキュアトンネリング」を参照してください。

4. クライアントアクセストークンをダウンロードし、[Done] (完了) を選択します。[Done] (完了) を選択した後、トークンはダウンロードできません。

   これらのトークンは、トンネルに接続するためだけに使用できます。トークンを紛失した場合やトンネルが切断された場合は、トンネルに再接続するために新しいトークンを生成してリモートデバイスに送信できます。



API を使用してトンネルを開くには

新しいトンネルを開くには、OpenTunnel API オペレーションを使用できます。API を使用して、トンネルの継続時間や送信先設定などの追加の設定を指定することもできます。

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
```

このコマンドを実行すると、新しいトンネルの作成と、送信元と送信先のアクセストークンが提供されます。

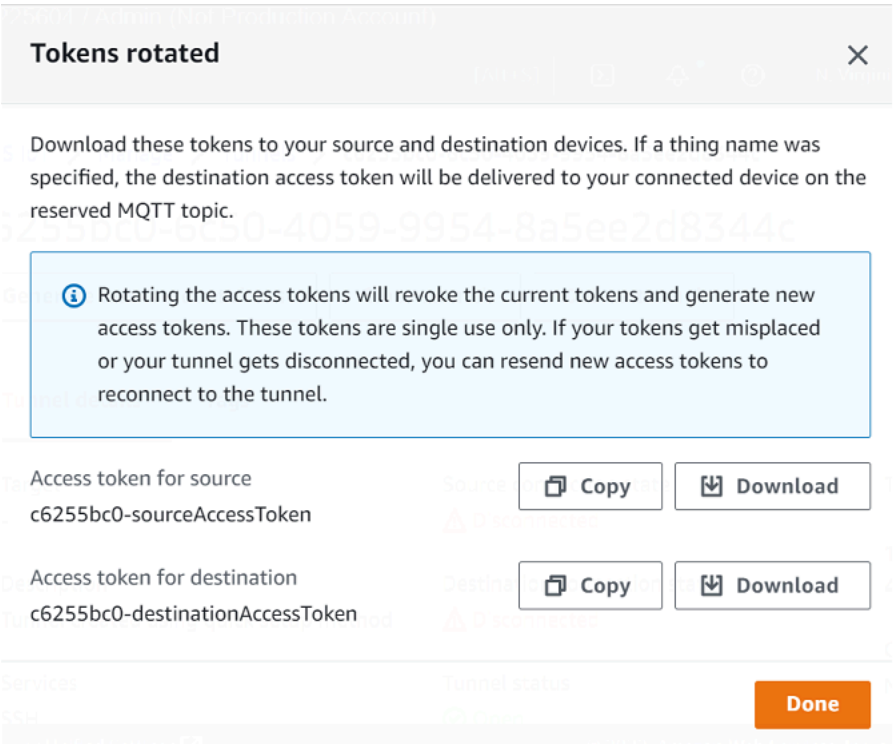
```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

トンネルのアクセストークンを再送信する

トンネル作成時に取得したトークンは、トンネルの接続に一度だけ使用できます。アクセストークンを紛失した場合、またはトンネルが切断された場合、追加料金なしで MQTT を使用して新しいアクセストークンをリモートデバイスに再送信できます。 AWS IoT セキュアトンネリングは現在のトークンを取り消し、トンネルに再接続するための新しいアクセストークンを返します。

コンソールからトークンをローテーションするには

1. AWS IoT コンソールのトンネルハブに移動し、作成したトンネルを選択します。
2. トンネルの詳細ページで、[Generate new access tokens] (新しいアクセストークンを生成する) を選択し、[Next] (次へ) を選択します。
3. トンネル用の新しいアクセストークンをダウンロードし、[Done] (完了) を選択します。これらのトークンは一度だけ使用できます。トークンを紛失した場合やトンネルの接続が切断された場合、新しいアクセストークンを再送信できます。



API を使用してアクセストークンをローテーションするには

トンネルアクセストークンをローテーションするには、RotateTunnelAccessToken API オペレーションを使用して現在のトークンを取り消し、トンネルに再接続するための新しいアクセストークンを返します。例えば、次のコマンドは送信先デバイス *RemoteThing1* のアクセストークンをローテーションします。

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

このコマンドを実行すると、次の例に示すように新しいアクセストークンが生成されます。その後、デバイスエージェントが正しく設定されていれば、トークンは MQTT を使用してデバイスに配信され、トンネルに接続します。

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

アクセストークンをいつどのようにローテーションするかを示す例については、「クライアントアクセストークンをローテーションして AWS IoT 安全なトンネリング接続の問題を解決する」を参照してください。

ローカルプロキシを設定、起動します

リモートデバイスに接続するには、ラップトップでターミナルを開き、ローカルプロキシを設定して起動します。ローカルプロキシは、WebSocketの安全な接続を介したセキュアトンネリングを使用して、送信元デバイスで実行中のアプリケーションによって送信されたデータを送信します。GitHub からローカルプロキシソースをダウンロードすることができます。

ローカルプロキシの設定後、送信元クライアントのアクセストークンをコピーし、それを使用してローカルプロキシを送信元モードで起動します。ローカルプロキシを起動するコマンドの例を次に示します。次のコマンドでは、ローカルプロキシは、ポート 5555 で新しい接続をリッスンするように設定されています。このコマンドで:

- `-r` は を指定します。これは AWS リージョントンネルが作成されたリージョンと同じである必要があります。
- `-s` によってプロキシが接続するポートを指定します。
- `-t` によってクライアントトークンのテキストを指定します。

```
./localproxy -r us-east-1 -s 5555 -t source-client-access-token
```

このコマンドを実行すると、送信元モードでローカルプロキシが起動します。コマンド実行後に次のエラーが表示された場合は、CA パスを設定します。詳細については、GitHub の「Secure tunneling local proxy」(セキュアトンネリングローカルプロキシ) を参照してください。

```
Could not perform SSL handshake with proxy server: certificate verify failed
```

以下は、ローカルプロキシを source モードで実行したサンプル出力です。

```
...
...

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 セッションの開始

別のターミナルを開き、次のコマンドを使用して、ポート 5555 のローカルプロキシに接続し、新しい SSH セッションを開始します。

```
ssh username@localhost -p 5555
```

SSH セッションのパスワードの入力が求められる場合があります。SSH セッションが終了したら、**exit** と入力してセッションを閉じます。

クリーンアップ

- トンネルを閉じる

  トンネルの使用が終わったら、トンネルを閉じることをお勧めします。トンネルを指定した継続期間を超えて開いたままにしていると、閉じた状態になることもあります。トンネルが閉じた場合、

再び開くことはできません。閉じたトンネルを開いて [Duplicate tunnel] (トンネルの複製) を選択すれば、トンネルを複製できます。使用するトンネルの継続時間を指定し、新しいトンネルを作成します。

- AWS IoT コンソールから個々のトンネルまたは複数のトンネルを閉じるには、トンネルハブに移動し、閉じるトンネルを選択して、[Close tunnel] (トンネルを閉じる) を選択します。
- AWS IoT API リファレンス API を使用して個々のトンネルまたは複数のトンネルを閉じるには、CloseTunnel API オペレーションを使用します。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- トンネルの削除

  からトンネルを完全に削除できます AWS アカウント。

> **Warning**
>
> 削除の操作は永続的で、元には戻せません。

  - AWS IoT コンソールから個々のトンネルまたは複数のトンネルを削除するには、トンネルハブに移動し、削除するトンネルを選択して、[Delete tunnel] (トンネルを削除する) を選択します。
  - AWS IoT API リファレンス API を使用して個々のトンネルまたは複数のトンネルを削除するには、CloseTunnel API オペレーションを使用します。API を使用する場合は、`delete` フラグを `true` に設定します。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

リモートデバイス用のトンネルを開き、ブラウザベースの SSH を使用する

AWS IoT コンソールから、トンネルハブまたは作成した IoT モノの詳細ページからトンネルを作成できます。 IoT トンネルハブからトンネルを作成する場合、クイックセットアップと手動セットアップのどちらを使用してトンネルを作成するかを指定できます。チュートリアルの例については、「トンネルを開き、リモートデバイスへの SSH セッションを開始します」を参照してください。

AWS IoT コンソールのモノの詳細ページからトンネルを作成する場合、このチュートリアルに示すように、新しいトンネルを作成するか、そのモノの既存のトンネルを開くかを指定することもできます。既存のトンネルを選択すると、このデバイス用に作成した最新のオープントンネルにアクセスできます。その後、ターミナル内のコマンドラインインターフェイスを使用して、デバイスに SSH で接続できます。

前提条件

- リモートデバイスが背後にあるファイアウォールは、ポート 443 でアウトバウンドトラフィックを許可する必要があります。作成されたトンネルは、このポートを使用してリモートデバイスに接続します。
- AWS IoT レジストリに IoT モノ (などRemoteDevice1) を作成しました。これは、クラウドでのリモートデバイスの表示に対応しています。詳細については、「AWS IoT レジストリにデバイスを登録する」を参照してください。
- デバイスゲートウェイに接続し、MQTT トピックサブスクリプションで設定されているリモートデバイスで実行されている IoT AWS IoT デバイスエージェント (「」を参照IoT エージェントスニペット) がある。詳細については、「デバイスを AWS IoT デバイスゲートウェイに接続する」を参照してください。
- リモートデバイスで SSH デーモンが実行されている必要があります。

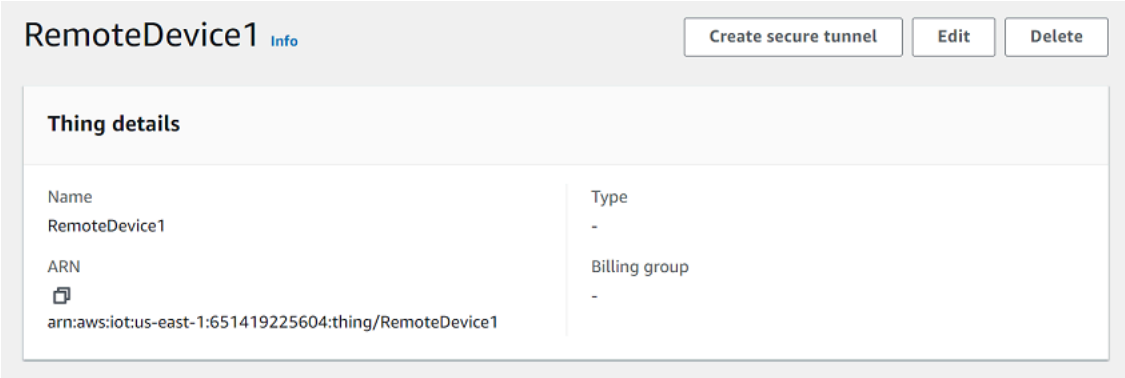
リモートデバイス用の新しいトンネルを開く

例えば、リモートデバイス RemoteDevice1 へのトンネルを開くとします。まず、 AWS IoT レジストリに RemoteDevice1 という名前で IoT のモノを作成します。その後、、 AWS IoT API リファレンス API AWS マネジメントコンソール、または を使用してトンネルを作成できます AWS CLI。

トンネル作成時に送信先を設定すると、セキュアトンネリングサービスにより MQTT および予約済み MQTT トピック ($aws/things/RemoteDeviceA/tunnels/notify) を介して、送信先クライアントアクセストークンがリモートデバイスに配信されます。詳細については、「AWS IoT コンソールでのトンネル作成方法」を参照してください。

コンソールからリモートデバイス用のトンネルを作成するには

1. RemoteDevice1 というモノを選択して詳細を表示し、[Create secure tunnel] (セキュアトンネルの作成) を選択します。



2. 新しいトンネルを作成するか、既存のトンネルを開くかを選択します。新しいトンネルを作成するには、[Create new tunnel] (新しいトンネルの作成) を選択します。次に、トンネルを作成するために、クイックセットアップ方式または手動セットアップ方式のどちらを使用するか選択できます。詳細については、「手動セットアップを使用してトンネルを開き、リモートデバイスに接続する」および「トンネルを開き、ブラウザベースの SSH を使用してリモートデバイスにアクセスします。」を参照してください。

API を使用してリモートデバイス用のトンネルを作成するには

新しいトンネルを開くには、OpenTunnel API オペレーションを使用できます。次のコードは、このコマンドの実行例を示しています。

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
    --cli-input-json file://input.json
```

次では、input.json ファイルの内容を示しています。destinationConfig パラメータを使用して、送信先デバイスの名前 (例えば *RemoteDevice1*) と、送信先デバイスへのアクセスに使用するサービス (例えば *SSH*) を指定できます。オプションで、トンネルの説明やタグなどの追加パラメータを指定することもできます。

input.json の内容

```
{
   "description": "Tunnel to remote device1",
   "destinationConfig": {
      "services": [ "SSH" ],
      "thingName": "RemoteDevice1"
   }
```

```
}
```

このコマンドを実行すると、新しいトンネルの作成と、送信元と送信先のアクセストークンが提供されます。

```
{
    "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>"
}
```

既存のトンネルを開き、ブラウザベースの SSH を使用する

たとえば、手動セットアップ方式または AWS IoT API リファレンス API を使用してRemoteDevice1、リモートデバイス のトンネルを作成したとします。次に、デバイスの既存のトンネルを開き、[Quick setup] (クイックセットアップ) を選択すると、ブラウザベースの SSH 機能を使用できます。既存のトンネル設定は編集できないため、手動セットアップ方式は使用できません。

ブラウザベースの SSH 機能を使用するには、ソースアクセストークンのダウンロードや、ローカルプロキシの設定は必要ありません。ウェブベースのローカルプロキシが自動的に設定されるので、リモートデバイスとのやり取りを開始できます。

クイックセットアップ方式とブラウザベースの SSH を使用するには

1. 作成した RemoteDevice1 というモノの詳細ページに移動し、[Create secure tunnel] (セキュアトンネルの作成) を選択します。
2. [Use existing tunnel] (既存のトンネルを使用する) を選択して、リモートデバイス用に作成した最新のオープントンネルを開きます。トンネル設定は編集できないため、トンネルの手動セットアップ方式を使用することはできません。クイックセットアップ方式を使用するには、[Quick setup] (クイックセットアップ) を選択します。
3. トンネル設定の詳細を確認して確定し、トンネルを作成します。トンネルの設定は編集できません。

   トンネルを作成すると、セキュアトンネリングでは rotateTunnelAccessToken API オペレーションを使用して元のアクセストークンを取り消し、新しいアクセストークンを生成します。リモートデバイスが MQTT を使用している場合、これらのトークンは、リモートデバイスがサブ

スクライブしている MQTT トピックに自動的に配信されます。これらのトークンを送信元デバイスに手動でダウンロードすることもできます。

トンネルを作成したら、ブラウザベースの SSH を使用して、コンテキスト内のコマンドラインインターフェイスを使用してコンソールから直接リモートデバイスとやり取りできます。このコマンドラインインターフェイスを使用するには、作成したモノのトンネルを選択し、詳細ページで [Command-Line Interface] (コマンドラインインターフェイス) セクションを展開します。ローカルプロキシは既に設定されているので、コマンドを入力して、リモートデバイス `RemoteDevice1` へのアクセスや、やり取りをすぐに開始できます。

クイックセットアップ方式とブラウザベースの SSH の使用方法の詳細については、「トンネルを開き、ブラウザベースの SSH を使用してリモートデバイスにアクセスします。」を参照してください。

クリーンアップ

- トンネルを閉じる

  トンネルの使用が終わったら、トンネルを閉じることをお勧めします。トンネルを指定した継続期間を超えて開いたままにしていると、閉じた状態になることもあります。トンネルが閉じた場合、再び開くことはできません。閉じたトンネルを開いて [Duplicate tunnel] (トンネルの複製) を選択すれば、トンネルを複製できます。使用するトンネルの継続時間を指定し、新しいトンネルを作成します。

  - AWS IoT コンソールから個々のトンネルまたは複数のトンネルを閉じるには、トンネルハブに移動し、閉じるトンネルを選択して、[Close tunnel] (トンネルを閉じる) を選択します。
  - AWS IoT API リファレンス API を使用して個々のトンネルまたは複数のトンネルを閉じるには、CloseTunnel API オペレーションを使用します。

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- トンネルの削除

  からトンネルを完全に削除できます AWS アカウント。

  > **Warning**
  >
  > 削除の操作は永続的で、元には戻せません。

- AWS IoT コンソールから個々のトンネルまたは複数のトンネルを削除するには、トンネルハブに移動し、削除するトンネルを選択して、[Delete tunnel] (トンネルを削除する) を選択します。
- AWS IoT API リファレンス API を使用して個々のトンネルまたは複数のトンネルを削除するには、CloseTunnel API オペレーションを使用します。API を使用する場合は、`delete` フラグを `true` に設定します。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

ローカルプロキシ

ローカルプロキシは、WebSocketの安全な接続を介したセキュアトンネリングを使用して、送信元デバイスで実行中のアプリケーションによって送信されたデータを送信します。GitHub からローカルプロキシソースをダウンロードすることができます。

ローカルプロキシは、`source` または `destination` の 2 つのモードで実行できます。送信元モードでは、ローカルプロキシは TCP 接続を開始するクライアントアプリケーションと同じデバイスまたはネットワーク上で実行されます。宛先モードでは、ローカルプロキシは宛先アプリケーションとともにリモートデバイス上で実行されます。トンネル多重化を使用すると、1 つのトンネルで一度に最大 3 つのデータストリームをサポートできます。セキュアトンネリングでは、データストリームごとに複数の TCP 接続が使用されるため、タイムアウトの可能性が低くなります。詳細については、「セキュアトンネル内でのデータストリームの多重化と同時 TCP 接続の使用」を参照してください。

ローカルプロキシの使用方法

送信元デバイスおよび宛先デバイスでローカルプロキシを実行して、セキュアトンネリングエンドポイントにデータを送信できます。デバイスが、ウェブプロキシを使用するネットワーク内にある場合、ウェブプロキシは接続をインターネットに転送する前にそれらをインターセプトできます。この場合、ウェブプロキシを使用するようにローカルプロキシを設定する必要があります。詳細については、「ウェブプロキシを使用するデバイスのローカルプロキシを設定します」を参照してください。

ローカルプロキシのワークフロー

次の手順は、送信元デバイスおよび宛先デバイスでローカルプロキシがどのように実行されるかを説明します。

1. ローカルプロキシをセキュアトンネリングに接続する

   まず、ローカルプロキシはセキュアトンネリングへの接続を確立する必要があります。ローカルプロキシを起動するとき、次の引数を使用します。

   - トンネルが開かれる AWS リージョン を指定する`-r`引数。
   - `OpenTunnel` から返された送信元または宛先のクライアントアクセストークンを渡す `-t` 引数。

   > **Note**
   >
   > 同じクライアントアクセストークン値を使用する 2 つのローカルプロキシを同時に接続することはできません。

2. 送信元または宛先アクションの実行

   WebSocket 接続が確立されると、ローカルプロキシは、その設定に応じて、送信元モードまたは宛先モードのいずれかのアクションを実行します。

   デフォルトでは、入力/出力 (I/O) エラーが発生した場合、または WebSocket 接続が予期せず閉じられた場合、ローカルプロキシは、セキュアトンネリングに再接続しようとします。これにより、TCP 接続が閉じられます。TCP ソケットエラーが発生した場合、ローカルプロキシはトンネルを経由してメッセージを送信し、相手側に TCP 接続を閉じるように通知します。デフォルトでは、ローカルプロキシは常に SSL 通信を使用します。

3. ローカルプロキシの停止

   トンネルを使用した後は、ローカルプロキシプロセスを停止しても安全です。`CloseTunnel` を呼び出して、トンネルを明示的に閉じることを推奨します。アクティブなトンネルクライアントは、`CloseTunnel` を呼び出した直後に閉じられないことがあります。

を使用してトンネル AWS マネジメントコンソール を開き、SSH セッションを開始する方法の詳細については、「」を参照してくださいトンネルを開き、リモートデバイスへの SSH セッションを開始します。

ローカルプロキシのベストプラクティス

ローカルプロキシを実行する場合は、以下のベストプラクティスに従ってください。

- `-t` ローカルプロキシ引数を使用してアクセストークンを渡すことは避けてください。`AWSIOT_TUNNEL_ACCESS_TOKEN` 環境変数を使用して、ローカルプロキシのアクセストークンを設定することをお勧めします。
- オペレーティングシステムまたは環境で、ローカルプロキシ実行可能ファイルを最小特権で実行します。
  - Windows では、ローカルプロキシを管理者として実行することは避けてください。
  - Linux および macOS の場合は、ローカルプロキシを root として実行することは避けてください。
- 個別のホスト、コンテナ、サンドボックス、chroot jail、または仮想化環境でローカルプロキシを実行することを検討してください。
- ツールチェーンに応じて、関連するセキュリティフラグを持つローカルプロキシを構築します。
- 複数のネットワークインターフェイスを持つデバイスでは、`-b`引数を使用して、送信先アプリケーションとの通信に使用されるネットワークインターフェイスに、TCP ソケットをバインドします。

コマンドと出力の例

以下は、実行するコマンドと、対応する出力の例です。この例は、ローカルプロキシを `source` および `destination` モードの両方で設定する方法を示しています。ローカルプロキシは HTTPS プロトコルを WebSockets にアップグレードして存続期間が長い接続を確立し、接続を介して安全なトンネリングデバイスエンドポイントへのデータの送信を開始します。

次のコマンドを実行する前に:

トンネルをオープンし、送信元と送信先のクライアントアクセストークンを取得しておく必要があります。また、前述の説明に従ってローカルプロキシを構築しておく必要もあります。ローカルプロキシを構築するには、GitHub レポジトリでローカルプロキシのソースコードを開き、ローカルプロキシを構築、インストールするための手順に従います。

> **Note**
>
> この例で使用される以下のコマンドは、ローカルプロキシを実行した後の異なるステップ (先ほど説明したもの) の概要を説明するために verbosity フラグを使用します。このフラグはテストのみに使用することが推奨されます。

出典モードでのローカルプロキシの実行

以下のコマンドは、ローカルプロキシを送信元モードで実行する方法を示しています。

Linux/macOS

Linux または macOS では、ターミナルで次のコマンドを実行して、送信元でローカルプロキシを設定および起動します。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -s 5555 -v 5 -r us-west-2
```

コードの説明は以下のとおりです。

- `-s` は送信元リッスンポートです。ローカルプロキシを送信元モードで起動します。
- `-v` は出力の冗長性です。0 ~ 6 の値を指定できます。
- `-r` は、トンネルが開かれるエンドポイントリージョンです。

これらのパラメータの詳細については、「コマンドライン引数を使用して設定されるオプション」を参照してください。

Windows

Windows では、Linux や macOS の場合と同様にローカルプロキシを設定しますが、環境変数の定義方法は、他のプラットフォームとは異なります。cmd ウィンドウで以下のコマンドを実行し、送信元のローカルプロキシを設定して起動します。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -s 5555 -v 5 -r us-west-2
```

コードの説明は以下のとおりです。

- `-s` は送信元リッスンポートです。ローカルプロキシを送信元モードで起動します。
- `-v` は出力の冗長性です。0 ~ 6 の値を指定できます。
- `-r` は、トンネルが開かれるエンドポイントリージョンです。

これらのパラメータの詳細については、「コマンドライン引数を使用して設定されるオプション」を参照してください。

> **Note**
>
> ソースモードで最新バージョンのローカルプロキシを使用する場合は、下位互換性のために`--destination-client-type V1`ソースデバイスに AWS CLI パラメータを含める必要があります。これは、次のいずれかの送信先モードに接続する場合に適用されます。
>
> - AWS IoT デバイスクライアント
> - AWS IoT セキュアトンネリングコンポーネントまたは AWS IoT Greengrass Version 2 セキュアトンネリングコンポーネント
> - 2022 年以前に作成された AWS IoT Secure Tunneling デモコード
> - ローカルプロキシの 1.X バージョン
>
> このパラメータにより、更新された送信元のプロキシと古い送信先クライアント間の適切な通信が確保されます。ローカルプロキシのバージョンの詳細については、GitHub の「AWS IoT Secure Tunneling」を参照してください。

以下は、ローカルプロキシを `source` モードで実行した出力の例です。

```
...
...

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
```

送信先モードでのローカルプロキシの実行

以下のコマンドは、送信先モードでローカルプロキシを実行する方法を示しています。

Linux/macOS

Linux または macOS では、ターミナルで次のコマンドを実行して、宛先でローカルプロキシを設定および起動します。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -d 22 -v 5 -r us-west-2
```

コードの説明は以下のとおりです。

- -d は宛先アプリケーションです。ローカルプロキシを宛先モードで起動します。
- -v は出力の冗長性です。0 ~ 6 の値を指定できます。
- -r は、トンネルが開かれるエンドポイントリージョンです。

これらのパラメータの詳細については、「コマンドライン引数を使用して設定されるオプション」を参照してください。

Windows

Windows では、Linux や macOS の場合と同様にローカルプロキシを設定しますが、環境変数の定義方法は、他のプラットフォームとは異なります。cmd ウィンドウで以下のコマンドを実行して、宛先のローカルプロキシを設定して起動します。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -d 22 -v 5 -r us-west-2
```

コードの説明は以下のとおりです。

- `-d` は宛先アプリケーションです。ローカルプロキシを宛先モードで起動します。
- `-v` は出力の冗長性です。0 ~ 6 の値を指定できます。
- `-r` は、トンネルが開かれるエンドポイントリージョンです。

これらのパラメータの詳細については、「コマンドライン引数を使用して設定されるオプション」を参照してください。

> **Note**
>
> 宛先モードで最新バージョンのローカルプロキシを使用する場合は、下位互換性のために`--destination-client-type V1`宛先デバイスに AWS CLI パラメータを含める必要があります。これは、次のいずれかの送信元モードに接続する場合に適用されます。
>
> - AWS コンソールからのブラウザベースのセキュアトンネリング。
> - ローカルプロキシの 1.X バージョン
>
> このパラメータにより、更新された送信先プロキシと古い送信元クライアント間の適切な通信が保証されます。ローカルプロキシのバージョンの詳細については、GitHub の「AWS IoT Secure Tunneling」を参照してください。

以下は、ローカルプロキシを `destination` モードで実行した出力の例です。

```
...
```

```
...

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...
```

ウェブプロキシを使用するデバイスのローカルプロキシを設定します

AWS IoT デバイスでローカルプロキシを使用して、 AWS IoT セキュアトンネリング APIsと通信できます。ローカルプロキシは、WebSocketの安全な接続を介したセキュアトンネリングを通して、デバイスアプリケーションによって送信されたデータを送信します。ローカルプロキシは `source` または `destination` モードで実行できます。`source` モードでは、TCP 接続を開始するものと同じデバイスまたはネットワークで実行されます。`destination` モードでは、ローカルプロキシが送信先アプリケーションと共にリモートデバイス上で実行されます。詳細については、「ローカルプロキシ」を参照してください。

AWS IoT セキュアトンネリングを使用するには、ローカルプロキシがインターネットに直接接続する必要があります。セキュアトンネリングを使用した存続期間が長い TCP 接続については、ローカ

ルプロキシが HTTPS リクエストをアップグレードして、セキュアトンネリングデバイス接続エンドポイントのいずれかへの WebSockets 接続を確立します。

デバイスが、ウェブプロキシを使用するネットワーク内にある場合、ウェブプロキシは接続をインターネットに転送する前にそれらをインターセプトできます。セキュアトンネリングデバイス接続エンドポイントに対して存続期間が長い接続を確立するには、WebSocket の仕様で説明されているとおり、ウェブプロキシを使用するようにローカルプロキシを設定します。

> **Note**
>
> AWS IoT Device Clientは、ウェブプロキシを使用するデバイスをサポートしていません。ウェブプロキシに対応させるには、以下で説明されているように、ローカルプロキシを使用してウェブプロキシに対応するように設定する必要があります。

以下の手順は、ローカルプロキシとウェブプロキシがどのように連動するかを説明するものです。

1. ローカルプロキシは、HTTP CONNECT リクエストをウェブプロキシに送信します。このリクエストには、セキュアトンネリングサービスのリモートアドレスと共に、ウェブプロキシ認証情報が含まれています。
2. 次に、ウェブプロキシがリモートセキュアトンネリングエンドポイントに対する存続期間が長い接続を作成します。
3. TCP 接続が確立され、ローカルプロキシは、データ伝送の送信元モードと送信先モードの両方で稼働します。

この手順を完了するには、次のステップを実行します。

- ローカルプロキシを構築します
- ウェブプロキシを設定します
- ローカルプロキシを設定、起動します

ローカルプロキシを構築します

GitHub リポジトリで、ローカルプロキシのソースコードを開き、ローカルプロキシを構築、インストールするための手順に従います。

ウェブプロキシを設定します

ローカルプロキシは、HTTP/1.1 の仕様で説明されている HTTP トンネリングメカニズムに依存しています。この仕様に準拠するには、ウェブプロキシが CONNECT メソッドの使用をデバイスに許可する必要があります。

ウェブプロキシの設定方法は、使用しているウェブプロキシと Wウェブプロキシのバージョンによって異なります。ウェブプロキシを正しく設定されるように、ウェブプロキシのドキュメントを確認してください。

Web プロキシを設定するには、まず ウェブプロキシ URLを特定し、ウェブプロキシが、HTTP トンネリングをサポートしているかどうかを確認します。ウェブプロキシURL は、後でローカルプロキシを設定して起動する時に使用されます。

1. ウェブプロキシの URL を特定します

   ウェブプロキシ URL は以下の形式になるでしょう。

   ```
   protocol://web_proxy_host_domain:web_proxy_port
   ```

   AWS IoT セキュアトンネリングは、ウェブプロキシの基本認証のみをサポートします。ベーシック認証を使用するには、**username**および**password**をウェブプロキシ URLの一部として指定する必要があります。ウェブプロキシ URL は、次の形式になります。

   ```
   protocol://username:password@web_proxy_host_domain:web_proxy_port
   ```

   - *protocol* は、http または https にすることが可能です。https を使用することをお勧めします。
   - [*proxy_host_domain*]は、ウェブプロキシの IP アドレス、またはウェブプロキシの IP アドレスを解決する DNS 名です。
   - [*web_proxy_port*]は、ウェブプロキシがリッスンするポートです。
   - ウェブプロキシはこの**username**および**password**を使用してリクエストを認証します。

2. ウェブプロキシの URL をテストします

   Web プロキシが TCP トンネリングをサポートしているかどうかを確認するには、curlコマンドを使用し、必ず2xxまたは3xxレスポンスを受け取る事を確認してください。

例えば、ウェブプロキシ URL が https://server.com:1235 の場合は、curl コマンドで proxy-insecure フラグを使用します。これは、ウェブプロキシが自己署名証明書に依存する可能性があるためです。

```
export HTTPS_PROXY=https://server.com:1235
curl -I https://aws.amazon.com --proxy-insecure
```

ウェブプロキシ URL にhttpポート (例、http://server.com:1234)がある場合は、proxy-insecureフラグを使う必要はありません。

```
export HTTPS_PROXY=http://server.com:1234
curl -I https://aws.amazon.com
```

ローカルプロキシを設定、起動します

ウェブプロキシを使用するためにローカルプロキシを設定するには、DNS ドメイン名、またはウェブプロキシが使用する IP アドレスとポート番号でHTTPS_PROXY 環境変数を設定する必要があります。

ローカルプロキシを設定したら、この README ドキュメントに説明されているようにローカルプロキシを使用できます。

> **Note**
>
> 環境変数の宣言では、大文字と小文字が区別されます。一度、すべて大文字またはすべて小文字のいずれかを使用して、それぞれの変数を定義する事をお勧めいたします。以下の例は、大文字で宣言された環境変数を示しています。同じ変数が大文字と小文字の両方を使用して指定されている場合は、小文字を使用して指定された変数が優先されます。

以下のコマンドは、ウェブプロキシを使用するように送信先で実行されているローカルプロキシを設定して、ローカルプロキシを開始する方法を示しています。

- AWSIOT_TUNNEL_ACCESS_TOKEN: この変数は、送信先のクライアントアクセストークン (CAT) を保持します。
- HTTPS_PROXY: この変数は、ローカルプロキシを設定するための Web プロキシ URL または IP アドレスを保持します。

以下の例にあるコマンドは、使用するオペレーティングシステムと、ウェブプロキシが HTTP ポートまたは HTTPS ポートのどちらでリッスンしているかに応じて異なります。

HTTP ポートでリッスンする ウェブプロキシ

ウェブプロキシが HTTP ポートでリッスンしている場合は、HTTPS_PROXY 変数にウェブプロキシ URL または IP アドレスを指定できます。

Linux/macOS

Linux または macOS では、ターミナルで次のコマンドを実行して、HTTP ポートをリッスンする Web プロキシを使用するように宛先でローカルプロキシを設定および起動します。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

プロキシで認証する必要がある場合は、**username**および**password**をHTTPS_PROXY変数の一部として実行します。

```
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

Windows では、Linux や macOS の場合と同様にローカルプロキシを設定しますが、環境変数の定義方法は、他のプラットフォームとは異なります。cmd ウィンドウで以下のコマンドを実行して、HTTP ポートをリッスンするウェブプロキシを使用するように送信先のローカルプロキシを設定して、ローカルプロキシを開始します。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22
```

プロキシで認証する必要がある場合は、**username**および**password**をHTTPS_PROXY変数の一部として実行する必要があります。

```
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 ポートでリッスンする ウェブプロキシ

ウェブプロキシが HTTPS ポートでリッスンしている場合は、以下のコマンドを実行します。

> **Note**
>
> Web プロキシに自己署名証明書を使用している場合、またはネイティブの OpenSSL サポートおよびデフォルト設定がない OS でローカルプロキシを実行している場合は、[Certificate setup](証明書セットアップ)セクションを GitHub リポジトリに追加します。

以下のコマンドは、HTTP プロキシ用にウェブプロキシを設定した方法と似ていますが、上記の説明どおりにインストールされた証明書ファイルへのパスも指定するという点が異なります。

Linux/macOS

Linux または macOS では、ターミナルで以下のコマンドを実行して、HTTPS ポートをリッスンするウェブプロキシを使用するように送信先で実行されているローカルプロキシを設定します。

```
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
```

プロキシで認証する必要がある場合は、HTTPS_PROXY 変数の一部として **username** および **password** を指定する必要があります。

```
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

Windows では、cmd ウィンドウで以下のコマンドを実行して、HTTP ポートをリッスンするウェブプロキシを使用するように送信先で実行されているローカルプロキシを設定し、ローカルプロキシを開始します。

```
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
```

プロキシで認証する必要がある場合は、HTTPS_PROXY 変数の一部として **username** および **password** を指定する必要があります。

```
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
```

コマンドと出力の例

以下は、Linux OS で実行するコマンドと、対応する出力の例です。この例では、HTTP ポートでリッスンしているウェブプロキシと、ウェブプロキシを source および destination モードの両方で使用するようにローカルプロキシを設定する方法を説明します。これらのコマンドを実行する前に、トンネルをオープンし、送信元と宛先のクライアントアクセストークンを取得しておく必要があります。また、前述の説明に従ってローカルプロキシを構築し、ウェブプロキシを設定しておく必要もあります。

ここでは、ローカルプロキシを起動した後の手順の概要を示します。ローカルプロキシ:

- ウェブプロキシURLを識別し、そのURLを使用してプロキシサーバーに接続できるようにします。
- ウェブプロキシとの TCP 接続を確立します。
- HTTP CONNECT リクエストをウェブプロキシに送信し、接続が確立されたことを示す HTTP/1.1 200 レスポンスを待ちます。
- HTTPS プロトコルを WebSockets にアップグレードして、存続時間が長い接続を確立します。
- セキュアトンネリングデバイスエンドポイントへの接続を介してデータの送信を開始します。

> **Note**
>
> この例で使用される以下のコマンドは、ローカルプロキシを実行した後の異なるステップ (先ほど説明したもの) の概要を説明するために verbosity フラグを使用します。このフラグはテストのみに使用することが推奨されます。

出典モードでのローカルプロキシの実行

以下のコマンドは、ローカルプロキシを出典モードで実行する方法を示しています。

```
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
```

以下は、ローカルプロキシを source モードで実行したサンプル出力です。

```
...

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
```

送信先モードでのローカルプロキシの実行

以下のコマンドは、送信先モードでローカルプロキシを実行する方法を示しています。

```
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
```

以下は、ローカルプロキシを destination モードで実行したサンプル出力です。

```
...

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...
```

セキュアトンネル内でのデータストリームの多重化と同時 TCP 接続の使用

セキュアトンネリング多重化機能を使用すると、トンネルごとに複数のデータストリーミングを使用できます。多重化により、複数のデータストリームを使用してデバイスをトラブルシューティングできます。また、複数のローカルプロキシを構築、デプロイ、開始したり、同じデバイスに対して複

数のトンネルを開いたりする必要がなくなるため、運用負荷を緩和できます。例えば、複数の HTTP および SSH ストリーミングを送信する必要があるウェブブラウザの場合、多重化を使用できます。

データストリームごとに、 AWS IoT セキュアトンネリングは同時 TCP 接続をサポートします。同時接続を使用すると、クライアントから複数のリクエストがあった場合にタイムアウトする可能性が低くなります。例えば、送信先デバイスに対してローカルなウェブサーバーにリモートアクセスするときのロード時間を短縮できます。

次のセクションでは、多重化と同時 TCP 接続の使用、およびそれらのさまざまな使用例について詳しく説明します。

トピック

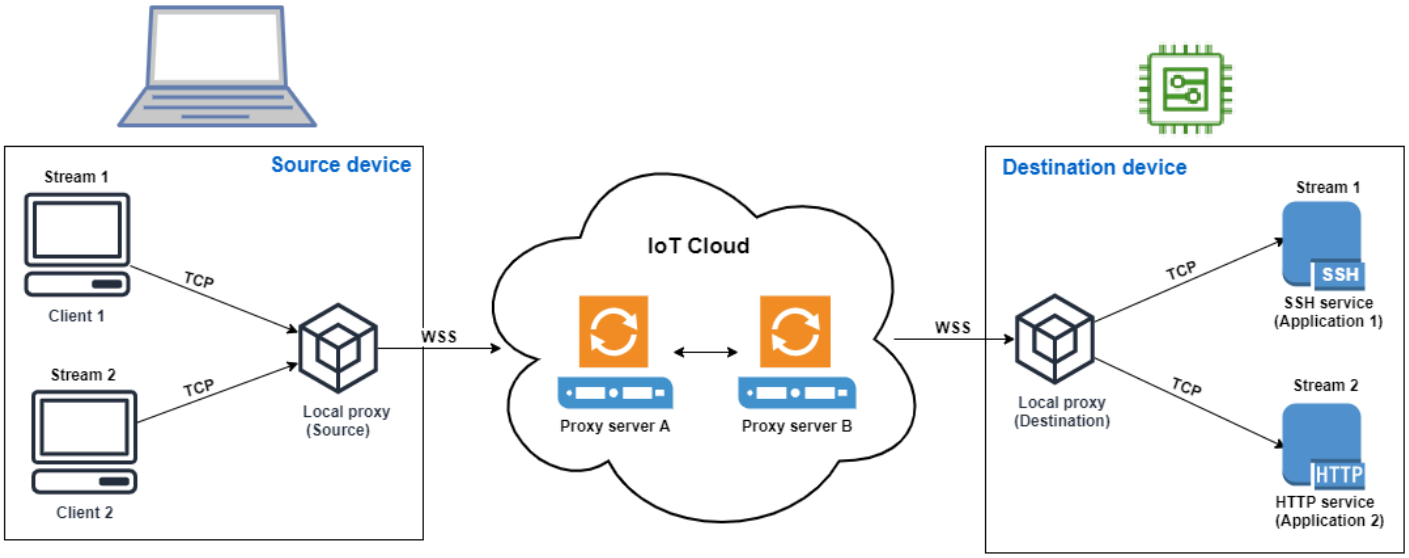
- セキュアトンネル内の複数のデータストリームを多重化する
- セキュアトンネルでの同時 TCP 接続の使用

セキュアトンネル内の複数のデータストリームを多重化する

多重化機能は、複数の接続またはポートを使用するデバイスに使用できます。多重化は、問題のトラブルシューティングのためにリモートデバイスへの複数の接続が必要な場合にも使用できます。例えば、複数の HTTP および SSH データストリーミングを送信する必要があるウェブブラウザの場合、多重化を使用できます。両方のストリームからのアプリケーションデータが、多重化されたトンネルを介して同時にデバイスに送信されます。

ユースケースの例

例えば、デバイス上のウェブアプリケーションに接続して、一部のネットワーキングパラメータを変更すると同時に、ターミナルからシェルコマンドを発行して、デバイスが新しいネットワーキングパラメータで正常に動作していることを確認する必要がある場合があります。このシナリオでは、ウェブアプリケーションとターミナルに同時にアクセスするために、HTTP と SSH の両方を介してデバイスに接続し、2 つの並列データストリームを転送する必要がある場合があります。多重化機能を使用すると、これらの 2 つの独立したストリーミングを同時に同じトンネルを介して転送できます。



多重化トンネルの設定方法

次の手順では、複数のポートへの接続を必要とするアプリケーションを使用してデバイスをトラブルシューティングするための多重化トンネルを設定する方法について説明します。HTTP ストリーミングと SSH ストリーミングの 2 つの多重化されたストリーミングを持つ 1 つのトンネルを設定します。

1. オプションの設定ファイル

   設定ファイルを使用して送信元および送信先デバイスを設定できます。ポートマッピングが頻繁に変更される可能性がある場合は、設定ファイルを使用します。CLI を使用して明示的にポートマッピングを指定する場合、または指定されたリスニングポートでローカルプロキシを開始する必要がない場合は、このステップを省略できます。設定ファイルの使用方法の詳細については、GitHub の「 Options set via --config」(--config を介して設定されるオプション) を参照してください。

   1. 送信元デバイスでローカルプロキシを実行するフォルダに、`Config` という設定フォルダを作成します。このフォルダ内で、次の内容を含む `SSHSource.ini` というファイルを作成します。

      ```
      HTTP1 = 5555
      SSH1 = 3333
      ```

2. 送信先デバイスのローカルプロキシを実行するフォルダに、Config という設定フォルダを作成します。このフォルダ内で、次の内容を含む SSHDestination.ini というファイルを作成します。

```
HTTP1 = 80
SSH1 = 22
```

2. トンネルを開く

OpenTunnel API オペレーションまたは open-tunnel CLI コマンドを使用してトンネルを開きます。SSH1 および をサービスHTTP1として指定し、リモートデバイスに対応する AWS IoT モノの名前を指定して、送信先を設定します。SSH アプリケーションと HTTP アプリケーションはこのリモートデバイスで実行されています。 AWS IoT レジストリに IoT モノがすでに作成されている必要があります。詳細については、「レジストリを使用したモノの管理」を参照してください。

```
aws iotsecuretunneling open-tunnel \
 --destination-config thingName=RemoteDevice1,services=HTTP1,SSH1
```

このコマンドを実行すると、ローカルプロキシの実行に使用する送信元と送信先のアクセストークンが生成されます。

```
{
 "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. ローカルプロキシを設定、起動します

ローカルプロキシを実行する前に、 AWS IoT Device Client を設定するか、GitHub からローカルプロキシソースコードをダウンロードして、任意のプラットフォーム用に構築します。その後、送信先と送信元のローカルプロキシを起動して、セキュアトンネルに接続できます。ローカルプロキシの設定と使用の詳細については、「ローカルプロキシの使用方法」を参照してください。

> **Note**
>
> 送信元デバイスでは、設定ファイルを使用しないか、CLI を使用してポートマッピングを指定しない場合でも、同じコマンドを使用してローカルプロキシを実行できます。ソースモードのローカルプロキシは、自動的に使用可能なポートを選択して、マッピングを使用します。

Start local proxy using configuration files

次のコマンドを実行し、設定ファイルを使用して送信元モードと送信先モードでローカルプロキシを実行します。

```
// ----------------- 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

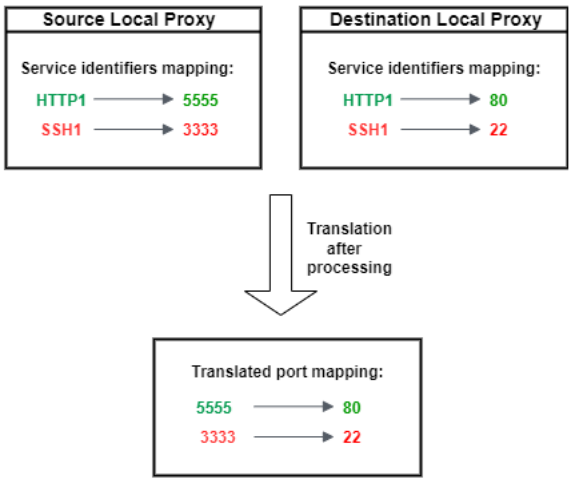
次のコマンドを実行し、CLI を使用してポートマッピングを明示的に指定することにより、送信元モードと送信先モードでローカルプロキシを実行します。

```
// ----------------- 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
```

SSH および HTTP 接続からのアプリケーションデータを、多重化されたトンネルを介して同時に転送できるようになりました。以下のマップからわかるように、サービス識別子は、送信元デバイスと送信先デバイス間のポートマッピングを変換する読み取り可能な形式として機能します。この設定では、安全なトンネリングは、送信元デバイスのポート *5555* からのすべての着信 HTTP トラフィッ

クを送信先デバイスのポート *80* に転送し、ポート *3333* からのすべての着信 SSH トラフィックを送信先デバイスのポート *22* に転送します。



セキュアトンネルでの同時 TCP 接続の使用

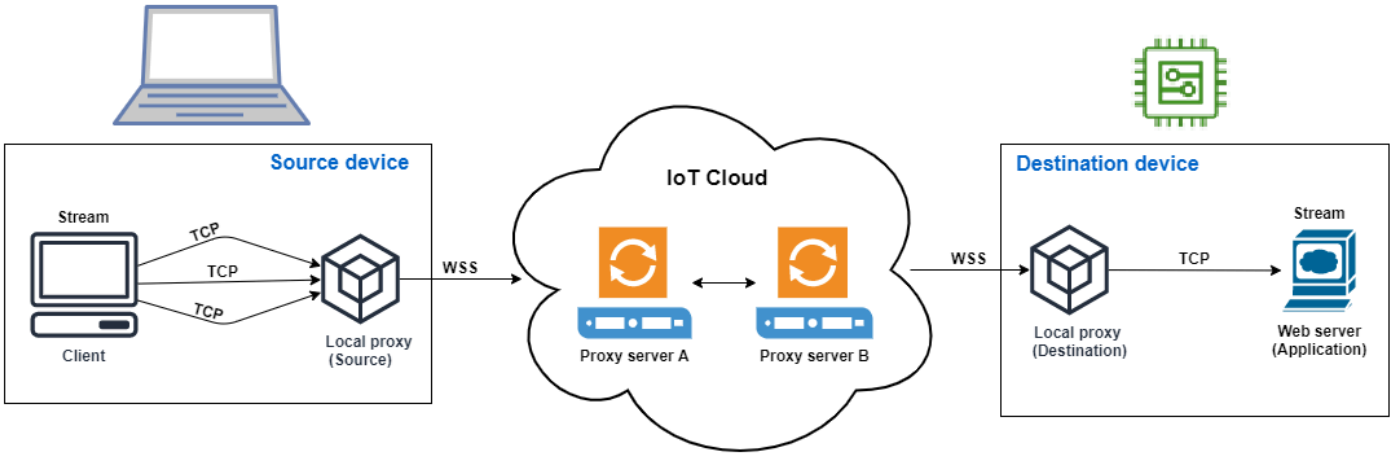
AWS IoT セキュアトンネリングは、データストリームごとに複数の TCP 接続を同時にサポートします。この機能は、リモートデバイスへの同時接続が必要な場合に使用できます。同時 TCP 接続を使用すると、クライアントから複数のリクエストがあった場合にタイムアウトする可能性が低くなります。例えば、複数のコンポーネントが実行されているウェブサーバーにアクセスする場合、同時 TCP 接続を行うと、サイトのロードにかかる時間を短縮できます。

> **Note**
>
> 同時 TCP 接続の帯域幅制限は、それぞれ 1 秒あたり 800 KB です AWS アカウント。 AWS IoT セキュアトンネリングでは、受信リクエストの数に応じてこの制限を設定できます。

ユースケースの例

送信先デバイスに対してローカルで、複数のコンポーネントが実行されているウェブサーバーにリモートにアクセスする必要があるとします。単一の TCP 接続でウェブサーバーにアクセスを試みているときに、シーケンシャルロードを行うと、サイト上のリソースのロードにかかる時間が長くなる可能性があります。同時 TCP 接続により、サイトのリソース要件を満たすことでロード時間を短縮でき、アクセス時間を短縮できます。次の図は、リモートデバイスで実行されているウェブサーバーアプリケーションへのデータストリームに対して、同時 TCP 接続がどのようにサポートされるかを示しています。

> **Note**
>
> トンネルを使用してリモートデバイスで実行されている複数のアプリケーションにアクセスする場合は、トンネル多重化を使用できます。詳細については、「セキュアトンネル内の複数のデータストリームを多重化する」を参照してください。



同時 TCP 接続を使用する方法

次の手順では、同時 TCP 接続を使用してリモートデバイスのウェブブラウザにアクセスする方法について説明します。クライアントからのリクエストが複数ある場合、 AWS IoT セキュアトンネリングはリクエストを処理するための同時 TCP 接続を自動的にセットアップするため、ロード時間が短縮されます。

1. トンネルを開く

   OpenTunnel API オペレーションまたは open-tunnel CLI コマンドを使用してトンネルを開きます。HTTP をサービスとして指定し、リモートデバイスに対応する AWS IoT のモノの名前を指定して、送信先を設定します。ウェブサーバーアプリケーションはこのリモートデバイスで実行されています。 AWS IoT レジストリに IoT モノを作成しておく必要があります。詳細については、「レジストリを使用したモノの管理」を参照してください。

   ```
   aws iotsecuretunneling open-tunnel \
    --destination-config thingName=RemoteDevice1,services=HTTP
   ```

このコマンドを実行すると、ローカルプロキシの実行に使用する送信元と送信先のアクセストークンが生成されます。

```
{
 "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. ローカルプロキシを設定、起動します

ローカルプロキシを実行する前に、GitHub からローカルプロキシソースコードをダウンロードし、選択したプラットフォーム用にビルドします。その後、送信先と送信元のローカルプロキシを起動して、セキュアトンネルに接続し、リモートウェブサーバーアプリケーションの使用を開始できます。

> **Note**
>
> AWS IoT 安全なトンネリングで同時 TCP 接続を使用するには、ローカルプロキシの最新バージョンにアップグレードする必要があります。 AWS IoT Device Client を使用してローカルプロキシを設定する場合、この機能は使用できません。

```
// 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
```

ローカルプロキシの設定と使用の詳細については、「ローカルプロキシの使用方法」を参照してください。

トンネルを使用してウェブサーバーアプリケーションにアクセスできるようになりました。 AWS IoT セキュアトンネリングは、クライアントから複数のリクエストがある場合、同時 TCP 接続を自動的にセットアップして処理します。

リモートデバイスの設定と IoT エージェントの使用

IoT エージェントは、クライアントアクセストークンを含む MQTT メッセージを受信し、リモートデバイスでローカルプロキシを起動するために使用されます。MQTT を使用してセキュアトンネリングでクライアントアクセストークンを配信する場合は、リモートデバイスに IoT エージェントをインストールして実行する必要があります。IoT エージェントは、次の予約済み IoT MQTT トピックにサブスクライブする必要があります。

> **Note**
>
> 予約済み MQTT トピックへのサブスクライブ以外の方法で送信先クライアントアクセストークンをリモートデバイスに配信する場合は、送信先クライアントアクセストークン (CAT) リスナーおよびローカルプロキシが必要になることがあります。CAT リスナーは、選択したクライアントアクセストークン配信メカニズムを使用して、送信先モードでローカルプロキシを起動できる必要があります。

IoT エージェントスニペット

IoT エージェントは、次の予約済み IoT MQTT トピックにサブスクライブする必要があります。これにより、MQTT メッセージを受信してローカルプロキシを起動できます。

`$aws/things/thing-name/tunnels/notify`

ここで、 `thing-name`はリモートデバイスに関連付けられた AWS IoT モノの名前です。

MQTT メッセージペイロードの例を次に示します。

```
{
    "clientAccessToken": "destination-client-access-token",
    "clientMode": "destination",
    "region": "aws-region",
    "services": ["destination-service"]
}
```

MQTT メッセージを受信した後、IoT エージェントは適切なパラメータを使用してリモートデバイスでローカルプロキシを起動する必要があります。

次の Java コードは、Java ライブラリのAWS IoT デバイス SDK および ProcessBuilder を使用して、セキュアトンネリングと連携する簡単な IoT エージェントを構築する方法を示しています。

```
// 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);
        }
    }
}
```

トンネルへのアクセスの制御

セキュアトンネリングは、IAM アクセス許可ポリシーで使用するために、サービス固有のアクション、リソース、および条件コンテキストキーを提供します。

トンネルアクセスの前提条件

- IAM ポリシーを使用して AWS リソースを保護する方法について説明します。
- IAM 条件を作成および評価する方法について説明します。
- AWS リソースタグを使用してリソースを保護する方法について説明します。

トンネルアクセスポリシー

セキュアトンネリング API を使用する権限を認証するには、次のポリシーを使用する必要があります。 AWS IoT セキュリティの詳細については、「」を参照してくださいの ID とアクセスの管理 AWS IoT。

iot:OpenTunnel

iot:OpenTunnel ポリシーアクションは、 OpenTunnel を呼び出すためのプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で以下を実行します。

- ワイルドカードトンネル ARN を指定します。

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

- 特定の IoT のモノの OpenTunnel 権限を管理するためのモノの ARN を指定します。

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

たとえば、次のポリシーステートメントでは、TestDevice という IoT モノへのトンネルを開くことができます。

```
{
    "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 ポリシーアクションは、次の条件キーをサポートします。

- iot:ThingGroupArn
- iot:TunnelDestinationService
- aws:RequestTag/*tag-key*
- aws:SecureTransport
- aws:TagKeys

以下のポリシーステートメントでは、モノが TestGroup で始まる名前のモノのグループに属し、そのトンネルの設定済み送信先サービスが SSH である場合、モノにトンネルを開くことができます。

```
{
```

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

また、リソースタグを使用して、トンネルを開くアクセス許可を制御することもできます。たとえば、次のポリシーステートメントでは、タグキー Owner が Admin の値で存在し、他にタグが指定されていない場合に、トンネルを開くことができます。タグの使用に関する一般情報については、「AWS IoT リソースのタグ付け」を参照してください。

```
{
    "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 ポリシーアクションは、RotateTunnelAccessToken を呼び出すためのプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で以下を実行します。

- 完全修飾トンネル ARN を指定します。

  arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

  また、次のワイルドカードトンネル ARN を使用することもできます。

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*
- 特定の IoT のモノの RotateTunnelAccessToken 権限を管理するためのモノの ARN を指定します。

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

たとえば、次のポリシーステートメントでは、トンネルの送信元アクセストークンまたはクライアントの宛先アクセストークンを、TestDevice という IoT のモノに対してローテーションできます。

```
{
    "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 ポリシーアクションは、次の条件キーをサポートします。

- iot:ThingGroupArn
- iot:TunnelDestinationService
- iot:ClientMode
- aws:SecureTransport

以下のポリシーステートメントでは、モノが TestGroup で始まる名前のモノのグループに属し、そのトンネルの設定済み宛先サービスが SSH で、クライアントが DESTINATION モードである場合、モノに対する宛先アクセストークンをローテーションすることができます。

```
{
    "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 ポリシーアクションは、DescribeTunnel を呼び出すためのプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で、完全修飾 ARN を指定します。

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

また、次のワイルドカード ARN を使用することもできます。

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:DescribeTunnel ポリシーアクションは、次の条件キーをサポートします。

- aws:ResourceTag/*tag-key*
- aws:SecureTransport

次のポリシーステートメントでは、要求されたトンネルに DescribeTunnel の値が Owner のキーでタグ付けされている場合に Admin を呼び出すことができます。

```
{
    "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 ポリシーアクションは、ListTunnels を呼び出すプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で以下を実行します。

- ワイルドカードトンネル ARN を指定します。

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*
- 特定の IoT のモノの ListTunnels 権限を管理するためのモノの ARN を指定します。

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

iot:ListTunnels ポリシーアクションは、条件キー aws:SecureTransport をサポートします。

以下のポリシーステートメントでは、TestDevice という名前のモノのトンネルを一覧表示することができます。

```
{
    "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 ポリシーアクションは、ListTagsForResource を呼び出すためのプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で、完全修飾 ARN を指定します。

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

また、次のワイルドカードトンネル ARN を使用することもできます。

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:ListTagsForResource ポリシーアクションは、条件キー aws:SecureTransport をサポートします。

iot:CloseTunnel

iot:CloseTunnel ポリシーアクションは、CloseTunnel を呼び出すプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で、完全修飾 ARN を指定します。

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

また、次のワイルドカードトンネル ARN を使用することもできます。

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:CloseTunnel ポリシーアクションは、次の条件キーをサポートします。

- iot:Delete
- aws:ResourceTag/*tag-key*
- aws:SecureTransport

次のポリシーステートメントでは、リクエストの Delete パラメータが false で、リクエストされたトンネルに値が QATeam のキー Owner でタグ付けされている場合に、CloseTunnel を呼び出すことができます。

```
{
    "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 ポリシーアクションは、TagResource を呼び出すためのプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で、完全修飾 ARN を指定します。

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

また、次のワイルドカードトンネル ARN を使用することもできます。

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:TagResource ポリシーアクションは、条件キー aws:SecureTransport をサポートします。

iot:UntagResource

iot:UntagResource ポリシーアクションは、UntagResource を呼び出すためのプリンシパルアクセス許可を付与します。

IAM ポリシーステートメントの Resource 要素で、完全修飾 ARN を指定します。

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

また、次のワイルドカードトンネル ARN を使用することもできます。

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

`iot:UntagResource` ポリシーアクションは、条件キー `aws:SecureTransport` をサポートします。

クライアントアクセストークンをローテーションして AWS IoT 安全なトンネリング接続の問題を解決する

AWS IoT セキュアトンネリングを使用すると、トンネルが開いている場合でも接続の問題が発生する可能性があります。次のセクションでは、発生する可能性のある問題と、クライアントアクセストークンをローテーションして解決する方法について説明します。クライアントアクセストークン (CAT) をローテーションするには、RotateTunnelAccessToken API または rotate-tunnel-access-token AWS CLIを使用します。クライアントを送信元モードまたは宛先モードのどちらで使用する場合にエラーが発生するかに応じて、送信元モードまたは宛先モード、あるいはその両方で CAT をローテーションできます。

> **Note**
>
> - 送信元と宛先のどちらで CAT をローテーションする必要があるか不明な場合は、`RotateTunnelAccessToken` API 使用時に `ClientMode` を ALL に設定することで、送信元と宛先の両方で CAT をローテーションできます。
> - CAT をローテーションしてもトンネル期間は延長されません。例えば、トンネル期間が 12 時間で、トンネルが既に 4 時間開かれているとします。アクセストークンをローテーションすると、生成された新しいトークンは残りの 8 時間だけ使用できます。

トピック

- 無効なクライアントアクセストークンのエラー
- クライアントトークンの不一致エラー
- リモートデバイスの接続性に関する問題

無効なクライアントアクセストークンのエラー

AWS IoT セキュアトンネリングを使用する場合、同じクライアントアクセストークン (CAT) を使用して同じトンネルに再接続するときに、接続エラーが発生することがあります。この場合、ローカルプロキシはセキュアトンネリングプロキシサーバーに接続できません。クライアントを送信元モードで使用している場合は、次のエラーメッセージが表示されることがあります。

```
Invalid access token: The access token was previously used and cannot be used again
```

このエラーは、クライアントアクセストークン (CAT) がローカルプロキシで一度しか使用できず、その後無効となるために発生します。このエラーを解決するには、クライアントアクセストークンを SOURCE モードでローテーションさせ、送信元に新しい CAT を生成します。送信元 CAT をローテーションする方法の例については、「送信元 CAT のローテーションの例」を参照してください。

クライアントトークンの不一致エラー

> **Note**
>
> クライアントトークンを使用して CAT を再利用することは推奨されていません。代わりに、RotateTunnelAccessToken API を使用してクライアントアクセストークンをローテーションし、トンネルに再接続することをお勧めします。

クライアントトークンを使用している場合は、トンネルへの再接続に CAT を再利用できます。CAT を再利用するには、セキュアトンネリングに初めて接続するときに、クライアントトークンに CAT を提供する必要があります。セキュアトンネリングはクライアントトークンを保存するため、同じトークンを使用してその後に接続を試みる場合は、クライアントトークンも提供する必要があります。クライアントトークンの使用方法の詳細については、「GitHub でのローカルプロキシリファレンスの実装」を参照してください。

クライアントトークンを使用している場合、クライアントを送信元モードで使用していると、次のエラーが表示されることがあります。

```
Invalid client token: The provided client token does not match the client token
    that was previously set.
```

このエラーは、提供されたクライアントトークンが、トンネルにアクセスするときに CAT で提供されたクライアントトークンと一致しないために発生します。このエラーを解決するには、CAT を SOURCE モードでローテーションさせ、送信元に新しい CAT を生成します。例を以下に示します。

送信元 CAT のローテーションの例

以下は、RotateTunnelAccessToken API を SOURCE モードで実行して、送信元の新しい CAT を生成する方法の例です。

```
aws iotsecuretunneling rotate-tunnel-access-token \
```

```
    --region <region> \
    --tunnel-id <tunnel-id> \
    --client-mode SOURCE
```

このコマンドを実行すると、新しい送信元アクセストークンが生成され、トンネルの ARN が返されます。

```
{
    "sourceAccessToken": "<source-access-token>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/<tunnel-id>"
}
```

これで、新しい送信元トークンを使用して、ローカルプロキシを送信元モードで接続できます。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=<source-access-token>
./localproxy -r <region> -s <port>
```

以下は、ローカルプロキシを実行したサンプル出力です。

```
...

[info]    Starting proxy in source mode
...
[info]    Successfully established websocket connection with proxy server ...
[info]    Listening for new connection on port <port>
...
```

リモートデバイスの接続性に関する問題

AWS IoT セキュアトンネリングを使用すると、トンネルが開いている場合でもデバイスが予期せず切断される可能性があります。デバイスがトンネルにまだ接続されているかどうかを確認するには、DescribeTunnel API または describe-tunnel AWS CLIを使用できます。

デバイスは、複数の理由で切断されることがあります。デバイスが以下の考えられる理由によって切断された場合、接続性の問題を解決するには、宛先の CAT をローテーションさせます。

- 宛先の CAT が無効になった。
- トークンがセキュアトンネリングの予約済み MQTT トピックを介してデバイスに配信されなかった。

```
$aws/things/<thing-name>/tunnels/notify
```

次の例は、この問題を解決する方法を示しています。

宛先 CAT のローテーションの例

リモートデバイス *<RemoteThing1>* を考えてみます。このモノへのトンネルを開くには、次のコマンドを使用できます。

```
aws iotsecuretunneling open-tunnel \
    --region <region> \
    --destination-config thingName=<RemoteThing1>,services=SSH
```

このコマンドを実行すると、トンネルの詳細と、送信元と宛先の CAT が生成されます。

```
{
    "sourceAccessToken": "<source-access-token>",
    "destinationAccessToken": "<destination-access-token>",
    "tunnelId": "<tunnel-id>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/tunnel-id"
}
```

ただし、DescribeTunnel API の場合、次の図に示すように、出力はデバイスが切断されたことを示します。

```
aws iotsecuretunneling describe-tunnel \
    --tunnel-id <tunnel-id> \
    --region <region>
```

このコマンドを実行すると、デバイスがまだ接続されていないことが表示されます。

```
{
    "tunnel": {
        ...
        "destinationConnectionState": {
            "status": "DISCONNECTED"
        },
        ...
    }
```

```
}
```

このエラーを解決するには、クライアントを DESTINATION モードにし、宛先の設定を変更した状態で RotateTunnelAccessToken API を実行します。このコマンドを実行すると、古いアクセストークンが取り消され、新しいトークンが生成され、このトークンが MQTT トピックに再送信されます。

$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>
```

このコマンドを実行すると、次に示すように、新しいアクセストークンが生成されます。その後、デバイスエージェントが正しく設定されていれば、トークンはデバイスに配信され、トンネルに接続します。

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

デバイスプロビジョニング

AWS には、デバイスをプロビジョニングし、一意のクライアント証明書をインストールするさまざまな方法が用意されています。このセクションでは、各方法と、IoT ソリューションに最適な方法を選択する方法について説明します。これらのオプションについては、「Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core」(IoTCore における X.509 証明書を使ったデバイス製造およびプロビジョニング) というタイトルのホワイトペーパーで詳しく説明しています。

状況に最も適したオプションを選択してください

- 証明書は、配送前に IoT デバイスにインストールできます。

  エンドユーザーによる使用のために配送される前に IoT デバイスに一意のクライアント証明書を安全にインストールできる場合は、ジャストインタイムプロビジョニング (JITP) またはジャストインタイム登録 (JITR) を使用する必要があります。

  JITP と JITR を使用すると、デバイス証明書の署名に使用される認証機関 (CA) は に登録 AWS IoT され、デバイスが最初に接続 AWS IoT したときに によって認識されます。デバイスは、プロビジョニングテンプレートの詳細を使用して、最初の接続 AWS IoT で にプロビジョニングされます。

  一意の証明書を持つデバイスの単一のモノ、JITP、JITR、およびバルクプロビジョニングの詳細については、「the section called “デバイス証明書があるデバイスのプロビジョニング”」を参照してください。
- エンドユーザーまたはインストーラは、アプリケーションを使用して IoT デバイスに証明書をインストールできます

  エンドユーザーに配信される前に IoT デバイスに一意のクライアント証明書を安全にインストールできないが、エンドユーザーまたはインストーラがアプリケーションを使用してデバイスを登録し、一意のデバイス証明書をインストールできる場合は、信頼できるユーザープロセスによるプロビジョニングを使用する必要があります。

  エンドユーザーや既知のアカウントを持つインストーラなどの信頼できるユーザーを使用すると、デバイスの製造プロセスを簡素化できます。一意のクライアント証明書の代わりに、デバイスには一時的な証明書があり、デバイスが に接続できる AWS IoT 時間は 5 分のみです。この 5 分間で、信頼されたユーザーは有効期限の長い一意のクライアント証明書を取得し、デバイスにインストールします。クレーム証明書には有効期限があるため、証明書のセキュリティが侵害された場合のリスクを最小限は最小限に抑えられます。

詳細については、「the section called “信頼されたユーザーによるプロビジョニング”」を参照してください。

- エンドユーザーは、アプリケーションを使用して IoT デバイスに証明書をインストールすることはできません

  上記のオプションのいずれも IoT ソリューションで機能しない場合は、クレームプロセスによるプロビジョニングを使用します。このプロセスでは、IoT デバイスには、フリート内の他のデバイスによって共有されるクレーム証明書があります。デバイスがクレーム証明書に初めて接続すると、はプロビジョニングテンプレートを使用してデバイスを AWS IoT 登録し、 への後続のアクセスのために一意のクライアント証明書を発行します AWS IoT。

  このオプションを使用すると、デバイスの接続時にデバイスの自動プロビジョニングが可能になりますが AWS IoT、クレーム証明書が侵害された場合、より大きなリスクが生じる可能性があります。クレーム証明書のセキュリティが侵害された場合は、証明書を無効化できます。クレーム証明書を無効化すると、そのクレーム証明書を持つすべてのデバイスが今後登録されなくなります。ただし、クレーム証明書を無効にしても、既にプロビジョニングされているデバイスはブロックされません。

  詳細については、「the section called “クレームによるプロビジョニング”」を参照してください。

でのデバイスのプロビジョニング AWS IoT

を使用してデバイスをプロビジョニングするときは AWS IoT、デバイスと が安全に AWS IoT 通信できるようにリソースを作成する必要があります。デバイスフリートの管理に役立つその他のリソースを作成できます。プロビジョニングプロセスでは、次のリソースを作成できます。

- IoT モノ。

  IoT モノは、 AWS IoT デバイスレジストリのエントリです。各モノには一意の名前と属性のセットがあり、物理デバイスに関連付けられています。モノはモノの種類を使用して定義することも、モノグループにグループ化することもできます。詳細については、「を使用したデバイスの管理 AWS IoT」を参照してください。

  必須ではありませんが、モノを作成することで、モノの種類、モノグループ、およびモノ属性でデバイスを検索し、デバイスフリートをより効率的に管理できます。詳細については、「フリートインデックス作成」を参照してください。

> **Note**
>
> モノの接続ステータスデータのインデックスを作成するには、モノをプロビジョニングし、モノの名前が Connect リクエストで使用されるクライアント ID と一致するように設定します。

- X.509 証明書。

  デバイスは X.509 証明書を使用して相互認証を実行します AWS IoT。既存の証明書を登録するか、 で新しい証明書 AWS IoT を生成して登録できます。証明書を、デバイスを表すモノにアタッチすることで、デバイスと関連付けることができます。また、証明書および関連付けられたプライベートキーをデバイスにコピーする必要もあります。デバイスは接続時に証明書を提示します AWS IoT。詳細については、「認証」を参照してください。

- IoT ポリシー。

  IoT ポリシーは、デバイスが AWS IoTで実行できるオペレーションを定義します。IoT ポリシーはデバイス証明書にアタッチされます。デバイスが証明書を に提示すると AWS IoT、ポリシーで指定されたアクセス許可が付与されます。詳細については、「Authorization」を参照してください。各デバイスには、 AWS IoTと通信する証明書が必要です。

AWS IoT は、プロビジョニングテンプレートを使用したフリートの自動プロビジョニングをサポートしています。プロビジョニングテンプレートは、デバイスをプロビジョニング AWS IoT するために必要なリソースを記述します。テンプレートには、1 つのテンプレートを使用して複数のデバイスをプロビジョニングできる変数が含まれています。デバイスをプロビジョニングするときは、ディクショナリまたはマップを使用して、デバイスに固有の変数の値を指定します。別のデバイスをプロビジョニングするには、ディクショナリに新しい値を指定します。

デバイスに固有の証明書 (および関連するプライベートキー) があるかどうかにかかわらず、自動プロビジョニングを使用できます。

フリートプロビジョニング API

フリートプロビジョニングで使用されるAPIには、いくつかのカテゴリがあります。

- これらのコントロールプレーン機能は、フリートのプロビジョニングテンプレートを作成および管理し、信頼できるユーザーポリシーを設定します。
  - CreateProvisioningTemplate

  - CreateProvisioningTemplateVersion
  - DeleteProvisioningTemplate
  - DeleteProvisioningTemplateVersion
  - DescribeProvisioningTemplate
  - DescribeProvisioningTemplateVersion
  - ListProvisioningTemplates
  - ListProvisioningTemplateVersions
  - UpdateProvisioningTemplate
- 信頼されたユーザーは、このコントロールプレーン機能を使用して、一時的なオンボーディング要求を生成できます。この一時的なクレームは、Wi-Fi 設定または同様の方法を通じてデバイスに渡されます。
  - CreateProvisioningClaim
- プロビジョニング処理中に使用される MQTT API。プロビジョニング要求証明書が埋め込まれているデバイスによって使用され、信頼されたユーザーによってそのデバイスに渡されます。
  - the section called “CreateCertificateFromCsr”
  - the section called “CreateKeysAndCertificate”
  - the section called “RegisterThing”

フリートプロビジョニングを使用したデバイス証明書がないデバイスのプロビジョニング

AWS IoT フリートプロビジョニングを使用すると、 は、 に AWS IoT 初めて接続するときにデバイス証明書とプライベートキーを生成して安全にデバイスに配信 AWS IoT できます。 は、Amazon ルート認証局 (CA) によって署名されたクライアント証明書 AWS IoT を提供します。

フリートプロビジョニングを使用するには、次の 2 つの方法があります。

- クレームによるプロビジョニング
- 信頼されたユーザーによるプロビジョニング

クレームによるプロビジョニング

デバイスは、プロビジョニングクレーム証明書とプライベートキー（特別な目的の認証情報）が埋め込まれた状態で製造できます。これらの証明書が に登録されている場合 AWS IoT、サービスはそれらの証明書を、デバイスが通常のオペレーションに使用できる一意のデバイス証明書と交換できます。このプロセスには、以下のステップが含まれます。

デバイスを配送する前に

1. CreateProvisioningTemplate を呼び出して、プロビジョニングテンプレートを作成します。この API はテンプレート ARN を返します。詳細については、「デバイスプロビジョニング MQTT API」を参照してください。

   AWS IoT コンソールでフリートプロビジョニングテンプレートを作成することもできます。

   a. ナビゲーションペインから、[多数のデバイスを接続] ドロップダウンを選択します。次に、[複数のデバイスを接続] を選択します。
   b. [プロビジョニングテンプレートを作成] を選択します。
   c. [プロビジョニングシナリオ] で、インストールプロセスに最適なものを選択します。その後、[Next] を選択します。
   d. テンプレートワークフローを完了します。
2. プロビジョニングクレーム証明書として使用する証明書および関連付けられたプライベートキーを作成します。
3. これらの証明書を に登録 AWS IoT し、証明書の使用を制限する IoT ポリシーを関連付けます。次の IoT ポリシーの例では、このポリシーに関連付けられた証明書の使用をプロビジョニングデバイスに制限しています。

```
{
    "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. デバイスのプロビジョニング時に、アカウントのモノや証明書などの IoT リソースを作成または更新するアクセス許可を AWS IoT サービスに付与します。これを行うには、 AWS IoT サービスプリンシパルを信頼する IAM ロール (プロビジョニングロールと呼ばれます) に AWSIoTThingsRegistrationマネージドポリシーをアタッチします。
5. プロビジョニングクレーム証明書が安全に埋め込まれたデバイスを製造します。

これで、デバイスを設置して使用する場所に配送する準備ができました。

> **Important**
>
> プロビジョニングクレームプライベートキーは、デバイス上を含め、常に保護する必要があります。 AWS IoT CloudWatch メトリクスとログを使用して、誤用の兆候をモニタリングすることをお勧めします。誤用を検出した場合は、プロビジョニングクレーム証明書を無効にして、デバイスのプロビジョニングに使用できないようにします。

デバイスを使用できるように初期化するには

1. デバイスは を使用して AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Clientに接続し、デバイスにインストールされているプロビジョニングクレーム証明書 AWS IoT を使用して で認証します。

   > **Note**
   >
   > セキュリティ上の理由から、CreateCertificateFromCsr および CreateKeysAndCertificate によって返される certificateOwnershipToken は 1 時間後に有効期限切れになります。certificateOwnershipToken が有効期限切れになる前に、RegisterThing を呼び出す必要があります。CreateCertificateFromCsr または CreateKeysAndCertificate によって作成された証明書がトークンの有効期限が切れるまでにアクティベートされず、ポリシーまたはモノにアタッチされない場合、証明書は削除されます。トークンの有効期限が切れた場合、デバイスは CreateCertificateFromCsr または CreateKeysAndCertificate を呼び出して新しい証明書を生成します。

2. デバイスは、以下のオプションのいずれかを使用して、永続的な証明書とプライベートキーを取得します。デバイスは、今後のすべての認証に証明書とキーを使用します AWS IoT。

   a. を呼び出しCreateKeysAndCertificateて、認証機関を使用して新しい AWS 証明書とプライベートキーを作成します。

      または

   b. CreateCertificateFromCsr を呼び出して、プライベートキーを安全に保つ証明書署名リクエストから証明書を生成します。

3. デバイスから RegisterThing を呼び出してデバイスを AWS IoT に登録し、クラウドリソースを作成します。

   フリートプロビジョニングサービスでは、プロビジョニングテンプレートを使用して、IoT のモノなどのクラウドリソースを定義および作成します。このテンプレートでは、モノが属する属性とグループを指定できます。新しいモノを追加するには、モノのグループが存在している必要があります。

4. デバイスに永続的な証明書を保存した後、デバイスはプロビジョニングクレーム証明書を使用して開始したセッションから切断し、永続的な証明書を使用して再接続する必要があります。

これで、デバイスは正常に通信する準備が整いました AWS IoT。

信頼されたユーザーによるプロビジョニング

多くの場合、エンドユーザーやインストール技術者などの信頼されたユーザーがモバイルアプリを使用してデプロイされた場所にデバイスを設定すると、デバイスは AWS IoT 初めて に接続します。

> **Important**
>
> この手順を実行するには、信頼されたユーザーのアクセスとアクセス許可を管理する必要があります。そのための 1 つの方法は、信頼されたユーザーに対して、これらのユーザーを認証し、この手順の実行に必要な AWS IoT 機能および API オペレーションへのアクセスを許可するアカウントを提供して維持することです。

デバイスを配送する前に

1. CreateProvisioningTemplate を呼び出してプロビジョニングテンプレートを作成し、その *templateArn* および *templateName* を返します。
2. 信頼されたユーザーがプロビジョニングプロセスを開始するために使用する IAM ロールを作成します。プロビジョニングテンプレートでは、そのユーザーだけがデバイスをプロビジョニングできます。例えば、次のようになります。

   ```
   {
       "Effect": "Allow",
       "Action": [
           "iot:CreateProvisioningClaim"
       ],
       "Resource": [
           "arn:aws:iot:aws-region:aws-account-id:provisioningtemplate/templateName"
       ]
   }
   ```

3. デバイスのプロビジョニング時に、アカウントのモノや証明書などの IoT リソースを作成または更新するアクセス許可を AWS IoT サービスに付与します。これを行うには、 AWS IoT サービスプリンシパルを信頼する IAM ロール (プロビジョニングロールと呼ばれる) に AWSIoTThingsRegistration管理ポリシーをアタッチします。

4. 信頼されたユーザーを識別する手段を提供します。例えば、これらのユーザーを認証し、デバイスの登録に必要な AWS API オペレーションとのやり取りを許可できるアカウントを提供します。

デバイスを使用できるように初期化するには

1. 信頼されたユーザーは、プロビジョニングモバイルアプリまたはウェブサービスにサインインします。
2. モバイルアプリケーションまたはウェブアプリケーションは、IAM ロールを使用し、CreateProvisioningClaim を呼び出して、 AWS IoTから一時的なプロビジョニングクレーム証明書を取得します。

   > **Note**
   >
   > セキュリティ上の理由から、CreateProvisioningClaim から返される一時的なプロビジョニングクレーム証明書は 5 分後に有効期限切れになります。以下の手順では、一時的なプロビジョニングクレーム証明書の有効期限が切れる前に、有効な証明書を正常に返す必要があります。一時的なプロビジョニングクレーム証明書はアカウントの証明書のリストには表示されません。

3. モバイルアプリまたはウェブアプリケーションは、Wi-Fi 認証情報などの必要な設定情報と共に、一時的なプロビジョニングクレーム証明書をデバイスに提供します。
4. デバイスは、一時的なプロビジョニングクレーム証明書と AWS IoT を使用して AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Client に接続します。
5. デバイスは、一時的なプロビジョニングクレーム証明書 AWS IoT を使用して に接続してから 5 分以内に、これらのオプションのいずれかを使用して永続的な証明書とプライベートキーを取得します。デバイスは証明書を使用し、これらのオプションが返すキーを今後のすべての認証に使用します AWS IoT。

   a. を呼び出しCreateKeysAndCertificateて、認証機関を使用して新しい AWS 証明書とプライベートキーを作成します。

      または

   b. CreateCertificateFromCsr を呼び出して、プライベートキーを安全に保つ証明書署名リクエストから証明書を生成します。

> **Note**
>
> CreateKeysAndCertificate または は、一時的なプロビジョニングクレーム証明書 AWS IoT を使用して に接続してから 5 分以内に有効な証明書を返すCreateCertificateFromCsr必要があります。

6. デバイスは RegisterThingを呼び出してデバイスを に登録 AWS IoT し、クラウドリソースを作成します。

   フリートプロビジョニングサービスでは、プロビジョニングテンプレートを使用して、IoT のモノなどのクラウドリソースを定義および作成します。このテンプレートでは、モノが属する属性とグループを指定できます。新しいモノを追加するには、モノのグループが存在している必要があります。

7. 永続的な証明書をデバイスに保存した後、デバイスは一時的なプロビジョニングクレーム証明書を使用して開始したセッションから切断し、永続的な証明書を使用して再接続する必要があります。

これで、デバイスは正常に通信する準備が整いました AWS IoT。

CLI AWS での事前プロビジョニングフックの使用

次の手順では、事前プロビジョニングフックを使用してプロビジョニングテンプレートを作成します。ここで使用されている Lambda 関数の例は変更可能です。

事前プロビジョニングフックを作成してプロビジョニングテンプレートに適用するには

1. 定義された入力と出力を持つ Lambda 関数を作成します。Lambda 関数は高度にカスタマイズ可能であり、`allowProvisioning` および `parameterOverrides` は事前プロビジョニングフックを作成するために必要です。Lambda 関数の作成の詳細については、AWS 「 コマンドラインインターフェイス AWS Lambda で を使用する」を参照してください。

   Lambda 関数出力の例を次に示します。

```
{
  "allowProvisioning": True,
  "parameterOverrides": {
    "incomingKey0": "incomingValue0",
```

```
    "incomingKey1": "incomingValue1"
  }
}
```

2. AWS IoT はリソースベースのポリシーを使用して Lambda を呼び出すため、Lambda 関数を呼び出すアクセス許可を付与 AWS IoT する必要があります。

> **Important**
>
> Lambda アクションにアタッチされたポリシーのグローバル条件コンテキストキーに `source-arn` または `source-account` を必ず含め、アクセス許可の操作を防止します。詳細については、「サービス間での不分別な代理処理の防止」を参照してください。

以下は、add-permission を使用して Lambda に IoT アクセス許可を付与する例です。

```
aws lambda add-permission \
    --function-name myLambdaFunction \
    --statement-id iot-permission \
    --action lambda:InvokeFunction \
    --principal iot.amazonaws.com
```

3. create-provisioning-template または update-provisioning-template コマンドを使用して、テンプレートに事前プロビジョニングフックを追加します。

次の CLI 例では、create-provisioning-template を使用して、事前プロビジョニングフックを持つプロビジョニングテンプレートを作成します。

```
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
```

このコマンドの出力は以下のようになります。

```
{
    "templateArn": "arn:aws:iot:us-east-1:1234564789012:provisioningtemplate/
myTemplate",
```

```
    "defaultVersionId": 1,
    "templateName": myTemplate
}
```

また、パラメータをすべてコマンドラインパラメータ値として入力する代わりに、ファイルからロードして、時間を節約することもできます。詳細については、「ファイルから AWS CLI パラメータをロードする」を参照してください。次に、拡張された JSON 形式の template パラメータを示します。

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

次に、拡張された JSON 形式の pre-provisioning-hook パラメータを示します。

```
{
     "targetArn" : "arn:aws:lambda:us-
east-1:765219403047:function:pre_provisioning_test",
     "payloadVersion" : "2020-04-01"
}
```

デバイス証明書があるデバイスのプロビジョニング

AWS IoT には、デバイスにデバイス証明書 (および関連するプライベートキー) が既にある場合にデバイスをプロビジョニングする 3 つの方法があります。

- プロビジョニングテンプレートを使用した単一のモノのプロビジョニング。このオプションは、一度に 1 つずつデバイスをプロビジョニングするだけの場合に適しています。
- デバイスを最初に接続するときにプロビジョニングするテンプレートを使用したJust-in-timeプロビジョニング (JITP) AWS IoT。このオプションは、大量のデバイスを登録する必要があるが、それらのデバイスに関して一括プロビジョニングリストにまとめることができる情報がない場合に適しています。
- 一括登録。このオプションを使用すると、S3 バケットのファイルに保存された単一のモノのプロビジョニングテンプレートの値のリストを指定できます。このアプローチは、目的の特性をリストにまとめることができる、既知のデバイスが大量にある場合に適しています。

トピック

- 単一のモノプロビジョニング
- ジャストインタイムプロビジョニング
- 一括登録

単一のモノプロビジョニング

モノをプロビジョニングするには、RegisterThing API または `register-thing` CLI コマンドを使用します。`register-thing` CLI コマンドは次の引数をとります。

--template-body

プロビジョニングテンプレート。

--parameters

プロビジョニングテンプレートで使用される、JSON 形式のパラメータの名前/値のペアのリスト (例: `{"ThingName" : "MyProvisionedThing", "CSR" : "csr-text"}`)。

「プロビジョニングテンプレート」を参照してください。

RegisterThing または `register-thing` は、作成した証明書のリソースとテキストの ARN を返します。

```
{
    "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"
    }
}
```

ディクショナリからパラメータを省略すると、デフォルト値が使用されます。デフォルト値が指定されていない場合、パラメータは値に置き換えられません。

ジャストインタイムプロビジョニング

ジャストインタイムプロビジョニング (JITP) を使用して、デバイスが最初に AWS IoTに接続しようとするときにデバイスをプロビジョニングできます。デバイスをプロビジョニングするには、自動登録を有効にして、プロビジョニングテンプレートを、デバイス証明書に署名するために使用される CA 証明書に関連付ける必要があります。プロビジョニングの成功とエラーは、Amazon CloudWatch の デバイスプロビジョニングのメトリクス と同様にログに記録されます。

トピック

- JITP の概要
- プロビジョニングテンプレートを使用して CA を登録する
- プロビジョニングテンプレート名を使用して CA を登録する

JITP の概要

登録された CA 証明書によって署名された証明書 AWS IoT を使用してデバイスが に接続しようとすると、 は CA 証明書からテンプレートを AWS IoT ロードし、それを使用して RegisterThing を呼び出します。JITP ワークフローは、最初に `PENDING_ACTIVATION` というステータス値で証明書を登録します。デバイスのプロビジョニングフローが完了すると、証明書のステータスは `ACTIVE` に変わります。

AWS IoT は、プロビジョニングテンプレートで宣言および参照できる以下のパラメータを定義します。

- `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`

これらのプロビジョニングテンプレートパラメータの値は、プロビジョニング対象のデバイスの証明書の件名フィールドから JITP が抽出できるものに限られます。証明書には、テンプレート本体のすべてのパラメータの値が含まれている必要があります。`AWS::IoT::Certificate::Id` パラメータは、証明書に含まれている ID ではなく、内部で生成された ID を参照します。この ID の値は、AWS IoT ルール内の `principal()`関数を使用して取得できます。

> **Note**
>
> デバイスの最初の接続で信頼チェーン全体を送信しなくても、 AWS IoT Core just-in-time プロビジョニング (JITP) 機能を使用してデバイスをプロビジョニングできます AWS IoT Core。CA 証明書の提示はオプションですが、デバイスが AWS IoT Coreに接続するときに [Server Name Indication (SNI)] (サーバーネームインディケーション (SNI)) エクステンションを送信する必要があります。

テンプレート本文の例

次の JSON ファイルは、完全な JITP テンプレートのテンプレート本文の例です。

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

このサンプルテンプレートでは、証明書から抽出され、AWS::IoT::Certificate::CommonName セクションで使用されている AWS::IoT::Certificate::SerialNumber、AWS::IoT::Certificate::Country、AWS::IoT::Ce および Resources プロビジョニングパラメータの値を宣言します。次に、JITP ワークフローはこのテンプレートを使用して次のアクションを実行します。

- 証明書を登録し、そのステータスを PENDING_ACTIVE に設定します。
- 1 つのモノのリソースを作成します。
- 1 つのポリシーのリソースを作成します。
- ポリシーを証明書にアタッチします。
- 証明書をモノにアタッチします。
- 証明書のステータスを ACTIVE に更新します。

templateBody の Parameters セクションに記載されているすべてのプロパティが証明書に含まれていない場合、デバイスのプロビジョニングは失敗します。例えば、AWS::IoT::Certificate::Country がテンプレートに含まれていても、証明書に Country プロパティがない場合、デバイスのプロビジョニングは失敗します。

CloudTrail を使用して、JITP テンプレートに関する問題をトラブルシューティングすることもできます。Amazon CloudWatch にログ記録されるメトリクスについては、「デバイスプロビジョニングのメトリクス」を参照してください。プロビジョニングテンプレートの詳細については、「プロビジョニングテンプレート」を参照してください。

> **Note**
>
> プロビジョニングプロセス中に、just-in-timeプロビジョニング (JITP) は他の AWS IoT コントロールプレーン API オペレーションを呼び出します。これらの呼び出しは、アカウントに設定されたAWS IoT スロットリングクォータを超過し、スロットリングされた呼び出しが発生する可能性があります。必要に応じて、AWS カスタマーサポートに連絡して、スロットリングのクォータを引き上げてください。

プロビジョニングテンプレートを使用して CA を登録する

完全なプロビジョニングテンプレートを使用して CA を登録するには、次の手順に従います。

1. 次の例に示すようなプロビジョニングテンプレートとロール ARN 情報を JSON ファイルとして保存します。

```
{
     "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"
}
```

この例で、templateBody フィールドの値は、エスケープ文字列として指定された JSON オブジェクトである必要があり、前のリストに示した値のみを使用できます。json.dumps (Python) や JSON.stringify（ノード）など、必要な JSON 出力を作成するには、さまざまなツールを使用できます。roleARN フィールドの値は、AWSIoTThingsRegistration がアタッチされたロールの ARN である必要があります。また、テンプレートでは例のインライン PolicyName の代わりに、既存の PolicyDocument を使用できます

2. RegisterCACertificate API オペレーションまたは register-ca-certificate CLI コマンドを使用して CA 証明書を登録します。前のステップで保存したプロビジョニングテンプレートとロール ARN 情報のディレクトリを指定します。

   以下に、 AWS CLIを使用して DEFAULT モードで CA 証明書を登録する方法の例を示します。

   ```
   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
   ```

   以下に、 AWS CLIを使用して SNI_ONLY モードで CA 証明書を登録する方法の例を示します。

   ```
   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
   ```

   詳細については、「Register your CA Certificates」(CA 証明書の登録) を参照してください。

3. (オプション) UpdateCACertificate API オペレーションまたは update-ca-certificate CLI コマンドを使用して、CA 証明書の設定を更新します。

   AWS CLIを使用して CA 証明書を更新する方法の例を次に示します。

   ```
   aws iot update-ca-certificate --certificate-id caCertificateId
   ```

```
                --new-auto-registration-status ENABLE --registration-config
 file://your-template
```

プロビジョニングテンプレート名を使用して CA を登録する

プロビジョニングテンプレート名を使用して CA を登録するには、次の手順に従います。

1. プロビジョニングテンプレート本文を JSON ファイルとして保存します。テンプレート本文の例は「テンプレート本文の例」にあります。
2. プロビジョニングテンプレートを作成するには、CreateProvisioningTemplate API または create-provisioning-templateCLI コマンドを使用します。

```
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**
>
> ジャストインタイムプロビジョニング (JITP) の場合は、プロビジョニングテンプレートを作成するときにテンプレートタイプとして JITP を指定する必要があります。テンプレートタイプの詳細については、「AWS API リファレンス」の「CreateProvisioningTemplate」を参照してください。

3. CA をテンプレート名で登録するには、RegisterCACertificate API または register-ca-certificate CLI コマンドを使用します。

```
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
```

一括登録

start-thing-registration-task コマンドを使用して、モノを一括で登録できます。このコマンドは、プロビジョニングテンプレート、S3 バケット名、キー名、ロール ARN (S3 バケット内のファイルへのアクセスを許可する) を使用します。S3 バケットのファイルには、テンプレート内のパラメータを置き換えるために使用される値が含まれています。このファイルは、改行で区切られた

JSON ファイルでなければなりません。各行には、単一のデバイスを登録するためのすべてのパラメータ値が含まれています。例:

```
{"ThingName": "foo", "SerialNumber": "123", "CSR": "csr1"}
{"ThingName": "bar", "SerialNumber": "456", "CSR": "csr2"}
```

以下の一括登録関連の API オペレーションが役立つ場合があります。

- ListThingRegistrationTasks: 現在のモノの一括プロビジョニングタスクを一覧表示します。
- DescribeThingRegistrationTask: 特定のモノの一括登録タスクに関する情報を提供します。
- StopThingRegistrationTask: 一括登録タスクを停止します。
- ListThingRegistrationTaskReports: モノの一括登録タスクの結果、失敗、またはその両方を確認するために使用されます。

> **Note**
>
> - 一度に実行できる一括登録オペレーションタスクは、(アカウントごとに) 1 つだけです。
> - 一括登録オペレーションは、他の AWS IoT コントロールプレーン API オペレーションを呼び出します。これらの呼び出しは、アカウントの AWS IoT スロットリングクォータを超過し、スロットルエラーが発生する可能性があります。必要に応じてAWS 、カスタマーサポートに連絡して AWS IoT スロットリングクォータを引き上げます。

プロビジョニングテンプレート

プロビジョニングテンプレートは、 パラメータを使用して、デバイスが操作するために使用する必要があるリソースを記述する JSON ドキュメントです AWS IoT。プロビジョニングテンプレートには `Parameters` と `Resources` の 2 つのセクションがあります。には 2 種類のプロビジョニングテンプレートがあります AWS IoT。1 つはジャストインタイムプロビジョニング (JITP) および一括登録に使用し、もう 1 つはフリートプロビジョニングに使用します。

トピック

- Parameters セクション
- Resources セクション

- 一括登録のテンプレート例
- ジャストインタイムプロビジョニング (JITP) のテンプレート例
- フリートプロビジョニング

Parameters セクション

Parameters セクションでは、Resources セクションで使用されるパラメータを宣言します。各パラメータは、名前、タイプ、およびオプションのデフォルト値を宣言します。デフォルト値は、テンプレートで渡されたディクショナリにパラメータの値が含まれていない場合に使用されます。テンプレートドキュメントの Parameters セクションは、次のようになります。

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

このテンプレート本文のスニペットでは、4 つのパラメータ (ThingName、SerialNumber、Location、CSR) を宣言します。これらのすべてのパラメータタイプは String です。Location パラメータは、デフォルト値 "WA" を宣言します。

Resources セクション

テンプレート本文の Resourcesセクションは、デバイスが通信するために必要なリソースを宣言します AWS IoT。モノ、証明書、および 1 つ以上の IoT ポリシーです。各リソースは、論理名、タイプ、および一連のプロパティを指定します。

論理名を使用すると、テンプレートの別の場所でリソースを参照できます。

タイプは、宣言するリソースのタイプを指定します。有効なタイプは次のとおりです。

- `AWS::IoT::Thing`
- `AWS::IoT::Certificate`
- `AWS::IoT::Policy`

指定するプロパティは、宣言するリソースのタイプによって異なります。

モノのリソース

モノのリソースは、次のプロパティを使用して宣言されます。

- `ThingName`: 文字列。
- `AttributePayload`: オプション。名前と値のペアのリスト。
- `ThingTypeName`: オプション。モノに関連するモノのタイプ型の文字列。
- `ThingGroups`: オプション。モノが属するグループのリスト。
- `BillingGroup`: オプション。関連する請求グループ名の文字列。
- `PackageVersions`: オプション。関連するパッケージとバージョン名の文字列。

証明書リソース

証明書は、次のいずれかの方法で指定できます。

- 証明書署名リクエスト (CSR)。
- 既存のデバイス証明書の証明書 ID。(フリートプロビジョニングテンプレートで使用できるのは証明書 ID のみです)。
- AWS IoTで登録された CA 証明書で作成されたデバイス証明書。同じ件名フィールドに複数の CA 証明書が登録されている場合は、デバイス証明書の署名に使用された CA 証明書も渡す必要があります。

> **Note**
>
> テンプレートで証明書を宣言する場合は、これらのいずれかの方法のみを使用してください。たとえば、CSR を使用する場合は、証明書 ID またはデバイス証明書を指定することもできません。詳細については、「X.509 クライアント証明書」を参照してください。

詳細については、「X.509 証明書の概要」を参照してください。

証明書リソースは、次のプロパティを使用して宣言されます。

- CertificateSigningRequest: 文字列。
- CertificateId: 文字列。
- CertificatePem: 文字列。
- CACertificatePem: 文字列。
- Status: オプション。ACTIVE または INACTIVE を指定できる文字列。デフォルトは ACTIVE です。
- ThingPrincipalType: オプション。モノとプリンシパル (証明書) の関係のタイプを指定する文字列。
  - EXCLUSIVE_THING: 排他的な関係を確立します。プリンシパルは、この特定のモノにのみアタッチでき、これ以外のモノにはアタッチできません。
  - NON_EXCLUSIVE_THING: 指定されたプリンシパルをモノにアタッチします。プリンシパルに複数のモノをアタッチできます。値が指定されていない場合は、これがデフォルト値です。

> **Note**
>
> デバイス証明書なしでデバイスをプロビジョニングすることもできます。詳細については、「フリートプロビジョニングを使用したデバイス証明書がないデバイスのプロビジョニング」を参照してください。

例:

- CSR で指定された証明書:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateSigningRequest": {"Ref" : "CSR"},
            "Status" : "ACTIVE"
        }
    }
}
```

- 既存の証明書 ID で指定された証明書:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateId": {"Ref" : "CertificateId"}
        }
    }
}
```

- 既存の証明書 .pem および CA 証明書 .pem で指定された証明書:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CACertificatePem": {"Ref" : "CACertificatePem"},
            "CertificatePem": {"Ref" : "CertificatePem"}
        }
    }
}
```

- プリンシパルに 1 つのモノを排他的にアタッチします。

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "ThingPrincipalType" : "EXCLUSIVE_THING"
        }
    }
}
```

ポリシーリソース

ポリシーリソースは、以下のいずれかのプロパティを使用して宣言されます。

- PolicyName: オプション。文字列。デフォルトはポリシードキュメントのハッシュです。PolicyName は AWS IoT ポリシーのみ参照可能で、IAM ポリシーは参照可能ではありませ

ん。既存の AWS IoT ポリシーを使用している場合は、PolicyName プロパティに、ポリシーの名前を入力します。PolicyDocument プロパティを含めないでください。

- PolicyDocument: オプション。エスケープした文字列として指定された JSON オブジェクト。PolicyDocument が指定されていない場合は、ポリシーを作成しておく必要があります。

> **Note**
>
> Policy セクションが存在する場合、PolicyName または PolicyDocument を指定する必要がありますが、両方を指定することはできません。

上書き設定

テンプレートに既に存在するリソースが指定されている場合、OverrideSettings セクションでは、実行するアクションを指定できます。

DO_NOTHING

リソースはそのままにしておきます。

REPLACE

リソースをテンプレートで指定されたリソースに置き換えます。

FAIL

リクエストが ResourceConflictsExceptionで失敗します。

MERGE

ThingGroups の AttributePayload および thing プロパティにのみ有効です。モノの既存の属性またはグループメンバーシップを、テンプレートで指定された属性またはグループメンバーシップとマージします。

モノのリソースを宣言する場合は、次のプロパティに OverrideSettings を指定できます。

- ATTRIBUTE_PAYLOAD
- THING_TYPE_NAME
- THING_GROUPS

モノの証明書リソースを宣言する場合は、OverrideSettings プロパティに Status を指定できます。

OverrideSettings をこのポリシーリソースに使用することはできません。

リソースの例

次のテンプレートスニペットでは、モノ、証明書、およびポリシーを宣言します。

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

モノは、次のもので宣言されます。

- 論理名 "thing"。
- 型 AWS::IoT::Thing。
- モノのプロパティのセット。

  モノのプロパティには、モノの名前、属性セット、オプションのモノのタイプ名、モノが属するモノのグループのオプションのリストが含まれます。

パラメータは、{"Ref":"*parameter-name*"} によって参照されます。テンプレートが評価されると、パラメータは、テンプレートと共に渡されたディクショナリのパラメータの値に置き換えられます。

証明書は、次のもので宣言されます。

- 論理名 "certificate"。
- 型 AWS::IoT::Certificate。
- プロパティのセット。

  プロパティには証明書の CSR を含めて、ステータスを ACTIVE に設定します。CSR テキストは、テンプレートと共に渡されたディクショナリのパラメータとして渡されます。

ポリシーは、次のもので宣言されます。

- 論理名 "policy"。
- 型 AWS::IoT::Policy。
- 既存のポリシー名またはポリシードキュメントの名前。

一括登録のテンプレート例

以下の JSON ファイルは、CSR で証明書を指定する完全なプロビジョニングテンプレートの例です。

(PolicyDocument フィールドの値は、エスケープ文字列として指定された JSON オブジェクトである必要があります。)

```
{
```

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

```
}
```

ジャストインタイムプロビジョニング (JITP) のテンプレート例

以下の JSON ファイルは、証明書 ID で既存の証明書を指定する完全なプロビジョニングテンプレートの例です。

```
{
   "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 プロビジョニング用のテンプレートでは CertificateId を使用する必要があります。

プロビジョニングテンプレートのタイプの詳細については、CreateProvisioningTemplate AWS API リファレンスの「」を参照してください。

このテンプレートをジャストインタイムプロビジョニングに使用する方法の詳細については、「ジャストインタイムプロビジョニング」を参照してください。

フリートプロビジョニング

フリートプロビジョニングテンプレートは、クラウドおよびデバイス設定をセットアップ AWS IoT するために によって使用されます。これらのテンプレートは、JITP テンプレートおよび一括登録

テンプレートと同じパラメータおよびリソースを使用します。詳細については、「プロビジョニングテンプレート」を参照してください。フリートプロビジョニングテンプレートには、Mapping セクションと DeviceConfiguration セクションを含めることができます。フリートプロビジョニングテンプレート内で組み込み関数を使用して、デバイス固有の設定を生成できます。フリートプロビジョニングテンプレートは名前付きリソースで、ARN によって識別されます (例: `arn:aws:iot:us-west-2:1234568788:provisioningtemplate/templateName`)。

Mappings

任意の Mappings セクションでは、キーと名前付きの一連の値とが対応付けられます。たとえば、リージョンに基づいて AWS 値を設定する場合は、その AWS リージョン 名前をキーとして使用し、特定のリージョンごとに指定する値を含むマッピングを作成できます。マップ内の値を取得するには、Fn::FindInMap 組み込み関数を使用します。

Mappings セクションにパラメータ、擬似パラメータを含めること、または組み込み関数を呼び出すことはできません

デバイス設定

デバイス設定セクションには、プロビジョニング時にデバイスに送信する任意のデータが含まれています。例えば、次のようになります。

```
{
    "DeviceConfiguration": {
        "Foo":"Bar"
    }
}
```

JavaScript Object Notation (JSON) ペイロード形式を使用してデバイスにメッセージを送信する場合、 はこのデータを JSON として AWS IoT Core フォーマットします。Concise Binary Object Representation (CBOR) ペイロード形式を使用している場合、 AWS IoT Core はこのデータを CBOR としてフォーマットします。DeviceConfiguration セクションは、ネストされた JSON オブジェクトをサポートしていません。

組み込み関数

組み込み関数は、Mappings セクションを除くプロビジョニングテンプレートの任意のセクションで使用されます。

`Fn::Join`

一連の値を特定の区切り文字で区切って 1 つの値に追加します。区切り文字が空の文字列の場合、値は区切り文字を使用することなく連結されます。

> **Important**
>
> `Fn::Join` は the section called “ポリシーリソース” に対してサポートされていません。

`Fn::Select`

インデックスによってオブジェクトのリストから単一のオブジェクトを返します。

> **Important**
>
> `Fn::Select` では、null 値のチェックや、インデックスが配列の範囲外であるかどうかのチェックは行われません。どちらの条件もプロビジョニングエラーになるため、有効なインデックス値を選択し、リストに null 以外が含まれていることを確認してください。

`Fn::FindInMap`

`Mappings` セクションで宣言された 2 つのレベルのマッピングのキーに対応する値を返します。

`Fn::Split`

文字列を文字列値のリストに分割して、文字列リストから要素を選択できるようにします。文字列の分割位置を決定する区切り文字 (カンマなど) を指定します。文字列を分割した後、`Fn::Select` を使用して要素を選択します。

たとえば、サブネット ID のカンマ区切りの文字列がスタックテンプレートにインポートされる場合は、各カンマで文字列を分割できます。サブネット ID のリストから、`Fn::Select` を使用してリソースのサブネット ID を指定します。

`Fn::Sub`

特定した値の入力文字列にある変数の代わりになります。スタックを作成または更新するまで使用できない値を含むコマンドまたは出力を作成するために、この関数を使用できます。

フリートプロビジョニングのテンプレート例

```
{
    "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**
>
> 既存のプロビジョニングテンプレートを更新して、事前プロビジョニングフックを追加できます。

事前プロビジョニングフック

AWS では、プロビジョニングテンプレートを作成するときに事前プロビジョニングフック関数を使用することをお勧めします。これにより、アカウントがオンボードするデバイスとデバイスの数をより詳細に制御できます。事前プロビジョニングフックは、デバイスのプロビジョニングを許可する前に、デバイスから渡されたパラメータを検証する Lambda 関数です。この Lambda 関数は、デバイ

スが the section called “RegisterThing” を介してリクエストを送信するたびに呼び出されるため、デバイスをプロビジョニングする前にアカウントに存在する必要があります。

> **Important**
>
> Lambda アクションにアタッチされたポリシーのグローバル条件コンテキストキーに `source-arn` または `source-account` を必ず含め、アクセス許可の操作を防止します。詳細については、「サービス間での不分別な代理処理の防止」を参照してください。

デバイスをプロビジョニングするには、Lambda 関数が入力オブジェクトを受け入れ、このセクションで説明する出力オブジェクトを返す必要があります。プロビジョニングは、Lambda 関数が `"allowProvisioning": True` のオブジェクトを返す場合にのみ続行されます。

事前プロビジョニングフックの入力

AWS IoT デバイスが に登録されると、 はこのオブジェクトを Lambda 関数に送信します 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 関数に渡される `parameters` オブジェクトには、the section called “RegisterThing” リクエストペイロードで渡される `parameters` 引数のプロパティが含まれています。

事前プロビジョニングフックの戻り値

この Lambda 関数は、プロビジョニングリクエスト、およびオーバーライドするプロパティの値を承認したかどうかを示す応答を返す必要があります。

次に、事前プロビジョニング機能からの正常な応答の例を示します。

```
{
    "allowProvisioning": true,
    "parameterOverrides" : {
        "Key": "newCustomValue",
        ...
    }
}
```

"parameterOverrides" 値は、the section called “RegisterThing” リクエストペイロードの "parameters" パラメータに追加されます。

> **Note**
>
> - Lambda 関数が失敗すると、プロビジョニングリクエストは ACCESS_DENIED で失敗し、エラーが CloudWatch Logs に記録されます。
> - Lambda 関数が応答で "allowProvisioning": "true" を返さない場合、プロビジョニング要求は ACCESS_DENIED で失敗します。
> - Lambda 関数の実行が終了し、5 秒以内に戻る必要があります。そうでない場合は、プロビジョニングリクエストは失敗します。

事前プロビジョニングフック Lambda の例

Python

Python での事前プロビジョニングフック Lambda の例。

```
import json

def pre_provisioning_hook(event, context):
    print(event)

    return {
        'allowProvisioning': True,
        'parameterOverrides': {
            'DeviceLocation': 'Seattle'
        }
    }
```

Java

Java での事前プロビジョニングフック Lambda の例。

ハンドラークラス:

```
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;
    }

}
```

リクエストクラス:

```
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;
}
```

Response クラス:

```
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

JavaScript での事前プロビジョニングフック Lambda の例。

```
exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    var reply = {
        allowProvisioning: true,
        parameterOverrides: {
            DeviceLocation: 'Seattle'
        }
     };
```

```
    callback(null, reply);
}
```

証明書プロバイダーを使用したセルフマネージド AWS IoT Core 証明書署名

AWS IoT Core 証明書プロバイダーを作成して、 AWS IoT フリートプロビジョニング で証明書署名リクエスト (CSRsに署名できます。証明書プロバイダーは、Lambda 関数と`CreateCertificateFromCsr`フリートプロビジョニング の MQTT API を参照します。Lambda 関数は CSR を受け入れ、署名付きクライアント証明書を返します。

に証明書プロバイダーがない場合 AWS アカウント、CreateCertificateFromCsr MQTT API はフリートプロビジョニングで呼び出され、CSR から証明書を生成します。証明書プロバイダーを作成すると、CreateCertificateFromCsr MQTT API の動作が変更され、この MQTT API へのすべての呼び出しは証明書プロバイダーを呼び出して証明書を発行します。

AWS IoT Core 証明書プロバイダーを使用すると、、他のパブリックに信頼された CAsAWS Private CA、または独自のパブリックキーインフラストラクチャ (PKI) などのプライベート認証機関 (CAs) を利用して CSR に署名するソリューションを実装できます。さらに、証明書プロバイダーを使用して、有効期間、署名アルゴリズム、発行者、拡張機能などのクライアント証明書のフィールドをカスタマイズできます。

> **Important**
>
> 作成できる証明書プロバイダーは、 AWS アカウントごとに 1 つだけです。署名動作の変更は、 AWS アカウントから証明書プロバイダーを削除するまで CreateCertificateFromCsr MQTT API を呼び出すフリート全体に適用されます。

このトピックの内容

- フリートプロビジョニングでのセルフマネージド証明書署名の仕組み
- 証明書プロバイダーの Lambda 関数入力
- 証明書プロバイダーの Lambda 関数の戻り値
- Lambda 関数の例
- フリートプロビジョニング用のセルフマネージド証明書署名
- AWS CLI 証明書プロバイダーの コマンド

フリートプロビジョニングでのセルフマネージド証明書署名の仕組み

主要なコンセプト

以下の概念は、フリー AWS IoT トプロビジョニングでのセルフマネージド証明書署名の仕組みを理解するのに役立つ詳細を提供します。詳細については、「フリートプロビジョニングを使用したデバイス証明書がないデバイスのプロビジョニング」を参照してください。

AWS IoT フリートのプロビジョニング

AWS IoT フリートプロビジョニング (フリートプロビジョニングの略) では、 は、初めて に接続するときにデバイス証明書 AWS IoT Core を生成し AWS IoT Core 、安全にデバイスに配信します。フリートプロビジョニングを使用して、デバイス証明書のないデバイスを AWS IoT Coreに接続できます。

証明書署名リクエスト (CSR)

フリートプロビジョニングの過程で、デバイスはフリートプロビジョニング MQTT API を介して AWS IoT Core にリクエストします。このリクエストには、クライアント証明書を作成するために署名される証明書署名リクエスト (CSR) が含まれます。

AWS フリートプロビジョニングでの マネージド証明書署名

AWS managed は、フリートプロビジョニングでの証明書署名のデフォルト設定です。 AWS マネージド証明書署名では、 AWS IoT Core は独自の CA を使用して CSRs に署名します。 CAs

フリートプロビジョニングでのセルフマネージド証明書署名

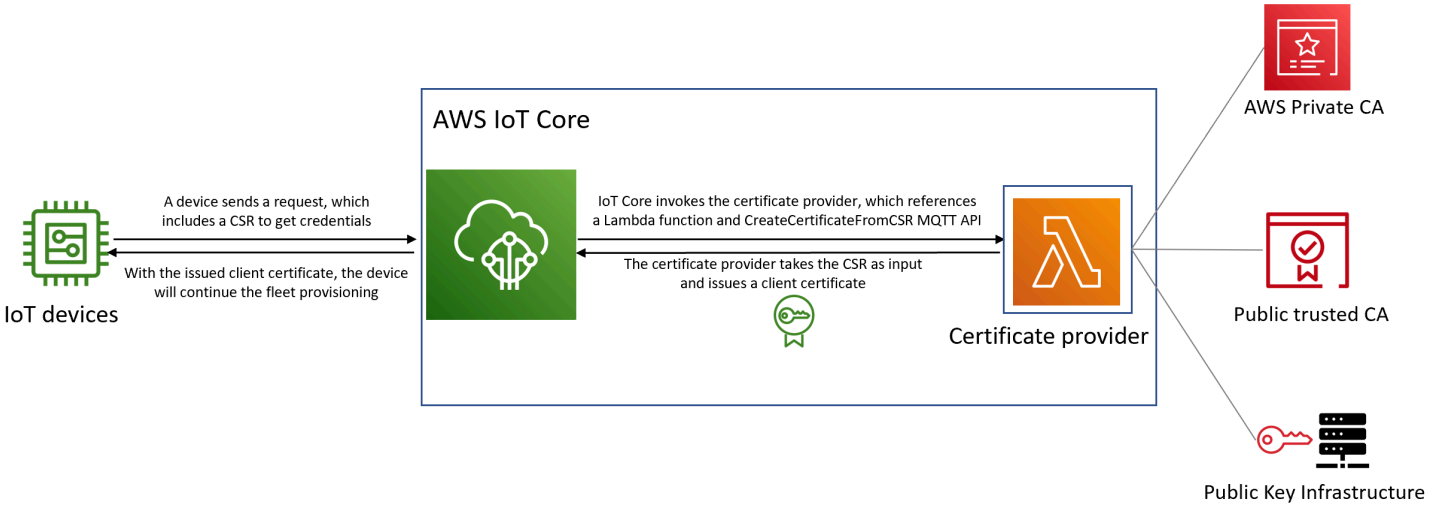
セルフマネージドは、フリートプロビジョニングで証明書に署名するためのもう 1 つのオプションです。セルフマネージド証明書署名では、CSR に署名するための AWS IoT Core 証明書プロバイダーを作成します。セルフマネージド証明書署名を使用して、 AWS プライベート CA、その他のパブリックに信頼された CA、または独自のパブリックキーインフラストラクチャ (PKI) によって生成された CA を使用して CSR に署名できます。

AWS IoT Core 証明書プロバイダー

AWS IoT Core 証明書プロバイダー (証明書プロバイダーの略) は、フリートプロビジョニングのセルフマネージド証明書署名に使用されるカスタマーマネージドリソースです。

図

次の図は、 AWS IoT フリートプロビジョニングでの自己証明書署名の仕組みを簡略化したものです。



- 新しい IoT デバイスが製造またはフリートに導入された場合、それ自体を認証するにはクライアント証明書が必要です AWS IoT Core。
- フリートプロビジョニングプロセスの一環として、デバイスはフリートプロビジョニング MQTT API を通じてクライアント証明書を AWS IoT Core にリクエストします。 APIs このリクエストには、証明書署名リクエスト (CSR) が含まれます。
- AWS IoT Core は証明書プロバイダーを呼び出し、CSR を入力としてプロバイダーに渡します。
- 証明書プロバイダーは CSR を入力として受け取り、クライアント証明書を発行します。

  AWS マネージド証明書署名の場合、 は独自の CA を使用して CSR AWS IoT Core に署名し、クライアント証明書を発行します。
- 発行されたクライアント証明書により、デバイスはフリートのプロビジョニングを継続し、 AWS IoT Coreとの安全な接続を確立します。

証明書プロバイダーの Lambda 関数入力

AWS IoT Core は、デバイスが Lambda 関数に登録するときに、次のオブジェクトを Lambda 関数に送信します。`certificateSigningRequest` の値は、`CreateCertificateFromCsr` リクエストで提供されるプライバシー強化メール (PEM) 形式の CSR です。`principalId` は、`CreateCertificateFromCsr`リクエストの実行 AWS IoT Core 時に に接続するために使用されるプリンシパルの ID です。 `clientId`は、MQTT 接続に設定されたクライアント ID です。

```
{
 "certificateSigningRequest": "string",
 "principalId": "string",
```

```
 "clientId": "string"
}
```

証明書プロバイダーの Lambda 関数の戻り値

Lambda 関数は、certificatePem 値を含むレスポンスを返す必要があります。以下は、成功したレスポンスの例です。 AWS IoT Core は戻り値 (certificatePem) を使用して証明書を作成します。

```
{
 "certificatePem": "string"
}
```

登録が成功すると、CreateCertificateFromCsr は CreateCertificateFromCsr レスポンスで同じ certificatePem を返します。詳細については、CreateCertificateFromCsr のレスポンスペイロードの例を参照してください。

Lambda 関数の例

証明書プロバイダーを作成する前に、CSR に署名するための Lambda 関数を作成する必要があります。次のコードは、Python での Lambda 関数の例です。この関数は AWS Private CA を呼び出して、プライベート CA と SHA256WITHRSA 署名アルゴリズムを使用して入力 CSR に署名します。返されたクライアント証明書は 1 年間有効です。の詳細 AWS Private CA とプライベート CA の作成方法については、AWS 「プライベート CA とは」および「プライベート CA の作成」を参照してください。

```
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 関数によって返される証明書には、証明書署名リクエスト (CSR) と同じサブジェクト名とパブリックキーが必要です。
- Lambda 関数の実行は 5 秒で完了する必要があります。
- Lambda 関数は、証明書プロバイダーリソースと同じ AWS アカウント およびリージョンにある必要があります。
- AWS IoT サービスプリンシパルには、Lambda 関数への呼び出しアクセス許可を付与する必要があります。混乱した代理問題を回避するため、 呼び出しアクセス許可には `sourceArn`と `sourceAccount` を設定することをお勧めします。詳細については、クロスサービスでの混乱した代理処理を防止するを参照してください。

Lambda の次のリソースベースのポリシー例では、Lambda 関数を呼び出すアクセス許可を AWS IoT に付与します。

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

フリートプロビジョニング用のセルフマネージド証明書署名

AWS CLI または AWS マネジメントコンソールを使用して、フリートプロビジョニング用のセルフマネージド証明書署名を選択できます。

AWS CLI

セルフマネージド証明書署名を選択するには、フリートプロビジョニングで CSRs に署名する証明書プロバイダーを作成 AWS IoT Core する必要があります。 は証明書プロバイダーを AWS IoT Core 呼び出し、CSR を入力として受け取り、クライアント証明書を返します。証明書プロバイダーを作成するには、`CreateCertificateProvider` API オペレーションまたは `create-certificate-provider` CLI コマンドを使用します。

> **Note**
>
> 証明書プロバイダーを作成すると、CreateCertificateFromCsrフリートプロビジョニング用の API の動作が変更され、`CreateCertificateFromCsr` へのすべての呼び出しが証明書プロバイダーを呼び出して証明書を作成します。この動作は、証明書プロバイダーの作成後に変更されるまでに数分かかる場合があります。

```
aws iot create-certificate-provider \
                --certificateProviderName my-certificate-provider \
```

```
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

次は、このコマンドの出力の例を示しています。

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

詳細については、「AWS IoT API リファレンス」の「CreateCertificateProvider」を参照してください。

AWS マネジメントコンソール

を使用したセルフマネージド証明書署名を選択するには AWS マネジメントコンソール、次の手順に従います。

1. [AWS IoT コンソール] に移動します。
2. 左のナビゲーションペインの [セキュリティ] で、[証明書] を選択します。
3. [証明書署名] ページで、[証明書署名の詳細] で [証明書署名方法の編集] を選択します。
4. [証明書署名方法の編集] ページで、[証明書署名方法] で [セルフマネージド] を選択します。
5. [セルフマネージド設定] セクションで、証明書プロバイダーの名前を入力し、Lambda 関数を作成または選択します。
6. [証明書署名を更新] を選択します。

AWS CLI 証明書プロバイダーの コマンド

証明書プロバイダーを作成する

証明書プロバイダーを作成するには、CreateCertificateProvider API オペレーションまたは create-certificate-provider CLI コマンドを使用します。

> **Note**
>
> 証明書プロバイダーを作成すると、CreateCertificateFromCsrフリートプロビジョニング用の API の動作が変更され、CreateCertificateFromCsr へのすべての呼び出しが証

明書プロバイダーを呼び出して証明書を作成します。この動作は、証明書プロバイダーの作成後に変更されるまでに数分かかる場合があります。

```
aws iot create-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

次は、このコマンドの出力の例を示しています。

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

詳細については、「AWS IoT API リファレンス」の「CreateCertificateProvider」を参照してください。

証明書プロバイダーを更新する

証明書プロバイダーを更新するには、UpdateCertificateProvider API オペレーションまたは update-certificate-provider CLI コマンドを使用します。

```
aws iot update-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-2 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

次は、このコマンドの出力の例を示しています。

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

詳細については、「AWS IoT API リファレンス」の「UpdateCertificateProvider」を参照してください。

証明書プロバイダーを記述する

証明書プロバイダーを記述するには、DescribeCertificateProvider API オペレーションまたは describe-certificate-provider CLI コマンドを使用します。

```
aws iot describe-certificate-provider --certificateProviderName my-certificate-provider
```

次は、このコマンドの出力の例を示しています。

```
{
 "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"
}
```

詳細については、「AWS IoT API リファレンス」の「DescribeCertificateProvider」を参照してください。

証明書プロバイダーを削除する

証明書プロバイダーを削除するには、DeleteCertificateProvider API オペレーションまたは delete-certificate-provider CLI コマンドを使用します。証明書プロバイダーリソースを削除すると、 の動作CreateCertificateFromCsrが再開され、CSR AWS IoT から によって署名された証明書 AWS IoT が作成されます。

```
aws iot delete-certificate-provider --certificateProviderName my-certificate-provider
```

このコマンドでは、出力が生成されません。

詳細については、「AWS IoT API リファレンス」の「DeleteCertificateProvider」を参照してください。

証明書プロバイダーを一覧表示する

内の証明書プロバイダーを一覧表示するには AWS アカウント、 `ListCertificateProviders` API オペレーションまたは `list-certificate-providers` CLI コマンドを使用します。

```
aws iot list-certificate-providers
```

次は、このコマンドの出力の例を示しています。

```
{
 "certificateProviders": [
  {
   "certificateProviderName": "my-certificate-provider",
   "certificateProviderArn": "arn:aws:iot:us-
east-1:123456789012:certificateprovider:my-certificate-provider"
  }
 ]
}
```

詳細については、「AWS IoT API リファレンス」の「`ListCertificateProvider`」を参照してください。

デバイスをインストールするユーザーの IAM ポリシーとロールの作成

> **Note**
>
> これらの手順は、 AWS IoT コンソールから指示された場合にのみ使用します。
> コンソールからこのページに移動するには、[create a new provisioning template] (新しいプロビジョニングテンプレートを作成) を開きます。

AWS IoT これをコンソールで実行できないのはなぜですか?

最も安全な操作のために、IAM アクションは IAM コンソールで実行されます。このセクションの手順では、プロビジョニングテンプレートを使用するために必要な IAM ロールとポリシーを作成するステップについて説明します。

デバイスをインストールするユーザーの IAM ポリシーの作成

この手順では、プロビジョニングテンプレートを使用してデバイスをインストールすることをユーザーに許可する IAM ポリシーの作成方法について説明します。

この手順の実行中に、IAM コンソールと AWS IoT コンソールを切り替えます。この手順を完了するまで、両方のコンソールを同時に開いておくことをお勧めします。

デバイスをインストールするユーザーの IAM ポリシーを作成するには

1. IAM コンソールのポリシーハブを開きます。
2. [ポリシーを作成] を選択します。
3. [ポリシーの作成] ページで、[JSON] タブを選択します。
4. ユーザーポリシーとロールの設定を選択した AWS IoT コンソールのページに切り替えます。
5. [Sample provisioning policy] (プロビジョニングポリシーのサンプル) で、[Copy] (コピー) を選択します。
6. IAM コンソールに切り替えます。
7. JSON エディタで、 AWS IoT コンソールからコピーしたポリシーを貼り付けます。このポリシーは、 AWS IoT コンソールで作成するテンプレートに固有です。
8. 続行するには、[Next: Tags] (次へ: タグ) を選択します。
9. [Add tags (Optional)] (タグの追加 (オプション)) ページで、このポリシーに追加するタグごとに [Add tag] (タグを追加) を選択します。追加するタグがない場合は、このステップをスキップできます。
10. [次へ: 確認] を選択して続行します。
11. [ポリシーの確認] ページで、以下の作業を行います。
    a. [Name*] (名前*) に、ポリシーの目的を簡単に示すポリシー名を入力します。

       次の手順で使用するため、このポリシーの名前を書き留めておきます。
    b. 作成するポリシーの説明 (オプション) を入力できます。
    c. このポリシーの残りの部分とそのタグを確認します。
12. ポリシーの作成を完了するには、[Create Policy] (ポリシーの作成) を選択します。

新しいポリシーを作成したら、引き続き「the section called “デバイスをインストールするユーザーの IAM ロールの作成”」に移動して、このポリシーをアタッチするユーザーのロールエントリを作成します。

デバイスをインストールするユーザーの IAM ロールの作成

以下の手順では、プロビジョニングテンプレートを使用してデバイスをインストールするユーザーを認証する IAM ロールの作成方法について説明します。

デバイスをインストールするユーザーの IAM ポリシーを作成するには

1. IAM コンソールのロールハブを開きます。
2. [ロールの作成] を選択してください。
3. [Select trusted entity] (信頼されたエンティティを選択) で、作成するテンプレートへのアクセスを許可する信頼されたエンティティのタイプを選択します。
4. アクセスを許可する信頼されたエンティティの ID を選択または入力して、[Next] (次へ) を選択します。
5. [Add permissions] (許可を追加) ページの [Permission policies] (許可ポリシー) で、検索ボックスに前の手順で作成したポリシーの名前を入力します。
6. ポリシーのリストで、前の手順で作成したポリシーを選択し、[Next] (次へ) を選択します。
7. [Name, review, and create] (名前、確認、および作成) セクションで、以下の操作を実行します。

    a. [Role name] (ロール名) に、このロールの目的を簡単に示すロール名を入力します。

    b. [Description] (説明) に、ロールの説明 (オプション) を入力できます。これを省略しても続行できます。

    c. ステップ 1 とステップ 2 の値を確認します。

    d. [Add tags (Optional)] (タグの追加 (オプション)) では、このロールにタグを追加することを選択できます。これを省略しても続行できます。

    e. このページの情報が完全で正しいことを確認して、[Create role] (ロールを作成) を選択します。

新しいロールを作成したら、 AWS IoT コンソールに戻ってテンプレートの作成を続行します。

新しいテンプレートを許可するように既存のポリシーを更新する

次の手順では、プロビジョニングテンプレートを使用してデバイスをインストールすることをユーザーに許可する新しいテンプレートを IAM ポリシーに追加する方法について説明します。

既存の IAM ポリシーに新しいテンプレートを追加するには

1. IAM コンソールのポリシーハブを開きます。
2. 検索ボックスに、更新するポリシーの名前を入力します。
3. 検索ボックスの下のリストで、更新するポリシーを見つけ、そのポリシー名を選択します。
4. [Policy summary] (ポリシー概要) で、[JSON] タブを選択します (まだ表示されていない場合)。
5. ポリシードキュメントを編集するには、[Edit policy] (ポリシーの編集) を選択します。
6. エディタで、[JSON] タブを選択します (まだ表示されていない場合)。
7. ポリシードキュメントで、`iot:CreateProvisioningClaim` アクションが含まれているポリシーステートメントを見つけます。

   ポリシードキュメント内に `iot:CreateProvisioningClaim` アクションが含まれているポリシーステートメントがない場合は、次のステートメントスニペットをコピーして、ポリシードキュメント内の `Statement` 配列に追加のエントリとして貼り付けます。

   > **Note**
   >
   > このスニペットは、`Statement` 配列の閉じ文字 `]` の前に配置する必要があります。構文エラーを修正するには、このスニペットの前または後にカンマの追加が必要になる場合があります。

```
{
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "--PUT YOUR NEW TEMPLATE ARN HERE--"
    ]
}
```

8. ユーザーロールのアクセス許可の変更を選択した AWS IoT コンソールのページに切り替えます。
9. テンプレートのリソース ARN を見つけて [Copy] (コピー) を選択します。
10. IAM コンソールに切り替えます。
11. コピーした Amazon リソースネーム (ARN) を、`Statement` 配列内でテンプレート ARN のリストの先頭に貼り付けて最初のエントリにします。

    これが配列内で唯一の ARN である場合は、貼り付けた値の末尾にあるカンマを削除します。
12. 更新したポリシーステートメントを確認し、エディタによって示されたエラーがあれば修正します。
13. 更新したポリシードキュメントを保存するには、[Review policy] (ポリシーの確認) を選択します。
14. ポリシーを確認して、[Save changes] (変更の保存) を選択します。
15. AWS IoT コンソールに戻ります。

デバイスプロビジョニング MQTT API

フリートプロビジョニングサービスは、次の MQTT API オペレーションをサポートしています。

- the section called “CreateCertificateFromCsr”
- the section called “CreateKeysAndCertificate”
- the section called “RegisterThing”

この API は、トピックの*#######*に応じて、簡潔なバイナリオブジェクト表現 (CCOR) 形式および JavaScript Object Notation (JSON) 形式のレスポンスバッファをサポートします。わかりやすくするために、このセクションのレスポンスとリクエストの例を JSON 形式で示します。

*******	レスポンス形式のデータ型
cbor	簡潔なバイナリオブジェクトの表現 (CCOR)
json	JavaScript Object Notation (JSON)

> **Important**
>
> リクエストメッセージトピックを発行する前に、応答トピックをサブスクライブしてレスポンスを受信します。この API で使用されるメッセージは、MQTT のパブリッシュ/サブスクライブプロトコルを使用して、リクエストとレスポンスの相互作用を提供します。
> リクエストを発行する前にレスポンストピックをサブスクライブしていない場合、そのリクエストの結果を受信できない可能性があります。
> IoT Core フリートプロビジョニングは、API リクエストの発行に使用したのと同じ MQTT 接続を介して、デバイスのプロビジョニング MQTT API 結果を返します。

CreateCertificateFromCsr

証明書署名リクエスト (CSR) から証明書を作成します。 は、Amazon ルート認証局 (CA) によって署名されたクライアント証明書 AWS IoT を提供します。新しい証明書には PENDING_ACTIVATION ステータスがあります。RegisterThing を呼び出して、この証明書を使用してモノをプロビジョニングすると、テンプレートで説明されているように、証明書のステータスが ACTIVE または INACTIVE に変わります。

認証局証明書と証明書署名リクエストを使用してクライアント証明書を作成する方法の詳細については、「CA 証明書を使用してクライアント証明書を作成する」を参照してください。

> **Note**
>
> セキュリティ上の理由から、CreateCertificateFromCsr によって返される certificateOwnershipToken は 1 時間後に有効期限切れになります。certificateOwnershipToken が有効期限切れになる前に、RegisterThing を呼び出す必要があります。CreateCertificateFromCsr によって作成された証明書がトークンの有効期限が切れるまでにアクティブにならず、ポリシーまたはモノにアタッチされない場合、証明書は削除されます。トークンの有効期限が切れた場合、デバイスは CreateCertificateFromCsr を呼び出して新しい証明書を生成します。

CreateCertificateFromCsr リクエスト

`$aws/certificates/create-from-csr/`*`payload-format`* トピックを含むメッセージを発行します。

`payload-format`

メッセージペイロード形式 (`cbor` または `json`)。

CreateCertificateFromCsr リクエストペイロード

```
{
    "certificateSigningRequest": "string"
}
```

`certificateSigningRequest`

PEM 形式の CSR。

CreateCertificateFromCsr レスポンス

`$aws/certificates/create-from-csr/`*`payload-format`*`/accepted` をサブスクライブします。

`payload-format`

メッセージペイロード形式 (`cbor` または `json`)。

CreateCertificateFromCsr レスポンスペイロード

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

`certificateOwnershipToken`

プロビジョニング中に証明書の所有権を証明するトークン。

`certificateId`

証明書の ID。証明書管理オペレーションでは、証明書 ID のみが使用されます。

`certificatePem`

PEM 形式の証明書データ。

CreateCertificateFromCsr エラー

エラーレスポンスを受信するには、$aws/certificates/create-from-csr/*payload-format*/rejected をサブスクライブします。

payload-format

メッセージペイロード形式 (cbor または json)。

CreateCertificateFromCsr エラーペイロード

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

statusCode

ステータスコード。

errorCode

エラーコード。

errorMessage

エラーメッセージです。

CreateKeysAndCertificate

新しいキーと証明書を作成します。Amazon ルート認証局 (CA) によって署名されたクライアント証明書 AWS IoT を提供します。新しい証明書には PENDING_ACTIVATION ステータスがあります。RegisterThing を呼び出して、この証明書を使用してモノをプロビジョニングすると、テンプレートで説明されているように、証明書のステータスが ACTIVE または INACTIVE に変わります。

> **Note**
>
> セキュリティ上の理由から、CreateKeysAndCertificate によって返される certificateOwnershipToken は 1 時間後に有効期限切れになりま

す。certificateOwnershipToken が有効期限切れになる前に、RegisterThing を呼び出す必要があります。CreateKeysAndCertificate によって作成された証明書がトークンの有効期限が切れるまでにアクティブにならず、ポリシーまたはモノにアタッチされない場合、証明書は削除されます。トークンの有効期限が切れた場合、デバイスは CreateKeysAndCertificate を呼び出して新しい証明書を生成します。

CreateKeysAndCertificate リクエスト

空のメッセージペイロードで $aws/certificates/create/*payload-format* にメッセージを発行します。

`payload-format`

メッセージペイロード形式 (cbor または json)。

CreateKeysAndCertificate レスポンス

$aws/certificates/create/*payload-format*/accepted をサブスクライブします。

`payload-format`

メッセージペイロード形式 (cbor または json)。

CreateKeysAndCertificate レスポンス

```
{
    "certificateId": "string",
    "certificatePem": "string",
    "privateKey": "string",
    "certificateOwnershipToken": "string"
}
```

`certificateId`

証明書 ID。

`certificatePem`

PEM 形式の証明書データ。

`privateKey`

プライベートキー。

`certificateOwnershipToken`

プロビジョニング中に証明書の所有権を証明するトークン。

CreateKeysAndCertificate エラー

エラーレスポンスを受信するには、`$aws/certificates/create/`*`payload-format`*`/rejected` をサブスクライブします。

`payload-format`

メッセージペイロード形式 (`cbor` または `json`)。

CreateKeysAndCertificate エラーペイロード

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

ステータスコード。

`errorCode`

エラーコード。

`errorMessage`

エラーメッセージです。

RegisterThing

事前定義されたテンプレートを使用してモノをプロビジョニングします。

RegisterThing リクエスト

`$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`* にメッセージを発行します。

`payload-format`

メッセージペイロード形式 (`cbor` または `json`)。

`templateName`

プロビジョニングテンプレート名。

RegisterThing リクエストペイロード

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
        "string": "string",
        ...
    }
}
```

`certificateOwnershipToken`

証明書の所有権を証明するトークン。MQTT で証明書を作成すると、 AWS IoT によってトークンが生成されます。

`parameters`

オプション。登録リクエストを評価するために事前プロビジョニングフックで使用されるデバイスからの、キーと値のペア。

RegisterThing レスポンス

`$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/accepted` をサブスクライブします。

`payload-format`

メッセージペイロード形式 (`cbor` または `json`)。

`templateName`

プロビジョニングテンプレート名。

RegisterThing レスポンスペイロード

```
{
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

`deviceConfiguration`

テンプレートで定義されているデバイス設定。

`thingName`

プロビジョニング中に作成される IoT モノの名前。

RegisterThing エラーレスポンス

エラーレスポンスを受信するには、`$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/rejected` をサブスクライブします。

`payload-format`

メッセージペイロード形式 (`cbor` または `json`)。

`templateName`

プロビジョニングテンプレート名。

RegisterThing エラーレスポンスペイロード

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
```

```
}
```

`statusCode`

ステータスコード。

`errorCode`

エラーコード。

`errorMessage`

エラーメッセージです。

フリートインデックス作成

フリートインデックス作成を使用して、次のソースからデバイスのデータのインデックス作成、検索、集計を実行できます。AWS IoT レジストリ、AWS IoT デバイスシャドウ、AWS IoT 接続性、AWS IoT Device Management Software Package Catalog、および AWS IoT Device Defender 違反。デバイスのグループをクエリし、状態、接続性、デバイスの違反など、さまざまなデバイス属性の組み合わせに基づくデバイスレコードの統計情報を集計できます。フリートインデックス作成を使用すると、デバイスにおけるフリートの整理、調査、トラブルシューティングを実行できます。

フリートインデックス作成には、次の機能があります。

インデックスの更新の管理

モノのグループ、モノのレジストリ、デバイスシャドウ、デバイスの接続性、デバイスの違反における更新をインデックス化するようにフリートインデックスを設定できます。フリートインデックス作成をアクティブにすると、 AWS IoT によりモノまたはモノのグループのインデックスが作成されます。AWS_Things はすべてのモノに対して作成されるインデックスで、AWS_ThingGroups はすべてのモノのグループが含まれるインデックスです。フリートインデックス作成がアクティブになると、インデックスに対してクエリを実行できます。たとえば、ハンドヘルドでバッテリー残量が 70% を超えるすべてのデバイスを見つけることができます。 は、最新のデータでインデックスを継続的に AWS IoT 更新します。詳細については、「Managing fleet indexing」を参照してください。

特定のデバイスの接続ステータスのクエリ

この API は、最新のデバイス固有の接続情報への低レイテンシーで高スループットのアクセスを提供します。詳細については、「デバイスの接続ステータス」 を参照してください。

データソース全体での検索

クエリ言語に基づいてクエリ文字列を作成し、それを使用してデータソース全体を検索できます。また、フリートインデックス作成の設定でデータソースを設定して、インデックス作成の設定に検索するデータソースを含める必要があります。クエリ文字列には、検索したいモノが記述されます。クエリは、 AWS マネージドフィールド、カスタムフィールド、およびインデックス化されたデータソースからの任意の属性を使用して作成できます。フリートのインデックス作成をサポートするデータソースの詳細については、「モノのインデックス作成の管理」を参照してください。

集計データのクエリ

デバイスで集計データを検索し、特定のフィールドに関連した検索クエリを使用して、統計情報、パーセンタイル、カーディナリティ、またはモノのリストを返すことができます。 AWS マネージドフィールドまたはフリートインデックス作成設定内のカスタムフィールドとして設定した属性で集計を実行できます。集計クエリの詳細については、「集計データのクエリ」を参照してください。

フリートメトリクスを使用した集計データのモニタリングとアラームの作成

フリートメトリクスを使用して CloudWatch に集計データを自動的に送信し、トレンドを分析し、事前定義されたしきい値に基づいてフリートの集計状態をモニタリングするためのアラームを作成できます。フリートメトリクスの詳細については、「フリートメトリクス」を参照してください。

フリートのインデックス作成の管理

フリートインデックス作成では、モノのインデックス作成およびモノのグループのインデックス作成の 2 種類のインデックスを管理できます。

モノのインデックス作成

すべてのモノに対して作成されるインデックスは `AWS_Things` と呼ばれます。モノのインデックス作成では、次のデータソースがサポートされています。AWS IoT レジストリデータ、AWS IoT デバイスシャドウデータ、AWS IoT 接続性データ、AWS IoT Device Defender 違反データ。これらのデータソースをフリートのインデックス作成設定に追加することで、モノの検索、集計データのクエリを行い、検索クエリに基づく動的なモノのグループとフリートメトリクスを作成できます。

Registry は、モノの管理に役立つレジストリAWS IoT を提供します。レジストリデータをフリートのインデックス作成設定に追加して、モノの名前、説明、およびその他のレジストリ属性に基づいてデバイスを検索できます。レジストリの詳細については、「レジストリによるモノの管理方法」を参照してください。

シャドウ — AWS IoT Device Shadow サービスでは、デバイスの状態についてのデータの保存に役立つシャドウが提供されています。モノのインデックス作成では、名前のないクラシックシャドウおよび名前付きシャドウの両方がサポートされています。名前付きシャドウのインデックスを作成するには、名前付きシャドウの設定を有効にし、モノのインデックス作成設定でシャドウ名を指定します。デフォルトでは、1 つにつき最大 10 個のシャドウ名を追加できます AWS アカウント。

シャドウ名数の上限を増やす方法については、「AWS 全般のリファレンス」の「AWS IoT Device Management クォータ」を参照してください。

インデックス作成のために名前付きシャドウを追加するには:

- AWS IoT コンソールを使用する場合、[Thing indexing] (モノのインデックス作成) を有効化し、[Add named shadows] (名前付きシャドウの追加) を選択して、[Named shadow selection] (名前付きシャドウの選択) からシャドウの名前を追加します。
- AWS Command Line Interface (AWS CLI) を使用する場合は、 `namedShadowIndexingMode` をに設定し`ON`、 でシャドウ名を指定します`IndexingFilter`。CLI コマンドの例を確認するには、「モノのインデックス作成の管理」を参照してください。

> **Important**
>
> 2022 年 7 月 20 日は、 AWS IoT Device Management フリートインデックス作成と AWS IoT Core 名前付きシャドウの統合の一般提供 (GA) リリースであり、違反 AWS IoT Device Defender を検出します。この一般提供版 (GA) リリースにより、ユーザーはシャドウ名を指定して、特定の名前付きシャドウにインデックスを付けることができます。2021 年 11 月 30 日から 2022 年 7 月 19 日までのこの機能のパブリックプレビュー期間中に、インデックス作成のために名前付きシャドウを追加した場合は、フリートのインデックス作成設定を再構成して特定のシャドウ名を選択することにより、インデックス作成コストを削減してパフォーマンスを最適化するようお勧めします。

シャドウの詳細については、「AWS IoT Device Shadow サービス」を参照してください。

接続性 — デバイスの接続性に関するデータは、デバイスの接続ステータスを特定するのに役立ちます。この接続性に関するデータは、ライフサイクルイベントによって扱われます。クライアントが接続または切断すると、 は MQTT トピックにメッセージを含むライフサイクルイベント AWS IoT を発行します。接続または切断のメッセージは、接続ステータスの詳細を提供する JSON 要素のリストになります。デバイスの接続性についての詳細は、「ライフサイクルイベント」を参照してください。

Device Defender 違反データAWS IoT Device Defender を使用すると、セキュリティプロファイルで定義した通常の動作に対して異常なデバイス動作を特定できます。Security Profile には、予想される一連のデバイスの動作が含まれています。各動作では、デバイスにおける通常の動作を指定するメトリクスを使用します。Device Defender 違反の詳細については、「AWS IoT Device Defender 検出」を参照してください。

詳細については、「モノのインデックス作成の管理」を参照してください。

モノのグループのインデックス作成

AWS_ThingGroups は、すべてのモノのグループが含まれるインデックスです。このインデックスを使用して、グループ名、説明、属性、すべての親グループ名に基づいてグループを検索することができます。

詳細については、「モノのグループのインデックス作成の管理」を参照してください。

管理対象フィールド

マネージドフィールドには、モノ、モノのグループ、デバイスシャドウ、デバイス接続、Device Defender 違反に関連するデータが含まれます。 は、マネージドフィールドのデータ型 AWS IoT を定義します。 AWS IoT モノを作成するときに、各マネージドフィールドの値を指定します。例えば、モノの名前、モノのグループ、モノの説明はすべて管理対象フィールドです。フリートインデックス作成では、指定したインデックス作成モードに基づいて、管理対象フィールドがインデックス化されます。customFields では、管理対象フィールドを変更したり表示することはできません。詳細については、「Custom fields」を参照してください。

モノのインデックス作成の管理対象フィールドを次に示します。

- レジストリの管理対象フィールド

```
"managedFields" : [
  {name:thingId, type:String},
  {name:thingName, type:String},
  {name:registry.version, type:Number},
  {name:registry.thingTypeName, type:String},
  {name:registry.thingGroupNames, type:String},
]
```

- 名前のないクラシックシャドウの管理対象フィールド

```
"managedFields" : [
  {name:shadow.version, type:Number},
  {name:shadow.hasDelta, type:Boolean}
]
```

- 名前付きシャドウの管理対象フィールド

```
"managedFields" : [
```

```
  {name:shadow.name.shadowName.version, type:Number},
  {name:shadow.name.shadowName.hasDelta, type:Boolean}
]
```

- モノの接続性の管理対象フィールド

```
"managedFields" : [
  {name:connectivity.timestamp, type:Number},
  {name:connectivity.version, type:Number},
  {name:connectivity.connected, type:Boolean},
  {name:connectivity.disconnectReason, type:String}
]
```

- Device Defender の管理フィールド

```
"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}
]
```

- モノのグループの管理対象フィールド

```
"managedFields" : [
  {name:description, type:String},
  {name:parentGroupNames, type:String},
  {name:thingGroupId, type:String},
  {name:thingGroupName, type:String},
  {name:version, type:Number},
]
```

次の表は、検索できない管理フィールドです。

データソース	検索できない管理フィールド
レジストリ	`registry.version`
名前のないシャドウ	`shadow.version`

データソース	検索できない管理フィールド
名前付きシャドウ	`shadow.name.*.version`
Device Defender	`deviceDefender.version`
モノのグループ	`version`

カスタムフィールド

モノの属性、Device Shadow データ、Device Defender 違反のデータを集計するには、カスタムフィールドを作成してそれらをインデックス化します。`customFields` 属性は、フィールド名とデータ型のペアのリストです。データ型に基づいて、集計クエリを実行できます。選択するインデックス作成モードは、`customFields` で指定できるフィールドに影響します。例えば、`REGISTRY` インデックス作成モードを指定した場合、Thing Shadow からカスタムフィールドを指定することはできません。update-indexing-configuration CLI コマンドを使用して、カスタムフィールドを作成または更新できます (Updating indexing configuration examples のコマンド例を参照してください)。

- カスタムフィールド名

モノおよび モノのグループ属性のカスタムフィールド名は `attributes.` で始まり、属性名が続きます。名前のないシャドウのインデックス作成がオンの場合、モノのカスタムフィールド名は `shadow.desired` または `shadow.reported` で始まり、名前のないシャドウデータの値名が続きます。名前付きシャドウのインデックス作成がオンの場合、モノのカスタムフィールド名は `shadow.name.*.desired.` または `shadow.name.*.reported.` で始まり、名前付きシャドウデータの値が続きます。Device Defender 違反のインデックス作成がオンの場合、モノのカスタムフィールド名は `deviceDefender.` で始まり、Device Defender 違反のデータの値が続きます。

プレフィックスに続く属性名またはデータの値名には、英数字、— (ハイフン)、_ (アンダースコア) のみを使用できます。スペースを含めることはできません。

設定内のカスタムフィールドとインデックス化される値のタイプに不整合がある場合、フリートインデックス作成では集計クエリの不整合値が無視されます。CloudWatch ログは、集約クエリの問題のトラブルシューティングに役立ちます。詳細については、「フリートインデックス作成サービスの集計クエリのトラブルシューティング」を参照してください。

- カスタムフィールドの型

カスタムフィールドのタイプには、次の値がサポートされています: Number、String、Boolean

モノのインデックス作成の管理

すべてのモノに対して作成されるインデックスは AWS_Things です。次のデータソースからインデックス化する対象を制御できます。AWS IoT レジストリデータ、AWS IoT デバイスシャドウデータ、AWS IoT 接続性データ、および AWS IoT Device Defender 違反データ。

このトピックの内容

- モノのインデックス作成の有効化
- モノのインデックスの説明
- モノのインデックスのクエリ
- 制約と制限
- Authorization

モノのインデックス作成の有効化

update-indexing-configuration CLI コマンドまたは UpdateIndexingConfiguration API オペレーションを使用して、AWS_Things インデックスを作成し、その設定を管理できます。--thing-indexing-configuration (thingIndexingConfiguration) パラメータを使用すると、インデックス化されるデータの種類 (レジストリ、シャドウ、デバイスの接続性データ、Device Defender 違反のデータなど) を管理できます。

--thing-indexing-configuration パラメータは、次の構造を持つ文字列を取ります。

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

モノのインデックス作成モード

インデックス作成設定では、デバイスのインデックス作成と検索を行うデータソースに応じて、さまざまなモノのインデックス作成モードを指定できます。

- thingIndexingMode: レジストリまたはシャドウがインデックス作成されているかどうかを制御します。thingIndexingMode が OFF に設定されている場合、モノのインデックス作成は無効になります。

- thingConnectivityIndexingMode: 属性は、モノの接続性データにインデックスを付けるかどうかを指定します。

- deviceDefenderIndexingMode: 属性は、Device Defender 違反のデータにインデックスを付けるかどうかを指定します。
- namedShadowIndexingMode: 属性は、名前付きシャドウデータにインデックスを付けるかどうかを指定します。フリートのインデックス作成設定に追加する名前付きシャドウを選択するには、namedShadowIndexingMode を ON に設定して、filter で名前付きのシャドウ名を指定します。

次の表は、各インデックス作成モードの有効な値と、各値に対してインデックス化されたデータソースを示しています。

属性	有効値	レジストリ	シャドウ	接続	DD 違反	名前付きシャドウ
thingIndexingMode	VOFF					
	REGISTRY	✓				
	REGISTRY_AND_SHADOW	✓	✓			
thingConnectivityIndexingMode	未指定。					
	VOFF					
	STATUS			✓		
deviceDefenderIndexingMode	未指定。					
	VOFF					
	VIOLATIONS				✓	
namedShadowIndexingMode	未指定。					
	VOFF					
	ON					✓

管理対象フィールドとカスタムフィールド

管理対象フィールド

マネージドフィールドには、モノ、モノのグループ、デバイスシャドウ、デバイス接続、Device Defender 違反に関連するデータが含まれます。 は、マネージドフィールドのデータ型 AWS IoT を定義します。 AWS IoT モノを作成するときに、各管理対象フィールドの値を指定します。例えば、モノの名前、モノのグループ、モノの説明はすべて管理対象フィールドです。フリートインデックス作成では、指定したインデックス作成モードに基づいて、管理対象フィールドがインデックス化されます。customFields では、管理対象フィールドを変更したり表示することはできません。

カスタムフィールド

カスタムフィールドを作成してインデックスを作成し、属性、Device Shadow データ、および Device Defender 違反のデータを集計できます。customFields 属性は、フィールド名とデータ型のペアのリストです。データ型に基づいて、集計クエリを実行できます。選択するインデックス作成モードは、customFields で指定できるフィールドに影響します。例えば、REGISTRY インデックス作成モードを指定した場合、Thing Shadow からカスタムフィールドを指定することはできません。update-indexing-configuration CLI コマンドを使用して、カスタムフィールドを作成または更新できます (Updating indexing configuration examples のコマンド例を参照してください)。詳細については、「Custom fields」を参照してください。

インデックス作成フィルター

インデックス作成フィルターは、名前付きシャドウと位置情報データに追加の選択を提供します。

namedShadowNames

フリートのインデックス作成設定に追加する名前付きシャドウを追加するには、namedShadowIndexingMode を ON に設定して、namedShadowNames フィルターで名前付きのシャドウ名を指定します。

例

```
"filter": {
     "namedShadowNames": [ "namedShadow1", "namedShadow2" ]
}
```

geoLocations

フリートのインデックス作成設定に位置情報データを追加するには:

- 位置情報データがクラシック (名前なし) シャドウに保存されている場合は、thingIndexingMode を REGISTRY_AND_SHADOW に設定し、geoLocations フィルターで位置情報データを指定します。

  次のフィルター例では、クラシック (名前なし) シャドウ内の geoLocation オブジェクトを指定します。

  ```
  "filter": {
       "geoLocations": [
          {
              "name": "shadow.reported.location",
  ```

```
            "order": "LonLat"
        }
    ]
}
```

- 位置情報データが名前付きシャドウに保存されている場合は、namedShadowIndexingMode を ON に設定し、シャドウ名を namedShadowNames フィルターに追加し、位置情報データを geoLocations フィルターに指定します。

  以下のフィルター例では、名前付きシャドウ (nameShadow1) 内の geoLocation オブジェクトを指定します。

```
"filter": {
    "namedShadowNames": [ "namedShadow1" ],
    "geoLocations": [
       {
           "name": "shadow.name.namedShadow1.reported.location",
           "order": "LonLat"
       }
    ]
}
```

詳細については、「AWS IoT API リファレンス」の「IndexingFilter」を参照してください。

インデックス作成の設定例を更新する

インデックス作成設定を更新するには、 update-indexing-configuration CLI コマンド を使用します AWS IoT 。次の例では、update-indexing-configuration の使用方法を示します。

短い構文:

```
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 構文:

```
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} ] } }'
```

このコマンドでは、出力が生成されません。

モノのインデックスのステータスを確認するには、describe-index CLI コマンドを実行します。

```
aws iot describe-index --index-name "AWS_Things"
```

describe-index コマンドの出力は以下のようになります。

```
{
    "indexName": "AWS_Things",
    "indexStatus": "ACTIVE",
    "schema": "MULTI_INDEXING_MODE"
}
```

> **Note**
>
> フリートインデックス作成でのフリートインデックスの更新には、しばらく時間がかかる場合があります。使用する前に indexStatus が ACTIVE になるまで待つことをお勧めします。スキーマフィールドには、設定したデータソースに応じて異なる値を指定できます。詳細については、「モノのインデックスの説明」を参照してください。

モノのインデックス設定についての詳細を取得するには、get-indexing-configuration CLI コマンドを実行します。

```
aws iot get-indexing-configuration
```

get-indexing-configuration コマンドの出力は以下のようになります。

```
{
    "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"
    }
}
```

カスタムフィールドを更新するには、update-indexing-configuration コマンドを実行します。例は次のとおりです。

```
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}]'
```

このコマンドは、インデックス作成設定 shadow.desired.intensity に追加されました。

> **Note**
>
> カスタムフィールドのインデックス作成設定が更新されると、すべての既存のカスタムフィールドが上書きされます。update-indexing-configuration を呼び出すときは、必ずすべてのカスタムフィールドを指定してください。

インデックスを再構築したら、新しく追加したフィールドに集計クエリを使用して、レジストリデータ、シャドウデータ、およびモノの接続ステータスについてのデータを検索できます。

インデックス作成モードを変更する場合、新しいインデックス作成モードを使用して、すべてのカスタムフィールドが有効であることを確認してください。例えば、shadow.desired.temperature というカスタムフィールドを使用して REGISTRY_AND_SHADOW モードを開始する場合、インデックス作成モードを REGISTRY に変更する前に shadow.desired.temperature カスタムフィールドを削除する必要があります。インデックス作成設定にインデックス作成モードによってインデックス化されていないカスタムフィールドが含まれている場合、更新は失敗します。

モノのインデックスの説明

次のコマンドは、describe-index CLI コマンドを使用して現在のモノのインデックスのステータスを取得する方法を示しています。

```
aws iot describe-index --index-name "AWS_Things"
```

コマンドのレスポンスは次のようになります。

```
{
    "indexName": "AWS_Things",
    "indexStatus": "BUILDING",
    "schema": "REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS"
}
```

初めてフリートインデックスを作成すると、 はインデックス AWS IoT を構築します。indexStatus が BUILDING の状態の場合、インデックスに対してクエリを実行することはできません。モノのインデックスの schema はどのタイプのデータのインデックスが作成されるかを示します (REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS)。

インデックスの設定を変更すると、インデックスが再構築されます。このプロセス中の indexStatus は REBUILDING です。再構築中に、モノのインデックスのデータに対してクエリを実行できます。たとえば、インデックスの再構築中にインデックス設定を REGISTRY から REGISTRY_AND_SHADOW に変更した場合、最新の更新を含むレジストリデータのクエリを行うことができます。ただし、再構築が完了するまで Shadow データのクエリを行うことはできません。インデックスの作成または再構築に要する時間は、データの量によって異なります。

スキーマフィールドには、設定したデータソースに応じて、さまざまな値を表示できます。次の表には、さまざまなスキーマの値と対応する説明が記載されています。

Schema	説明
VOFF	データソースが設定またはインデックス化されていません。
REGISTRY	レジストリデータがインデックス化されます。

Schema	説明
REGISTRY_AND_SHADOW	レジストリデータおよび名前のない (クラシック) シャドウデータがインデックス化されます。
REGISTRY_AND_CONNECTIVITY	レジストリデータおよび接続性データがインデックス化されます。
REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS	レジストリデータ、名前のない (クラシック) シャドウデータ、および接続性データがインデックス化されます。
MULTI_INDEXING_MODE	レジストリ、名前のない (クラシック) シャドウ、または接続性データに加えて、名前付きシャドウまたは Device Defender 違反のデータがインデックス化されます。

モノのインデックスのクエリ

インデックスのデータにクエリを実行するには、CLI の search-index コマンドを使用します。

```
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"
}
```

この JSON レスポンスでは、"connectivity"は、(thingConnectivityIndexingMode=STATUS 設定で有効にされるように) デバイスが AWS IoT Coreに接続されているかどうかを示すブール値、タイムスタンプ、あるいは、disconnectReason をを提供します。デバイス "mything1" は、POSIX 時間 1556649874716 にCONNECTION_LOSTのために切断されました (false)。切断理由の詳細については、「ライフサイクルイベント」を参照してください。

```
"connectivity": {
    "connected":false,
    "timestamp":1556649874716,
    "disconnectReason": "CONNECTION_LOST"
}
```

デバイス"mything2" は、POSIX 時間 1556649855046に接続されました (true)。

```
"connectivity": {
    "connected":true,
    "timestamp":1556649855046
}
```

タイムスタンプは、エポックからの経過をミリ秒単位で提供されるため、1556649855046は、2019 年 4 月 30 日火曜日の午後 6 時 44 分 15. 046秒 (UTC) を表します。

> **Important**
>
> デバイスが約 1 時間切断されていた場合、"timestamp"値と接続ステータスの"disconnectReason"値が無くなっている可能性があります。

制約と制限

以下は、AWS_Things の制約と制限です。

複合型のシャドウフィールド

シャドウフィールドは、シンプルな型 (配列を含まない JSON オブジェクトやシンプルな型で全体が構成されている配列など) である場合のみインデックス化されます。「シンプルな型」とは、文字列、数値、または、true あるいは false を意味します。例えば、次のシャドウステータスについては、フィールド "palette" の値は復号型の項目を含む配列であるためインデックス化されません。"colors" フィールドの値は、配列の各値が文字列であるため、インデックス化されます。

```
{
    "state": {
        "reported": {
            "switched": "ON",
            "colors": [ "RED", "GREEN", "BLUE" ],
            "palette": [
                {
                    "name": "RED",
                    "intensity": 124
                },
                {
                    "name": "GREEN",
                    "intensity": 68
                },
                {
                    "name": "BLUE",
                    "intensity": 201
                }
            ]
        }
    }
}
```

ネストされたシャドウフィールドの名前

ネストされたシャドウフィールドの名前は、ピリオド (.) で区切られた文字列として保存されます。たとえば、シャドウドキュメントがあるとします。

```
{
  "state": {
    "desired": {
      "one": {
        "two": {
          "three": "v2"
        }
      }
    }
  }
}
```

フィールドの名前 three は desired.one.two.three として保存されます。また、シャドウドキュメントがある場合、次のように保存されます。

```
{
  "state": {
    "desired": {
      "one.two.three": "v2"
    }
  }
}
```

両方が shadow.desired.one.two.three:v2 のクエリに一致します。ベストプラクティスとして、シャドウフィールドの名前にはピリオドを使用しないでください。

シャドウメタデータ

シャドウのメタデータセクションのフィールドはインデックス化されますが、これが行われるのは、シャドウの "state" セクションの対応するフィールドがインデックス化される場合のみです。(前の例では、シャドウのメタデータセクションの "palette" フィールドもインデックス化されません。)

未登録のデバイス

フリートインデックスは、接続 clientId がレジストリに登録されているモノの thingName と同じであるデバイスの接続ステータスをインデックス化します。

未登録シャドウ

UpdateThingShadow を使用して AWS IoT 、アカウントに登録されていないモノの名前を使用してシャドウを作成する場合、このシャドウのフィールドはインデックス化されません。これは、名前のないクラシックシャドウと名前付きシャドウの両方に適用されます。

数値

レジストリまたはシャドウデータがサービスによって数値として認識される場合、そのようにインデックス化されます。数値の範囲および比較演算子を含むクエリを作成できます (例: `"attribute.foo<5"` または `"shadow.reported.foo:[75 TO 80]"`)。数値として認識されるには、データの値が有効なリテラルタイプの JSON 番号である必要があります。値は、範囲 -2^53...2^53-1 の整数、オプションの指数関数表記を使用した倍精度浮動小数点、またはそのような値のみを含む配列の一部である必要があります。

Null 値

Null 値はインデックス化されません。

最大値

集計クエリのカスタムフィールドの最大数は 5 です。

集計クエリで要求されるパーセンタイルの最大数は 100 です。

Authorization

次のように、 AWS IoT ポリシーアクションでモノインデックスを Amazon リソースネーム (ARN) として指定できます。

Action	リソース
`iot:SearchIndex`	インデックス ARN (例: `arn:aws:iot:` *`your-aws-region your-aws-account`* `:index/AWS_Things`)。
`iot:DescribeIndex`	インデックス ARN (例: `arn:aws:iot:` *`your-aws-region`* `:index/AWS_Things`)。

> **Note**
>
> フリートインデックスに対してクエリを実行するアクセス許可があるユーザーは、フリート全体でモノのデータにアクセスできます。

モノのグループのインデックス作成の管理

AWS_ThingGroups は、すべてのモノのグループが含まれるインデックスです。このインデックスを使用して、グループ名、説明、属性、すべての親グループ名に基づいてグループを検索することができます。

モノのグループのインデックス作成の有効化

UpdateIndexingConfiguration API の thing-group-indexing-configuration 設定を使用して、AWS_ThingGroups インデックスを作成し、その設定を管理できます。GetIndexingConfiguration API を使用して、現在のインデックス作成設定を取得できます。

モノのグループのインデックス作成設定を更新するには、update-indexing-configuration CLI コマンドを使用します。

```
aws iot update-indexing-configuration --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

次のように、1 つのコマンドでモノとモノのグループのインデックス作成の両方の設定を更新することもできます。

```
aws iot update-indexing-configuration --thing-indexing-configuration
 thingIndexingMode=REGISTRY --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

次に示すのは、thingGroupIndexingMode の有効な値です。

VOFF

インデックスの作成/インデックスの削除がありません。

ON

AWS_ThingGroups インデックスを作成または設定します。

現在のモノおよびモノのグループのインデックス作成設定を取得するには、get-indexing-configuration CLI コマンドを実行します。

```
aws iot get-indexing-configuration
```

コマンドのレスポンスは次のようになります。

```
{
   "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "ON"
    }
}
```

グループインデックスの説明

現在の AWS_ThingGroups インデックスのステータスを取得するには、describe-index CLI コマンドを使用します。

```
aws iot describe-index --index-name "AWS_ThingGroups"
```

コマンドのレスポンスは次のようになります。

```
{
   "indexStatus": "ACTIVE",
   "indexName": "AWS_ThingGroups",
   "schema": "THING_GROUPS"
}
```

AWS IoT は、インデックスを初めて作成するときにインデックスを構築します。indexStatus が BUILDING の場合、インデックスに対してクエリを実行することはできません。

モノのグループのインデックスのクエリ

インデックスのデータにクエリを実行するには、search-index CLI コマンドを使用します。

```
aws iot search-index --index-name "AWS_ThingGroups" --query-string
 "thingGroupName:mythinggroup*"
```

Authorization

次のように、 AWS IoT ポリシーアクションでモノのグループのインデックスをリソース ARN として指定できます。

Action	リソース
`iot:SearchIndex`	インデックス ARN (例: `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`)。
`iot:DescribeIndex`	インデックス ARN (例: `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`)。

デバイス接続ステータスのクエリ

AWS IoT フリートインデックス作成は個々のデバイス接続クエリをサポートしているため、特定のデバイスの接続ステータスと関連メタデータを効率的に取得できます。この機能は、既存のフリート全体のインデックス作成およびクエリ機能を補完します。

仕組み

デバイス接続クエリのサポートは、最適化された単一デバイス接続ステータスの取得に使用することができます。この API は、最新のデバイス固有の接続情報への低レイテンシーで高スループットのアクセスを提供します。接続インデックス作成を有効にすると、標準クエリとして課金されるこのクエリ API にアクセスできます。詳細については、「AWS IoT Device Management 料金表」を参照してください。

機能

デバイス接続クエリのサポートにより、次のことが可能になります。

1. `thingName` を使用して、特定のデバイスの現在の接続状態 (接続または切断) をクエリする。
2. 以下を含む追加の接続メタデータを取得する。
   a. 切断の理由

b. 最新の接続または切断イベントのタイムスタンプ。

> **Note**
>
> フリートインデックスは、接続 `clientId` がレジストリに登録されているモノの `thingName` と同じであるデバイスの接続ステータスをインデックス化します。

利点

1. 低レイテンシー: 最新のデバイス接続状態を反映し、IoT Core からの接続状態の変化を反映するための低レイテンシーを提供します。IoT Core は、デバイスから切断リクエストを受信すると即時に、または切断リクエストを送信せずにデバイスが切断された場合に、デバイスを切断状態と見なします。IoT Core は、クライアントが切断されたと判断されるまで、設定されたキープアライブ時間の 1.5 倍を待機します。接続ステータス API は、通常 IoT Core がデバイスの接続状態の変更を判定してから 1 秒以内にこれらの変更を反映します。
2. 高スループット: デフォルトでは 350 トランザクション/秒 (TPS) をサポートしており、リクエストに応じてより高い値に調整できます。
3. データ保持: フリートインデックス作成 (FI) ConnectivityIndexing モードが有効で、モノが削除されていない場合に、イベントデータを無期限に保存します。接続インデックス作成を無効にすると、レコードは保持されません。

> **Note**
>
> この API の起動前に接続ステータスインデックス作成が有効になっている場合、フリートインデックス作成は API の起動後に接続ステータスの変更の追跡を開始し、それらの変更に基づいて更新されたステータスを反映します。

前提条件

デバイス接続クエリのサポートを使用するには:

1. AWS アカウントを設定する
2. 任意のリージョン AWS IoT Core の にデバイスをオンボードして登録する
3. 接続インデックス作成を使用して、フリートインデックス作成を有効にする

> **Note**
>
> 接続インデックス作成が既に有効になっている場合は、追加のセットアップは必要ありません。

詳細な手順については、「AWS IoT デベロッパーガイド」を参照してください。

例

```
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: リクエストで示されるデバイスの名前。これは、 AWS IoT Coreへの接続に使用される clientId とも一致します。
- disconnectReason: 切断の理由。接続されたデバイスでは NONE になります。
- connected: ブール値の true は、このデバイスが現在接続されていることを示します。
- timestamp: デバイスの直近の切断をミリ秒単位で表すタイムスタンプ。

```
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: リクエストで示されるデバイスの名前。これは、 AWS IoT Coreへの接続に使用される clientId とも一致します。
- disconnectReason: 切断の理由は CLIENT_INITIATED_DISCONNECT であり、クライアントが切断 AWS IoT Core することを示すことを示します。
- connected: ブール値の false は、このデバイスが現在切断されていることを示します。
- timestamp: デバイスの直近の切断をミリ秒単位で表すタイムスタンプ。

```
aws iot get-thing-connectivity-data --thing-name neverConnectedThing
```

```
{
   "connected": false,
   "disconnectReason": "UNKNOWN",
   "thingName": "neverConnectedThing"
}
```

- thingName: リクエストで示されるデバイスの名前。これは、 AWS IoT Coreへの接続に使用される clientId とも一致します。
- disconnectReason: 切断の理由。一度も接続されていないデバイスの場合、またはフリートインデックス作成に最後の切断理由が保存されていないデバイスの場合、「UNKNOWN」になります。
- connected: ブール値の false は、このデバイスが現在切断されていることを示します。
- timestamp: 一度も接続されていないデバイスの場合、またはフリートインデックス作成に最後のタイムスタンプが保存されていないデバイスの場合は、タイムスタンプは返されません。

集計データのクエリ

AWS IoT にはGetStatistics、デバイスフリートで集計データを検索GetBucketsAggregationできる 4 つの APIs (GetCardinality、、GetPercentiles、) が用意されています。

> **Note**
>
> 集約 API の値が不足しているか、予期しない値になっているという問題については、フリートインデックス作成トラブルシューティングガイドを参照してください。

GetStatistics

GetStatistics API および get-statistics CLI コマンドは、指定された集計フィールドのカウント、平均、合計、最小、最大、平方和、分散、および標準偏差を返します。

get-statistics CLI コマンドでは、以下のパラメータを使用します。

`index-name`

検索を実行するインデックスの名前。デフォルト値は `AWS_Things` です。

`query-string`

インデックスを検索するために使用されるクエリ。を指定"*"して、 内のすべてのインデックス付きモノの数を取得できます AWS アカウント。

`aggregationField`

(オプション) 集計するフィールド。このフィールドは、update-indexing-configuration を呼び出すときに定義される管理フィールドまたはカスタムフィールドである必要があります。集計フィールドを指定しない場合、`registry.version` が集計フィールドとして使用されます。

`query-version`

使用するクエリのバージョン。デフォルト値は `2017-09-30` です。

集計フィールドのタイプは、返される統計に影響します。

文字列値を使用した GetStatistics

文字列フィールドを集計する場合、GetStatistics を呼び出すと、クエリに一致する属性を持つデバイスの数が返されます。以下に例を示します。

```
aws iot get-statistics --aggregation-field 'attributes.stringAttribute'
            --query-string '*'
```

このコマンドは、`stringAttribute` という名前の属性を含むデバイスの数を返します。

```
{
  "statistics": {
    "count": 3
  }
}
```

ブール値を使用した GetStatistics

ブール値の集計フィールドを使用して `GetStatistics` を呼び出すと、次のようになります。

- AVERAGE は、クエリに一致するデバイスの割合です。
- MINIMUM は、次の規則に従って 0 または 1 です。
  - 集計フィールドのすべての値が `false` の場合、MINIMUM は 0 です。
  - 集計フィールドのすべての値が `true` の場合、MINIMUM は 1 です。
  - 集計フィールドの値が `false` と `true` の混合である場合、MINIMUM は 0 です。
- MAXIMUM は、次の規則に従って 0 または 1 です。
  - 集計フィールドのすべての値が `false` の場合、MAXIMUM は 0 です。
  - 集計フィールドのすべての値が `true` の場合、MAXIMUM は 1 です。
  - 集計フィールドの値が `false` と `true` の混合である場合、MAXIMUM は 1 です。
- SUM は、ブール値に相当する整数の合計です。
- COUNT は、クエリ文字列条件に一致し、有効な集計フィールド値を含むモノの数です。

数値を使用した GetStatistics

`GetStatistics` を呼び出してタイプ `Number` の集計フィールドを指定すると、`GetStatistics` は次の値を返します。

数

クエリ文字列条件に一致し、有効な集計フィールドの値を含むモモノの数。

平均

クエリに一致する数値の平均。

sum

クエリに一致する数値の合計。

minimum

クエリに一致する数値のうち最小の値。

maximum

クエリに一致する数値のうち最大値。

sumOfSquares

クエリに一致する数値の二乗の合計。

分散

クエリに一致する数値の分散。値の集合の分散は、集合の平均値からの各値の差の二乗の平均です。

stdDeviation

クエリに一致する数値の標準偏差。値のセットの標準偏差は、値がどの程度広がっているかを示す尺度です。

次の例は、数値カスタムフィールドを使用して get-statistics を呼び出す方法を示しています。

```
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
  }
}
```

数値集計フィールドの場合、フィールド値が最大倍精度値を超えた場合、統計値は空です。

GetCardinality

GetCardinality API および get-cardinality CLI コマンドは、クエリに一致する一意の値の概算数を返します。たとえば、バッテリー残量が 50% 未満のデバイスの数を調べるとします。

```
aws iot get-cardinality --index-name AWS_Things --query-string "batterylevel
          > 50" --aggregation-field "shadow.reported.batterylevel"
```

このコマンドは、バッテリー残量が 50% を超えるものの数を返します。

```
{
    "cardinality": 100
}
```

cardinality は、一致するフィールドがない場合でも、常に get-cardinality によって返されます。以下に例を示します。

```
aws iot get-cardinality --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_STR"
```

```
{
    "cardinality": 0
}
```

get-cardinality CLI コマンドでは、以下のパラメータを使用します。

`index-name`

検索を実行するインデックスの名前。デフォルト値は `AWS_Things` です。

`query-string`

インデックスを検索するために使用されるクエリ。を指定"*"して、 内のすべてのインデックス付きモノの数を取得できます AWS アカウント。

`aggregationField`

集計するフィールド。

`query-version`

使用するクエリのバージョン。デフォルト値は `2017-09-30` です。

GetPercentiles

GetPercentiles API および get-percentiles CLI コマンドは、クエリに一致する集計値をパーセンタイルグループにグループ化します。デフォルトのパーセンタイルのグループ化は 1,5,25,50,75,95,99 ですが、`GetPercentiles` を呼び出すときに独自のグループを指定することもできます。この関数は、指定された各パーセンタイルグループ (またはデフォルトのパーセンタイルグループ) の値を返します。パーセンタイルグループ「1」には、クエリに一致する値の約 1% に含まれる集計フィール

ド値が含まれます。パーセンタイルグループ「5」には、クエリに一致する値の約 5% で発生する集計フィールド値が含まれます。結果は近似値になります。クエリに一致する値が多いほど、パーセンタイルの値が正確になります。

次に、get-percentiles CLI コマンドを呼び出す例を示します。

```
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
        }
    ]
}
```

次のコマンドは、一致するドキュメントがない場合に get-percentiles から返される出力を示します。

```
aws iot get-percentiles --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_NUM"
```

```
{
    "percentiles": []
}
```

get-percentile CLI コマンドでは、以下のパラメータを使用します。

`index-name`

検索を実行するインデックスの名前。デフォルト値は `AWS_Things` です。

`query-string`

インデックスを検索するために使用されるクエリ。を指定`"*"`して、 内のすべてのインデックス付きモノの数を取得できます AWS アカウント。

`aggregationField`

集計するフィールド。`Number` 型である必要があります。

`query-version`

使用するクエリのバージョン。デフォルト値は `2017-09-30` です。

`percents`

(オプション) このパラメータを使用して、カスタムのパーセンタイルグループを指定できます。

GetBucketsAggregation

GetBucketsAggregationAPI およびget-buckets-aggregationCLIコマンドは、バケットのリストとクエリ文字列条件に一致するものの合計数を返します。

以下の例は、get-buckets-aggregation CLIコマンドの呼び出し方を示しています。

```
aws iot get-buckets-aggregation --query-string '*' --index-name AWS_Things --
aggregation-field 'shadow.reported.batterylevelpercent' --buckets-aggregation-type
 'termsAggregation={maxBuckets=5}'
```

このコマンドは、次のサンプル出力を返します。

```
{
    "totalCount": 20,
    "buckets": [
        {
            "keyValue": "100",
            "count": 12
        },
        {
            "keyValue": "90",
            "count": 5
        },
        {
            "keyValue": "75",
            "count": 3
        }
    ]
}
```

get-buckets-aggregation CLI コマンドでは、以下のパラメータを使用します。

`index-name`

検索を実行するインデックスの名前。デフォルト値は `AWS_Things` です。

`query-string`

インデックスを検索するために使用されるクエリ。を指定"*"して、 内のすべてのインデックス付きモノの数を取得できます AWS アカウント。

`aggregation-field`

集計するフィールド。

`buckets-aggregation-type`

レスポンスの形の基本的な制御と実行するバケット集計タイプ。

Authorization

次のように、 AWS IoT ポリシーアクションでモノのグループのインデックスをリソース ARN として指定できます。

Action	リソース
`iot:GetStatistics`	インデックス ARN (例: `arn:aws:iot:` *`your-aws-region`* `:index/AWS_Things` または `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`)。

クエリ構文

フリートインデックス作成では、クエリ構文を使用してクエリを指定します。

サポートされている機能

クエリ構文では次の機能がサポートされています。

- 規約とフレーズ
- フィールド検索
- プレフィックス検索
- 範囲検索
- ブール演算子 AND、OR、NOT および -。検索結果から項目を除外するにはハイフンを使用します (例: `thingName:(tv* AND -plasma)`)。
- グループ化
- フィールドのグループ分け

- 特殊文字のエスケープ (\ など)
- 先頭ワイルドカードの使用は、1 つのクエリにつき 1 つのクエリ用語に制限されます。例えば、同一のクエリ内で、`thingName:*my` と `thingGroupNames:*echo` の両方を検索することはできません。先頭ワイルドカードを含むクエリの最大クエリ長は 100 文字です。

> **Note**
>
> フリートインデックス作成クエリ API の制限が 1 秒あたり 30 リクエストを超えるアカウントは、先行ワイルドカードクエリ条件を使用する場合にスロットリングの対象となります。この 30 requests-per-secondの制限は、すべてのフリートインデックス作成クエリ APIs のアカウントレベルで適用されます。

サポートされていない 機能

クエリ構文では次の機能がサポートされていません。

- 正規表現
- ブースト
- ランキング
- あいまい検索
- 近接検索
- ソート
- 集約
- 特殊文字: `、@、#、%、\、/、'、;、および ,。, はジオクエリでのみサポートされています。

注意事項

クエリ言語に関する注意事項がいくつかあります。

- デフォルトの演算子は AND です。`"thingName:abc thingType:xyz"` のクエリは `"thingName:abc AND thingType:xyz"` と同等です。
- フィールドが指定されていない場合、 はすべてのレジストリ、Device Shadow、および Device Defender フィールドで用語 AWS IoT を検索します。
- すべてのフィールド名で大文字と小文字が区別されます。

- 検索では大文字と小文字は区別されません。単語は、Java の `Character.isWhitespace(int)` で定義されているように空白文字で区切られます。
- Device Shadow データ (名前のないシャドウおよび名前付きシャドウ) のインデックス作成には、報告されたセクション、目的のセクション、デルタおよびメタデータのセクションが含まれます。
- デバイスシャドウおよびレジストリのバージョンは検索不可能ですが、レスポンスには存在します。
- クエリの語の最大数は 12 です。
- 特殊文字 , は、ジオクエリでのみサポートされています。

モノのクエリの例

クエリ構文を使用して、クエリ文字列にクエリを指定します。クエリは、`SearchIndex` API に渡されます。次の表に、クエリ文字列の例を示します。

クエリ文字列	結果
`abc`	レジストリ、シャドウ (従来の名前のないシャドウと名前付きシャドウ)、またはデバイスディフェンダー違反フィールドで「abc」をクエリします。
`thingName:myThingName`	「myThingName」という名前のモノをクエリします。
`thingName:my*`	「my」で始まる名前のモノをクエリします。
`thingName:*my`	「my」で終わる名前のモノをクエリします。先頭ワイルドカードの使用は、1 つのクエリにつき 1 つのクエリ用語に制限されます。先頭ワイルドカードを含むクエリの最大クエリ長は 100 文字です。
`thingName:*my*`	部分文字列「my」を含むモノをクエリします。先頭ワイルドカードの使用は、1 つのクエリにつき 1 つのクエリ用語に制限されます。先頭ワイルドカードを含むクエリの最大クエリ長は 100 文字です。
`thingName:ab?`	「aba」、「abb」、「abc」など、名前に「ab」と追加で 1 つの文字を加えたモノをクエリします。

クエリ文字列	結果
`thingTypeName:aa`	「aa」タイプに関連付けられているモノをクエリします。
`thingGroupNames:a`	親モノグループまたは請求グループ名「a」を持つモノのクエリ。
`thingGroupNames:a*`	パターン「a*」に一致する親モノグループまたは請求グループ名を持つモノのクエリ。
`attributes.myAttribute:75`	値が 75 の「myAttribute」という属性を持つモノをクエリします。
`attributes.myAttribute:[75 TO 80]`	値が数値範囲 (75—80) 内にある「myAttribute」という属性を持つモノをクエリします。
`attributes.myAttribute:{75 TO 80]`	値が数値範囲 (>75 および <=80) 内にある「myAttribute」という属性を持つモノをクエリします。
`attributes.serialNumber:["abcd" TO "abcf"]`	値が英数字の文字列範囲内にある「serialNumber」という属性を持つモノをクエリします。このクエリは、値が「abcd」、「abce」、または「abcf」の「serialNumber」属性を持つモノを返します。
`attributes.myAttribute:i*t`	値が「i」で、その後に任意の文字数、そして「t」が続く「myAttribute」という属性を持つモノをクエリします。
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	ブール式を使用して、用語の組み合わせとなるモノをクエリします。このクエリは、値が「abc」の属性「attr1」、値が 5 未満の属性「attr2」、および値が 10 以下の属性「attr3」のモノを返します。
`shadow.hasDelta:true`	デルタ要素を持つ名前のないシャドウを持つモノをクエリします。
`NOT attributes.model:legacy`	「model」という属性が「legacy」でないモノをクエリします。

クエリ文字列	結果
`shadow.reported.stats.battery:{70 TO 100} (v2 OR v3) NOT attributes.model:legacy`	以下を持つモノをクエリします。 • 70 ~ 100 の値を持つ Thing Shadow の `stats.battery` 属性。 • モノの名前、タイプ名、または属性値で発生するテキスト「v 2」または「v 3」。 • モノの `model` 属性は「legacy」に設定されません。
`shadow.reported.myvalues:2`	Shadow の reported セクションの `myvalues` 配列の値に 2 が含まれているモノをクエリします。
`shadow.reported.location:* NOT shadow.desired.stats.battery:*`	以下を持つモノをクエリします。 • Shadow の `location` セクションに `reported` 属性が存在します。 • Shadow の `desired` セクションに `stats.battery` 属性が存在しません。
`shadow.name.<shadowName>.hasDelta:true`	指定された名前とデルタ要素のシャドウを持つモノをクエリします。
`shadow.name.<shadowName>.desired.filament:*`	指定された名前および目的のフィラメントプロパティのシャドウを持つモノをクエリします。
`shadow.name.<shadowName>.reported.location:*`	名前付きシャドウの報告されたセクションに `location` 属性が存在し、指定された名前のシャドウを持つモノをクエリします。
`connectivity.connected:true`	接続されているすべてのデバイスに対するクエリです。
`connectivity.connected:false`	切断されたすべてのデバイスに対するクエリです。

クエリ文字列	結果
`connectivity.connected:true AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	特定の範囲 (1557651600000 ~ 1557867600000) 内の接続タイムスタンプを持つ、接続されているすべてのデバイスに対するクエリです。タイムスタンプはエポックからのミリ秒単位で指定されます。
`connectivity.connected:false AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	特定の範囲 (1557651600000 ~ 1557867600000) 内の切断タイムスタンプを持つ、切断されているすべてのデバイスに対するクエリです。タイムスタンプはエポックからのミリ秒単位で指定されます。
`connectivity.connected:true AND connectivity.timestamp > 1557651600000`	特定の範囲 (1557651600000 より大) 内の接続タイムスタンプを持つ、接続されているすべてのデバイスに対するクエリです。タイムスタンプはエポックからのミリ秒単位で指定されます。
`connectivity.connected:*`	接続情報があるすべてのデバイスに対するクエリです。
`connectivity.disconnectReason:*`	connectivity.disconnectReason のあるすべてのデバイスに対するクエリです。
`connectivity.disconnectReason:CLIENT_INITIATED_DISCONNECT`	CLIENT_INITIATED_DISCONNECTが原因で切断されたすべてのデバイスのクエリです。
`deviceDefender.violationCount:[0 TO 100]`	数値範囲 (0—100) 内の Device Defender 違反のカウント値を持つモノをクエリします。
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.inViolation:true`	セキュリティプロファイル `device-SecurityProfile` で定義されているように、動作 `disconnectBehavior` に違反しているモノをクエリします。inViolation:false は有効なクエリではないことに注意してください。

クエリ文字列	結果
deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationValue.number>2	最新の違反のイベント値が 2 より大きく、セキュリティプロファイル device-SecurityProfile で定義されているように動作 disconnectBehavior に違反しているモノをクエリします。
deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationTime>1634227200000	指定したエポックタイムの後に最新の違反イベントがあり、セキュリティプロファイル device-SecurityProfile で定義されているように動作 disconnectBehavior に違反しているモノをクエリします。
shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km	47.6204,-122.3491 の座標から 15.5 km の半径距離内にあるモノをクエリします。このクエリ文字列は、位置データが名前付きシャドウに保存されている場合に適用されます。
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km	47.6204,-122.3491 の座標から 15.5 km の半径距離内にあるモノをクエリします。このクエリ文字列は、位置データがクラシックシャドウに保存されている場合に適用されます。

モノのグループのクエリの例

クエリは、クエリ構文を使用してクエリ文字列に指定され、SearchIndex API に渡されます。次の表に、クエリ文字列の例を示します。

クエリ文字列	結果
abc	任意のフィールドでの「abc」のクエリ
thingGroupName:myGroupThingName	「myGroupThingName」という名前のモノのグループをクエリします。
thingGroupName:my*	「my」で始まる名前のモノのグループをクエリします。

クエリ文字列	結果
`thingGroupName:ab?`	名前に「ab」と 1 つの追加文字 (例：「aba」、「abb」、「abc」など)を加えたモノのグループをクエリします)。
`attributes.myAttribute:75`	値が 75 の「myAttribute」という属性を持つモノのグループをクエリします。
`attributes.myAttribute:[75 TO 80]`	値が数値範囲 (75–80) の「myAttribute」という属性を持つモノのグループをクエリします。
`attributes.myAttribute:[75 TO 80]`	値が数値範囲 (>75 および =80) の「myAttribute」という属性を持つモノのグループをクエリします。
`attributes.myAttribute: ["abcd" TO "abcf"]`	値が英数字の文字列範囲内にある「myAttribute」という属性を持つモノのグループをクエリします。このクエリは、値が「abcd」、「abce」、または「abcf」の「serialNumber」属性を持つモノのグループを返します。
`attributes.myAttribute:i*t`	「myAttribute」という属性を持つモノのグループを検索します。値は「i」で、その後に任意の文字数が続き、その後に「t」が続きます。
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	ブール式を使用して、用語の組み合わせとなるモノのグループをクエリします。このクエリは、値が「abc」の属性「attr1」、値が 5 未満の属性「attr2」、および値が 10 以下の属性「attr3」をモノのグループを返します。
`NOT attributes.myAttribute:cde`	「myAttribute」という属性が「cde」でないモノのグループをクエリします。
`parentGroupNames:(myParentThingGroupName)`	親グループ名が「myParentThingGroupName」に一致するモノのグループのクエリ。
`parentGroupNames:(myParentThingGroupName OR myRootThingGroupName)`	親グループ名が「myParentThingGroupName」または「myRootThingGroupName」に一致するモノのグループのクエリ。

クエリ文字列	結果
`parentGroupNames:(myParentThingGroupNa*)`	親グループ名の先頭が「myParentThingGroupNa」のモノのグループのクエリ。

位置データのインデックス作成

AWS IoT フリートのインデックス作成を使用して、デバイスの最後に送信された位置データのインデックスを作成し、ジオクエリを使用してデバイスを検索できます。この機能は、位置追跡や近接検索などのデバイスモニタリングと管理のユースケースを解決します。位置インデックス作成は、他のフリートインデックス作成機能と同様に機能し、モノのインデックス作成で を指定する追加の設定で機能します。

一般的なユースケースには、目的の地理的境界内に位置するデバイスを検索および集約する、インデックス作成されたデータソースから、デバイスのメタデータと状態に関連するクエリ用語を使用して、位置固有のインサイトを取得する、結果を特定の地理的エリアにフィルタリングするなどの詳細なビューを提供し、フリートモニタリングマップ内のレンダリングの遅延を減らし、最後に報告されたデバイスの位置を追跡する、目的の境界制限外のデバイスを特定し、フリートメトリクスを使用してアラームを生成するなどがあります。位置インデックス作成とジオクエリを開始するには、「???」を参照してください。

サポートされているデータ形式

AWS IoT フリートインデックス作成は、次の位置データ形式をサポートしています。

1. 座標参照システムのよく知られたテキスト表現

   地理情報 - 座標参照システム形式のよく知られたテキスト表現の形式に従う文字列。例として、`"POINT(long lat)"` を使用できます。

2. 座標を表す文字列

   `"latitude, longitude"` または `"longitude, latitude"` の形式の文字列。`"longitude, latitude"` を使用する場合は、`geoLocations` で `order` も指定する必要があります。例として、`"41.12,-71.34"` を使用できます。

3. lat (緯度) 、lon (経度) キーのオブジェクト

   この形式は、クラシックシャドウと名前付きシャドウに適用されます。サポートされるキー: `lat`、`latitude`、`lon`、`long`、`longitude`。例として、`{"lat": 41.12, "lon": -71.34}` を使用できます。

4. 座標を表す配列

   形式が `[lat,lon]` または `[lon,lat]` の配列。GeoJSON の座標 (クラシックシャドウと名前付きシャドウに適用) と同じ形式 `[lon,lat]` を使用する場合は、`geoLocations` で `order` も指定する必要があります。

   例としては、次のようなものがあります。

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

位置データのインデックス作成方法

次の手順では、位置データのインデックス作成の設定を更新し、ジオクエリを使用してデバイスを検索する方法を示します。

1. 位置情報データが保存されている場所を把握する

   現在、フリートのインデックス作成は、クラシックシャドウまたは名前付きシャドウに保存されている位置データのインデックス作成をサポートしています。

2. サポートされている位置データ形式を使用する

   位置データ形式がサポートされているデータ形式 のいずれかに従っていることを確認します。

3. インデックス作成の設定の更新

   少なくとも、モノ (レジストリ) インデックス作成の設定を有効にします。また、位置データを含むクラシックシャドウまたは名前付きシャドウでインデックス作成を有効にする必要があります。モノのインデックス作成を更新するときは、インデックス作成の設定に位置情報データを含める必要があります。

4. ジオクエリを作成して実行する

   ユースケースに応じて、ジオクエリを作成して実行し、デバイスを検索します。作成するジオキューは、クエリ構文に従う必要があります。??? にいくつかの例があります。

モノのインデックス作成を設定する

位置データのインデックスを作成するには、インデックス作成の設定を更新し、位置データを含める必要があります。位置データが保存されている場所に応じて、手順に従ってインデックス作成の設定を更新します。

クラシックシャドウに保存されている位置データ

位置データがクラシックシャドウに保存されている場合は、thingIndexingMode を REGISTRY_AND_SHADOW に設定し、filter の geoLocations フィールド (name および order) に位置データを指定する必要があります。

次のモノのインデックス作成の設定例では、位置データパス shadow.reported.coordinates を name、 LonLat を order として指定します。

```
{
 "thingIndexingMode": "REGISTRY_AND_SHADOW",
 "filter": {
  "geoLocations": [
   {
    "name": "shadow.reported.coordinates",
    "order": "LonLat"
```

```
    }
   ]
  }
 }
```

- thingIndexingMode

  インデックス作成モードは、レジストリまたはシャドウがインデックス作成されているかどうかを制御します。thingIndexingMode が OFF に設定されている場合、モノのインデックス作成は無効になります。

  クラシックシャドウに保存されている位置データをインデックス作成するには、thingIndexingMode を REGISTRY_AND_SHADOW に設定する必要があります。詳細については、「???」を参照してください。

- filter

  インデックス作成フィルターは、名前付きシャドウと位置情報データに追加の選択を提供します。詳細については、「???」を参照してください。

- geoLocations

  インデックス作成のために選択した位置情報ターゲットのリスト。インデックス作成の位置情報ターゲットのデフォルトの最大数は 1 です。制限の引き上げをリクエストするには、「AWS IoT Device Management クォータ」を参照してください。

- name

  位置情報ターゲットフィールドの名前。name の値は、シャドウの位置データパス (shadow.reported.coordinates) にすることができます。

- order

  位置情報ターゲットフィールドの順序。有効な値: LatLon および LonLat。LatLon は緯度と経度を意味し、LonLat は経度と緯度を意味します。このフィールドはオプションです。デフォルト値は LatLon です。

名前付きシャドウに保存されている位置データ

位置データが名前付きシャドウに保存されている場合は、namedShadowIndexingMode を ONに設定し、名前付きシャドウ名 (複数可) を filter の namedShadowNames フィールドに追加し、filter の geoLocations フィールドに位置データパスを指定します。

次のモノのインデックス作成の設定例では、位置データパス shadow.name.namedShadow1.reported.coordinates を name、 LonLat を order として指定します。

```
{
 "thingIndexingMode": "REGISTRY",
 "namedShadowIndexingMode": "ON",
 "filter": {
  "namedShadowNames": [
   "namedShadow1"
  ],
  "geoLocations": [
   {
    "name": "shadow.name.namedShadow1.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- thingIndexingMode

  インデックス作成モードは、レジストリまたはシャドウがインデックス作成されているかどうかを制御します。thingIndexingMode が OFF に設定されている場合、モノのインデックス作成は無効になります。

  名前付きシャドウに保存されている位置データのインデックスを作成するには、thingIndexingMode を REGISTRY (または REGISTRY_AND_SHADOW) に設定する必要があります。詳細については、「???」を参照してください。

- filter

  インデックス作成フィルターは、名前付きシャドウと位置情報データに追加の選択を提供します。詳細については、「???」を参照してください。

- geoLocations

  インデックス作成のために選択した位置情報ターゲットのリスト。インデックス作成の位置情報ターゲットのデフォルトの最大数は 1 です。制限の引き上げをリクエストするには、「AWS IoT Device Management クォータ」を参照してください。

- name

位置情報ターゲットフィールドの名前。name の値は、シャドウの位置データパス (shadow.name.namedShadow1.reported.coordinates) にすることができます。

- order

  位置情報ターゲットフィールドの順序。有効な値: LatLon および LonLat。LatLon は緯度と経度を意味し、LonLat は経度と緯度を意味します。このフィールドはオプションです。デフォルト値は LatLon です。

ジオクエリの例

位置データのインデックス作成の設定が完了したら、ジオクエリを実行してデバイスを検索します。ジオクエリを他のクエリ文字列と組み合わせることもできます。詳細については、「???」および「???」を参照してください。

クエリの例 1

この例では、位置データが名前付きシャドウ gps-tracker に保存されていることを前提としています。このコマンドの出力は、座標 (47.6204,-122.3491) を持つ中心点から 15.5 km の半径距離内にあるデバイスのリストです。

```
aws iot search-index --query-string \
"shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

クエリの例 2

この例では、位置データがクラシックシャドウに保存されていることを前提としています。このコマンドの出力は、座標 (47.6204,-122.3491) を持つ中心点から 15.5 km の半径距離内にあるデバイスのリストです。

```
aws iot search-index --query-string \
"shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

クエリの例 3

この例では、位置データがクラシックシャドウに保存されていることを前提としています。このコマンドの出力は、接続されておらず、座標 (47.6204、-122.3491) を持つ中心点から 15.5 km の半径距離外にあるデバイスのリストです。

```
aws iot search-index --query-string \
"connectivity.connected:false AND (NOT
 shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km)"
```

入門チュートリアル

このチュートリアルでは、フリートのインデックス作成を使用して位置データのインデックス作成方法を示します。簡単にするために、デバイスを表すモノを作成し、位置データを名前付きシャドウに保存し、位置インデックス作成のモノインデックス設定を更新し、サンプルジオクエリを実行して半径境界内のデバイスを検索します。

このチュートリアルの完了には 15 分ほどかかります。

このトピックの内容

- 前提条件
- モノとシャドウを作成する
- モノのインデックス作成を設定する
- ジオクエリの実行

前提条件

- AWS CLI の最新バージョンをインストールします。
- 位置インデックス作成とジオクエリ、モノのインデックス作成の管理、およびクエリ構文を理解しておいてください。

モノとシャドウを作成する

デバイスを表すモノと、その位置データを保存する名前付きシャドウを作成します (座標 47.61564、-122.33584)。

1. 次のコマンドを実行して、Bike-1 という名前のバイクを表すモノを作成します。を使用してモノを作成する方法の詳細については AWS CLI、「 AWS CLIリファレンス」の「Create-thing」を参照してください。

   ```
   aws iot create-thing --thing-name "Bike-1" \
   --attribute-payload '{"attributes": {"model":"OEM-2302-12", "battery":"35",
    "acqDate":"06/09/23"}}'
   ```

このコマンドの出力は以下のようになります。

```
{
    "thingName": "Bike-1",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/Bike-1",
    "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df"
}
```

2. 次のコマンドを実行して、Bike-1 の位置データ (座標 47.61564、-122.33584) を保存する名前付きシャドウを作成します。を使用して名前付きシャドウを作成する方法の詳細については AWS CLI、「 AWS CLIリファレンス」のupdate-thing-shadow」を参照してください。

```
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" \
```

このコマンドでは、出力が生成されません。作成した名前付きシャドウを表示するには、list-named-shadows-for-thing CLI コマンドを実行します。

```
aws iot-data list-named-shadows-for-thing --thing-name Bike-1
```

このコマンドの出力は以下のようになります。

```
{
    "results": [
        "Bike1-shadow"
    ],
    "timestamp": 1699574309
}
```

モノのインデックス作成を設定する

位置データのインデックスを作成するには、モノのインデックス作成設定を更新して、位置データを含める必要があります。このチュートリアルでは、位置情報データは名前付きシャドウに保存されるため、thingIndexingMode を REGISTRY (最小要件で) に設定し、namedShadowIndexingMode

を ON に設定して、位置情報データを設定に追加します。この例では、名前付きシャドウの名前とシャドウの位置データパスを filter に追加する必要があります。

1. コマンドを実行して、位置インデックスの作成のインデックス作成設定を更新します。

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

   コマンドでは、出力が生成されません。更新が完了するまで少し時間を要する場合があります。ステータスを確認するには、describe-index CLI コマンドを実行します。indexStatus が ACTIVE と表示された場合は、モノのインデックスの更新が完了しました。

2. コマンドを実行して、インデックス作成の設定を確認します。この手順は省略可能です。

```
aws iot get-indexing-configuration
```

   出力は次のようになります。

```
{
    "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"
```

```
    }
}
```

ジオクエリの実行

これで、モノのインデックスの作成設定を更新して、位置データを含めました。ジオクエリを作成して実行し、目的の検索結果を取得できるかどうかを確認してください。ジオクエリは、クエリ構文に従う必要があります。??? に有用なジオクエリの例があります。

次のコマンド例では、ジオクエリ `shadow.name.Bike1-shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km` を使用して、座標 (47.6204,-122.3491) を使用して中心点から 15.5 km の半径距離内にあるデバイスを検索します。

```
aws iot search-index --query-string "shadow.name.Bike1-
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

座標「lat」: 47.6153、「lon」: -122.3333 にデバイスがあり、これはは中心点から 15.5 km の距離以内となるため、このデバイス (Bike-1) を出力で確認できます。出力は次のようになります。

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

詳細については、「???」を参照してください。

フリートメトリクス

フリートメトリクスは、デバイスのデータのインデックス作成、検索、集計を可能にするマネージドサービスであるフリートインデックス作成の機能です AWS IoT。フリートメトリクスを使用すると、CloudWatch でフリートデバイスの集約状態を経時的にモニタリングできます。これには特定期間におけるフリートデバイスの切断率、または平均的なバッテリーレベルの変化の確認などが含まれます。

フリートメトリクスを使用することによって、集計クエリが構築できます。クエリからの結果は、傾向を分析し、アラームを作成するためのメトリクスとして CloudWatch に継続的に出力されます。モニタリングタスクでは、異なるタイプ([Statistics](統計）、[Cardinality]（濃度）、および[Percentile](パーセンタイル)）の集計クエリが指定できます。将来の再使用のために、フリートメトリクスを作成するためのすべての集計クエリを保存することができます。

入門チュートリアル

このチュートリアルでは、センサーの温度を監視し、潜在的な異常を検出するために、フリートメトリクスを作成します。フリートメトリクスを作成するときは、華氏80度を超える温度を持つセンサーの数を検出する集計クエリを定義します。クエリが 60 秒ごとに実行され、クエリ結果が CloudWatch に出力されるように指定します。そうすると、高温リスクの可能性があるセンサーの数を確認し、アラームを設定する事ができます。このチュートリアルを完了するには、AWS CLIを使用します。

このチュートリアルの学習内容は次のとおりです。

- セットアップする
- フリートメトリクスを作成します
- CloudWatch でメトリクスを表示する
- リソースをクリーンアップする

このチュートリアルの完了には 15 分ほどかかります。

前提条件

- AWS CLIの最新バージョンをインストールします
- 集計データのクエリについて理解する
- Amazon CloudWatch メトリクスの使用について理解する

セットアップする

フリートメトリクスを使用するには、フリートインデックス作成を有効にします。指定されたデータソースと関連付けられた設定を持つモノ、またはモノのグループに対してフリートのインデックス作成を有効にするには、「モノのインデックス作成の管理」と「モノのグループのインデックス作成の管理」にある手順に従ってください。

をセットアップするには

1. 次のコマンドを実行して、フリートインデックス作成を有効にし、検索元のデータソースを指定します。

   ```
   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" \
   ```

   前の CLI コマンドの例は、AWS_Things インデックスを使用して、フリートインデックス作成に、レジストリーデータ、シャドウデータ、およびモノの接続ステータスの検索サポートをさせます。

   設定の変更が完了するまで数分かかることがあります。フリートメトリクスを作成する前に、フリートインデックス作成が有効になっていることを確認してください。

   フリートインデックスが有効になっているかどうかを確認するには、次の CLI コマンドを実行します：

   ```
   aws --region us-east-1 iot describe-index --index-name "AWS_Things"
   ```

   詳細については、「モノのインデックス作成を有効にする」をご覧ください。
2. 次のbashスクリプトを実行して、10個のものを作成し、それらを記述します。

   ```
   # 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
```

このスクリプトは、10 個のセンサーを表す 10 個のものを作成します。それぞれのモノは、次の表で説明されているようtemperature、rackId、 およびstateNormalの属性を持っています。

属性	データ型	説明
temperature	Number	温度値 (華氏)
rackId	String	センサーを含むサーバーラックの ID
stateNormal	ブール値	センサーの温度値が正常かどうか

このスクリプトの出力には、10 個の JSON ファイルが含まれています。JSON ファイルのうちの 1 つが次のようになります。

```
{
    "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"
}
```

詳細については、「モノの作成」を参照してください。

フリートメトリクスを作成します

フリートメトリクスを作成します

1. 次のコマンドを実行して、*high_temp_FM*という名前のフリーメトリクスを作成します。CloudWatch で華氏 80 度を超える温度を持つセンサーの数を監視するフリートメトリクスを作成します。

```
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
```

–-メトリクス名

データ型 – 文字列 `--metric-name`パラメーターは、フリートメトリクス名を指定します。この例では、high_temp_FMという名前のフリートメトリクス を作成しています。

--query <string>

データ型 – 文字列 `--query-string`パラメーターは、クエリ文字列を指定します。この例でクエリ文字列は、TempSensor で始まる名前が付けられたすべてのモノと、温度が華氏 80 度を超えるすべてのモノをクエリすることを意味します。詳細については、「クエリ構文」を参照してください。

–-期間

データ型: 整数 `--period`パラメーターは、集計データを取得する時間を秒単位で指定します。この例では、作成するフリートメトリクスが 60 秒ごとに集計データを取得するように指定します。

–aggregation-field

データ型:文字列 `--aggregation-field`パラメーターは、評価する属性を指定します。この例では、温度属性を評価します。

–-aggregation-type

`--aggregation-type`パラメーターは、フリートメトリクスに表示する統計概要を指定します。モニタリングタスクでは、異なる集計タイプ ([Statistics] (統計)、[Cardinality] (濃度)、および [Percentile] (パーセンタイル)の集計クエリプロパティをカスタマイズできます。この

例では、集計タイプ [統計] に [数] を指定して、クエリに一致する属性を持つデバイス、つまり、TempSensor で始まる名前が付けられており、温度が華氏 80 度を超えるデバイスの数を返します。詳細については、「集計データのクエリ」をご覧ください。

このコマンドの出力は以下のようになります。

```
{
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "metricName": "high_temp_FM"
}
```

> **Note**
>
> CloudWatch でデータポイントが表示されるまでに時間がかかることがあります。

フリートメトリクスの作成方法については、「フリートメトリクスの管理」を参照してください。。

フリートメトリックスを作成できない場合は、「フリートメトリクスのトラブルシューティング」を参照してください。

2. (オプション) 次のコマンドを実行して、high_temp_FMという名前のフリートメトリクスを記述します。

```
aws iot describe-fleet-metric --metric-name "high_temp_FM"
```

このコマンドの出力は以下のようになります。

```
{
    "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"
}
```

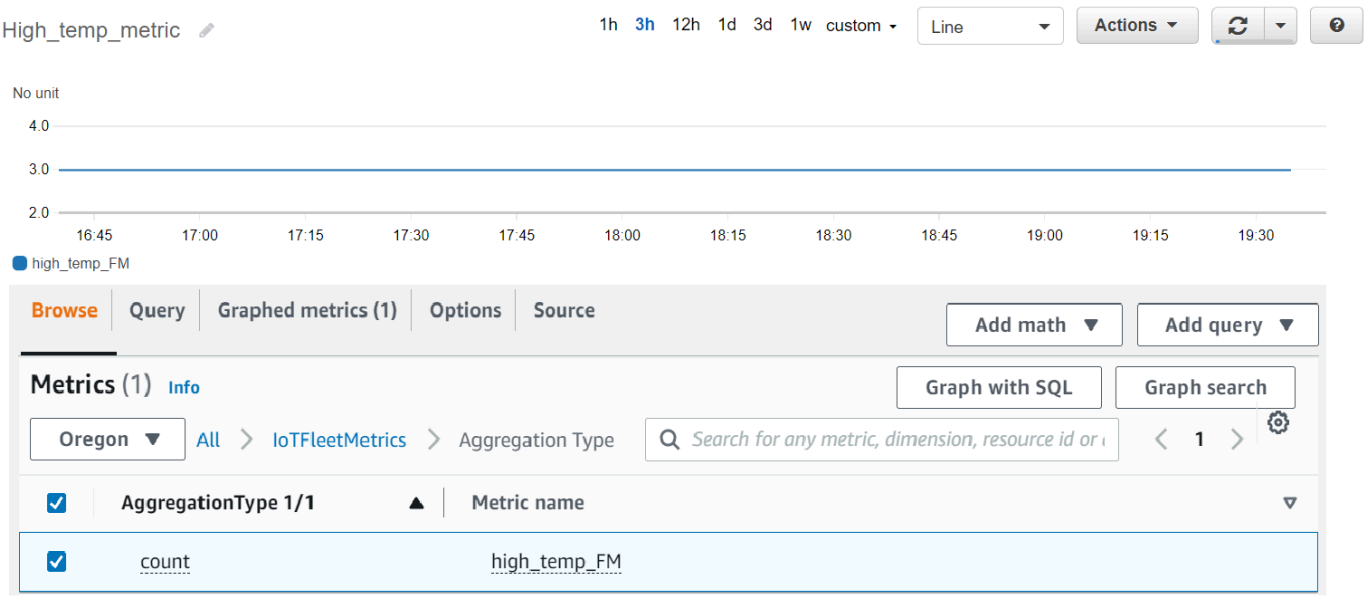
CloudWatch でフリートメトリクスを表示する

フリートメトリクスを作成したら、CloudWatch コンソールでメトリクスデータを表示できます。このチュートリアルでは、TempSensor で始まる名前が付けられており、温度が華氏 80 度を超えるセンサーの数を示すメトリクスが表示されます。

CloudWatch でデータポイントを表示するには

1. CloudWatch コンソールの https://console.aws.amazon.com/cloudwatch/ を開いてください。
2. 左側の [CloudWatch] メニューで、[メトリクス] をクリックしてサブメニューを展開し、[すべてのメトリクス] を選びます。これにより、上半分はグラフが表示され、下半分は4つのタブ付きセクションが表示されたページが開きます。
3. 最初のタブ付きセクション[すべてのメトリクス] には、グループで表示できるすべてのメトリクスが一覧表示されます。[IoTFleetMetrics] を選びます。これにはすべてのフリートメトリクスが含まれます。
4. [All metrics] (すべてのメトリクス) タブの [Aggregation type] (集計タイプ) セクションで、[Aggregation type] (集計タイプ) を選択して、作成したすべてのフリートメトリクスを表示します。
5. フリート指標を選択して、[Aggregation type](集計タイプ)セクションの左側にグラフを表示します。[メトリクス名] の左に、値　####が表示されます。これは、このチュートリアルの「フリートメトリクスを作成する」セクションで指定された集約タイプの値です。
6. [All metrics](すべてのメトリクス) タブの右にある[Graphed metrics](グラフ化したメトリクス) という名前の2番目のタブを選び、前のステップで選んだフリートメトリクスを表示します。

   以下の通り、華氏 80 度以上の温度を持つセンサーの数を表示するグラフを確認できます。



> **Note**
>
> CloudWatchの[Period] (期間) 属性のデフォルトは 5 分です。これは、CloudWatch に表示されているデータポイント間の時間間隔です。必要に応じて、[Period](期間)設定を変更する事ができます。

7. (オプション) メトリクスアラームを設定できます。

   1. 左側の CloudWatch メニューで、[アラーム] をクリックしてサブメニューを展開し、[すべてのアラーム] を選びます。
   2. [Alarms](アラーム) ページで、右上のコーナーの [Create alarm] (アラームを作成する)を選びます。コンソールの[Create alarm](アラームを作成する)の指示に従って、必要に応じてアラームを作成します。詳細については、「Using Amazon CloudWatch alarms」(Amazon CloudWatch アラームを使用する) を参照してください。

詳細については、[Using Amazon CloudWatch metrics](Amazon CloudWatch メトリクスを使う) を参照してください。

CloudWatch でデータポイントが表示されない場合は、[Troubleshooting fleet metrics](フリートメトリクスのトラブルシューティング) を参照してください。。

クリーンアップ

フリートメトリクスを削除するには

delete-fleet-metricCLI コマンドを使用して、フリートメトリクスを削除します。

high_temp_FM という名前のフリートメトリクスを削除するには、次のコマンドを実行します。

```
aws iot delete-fleet-metric --metric-name "high_temp_FM"
```

モノをクリーンアップするには

delete-thing CLIコマンドを使用してモノを削除します。

作成した 10 個のモノを削除するには、次のスクリプトを実行します。

```
# 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
```

CloudWatch でメトリクスをクリーンアップするには

CloudWatch は、メトリクスの削除をサポートしません。メトリクスは、保持スケジュールに基づいて期限切れになります。詳細については、「Amazon CloudWatch メトリクスを使用する」を参照してください。

フリートのメトリクスの管理

このトピックでは、 AWS IoT コンソールと AWS CLI を使用してフリートメトリクスを管理する方法について説明します。

トピック

- フリートメトリクスの管理 (コンソール)
- フリートのメトリクスの管理 (CLI)
- IoT リソースのタグ付けを承認する

フリートメトリクスの管理 (コンソール)

以下のセクションでは、 AWS IoT コンソールを使用してフリートメトリクスを管理する方法を示します。フリートメトリックスを作成する前に、関連するデータソースでフリートインデックス作成が有効になっていることを確認してください。

フリートインデックス作成を有効にする

フリートインデックス作成がすでに有効になっている場合は、このセクションをスキップしてください。

フリートのインデックス作成を有効にしていない場合は、次の手順に従ってください。

1. https://console.aws.amazon.com/iot/ で AWS IoT コンソールを開きます。
2. AWS IoT メニューで、設定 を選択します。
3. 詳細設定を表示するには、[Settings](設定)ページで、[Fleet indexing](フリートインデックスを作成する)セクションまでスクロールダウンします。
4. フリートインデックス作成設定を更新するには、[Fleet indexing](フリートインデックスの作成)セクションで、[Manage indexing](インデックス作成の管理)を選択します。
5. [Manage fleet indexing] (フリートインデックス作成の管理) ページで、必要に基づいてフリートインデックス作成の設定を更新します。
   - 設定

     モノのインデックス作成を有効にするには、[Thing indexing](モノのインデックス作成)をオンにして、インデックスを作成するデータソースを選択します。

     モノグループのインデックス作成を有効にするには、[Thing group indexing](モノグループのインデックス作成)をオンに切り替えます。
   - 集約のカスタムフィールド - オプション

     カスタムフィールドは、フィールド名とフィールドタイプのペアのリストです。

     カスタムフィールドのペアを追加するには、[Add New field](新しいフィールドを追加)を選びます。`attributes.temperature` のようなカスタムフィールド名を入力してから、[Field type] (フィールドタイプ) メニューからフィールドタイプを選択します。カスタムフィールド名は `attributes.` で始まり、モノの集計クエリを実行するための属性として保存されることに注意してください。

     設定を更新して保存するには、[Update](更新)を選びます。

フリートメトリクスを作成します

1. https://console.aws.amazon.com/iot/ で AWS IoT コンソールを開きます。
2. AWS IoT メニューで、管理を選択し、フリートメトリクスを選択します。

3. [Fleet metrics] (フリートメトリクス) ページで, [Create fleet metric] (フリートメトリクスの作成) をクリックして、作成ステップを完了します。
4. ステップ 1で、[Configure fleet metrics](フリートメトリクス)を設定します。
   - Query(クエリ)セクションで、集約検索を実行したいモノまたは、モノのグループを指定するクエリ文字列を入力します。クエリ文字列は、属性と値で構成されます。[Properties] (プロパティ) には、使用したい属性を選びます。あるいは、リストに属性が表示されていない場合は、フィールドに属性を入力します。:の後に値を入力してください。クエリ文字列の例として、`thingName:TempSensor*`があります。入力するクエリ文字列ごとに、キーボードのEnterを押します。複数のクエリ文字列を入力する場合は、それらの間に、and、or、and not、または or not を選択してそれらの関係を指定します。
   - [レポートのプロパティ] で、[インデックス名]、[集計タイプ]、[集計フィールド] をそれぞれのリストから選択します。次に、[Select data](データの選択)で集計したいデータを選択します。そこでは、複数のデータ値を選択できます。
   - [Next] (次へ) を選択します。
5. ステップ 2で、[Specify fleet metric properties](フリートメトリクスプロパティを指定)
   - [Fleet metric name](フリートメトリクス)フィールドに、作成しているフリートメトリクスの名前を入力します。
   - [Description-optional](説明 - オプション)フィールドに、作成しているフリートメトリクスの説明を入力します。このフィールドはオプションです。
   - [Hours](時間)および[Minutes](分)フィールドに、フリートメトリクスがCloudWatchにデータを送信する時間（頻度）を入力します。
   - [次へ] を選択します。
6. ステップ3の [Review and create](確認と作成)
   - ステップ 1 とステップ 2 の設定を確認します。設定を編集するには、[Edit](編集)を選びます。
   - [Create fleet metric](フリートメトリクスの作成)を選びます。

作成に成功すると、[Fleet metric](フリートメトリクス)ページで、フリートメトリックスがリスト化されます。

フリートメトリクスを更新します

1. [フリートメトリクス] ページで、更新したいフリートメトリクスを選択します。
2. [Fleet Details](フリートの詳細) タブで、[Edit]（編集）を選びます。これにより、作成ステップが開き、3 つのステップのいずれかでフリートメトリクスを更新する事ができます。

3. フリートメトリクスの更新が完了したら、[Update fleet metric](フリートメトリクスの更新)を選択します。

フリートメトリクスを削除する

1. [フリートメトリクス] ページで、削除したいフリートメトリクスを選びます。
2. フリートメトリクスの詳細を表示する次のページで、[Delete](削除)を選びます。
3. ダイアログボックスで、削除を確認するフリートメトリクスの名前を入力します。
4. [Delete](削除) を選びます。このステップは、フリートメトリクスを永続的に削除します。

フリートのメトリクスの管理 (CLI)

以下のセクションでは、 を使用してフリートメトリクス AWS CLI を管理する方法を示します。フリートメトリクスを作成する前に、関連するデータソースおよび設定と一緒に、フリートインデックス作成が有効になっていることを確認してください。モノまたはモノのグループのフリートのインデックス作成を有効にするために、[Managing thing indexing](モノのインデックス化管理)または[Managing thing group indexing](モノのグループインデックス化管理)の手順に従ってください。

フリートメトリクスを作成します

フリートメトリクスを作成するには、create-fleet-metric CLI コマンドを使用できます。

```
aws iot create-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 60 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum
```

このコマンドの出力には、フリートメトリクスの名前と Amazon リソースネーム (ARN) が含まれます。出力は次のようになります。

```
{
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "metricName": "YourFleetMetricName"
}
```

フリートメトリクスを一覧表示します

list-fleet-metric CLI コマンドを使用して、アカウント内のすべてのフリートメトリクスを一覧表示できます。

```
aws iot list-fleet-metrics
```

このコマンドの出力には、すべてのフリートのメトリクスが含まれます。出力は次のようになります。

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

フリートのメトリクスを説明する

describe-fleet-metric CLI コマンドを使用して、フリートメトリクスに関するより詳細な情報を表示できます。

```
aws iot describe-fleet-metric --metric-name "YourFleetMetricName"
```

コマンドの出力には、指定されたフリートメトリクスに関する詳細情報が含まれます。出力は次のようになります。

```
{
    "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"
}
```

フリートメトリクスを更新します

フリートメトリクスを更新するには、update-fleet-metric CLI コマンドを使用できます。

```
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-メトリクスコマンドは、出力を生成しません。describe-fleet-メトリクス CLI コマンドを使用して、結果を表示できます。

```
{
    "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"
}
```

フリートメトリクスを削除する

delete-fleet-metric CLI コマンドを使用してフリートメトリクスを削除します。

```
aws iot delete-fleet-metric --metric-name "YourFleetMetricName"
```

このコマンドは、削除が成功した場合、または存在しないフリートメトリクスを指定した場合、出力を生成しません。

詳細については、「フリートメトリクスのトラブルシューティング」をご覧ください。

IoT リソースのタグ付けを承認する

作成、変更、または使用できるフリートメトリクスをより適切に制御するために、フリートメトリクスにタグをアタッチすることができます。

AWS マネジメントコンソール または を使用して作成したフリートメトリクスにタグを付けるには AWS CLI、IAM ポリシーに `iot:TagResource`アクションを含めて、ユーザーアクセス許可を付与する必要があります。IAM ポリシーに `iot:TagResource` が含まれていない場合、タグを使用してフリートメトリクスを作成するアクションは `AccessDeniedException` エラーを返します。

リソースへのタグ付けに関する一般情報については、「AWS IoT リソースのタグ付け」を参照してください。

IAM ポリシーの例

フリートメトリクスの作成時にタグ付けのアクセス許可を付与する IAM ポリシーの例を参照してください。

```
{
 "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/*"
    ]
   }
  ]
}
```

詳細については、AWS IoTのアクション、リソース、および条件キーを参照してください。

MQTT ベースのファイル配信

ファイルを管理し、フリート内の AWS IoT デバイスに転送するために使用できる 1 つのオプションは、MQTT ベースのファイル配信です。 AWS クラウドのこの機能を使用すると、複数のファイルを含むストリームを作成したり、ストリームデータ (ファイルリストと説明) を更新したり、ストリームデータを取得したりできます。 AWS IoT MQTT ベースのファイル配信は、JSON または CBOR のリクエストおよびレスポンスメッセージをサポートする MQTT プロトコルを使用して、小さなブロックでデータを IoT デバイスに転送できます。

を使用して IoT デバイスとの間でデータを転送する方法の詳細については AWS IoT、「」を参照してくださいデバイスを に接続する AWS IoT。

トピック

- ストリーミングとは
- AWS クラウドでのストリームの管理
- デバイスで AWS IoT の MQTT ベースのファイル配信の使用
- FreeRTOS OTA のユースケースの例

ストリーミングとは

では AWS IoT、ストリームはパブリックにアドレス可能なリソースであり、IoT デバイスに転送できるファイルのリストの抽象化です。一般的なストリーミングには、以下の情報が含まれています。

- Amazon リソースネーム (ARN) は、特定の時刻にストリーミングを一意に識別します。この ARN のパターンは `arn:partition:iot:region:account-ID:stream/stream ID` です。
- ストリームを識別し、() または SDK コマンドで使用される (通常は必須) ストリーム ID。 AWS Command Line Interface AWS CLI
- ストリーミングリソースの説明を提供するストリーミングの説明。
- ストリーミングの特定のバージョンを識別するストリーミングバージョン。ストリーミングデータはデバイスがデータ転送を開始する直前に変更できるため、デバイスはストリーミングバージョンを使用して整合性チェックを実施できます。
- デバイスに転送できるファイルのリスト。リスト内の各ファイルについて、ストリーミングはファイル ID、ファイルサイズ、および Amazon S3 バケット名、オブジェクトキー、オブジェクトバージョンなどで構成されるファイルのアドレス情報を記録します。

- データストレージに保存されているストリームファイルを読み取るアクセス許可を MQTT ベースのファイル配信に付与 AWS IoT する AWS Identity and Access Management (IAM) ロール。

AWS IoT MQTT ベースのファイル配信は、デバイスが クラウドから AWS データを転送できるように、次の機能を提供します。

- MQTT プロトコルを使用したデータ転送。
- JSON または CBOR 形式のサポート。
- ストリーミングファイルリスト、ストリーミングバージョン、および関連情報を取得するためのストリーミング (`DescribeStream` API) を記述する機能。
- ハードウェア制約のあるデバイスがブロックを受信できるように、小さなブロック (`GetStream` API) でデータを送信する機能。
- 異なるメモリ容量を持つデバイスをサポートするための、リクエストごとの動的ブロックサイズのサポート。
- 複数のデバイスが同じストリーミングファイルからデータブロックをリクエストする場合の同時実行ストリーミングリクエストの最適化。
- ストリーミングファイルのデータストレージとしての Amazon S3。
- AWS IoT MQTT ベースのファイル配信から CloudWatch へのデータ転送ログ発行のサポート。

MQTT ベースのファイル配信クォータについては、「AWS 全般のリファレンス」の「AWS IoT Core サービスクォータ」を参照してください。

AWS クラウドでのストリームの管理

AWS IoT には、 AWS クラウドでストリームを管理するために使用できる AWS SDK と AWS CLI コマンドが用意されています。これらのコマンドを使うと、次のことができます。

- ストリーミングを作成します。CLI / SDK
- ストリーミングを記述して、その情報を取得します。CLI / SDK
- のストリームを一覧表示します AWS アカウント。CLI / SDK
- ストリーミング内のファイルリストまたはストリーミングの説明を更新します。CLI / SDK
- ストリーミングを削除します。CLI / SDK

> **Note**
>
> 現時点では、ストリーミングは AWS マネジメントコンソールに表示されません。ストリームを管理するには、 AWS CLI または AWS SDK を使用する必要があります AWS IoT。また、Embedded C SDK は、MQTT ベースのファイル転送をサポートする唯一の SDK です。

デバイスから AWS IoT MQTT ベースのファイル配信を使用する前に、次のセクションに示すように、デバイスで次の条件が満たされていることを確認する必要があります。

- MQTT 経由でデータを送信するために必要な正しいアクセス許可を反映するポリシー。
- デバイスは AWS IoT Device Gateway に接続できます。
- リソースにタグ付けできることを示すポリシーステートメント。`CreateStream` がタグで呼び出された場合、`iot:TagResource` は必須です。

デバイスから AWS IoT MQTT ベースのファイル配信を使用する前に、次のセクションの手順に従って、デバイスが適切に承認され、 AWS IoT Device Gateway に接続できることを確認する必要があります。

デバイスにアクセス許可を付与する

AWS IoT ポリシー作成r.の手順に従って、デバイスポリシーを作成することも、既存のデバイスポリシーを使用することもできます。デバイスに関連付けられている証明書にポリシーをアタッチし、デバイスポリシーに次のアクセス許可を追加します。

```
{
    "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/*"
            ]
        }
    ]
}
```

デバイスを に接続する AWS IoT

と接続するには、 AWS IoT MQTT ベースのファイル配信を使用するデバイスが必要です AWS IoT。AWS IoT MQTT ベースのファイル配信は AWS クラウド AWS IoT で と統合されるため、デバイスはAWS IoT データプレーンのエンドポイントに直接接続する必要があります。

> **Note**
>
> AWS IoT データプレーンのエンドポイントは、 AWS アカウント および リージョンに固有です。のエンドポイント AWS アカウント と、デバイスが登録されているリージョンを使用する必要があります AWS IoT。

詳細については「に接続する AWS IoT Core」を参照してください。

TagResource の使用

`CreateStream` API アクションは、MQTT を介して 1 つ以上の大きなファイルをチャンクで配信するためのストリームを作成します。

CreateStream API コールを成功させるには、次のアクセス許可が必要です。

- iot:CreateStream
- iot:TagResource (CreateStream にタグがある場合)

これら 2 つのアクセス許可をサポートするポリシーを次に示します。

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iot:CreateStream",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:stream/streamId"
    }
}
```

iot:TagResource ポリシーステートメントアクションは、ユーザーが適切なアクセス許可なしでリソースのタグを作成または更新できないようにするために必要です。iot:TagResource の特定のポリシーステートメントアクションがない場合、リクエストにタグが付属している場合、CreateStream API コールは AccessDeniedException を返します。

詳細については、次のリンクを参照してください。

- CreateStream
- TagResource
- タグ:

デバイスで AWS IoT の MQTT ベースのファイル配信の使用

データ転送プロセスを開始するには、デバイスは、少なくともストリーミング ID を含む初期データセットを受信する必要があります。AWS IoT ジョブ を使用して、ジョブドキュメントに初期データセットを含めることで、デバイスのデータ転送タスクをスケジュールできます。デバイスが最初のデータセットを受け取ると、 AWS IoT MQTT ベースのファイル配信とのやり取りを開始する必要が

あります。 AWS IoT MQTT ベースのファイル配信でデータを交換するには、デバイスは以下を行う必要があります。

- MQTT ベースのファイル配信のトピック にサブスクライブするには、MQTT プロトコルを使用します。
- リクエストを送信し、MQTT メッセージを使用して応答を受信するのを待機します。

必要に応じて、ストリーミングファイル ID とストリーミングバージョンを初期データセットに含めることができます。ストリーミングファイル ID をデバイスに送信すると、デバイスからこの ID を取得するための DescribeStream リクエストを行う必要がなくなるため、デバイスのファームウェア/ソフトウェアのプログラミングを簡素化できます。ストリーミングが予期せず更新された場合に備えて、デバイスは GetStream リクエストでストリーミングバージョンを指定して、整合性チェックを実施できます。

DescribeStream を使用してストリーミングデータを取得する

AWS IoT MQTT ベースのファイル配信は、ストリームデータをデバイスに送信する DescribeStream API を提供します。この API によって返されるストリーミングデータには、ストリーミング ID、ストリーミングバージョン、ストリーミングの説明、およびストリーミングファイルのリストが含まれており、それぞれにファイル ID とファイルサイズ (バイト単位) が含まれています。この情報を使用して、デバイスは任意のファイルを選択してデータ転送プロセスを開始できます。

> **Note**
>
> デバイスが初期データセットで必要なすべてのストリーミングファイル ID を受信する場合、DescribeStream API を使用する必要はありません。

DescribeStream リクエストを実行するには、以下の手順に従います。

1. 「承諾済み」トピックフィルター を$aws/things/*ThingName*/streams/*StreamId*/description/jsonサブスクライブします。
2. 「拒否済み」トピックフィルター $aws/things/*ThingName*/streams/*StreamId*/rejected/json をサブスクライブします。
3. $aws/things/*ThingName*/streams/*StreamId*/describe/jsonにメッセージを送信して、DescribeStream リクエストを発行します。

4. リクエストが受け入れられた場合、デバイスは「承諾済み」トピックフィルターで DescribeStream 応答を受け取ります。
5. リクエストが拒否された場合、デバイスは「拒否済み」トピックフィルターでエラー応答を受け取ります。

> **Note**
>
> 表示されているトピックとトピックフィルターで json を cbor に置き換えると、デバイスは JSON よりもコンパクトな CBOR 形式でメッセージを受信します。

DescribeStream リクエスト

JSON での典型的な DescribeStream リクエストは、次の例のようになります。

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039"
}
```

- (オプション)「c」はクライアントトークンフィールドです。

  クライアントトークンは 64 バイトを超えない範囲にします。64 バイトより長いクライアントトークンは、エラー応答と InvalidRequest エラーメッセージを引き起こします。

DescribeStream レスポンス

JSON での DescribeStream 応答は次の例のようになります。

```
{
    "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」はクライアントトークンフィールドです。DescribeStream リクエストで指定された場合、これが返されます。クライアントトークンを使用して、応答をそのリクエストに関連付けます。
- 「s」は整数としてのストリーミングバージョンです。このバージョンを使用して、GetStream リクエストの整合性チェックを実行できます。
- 「r」にはストリーミング内のファイルのリストが含まれています。
  - 「f」は整数としてのストリーミングファイル ID です。
  - 「z」はストリーミングファイルのサイズ (バイト数) です。
- 「d」にはストリーミングの説明が含まれています。

ストリーミングファイルからデータブロックを取得する

GetStream API を使用すると、デバイスが小さなデータブロックでストリーミングファイルを受信できるため、大きなブロックサイズの処理に制約があるデバイスで使用できます。データファイル全体を受信するには、すべてのデータブロックが受信されて処理されるまで、デバイスは複数のリクエストと応答を送受信する必要がある場合があります。

GetStream リクエスト

GetStream リクエストを実行するには、以下の手順に従います。

1. 「承諾済み」トピックフィルター を$aws/things/*ThingName*/streams/*StreamId*/data/jsonサブスクライブします。
2. 「拒否済み」トピックフィルター $aws/things/*ThingName*/streams/*StreamId*/rejected/json をサブスクライブします。
3. トピック $aws/things/*ThingName*/streams/*StreamId*/get/json に GetStream リクエストを発行します。
4. リクエストが承諾された場合、デバイスは「承諾済み」トピックフィルターで 1 つ以上の GetStream 応答を受け取ります。各応答メッセージには、1 つのブロックの基本情報とデータペイロードが含まれます。

5. ステップ 3 と 4 を繰り返して、すべてのデータブロックを受信します。リクエストされたデータ量が 128 KB を超える場合は、これらの手順を繰り返す必要があります。リクエストされたすべてのデータを受信するには、複数の `GetStream` リクエストを使用するようにデバイスをプログラムする必要があります。
6. リクエストが拒否された場合、デバイスは「拒否済み」トピックフィルターでエラー応答を受け取ります。

> **Note**
>
> - 表示されているトピックとトピックフィルターで「json」を「cbor」に置き換えると、デバイスは JSON よりもコンパクトな CBOR 形式でメッセージを受信します。
> - AWS IoT MQTT ベースのファイル配信は、ブロックのサイズを 128 KB に制限します。128 KB を超えるブロックをリクエストすると、リクエストは失敗します。
> - 合計サイズが 128 KB を超える複数のブロックをリクエストできます (例えば、合計 160 KB のデータに対して、それぞれ 32 KB の 5 つのブロックをリクエストする場合)。この場合、リクエストは失敗しませんが、リクエストされたすべてのデータを受信するには、デバイスが複数のリクエストを行う必要があります。デバイスが追加のリクエストを行うと、サービスは追加のブロックを送信します。以前の応答が正しく受信されて処理された後でのみ、新しいリクエストを続行することをお勧めします。
> - リクエストされたデータの合計サイズに関係なく、ブロックが受信されなかった場合、または正しく受信されなかった場合に再試行を開始するようにデバイスをプログラムする必要があります。

JSON での典型的な `GetStream` リクエストは、次の例のようになります。

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "s": 1,
    "f": 0,
    "l": 4096,
    "o": 2,
    "n": 100,
    "b": "..."
}
```

- [オプション]「c」はクライアントトークンフィールドです。

  クライアントトークンは 64 バイトを超えない範囲にします。64 バイトより長いクライアントトークンは、エラー応答と InvalidRequest エラーメッセージを引き起こします。

- [オプション]「s」はストリーミングバージョンフィールド (整数) です。

  MQTT ベースのファイル配信では、このリクエストされたバージョンとクラウド内の最新のストリーミングバージョンに基づいて整合性チェックが適用されます。GetStream リクエストでデバイスから送信されたストリーミングバージョンがクラウド内の最新のストリーミングバージョンと一致しない場合、サービスはエラー応答と VersionMismatch エラーメッセージを送信します。通常、デバイスは、初期データセットまたは DescribeStream への応答で、想定される (最新の) ストリーミングバージョンを受け取ります。

- 「f」はストリーミングファイル ID (0 ~ 255 の整数) です。

  ストリームファイル ID は、 AWS CLI または SDK を使用してストリームを作成または更新するときに必要です。デバイスが存在しない ID でストリーミングファイルをリクエストする場合、サービスはエラー応答と ResourceNotFound エラーメッセージを送信します。

- 「l」はデータブロックサイズ (バイト単位) です (256 ~ 131,072 の範囲の整数)。

  ビットマップフィールドを使用して、GetStream 応答で返されるストリーミングファイルの部分を指定する方法については、「GetStream リクエストのビットマップを構築します」を参照してください。デバイスが範囲外のブロックサイズを指定した場合、サービスはエラー応答と BlockSizeOutOfBounds エラーメッセージを送信します。

- [オプション]「o」は、ストリーミングファイル内のブロックのオフセット (0 ~ 98,304 の範囲の整数) です。

  ビットマップフィールドを使用して、GetStream 応答で返されるストリーミングファイルの部分を指定する方法については、「GetStream リクエストのビットマップを構築します」を参照してください。最大値 98,304 は、24 MB のストリーミングファイルサイズ制限と最小ブロックサイズの 256 バイトに基づいています。指定されなかった場合、デフォルト値は 0 です。

- [オプション]「n」は、リクエストされたブロックの数です (0 ~ 98,304 の範囲の整数)。

  「n」フィールドは、(1) リクエストされたブロック数、または (2) ビットマップリクエストによって返されるブロック数の制限 (ビットマップフィールド (「b」) が使用されている場合) のいずれかを指定します。この 2 つ目の使用はオプションです。定義されていない場合、デフォルトで 131072/*DataBlockSize* になります。

- [オプション]「b」は、リクエストされているブロックを表すビットマップです。

ビットマップを使用すると、デバイスは非連続ブロックをリクエストできるため、エラー後の再試行の処理がより便利になります。ビットマップフィールドを使用して、GetStream 応答で返されるストリーミングファイルの部分を指定する方法については、「GetStream リクエストのビットマップを構築します」を参照してください。このフィールドでは、ビットマップを 16 進表記でビットマップの値を表す文字列に変換します。ビットマップは 12,288 バイト未満である必要があります。

> **Important**
>
> 「n」か「b」のいずれかを指定する必要があります。どちらも指定されていない場合、GetStream要求は、ファイルサイズが 131072 バイト (128 KB) 未満の場合は有効ではない可能性があります。

GetStream レスポンス

リクエストされた各データブロックに対する JSON の GetStream 応答は、次の例のようになります。

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "f": 0,
    "l": 4096,
    "i": 2,
    "p": "..."
}
```

- 「c」はクライアントトークンフィールドです。GetStream リクエストで指定された場合、これが返されます。クライアントトークンを使用して、応答をそのリクエストに関連付けます。
- 「f」は現在のデータブロックペイロードが属するストリーミングファイルの ID です。
- 「l」はデータブロックペイロードのサイズ (バイト単位) です。
- 「i」はペイロードに含まれるデータブロックの ID です。データブロックは 0 から番号付けされます。
- 「p」には、データブロックのペイロードが含まれます。このフィールドは、Base64 でエンコードされたデータブロックの値を表す文字列です。

GetStream リクエストのビットマップを構築します

GetStream リクエストでビットマップフィールド (b) を使用して、ストリーミングファイルから連続しないブロックを取得できます。これは、RAM 容量が限られているデバイスがネットワーク配信の問題に対処するのに役立ちます。デバイスは、受信されなかったブロックまたは正しく受信されなかったブロックのみをリクエストできます。ビットマップは、ストリーミングファイルのどのブロックが返されるかを決定します。ビットマップで 1 に設定されているビットごとに、ストリーミングファイルの対応するブロックが返されます。

GetStreamリクエストでビットマップとそのサポートフィールドを指定する方法の例を次に示します。例えば、ストリーミングファイルを 256 バイト (ブロックサイズ) のチャンクで受信するとします。256 バイトの各ブロックには、ファイル内の位置を指定する番号が付いていると考えてください。この番号は 0 から始まります。したがって、ブロック 0 はファイル内の 256 バイトの最初のブロックであり、ブロック 1 は 2 番目のブロックであり、以下同様です。ファイルからブロック 20、21、24、および 43 をリクエストします。

ブロックオフセット

最初のブロックの番号は 20 であるため、オフセット (フィールド o) を 20 に指定して、ビットマップのスペースを節約します。

ブロックの数

デバイスが限られたメモリリソースで処理できる以上のブロックを受信しないようにするために、MQTT ベースのファイル配信によって送信される各メッセージで返されるブロックの最大数を指定できます。ビットマップ自体が指定する値がこのブロック数より少ない場合、または MQTT ベースのファイル配信によって送信される応答メッセージの合計サイズが、GetStream リクエストごとのサービス制限である 128 KB よりも大きくなる場合、この値は無視されることに注意してください。

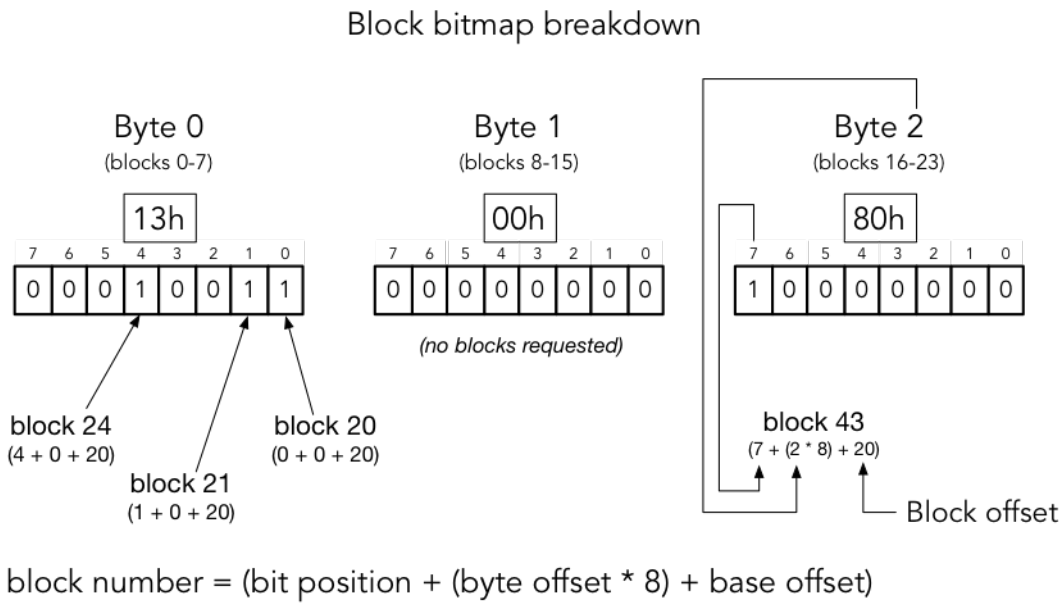
ブロックビットマップ

ビットマップ自体は、16 進数表記で表現された符号なしバイトの配列であり、数値の文字列表現として GetStream リクエストに含まれます。しかし、この文字列を構築するために、ビットマップをビットの長いシーケンス (2 進数) と考えることから始めましょう。このシーケンスのビットが 1 に設定されている場合、ストリーミングファイルの対応するブロックがデバイスに返送されます。この例では、ブロック 20、21、24、および 43 を受信する必要があるため、ビットマップでビット 20、21、24、および 43 を設定する必要があります。ブロックオフセットを使用してスペースを節約できるため、各ブロック番号からオフセットを差し引いた後、次の例のようにビット 0、1、4、および 23 を設定します。

```
1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

一度に 1 バイト (8 ビット) を取ると、これは通常、「0b00010011」、「0b00000000」、「0b10000000」のように記述されます。ビット 0 は、最初のバイトの終わりにバイナリ表現で表示され、最後のバイトの最初にビット 23 が表示されます。慣習を知らなければ、これは混乱を招く可能性があります。最初のバイトにはビット 7～0 が (この順序で) 含まれ、2 番目のバイトにはビット 15～8 が含まれ、3 番目のバイトにはビット 23～16 が含まれ、以降同様です。これは 16 進数表記では「0x130080」に変換されます。

> **Tip**
>
> 標準のバイナリを 16 進表記に変換できます。一度に 4 桁の 2 進数を取り、それらを同等の 16 進数に変換します。例えば、「0001」は「1」になり、「0011」は「3」になり、以降同様です。



これをすべてまとめると、`GetStream` リクエストの JSON は次のようになります。

```
{
    "c" : "1",
    "s" : 1,
    "l" : 256,
    "f" : 1,
    "o" : 20,
```

```
    "n" : 32,
    "b" : "130080"
}
```

- 「c」はクライアントトークンフィールドです。
- 「s」は想定されるストリームバージョンです。
- 「l」はデータブロックペイロードのサイズ (バイト単位) です。
- 「f」はソースファイルインデックスの ID です。
- 「o」はブロックオフセットです。
- 「n」はブロック数です。
- 「b」はオフセットから始まる欠落している blockId ビットマップです。この値は base64 でエンコードする必要があります。

AWS IoT MQTT ベースのファイル配信によるエラーの処理

DescribeStream API と GetStream API の両方のデバイスに送信されるエラー応答には、クライアントトークン、エラーコード、およびエラーメッセージが含まれます。典型的なエラー応答は、次の例のようになります。

```
{
    "o": "BlockSizeOutOfBounds",
    "m": "The block size is out of bounds",
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380"
}
```

- 「o」はエラーが発生した理由を示すエラーコードです。詳細については、このセクションの後半にあるエラーコードを参照してください。
- 「m」はエラーの詳細を含むエラーメッセージです。
- 「c」はクライアントトークンフィールドです。DescribeStream リクエストで指定されている場合は、これが返される場合があります。クライアントトークンを使用して、応答をそのリクエストに関連付けることができます。

  クライアントトークンフィールドは、常にエラー応答に含まれるわけではありません。リクエストで指定されたクライアントトークンが無効または不正な形式の場合、エラーレスポンスで返されません。

> **Note**
>
> 下位互換性のために、エラー応答のフィールドは省略形ではない場合があります。例えば、エラーコードは「code」または「o」フィールドで指定される場合があり、クライアントトークンフィールドは「clientToken」または「c」フィールドで指定される場合があります。上記の省略形を使用することをお勧めします。

InvalidTopic

ストリーミングメッセージの MQTT トピックが無効です。

InvalidJson

Stream リクエストは有効な JSON ドキュメントではありません。

InvalidCbor

Stream リクエストは有効な CBOR ドキュメントではありません。

InvalidRequest

通常、リクエストは形式が不正であると識別されます。詳細については、エラーメッセージを参照してください。

Unauthorized

リクエストは、Amazon S3 などのストレージメディア内のストリーミングデータファイルへのアクセスを許可されていません。詳細については、エラーメッセージを参照してください。

BlockSizeOutOfBounds

ブロックサイズが範囲外です。AWS IoT Core Service Quotas の「MQTT ベースのファイル配信」のセクションを参照してください。

OffsetOutOfBounds

オフセットが範囲外です。AWS IoT Core Service Quotas の「MQTT ベースのファイル配信」のセクションを参照してください。

BlockCountLimitExceeded

リクエストブロックの数が範囲外です。AWS IoT Core Service Quotas の「MQTT ベースのファイル配信」のセクションを参照してください。

BlockBitmapLimitExceeded

リクエストビットマップのサイズが範囲外です。AWS IoT Core Service Quotas の「MQTT ベースのファイル配信」のセクションを参照してください。

ResourceNotFound

リクエストされたストリーミング、ファイル、ファイルバージョン、またはブロックが見つかりませんでした。詳細については、エラーメッセージを参照してください。

VersionMismatch

リクエストのストリーミングバージョンが、MQTT ベースのファイル配信機能のストリーミングバージョンと一致しません。これは、ストリーミングバージョンがデバイスによって最初に受信されてから、ストリーミングデータが変更されたことを示します。

ETagMismatch

ストリーミング内の S3 ETag が最新の S3 オブジェクトバージョンの ETag と一致しません。

InternalError

MQTT ベースのファイル配信で内部エラーが発生しました。

FreeRTOS OTA のユースケースの例

FreeRTOS OTA (over-the-air) エージェントは、 AWS IoT MQTT ベースのファイル配信を使用して FreeRTOS ファームウェアイメージを FreeRTOS デバイスに転送します。初期データセットをデバイスに送信するには、 AWS IoT ジョブサービスを使用して FreeRTOS デバイスに OTA 更新ジョブをスケジュールします。

MQTT ベースのファイル配信クライアントのリファレンス実装については、FreeRTOS ドキュメントの FreeRTOS OTA エージェントコードを参照してください。

Device Advisor

Device Advisor は、デバイスソフトウェア開発中に IoT デバイスを検証するための、クラウドベースのフルマネージドテスト機能です。Device Advisor は、デバイスを本番環境にデプロイする前に AWS IoT Core、IoT デバイスとの信頼性と安全性を検証するために使用できる事前構築されたテストを提供します。Device Advisor の事前構築されたテストにより、デバイスソフトウェアをTLS,MQTT,デバイスシャドウ, およびIoT ジョブズの使用のベストプラクティスに照らして検証できます。また、署名された認定レポートをダウンロードして AWS パートナーネットワークに提出し、デバイスを送付してテストを待つことなく、AWS Partner Device Catalog の認定を受けることもできます。

> **Note**
>
> Device Advisor は、us-east-1、us-west-2、ap-northeast-1、および eu-west-1 リージョンでサポートされています。
> Device Advisor は、MQTT および MQTT over WebSocket Secure (WSS) プロトコルを使用してメッセージを発行およびサブスクライブするデバイスとクライアントをサポートします。すべてのプロトコルでは、IPv4 および IPv6 がサポートされています。
> デバイスアドバイザーは RSA サーバー証明書をサポートしています。

に接続するために構築されたデバイスは、Device Advisor を活用 AWS IoT Core できます。Device Advisor には、AWS IoT コンソールから、または AWS CLI または SDK を使用してアクセスできます。デバイスをテストする準備ができたら、 に登録 AWS IoT Core し、Device Advisor エンドポイントでデバイスソフトウェアを設定します。その後、事前に構築されたテストを選択して設定し、デバイスでテストを実行して、詳細なログまたは認定レポートとともにテスト結果を取得します。

Device Advisor はクラウド内のテストエンドポイントです AWS 。Device Advisor が提供するテストエンドポイントに接続するようにデバイスを設定することで、デバイスをテストできます。デバイスがテストエンドポイントに接続するように設定されたら、Device Advisor のコンソールにアクセスするか、 AWS SDK を使用してデバイスで実行するテストを選択できます。その後、Device Advisor は、リソースのプロビジョニング、テストプロセスのスケジューリング、ステートマシンの管理、デバイスの動作の記録、結果の記録、テストレポートの形式での最終結果の提供など、テストのライフサイクル全体を管理します。

TLS プロトコル

Transport Layer Security (TLS) プロトコルは、インターネットなどの安全性が低いネットワークでの機密データの暗号化に使用されます。TLS プロトコルは、Secure Sockets Layer (SSL) プロトコルの後継です。

Device Advisor は以下の TLS プロトコルをサポートしています。

- TLS1.3 (推奨)
- TLS 1.2

プロトコル、ポートマッピング、認証

デバイス通信プロトコルは、デバイスまたはクライアントがデバイスエンドポイントを使用してメッセージブローカーに接続するために使用されます。次の表は、Device Advisor エンドポイントがサポートするプロトコルと、それらが使用する認証方法とポートを示しています。

プロトコル、認証、ポートマッピング

プロトコル	サポートされているオペレーション	認証	ポート	ALPN プロトコル名
MQTT over WebSocket	発行、サブスクライブ	署名バージョン 4	443	該当なし
MQTT	発行、サブスクライブ	X.509 クライアント証明書	8883	`x-amzn-mqtt-ca`
MQTT	発行、サブスクライブ	X.509 クライアント証明書	443	該当なし

この章には、以下のセクションが含まれています。

- 設定
- コンソールでの Device Advisor の開始方法
- Device Advisor ワークフロー
- Device Advisor の詳細コンソールワークフロー
- 長期テストコンソールのワークフロー
- デバイスアドバイザー VPC エンドポイント (AWS PrivateLink)

- Device Advisor テストケース

設定

Device Advisor を初めて使用する場合は、事前に以下のタスクをすべて実行してください。

IoT モノの作成

まず、IoT モノを作成し、モノに証明書をアタッチします。モノの作成方法のチュートリアルについては、「モノのオブジェクトを作成する」を参照してください。

デバイスロールとして使用する IAM ロールを作成する

> **Note**
>
> Device Advisor コンソールを使用して、デバイスロールをすばやく作成できます。Device Advisor コンソールを使用してデバイスロールを設定する手順については、「コンソールでの Device Advisor の開始方法」を参照してください。

1. AWS Identity and Access Management コンソールに移動し、Device Advisor テスト AWS アカウント に使用する にログインします。
2. 左側のナビゲーションペインで、[Policies] (ポリシー) を選択します。
3. [Create policy] (ポリシーの作成) を選択します。
4. [Create policy] (ポリシーの作成) で、次の操作を実行します。

   a. [Service] (サービス) で、[IoT] を選択します。

   b. [アクション] で、次のいずれかを実行します。

      - (推奨) IoT Thing に添付されているポリシーまたは前のセクションで作成した証明書に基づいてアクションを選択します。
      - [フィルターアクション] ボックスで次のアクションを検索して選択します。

        - `Connect`
        - `Publish`
        - `Subscribe`
        - `Receive`

- `RetainPublish`

c. [リソース] で、クライアント、トピック、およびトピックのリソースを制限します。これらのリソースを制限することは、セキュリティのベストプラクティスです。リソースを制限するには、次の手順を実行します。

i. Connect アクションの [Specify client resource ARN] (クライアントリソース ARN を指定) を選択します。

ii. [ARN の追加] を選択し、次のいずれかを実行します。

> **Note**
>
> clientId は、デバイスが Device Advisor とやり取りするために使用する MQTT クライアント ID です。

- ビジュアル ARN エディタで、[リージョン]、[accountID]、および [clientID] を指定します。
- テストケースを実行する IoT トピックの Amazon リソースネーム (ARN) を手動で入力します。

iii. [追加] を選択します。

iv. [受信およびさらに 1 つのアクションのためにトピックリソース ARN を指定] を選択します。

v. [ARN の追加] を選択し、次のいずれかを実行します。

> **Note**
>
> トピック名は、デバイスがメッセージを発行する MQTT トピックです。

- ビジュアル ARN エディタで、[リージョン]、[accountID]、および [トピック名] を指定します。
- テストケースを実行する IoT トピックの ARN を手動で入力します。

vi. [Add] (追加) をクリックします。

vii. [サブスクライブアクションのトピックフィルターリソース ARN を指定] を選択します。

viii. [ARN の追加] を選択し、次のいずれかを実行します。

> **Note**
>
> トピック名は、デバイスがサブスクライブする MQTT トピックです。

- ビジュアル ARN エディタで、[リージョン]、[accountID]、および [トピック名] を指定します。
- テストケースを実行する IoT トピックの ARN を手動で入力します。

ix. [Add] (追加) を選択します。

5. [Next: Tags (次へ: タグ)] を選択します。
6. [次へ: レビュー] を選択します。
7. [ポリシーの確認] でポリシーの [名前] を入力します。
8. [Create policy] (ポリシーの作成) を選択します。
9. 左のナビゲーションペインで、[Roles] (ロール) を選択します。
10. [ロールの作成] を選択します。
11. [信頼されたエンティティの選択] で、[カスタム信頼ポリシー] を選択します。
12. 次の信頼ポリシーを [カスタム信頼ポリシー] ボックスに入力します。混乱した代理問題から保護するために、グローバル条件コンテキストキー `aws:SourceArn` と `aws:SourceAccount` をポリシーに追加します。

> **Important**
>
> `aws:SourceArn` は、format: `arn:aws:iotdeviceadvisor:region:account-id:*` . と一致する必要があります。*region* がお客様の AWS IoT リージョンと一致し、*account-id* がお客様のカスタマーアカウント ID と一致することを確認してください。詳細については、クロスサービスでの混乱した代理処理を防止するを参照してください。

```
{
    "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. [次へ] を選択します。
14. ステップ 4 で作成したポリシーを選択します。
15. (オプション) [アクセス許可の境界の設定] で、[アクセス許可の境界を使用してロールのアクセス許可の上限を設定する] を選択し、作成したポリシーを選択します。
16. [次へ] を選択します。
17. [Role name] (ロール名) と [Role description] (ロールの説明) を入力します。
18. [ロールの作成] を選択します。

IAM ユーザーが Device Advisor を使用するためのカスタムマネージドポリシーを作成する

1. IAM コンソール (https://console.aws.amazon.com/iam/) に移動します。プロンプトが表示されたら、 AWS 認証情報を入力してサインインします。
2. 左のナビゲーションペインの [Policies (ポリシー)] を選択します。
3. [Create Policy] (ポリシーの作成) を選択し、[JSON] タブを選択します。
4. Device Advisor を使用するために必要なアクセス許可を追加します。ポリシードキュメントは、トピック [Security best practices] (セキュリティのベストプラクティス) にあります。
5. ポリシーのレビュー を選択します。

6. [Name (名前)] と [Description (説明)] を入力します。
7. [Create Policy (ポリシーの作成)] を選択します。

Device Advisor のテストの実行に使用する IAM ユーザーを作成する

> **Note**
>
> Device Advisor テストを実行するときに使用する IAM ユーザーを作成することをお勧めします。管理者ユーザーから Device Advisor のテストを実行すると、セキュリティ上のリスクが生じる可能性があるため、お勧めしません。

1. IAM コンソール (https://console.aws.amazon.com/iam/) に移動します。プロンプトが表示されたら、 AWS 認証情報を入力してサインインします。
2. 左のナビゲーションペインで、[Users] (ユーザー) を選択します。
3. [Add User] を選択します。
4. ユーザー名を入力します。
5. ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS マネジメントコンソール。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。

   ユーザーにプログラムによるアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(推奨) コンソール認証情報を一時的な認証情報として使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、AWS Command Line Interface 「 ユーザーガイド」のAWS 「ローカル開発のためのログイン」を参照してください。

プログラムによるアクセス権を必要とするユーザー	目的	方法
		• AWS SDKs「 SDK およびツールリファレンスガイド」のAWS 「ローカル開発のためのログイン」を参照してください。 AWS SDKs
ワークフォースアイデンティティ (IAM アイデンティティセンターで管理されているユーザー)	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 を使用する AWS CLI ように を設定する AWS IAM Identity Center」を参照してください。 • AWS SDKs、ツール、API については、 AWS APIs 「 SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」を参照してください。 AWS SDKs
IAM	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。

プログラムによるアクセス権を必要とするユーザー	目的	方法
IAM	(非推奨) 長期認証情報を使用して、AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • API AWS APIs「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

6. [Next: Permissions] (次のステップ: 許可) を選択します。
7. アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

   • 以下のユーザーとグループ AWS IAM Identity Center:

     アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セットを作成する」の手順に従ってください。

   • IAM 内で、ID プロバイダーによって管理されているユーザー:

     ID フェデレーションのロールを作成します。詳細については IAM ユーザーガイド の サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する を参照してください。

   • IAM ユーザー:

- ユーザーが担当できるロールを作成します。手順については IAM ユーザーガイド の IAM ユーザーのロールの作成 を参照してください。
- (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。IAM ユーザーガイド の ユーザー (コンソール) へのアクセス許可の追加 の指示に従います。

8. 作成したカスタム管理ポリシーの名前を検索ボックスに入力します。次に、[ポリシー名] のチェックボックスをオンにします。
9. [Next: Tags (次へ: タグ)] を選択します。
10. [Next: Review] を選択します。
11. [Create user] を選択します。
12. [閉じる] を選択します。

Device Advisor では、ユーザーに代わって AWS リソース (モノ、証明書、エンドポイント) にアクセスする必要があります。IAM ユーザーは必要なアクセス許可を備えている必要があります。また、必要なアクセス許可ポリシーを IAM ユーザーにアタッチすると、Device Advisor は Amazon CloudWatch にログを発行します。

デバイスを設定する

Device Advisor は、サーバー名表示 (SNI) TLS 拡張を使用して TLS 設定を適用します。デバイスは、接続時にこの拡張を使用し、Device Advisor のテストエンドポイントと同じサーバー名を渡す必要があります。

Device Advisor は、テストが `Running` のステータスにあるときに TLS 接続を許可します。Device Advisor は、各テストの実行前と実行後に TLS 接続を拒否します。このため、Device Advisor で完全に自動化されたテストを行うために、デバイス接続の再試行メカニズムを使用することをお勧めします。TLS 接続、MQTT 接続、MQTT パブリッシュなど、複数のテストケースを含むテストスイートを実行できます。複数のテストケースを実行する場合、デバイスが 5 秒ごとにテストエンドポイントに接続することを推奨します。その後、自動化された方法で順番に複数のテストケースを実行できます。

> **Note**
>
> テスト用にデバイスソフトウェアを準備するには、 AWS IoT Coreに接続できる SDK を使用することをお勧めします。次に、 AWS アカウント用に提供された Device Advisor テストエンドポイントで SDK を更新する必要があります。

Device Advisor が、アカウントレベルのエンドポイントとデバイスレベルのエンドポイントの 2 種類のエンドポイントをサポートします。ユースケースに最も適したエンドポイントを選択します。異なるデバイスを使用して複数のテストスイートを同時に実行するには、デバイスレベルのエンドポイントを使用します。

次のコマンドを実行して、デバイスレベルのエンドポイントを取得します。

X.509 クライアント証明書を使用する MQTT のお客様の場合:

```
aws iotdeviceadvisor get-endpoint --thing-arn your-thing-arn
```

または

```
aws iotdeviceadvisor get-endpoint --certificate-arn your-certificate-arn
```

シグネチャバージョン 4 を使用している MQTT over WebSocket のお客様の場合:

```
aws iotdeviceadvisor get-endpoint --device-role-arn your-device-role-arn --
authentication-method SignatureVersion4
```

一度に 1 つのテストスイートを実行するには、アカウントレベルのエンドポイントを選択します。次のコマンドを実行して、アカウントレベルのエンドポイントを取得します。

```
aws iotdeviceadvisor get-endpoint
```

コンソールでの Device Advisor の開始方法

このチュートリアルは、 コンソール AWS IoT Core Device Advisor で の使用を開始するのに役立ちます。Device Advisor は、必要なテストや署名済み認定レポートなどの機能を提供します。これらのテストとレポートを使用して、AWS IoT Core 認定プログラムで詳述されているように、AWS Partner Device Catalog でデバイスを認定して一覧表示できます。

Device Advisor の使用の詳細については、Device Advisor ワークフロー および Device Advisor の詳細コンソールワークフロー を参照してください。

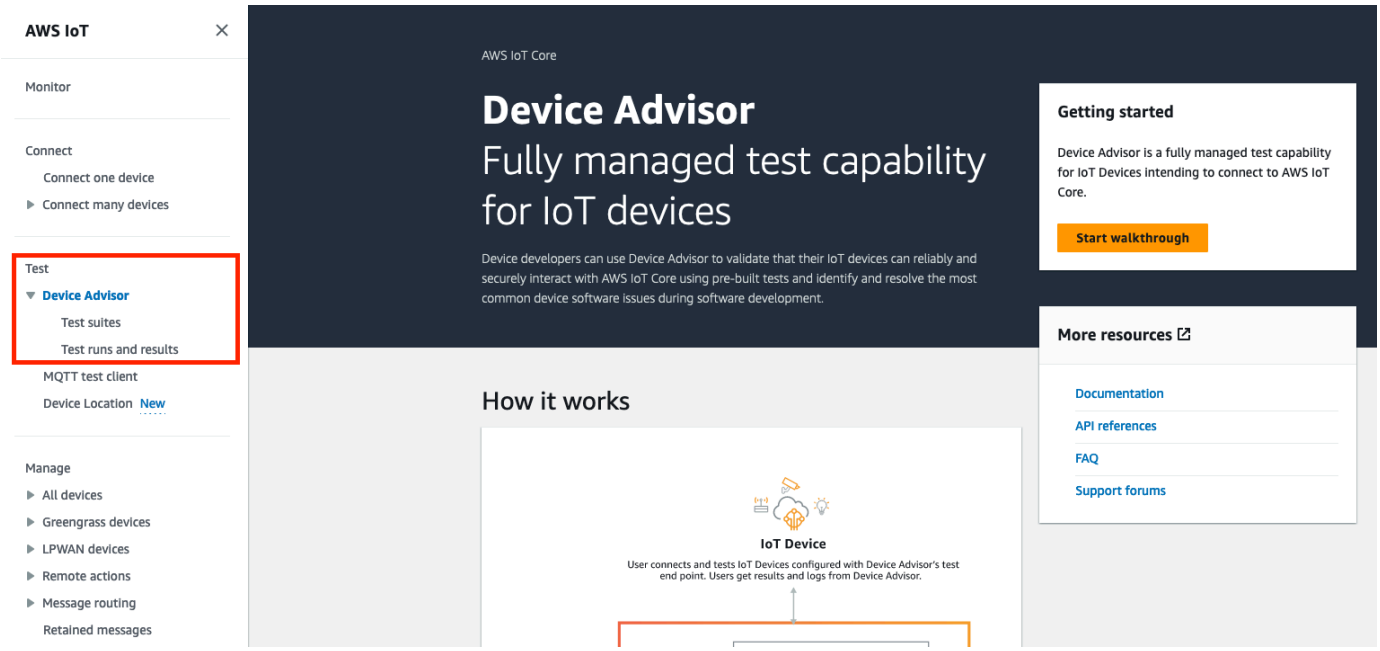
このチュートリアルを完了するには、設定 で概説されている手順に従います。

> **Note**
>
> Device Advisor は、以下でサポートされています AWS リージョン。

- 米国東部 (バージニア北部)
- 米国西部 (オレゴン)
- アジアパシフィック (東京)
- 欧州 (アイルランド)

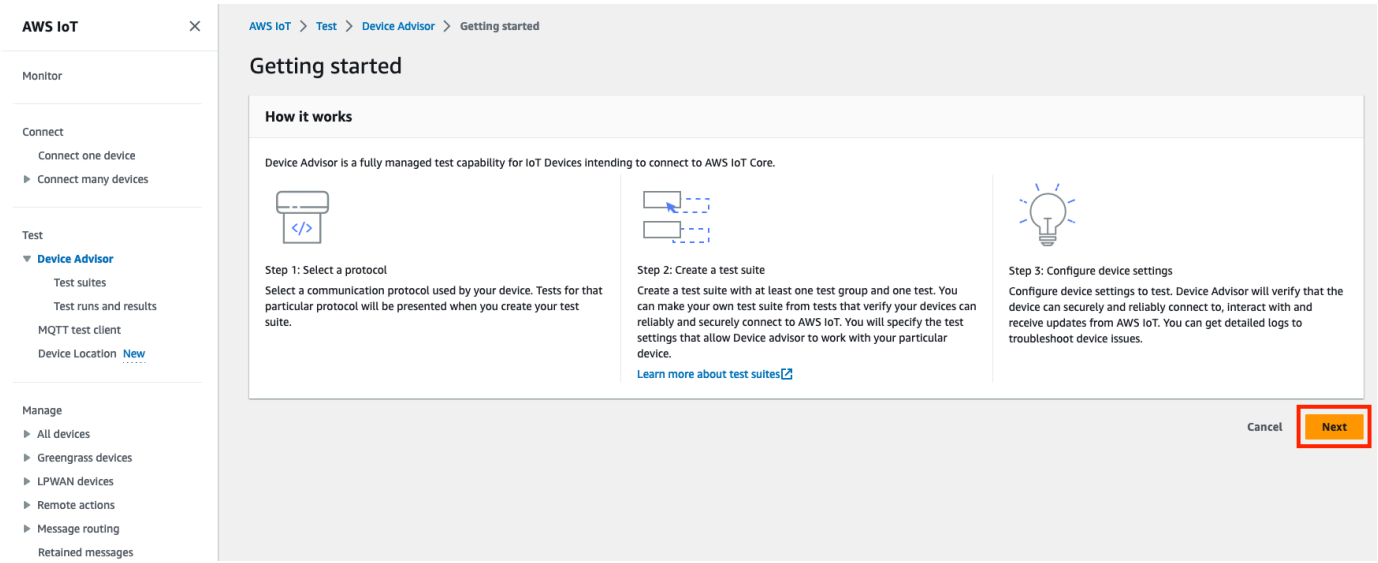
開始方法

1. AWS IoT コンソールのナビゲーションペインの [テスト] の下で、[Device Advisor] を選択します。次に、コンソールの [チュートリアルの開始] ボタンを選択します。

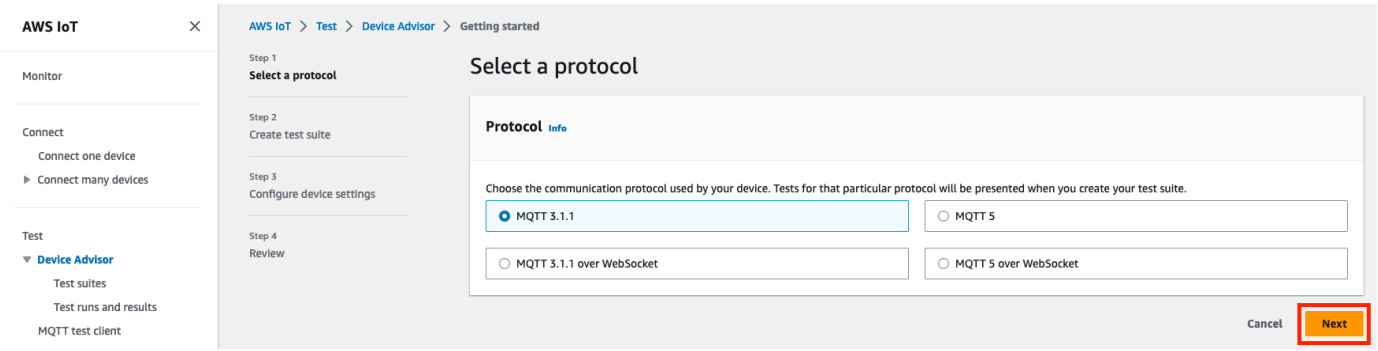


2. [Device Advisor の開始方法] ページでは、テストスイートを作成し、デバイスに対してテストを実行するために必要な手順の概要を説明しています。アカウントの Device Advisor テストエンドポイントもここで見つけることができます。このテストエンドポイントに接続するには、テストに使用するデバイスのファームウェアまたはソフトウェアを設定する必要があります。

   このチュートリアルを完了するには、まずモノと証明書を作成します。[仕組み] の情報を確認したら、[次へ] を選択します。

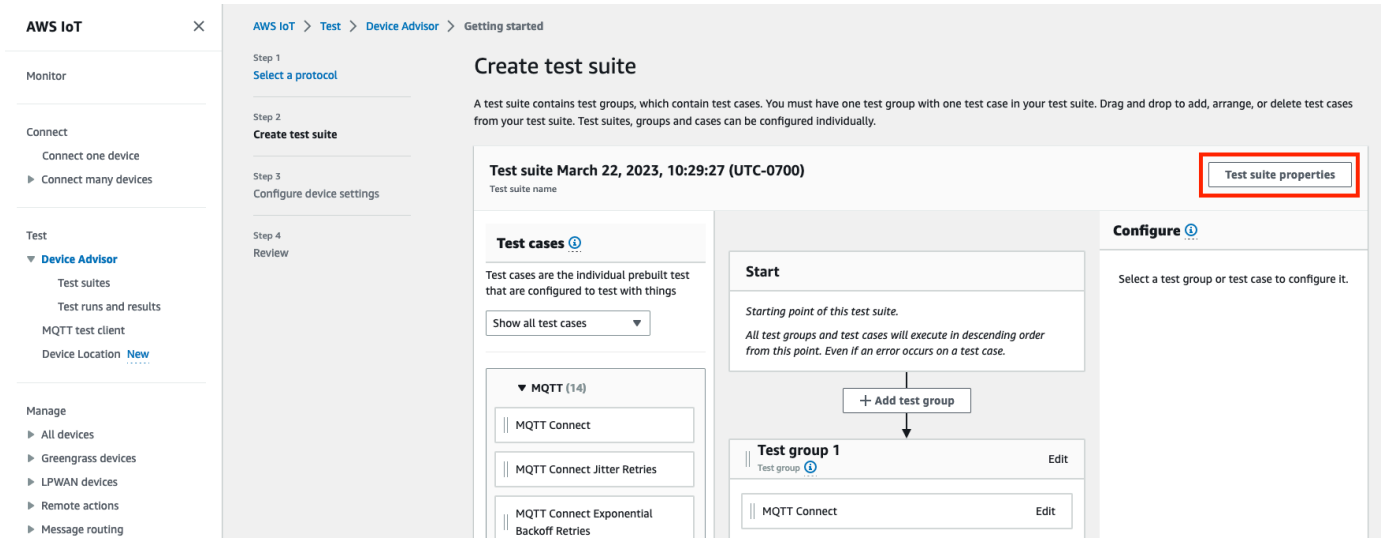


3. [ステップ 1: プロトコルの選択] で、表示されるオプションからプロトコルを選択します。その後、[Next] を選択します。



4. ステップ 2 では、カスタムテストスイートの作成および設定を行います。カスタムテストスイートには少なくとも 1 つのテストグループがなければならず、各テストグループには少なくとも 1 つのテストケースが必要です。使用を開始できるように、MQTT Connect テストケースを追加しました。

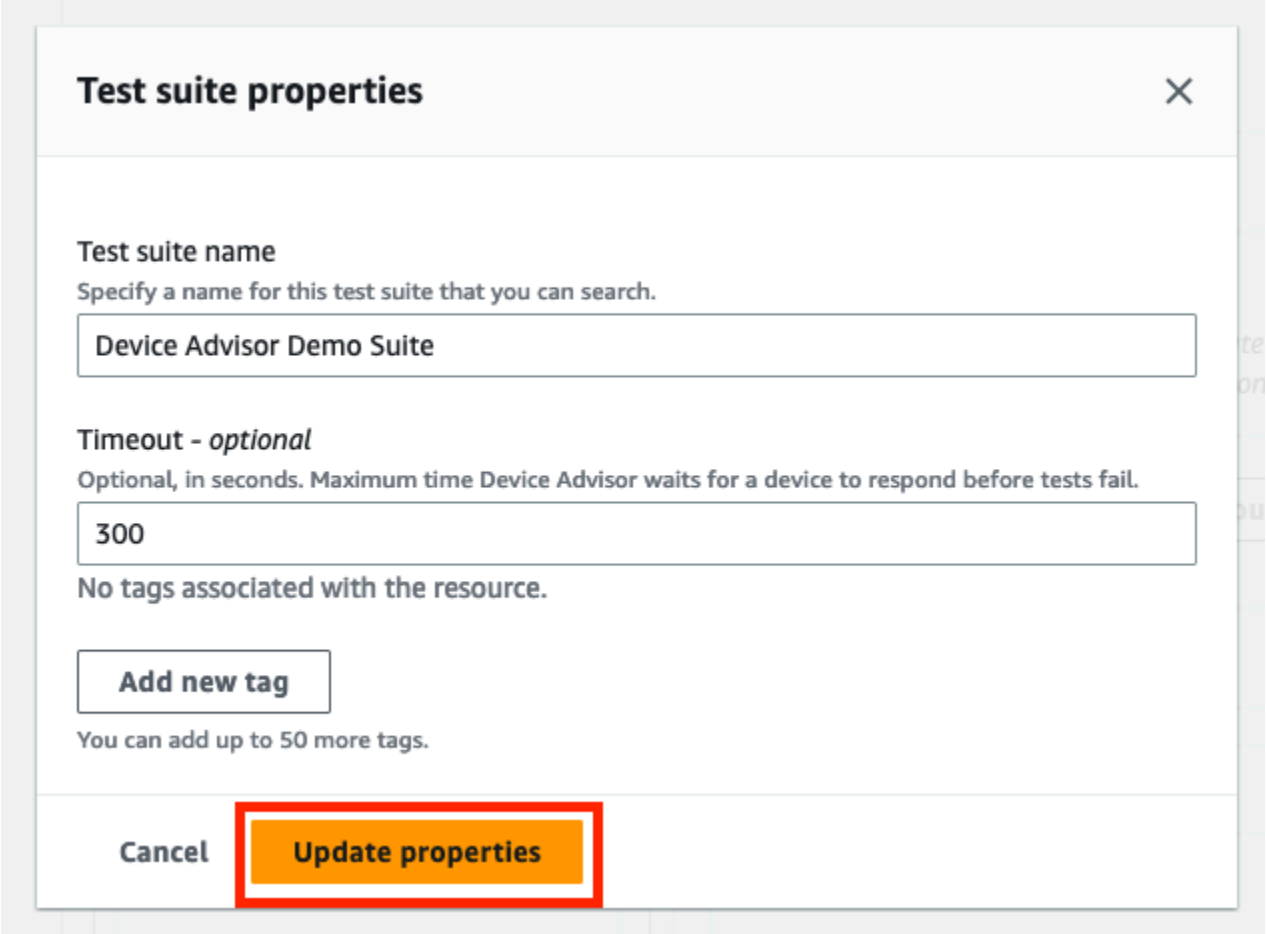
   [Test suite properties] (テストスイートのプロパティ) を選択します。



テストスイートを作成するときは、テストスイートのプロパティを指定します。設定できるスイートレベルのプロパティは次の通りです。

- テストスイート名: テストスイートの名前を入力します。
- [タイムアウト] (オプション): 現在のテストスイートの各テストケースの秒単位でのタイムアウト。タイムアウト値を指定しない場合、デフォルト値を使用します。
- [Tags] (タグ) (オプション): テストスイートにタグを追加します。

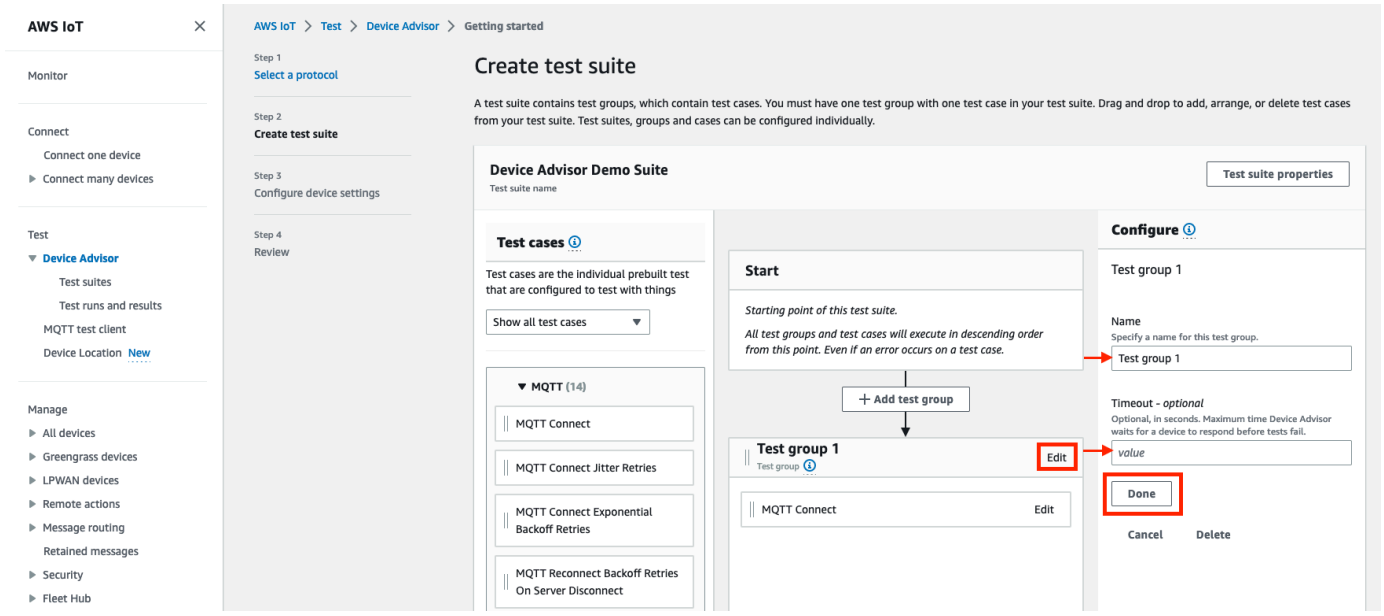
完了したら、[Update properties] (プロパティの更新) を選択します。



5. (オプション) テストスイートグループの設定を更新するには、テストグループ名の横にある [編集] ボタンを選択します。

   - 名前: テストスイートグループのカスタム名を入力します。
   - [タイムアウト] (オプション): 現在のテストスイートの各テストケースの秒単位でのタイムアウト。タイムアウト値を指定しない場合、デフォルト値を使用します。

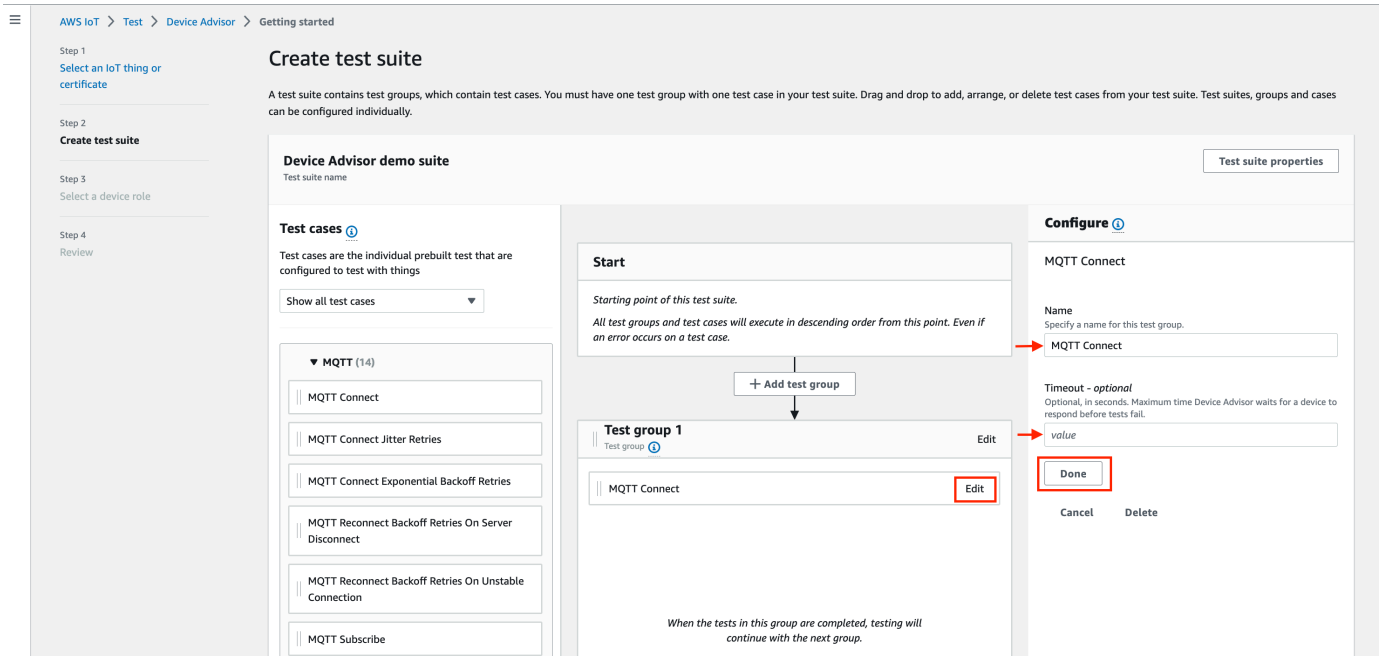
   終了したら、[完了] を選択して続行します。



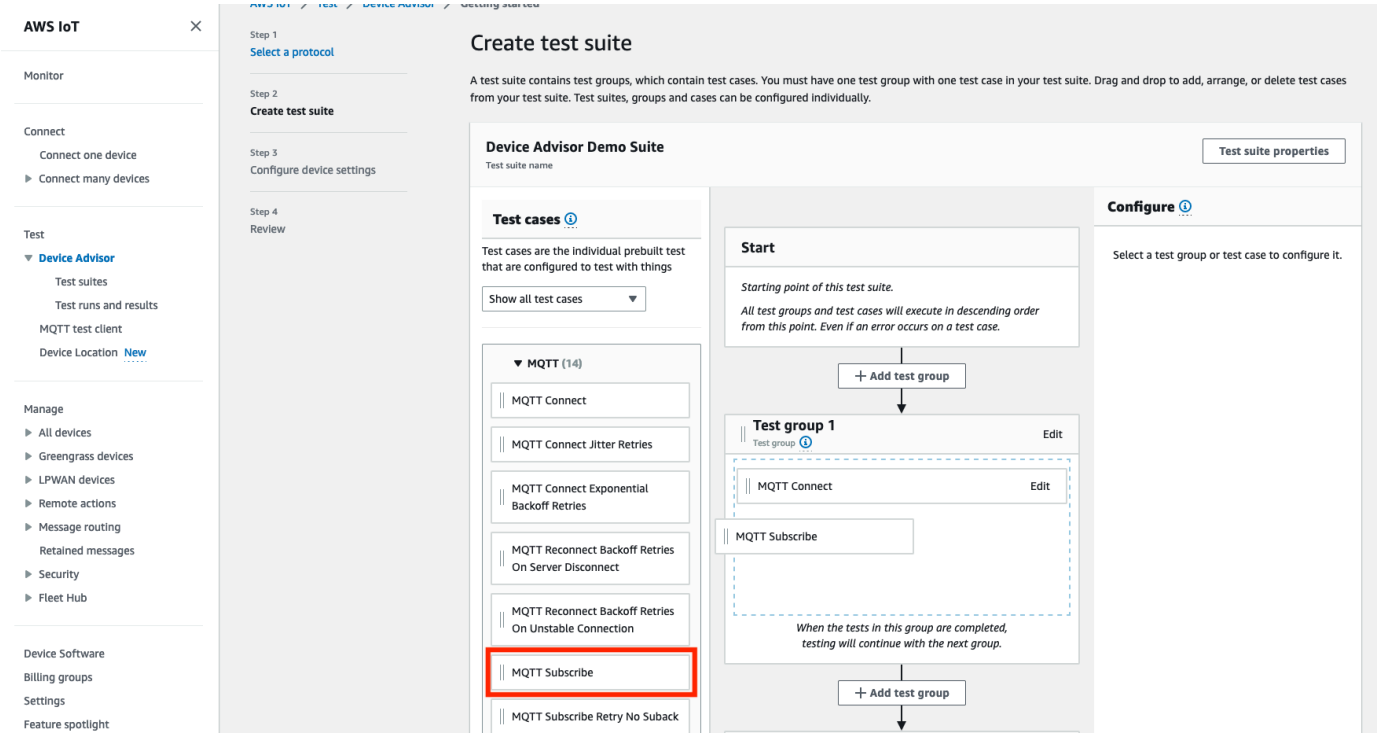
6. (オプション) テストケースの設定を更新するには、テストケース名の横にある [編集] ボタンを選択します。

   - 名前: テストスイートグループのカスタム名を入力します。
   - [タイムアウト] (オプション): 選択したテストケースのタイムアウト (秒)。タイムアウト値を指定しない場合、デフォルト値を使用します。

   終了したら、[完了] を選択して続行します。

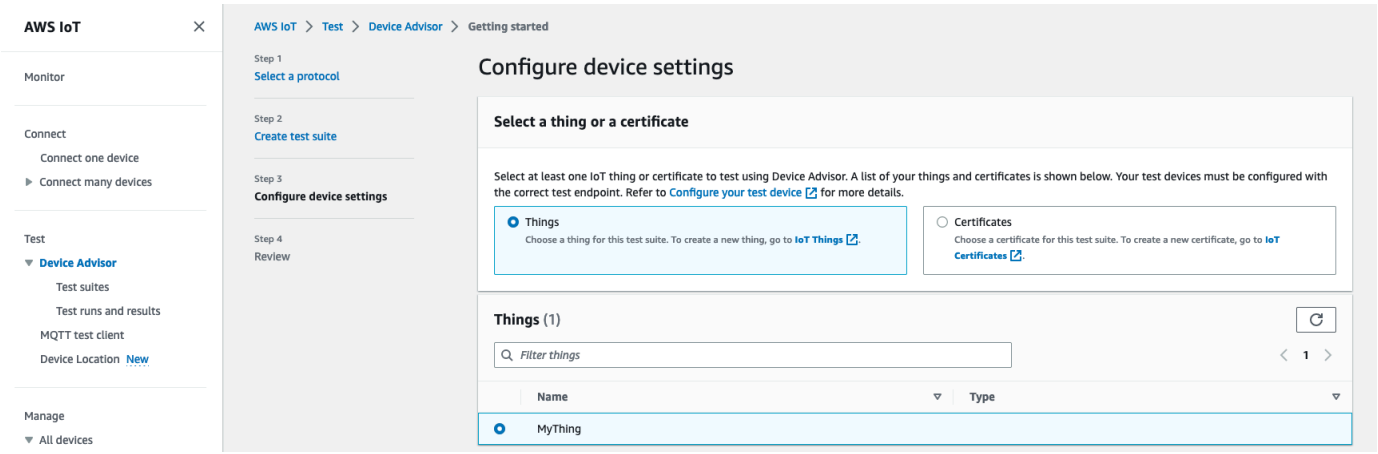


7. (オプション) テストスイートにさらにテストグループを追加するには、[テストグループの追加] を選択し、ステップ 5 の手順に従います。
8. (オプション) テストケースをさらに追加するには、[テストケース] セクションのテストケースを、任意のテストグループにドラッグします。



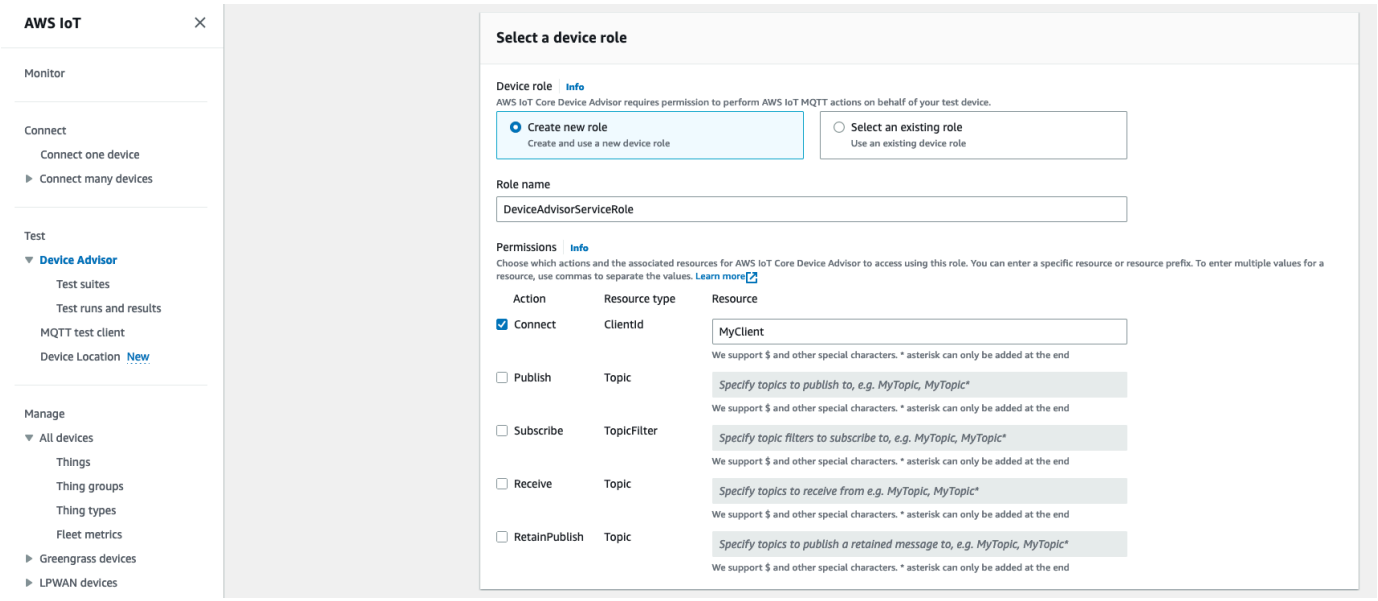
9. テストグループとテストケースの順序を変更できます。変更するには、リストされているテストケースをリストの上または下にドラッグします。Device Advisor は、リストされている順序でテストを実行します。

   テストスイートを設定したら、[Next] (次へ) を選択します。
10. ステップ 3 で、Device Advisor を使用してテストする AWS IoT モノまたは証明書を選択します。既存のモノまたは証明書がない場合は、セットアップ を参照してください。

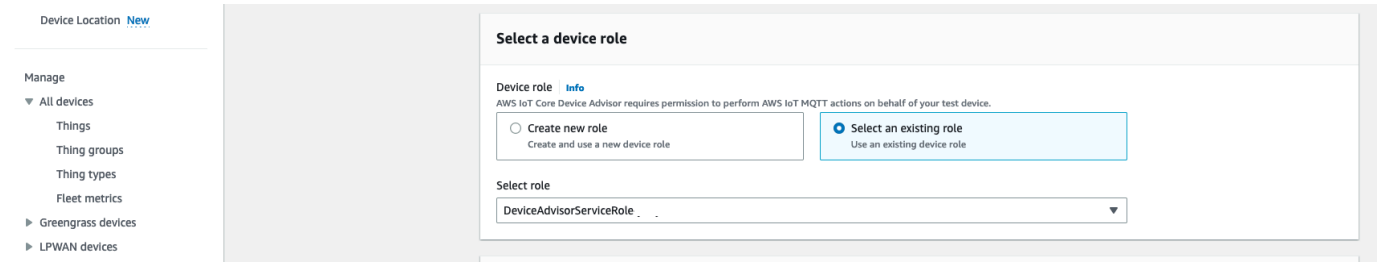


11. Device Advisor がテストデバイスに代わって AWS IoT MQTT アクションを実行するために使用するデバイスロールを設定できます。[MQTT Connect] テストケースの場合のみ、[接続] アクションが自動的に選択されます。これは、デバイスロールがテストスイートを実行するためにこの権限を必要とするためです。他のテストケースでは、対応するアクションが選択されます。

    選択した各アクションのリソース値を指定します。例えば、[接続] アクションでは、デバイスと Device Advisor エンドポイントの接続に使用するクライアント ID を指定します。カンマ区切りの値を使用して複数の値を指定したり、値のプレフィックスにワイルドカード (*) 文字を使用したりできます。例えば、`MyTopic` で始まる任意のトピックで発行するためのアクセス許可を付与する場合は、リソース値として「**`MyTopic*`**」を指定できます。



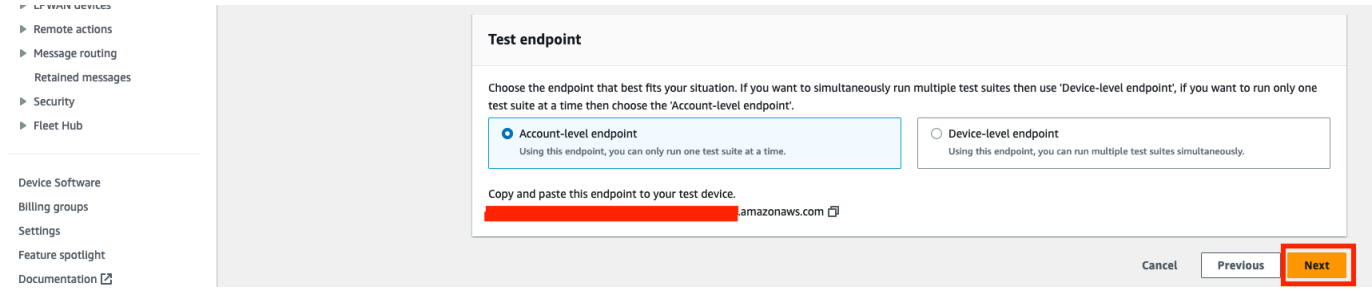
    セットアップで以前に作成したデバイスロールを使用するには、[既存のロールを選択] を選択します。次に、[ロールの選択] でデバイスロールを選択します。



    提供されている 2 つのオプションのいずれかを使用してデバイスロールを設定し、[次へ] を選択します。

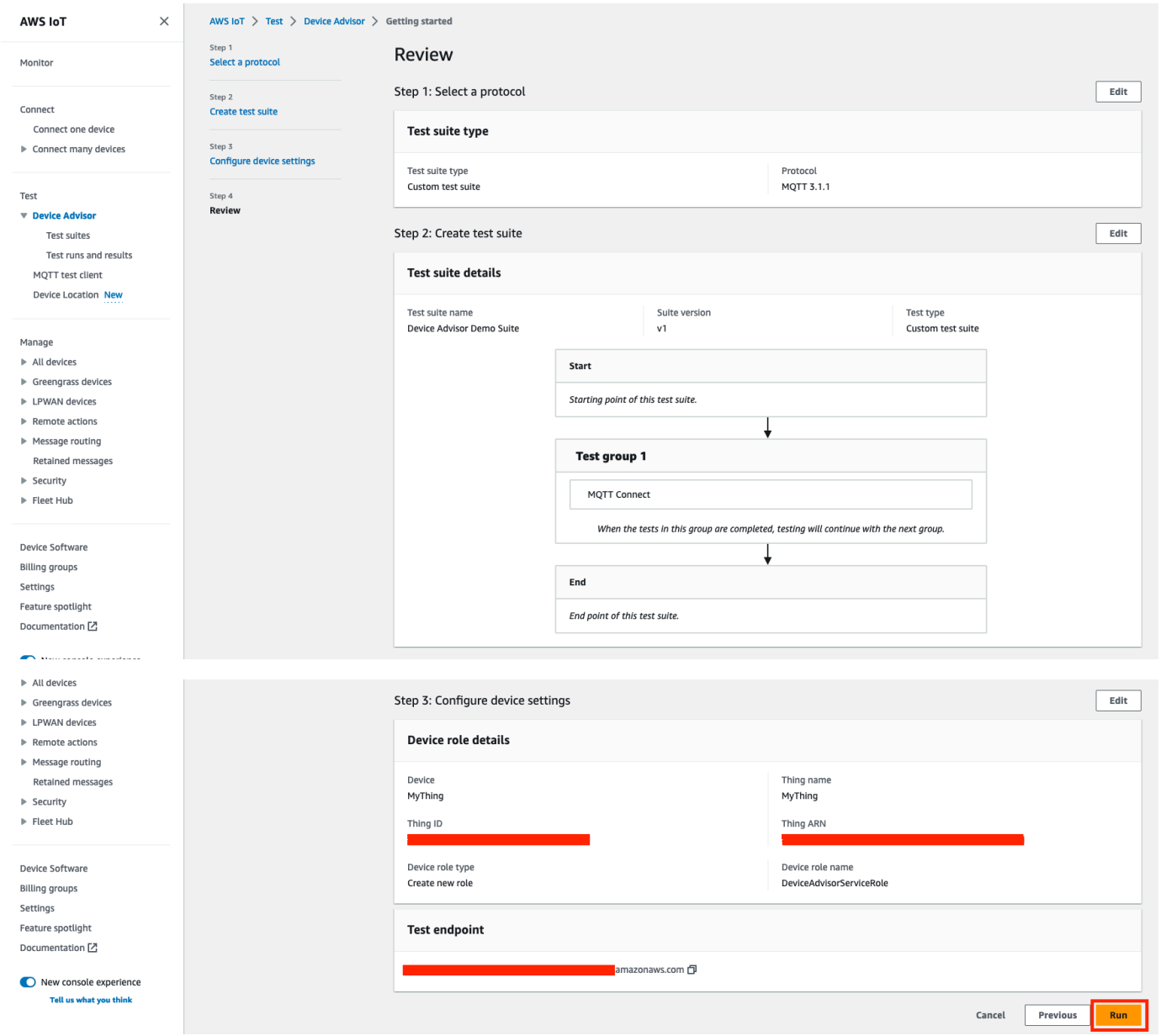
12. [テストエンドポイント] セクションで、ユースケースに最適なエンドポイントを選択します。同じ で複数のテストスイートを同時に実行するには AWS アカウント、デバイスレベルのエンド

ポイントを選択します。一度に 1 つのテストスイートを実行する場合、[アカウントレベルのエンドポイント] を選択します。

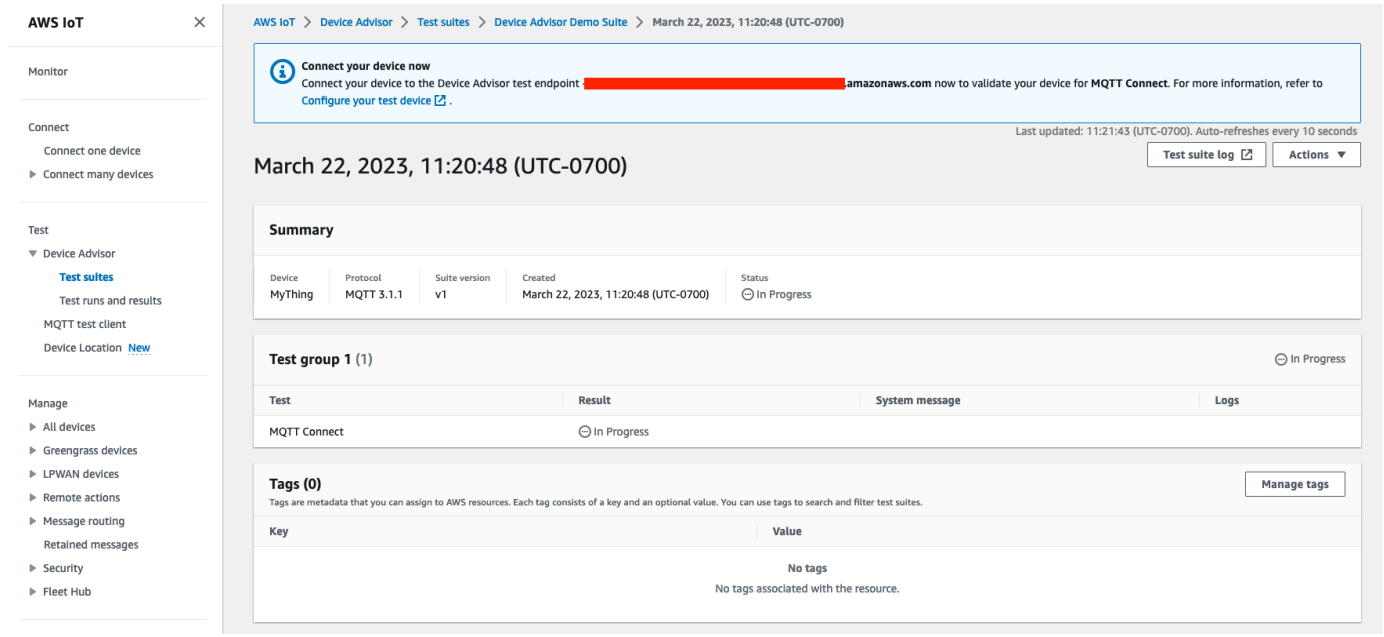


13. [ステップ 4] では、選択したテストデバイス、テストエンドポイント、テストスイート、および設定したテストデバイスロール設定の概要が表示されます。セクションに変更を加える場合は、編集するセクションの [編集] ボタンを選択します。テスト構成を確認したら、[実行] を選択してテストスイートを作成し、テストを実行します。

> **Note**
>
> 最良の結果を得るために、テストスイートの実行を開始する前に、選択したテストデバイスを Device Advisor テストエンドポイントに接続できます。1 ~ 2 分間、デバイスが 5 秒ごとにテストエンドポイントへの接続を試行するメカニズムを構築することをお勧めします。



14. ナビゲーションペインの [テスト] の下で、[Device Advisor]、[テストの実行と結果] の順に選択します。実行の詳細とログを表示するには、実行を開始したテストスイートを選択します。



15. スイート実行の Amazon CloudWatch ログにアクセスするには:

    - [Test suite log] (テストスイートのログ) を選択して、テストスイートの実行の CloudWatch ログを表示します。
    - テストケース固有の CloudWatch ログを表示するには、テストケースの [Test case log] (テストケースログ) を選択します。

16. テスト結果に基づいて、すべてのテストに合格するまでデバイスのトラブルシューティングを行います。

Device Advisor ワークフロー

このチュートリアルでは、カスタムテストスイートを作成し、コンソールでテストするデバイスに対してテストを実行する方法について説明します。テストが完了したら、テスト結果と詳細ログを表示できます。

前提条件

このチュートリアルを開始する前に、「設定」で説明されている手順を完了してください。

テストスイート定義を作成する

まず、 AWS SDK をインストールします。

rootGroup の構文

ルートグループは、テストスイートに含めるテストケースを指定する JSON 文字列です。また、これらのテストケースに必要な構成も指定します。ルートグループを使用して、テストスイートを任意の態様で構造化し、順序付けます。テストスイートの階層は次のとおりです。

```
test suite # test group(s) # test case(s)
```

テストスイートには少なくとも 1 つのテストグループがなければならず、各テストグループには少なくとも 1 つのテストケースが必要です。Device Advisor は、テストグループとテストケースを定義する順序でテストを実行します。

各ルートグループは、次の基本的な構造に従います。

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

ルートグループでは、グループに含まれる name、configuration、および tests を使用してテストスイートを定義します。tests グループには、個々のテストの定義が含まれています。各テストは、name、configuration およびそのテストのテストケースを定義する test ブロックを使用して定義します。最後に、各テストケースは id と version で定義されます。

各テストケース (test ブロック) の "id" フィールドと "version" フィールドの使用方法については、「Device Advisor テストケース」を参照してください。そのセクションには、使用可能な configuration 設定に関する情報も含まれています。

次のブロックは、ルートグループ設定の例です。この設定では、設定フィールドについての説明とともに、MQTT Connect Happy Case および MQTT Connect Exponential Backoff Retries テストケースを指定します。

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

テストスイート定義を作成するときに、ルートグループ設定を指定する必要があります。応答オブジェクトで返された suiteDefinitionId を保存します。この ID を使用して、テストスイートの定義情報を取得し、テストスイートを実行できます。

Java SDK の例を次に示します。

```
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()
)
```

テストスイート定義を取得する

テストスイート定義を作成した後、CreateSuiteDefinition API オペレーションの応答オブジェクトで suiteDefinitionId を受け取ります。

オペレーションが suiteDefinitionId を返すと、各グループ内に新しい id フィールドが表示され、ルートグループ内にテストケース定義が表示される場合があります。これらの ID を使用してテストスイート定義のサブセットを実行できます。

Java SDK の例:

```
response = iotDeviceAdvisorClient.GetSuiteDefinition(
    GetSuiteDefinitionRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .build()
```

```
)
```

テストエンドポイントを取得する

GetEndpoint API オペレーションを使用して、デバイスで使用されるテストエンドポイントを取得できます。テストに最適なエンドポイントを選択します。複数のテストスイートを同時に実行する場合は、thing ARN、certificate ARN、または device role ARN を指定してデバイスレベルのエンドポイントを使用します。単一のテストスイートを実行するには、GetEndpoint オペレーションに引数を指定せずに、アカウントレベルのエンドポイントを選択します。

SDK の例:

```
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())
```

テストスイートの実行を開始する

テストスイートの定義を作成し、Device Advisor のテストエンドポイントに接続するためにテストデバイスを設定したら、StartSuiteRun API を使用してテストスイートを実行します。

MQTT のお客様は、certificateArn または thingArn を使用してテストスイートを実行します。両方が設定されている場合、証明書がモノに属している場合は証明書が使用されます。

MQTT over WebSocket をご利用のお客様は、deviceRoleArn を使用してテストスイートを実行してください。指定されたロールがテストスイート定義で指定されたロールと異なる場合、指定されたロールは定義されたロールよりも優先されます。

.parallelRun() の場合、デバイスレベルのエンドポイントを使用して、1 つの AWS アカウントアカウントで複数のテストスイートを並列して実行する場合、true を使用します。

SDK の例:

```
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())
```

レスポンスから suiteRunId を保存します。これを使用して、このテストスイートの実行の結果を取得します。

テストスイートの実行を取得する

テストスイートの実行を開始したら、その進行状況を確認し、GetSuiteRun API を使用してその結果を確認できます。

SDK の例:

```
// Using the SDK, call the GetSuiteRun API.

response = iotDeviceAdvisorClient.GetSuiteRun(
GetSuiteRunRequest.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

テストスイートの実行を停止する

まだ進行中のテストスイートの実行を停止する場合は、StopSuiteRun API オペレーションを呼び出します。StopSuiteRun オペレーションを呼び出すと、サービスはクリーンアッププロセスを開始します。サービスがクリーンアップ処理を実行している間、テストスイートの実行ステータスが Stopping に更新されます。このクリーンアッププロセスには、数分以上かかることがあります。プロセスが完了すると、テストスイートの実行ステータスが Stopped に更新されます。テストの実行が完全に停止したら、別のテストスイートの実行を開始できます。前のセクションの説明どおりに GetSuiteRun API オペレーションを使用して、スイートの実行ステータスを定期的にチェックすることができます。

SDK の例:

```
// Using the SDK, call the StopSuiteRun API.
```

```
response = iotDeviceAdvisorClient.StopSuiteRun(
StopSuiteRun.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

成功した認定テストスイートの実行の認定レポートを取得する

正常に完了した認定テストスイートを実行すると、GetSuiteRunReport API オペレーションを使用して認定レポートを取得できます。この認定レポートを使用して、 AWS IoT Core 認定プログラムでデバイスを認定します。テストスイートが認定テストスイートであるかどうかを判断するには、intendedForQualification パラメータが true に設定されているかどうかを確認します。GetSuiteRunReport API オペレーションを呼び出した後、返された URL からレポートを最大 90 秒間ダウンロードできます。GetSuiteRunReport API オペレーションを最後に呼び出してから 90 秒を超える時間が経過した場合は、そのオペレーションを再度呼び出して有効な URL を取得します。

SDK の例:

```
// 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 の詳細コンソールワークフロー

このチュートリアルでは、カスタムテストスイートを作成し、コンソールでテストするデバイスに対してテストを実行します。テストが完了したら、テスト結果と詳細ログを表示できます。

チュートリアル

- 前提条件
- テストスイート定義を作成する
- テストスイートの実行を開始する
- テストスイートの実行を停止する (オプション)

- テストスイートの実行の詳細とログを表示する
- AWS IoT 認定レポートをダウンロードする

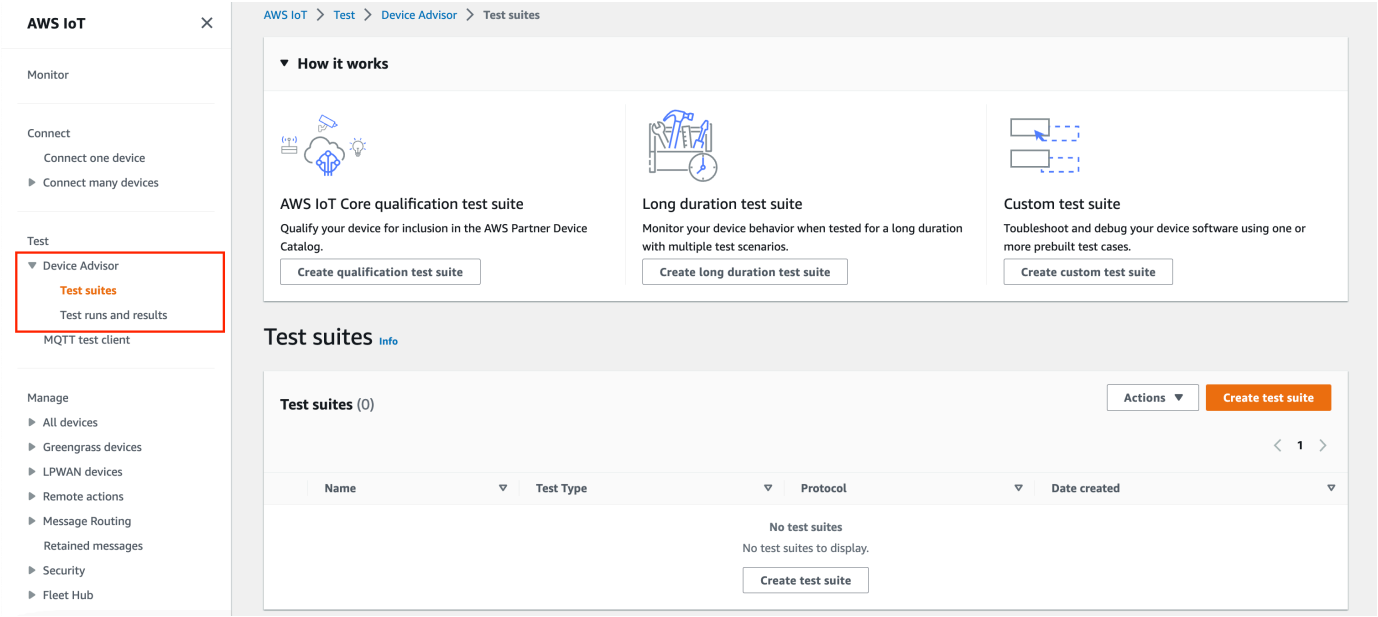
前提条件

このチュートリアルを完了するには、モノや証明書を作成する必要があります。

テストスイート定義を作成する

テストスイートスイートを作成して、デバイス用に実行して検証を実行できるようにします。

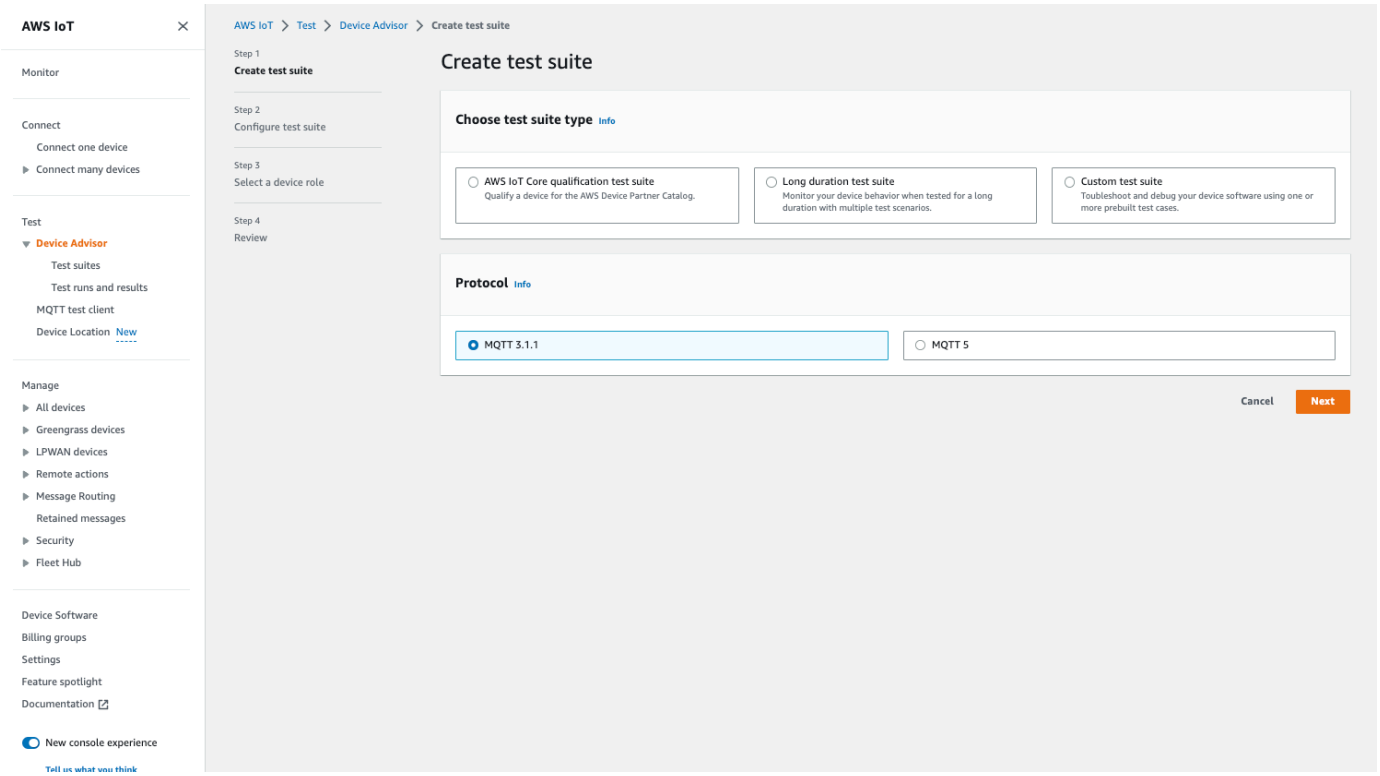
1. AWS IoT コンソールのナビゲーションペインで、[Test] (テスト)、[Device Advisor] の順に展開し、[Test suites] (テストスイート) を選択します。



   [Create test suite] (テストスイートの作成) を選択します。

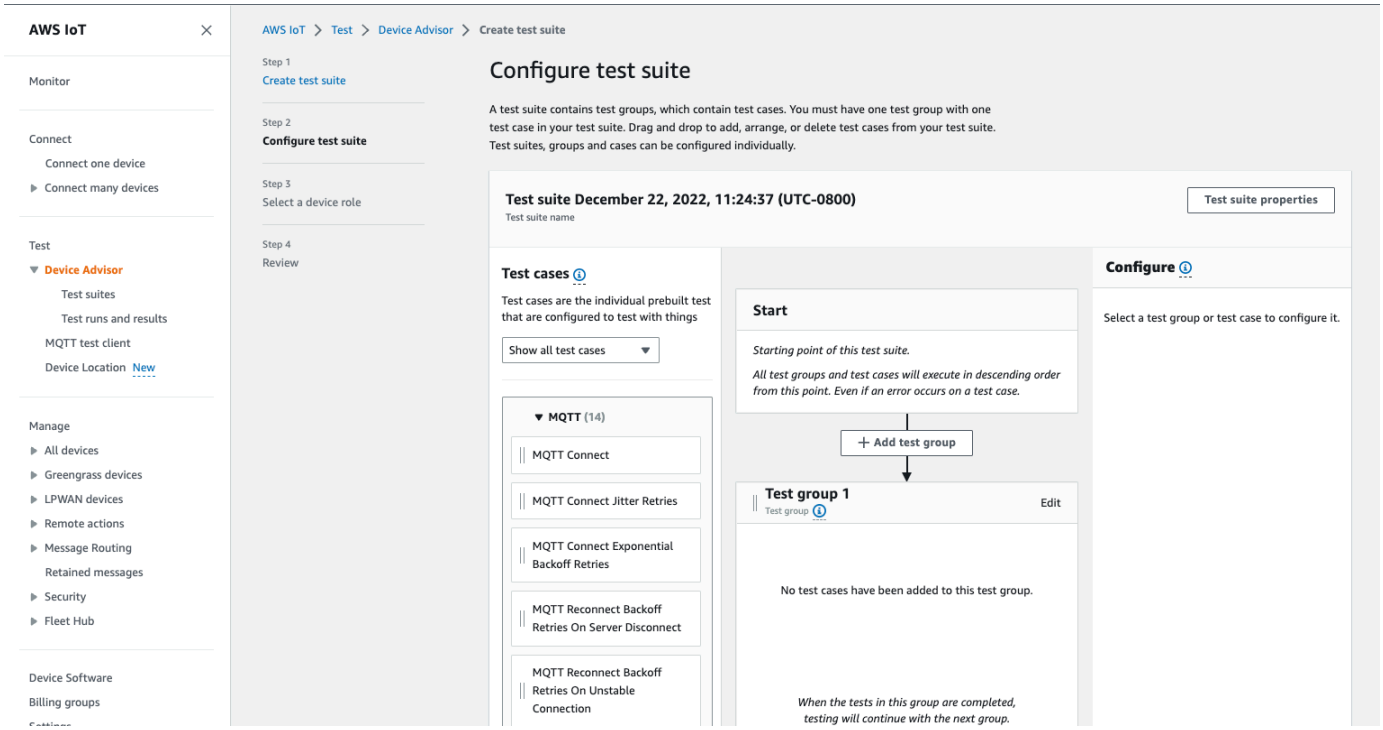
2. [Use the AWS Qualification test suite] または [Create a new test suite] のいずれかを選択します。

   プロトコルについては、[MQTT 3.1.1] または [MQTT 5] のいずれかを選択します。

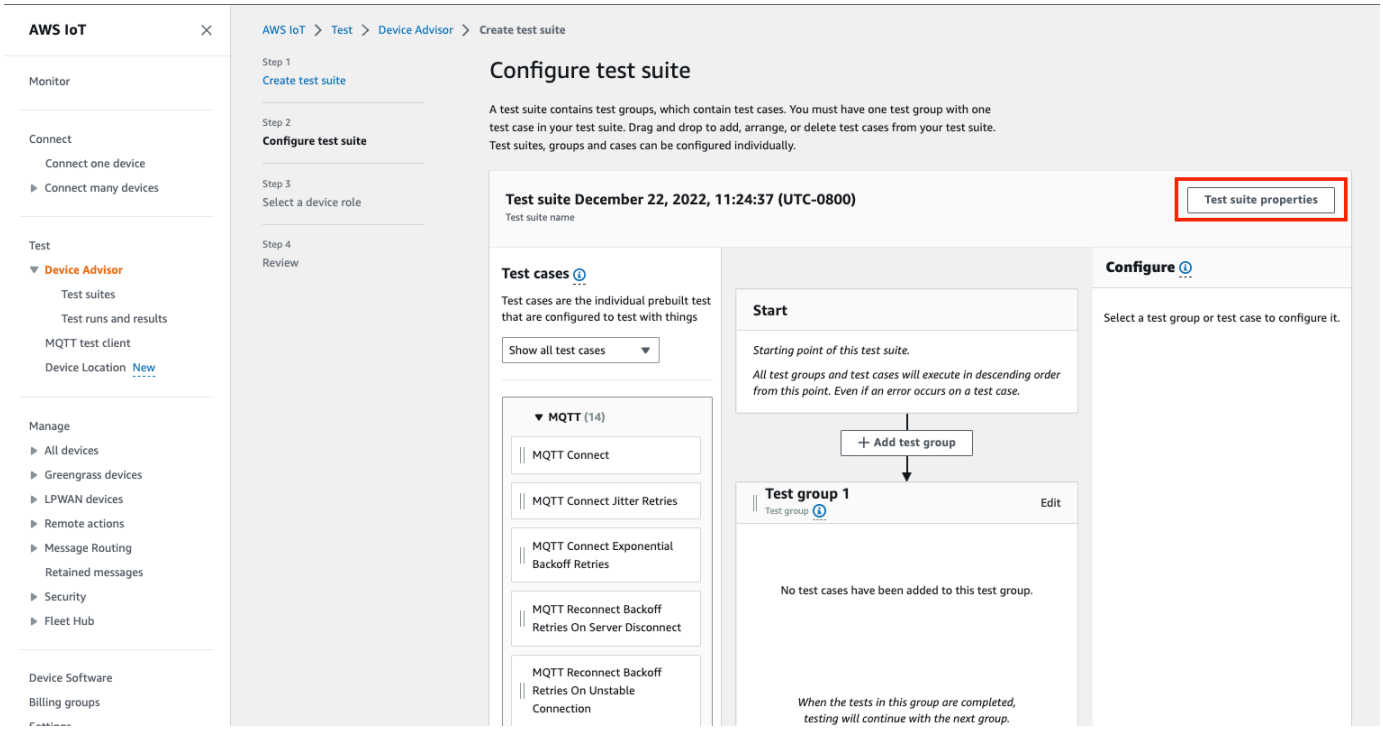


`Use the AWS Qualification test suite` を選択してデバイスを認定し、 AWS Partner Device Catalog に一覧表示します。このオプションを選択すると、 AWS IoT Core 認定プログラムへのデバイスの認定に必要なテストケースが事前に選択されます。テストグループおよびテストケースを追加または削除することはできません。テストスイートのプロパティを設定する必要があります。

`Create a new test suite` を選択して、カスタムテストスイートを作成および設定します。最初のテストとトラブルシューティングを行う場合は、このオプションから始めることをお勧めします。カスタムテストスイートには少なくとも 1 つのテストグループがなければならず、各テストグループには少なくとも 1 つのテストケースが必要です。このチュートリアルでは、このオプションを選択し、[Next] (次へ) を選択します。



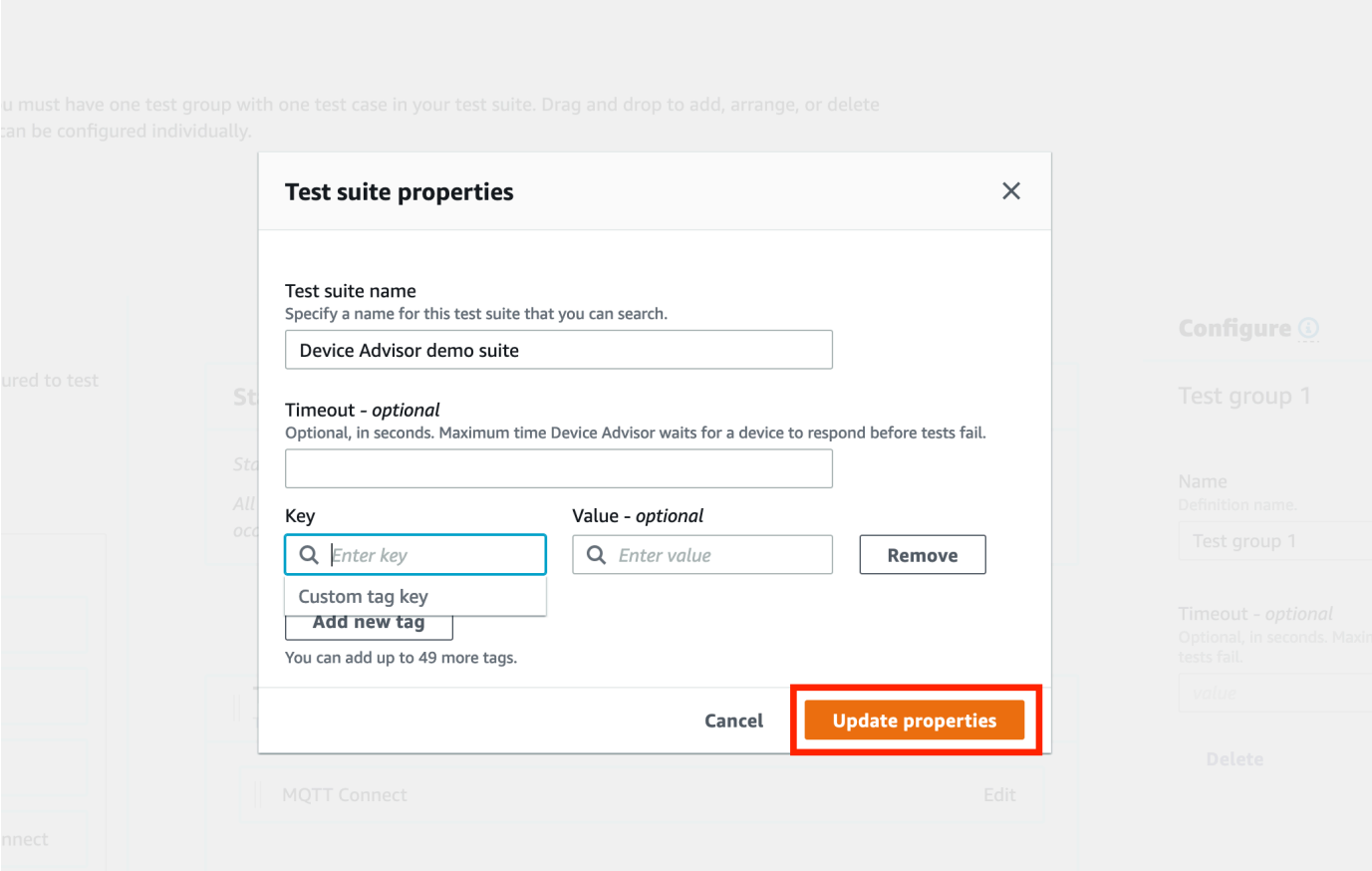
3. [Test suite properties] (テストスイートのプロパティ) を選択します。テストスイートを作成するときに、テストスイートのプロパティを作成する必要があります。



[Test suite properties] (テストスイートのプロパティ) で、次の情報を入力します。

- テストスイートの名前: カスタム名を使用してスイートを作成できます。

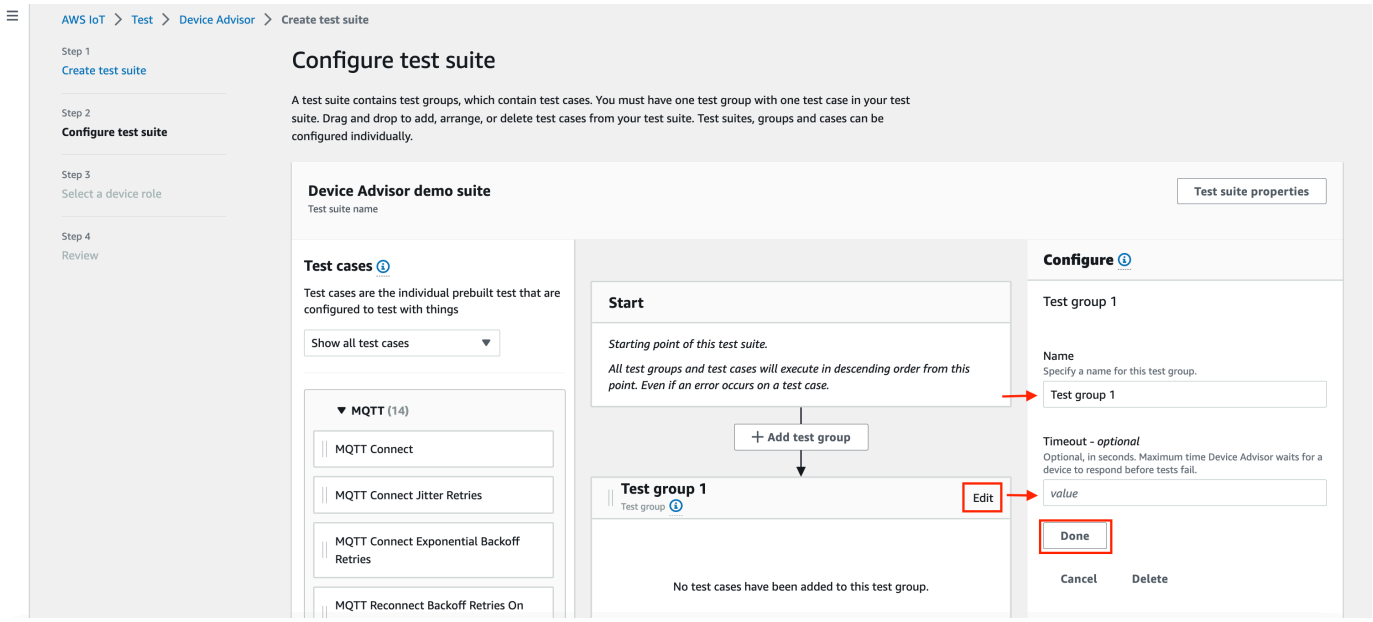
- タイムアウト (オプション): 現在のテストスイートの各テストケースの秒単位でのタイムアウト 。タイムアウト値を指定しない場合、デフォルト値を使用します。
- [Tags] (タグ) (オプション): テストスイートにタグを追加します。



完了したら、[Update properties] (プロパティの更新) を選択します。

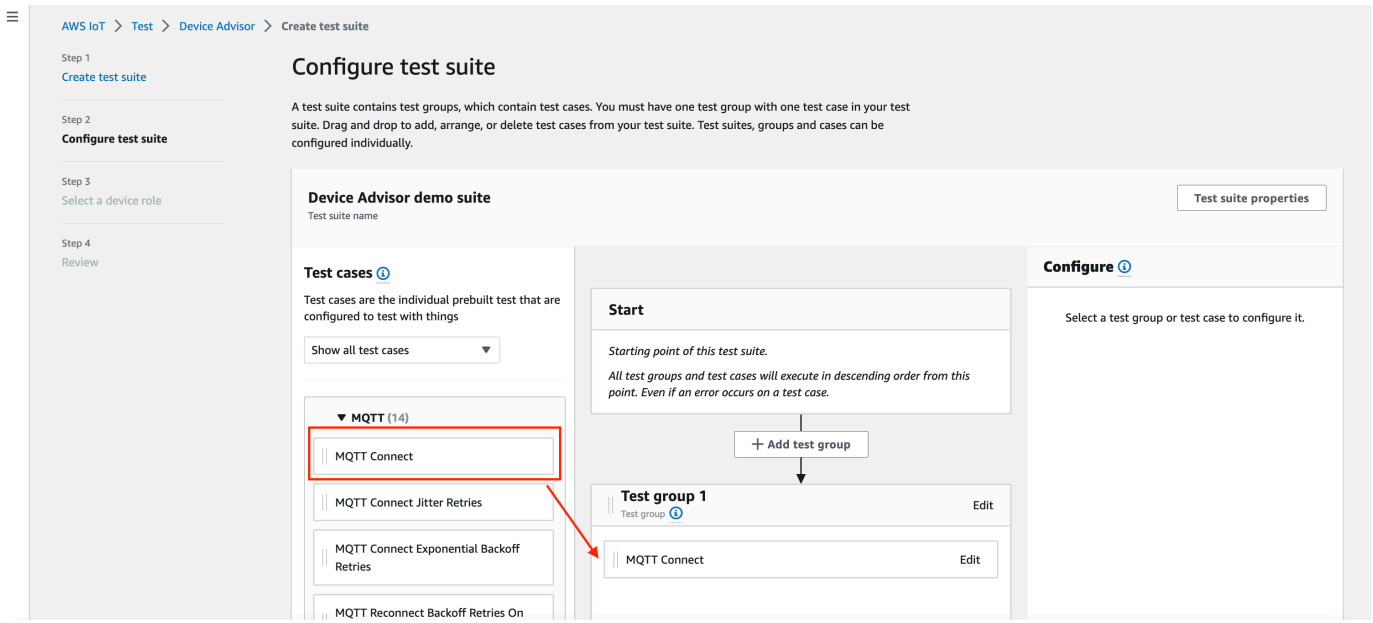
4. グループレベルの設定を変更するには、`Test group 1` で [Edit] (編集) を選択します。その後、名前を入力して、グループにカスタム名を付けます。

   オプションで、選択したテストグループの下に秒単位で [Timeout] (タイムアウト) 値を入力することもできます。タイムアウト値を指定しない場合、デフォルト値を使用します。



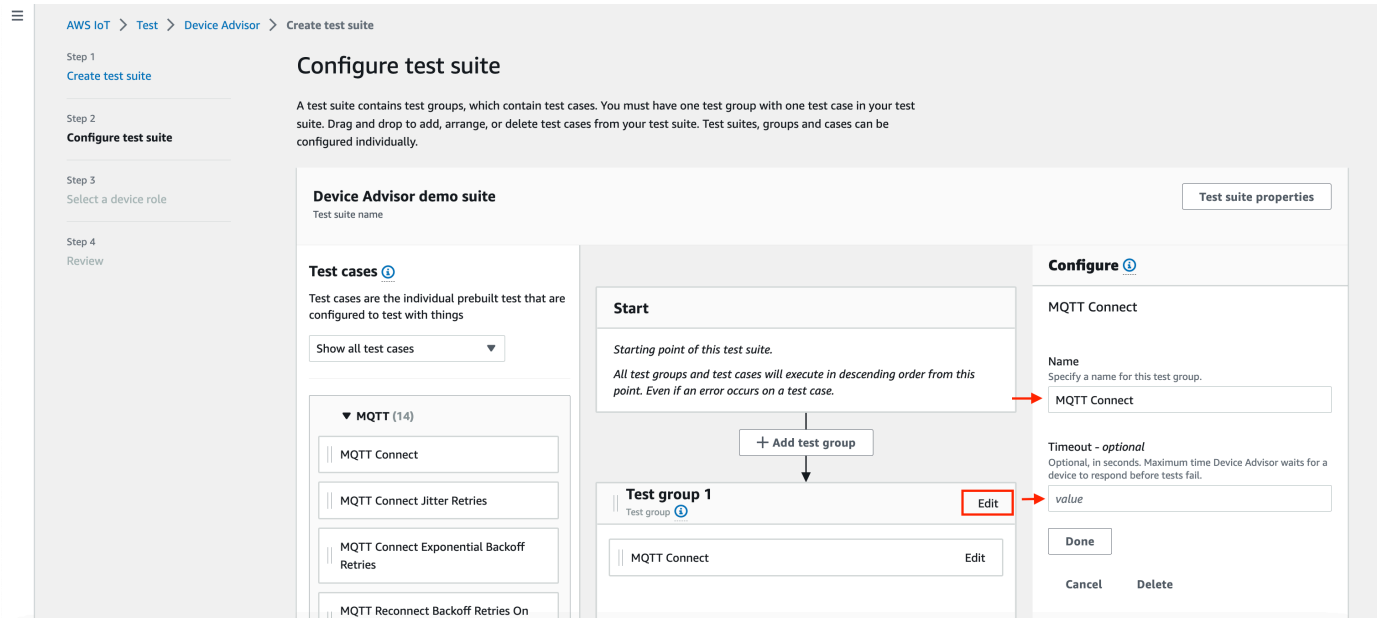
[Done] を選択します。

5. 使用可能ないずれかのテストケースを [Test cases] (テストケース) からテストグループにドラッグします。



6. テストグループに追加したテストケースのテストケースレベルの設定を変更するには、[Edit] (編集) を選択します。その後、名前を入力して、グループにカスタム名を付けます。

   オプションで、選択したテストグループの下に秒単位で [Timeout] (タイムアウト) 値を入力することもできます。タイムアウト値を指定しない場合、デフォルト値を使用します。



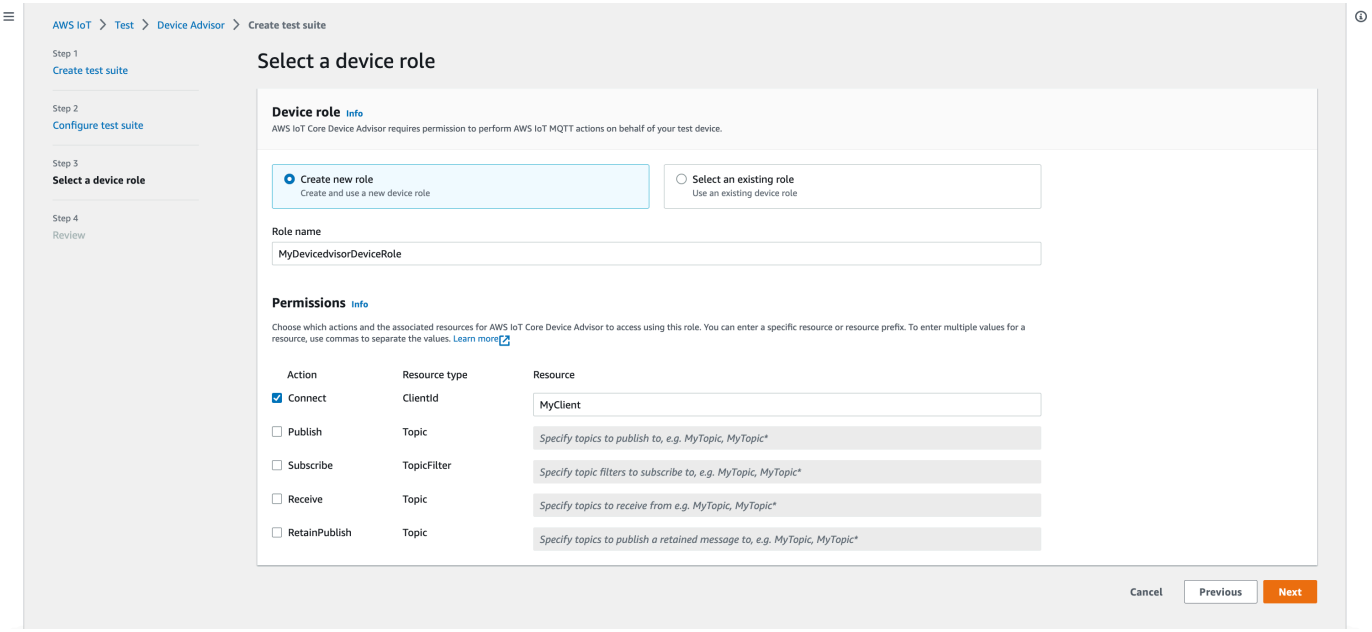
[Done] を選択します。

> **Note**
>
> テストスイートにさらにテストグループを追加するには、[Add test group] (テストグループの追加) を選択します。前の手順に従って、さらにテストグループを作成して設定するか、1 つ以上のテストグループにテストケースを追加します。テストケースを選択し、目的の位置にドラッグすることによって、テストグループとテストケースを並べ替えることができます。Device Advisor は、テストグループとテストケースを定義する順序でテストを実行します。

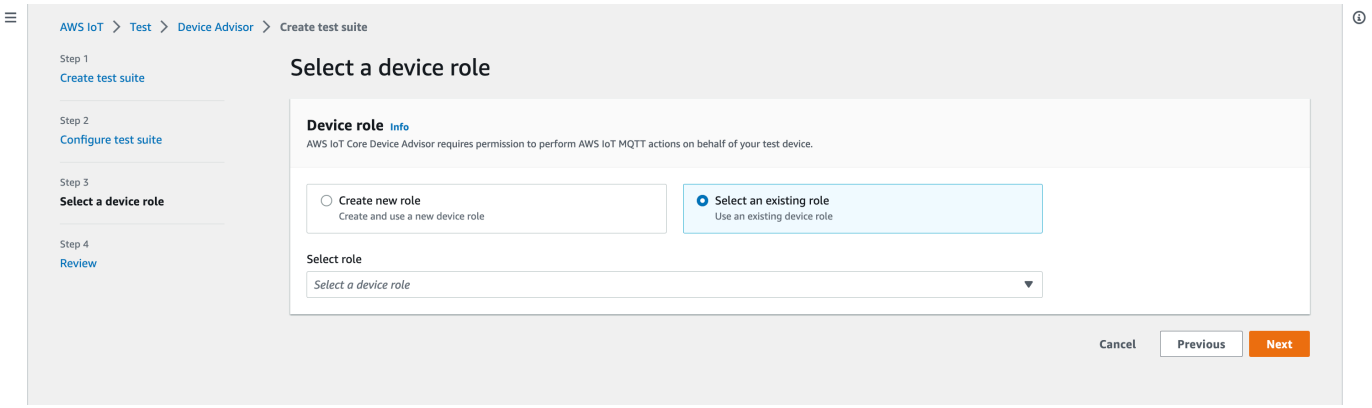
7. [次へ] を選択します。
8. ステップ 3 で、Device Advisor がテストデバイスに代わって AWS IoT MQTT アクションを実行するために使用するデバイスロールを設定します。

   ステップ 2 で MQTT Connect テストケースのみを選択した場合、[Connect] (接続) アクションが自動的にチェックされます。このテストスイートを実行するために、デバイスロールに対してそのアクセス許可が必要になるためです。他のテストケースを選択した場合、対応する必須アクションがチェックされます。各アクションのリソース値が提供されていることを確認します。例えば、[Connect] (接続) アクションでは、デバイスと Device Advisor エンドポイントの接続に使用するクライアント ID を指定します。カンマを使用して値を区切ることで、複数の値を指定できます。また、ワイルドカード (*) 文字を使用して、プレフィックス値を指定することもできま

す。例えば、MyTopic で始まる任意のトピックで発行するためのアクセス許可を付与する場合は、リソース値として「MyTopic*」を指定できます。



以前にデバイスロールを作成していて、そのロールを使用する場合は、[Select an existing role] (既存のロールを選択) を選択し、[Select role] (ロールを選択) でデバイスロールを選択します。



提供されている 2 つのオプションのいずれかを使用してデバイスロールを設定し、[Next] (次へ) をクリックします。

9. ステップ 4 では、各ステップで指定した設定が正確であることを確認します。特定のステップの指定した設定を編集するには、対応するステップの [Edit] (編集) を選択します。

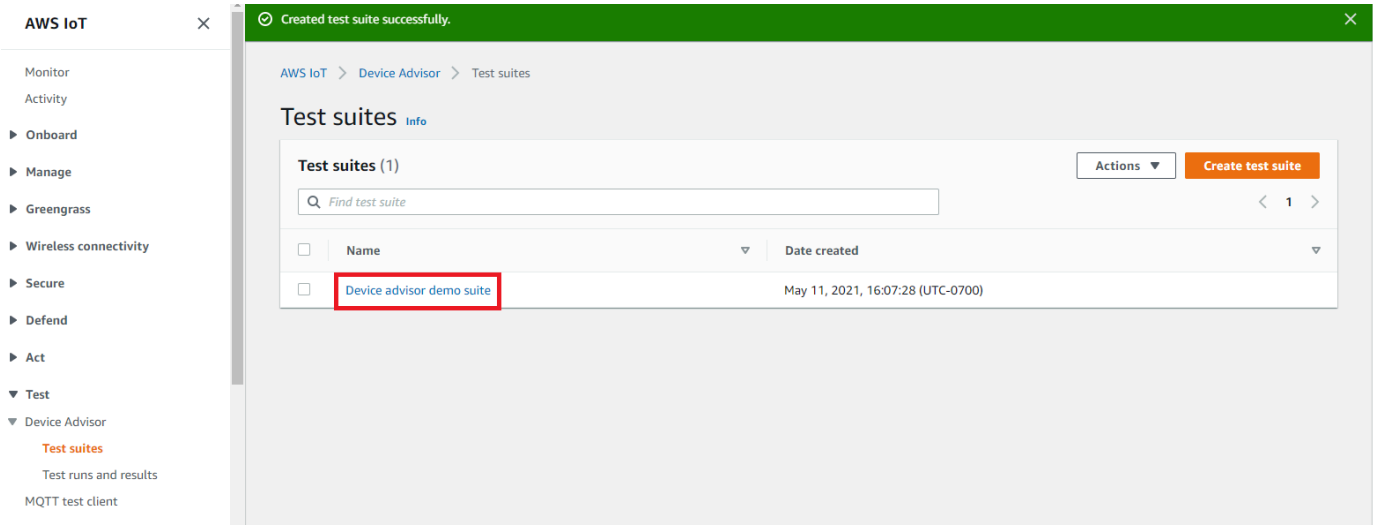
   設定を確認したら、[Create Test Suite] (テストスイートの作成) を選択します。

   テストスイートが正常に作成され、作成されたすべてのテストスイートを表示できる [Test suites] (テストスイート) ページにリダイレクトされます。

テストスイートの作成に失敗した場合は、テストスイート、テストグループ、テストケース、およびデバイスロールが、前述の指示に従って設定されていることを確認します。

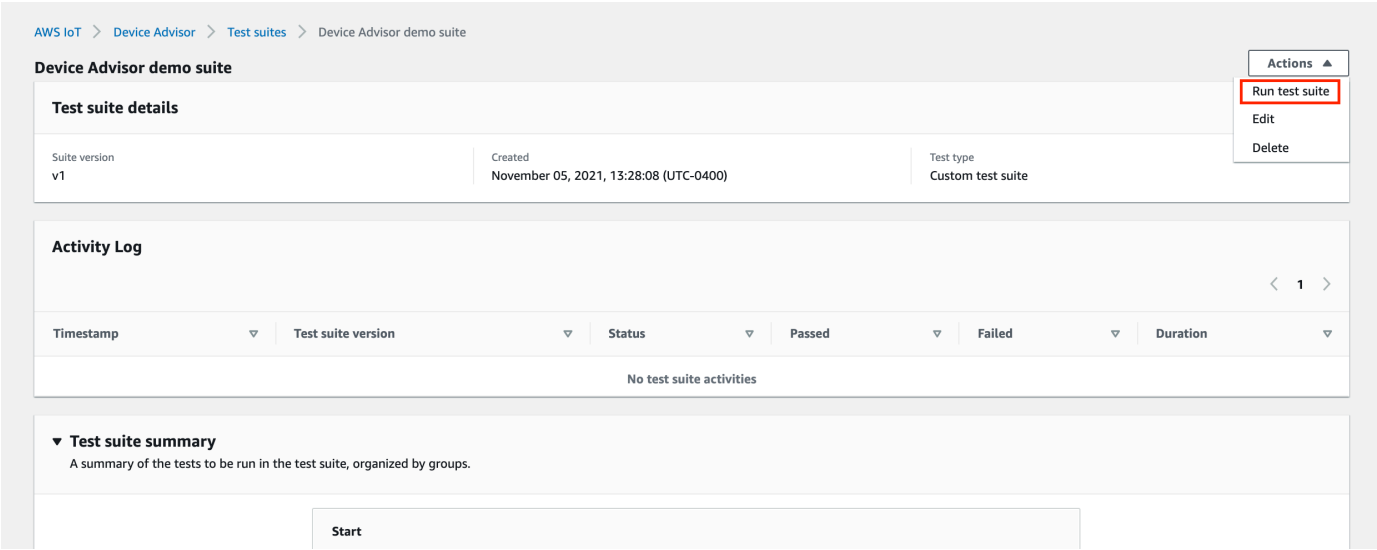
テストスイートの実行を開始する

1. AWS IoT コンソールのナビゲーションペインで、[Test] (テスト)、[Device Advisor] の順に展開し、[Test suites] (テストスイート) を選択します。
2. テストスイートの詳細を表示するテストスイートを選択します。



テストスイートの詳細ページには、テストスイートに関連するすべての情報が表示されます。

3. [Actions] (アクション)、[Run test suite] (テストスイートの実行) の順に選択します。

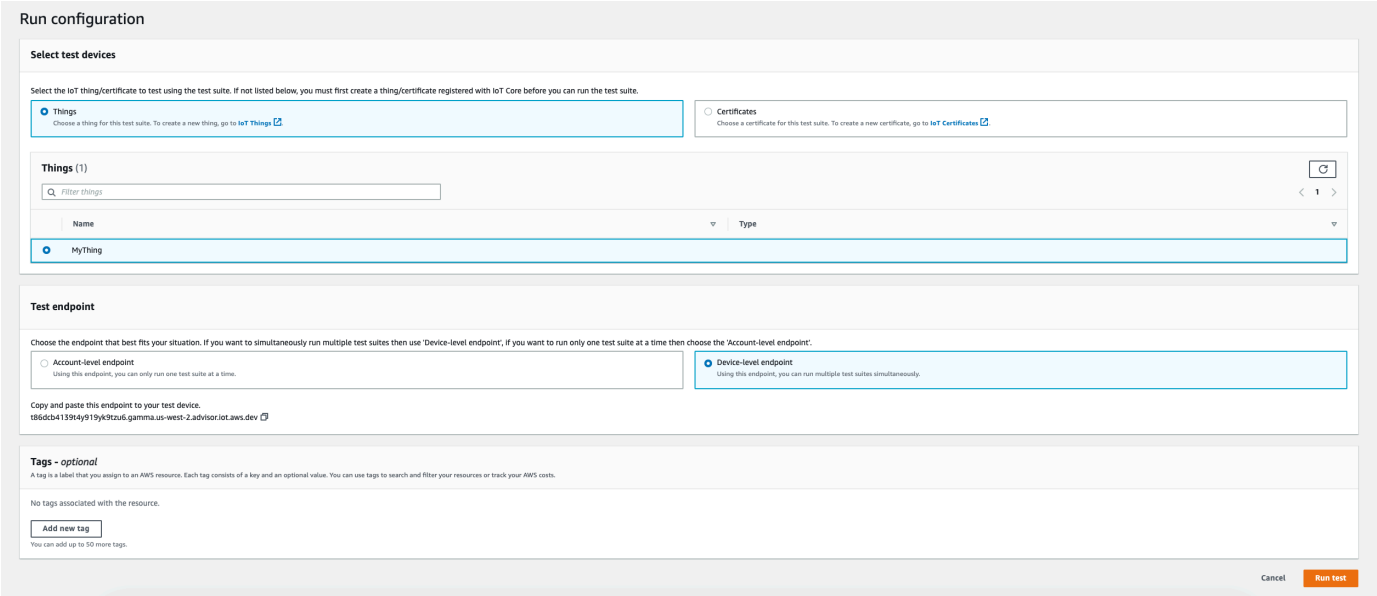


4. 実行設定では、Device Advisor を使用してテストする AWS IoT モノまたは証明書を選択する必要があります。既存のモノや証明書がない場合は、まずAWS IoT Core リソースを作成します。

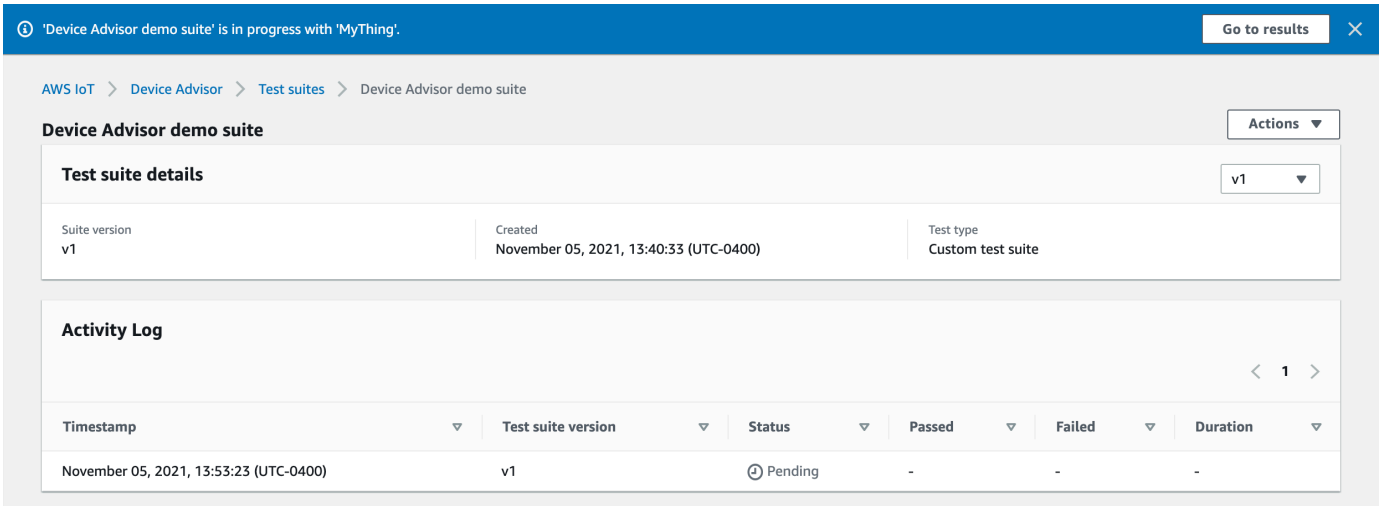
   [Test endpoint] (テストエンドポイント) セクションで、ケースに最適なエンドポイントを選択します。将来、同じ AWS アカウントを使用して複数のテストスイートを同時に実行する予定がある場合は、デバイスレベルのエンドポイントを選択します。別の方法として、一度に 1 つのテストスイートのみを実行する場合は、[Account-level endpoint] (アカウントレベルのエンドポイント) を選択します。

   選択した Device Advisor のテストエンドポイントでテストデバイスを設定します。

   モノまたは証明書を選択し、Device Advisor エンドポイントを選択したら、[Run test] (テストの実行) を選択します。

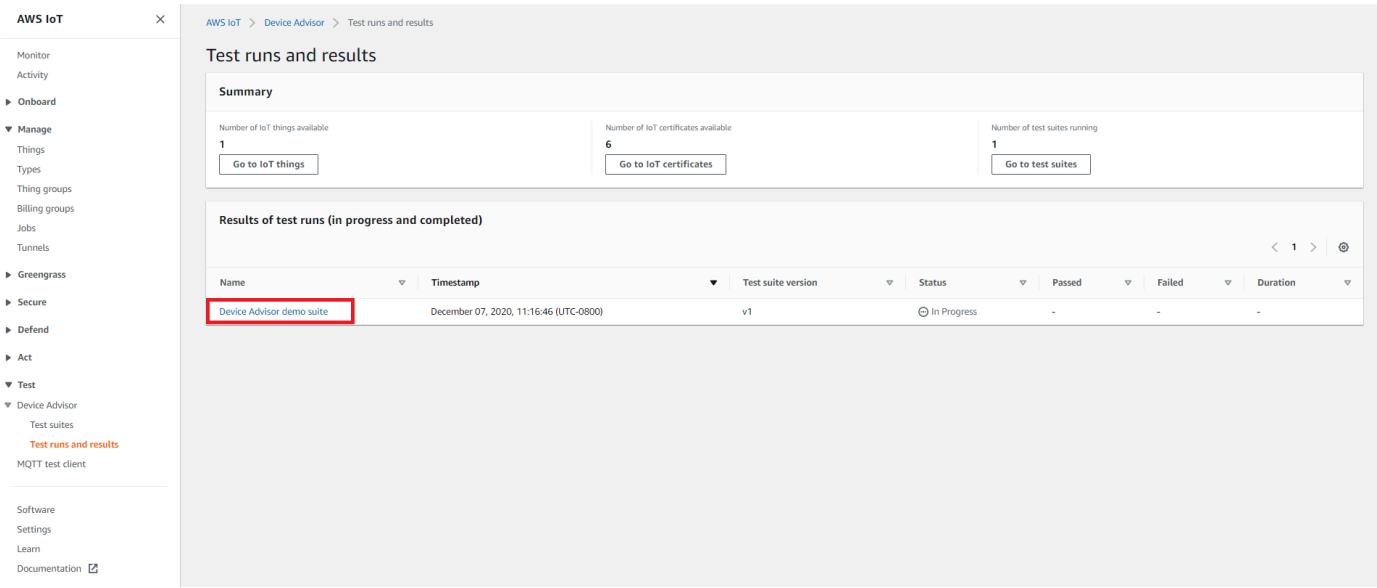


5. テスト実行の詳細を表示するには、上部にあるバナーの [Go to results] (結果に移動) を選択します。

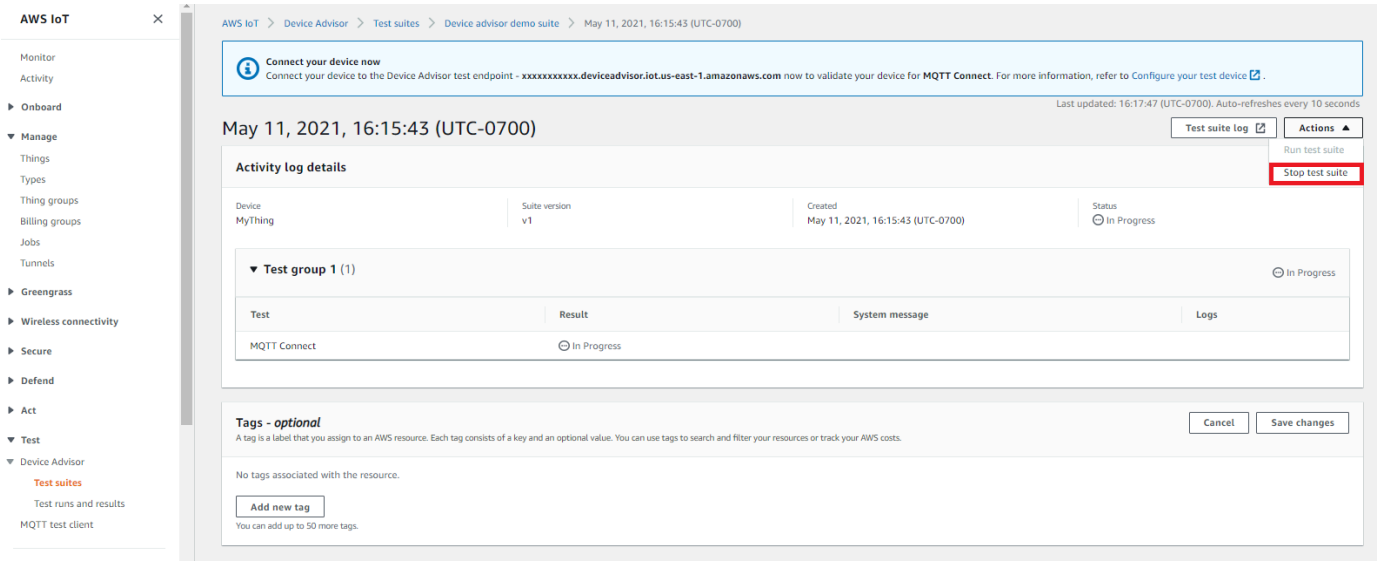


テストスイートの実行を停止する (オプション)

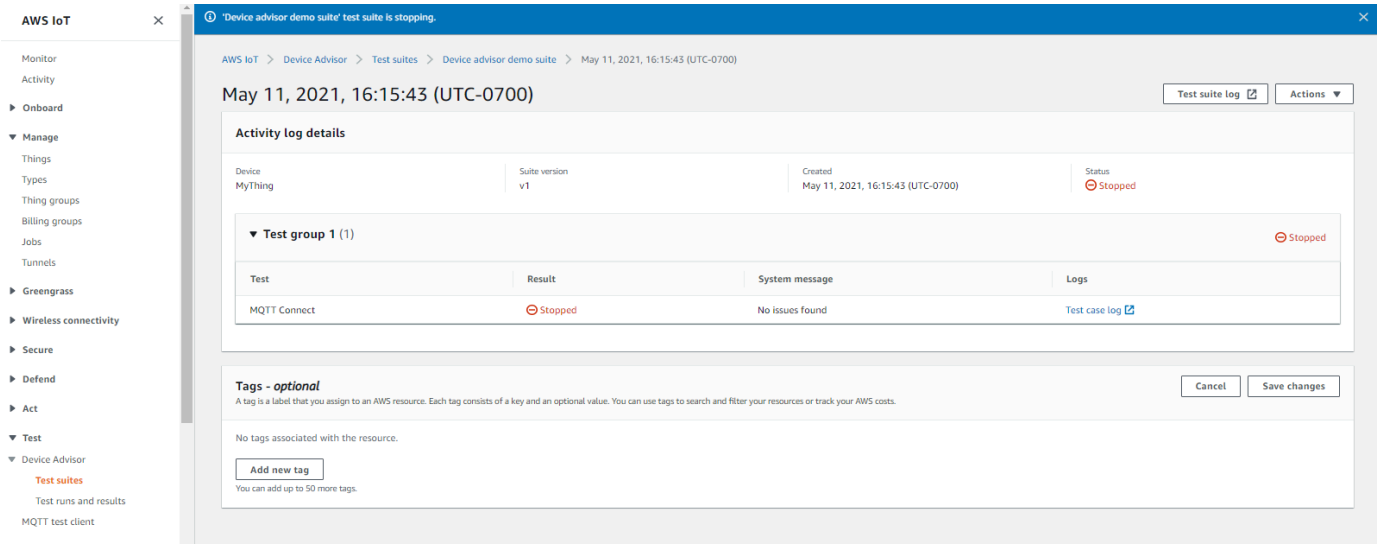
1. AWS IoT コンソールのナビゲーションペインで、[Test] (テスト)、[Device Advisor] の順に展開し、[Test runs and results] (テストの実行と結果) を選択します。
2. 停止する進行中のテストスイートを選択します。



3. [Actions] (アクション)、[Stop test suite] (テストスイートを停止) の順に選択します。



4. このクリーンアップ処理は完了までに数分かかります。クリーンアップ処理の実行中、テストの実行ステータスは STOPPING になります。クリーンアップ処理が完了し、テストスイートのステータスが STOPPED ステータスに変わった後に、新しいスイートの実行を開始します。



テストスイートの実行の詳細とログを表示する

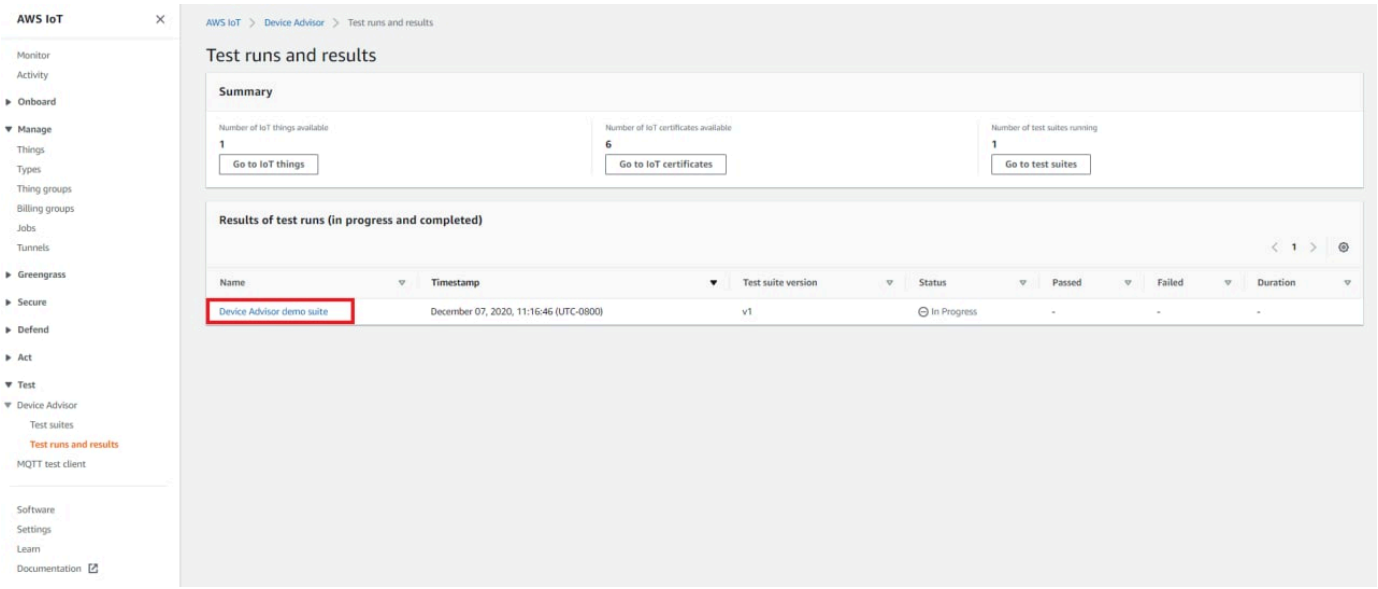
1. AWS IoT コンソールのナビゲーションペインで、[Test] (テスト)、[Device Advisor] の順に展開し、[Test runs and results.] (テストの実行と結果) を選択します。

   このページが表示されます。

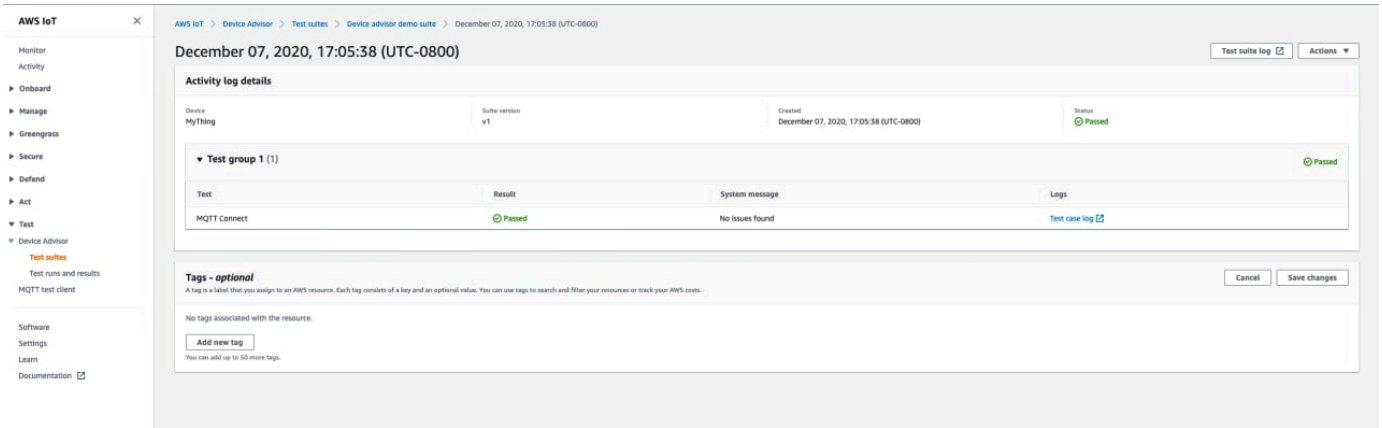
   - IoT モノの数
   - IoT 証明書の数

- 現在実行中のテストスイートの数
- 作成されたすべてのテストスイートの実行

2. 実行の詳細とログを表示するテストスイートを選択します。



実行の概要のページには、現在のテストスイート実行のステータスが表示されます。このページは 10 秒ごとに自動更新されます。1 ~ 2 分間、デバイスが 5 秒ごとにテストエンドポイントへの接続を試行するメカニズムを構築することをお勧めします。その後、自動化された方法で順番に複数のテストケースを実行できます。



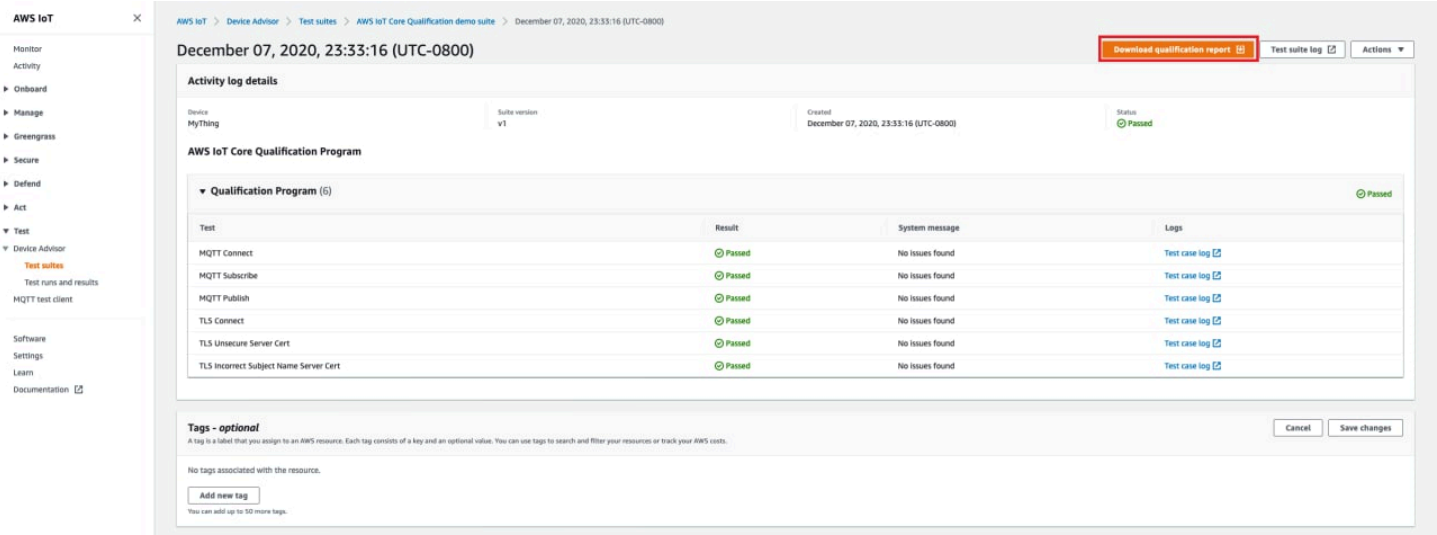
3. テストスイート実行の CloudWatch ログにアクセスするには、[Test suite log] (テストスイートログ) を選択します。

   テストケースの CloudWatch ログにアクセスするには、[Test case log] (テストケースログ) を選択します。

4. テスト結果に基づいて、すべてのテストに合格するまでデバイスのトラブルシューティングを行います。

AWS IoT 認定レポートをダウンロードする

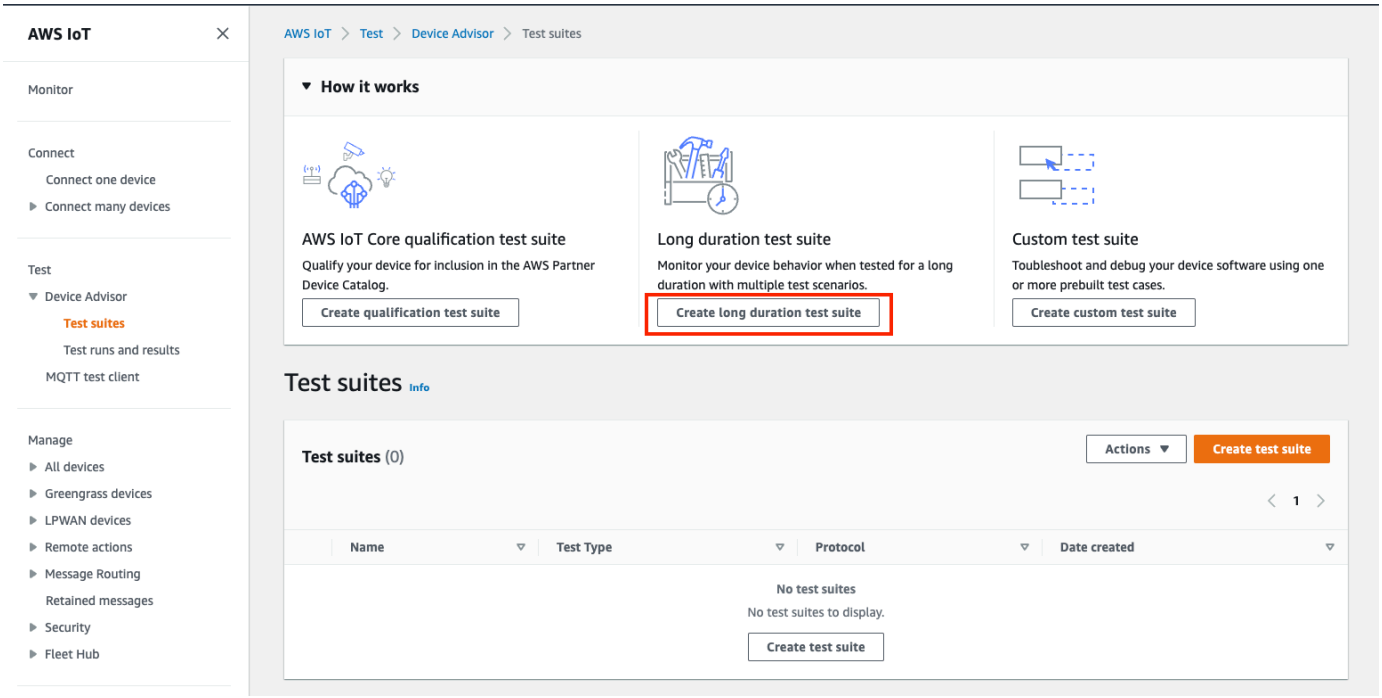
テストスイートの作成時に AWS IoT 認定テストスイートの使用オプションを選択し、認定テストスイートを実行できた場合は、テスト実行の概要ページで認定レポートのダウンロードを選択して認定レポートをダウンロードできます。



長期テストコンソールのワークフロー

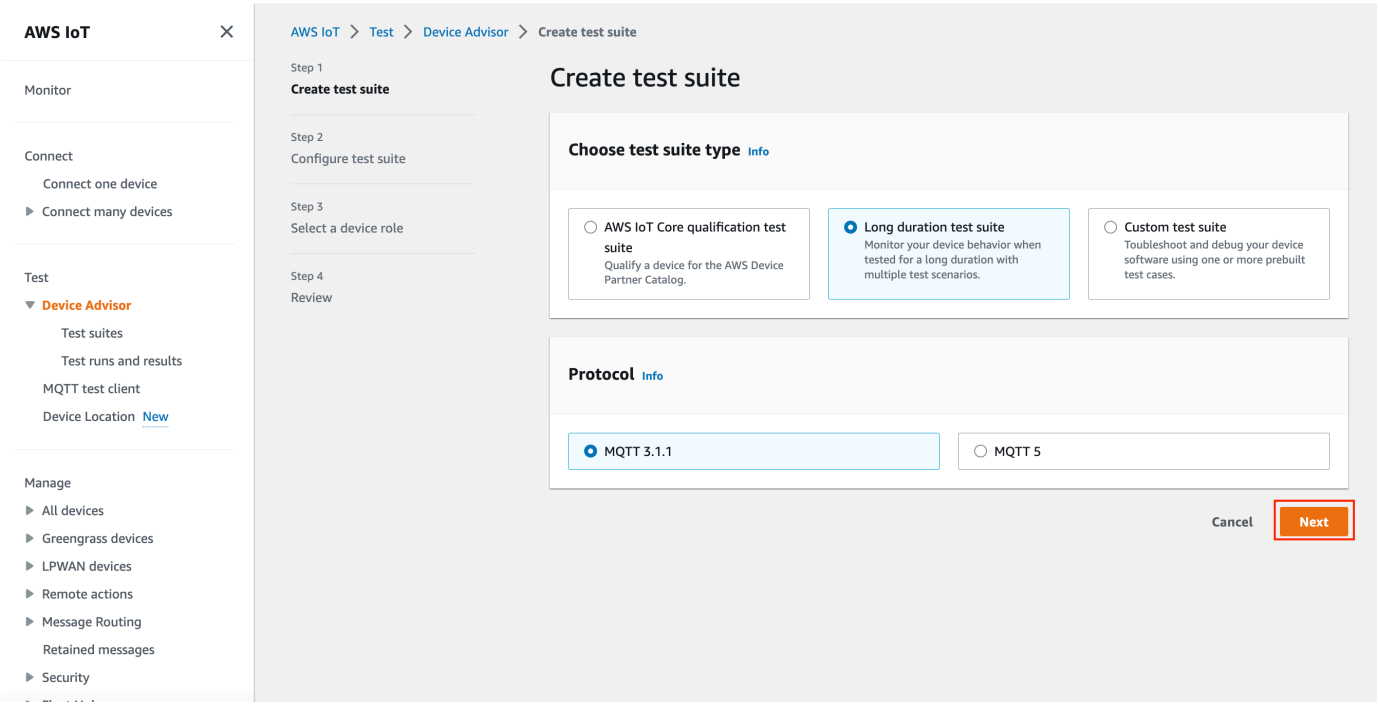
このチュートリアルは、コンソールを使用して Device Advisor で長時間テストを開始するのに役立ちます。このチュートリアルを完了するには、設定 のステップに従ってください。

1. AWS IoT コンソール のナビゲーションペインで、[Test] (テスト)、[Device Advisor] の順に展開し、[Test suites] (テストスイート) を選択します。ページで、[Create long duration test suite] (長期テストスイートの作成) を選択します。



2. [Create test suite] (テストスイートの作成) ページで、[Long duration test suite] (長期テストスイート) を選択し、[Next] (次へ) を選択します。

   プロトコルについては、[MQTT 3.1.1] または [MQTT 5] のいずれかを選択します。



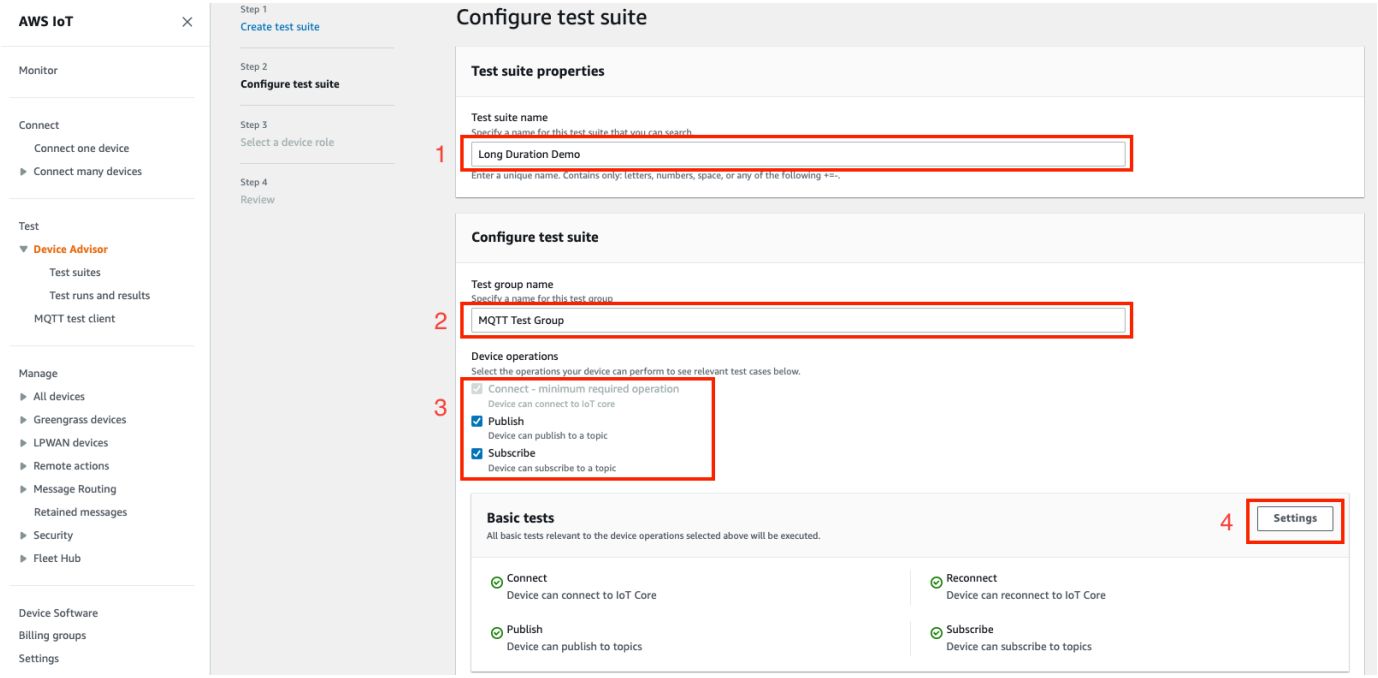
3. [Configure test event] (テストイベントの設定) ページで、以下の操作を行います。

   a. テストスイート名フィールドを更新します。

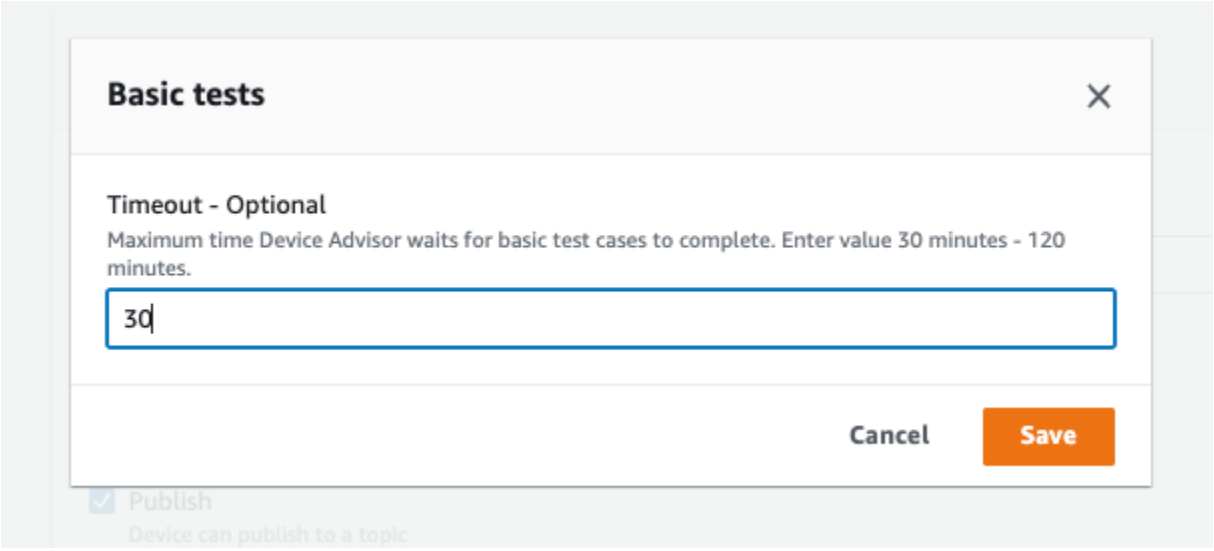
b. テストグループ名フィールドを更新します。

c. デバイスが実行できるデバイスオペレーションを選択します。これにより、実行するテストが選択されます。

d. [Settings] (設定) オプションを選択します。



4. (オプション) Device Advisor が基本テストを完了するまで待機する必要のある最大時間を入力します。[保存] を選択します。



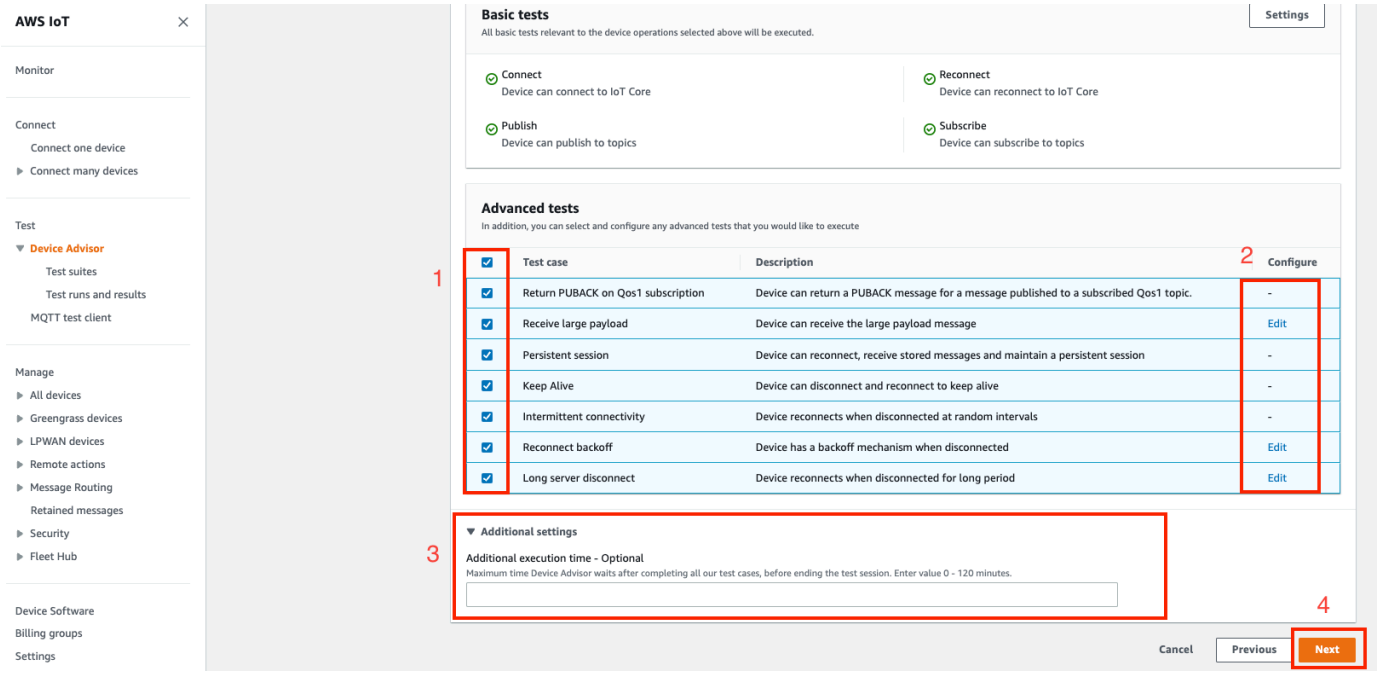
5. [Advanced tests] (詳細テスト) セクションと [Additional settings] (追加設定) セクションで次の操作を行います。

a. このテストの一部として実行する[Advanced tests] (詳細テスト) を選択または選択解除します。

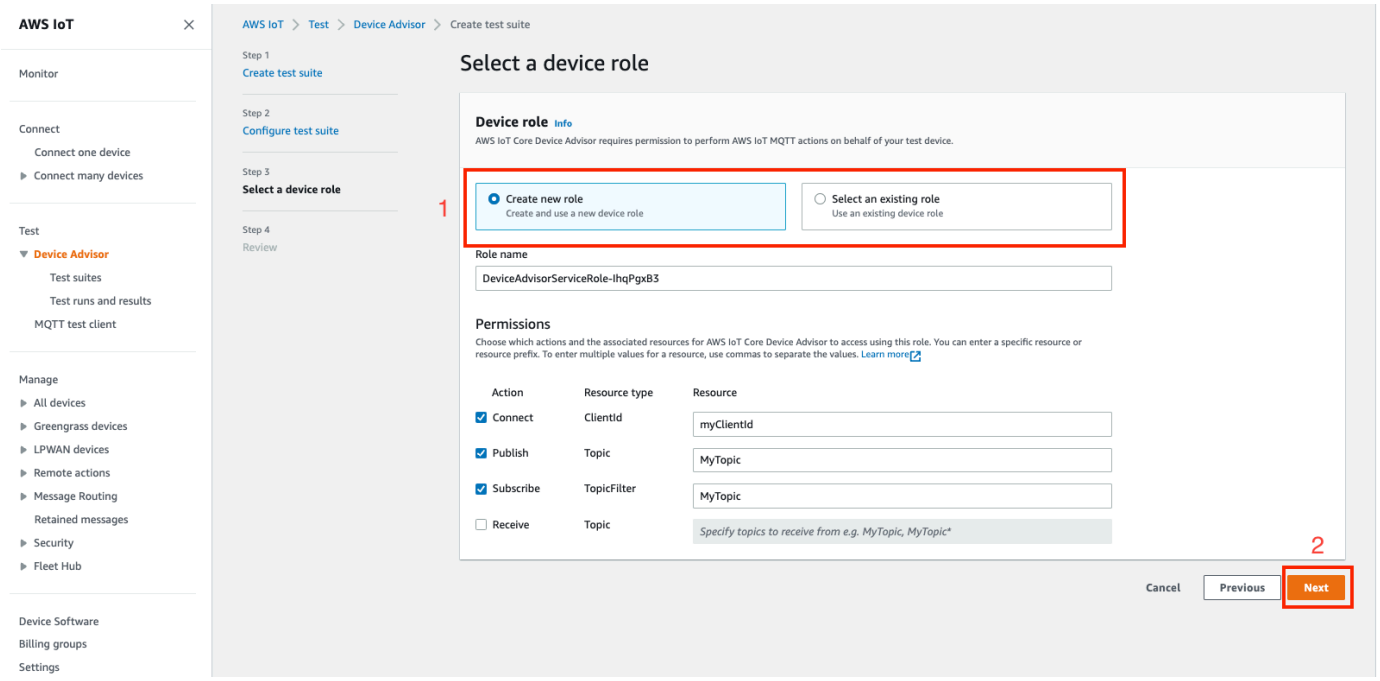
b. 必要に応じて、テストの設定を編集します。

c. [Additional settings] (追加設定) セクションで [Additional execution time] (追加実行時間) を設定します。

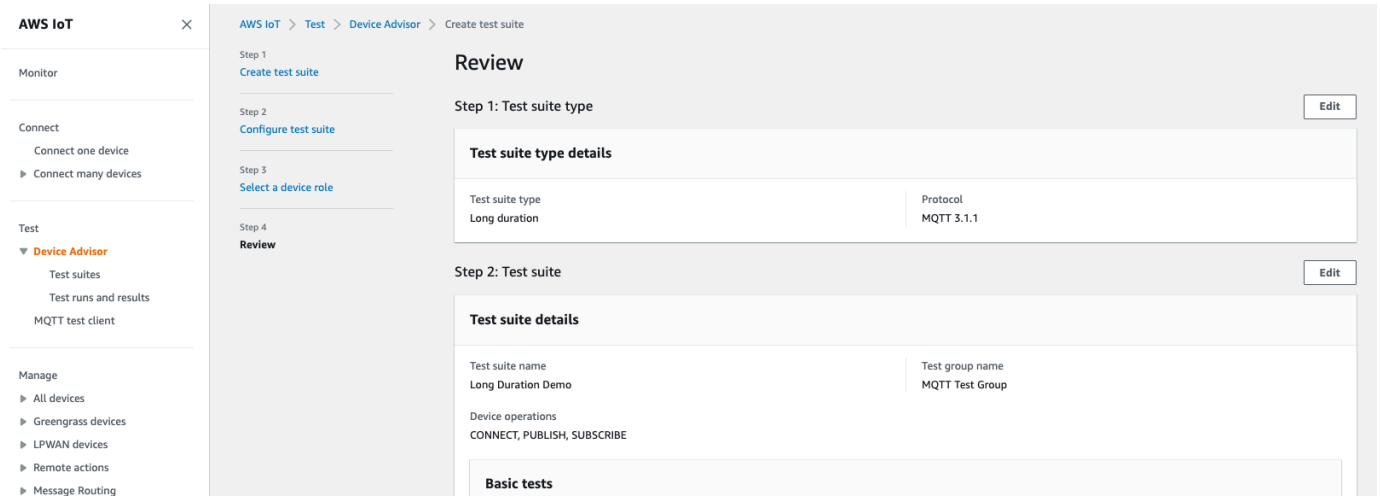
d. [Next] (次へ) を選択して、次のステップに進みます。

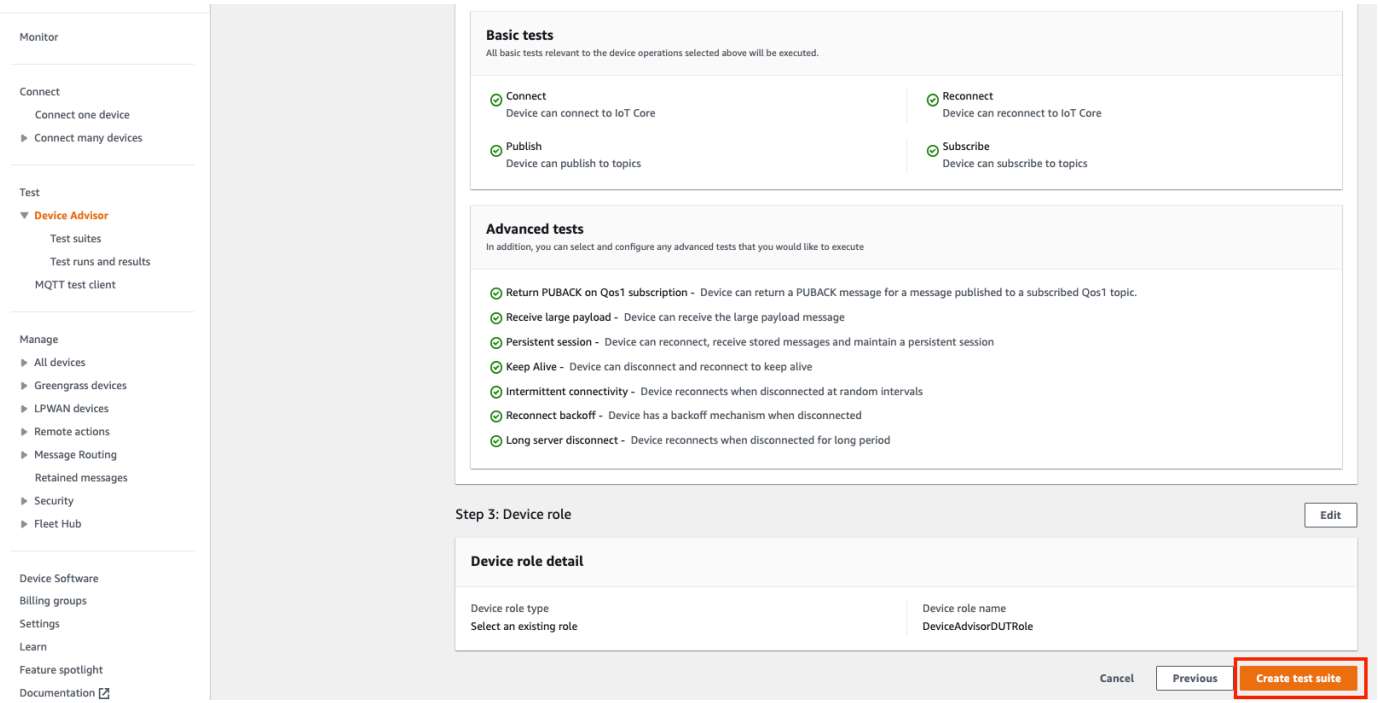


6. このステップでは、新しいロールを作成するか、既存のロールを選択します。詳細については、「デバイスロールとして使用する IAM ロールを作成する」を参照してください。

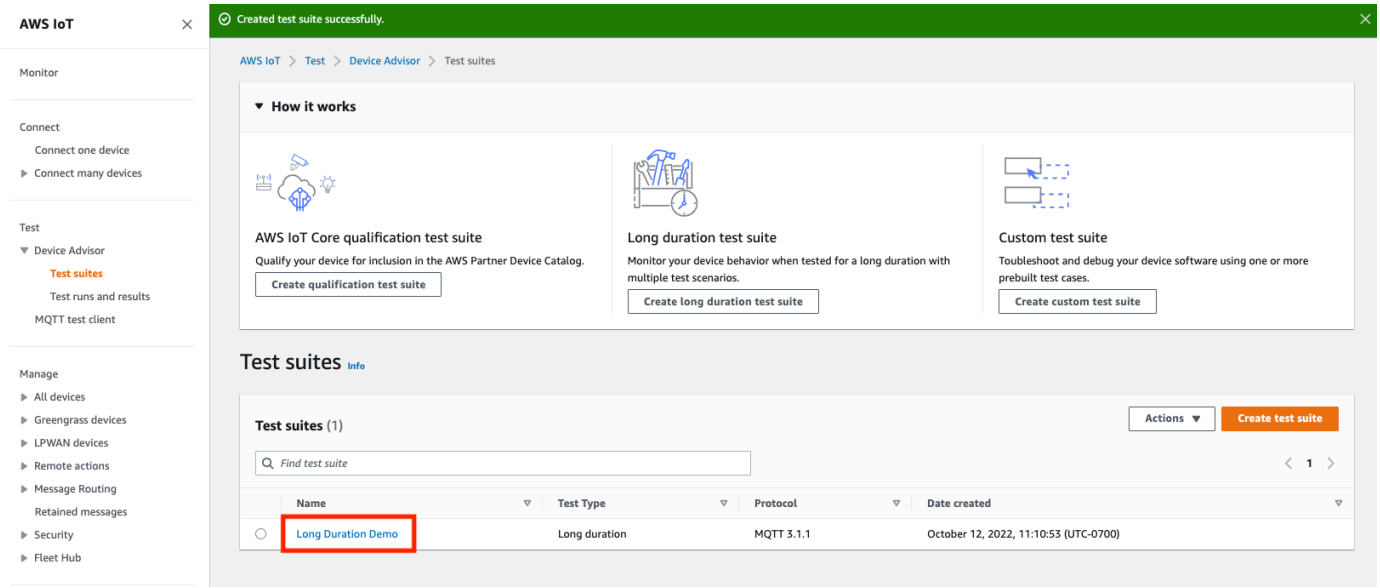


7. このステップまでに作成したすべての設定を確認し、[Create test suite] (テストスイートの作成) を選択します。

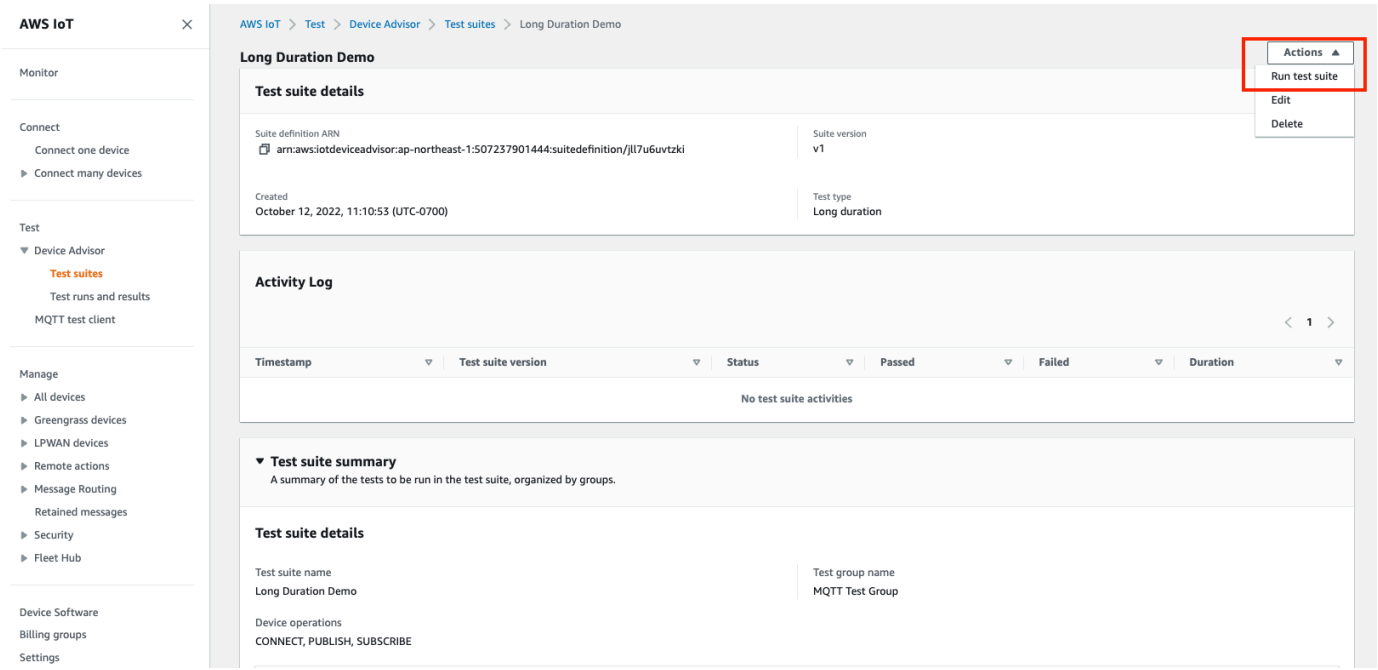




8. 作成されたテストスイートは、[Test suites] (テストスイート) セクションにあります。スイートを選択すると、詳細を表示できます。



9. 作成したテストスイートを実行するには、[Actions] (アクション) を選択し、[Run test suite] (テストスイートを実行) を選択します。

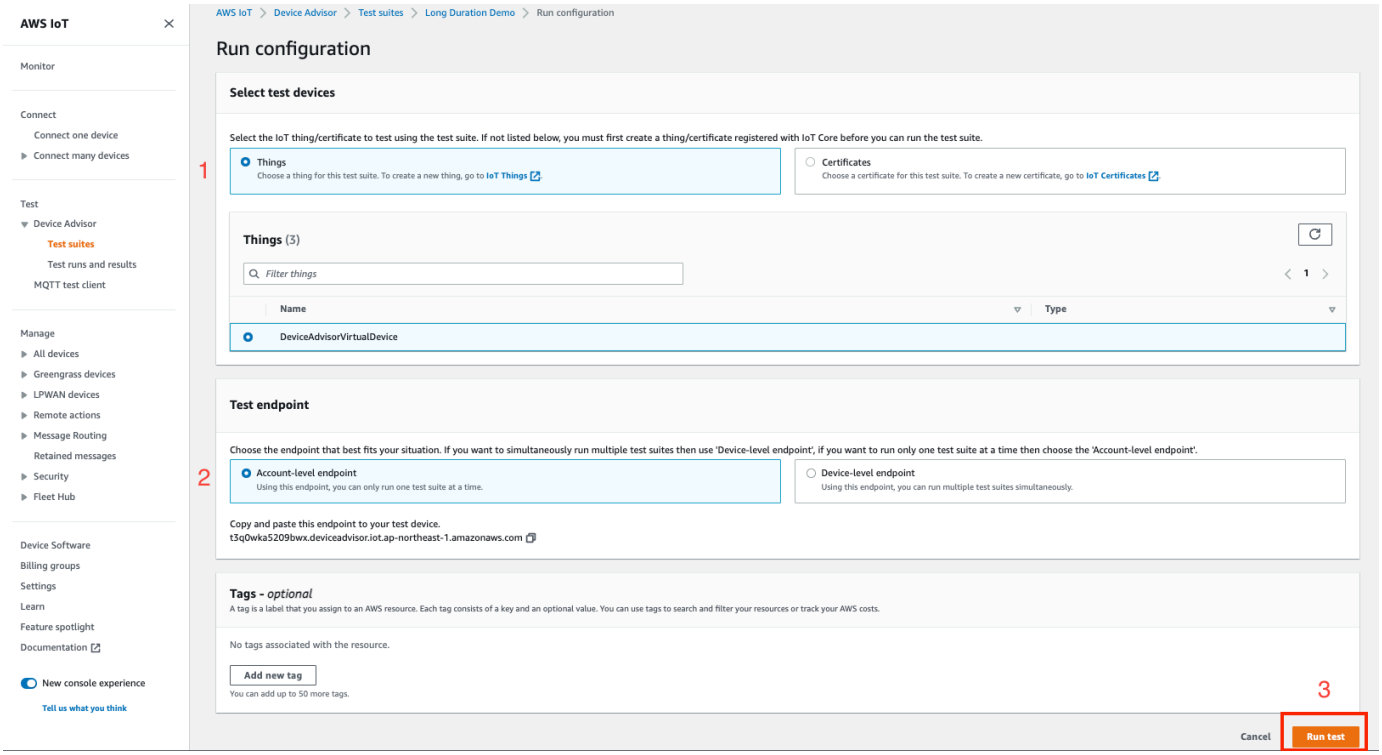


10. [Run configuration] (実行設定) ページで設定オプションを選択します。

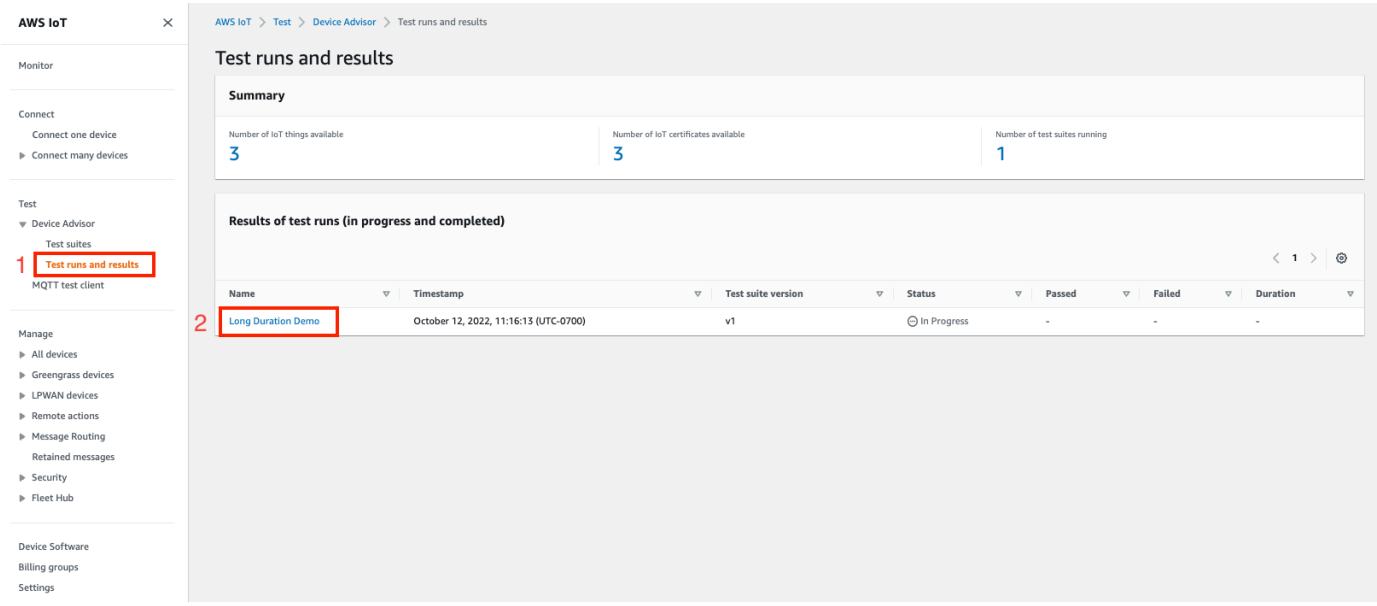
    a. テストを実行する対象の [Things] (モノ) または [Certificate] (証明書) を選択します。

    b. [Account-level endpoint] (アカウントレベルのエンドポイント) または [Device-level endpoint] (デバイスレベルのエンドポイント) を選択します。

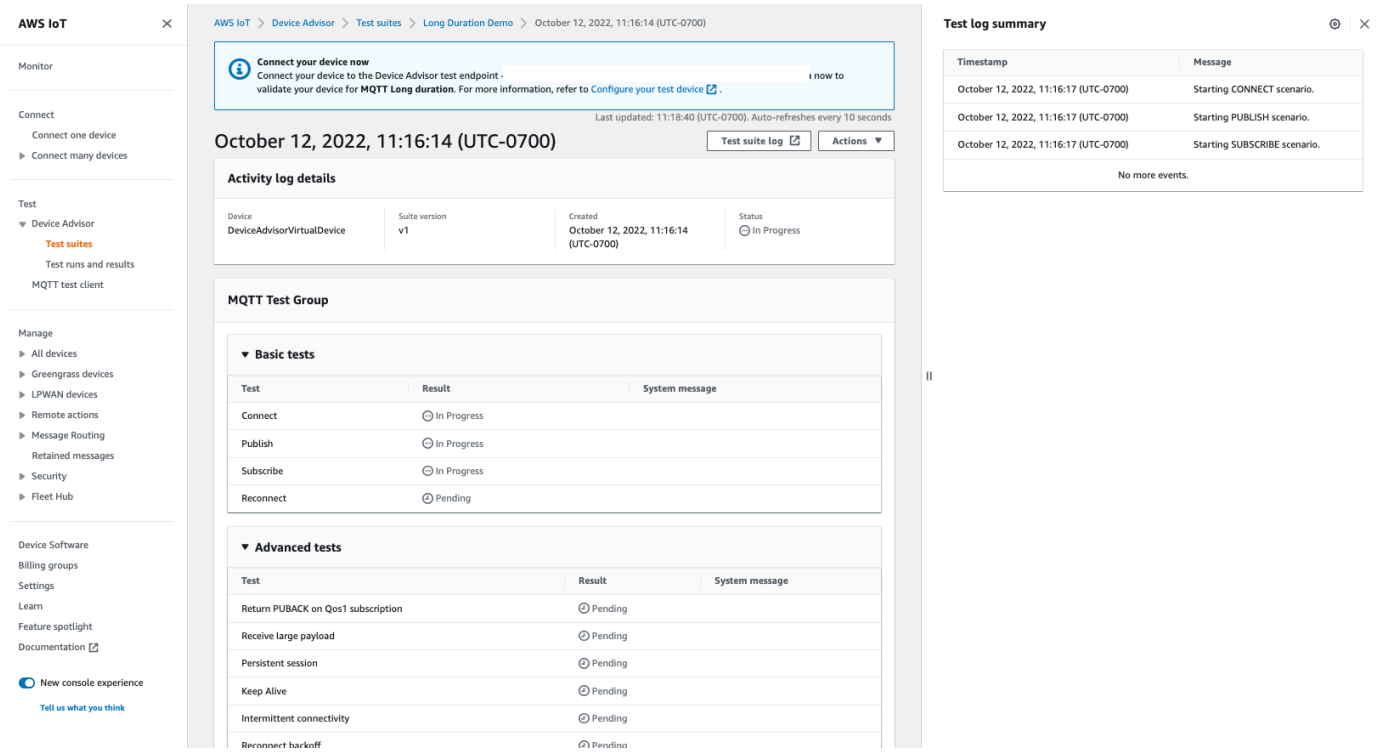
    c. [Run test] (テストの実行) を選択して、テストを実行します。



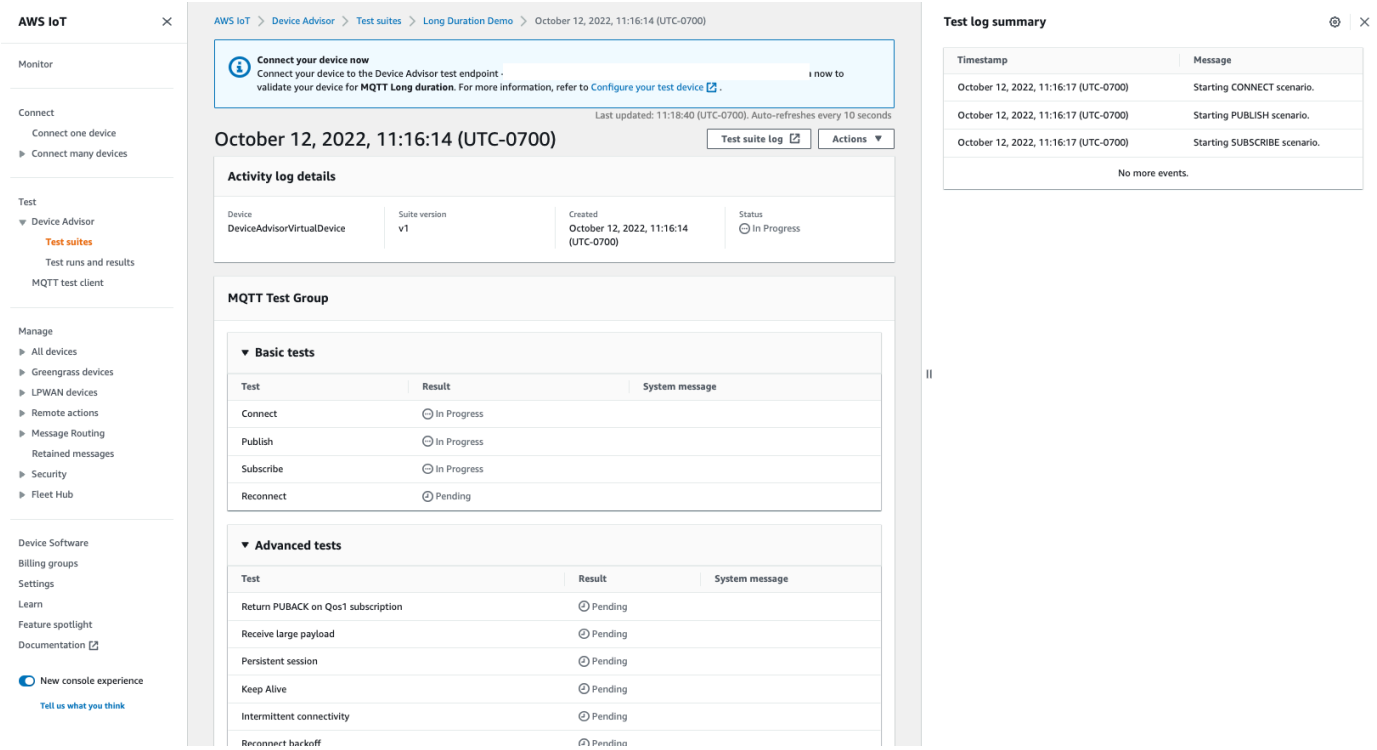
11. テストスイートの実行結果を表示するには、左側のナビゲーションペインで [Test runs and results] (テストの実行と結果) を選択します。実行したテストスイートを選択すると、結果の詳細が表示されます。



12. 前のステップでは、テストの概要ページが表示されます。テスト実行の詳細はすべてこのページに表示されます。コンソールにデバイス接続の開始を求めるメッセージが表示されたら、デバイスを指定されたエンドポイントに接続します。テストの進捗状況はこのページに表示されます。



13. 長時間テストでは、サイドパネルにテストログの概要が追加され、デバイスとブローカーの間で発生するすべての重要なイベントがほぼリアルタイムで表示されます。より詳細なログを表示するには、[Test case log] (テストケースログ) をクリックします。



デバイスアドバイザー VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを作成することで、VPC と AWS IoT Core Device Advisor テストエンドポイント (データプレーン) の間にプライベート接続を確立できます。このエンドポイントを使用して、 AWS IoT デバイスを本番環境にデプロイ AWS IoT Core する前に、 との信頼性の高い安全な接続についてデバイスを検証できます。Device Advisor の事前構築されたテストにより、デバイスソフトウェアを TLS、MQTT、デバイスシャドウ、および AWS IoT Jobs の使用に関するベストプラクティスに照らして検証できます。

AWS PrivateLink は、IoT デバイスで使用されるインターフェイスエンドポイントを強化します。このサービスにより、インターネットゲートウェイ、NAT デバイス、VPN 接続、または Direct Connect 接続なしで、 AWS IoT Core Device Advisor テストエンドポイントにプライベートにアクセスできます。TCP パケットと MQTT パケットを送信する VPC 内のインスタンスは、 AWS IoT Core Device Advisor テストエンドポイントと通信するためにパブリック IP アドレスを必要としません。VPC と AWS IoT Core Device Advisor の間のトラフィックは離れません AWS クラウド。IoT デバイスと Device Advisor テストケース間の TLS および MQTT 通信はすべて、 AWS アカウントのリソース内にとどまります。

各インターフェイスエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。

インターフェイス VPC エンドポイントの使用の詳細については、Amazon VPC ユーザーガイドの「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

AWS IoT Core Device Advisor VPC エンドポイントに関する考慮事項

インターフェイス VPC エンドポイントを設定する前に、Amazon VPC ユーザーガイドの「インターフェイスエンドポイントのプロパティと制限」を確認してください。次に進む前に、以下を検討してください。

- AWS IoT Core Device Advisor は現在、VPC からの Device Advisor テストエンドポイント (データプレーン) への呼び出しをサポートしています。メッセージブローカーは、データプレーン通信を使用してデータを送受信します。これは TLS と MQTT パケットの助けを借りて行われます。AWS IoT デバイスを Device Advisor テストエンドポイント AWS IoT Core Device Advisor に接続するための VPC エンドポイント。コントロールプレーン API アクションは、この VPC エンドポイントでは使用されません。テストスイートやその他のコントロールプレーン API を作成または実行するには、コンソール、 AWS SDK、またはパブリックインターネット上の AWS コマンドラインインターフェイスを使用します。

- 以下では、VPC エンドポイント AWS リージョン がサポートされています AWS IoT Core Device Advisor。
  - 米国東部 (バージニア北部)
  - 米国西部 (オレゴン)
  - アジアパシフィック (東京)
  - 欧州 (アイルランド)
- Device Advisor は、X.509 クライアント証明書と RSA サーバー証明書による MQTT をサポートします。
- VPC エンドポイントポリシーは、今のところサポートされていません。
- VPC エンドポイントを接続するリソースの作成方法については、VPC エンドポイントの前提条件を確認してください。VPC エンドポイントを使用するには、 AWS IoT Core Device Advisor VPC とプライベートサブネットを作成する必要があります。
- AWS PrivateLink リソースにはクォータがあります。詳細については、AWS PrivateLink クォータを参照してください。
- VPC エンドポイントは IPv4 トラフィックのみをサポートします。

のインターフェイス VPC エンドポイントを作成する AWS IoT Core Device Advisor

VPC エンドポイントの使用を開始するには、インターフェイス VPC エンドポイントを作成します。次に、 AWS IoT Core Device Advisor として を選択します AWS のサービス。を使用している場合は AWS CLI、describe-vpc-endpoint-services を呼び出して、 AWS IoT Core Device Advisor が のアベイラビリティーゾーンに存在することを確認します AWS リージョン。エンドポイントに接続されているセキュリティグループが MQTT および TLS トラフィックの TCP プロトコル通信を許可していることを確認します。例えば、米国東部 (バージニア北部) リージョンでは、以下のコマンドを使用します。

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.deviceadvisor.iot
```

次のサービス名 AWS IoT Core を使用して、 の VPC エンドポイントを作成できます。

- com.amazonaws.region.deviceadvisor.iot

デフォルトでは、エンドポイントのプライベート DNS は有効になっています。これにより、デフォルトのテストエンドポイントの使用がプライベートサブネット内にとどまることが保証されます。アカウントまたはデバイスレベルのエンドポイントを取得するには、 コンソール AWS CLI または AWS SDK を使用します。例えば、パブリックサブネット内またはパブリックインターネット上で get-endpoint を実行する場合、エンドポイントを取得し、それを使用して Device Advisor に接続できます。詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

MQTT クライアントを VPC エンドポイントインターフェイスに接続するために、 AWS PrivateLink サービスは VPC にアタッチされたプライベートホストゾーンに DNS レコードを作成します。これらの DNS レコードは、 AWS IoT デバイスのリクエストを VPC エンドポイントに転送します。

VPC エンドポイント AWS IoT Core Device Advisor を介した へのアクセスの制御

VPC 条件コンテキストキーを使用して、VPC エンドポイントを介してのみデバイスアクセスを制限 AWS IoT Core Device Advisor し、アクセスを許可できます。 は、次の VPC 関連のコンテキストキー AWS IoT Core をサポートしています。

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core Device Advisor は現在、VPC エンドポイントポリシーをサポートしていません。

次のポリシーは、モノの名前に一致するクライアント ID AWS IoT Core Device Advisor を使用してに接続するアクセス許可を付与します。また、モノ名のプレフィックスが付いた任意のトピックにも公開されます。このポリシーは、デバイスが特定の VPC エンドポイント ID を持つ VPC エンドポイントに接続することを条件としています。このポリシーでは、パブリック AWS IoT Core Device Advisor テストエンドポイントへの接続試行が拒否されます。

```
{
```

```
"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 テストケース

Device Advisor は、6 つのカテゴリの事前構築済みテストを提供します。

- TLS
- MQTT
- シャドウ
- ジョブの実行

- アクセス許可とポリシー
- 長期テスト

Device Qualification Program の対象となる AWS Device Advisor テストケース。

お使いのデバイスが認定を受けるには、AWS デバイス認定プログラムに従って、次のテストに合格する必要があります。

> **Note**
>
> これは資格試験の改訂されたリストです。

- TLS Connect (「TLSConnect」)
- TLS 不正なサブジェクト名サーバー証明書 (「サブジェクトの共通名 (CN)/サブジェクトの別名 (SAN) が正しくありません」)
- TLS 非セキュアサーバー証明書 (「認識された CA によって署名されていません」)
- 暗号スイートの AWS IoT TLS デバイスサポート (AWS IoT 「推奨暗号スイートの TLS デバイスサポート」)
- TLS が受信する最大サイズのフラグメント (「TLS が受信する最大サイズのフラグメント」)
- TLS 期限切れサーバー証明書 (「期限切れサーバー証明書」)
- TLS 大きなサイズのサーバー証明書 (「TLS 大きなサイズのサーバー証明書」)
- MQTT Connect (「デバイスが CONNECT を AWS IoT Core (ハッピーケース) に送信する」)
- MQTT Subscribe (「Can Subscribe (Happy Case)」)
- MQTT Publish (「QoS0 (Happy Case)」)
- MQTT 接続ジッター再試行 (「ジッターバックオフを使用したデバイス接続再試行 - CONNACK 応答なし」)

TLS

これらのテストを使用して、デバイスと 間のトランスポートレイヤーセキュリティプロトコル (TLS) AWS IoT が安全かどうかを判断します。

> **Note**
>
> Device Advisor が TLS1.3 をサポートするようになりました。

Happy Path

TLS Connect

テスト対象のデバイスが TLS ハンドシェイクを完了できるかどうかを検証します AWS IoT。このテストでは、クライアントデバイスの MQTT 実装は検証されません。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。最良の結果を得るには、タイムアウト値 を 2 分とすることをお勧めします。

```
"tests":[
   {
      "name":"my_tls_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Connect",
         "version":"0.0.0"
      }
   }
]
```

Exampleテストケースの出力:

- 合格 — テスト対象のデバイスが TLS ハンドシェイクを完了しました AWS IoT。
- 警告付きで合格 — テスト対象のデバイスは TLS ハンドシェイクを完了しましたが AWS IoT、デバイスまたは からの TLS 警告メッセージがありました AWS IoT。

- 失敗 — ハンドシェイクエラー AWS IoT により、テスト対象のデバイスが で TLS ハンドシェイクを完了できませんでした。

TLS が受信する最大サイズのフラグメント

このテストケースは、デバイスが TLS 最大サイズのフラグメントを受信して処理できることを検証します。大きなペイロードを受信するには、テストデバイスが QoS 1 で事前設定されたトピックをサブスクライブする必要があります。${payload} 設定を使用して、ペイロードをカスタマイズできます。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。最良の結果を得るには、タイムアウト値 を 2 分とすることをお勧めします。

```
"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"
      }
   }
]
```

暗号スイート

AWS IoT 推奨される暗号スイートの TLS デバイスサポート

テスト対象デバイスからの TLS Client Hello メッセージの暗号スイートに、推奨される AWS IoT 暗号スイートが含まれていることを検証します。デバイスでサポートされている暗号スイートに関するさらなる洞察を提供します。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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"
      }
   }
]
```

Exampleテストケースの出力:

- 合格 — テスト対象のデバイス暗号スイートには、推奨される暗号スイートが少なくとも 1 AWS IoT つ含まれており、サポートされていない暗号スイートは含まれていません。
- 警告付きで合格 – デバイス暗号スイートには少なくとも 1 つの AWS IoT 暗号スイートが含まれていますが、次の点に注意してください。
  1. 推奨される暗号スイートは含まれていません
  2. これには、 でサポートされていない暗号スイートが含まれています AWS IoT。

  サポートされていない暗号スイートが安全であることを確認することをお勧めします。

- 失敗 — テスト対象のデバイス暗号スイートには、 AWS IoT サポートされている暗号スイートは含まれていません。

より大きなサイズのサーバー証明書

TLS 大きなサイズのサーバー証明書

お使いのデバイスが、より大きなサイズのサーバー証明書を受け取って処理する場合に、 AWS IoT との TLS ハンドシェイクを完了できるかどうかを検証します。このテストで使用されるサーバー証明書のサイズ (バイト単位) は、TLS Connect テストケースおよび IoT Core で現在使用されているサイズよりも大きく、20 倍です。このテストケースでは、 は TLS のデバイスのバッファスペースを AWS IoT テストします。バッファスペースが十分大きい場合、TLS ハンドシェイクはエラーなしで枯渇します。このテストでは、デバイスの MQTT 実装は検証されません。テストケースは、TLS ハンドシェイクプロセスが完了した後に終了します。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。最良の結果を得るには、タイムアウト値 を 2 分とすることをお勧めします。このテストケースが失敗し、TLS Connect テストケースが成功した場合、お使いのデバイスの TLS 用バッファ領域の制限を増やすことをお勧めします。バッファ領域の制限を増やすと、サイズが大きくなった場合にデバイスがより大きなサイズのサーバー証明書を処理できるようになります。

```
"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"
      }
   }
]
```

Exampleテストケースの出力:

- 合格 – テスト対象のデバイスが AWS IoTとの TLS ハンドシェイクを完了しました。
- 警告付きで合格 — テスト対象のデバイスは TLS ハンドシェイクを完了しましたが AWS IoT、デバイスまたは からの TLS 警告メッセージがあります AWS IoT。
- 失敗 — ハンドシェイクプロセス中にエラーが発生した AWS IoT ため、テスト対象のデバイスが との TLS ハンドシェイクを完了できませんでした。

TLS 非セキュアサーバー証明書

認識された CA によって署名されていません

ATS CA からの有効な署名がないサーバー証明書が提示された場合にテスト対象のデバイスが接続を閉じることを検証します。デバイスは、有効な証明書を提示するエンドポイントにのみ接続する必要があります。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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"
      }
   }
]
```

Exampleテストケースの出力:

- 合格 – テスト対象のデバイスは接続を閉じました。

- 失敗 — テスト対象のデバイスが TLS ハンドシェイクを完了しました AWS IoT。

TLS 不正なサブジェクト名サーバー証明書/不正なサブジェクトの共通名 (CN)/サブジェクトの別名 (SAN)

リクエストされたものとは異なるドメイン名のサーバー証明書が提示された場合にテスト対象のデバイスが接続を閉じることを検証します。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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"
      }
   }
]
```

Exampleテストケースの出力:

- 合格 – テスト対象のデバイスは接続を閉じました。
- 失敗 — テスト対象のデバイスが TLS ハンドシェイクを完了しました AWS IoT。

TLS 期限切れサーバー証明書

期限切れサーバー証明書

期限切れサーバー証明書が提示された場合にテスト対象のデバイスが接続を閉じることを検証します。

Example API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"tests":[
   {
      "name":"my_tls_expired_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Expired_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Exampleテストケースの出力:

- 合格 — テスト対象のデバイスは TLS ハンドシェイクの完了を拒否します AWS IoT。デバイスは、接続を閉じる前に TLS アラートメッセージを送信します。
- 警告付きで合格 – テスト対象のデバイスが AWS IoTとの TLS ハンドシェイクを完了することを拒否します。ただし、接続を閉じる前に TLS 警告メッセージは送信されません。
- 失敗 — テスト対象のデバイスは TLS ハンドシェイクを完了します AWS IoT。

MQTT

CONNECT、DISCONNECT、および RECONNECT

「デバイスが CONNECT を AWS IoT Core (ハッピーケース) に送信する」

テスト対象のデバイスが CONNECT リクエストを送信することを検証します。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"tests":[
   {
      "name":"my_mqtt_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"MQTT_Connect",
         "version":"0.0.0"
      }
   }
]
```

「デバイスは、QoS1 のための任意のトピックに PUBACK を返すことができます」

このテストケースでは、デバイス (クライアント) が QoS1 でトピックにサブスクライブした後にブローカーから発行メッセージを受信した場合、PUBACK メッセージを返すことができるかどうかをチェックします。

このテストケースでは、ペイロードコンテンツとペイロードサイズを設定できます。ペイロードサイズが設定されている場合、Device Advisor はペイロードコンテンツの値を上書きし、事前定義済みのペイロードを目的のサイズでデバイスに送信します。ペイロードサイズは 0 から 128 までの値で、128 KB を超えることはできません。AWS IoT Core メッセージブローカーとプロトコルの制限とクォータのページで示されているように、 AWS IoT Core では、128 KB を超えるリクエストの発行および接続は拒否されます。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分を推奨します。PAYLOAD_SIZE は 0 ~ 128 KB の値に設定できます。ペイロードサイズを定義する

と、Device Advisor が指定されたサイズの事前定義されたペイロードをデバイスに送り返すため、ペイロードコンテンツが上書きされます。

```
"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"
        }
    }
]
```

「ジッターバックオフを使用したデバイス接続再試行 - CONNACK 応答なし」

ブローカーに少なくとも 5 回再接続するときに、テスト対象のデバイスが適切なジッターバックオフを使用することを検証します。ブローカーは、テスト対象のデバイスの CONNECT リクエストのタイムスタンプを記録し、パケット検証を実行し、テスト対象デバイスに CONNACK を送信せずに一時停止し、テスト対象デバイスがリクエストを再送信するのを待機します。6 回目の接続試行はパススルーでき、CONNACK はテスト対象のデバイスにフローバックできます。

上記のプロセスが再び実行されます。合計で、このテストケースでは、デバイスが合計で少なくとも 12 回接続する必要があります。収集されたタイムスタンプは、ジッターバックオフがテスト対象デバイスによって使用されていることを検証するために使用されます。テスト対象のデバイスにエクスポネンシャルバックオフ遅延のみがある場合、警告付きでこのテストケースに合格します。

このテストケースに合格するには、テスト対象のデバイスにエクスポネンシャルバックオフとジッターメカニズムを実装することをお勧めします。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUT のデフォルト値は 5 分です。タイムアウト値は 4 分であることを推奨します。

```
"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"
      }
   }
]
```

「エクスポネンシャルバックオフを使用したデバイス接続再試行 - CONNACK 応答なし」

ブローカーに少なくとも 5 回再接続するときに、テスト対象のデバイスが適切なエクスポネンシャルバックオフを使用することを検証します。ブローカーは、テスト対象のデバイスの CONNECT リクエストのタイムスタンプを記録し、パケット検証を実行し、クライアントデバイスに CONNACK を送信せずに一時停止し、テスト対象デバイスがリクエストを再送信するのを待機します。収集されたタイムスタンプは、エクスポネンシャルバックオフがテスト対象のデバイスによって使用されていることを検証するために使用されます。

このテストケースに合格するには、テスト対象のデバイスにエクスポネンシャルバックオフとジッターメカニズムを実装することをお勧めします。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUT のデフォルト値は 5 分です。タイムアウト値は 4 分であることを推奨します。

```
"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"
      }
   }
]
```

「デバイスはジッターバックオフを使用して再接続します-サーバーの切断後」

テスト対象のデバイスが、サーバーから切断された後の再接続時に必要なジッターとバックオフを使用しているかどうかを検証します。Device Advisor が、デバイスをサーバーから少なくとも 5 回切断し、MQTT 再接続のためにデバイスの動作を監視します。Device Advisor は、テスト対象デバイスの CONNECT リクエストのタイムスタンプを口グに記録し、パケット検証を実行し、クライアントデバイスに CONNACK を送信せずに一時停止して、テスト対象デバイスがリクエストを再送信するのを待ちます。収集されたタイムスタンプは、テスト対象デバイスが再接続中にジッターとバックオフを使用することを検証するために使用されます。テスト対象のデバイスがエクスポネンシャルバックオフのみを使用している、または適切なジッターバックオフメカニズムを実装していない場合は、警告付きでこのテストケースに合格します。テスト対象のデバイスが、線形バックオフまたは固定バックオフメカニズムを実装している場合、テストは失敗します。

このテストケースに合格するには、テスト対象のデバイスに、エクスポネンシャルバックオフとジッターメカニズムを実装することをお勧めします。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUT のデフォルト値は 5 分です。タイムアウト値は 4 分であることを推奨します。

バックオフを検証する再接続試行回数は、RECONNECTION_ATTEMPTS を指定する事によって変更できます。その数は 5 ~ 10 の間である必要があります。デフォルト値は 5 です。

```
"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"
      }
   }
]
```

「ジッターバックオフを使用したデバイスの再接続 - 不安定な接続時」

不安定な接続での再接続中に、テスト対象のデバイスが必要なジッターとバックオフを使用するかどうかを検証します。Device Advisor は、5 回接続に成功するとサーバーからデバイスを切断し、MQTT 再接続のためにデバイスの動作を監視します。Device Advisor は、テスト対象デバイスの CONNECT リクエストのタイムスタンプをログに記録し、パケット検証を実行し、CONNACK を返し、切断し、切断のタイムスタンプをログに記録し、テスト対象デバイスがリクエストを再送信するのを待ちます。収集されたタイムスタンプは、テスト対象デバイスが、接続は成功したが不安定になった後の再接続中にジッターとバックオフを使用していることを検証するために使用されます。テスト対象のデバイスがエクスポネンシャルバックオフのみを使用している、または適切なジッターバックオフメカニズムを実装していない場合は、警告付きでこのテストケースに合格します。テスト対象のデバイスが、線形バックオフまたは固定バックオフメカニズムを実装している場合、テストは失敗します。

このテストケースに合格するには、テスト対象のデバイスに、エクスポネンシャルバックオフとジッターメカニズムを実装することをお勧めします。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUT のデフォルト値は 5 分です。タイムアウト値は 4 分であることを推奨します。
> バックオフを検証する再接続試行回数は、RECONNECTION_ATTEMPTS を指定する事によって変更できます。その数は 5 ~ 10 の間である必要があります。デフォルト値は 5 です。

```
"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"
      }
   }
]
```

発行

「QoS0 (Happy Case)」

テスト対象デバイスが QoS0 または QoS1 でメッセージを発行することを検証します。テスト設定でトピック値とペイロードを指定することで、メッセージとペイロードのトピックを検証することもできます。

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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 発行の再試行 - PUBACK なし」

ブローカーが PUBACK を送信しない場合、テスト対象のデバイスが QoS1 で送信されたメッセージを再発行することを検証します。また、テスト設定でこのトピックを指定することで、メッセージのトピックを検証することもできます。メッセージを再発行する前に、クライアントデバイスを切断しないでください。このテストでは、再発行されたメッセージが、元のメッセージと同じパケット ID を持つことも検証されます。テスト実行中にデバイスが接続を失って再接続した場合、テストケースは失敗することなくリセットされます。そのため、デバイスはテストケースのステップを再実行する必要があります。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。少なくとも 4 分間お勧めします。

```
"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"
      }
   }
]
```

「保持されたメッセージの発行」

テスト対象のデバイスが、retainFlag が true に設定されたメッセージを発行することを検証します。テスト設定でトピック値とペイロードを設定することで、メッセージのトピックとペイロードを検証できます。PUBLISH パケット内で送信された retainFlag が true に設定されていない場合、テストケースは失敗します。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。このテストケースを実行するには、デバイスロールに iot:RetainPublish アクションを追加します。

```
"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"
      }
   }
]
```

「ユーザープロパティを使用して発行」

テスト対象デバイスが正しいユーザープロパティでメッセージを発行することを検証します。テスト設定で名前と値のペアを設定することで、ユーザープロパティを検証できます。ユーザープロパティが指定されていないか、一致しない場合、テストケースは失敗します。

API テストケースの定義:

> **Note**
>
> これは MQTT5 のみのテストケースです。
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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

「Can Subscribe (Happy Case)」

テスト対象のデバイスが MQTT トピックにサブスクライブしていることを検証します。テスト設定でこのトピックを指定することで、テスト対象のデバイスがサブスクライブするトピックを検証することもできます。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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"
      }
   }
]
```

「サブスクライブ再試行 - SUBACK なし」

テスト対象のデバイスが MQTT トピックへの失敗したサブスクリプションを再試行することを検証します。サーバーは待機し、SUBACK を送信しません。クライアントデバイスがサブスクリプションを再試行しない場合、テストは失敗します。クライアントデバイスは、失敗したサブスクリプションを同じパケット ID で再試行する必要があります。テスト設定でこのトピックを指定することで、テスト対象のデバイスがサブスクライブするトピックを検証することもできます。テスト実行中にデバイスが接続を失って再接続した場合、テストケースは失敗することなくリセットされます。そのため、デバイスはテストケースのステップを再実行する必要があります。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUT のデフォルト値は 5 分です。タイムアウト値は 4 分であることを推奨します。

```
"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」

このテストケースでは、テスト対象のデバイスが ping 応答を受信しないときに切断されるかどうかを検証します。このテストケースの一環として、Device Advisor は発行、サブスクライブ、および ping リクエスト AWS IoT Core のために から送信されたレスポンスをブロックします。テスト対象のデバイスが、MQTT 接続を切断しているかどうかも検証します。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウトは keepAliveTime 値の 1.5 倍超とすることをお勧めします。
> このテストでは、最大値 keepAliveTime は 230 秒以下にする必要があります。

```
"tests":[
    {
       "name":"Mqtt No Ack PingResp",
       "configuration":
          //optional:
          "EXECUTION_TIMEOUT":"306",   // in seconds
```

```
        },
        "test":{
           "id":"MQTT_No_Ack_PingResp",
           "version":"0.0.0"
        }
    }
]
```

永続セッション

「永続セッション (Happy Case)」

このテストケースは、永続セッションから切断されたときのデバイスの動作を検証します。テストケースは、デバイスの再接続、明示的な再サブスクライブなしでのトリガートピックへのサブスクライブの再開、トピックに保存済みのメッセージの受信、および永続セッション中に期待どおりの動作が可能かどうかをチェックします。このテストケースに合格すると、クライアントデバイスが予想される方法で AWS IoT Core ブローカーとの永続的セッションを維持できることを示します。永 AWS IoT 続セッションの詳細については、「MQTT 永続セッションの使用」を参照してください。

このテストケースでは、クライアントデバイスは、クリーンセッションのフラグを false に設定して AWS IoT Core で接続し、トリガートピックにサブスクライブすることが予期されます。サブスクリプションが成功すると、デバイスは AWS IoT Core Device Advisor によって切断されます。デバイスが切断状態の間、そのトピックに QoS 1 メッセージペイロードが保存されます。その後、Device Advisor は、クライアントデバイスがテストエンドポイントとの再接続を許可します。この時点で、クライアントデバイスは、永続セッションが存在するため、追加の SUBSCRIBE パケットを送信せずに、トピックサブスクリプションを再開し、ブローカーから QoS 1 メッセージを受信することが予期されます。再接続後、クライアントデバイスが追加の SUBSCRIBE パケットを送信することでトリガートピックに再サブスクライブした場合、および/またはクライアントがトリガートピックから保存済みメッセージを受信できなかった場合、テストケースは失敗します。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 4 分以上にすることを推奨します。最初の接続で、クライアントデバイスは、以前サブスクライブされていなかった TRIGGER_TOPIC に明示的にサブスクライブする必要があります。テストケー

スに合格するには、クライアントデバイスが QoS 1 で TRIGGER_TOPIC に正常にサブスクライブする必要があります。再接続後、クライアントデバイスは、アクティブな永続セッションがあると認識することが予期されます。そのため、トリガートピックによって送信された保存済みメッセージを受け入れ、その特定のメッセージに対して PUBACK を返します。

```
"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"
      }
   }
]
```

「永続セッション - セッションの有効期限」

このテストケースは、切断されたデバイスが期限切れの永続セッションに再接続するときのデバイスの動作を検証するのに役立ちます。セッションが期限切れになると、デバイスは新しい SUBSCRIBE パケットを明示的に送信して、以前にサブスクライブしたトピックに再サブスクライブすることが予想されます。

最初の接続中、テストデバイスは AWS IoT ブローカーと接続することが想定されます。これは、永続セッションを開始するフラグCleanSessionが false に設定されているためです。その後、デバイスはトリガートピックをサブスクライブする必要があります。その後、サブスクリプションが成功し、永続セッションが開始されると、デバイスは AWS IoT Core Device Advisor によって切断されます。切断後、 AWS IoT Core Device Advisor はテストデバイスがテストエンド

ポイントに再接続できるようにします。この時点で、テストデバイスが別の CONNECT パケットを送信すると、 AWS IoT Core Device Advisor は永続セッションの有効期限が切れていることを示す CONNACK パケットを返します。テストデバイスはこのパケットを適切に解釈する必要があり、永続セッションが終了すると、同じトリガートピックに再サブスクライブすることが予期されます。テストデバイスがトピックトリガーに再サブスクライブしない場合、テストケースは失敗します。テストに合格するには、デバイスは永続セッションが終了したことを認識し、2 番目の接続で同じトリガートピックに対して新しい SUBSCRIBE パケットを送り返す必要があります。

テストデバイスでこのテストケースに合格した場合、永続セッションの期限が切れる際に、デバイスが予期された方法で再接続を処理できることを示しています。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 4 分以上にすることを推奨します。クライアントデバイスは、以前サブスクライブされていなかった TRIGGER_TOPIC に明示的にサブスクライブする必要があります。テストケースに合格するには、テストデバイスで CleanSession フラグを false に設定して CONNECT パケットを送信し、QoS 1 でトリガートピックに正常にサブスクライブする必要があります。接続が成功すると、 AWS IoT Core Device Advisor はデバイスを切断します。切断後、 AWS IoT Core Device Advisor はデバイスの再接続を許可し、 AWS IoT Core Device Advisor が永続セッションを終了するTRIGGER_TOPICため、デバイスは同じ に再サブスクライブすることが期待されます。

```
"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"
      }
```

```
    }
]
```

シャドウ

これらのテストを使用して、テスト対象のデバイスが AWS IoT Device Shadow サービスを正しく使用していることを確認します。詳細については、「AWS IoT Device Shadow サービス」を参照してください。これらのテストケースがテストスイートで設定されている場合は、スイートの実行を開始するときにモノを指定する必要があります。

現時点では、MQTT over WebSocket はサポートされていません。

発行

「デバイスは接続後に状態を発行します (Happy case)」

デバイスが接続後にその状態を発行できるかどうかを検証します AWS IoT Core

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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_STATE は、接続後に、デバイスの正確なシャドウ状態をさらに検証するために提供できます。デフォルトでは、このテストケースはデバイスの発行状態を検証します。

SHADOW_NAME が指定されていない場合、テストケースはデフォルトで名前なし (クラシック) シャドウタイプのトピックプレフィックスに発行されたメッセージを検索します。デバイスで名前の付いたシャドウタイプを使用する場合は、シャドウの名前を指定します。詳細については、デバイスでのシャドウの使用を参照してください。

更新

「デバイスは報告された状態を望ましい状態に更新します (Happy case)」

デバイスが受信したすべての更新メッセージを読み取ったかどうかを検証し、デバイスの状態を同期して、必要な状態のプロパティと一致させます。デバイスは、同期後に最新の報告された状態を発行します。テストを実行する前に、デバイスに既にシャドウが存在する場合は、テストケースに設定された目的の状態と、報告された既存の状態がまだ一致していないことを確認します。Device Advisor から送信されたシャドウ更新メッセージは、シャドウドキュメントの ClientToken フィールドが DeviceAdvisorShadowTestCaseSetup になるのを確認することで識別できます。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 2 分であることを推奨します。

```
"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_STATE には、少なくとも 1 つの属性と関連付けられた値が必要です。

SHADOW_NAME が指定されていない場合、テストケースはデフォルトで [Unnamed] (無名) (クラシック) シャドウタイプのトピックプレフィックスに発行されたメッセージを検索します。デバイスで名前の付いたシャドウタイプを使用する場合は、シャドウの名前を指定します。詳細については、デバイスでのシャドウの使用を参照してください。

ジョブの実行

「デバイスはジョブの実行を完了できます」

このテストケースは、デバイスが AWS IoT Jobs を使用して更新を受信できるかどうかを検証し、成功した更新のステータスを発行するのに役立ちます。 AWS IoT ジョブの詳細については、「 ジョブ」を参照してください。

このテストケースを正常に実行するには、[デバイスロール] を付与する必要がある予約済みの AWS トピックが 2 つあります。ジョブアクティビティ関連のメッセージをサブスクライブするには、notify および notify-next トピックを使用します。デバイスロールは、次のトピックで PUBLISH アクションを付与する必要があります。

- $aws/things/thingName/jobs/jobId/get
- $aws/things/thingName/jobs/jobId/update

次のトピックで SUBSCRIBE アクションと RECEIVE アクションを付与することをお勧めします。

- $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

次のトピックについては、SUBSCRIBE アクションを許可することをお勧めします。

- $aws/things/thingName/jobs/notify-next

これらの予約トピックの詳細については、「AWS IoT ジョブ」で予約トピックについて参照してください。

現時点では、MQTT over WebSocket はサポートされていません。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUTのデフォルト値は 5 分です。タイムアウト値は 3 分とすることを推奨します。提供された AWS IoT ジョブドキュメントまたはソースに応じて、タイムアウト値を調整します (たとえば、ジョブの実行に時間がかかる場合は、テストケースのより長いタイムアウト値を定義します) 。テストを実行するには、有効な AWS IoT ジョブドキュメントまたは既存のジョブ ID が必要です。 AWS IoT ジョブドキュメントは、JSON ドキュメントまたは S3 リンクとして提供できます。ジョブドキュメントが提供されている場合、ジョブ ID の提供は任意です。ジョブ ID が指定されている場合、Device Advisor はユーザーに代わって AWS IoT ジョブを作成するときにその ID を使用します。ジョブドキュメントが提供されていない場合は、テストケースを実行しているのと同じリージョンにある既存の ID を提供できます。この場合、Device Advisor はテストケースの実行中にその AWS IoT ジョブを使用します。

```
"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"
      }
   }
]
```

ジョブドキュメントの作成および使用の詳細については、「ジョブドキュメント」を参照してください。

アクセス許可とポリシー

次のテストを使用して、デバイスの証明書にアタッチされたポリシーがスタンダードベストプラクティスに従っているかどうかを判断します。

現時点では、MQTT over WebSocket はサポートされていません。

「デバイス証明書にアタッチされたポリシーにはワイルドカードが含まれていません」

デバイスに関連付けられたアクセス許可ポリシーがベストプラクティスに従っており、必要以上のアクセス許可をデバイスに付与していないかどうかを検証します。

API テストケースの定義:

> **Note**
>
> EXECUTION_TIMEOUT のデフォルト値は 1 分です。タイムアウトは 30 秒以上で設定することをお勧めします。

```
"tests":[
   {
        "name":"my_security_device_policies",
        "configuration": {
            // optional:
            "EXECUTION_TIMEOUT":"60"    // in seconds
        },
        "test": {
            "id": "Security_Device_Policies",
            "version": "0.0.0"
        }
    }
]
```

長期テスト

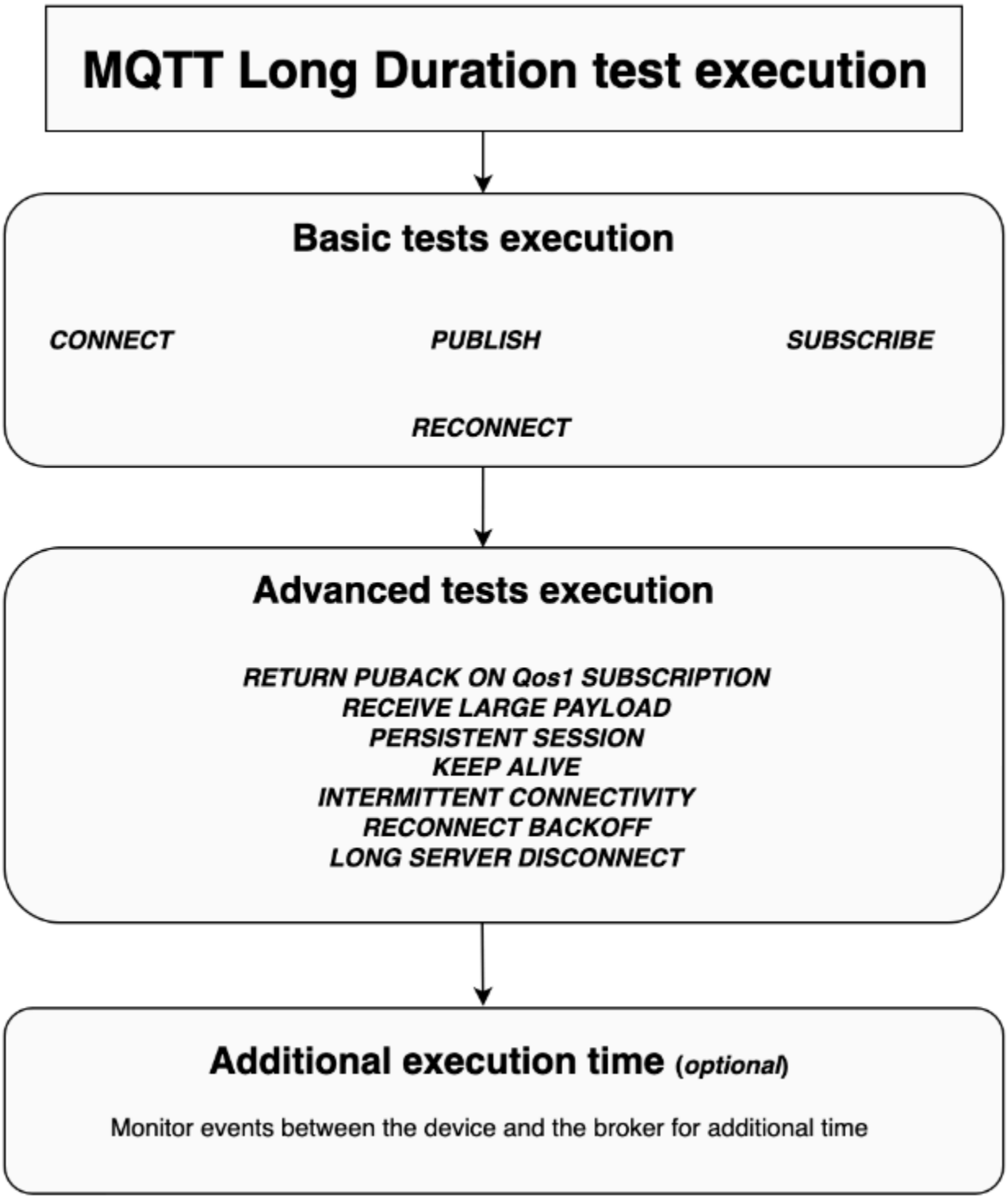
長期テストは、デバイスが長期間動作しているときの動作を監視する新しいテストスイートです。デバイスの特定の動作に焦点を当てた個別のテストを実行する場合と比較して、長期テストでは、デバイスの寿命全体にわたるさまざまな現実世界のシナリオにおけるデバイスの動作を調べます。Device Advisor は、可能な限り効率的な順序でテストを調整します。テストでは、テスト中のデバイスのパフォーマンスに関する有用なメトリクスを含む概要ログを含む結果とログが生成されます。

MQTT 長期テストケース

MQTT の長期テストケースでは、MQTT Connect、サブスクライブ、発行、再接続などのハッピーケースシナリオでデバイスの動作が最初に確認されます。次に、デバイスが MQTT 再接続バックオフ、長期のサーバー切断、断続的な接続など、複数の複雑な障害シナリオに見舞われます。

MQTT 長期テストケース実行フロー

MQTT 長期テストケースの実行には、次の 3 つのフェーズがあります。


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

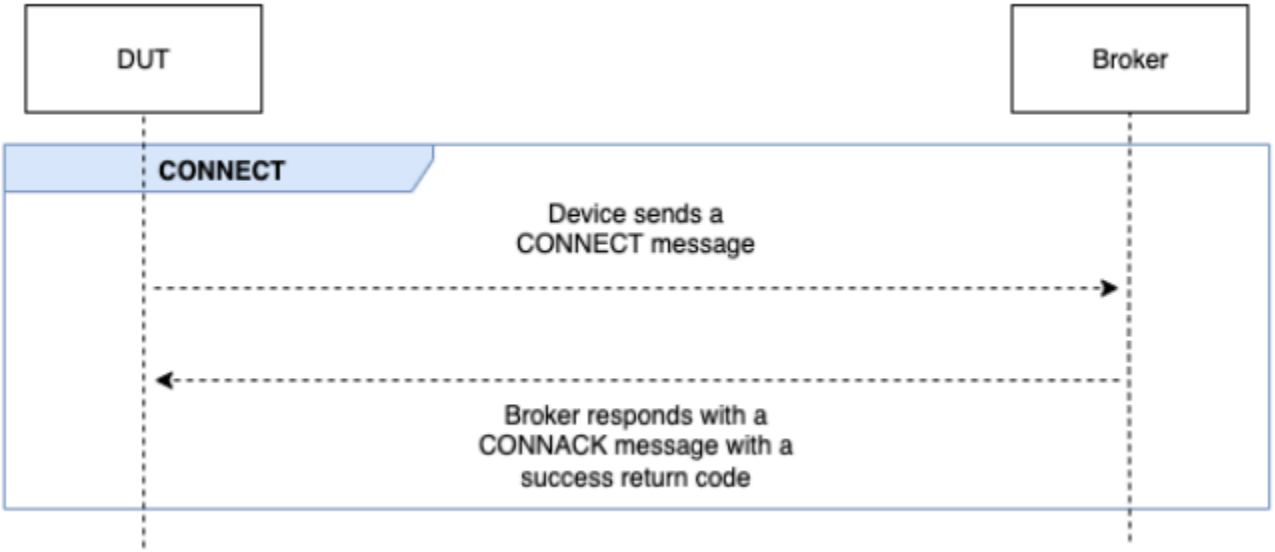
基本テストの実行

このフェーズでは、テストケースは簡単なテストを並行して実行します。このテストでは、設定で選択したオペレーションがデバイスにあるかどうかを検証します。

基本テストのセットには、選択したオペレーションに基づいて次の内容が含まれる場合があります。

CONNECT

このシナリオでは、デバイスがブローカーと正常に接続できるかどうかを検証します。

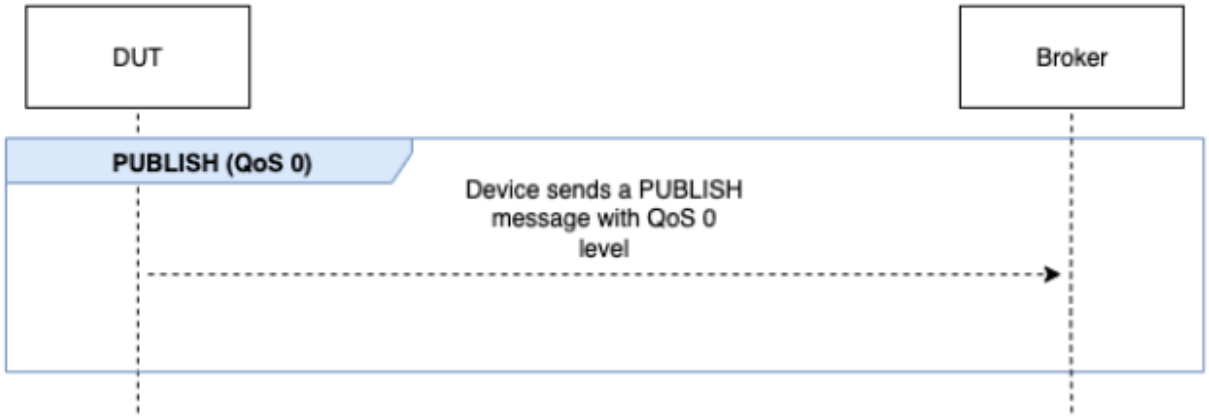


発行

このシナリオでは、デバイスがブローカーに対して正常に発行されているかどうかを検証します。

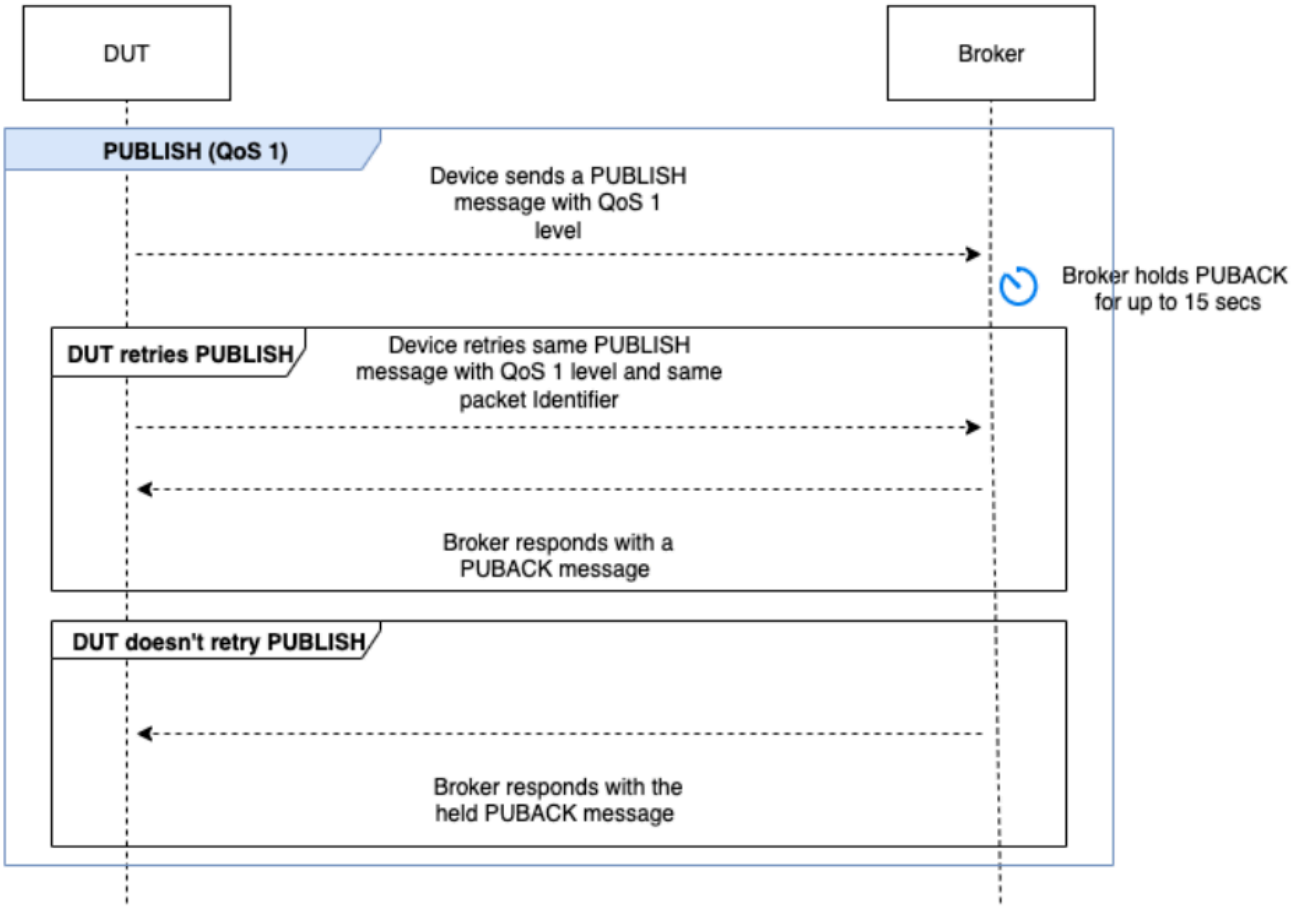
QoS 0

このテストケースは、QoS 0 での発行中に、デバイスがブローカーに PUBLISH メッセージを正常に送信するかどうかを検証します。このテストでは、デバイスが PUBACK メッセージを受信するまで待ちません。



QoS 1

このテストケースでは、デバイスは QoS 1 で 2 つの PUBLISH メッセージをブローカーに送信することが想定されます。最初の PUBLISH メッセージの後、ブローカーは最大 15 秒待ってから、応答します。デバイスは、15 秒以内に同じパケット ID を使用して元の PUBLISH メッセージを再試行する必要があります。その場合、ブローカーは PUBACK メッセージを返し、テストが検証します。デバイスが PUBLISH を再試行しない場合、元の PUBACK がデバイスに送信され、テストはシステムメッセージとともに警告付きで合格とマークされます。テスト実行中にデバイスが接続を失って再接続した場合、テストシナリオは失敗することなくリセットされます。そのため、デバイスはテストシナリオのステップを再実行する必要があります。

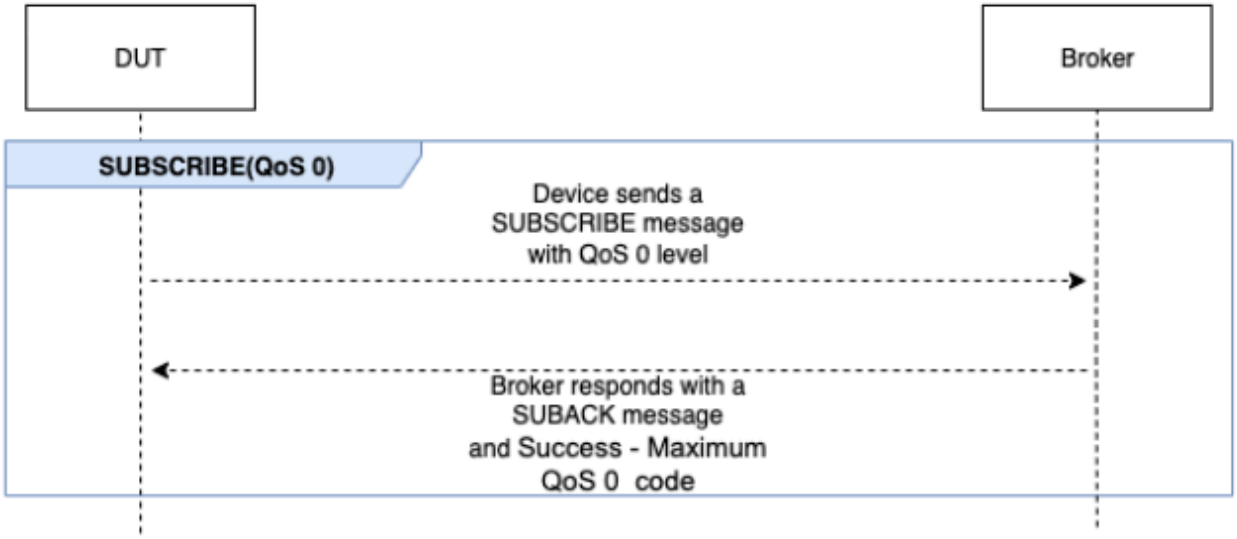


サブスクライブ

このシナリオでは、デバイスがブローカーに対して正常にサブスクライブしているかどうかを検証します。

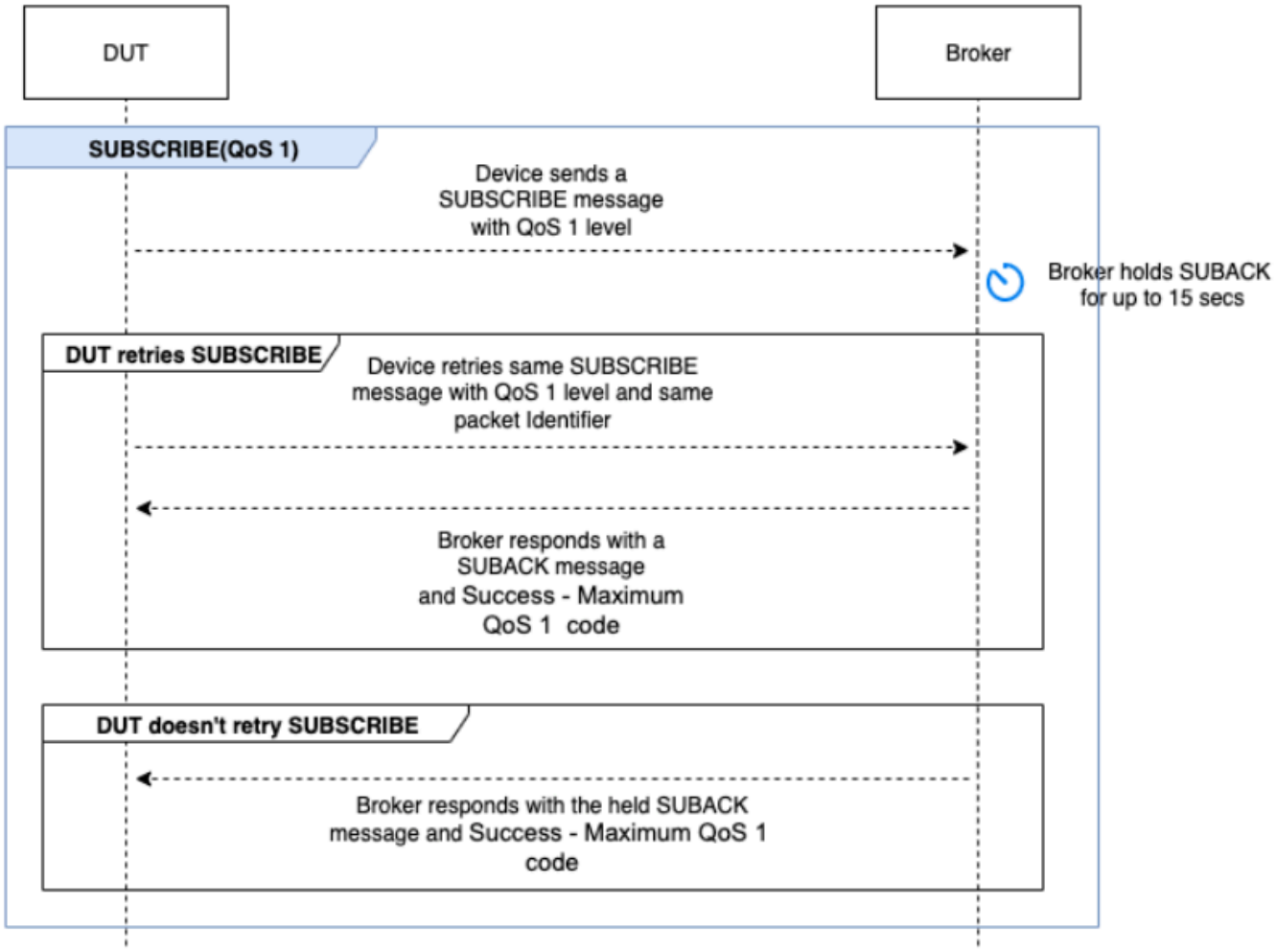
QoS 0

このテストケースは、QoS 0 でのサブスクライブ中にデバイスがブローカーに SUBSCRIBE メッセージを正常に送信するかどうかを検証します。このテストでは、デバイスが SUBACK メッセージを受信するまで待ちません。



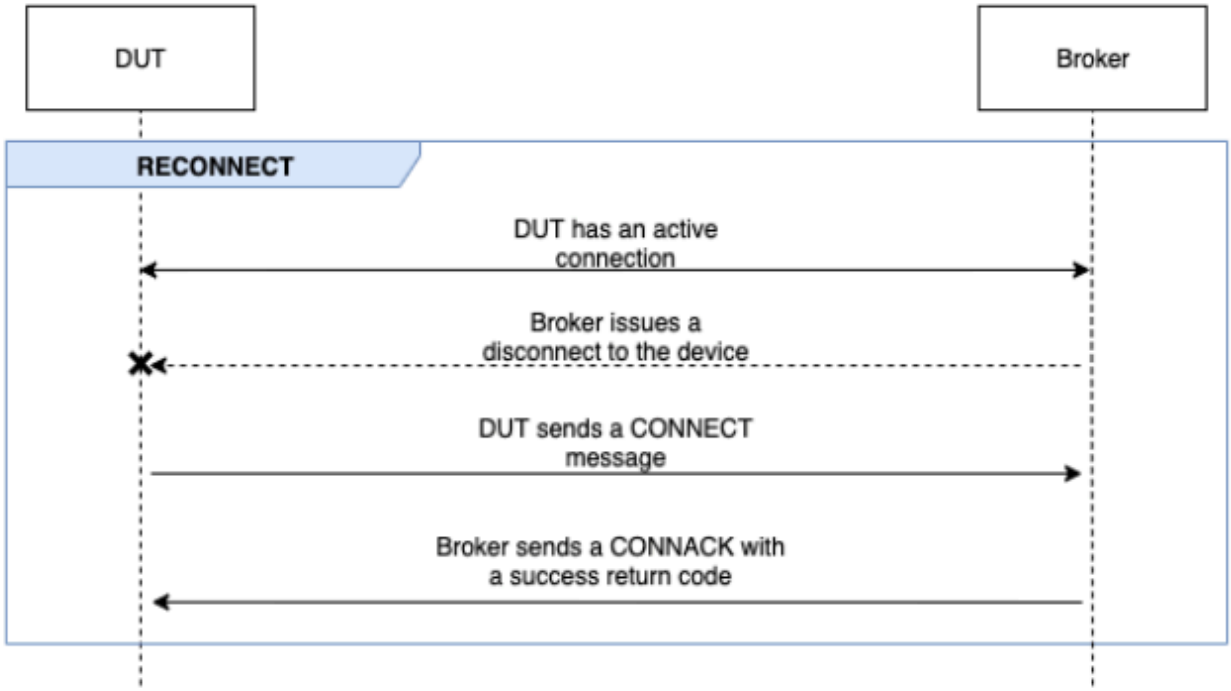
QoS 1

このテストケースでは、デバイスは QoS 1 で 2 つの SUBSCRIBE メッセージをブローカーに送信することが想定されます。最初の SUBSCRIBE メッセージの後、ブローカーは最大 15 秒待ってから、応答します。デバイスは、15 秒以内に同じパケット ID を使用して元の SUBSCRIBE メッセージを再試行する必要があります。その場合、ブローカーは SUBACK メッセージを返し、テストが検証します。デバイスが SUBSCRIBE を再試行しない場合、元の SUBACK がデバイスに送信され、テストはシステムメッセージとともに警告付きで合格とマークされます。テスト実行中にデバイスが接続を失って再接続した場合、テストシナリオは失敗することなくリセットされます。そのため、デバイスはテストシナリオのステップを再実行する必要があります。



再接続

このシナリオでは、デバイスが正常に接続から切断された後に、デバイスがブローカーと正常に再接続するかどうかを検証します。テストスイート中にデバイスを複数回接続しても、Device Advisor はデバイスを切断しません。代わりに、テストを [Pass] (合格) としてマークします。



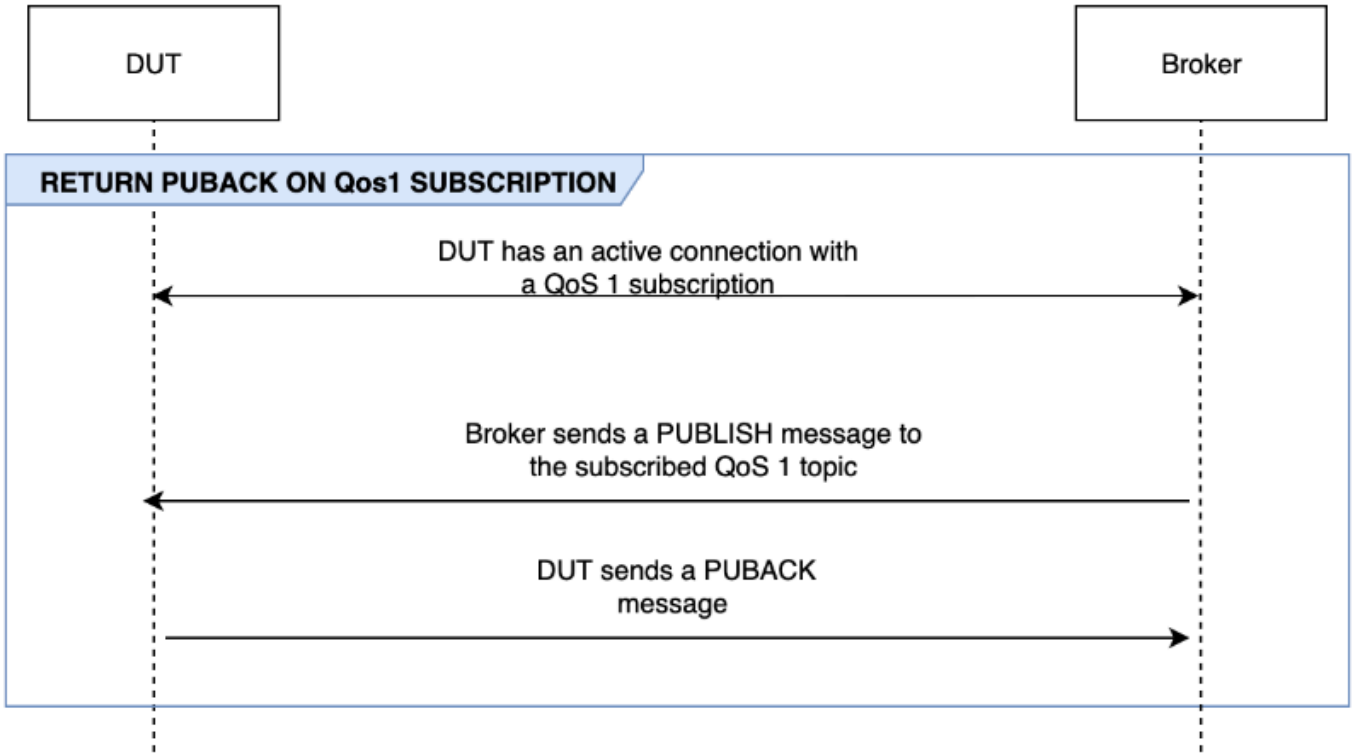
高度なテストの実行

このフェーズでは、テストケースはより複雑なテストを連続して実行し、デバイスがベストプラクティスに従っているかどうかを検証します。これらの高度なテストは選択可能で、必要ない場合はオプトアウトできます。それぞれの高度なテストには、シナリオの要求に応じて独自のタイムアウト値があります。

QoS 1 サブスクリプションで PUBACK を返す

> **Note**
>
> このシナリオは、デバイスが QoS 1 サブスクリプションを実行できる場合にのみ選択してください。

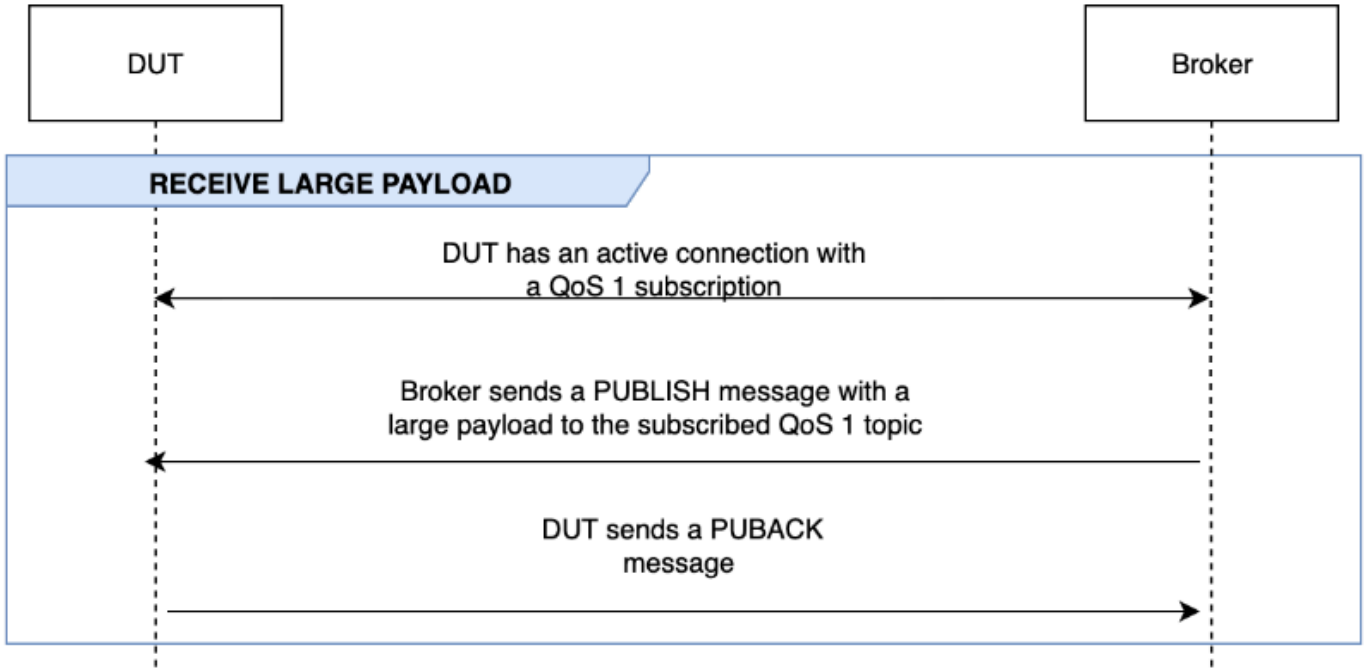
このシナリオでは、デバイスがトピックをサブスクライブしてブローカーから PUBLISH メッセージを受信した後に PUBACK メッセージを返すかどうかを検証します。



大きなペイロードを受け取る

> **Note**
>
> このシナリオは、デバイスが QoS 1 サブスクリプションを実行できる場合にのみ選択してください。

このシナリオでは、ペイロードが大きい QoS 1 トピックの PUBLISH メッセージをブローカーから受信した後、デバイスが PUBACK メッセージで応答するかどうかを検証します。想定されるペイロードの形式は、LONG_PAYLOAD_FORMAT オプションを使用して設定できます。



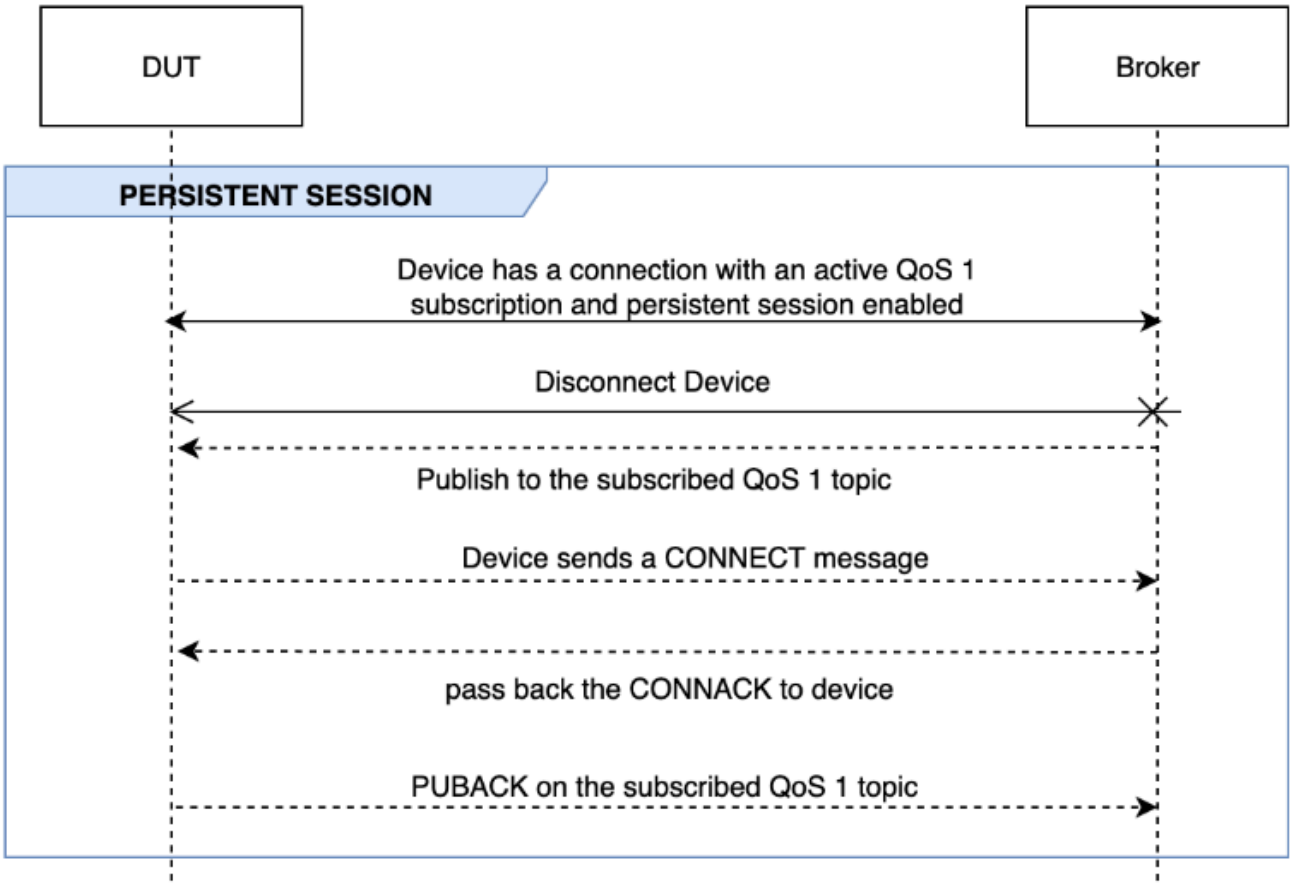
永続セッション

> **Note**
>
> このシナリオは、デバイスが QoS 1 サブスクリプションを実行でき、永続セッションを維持できる場合にのみ選択してください。

このシナリオは、永続セッションを維持する際のデバイスの動作を検証します。以下の条件が満たされると、テストは検証されます。

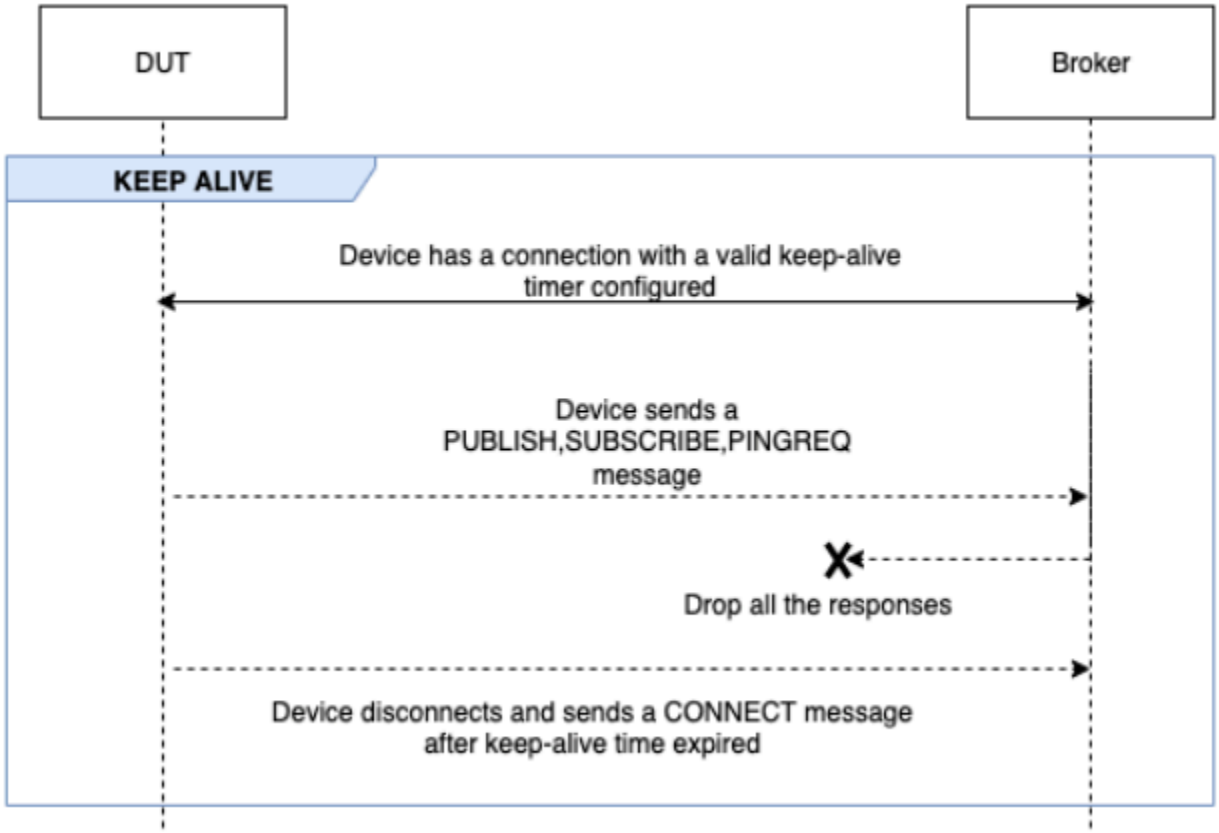
- デバイスは、アクティブな QoS 1 サブスクリプションと永続セッションが有効になっているブローカーに接続します。
- デバイスはセッション中にブローカーから正常に切断されます。
- デバイスはブローカーに再接続し、そのトリガートピックへのサブスクリプションを再開します。これらのトピックを明示的に再サブスクライブする必要はありません。
- デバイスは、サブスクライブされたトピックについてブローカーに保存されたメッセージを正常に受信し、想定どおりに動作します。

永 AWS IoT 続セッションの詳細については、「MQTT 永続セッションの使用」を参照してください。



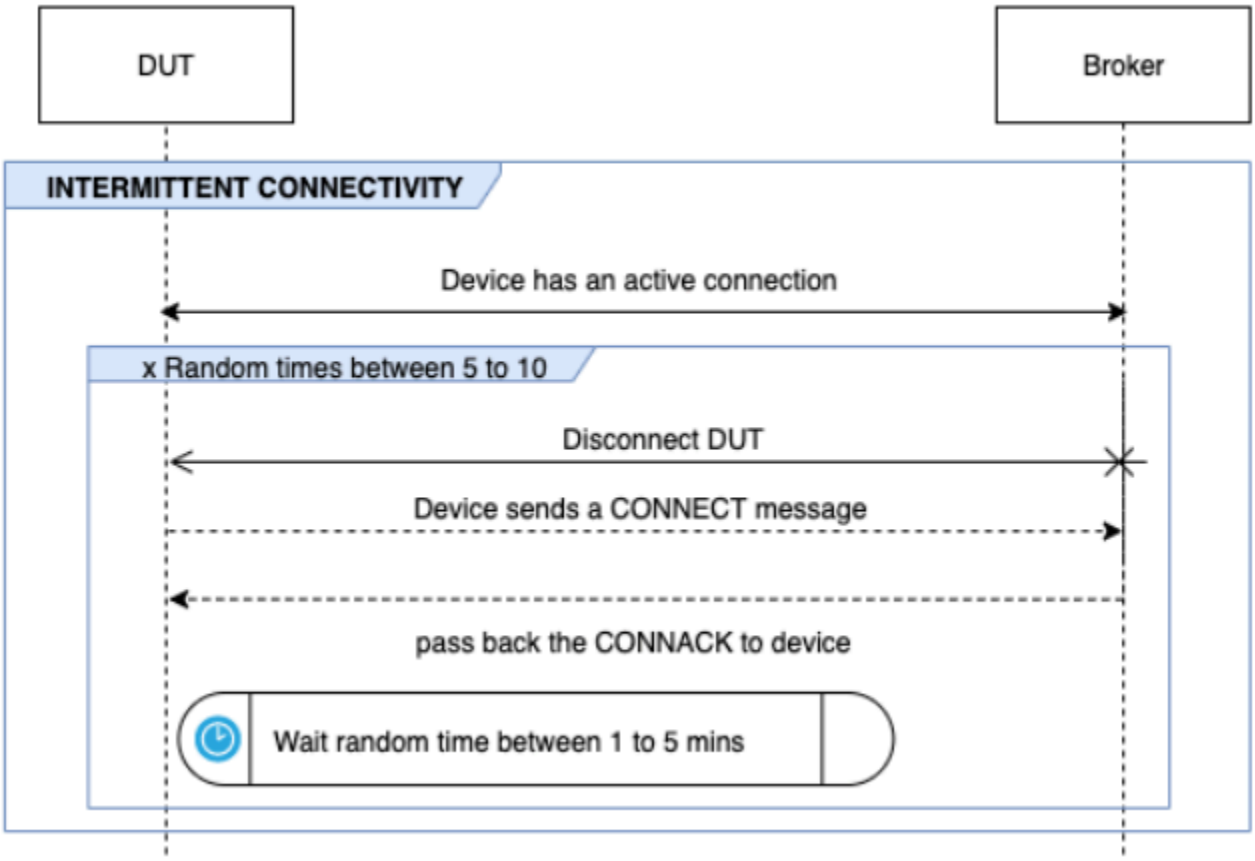
KEEP ALIVE

このシナリオでは、デバイスがブローカーから ping 応答を受信しない後に正常に切断されるかどうかを検証します。接続には有効なキープアライブタイマーが設定されている必要があります。このテストの一環として、ブローカーは、PUBLISH、SUBSCRIBE、および PINGREQ メッセージに送信されるすべての応答をブロックします。テスト対象のデバイスが、MQTT 接続を切断しているかどうかも検証します。



断続的な接続

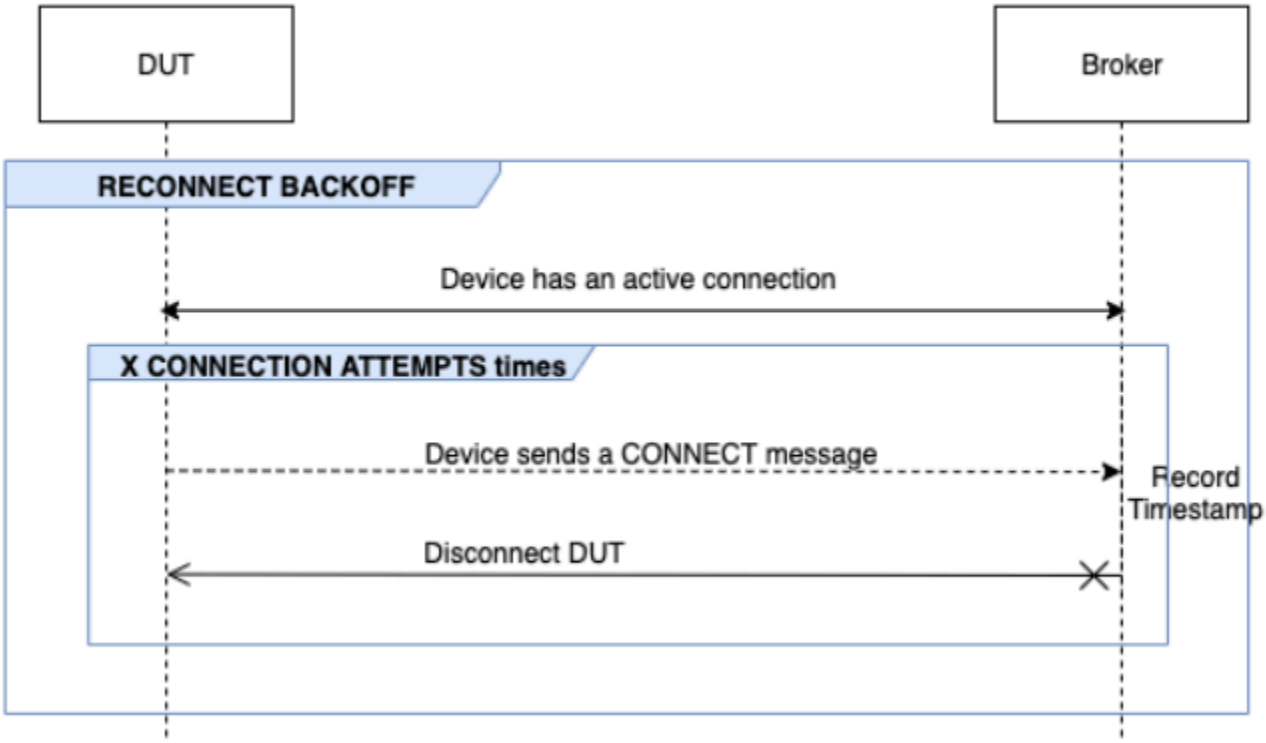
このシナリオでは、ブローカーがデバイスをランダムな間隔で一定時間切断した後に、デバイスがブローカーに再び接続できるかどうかを検証します。



再接続のバックオフ

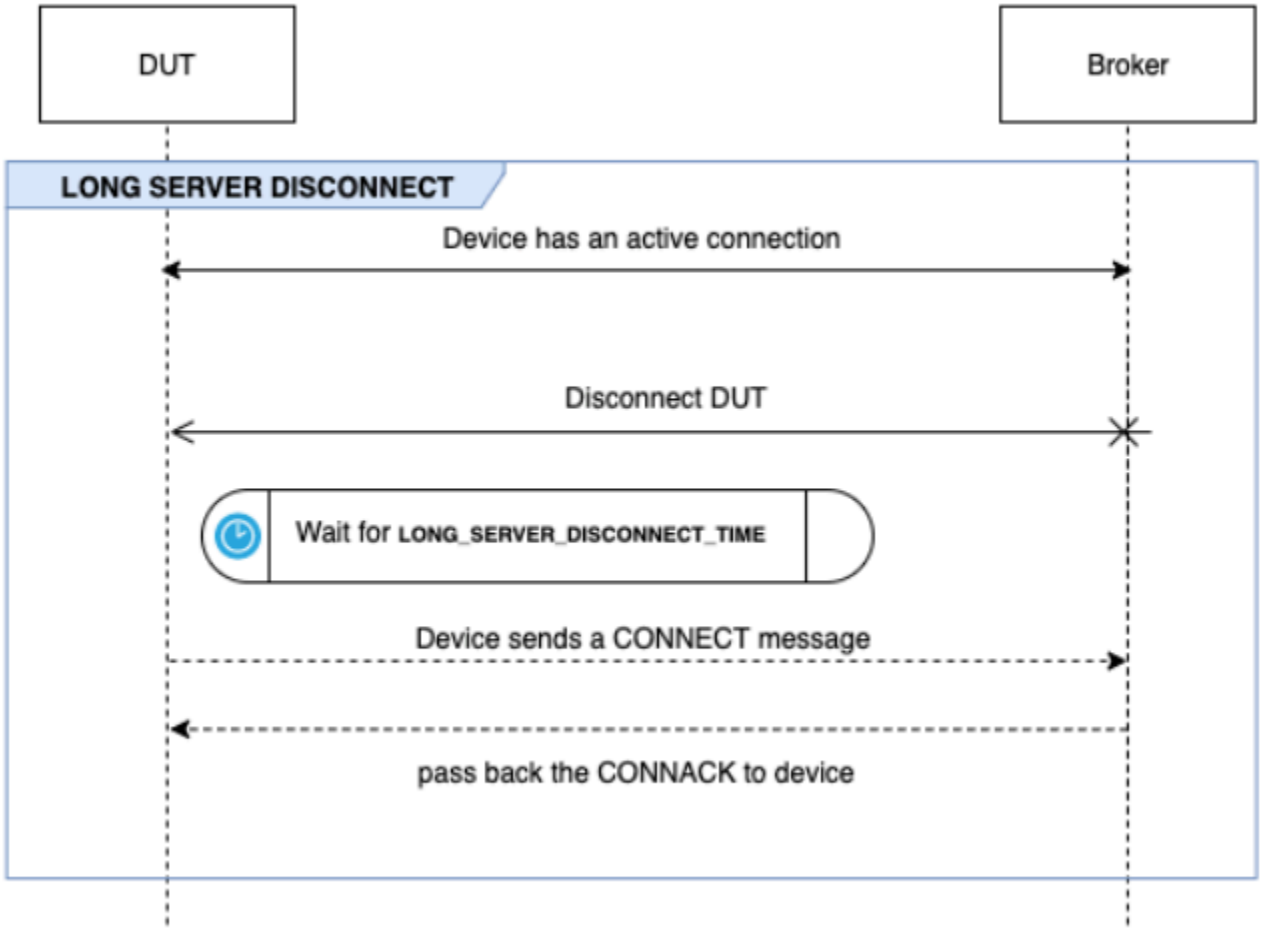
このシナリオでは、ブローカーが複数回接続を切断したときに、デバイスにバックオフメカニズムが実装されているかどうかを検証します。Device Advisor は、バックオフタイプを指数関数、ジッター、線形、または定数として報告します。バックオフの試行回数は、BACKOFF_CONNECTION_ATTEMPTS オプションを使用して設定できます。デフォルト値は 5 です。この値は 5 ~ 10 の間で設定できます。

このテストに合格するには、テスト対象のデバイスに、エクスポネンシャルバックオフとジッターメカニズムを実装することをお勧めします。



長期のサーバー切断

このシナリオでは、ブローカーがデバイスを長時間 (最大 120 分) 切断した後に、デバイスが正常に再接続できるかどうかを検証します。サーバーを切断する時間は、LONG_SERVER_DISCONNECT_TIME オプションを使用して設定できます。デフォルト値は 120 分です。この値は 30 分から 120 分まで設定できます。



追加実行時間

追加実行時間は、上記のすべてのテストを完了してからテストケースを終了するまでにテストが待機する時間です。顧客はこの追加時間を利用して、デバイスとブローカーとの間のすべての通信を監視および記録します。追加実行時間は、ADDITIONAL_EXECUTION_TIME オプションを使用して設定できます。デフォルトでは、このオプションは 0 分に設定されており、0 ~ 120 分に設定できます。

MQTT 長期テスト設定オプション

MQTT 長期テストで提供される設定オプションはすべてオプションです。以下のオプションが利用できます。

オペレーション

デバイスが実行するオペレーションのリスト (CONNECT、PUBLISH および SUBSCRIBE など)。テストケースは、指定されたオペレーションに基づいてシナリオを実行します。指定されていないオペレーションは有効とみなされます。

```
{
"OPERATIONS": ["PUBLISH", "SUBSCRIBE"]
//by default the test assumes device can CONNECT
}
```

シナリオ

選択したオペレーションに基づいて、テストケースはシナリオを実行してデバイスの動作を検証します。シナリオには、次の 2 つのタイプがあります。

- 基本シナリオは、デバイスが設定の一部として上記で選択したオペレーションを実行できるかどうかを検証する簡単なテストです。これらは、構成で指定されたオペレーションに基づいて事前に選択されています。設定に追加の入力は必要ありません。
- 高度なシナリオは、デバイスに対して実行されるより複雑なシナリオで、デバイスが実際の条件を満たしたときにベストプラクティスに従っているかどうかを検証します。これらはオプションで、シナリオの配列としてテストスイートの設定入力に渡すことができます。

```
{
    "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:

すべての基本テストが完了するまでのテストケースの最大待機時間。デフォルト値は 60 分です。この値は 30 分から 120 分まで設定できます。

LONG_SERVER_DISCONNECT_TIME:

長期のサーバー切断テスト中に、テストケースがデバイスを切断して再接続するまでにかかった時間。デフォルト値は 60 分です。この値は 30 分から 120 分まで設定できます。

ADDITIONAL_EXECUTION_TIME:

このオプションを設定すると、すべてのテストが完了した後、デバイスとブローカー間のイベントを監視するための時間ウィンドウが設けられます。デフォルト値は 0 分です。この値は 0 から 120 分まで設定できます。

BACKOFF_CONNECTION_ATTEMPTS:

このオプションは、テストケースによってデバイスが切断される回数を設定します。これは再接続バックオフテストで使用されます。デフォルト値は 5 回です。この値は 5 ~ 10 の間で設定できます。

LONG_PAYLOAD_FORMAT:

デバイスがサブスクライブしている QoS 1 トピックにテストケースを発行するときにデバイス想定するメッセージペイロードの形式。

API テストケースの定義:

```
{
"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 長期テストケース概要ログ

MQTT 長期テストケースは、通常のテストケースよりも長時間実行されます。実行中のデバイス接続、発行、サブスクライブなどの重要なイベントを一覧表示する概要ログが別途提供されます。詳細には、テストされた内容、テストされなかったもの、失敗したものが含まれます。ログの最後には、テストケースの実行中に発生したすべてのイベントの概要がテストに含まれます。これには、以下が含まれます。

- デバイスに設定されているキープアライブタイマー。
- デバイスに設定された永続セッションフラグ。
- テスト実行中のデバイス接続数。
- デバイス再接続バックオフタイプ (再接続バックオフテストで検証された場合)。
- テストケースの実行中にデバイスが発行したトピック。
- テストケースの実行中にデバイスがサブスクライブしたトピック。

AWS IoT Core デバイスの場所

AWS IoT Core Device Location 機能を使用する前に、この機能の利用規約を確認してください。AWS は、検索の実行に使用される位置データなどの位置情報検索リクエストパラメータや、AWS リージョン 現在使用している の外部にある、選択したサードパーティーのデータプロバイダーにその他の情報を送信する場合があることに注意してください。サードパーティープロバイダーと使用するソルバーは、受信した入力ペイロードに基づいて選択されます。詳細については、「AWS サービス条件」を参照してください。

AWS IoT Core Device Location を使用して、サードパーティーのソルバーを使用して IoT デバイスの場所をテストします。ソルバーは、測定データを解決してデバイスの位置を推定するサードパーティーベンダーが提供するアルゴリズムです。デバイスの位置を特定することで、現場でデバイスを追跡してデバッグし、問題をトラブルシューティングできます。

さまざまなソースから収集された測定データが解決され、位置情報が GeoJSON ペイロードとして報告されます。GeoJSON 形式は、地理データ構造をエンコードするために使用される形式です。ペイロードには、世界測地系 (座標系) (WGS84) に基づくデバイス位置の緯度と経度の座標が含まれています。

トピック

- 測定タイプとソルバー
- AWS IoT Core Device Location の仕組み
- AWS IoT Core Device Location の使用方法
- IoT デバイスの位置を解決する
- Device Location MQTT トピックを使用した AWS IoT Core デバイスロケーションの解決
- 位置ソルバーとデバイスペイロード

測定タイプとソルバー

AWS IoT Core Device Location はサードパーティーベンダーと提携して、測定データを解決し、推定されるデバイスの場所を提供します。次の表は、測定タイプとサードパーティー製のロケーションソルバー、およびサポートされているデバイスに関する情報を示しています。LoRaWAN デバイスとそのデバイスロケーションの設定については、「LoRaWAN リソースの位置を設定する」を参照してください。

> **Note**
>
> 一般的な IoT デバイスと Sidewalk デバイスは、デバイスロケーション MQTT トピックを使用して位置情報を取得できます。Wi-Fi、セルラー、および IP アドレスの測定タイプで、デバイスが定義された GeoJSON 形式で予約済みトピックに測定データを発行する場合、AWS IoT Core デバイスの場所はデバイスの場所を解決できます。GNSS 測定タイプの場合、デバイスは LR11xx チップを使用して測定データをスキャンし、GNSS ソルバーを使用して解決された位置情報を取得する必要があります。LoRaWAN デバイスの位置情報を取得する方法については、 AWS IoT Wireless ドキュメントのLoRaWAN リソースの位置の設定」を参照してください。

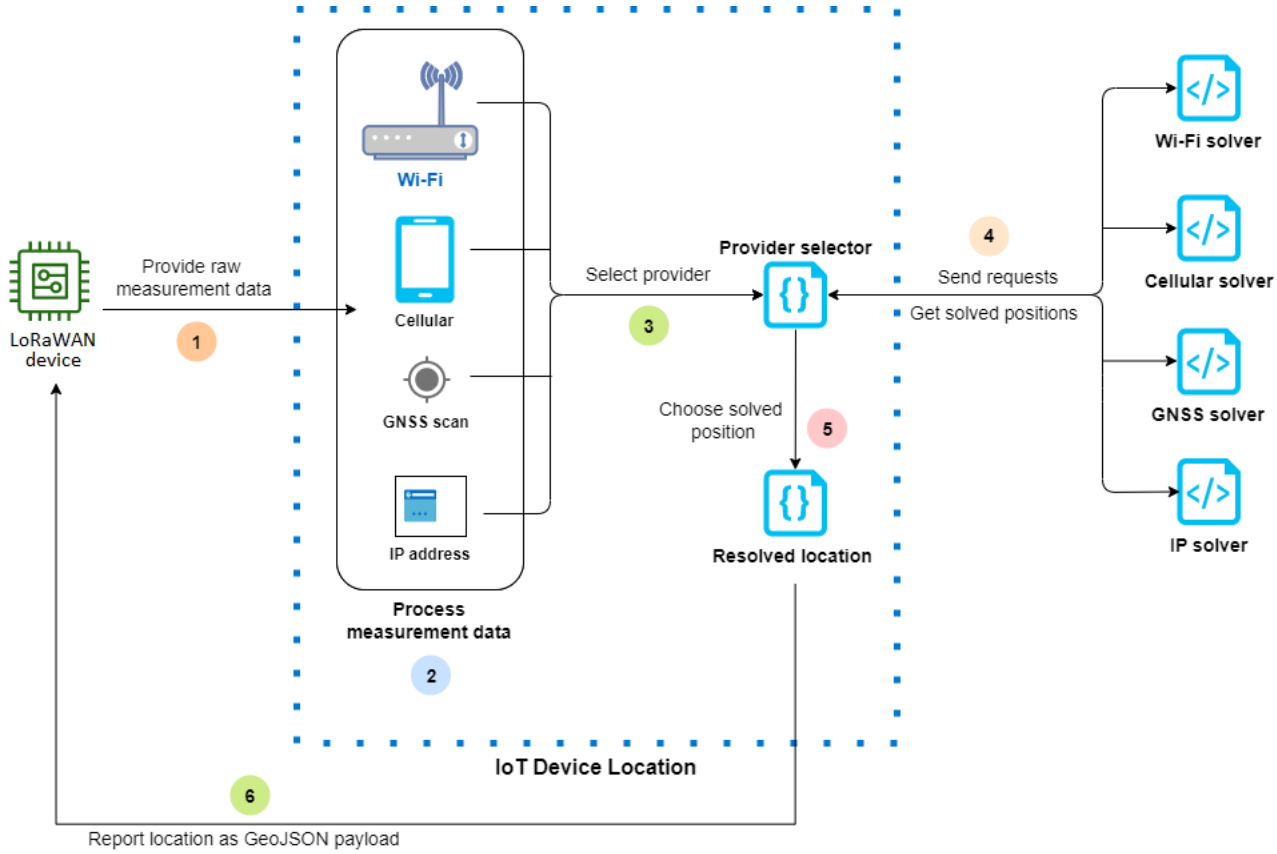
測定タイプとソルバー

計測タイプ	サードパーティーのソルバー	サポートされるデバイス
Wi-Fi アクセスポイント	Wi-Fi ベースのソルバー	一般的な IoT デバイス、LoRa WAN、Amazon Sidewalk デバイス
セルラー無線タワー: GSM、LTE、CDMA、SCDMA、WCMDA、および TD-SCDMA データ	セルラーベースのソルバー	一般的な IoT デバイス、LoRa WAN、Amazon Sidewalk デバイス
IP アドレス	IP リバースルックアップソルバー	TCP/IP 経由で接続する IoT デバイス
GNSS スキャンデータ (NAV メッセージ)	GNSS ソルバー	一般的な IoT デバイス、LoRa WAN、Amazon Sidewalk デバイス
Bluetooth Low Energy (BLE)	BLE ベースのソルバー	Amazon Sidewalk デバイス

ロケーションソルバーの詳細と、さまざまな測定タイプのデバイスペイロードを示す例については、「位置ソルバーとデバイスペイロード」を参照してください。

AWS IoT Core Device Location の仕組み

次の図は、 AWS IoT Core Device Location が測定データを収集し、デバイスの位置情報を解決する方法を示しています。



次の手順は、 AWS IoT Core Device Location の仕組みを示しています。

1. 測定データを受信する

   デバイスの位置に関連する未加工の測定データは、まずデバイスから送信されます。測定データは JSON ペイロードとして指定されます。

2. 測定データを処理する

   測定データが処理され、 AWS IoT Core デバイスロケーションは使用する測定データを選択します。これは、Wi-Fi、セルラー、GNSS スキャン、または IP アドレス情報です。

3. ソルバーを選択する

   サードパーティーのソルバーは、測定データに基づいて選択されます。例えば、測定データに Wi-Fi と IP アドレスの情報が含まれている場合は、Wi-Fi ソルバーと IP リバースルックアップソルバーが選択されます。

4. 解決済み口ケーションを取得する

   API リクエストがソルバープロバイダーに送信され、ロケーションの解決がリクエストされます。 AWS IoT Core デバイスロケーションはソルバーから推定位置情報を取得します。

5. 解決済みの位置を選択する

   解決された位置情報とその精度が比較され、 AWS IoT Core Device Location は最高の精度で位置情報結果を選択します。

6. 位置情報を出力する

   位置情報が GeoJSON ペイロードとして送信されます。ペイロードには、WGS84 の地理座標、精度情報、信頼度レベル、および解決済みの位置が取得されたタイムスタンプが含まれています。

AWS IoT Core Device Location の使用方法

次の手順は、 AWS IoT Core Device Location の使用方法を示しています。

1. 測定データを提供する

   デバイスの位置に関連する未加工の測定データを JSON ペイロードとして指定します。ペイロード測定データを取得するには、デバイスログに移動するか、CloudWatch Logs を使用してペイロードデータ情報をコピーします。JSON ペイロードには 1 つ以上のタイプのデータ測定が含まれている必要があります。さまざまなソルバーのペイロード形式を示す例については、「位置ソルバーとデバイスペイロード」を参照してください。

2. 位置情報を解決する

   AWS IoT コンソールの Device Location ページまたは GetPositionEstimate API オペレーションを使用して、ペイロード測定データを渡してデバイスの位置を解決します。 AWS IoT Core Device Location は、最も高い精度でソルバーを選択し、デバイスの位置を報告します。詳細については、「IoT デバイスの位置を解決する」を参照してください。

3. 位置情報をコピーする

   AWS IoT Core Device Location によって解決され、GeoJSON ペイロードとして報告された位置情報を確認します。ペイロードをコピーして、アプリケーションやその他の AWS のサービスで使用できます。たとえば、ロケーション AWS IoT ルールアクションを使用して、地理的位置データを Amazon Location Service に送信できます。

以下のトピックでは、 AWS IoT Core デバイスロケーションの使用方法とデバイスロケーションペイロードの例を示します。

- IoT デバイスの位置を解決する
- 位置ソルバーとデバイスペイロード

IoT デバイスの位置を解決する

AWS IoT Core Device Location を使用してデバイスからの測定データをデコードし、サードパーティーのソルバーを使用してデバイスの位置を解決します。解決された位置は、地理座標と精度情報を含む GeoJSON ペイロードとして生成されます。デバイスの場所は、 AWS IoT コンソール、AWS IoT ワイヤレス API、または から解決できます AWS CLI。

トピック

- デバイスの位置を解決する (コンソール)
- デバイス位置の解決 (API)
- 位置の解決時のトラブルシューティング

デバイスの位置を解決する (コンソール)

デバイスの位置を解決するには (コンソール)

1. AWS IoT コンソールの Device Location ページに移動します。
2. デバイスログまたは CloudWatch Logs からペイロード測定データを取得し、[ペイロードを介して位置を解決する] セクションに入力します。

   以下のコードは、JSON ペイロードのサンプルを示しています。ペイロードには、セルラーと Wi-Fi の測定データが含まれています。ペイロードに他の種類の測定データが含まれている場合

は、最も精度の高いソルバーが使用されます。詳細な説明とペイロードの例については、「the section called “位置ソルバーとデバイスペイロード”」を参照してください。

> **Note**
>
> JSON ペイロードには、少なくとも 1 つのタイプの測定データが含まれている必要があります。

```
{
    "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. 位置情報を解決するには、[Resolve] (解決) を選択します。

位置情報は、タイプブロブのものであり、地理的データ構造のエンコードに使用される形式である GeoJSON 形式を使用するペイロードとして返されます。ペイロードには以下が含まれます。

- 緯度と経度の情報を含む WGS84 の地理座標。高度情報も含まれる場合があります。
- レポートされる位置情報のタイプ (ポイントなど)。ポイント位置タイプは、位置を GeoJSON ポイントとしてエンコードされた WGS84 の緯度と経度として表します。
- ソルバーによって推定された位置情報と実際のデバイスの位置との差を示す、水平および垂直の精度情報 (メートル単位)。
- 位置推定レスポンスの不確実性を示す信頼度レベル。デフォルト値は 0.68 です。これは、実際のデバイス位置が推定位置の不確実性半径内にある確率が 68% であることを示しています。
- デバイスが位置している都市、州、国、および郵便番号。この情報は、IP リバースルックアップソルバーが使用されている場合にのみ報告されます。
- 位置が解決された日時に対応するタイムスタンプ情報。Unix タイムスタンプ形式を使用します。

以下のコードは、位置の解決によって返される GeoJSON ペイロードのサンプルを示しています。

> **Note**
>
> ロケーションを解決しようとしたときに AWS IoT Core Device Location がエラーを報告した場合は、エラーをトラブルシューティングしてロケーションを解決できます。詳細については、「位置の解決時のトラブルシューティング」を参照してください。

```
{
    "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. リソースロケーションセクションに移動し、 AWS IoT Core Device Location によって報告された位置情報を確認します。ペイロードをコピーして、他のアプリケーションや AWS のサービスで使用できます。例えば、ロケーション を使用して位置データを Amazon Location Service に送信できます。

デバイス位置の解決 (API)

AWS IoT Wireless API を使用してデバイスの位置を解決するには、GetPositionEstimate API オペレーションまたは get-position-estimate CLI コマンドを使用します。ペイロード測定データを入力として指定し、API オペレーションを実行してデバイスの位置を解決します。

> **Note**
>
> `GetPositionEstimate` API オペレーションにはデバイスや状態の情報は保存されず、過去の位置データを取得することもできません。1 回限りのオペレーションを実行して測定データを解決し、推定位置を生成します。位置情報を取得するには、この API オペレーションを実行するたびにペイロード情報を指定する必要があります。

次のコマンドは、この API オペレーションを使用して位置を解決する方法の例を説明しています。

> **Note**
>
> `get-position-estimate` CLI コマンドを実行する場合、出力 JSON ファイルを最初の入力として指定する必要があります。この JSON ファイルには、CLI からの応答として取得した推定位置情報が GeoJSON 形式で格納されます。例えば、次のコマンドは位置情報を *`locationout.json`* ファイルに保存します。

```
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
```

この例では、測定タイプとして Wi-Fi アクセスポイントと IP アドレスの両方が含まれています。AWS IoT Core デバイスの場所は Wi-Fi ソルバーと IP リバースルックアップソルバーから選択し、精度の高いソルバーを選択します。

解決された位置は、地理的データ構造のエンコードに使用される形式である GeoJSON 形式を使用するペイロードとして返されます。その後、*locationout.json* ファイルに保存されます。ペイロードには、WGS84 の緯度と経度の座標、精度と信頼度レベルの情報、位置データタイプ、および位置が解決されたタイムスタンプが含まれています。

```
{
    "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"
    }
}
```

位置の解決時のトラブルシューティング

ロケーションを解決しようとすると、次のいずれかのエラーコードが表示されることがあります。AWS IoT Core デバイスロケーションは GetPositionEstimate API オペレーションの使用時にエラーを生成するか、 AWS IoT コンソールでエラーに対応する行番号を参照することがあります。

- 400 エラー

このエラーは、デバイスペイロード JSON の形式を AWS IoT Core Device Location で検証できないことを示します。このエラーは、次の理由で発生する可能性があります。

- JSON 測定データの形式が正しくない。
- ペイロードにタイムスタンプ情報のみが含まれている。
- IP アドレスなどの測定データパラメータが無効である。

このエラーを解決するには、JSON が正しくフォーマットされ、1 つ以上の測定タイプのデータが入力として含まれているかどうかを確認してください。IP アドレスが無効な場合、有効な IP アドレスを指定してエラーを解決する方法については、「IP リバースルックアップソルバー」を参照してください。

- 403 エラー

  このエラーは、API オペレーションを実行したり、 AWS IoT コンソールを使用してデバイスの場所を取得したりするアクセス許可がないことを示します。このエラーを解決するには、このアクションを実行するために必要なアクセス許可があることを確認してください。このエラーは、AWS マネジメントコンソール セッションまたは AWS CLI セッショントークンの有効期限が切れている場合に発生する可能性があります。このエラーを解決するには、 を使用するようにセッショントークンを更新するか AWS CLI、 からログアウト AWS マネジメントコンソール してから、 認証情報を使用してログインします。

- 404 エラー

  このエラーは、 AWS IoT Core Device Location によって位置情報が見つからないか、解決されなかったことを示します。このエラーは、測定データ入力のデータが十分ではないなどの場合に発生する可能性があります。例えば、次のようになります。

  - MAC アドレスまたはセルラータワーの情報が不十分です。
  - この IP アドレスでは、位置を検索したり取得したりすることはできません。
  - GNSS ペイロードが不十分です。

  このような場合のエラーを解決するには、測定データにデバイスの位置を特定するのに必要な情報が十分に含まれているかどうかを確認してください。

- 500 エラー

  このエラーは、 AWS IoT Core Device Location が位置を解決しようとしたときに、内部サーバー例外が発生したことを示しています。このエラーを修正するには、セッションを更新して、解決する測定データを送信し直してください。

Device Location MQTT トピックを使用した AWS IoT Core デバイスロケーションの解決

リザーブド MQTT トピックを使用して、 Device Location 機能を使用して AWS IoT Core デバイスの最新の位置情報を取得できます。

デバイスの位置情報 MQTT トピックの形式

AWS IoT Core Device Location の予約済みトピックでは、次のプレフィックスを使用します。

`$aws/device_location/{customer_device_id}/`

完全なトピックを作成するには、まず *customer_device_id* を、デバイスを識別するために使用する固有の ID に置き換えてください。for LoRaWAN デバイスや Sidewalk デバイスWirelessDeviceIdなどの を指定し、デバイスが AWS IoT モノとして登録されている場合は *thingName*を指定することをお勧めします。次に、以下のセクションに示すように、get_position_estimate または get_position_estimate/accepted などのトピックスタブをトピックに追加します。

> **Note**
>
> *{customer_device_id}* に含むことができるのは、英文字、数字、およびダッシュのみです。デバイス位置トピックに登録する場合、プラス記号 (+) をワイルドカードとして使用することのみが可能です。例えば、*{customer_device_id}* に対して + ワイルドカードを使用して、デバイスの位置情報を取得できます。トピック $aws/device_location/*+*/get_position_estimate/accepted にサブスクライブすると、正常に解決された場合、任意のデバイス ID と一致するデバイスの位置情報を含むメッセージが発行されます。

Device AWS IoT Core Location を操作するために使用される予約済みトピックを次に示します。

デバイス位置 MQTT トピック

Topic	許可されている操作	説明
$aws/device_location/*customer_*	発行	デバイスは、このトピックに発行して、スキャンされた未加工の測定データを AWS IoT Core Device Location で解決します。

Topic	許可されている操作	説明
device_id /get_position_estimate		
$aws/device_location/*customer_device_id* /get_position_estimate/accepted	Subscribe	AWS IoT Core Device Location は、デバイスの位置が正常に解決されると、位置情報をこのトピックに発行します。
$aws/device_location/*customer_device_id* /get_position_estimate/rejected	Subscribe	AWS IoT Core Device Location は、デバイスの場所の解決に失敗すると、エラー情報をこのトピックに発行します。

デバイスの位置情報 MQTT トピックのポリシー

デバイスの場所トピックからメッセージを受信するには、デバイスが AWS IoT デバイスゲートウェイに接続し、MQTT トピックをサブスクライブすることを許可するポリシーを使用する必要があります。

以下は、さまざまなトピックのメッセージを受信するために必要なポリシーの例です。

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

デバイスの位置情報トピックとペイロード

以下は、 AWS IoT Core Device Location トピック、メッセージペイロードの形式、各トピックのポリシー例を示しています。

トピック

- /get_position_estimate
- /get_position_estimate/accepted
- /get_position_estimate/rejected

/get_position_estimate

このトピックにメッセージを発行して、デバイスロケーションによって解決される未加工の測定データを AWS IoT Core デバイスから取得します。

```
$aws/device_location/customer_device_id/get_position_estimate
```

AWS IoT Core Device Location は、 /get_position_estimate/acceptedまたは のいずれかに発行することで応答します/get_position_estimate/rejected。

> **Note**
>
> このトピックに発行されるメッセージは、有効な JSON ペイロードでなければなりません。入力メッセージが有効な JSON 形式でない場合、レスポンスは返されません。詳細については、「メッセージペイロード」を参照してください。

メッセージペイロード

メッセージペイロード形式は、 AWS IoT Wireless API オペレーションリクエスト本文 と同様の構造に従いますGetPositionEstimate。以下を含みます。

- 位置が解決された日時に対応するオプションの Timestamp 文字列。Timestamp 文字列の最小長は 1、最大長は 10 です。
- リクエストをレスポンスにマッピングできる、オプションの MessageId 文字列。この文字列を指定すると、get_position_estimate/accepted または get_position_estimate/rejected トピックにパブリッシュされるメッセージにはこの MessageId が含まれます。MessageID 文字列の最小長は 1、最大長は 256 です。
- 次の 1 つ以上の測定タイプを含む、デバイスからの測定データ。
  - WiFiAccessPoint
  - CellTowers
  - IpAddress
  - Gnss

以下は、メッセージペイロードのサンプルを示しています。

```
{
    "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
    }
}
```

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 Location は、デバイスの解決された位置情報を返すときに、このトピックへのレスポンスを発行します。位置情報は GeoJSON 形式で返されます。

```
$aws/device_location/customer_device_id/get_position_estimate/accepted
```

次は、メッセージペイロードとポリシーの例を示しています。

メッセージペイロード

次は、GeoJSON 形式のメッセージペイロードの例を示しています。raw 測定データMessageIdでを指定し、 AWS IoT Core Device Location が位置情報を正常に解決した場合、メッセージペイロードは同じMessageId情報を返します。

```
{
    "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"
    }
}
```

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

```
{
    "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 Device Location は、デバイスの場所の解決に失敗すると、このトピックにエラーレスポンスを発行します。

```
$aws/device_location/customer_device_id/get_position_estimate/rejected
```

次は、メッセージペイロードとポリシーの例を示しています。これらのエラーの詳細については、「位置の解決時のトラブルシューティング」を参照してください。

メッセージペイロード

以下は、エラーコードとメッセージを提供するメッセージペイロードの例です。これは、 AWS IoT Core Device Location が位置情報の解決に失敗した理由を示しています。raw 測定データを提供するMessageIdときに を指定し、 AWS IoT Core Device Location が位置情報の解決に失敗した場合、メッセージペイロードに同じMessageId情報が返されます。

```
{
    "errorCode": 500,
    "errorMessage":"Internal server error",
    "messageId": "ABCD1"
}
```

ポリシーの例

以下に示しているのは、必要なポリシーの例です。

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

位置ソルバーとデバイスペイロード

ロケーションソルバーは、IoT デバイスの位置を解決するために使用できるアルゴリズムです。AWS IoT Core デバイスロケーションは、次のロケーションソルバーをサポートしています。これらの測定タイプの JSON ペイロード形式の例、ソルバーがサポートするデバイス、および位置の解決方法が表示されます。

デバイスの位置を特定するには、これらの測定データタイプを 1 つ以上指定してください。すべての測定データを組み合わせた単一の解決済み位置が返されます。

トピック

- Wi-Fi ベースのソルバー
- セルラーベースのソルバー

- IP リバースルックアップソルバー
- GNSS ソルバー
- Amazon Sidewalk 対応デバイスの BLE ソルバー

Wi-Fi ベースのソルバー

Wi-Fi ベースのソルバーを使用して、Wi-Fi アクセスポイントからのスキャン情報を使用して位置を解決します。ソルバーは DDoS テクノロジーをサポートしており、TCP/IP、LoRaWAN ワイヤレスデバイス、Amazon Sidewalk 対応デバイスを介して接続する一般的な IoT デバイスの位置を計算するために使用できます。

LoRaWAN デバイスには、受信した Wi-Fi スキャン情報をデコードする LoRa Edge チップセットが必要です。LoRa Edge は、位置情報アプリケーションを対象とした長距離 LoRa トランシーバー、マルチコンステレーション GNSS スキャナー、パッシブ Wi-Fi MAC スキャナーを統合した超低電力プラットフォームです。デバイスからアップリンクメッセージを受信すると、Wi-Fi スキャンデータが AWS IoT Core Device Location に送信され、その場所は Wi-Fi スキャン結果に基づいて推定されます。次に、デコードされた情報は Wi-Fi ベースのソルバーに渡され、位置情報が取得されます。

Amazon Sidewalk 対応デバイスの位置情報データの解決の詳細については、「 デAWS IoT Core for Amazon Sidewalk ベロッパーガイド」を参照してください。

Wi-Fi ベースのソルバーペイロードの例

次のコードは、測定データを含むデバイスからの JSON ペイロードの例を示しています。 AWS IoT Core Device Location はこのデータを入力として受信すると、ソルバープロバイダーに HTTP リクエストを送信して位置情報を解決します。情報を取得するには、MAC アドレスと RSS (受信信号強度) の値を指定します。そのためには、この形式を使用して JSON ペイロードを指定するか、GetPositionEstimate API オペレーションの WifiAccessPoints オブジェクトパラメータを使用してください。

```
{
    "Timestamp": 1664313161,    // optional
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",  // required
            "Rss": -75                          // required
        }
    ]
}
```

セルラーベースのソルバー

セルラーベースのソルバーを使用すると、セルラーラジオタワーから取得した測定データを使用して位置を解析できます。ソルバーは以下の技術をサポートしています。これらのテクノロジーのいずれかまたはすべてからの測定データを含めた場合でも、解決された単一の位置情報が取得されます。

- GSM
- CDMA
- WCDMA
- TD-SCDMA
- LTE

セルラーベースのソルバーペイロードの例

次のコードは、セルラー測定データを含むデバイスからの JSON ペイロードの例を示しています。AWS IoT Core Device Location はこのデータを入力として受信すると、ソルバープロバイダーに HTTP リクエストを送信して位置情報を解決します。情報を取得するには、コンソールでこの形式を使用して JSON ペイロードを指定するか、GetPositionEstimate API オペレーションの CellTowers パラメータの値を指定します。これらのセルラーテクノロジーのいずれかまたはすべてを使用してパラメータの値を指定することにより、測定データを提供できます。

LTE (長期的進化)

この測定データを使用するときは、モバイルネットワークのネットワークや国コードなどの情報、およびローカル ID に関する情報を含むオプションの追加パラメータを指定する必要があります。次のコードは、ペイロード形式の例を示しています。これらのパラメータの詳細については、「LTE object」(LTE オブジェクト) を参照してください。

```
{
    "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 (モバイル通信用グローバルシステム)

この測定データを使用するときは、モバイルネットワークのネットワークと国コード、基地局情報、オプションの追加パラメータなどの情報を指定する必要があります。次のコードは、ペイロード形式の例を示しています。これらのパラメータの詳細については、「GSM object」(GSM オブジェクト)を参照してください。

```
{
    "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 (符号分割多元接続)

この測定データを使用するときは、信号電力や識別情報、基地局情報、オプションの追加パラメータなどの情報を指定する必要があります。次のコードは、ペイロード形式の例を示しています。これらのパラメータの詳細については、「CDMA object」(CDMA オブジェクト) を参照してください。

```
{
    "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 (広帯域符号分割多元接続)

この測定データを使用するときは、ネットワークと国コード、信号電力と識別情報、基地局情報、オプションの追加パラメータなどの情報を指定する必要があります。次のコードは、ペイロード形式の例を示しています。これらのパラメータの詳細については、「CDMA object」(CDMA オブジェクト)を参照してください。

```
{
    "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 (時分割同期符号分割多元接続)

この測定データを使用するときは、ネットワークと国コード、信号電力と識別情報、基地局情報、オプションの追加パラメータなどの情報を指定する必要があります。次のコードは、ペイロード形式の例を示しています。これらのパラメータの詳細については、「CDMA object」(CDMA オブジェクト)を参照してください。

```
{
    "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 リバースルックアップソルバー

IP リバースルックアップソルバーを使用すると、IP アドレスを入力として使用して位置を特定できます。ソルバーは、プロビジョニングされたデバイスから位置情報を取得できます AWS IoT。IPv4 または IPv6 の標準パターン、または IPv6 の 16 進圧縮パターンのいずれかの形式を使用して、IP ア

ドレス情報を指定します。次に、デバイスが置かれている都市や国などの追加情報を含む、解決済みの推定位置情報を取得します。

> **Note**
>
> IP リバースルックアップを使用することで、特定の家庭や通りの住所を特定したり探し出したりする目的で利用しないことに同意するものとします。

IP リバースルックアップソルバーペイロードの例

次のコードは、測定データを含むデバイスからの JSON ペイロードの例を示しています。 AWS IoT Core Device Location は、測定データ内の IP アドレス情報を受信すると、ソルバープロバイダーのデータベースでこの情報を検索し、位置情報の解決に使用されます。情報を取得するには、この形式を使用して JSON ペイロードを指定するか、GetPositionEstimate API オペレーションの Ip パラメータの値を指定します。

> **Note**
>
> このソルバーを使用すると、座標に加えてデバイスが配置されている都市、州、国、郵便番号も報告されます。例については、デバイスの位置を解決する (コンソール)を参照してください。

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress":"54.240.198.35"
      }
}
```

GNSS ソルバー

GNSS (グローバルナビゲーションサテライトシステム) ソルバーを使用して、GNSS スキャン結果メッセージまたは NAV メッセージに含まれる情報を使用してデバイスの位置を取得します。オプションで追加の GNSS 支援情報を指定できます。これにより、ソルバーが信号を検索するために使用する必要のある変数の数が減ります。位置、高度、キャプチャ時間、精度情報を含むこの支援情報を提供することで、ソルバーは表示されている衛星を簡単に識別し、デバイスの位置を計算できます。

このソルバーは、LoRaWAN デバイスと Amazon Sidewalk 対応デバイス、およびプロビジョニングされた AWS IoT デバイスでプロビジョニングされた他のデバイスで使用できます AWS IoT。一般的な IoT デバイスでは、デバイスが GNSS による位置推定をサポートしている場合、デバイスから GNSS スキャン情報を受信すると、トランシーバーが位置情報を解決します。LoRaWAN デバイスの場合、デバイスには LoRa Edge チップセットが搭載されている必要があります。デバイスからアップリンクメッセージを受信すると、GNSS スキャンデータが AWS IoT for LoRaWAN または AWS IoT for Amazon Sidewalk に送信され、トランシーバーからのスキャン結果に基づいて場所が推定されます。

GNSS ソルバーペイロードの例

次のコードは、測定データを含むデバイスからの JSON ペイロードの例を示しています。 AWS IoT Core Device Location は、測定データ内のペイロードを含む GNSS スキャン情報を受信すると、トランシーバーと含まれる追加の支援情報を使用してシグナルを検索し、位置情報を解決します。情報を取得するには、この形式を使用して JSON ペイロードを指定するか、GetPositionEstimate API オペレーションの Gnss パラメータの値を指定します。

> **Note**
>
> AWS IoT Core Device Location がデバイスの位置を解決する前に、ペイロードから送信先バイトを削除する必要があります。

```
{
    "Timestamp": 1664313161,                // optional
    "Gnss": {
        "AssistAltitude": number,           // optional
        "AssistPosition": [ number ],       // optional
        "CaptureTime": number,              // optional
        "CaptureTimeAccuracy": number,      // optional
        "Payload": "string",                // required
        "Use2DSolver": boolean              // optional
   }
}
```

Amazon Sidewalk 対応デバイスの BLE ソルバー

> **Note**
>
> Bluetooth Low Energy ベースのロケーションの場合、 は Amazon Sidewalk に接続され、コミュニティ検出機能が有効になっている近くの Sidewalk ゲートウェイのおおよその場所に基づいてロケーション座標 AWS IoT を返します。Gateway Location Data は AWS コンテンツであり、Amazon Sidewalk に接続されているデバイスの検索を支援する目的でのみ提供され、その目的でのみデータを使用する必要があります。位置情報データを使用およびアクセスできるのは、当社が通常提供するインターフェイスおよび機能のみです。また、当社が提供する Gateway 位置情報データの再識別、リバースエンジニアリング、再マッピングを試みてはなりません。

経由で接続する Amazon Sidewalk エンドデバイスは、Amazon Sidewalk エンドデバイスによって発行された BLE、Wi-Fi、または GNSS アップリンクメッセージを使用して位置データを解決 AWS IoT Core for Amazon Sidewalk できます。詳細については、 デAWS IoT Core for Amazon Sidewalk ベロッパーガイドを参照してください。

イベントメッセージ

このセクションには、モノまたはジョブが更新または変更された AWS IoT ときに によって発行されるメッセージに関する情報が含まれています。ディテクターを作成して、動作中の障害や変更がないかデバイスをモニタリングし、発生したときにアクションをトリガーできる AWS IoT Events サービスについては、「」を参照してくださいAWS IoT Events。

イベントメッセージが生成される方法

AWS IoT は、特定のイベントが発生したときにイベントメッセージを発行します。例えば、モノが追加、更新、または削除されると、イベントがレジストリによって生成されます。各イベントによって、単一のイベントメッセージが送信されます。イベントメッセージは、MQTT を介して JSON ペイロードを使用して公開されます。ペイロードのコンテンツは、イベントの種類によって異なります。

> **Note**
>
> イベントメッセージは一度公開されることが保証されています。複数回発行することが可能です。イベントメッセージの順序は保証されません。

イベントメッセージを受信するためのポリシー

イベントメッセージを受信するには、デバイスが AWS IoT デバイスゲートウェイに接続し、MQTT イベントトピックをサブスクライブできるようにする適切なポリシーを使用する必要があります。また、適切なトピックフィルターを受信登録する必要があります。

以下に示しているのは、ライフサイクルイベントの受信に必要なポリシーの例です。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:/$aws/events/*"
            ]
        }
    ]
}
```

のイベントを有効にする AWS IoT

予約済みトピックのサブスクライバーがメッセージを受信する前に、 API AWS マネジメントコンソール または CLI を使用して、 または からイベントメッセージを有効にする必要があります。さまざまなオプションが管理するイベントメッセージの詳細については、AWS IoT 「イベント設定表」を参照してください。

- イベントメッセージを有効にするには、 AWS IoT コンソールの設定タブに移動し、イベントベースのメッセージセクションでイベントの管理を選択します。管理したいイベントを指定ができます。
- API または CLI を使用して発行されるイベントタイプを制御するには、UpdateResourceEventConfiguration API を呼び出すか、update-event-configurations CLI コマンドを使用します。例:。

```
aws iot update-event-configurations --event-configurations "{\"THING\":{\"Enabled\":
 true}}"
```

> **Note**
>
> 二重引用符 (") はバックスラッシュ (\) でエスケープされます。

現在のイベント設定は、DescribeEventConfigurations API を呼び出すか、describe-event-configurations CLI コマンドを使用することで取得できます。例 :

```
aws iot describe-event-configurations
```

AWS IoT イベント構成設定表

イベントカテゴリ (AWS IoT コンソール: 設定: イベントベースのメッセージ)	**eventConfigurations** キーバリュー (AWS CLI/API)	イベントメッセージのトピック
(AWS CLI/API を使用してのみ設定可能)	CA_CERTIFICATE	$aws/events/certificates/registered/ *caCertificateId*
(AWS CLI/API を使用してのみ設定可能)	CERTIFICATE	$aws/events/presence/connected/ *clientId*
(AWS CLI/API を使用してのみ設定可能)	CERTIFICATE	$aws/events/presence/disconnected/ *clientId*
(AWS CLI/API を使用してのみ設定可能)	CERTIFICATE	$aws/events/subscriptions/subscribed/ *clientId*
(AWS CLI/API を使用してのみ設定可能)	CERTIFICATE	$aws/events/subscriptions/unsubscribed/ *clientId*
ジョブ完了、キャンセル済み	JOB	$aws/events/job/*jobID*/canceled
ジョブ完了、キャンセル済み	JOB	$aws/events/job/*jobID*/cancellation_in_progress
ジョブ完了、キャンセル済み	JOB	$aws/events/job/*jobID*/completed
ジョブ完了、キャンセル済み	JOB	$aws/events/job/*jobID*/deleted

イベントカテゴリ (AWS IoT コンソール: 設定: イベントベースのメッセージ)	**eventConfigurations** キーバリュー (AWS CLI/API)	イベントメッセージのトピック
ジョブ完了、キャンセル済み	JOB	$aws/events/job/*jobID*/deletion_in_progress
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/canceled
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/deleted
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/failed
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/rejected
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/removed
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/succeeded
ジョブ実行 : 成功、失敗、拒否、キャンセル、削除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/timed_out
モノ : 作成、更新、削除	THING	$aws/events/thing/ *thingName* /created
モノ : 作成、更新、削除	THING	$aws/events/thing/ *thingName* /updated

イベントカテゴリ (AWS IoT コンソール: 設定: イベントベースのメッセージ)	**eventConfigurations** キーバリュー (AWS CLI/API)	イベントメッセージのトピック
モノ : 作成、更新、削除	THING	$aws/events/thing/ *thingName* /deleted
モノグループ : 追加、削除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /created
モノグループ : 追加、削除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /updated
モノグループ : 追加、削除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /deleted
モノグループ階層 : 追加、削除	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /added
モノグループ階層 : 追加、削除	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /removed

イベントカテゴリ (AWS IoT コンソール: 設定: イベントベースのメッセージ)	**eventConfigurations** キーバリュー (AWS CLI/API)	イベントメッセージのトピック
モノグループメンバーシップ : 追加、削除	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /added
モノグループメンバーシップ : 追加、削除	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /removed
モノタイプ : 作成、更新、削除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /created
モノタイプ : 作成、更新、削除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /updated
モノタイプ : 作成、更新、削除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /deleted

イベントカテゴリ (AWS IoT コンソール: 設定: イベントベースのメッセージ)	**eventConfigurations** キーバリュー (AWS CLI/API)	イベントメッセージのトピック
モノタイプの関連付け : 追加、削除	THING_TYPE_ASSOCIATION	$aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /added $aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /removed

登録イベント

モノ、モノのタイプ、モノのグループが作成、更新、または削除されたときに、レジストリでのイベントメッセージ発行が可能になります。ただし、これらのイベントはデフォルトでは使用できません。これらのイベントを有効にする方法の詳細については、のイベントを有効にする AWS IoT を参照してください。

レジストリでは、次のイベントタイプを指定できます。

- モノのイベント
- モノのタイプのイベント
- モノのグループのイベント

モノのイベント

モノの作成/更新/削除

レジストリは、モノが作成、更新、または削除されると次のイベントメッセージを発行します。

- $aws/events/thing/*thingName*/created

- $aws/events/thing/*thingName*/updated
- $aws/events/thing/*thingName*/deleted

メッセージには、次のペイロード例が含まれています。

```
{
    "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"
    }
}
```

ペイロードには次の属性が含まれます。

eventType

「THING_EVENT」に設定します。

eventId

一意のイベント ID (文字列)。

timestamp

イベントが発生したときの UNIX タイムスタンプ。

オペレーション

イベントをトリガーしたオペレーション。有効な値は次のとおりです。

- CREATED
- UPDATED
- 削除済み

accountId

AWS アカウント ID。

thingId

作成、更新、または削除されているモノの ID。

thingName

作成、更新、または削除されているモノの名前。

versionNumber

作成、更新、または削除されているモノのバージョン。この値は、モノが作成されると 1 に設定されます。これは、モノが更新されるたびに 1 ずつ増加します。

thingTypeName

新しいモノに関連付けられたモノのタイプの名前 (存在する場合)。そうでない場合は、`null` です。

個の属性

モノに関連付けられた名前と値のペアの集合。

モノのタイプのイベント

モノのタイプ関連のイベント:

- 作成/更新/廃止/復帰/削除されたモノのタイプ
- モノに関連付けまたは関連付け解除されたモノのタイプ

作成/更新/廃止/復帰/削除されたモノのタイプ

レジストリは、モノのタイプが作成、更新、廃止、復帰、または削除されると、次のイベントメッセージを発行します。

- `$aws/events/thingType/thingTypeName/created`
- `$aws/events/thingType/thingTypeName/updated`
- `$aws/events/thingType/thingTypeName/deleted`

メッセージには、次の例のペイロードが含まれています。

```
{
    "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"
}
```

ペイロードには次の属性が含まれます。

eventType

「THING_TYPE_EVENT」に設定します。

eventId

一意のイベント ID (文字列)。

timestamp

イベントが発生したときの UNIX タイムスタンプ。

オペレーション

イベントをトリガーしたオペレーション。有効な値は次のとおりです。

- CREATED
- UPDATED
- 削除済み

accountId

AWS アカウント ID。

thingTypeId

作成、更新、廃止、または削除されているモノのタイプの ID。

thingTypeName

作成、更新、廃止、または削除されているモノのタイプの名前。

isDeprecated

`true` モノのタイプは廃止されている場合。そうでない場合は、`false` です。

deprecationDate

モノのタイプが廃止されたときの UNIX タイムスタンプ。

searchableAttributes

検索に使用できるモノのタイプに関連付けられた名前と値のペアの集合。

propagatingAttributes

伝播属性のリスト。伝達属性には、モノの属性、接続属性、およびユーザープロパティキーを含めることができます。詳細については、「メッセージエンリッチメントの伝播属性の追加」を参照してください。

説明

モノのタイプの説明。

モノに関連付けまたは関連付け解除されたモノのタイプ

レジストリは、モノのタイプがモノに関連付けまたは関連付け解除されると、次のイベントメッセージを発行します。

- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/thingType/`*`typeName`*`/added`
- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/thingType/`*`typeName`*`/removed`

`added` ペイロードの例を次に示します。`removed` メッセージのペイロードは類似しています。

```
{
    "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,
}
```

ペイロードには次の属性が含まれます。

eventId

一意のイベント ID (文字列)。

eventType

「THING_TYPE_ASSOCIATION_EVENT」に設定します。

オペレーション

イベントをトリガーしたオペレーション。次の値を指定できます。

- 追加
- 削除済み

thingId

タイプの関連付けが変更されたモノの ID。

thingName

タイプの関連付けが変更されたモノの名前。

thingTypeName

モノと関連付けられた、または関連付けが解除されたモノのタイプ。

timestamp

イベントが発生したときの UNIX タイムスタンプ。

モノのグループのイベント

モノのグループ関連のイベント:

- 作成/更新/削除されたモノのグループ
- モノのグループに対して追加または削除されたモノ
- モノのグループに対して追加または削除されたモノのグループ

作成/更新/削除されたモノのグループ

レジストリは、モノのグループが作成、更新、または削除されると次のイベントメッセージを発行します。

- $aws/events/thingGroup/*groupName*/created
- $aws/events/thingGroup/*groupName*/updated
- $aws/events/thingGroup/*groupName*/deleted

updated ペイロードの例を次に示します。created と deleted メッセージのペイロードは類似しています。

```
{
  "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
}
```

ペイロードには次の属性が含まれます。

eventType

「THING_GROUP_EVENT」に設定します。

eventId

一意のイベント ID (文字列)。

timestamp

イベントが発生したときの UNIX タイムスタンプ。

オペレーション

イベントをトリガーしたオペレーション。有効な値は次のとおりです。

- CREATED
- UPDATED
- 削除済み

accountId

AWS アカウント ID。

thingGroupId

作成、更新、または削除されているモノのグループの ID。

thingGroupName

作成、更新、または削除されているモノのグループの名前。

versionNumber

モノのグループのバージョン。この値は、モノのグループが作成されると 1 に設定されます。これは、モノのグループが更新されるたびに 1 ずつ増加します。

parentGroupName

親モノグループの名前 (存在する場合)。

parentGroupId

親モノグループの ID (存在する場合)。

description

モノのグループの説明。

rootToParentThingGroups

親モノのグループについての情報の配列。モノの親グループごとに 1 つの要素があり、モノのルートグループから始まり、そのモノのグループの親まで続きます。各エントリには、モノのグループの `groupArn` と `groupId` が含まれています。

個の属性

モノのグループに関連付けられた名前と値のペアの集合。

モノのグループに対して追加または削除されたモノ

レジストリは、モノがモノのグループに対して追加または削除されると、次のイベントメッセージを発行します。

- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/added`
- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/removed`

メッセージには、次のペイロード例が含まれています。

```
{
```

```
    "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"
}
```

ペイロードには次の属性が含まれます。

eventType

「THING_GROUP_MEMBERSHIP_EVENT」に設定します。

eventId

イベントの ID。

timestamp

イベントが発生したときの UNIX タイムスタンプ。

オペレーション

ADDED モノのグループにモノが追加されると送信されます。REMOVED モノのグループからモノが削除されると送信されます。

accountId

AWS アカウント ID。

groupArn

モノのグループの ARN。

groupId

グループの ID。

thingArn

モノのグループに追加または削除されたモノの ARN。

thingId

モノのグループに追加または削除されたモノの ID。

membershipId

モノとモノの間の関係を表す ID。この値は、モノをモノのグループに追加するときに生成されます。

モノのグループに対して追加または削除されたモノのグループ

レジストリは、モノのグループが別のモノのグループに対して追加または削除されると、次のイベントメッセージを発行します。

- $aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/added
- $aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/removed

メッセージには、次の例のペイロードが含まれています。

```
{
    "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"
}
```

ペイロードには次の属性が含まれます。

eventType

「THING_GROUP_HIERARCHY_EVENT」に設定します。

eventId

イベントの ID。

timestamp

イベントが発生したときの UNIX タイムスタンプ。

オペレーション

ADDED モノのグループにモノが追加されると送信されます。REMOVED モノのグループからモノが削除されると送信されます。

accountId

AWS アカウント ID。

thingGroupId

親モノのグループの ID

thingGroupName

親モノのグループの名前。

childGroupId

子モノのグループの ID

childGroupName

子モノのグループの名前。

ジョブイベント

AWS IoT ジョブサービスは、ジョブが保留中、完了、またはキャンセルされたとき、およびジョブの実行時にデバイスが成功または失敗を報告するときに、MQTT プロトコルの予約済みトピックに発行します。デバイスまたは管理および監視アプリケーションは、これらのトピックにサブスクライブする事によって、ジョブの状態を追跡することができます。

ジョブイベントを有効にする方法

AWS IoT Jobs サービスからのレスポンスメッセージはメッセージブローカーを通過せず、他のクライアントやルールによってサブスクライブすることはできません。ジョブアクティビティ関連のメッセージをサブスクライブするには、notify および notify-nextトピックを使用します。トピックの詳細については、ジョブのトピックを参照してください。

ジョブの更新を通知するには、 を使用するか AWS マネジメントコンソール、 API または CLI を使用して、これらのジョブイベントを有効にします。詳細については、「のイベントを有効にする AWS IoT」を参照してください。

ジョブイベントの仕組み

ジョブのキャンセルと削除には少し時間がかかることがあるため、リクエストの開始と終了を示す 2 つのメッセージが送信されます。例えば、キャンセルリクエストが開始されたら、$aws/events/job/jobID/cancellation_in_progress トピックにメッセージが送信されます。キャンセルリクエストが完了したら、$aws/events/job/jobID/canceled トピックにメッセージが送信されます。

ジョブ削除リクエストにも同様のプロセスが発生します。管理およびモニタリングアプリケーションは、これらのトピックに登録することによって、ジョブの状態を追跡することができます。MQTT トピックのパブリッシュとサブスクライブの詳細については、「the section called “デバイス通信プロトコル”」を参照してください。

Job イベントのタイプ

以下に、ジョブイベントの様々なタイプの例を示しています。

ジョブの完了/キャンセル/削除

AWS IoT ジョブサービスは、ジョブが完了、キャンセル、削除されたとき、またはキャンセルまたは削除が進行中のときに、MQTT トピックにメッセージを発行します。

- $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メッセージには、次の例のペイロードが含まれています：

```
{
  "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メッセージには、次の例のペイロードが含まれています：

```
{
  "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メッセージには、次の例のペイロードが含まれています：

```
{
      "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メッセージには、次の例のペイロードが含まれています：

```
{
      "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メッセージには、次の例のペイロードが含まれています：

```
{
      "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"
    }
```

ジョブ実行ターミナルステータス

AWS IoT Jobs サービスは、デバイスがジョブ実行を終了ステータスに更新すると、メッセージを発行します。

- $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

メッセージには、次の例のペイロードが含まれています。

```
{
  "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"
  }
}
```

ライフサイクルイベント

AWS IoT は、MQTT トピックでライフサイクルイベントを発行できます。これらのイベントはデフォルトで利用可能で、無効にすることはできません。

> **Note**
>
> ライフサイクルメッセージは順不同で送信される場合があります。重複したメッセージを受信することがあります。
> `thingName` は、クライアントが排他的なモノの機能を使用して接続している場合にのみ含まれます。

このトピックの内容

- 接続/切断イベント
- 接続試行失敗イベント
- サブスクライブ/サブスクライブ解除イベント

接続/切断イベント

> **Note**
>
> AWS IoT Device Management フリートインデックス作成を使用すると、モノの検索、集計クエリの実行、モノの Connect/Disconnect イベントに基づく動的グループの作成を行うことができます。詳細については、「フリートインデックス作成」を参照してください。

AWS IoT クライアントが接続または切断すると、 は次の MQTT トピックにメッセージを発行します。

- `$aws/events/presence/connected/`*`clientId`* – クライアントがメッセージブローカーに接続しました。

- `$aws/events/presence/disconnected/clientId` – クライアントがメッセージブローカーから切断されました。

以下に示しているのは、`$aws/events/presence/connected/clientId` トピックにパブリッシュされる接続/切断メッセージに含まれる JSON 要素のリストです。

clientId

接続/切断するクライアントの ID。

> **Note**
>
> # または + が含まれているクライアント ID はライフサイクルイベントを受信しません。

thingName

IoT モノの名前。`thingName` は、クライアントが排他的なモノの機能を使用して接続している場合にのみ含まれます。

clientInitiatedDisconnect

クライアントによって切断が開始された場合は True、それ以外の場合は、false を返します。接続解除メッセージのみで見つかります。

disconnectReason

クライアントが切断する理由。接続解除メッセージのみで見つかります。次の表には、有効な値と、接続が切断されたときにブローカーが Last Will and Testament (LWT) メッセージを送信するかどうかが含まれています。

切断の理由	説明	ブローカーは LWT メッセージを送信します
`AUTH_ERROR`	クライアントが認証に失敗したか、または認可が失敗しました。	あり [1]
`CLIENT_INITIATED_DISCONNECT`	クライアントが切断することを示します。クライアントは、MQTT `DISCONNECT` 制御パケットを送信するか、クライアントが WebSocket 接続を	いいえ

切断の理由	説明	ブローカーは LWT メッセージを送信します
	使用している場合は Close frame を送信することでこれを実行できます。	
CLIENT_ERROR	クライアントに何か問題があり、切断されました。例えば、同じ接続で 1 つ以上の MQTT CONNECT パケットを送信する場合、またはクライアントがペイロード制限を超えるペイロードでパブリッシュしようとした場合、クライアントは切断されます。	はい
CONNECTION_LOST	クライアント/サーバー接続が切断されます。これは、ネットワークのレイテンシーが長い間、またはインターネット接続が失われた場合に発生する可能性があります。	はい
DUPLICATE_CLIENTID	クライアントは、すでに使用されているクライアント ID を使用しています。この場合、すでに接続されているクライアントは、この切断理由により切断されます。	はい
FORBIDDEN_ACCESS	クライアントは接続できません。例えば、IP アドレスが拒否されたクライアントは接続に失敗します。	あり [1]
MQTT_KEEP_ALIVE_TIMEOUT	クライアントのキープアライブ時間の 1.5 倍の期間にクライアントとサーバー間の通信がない場合、クライアントは切断されます。	はい
SERVER_ERROR	予期しないサーバーの問題が発生したため、切断されました。	はい
SERVER_INITIATED_DISCONNECT	サーバーは、運用上の理由から、意図的にクライアントを切断します。	はい

切断の理由	説明	ブローカーは LWT メッセージを送信します
API_INITIATED_DISCONNECT	クライアントは DeleteConnection API を使用して切断されました。	あり[2]
THROTTLED	スロットリング制限を超えたため、クライアントは切断されます。	はい
WEBSOCKET_TTL_EXPIRATION	WebSocket が存続可能時間値よりも長く接続されているため、クライアントが切断されます。	はい
CUSTOMAUTH_TTL_EXPIRATION	クライアントは、カスタムオーソライザーの存続可能時間値よりも長く接続されているため、切断されます。	はい

[1]このエラーが表示される前にデバイスの接続がアクティブだった場合。

[2]Last Will and Testament (LWT) メッセージを防ぐには、DeleteConnection API のデフォルトの LWT 送信動作を上書きするように preventWillMessage=true を設定します。

eventType

イベントのタイプ。有効な値は connected または disconnected です。

ipAddress

接続しているクライアントの IP アドレス。これは、IPv4 形式または IPv6 形式にすることができます。接続メッセージでのみ見つかります。

principalIdentifier

認証に使用された認証情報。TLS 相互認証の場合、これは使用された証明書の ID です。その他の認証の場合、これは IAM 認証情報です。

sessionIdentifier

セッションの存続中 AWS IoT に存在する のグローバルに一意の識別子。

timestamp

イベントが発生したおおよその日時。

versionNumber

ライフサイクルイベントのバージョン番号。これは、各クライアント ID 接続ごとに単調に増加する長い整数値です。バージョン番号は、ライフサイクルイベントの順序を推測するために加入者が使用できます。

> **Note**
>
> クライアント接続の Connect メッセージと Disconnect メッセージのバージョン番号は同じです。
> バージョン番号は値をスキップする可能性があり、イベントごとに 1 ずつ増加するとは限りません。
> クライアントが約 1 時間接続されない場合、バージョン番号は 0 にリセットされます。
> 永続的なセッションでは、永続的なセッションに設定された有効期間 (TTL) よりも長い時間クライアントが切断された後、バージョン番号は 0 にリセットされます。

接続メッセージの構造は次のとおりです。

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "ipAddress": "192.0.2.0",
    "versionNumber": 0
}
```

切断メッセージの構造は次のとおりです。

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002340451,
    "eventType": "disconnected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
```

```
    "principalIdentifier": "12345678901234567890123456789012",
    "clientInitiatedDisconnect": true,
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
    "versionNumber": 0
}
```

クライアントの切断の処理

ベストプラクティスは、Last Will and Testament (LWT) メッセージを含むライフサイクルイベントに対して、常に待機状態を実装することです。切断メッセージが受信されると、コードは一定期間待機し、デバイスがオフラインのままであることを確認してからアクションを実行します。これを行う 1 つの方法は、SQS 遅延キューの使用です。クライアントが LWT またはライフサイクルイベントを受信したら、例えば 5 秒間メッセージをキューに追加できます。そのメッセージが使用可能になり、(Lambda または別のサービスによって) 処理されたら、さらにアクションを実行する前に、最初にデバイスがまだオフラインかどうか確認できます。

接続試行失敗イベント

AWS IoT は、クライアントが接続する権限がない場合、または最後の Will と Testament が設定されていて、クライアントがその最後の Will トピックに発行する権限がない場合に、次の MQTT トピックにメッセージを発行します。

```
$aws/events/presence/connect_failed/clientId
```

以下に、$aws/events/presence/connect_failed/*clientId* トピックに発行される接続許可メッセージに含まれる JSON 要素のリストを示しています。

clientId

接続を試みて失敗したクライアントのクライアント ID。

> **Note**
>
> # または + が含まれているクライアント ID はライフサイクルイベントを受信しません。

thingName

IoT モノの名前。thingName は、クライアントが排他的なモノの機能を使用して接続している場合にのみ含まれます。

timestamp

イベントが発生したおおよその日時。

eventType

イベントのタイプ。有効な値はconnect_failedです。

connectFailureReason

接続が失敗する理由。有効な値はAUTHORIZATION_FAILEDです。

principalIdentifier

認証に使用された認証情報。TLS 相互認証の場合、これは使用された証明書の ID です。その他の認証の場合、これは IAM 認証情報です。

sessionIdentifier

セッションの存続中 AWS IoT に存在する のグローバルに一意の識別子。

ipAddress

接続しているクライアントの IP アドレス。これは、IPv4 形式または IPv6 形式にすることができます。接続メッセージでのみ見つかります。

接続失敗メッセージの構造は次のとおりです。

```
{
    "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"
}
```

サブスクライブ/サブスクライブ解除イベント

AWS IoT クライアントが MQTT トピックをサブスクライブまたはサブスクライブ解除すると、 は次の MQTT トピックにメッセージを発行します。

```
$aws/events/subscriptions/subscribed/clientId
```

または

```
$aws/events/subscriptions/unsubscribed/clientId
```

ここで、clientId は、 AWS IoT メッセージブローカーに接続する MQTT クライアントの ID です。

このトピックにパブリッシュされたメッセージには、以下の構造があります。

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "subscribed" | "unsubscribed",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "topics" : ["foo/bar","device/data","dog/cat"]
}
```

以下に示しているのは、$aws/events/subscriptions/subscribed/*clientId* および $aws/events/subscriptions/unsubscribed/*clientId* トピックにパブリッシュされるサブスクライブ/サブスクライブ解除メッセージに含まれる JSON 要素のリストです。

clientId

サブスクライブまたはサブスクライブ解除するクライアントの ID。

> **Note**
>
> # または + が含まれているクライアント ID はライフサイクルイベントを受信しません。

thingName

IoT モノの名前。thingName は、クライアントが排他的なモノの機能を使用して接続している場合にのみ含まれます。

eventType

イベントのタイプ。有効な値は subscribed または unsubscribed です。

principalIdentifier

認証に使用された認証情報。TLS 相互認証の場合、これは使用された証明書の ID です。その他の認証の場合、これは IAM 認証情報です。

sessionIdentifier

セッションの存続中 AWS IoT に存在する のグローバルに一意の識別子。

timestamp

イベントが発生したおおよその日時。

トピック

クライアントがサブスクライブした MQTT トピックの配列。

> **Note**
>
> ライフサイクルメッセージは順不同で送信される場合があります。重複したメッセージを受信することがあります。

トラブルシューティング AWS IoT

このトピックの改善にご協力ください
より良いものにするために必要なことを教えてください

以下の情報は、 での一般的な問題のトラブルシューティングに役立ちます AWS IoT

タスク

- AWS IoT Core トラブルシューティングガイド
- AWS IoT Device Management トラブルシューティングガイド
- AWS IoT Device Advisor トラブルシューティングガイド
- AWS IoT エラー

AWS IoT Core トラブルシューティングガイド

このトピックの改善にご協力ください
より良いものにするために必要なことを教えてください

これは のトラブルシューティングセクションです AWS IoT Core。

トピック

- 接続関連の問題の診断
- ルール関連の問題の診断
- シャドウ関連の問題の診断
- Salesforce IoT 入力ストリームアクションの問題の診断
- ストリーム制限の診断
- デバイス群切断のトラブルシューティング

接続関連の問題の診断

> このトピックの改善にご協力ください
> より良いものにするために必要なことを教えてください

への接続を成功させるには、以下 AWS IoT が必要です。

- 有効な接続
- 有効かつアクティブな証明書
- 必要な接続とオペレーションを許可するポリシー

接続

正しいエンドポイントを見つけるにはどうすればよいですか?

- endpointAddress によって返される `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

  または

- domainName によって返される `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"`

正しい Server Name Indication (SNI) 値を見つけるにはどうすればよいですか?

正しい SNI 値は、describe-endpoint によって返される endpointAddress、または describe-domain-configuration コマンドによって返される domainName です。これは、前のステップのエンドポイントと同じアドレスです。デバイスを に接続する場合 AWS IoT Core、クライアントは Server Name Indication (SNI) 拡張機能を送信できます。これは必須ではありませんが、強くお勧めします。マルチアカウント登録、カスタムドメイン、VPC エンドポイントなどの機能を使用するには、SNI 拡張機能を使用する必要があります。詳細については、「 のトランスポートセキュリティ AWS IoT」を参照してください。

持続する接続性の問題を解決するにはどうすればよいですか?

AWS Device Advisor を使用してトラブルシューティングを行うことができます。Device Advisor の事前構築されたテストにより、デバイスソフトウェアを TLS、MQTT、AWS IoT デバイスシャドウ、および AWS IoT Jobs の使用に関するベストプラクティスに照らして検証できます。

ここに既存の Device Advisor コンテンツを参照してください。

認証

AWS IoT エンドポイントに接続するには、デバイスを認証する必要があります。認証X.509 クライアント証明書に を使用するデバイスの場合、証明書は に登録 AWS IoT され、アクティブである必要があります。

デバイスが AWS IoT エンドポイントを認証する方法

AWS IoT CA 証明書をクライアントの信頼ストアに追加します。AWS IoT Coreのサーバー認証のドキュメントを参照して、リンクから適切な CA 証明書をダウンロードします。

デバイスが に接続するときにチェックされるものは何ですか AWS IoT?

デバイスが AWS IoTに接続しようとすると、次のようになります。

1. AWS IoT は、有効な証明書と Server Name Indication (SNI) 値をチェックします。
2. AWS IoT は、使用された証明書がアカウント AWS IoT に登録され、アクティブ化されたことを確認します。
3. デバイスがメッセージのサブスクライブや発行など AWS IoT、 でアクションを実行しようとすると、接続に使用した証明書にアタッチされたポリシーをチェックして、デバイスがそのアクションを実行する権限があることを確認します。

正しく設定された証明書を検証するにはどうすればよいですか?

OpenSSL の `s_client` コマンドを使用して、 AWS IoT エンドポイントへの接続をテストします。

```
openssl s_client -connect custom_endpoint.iot.aws-region.amazonaws.com:8443 -
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

`openssl s_client` の使用の詳細については、OpenSSL s_client ドキュメントを参照してください。

証明書のステータスを確認するにはどうすればよいですか?

- 証明書を一覧表示する

  証明書 ID がわからない場合は、`aws iot list-certificates` コマンドを使用してすべての証明書のステータスを確認できます。

- 証明書の詳細を表示する

証明書の ID がわかっている場合、このコマンドは証明書に関するより詳細な情報を表示します。

```
aws iot describe-certificate --certificate-id "certificateId"
```

- AWS IoT コンソールで証明書を確認する

  AWS IoT コンソールの左側のメニューで、[Secure] (安全性) を選択し、[Certificates] (証明書) を選択します。

  接続に使用している証明書をリストから選択して、その詳細ページを開きます。

  証明書の詳細ページで、現在のステータスを確認できます。

  証明書のステータスは、詳細ページの右上にある [Actions] (アクション) メニューを使用して変更できます。

Authorization

AWS IoT リソースは を使用してAWS IoT Core ポリシー、これらのリソースがアクションを実行することを許可します。アクションを承認するには、指定された AWS IoT リソースに、そのアクションを実行するアクセス許可を付与するポリシードキュメントがアタッチされている必要があります。

ブローカーから PUBNACK または SUBNACK レスポンスを受信しました。何をすればよいですか?

呼び出しに使用する証明書にポリシーがアタッチされていることを確認します AWS IoT。すべてのパブリッシュ/サブスクライブオペレーションはデフォルトで拒否されます。

添付されたポリシーが、実行しようとしている[action](アクション)を承認していることを確認してください。

アタッチされたポリシーが、承認されたアクションを実行しようとしているリソースを承認していることを確認してください。

ログに AUTHORIZATION_FAILURE エントリがあります。

呼び出しに使用する証明書にポリシーがアタッチされていることを確認します AWS IoT。すべてのパブリッシュ/サブスクライブオペレーションはデフォルトで拒否されます。

添付されたポリシーが、実行しようとしている[action](アクション)を承認していることを確認してください。

アタッチされたポリシーが、承認されたアクションを実行しようとしているリソースを承認していることを確認してください。

ポリシーで承認される内容を確認するにはどうすればよいですか?

AWS IoT コンソールの左側のメニューで、[セキュリティ] を選択し、[証明書] を選択します。

接続に使用している証明書をリストから選択して、その詳細ページを開きます。

証明書の詳細ページで、現在のステータスを確認できます。

証明書の詳細ページの左側のメニューで、[Policies] (ポリシー) を選択して、証明書にアタッチされているポリシーを表示します。

目的のポリシーを選択して、その詳細ページを表示します。

ポリシーの詳細ページで、ポリシーの[Policy document] (ポリシードキュメント) を確認して、何が許可されているかを確認します。

ポリシードキュメントを変更するには、[Edit policy document] (ポリシードキュメントの編集) を選択します。

セキュリティと ID

AWS IoT カスタムドメイン設定のサーバー証明書を提供する場合、証明書には最大 4 つのドメイン名があります。

詳細については、「AWS IoT Core endpoints and quotas」を参照してください。

ルール関連の問題の診断

> このトピックの改善にご協力ください
> より良いものにするために必要なことを教えてください

このセクションでは、ルールに関する問題が発生した場合に確認する必要のあるいくつかの事項について説明します。

トラブルシューティングのための CloudWatch Logs の設定

ルール関連の問題をデバッグするのに最適な方法は、CloudWatch Logs を使用することです。CloudWatch Logs を有効にすると AWS IoT、トリガーされるルールとその成功または失敗を確認できます。また、WHERE 句の条件の一致について情報も得られます。詳細については、「CloudWatch Logs AWS IoT を使用したモニタリング」を参照してください。

最も一般的なルールの問題は権限付与です。ログを確認することで、ロールがリソースに対して AssumeRole を実行するための権限が付与されていないことがわかります。きめ細かな口グ記録で生成される口グの例を次に示します。

```
{
    "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)"
}
```

グローバルログ記録で生成される同様のログの例を次に示します。

```
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.
```

詳細については、「the section called “CloudWatch コンソールでの AWS IoT ログの表示”」を参照してください。

外部サービスの診断

外部サービスはエンドユーザーによって制御されます。ルールを実行する前に、ルールにリンクした外部サービスが設定されており、アプリケーションのために十分なスループットとキャパシティーユニットがあることを確認してください。

SQL 問題の診断

SQL クエリが想定するデータを返さない場合:

- ログでエラーメッセージを確認します。
- SQL 構文がメッセージ内の JSON ドキュメントと一致することを確認します。

  クエリで使用されているオブジェクト名とプロパティ名を、トピックのメッセージペイロードの JSON ドキュメントで使用されている名前とともに確認してください。SQL クエリでの JSON フォーマットの詳細については、「JSON 拡張」を参照してください。
- JSON オブジェクトまたはプロパティ名に予約文字または数字が含まれているかどうかを確認します。

  SQL クエリでの JSON オブジェクト参照の予約文字の詳細については、「JSON 拡張」を参照してください。

シャドウ関連の問題の診断

このトピックの改善にご協力ください

より良いものにするために必要なことを教えてください

シャドウの診断

問題	トラブルシューティングのガイドライン
デバイスのシャドウドキュメントが `Invalid JSON document` で拒否されます。	JSON についてよくわからない場合は、このガイドで示されている例を用途に合わせて変更します。詳細については、「シャドウドキュメントの例」を参照してください。
正しい JSON を送信したが、そのすべてまたは一部がデバイスのシャドウドキュメントに保存されません。	JSON 形式のガイドラインに従っていることを確認します。`desired` および `reported` セクションの JSON フィールドのみが保存されます。これらのセクション以外の JSON の内容は (形式が正しい場合でも) 無視されます。
デバイスのシャドウが許容サイズを超えているというエラーを受け取りました。	デバイスのシャドウは、8 KB のデータのみがサポートされています。JSON ドキュメント内のフィールド名を短くするか、より多くのモノを作成してさらにシャドウを作成します。デバイスに関連付けられるモノ/シャドウの数に制限はありません。唯一の要件は、各モノ名はアカウント内で一意でなければならないことです。
デバイスのシャドウを受け取ると、そのサイズが 8 KB を超えています。どのようにしてこの状況になるのでしょうか?	受信すると、 AWS IoT サービスはデバイスのシャドウにメタデータを追加します。サービスはそのレスポンスにメタデータを含めますが、このデータは 8 KB の制限にはカウントされません。デバイスのシャドウに送信された状態ドキュメント内の `desired` 状態および

問題	トラブルシューティングのガイドライン
	reported 状態のみが、この制限にカウントされます。
リクエストは正しくないバージョンのため拒否されました。どうすればよいですか?	GET オペレーションを実行して、最新の状態ドキュメントのバージョンに同期させます。MQTT の使用時は、./update/accepted トピックにサブスクライブすると、状態の変更について通知され、JSON ドキュメントの最新バージョンが返されます。
タイムスタンプが数秒ずれます。	個々のフィールドと JSON ドキュメント全体のタイムスタンプは、ドキュメントが AWS IoT サービスによって受信されたとき、または状態ドキュメントが ./update/accepted および ./update/delta メッセージに公開されたときに更新されます。メッセージはネットワークを介して表示でき、それにより、タイムスタンプが数秒ずれることがあります。
デバイスは対応する Shadow トピックにパブリッシュおよびサブスクライブできますが、HTTP REST API を介して Shadow ドキュメントを更新しようとすると、HTTP 403 が返されます。	IAM でこれらのトピックへのアクセスを許可するポリシーを作成し、使用する認証情報に対するアクション (UPDATE/GET/DELETE) を許可するポリシーを作成したことを確認してください。IAM ポリシーと証明書ポリシーは独立しています。
その他の問題。	Device Shadow サービスで CloudWatch Logs にエラーが記録されます。デバイスと設定の問題を特定するには、CloudWatch Logs を有効にし、ログを表示して、デバッグ情報を見つけます。

Salesforce IoT 入力ストリームアクションの問題の診断

このトピックの改善にご協力ください

より良いものにするために必要なことを教えてください

実行トレース

Salesforce のアクションの実行トレースを確認するにはどうすればよいですか。

「CloudWatch Logs AWS IoT を使用したモニタリング」セクションを参照してください。ログを有効にすると、Salesforce アクションの実行トレースを確認できます。

アクションの成功と失敗

メッセージが Salesforce IoT 入力ストリームに正常に送信されたことを確認するにはどうすればよいですか。

CloudWatch Logs で Salesforce アクションの実行によって生成されたログを表示します。が表示された場合`Action executed successfully`、 AWS IoT ルールエンジンは、メッセージがターゲットの入力ストリームに正常にプッシュされたことを Salesforce IoT から確認したことを意味します。

Salesforce IoT プラットフォームに問題が発生した場合は、Salesforce IoT のサポートにお問い合せください。

メッセージが Salesforce IoT 入力ストリームに正常に送信されていない場合にはどうすればよいですか。

CloudWatch Logs で Salesforce アクションの実行によって生成されたログを表示します。ログエントリに応じて次のアクションを試すことができます。

`Failed to locate the host`

アクションの `url` パラメータが正しく、Salesforce IoT 入力ストリームが存在することを確認してください。

`Received Internal Server Error from Salesforce`

再試行。問題が解決しない場合は、Salesforce IoT サポートにお問い合わせください。

`Received Bad Request Exception from Salesforce`

送信するペイロードにエラーがないかどうかを確認してください。

`Received Unsupported Media Type Exception from Salesforce`

Salesforce IoT は現在バイナリペイロードをサポートしていません。JSON ペイロードを送信していることを確認してください。

`Received Unauthorized Exception from Salesforce`

アクションの `token` パラメータが正しく、トークンがまだ有効であることを確認してください。

`Received Not Found Exception from Salesforce`

アクションの `url` パラメータが正しく、Salesforce IoT 入力ストリームが存在することを確認してください。

ここに記載されていないエラーが表示された場合は、 AWS IoT サポートにお問い合わせください。

ストリーム制限の診断

「 AWS アカウントのストリーミング制限を超えました」のトラブルシューティング

「`"Error: You have exceeded the limit for the number of streams in your AWS account."`」と表示された場合は、制限の引き上げをリクエストする代わりに、アカウント内の未使用ストリームをクリーンアップできます。

AWS CLI または SDK を使用して作成した未使用のストリームをクリーンアップするには:

```
aws iot delete-stream -stream-id value
```

詳細については、「delete-stream」を参照してください。

> **Note**
>
> `list-streams` コマンドを使用して、ストリーミング ID を見つけることができます。

デバイス群切断のトラブルシューティング

> このトピックの改善にご協力ください
>
> より良いものにするために必要なことを教えてください

AWS IoT デバイスフリートの切断は、複数の理由で発生する可能性があります。この記事では、切断理由の診断方法と、 AWS IoT サービスの定期的なメンテナンスまたはスロットリング制限による切断の処理方法について説明します。

切断理由を診断するには

CloudWatch の AWSIotLogsV2 ロググループを確認して、ログエントリの `disconnectReason` フィールドで切断理由を特定できます。

AWS IoTのライフサイクルイベント機能を使用して、切断理由を特定することもできます。ライフサイクルの切断イベント (`$aws/events/presence/disconnected/clientId`) にサブスクライブしている場合は、切断が発生した AWS IoT ときに から通知を受け取ります。通知の `disconnectReason` フィールドで切断理由を特定できます。

詳細については、CloudWatch AWS IoT ログエントリとライフサイクルイベント」を参照してください。

AWS IoT サービスメンテナンスによる切断のトラブルシューティングを行うには

AWS IoTのサービスメンテナンスによって発生した切断は、 AWS IoTのライフサイクルイベントとCloudWatch `SERVER_INITIATED_DISCONNECT`に記録されます。これらの切断を処理するには、クライアント側のセットアップを調整して、デバイスを自動的に AWS IoT プラットフォームに再接続できるようにします。

スロットリング制限による切断のトラブルシューティングを行うには

スロットリング制限による切断は、 AWS IoTのライフサイクルイベントと CloudWatch `THROTTLED` に記録されます。これらの切断に対処するために、デバイス数の増加に合わせてメッセージブローカーの制限緩和を要求することができます。

詳細については、AWS IoT Core メッセージブローカーを参照してください。

AWS IoT Device Management トラブルシューティングガイド

このトピックの改善にご協力ください
より良いものにするために必要なことを教えてください

これは のトラブルシューティングセクションです AWS IoT Device Management。

トピック

- AWS IoT ジョブのトラブルシューティング
- フリートインデックス作成のトラブルシューティング
- AWS IoT デバイス管理ソフトウェアパッケージカタログのトラブルシューティング

AWS IoT ジョブのトラブルシューティング

これは、 AWS IoT ジョブのトラブルシューティングセクションです。

AWS IoT ジョブエンドポイントを見つけるにはどうすればよいですか?

AWS IoT Jobs コントロールプレーンエンドポイントを見つけるにはどうすればよいですか?

AWS IoT ジョブは、HTTPS プロトコルを使用したコントロールプレーン API オペレーションをサポートします。HTTPS プロトコルを使用して、正しいコントロールプレーンエンドポイントに接続していることを確認します。

AWS リージョン固有のエンドポイントのリストについては、AWS IoT 「 Core - コントロールプレーンエンドポイント」を参照してください。

FIPS 準拠の AWS IoT Jobs コントロールプレーンエンドポイントのリストについては、「サービス別の FIPS エンドポイント」を参照してください。

> **Note**
>
> AWS IoT ジョブと は、同じ AWS リージョン固有のエンドポイント AWS IoT Core を共有します。

AWS IoT Jobs データプレーンエンドポイントを見つけるにはどうすればよいですか?

AWS IoT ジョブは、HTTPS プロトコルと MQTT プロトコルを使用したデータプレーン API オペレーションをサポートします。HTTPS または MQTT プロトコルを使用して、正しいデータプレーンエンドポイントに接続していることを確認します。

- HTTPS プロトコル
  - 以下の describe-endpoint CLI コマンドまたは `DescribeEndpoint` REST API を使用します。エンドポイントタイプには、`iot:Jobs` を使用します。

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

- MQTT プロトコル
  - 以下の describe-endpoint CLI コマンドまたは `DescribeEndpoint` REST API を使用します。エンドポイントタイプには、`iot:Data-ATS` を使用します。

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

FIPS 準拠の AWS IoT Jobs データプレーンエンドポイントのリストについては、「サービス別の FIPS エンドポイント」を参照してください。

AWS IoT ジョブのアクティビティをモニタリングし、メトリクスを提供するにはどうすればよいですか?

Amazon CloudWatch を使用した AWS IoT ジョブアクティビティのモニタリングは、進行中の AWS IoT ジョブオペレーションをリアルタイムで可視化し、 AWS IoT ルールを介して CloudWatch アラームでコストを制御するのに役立ちます。 AWS IoT Jobs アクティビティをモニタリングし、CloudWatch アラームを設定する前に、ログ記録を設定する必要があります。ログ記録の設定の詳細については、「AWS IoT ログ記録の設定」を参照してください。

Amazon CloudWatch および CloudWatch リソースを使用するためのアクセス許可を IAM ユーザーロール経由で設定する方法については、「Amazon CloudWatch 用 Identity and Access Management」を参照してください。

Amazon CloudWatch を使用して AWS IoT Jobs メトリクスとモニタリングをセットアップするにはどうすればよいですか?

AWS IoT ログ記録を設定するには、「Configure AWS IoT logging. AWS IoT logging set up can be done in the AWS マネジメントコンソール AWS CLI」、または「API. AWS IoT logging set up for specific thing groups」で説明されているステップに従います AWS CLI 。

AWS IoT ジョブメトリクスセクションには、 AWS IoT ジョブアクティビティのモニタリングに使用される AWS IoT ジョブメトリクスが含まれています。ここでは、 AWS マネジメントコンソール および でメトリクスを表示する方法について説明します AWS CLI。

さらに、CloudWatch アラームをセットアップして、詳細にモニタリングする特定のメトリクスを警告できます。アラームのセットアップのガイダンスについては、「Amazon CloudWatch アラームの使用」を参照してください。

デバイスフリートと単一デバイスのトラブルシューティング

ジョブ実行は **QUEUED** のステータスを無期限に維持します

ステータス状態が QUEUED のジョブ実行が IN_PROGRESS、FAILED、TIMED_OUT などの次の論理ステータス状態に移行しない場合、次のいずれかのシナリオが原因である可能性があります。

- [CloudWatch コンソール] にある CloudWatch ログでデバイスアクティビティを確認します。詳細については、CloudWatch Logs AWS IoT を使用したモニタリング」を参照してください。
- ジョブおよび後続のジョブ実行に関連付けられた IAM ロールには、その IAM ロールにアタッチされた IAM ポリシーのポリシーステートメントのいずれかに正しいアクセス許可がリストされていない可能性があります。`describe-job` API を使用して、そのジョブとその後のジョブ実行にリンクされた IAM ロールを特定し、IAM ポリシーで正しいアクセス許可を確認します。ポリシーアクセス許可ステートメントが更新されると、リソースで `AssumeRole` API コマンドを実行できるようになります。

モノまたはモノのグループに対してジョブ実行が作成されませんでした

ジョブがステータスを IN_PROGRESS に更新すると、ターゲットグループ内のすべてのデバイスへのジョブドキュメントのロールアウトが開始されます。このステータス状態の更新により、ターゲットデバイスごとにジョブ実行が作成されます。ターゲットデバイスの 1 つに対してジョブ実行が作成されていない場合は、次のガイダンスを参照してください。

- thing はジョブによって直接ターゲットにされ、ジョブのステータス状態は IN_PROGRESS で、ジョブは同時ですか？ 3 つの条件がすべて満たされている場合、ジョブはターゲットグ

ループ内のすべてのデバイスにジョブ実行を送信し、その特定の thing はまだジョブ実行を受信していません。

- AWS マネジメントコンソールのジョブとジョブステータスの状態について、ターゲットグループ内のデバイスを確認するか、 describe-job API コマンドを使用します。
- describe-job API コマンドを使用して、ジョブの IsConcurrent プロパティが true または false に設定されているかどうかを確認します。詳細については、「Job limits」を参照してください。

- thing はジョブによって直接ターゲットにされていません。
  - Thing が ThingGroup に追加され、ジョブが ThingGroup をターゲットにしている場合は、Thing が ThingGroup の一部であることを確認します。
  - ジョブがステータス状態が IN_PROGRESS のスナップショットジョブで、同時である場合、ジョブはターゲットグループ内のすべてのデバイスにジョブ実行を送信し、その特定の Thing はまだジョブ実行を受信していません。
  - ジョブがステータス状態が IN_PROGRESS の連続ジョブで、同時である場合、ジョブはターゲットグループ内のすべてのデバイスにジョブ実行を送信し、その特定の Thing はまだジョブ実行を受信していません。連続ジョブの場合のみ、ThingGroup から Thing を削除し、Thing を ThingGroup に追加することもできます。
  - ジョブがステータスが のスナップショットジョブで、同時IN_PROGRESSでない場合、Thingまたは ThingGroupメンバーシップ関係は AWS IoT ジョブによって確認されない可能性があります。Job を作成する前に、AddThingToThingGroup 呼び出し後に数秒の待機時間を追加することをお勧めします。または、ターゲット選択を Continuous に切り替えることで、遅延 Thing および ThingGroup メンバーシップアタッチメントイベントをサービスバックフィルできます。

LimitedExceededException エラーにより新しいジョブが失敗する

ジョブの作成が LimitedExceededException のエラーレスポンスで失敗した場合は、 list-jobs API を呼び出し、isConcurrent=true ですべてのジョブを確認して、ジョブの同時実行数の上限に達しているかどうかを確認します。同時ジョブの詳細については、「ジョブの制限」を参照してください。ジョブの同時実行制限を表示し、制限の引き上げをリクエストするには、「AWS IoT Device Management ジョブの制限とクォータ」を参照してください。

ジョブドキュメントのサイズの制限

ジョブドキュメントのサイズは MQTT ペイロードサイズによって制限されます。32 kB (キロバイト) 、32,000 B (バイト) を超えるジョブドキュメントが必要な場合は、そのジョブドキュメントを作成して Amazon S3 に保存し、`CreateJob` API の `documentSource` フィールドまたは AWS CLIを使用して Amazon S3 オブジェクト URL を追加します。では AWS マネジメントコンソール、ジョブの作成時に Amazon S3 URL テキストボックスに Amazon S3 オブジェクト URL を追加します。

- AWS マネジメントコンソール ジョブドキュメントの作成: を使用してジョブを作成および管理します。 AWS マネジメントコンソール
- AWS CLI ジョブドキュメントの作成: を使用したジョブの作成と管理 AWS CLI
- `CreateJob` API ドキュメント: CreateJob

Device Side MQTT メッセージリクエストのスロットル制限

エラーコード 400 `ThrottlingException` を受け取った場合、デバイス側の同時リクエストの制限に達したため、デバイス側の MQTT メッセージは失敗しました。スロットル制限と調整可能かどうかの詳細については、「AWS IoT Device Management jobs limits and quotas」を参照してください。

接続タイムアウトエラー

エラーコード 400 `RequestExpired` は、高いレイテンシーまたは低いクライアント側のタイムアウト値による接続障害を示します。

- クライアント側とサーバー側の接続のテストについては、「デバイスデータエンドポイントとの接続をテストする」を参照してください。

無効な API コマンド

API コマンドが無効であることを示すエラーメッセージが表示されないように、正しい API コマンドが入力されていることを確認します。すべての AWS IoT API コマンドの包括的なリストについては、「AWS IoT API リファレンス」を参照してください。

サービス側の接続エラー

エラーコード 503 `ServiceUnavailable` は、サーバー側から発生したエラーを示します。

- AWS Health Dashboard すべての AWS サービスの現在のステータスについては、「 (すべてのサービス)」を参照してください。 AWS
- 個人の現在のステータスについては、 AWS Health Dashboard (個人 AWS アカウント) を参照してください AWS アカウント。

フリートインデックス作成のトラブルシューティング

フリートインデックス作成サービスの集計クエリのトラブルシューティング

タイプの不一致エラーがある場合は、CloudWatch Logs を使用して問題のトラブルシューティングを行うことができます。フリートインデックス作成サービスによってログが書き込まれる前に、CloudWatch Logs を有効にする必要があります。詳細については、「CloudWatch Logs AWS IoT を使用したモニタリング」を参照してください。

非管理対象フィールドに対して集計クエリを行うには、`customFields` または `UpdateIndexingConfiguration` に渡される update-indexing-configuration 引数で定義したフィールドを指定する必要があります。フィールド値が設定されたフィールドのデータ型と一致しない場合、集計クエリの実行時にこの値は無視されます。

型が一致しないためにフィールドをインデックス化できない場合、フリートインデックス作成サービスは、CloudWatch Logs にエラーログを送信します。エラーログには、フィールド名、変換できなかった値、デバイスのモノ名が含まれます。以下に、エラーログの例を示します。

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

デバイスが約 1 時間切断されていた場合、接続ステータス値 `timestamp` が含まれていない場合があります。永続的なセッションでは、この値は、永続的なセッションに設定された有効期間 (TTL) よりも長い時間クライアントが切断されると表示されません。接続ステータスのデータのインデックスは、クライアント ID に一致するモノの名前が含まれる接続にのみ作成されます。(クライアント ID は、デバイスの接続に使用される値です AWS IoT Core。)

フリートインデックス作成のトラブルシューティング

フリートインデックス作成の設定をダウングレードできない

フリートメトリクスまたは動的グループに関連付けられているデータソースを削除する場合、フリートインデックス作成の設定のダウングレードはサポートされていません。

例えば、インデックス作成の設定にレジストリデータ、シャドウデータ、接続データがあり、クエリ `thingName:TempSensor* AND shadow.desired.temperature>80` にフリートメトリクスが存在する場合、レジストリデータのみを含むようにインデックス作成の設定を更新するとエラーが発生します。

既存のフリートメトリクスが使用するカスタムフィールドの変更はサポートされていません。

フリートメトリクスまたは動的グループに互換性がないため、インデックス作成の設定を更新できません

互換性のないフリートメトリクスまたは動的グループが原因でインデックス作成の設定を更新できない場合は、インデックス作成の設定を更新する前に、互換性のないフリートメトリクスまたは動的グループを削除してください。

位置インデックス作成とジオクエリのトラブルシューティング

位置インデックス作成とジオクエリの不一致タイプエラーをトラブルシューティングするには、CloudWatch ログを有効にします。CloudWatch AWS IoT を使用してモニタリングする方法の詳細については、step-by-stepガイド」を参照してください。

ジオクエリを使用して位置データのインデックス作成をする場合、`geoLocations` で指定する位置フィールドは、`UpdateIndexingConfiguration` に渡す位置フィールドと一致する必要がありま

す。不一致がある場合、フリートのインデックス作成は不一致タイプエラーを CloudWatch に送信します。エラーログには、フィールド名、変換できなかった値、デバイスのモノ名が含まれます。

以下に、エラーログの例を示します。

```
{
"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."
}
```

詳細については、「位置データのインデックス作成」を参照してください。

フリートメトリクスのトラブルシューティング

CloudWatch でデータポイントが表示されない

フリートメトリックスを作成できるが、CloudWatch でデータポイントが表示されない場合は、クエリ文字列条件を満たすものがない可能性があります。

フリートメトリクスの作成方法を示す以下のサンプルコマンドを参照してください。

```
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
```

クエリ文字列条件を満たすモノがない場合`--query-string "thingName:TempSensor* AND attributes.temperature>80"`:

- values=countを使用すると、フリートメトリックスを作成でき、CloudWatch に表示するデータポイントが得られます。値countのデータポイントは常に 0 です。
- count 以外の values を使用すると、フリートメトリクスを作成することはできますが、CloudWatch にはフリートメトリクスが表示されず、CloudWatch に表示するデータポイントも得られません。

AWS IoT デバイス管理ソフトウェアパッケージカタログのトラブルシューティング

これは、 AWS IoT Device Management Software Package Catalog のトラブルシューティングセクションです。

エラーメッセージの一般的なトラブルシューティング

このセクションでは、ソフトウェアパッケージのバージョンライフサイクル全体で見られる一般的なエラーを一覧表示します。

HeadBucket エラー

HeadBucket API オペレーションまたは head-bucket CLI コマンドを呼び出して、ジョブのデプロイ中にファイルのアップロードに使用される Amazon S3 バケットを検証すると、次のエラーメッセージが表示されます。

ジョブのデプロイ中に Amazon S3 バケットを使用してファイルをアップロードする方法の詳細については、「ファイルアップロード用の署名付き URL」を参照してください。

```
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

無効な引数が指定されると、次のエラーメッセージが表示され、Amazon S3 `GetObject` API オペレーションが失敗します。

```
InvalidRequestException
    "Provided argument for presigned url is invalid"
```

Amazon S3 バージョン ID サポート

バージョニングコントロールを使用して Amazon S3 バケットへのアクセスをリクエストする場合、`versionId` を含めないと、以下のエラーが表示されることがあります。

Amazon S3 バケットでのバージョニングコントロールの使用に関する詳細については、「Amazon S3 バケットでのバージョニングの使用」を参照してください。

```
InvalidRequestException
    "VersionId not found when attempting to access s3 url"
```

ファイルアップロード用の署名付き URL 内のプレースホルダー

次のエラーメッセージは、ジョブのデプロイ中に送信先の Amazon S3 バケットにファイルをアップロードするために使用される署名付き URL 内のプレースホルダーに問題があると表示されます。ジョブのデプロイ中に Amazon S3 バケットを使用してファイルをアップロードする方法と、ローカルプレースホルダーの詳細については、「ファイルアップロード用の署名付き URL」を参照してください。

ローカルプレースホルダーが認識されない場合、以下のエラーメッセージが表示されます。

```
InvalidJobDocumentException
    "Undefined placeholder, ${...}, inside of presign url upload parameter"
```

ファイルのアップロードではなく、署名付き URL でローカルプレースホルダーを使用しようとすると、以下のエラーメッセージが表示されます。

```
InvalidJobDocumentException
```

```
    "Local placeholder, ${...}, is only valid inside of presign url upload"
```

Amazon S3 URL のネストが誤っている

Amazon S3 URL が別のプレースホルダー内に誤ってネストされると、次のエラーメッセージが表示されます。

```
InvalidJobDocumentException
    "${aws:%s[...]} should not be the second layer pattern."
```

パッケージバージョンのアーティファクトネスト

パッケージバージョンのアーティファクト署名付き URL が別のプレースホルダー内に誤ってネストされると、次のエラーメッセージが表示されます。

```
InvalidJobDocumentException
    "${aws:iot:package:[...]:artifact:s3-presigned-url} cannot be nested inside another
 placeholder."
```

パッケージバージョンのアーティファクトがない

参照されるパッケージバージョンアーティファクトが見つからない場合、次のエラーメッセージが表示されます。

```
InvalidJobDocumentException
    "Package %s version %s does not have an associated artifact to generate an S3
 presigned url."
```

ソフトウェアパッケージとパッケージバージョンのプレースホルダー

次のエラーメッセージは、destinationPackageVersions パラメータまたはパッケージバージョンの詳細ページの [バージョン ARN] で参照されている複数のソフトウェアパッケージとパッケージバージョンが原因で、ソフトウェアパッケージとパッケージバージョンのジョブドキュメントプレースホルダーがジョブデプロイに必要な有効な値に解決できない場合に表示されます。

```
InvalidJobDocumentException
    "Cannot resolve empty package name and version name given multiple elements in
 destination package versions."
```

空のソフトウェアパッケージとパッケージバージョンの使用

次のエラーメッセージは、空のパッケージまたはパッケージバージョンを、ジョブドキュメントのもう 1 つのバージョンを使用せずに使用しようとすると表示されます。

```
InvalidJobDocumentException
    "Empty package name and version name have to be used in pair."
```

ジョブドキュメントでの Null の使用

ジョブドキュメントでパッケージバージョンとして $null を指定しようとすると、次のエラーメッセージが表示されます。$null は CreateJob API オペレーションを使用するときに destinationPackageVersions パラメータ内でのみ使用できます。

```
InvalidJobDocumentException
    "$null is not allowed to be referenced as a package version in job documents."
```

パッケージバージョンのすべての属性

パッケージバージョンのすべての属性を使用し、追加のテキストまたはプレースホルダーで囲もうとすると、次のエラーメッセージが表示されます。

ソフトウェアパッケージバージョンですべての属性を使用する方法の詳細については、「AWS IoT ジョブの置換パラメータ」を参照してください。

```
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."
```

ファイルアップロード用の署名付き URL のローカルプレースホルダーの制限

ジョブのデプロイ中にファイルアップロード用に署名付き URL で使用されるローカルプレースホルダーの数の制限を超えると、次のエラーメッセージが表示されます。

ジョブのデプロイ中にファイルのアップロードに署名付き URL を使用する方法の詳細については、「ファイルアップロード用の署名付き URL」を参照してください。

```
InvalidJobDocumentException
    "The occurrence of local placeholder %s within S3 presigned url upload placeholder
 exceeds limit of %d."
```

Amazon S3 バケットのローカルプレースホルダー

ジョブのデプロイ中にファイルのアップロードに使用される署名付き URL プレースホルダーの Amazon S3 バケット名にローカルプレースホルダー URL を配置しようとすると、次のエラーメッセージが表示されます。

ジョブのデプロイ中にファイルのアップロードに署名付き URL を使用する方法の詳細については、「ファイルアップロード用の署名付き URL」を参照してください。

```
InvalidJobDocumentException
    "S3 bucket name in presigned url upload is not allowed to contain any placeholders"
```

開括弧および閉括弧

閉括弧「}」なしでジョブドキュメントにパラメータまたはプレースホルダーを追加すると、次のエラーメッセージが表示されます。

```
InvalidJobDocumentException
    "One or more parameters or placeholders are not terminated."
```

Amazon S3 署名付き URL を使用した IAM ロール

IAM ロールのないジョブドキュメントで Amazon S3 署名付き URL を使用しようとすると、次のエラーメッセージが表示されます。

Amazon S3 署名付き URL の詳細については、「署名付き URL の使用」を参照してください。

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document."
```

パッケージバージョンアーティファクトの Amazon S3 署名付き URL を持つ IAM ロール

IAM ロールのないジョブドキュメントでパッケージバージョンアーティファクトを表す Amazon S3 署名付き URL を使用しようとすると、次のエラーメッセージが表示されます。

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document for package %s version %s artifact."
```

ソフトウェア部品表エラーメッセージ

このセクションでは、パッケージバージョンにリンクされたソフトウェア部品表 (SBOM) に関連する一般的なエラーを一覧表示します。

SBOM 関連付けリクエストの入力検証

AssociateSbomWithPackageVersion API オペレーションを使用し、s3Location パラメータが null の場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "Associate request needs to include SBOM reference"
```

AssociateSbomWithPackageVersion API オペレーションの詳細については、AssociateSbomWithPackageVersion」を参照してください。

SBOM 検証エラー

このセクションでは、ソフトウェアパッケージバージョンに関連付けられている場合に、ソフトウェア部品表 (SBOM) の初期検証中に見られる一般的なエラーを一覧表示します。

AssociateSbomWithPackageVersion API オペレーションを使用し、s3Location パラメータの bucket が null の場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 bucket name for SBOM cannot be null"
```

AssociateSbomWithPackageVersion API オペレーションの s3Location パラメータの bucket の文字列が長すぎると、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 bucket name for SBOM is illegal. String length exceeds
 limit"
```

key パラメータが null の場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 key name for SBOM cannot be null"
```

AssociateSbomWithPackageVersion API オペレーションの s3Location パラメータの key の文字列が長すぎると、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 key name for SBOM is illegal. String length exceeds limit"
```

AssociateSbomWithPackageVersion API オペレーションの s3Location パラメータの version の文字列が null の場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 object version for SBOM cannot be null"
```

AssociateSbomWithPackageVersion API オペレーションの s3Location パラメータの version の文字列が長すぎると、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 object version for SBOM is illegal. String length exceeds
 limit"
```

Amazon S3 バケットに保存されている SBOM zip アーカイブファイルのサイズが大きすぎると、次のエラーメッセージが表示されます。

```
InvalidRequestException "S3 object file size exceeds limit"
```

AssociateSbomWithPackageVersion API オペレーションを使用し、進行中の SBOM 検証の現在の数が既に上限に達している場合、次のエラーメッセージが表示されます。

```
LimitExceededException "Too many ongoing SBOM validation workflows. Please wait and
 retry"
```

Amazon S3 バケットの SBOM ファイルの問題にアクセスする

Amazon S3 バケットが存在しないか、Amazon S3 バケットへのアクセスに適切なアクセス許可が付与されていないために、別のエンティティが Amazon S3 バケットにアクセスできない場合、次のエラーメッセージが表示されます。

Amazon S3 バケットにアクセスするために必要なアクセス許可ポリシーの詳細については、「ソフトウェア部品表のストレージ」を参照してください。

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket exists and S3 permission is granted."
```

Amazon S3 バケットが存在しないか、Amazon S3 バケットに保存されているコンテンツにアクセスするための適切なアクセス許可が付与されていないために、別のエンティティが key パラメータの SBOM zip アーカイブファイルにアクセスできない場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the key
 exists and S3 permission is granted."
```

バケット、キー、バージョン ID が存在しないか、Amazon S3 バケットへのアクセスに適切なアクセス許可が付与されていないために、別のエンティティが Amazon S3 バケットにアクセスできない場合、次のエラーメッセージが表示されます。さらに、付与されたアクセス許可が Amazon S3 バケットの SBOM zip アーカイブファイルへのアクセスに不十分な場合、このエラーメッセージが表示されることがあります。

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket/key/version exists and S3 permission is granted."
```

バケットが別のリージョンにあるため、別のエンティティが Amazon S3 バケットにアクセスできない場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "Cross-region S3 bucket for %s is not supported."
```

`AssociateSbomWithPackageVersion` API オペレーションの使用時に `bucket`、`key`、または `version` パラメータのスペルが正しくないため、別のエンティティが Amazon S3 バケットにアクセスできない場合、次のエラーメッセージが表示されます。

```
InvalidRequestException "Please make sure SBOM S3 bucket name/key length/version is
 valid"
```

AWS IoT Device Advisor トラブルシューティングガイド

> このトピックの改善にご協力ください
>
> より良いものにするために必要なことを教えてください

General

Q: 複数のテストスイートを並行して実行できますか?

A: はい。Device Advisor で、デバイスレベルのエンドポイントを使用して異なるデバイスでの複数のテストスイートの実行がサポートされるようになりました。アカウントレベルのエンドポイ

ントを使用する場合は、各アカウントで使用できるの Device Advisor エンドポイントは 1 つなので、一度に実行できるスイートは 1 つです。詳細については、「デバイスを設定する」を参照してください。

Q: デバイスから、TLS 接続が Device Advisor によって拒否されていたことがわかりました。これは想定どおりですか?

A: はい。Device Advisor は、各テストの実行前と実行後に TLS 接続を拒否します。Device Advisor で完全に自動化されたテストを行うために、デバイスの再試行メカニズムを実装することをお勧めします。複数のテストケース (TLS 接続、MQTT 接続、および MQTT 発行など) を使用してテストスイートを実行する場合、デバイス向けのメカニズムを構築することをお勧めします。このメカニズムでは、5 秒ごとにテストエンドポイントに 1 分から 2 分間の接続を試行できます。この方法を使用して、自動化された方法で順番に複数のテストケースを実行できます。

Q: セキュリティ分析および運用に関するトラブルシューティングを行うために、アカウントで実行された Device Advisor API 呼び出しの履歴を取得できますか?

A: はい。アカウントで行われた Device Advisor API コールの履歴を受信するには、 AWS IoT マネジメントコンソールで CloudTrail を有効にし、イベントソースを にフィルタリングしますiotdeviceadvisor.amazonaws.com。

Q: CloudWatch で Device Advisor ログを表示するにはどうすればよいですか?

A: 必要なポリシー (CloudWatchFullAccess など) をサービスロールに追加すると、テスト スイートの実行中に生成されたログが CloudWatch にアップロードされます (「設定」 を参照してください)。テストスイートに 1 つ以上のテストケースがある場合、2 つのログストリームを含むロググループ「aws/iot/deviceadvisor/$testSuiteId」が作成されます。1 つのストリーミングは「$testRunId」という名前で、セットアップやクリーンアップ手順など、テストスイートでテストケースを実行する前と後に実行されたアクションのログが含まれます。もう 1 つのログストリーミングは、テストスイートの実行に固有の「$suiteRunId_$testRunId」です。デバイスから送信されたイベントと AWS IoT Core は、このログストリームに記録されます。

Q: デバイスのアクセス許可ロールの目的は何ですか?

A: Device Advisor はテストデバイスと の間にあり、テストシナリオ AWS IoT Core をシミュレートします。テストデバイスからの接続とメッセージを受け入れ、デバイスのアクセス許可ロールを引き受けてユーザーに代わり接続を開始することで、それらを AWS IoT Core に転送します。デバイスロールのアクセス許可が、テストの実行に使用する証明書のアクセス許可と同じであることを確認することが重要です。Device Advisor がデバイスアクセス許可ロールを使用して AWS IoT Core ユーザーに代わって への接続を開始する場合、 AWS IoT 証明書ポリシーは適用されません。ただし、設定したデバイスのアクセス許可ロールの権限は適用されます。

Q: Device Advisor はどのリージョンでサポートされていますか?

A: Device Advisor は、us-east-1、us-west-2、ap-northeast-1、eu-west-1 リージョンでサポートされています。

Q: 一貫性のない結果が表示される理由は何ですか?

A: 一貫性のない結果の主な原因の1つは、テストの EXECUTION_TIMEOUT を低すぎる値に設定していることです。EXECUTION_TIMEOUT の推奨値およびデフォルト値の詳細については、Device Advisor テストケースを参照してください。

Q: Device Advisor はどの MQTT プロトコルをサポートしていますか?

A: Device Advisor は、X509 クライアント証明書による MQTT バージョン 3.1.1 をサポートしています。

Q: デバイスをテストエンドポイントに接続しようとした場合でもテストケースが実行タイムアウトメッセージで失敗した場合はどうなりますか?

A: 「デバイスロールとして使用する IAM ロールを作成する」のすべてのステップを検証します。それでもテストが失敗した場合、Device Advisor が機能するために必要な正しいサーバーネームインディケーション (SNI) 拡張子がデバイスから送信されていない可能性があります。正しい SNI 値は、「デバイスの設定」セクションに従って返されるエンドポイントアドレスです。AWS IoT また、 では、デバイスが Server Name Indication (SNI) 拡張機能を Transport Layer Security (TLS) プロトコルに送信する必要があります。詳細については、「Transport security in AWS IoT」を参照してください。

Q: MQTT 接続が「libaws-c-mqtt: AWS_ERROR_MQTT_UNEXPECTED_HANGUP」エラーで失敗します。(または) デバイスの MQTT 接続が Device Advisor のエンドポイントから自動的に切断されます。このエラーはどのように解決できますか?

A: この特定のエラーコードと予期しない切断は、さまざまな原因で発生する可能性がありますが、ほとんどの場合、デバイスに関連付けられているデバイスのロールに関連している可能性があります。以下のチェックポイント (優先度順) でこの問題を解決します。

- デバイスにアタッチされたデバイスロールには、テストを実行するために必要な最低限の IAM 権限が必要です。Device Advisor は、接続されたデバイスのロールを使用して、テストデバイスに代わって AWS IoT MQTT アクションを実行します。必要な権限がない場合、AWS_ERROR_MQTT_UNEXPECTED_HANGUP エラーが表示されるか、デバイスが Device Advisor のエンドポイントに接続しようとするときに予期しない切断が発生します。例えば、MQTT Publish テストケースの実行を選択した場合、Connect アクションと Publish アクションの両方を、対応する ClientId と Topic とともにロールに含める必要があります (コンマ

で区切ることで複数の値を指定でき、ワイルドカード (*) 文字を使用してプレフィックス値を指定することもできます)。例えば、`TestTopic` で始まる任意のトピックで公開するためのアクセス許可を付与する場合は、リソース値として `TestTopic*` を指定できます。ポリシーの例を次に示します。

- リソースタイプのデバイスロールで定義されている値と、コードで使用される実際の値が一致しません。例: ロールで定義された ClientId とデバイスコードで使用されている実際の ClientId の不一致。ClientId、Topic、TopicFilter などの値は、デバイスロールとコードで同一である必要があります。
- デバイスに添付されているデバイス証明書は、アクティブであり、リソースに必要なアクション許可を持つポリシーが添付されている必要があります。デバイス証明書ポリシーは、 AWS IoT リソースおよび AWS IoT Core データプレーンオペレーションへのアクセスを許可または拒否することに注意してください。Device Advisor では、テストケース中に使用されるアクション権限を付与するアクティブなデバイス証明書をデバイスに添付する必要があります。

AWS IoT エラー

> このトピックの改善にご協力ください
>
> より良いものにするために必要なことを教えてください

このセクションでは、 によって送信されたエラーコードを一覧表示します AWS IoT。

メッセージブローカーに関するエラーコード

エラーコード	エラーの説明
400	Bad request。
401	Unauthorized。
403	Forbidden。
426	Upgrade required.
503	Service unavailable。

ID とセキュリティに関するエラーコード

エラーコード	エラーの説明
401	Unauthorized。

Device Shadow に関するエラーコード

エラーコード	エラーの説明
400	Bad request。
401	Unauthorized。
403	Forbidden。
404	見つかりません。
409	Conflict。
413	Request too large。
422	Failed to process request。
429	Too many requests。
500	Internal error。
503	Service unavailable。

AWS IoT Device SDK、Mobile SDK、および AWS IoT Device Client

このページでは、AWS IoT と選択したハードウェアプラットフォームを使用して革新的な IoT ソリューションを構築するのに役立つ、AWS IoT Device SDK、オープンソースライブラリ、デベロッパーガイド、サンプルアプリケーション、および移植ガイドの概要を説明します。

これらの SDK は、IoT デバイスで使用するためのものです。モバイルデバイスで使用する IoT アプリケーションを開発している場合は、「AWS Mobile SDK」を参照してください。IoT アプリケーションまたはサーバー側プログラムを開発している場合は、AWS SDKs を参照してください。

AWS IoTDevice SDKs

AWS IoT Device SDK には、オープンソースライブラリ、サンプル付きの開発者ガイド、ガイドが含まれているので、選択したプラットフォーム上で革新的な IoT 製品またはソリューションを構築できます。

> **Note**
>
> AWS IoT デバイス SDK は、MQTT 5 クライアントをリリースしました。AWS IoT Device SDK は、macOS での TLS 1.3 の使用をサポートしていません。

これらの SDK は、MQTT および WSS プロトコルを使用して、IoT デバイスを AWS IoT に接続するのに役立ちます。

C++

AWS IoT C++ Device SDK

AWS IoT C++ Device SDK を使用すると、デベロッパーは AWS および AWS IoT API を使用して接続されたアプリケーションを構築できます。具具体的には、この SDK にはリソース制約がなく、メッセージキュー、マルチスレッドサポート、最新の言語機能などの高度な機能が必要なデバイス向けに設計されています。詳細については、以下を参照してください:

- AWS IoT C++ Device SDK v2 (GitHub)
- AWS IoT C++ Device SDK v2 Readme
- AWS IoT C++ Device SDK v2 サンプル

- AWS IoTDevice SDK C++ v2 API ドキュメント

Python

AWS IoT Device SDK for Python

AWS IoTDevice SDK for Python では、開発者が デバイスを使用して MQTT または WebSocket プロトコル経由の MQTT で AWS IoTプラットフォームにアクセスできるように Python スクリプトを記述できます。そのデバイスを AWS IoT に接続することで、ユーザーは AWS IoT が提供するメッセージブローカー、ルール、Shadows、および、AWS Lambda、Kinesis、Amazon S3 などその他の AWS のサービスを安全に使用できます。

- GitHub の Python v2 用 AWS IoT Device SDK
- AWS IoT Device SDK for Python v2 Readme
- AWS IoT Device SDK for Python v2 サンプル
- AWS IoT Device SDK for Python v2 API ドキュメント

JavaScript

AWS IoT Device SDK for JavaScript

aws-iot-device-sdk.js パッケージでは、開発者が MQTT または WebSocket プロトコル経由の MQTT を使用して AWS IoT にアクセスする JavaScript アプリケーションを記述できます。これは、Node.js 環境およびブラウザーアプリケーションで使用できます。詳細については、以下を参照してください:

- GitHub の AWS IoT Device SDK for JavaScript v2
- AWS IoT Device SDK for JavaScript v2 Readme
- AWS IoT Device SDK for JavaScript v2 サンプル
- AWS IoT Device SDK for JavaScript v2 API ドキュメント

Java

AWS IoT Device SDK for Java

AWS IoT Device SDK for Java では、Java 開発者が MQTT または WebSocket プロトコル経由の MQTT で AWS IoT プラットフォームにアクセスできます SDK はシャドウをサポートするように構築されています。GET、UPDATE、DELETE を含む HTTP メソッドを使用して、 Shadows に

アクセスできます。SDK では、簡略化された Shadow アクセスモデルもサポートしていて、開発者が、JSON ドキュメントをシリアル化または逆シリアル化することなく、ゲッターメソッドとセッターメソッドを使用するだけで Shadows とデータを交換できます。

> **Note**
>
> AWS IoT Device SDK for Java v2 が Android 開発をサポートするようになりました。詳細については、「AWS IoT Device SDK for Android」を参照してください。

詳細については、以下を参照してください:

- GitHub の AWS IoT Device SDK for Java v2
- AWS IoT Device SDK for Java v2 Readme
- AWS IoT Device SDK for Java v2 サンプル
- AWS IoT Device SDK for Java v2 API ドキュメント

Swift

AWS IoT デバイス SDK for Swift

AWS IoT デバイス SDK for Swift では、Swift 開発者が、MQTT 5 プロトコルを使用して Linux、Apple macOS、iOS、tvOS プラットフォーム用の AWS IoT アプリケーションを作成できます。

詳細については、以下を参照してください:

- AWS IoT Device SDK for Swift (GitHub)
- AWS IoT Device SDK for Swift Readme
- AWS IoT Device SDK for Swift Samples

AWS IoT Device SDK for Embedded C

> **Note**
>
> この SDK は、経験豊富な組み込みソフトウェアデベロッパーによる使用を想定しています。

AWS IoT Device SDK for Embedded C (C-SDK) は、IoT デバイスを AWS IoT Core に安全に接続するために組み込みアプリケーションで使用できる、MIT オープンソースライセンスに基づく C ソースファイルのコレクションです。MQTT クライアント、JSON パーサー、および AWS IoT デバイスシャドウ、AWS IoT ジョブ、AWS IoT フリートプロビジョニング、AWS IoT Device Defender ライブラリが含まれています。この SDK はソース形式で配布され、アプリケーションコード、その他のライブラリ、および任意のオペレーティングシステム (OS) とともにお客様のファームウェアに組み込まれることが意図されています。

AWS IoT Device SDK for Embedded C は通常、最適化された C 言語ランタイムを必要とするリソース制約のあるデバイスを対象としています。この SDK は、任意のオペレーティングシステムで使用でき、任意のプロセッサタイプ (MCU や MPU など) でホストできます。

詳細については、以下を参照してください:

- GitHub の Embedded C 用 AWS IoT Device SDK
- Embedded C Readme用AWS IoTDevice SDK
- Embedded C サンプル用 AWS IoT Device SDK

AWS Mobile SDK

AWS Mobile SDK は、モバイルアプリケーションデベロッパーに、AWS IoT Core サービスの API、MQTT を使用した IoT デバイス通信、および他の AWS のサービスの API に対するプラットフォーム固有のサポートを提供します。

Android

AWS Mobile SDK for Android

AWS Mobile SDK for Android には、デベロッパーが AWS を使用して関連するモバイルアプリケーションを構築できるように、ライブラリ、サンプル、およびドキュメントが含まれています。この SDK には、MQTT デバイス通信と AWS IoT Core サービスの API 呼び出しのサポートも含まれています。詳細については、以下を参照してください:

- AWS Mobile SDK for AndroidGitHub の
- AWS Mobile SDK for AndroidReadme
- AWS Mobile SDK for Androidサンプル
- AWS Mobile SDK for Android API リファレンス

- AWSIoTClient Class リファレンスドキュメント

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS はオープンソースの開発キットで、Apache オープンソースライセンスのもとに配布されています。AWS Mobile SDK for iOS には、デベロッパーが AWS を使用して関連するモバイルアプリケーションを構築できるように、ライブラリ、コードサンプル、およびドキュメントが含まれています。この SDK には、MQTT デバイス通信と AWS IoT Core サービスの API 呼び出しのサポートも含まれています。詳細については、以下を参照してください:

- AWS Mobile SDK for iOSGitHub の
- AWS Mobile SDK for iOSReadme
- AWS Mobile SDK for iOSサンプル
- AWSIoT Class リファレンスドキュメントAWS Mobile SDK for iOS

AWS IoT Device Client

AWS IoT Device Client は、デバイスが AWS IoT に接続し、フリートプロビジョニングタスクを実行し、デバイスセキュリティポリシーをサポートし、セキュアトンネリングを使用して接続し、デバイスでジョブを処理するのに役立つコードを提供します。このソフトウェアをデバイスにインストールして、これらの日常的なデバイスタスクを処理できるため、特定のソリューションに集中できます。

> **Note**
>
> AWS IoTDevice Client は、x86_64 または ARM プロセッサと一般的な Linux オペレーティングシステムを備えたマイクロプロセッサベースの IoT デバイスで動作します。

C++

AWS IoT Device Client

C++ の AWS IoT Device Client の詳細については、以下を参照してください。

- AWS IoTC++ ソースコードの Device Client (GitHub)
- AWS IoTC++ Readme の Device Client

AWS IoT Device SDK の以前のバージョン

これらは、上記の新しいバージョンに置き換えられた AWS IoT Device SDK の以前のバージョンです。これらの SDK は、メンテナンスおよびセキュリティ更新プログラムのみを受信します。新しい機能を含むように更新されることはないため、新しいプロジェクトでは使用しないでください。

- AWS IoT C++ Device SDK (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 を使用するためのコード例

次のコード例は、 AWS Software Development Kit (SDK) AWS IoT で を使用する方法を示しています。

基本 は、重要なオペレーションをサービス内で実行する方法を示すコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードの例

- AWS SDKs AWS IoT を使用するための基本的な例
  - こんにち AWS IoTは
  - AWS SDK AWS IoT を使用した の基本について説明します。
  - AWS SDKs AWS IoT を使用するためのアクション
    - AWS SDK または CLI AttachThingPrincipalで を使用する
    - AWS SDK または CLI CreateKeysAndCertificateで を使用する
    - AWS SDK または CLI CreateThingで を使用する
    - AWS SDK または CLI CreateTopicRuleで を使用する
    - AWS SDK または CLI DeleteCertificateで を使用する
    - AWS SDK または CLI DeleteThingで を使用する
    - AWS SDK または CLI DeleteTopicRuleで を使用する
    - AWS SDK または CLI DescribeEndpointで を使用する
    - AWS SDK または CLI DescribeThingで を使用する
    - AWS SDK または CLI DetachThingPrincipalで を使用する
    - AWS SDK または CLI ListCertificatesで を使用する
    - AWS SDK または CLI ListThingsで を使用する
    - AWS SDK または CLI SearchIndexで を使用する
    - AWS SDK または CLI UpdateIndexingConfigurationで を使用する

- AWS SDK または CLI UpdateThingで を使用する

AWS SDKs AWS IoT を使用するための基本的な例

次のコード例は、 SDKs AWS IoT で AWS の基本を使用する方法を示しています。

例

- こんにち AWS IoTは
- AWS SDK AWS IoT を使用した の基本について説明します。
- AWS SDKs AWS IoT を使用するためのアクション
  - AWS SDK または CLI AttachThingPrincipalで を使用する
  - AWS SDK または CLI CreateKeysAndCertificateで を使用する
  - AWS SDK または CLI CreateThingで を使用する
  - AWS SDK または CLI CreateTopicRuleで を使用する
  - AWS SDK または CLI DeleteCertificateで を使用する
  - AWS SDK または CLI DeleteThingで を使用する
  - AWS SDK または CLI DeleteTopicRuleで を使用する
  - AWS SDK または CLI DescribeEndpointで を使用する
  - AWS SDK または CLI DescribeThingで を使用する
  - AWS SDK または CLI DetachThingPrincipalで を使用する
  - AWS SDK または CLI ListCertificatesで を使用する
  - AWS SDK または CLI ListThingsで を使用する
  - AWS SDK または CLI SearchIndexで を使用する
  - AWS SDK または CLI UpdateIndexingConfigurationで を使用する
  - AWS SDK または CLI UpdateThingで を使用する

こんにち AWS IoTは

次のコード例は、 AWS IoTの使用を開始する方法を示しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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 の詳細については、AWS SDK for .NET API リファレンスの「listThings」を参照してください。

C++

SDK for C++

CMakeLists.txt CMake ファイルのコード。

```
# 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 ソースファイルのコード。

```
#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 の詳細については、AWS SDK for C++ API リファレンスの「listThings」を参照してください。

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「listThings」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、[AWS SDK for Kotlin API リファレンス]の「listThings」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「listThings」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK AWS IoT を使用した の基本について説明します。

次のコード例は、以下を実行する方法を示しています。

- AWS IoT モノを作成します。
- デバイス証明書を生成します。
- 属性を使用して AWS IoT モノを更新します。
- 一意のエンドポイントを返します。
- AWS IoT 証明書を一覧表示します。
- AWS IoT シャドウを更新します。
- 状態情報を書き込みます。
- ルールを作成する。
- ルールを一覧表示します。

- モノの名前を使用してモノを検索します。
- AWS IoT モノを削除します。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

AWS IoT 機能を示すインタラクティブなシナリオを実行します。

```
/// <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 メソッドのラッパークラス。

```
/// <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**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

AWS IoT モノを作成します。

```
    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();
}
```

デバイス証明書を生成してアタッチします。

```
    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 モノに対してさまざまなオペレーションを実行します。

```
    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;
}
```

リソースをクリーンアップします。

```
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**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

AWS IoT 機能を示すインタラクティブなシナリオを実行します。

```
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 メソッドのラッパークラス。

```
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**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

オペレーションを管理する IoT ラッパークラスを作成します。

```
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 の基本を示すインタラクティブなシナリオを実行します。

```
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 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDKs AWS IoT を使用するためのアクション

次のコード例は、 AWS SDKs で個々の AWS IoT アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「AWS IoT API リファレンス」を参照してください。

例

- AWS SDK または CLI AttachThingPrincipalで を使用する
- AWS SDK または CLI CreateKeysAndCertificateで を使用する
- AWS SDK または CLI CreateThingで を使用する
- AWS SDK または CLI CreateTopicRuleで を使用する
- AWS SDK または CLI DeleteCertificateで を使用する
- AWS SDK または CLI DeleteThingで を使用する
- AWS SDK または CLI DeleteTopicRuleで を使用する
- AWS SDK または CLI DescribeEndpointで を使用する
- AWS SDK または CLI DescribeThingで を使用する
- AWS SDK または CLI DetachThingPrincipalで を使用する
- AWS SDK または CLI ListCertificatesで を使用する
- AWS SDK または CLI ListThingsで を使用する
- AWS SDK または CLI SearchIndexで を使用する
- AWS SDK または CLI UpdateIndexingConfigurationで を使用する
- AWS SDK または CLI UpdateThingで を使用する

AWS SDK または CLI **`AttachThingPrincipal`**で を使用する

次のサンプルコードは、`AttachThingPrincipal` を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「AttachThingPrincipal」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「AttachThingPrincipal」を参照してください。

CLI

AWS CLI

モノに証明書をアタッチするには

次の attach-thing-principal 例では、MyTemperatureSensor モノに証明書をアタッチします。証明書は ARN によって識別されます。証明書の ARN は、 AWS IoT コンソールで確認できます。

```
aws iot attach-thing-principal \
    --thing-name MyTemperatureSensor \
    --principal arn:aws:iot:us-
west-2:123456789012:cert/2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8
```

このコマンドでは何も出力されません。

詳細については、「AWS IoT ディベロッパーガイド」の「レジストリによるモノの管理方法」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「AttachThingPrincipal」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /**
    * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「AttachThingPrincipal」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「AttachThingPrincipal」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンス の「AttachThingPrincipal」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateKeysAndCertificate**で を使用する

次のサンプルコードは、CreateKeysAndCertificate を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「CreateKeysAndCertificate」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「CreateKeysAndCertificate」を参照してください。

CLI

AWS CLI

RSA キーペアを作成して X.509 証明書を発行するには

以下の `create-keys-and-certificate` は、2048 ビットの RSA キーペアを作成し、発行されたパブリックキーを使用して X.509 証明書を発行します。IoT がこの証明書のプライベートキー AWS IoT を提供するのは今回だけです。そのため、必ず安全な場所に保管してください。

```
aws iot create-keys-and-certificate \
    --certificate-pem-outfile "myTest.cert.pem" \
    --public-key-outfile "myTest.public.key" \
    --private-key-outfile "myTest.private.key"
```

出力:

```
{
    "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"
    }
}
```

詳細については、AWS 「 IoT デベロッパーガイド」の「IoT デバイス証明書の作成と登録」を参照してください。 AWS IoT

- API の詳細については、「AWS CLI コマンドリファレンス」の「CreateKeysAndCertificate」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
```

```
    * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateKeysAndCertificate」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateKeysAndCertificate」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンス の「CreateKeysAndCertificate」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateThing**で を使用する

次のサンプルコードは、CreateThing を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「CreateThing」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「CreateThing」を参照してください。

CLI

AWS CLI

例 1: レジストリにモノのレコードを作成するには

次のcreate-thing例では、 AWS IoT モノレジストリにデバイスのエントリを作成します。

```
aws iot create-thing \
    --thing-name SampleIoTThing
```

出力:

```
{
```

```
    "thingName": "SampleIoTThing",
    "thingArn": "arn:aws:iot:us-west-2: 123456789012:thing/SampleIoTThing",
    "thingId": " EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE "
}
```

例 2: モノのタイプに関連付けられているモノを定義するには

次の create-thing 例では、指定されたモノのタイプとその属性を持つモノを作成します。

```
aws iot create-thing \
    --thing-name "MyLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

出力:

```
{
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797"
}
```

詳細については、「AWS IoT ディベロッパーガイド」の「レジストリによるモノの管理方法」と「モノのタイプ」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「CreateThing」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
```

```
    * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateThing」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateThing」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスの「CreateThing」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateTopicRule**で を使用する

次のサンプルコードは、CreateTopicRule を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「CreateTopicRule」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「CreateTopicRule」を参照してください。

CLI

AWS CLI

Amazon SNS アラートを送信するルールを作成するには

次のcreate-topic-rule 例では、デバイスシャドウにある土壌湿度レベルの読み取り値が低くなったときに Amazon SNS メッセージを送信するルールを作成します。

```
aws iot create-topic-rule \
    --rule-name "LowMoistureRule" \
    --topic-rule-payload file://plant-rule.json
```

この例では、次の JSON コードを plant-rule.json という名前のファイルに保存する必要があります。

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

このコマンドでは何も出力されません。

詳細については、AWS 「 IoT デベロッパーガイド」の「Creating anIoT RuleAWS IoT」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「CreateTopicRule」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
     * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateTopicRule」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「CreateTopicRule」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンス の「CreateTopicRule」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteCertificate**で を使用する

次のサンプルコードは、DeleteCertificate を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「DeleteCertificate」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteCertificate」を参照してください。

CLI

AWS CLI

デバイス証明書を削除するには

次の delete-certificate 例は、指定した ID のデバイス証明書を削除します。

```
aws iot delete-certificate \
    --certificate-
id c0c57bbc8baaf4631a9a0345c957657f5e710473e3ddbee1428d216d54d53ac9
```

このコマンドでは何も出力されません。

詳細については、「AWS IoT API リファレンス」の「DeleteCertificate」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteCertificate」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /**
    * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteCertificate」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteCertificate」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteCertificate」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteThing**で を使用する

次のサンプルコードは、DeleteThing を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/// <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 の詳細については、AWS SDK for .NET API リファレンスの「DeleteThing」を参照してください。https://docs.aws.amazon.com/goto/DotNetSDKV4/iot-2015-05-28/DeleteThing

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteThing」を参照してください。

CLI

AWS CLI

モノに関する詳細情報を表示するには

次の のdelete-thing例では、 AWS アカウントの AWS IoT レジストリからモノを削除します。

aws iot delete-thing --thing-name "FourthBulb"

このコマンドでは何も出力されません。

詳細については、「AWS IoT ディベロッパーガイド」の「レジストリによるモノの管理方法」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteThing」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
     * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteThing」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「DeleteThing」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスのDeleteThing」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteTopicRule**で を使用する

次のサンプルコードは、DeleteTopicRule を使用する方法を説明しています。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteTopicRule」を参照してください。

CLI

AWS CLI

ルールを削除するには

次の delete-topic-rule の例は、指定されたルールを削除します。

```
aws iot delete-topic-rule \
    --rule-name "LowMoistureRule"
```

このコマンドでは何も出力されません。

詳細については、「AWS IoT デベロッパーガイド」の「ルールの削除」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DeleteTopicRule」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスのDeleteTopicRule」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeEndpoint**で を使用する

次のサンプルコードは、DescribeEndpoint を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「DescribeEndpoint」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、AWS SDK for C++ API リファレンスの「DescribeEndpoint」を参照してください。

CLI

AWS CLI

例 1: 現在の AWS エンドポイントを取得するには

次の のdescribe-endpoint例では、すべてのコマンドが適用されるデフォルトの AWS エンドポイントを取得します。

```
aws iot describe-endpoint
```

出力:

```
{
    "endpointAddress": "abc123defghijk.iot.us-west-2.amazonaws.com"
}
```

詳細については、「AWS IoT ディベロッパーガイド」の「DescribeEndpoint」を参照してください。

例 2: ATS エンドポイントを取得するには

次の describe-endpoint の例は、Amazon Trust Services (ATS) エンドポイントを取得します。

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

出力:

```
{
    "endpointAddress": "abc123defghijk-ats.iot.us-west-2.amazonaws.com"
```

```
}
```

詳細については、AWS 「 IoT デベロッパーガイド」の「X.509 証明書と AWS IoT」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DescribeEndpoint」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
     * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeEndpoint」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「DescribeEndpoint」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスの「DescribeEndpoint」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、AWS SDK for Rust API リファレンスの「DescribeEndpoint」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeThing**で を使用する

次のサンプルコードは、DescribeThing を使用する方法を説明しています。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeThing」を参照してください。

CLI

AWS CLI

モノに関する詳細情報を表示するには

次の のdescribe-thing例では、 AWS アカウントの AWS IoT レジストリで定義されているモノ (デバイス) に関する情報を表示します。

aws iot describe-thing --thing-name "MyLightBulb"

出力:

```
{
```

```
    "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
}
```

詳細については、「AWS IoT ディベロッパーガイド」の「レジストリによるモノの管理方法」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DescribeThing」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
     * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeThing」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「DescribeThing」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetachThingPrincipal**で を使用する

次のサンプルコードは、DetachThingPrincipal を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「DetachThingPrincipal」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ コマンドリファレンス」の「DetachThingPrincipal」を参照してください。

CLI

AWS CLI

モノから証明書/プリンシパルをデタッチするには

次の detach-thing-principal 例では、指定されたモノからプリンシパルを表す証明書を削除します。

```
aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
```

このコマンドでは何も出力されません。

詳細については、「AWS IoT ディベロッパーガイド」の「レジストリによるモノの管理方法」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「DetachThingPrincipal」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
```

```
     * 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 の詳細については、「AWS SDK for Java 2.x コマンドリファレンス」の「DetachThingPrincipal」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「DetachThingPrincipal」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンス のDetachThingPrincipal」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListCertificates**で を使用する

次のサンプルコードは、ListCertificates を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   /// <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 の詳細については、AWS SDK for .NET API リファレンスの「ListCertificates」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、AWS SDK for C++ API リファレンスの「ListCertificates」を参照してください。

CLI

AWS CLI

例 1: AWS アカウントに登録されている証明書を一覧表示するには

次の list-certificates 例では、アカウントに登録されているすべての証明書を一覧表示します。デフォルトのページング制限数が 25 を超える場合は、このコマンドの nextMarker レスポンス値を使用して次のコマンドに指定し、結果の次のバッチを取得できます。nextMarker が値なしで返すまで繰り返します。

```
aws iot list-certificates
```

出力:

```
{
    "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 の詳細については、「AWS CLI コマンドリファレンス」の「ListCertificates」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
     * 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 の詳細については、AWS SDK for Java 2.x API リファレンスの「ListCertificates」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「ListCertificates」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「ListCertificates」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListThings**で を使用する

次のサンプルコードは、ListThings を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET 「 API リファレンス」のListThings」を参照してください。

CLI

AWS CLI

例 1: レジストリ内のすべてのモノを一覧表示するには

次の のlist-things例では、 AWS アカウントの AWS IoT レジストリで定義されているモノ (デバイス) を一覧表示します。

```
aws iot list-things
```

出力:

```
{
    "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
        }
    ]
}
```

例 2: 特定の属性を持つ定義済みのモノを一覧表示するには

次の `list-things` の例は、`wattage` という名前の属性を持つモノのリストを表示します。

```
aws iot list-things \
    --attribute-name wattage
```

出力:

```
{
    "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
        }
    ]
}
```

詳細については、「AWS IoT ディベロッパーガイド」の「レジストリによるモノの管理方法」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「ListThings」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスのListThings」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、AWS SDK for Rust API リファレンスの「ListThings」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **SearchIndex**で を使用する

次のサンプルコードは、SearchIndex を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「SearchIndex」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「SearchIndex」を参照してください。

CLI

AWS CLI

モノのインデックスをクエリするには

次の search-index 例では、LightBulb タイプを持つモノについて AWS_Things インデックスをクエリします。

```
aws iot search-index \
    --index-name "AWS_Things" \
    --query-string "thingTypeName:LightBulb"
```

出力:

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

詳細については、「AWS IoT デベロッパーガイド」の「モノのインデックス作成の管理」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「SearchIndex」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /**
     * 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 の詳細については、「AWS SDK for Java 2.x API リファレンス」の「SearchIndex」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「SearchIndex」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスのSearchIndex」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **UpdateIndexingConfiguration**で を使用する

次のサンプルコードは、UpdateIndexingConfiguration を使用する方法を説明しています。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「UpdateIndexingConfiguration」を参照してください。

CLI

AWS CLI

モノのインデックス作成を有効にするには

次のupdate-indexing-configuration例では、モノのインデックス作成が、AWS_Things インデックスを使用したレジストリデータ、シャドウデータ、モノの接続ステータスの検索をサポートできるようにします。

```
aws iot update-indexing-configuration
    --thing-indexing-
configuration thingIndexingMode=REGISTRY_AND_SHADOW,thingConnectivityIndexingMode=STATUS
```

このコマンドでは何も出力されません。

詳細については、「AWS IoT デベロッパーガイド」の「モノのインデックス作成の管理」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「UpdateIndexingConfiguration」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、 AWS SDK for Python (Boto3) API リファレンスのUpdateIndexingConfiguration」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **UpdateThing**で を使用する

次のサンプルコードは、UpdateThing を使用する方法を説明しています。

.NET

SDK for .NET (v4)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <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 の詳細については、AWS SDK for .NET API リファレンスの「UpdateThing」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
//! 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 の詳細については、「AWS SDK for C++ API リファレンス」の「UpdateThing」を参照してください。

CLI

AWS CLI

モノのタイプにモノを関連付けるには

次のupdate-thing例では、 AWS IoT レジストリ内のモノをモノのタイプに関連付けます。関連付けを行うときは、モノのタイプで定義された属性の値を指定します。

```
aws iot update-thing \
```

```
    --thing-name "MyOtherLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

このコマンドでは、出力が生成されません。describe-thing コマンドを使用して結果を表示します。

詳細については、「AWS IoT デベロッパーガイド」「モノのタイプ」を参照してください。

- API の詳細については、「AWS CLI コマンドリファレンス」の「UpdateThing」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
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 の詳細については、「AWS SDK for Kotlin API リファレンス」の「UpdateThing」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK AWS IoT での の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS IoT のクォータ

AWS IoT のクォータについての情報は、「AWS 全般リファレンス」を参照してください。

- AWS IoT Core のクォータの詳細については、「AWS IoT Core エンドポイントとクォータ」を参照してください。
- AWS IoT Device Management のクォータの詳細については、「AWS IoT Device Management エンドポイントとクォータ」を参照してください。
- AWS IoT Device Defender のクォータの詳細については、「AWS IoT Device Defender エンドポイントとクォータ」を参照してください。

AWS IoT Core の料金

AWS IoT Core 価格設定に関する情報は、AWSマーケティングページとAWS価格計算ツールで確認できます。

- AWS IoT Core 価格情報を確認するには、「AWS IoT Core価格設定」を参照してください。
- アーキテクトソリューションのコストを見積もるには、「AWS価格計算ツール」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。